TECHNISCHE
UNIVERSITÄT
DARMSTADT

PROACTIVE MECHANISMS FOR VIDEO-ON-DEMAND CONTENT DELIVERY

Efficient Proactive Content Distribution
and Placement of Video-on-Demand Content

Vom Fachbereich Elektrotechnik und Informationstechnik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Dissertation

von

CHRISTIAN KOCH, M.SC., M.SC.

Geboren am 5. April 1988 in Mühlhausen, Deutschland

Vorsitzender:	Prof. Dr.-Ing. Tran Quoc Khanh
Erstgutachter:	Prof. Dr.-Ing. Ralf Steinmetz
Zweitgutachter:	Prof. Dr. David Hausheer (nach Absprache vorgeschlagen)

Tag der Einreichung: 26. Juni 2018
Tag der Disputation: 17. Oktober 2018

Hochschulkennziffer: D17
Darmstadt 2018

Christian Koch: Proactive Mechanisms for Video-on-Demand Content Delivery
Darmstadt, Technische Universität Darmstadt,
Jahr der Veröffentlichung der Dissertation auf TUprints: 2018
URN: urn:nbn:de:tuda-tuprints-82127
URL: https://tuprints.ulb.tu-darmstadt.de/id/eprint/8212
Tag der mündlichen Prüfung: 17. Oktober 2018

ABSTRACT

VIDEO delivery over the Internet is the dominant source of network load all over the world. Especially Video-on-Demand (VoD) streaming services such as YouTube, Netflix, and Amazon Video have propelled the proliferation of VoD in many peoples' everyday life. VoD allows watching video from a large quantity of content at any time and on a multitude of devices, including smart TVs, laptops, and smartphones. Studies show that many people under the age of 32 grew up with VoD services and have never subscribed to a traditional cable TV service. This shift in video consumption behavior is continuing with an ever-growing number of users. To satisfy this large demand, VoD service providers usually rely on Content Delivery Networks (CDNs), which make VoD streaming scalable by operating a geographically distributed network of several hundreds of thousands of servers. Thereby, they deliver content from locations close to the users, which keeps traffic local and enables a fast playback start. CDNs experience heavy utilization during the day and are usually reactive to the user demand, which is not optimal as it leads to expensive over-provisioning, to cope with traffic peaks, and overreacting content eviction that decreases the CDN's performance. However, to sustain future VoD streaming projections with hundreds of millions of users, new approaches are required to increase the content delivery efficiency. To this end, this thesis identifies three key research areas that have the potential to address the future demand for VoD content.

Our first contribution is the design of vFETCH, a privacy-preserving prefetching mechanism for mobile devices. It focuses explicitly on Over-the-Top (OTT) VoD providers such as YouTube. vFETCH learns the user interest towards different content channels and uses these insights to prefetch content on a user terminal. To do so, it continually monitors the user behavior and the device's mobile connectivity pattern, to allow for resource-efficient download scheduling. Thereby, vFETCH illustrates how personalized prefetching can reduce the mobile data volume and alleviate mobile networks by offloading peak-hour traffic.

Our second contribution focuses on proactive in-network caching. To this end, we present the design of the PROCACHE mechanism that divides the available cache storage concerning separate content categories. Thus, the available storage is allocated to these divisions based on their contribution to the overall cache efficiency. We propose a general work-flow that emphasizes multiple categories of a mixed content workload in addition to a work-flow tailored for music video content, the dominant traffic source on YouTube. Thereby, PROCACHE shows how content-awareness can contribute to efficient in-network caching.

Our third contribution targets the application of multicast for VoD scenarios. Many users request popular VoD content with only small differences in their playback start time which offers a potential for multicast. Therefore, we present the design of the VoDCAST mechanism that leverages this potential to multicast parts of popular VoD content. Thereby, VoDCAST illustrates how Internet Service Providers (ISPs) can collaborate with CDNs to coordinate on content that should be delivered by ISP-internal multicast.

KURZFASSUNG

Die Übertragung von Videoinhalten über das Internet ist maßgeblich verantwortlich für die Last von Internetanbietern auf der ganzen Welt. Vor allem Video-on-Demand (VoD) Streaming-Dienste wie YouTube, Netflix und Amazon Video haben dazu beigetragen, dass VoD-Dienste zu einem festen Bestandteil der täglichen Routine vieler Menschen geworden sind. Mittels dieser Dienste kann jederzeit eine Vielzahl von Inhalten auf unterschiedlichen Geräten wie Smart-TVs, Laptops und Smartphones angesehen werden. Eine US-Studie zeigt, dass viele Personen, die jünger als 32 Jahre sind, bereits mit VoD-Diensten aufgewachsen sind und mit zunehmendem Anteil noch nie einen traditionellen TV-Bezahldienst abonniert haben. Diese Verschiebung des Video-Konsumverhaltens setzt sich fort und wird von einer wachsenden Anzahl von Benutzern begleitet. Um die daraus resultierende große Nachfrage nah VoD-Inhalten zu bedienen, verlassen sich VoD-Dienste typischerweise auf CDN-Netzwerke, welche VoD-Streaming skalierbar machen, indem sie ein geografisch verteiltes Netzwerk von mehreren hunderttausend Servern betreiben. Dadurch liefern CDN-Netzwerke Inhalte von Servern in der Nähe der Benutzer aus, was den Datenverkehr lokal hält und einen schnellen Start der Wiedergabe ermöglicht. CDNs sind tagsüber stark ausgelastet und verhalten sich in der Regel reaktiv zu den Benutzeranfragen, was nicht optimal ist, da dies zu kostspieliger Überprovisionierung auf Seiten der CDN- und Internetanbieter-Netze führt, um die Verkehrsspitzen bedienen zu können. Damit jedoch zukünftige VoD-Streaming-Szenarien mit Millionen von Benutzern ermöglicht werden, sind neue Ansätze erforderlich, um die Effizienz der Inhaltsbereitstellung zu erhöhen. Zu diesem Zweck identifiziert diese Arbeit drei Forschungsschwerpunkte, welche die zukünftige Nachfrage nach VoD-Inhalten adressieren.

Der erste Beitrag ist das Design von vFetch, einem Vorlade-Mechanismus für mobile Geräte, der die Privatsphäre der Nutzer wahrt. vFetch konzentriert sich speziell auf Over-the-Top (OTT) VoD-Dienste wie YouTube. Dabei lernt der Mechanismus das Nutzerinteresse an verschiedenen Inhaltskanälen und nutzt diese Erkenntnisse, um Videoinhalte auf mobilen Endgeräten prädiktiv herunterzuladen. Dazu überwacht vFetch kontinuierlich das Benutzerverhalten und die zur Verfügung stehenden mobilen Verbindungskanäle des Geräts, um eine effiziente und ressourcenschonende Download-Planung zu ermöglichen. vFetch zeigt unter anderem, wie personalisiertes Vorladen von Videoinhalten das mobile Datenvolumen schonen und mobile Netze entlasten kann, während es gleichzeitig eine netzunabhängige und hochqualitative Videowiedergabe gewährleistet.

Der zweite Beitrag konzentriert sich auf proaktives Zwischenspeichern von Videoinhalten in Netzwerken. Zu diesem Zweck wird das Design des ProCache-Mechanismus präsentiert, der den Speicherbereich von Caches basierend auf Inhaltskategorien in separate Speicherbereiche aufteilt. Der verfügbare Speicher wird diesen Bereichen basierend auf ihrem Beitrag zur Effizienz des Gesamtsystems zugeordnet. Wir schlagen die Unterstützung von mehreren Videokategorien sowie einen auf Musikvideoinhalte zugeschnittenen Arbeitsablauf vor, da Musikvideos die Haupt-

last von YouTube darstellen. Damit zeigt ProCache, wie die Berücksichtigung von Inhaltskategorien zu einem effizienten Zwischenspeichern von Inhalten in Netzen beitragen kann.

Der dritte Beitrag zielt auf die Anwendung von Multicast für VoD-Szenarien ab. Populäre VoD-Inhalte werden von vielen Nutzern mit nur geringen Unterschieden in ihrer Wiedergabe-Startzeit geschaut, was ein Potential für Multicast eröffnet. Der vorgeschlagene Mechanismus, VoDCast, nutzt dieses Potenzial, indem Teile beliebter Videos per Multicast übertragen werden. Dadurch zeigt VoDCast, wie Internet-Anbieter mit CDN-Netzwerken kooperieren können, um die VoDCast-basierte Videoübertragung zu koordinieren.

ACKNOWLEDGMENTS

There are numerous people who have helped and supported me working on this thesis. First of all, I would like to thank my supervisors Prof. Dr.-Ing. Ralf Steinmetz and Prof. Dr. David Hausheer for giving me the opportunity to become a doctor at the TU Darmstadt and for providing valuable and productive work places. Furthermore, I would like to thank Dr. Amr Rizk for his support, advice, and guidance. Especially the Multimedia Communications Lab (KOM) and the collaborative research center MAKI have strongly promoted my research. I would also like to thank Prof. Klara Nahrstedt for her advice, help, and for the opportunity to work at the Multimedia Operating Systems and Networking (MONET) Research Group at the University of Illinois Urbana-Champaign. A special thank you goes to Prof. Wolfgang Effelsberg, Prof. Andreas Mauthe, and Prof. Michael Zink from them I got many inspirations during our fruitful discussions. Furthermore, I would like to thank all my current and past colleagues at KOM – especially Alex, Denny, Fabian, Jeremias, Julius, Leo, Manisha, Matthias, Rhaban, Sounak and Stefan – Mrs. Scholz-Schmidt, Mrs. Ehlhardt, and the administrative staff! And further thanks to my students, especially Andreas, Benedikt, Ganna, Johannes, Moritz, and Stefan. I am especially grateful to my parents, my grandparents, and my friends Christof, Jakob, Lukas, and Sacha for their constant support and encouragement. Last but not least, I would like to thank, Hans-Jörg Kolbe, Georg Schyguda and Gerhard Hasslinger from T-Systems (Deutsche Telekom) as well as Ali Gouta for their valuable input and fruitful discussions.

CONTENTS

1 INTRODUCTION

VIDEO STREAMING is the transmission of live or prerecorded video content over the Internet and an essential part of many people's daily routine. Just a decade ago, television was the undisputed single source of multimedia entertainment in people's homes. At this time, video content was delivered to customers by satellite or by cable. The content with the highest perceived quality, in resolution as well as substance, was offered by pay television. What people watched was decided only by broadcast and television networks. Nowadays, smartphones, tablets, PCs as well as TVs connected to the Internet allow people to consume video from more than one device and content source [Con16]. Using multiple of these devices at the same time is a new trend known as *second screen* watching. For many people, Video-on-Demand (VoD) services such as YouTube, Netflix, or Amazon Video are used daily and often are the only possibility to watch the highest video quality, which is at this time a 4k resolution. In the US, more people have a Netflix subscription than a cable TV subscription [Hud17] and television does no longer dominate the total screen time since 2016 [Con16]. Overall, the so-called "cable-nevers", who grew up with VoD services and never have subscribed to cable TV services are forecasted to account for 50% of the people under the age of 32 in the US by 2025 [Lyn15].

Benefits of video streaming

The increasing popularity of video streaming services is propelled mostly by faster end-user Internet connections and higher bandwidths available at Internet Service Provider (ISP) core and access networks, as well as a multitude of affordable mobile devices, highly compressive video codecs [GMM+13; ES98], and a rapidly growing amount of, e.g., music, movies, tutorials, and information available as video [Eri13; Eri15]. Recent reports show that multimedia streaming constitutes the largest share of the global Internet traffic [San15a; San15b; Cis17c]. In addition to that, they forecast an ongoing and robust growth of Internet traffic, mainly caused by video streaming. Overall, Internet Protocol (IP) traffic is forecasted to grow with a Cumulative Average Growth Rate of 24% [Cis17c] between 2016 and 2021. Video traffic is forecasted to grow even faster with a Cumulative Average Growth Rate of 31% and, hence, to account for 82% of total consumer Internet traffic by 2021 [Cis17d].

Video traffic on the rise

VoD streaming services are experiencing increasingly high user-adoption due to several inherent advantages. Foremost, time-independent and user-initiated content consumption has become possible by VoD, in contrast to, broadcast or live video streaming. Taking fixed networks in North America as an example, two major players cause more than half of the Internet traffic: Netflix[1] and YouTube[2] [San15a]. Netflix offers on-demand streaming of TV series and movies. In contrast to Netflix, YouTube delivers mainly User-generated Content (UGC).

VoD dominates video traffic

Remarkably, in mobile networks, YouTube was the dominant source of video traffic accounting for 40%–70% of all mobile video traffic by 2017 [Eri17]. This is a large traffic share compared with Netflix accounting for just 10%-20%. Among all VoD

YouTube as a major VoD service

[1]Netflix Inc., `https://www.netflix.com/` [Accessed: November 19, 2018]
[2]YouTube, `https://www.youtube.com/` [Accessed: November 19, 2018]

services, YouTube is the most used service, with 70% of their users utilize it at least on a weekly basis. With more than one billion users and more than one billion hours of watch time per day, on mobile devices alone, YouTube reaches more people aging between 18 and 34 years than any cable TV network in the US[3]. Both YouTube and Netflix are Over-the-Top (OTT) content providers, meaning that they have to transfer their content through the Internet's transit and broadband access ISPs [Cla16] instead of IP Television (IPTV), serving the video content from an ISP directly to its customers [DRC10; LLW+11].

Summarizing multiple reports, the delivery of OTT VoD content through the Internet is already highly important and forecasted to grow further. The support of efficient OTT VoD mechanisms is fundamental considering the expected traffic volume which is driven by trends, such as the demand for higher quality video streams, larger display sizes, better cameras, 360° videos, and virtual reality [Ste12; Ste92]. Given the vast amount of VoD traffic in the foreseeable future, new content distribution and placement mechanisms are required to act proactively by reflecting the dynamic and heterogeneous user demands. In this context, the term proactive refers to a popularity prediction for known content. This includes mechanisms applied on Content Delivery Networks (CDNs), which help to scale video streaming and bring content to the users operating hundreds of thousands of globally deployed cache servers. Furthermore, ISPs and user terminals can benefit from proactive mechanisms.

1.1 MOTIVATION AND PROBLEM STATEMENT

Both industry and the scientific community have extensively studied the distribution of VoD content on an Internet-scale due to its fundamental importance and implied challenges for the future. According to IneoQuest, by 2016, 51% of streaming viewers experienced "buffer rage", an industry term for uncontrollable fury or violent anger induced by the delayed or interrupted enjoyment of streaming OTT video content [Ine16; Con16]. Men watching sports videos experience the strongest buffer rage, the most often. Further, studies also found dissatisfying experiences of users watching videos on YouTube [CDF+14b; CDF+14a]. This indicates that user demand grows faster than the available bandwidth [BBS14; GEF16]. In addition to insufficient bandwidth, packet loss and high latency caused by long transmission paths or request redirection [AJC+12] impair the video quality by introducing initial playback stalling [HSB+13; SFF+16]. This demonstrates that efficient video streaming is still an open research area with high relevance for the industry since high-quality video delivery is a crucial differentiator for streaming service providers [Gri17].

OTT VoD delivery is an open research area

In the scope of this thesis, three complementary key research areas were identified focusing on individual aspects of the problem and, thus, define the focal points of this thesis. Figure 1 depicts the overall OTT VoD delivery scenario and highlights the conceptual focus on these three areas. On the figure's left side, the logical layers are depicted to classify the three areas depicted on the right side. This thesis is motivated by an observed lack of proactive and efficient OTT VoD support in all three areas. Without proactivity, caching mechanisms are prone to ephemeral popularity changes, multicast mechanisms are inefficient to use for the large number of OTT VoD content

Three key research areas for OTT VoD

[3]YouTube, `https://www.youtube.com/yt/about/press/` [Accessed: November 19, 2018]

available, and prefetching performs poorly because of its inability to adapt to a user's interests in the absence of proactivity [WSS+16; WRT+15]. By addressing the research areas at the application and network layer, large-scale OTT VoD streaming will stay feasible given the large amount of traffic estimated. In each of the three areas, several specific individual sub-challenges have to be solved toward this goal. In the first area, adaptability and accuracy are challenging as individual persons are highly diverse in interest, taste, and predictability. Regarding the second area, a detailed content-oriented understanding of how proactive caching can contribute to efficient content delivery is missing. Considering the third area, selecting VoD content and efficiently using Software-defined Networking (SDN)-based multicast for OTT video delivery is challenging. To leverage the underutilized potential of proactivity, in contrast to traditional reactive mechanisms, we address the areas presented in Figure 1.

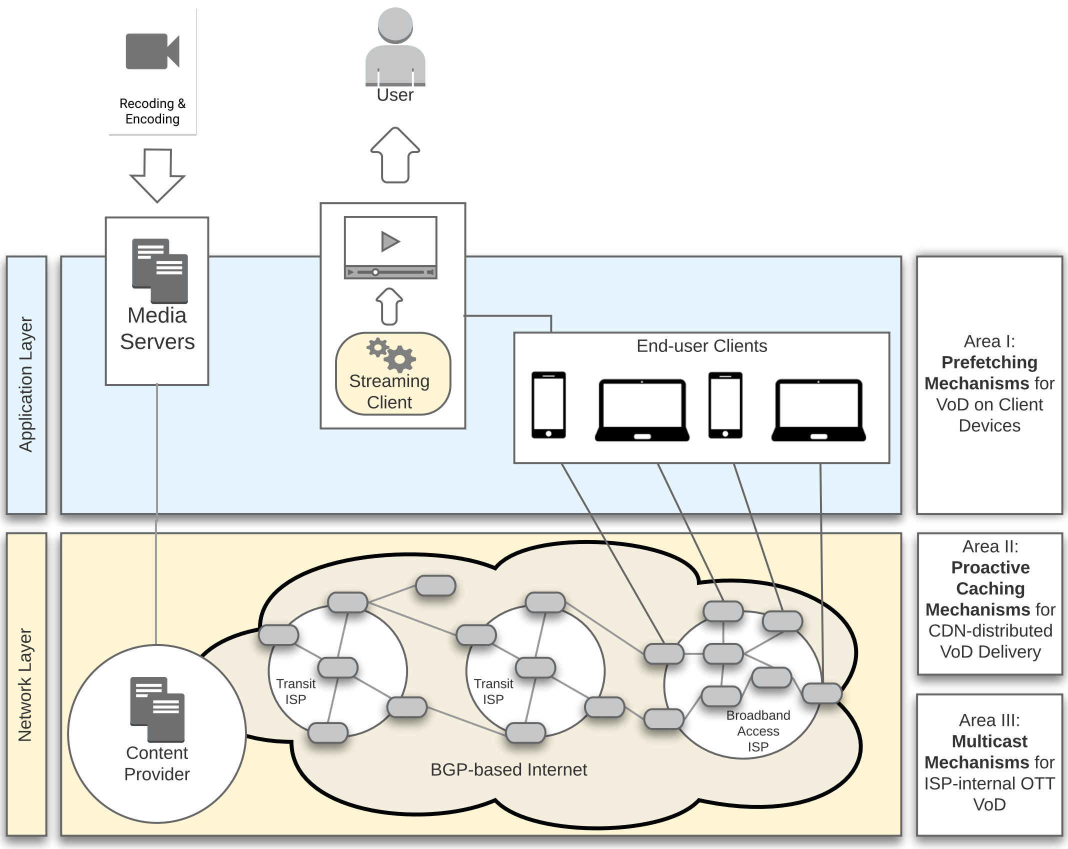

Recoding & Encoding
User
Media Servers
Streaming Client
End-user Clients
Application Layer
Network Layer
Transit ISP
Transit ISP
Broadband Access ISP
Content Provider
BGP-based Internet
Area I: **Prefetching Mechanisms** for VoD on Client Devices
Area II: **Proactive Caching Mechanisms** for CDN-distributed VoD Delivery
Area III: **Multicast Mechanisms** for ISP-internal OTT VoD

Figure 1: Relevant areas of flexible and efficient OTT VoD delivery (based on [Rüc16])

Research Area I: Prefetching Mechanisms for VoD on Client Devices

The first research area addresses the "last mile" and refers to the comparable low capacities of the aggregation and access network compared with the ISP core network. Thus, if many users demand bandwidth at the same time is it likely that the per-user bandwidth decreases. This applies to mobile as well as to fixed networks. Just in-

Traffic peaks are expensive

creasing the Capital Expenditures (CAPEXs) and overprovision networks by buying more hardware is rather inefficient as the bandwidth demand is not constant but fluctuates on a daily basis. Furthermore, traffic is costly as ISPs typically pay traffic from and to another ISP by burstable billing[4], i.e., traffic peaks are most expensive.

Prefetching circumvents low-quality Internet connections

Prefetching mechanisms have been proposed to transfer network load to off-peak hours and, hence, reduce the overall transit costs. Furthermore, when prefetching is applied on client devices, prefetched content can be played back without relying on a potential low-quality mobile connection. Thereby, Quality of Service (QoS) impairing factors such as insufficient bandwidth, high latency, and packet loss can be avoided. This usually correlates with a high Quality of Experience (QoE), i.e., the individual user-perceived quality of the video playback. Especially for mobile devices, prefetching can help to save a large amount of energy, a limited resource on mobile terminals. When Wi-Fi is used for offloading instead of streaming via 4G, the energy costs are about 20-times lower [HQG+12], thereby avoiding a fast battery depletion.

VoD prefetching is challenging

However, accurately predicting what content the user is interested in is challenging. While it is likely that a Netflix or an Amazon Video customer watches the next episode of a series the customer regularly watches [DKA+15], the prediction for services like YouTube is more complex. Notably, the potential for caching YouTube content is significantly higher than for Netflix content [RDH+13]. For example, the intuitive idea of prefetching recent videos from YouTube channels the user has subscribed to is generally inefficient [KLR+17]. Although YouTube recommends videos on its landing page, prefetching based on this information has also shown a poor performance [WSS+16] similarly to selecting videos by their like count [WRT+15]. Native video features, such as the video's channel subscription status [KLR+17] and the global popularity [WRT+15], are rather ineffective for prefetching. This demonstrates that individual video prefetching is still challenging for efficient traffic offloading.

Prefetching for individual users

Research Area II: Proactive Caching Mechanisms for CDN-distributed VoD Delivery

CDNs make video streaming scalable

Content Delivery Networks (CDNs) provide a globally distributed overlay network of cache servers. Thereby, they help video streaming to become scalable and reliable as CDNs are capable of serving millions of users at the same time and have the possibility to load-balance the traffic among their cache servers [MS15]. As the content can be streamed from nearby cache servers, also the transmission latency is lower than when streaming it from the potentially far-away content source. This leads to enhanced QoS and user satisfaction, as users expect a web page or a video to load timely [Kis17; SKL+14].

Proactive caching as a promising technique for caching

However, efficient caching at the ISP's access and aggregation network is still challenging. In the future Internet, it is likely that many in-network caches are deployed at different locations, e.g., at base stations or per metropolitan area of a country as well as in home routers in the form of in-network caches serving femtocells [BBD14; GSD+12]. Here, cache capacities are limited and smaller than in data centers. Hence,

[4] http://www.netrepid.com/burstable-billing-95th-percentile/ [Accessed: November 19, 2018]

varying content demand to a large content catalog decreases the cache performance because most caching approaches are designed for large data center-scale caches and not for the smaller storage capacities at the network's edge. The most common cache performance metric is the Cache Hit Rate (CHR) which is the ratio of content delivery by the cache to the overall content requests to the cache. For small caches but also for larger caches, proactive caching has shown to outperform reactive caching [Gou+15; HNH14; KWR+18]. Preliminary work shows that proactively placing content on servers [SBE+16], femtocells [GSD+12], and home routers [LPB+15; SBH13] has the potential to unburden content servers and networks efficiently. Here, content is loaded and statically kept within the cache based on, e.g., social information or content properties. Since small caches are operated at the edge of the network, they can efficiently unburden the core network and reduce costly transit traffic and at the same time enhance the video service quality due to a low transmission latency.

Research Area III: Multicast Mechanisms for ISP-internal Delivery of OTT VoD

Missing multicast support for OTT VoD services

The third area is motivated by the potential of cooperation between content providers or CDNs and ISPs. CDNs are used for streaming of, both, on-demand as well as live video content, among others. Though, they are often not allowed to place and operate their reverse proxies inside ISPs because they consider CDNs as unpredictable and hard to manage traffic sources [HH11]. Consequently, CDNs are often not present inside ISP networks and, hence, cannot help to optimize the content distribution there. However, ISPs often use multicast for efficient delivery of their own IPTV services which are available only for the ISP's customers [LLW+11]. Nevertheless, limited resources make traditional IP multicast unlikely to scale for OTT services, because each router on the delivery path needs to offer multicast capabilities [DLL+00].

SDN as an enabling technology for multicast

ISPs are widely adopting SDN [JKM+13] which is an enabling technology for a multitude of network services, also for multicast. Here, SDN enables ISPs to deliver content based on efficient network-level multicast in cooperation with content providers or CDNs. So far, Software-defined Multicast (SDM) has been proposed and evaluated for OTT live streaming [Rüc16; RBH+16; YYR+15], but has not yet been adapted to the OTT VoD delivery scenario. Popular VoD content is typically watched thousands of times per day solely within a single broadband access ISP network. To this end, it is likely that different persons stream the same content with only minor differences in their playback positions. This case offers the potential to efficiently multicast parts of these videos and, thereby, reduce the ISP's and CDN's link utilization, and consequently decreasing operational expenditures.

Multicast for OTT VoD

Challenges for SDN-based multicast for OTT VoD

A further distinguishing criterion between live and on-demand is that much more on-demand content, e.g., about 1.2 billion[5] videos in the case of YouTube, exists compared with the much smaller content catalog of IPTV services. Since the number of multicast groups that can be supported by SDN-enabled switches is limited by the amount of memory they are equipped with [KRE+15], network-based

[5] http://tinyurl.com/j63ffxd [Accessed: November 19, 2018]

multicast cannot deliver all of these videos. In addition, VoD popularity is Zipf-distributed [GHM13; ACG+09; GAL+07]. Therefore, only a small fraction of the content catalog is popular and, thus, efficient to be delivered by multicast. Further, efficiently selecting videos for multicast is challenging as popularity changes dynamically based on, e.g., video age and daytime [CKR+07]. So far, none of the existing works explores the potential of SDN-based multicast of VoD content streams with a small playback distance and Zipf-distributed popularity. Furthermore, related approaches focus mostly on live video streaming [RBH+16; YYR+15] or ignore network resources limitations and ignore the user's need for a small initial buffering time [DSS96; HCS98; ACG+09]. Summarizing, an efficient SDM-based system for VoD delivery needs to, first, adapt to the video popularity, second, consider the limited number of possible multicast groups, and, third, reduce the ISP and CDN bandwidth utilization.

1.2 RESEARCH GOALS

After the three research areas have been introduced in the preceding section, now we define the primary goals of this thesis. Based on the problem statement above and the previously identified research areas, three primary goals are formulated for this thesis. These goals imply a number of research questions, presented in the following.

Research Goal 1: *Support of Prefetching for Individual Users on End Devices*

The first research goal is to design a predictive prefetching mechanism that determines user interests in a way that videos, interesting to the user, can be detected and downloaded in advance of the actual user video request. The term "predictive model" refers to a user-specific and adaptive mechanism that regularly adapts to the user's interests and behavior. To this end, textual features, regular user habits, as well as connectivity patterns have to be considered to maximize the efficiency. Research questions that are of interest in the context of the first research goal are:

RQ 1.1: How accurate can video requests of different users be predicted, considering user interests?

RQ 1.2: Which CHR gain can be achieved by considering content properties for content access prediction?

RQ 1.3: How much of a user's mobile video traffic can be saved using a predictive prefetching model?

Research Goal 2: *Support of Proactive Caching for Video-on-Demand Content*

The second research goal is to design a proactive caching mechanism that considers content popularity dynamics to increase the CHR and decreases write operations on the cache storage. In this context, the term "proactive" refers to a popularity prediction of known content for a fixed set of users whose requests pass the caching system. To study the effect of proactive caching, a distinct storage share of the caches is filled proactively with content according to the prediction, while the remainder is managed in a traditional reactive fashion. Since CDN caches are often connected by an overlay, common overlay topologies have to be considered. Research questions that are of interest in the context of the second goal are:

RQ 2.1: How can proactivity be used to enhance caching performance?

RQ 2.2: Which prediction policies show the highest performance regarding the CHR?

RQ 2.3: How does cache storage size contribute to the CHR gain of proactive caching?

Research Goal 3: *Support of Multicast for Popular Video-on-Demand Content*

The goal is to design a multicast mechanism that supports broadband access ISPs to deliver OTT VoD content to their customers efficiently. Here, the ISP is assumed to be explicitly involved in the content delivery process and to collaborate with the content origin or the CDN hosting the content. Additionally, to the network load reduction, we also address costs occurring by providing and managing multicast groups. Research questions that are of interest in the context of the third goal are:

RQ 3.1: How much does multicast lower the traffic volume caused by VoD content within ISP networks?

RQ 3.2: How can VoD multicast be realized using SDN?

RQ 3.3: Which VoD content should be selected for being delivered by multicast to decrease the data traffic volume?

1.3 METHODOLOGY

To answer the proposed research questions, the related work in the respective areas is studied and reviewed. In the scope of this thesis, we propose innovative mechanisms considering existing systems, realistic workloads, and the different stakeholders in the VoD delivery process. Our mechanisms are designed for the application and network layer, and they contain communication protocols and policies. Thereby, they consider suitable measures of content popularity to anticipate and react to dynamic network load and demand changes proactively. We evaluate the mechanisms' performances and costs by implementing them as a simulation model, using real-world workloads from either user traces or a network trace from a large European mobile ISP. We laid a particular focus on evaluating the proposed mechanisms in relevant but exemplary simulation environments. However, an evaluation of the proposed mechanisms in real-world productive deployments was not conducted due to the large number of required resources and implementation effort for CDNs, ISPs, and end users.

We developed and presented individual functional prototypes and technical demonstrations [KRB+14; KBR+15; KPR+18] to the scientific community to highlight the practicability of selected mechanism aspects and their deployment.

1.4 CONTRIBUTIONS

This thesis proposes three contributions that aim to address the presented research questions:

- **Design of a novel privacy-preserving & individual prefetching mechanism:** In Chapter 4, we propose the privacy-preserving prefetching mechanism vFetch. It accurately determines the user interests and downloads matching content to the user's mobile terminal. Thereby, Wi-Fi-offloading is preferred to save energy and mobile data plane.

- **Design of an innovative proactive caching mechanism for CDNs and ISPs:** In Chapter 5, we propose the ProCache mechanism which uses efficient proactive caching by adapting its cache storage allocation to dynamic content popularity distributions.

- **Design of a new SDN-based multicast mechanism for OTT VoD content:** In Chapter 6, the SDN-based multicast mechanism VoDCast is proposed, enabling an efficient delivery of OTT VoD content to customers of broadband access ISPs.

Popularity-triggered mechanism transition

Each of the proposed mechanisms for the individual research areas can contribute to more efficient OTT VoD delivery on its own. However, all three of these approaches can coexist to achieve a higher impact than each approach on its own, thereby allowing an even more efficient and high-quality delivery of OTT VoD content for a significantly more demanding audience than observed in today's Internet. While vFetch runs independently of VoDCast and ProCache on user terminals, ProCache and VoDCast are ISP/CDN-operated and are likely to benefit from coordination [FCL17]. In Figure 2, we find that the different mechanisms address content of different popularity. Hence, a transition (cf. Section 2.6) between the three proposed mechanisms over a content's popularity life cycle is desirable. Thereby, content that changes its popularity, e.g., due to its age, should be served by the most suitable of the three mechanisms. VoDCast addresses highly popular content such as new music releases from popular artists, e.g., the "Gangnam Style" video[6]. ProCache handles content that is still requested by many users, but the Inter-Request Time (IRT) requirements are relaxed compared to VoDCast, i.e., no overlapping user video playbacks are required. In the absence of VoDCast, ProCache can also manage highly popular content efficiently. vFetch manages content belonging to the popularity distribution's long tail efficiently, as it can adapt to highly diverse and individual user interests.

[6] https://www.youtube.com/watch?v=9bZkp7q19f0 [Accessed: November 19, 2018]

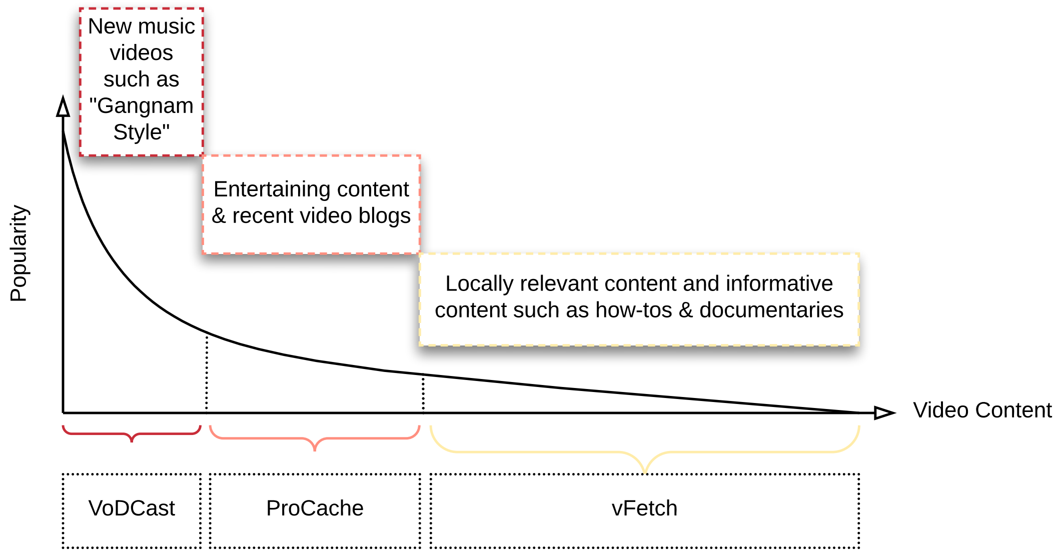

New music videos such as "Gangnam Style"
Entertaining content & recent video blogs
Locally relevant content and informative content such as how-tos & documentaries
Popularity
Video Content
VoDCast
ProCache
vFetch

Figure 2: Coexistence of the three proposed contributions: VoDCast, ProCache, and vFetch

1.5 THESIS ORGANIZATION

The remainder of this thesis is structured as follows: Chapter 2 describes background information on, e.g., ISPs, CDNs, SDN, and video streaming. Moreover, this chapter introduces fundamental concepts of the proposed research contributions, such as caching strategies and machine learning. Chapter 3 describes related work in the areas of the different research contributions. Chapter 4 presents the design and evaluates the vFetch mechanism. Efficient and proactive network caching is presented and evaluated in Chapter 5. In Chapter 6, the SDN-based multicast of popular VoD content is designed and evaluated. The three contribution chapters are concluded by Chapter 7, summarizing the obtained insights and results. Additionally, promising directions of future research are presented by an outlook.

2 BACKGROUND

This chapter provides the necessary information to follow and understand the research contributions presented in this thesis. To this end, the following presents basics of video streaming, Software-defined Networking (SDN), machine learning, and multi-mechanism transitions.

2.1 INTERNET SERVICE PROVIDER

Definition 1: *Internet Service Provider*

An Internet Service Provider (ISP), in the context of this thesis, is understood as a company providing broadband Internet access to its customers that are typically private customers but can also be commercial customers.

ISPs provide broadband access to different networks and, thereby, create a network of networks known as the Internet. Usually, the term ISP refers to broadband access providers like AT&T, Verizon, and Deutsche Telekom. However, next to these broadband access ISPs, another kind exists that provides access, not for private customers but other ISPs and companies. ISPs are autonomously managed networks, identified by a unique Autonomous System Number that allows routing between different Autonomous Systems (ASes). When two ASes connect, the Exterior Gateway Protocol exchanges sets of Internet Protocol (IP) prefixes reachable within and through the ASes [HB96]. Thereby, connectivity between different ISPs is provided.

The architecture of modern ISP networks is presented in the following. This section's content is partially based on the works of Doverspike et al. [DRC10] on AT&T's ISP network and Betker et al. [BGK+14] providing insights in the ISP topology of Deutsche Telekom. Thereby, we focus on the transport layer because of the relevance for this thesis and the sake of simplicity.

2.1.1 *ISP Network Architecture*

An ISP network's topological structure is by no means fixed and can vary between different countries depending on, e.g., the population density and country size. Though, an ISP-internal network architecture comprises typically three different parts: the access network, the aggregation network, and the core network [DRC10]. Figure 3 gives an overview of a typical ISP network topology.

Access Network

The access network provides Internet access to the ISP customers via the Broadband Network Gateway (BNG), which ensures the customer can authenticate by providing its credentials and does not consume more bandwidth than the customer pays for. The connection between a customer and a BNG is established by the customer's Digital Subscriber Line modem which initiates a Point-to-Point Protocol over Ethernet

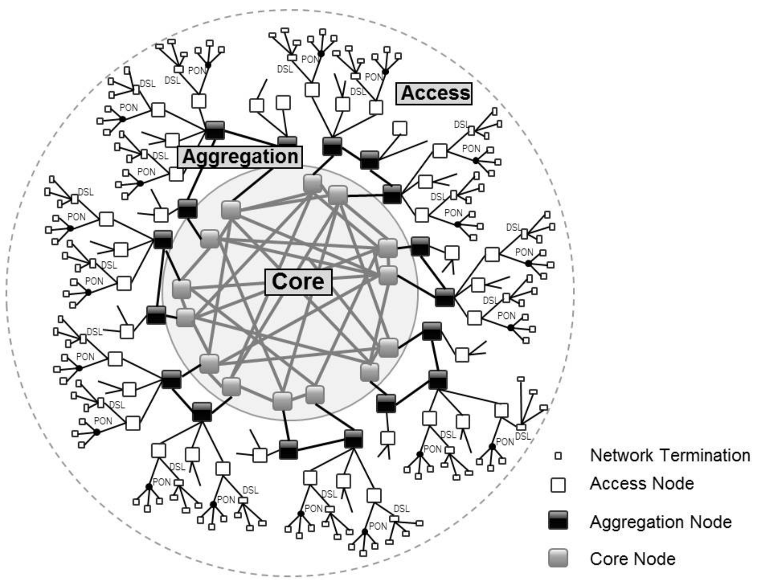


Figure 3: Typical ISP network topology: Star-/tree-like and meshed network parts [BGK+14]

connection with the BNG. An access network is typically tree-shaped and aggregates several customers along the way back to the BNG by so-called concentration nodes [BGK+14].

Aggregation Network

BNGs and Label Edge Routers route Internet traffic between the access network and the aggregation network, also known as Metropolitan Area Network, using Multiprotocol Label Switching as the routing protocol. Physically, an aggregation network is built by fiber optic cables which are formed in a ring topology connected to a core router. While multiple Label Switching Routers are placed on the ring, on the network layer, only one hop to a dedicated core network router is possible.

Core Network

The core network is the ISP's backbone that provides high bandwidth capacities. These core network capacities typically largely exceed the demand needed to route the traffic. This is called over-provisioning and avoids customer connection impairments in unlikely worst-case scenarios of network congestions or outages. In case of Deutsche Telekom, the complete backbone exists twice, i.e., for each core router, a second core router exists for safety and redundancy reasons. In case of a core router defect or an optical link failure, the routes can be re-configured within 50ms[BGK+14] and take over. After the traffic reaches the core network, two possibilities exist. If the destination is within the ISP network, the core network routes the traffic to the corresponding aggregation network, where it is routed to the corresponding access network and eventually delivered. If the destination is outside of the ISP network, the traffic has to be routed to a Border Gateway which is able to forward the traffic outside the ISP network and find the destination ISP.

2.1.2 *Transit and Peering Agreements*

To allow traffic routing from and to other ASes, two kinds of business agreements exist, peering and transit. Peering agreements are often seen between ISPs of equal size which transmit about the same amount of traffic in each other's ISP network. However, if the ISPs are unequal in size and traffic, transit agreements are applied.

Transit

Overall, transit agreements represent the majority of intra-ISP agreements [Nor14]. When using transit agreements, the small ISP pays the large ISP for providing it access to the rest of the Internet. Thereby, the large ISP becomes the so-called upstream provider of the smaller ISP. Measuring and billing of Internet transit traffic is a well-established business model, especially for large ISPs, so-called Tier 1 ISPs, whose only business model is to provide smaller ISPs with access to the rest of the Internet.

Peering

In contrast to this, peering is an interconnection between ISPs of similar data transfer volume without charging costs between each other. A further difference to transit agreements is that peering is not transitive [Nor14]. This means the peering ISPs do not provide access to other ISPs to which they might be connected, e.g., by a cost-charging transit agreement. Instead, peering provides only access between the two ISPs and, hence, to each of their customers.

2.1.3 *OTT IP Multicast*

Multicast is the delivery of the same data to a set of clients without transferring it redundantly over the same link. Instead, the data is only transferred once along the delivery path and replicated at the latest possible point in the network to deliver it individually to the requesting clients. In the IP multicast model proposed by Deering [Dee89], a packet is transmitted at most once over a link. Figure 4 gives an example by visualizing the paths taken by packets in case of unicast and IP multicast. Thereby, the number of clients can vary from only a few to millions. Parallel to the number of clients, the state needed for multicast management grows. Technically, multicast is realized by placing multicast-capable routers within the ISP network. This approach has two major limitations. First, multicast routers are only capable of serving a limited amount of clients. Second, also the user churn, i.e., new users joining the multicast stream and leaving it creates a management overhead and is limited.

However, this is a viable approach for scenarios with a limited number of users and streams. One example for such a scenario are ISP-internal IP Television (IPTV) services [LLW+11; RCW16], such as EntertainTV[1] from Deutsche Telekom. However, for Over-the-Top (OTT) streaming, IP multicast is not used as ISPs fear external content sources such as Content Delivery Networks (CDNs) as unpredictable and hard to manage traffic sources [HH11]. The term OTT emphasizes that the content provider uses the public Internet as transport for data delivery (cf. Definition 3).

This is likely to be the reason that none of the proposed solution models developed by researchers [HC99; DLL+00] has been applied in reality so far. To this end, OTT video content is commonly not served by ISP-internal multicast and, hence, is delivered using unicast. This does not scale well for live events with millions of viewers such as an Apple keynote, where even a single data center can become a bandwidth bottleneck [AMM+03] as each client gets the content delivered by a separate unicast transmission.

With the advent of the SDN paradigm, ISP-internal multicast for OTT content has been reconsidered and new promising solution models have been developed as discussed further in Section 3.3.

[1] `https://www.telekom.de/zuhause/fernsehen/entertaintv` [Accessed: November 19, 2018]

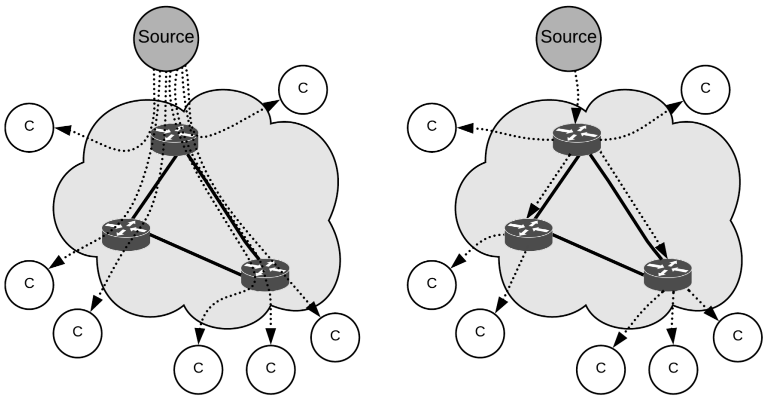


Figure 4: Typical ISP data dissemination using unicast (left) and IP multicast (right)

2.2 CONTENT DELIVERY NETWORKS

Content Delivery Networks (CDNs) carry more than half of the multimedia content in today's Internet and are predicted to deliver 71% of the global Internet traffic by 2021 [Cis17d]. In the course of this thesis, we define CDNs based on Buyya et al. [BPV08] as follows.

Definition 2: *Content Delivery Network*

A Content Delivery Network (CDN) is a collaborative collection of servers spanning the Internet, replicating content between them to perform transparent and effective delivery of files to end users.

Benefits of CDNs

CDNs have advantages for content providers, ISPs, and users. By placing content close to users, CDNs offload the content provider. Thereby, their service becomes scalable and reliable as CDNs are capable of serving millions of users at the same time and have the possibility to load-balance the traffic amongst its reverse proxy servers [MS15]. As the content can be streamed from nearby reverse proxies, also the delay is much lower than streaming it from the potentially far-away content source. This leads to enhanced Quality of Service (QoS) [Fur98] and user satisfaction as users expect a web page or a video to load timely [Kis17; SKL+14].

Especially in the case of video streaming, low delays influence not just the time till the video playback starts. Besides, it is likely to positively affect the Quality of Experience (QoE) during the entire streaming session [SFF+16; Ste96].

Tasks of CDNs

For ISPs, the main advantage of CDNs is that they reduce the amount of traffic to carry as near-by reverse proxies deliver the content instead of the potentially far-away content source. Thereby, the amount of Internet traffic and potential transit costs are reduced. To achieve the benefits mentioned above, CDNs have to fulfill specific operational tasks. Vakali and Pallis define four essential tasks a CDN has to take care of [VP03]:

1. **Caching** of the content source's content at the CDN's cache proxies. Not much is known about how exactly cache management is operated because this is the key business secret of a CDN. However, Akamai revealed that Least Re-

cently Used (LRU) is used as cache eviction policy at most of their cache proxies [MS15].

2. **Routing** of client requests to a suitable cache proxy. Large CDNs operate thousands of cache proxies distributed in data centers around the world. Here, user request assignment is performed considering multiple metrics, such as the distance to the user but also the load of the proxies [MS15].

3. **Content distribution** of the content from the content source to the cache proxies is performed by a high capacity backbone managed by the CDN.

4. **Accounting** provides mechanisms for the content source to get informed on its contents' popularity and potential costs that have to be paid to the CDN.

More details on how CDNs work internally can be found in the paper of Maggs and Sitaraman [MS15] as well as in Vakali's and Pallis' paper [VP03].

A term that gains importance in this context is Over-the-Top (OTT) video streaming. This term means that a video stream is delivered from a third party, i.e., over the public Internet and not from within the user's broadband access ISP [NLB12]. Examples of OTT streaming providers are YouTube, Netflix, and Amazon Video. YouTube and Netflix are the two largest sources of Internet traffic in North America and account for more than half of the peak downstream traffic [Cis17b]. Hence, the delivery of Video-on-Demand (VoD) content, as defined by Steinmetz et al. [Ste12], is a major use case for CDNs. In contrast to ISP-internal IPTV services, when using OTT video streaming services, no guarantees for the delivered video quality can be given. Here, ISPs only deliver the OTT video stream but typically have no control over the delivery method. In the course of this thesis, we define OTT as follows.

Over-the-Top Streaming

Definition 3: *Over-the-Top*

Over-the-Top (OTT) video streaming delivers video content using the public Internet. Thereby, the content provider is a third party different from the requesting user's broadband access ISP.

2.2.1 *Video Caching*

CDNs are often used to provide content caching for OTT video streaming services. Depending on the ISP policies, caches can be operated for OTT services within the ISP's network or just close to it. Especially large ISPs tend to avoid having caches for OTT services in their networks. However, they are often not allowed to place and operate their reverse proxies inside ISPs because they consider CDNs as unpredictable and hard to manage traffic sources [HH11]. However, they may use caches for their own IPTV services for their customers [LLW+11]. In the following, we will focus on OTT VoD content caching, though, live video streaming, ISP-owned IPTV services, and other web services also make use of caching. It is expected that caching will stay and even rise as a major part of the video streaming process. This is foremost reasoned by the memory and bandwidth cost projections [RS13]. However, the traffic decreasing effect of a single cache is limited by its capacity, i.e., the number of bytes or items it can contain. To this end, most research in this area focuses on increasing the Cache Hit Rate (CHR) or decreasing the write operations on the cache's disks to increase their lifetime and, hence, reduce Capital Expenditure (CAPEX).

Since most videos are just watched partially [KH14; MGP+18] and different quality demands exist, it is state-of-the-art to cache not entire videos but video segments of about 1-10 seconds length and a certain quality (ref. Section 2.3.1). Especially in VoD, it is common that users crawl over videos sequentially and watch just the first segments till they find an interesting video which they are interested in for a longer time.

Definition 4: *Caching Strategy*

A caching strategy is a combination of an admission policy, determining if a newly requested item is cached, and an eviction policy, determining which item has to be deleted to allow inserting a new item.

In the remainder of this thesis, we define a caching strategy as a combination of an admission and an eviction policy. An admission policy determines if a newly requested item is cached (ref. Definition 4). In case the cache does not provide enough free storage to insert this item into the cache, an eviction policy determines which item has to be deleted to allow inserting the item to be inserted. In the following, we give an overview of the most established admission and eviction policies. One of our papers [KPR+18] contributes to the following two sections introducing the most established admission (ref. Section 2.2.1.1) and eviction policies (ref. Section 2.2.1.2).

2.2.1.1 *Admission Policies*

In reality, caches within a CDN's delivery network are connected within an overlay and, thereby, build a particular topology. Consequently, a user request can be served by any of the caches on the delivery path. When the item is found in one of the caches, the reply to the user's request can also pass multiple caches on the reply path. Here, an admission policy defines on which of these caches the transferred content contained in the reply is stored. The most common admission policies are introduced in the following:

1. **Leave Copy Everywhere** stores the content on every intermediate cache it passes [LSS04; CTW02]

2. **Leave a Copy Down** stores the content only on the cache that is the direct successor of the cache on the answer path that generated is in possession of the content or, in case none of the caches holds the content, on the cache that is connected to the *Origin Server* [LSS04; LCS06].

3. **Move Copy Down** moves the requested content to the cache that succeeds the cache that serves the content and, thereby, brings the content closer to the requesting user [LSS04; ZLL13].

4. **Probability Admission (Prob)** assigns a probability to each of the caches on the response path that defines how likely it is for the respective cache to store the bypassing content [LSS04; LSS04]. If the content is requested for the first time, it is cached at the cache succeeding the *Origin Server*. One variant of Prob is PProb. Here, for each cache on the reply path, the caching probability is determined based on the cache's distance to the requesting user. Therefore, the probability is chosen proportionally to the number of caches it takes to get to the content source.

5. **NHIT Caching** stores the content if it a request to it has passed the cache at least N-times within a time interval of a specific length. Only the N+1st request in this interval causes the content to be cached. Thereby, the performance-impairing effect of caching content of low popularity [MS15] is reduced. Depending on the content popularity distribution, N can be chosen appropriately. In the course of this thesis, we denote NHIT caching with $N = 1$ as NHIT1 and with $N = 2$ as NHIT2.

2.2.1.2 *Eviction Policies*

Cache eviction policies define which content is deleted from a cache in case new content has to be inserted and no free storage is left. The most popular eviction policies are LRU and Least Frequently Used (LFU), while also extensions such as LFU with Dynamic Aging (LFUDA) [DA99] exist. LRU evicts the content that has not been requested for the longest time, while LFU evicts the content that has the lowest request frequency. In the following, we describe two LRU enhancements that are sensitive to the video popularity distribution: Segmented Least Recently Used (SLRU) [KLW94] and Adaptive Replacement Cache (ARC) [MM04]. Both approaches aim to maximize the hit rate, by splitting the cache storage into two parts.

- **Segmented Least Recently Used** uses a constant storage division between the two parts. Hence, both storage part sizes are constant. We denote the first storage part as the probationary part. It stores content on its first request. If a second request occurs, i.e., a hit in the probationary part, the content is moved to the second cache part, which we denote as the protected part. Thereby, the performance-impairing effect of content that is requested just once can be decreased. If content is evicted from the protected part, it is moved to the probationary part to provide it with a second chance and, thereby, replaces the probationary part's last element in a First In - First Out (FIFO)-fashion.

- **Adaptive Replacement Cache** is an extension of SLRU that introduces a ghost lists, which keeps track of recently evicted content. Initially, the cache storage is split into two cache divisions where each has its own ghost list. The first cache division is a probationary cache, like introduced before for SLRU, which stores content that is requested for the first time. If new content has to be cached and the probationary part is full, the oldest content is evicted in a FIFO-manner and the new content is stored. In this case, the ID of the evicted item is stored in the probationary part's ghost list. Using ARC, ghost lists are limited in the number of items and cannot exceed the size of the corresponding cache division. A cache hit in the probationary part causes the content to be moved inside the LRU cache assuming that it will be requested again soon. In case the LRU cache evicts content, the content ID is stored in the LRU cache's ghost list. If the requested content is stored in the probationary part but its ID is already present in its ghost list, this indicates that the probationary part is too small. Therefore, its size needs to be increased. Consequently, the LRU cache's size needs to be decreased to not exceed the available storage space. Similarly, in case content is moved from the probationary to the LRU cache and is present in the LRU cache's ghost list, the LRU cache is increased and its probationary part decreased in size [Eva14; MM04].

2.2.2 *Cache Hierarchies*

CDNs operate a globally distributed set of caches, which are ordered in a particular topology consisting of one or many levels. Figure 5 provides an overview of typical CDN cache hierarchies. We see that the first level caches are directly connected with the content source on the one side and the second level caches on the other side. It is common that caches communicate only with caches of neighboring levels, but not with caches on the same level [SKL+14; RSB99]. We introduce three different roles present in a typical cache hierarchy. Note that the nomenclature varies in different works.

1. **Content source:** Stores the entire content catalog, but is limited in connectivity. A world-wide delivery using only the content source cannot scale for popular content.

2. **Edge cache:** Is connected to another edge cache of a different layer or directly to the content source.

3. **Leaf cache:** Is connected to one edge cache. In contrast to the other roles, leaf caches are connected with the clients [SKL+14] and serve them.

If any of the caches does not store the requested content, it demands it from a cache that is one level higher in the hierarchy. In case none of the caches on the way back to the content source stores the requested content, it is retrieved directly from the content source.

In the following, we give a brief overview of selected and relevant related works in the area of cache hierarchies. Rodriguez [RSB99] et al. report on the performance differences between distributed and hierarchical web document caching. They found that hierarchical caching has the advantage of shorter connection times. Che et al. [CTW02] present a hierarchical web caching system and model caches as low-pass filters that allow objects with large inter-request times to pass to a cache higher in the hierarchy but absorbing, i.e., storing content that is frequently requested. Laoutaris et al. [LSS04] present three admission policies: Leave Copy Everywhere (LCE), Leave a Copy Down (LCD), Move Copy Down (MCD), and Probabilistic Admission (Prob). In their evaluation, LCD achieves the highest performance and, thereby, outperforms Che's approach [CTW02]. Dai et al. [DHL+12] enhance caching systems to support dynamic request routing. To do so, they design an efficient distributed caching system relying on a realistic IPTV network topology. During their experiments, they notice three key performance influencer: i) topology characteristics, ii) heterogeneous

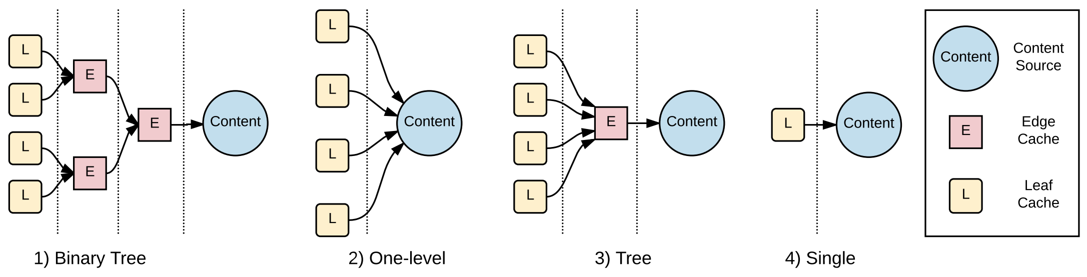


Figure 5: Overview of typical topology components in CDN cache hierarchies [RZS17]

request patterns at different locations, and iii) heterogeneous network capacities. Hu et al. [HGC+15] and Sarkar et al. [SH00] stress that the effort for collaboration and coordination between caches within a hierarchy have to be kept low as they are likely to decrease the performance, e.g., by introducing additional network latency and, thereby, delaying content delivery. Rizk et al. [RZS17] addressed that cache hierarchies are difficult to analyze, though, they have been in use for a few decades by CDNs. Due to the lack of analytical models, caches hierarchies are currently designed in an ad-hoc fashion. To this end, they design a novel model that allows them to estimate the CHR for caches within a hierarchy. Their model is based on single cache models [CTW02; MGL14; GLM16].

2.2.3 *Video Popularity*

For efficient caching, it is important to consider the content popularity. Gill et al. [GAL+07] study content popularity changes on a daily basis since the monthly and longer variations are only marginal. The authors found that popular videos have a high rating on average and a length below the maximum of ten minutes. The popularity of the content also correlated with the user engagement according to Cheng et al. [CMM+14]. Regarding the video metadata, Abhari et al. [AS10] observe only a weak correlation between view count, rating, or video length and the video popularity. Hence, it is hard to determine a video's popularity just by its metadata.

When comparing the local popularity in YouTube's edge network to the global content popularity, no significant correlation was found [GAL+07]. This finding was confirmed in a more recent work of Zink et al. [ZSG+09]. A reason for this observation is that within User-generated Content (UGC) platforms, only little social networking between users happens. Thus, network effects, which affect a broader community do not appear. Further insights on social dynamics of online media sharing can be found in the paper of Halvey et al. [HK07].

User demand patterns are studied by Arvidsson et al. [ADA+13]. Considering the number of repetitive requests, they found differences at city access points and the ones located at a university campus indicating that socially-driven content sharing works in on a small scale. Li et al. [LLX+12] study the propagation of video in Online Social Networks (OSNs). They found that requests coming from an OSN contribute to amplifying the skewness of video popularity: Only 0.31 % of the most popular video content accounts for almost 80 % of all views. However, prediction of popularity from videos shared in OSNs is hard due to the unpredictability of the sharing behavior.

2.3 OVER-THE-TOP VIDEO-ON-DEMAND STREAMING

The three most important stakeholders within the video streaming ecosystem are the content source, the CDN, the ISP, and the end users. In the following, we provide an overview of the major interests of each stakeholder mentioned above.

Playback stalling decreases the user experience

On the user's side, video streaming is a continuous consumption of video data from the video player's buffer. However, this buffer is restricted in size and provides only a limited relaxation of immediate delivery of video data. If the buffer is empty, the video playback stalls temporarily. This is a severe impairment of the QoE perceived by the user. Consequently, the user might leave or abandon the streaming service. Two types of video playback stalling can be distinguished based on the video's playback position.

1. **Initial stalling:** The video pauses when it is started before it starts playing back. This is also known as *startup delay* or *initial buffering delay*. The duration of the initial stalling has been shown to correlate with the probability that a user leaves the video service [Sit13] and, hence, is undesirable.

2. **Playback stalling:** The video can pause during the video playback. For the second type of playback stalling, not only the length but also the frequency impacts the QoE [HSB+13]. Generally, frequent and short stalling decreased the QoE more than one long stalling event [QD06]. However, stalling is generally worse than a reduction of the video quality, e.g., a frame rate reduction [HG08].

Content caching increases the user experience

One way to reduce stalling events is caching, which can contribute to decreased latencies, overcome network congestions, and hence increase the user-perceived QoE in many cases. Cost-efficiency is a major concern of today's content delivery systems as the traffic volume grows about 24% annually [Cis17c]. Therefore, their deployment strategy is driven by bandwidth and energy cost reduction [LWY+12; QWB+09] as well as constant hardware upgrading and reducing peering costs [NSS10]. Thereby, CDNs aim to deliver content in a timely fashion as a standard requirement for streaming services [Lei09; NSS10] since it is a primary influencer of the user engagement [DSA+11]. However, their cache-user assignment is mainly economically driven and, hence, end users are not always being assigned to the cache server whose selection for delivery would result in the highest QoE. CDNs have limited information about the delivery inside ISP networks to optimize their content delivery process. However, they aim to estimate end-to-end characteristics using, e.g., active bandwidth and delay measurements as well as user-side reports to predict the resulting performance [NSS10; KMS+]. Still, this does not provide a guarantee for better service quality and can even worsen existing bottlenecks or create new ones [FPL+13].

ISPs-internal caching is mostly done for ISP-owned services

By providing connectivity between, e.g., CDNs and end users, ISPs have to carry the user's video streaming traffic. Thereby, higher qualities also bring higher network load and costs. Since today's video streaming is done mostly using Hypertext Transfer Protocol Secure (HTTPS), the actual data transferred is not visible to the ISPs. Hence, caching and multicasting schemes are not effective anymore. However, a collaboration between the content providers, e.g., the CDNs and the ISPs in a fashion that the CDNs are allowed to place caches within the ISP's network can severely decrease inter-ISP network load and, hence, potential transit costs. Additionally, it results in a decreased content delivery latency for content cached inside the ISP and, thereby, is likely to increase the user's QoE [SFF+16]. Consequently, this is likely to increase the user's attitude towards the ISP, as for the user it is often not transparent which of the stakeholders within the video streaming ecosystem is responsible for a poor video stream quality.

2.3.1 *Adaptive Video Streaming*

As mentioned in Section 2.2, video dominates the Internet traffic and has become a part of most people's daily life [Eri14]. A video stream typically starts with a video player that requests video data from a content provider, which replies with the requested content in a way that the video player can start the video playback while the video content is still downloading. As the Internet can only offer best-effort delivery, fluctuating available bandwidth has to be considered during video streaming. To this end, the media player has a buffer which can store a certain portion of the video in advance, before the player consumes the video data. More insights about how the video player's buffer affects the QoE can be found in the article from Sani et al. [SME17]. Note that the bandwidth available for the video transmission needs to be higher than the video bitrate to avoid playback interruptions, so-called stalling events, which negatively impact the user experience [HSB+13]. Today's prevalent streaming technique is adaptive video streaming, which we define as follows.

Definition 5: *Adaptive Video Streaming*

A dynamic adaptation of the video stream quality during the transmission via the Internet aiming to utilize but not overburden the available network bandwidth.

The term QoE, in the context of video streaming, refers to the individual perception of the video quality. The International Telecommunication Union ITU's definition [Rec08] based on Brunnström et al. [BBD+13] is given below.

Definition 6: *Quality of Experience*

The degree of delight or annoyance of the user of an application or service. It results from the fulfillment of the user's expectations concerning the utility and / or enjoyment of the application or service in the light of the user's personality and current state.

Note that a technical solution can not optimize some QoE influencing factors such as personal interest in the content, cognitive abilities, and socio-cultural background as well as display brightness, visual abilities, and display-eye distance. However, most of the QoE can be explained by QoS parameters, i.e., bandwidth, delay, loss rate, and jitter [KC10; KLL+08; BSR+06; Ste96; SW97]. Figure 6 presents a taxonomy of QoE influencing factors based on the work of Seufert et al. [SES+15].

The most essential adaptation dimensions for video quality are the image quality, i.e., the encoding quality, the spatial resolution, and the temporal resolution, i.e.,

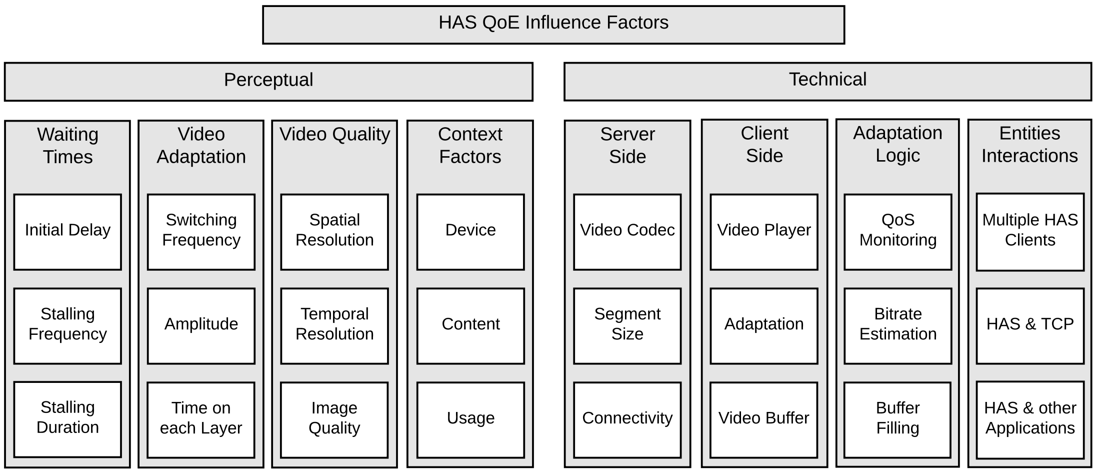


Figure 6: Taxonomy of QoE influence factors in HAS [SES+15]

the frame rate [SES+15; ZHA+10]. Additionally, stalling times decrease the video quality, to an even larger extent than, e.g., the video frame rate [HG08] and negatively impact the rate at which users abandon a video streaming service [Sit13; KS13]. Furthermore, the perceived quality varies between users and between devices. For example, video content of a low resolution can be suitable for a small smartphone display, while on a 4k large TV screen it is considered as low quality. Hence, adaptive video streaming allows also for video delivery in different resolutions depending on the device demands. To assess the quality perceived by the users, typically crowdsourcing [HSH+11] or lab environment assessments are conducted to derive models that allow estimating the perceived QoE based on QoS measurements [HHS+17; AFA16]. In addition to the device, the quality can also be adapted due to a low or unstable bandwidth. While in the case of Constant Bitrate (CBR), users with a lower bandwidth than the video bitrate cannot receive the video in a streaming-fashion, adaptive streaming allows selecting a video quality that meets the user's bandwidth capacities. This can also help in case of an unstable wireless connection, a congested network link, or other apps demanding bandwidth on the user device.

Playback stalling as the most prominent QoE impairment

Several application protocols for multimedia streaming have been proposed during the last decades, e.g., Microsoft Media Server Protocol (MMS)[2] and Real-time Transport Protocol (RTP) [SCF+03]. However, nowadays the de-facto standard has become another group of protocols, i.e., adaptive video streaming, which is based on Hypertext Transfer Protocol (HTTP). Hence, these protocols rely on Transmission Control Protocol (TCP) by definition. Two different basic approaches to implement adaptive video streaming exist: Scalable Video Coding (SVC) and HTTP Adaptive Streaming (HAS). Both approaches slice the video into video segments along the time dimension. Depending on the implementation these segments are between 2 seconds (Microsoft Smooth Streaming[3]) and 10 seconds (Apple HLS[4]) long. The optimal segment size for a video streaming system depends on many influencing factors, whereby bandwidth fluctuation is considered as the most important one [Led15]. On the one hand, if the segment duration is chosen too large, a bandwidth change is

[2] https://msdn.microsoft.com/en-us/library/cc239490.aspx [Accessed: November 19, 2018]
[3] http://www.iis.net/downloads/microsoft/smooth-streaming [Accessed: November 19, 2018]
[4] https://developer.apple.com/streaming/ [Accessed: November 19, 2018]

more likely to cause a playback stalling as quality adaptation can only happen on a per-segment basis. On the other hand, a smaller segment size increases overhead introduced by the segment-encoding, header information as well as more requests traversing the network. Lederer [Led15] demonstrates that for persistent HTTP 1.1 connections the optimal segment duration should be in the range of $2-3$ s. For non-persistent HTTP 1.0 connections, a segment length of $5-8$ s is recommended.

Furthermore, for each segment, multiple quality options are generated which allows adapting the video quality during streaming. Using SVC requires the client to download the base layer, containing the video segments in low quality. If the bandwidth allows, additional enhancement layers can be downloaded to increase the video quality. However, for the enhancement layers, all preceding quality layers down to the base layer are required to allow decoding the video. In contrast to this, using HAS, the different quality layers are independent from each other and only one segment representing the desired quality needs to be downloaded at a time. Today's video services use HAS as a de-facto standard.

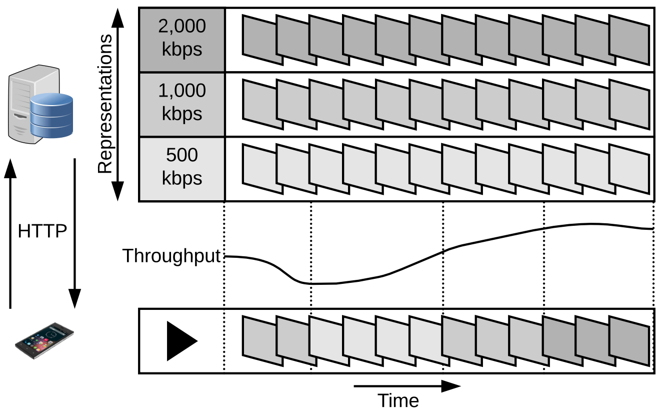


Figure 7: DASH example with three video qualities and fluctuating client bandwidth based on [HSS+15]. Dashed vertical lines mark quality adaptations.

The most prominent HAS implementation is Dynamic Adaptive Streaming over HTTP (DASH). DASH defines how videos should be encoded and in which format the client is informed about the different representations available as well as their locations. This information is encapsulated in a so-called DASH manifest file. Depending on the adaptation algorithm used, the clients select the video segment of a specific quality and download it into their video buffer. Since the adaptation logic is on the client's side, DASH is state-less on the server side. An example of one client's behavior is given by Figure 7. Here, the client adapts only based on the available bandwidth and downloads one of the available video segments at a time. Video quality adaptation algorithms aim for maximizing the user-perceived quality by monitoring the current network conditions, video bitrate, and playback buffer status [HSS+15]. This information is used to determine the video segment which is requested next that will not lead to a playback stalling but of highest possible quality. Note that adaptive streaming mechanisms cannot entirely eliminate the occurrence of video stalling events as they can have many causes, e.g., network congestions, fluctuating network connectivity, or high transmission latencies. More information on recent adaptation algorithms and their user-perceived quality can be found in the paper of Hoßfeld et al. [HSS+15] and the dissertation of Konstantin Miller [Mil16].

Video quality adaptation algorithms

2.4 MACHINE LEARNING

In this thesis, we define Machine Learning as proposed by Andrew Ng [Ng09]:

Definition 7: *Machine Learning*

Machine learning is the science of getting computers to act without being explicitly programmed.

In traditional programming, a programmer designs a program which takes data as an input and outputs a result. Machine learning breaks with this concept by taking data $\mathbf{x}$ and its output value(s) y to learn a program [Dom12] in case of supervised learning which is the most common form of machine learning. The most prominent taxonomy distinguishes three major classes: supervised, unsupervised, and semi-supervised techniques. Supervised techniques require the training of a machine learning model using labeled data. In contrast to this, unsupervised learning does not require such a training phase and can directly be applied to a dataset. Semi-supervised techniques require human intervention in cases where the machine learning model's certainty about the output is low. In the following, we give an overview of supervised and unsupervised machine learning methods as they are relevant for this thesis. Further information on semi-supervised methods can be found in the book of Chapelle et al. [CSZ10].

2.4.1 *Supervised Learning*

Supervised learning and, especially, classification is the most common form of machine learning. To also classify previously unseen observations, machine learning algorithms are required to generalize from provided training examples [Dom12]. This generalization allows judging on previously unseen data to compute the estimated output $\hat{y}$. In case this output is a class, e.g., determining the genre of a music track, this is a classification task. If the estimated output is a numerical value, e.g., the probability that a smartphone is connected to Wi-Fi, this is a regression task. In the following, we introduce these two supervised learning concepts in detail.

2.4.1.1 *Classification*

Classification is the task of deciding to which category or set of categories a certain observation belongs. These classes have to be categorical, e.g., blood groups or if an email is spam or not. To do so, a classifier needs to be trained on observations with known categories, also known as labels. This dataset is denoted as the training dataset. Hence, a classification is a supervised learning task relying on the existence of training data. The observations on which the classifier is trained is usually represented by a feature vector that contains numerical and/or categorical variables. Depending on the training algorithm used, categorical features need to be embedded in a pure numerical feature vector, e.g., by one-hot encoding. After the training, the classification model is able to estimate the true class of a given input. In the simplest case an observation belongs just to one class. In this case, the model outputs a vector containing one likelihood value for each possible class. The estimated class can be determined by taking the highest value of the class values as the classifiers estimate. In case of a multi-class classification, an item can belong to multiple classes. Therefore, the likelihood for each available class is determined. Often, the term likelihood is used interchangeably with the term score. The most popular classification class is denoted as linear classifiers. The name arises from the underlying computation of a so-called score value, expressing the probability that an observation belongs to the model's class k. Thereby, the input vector X_i is multiplied by the coefficients β_k of the learned linear model using the dot product: $\text{score}(X_i) = \beta_k \cdot X_i$.

Linear Classifiers

Decision tree [SL91; HMS66; BFS+84] is a term that comprises two different tree types: i) Classification trees, if the variable to predict is categorical and ii) regression trees, if the variable to predict is continuous. In the following, we will focus on classification trees due to their importance for this thesis and use the term decision tree and classification tree interchangeably. One advantage of decision trees is that the decisions can be visualized in form of a tree and, thereby, are easy to comprehend compared with, e.g., Deep Neural Networks (DNNs) embedding the learned decision logic in a concatenated sequence of weighted sums. The decision tree model is learned in a greedy-fashion by testing which split along all available input dimensions decreases the classification error. Each node in the tree refers to a decision that splits the data into further nodes or a leaf. This decision can be a numerical comparison ($<, >, \leqslant, \geqslant$) or a categorical comparison ($=, \neq$). Typically, leafs are nodes which cannot profit from any more splits. Hence, decision trees are limited by splitting the feature space along individual feature axis in a consecutive fashion.

Decision Tree

Support Vector Machines (SVMs) [SC08; CV95] are also commonly used for classification tasks. They fundamentally differ from linear classifiers by not learning a score function but discriminating hyperplanes in the multi-dimensional feature space. The SVM training algorithm aims to find a decision boundary, i.e., a hyperplane that lies in the gap between observations belonging to the class or not. At this point, this is still a linear classification. However, SVMs can be extended to non-linear classifiers by applying the so-called *kernel trick*. The kernel trick maps the SVM's numerical input to a higher-dimensional space using a kernel function. Thereby, the datasets original dimensions, e.g., x_1 and x_2 are combined into a new dimension z by a kernel function: $z(x_1, x_2)$ which maps the input dimensions to a scalar, i.e., $\mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$. An example is given by Figure 8. Here, the two-dimensional dataset is extended by a third dimension, i.e., the z-axis which is computed using the kernel function $z(x_1, x_2) = x_1^2 + x_2^2$. As depicted on the figure's left side, the dataset is not linearly separable and, hence, no meaningful decision boundary can be learned using a linear classifier. On the right side, the dataset is transformed into $\mathbb{R}^3$ and becomes easily separable by a linear function in $\mathbb{R}^3$, which corresponds to a non-linear function in $\mathbb{R}^2$, i.e., a discontinuous function. Thus, the kernel trick allows learning a classifier that can correctly represent the data-inherent pattern that decides if an observation belongs to the class or not. The distance from the SVM's decision boundary can, thereby, serve as an indication of how confident the classification is. Note that the example given uses just two input dimensions while in reality, the input dimensionality is typically higher. However, applying a vast number of kernel functions to the dataset and, thereby, increasing its dimensionality in the hope that a meaningful kernel function is amongst them is not a feasible solution. The drawback of this method and in general huge feature spaces is that training becomes harder, i.e., the accuracy can be expected to decrease. This phenomenon is known as the *curse of dimensionality*

Support Vector Machines

Curse of dimensionality

and refers to the fact that with increasing dimensions their contained space increases so fast that observations become sparse and dissimilar in many of the dimensions. This renders the training of machine learning models severely harder.

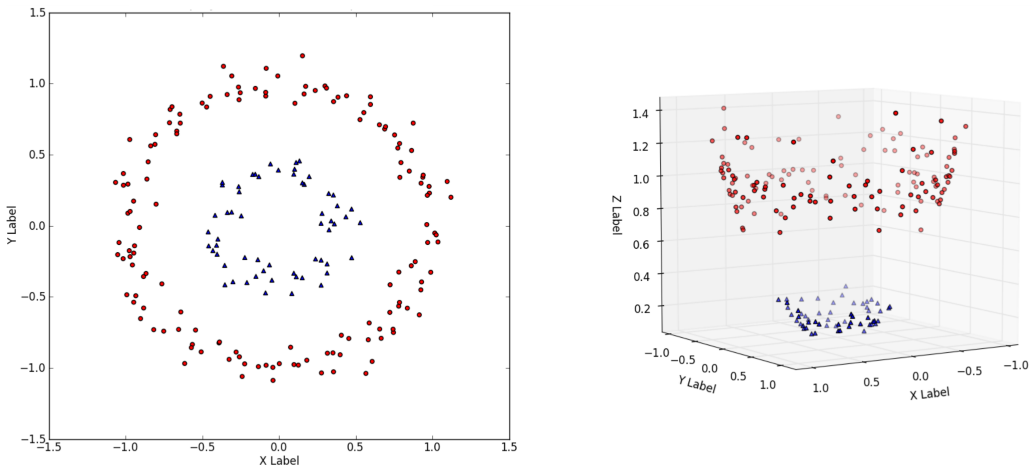


Figure 8: Left: A dataset in $\mathbb{R}^2$, not linearly separable. Right: The same dataset transformed into $\mathbb{R}^3$ by the kernel function $z(x_1, x_2) = x_1^2 + x_2^2$. [Kim15]

2.4.1.2 *Regression*

Regression is the problem of estimating a continuous quantity output for an observation. Some machine learning algorithms can be used for, both, classification as well as regression with small modifications. Examples are decision trees and Artificial Neural Network. However, for some algorithms, it is not as easy or even impossible to be used for both problem types. Some learning tasks can also be transferred from a regression task to a classification task. One example is the prediction of house prices, a regression task, that can be converted into a classification problem by introducing price classes, e.g., luxury, expensive, average, and cheap. A further difference between regression and classification is how the respective machine learning models are evaluated. While classification models can be evaluated using the accuracy, regression models cannot and, instead, use, e.g., the Root Mean Square Error.

Regression vs. Classification

The simplest regression model is the linear regression which assumes a linear relationship between the observation's input variables $\mathbf{x}$ and a single output variable y. Therefore, the model's training tasks compile a search for the optimal linear combination of the input variables to estimate the output. The most common training algorithm for linear regression models is Ordinary Least Squares which aims to minimize the sum of the squared residuals, i.e., the difference between the given correct output and the predicted output. For observations of high input dimensionality, also Gradient Descent is an efficient and common training approach.

Linear Regression

2.4.2 *Unsupervised Learning*

In contrast to supervised learning, in unsupervised learning, no model of the data-inherent pattern is learned and only unlabeled input data is provided. Instead, the learning is based on the data provided and seeks to determine how the data is organized.

2.4.2.1 *Clustering*

Clustering or cluster analysis divides a given set of observations into clusters, i.e., subsets of similar items. Most common is that an item can only belong to one cluster which is called hard clustering. Less popular are soft clustering approaches which assign each observation for each cluster a probability to belong to it. Thereby, the clustering algorithm works without prior knowledge about potentially existing clusters or what is expected. Hence, clustering is a method for knowledge discovery and provides insights into data-inherent groupings. Examples of clustering tasks are customer segmentation and anomaly detection. The three most used clustering model types are introduced and explained in the following.

- **Connectivity models:** Observations close to each other are considered to belong to a cluster. The cluster creation is done in one out of two ways. The first option is to assign each observation to a cluster and aggregate them based on the smallest distance. The second option is to assign all data points to a single cluster and split it iteratively based on the largest distance between the resulting clusters. An example of this clustering type is hierarchical clustering.

- **Centroid models:** As the name suggests, these type of clustering models assigns observations around centroids in the dataset based on the distance to the centroid. This is done iteratively until all observations are assigned to one of the clusters. The inherent weakness of this approach is to determine good centroids. Furthermore, the number of cluster centroids k has to be provided as an input and, hence, prior knowledge of the dataset is required. The most prominent example of this class is k-means clustering. Notably, mechanisms to determine an optimal k exist, e.g., X-means [PM00] and k-means++ [AV07].

- **Density models:** These models identify regions of high density, i.e., a large share of the observations and assign remaining observations by their distance to the density-based clusters. The most prominent algorithm in this class is Density-based Spatial Clustering of Applications with Noise. This algorithm performs optimal for many observation distributions. A visual and intuitive evaluation of a variety of clustering algorithms can be found here[5].

2.4.3 *Music Classification*

In parts of this thesis, we rely on existing methods of music classification and apply them in the context of proactive music video caching. Therefore, we present a brief overview of relevant works in the areas of music mood and genre classification.

Music classification aims to assign a label from a pre-specified set of labels to a music track referring to, e.g., the track's genre or mood. For many tracks, such labels can be retrieved from online music databases, e.g., from last.fm[6]. However, for very recent tracks, this information is unlikely to be available as these databases rely on tags assigned to the tracks by users. Thus, automated classification is preferable as it can also assign labels to new tracks. To this end, we use a set of music tracks together and their known labels to train a classifier that is able to estimate the

[5] `http://scikit-learn.org/stable/modules/clustering.html` [Accessed: November 19, 2018]
[6] `https://www.last.fm` [Accessed: November 19, 2018]

respective labels. Most existing research in the area of music information retrieval aims at genre or mood classification. The features on which the classifier is trained are traditionally characteristic audio features such as the Mel Frequency Cepstral Coefficients (MFCC) [Log00]. Wang et al. [Wan+13] evaluate the impact of different acoustic feature sets regarding classification accuracy. Therefore, they extracted features from, both, the signal processing and musical dimension using MIRtoolbox [LTE08]. Next, they train SVMs on different feature subsets. A combination of, e.g., Rhythm, Timbre, and Tonality achieves 79.5% accuracy using a labeled dataset of 1,000 tracks [Stu13]. Note that the classification accuracy strongly depends on the experts who annotated the training dataset and its size. The spectrum for, e.g., mood classification ranges between 25% [GS15] and 90.44% [Lau+10]. Recent works use neural networks to derive music features and show a more stable and higher performance compared to traditional approaches. This way, Costa et al. [Cos+17] achieve 87.4% accuracy. Choi et al. [Cho+17] present a transfer learning approach for music classification using ConvNets. These networks take the music's Mel spectrograms as an input, which efficiently approximates human auditory perception. The training uses several publicly available datasets and aims at music tagging. However, the resulting 160-dimensional feature vector of their network is useful for any music information retrieval task. The authors use this feature vector as an input for music genre classification and achieve 89.8% accuracy. Hence, we conclude that neural networks are in general more powerful to retrieve expressive music features.

Framework Name	#Descriptors	Last update
MPEG-7 descriptors [KMS05]	17	2004
Marsyas [TC02]	30	2015
jAudio [MMF+05]	40	2009
MIRtoolbox [LTE08]	55	2014

Table 1: Music feature extraction frameworks

In most of the related works discussed, feature extraction frameworks are used to determine low-level features. Most commonly used frameworks in the scientific literature are depicted in Table 1 together with the number of audio descriptors, and the year of their last update.

To classify the emotion or genre of a music track, the following procedure is widely used: First, a representative sample of the music file is selected, e.g., seconds 30-60 to avoid the often not representative, intro. Second, the loudness is normalized to make loudness-sensitive metrics comparable between different tracks. Third, music features are derived using one of the feature extraction frameworks shown in Table 1. Fourth, a machine learning model, e.g., a Support Vector Machine (SVM) is trained which is able to classify a mood state based on music features.

Genres are well-defined classes to which a music track can belong to. In contrast to this, mood states are more complex to represent. One widely accepted model representing mood states is Thayer's mood model [Tha90]. As depicted in Figure 9, the model defines a mood by a point in a 2D coordinate system with a certain intensity of valence on the x-axis and arousal on the y-axis. The arousal value is defined as the intensity of the emotion, while the valence value refers to how positive or negative

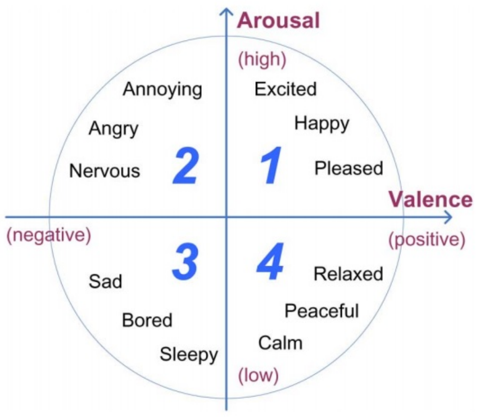


Figure 9: Thayer's mood model [YLS+08]

the emotion is perceived. For different domains, variants of this model have been derived. A typical representation is depicted in Figure 9.

Music mood classification

Yang et al. propose a fuzzy approach for music emotion recognition [YLC06]. Fuzzy classifiers are characterized by not only computing the most probable class but returning a fuzzy vector containing the probability of the music sample belonging to each of the classes. They evaluate two classifiers: fuzzy k-nearest neighbors and fuzzy nearest mean. Therefore, a dataset containing 195 popular music samples of 25 seconds length is used. Each sample is manually annotated with a mood. The four mood classes considered represent the four quadrants of Thayer's mood model. Additionally, to track mood changes within a song, a track is split in 10 seconds samples overlapping by $\frac{1}{3}$ of the previous sample and the mood for each sample is evaluated. The results show that a fuzzy nearest mean classifier with an accuracy of 78.33% is superior to fuzzy k-nearest neighbors.

Trohidis et al. [TTK+08] extend Thayer's mood model by two dimensions: pleasant/unpleasant and engaging/disengaging. The authors select a multi-label classification approach, using six labels: amazed-surprised, happy-pleased, relaxing-calm, quiet-still, sad-lonely, and angry-fearful. The dataset used consists of 593 expert-annotated songs from the genres Classical, Reggae, Rock, Pop, Hip-Hop, Techno, and Jazz. These genres serve as features for their multi-label classifier evaluation. As classifiers, they choose binary relevance (BR), label powerset (LP), random k-labelsets (RAKEL), and multilabel k-nearest neighbor (MLkNN). Among these classifiers, RAKEL provides the highest prediction accuracy with 80% but different accuracies for each class.

Laurier et al. [LMS+10] categorize music into one of Thayer's mood model quadrants by using SVM, Decision Trees and Random Forests, k-nearest neighbors, logistic regression, and Gaussian mixture models. Out of the chosen models, an SVM with polynomial kernel achieved highest mean accuracy of 90.44% for the categories angry, relaxed, and sad.

The highest accuracy is achieved by [LMS+10] compared with [RHH09; HRJ+10; YLS+08; SDP12], even though they also use SVMs. Table 2 summarizes mood classification works showing their best performing algorithm and the corresponding accuracy. Most works achieve a high accuracy by manually annotating music tracks and using a small training set in the range of a few hundred tracks only. However,

Paper	Algorithm	Accuracy
Laurier et al. [LMS+10]	SVM with polynomial kernel	90.44%
Trohidis et al. [TTK+08]	Random k-label set	76–90%
Rho et al. [RHH09]	SVM with radial kernel	87.8%
Han et al. [HRJ+10]	SVM with polynomial kernel	87.78%
Eerola et al. [ELT09]	Partial least squares regression	75–85%
Yang et al. [YLC06]	Fuzzy nearest mean clustering	78.33%
Yang et al. [YLS+08]	SVR	58.3%
Song et al. [SDP12]	SVM with polynomial kernel	54%
Gillhofer et al. [GS15]	Random Forest	25%

Table 2: Music mood classification approaches and the best performing algorithm per paper

[GS15] achieves only 25% accuracy by using on average ten mood classes. Therefore, the classification performance is assumed to vary strongly depending on the data and algorithm used as well as on the number of classes chosen.

Music genre classification

Gillhofer et al. [GS15] collect a dataset of 7,628 listening events to 4,149 music tracks, obtained through a mobile app provided in the scope of a user study. For each event, time, location, weather, device, network, and motion are logged. Additionally, they acquired the genre and mood information for each track from last.fm. For genre classification, k-NN, decision tree and random forest, rule learner, and ZeroR are evaluated. They achieve an accuracy of about 60% for the genre prediction using a decision tree approach.

Huang et al. [HLW+14] propose a genre classification system using separate feature-selection for each genre class. The features used are intensity, pitch, timbre, tonality, and rhythm. For each pair of two genres, a local feature set is derived by their self-adapting harmony search (SAHS) algorithm. To get accurate results even for ambiguous genres, multiple one-against-one SVM classifiers are trained. The final classification is computed by a classifier ensemble containing the SVMs mentioned above. Evaluations of multiple strategies are conducted on the GZTAN dataset[7] published 2002. They achieved an accuracy of 97.2% for ten different music genre classes. This is a 13% increase compared with just using the original feature set.

To the best of our knowledge, the work presented in this thesis is the first deriving audio features from a recent dataset containing video requests from a mobile network to YouTube. The data used for genre classification is more recent than, e.g., the GTZAN dataset used by Huang et al. [HLW+14] and considerably larger with over 4 million requests, than the datasets used in the related work for mood classification.

2.5 SOFTWARE-DEFINED NETWORKING

Traditional network equipment such as switches and routers are often distributed and heterogeneous which makes them hard to configure as vendor-specific commands have to be used. To do so, a specialist needs to configure each device involved in, e.g., a firewall or a load balancing network service separately. The reason for this

[7] `http://marsyasweb.appspot.com/download/data_sets/` [Accessed: November 19, 2018]

is that both the control and the data plane are implemented in each device. Here, the forwarding plane refers to fast packet processing and forwarding, i.e., switching. The control plane defines where a packet has to be forwarded by implementing, managing, and updating routing logic, e.g., a specific routing protocol. As each vendor provides different control logic elements, no common control primitives and control Application Programming Interface (API) exists. However, this is a requirement to allow for innovation in IP networks. We provide an abstract example of an SDN-enabled network in Figure 10.

This is exactly where SDN improves upon traditional network architectures, offering a common API and further innovation-fostering features. We define SDN in this work following the definition of Kreutz et al. [KRE+15].

Definition 8: *Software-defined Networking*

Software-defined Networking (SDN) is a network paradigm based on four pillars: i) decoupling of control and data plane, ii) flows as a key concept for packet forwarding, iii) control logic is moved outside the forwarding devices, and iv) programmability of the network.

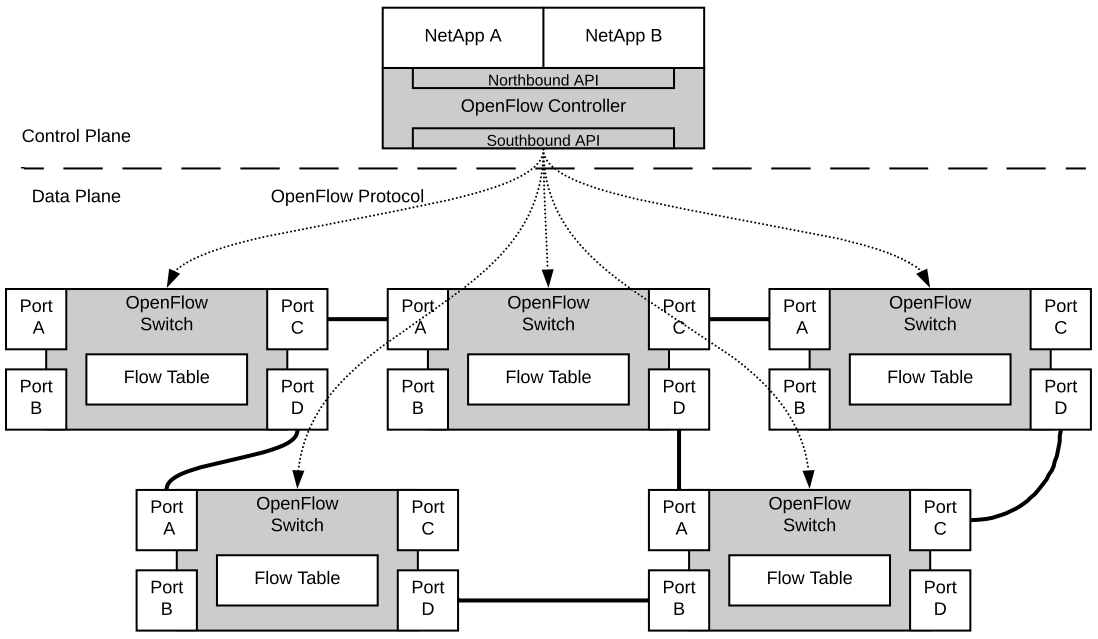


Figure 10: Major components of an OpenFlow-supporting network

The first pillar is the decoupling of the control logic from network devices. Hence, SDN assumes networking devices to be pure forwarding devices without own local control logic at the devices. To this end, the data plane has to offer a standardized interface to allow receiving control signals from an external controller. One of the most popular standardized interfaces nowadays is OpenFlow.

Decoupled control and data plane

Traditional network devices make forwarding decisions on a per-packet basis. In contrast to this, SDN uses the flow concept for packet forwarding decisions. Here, a flow is defined by a so-called matcher which comprises a set of header values

Flow-based packet forwarding

or value ranges. All packets matching a flow can be applied to one or a set of actions, e.g., to forward it to a specific port, rewrite protocol fields, or drop the packet. Two exemplary actions are packet duplication and replacement of the destination IP address.

The control plane consists of one or more SDN controllers, i.e., a software that may run on commodity hardware. The controller communicates with the SDN-enabled switches by its southbound API, while providing access to NetApps, i.e., control applications by its northbound API. The actual network logic resides in the control applications which use well-defined abstractions and primitives offered by the controller. Therefore, the controller has to translate these commands to the respective SDN flow entries and implement them in the involved SDN switches. Furthermore, the controller takes care of security tasks such as NetApp isolation, deciding how to handle previously unseen flows, and how to resolve conflicts between NetApps.

External control logic

By doing so, the network becomes programmable. Foremost, network applications have two benefits. First, they lower the costs of network management as the network can be controlled in a standardized and automatic fashion. Second, standardized programming interfaces allow for innovative network applications, abstracting from different vendors and hardware, thereby allowing innovative network software that would be hard to realize without SDN.

Programmable network

2.5.1 *OpenFlow*

The OpenFlow protocol is the most popular implementation of the SDN paradigm. The following section is based on the OpenFlow protocol specification [Ope15] standardized by the Open Networking Foundation (ONF). The core element of this specification is an OpenFlow switch. However, the standard only presents concepts and ideas but does not provide information on how to implement them.

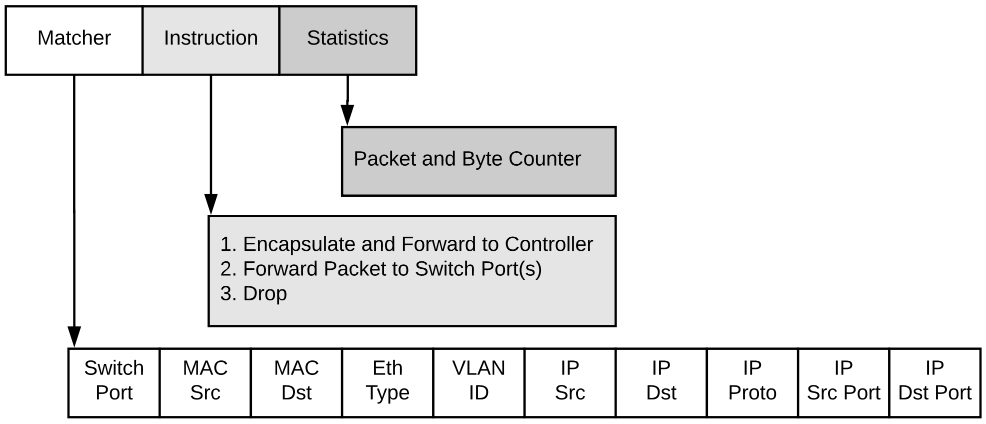


Figure 11: Conceptual components of an OpenFlow flow entry

Simplified, an OpenFlow switch contains a number of flow tables and network ports. A flow table contains flow entries which are set by the OpenFlow controller and define how incoming packets are handled. Figure 11 provides a conceptual overview of an SDN flow entry. Each flow consists of three components: a matcher, an instruction, and statistic counters. The controller can install these entries, modify them later on, and read their statistic counters by communicating via the OpenFlow protocol with the switch. Upon incoming packets, an OpenFlow switch tries to find a matching flow entry by comparing the packets' header field with the flow entries in its flow tables. If a matching flow entry is found, one or several actions are applied to the packet, e.g., forwarding it to a specific port, modifying its header fields or dropping it. OpenFlow has evolved considerably between the first version 1.0 [Ope09] and its current version 1.5.1 [Ope15]. While in version 1.0 only a single flow table per switch exists and the set of actions and instructions are limited, OpenFlow 1.5.1 offers plenty of features and performance improvements. For example, multiple concatenated flow tables and group tables allow storing flow entries much more efficient than a single flow table. Additionally, a richer set of matchers and more actions are available.

2.5.2 *SDN in Hardware*

It is crucial to guarantee a packet forwarding performance close to line speed and not to slow down the forwarding process by OpenFlow unnecessarily. Thus, a fast and efficient technique is required to match, process, and forward packets to determine if they belong to a particular flow. In today's SDN-enabled hardware switches, fast packet matching is allowed by Content-Addressable Memory (CAM). This special memory allows using arbitrary addresses to access memory, a concept similar to HashMaps known from programming languages. If a packet has to be matched, all memory entries are searched within one clock cycle and the result is returned [ACS03]. While CAM allows a lookup in constant time, inserting new data is comparably slow, similar to HashMaps. Hence, using CAM is avoided in case of the flow statistic counters, which are instead stored in the switch's Static Random-Access Memory. CAM has an additional disadvantage as it allows only for exact matching but not partial matching. Therefore, tasks like matching a subnet that includes wildcards are not possible. Therefore, modern SDN switches store their flow tables in Ternary Content-Addressable Memory (TCAM) which allows setting specific parts of a key as "don't care" values, which allows partial key matching. At present, optimizing TCAM memory usage, energy consumption, and manufacturing costs are on-going research challenges [ACS03].

2.6 MULTI-MECHANISMS TRANSITIONS

The concept of multi-mechanism transition used in this thesis is taken from the definition provided by the members of the Collaborative Research Centre "MAKI"[8] [Ric17; Wik16; SHK+15; FHK+16]. Transitions are a paradigm adapting a communication system's service or protocol components. Hence, a group of protocols or services that offer the same functionality is required. This group, offering exchangeable functionality, is denoted as a multi-mechanism [Wik16]. The transition between these mechanisms is initiated by a context change which can be, e.g., varying bandwidth, changing requirements, or different QoE needs and aims at a stable system performance. To this end, the mechanisms which perform best for a given context is chosen. The performance can be for example the user-perceived QoE, fairness, or service stability. However, costs must also be taken into account when selecting the optimal

[8] https://www.maki.tu-darmstadt.de [Accessed: November 19, 2018]

mechanism. Thereby, costs depend on the application scenario, available resources, and environmental constraints. If the environment changes often and rapidly, the mechanism transition may not pay off and be kept constant to avoid continuous and potentially costly oscillations between mechanisms and their potentially high costs. In the scope of this thesis, transitions occur in the exchange of predictive models in the scope of Prefetching systems (ref. Section 4) and when considering different proactive caching mechanisms in the presence of varying cache storage sizes (ref. Section 5).

3 RELATED WORK

In this chapter, we discuss the related work of the contributions proposed in Chapter 1. Hence, the related work chapter presents three main parts. Section 3.1 discusses the related work on mobile video prefetching. In Section 3.2, we discuss recent works in the area of proactive caching, and Section3.3 presents the related work on SDN-based multicasting of Video-on-Demand (VoD) content. In each section, we first identify, describe, and discuss design goals from existing works in the context of to the research challenges presented in Chapter 1. Subsequently, we detail and discuss individual works that are most relevant to the three main parts of this thesis.

3.1 MOBILE VIDEO PREFETCHING

Prefetching is a technique applied in a variety of application areas, e.g., memory architectures [Smi82], file systems [KE93], databases, cloud computing, and distributed systems [KS92]. The commonality between prefetching in the context of these application domains is that they anticipate the need for data in advance to load it speculatively into a storage area, i.e., a cache, that is closer to the location where the data might be eventually needed. We define mobile video prefetching as estimating and downloading a user's future demand for VoD content. If the estimation is correct, the requested content is provided on request without the need to use the potentially low-quality cellular Internet connection.

3.1.1 *Taxonomy*

Figure 12 depicts the taxonomy of mobile video prefetching used in this thesis. Here, the first level (light gray) of the taxonomy tree depicts the *Optimization Goals*, *Candidate Source*, *Candidate Selection*, *Download Scheduling*, and *Caching*. In the following, we discuss these categories and their manifestations (gray).

3.1.1.1 *Optimization Goals*

The idea of mobile video prefetching is to serve videos from local storage to avoid an unstable Quality of Service (QoS) caused by a potentially low-quality cellular Internet connection. We characterize a low-quality mobile connection by low bandwidth, high delay, or even interruptions. These QoS parameters have a direct impact on the Quality of Experience (QoE) as we detail in the context of video streaming, in Section 2.3.1. Under these circumstances, prefetching can avoid the drawbacks of such a connection by fetching the data in advance instead of on-demand. Note that low-quality mobile connectivity is not the only reason for decreased video quality or playback interruptions as they also occur when using fixed-network Internet connections, e.g., when being connected over Wi-Fi [SSF08]. The goal of prefetching is

QoE

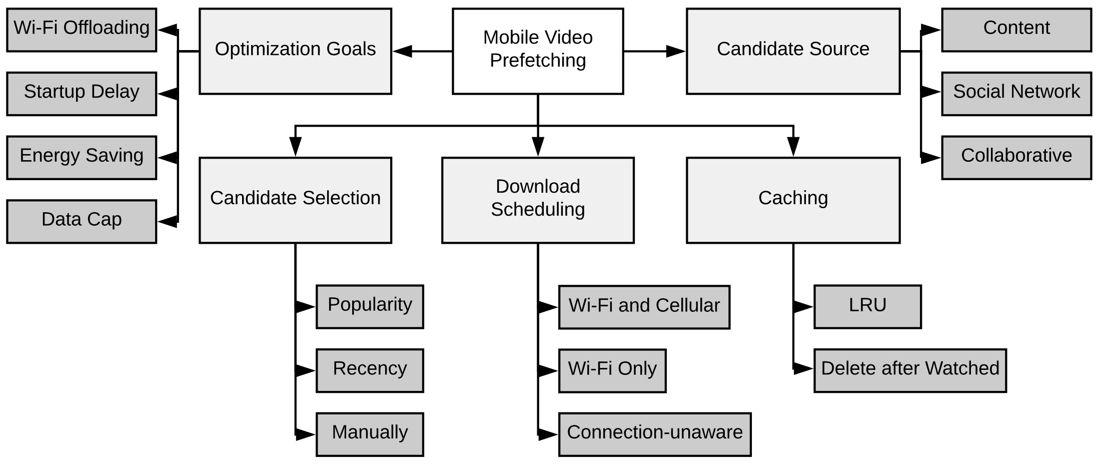


Figure 12: Taxonomy of mobile video prefetching mechanisms

to deliver video content fast and stable to the video player. This implies a low initial stalling as it might occur by using a low-quality mobile connection.

Energy saving

Mobile video prefetching allows to move data transmissions from the cellular network to Wi-Fi networks and, thereby, enhance the battery lifetime. This is commonly known as Wi-Fi-offloading. A mobile prefetching system has to consider the network capabilities as well as their energy properties to minimize its energy footprint. Using a cellular network connection is more energy consuming than using a Wi-Fi connection since the transceiver consumes more power. Furthermore, the device's 3G/4G transceiver stays in a high power-consuming state for a short period after the last data has been transmitted in anticipation that the next transmission begins soon. Thus, when using a cellular network, the connection to the base station is kept open longer than the actual data transfer takes. This additional time is denoted as the tail time [PHZ12] and additionally leads to energy consumption even though no data is going to be transmitted. The negative effect of this behavior can be mitigated by aggregating delay-tolerant traffic and send it as a batch [QWG+10]. Data transmissions that use Wi-Fi have often more stable network conditions and tend to be much more energy efficient. Gross et al. [Gro+13] investigate accurate energy models of smartphones and conclude that transmissions using the mobile 3G connection are one magnitude less energy-efficient than Wi-Fi transmissions. These results are still valid for state-of-the-art Long Term Evolution (LTE) networks. Huang et al. [HQG+12] measured the performance and power characteristics of LTE networks. They conclude that the client-side energy consumption is up to 32-times higher compared with Wi-Fi. Gautam et al. [GPN13] report an energy saving of up to 84% when using Wi-Fi instead of 3G and up to 98% compared to streaming via 2G.

From these results, we deduce that the energy demand between cellular and Wi-Fi transmissions cannot be expected to converge in the near future, even though this is part of the vision for future 5G networks.

To allow simulating regarding mobile device energy consumption, energy models which are in large parts device-specific have been proposed. One popular [HFG+12; ZAC+13] source for smartphone energy models is PowerTutor [ZTQ+10]. These models allow calculating the cost of downloading data over a certain network interface and, therefore, can be used as a valuable input for prefetching heuristics.

Mansy et al. [MAC+14] investigate the behavior of mobile streaming apps on iOS and Android in combination with the video players provided by YouTube, Netflix, and Hulu. They observe that redundant traffic causes up to 15% of traffic in video streaming sessions. This is caused by video quality adaptation due to varying network conditions. Hence, prefetching a single quality on a mobile connection can reduce energy consumption in case the video is requested later on.

Data cap

In some countries, there are only limited mobile data contracts available and, when unlimited rates are available, they are usually the most expensive tariff option. Prefetching is speculatively and can deplete the mobile data volume severely if not considered as a limited and costly resource. To this end, Wi-Fi offloading is preferable to using a cellular connection. Though, not every terminal may connect to a Wi-Fi regularly or too late to achieve an efficient prefetching performance.

The optimization goals mentioned before strongly correlate with the precision and the recall of the prefetching mechanism. Therefore, they are commonly used in the evaluation of mobile prefetching systems. They are superior to the Cache Hit Rate (CHR), which is commonly used to evaluate web caching since prefetching a single content that is afterward being requested already achieves the maximum CHR.

3.1.1.2 *Candidate Source*

Prefetching mechanisms need be informed about new content that becomes available. Content portals like YouTube can serve this purpose. Here, the subscribed channels or videos marked by the user for later watching can serve as relevant sources of video content. Similarly, Online Social Networks (OSNs) like Facebook, Twitter, or Instagram can serve this purpose. Here, content is posted by friends, colleagues, acquaintances, or groups and, thereby, shown to the user on his news feed. The two previously mentioned sources, both, have a certain familiarity for the user who knows the source from which the content comes personally. It has been shown that about 45% of all videos requested are selected from the related video list [KZG13] of videos watched before. Furthermore, 80% of these videos belong to the top 10 positions of the related video lists. A third approach, Collaborative Filtering (CF), identifies video content that the user might be interested in, even though he has no direct connection to the content source or the person sharing the content. The idea of CF is to identify users that show similar video preferences as the user at hand and consider content watched by these users as likely to be of interest to this user.

3.1.1.3 *Candidate Selection*

The video content available from the candidate sources have a set content and social properties. Examples on which content can be selected are the source, the topic of the content, the content's category, and interactions of friends with the content. Though, the features which determine if a content is especially appealing can vary among different persons. Generally, four different sources can serve the prefetching mechanism with video candidates [GPN13]:

- Automatic selection by an algorithm running on the user's device
- Manual selection by the user
- Selection by the content provider

- Selection by the network operator

Among the presented candidate sources, an automatized content selection on the user's device is most convenient and privacy preserving.

If the user selects the content manually, the CHR is likely to be close to 100% since the user is likely to select content he is interested in. The major drawback of this is that the user is burdened with more effort. Besides, contents on OSNs are often so important for the user that it is not acceptable to select them for prefetching first and watch them later. The user may want to consume the content immediately, to be informed and able to communicate with his peers about the content.

The content provider and the network operator have the ability to apply machine learning approaches like CF to a vast number of users. This allows determining similarities between users and making predictions based on content that a user similar to another user has already watched. Furthermore, the network operator has the global knowledge of his entire network which allows him to move data traffic to times of low network utilization and thereby achieving higher transmission bandwidth, less packet loss, and lower energy consumption. Gouta et al. [Gou+15] present a prefetching mechanism denoted as *CPSys* for mobile video prefetching. Inspired by Google's page rank algorithm, they propose using CF to predict videos for users based on user similarity according to the videos requested.

On the one hand, an automated content selection is the most desirable method, since it can happen in the background without introducing effort for the user. On the other hand, automatic prefetching has the drawback that it is speculative and a 100% CHR is unlikely to be achieved. Hence, cost and benefit of prefetching mechanisms have to be carefully evaluated.

In the following, we focus on prefetching mechanisms which run on the client's device which covers the majority of mechanisms. Furthermore, it respects the user's privacy requirements and is more convenient than a manual selection.

3.1.1.4 *Download Scheduling*

Especially for mobile devices, the potential rewards achievable with prefetching are high according to Higgins et al. [HFG+12]. Downloading video content during off-peak hours can alleviate carrier networks during times of high load.

In case a non-prefetched item is requested, the request time can be simplified modeled as $T_{fetch,t} = \frac{S}{BW_t} + L_t$, where S is the data size, BW_t is the bandwidth at time t, and L_t is the one-way channel delay, at time t, needed by the content request to arrive at the content source. Knowledge about the connectivity over the course of the day can be used to choose an optimal time for prefetching. Nicholson and Noble [NN08] follow this principle and forecast the cost of prefetching for a certain time in the future $t_{download}$, with the following condition: $t_0 \leqslant t_{download} \leqslant t_{watch}$. Here t_0 is the time when the system becomes aware of the data item, that might be requested in the future and $t_{download}$ is the time when the download occurs. There are also approaches using the average observed network capabilities, i.e., availability, bandwidth, and latency, as a baseline for the future. To this end, measurements have to be made. Ideally, these measurements are obtained passively, which determine the current network conditions without transmitting additional data. Keep-alive messages and small data packets are sent quite often on modern smartphones and, therefore,

Timely prefetching

might be used for this purpose. However, Transmission Control Protocol (TCP) does not leave the slow start phase when transmitting small data; hence these measurements are likely to be biased. Therefore, active measurements often provide a better quality of measurements.

3.1.1.5 *Caching*

Low storage utilization

Prefetched videos need to be stored locally. Thus, an imprecise mechanism would unnecessarily occupy storage on the client device. However, it is unrealistic to assume a perfect prefetching approach that only stores content that is requested by the user at the right time. Therefore, video content needs to be stored locally until it becomes unlikely to be watched or more promising video candidates become available. In the case of smartphones, large storage capabilities are common nowadays. Though, in the case of smartwatches, which are also capable of playing videos, storage is still a limited resource.

3.1.2 *Discussion of Selected and Representative Work*

In the following, we present and discuss selected and representative works in the area of video prefetching. While the focus of this thesis is on mobile prefetching, approaches from stationary clients, i.e., PCs are closely related. To make this distinction clear, we first present traditional approaches for stationary devices and, second, prefetching mechanisms tailored for mobile devices.

How much of a video to prefetch?

A fundamental question in prefetching is how much of a video, i.e., typically measured by the number of video segments should be prefetched. Wang et al. [WSY11] propose prefetching a fixed chunk length of 1200 bytes. However, a fixed byte size is not applicable in Dynamic Adaptive Streaming over HTTP (DASH)-based streaming platforms as they deliver several video qualities in the form of segments that last 2-10 seconds (cf. Section 2.3.1). Hence, a more applicable solution is prefetching a certain number of video segments instead of bytes. Khemmarat et al. [KZK+12] propose using prefix-prefetching. Based on the available bandwidth, the video duration, and the video bitrate, the prefix size is determined dynamically. Therefore, the prefix size is chosen in a way that allows downloading the rest of the video, i.e., the not yet prefetched part, if a user begins to watch the video.

Social network-based mechanisms

OSNs play a vital role in the distribution of video content. In 2009, 25% of the YouTube views came from person-to-person sharing in OSNs according to Broxton et al. [BIV+13]. The authors note that 90% of the total views are caused by 2% of the most popular videos in a Facebook-like OSN. This characteristic can be observed on many video platforms, e.g., the Korean OSN Daum (cf. Section A.5), and is known as the short-tail/long-tail distribution, i.e., the popularity distribution can be described as a power-law function (cf. Section A.5). Further research on the distribution of content through OSNs can be found for Twitter [RBC+11], YouTube [SMM+11], for Facebook [SRM+09], and Flickr [CMG09]. One interesting finding is that, e.g., images on Twitter, even the popular ones, spread significantly slower than video content [LWL+13]. This stressed the importance of VoD content for the Internet and for this thesis. On OSNs, there is a vast number of potentially interesting videos for a user, which is hard to predict. However, using OSN information has proven to be predictive of a user's watching behavior [ZDW+13]. Sedhain et al. [SSX+13] propose a content filtering mechanism, denoted as social affinity filtering, which is able to select promising photos and videos from the users' OSN. They investigate how predictive particular OSN features are, e.g., likes, comments, and tags. In addition, groups, pages, and favorites are investigated separately. One of their key insights is that interactions on videos are most predictive. Especially, a user's participating in small online groups are predictive for long-tail content. Wilk et al. [WRT+15] conduct a Facebook user study. They investigate social-aware multimedia prefetching by providing a smartphone application denoted as *SonNet* to their participants. They found that video watching behavior is quite heterogeneous across the participants and do not find a single predictive feature that would allow efficient prefetching for all participants. Therefore, a video's like and comment counts, as well as the content coming from a 1-hop Facebook friend alone, are not sufficiently predictive features. Paul et al. [PPW+15] confirmed this results using a browser plugin [1] that monitored Facebook user behavior within the scope of a user study.

Prefetching in P2P systems

Cheng and Liu propose *NetTube* [CL09], which uses an additional Peer-to-Peer (P2P) overlay [SW05] per video to support the video content distribution. Thereby, they focus on prefix-based prefetching to support a fast video playback start. They focus on increasing the user experience, which has shown to be sensitive to stalling events at the video playback start [HSK14]. Li et al. present *SocialTube* [LSW+12], a P2P video prefetching mechanism that focuses on social relationships. The authors observe that 90% of the video views can be explained by direct or 2-hops friends of the requesting user. *SocialTube* pushes a prefix of a newly published content of a user to all of its follower. If a follower starts the video playback, the remaining video content is retrieved in a BitTorrent [Coh03] fashion. Wang et al. [WSY11] rely on the Chinese Facebook variant *RenRen* to analyze the QoE gain achievable by using P2P-assisted video streaming. Their proposed mechanism strives to reduce the initial video stalling time similar to *NetTube* and, hence to increase the user-perceived QoE. Therefore, the authors argue to prefetch the first video segments since they are most likely to be requested. The prefetching candidates are determined using the social relationships between *RenRen* users and their preferences.

Prefetching related videos

Khemmarat et al. [KZK+12] propose two novel YouTube-specific prefetching approaches. The first approach uses a user's search results to prefetch the videos appearing in a search result on, e.g., YouTube, to prefetch their prefixes. The second approach uses the recommended videos of a currently watched video to prefetch prefixes of these. Thereby, the recommender system of YouTube is leveraged by their system. The authors demonstrated that prefetching videos retrieved from the related video list of an already watched video can efficiently offload networks when applied on network proxies and also on mobile devices. Plecsca et al. [PCO16] propose a similar mechanism which also determined prefetching candidates from the related video list of already watched videos. To this end, they use Markovian policies fed by the YouTube video recommendation list. However, when a user currently watches a video, it might be impossible, in case of, e.g., a low-quality mobile connection, to prefetch videos from the related video list in parallel. Furthermore, this might impair the quality of the currently watched video as they share the same mobile connection.

Prefetching for mobile devices

In the following, we present selected works that focus on prefetching for mobile devices specifically. Gautam et al. [GPN13] present an Android application that can

[1] http://www.daniel-puscher.de/fpw/ [Accessed: November 19, 2018]

download videos from a user's OSN feeds. The prefetched videos are presented to the user by the app. Thus, it is difficult to compare this approach with a transparent prefetching mechanism that does not change the look and feel of the native OSN feeds. Finamore et al. [FMG+13] investigate the exploitation of the wireless broadcast channel in 3G/4G networks. Based on this broadcast mechanism, they aim at prefetching popular content to mobile devices. The authors discuss if the estimated benefit is worth engineering such a system and use a real operator trace for their evaluation. However, the performance of their system is poor and the authors argue if such a system is worth the engineering effort. Zhao et al. [ZDW+13; DZW+14] design a middleware for mobile devices denoted as Offline Online Social Media (*O^2SM*) middleware. It estimates prefetching candidate videos by taking the user's Facebook feed as an input. They use commenting, sharing, and liking of posts, the number of private messages exchanged, the number of viewed videos from friends or pages, and the global post popularity to determine promising prefetch candidates. To derive the user's engagement, their own Facebook application needs to be used, which is likely to introduce a bias, since the post ordering and the look-and-feel differ from the native Facebook client. Thus, it remains unclear how *O^2SM* orders Facebook posts and, generally, which design choices the authors made to create the framework. *CPSys* [Gou+15] is designed for mobile video prefetching of YouTube videos. The system consists of two main modules, a prefetch agent running on smartphones and a central predictor which informs the agent which videos should be prefetched. The central predictor keeps track of all user requests and determines the most similar users to a given user, i.e., the nearest neighbors by using the Jaccard index as a similarity measure. The number of videos to prefetch is determined by the number of videos requested over the last 10 days for each user separately. The videos to prefetch are selected, per user, by a queue containing all the videos requested by a user's neighbors. This queue is ordered firstly by popularity and secondly by recency if there are multiple items with the same popularity. Downloading of the videos is only conducted when a Wi-Fi connection is available. *CPSys* achieves overall a correct prediction ratio of 18-20%. However, in the analysis of *CPSys*, music videos are explicitly excluded as they show a different and more persistent popularity pattern compared with other video categories. Furthermore, the approach requires continuous monitoring of many users to work efficiently. This does not respect the user's need for privacy.

3.1.3 *Summary*

In the following, we discuss the presented related works in the context of vFetch. A set of OSN-based prefetching-related works have been proposed, e.g., Facebook-based approaches [SSX+13; ZDW+13; DZW+14] and a *RenRen*-based approach [WSY11]. However, the video like and comment counts, as well as a Facebook friendship, are not sufficiently predictive features [WRT+15; PPW+15] for prefetching. Furthermore, the majority, i.e., 75% of YouTube video requests did not originate from social sharing [BIV+13]. Therefore, we decided to not rely on OSN information for the design of vFetch but to focus on content features and individual user preferences since they are more likely to be predictable for a larger share of a user's video views. Prefetching from the related video list [KZK+12; PCO16] shows a promising accuracy which we also observed for our users (cf. Section A.1), though it is only applicable if a video is currently played. Hence, content prefetching for a related video must happen at the same time before the user decides to watch a related video next. This puts an additional burden on the user's Internet connection and is likely to decrease the video quality of the currently watched video. For vFetch which strives to prefetch content hours in advance ideally over Wi-Fi, this is an inefficient approach. The prefetching application from Gautam et al. [GPN13] requires the user to use a different application that presents prefetched content to the user. This prevents the user from using the native YouTube application with its desired features, including look and feel. Furthermore, when we tried the app, it downloaded a vast amount of videos from our YouTube subscriptions and drained the battery. By reverse-engineering the app, we could not find any filter, selection, or machine learning-based mechanism. Finamore et al. [FMG+13] propose a wireless broadcast of popular content over 3G/4G networks. However, they discuss if the low performance of their system is worth the engineering effort. This matches the results of our user study in which almost no channel was subscribed by two users which indicates a very diverse user interest that cannot be efficiently served by globally or country-wide popular content. *CPSys* [Gou+15] does not focus on popular content but on individual user similarity to determine prefetching candidates. This results in a decent performance. However, the mechanism requires to store a user-item matrix that covers a large time span. Hence, the user's privacy is violated by tracing each individual user's requests. In some countries, this is likely to be illegal. Therefore, we design vFetch to run on a user's mobile device and only rely on this user's request history. Thereby, the user's data is kept local on his device and is not collected and stored externally as in the case of *CPSys*.

In Table 3 we provide a synopsis of the related work. Here, we distinguish prefetching mechanisms based on their application scenario which can be either general stationary devices or mobile devices. Additionally, we show to which extent efficiency metrics, i.e., Wi-Fi offloading and QoS metrics are considered in the design and evaluation of the mechanism, as well as cost metrics and research area-implied properties. To the best of the author's knowledge, there is currently no privacy-preserving mobile prefetching mechanism and no YouTube-specific mechanism solely focusing on content properties. This motivates the vFetch mechanism presented in Chapter 4.

Scenario	Approach	Efficiency		Costs			Research area implied						Description
		Wi-Fi Offloading	QoS	Energy Saving	Data Cap	Overhead	Automatized	User Interests	Network Dynamics	Privacy-preserving	Focus	Content Type	
Stationary Devices	NetTube [CL09]	−	+	◦	◦	◦	+	◦	◦	◦	Social	YouTube	Generates an additional P2P overlay per video for users to re-distribute their cached videos.
	SocialTube [LSW+12]	−	+	−	−	−	+	+	−	(+)	Social	Facebook	Generates an additional user interest-based P2P overlay; uses OSN-based video prefetching
	Wang et al. [WSY11]	−	+	◦	◦	◦	+	+	−	◦	Social	RenRen	A prefetching mechanism based on the user's social ties in an OSN is proposed.
	Khemmarat et al. [KZK+12]	−	−	−	−	+	+	−	−	+	Content	YouTube	Prefetching candidates are selected from the top videos from a watched video's related video list or YouTube search results
	Pleşca et al. [PCO16]	−	+	−	−	−	+	−	(+)	(+)	Content	YouTube	Theoretical optimal prefetching policies are derived using Markov chains to model user video request sequences
Mobile Devices	Incoming [GPN13]	+	−	+	−	−	+	−	−	−	Content	Multi	The energy efficiency of Wi-Fi offloading is investigated
	Finamore et al. [FMG+13]	−	−	−	−	+	+	−	−	−	Content	Multi	Wireless broadcast of 3G/4G networks is used to distribute popular content
	O²SM [ZDW+13]	+	+	+	+	+	+	+	−	−	Social	Facebook	Proposes a middleware learning a user's interests and considers energy and connectivity
	Do et al. [DZW+14]	+	◦	+	+	+	+	−	+	−	Social	Facebook	A broker-proxy architecture is proposed, determining prefetching candidates at the broker and prefetching them to the mobile device
	CPSys [Gou+15]	◦	◦	◦	◦	+	+	◦	◦	−	Social	YouTube	Prefetching based on user similarities considering past and recent video requests; ignores music videos
	vFetch [KLR+17]	+	+	−	+	+	+	+	−	+	Content	YouTube	Prefetching based on mobile Internet connectivity and content affinity using content features such as channels; operates privacy-preserving on user devices

Table 3: Classification of prefetching mechanisms, their design goals and key features as *supported/optimized per design* (+), *potentially compatible* ((+)), *unaware/not discussed/unclear* (◦) or *inefficient/incompatible with design* (−).

3.2 PROACTIVE CACHING

Video caching is a well-established technique to make high-quality video delivery scalable and decrease the delivery latency which comes with an increased QoE. Thereby, the content is served from nearby caches instead of a single, potentially far away source with limited bandwidth. Providing content by a geographically distributed network of caches allows scalable and fast delivery of video content but also, e.g., JavaScript files, images, and HTML pages. Content Delivery Networks (CDNs) provide globally-distributed content caching as a service, e.g., Google Global Cache[2] in the case of YouTube and Open Connect in the case of Netflix[3]. Caching can be done in several places within the content delivery ecosystem. Large Internet Service Providers (ISPs) also use intra-ISP caching to distribute the content of their own IP Television (IPTV) services [LLW+11]. However, at the moment large ISPs hesitate to allow CDNs to operate caches within their network as they fear CDN traffic is unpredictable and hard to manage [HH11]. In the past, ISPs also cached content from bypassing requests, e.g., from YouTube. However, since there is a trend to encrypt every connection, this is not feasible anymore since almost the entire web uses encrypted Hypertext Transfer Protocol Secure (HTTPS). A solution to this hurdle would be an ISP-CDN cooperation, which is a recent research topic.

Caching locations

Caching has been a research topic since the early 90s [LWY93]. Remarkably, the interest in this topic has not diminished due to the emergence of new technologies and the important use case of VoD delivery. CDNs [YLZ+09], P2P systems [LSW+12; HLL+07; LBW+09], IPTV [LLW+11], Information-centric Networking [ZLL13], and femto caching [GSD+12; BBD14] are among the most popular trends which rely on in-network content caching. A survey of traditional web caching replacement strategies can be found in the article of Podlipnig et al. [PB03] and more information on cache placement in the paper of Zhang et al. [ZW17] for wireless networks and in the paper of Aadhikari et al. [AJC+12] for YouTube specifically.

Caching research

In the vision of future 5G scenarios [BBD14], intra-ISP caching is likely to become a pay-per-use service. Cost trends suggest that caching generally becomes more cost-efficient since the cost of storage decreases faster than the cost of bandwidth [ER15]. Hence, the storage-bandwidth trade-off [CKS02; EGH+11; RS13] becomes more and more attractive from the cost perspective. Robers and Sbihi [RS13] investigate the trade-off between saved bandwidth and cache storage capacity. Their evaluation is based on Che's approximation [CTW02] using a realistic traffic model derived from BitTorrent popularity measurements.

Research visions

For the future, edge caches [LRD+13; MH10; EGH+11] at home routers[LPB+15; SBH13], cellular network antennas, mobile cloud nodes, and PDN Gateways are envisioned [Sar12; ER15; SBH13; LPB+15] as well as caching for individual users on their end devices [KZG13; HFG+12; FMG+13; GPN13; QQH+12], opportunistic caching for mobile devices [NLH15; TMB+13; HHK+12], and collaborative media sharing in public transportation [LC06; TCZ+13; TPS+16].

Currently, the most prominent position for caches for Over-the-Top (OTT) VoD content are peering nodes, e.g., the DE-CIX[4] in Frankfurt, Germany. However, traditional reactive caching is not efficient for nodes which receive low demand due to the

[2] `https://peering.google.com/#/infrastructure` [Accessed: November 19, 2018]
[3] `https://openconnect.netflix.com` [Accessed: November 19, 2018]
[4] `https://www.de-cix.net` [Accessed: November 19, 2018]

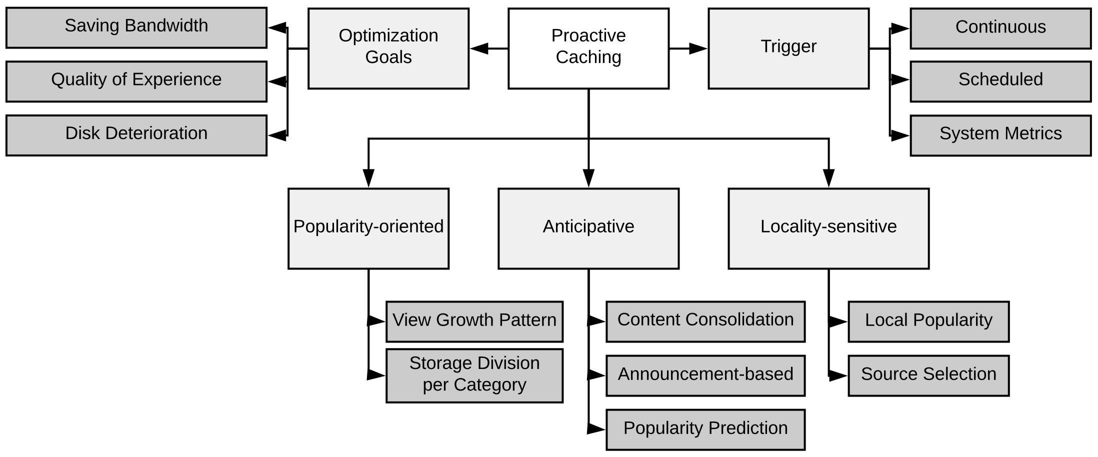


Figure 13: Taxonomy of proactive video caching

performance-decreasing impact of content churn [ER15]. Thus, it is suggested that base stations or mobile cloud node caches must be managed proactively, i.e., content that is estimated to be popular is placed and kept statically in the cache. Though, this is not the only variant that can be considered to be proactive. Further examples are proactive request routing [BRZ+17] or location-sensitive caching [DTK+16]. *Proactive caching*

Ramanan et al. [RDH+13] analyze mobile data traffic and different content types, e.g., application, audio, image, text, video, and others. They focus on selective caching based on content type and caching at the Serving Gateway in an LTE network's Evolved Packet Core. To this end, they used an LTE request trace. Furthermore, they use the *cacheability* metric, which indicates how useful it is to cache an item depending on its content type. Thereby, cacheability is defined as $\frac{\sum_1^n (k_i - 1) s_i}{\sum_1^n k_i - 1}$ as originally proposed by Ager at al. [ASK+10]. Here, k_i denotes the total number of downloads for item i of size s_i They found that 30% of all request responses are cacheable, accounting for 73% of the data volume. The content types showing the highest cacheability are application and video. In their analysis, YouTube content has shown to be even more valuable to cache than Netflix content, while both belong to the largest traffic sources. Before we review the related work in the area of proactive caching, we define a taxonomy. This taxonomy is used to categorize the related work with respect to their design choices. *Content-aware caching*

3.2.1 *Taxonomy*

Figure 13 depicts the taxonomy of proactive caching used in this thesis. On the taxonomy tree's first level, we see the *Optimization Goals*, the *Trigger* causing a proactive process, and the three classes of proactive mechanisms: *Content-oriented*, *Anticipative*, and *Locality-sensitive* mechanisms. The manifestations of each of the nodes mentioned above are discussed in the following.

3.2.1.1 *Optimization Goals*

Caches serve as a low-pass filter for popular content. Hence, depending on the cacheability of the request distribution observed, caches reduce the amount of network load measured in bandwidth between the cache and the content source which *Saving bandwidth* holds the entire content catalog. This property can be efficiently leveraged, e.g., in heterogeneous networks with network segments of different bandwidth availability. One prominent example are mobile networks, where the backhaul, i.e., access and aggregation network, is often considered to be or to become a bottleneck [Zha14; BNP+15; TMF+14]. Therefore, caches between the mobile backhaul network and the customers can reduce the load and, thereby, the required bandwidth at the backhaul network. Though, this is only possible if the content is cacheable, which requires redundant requests within a certain time interval as well as unencrypted transmissions or a collaboration of the ISP and the content provider.

The Cache Hit Rate is the most common metric when evaluating caching mechanisms and describes the percentage of requests that were served from the caching system instead of from the source. The CHR is defined as the ratio of cache hits of the cache or, in case of multiple caches, for the entire hierarchy to the overall number of requests N, i.e., $\frac{1}{N}\sum_{i=1}^{N}\mathbb{1}_{h_i}$ with h_i evaluates to `true` if request i is a hit and `false` otherwise.

Quality of experience

Accordingly, this metric indicates the origin offload, i.e., how much traffic can be saved at the content origin by caching. A high CHR does not only save bandwidth but also increases the transmission latency to the client which increases the user-perceived QoE (cf. Section 2.3.1) as videos tend to start playing back faster and the overall QoE is increased by a lower transmission latency as measured by Stohr et al. [SFF+16]. A weakness of the CHR is that it does not reflect the number of bytes but of cached content items. However, each content item can have a different size. To this end, the less prominent Byte Hit Rate (BHR) can be used if the absolute amount of bytes served from the caching system has to be evaluated. Similar to the CHR, the BHR is defined as the ratio of Bytes delivered from the cache to the overall number of Bytes requested B, i.e., $\frac{1}{B}\sum_{i=1}^{N} b_i$ with b_i as the size of the content in Bytes that belongs to request i.

Disk deterioration

During peak usage times, disk load is often a major performance bottleneck next to network latency. Here, insert and delete operations take much more time than read operations and, furthermore, lead to deterioration of Hard Disk Drives (HDDs) as well as Solid-State-Drives (SSDs). In recent years, the price gap between SSDs and HDDs has closed [Hac15], which makes SSDs attractive as storage for caches. One drawback when using SSDs is the deterioration. SSDs, as NAND flash devices, consist of storage cells, which store bits by assigning two different voltage levels: one representing a 1 and the other one representing a 0. These cells are limited in the number of times this value can change [Bas15]. Hence, the number of write operations is a vital cost metric considering the lifetime of SSDs. Read operations are far less costly and strongly correlate with the CHR.

3.2.1.2 *Proactivity Trigger*

Continuous trigger

In a practical deployment, a proactive mechanism is executed multiple times. Therefore, it needs to be defined which event triggers the mechanism execution. A continuous activation of the proactive mechanism is the most preferable from the performance perspective as it permanently adapts the respective mechanism. However, this might come at a high price, e.g., when the required computational effort is high, e.g., when re-training a machine learning model. Instead, a scheduled execution of the proactive mechanism can be a trade-off between optimal performance and low

Scheduled trigger

cost. Here, the frequency of the execution has to be carefully considered to keep the performance degradation bounded. Time To Live (TTL)-based cache eviction policies can be seen as a special case of scheduled triggering since each content item is assigned to a timer that determines when the content is removed from the cache. Recent works in this area address how cache networks can be optimized by analytically deriving optimized TTL-based eviction policies [BHC+15; BGS+14]. Besides a continuous and a scheduled execution, a third option is using an intermediate metric which is continuously monitored and triggers the execution if it exceeded or undershot a specified threshold. This metric can either be a native system metric or a composed metric considering multiple native metrics. A mix of scheduled triggering and metric-based execution can also be reasonable if the metric-based execution can be expected to rarely happen but is essential to prevent a severe performance depletion.

System metrics as a trigger

3.2.1.3 *Popularity-based Caching*

Chowdhury and Makaroff investigate popularity growth patterns in YouTube and, thereby, distinguish between YouTube categories [CM13]. For their analysis, they use a dataset of videos uploaded within two days and monitored the view count of these videos for the following five months. They found that YouTube categories exhibit different view growth patterns regarding maximum popularity as well as popularity evolution over time. Furthermore, they successfully applied time series clustering to understand growth pattern for different categories. One of their findings is that early video popularity is unsuitable for predicting future popularity.

View growth pattern

The findings of the authors motivate our category-aware caching mechanism, ProCache, proposed in Chapter 5 and confirm that content belonging to different YouTube categories are likely to profit from different caching strategies and storage size. A related storage division-based approach is presented by Shafiq et al. [SKL16]. They divide the cache storage into multiple adaptive segments for images, video, audio, compressed, and misc. A simple approach that is designed to optimize the cache performance for content of different popularity is Adaptive Replacement Cache as introduced in Section 2.2.1.2. When using Adaptive Replacement Cache (ARC), the cache storage is split into two cache divisions where each has its own ghost list that tracks recently evicted content. The first cache division is a probationary cache, which stores content that is requested for the first time. If new content has to be cached and the probationary division is full, the oldest content is evicted in a First In - First Out (FIFO)-manner and the new content is stored. In this case, the ID of the evicted item is stored in the probationary part's ghost list. The ghost lists are limited in the number of items and cannot exceed the size of the corresponding cache division. A cache hit in the probationary part causes the content to be moved inside the Least Recently Used (LRU) cache division, assuming that it will be requested again soon. In case the LRU cache evicts content, the content ID is stored in the LRU cache's ghost list. If the requested content is stored in the probationary part but its ID is already present in its ghost list, this indicates that the probationary part is too small. Therefore, its size needs to be increased. Consequently, the LRU cache's size needs to be decreased to not exceed the available storage space. Similarly, in case content is moved from the probationary to the LRU cache and is present in the LRU

Storage Divisioning

cache's ghost list, the LRU cache is increased and its probationary part decreased in size [Eva14; MM04].

3.2.1.4 *Anticipative Caching*

Content consolidation

Chatzieleftheriou et al. [CKK17] use a recommender system in a way that increases the CHR. To this end, a user's content search is answered such that content items which are of interest to many users are ranked higher and are recommended to the user. Thereby, the user may not get the content perfectly matching his interest but similar content. The divergence between the most suitable and the recommended content is measured and can be bound by a maximum tolerable distortion.

Announcement based

Bai et al. [BJS13] discuss caching mechanisms using information from OSNs for content published on Facebook and Yahoo. Their proposed approach is denoted as SOCR (Social On- line Cache Refreshing) and refreshed cache content in an online manner each time a user requests his news feed. A related approach for VoD services is proposed by Claeys et al. [CBV+15]. They use all users' current playback position t anticipate which video segments are going to be requested in the near future to cache them in advance.

Popularity prediction

Kaafar et al. [KBD13] propose a proactive caching approach for CDNs. Therefore, video candidates are determined based on either using a content-based recommendation for cold items with only a few requests or Collaborative Filtering for warm items. However, they do not propose a specific mechanism nor an evaluation of their proposal.

3.2.1.5 *Locality-sensitive Caching*

Local popularity

In scenarios with a location-bound content demand, locality-sensitive caching helps to offload source and the backhaul network. Here, clients can retrieve content from a nearby cache server instead of the source or a potentially far away cache. Besides, the local cache can adapt to the content popularity distribution of the specific area it serves. Golrezaei et al. [GSD+12] study this scenario empirically, while Dernbach et al. [DTK+16] propose an analytical performance model. Besides spatial-correlated content popularity, also for globally popular content, the distance from the cache to the users' location is an important metric to consider. A higher distance does not only create a higher delay that correlates with the user-perceived QoE but also uses more network resources and path capacity. Though, it can be reasonable to select a more distant cache for delivery in case the bandwidth of the nearby cache is significantly lower. This scenario is studied, e.g., by Bhat et al. [BRZ+17].

Source selection

3.2.2 *Discussion of Selected and Representative Work*

Content-aware caching

Shafiq et al. [SKL16] propose a content-aware caching replacement approach based on the insights gained from a CDN workload analysis. Motivated by the temporal dynamics for different content types, e.g., video, images, and compressed files, the authors divide the cache storage among different category types in a dynamic fashion. They allocate more storage to content types that generally contribute more to the cache performance. They propose to measure this performance using the CHR for content of equal size and the BHR for content of different sizes. To estimate a content type's performance contribution, they use a time series prediction model, i.e., Autoregressive Integrated Moving Average. The derived model is regularly updated because a continuous update would make the storage allocation unstable. Their findings motivate the category-aware storage-splitting approach we propose for ProCache. An approach that focuses on video popularity is proposed by Hasslinger et al. [HNH14]. They compare LRU with statistic-based caching strategies for Zipf-distributed popularity. They propose Score-gated LRU, defining a score for each object by its popularity. Thereby, the items in the cache are kept constant as long as content popularity distributions do not change. This is beneficial over LRU which always loads every newly requested object into the cache, if not present already. Using score-gated LRU, an object is only inserted in the cache if its score surpasses the lowest score of all objects in the cache. By implementing a variant of score-gated LRU, the authors achieve about 10% hit rate increase compared with LRU. However, it is questionable how the proposed score can be efficiently computed and how it performs on real-world workloads which exhibit quite diverse popularity distributions depending on a videos category [CM13] and age [HNH14].

Anticipative caching

A very different approach compared to the approaches mentioned before is presented by Chatzieleftheriou et al. [CKK17]. They propose using recommender systems as a traffic engineering tool. Specifically, they shape the demand of their users by recommending content that is of interest to a large number of users instead of only a few. Thereby, the cacheability of the requests arriving at the cache is increased and, consequently the CHR. To balance the user- and cache-centric objectives, a maximum distortion is defined, limiting the deviation of the user's ideal content and the recommended one. We refrain from adopting this idea in the design of ProCache since it most likely decreases the overall user satisfaction which we consider as the most important long-term factor for a VoD service's success. However, in cases of network congestions this might be a suitable temporary solution and superior to low-quality video delivery or stalling playbacks. An announcement-based caching approach for VoD content is presented by Claeys et al. [CBV+15]. By respecting the temporal structure of video segment requests, as well as the chance that a user watches multiple episodes of a series consecutively, announcements are created to inform the caching policy in advance. The evaluation is based on a dataset from 2010 containing 108,392 requests to 5,644 unique videos, which are assumed to be 50 minutes long, using 1 Mbit/s. Simulations are conducted using a realistic network topology and assuming an exponential distribution of video session durations. Thereby, the authors consider that most videos are only watched partially [KH14]. By considering announcements of the videos a user is going to watch in the near feature, the CHR is increased by 11% compared with LRU. In contrast to the paper proposed, the authors consider episodes of a VoD portal dedicated to TV series which results in a small and homogeneous content catalog compared with YouTube. Furthermore, videos offered by the most popular VoD services tend to be much shorter and, hence, the positive effect of segment announcement is limited or might even negatively impact the performance. Baştuğ et al. [BBD14; BBD16] design an analytical model to estimate the performance of caching at small cell base stations in cellular networks. In a further work, they discuss the trade-offs on deploying cache-equipped small cell networks [BBK+15].

Next, we discuss approaches that proactively cache by considering cache and user locations. Dernbach et al. [DTK+16] propose a model which is able to capture the overlap between intra- and inter-regional user preferences. This allows giving guidance to the question of whether a content provider should fill globally popular content into its caches or, in addition, also consider region-specific content popularity. Golrezaei et al. [GSD+12] investigate the potential of prefetching at femtocells for cooperative caching in mobile networks to relieve the backhaul network. For their simulative evaluation, they use the superposition of the two most active four-hour time intervals contained in a YouTube trace recorded in 2008 at a US university campus. Therefore, their evaluation results are hard to compare with a country-wide or global workload. Bhat et al. [BRZ+17] propose a mechanism denoted as Software-defined Networking (SDN)-assisted Adaptive Bitrate Streaming (SABR). It is based on an enhanced DASH manifest and an SDN-assisted control plane that monitors network conditions. The DASH manifest is extended by providing multiple locations of the video segments together with the available bandwidth of the delivery path. Hence, the client stays in control of the decision from which source to download the respective DASH segment. An SDN control application evaluates this bandwidth information. The proposed approach can efficiently react to network congestions. However, we find the general cache performance also in the absence of congestion more important and, thus focus on it in the design of ProCache. Summers et al. [SBE+16] analyze Netflix's workloads showing chains of sequential requests. Here, the authors present prefetching algorithms that reduce hard disk and main memory utilization, thereby not focusing on individual users.

Location-aware caching

3.2.3 *Summary*

In the related work discussed before, we have seen that content-aware caching is generally performance-increasing. While Shafiq et al. [SKL16] have shown this for different file types such as video, images, and compressed files, ProCache adapts this idea for VoD content categories. Thereby, we focus on one of the largest single sources of Internet traffic, namely YouTube and its specific set of subcategories. Proactively caching by using video popularity forecasts is an interesting and promising idea considering historic and a priori known popularity distributions. However, a simple score [HNH14] used for recent content is unlikely to work efficiently on real-world workloads. First, it is generally hard to conduct popularity predictions for recent contents with only a few requests [Pin+13]. Hence, it is questionable how efficient a popularity prediction approach is for real-world workloads. Furthermore, a single score function is not flexible enough to accurately model the differences in video popularity concerning video categories. Depending on the video category, the distribution of the time till a video reaches its popularity peak and the percentage of total views over time differ significantly [CM13]. Thus, in the design of ProCache, we emphasize different content categories by flexibly reacting to their different and changing popularity distributions. We refrain from changing the content presented to the user in favor of a higher cache performance [CKK17] a maximum user satisfaction is essential for the long-term success of a VoD platform. While we find announcement-based caching a good approach for VoD portals with small content catalogs and long playback durations, it is unlikely to perform well for typical VoD services such as YouTube, Vimeo, or Dailymotion. The primary reason for this is that their content is short and watched only partially. Hence, announcements are less predictive and, therefore are likely to have limited or negative effects on the cache performance since it occupies storage that can be used for actually requested content. The same applies to approaches that leverage a subsequent request behavior as observed for Netflix [SBE+16]. Furthermore, we decide to evaluate ProCache using a real-world user request trace and, thereby to follow an empirical approach. In contrast to analytical approaches [BBD14; BBD16; DTK+16], this guarantees realistic content popularity time patterns and workload which is generally hard to model analytically (cf. Section A.5).

Table 4 provides a comprehensive overview of the contributions of the distinct works as well as their efficiency metrics and research area-implied properties. We see that the related work does not focus on collaboration of different admission and eviction strategies and at the same time acknowledge that different types of video content, i.e., music or sport can profit from disjunct caching strategies. In Chapter 5, we aim to close this gap by the ProCache mechanism presented.

Contribution	Approach	Efficiency			Research area implied						Description
		Bandwidth Saving	Hit Rate	Write Operations	Trigger	Storage Size Variation	Segmented Videos	Multiple Qualities	Evaluation Data	Content Type	
Content	Shafiq et al. [SKL16]	+	+	+	Continuous	–	–	–	CDN Trace	Mixed	The cache storage is divided among different content types, e.g., images, videos, and audio
Content	Hasslinger et al. [HNH14]	–	+	–	Continuous	+	–	–	Synth. Req.	N/A	Content is evicted using a popularity score to keep the cached content static if the popularity does not change.
Content	ProCache [KPR+18]	+	+	+	Continuous	+	+	–	Req. Trace	YouTube	The cache storage is segmented according to content categories and their size adapts to the content popularity dynamically.
Anticipative	Chatzieleftheriou et al. [CKK17]	–	+	–	Content Similarity	+	–	–	Synth. Req.	N/A	Cache-aware recommendation of content similar to the user's interests but interesting to many other users.
Anticipative	Claeys et al. [CBV+15]	+	+	–	Continuous	–	+	–	Req. Trace	N/A	Content needed by most clients soon is preferred to be kept in the cache during eviction.
Anticipative	Bastug et al. [BBD14]	+	+	–	N/A	+	–	–	Numerical	N/A	An analytical model is designed to estimate the performance of proactive caching at small base stations.
Locality	Dernbach et al. [DTK+16]	+	+	–	Scheduled	+	–	–	MovieLens	N/A	A comprehensive analytical framework for studying the impact of locality-aware proactive caching is proposed.
Locality	Golrezaei et al. [GSD+12]	–	–	–	N/A	–	–	–	Req. Trace	YouTube	Distributed edge caches help to serve nearby clients by storing promising prefetch candidate videos.
Locality	Bhat et al. [BRZ+17]	–	+	–	Network Load	–	+	+	Synth. Req.	N/A	DASH manifests are extended by network bandwidth predictions for multiple caches storing the content

Table 4: Classification of proactive caching mechanisms: design goals, key features (*supported/optimized per design* (+) or *unaware/not discussed/unclear* (–))

3.3 SDN-BASED MULTICAST APPROACHES FOR VIDEO-ON-DEMAND

We want to provide a brief understanding of the major differences between traditional Internet Protocol (IP) multicast and SDN-based multicast in the context of OTT VoD content delivery since it is essential to understand the future works presented in this section. ISPs often use network layer multicast for their own internal IPTV services, for efficient content delivery [LLW+11]. In contrast to this, OTT content providers operate on the application layer and can only rely on the Internet routing mechanisms. IP multicast does not meet the requirements of OTT delivery, and hence is not available for this scenario [DLL+00]. The resources for managing multicast groups render traditional IP multicast for OTT services unlikely to scale, because each router needs to provide multicast capabilities. However, the advent of SDN allows revisiting network management concepts. This includes multicast which becomes more practical and easier to manage since an SDN-enabled switch is by default able to apply multicast mechanisms as a centralized SDN control application. Hence, the need for multicast-specific hardware is eliminated and each switch in the network capable of supporting multicast. Recently, SDN has found a broad adaption by network equipment vendors and network providers [JKM+13]. Though, SDN network architectures are centrally controlled. Thus, the deployment will most likely be per network operator, e.g., ISP-wide and not capture multiple ISPs and content provider. Thereby, each stakeholder stays in control of its network. However, ISP-controlled SDN networks open the possibility for SDN-enabled multicast of OTT content by ISP/CDN collaboration without the drawback of traditional IP multicast. An extensive overview of SDN-based multicast approaches and its specific use in an OTT content delivery scenario inside ISP networks can be found in the Ph.D. thesis of Julius Rückert [Rüc16] in Section 4.1. Before we review the related work in the area of SDN-based OTT VoD delivery, we define a taxonomy. This taxonomy is used to categorize the related work with respect to their design choices.

IP multicast not suitable for OTT content

SDN enables multicast of OTT content

3.3.1 *Taxonomy*

Figure 14 depicts the taxonomy of SDN-based multicast for OTT VoD used in this thesis. On the taxonomy tree's first level, we see the *Optimization Goals*, the *Client Aggregation*, as well as the *Content Selection* method used, the *Playback Start* time, and the *Cooperation* type. The manifestations of each of the aforementioned nodes are discussed in the following.

3.3.1.1 *Optimization Goals*

The most obvious property of multicast is that it avoids redundant transmissions of the same data. Hence, multicast saved bandwidth in scenarios where the same content is requested by multiple users connected to the same network. Assuming a geographically equally distributed demand, the bandwidth reduction is especially high at the network's core and the path to the content origin, while the last hop cannot profit from the bandwidth reduction since it constitutes the latest point of multicast stream duplication, i.e., one duplicate stream per requesting user. The bandwidth saved, when using multicast, can be used by other applications as well as by serving the content in a higher quality which fewer interruptions. Thereby, the QoE is likely

Saving bandwidth

QoE

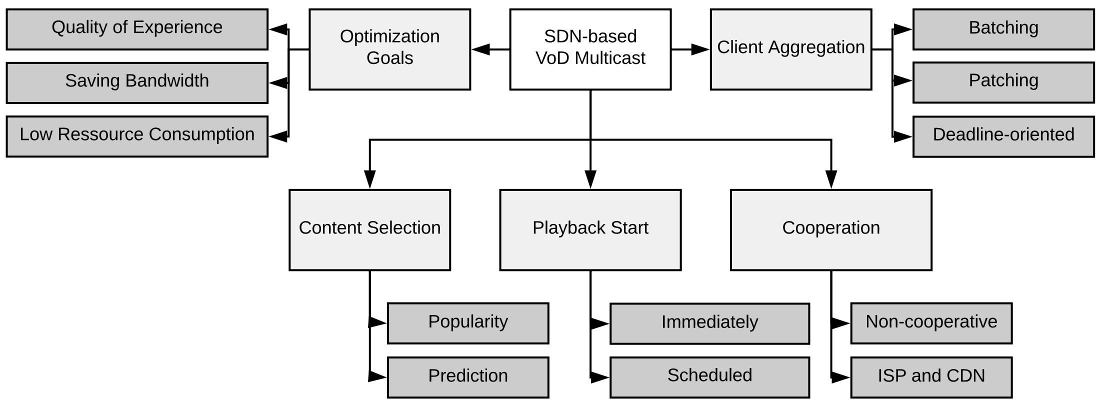


Figure 14: Taxonomy of SDN-based OTT VoD streaming

to be increased as potential network bottlenecks can be resolved in many cases, depending on the network topology, the geographically request distribution, and the connection path to the content source(es). More information on QoE in the context of video streaming can be found in Section 2.3.1.

The number of multicast flow entries and their changes state practical limitations of each SDN-enabled system [KRE+15], as the Ternary Content-Addressable Memory (TCAM) required for accessing these entries is a costly and limited resource on an SDN-enabled hardware switch. Therefore, a low resource footprint is desirable when using SDN-enabled hardware switches. Also, the management overhead, i.e., the SDN flow entry changes are limited as well and have to be within the capabilities of current hardware devices. Without respecting these limitations, an SDN-based system is not considered practicable especially in the context of other SDN applications running in parallel.

Low resource consumption

3.3.1.2 *Client Aggregation*

In the following, client aggregation mechanisms for aggregating clients are proposed. The scenario assumes that many clients request segmented videos, e.g., DASH from a limited content catalog.

Batching [DSS96] aims for aggregation of streams delivering the same content into one multicast group. Hence, clients are likely to wait until the next batching time window starts. Thereby, batching has a fixed maximum startup latency, i.e., the size of the batching time window. Letting users wait for more than a few seconds, until the playback starts, is impractical since it decreases the user-perceived QoE [HSB+13]. Different patching schemes are proposed which vary in their main objectives. MQLF [DSS94] schedules the segments requested by most clients first. Thereby, the maximum bandwidth can be saved at the cost of fairness, since requests to unpopular content are unlikely to be served. FCFS [DSS94] gives emphasis to the maximum waiting time and schedules the segment with the oldest pending request first and thereby more emphasis on fairness. More evolved schemes exist, e.g., in [AWY96b], [DF02], and [TV97], but are left out because they are of less relevance for this thesis.

Batching

To eliminate the major drawback of batching, patching [HCS98] reduces the initial waiting time till a video plays back. This is achieved by additional video delivery streams to overcome the gap, i.e. the patching window, between a user's current

Patching

playback position and the playback position of the existing multicast group(s) for the requested content. Some works have adapted patching to different scenarios, More evolved schemes exist, e.g., Pyramid Broadcasting [VI95; AWY96a], Skyscraper Broadcasting [HS97; EV98], Disk-conserving Broadcasting [GKT02] as well as approaches using more than two download channels, such as Controlled multicast [GT99] and Catching and Selective catching [GZT99], but are left out because they are of less relevance for this thesis.

Deadline-oriented

Deadline-oriented approaches give the highest priority to the video segments that are needed most recently to avoid playback stalling. Thereby, fairness is ensured by not letting certain video players starve. However, this comes at the cost of throughput in certain scenarios. For example, if segment A has a deadline earlier than segment B and segment A is scheduled to be multicasted. Assuming that no further resources for multicasting are available, not segment B might not be delivered even though much more clients are requesting it.

3.3.1.3 *Content Selection*

Much more on-demand content exists compared to IPTV channels, e.g., about 1.2 billion[5] videos on YouTube. As the number of multicast groups is limited on a hardware switch, not all of these videos can be delivered by network-based multicast. As VoD popularity is Zipf-distributed, only a small fraction of the content catalog is popular and, therefore, efficient to be delivered by multicast. Selecting videos for multicast is challenging as popularity changes dynamically based on, e.g., video age and daytime [CKR+07].

Popularity

Assuming that the video popularity distribution does not change rapidly, the current popularity can serve as an indication of which videos to consider for multicast. For content with slowly changing popularity distributions, e.g., Music content, this is likely to be a suitable approach. However, it is not able to anticipate the effect of popularity changes in the near future. A video might, e.g., have a decreasing popularity and can be assumed to be irrelevant for multicast within the next time. To detect such trends and consider them in the decision on which content to multicast, a prediction is required. Thereby, videos becoming more popular can be selected for multicast at an early stage. In addition, a prediction may consider that a new Music video is released at a given time and thousands of people are waiting to watch it as soon as it becomes available. A predictive approach allows considering such information in advance.

Prediction

3.3.1.4 *Playback Start*

Scheduled

When aggregating clients into a multicast group, they may be required to wait until a new multicast stream for the requested video is created and they get delivered the requested content. This has the drawback of letting clients wait, which renders them likely to quit the service with a bad impression. Hence, an immediate playback start is preferable. This can be achieved by serving the clients with an intermediate multicast group to patch the gap between the client and a potentially existing multicast group. However, this assumes that the client's video buffer can store the content

Immediately

[5] http://tubularinsights.com/youtube-changes-33-percent-a-year/ [Accessed: November 19, 2018]

that constitutes this gap and join an existing multicast group afterward. Another approach would be to rely on unicast for a fast playback start, which is also preferable in the context of many videos being watched only partially in common VoD services like YouTube [KH14].

3.3.1.5 *Cooperation*

Content Delivery Networks (CDNs) like Akamai[6], Open Connect[7], and Google Global Cache[8] help to deliver content globally. However, CDNs are often not allowed to operate inside ISPs because they fear CDNs as unpredictable and hard to manage traffic sources [HH11]. Therefore, CDNs are often not present within ISP networks and cannot help to optimize the network usage there. In the past, ISPs cached bypassing content from outside their network on the path to one of their customers. Nowadays, this has become inefficient since there is a trend to encrypted HTTPS connections. ISP/CDN cooperation is likely to solve this problem, e.g., by allowing group encryption schemes [DC06] to support a secure multicast transmission to the clients. Such a scheme prevents the ISP from reading the content transferred from the content source to the clients and at the same time allows efficient ISP-internal multicast.

Non-cooperative

ISP/CDN cooperation

3.3.2 *Discussion of Selected and Representative Work*

SDN-based multicast for VoD

Only a few mechanisms exist that consider SDN for multicast video streaming. Rückert et al. [RBH+16] proposes a mechanism called Software-defined Multicast. They design a one-to-many service which is used between CDNs and ISP to allow ISP-internal multicast of OTT live video content. To this end, the ISP offers a control Application Programming Interface (API) to the CDN. Thereby, the CDN informs the ISP about video streams that can be delivered using multicast. In a next step, this stream is sent from the CDN to the ISP and forwarded within the ISP network to the clients by using SDN-enabled switches. The Software-defined Multicast (SDM) controller manages the ISP-internal multicast groups. It maintains the required network-layer multicast trees which need to be established inside the ISP network. Therefore, the shortest-path tree is computed by using a variant of Dijkstra's algorithm. This tree consists of the paths on which the video stream is forwarded. Each multicast group is assigned to a unique IPv6 multicast address of the ISP which is used by the CDN to send the video segments which are multicasted within the ISP network. After computing the multicast tree, the SDM controller translates the tree topology to the corresponding SDN flow entries and forwards them to the SDN controller. These flow entries are responsible for forwarding the multicast groups' data through the data plane. One subtask of the forwarding is the video stream duplication, e.g., at the latest possible point in the network to each of the group's clients. Besides, the SDM controller maintains the multicast tree, e.g., when client join or leave and triggers the necessary flow entry changes by notifying the SDN controller. The VoDCast mechanism proposed in this thesis is based on the concept of SDM. However, SDM

[6]`https://www.akamai.com/` [Accessed: November 19, 2018]
[7]`https://openconnect.netflix.com` [Accessed: November 19, 2018]
[8]`https://peering.google.com` [Accessed: November 19, 2018]

focuses on live streaming only while VoDCast addresses the particular challenges to design an SDN-based OTT VoD delivery mechanism.

Yang et al. [YYR+15] propose an SDN-based Multimedia Multicast Streaming Framework denoted as SDM^{2}Cast. They highlight the scalability of their approach and its flexibility as they use Scalable Video Coding (SVC). Their evaluation is based on a prototypical implementation using three different video quality layers, 16 OpenFlow switches, and 60 client devices. During the evaluation, each client requests a pre-defined content quality. In the author's experiments, SDM^{2}Cast reduces the consumed bandwidth by 50% compared with IP multicast using a single video quality. However, SDM^{2}Cast is not evaluated using a realistic ISP workload and scenario.

In the following, we present scheduling mechanisms designed for VoD content delivery. Dan et al. [DSS96] propose a fixed batching time-window in which requests for the same video are aggregated, resulting in a separate multicast group for each video at the end of each time-window. Consequently, most client requests cannot be handled immediately and are queued. Two policies are proposed that schedule videos from this queue, either prioritizing popularity or waiting time for multicast delivery. Popularity-based scheduling has the drawback of high waiting times for users that request less popular videos. The higher the waiting time, the higher is the probability of a user leaving the streaming service, which is undesired. Prioritizing requests with longer waiting times has the drawback of many groups serving only a small number of users. A combination of both of these policies is proposed by Aggarwal et al. [AWY96b], taking waiting time and video popularity into account. Overall, the main drawback of batching is the additionally introduced waiting time for users till they are delivered with the first video segment. However, a fast playback is essential for the user experience. Furthermore, many video sessions only last a few seconds. Therefore, it is considered unreasonable to deliver the first segments by multicast as it would cause a lot of costly join and leave events.

Batching

Hua et al. [HCS98] present two patching approaches taking care of the limited buffer space B, denoted in the number of playback minutes. The first scheme, i.e., Greedy Patching does not create new groups after a certain network-wide group limit is met, respecting limited network resources. A client immediately joins an existing group to receive the next B minutes of the video to fill its buffer. Compared to batching this approach's benefit is that clients receive segments earlier, i.e., before they are needed for playback. However, the drawback is that the first received segments are unlikely to belong to the video start and occupy the client buffer until the video player consumes them. This decreases the probability of receiving more recent segments by multicast. The second approach by Hua et al. requires each client to receive two multicasts in play rate simultaneously. One is an existing multicast group like in the previous approach. The other one is temporary to receive the gap between video start and the playback position of the existing multicast. This decreases the waiting time existing in the first approach. However, it is only reasonable to join both multicast groups if the playback distance between the temporary multicast group and the already existing one is smaller than buffer space B. Therefore, this improvement is limited and comes at the cost of temporarily doubling the number of multicast groups which are a limited resource.

Patching

Eager et al. [EVZ01] propose two approaches focusing on minimal waiting time for popular content. Both approaches use a parameter n for the maximal number of groups a client can join. In the first approach, a client receives all segments with a

Deadline-oriented

playback position of maximal x seconds away. The parameter x can be configured. If a client can receive more than n segments simultaneously, only the n groups which deliver the segments needed earliest for playback are joined. This can lead to a fragmented reception of segments and requires a complex scheduler. According to the authors, this is impractical to implement. Therefore, a more practical approach is proposed, delivering a video stream by at least one multicast group and other clients can join it. Similar to the first approach of [HCS98], each new client creates a temporary multicast group to minimize its waiting time and additionally joins the most recently created multicast group for this video, streaming segments starting from playback position p. If a temporary multicast group reaches position $p - 1$ the client leaves it and stays a member of the previously existing multicast group only. This leave only happens, if no other client joined the temporary multicast group in the meantime. If there is another client in the group, the client which created the group has to stay in the group till the last member leaves this group. This approach neglects per-group costs and is likely to require more bandwidth caused by receiving segments twice. To this end, mobile devices with limited resources, e.g., mobile data volume are unlikely to use such a system.

Aggarwal et al. [ACG+09] propose two deadline-oriented schedulers. In the first scheduling approach, no new groups are created if the maximum number of groups, similar to the first approach of [HCS98], is reached. If a group becomes free, a segment which is needed for playback first, by most of the clients, is scheduled. Each client who requested this segment joins the multicast group and receives the segment. In case no group is free and a segment reaches its deadline, it is never scheduled causing a playback interruption. In the second approach, the number of groups is unlimited. Therefore, it outperforms the first approach measured by required bandwidth and initial waiting time as the first segment's deadline is set to the current time. However, most videos, e.g., on YouTube, are not watched fully. Therefore, issuing requests for all video segments seems unrealistic.

A comprehensive survey of traditional VoD multicast mechanisms can be found in the survey paper of Ma et al. [MS02] and a survey of broadcasting schemes for VoD is presented by Hu et al. [Hu01]. Other approaches, which are not in the focus of this thesis, as they discuss multicast in specific application domains, e.g., in combination with other delivery techniques are briefly described in the following. Feng et al. [HC16] propose a client caching-enabled adaptive multicast delivery scheme and Jayasundara et al. [JG13] and Ramesh et al. [RRG01] propose a joint approach of caching and multicast for ISPs. Multicast schemes for VoD focusing on broadcast delivery or the wireless channel [SC17; JL17; YWX17; THD04; RW16; RW13] are not discussed in detail. Farhad et al. [FAK09] propose client-assisted patching for VoD in enterprise networks. Griwodz et al. [Gri00] present an optimization for patching denoted as *Multilevel Patching* [GLZ+00] and consider the application of patching in conjunction with content caching [GZL+00]. Also, works focusing on client-assisted P2P multicast are proposed [HBY04; CLN04]. Jang et al. [JL17] design a batching-like scheme considering adaptive modulation and coding of the wireless channel. Yuan et al. [YWX17] present an algorithmic approach to minimize initial stalling of a video playback in wireless networks by combining fast broadcasting [JT98] with unicast.

3.3.3 *Summary*

The results presented by SDM2Cast [YYR+15] are a proof-of-concept and show the general feasibility of SDN-based multicast. However, the scenario and workload used are unrealistic and do not allow any conclusions regarding the cost and performance when applied to a real-world workload. A careful and thorough analysis of SDN-based multicast is provided by DynSDM [RBH+16] and ASDM [BRV+15] which conduct comprehensive evaluations to assess their systems' performance. However, they focus on live video streaming and do not address the OTT VoD case. VoDCast closes this gap by addressing the particular requirements of SDN-based multicast for the VoD scenario. To do so, clients need to be served by a common multicast group. Several techniques such as batching [DSS96], patching [HCS98], and deadline-oriented approaches [EVZ01] have been discussed in the related work. Batching requires the users to wait until the next batching time interval starts. This is not desirable since the initial stalling time is increased and, thereby the QoE is decreased. Patching improves batching by a faster playback start since an additional multicast group is created to provide a patch between the user's playback position and an already existing multicast group. However, this is not cost-efficient since a low of videos exist on current VoD services such as YouTube, Vimeo, and Dailymotion. The available resources do not support such a large number of distinct multicast groups. Further, this is not efficient since the popularity exhibits a large long-tail of unpopular content (cf. Section A.5) which is inefficient to multicast. Hence, an efficient content selection is missing which we address in the design of ProCache. Furthermore, we evaluate the resource requirements of our mechanism carefully since they are limited in real-world deployments which is often neglected in the related work [DSS96; YYR+15].

We see that only a few mechanisms exist that consider SDN for the delivery of VoD using multicast. The existing mechanisms mostly focus on live content or are proofs-of-concept at an early stage. Also, the scheduling approaches reviewed do not consider the characteristics of VoD-specific user behavior, e.g., partial viewing [KH14] and Zipf-like popularity distribution [GHM13; ACG+09; GAL+07] of a large content catalog. So far, none of the existing works explores the potential of VoD for OTT video streaming under realistic workloads. To address this relevant scenario, we propose the VoDCast mechanism in Chapter 6. We deduce that the related work does either not consider the user's need for a fast video start or does neglect that the number of multicast groups is a limited resource. For a realistic evaluation, the number of groups and group changes have to be discussed.

Table 5 provides a synopsis of the related work. It provides a comprehensive overview of the contributions of the distinct works as well as their efficiency metrics and research area-implied properties.

Scenario	Contribution	Approach	Efficiency			Research area implied			Description
			Traffic Reduction	Initial Stalling	Resource Limitations	Video Delivery	OTT Support	Real-world Workload	
Live Streaming	SDN-based Multicast	SDM^{2}Cast [YYR+15]	+	–	–	SVC	–	–	A prototype of the proposed framework is evaluated as a proof of concept using 16 switches, 3 video quality layer, and 60 terminal devices.
		ASDM [BRV+15]	+	–	+	–	+	–	The trade-off between bandwidth and network state is investigated using unicast as a fall-back solution when multicast cannot be used, e.g., due resource depletion.
		DynSDM [RBH+16]	+	–	+	–	+	–	DynSDM extends SDM by emphasizing ISP-internal traffic and service management procedures, e.g., load balancing and using unicast as a fall-back solution if SDN switch capacities are exhausted.
VoD Streaming	Scheduling	Batching [DSS96]	+	–	–	–	–	–	One multicast group serves requests within a specified time interval. New requesting clients have to wait for the beginning of the next time interval.
		Patching [HCS98]	+	+	+	–	–	–	Clients immediately join an existing multicast group. Additionally, a multicast group is created to patch the difference between the client's and the existing multicast group's playback position.
		Deadline-oriented [ACG+09]	+	–	+	Segmented	–	+	Two EDF-schedulers are proposed that maximize the waiting time till a video segment is delivered to aggregate a high number of interested clients.
VoD	SDN	VoDCast [KHH17]	+	+	+	Segmented	+	+	VoDCast uses SDM to multicast VoD content in OTT scenarios. Thereby, ISP-CDN collaboration, encrypted traffic, and resource limitations are considered.

Table 5: Classification of selected SDN-based video streaming mechanisms: design goals, key features (*supported/optimized per design* (+) or *unaware/not discussed/unclear* (–))

4 PRIVACY-PRESERVING MOBILE VIDEO PREFETCHING WITH VFETCH

In our review of the related work (cf. Section 3.1), we identified a gap in privacy-preserving and content-oriented prefetching mechanisms. This chapter strives to close this gap by presenting a novel privacy-preserving and personalized prefetching mechanism for video content on mobile devices. We denote this mechanism as vFetch. To achieve efficient prefetching, we have to accurately model the user interests. However, in current recommender systems, this is mostly done by Collaborative Filtering (CF), which models user interests by the user-behavior similarity to other users. Hence, content that is interesting to similar users is recommended to the user for which the recommendation is conducted. vFetch differs as it does not rely on a database that contains many users' video watch histories. Instead, we rely on the video content properties and model the user's interest by differentiating content that is watched and not watched by the user. Thereby, vFetch does not depend on other user's data. Furthermore, it models the user interests locally as only the data of one user needs to be considered. Thus, vFetch is privacy-preserving and designed to operate on user terminals where the data is kept and processed at the user's premises. A further important aspect of prefetching is efficient scheduling of the video content download as the content needs to be fetched timely to avoid downloading it by using a potentially low-quality cellular Internet connection. Furthermore, download scheduling has a strong effect on the mobile device's battery. Downloading via Wi-Fi is preferable since the energy costs of 4G networks are about 20-times higher [HQG+12]. This is likely to contribute to offload cellular data networks during peak hours, and thereby reduces their transit costs.

The vision of vFetch was first presented in [KH14]. A prototype showing the technical feasibility in a simple scenario was demonstrated in [KRB+14]. How vFetch and mobile network operators can benefit reciprocally was proposed in [KBR+15]. Later on, vFetch's core mechanism and a user study analysis were presented in [KLR+17]. Furthermore, a number of student theses contributed to the publications mentioned above, and the mechanism presented in the course of this chapter [Lin16; Kos15].

Chapter outline

In Section 4.1, we provide a general overview of prefetching and its benefits. These make prefetching especially valuable in a set of use cases that we discuss in Section 4.2. As we observe a lack of suitable, current, public datasets, we conduct a user study to gain insights into users' video watching behavior on YouTube. vFetch's system design and major conceptual components are discussed in Section 4.5. The evaluation of vFetch is presented in Section 4.6. The data collected in the scope of the user study allows us to evaluate vFetch using a real-world workload. This chapter is summarized and discussed in Section 4.7.

4.1 CONCEPTUAL OVERVIEW

Before we detail the use case scenarios and the mechanism design of vFetch, we want to convey a conceptual understanding of the approach. To this end, Figure 15 compares the transmission flow using traditional video streaming with the transmission flow of prefetching. Using video streaming, the video content is transferred

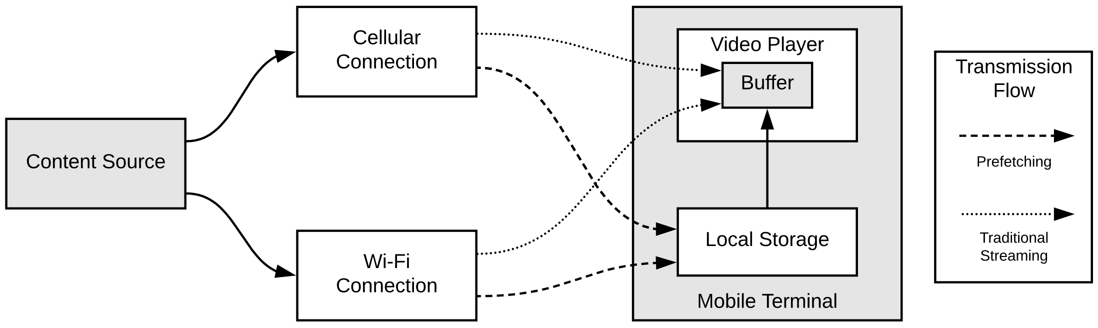


Figure 15: Video transmission flow of prefetching compared with video streaming

from a content source either by a cellular or a Wi-Fi connection to a mobile terminal. Here, small amounts of the content are stored in the video player's buffer to compensate for bandwidth variation and jitter. Though, this might not be sufficient to guarantee a fast video playback start and continuous video playback. Reasons for this are unstable connections and fluctuating or low bandwidth. This can be circumvented by content prefetching. Here, the video content is downloaded and stored locally in advance. In case a prefetched video is requested, the video player's buffer is filled from the locally stored video content; thus, playback impairments of traditional video streaming can be avoided.

From the abstract point of view, vFetch might appear very similar to downloading a video before watching it. In fact, the transmission path of the video content is the same. The major difference at a conceptual view is that using prefetching, the video content is selected from a large content set and downloaded automatically without user involvement and before the user wants to watch the video. Hence, the mechanism is transparent to the user and does not burden him with a manual content selection. Thereby, the content download is scheduled in a way that avoids costly conditions such as cellular connectivity and low battery levels. An additional difference to manual content selection is that the user can watch prefetched videos in a high quality even in situations where the cellular bandwidth would not even allow video streaming in its lowest supported quality.

4.2 USE CASES

Prefetching video content in advance to mobile terminals has a set of scenarios where it is useful concerning different objectives. In the following, we present three major use case scenarios where vFetch can contribute.

1. **Varying Connectivity:** Video streaming requires a continuous Internet connection. In some situations, this is not given, e.g., when commuting by subway or train [KFH17] which becomes an increasingly relevant scenario as a grow-

ing number of people travel to work. It is estimated that about 82% of commuters use their smartphone [Kel13] and a majority streams videos [JLT+16]. Thus, public transportation is a relevant case where people are likely to watch videos. Also in the area of self-driving cars, optimizing mobile connectivity is likely to stay challenging [Rüc17]. In this context, prefetched content has the advantage that the mobile connectivity is not required to playback prefetched videos. Therefore, vFetch can allow video playback in scenarios where traditional video streaming is impossible.

2. **Low Bandwidth:** In cases where mobile connectivity is given, the bandwidth might vary strongly. This can have two major reasons: i) The mobile resources, e.g., spectrum and access network bandwidth, are shared resources that tend to become scarce when many people use them, ii) Mobile data tariffs are often limited and allow, after exceeding the mobile traffic volume limit, only a limited speed that is insufficient for video streaming.

3. **Limited Energy:** Watching high-quality videos makes heavily use of the mobile transceiver which can drain be battery quickly [Gro+13]. Though, energy is a scarce and limited resource on mobile devices. Prefetching can avoid the energy-demanding cellular connection and instead give preference to download the video content via battery-sparing Wi-Fi or even while charging the device.

While the three use cases have been presented separately, in reality, a combination of them is most likely relevant. Which one is more or less relevant is highly user-specific. Hence, general statements about which use case is more or less important are hard to deduce.

4.3 USER STUDY DESIGN AND ANALYSIS

We designed and implemented an Android application denoted as *SocialMonitor*[1] to collect real-world user data of YouTube users in the scope of a user study. In this section, we use the results of this user study to deduce helpful insights for vFetch's design as well as its evaluation. We handed out the *SocialMonitor* application to our participants starting from November 2014. In the following summer and winter terms, we acquired additional participants by asking students in the scope of lectures and supervised bachelor and master theses to participate. After the *SocialMonitor* is installed and opened for the first time, it informs about which information is collected and for which purpose. In case the participant does not agree with this privacy consent, no data is collected. Furthermore, the application anonymizes all privacy-sensitive information directly on the device using a strong cryptographic hash function. Hence, only anonymized data is sent from the device to a collection server that regularly receives and stores the data of all participants. In addition to their agreement to the privacy consent, the *SocialMonitor* requests basic demographic information from the participants, which we cannot match with the previously described user request data. By doing so, we can tell that the majority of

[1] `https://play.google.com/store/apps/details?id=de.tudarmstadt.ps.mobilesocialprefetcher` [Accessed: November 19, 2018]

our participants live in Germany, India, and France. Further, 58% of the participants are male and 42% are female. They have an average age of 24.5 years with a standard deviation of 10.2 years.

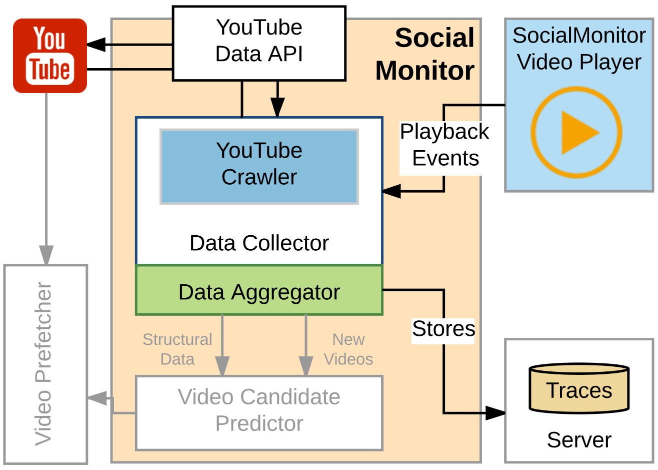


Figure 16: Conceptual architecture of the *SocialMonitor* Android app

Figure 16 exhibits the conceptual architecture of the *SocialMonitor*. In the following, we introduce the *SocialMonitor*'s main functional components. The *Data Collector* regularly retrieves information about recently watched YouTube videos as well as newly subscribed YouTube channels and canceled subscriptions. To accomplish these tasks, we crawl the list of subscribed channels and the watch history of the participants using the YouTube Data Application Programming Interface (API) v3[2]. The retrieved information is passed to the *Data Aggregator*, which filters, anonymizes, and stores the data in a local database on the device. A YouTube user's watch history can contain only one entry per video even if it has been watched multiple times. Thus, one important filtering task is to detect if a video has been watched twice with a different timestamp or if the watch events have been crawled and stored already by a previous crawl. This methodology detects if a video has been watched again in the current crawling interval. The *SocialMonitor* uploads the participant's data to the *Traces Server* every hour, if the device is connected over Wi-Fi. In the case that no Wi-Fi is available, the data is stored at most three days before it is sent using the device's cellular connection. Hence, Wi-Fi connectivity becoming available or unavailable triggers a mechanism transition (cf. Section 2.6). By using a YouTube user's watch history, it cannot be determined on which device a video was watched. Therefore, the *Social Monitor* collects all videos watched while the user is logged-in using, e.g., a mobile device, a browser, or a smart TV. However, this information is not necessary to derive insights and determine user-specific prefetching policies in general. Furthermore, the main share of YouTube requests comes from mobile devices[3]. In addition to the previously described components, the *SocialMonitor* provides a custom video player. We encouraged the participants to use YouTube's website instead of the YouTube application on their smartphones, allowing to choose the *SocialMonitor*'s video player. Thereby, further information about initial buffering times, pausing, fast-forward, screen rotations as well as if a video was watched on the smartphone,

[2]https://developers.google.com/youtube/v3/ [Accessed: November 19, 2018]
[3]https://www.youtube.com/yt/press/statistics.html [Accessed: November 19, 2018]

was determined. However, only a minority of the participants used this video player. Accordingly, the playback-specific information is not available for most video playbacks.

Some participants participated only for a short time or did not use YouTube actively. Therefore, we filtered out some participants for our further analysis. We consider only participants that participated for at least two consecutive weeks and watched more than one video per day on average. In addition, we removed one participant for whom more than 10% of the watched videos have been removed from YouTube mostly because of copyright infringement. This leaves us with 27 participants. Figure 17 depicts the participant's watch events and thereby indicates their participation duration Note that the duration and the number of participants surpass the related work which reports about 10-15 users observed during 10 days - 8 weeks [DZW+14; WRT+15; WSS+16].

Table 6 shows relevant statistics of our participants. We can see that the participants are quite diverse in participation duration, the number of the subscribed channels (#Subscriptions), and the number of channels from which they requested videos (#Channels watched). The average participation duration is 145.8 days (median). During this time, videos were watched on 123 days (median). The videos are provided by 464.5 different channels (median) with a large standard deviation of 763.9. The mean number of subscribed channels per participant is 23.9. Further, 11.8% of the videos provided by channels the participants have subscribed to were watched, while the maximum share is 70.8% for one participant. The row *%Watched Subs* shows that only a minority of 10% (median) of the videos from subscribed channels is watched. The rows *#Un-/Subscribe events* show how often users un-/subscribed channels during the user study.

4.3.1 *Dataset Analysis*

In the following, we focus our analysis on user behavior relevant for prefetching. Table 7 depicts the results of our descriptive analysis. The table's values refer to a daily scale and provide an overview of the participants' behavior. On average, 9.7 videos are provided to the users by subscriptions while the median is only 5. Overall, the participants watch 8.1 videos from 5 channels (mean). However, they watch only 16.7 of the videos offered from subscriptions on average and 83.3 from other sources. Interestingly, the mean values of these metrics indicate that at least half of the users

Metric	Mean	Median	StdDev	Minimum	Maximum
Participation (days)	219.5	145.8	193.4	26.2	639.8
#Days with views	151.7	123	129.2	22	449
#Subscriptions	23.9	10	43.2	0	172
#Channels watched	882.6	464.5	763.9	8	2,410
%Watched Subs.	11.8%	5.3%	17.1%	0%	70.8%
#Subscribe events	24	11	43.4	0	173
#Unsubscribe events	23.9	10	43.2	0	172

Table 6: Statistics of the user study participants

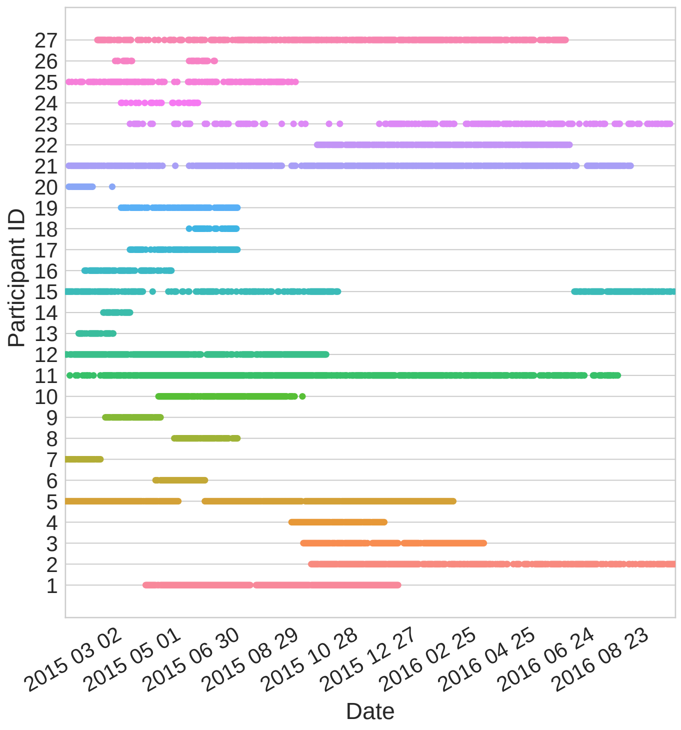


Figure 17: Video watch events of the user study's participants

do not watch videos from subscribed channels at all. The watch time shows that the distribution is shifted to the left since the median is 74.6 minutes while the mean is severely higher with 172.7 minutes. Please note that we do not have the watch time per video available. Thus, we calculated the *Watch time* as the sum of the duration of the videos watched. However, the maximum value exceeds the minutes of the day indicating that this user did not watch the videos entirely but skipped videos, which is a typical behavior for YouTube users observed before [KH14].

4.3.1.1 *Subscriptions*

It is a valid assumption that most users request popular content, next to content of individual interest. However, in our dataset, only 15 channels have been subscribed by three users and only two channels by four. This acknowledges the need to address content belonging to the long-tail of the video popularity distribution. Figure 18 depicts the CDF of the share of watched videos from subscribed channels. Here we see that from about 20% of the subscribed channels only 34% or less of the uploaded videos are watched. Note that the participants of our user study did not watch most videos from their subscriptions. Surprisingly, only for about 7% of the subscribed channels, the participants are interested enough to watch all videos. Further, for 19% of subscribed channels, no video was watched. Hence, we conclude that a user's channel subscription status alone is not an effective feature for deciding whether to prefetch a video. We argue that channel affinity cannot be expressed binary, i.e., subscribed or not, but the ratio of videos watched from a subscribed channel is more informative. This ratio is promising to determine if a video should be prefetched. Not

Metric	Mean	Median	StdDev	Minimum	Maximum
Subs. videos offered	9.7	5	69.0	0	6,293
Number of views	10.5	8.1	12.2	2.9	69
#Subscription views	3.6	0.7	10.1	0.1	48.32
#Channels watched	7.2	5	8.5	0	80
%Subs. watched	16.7%	0%	28.0%	0%	100%
%Others watched	83.3%	100%	28.0%	0%	100%
Watch time (min.)	172.7	74.6	269.3	0.15	3,644

Table 7: Participant statistics on a daily basis

that this ratio can continuously fluctuate and, hence cannot be adequately described by descriptive statistics.

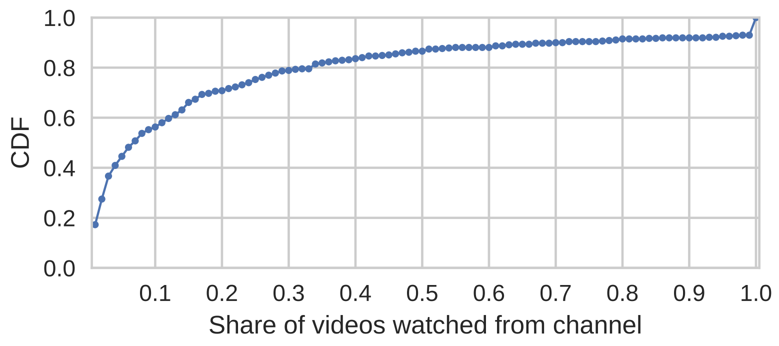


Figure 18: CDF of videos watched from subscribed channels

4.3.1.2 *Video Age*

We analyze the video age, i.e., the time between the video was uploaded and the time it was watched by the participants to constrain the potentially large number of videos that come into consideration for prefetching. Figure 19 depicts the distribution of the watched videos' age measured in days. In the figure, each bar corresponds to the requests of one participant. The x-axis depicts the participant IDs ordered by the number of their mean daily views. Thus, users 2, 23, and 9 have watched most YouTube videos among the participants. While there is a tendency to watch videos younger than three months, for most participants, the median age is between 10^2 and 10^3 days. Surprisingly, this indicates that most videos older than one month are watched and, therefore not the most recent ones.

YouTube video requests can have a variety of reasons, e.g., a new video on a subscribed channel, a search for an ephemeral interest, or a share on Facebook. Since the user has already expressed interest in a YouTube channel by subscribing to it, we further looked into the age of videos from subscribed channels. Figure 20 depicts the participant's watch events similarly as Figure 19 but contains only video watch events that belong to videos from subscribed channels. Note that the unit of the y-axis is hours instead of days. The reason for this is that the video age is lower compared to the general age of all videos watched. The maximum median value of 10^4 days refers roughly to 400 days. The number of depicted devices in Figure 20 is lower compared with Figure 19 because six participants in our study did not watch videos from subscriptions at all. About half of the participants watch subscribed videos with a median age of about 10^3 hours, i.e., 42 days. For seven participants, a median of about four days is observed, demonstrating a diverse behavior. However, for distinct participants, the second and third quartile are narrow, which suggests a stable per-user behavior. To investigate this further, we divide the participants' standard deviation by their mean. This results in values smaller than nine for all participants except participants 3, 4, 5, and 6 which range between 38 and 80. Hence, the ratio of the standard deviation and the mean is quite similar among the participants. Summarizing, we conclude that videos from subscriptions have to be prefetched within the first 7 hours after their upload to not miss more than 25% of videos requested, e.g., by participants 3, 4, 5, and 6 which are among the most active YouTube users in this study.

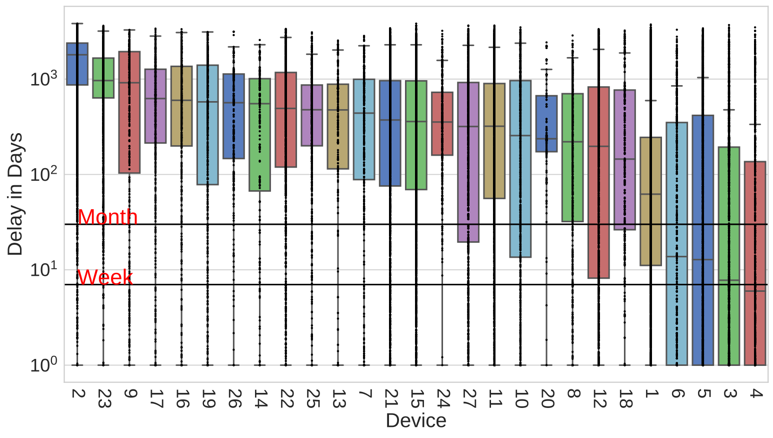


Figure 19: Age of all videos watched for distinct users (days)

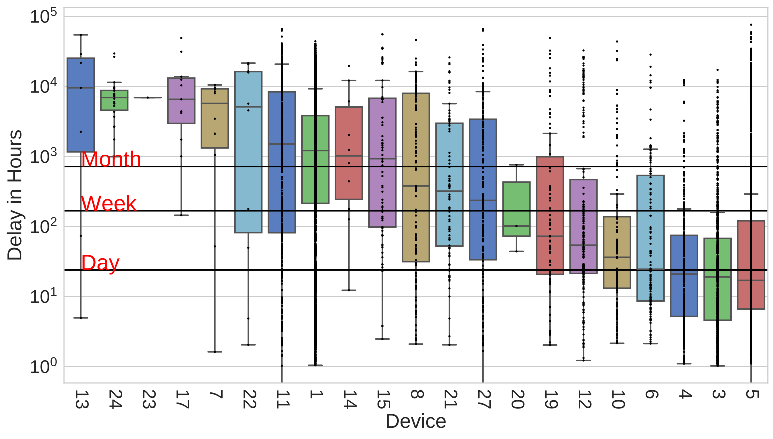


Figure 20: Age of subscription videos watched per users (hours)

4.3.1.3 *Video Origin*

A user's video request can have multiple causes, as mentioned before. From the user data at hand, we can determine if a video belongs to a subscribed channel or not. However, we observed a substantial number of video views to channels the users have not subscribed to but regularly request. Therefore, we deduce that the participants have a non-ephemeral interest in these channels. In the following, we refer to such YouTube channels as *pseudo subscriptions*. Overall, we distinguish between three different video categories regarding their origins: *1)* The video is published on a YouTube channel that a user has subscribed to. *2)* The video comes from a channel that the user regularly watches videos from but has not subscribed to. *3)* The video does not belong to the previous two categories and, therefore is considered as a random view and may come from a website embedding, a YouTube search, or a recommendation on the YouTube landing page.

In general, an additional video source is the related video list of a video which might motivate the user to watch another video. We deliberately neglected this category since it cannot help prefetching in advance but only very shortly before the related video is watched. Distinguishing category 1 from the other two categories is trivial as we know a user's subscribed channels. To distinguish between categories 2 and 3, we set the following two criteria for a match with category 2: First, the user has watched more than one videos of the channel. Second, the user has at least watched 80% of the videos published on this channel. Hence, the first video watched from a channel does not render the channel to a pseudo subscription. Note that the channel may not stay a pseudo subscription forever. Thus, videos from this channel may change to match category 3. Similarly, a category 3 video can also become a category 2 video.

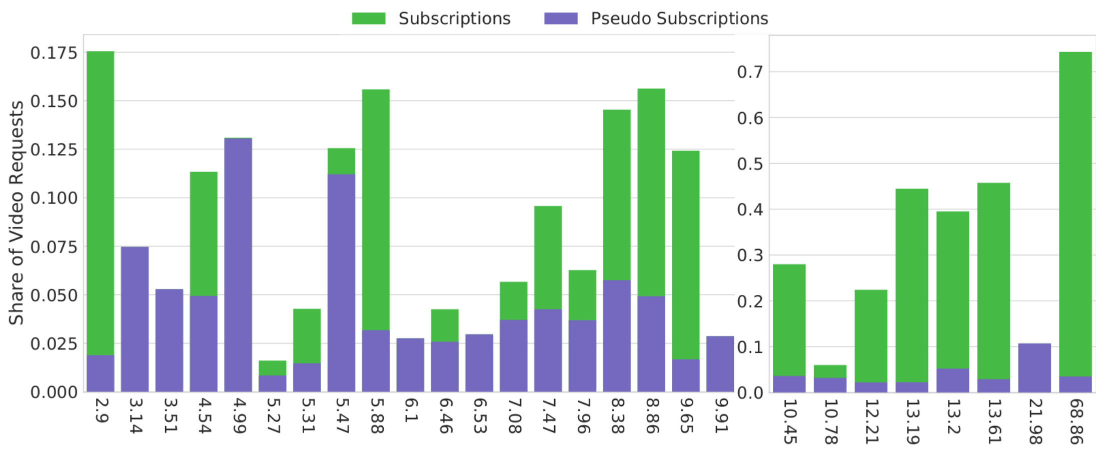


Figure 21: Share of videos requested per subscribed (green) and pseudo subscribed channels (blue), stacked per user and ordered by the users' average number of requests

Figure 21 depicts the share of videos belonging to the previously introduced source categories. Each bar belongs to one participant and the x-axis depicts the mean daily requests of the respective participant. Here, the share of videos from subscriptions is colored in green. The blue share belongs to videos from pseudo subscriptions. The remainder of requests do not belong to subscriptions or pseudo subscriptions and, thereby explain the gap to 1. Note that subscriptions and pseudo subscriptions cannot explain the majority of requests, which, therefore seem too unpredictable for a prefetching mechanism. For completeness, we provide a figure in the style of Figure 21 with, e.g., videos coming from the related video list in the appendix in Section A.1. However, subscriptions are responsible for about 10% of views for most participants with less than eight views per day. For participants watching more than seven videos per day, a higher share of videos from subscriptions is depicted. Four participants that requested more than ten videos per day reach more than 33% of their videos due to subscriptions. For the heaviest YouTube user among our participants, about 70% of the video requests can be explained by subscriptions. Videos can only be prefetched if user interest can be estimated in advance. Thus, we conclude that the bars shown in Figure 21 are the individual participant's maximum prefetching potential. This potential is very diverse and ranges between 3% and 77%.

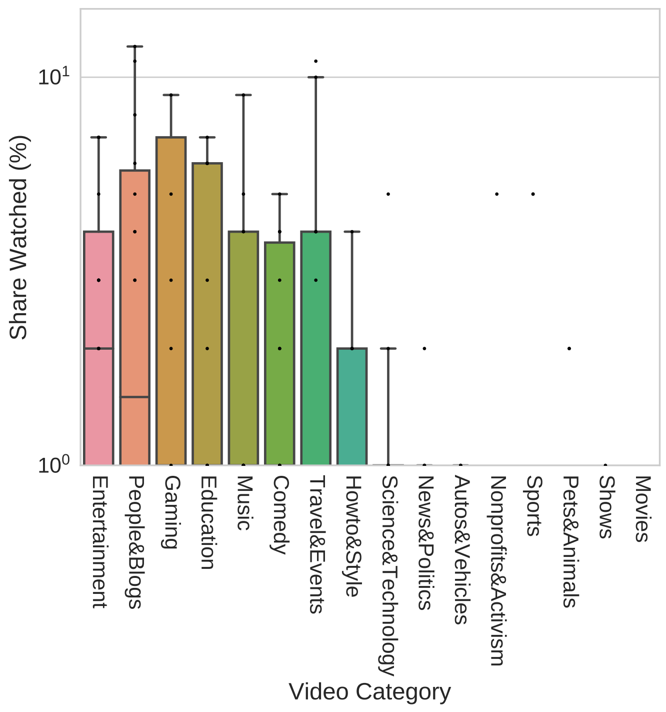


Figure 22: Watched subscriptions (%) by YouTube video category, the y-axis is log-scaled

A further interesting observation is that the participant's likelihood of requesting a video depends on its category. Figure 22 shows for each YouTube category the share of videos watched that are offered by subscribed channels. A YouTube category can be chosen out of a set of categories by the video uploader. Since not all participants watched content form all YouTube categories, the data is quite sparse. The categories Entertainment, People&Blogs, Gaming, Education, Music, and Comedy show a large share of watched videos compared with the other categories. We note that mostly videos from entertaining categories are requested and less from informative categories. Hence, we argue that is important to consider the watch ratio of different categories or even distinct channels for a prefetching mechanism to be efficient.

4.3.1.4 *Repeated Video Requests*

Videos that are repeatedly watched are especially valuable for prefetching as they increase the Cache Hit Rate (CHR) more than videos that are only watched once. This motivates our next analysis on repeated requests to the same video. Figure 23 exhibits the request frequency of videos which have been requested more than once, grouped by their YouTube category. On the one hand, videos belonging to Music and Entertainment show a high re-watch behavior. One explanation for this is that

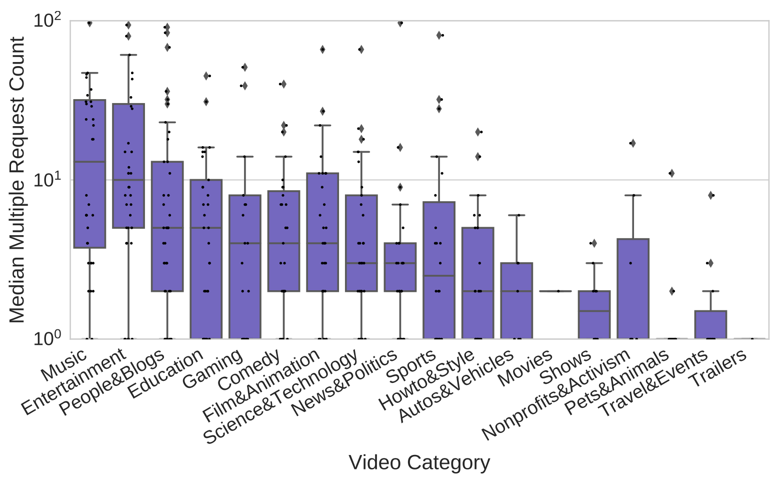


Figure 23: Videos requested more than once per YouTube category

these videos belong to entertaining categories. In contrast to this, informative video categories, e.g., Travel&Events, News&Politics, and Shows exhibit rarely repeated requests. This acknowledges our results from the previous Section (cf. Section 4.3.1.3), where entertaining videos showed to be watched most often out of all videos offered by subscribed channels. In Figure 23, music videos show the highest re-watch behavior by a median of 13-times. This confirms that most requests on YouTube address music videos [KKH17] with about 42%. Hence, we conclude that the prefetching system should not delete videos belonging to entertaining categories after they have been watched. One way to achieve this is by a Least Recently Used (LRU)-managed cache that stores prefetched videos and keeps repeatedly watched videos in the cache.

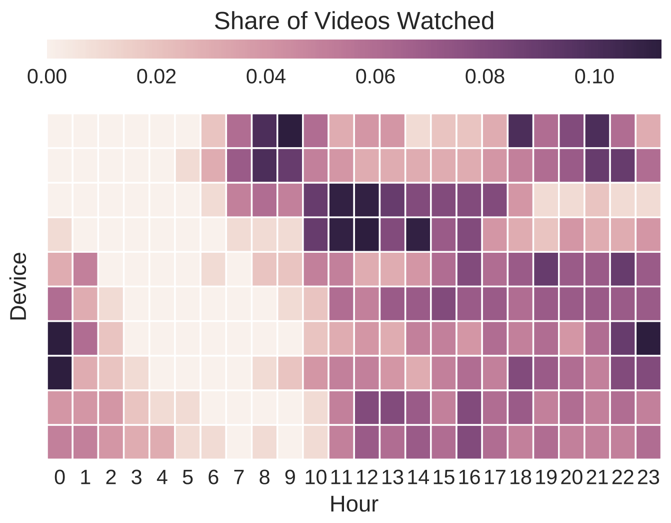


Figure 24: Views over the hours of the day

4.3.1.5 *User Request Time*

One important task of a prefetching mechanism is to timely prefetch content interesting to the user. Therefore, it is essential to know, when a user typically watches videos. Figure 24 depicts a heat map of our participant's views. The x-axis depicts the hours of the day for the ten most active participants measured by their mean views per day. The fields with the darkest colors, i.e., the hotspots range between 11%–17% due to the robust coloring scheme. For all participants, we observe hours with an increased likelihood of watching videos, e.g., 30% of views between 7 am and 10 am for the participant depicted in the first row. We present similar additional heat maps for distinct YouTube categories in the Appendix in Section A.2. To be aware of these hotspots is especially important for efficient prefetch scheduling in case of users that watch videos briefly after their upload.

4.4 DESIGN DECISIONS

In the following, we derive requirements for a prefetching mechanism based on the analysis conducted previously in this section.

D1) Videos older than one month should be removed from the candidate list, as it is unlikely that the user watches them compared with younger videos

In the following, we assuming a video candidate list that defines which videos should be prefetched. It is ordered according to the videos likelihood of being watched by the user. A simple First In - First Out (FIFO)-based mechanism would emphasize the oldest videos and prefetched them first. Though, this is not an efficient policy since a large number of videos watched from subscribed channels is younger than one month (cf. Section 4.3.1.2).

D2) The user's affinity to a subscribed channel must be considered during the content selection for prefetching

The next simple mechanism would be Last In - First Out (LIFO)-based, which is likely to show an increased prefetching efficiency compared with FIFO. However, just prefetching the most recent videos does not consider the user's various channel interests as shown in Figure 18. Thus, not only recency and subscribe status but also the channel affinity has to be considered when selecting content for prefetching. We define this ratio as $\frac{\text{videos watched from channel}}{\text{videos published by channel}}$ for a time interval starting with the first request of the user for this channel.

D3) Pseudo subscriptions must be considered as relevant sources of video views

While our participants subscribed to channels, only 16.7% of the videos published by these subscriptions are watched, as shown in Table 7. For the majority of the participants, videos watched from not subscribed channels dominate. In Section 4.3.1.3, we observed that a major share, i.e., up to 80% of videos are watched from a set of non-subscribed channels. Therefore, we consider not just subscriptions but also other channels from which the user regularly watched videos.

D4) Videos should be cached and kept in the cache after they were watched

We observed that, for some content, the participants seem to have a mid-term to long-term interest, leading to repeated requests to music and entertainment videos as shown in Section 4.3.1.4. Therefore, we deduce that videos should not be deleted after being watched but kept in a simple LRU-managed cache. This seems to contradict the advice of Gouta et al. [Gou+15] to remove videos after they have been watched from the local storage. Note that the authors did not consider music videos in their analysis which belong to the most repeatedly watched video category on YouTube as shown in Section 4.3.1.4.

D5) Videos must be prefetched timely after their upload and before the preferred times when the user watches videos

To serve videos by prefetching, they have to be downloaded timely to the user terminal. Based on the results of the analysis presented in Section 4.3.1.2 and 4.3.1.5, we find this design choice essential for an efficient prefetching mechanism.

4.5 SYSTEM DESIGN

This section discusses the design of vFETCH. To this end, first, the high-level architecture is presented. Second, a functional overview is given followed by an evaluation and a discussion of the results.

Assumptions and requirements

For the design of vFETCH, we consider only available technologies that are used by the industry to a certain extent already. Specifically, we assume that the video requests a user has made, i.e., the user's watch history can be monitored. Further, we assume that video metadata, as well as channels a user has subscribed to and their published videos, can be retrieved. This functionality is available by the YouTube Data API. On the mobile terminal's side, we require storage capabilities so that vFETCH can download videos to the device. Hence, we do not require a permanent but a temporary Internet connection that allows downloading videos, i.e., Wi-Fi, 3G, or 4G. The video player or the application which contains the video player that we require on the user's device is assumed to check if the requested content is available at the local prefetch storage first, before requesting the content using the device's Internet connection. An implicit assumption we have is that an efficient prefetching model can accurately capture a user's video preferences.

4.5.1 *Architectural Overview*

Data manager

We present vFETCH's conceptual architecture in Figure 25. The architecture consists of two main components: the *Data Manager* and the *Content Prefetcher*. The *Data Manager* is responsible for retrieving information about watched videos from the connected Online Social Networks (OSNs) and platforms. The metadata and watched videos are retrieved to provide a data foundation for identifying which properties of a video are most interesting to a user. This *Interest Model* is used to rank the prefetch candidate videos concerning the model's estimated user interest per video. Thereby, the user's data is processed on a user-owned device and, therefore the user privacy is preserved. To avoid a loss of information on YouTube's side, vFETCH is envisioned to send back user viewing statistics to YouTube allowing to distinguish between prefetched and actually watched videos as well as playback duration and video quality information. In principle, the *Data Manager* of vFETCH can operate on any kind of user terminal, e.g., smartphones, or stationary last mile hardware such as

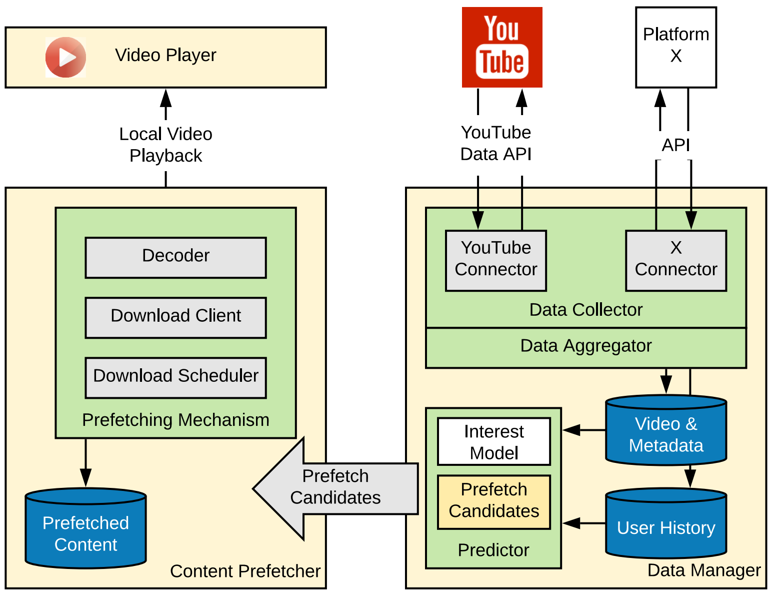


Figure 25: Architecture of the vFetch prefetching mechanism

home routers and cloudlets. In the course of this thesis, we focus on the application on a mobile terminal as the most relevant scenario.

A subset of the prefetch candidates with the highest estimated interest score is passed to the *Content Prefetcher* module that downloads the video content. To this end, it needs to fulfill a set of tasks, such as decoding, downloading, and download scheduling. Thereby, vFetch can emphasize downloading during hours when the mobile terminal is connected to Wi-Fi and charging to avoid battery depletion and draining the potentially limited mobile data plane.

Content prefetcher

4.5.2 *Functional Overview*

In the following, we provide details on the core functionalities of vFetch and the interactions between the involved functional components. In essence, vFetch is understood as a middleware that can provide prefetching functionalities to a set of OSNs and content portals.

4.5.2.1 *Data Collector and Aggregator*

An interface between the vFetch mechanism and the corresponding OSNs or content portals is required to retrieve information on available content. To this end, vFetch implements a *Data Collector* module that has one distinct *Connector* module per platform. Therefore, it uses the access permission of the user to the respective platforms. This way, platform-specific APIs can be addressed, which can have different concepts of, e.g., feeds and subscriptions. For example, YouTube allows the user to subscribe to YouTube channels and shows the videos from these subscriptions on a dedicated page. Facebook allows following persons and groups and presents the content from these sources on the user's Facebook feed. For vFetch, it is essential to be aware of recently published or uploaded content, to allow timely prefetching. Therefore, the *Data Collector* retrieves new information from the platform regularly. This information consists of the following parts:

1. Videos the user has watched in the past, i.e., the watch history
2. Recently published videos on channels that the user has subscribed to
3. Recently published videos on channels that the user has watched, including pseudo subscriptions
4. Metadata of the videos: title, upload date, channel, category

The watch history allows monitoring which videos a user has watched. To identify the source of a watched video, we additionally retrieve the channels a user has subscribed and the videos uploaded on those channels. Further, the videos from channels the user has recently watched are retrieved. For all videos retrieved by the steps before, the metadata is retrieved additionally, including the video's title, its upload date, the channel on which it is uploaded, and its YouTube category. The *Data Aggregator* is responsible for translating the different formats retrieved from the different platforms into a common data representation and to store it into a local database. Furthermore, the *Data Aggregator* serves the *Data Collector* with the timestamp and the entity of the most recently stored database entry. This makes it possible for the *Data Collector* to retrieve new data incrementally and stop the retrieval process if already stored information appears.

4.5.2.2 *Predictor*

The *Predictor* is responsible for estimating the probability that the user will watch a video. To this end, it uses an *Interest Model* that needs to be regularly updated. Recent videos retrieved by the *Data Collector* and stored by the *Data Aggregator* serve as an input to the *Predictor*'s candidate list. The *Interest Model* scores each new video observed on the user's feed. On the example of YouTube, which we will use in the following as the content platform, a set of different candidate videos can be retrieved, as we list in the following:

1. Videos from Subscriptions: YouTube has a subscription feature which allows users to get listed videos from subscribed channels, which they are supposedly interested in.
2. Videos from Pseudo Subscriptions: We define a pseudo subscription as a channel that the user has not subscribed to but regularly watches videos from. Two criteria have to be met: At least two videos must have been viewed from the channel and at least 80% of the videos published on the channel since the first video was watched by the user have to be watched as well. We describe these 80% as the pseudo subscription threshold.
3. Related videos: YouTube lists related videos on a video's web page which can be expected to be of similar content.
4. Playlist videos: Users can create or subscribe to playlists on YouTube containing typically music or video sequences of a common topic and order, e.g., a lecture.
5. Watch later list videos: Users can add videos which they do not want to watch right away add to a list called the watch later list. Hence, the user expresses interest in watching the video later on by adding it to this list.

The participants of our user study rarely watched videos from the watch later list or playlists. Further, the related video list of a currently played video becomes available too late for prefetching in advance. Thus, we exclude videos originating from these content sources from the *Prefetch Candidate* list. Acknowledging the design decisions *D2* and *D3* (cf. Section 4.4), we monitor videos from subscriptions and pseudo subscriptions as promising prefetch candidates. In addition, we filter videos older than 30 days from the candidates since they are less likely of being watched. Thereby, we acknowledge *D1* (cf. Section 4.4). To determine the user's affinity to a channel, the *channel affinity* score is computed. We define it as the ratio of $\frac{\text{videos watched from channel}}{\text{videos published by channel}}$ for a time span starting with the first request of the user for this channel. Accordingly, a user needs to request as lest the three most recent videos uploaded to render a channel to a pseudo subscription. The previously introduced channel affinity ratio is used as a user interest estimate for the videos published on it. Depending on the users watching behavior, the average number of watched videos per day, defined as d, is allowed to be prefetched. We also evaluated smaller and larger values but the average number of views per day showed the best CHR. The d video IDs with the highest estimated user interest are passed to the *Content Prefetcher*. In case two or more videos have the same estimated popularity, the most recent videos are emphasized.

4.5.2.3 *Content Prefetcher*

The *Content Prefetcher* comprises the necessary functionalities to download the video content. This module determines when vFetch downloads videos by keeping track of the user's diurnal connectivity patterns. Thereby, a download can be postponed either Wi-Fi becomes available or the user is predicted to watch videos soon. In the latter case, the *Content Prefetcher* does not wait until a Wi-Fi connection becomes available. Though, only videos with a high-interest score are considered for being prefetched via a 4G or 3G connection. To determine when a user is most likely to watch videos, we feed the previous video watch times into a clustering algorithm. For this purpose, we decided to use the clustering algorithm DBSCAN [EKS+96] because it shows a decent clustering performance[4]. Additionally, it is density based which fits our use case as we are seeking for time intervals in which many videos are watched, i.e., the density of the watch events is high. By identifying the periods when the user is most likely to watch videos, the *Download Scheduler* schedules the video download in advance, preferably during off-peak hours or when Wi-Fi is available. For a set of our participants, we provide a heat map indicating the video watch behavior per hour of the day in Section 4.3.1.5. Note that it is not optimal to download all d videos at the first best moment when Wi-Fi becomes available. By doing so, potential later published videos would be excluded from prefetching them on the same day. Therefore, we distribute the allowed number of videos to download d during the times before the user's watch clusters have been observed. The time interval of four hours before the estimated user watch time is split into sub-intervals of half an hour.

Download Scheduler

From these eight sub-intervals, the one with the highest probability of having a Wi-Fi connection at this time is chosen to execute the download. Once an interval is selected, the *Download Scheduler* assigns up to three videos to be prefetched to this

[4] http://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html [Accessed: November 19, 2018]

interval. Thus, vFetch can prefetch up to three videos per half an hour. We chose half an hour time intervals since they are by any connection suitable to download three YouTube videos, giving each video 10 minutes. The overall number of videos to prefetch is determined by the mean number of videos the user watches per day over the preceding, at most, 28 days. Thereby, downloading unnecessarily many videos is avoided. The prefetched videos are stored in an LRU-managed cache. By doing so, we address design decision *D4* (cf. Section 4.4). This supports vFetch to benefit from, e.g., music videos which are likely to be repeatedly watched.

The *Download Client* downloads the actual video content and handles connection interruptions. Also, it chooses the video's quality since usually a set of qualities are available. On many mobile devices, a decent video quality can be achieved by 720p with 30Hz which we choose as the target video quality. In case this quality is unavailable, the *Download Client* selects the next lowest quality. The *Decoder* module is helping the *Download Client* to find the video's URI, e.g., of the corresponding mp4 file belonging to a certain video ID and quality.

4.6 evaluation

This section presents the evaluation of the vFetch mechanism. The evaluation's objective is to provide a proof of concept of the designed mechanism and to quantify key performance indicators in relevant scenarios. To this end, vFetch is implemented as a prototype to demonstrate the general technical feasibility. In the following, first, the general evaluation methodology is described. Second, the evaluation of vFetch's general efficiency in scenarios of variable mobile device storage capacity is investigated in Section 4.6.2. Third, we vary the number of past days from which we consider videos as prefetch candidates in Section 4.6.3. In the fourth part of the evaluation, we investigate the storage overhead caused by vFetch in terms of the number of stored and watched or not watched videos in Section 4.6.4.

4.6.1 *Methodology*

We evaluate vFetch by implementing it within a discrete event simulator. This has two benefits. First, it facilitates us to use the real-world user traces collected during our user study in Section 4.3. Thereby, we can guarantee that the users are not influenced by, e.g., using a dedicated application with a different look and feel. Note that due to legal restrictions, we cannot allow our *SocialMonitor* application to download videos although this is technically possible as demonstrated in the scope of a demonstrator [KRB+14]. During the simulation, we keep track of the past and the current simulation time to use only information which is available at the simulation time and, hence, no information from the future. Second, a simulator supports the evaluation of a set of different resource and parameter configurations. This is not simple to realize by deploying vFetch as an Android app, since resources are not configurable and parameters need to be fixed. Evaluating vFetch by simulations makes it possible to assume an array of storage and parameter configurations for each participant of the user study.

Since prefetching strives for perfect prediction, all approaches have to be evaluated by an efficiency metric. A common prefetching efficiency metric is the ratio between correctly predicted videos and all predicted videos (cf. Equation 1). However, the precision can be easily tweaked by prefetching only a few videos, e.g., one. In case this single video is requested, the precision would be 100%. Thus, besides the precision, a further metric is needed which expresses the value of the methods. For this purpose, the recall (cf. Equation 2) is commonly used.

Since both metrics: precision and recall are important to consider in the evaluation of vFETCH, we use a metric that combines them, the F1-measure (cf. Equation 3). The F1-measure is the harmonic mean of the precision and the recall and, hence, a robust metric for the quality of the prefetching.

$$\text{precision} = \frac{|\text{watched contents}| \cap |\text{prefetched contents}|}{|\text{prefetched contents}|} \tag{1}$$

$$\text{recall} = \frac{|\text{watched contents}| \cap |\text{prefetched contents}|}{|\text{prefetched contents}|} \tag{2}$$

$$\text{F1-measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{3}$$

4.6.2 *Storage Size and Caching*

The storage capacity of a mobile device is expected to influence vFETCH's performance since larger storage capacities can store more videos for a longer time. To this end, we investigate the impact of different storage sizes on vFETCH's F1-Measure as well as the caching performance given by the Byte Hit Rate (BHR). We decide to use an array of storage sizes that cover a range of mobile devices, including storage-restricted smartwatches and smartphones with large storage capacities. The storage sizes used for the evaluation are $\{0.05, 0.1, 0.5, 1, 5, 10, 50, 100\}$ GB. It is important to note that a large storage capacity does not imply that more videos are prefetched as this number depends on the user's mean number of daily videos watched. Figure 26 depicts the impact of different storage sizes on vFETCH's performance. The first row shows the results for videos from predictable sources, i.e., subscriptions and pseudo subscriptions. Note that to count as a pseudo subscription, a user must watch more than one video and overall more than 80% of the channel videos uploaded since the first video of this channel was watched. Following Li et al. [LSW+12], 80% is a suitable value for the pseudo subscription threshold to assume the user being interested in a YouTube channel.

In the second row, we see the results considering all videos watched, also from unpredictable sources. In addition, we distinguish between two cases: prefetching and prefetching in combination with request-based caching, i.e., watched videos are stored in an LRU-managed cache additionally to the prefetched videos. For each row, the figure exhibits the F1-measure and the BHR). For these experiments, we observe precision and recall to be similar. This is also indicated by the similarity between F1-measure and BHR, because the BHR represents the precision considering repeated requests. From Figure 26's first row, we deduce that given a fixed storage size, introducing caching additionally to prefetching severely decreases the F1-measure and

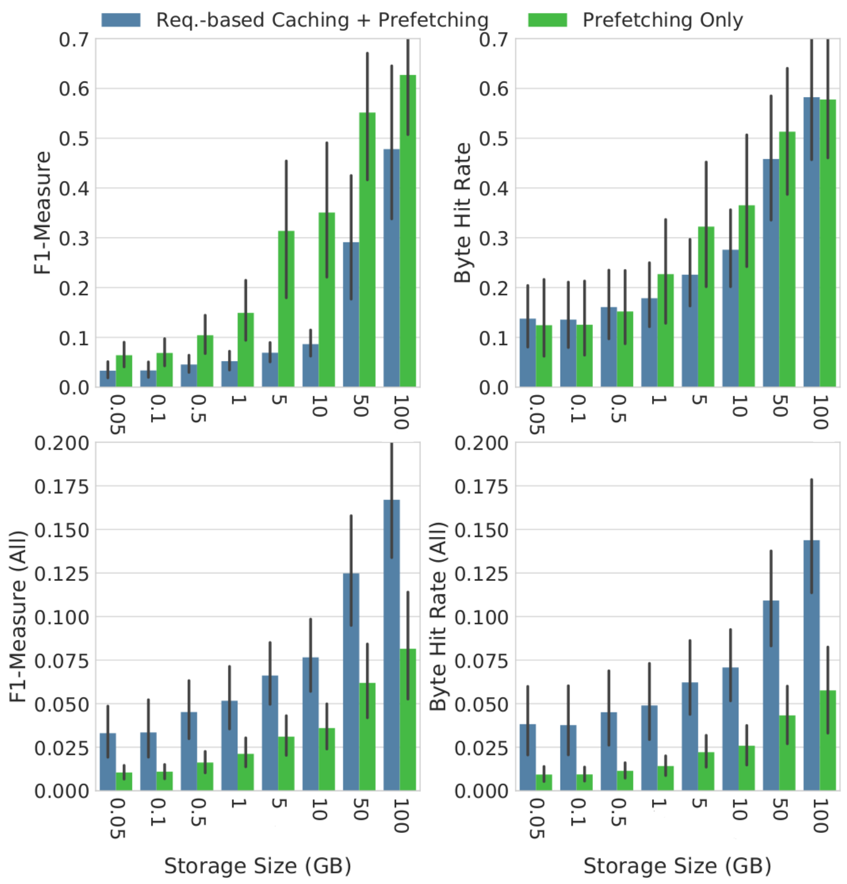


Figure 26: Performance impact of storage size, top: videos watched from predictable sources, bottom: all videos watched

the BHR for predictable videos. Caching combined with prefetching increases the performance metrics 2.8-times on average if we consider all requested videos. However, the absolute values of the F1-measure and the BHR are severely lower compared to the case where we consider only requests from predictable sources. This finding shows that predictability contains more valuable information, which is used by prefetching, than the mere object requests, which drive the LRU cache. Overall, for the mixture of predictable and ephemeral user interests, we measure that prefetching combined with caching results in the best performance. Therefore, we deduce that a differentiation based on predictability of the user interests leads to an adapted use of prefetching and caching. However, we argue that prefetching systems should be evaluated w.r.t. predictable video sources, as shown in the first row. Other sources, e.g., ephemeral user interests or random browsing are hard to predict and, hence, would skew a performance evaluation. vFetch achieves BHRs of up to 58% for very large caches of 100 GB and about 23% for caches with 1 GB. However, both metrics strongly spread among the participants, as the distance between second and third quartile indicates. We leave out the box plots for the sake of clarity but report on our observations. For example, the third quartile for 5 GB cache storage is 52% while the median is only 18.3%. For some participants, the whiskers reach even 100% for cache size $\geqslant$ 5 GB.

Since the BHR reflects the precision of vFetch considering repeated requests, the gap to 1, i.e., $1 - \text{BHR}$ indicates the number of videos downloaded but not watched. Figure 26 shows the average BHR for all participants and their common 95% confidence intervals. Additionally, to the exhibited results, we evaluated the same configuration while omitting pseudo subscriptions. Interestingly, the results significantly differ as the median BHR was almost 0. Thus, we deduce that pseudo subscriptions are the dominant source of videos for many participants. Summarizing, we demonstrated that vFetch outperforms existing prefetching mechanisms since vFetch shows a BHR between 0.3% and 14%, while the related work shows $\leqslant$ 0.03% [WSS+16; CL09; LSW+12] when applied to YouTube as demonstrated by Wilk et al. [WSS+16]. vFetch shows high-performance measures for storage sizes $\geqslant$ 5 GB, depending on the users. While few users show BHRs $\geqslant$ 90% for 5 GB, on average half of the prefetched videos are watched at 50 GB storage size.

4.6.3 *Watch History*

The watch history is a list of videos the user has watched in the past and is ordered by the watch time. It can be defined as a Time To Live (TTL) cache, i.e., the video watch time is refreshed upon a request. In case a video is not requested again, it remains on this list for a maximum lifetime denoted watch history threshold. We seek to adapt to the potentially dynamic user request behavior, e.g., during public holidays or vacations by using this threshold. Furthermore, vFetch determines the number of daily prefetches based on the watch history by using the average number of requested videos per day within this time interval. Accordingly, if the user requests fewer videos, fewer videos are prefetched.

Figure 27 exhibits the influence of the watch history threshold on vFetch's F1-measure and CHR. Here, we decided to investigate two distinct cases. The figure's first column shows the results for 1 GB cache size and the second column in the case of 50 GB. In addition, we depict both cases: pure prefetching and request-based caching combined with prefetching. When increasing the watch history threshold, we observe diminishing returns on the F1-measures and the CHRs. The results indicate that no optimal value for all users exists. However, we suggest a window of two weeks to balance performance and length of the watch history. Please note that for window sizes greater or equal than two weeks, prefetching surpasses request-based caching in combination with prefetching in terms of, both, CHR and F1-measure when comparing their performance with each other.

4.6.4 *Storage Overhead*

In the following, we investigate the storage overhead of vFetch. We measure this overhead by evaluating the share of bytes fetched and watched by a user as well as the prefetched and not watched bytes. Figure 28 depicts the results for the case of 50 GB storage. The participant IDs denote the x-axis, and the y-axis depicts the share of downloaded video data. Here, 100% represents the total prefetched bytes. Values higher than 100% are possible if videos are repeatedly watched. We see that the participants 23, 25, and 18 have watched videos multiple times.

Besides, we observe for half of the participants that the overhead in terms of bytes from prefetched but not watched videos is below 80%. We consider an overhead $\leqslant$ 80% as reasonable for a prefetching system, i.e., $\frac{1}{5}$ of prefetched bytes are

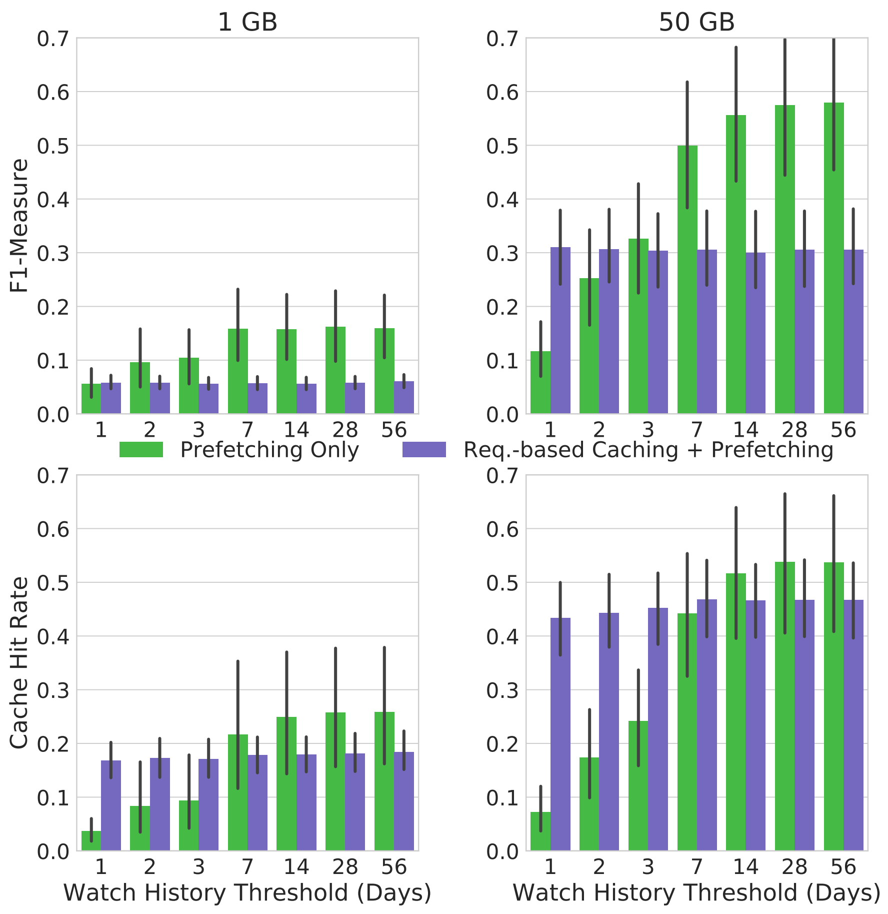


Figure 27: Performance impact of watch history threshold

consumed. Wi-Fi is about 23-times less power demanding than Long Term Evolution (LTE) [HQG+12]. Since vFETCH has an average overhead of 80%, it is about four times more energy efficient compared with streaming the videos over LTE. For ten participants, the share of fetched & watched bytes is below 10% and, hence, considered as not efficient. The share of vFETCH's prefetched & watched bytes is for three users even higher than the prefetched & not watched share, i.e., the overhead is less than 50%. Note that vFetch has a low average overhead of 70% compared with 82% in case of CPSys [Gou+15].

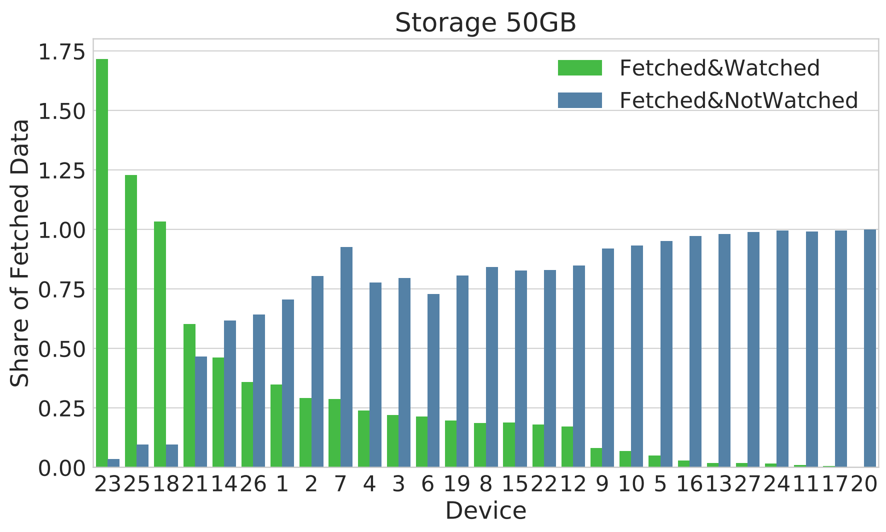


Figure 28: Share of fetched data watched and not watched

4.7 SUMMARY AND DISCUSSION

In this chapter, we first presented a thorough analysis of a user study focusing on people's YouTube usage behavior. From the insights learned, we derived requirements on the design of an efficient prefetching mechanism. Acknowledging these requirements, we designed the privacy-preserving and content-oriented prefetching mechanism vFetch and evaluated it using YouTube as an example. To the best of the author's knowledge, vFetch is the first mechanism that combines cellular network offload, an accurate user interest model and, at the same time, does not require revealing personal data to a third party. To achieve the latter point, vFetch runs locally on the user's device and learns the user's watching behavior, interests, and the mobile terminal's connectivity pattern. Thereby, vFetch differs from content recommender systems that store multiple users' data outside the user's device, e.g., in a cloud to correlate the user behavior of a vast amount of users. Instead, the proposed mechanism relies solely on content features, channel crawling, and locally available information. Thereby, vFetch outperforms existing works considering Byte Hit Rate (BHR) and overhead. vFetch adapts to changing user interests by continuously monitoring the user's behavior. Further, the number of videos to prefetch is tailed to the user's watching habits, to allow timely prefetching. vFetch already adapts to the number of videos watched and the time when videos are watched. Similarly, the *Download Scheduler* considers the user's daily connectivity pattern to identify efficient time intervals to download video content. While we investigated vFetch as a stand-alone-application in this chapter, it can be extended to work in cooperation with the mobile network operator to further decrease mobile network load. In case the mobile network operator supports cooperating with vFetch, a mechanism transition (cf. Section 2.6) from the stand-alone mechanism to the cooperative mechanism is triggered. We describe how this cooperative mechanism is envisioned in the appendix in Section A.3.

We demonstrated that using pseudo subscriptions as a source of prefetching candidates results in a significant performance increase compared to using videos from subscribed channels. Our trace-based evaluation shows the sensitivity of vFetch to key parameters such as the watch history lifetime, different cache sizes, and the performance impact of request caching in addition to prefetching. vFETCH achieves an average CHR of 54% on a 50 GB storage. The trend of storage costs decreasing faster than bandwidth costs [ER15] are expected to make vFETCH more beneficial in the future. Furthermore, assuming more storage to become available on modern smartphones[5,6], is expected to further decrease the storage footprint of vFETCH on modern mobile devices. We observe diverse results for different users, indicating that prefetching is efficient for only a subset of our participants, i.e., where predictability can be leveraged. The average overhead of the prefetching system of 70% when applied to all users remains below the overhead of 82% from the comparable related work in [Gou+15].

[5]The iPhone X is available with 256 GB storage: `https://goo.gl/neFV4z` [Accessed: November 19, 2018]

[6]Smartphones with 512 GB storage: `https://goo.gl/NjsooB` [Accessed: November 19, 2018]

5 EFFICIENT AND PROACTIVE NETWORK CACHING WITH PROCACHE

This chapter presents the design, discussion, and evaluation of a novel proactive caching mechanism for Video-on-Demand (VoD) content that adapts to a continuously varying content popularity distribution, i.e., the fluctuating number of requests to videos. This is caused by, e.g., changing user interests, heterogeneous content popularity life cycles, and new content being published [FBA11]. The term "proactive", in the context of ProCache, refers to the flexible storage allocation for content of specific categories which our mechanism continuously adapts by considering the current content popularity distribution. We denote the proposed mechanism as ProCache. This mechanism aims at improving the cache performance and reducing the transmission latency for individual caches and cache hierarchies. Thereby, network traffic can be kept local, and the amount of potentially costly transit traffic is reduced. Besides, the Quality of Experience (QoE) is positively influenced by a decreased latency that facilitates a fast video playback start and high video quality.

Several scientific publications present parts of ProCache, that jointly build the foundation for this chapter. The general functionality and its application on typical topological cache hierarchies used by Content Delivery Networks (CDNs) are presented in [KPR+18]. While the mechanism presented in this paper focuses on video content in general, we address the dominant content category on YouTube, namely music [CM13; CDL08b] in [KKH17] and proposed a Deep Neural Network (DNN)-based extension in [KWR+18].

Furthermore, a number of student theses contributed to the publications mentioned above and parts of ProCache as presented in the course of this chapter [Pfa17; Wer17; Kru16].

Chapter outline

The remainder of this chapter is structured as follows. In Section 5.1, we provide a general overview of proactive caching and its benefits. These benefits make proactive caching especially valuable for a set of use cases, that we discuss in Section 5.2. Section 5.3 discusses ProCache's system design and key functional components. In Section 5.4, we evaluate ProCache using a real-world YouTube request trace collected in a large European mobile network. This chapter is summarized and discussed in Section 5.5.

5.1 CONCEPTUAL OVERVIEW

We highlight the two key properties of ProCache by comparing it to traditional reactive caching, i.e., Least Recently Used (LRU) which is a simple and common eviction policy that shows decent caching performance for video content in the literature [SKL16; LAO+14]. The first property is ProCache's ability to emphasize popular content by preventing it from a premature eviction.

The performance-increasing effect of cache divisioning, which we leverage in the design of ProCache, has been demonstrated already by Adaptive Replacement

Cache (ARC) (cf. Section 3.2.1.3) and Segmented Least Recently Used (SLRU) (cf. Section 2.2.1.2). However, they either consider just two cache divisions or a fixed-size cache size allocation. An inherent limitation of reactive caching and a fixed cache storage allocation is the dynamic character of a real-world workload, where the number of content belonging to a category is likely to change continuously. Therefore, a fixed cache division scheme cannot adapt to growing popularity of a category or the decreasing number of popular content in the respective categories. To this end, ProCache introduces dynamic cache storage allocation as a key concept. This makes it possible for cache divisions to extend if they contribute to a higher extent to the cache performance, while cache divisions that experience decreasing popularity can shrink.

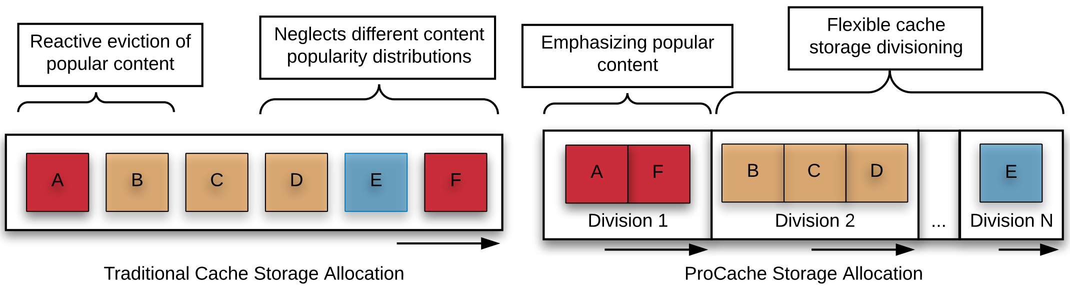

Reactive eviction of popular content
Neglects different content popularity distributions
Emphasizing popular content
Flexible cache storage divisioning
A
B
C
D
E
F
A
F
Division 1
B
C
D
Division 2
...
E
Division N
Traditional Cache Storage Allocation
ProCache Storage Allocation

Figure 29: Traditional vs. ProCache's cache management

Figure 29 gives an example that considers six video contents. The color of the cached items indicates their popularity. The red content belongs to the most popular content. Orange indicates the content of medium popularity and blue indicates unpopular content from the popularity distribution's long tail. Traditionally, caches consist only of one storage division responsible for the entirety of cached content. ProCache differs from this by splitting the cache storage into multiple cache divisions. As indicated by the black arrows, new content enters the left and is evicted on the right end of the respective cache division. On the left side, the figure depicts the traditional cache. We see that content A is the most recently requested content at this time, while the content B, C, D as well as E are less popular. However, since they have been requested recently, they have pushed the general popular content F close to its eviction. While this behavior enables the cache to anticipate rapid changes in the content popularity distribution quickly, e.g., flash crowds, it tends to penalize content F disproportionately. A more desirable behavior is to evict less popular content first. This motivates the cache division concept of ProCache. Thereby, content categories that show different popularity distributions, e.g., very popular, popular, and unpopular content, are stored separately. In the example given by Figure 29, ProCache protects content A and F from being evicted by an ephemeral popularity increase of less popular content. Note, the cache allocation does not necessarily correspond to the long-term popularity of content but can also orient on content categories if they show distinct popularity dynamics.

The sketched example uses LRU to compare ProCache for simplicity. However, state-of-the-art cache admission strategies prevent content from being cached upon the first request. This property is known as *scan resistance* [MM04] and realized, e.g., by a cache-on-second-hit admission policy (ref. Section 2.2.1.1). The balance between agility and scan resistance is a challenging task in the design of caching strategies;

however, the sketched example is also valid for scenarios where unpopular content is requested more than once since it can still pollute the cache with low-popularity content and, thereby decrease overall performance. Hence, ProCache is expected to show superior performance over simple cache-on-second-hit admission policies as well.

5.2 USE CASES

In the following, we discuss in which use case scenarios ProCache can operate. Either the CDN or the Internet Service Provider (ISP) commonly accomplishes the task of video content caching. Figure 30 depicts the location of ProCache for these two use cases.

The first use case addresses how CDNs can use ProCache. Thereby, the CDN can operate its cache appliances either outside or inside of broadband access ISPs while we consider caching outside of the ISP as the most common and regular case since large ISP hesitate to allow CDNs operating inside their network [HH11]. In this case, ProCache can be used instead of traditional caching mechanisms.

The second use case is operating ProCache inside an ISP. In this use case, the ISP can provide own caching appliances for either its own video services or offer its caching resources to external entities by a pay-per-use payment model. Since encrypted content transmissions are the default in today's Internet, the ISP cannot cache bypassing content transferred from the CDN to the ISP's customers. However, an ISP can still use caching for its own content or cooperate with the content provider to either allow the CDN using ProCache as a service to the CDN or to collaborate and, hence trust each other. Here, they can transfer the content provider traffic by encrypted transmissions to the ISP's cache appliances to cache it there in unencrypted form. The encryption endpoint for the user would be the cache appliance. Thereby the CDN allows the ISP to cache its content transparently. Hence, a collaboration requires privacy consents and appropriate technical measures of trust such as cryptographic certificates as well as strong encryption schemes.

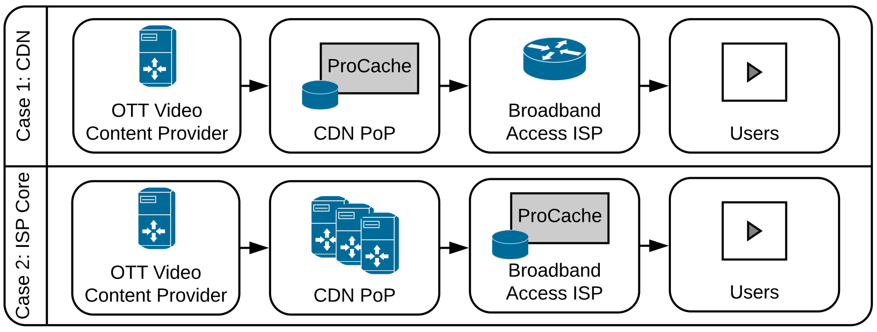


Figure 30: Two major use cases of ProCache, arrows indicate video content transmissions

5.3 SYSTEM DESIGN

This section discusses the system design of ProCache. Therefore, we first present ProCache's high-level architecture and its two key components. Second, we provide a functional overview followed by an evaluation and a discussion of the results. In the course of this chapter, we propose and discuss two workload-oriented application scenarios of ProCache. The first scenario addresses a mixed workload covering requests to various categories. In the second scenario, we emphasize using ProCache for music content specifically, since this is the dominant YouTube video category regarding requests [KKH17; CDL08b].

Assumptions and requirements

For the design of ProCache, we consider only available technologies that are used by the industry to a certain extent already. Specifically, we require computational capacities at the cache storage servers, to facilitate executing ProCache. Further, we require the YouTube category of a video, as a specific metadata available by the YouTube Data Application Programming Interface (API)[1]. In addition, we assume videos to be segmented, which is given in reality by HTTP Adaptive Streaming (HAS)-approaches such as Dynamic Adaptive Streaming over HTTP (DASH) (cf. Section 2.3.1). A common approach in the related work is to consider just one video quality being available. This increases the comparability in contrast to an arbitrary- or platform-depending number of video qualities, which can quickly reach more than 100 [KZS15]. We follow the same approach in this thesis. Therefore, we consider just one video quality and the associated video bitrate in the evaluation, which is either 1080p with 30 Hz or, if this is not available, the closest lower video quality.

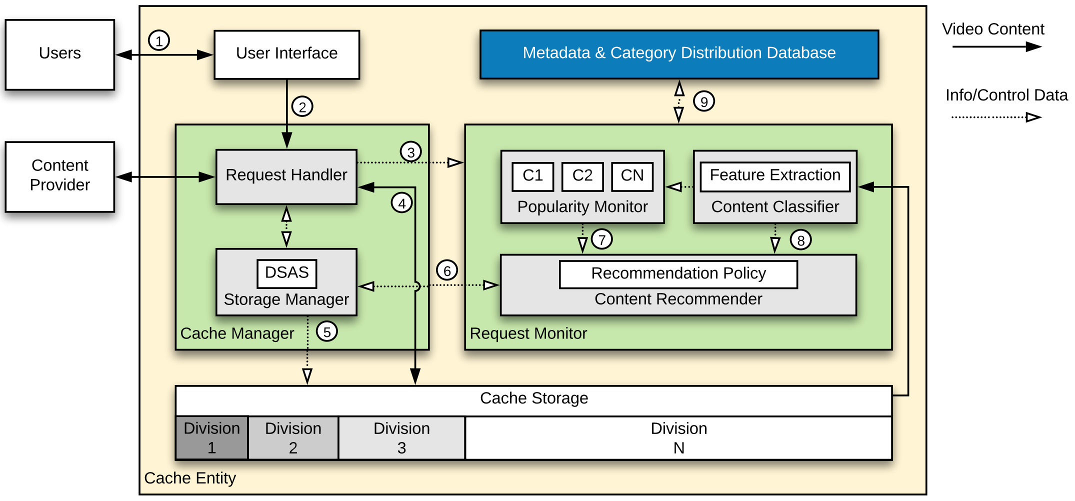


Figure 31: Architecture of ProCache (inspired by [MAS+17])

5.3.1 *Architectural Overview*

We present ProCache's conceptual architecture in Figure 31. It comprises two main components: the *Cache Manager* and the *Request Monitor*. Both modules are encapsulated within the *Cache Entity* that communicates with the *Users* and the *Content Provider*. Below, we detail the work-flow and the key tasks of the involved modules:

[1] https://developers.google.com/youtube/v3/ [Accessed: November 19, 2018]

1. The user requests are received by the *User Interface*, represented, e.g., by an Hypertext Transfer Protocol (HTTP) server offering a Representational State Transfer (REST) interface. Furthermore, the *User Interface* takes care of delivering requested video content to the clients.
2. The *User Interface* parses user requests and sends them to the *Cache Manager*.
3. Here, the *Request Handler* sub-module informs the *Request Monitor* about every incoming request.
4. The *Request Handler* retrieves the requested content by either fetching it from the *Cache Storage* or, if not present, downloads the content from the *Content Provider*. Besides, the *Request Handler* stores all request for a content that is currently fetched from the *Content Provider* so that only one request is sent to the *Content Provider*. As soon as the content is received, the waiting clients get served the content and it is stored at the *Cache Storage*. Besides, the *Request Handler* retrieves the video category from the *Content Provider*, e.g., YouTube.
5. The content request is passed to the *Storage Manager* as well. It decides to increase or decrease a cache division's size based on the popularity of all divisions' content and the used *Division Size Adaptation Strategy (DSAS)* (cf. Section 5.3.2.2). We use the YouTube video categories as a basis for the divisions.
6. The *Storage Manager* can optionally get assistance from the *Request Monitor*'s *Content Recommender*. Here, content that is predicted to be popular in the near future is used by the *Storage Manager* to prevent it from being evicted.
7. To this end, predefined content classes and their request patterns are monitored. The class *C1* – *CN* could refer to a YouTube category or a specific concept, i.e., features such as genre or mood of a music video.
8. To determine these features, a *Content Classifier* is required that extracts expressive features from the video which makes a classification possible.
9. Depending on the *Recommendation Policy* used, the *Popularity Monitor*'s time series, the content classifications, or both can be used.
10. The *Database* module is used to store the video request time series traced by the *Popularity Monitor* as well as the video classification results.

5.3.2 *Functional Overview*

In the following, we provide details on the core functionalities of ProCache and the interactions between its major modules. In essence, ProCache is understood as a network application that can provide cache management functionalities to a set of VoD streaming platforms, e.g., YouTube, Vimeo, or Dailymotion.

5.3.2.1 *Cache Manager*

The *Cache Manager* is responsible for three key properties of ProCache. First, ProCache has to be aware of video content categories in order to differentiate the content and handle it individually. Second, it supports to use distinct caching policies per content category and, thereby considers content category heterogeneity. Third, the amount of storage used per category must be size-adaptive, facilitating to adapt to fluctuating demand. In the following, we discuss each of ProCache's key properties mentioned above separately.

Content Category Awareness

Content-awareness is achieved by dividing the cache's storage into separate divisions that are responsible for one distinct content category. Besides, ProCache uses one probationary cache division similar to ARC and SLRU (cf. Section 2.2.1.2) because it has already shown to increase the cache performance. The probationary division is dedicated to content that is requested for the first time, independent of its category. Thereby, we prevent content that is rarely requested once from being cached into the category-specific cache divisions. To prevent the probationary division from consuming too much memory compared with the other divisions, we limit its size by the number of overall Bytes stored in the other cache divisions. Thus, it can not occupy the entire cache storage. However, in general, this division is expected to be rather small compared with the other divisions. This is a reasonably large share which considers the long-tail in the VoD content popularity distribution. In case a video in the probationary division causes a cache hit by being requested for the second time, it is moved into a category-specific division, which is allowed to use a more sophisticated eviction policy, e.g., LRU, ARC, or a Time To Live (TTL)-based eviction. An additional advantage of the probationary cache division is that it supports retrieving the video content category slower than it takes for ProCache to answer the request. Consequently, the video category is already at hand when the item is requested for the second time and can be moved to one of the category cache divisions. Since CDNs and content providers analyze their content, category descriptions often readily exist as part of the video's metadata. In this work, we focus on the publicly available YouTube video categories which associate a video with one out of 15 categories, e.g., Music, News, Entertainment, Sports, etc. We found that more fine-grained categories or semantic wikidata IDs[2] are less useful since only a fraction of YouTube requests belong to one out of several million of these categories.

Support of Distinct Caching Policies

ProCache supports the assignment of eviction policies to each division individually. We observe that many YouTube categories are responsible for only a small fraction of the overall request workload [KKH17; CDL08b] such that we decide to pool categories with relatively low request shares. Additionally to the probationary division, we create four cache divisions for the most popular categories: Music (42.5%), Entertainment (10.1%), People & Blogs (8.7%), Comedy (7.8%), and a fifth cache division to aggregate the remaining categories that are smaller than 7.5% as miscellaneous (30.8%). The values in brackets denote the share of requests towards the respective category [KKH17].

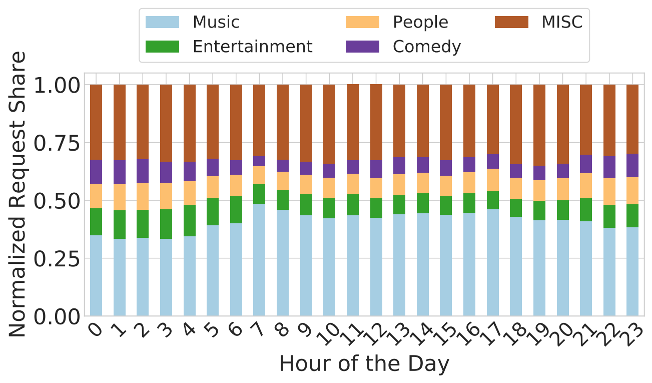


Figure 32: Popularity per category and hour of the day

Size-adaptive Cache Divisions

Assigning an equal amount of cache storage per category is unlikely to be the optimal storage distribution since we already know that, e.g., much more music is requested than news. Figure 32 shows that the category popularity distribution varies over the course of the day. Additionally, Figure 33 depicts the popularity distribution of the most popular YouTube categories. We observe a large long-tail for all categories. We note that music and entertainment have more popular content in general compared with people or comedy and the least popular content is observed in the comedy category. However, on the right side, we observe very different maximum view counts. We see that entertainment, misc, and music cover more popular content compared with people or comedy. As we observe different changes in the popularity dynamics of the different content categories over the course of the day, we consider a dynamic variation of category cache division sizes.

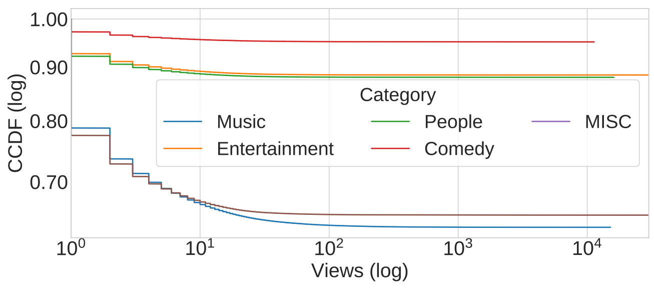


Figure 33: Non-normalized CCDF of the video content ranks, y-axis scaled linearly

2https://www.wikidata.org [Accessed: November 19, 2018]

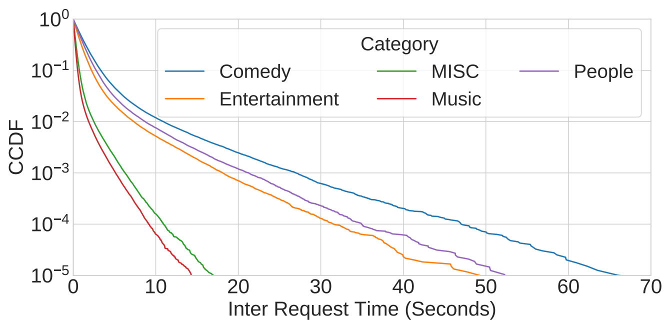


Figure 34: CCDF of the IRT for the most popular YouTube categories and the Misc category, y-axis scaled logarithmically

Figure 34 exhibits the Inter-Request Time's empirical distribution. We can see that popular YouTube categories, such as music, exhibit comparably low Inter-Request Times (IRTs). This means that their content is requested frequently. However, less popular categories show IRTs that are up to three times higher. Hence, these categories are dominated by content of low popularity which belongs to the popularity distribution's long tail. We can see that the popular categories have a smaller IRT as a consequence of more requests to these categories which do not have proportionally more items in their content catalog. This indicates a higher cacheability (cf. Section 3.2) for more popular categories. ProCache is flexible to adapt to diurnal popularity dynamics by adapting the size of its cache divisions according to the number of hits in their respective ghost lists. A ghost list contains the IDs (hashes) of evicted video segments from a given cache division over a pre-defined time interval. We propose a time-limited ghost list, which saves not only the ID of the evicted video segment of the cache division but also the time when the ID has entered the ghost list. When the IDs in these lists exceed the specified TTL threshold, they are evicted. We decided to set this value to four hours to keep track of evicted content within a reasonably large share of a day, e.g., the morning and the evening. Thereby it is guaranteed that ghost lists cannot grow indefinitely. Figure 35 depicts an exemplary state of the cache division storage allocation of ProCache. The blocks within the cache divisions represent video segments. The numbered block in the corresponding cache division's ghost list represents IDs of evicted video segments from this division. For example, the ghost list belonging to the category music can only contain IDs of this category that have been cached and evicted previously.

5.3.2.2 *Division Size Adaptation Strategy (DSAS)*

One of ProCache's key tasks is cache size adaptation over time if the belonging category popularity dynamics require it, e.g., the category gains or loses popularity. To this end, a transition of the amount of allocated storage per category needs to be conducted. However, since the cache storage is limited, increasing a cache division's size requires decreasing the other divisions. We define the term *Division Size Adaptation Strategy (DSAS)* as a policy that defines the trigger for storage size re-allocation and determines which cache division to decrease in this case. As it is not trivial to answer how the optimal DSAS looks like, we propose and discuss four different heuristics:

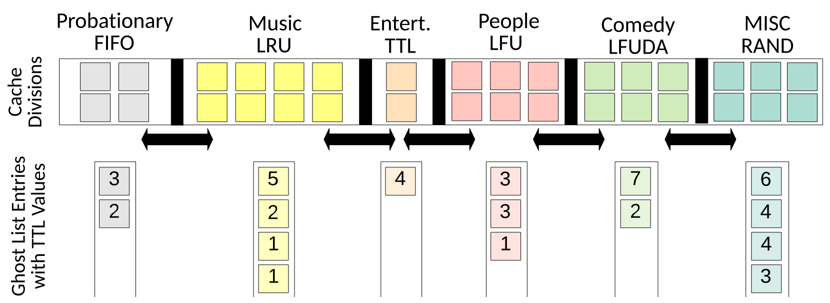


Figure 35: Example cache space assignment, numbers indicate the ghost list entry TTLs

Smallest Ghost List (SGL)

The cache division that has the smallest ghost list is decreased in size. Thereby, a comparably small ghost list indicates a too large category division because the ghost list is rarely used. Hence, it is likely that unpopular content is cached, i.e., content that exhibits a lower cacheability than the content of the division that is increased. This policy is likely to perform well in scenarios with a workload that has a rather short long tail, i.e., a comparably small amount of unpopular content since a large amount of unpopular content would quickly fill the ghost list and reduce the cache division size.

Largest Ghost List (LGL)

The cache division that has the largest ghost list is decreased in size. The underlying assumption is that a large ghost list indicates that video segments cached in this cache division are evicted more often than video segments cached in divisions with small ghost lists. This means that the corresponding cache division generates fewer hits than cache divisions with small ghost lists. This policy is expected to perform well in case of a large number of content that is rarely requested, e.g., just once or twice for some of the content categories.

Relative Smallest Ghost List (RSGL)

This DSAS decreases the cache division associated to the smallest ghost list relative to the division size. Thereby, not the absolute ghost list size is compared with the length of other ghost lists but the ratio of the ghost list size divided by the size of the cache division. Note, this emphasizes large cache divisions and, thereby introduces unfairness. However, it emphasizes the overall request workload heterogeneity. This policy is suitable for workloads of different content catalog sizes per content category as it considered the relative proportions between cache division and ghost list size.

Relative Largest Ghost List (RLGL)

The cache division that has the largest ghost list relative to its division size is decreased. Similar to Largest Ghost List, this relative measure considers that a cache division larger than another one is allowed to have a larger ghost list because it is responsible for serving more video segments. If requests for a more diverse content catalog arrive at one cache division compared to another, it is intuitive that video

```
Input: ID of cache division to increase: $S_i$
Input: Minimum cache division size: $M$
Output: ID of a cache division to reduce: $S_r$
if $S_i \neq$ Probationary then
    $S_r$ = Probationary
else
    $S_r$ = MISC
end
foreach cacheDivision s do
    if $s \neq S_i$ then
        if $s > M$ then
            $\Delta\text{current} = \frac{\text{s.ghostList.size}}{\text{s.size}}$
            $\Delta\text{candidate} = \frac{S_r\text{.ghostList.size}}{S_r\text{.size}}$
            if $\Delta candidate < \Delta current$ then
                $S_r = s$
            end
        end
    end
end
return $S_r$
```

Algorithm 1: Division Size Adaptation Strategy (DSAS): RLGL

segments get evicted more often, resulting in a large ghost list since also the number of content belonging to the popularity distribution's long-tail increases.

To determine which DSAS results in the highest performance, we conducted a set of experiments. Therefore, a cache with 10 TB storage was investigated. We consider only one category division in addition to the probationary cache division and a misc division pooling all other categories. The most popular categories: Music, Entertainment, and People & Blogs performed best with RLGL, which we use in the following. Note that less popular categories like Sports and Comedy showed the best performance with Smallest Ghost List, but overall RLGL performs best. We observed that DSASs often led to a convergence of smaller cache divisions to a tiny size. As a division with a size close to zero is unlikely to ensure meaningful measurements of relative ghost list size, which we need, we introduce a minimum cache division size M. The used DSAS is given in Algorithm 1.

Here, it takes the ID of the cache division to be increased S_i and the minimum cache division size M as an input. In case the ID does not refer to the probationary division, the probationary division is set as the default cache division to be decreased S_r; otherwise, the *MISC* category is selected. Next, it uses the ratio of the ghost list size to the cache division size of the current category s as well as the ratio of the ghost list size compared to the division size of S_r to decide on the category to be reduced next, i.e., setting the new S_r. Thereby, the division with the relatively largest ghost list is selected (in line 11 and 12).

5.3.3 *Supporting Music Video Content*

In this section, we introduce and discuss the design of ProCache's *Request Monitor* on the example of music videos. This module can be used by the *Cache Manager* (cf. Section 5.3.2.1) to get recommendations on content that should not be evicted from the cache as it is forecasted to stay or become popular. Since music videos are responsible for the major workload share of YouTube, we target this category specifically.

5.3.3.1 *Content Classifier*

The user *Request Handler* informs the *Request Monitor* upon every video request. Here, the videos are classified w.r.t. categories that are not available in the video's metadata. Since music can be described well by genre and mood, we use these two classes. For popular music content, genre and mood can be retrieved from the music website last.fm[3]. However, for recent or less popular content, this information is unlikely to be available. To this end, we need to extract features that serve as input for i) a genre classifier and ii) a mood classifier. These classifiers ensure that all music videos can be labeled.

5.3.3.2 *Music Feature Extraction*

We extract a music video's audio features for two reasons. First, we can train a classifier with known genre and mood using these features. Second, they serve as an input for the classifier for content where genre and mood are not retrievable.

Genre and Mood Retrieval from last.fm

The accuracy of the classifier we want to train depends on correct and descriptive music features together with a known label. To this end, we need a training dataset. We use a dataset that contains over 10 million requests to YouTube caused by 700k users. For each music video in this dataset, we use the title to request the tags annotated to this track by last.fm users. On last.fm, users can assign labels to a music track indicating, e.g., mood and genre of a song, but may also refer to the song's topic as they are free to choose a label. To search for our video titles on last.fm, we clean the video titles that we retrieved from YouTube in a previous step. To this end, we remove strings like "official clip", "officiel", and similar strings from the track titles. Besides, if a hyphen surrounded by spaces occurs in the title, the preceding part is assumed to be the artist, while the latter part is considered to be the title. Using these cleaned titles, all tags associated with them are retrieved from fast.fm[4]. Thereby, we retrieved tags for 13,553 tracks. Overall, these are 30% of the 44,704 tracks considered. This acknowledges our claim that not for every music tracks genre and mood can be retrieved. However, we need to have a mood and a genre label for all music tracks. Therefore, low-level audio features are derived and a genre and a mood classifier are trained on these features in conjunction with the dominant genre or mood information obtained from last.fm, i.e., the training dataset.

[3] `http://www.last.fm` [Accessed: November 19, 2018]

[4] We access the last.fm API (`http://www.last.fm/de/api`) using the python library pylast (`https://code.google.com/archive/p/pylast/`) [Accessed: November 19, 2018]

Happy	Sad	Angry	Relaxed
happy	sad	angry	relaxed
energetic	nostalgia	aggressive	calm
positive	depressive	banger	downtempo
fun	bittersweet	passion	chillout
cheerful	sentimental	quirky	dreamy
humorous	melancholic	annoying	longing
feel good	dramatic	gangsta rap	spiritual

Table 8: Subset of associations between Thayer's mood model and last.fm tags

Deriving mood

The freedom of users to assign tags to tracks on last.fm leads to a large variety of tags. To counter this, only tags assigned by at least 50 users are considered to avoid rarely used and less representative tags. The remaining tags are mapped to a smaller set of moods and genres. For the mood set, we choose to map each tag to one out of four quadrants of Thayer's mood model [Tha90] (cf. Section 2.4.3), a well-known psychological mood model[LMS+10; SBI14; YLC06]. The four quadrants of this model are: happy, sad, angry, and relaxed. For each of the most often used tags, a quadrant of the Thayer mood model is assigned following the mapping depicted in Table 8. For example, the tags: angry, aggressive, and banger are assigned to the quadrant: *angry*. Following this procedure, we generate a dataset of tracks and labels, which we consider as the ground truth.

Deriving genre

To derive the genre of a music track, we follow a similar approach as for deriving the mood. Following the approach proposed by Huang et al. [HLW+14], we use the following set of genre classes: rock, classical, pop, blues, jazz, country, disco, hip hop, metal, and reggae. Last.fm reveals that besides these genres, many songs in our dataset belong to the genres chanson, dance, electronic, and soul, which are not reflected in the aforementioned set of genre classes. To this end, we also consider these genre classes as well. To counter the noisy variety of genres in the tags that we retrieved from last.fm, we consolidate specific genres, as listed exemplary below. Thereby, we aggregate similar genres such as hip-hop and rap or reggae and reggaeton which are usually hard to distinguish from each other.

- **Metal**: metal, heavy metal
- **Rap**: hip-hop, hiphop, rap
- **Reggae**: reggae, reggaeton
- **Rock**: rock, classic rock
- **Soul**: soul, rnb

For example, tracks with the dominant labels hip hop, hiphop, and rap are considered as rap. For each track, the genre label assigned by most last.fm users, i.e., the dominant label is chosen. Following this approach, the genre is determined for 9,029 music video's tracks. The number of videos per genre is highly heterogeneous, e.g., Pop with 2,004 and Blues with 55 samples, as shown in Table 9.

Pop	Rock	Rap	Electronic	Soul	Chanson	Reggae	Dance	Metal	Jazz	Disco	Classic	Country	Blues
2,004	1,633	1,397	970	784	543	520	467	197	159	142	96	62	55
22%	18%	15%	11%	9%	6%	6%	5%	2%	2%	2%	1%	1%	1%

Table 9: Absolute and relative occurrence of samples per genre in the dataset

5.3.3.3 *Audio Feature Extraction*

The last step to generate a training dataset for our envisioned classifiers is to derive low-level audio features. In the following, we introduce the methodology used to extract these features. A set of frameworks for audio feature extraction are proposed as discussed in Section 2.4.3. We choose the Matlab package MIRtoolbox [LTE08] because it supports deriving a large set of audio features from an audio signal, e.g., the Mel Frequency Cepstral Coefficients values, tempo, spectral entropy, timbre, and pitch. In addition, statistical values of these features are offered such as the mean and the standard deviation. Overall, MIRtoolbox provides us with 392 features for each track. We use a representative sample of 30 seconds of each music video as input. This is a common procedure [LMS+10; TTK+08; SDP12]. The sample is taken from seconds 30 to 60 for tracks with a duration greater than 60 seconds. Thereby, we avoid the often not representative intro. For videos that have a duration of 60 seconds or less, we use the first 30 seconds. Following this procedure, we could retrieve the low-level audio features for 37,732 tracks. To reduce the number of features carrying similar information, sets of highly correlating features are determined. For each of this set, only the feature with the lowest entropy is used because it is most predictive. Thereby, we exclude correlating features that have the same or equal values for most tracks. This leaves us with 317 from the originally 392 features.

5.3.3.4 *Genre and Mood Classification*

This section discusses the design and performance of the genre and mood classifiers used by ProCache's *Content Classifier*. Using the previously created training dataset comprising the low-level audio features derived by MIRtoolBox and the determined genre or mood label, audio samples are used for training and testing of the classifiers. In a first step, the features are normalized, i.e., scaled to a value between 0 and 1, which is a common requirement for most machine learning algorithms. With the goal to avoid using less predictive and unnecessary many features, the classifier is trained on an iteratively increasing number of features, thereby following a common subset heuristic. If an audio feature can increase the classification accuracy, we add it to the feature set used; otherwise, it is discarded. However, the classification accuracy did not increase significantly by doing so. Therefore, all the 317 features are kept.

Based on the literature presented (cf. Section 2.4.3), a Support Vector Machine (SVM) is chosen as the classification model. Several combinations of parameters are tested to find the optimal SVM model and training parameter configuration. This includes the penalty parameter of the error term C, that defines how resistant the training is to overfitting, different kernels, i.e., linear, radial, polynomial, and sigmoid as well as the kernel coefficient γ which shapes the kernel function. We train the SVM model using 10-fold cross-validation to achieve a robust measure of classifi-

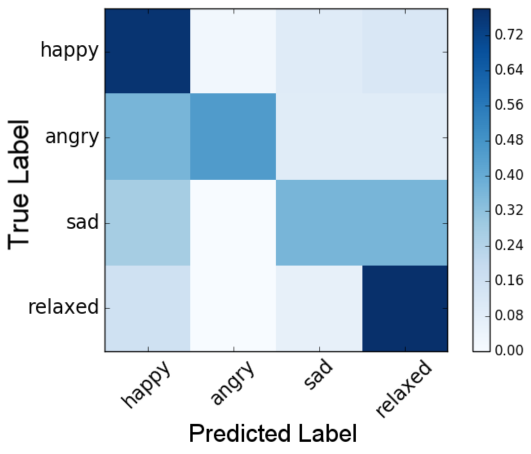


Figure 36: Confusion matrix of the mood classifier's precision

Mood	Precision	Recall	F1-score
Angry	0.50	0.60	0.55
Happy	0.82	0.77	0.79
Relaxed	0.63	0.73	0.68
Sad	0.53	0.47	0.50
Average	0.70	0.69	0.69

Table 10: Mood classifier performance metrics

cation accuracy. The python library scikit-learn[5] is used for training, cross-validation, and grid-search for hyper-parameter optimization as well as class balancing. Class balancing is especially necessary since the number of samples per class in the dataset vary, i.e., the dataset is unbalanced (cf. Table 9). Therefore, classes are weighted inversely proportional to their occurrence in the dataset to balance their influence and, thereby achieve a high classification accuracy for each class. Following this procedure, two classifiers are trained which can determine a track's mood and genre based on its low-level audio features. Note that the dataset used in this paper is significantly larger than the ones stated in the related work. The achieved accuracy of the mood classifier is 64% by using a radial basis function (RBF) kernel and a test set size of 10%. Figure 36 shows that the accuracy differs for each class. This figure shows a confusion matrix that indicates how many percents of the known labels, i.e., the true labels are classified correctly. For example, the category angry is easily mistaken for happy, while happy and relaxed music can be identified with a high accuracy of about 82% and 63%. Table 10 shows the achieved precision, recall, and F1-score per mood class.

Note that an SVM with a radial base function (rbf) kernel that we use for the mood classification outperforms the best results reported by Laurier et al. [LMS+10] and, thereby shows similar results to the approach of Rho et al. [RHH09]. One explanation for this is that the training data that we use is severely larger than in the related work. Furthermore, the approach presented by Laurier et al. [LMS+10] performs only 1.6 — 6.7% worse, depending on the hyperparameter settings chosen for the SVM. Note, this is a decent accuracy compared to a random classifier, which would result in 25% precision and a majority class classifier which would always predict happy and thereby achieve 40% precision.

The results for the genre classification show high precision values of up to 70% for the classes hip hop, pop, and electronic. On the other hand, blues and country, the two smallest categories, comprising about 50 samples in the training dataset, performed with 0% and are, therefore, usually misclassified. Still, overall the genre classifier shows an average precision of 50%, which is high for a classification task with that many classes. Here, a majority class classifier which would always predict

[5] http://scikit-learn.org [Accessed: November 19, 2018]

pop would result in 22% precision and a random classifier only in 7%. A detailed listing of precision, recall, and F1-measure is depicted in Table 11 for genres and in Table 10 for moods. The confusion matrix for all genres is depicted in Figure 37.

Genre	Precision	Recall	F1-score
Blues	0.00	0.00	0.00
Chanson	0.24	0.43	0.31
Classical	0.30	0.60	0.40
Country	0.00	0.00	0.00
Dance	0.20	0.43	0.27
Disco	0.29	0.48	0.36
Electronic	0.42	0.48	0.45
Hip hop	0.70	0.68	0.69
Jazz	0.14	0.26	0.19
Metal	0.23	0.48	0.31
Pop	0.63	0.34	0.44
Reggae	0.43	0.49	0.46
Rock	0.56	0.42	0.48
Soul	0.36	0.33	0.35
Average	0.50	0.44	0.45

Table 11: Performance statistics of the genre classifier

5.3.3.5 *Popularity Monitor*

The popularity monitor is a vital component of the *Request Monitor* (cf. Section 5.3.2.1). Its task is to monitor the distribution of pre-specified classes as well as all requests concerning their YouTube content categories over the course of a specified time interval, e.g., a day, a week, or a longer duration. While the YouTube categories can be retrieved from the YouTube Data API, the specific classes can refer to a more precise concept, e.g., genre and mood. To this end, the popularity monitor received this information from the *Content Classifier* module. Thereby, insights of the time-dependent content popularity concerning the video categories and the content classes can be deduced.

In the following, we emphasize the use of the popularity monitor for the case of music video caching. Music taste is observed to shift by the hours of the day, e.g., activating music during sports activities and relaxing music in the evening [GS15]. To validate this observation, the distributions of mood and genre over the course of the day are investigated. Figure 38 visualizes the distributions for a) genre and b) mood. Surprisingly, we observe only small variations for many of the genres over the course of the day. This observation holds true for all 14 days captured in the trace. While the popularity of most categories is relatively static, hip-hop and pop music exhibit more variation. While both genres vary in popularity, their summed share stays stable around 60%. One explanation for this is that the same users tend to request content from both genres.

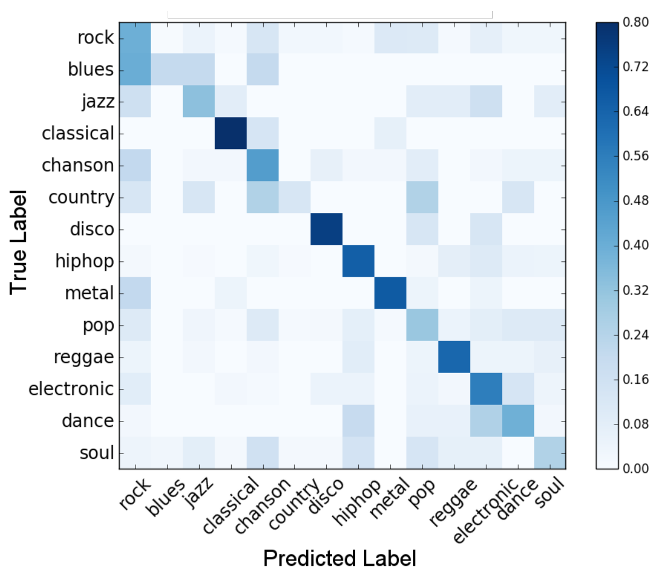


Figure 37: Confusion matrix of the genre classifier

5.3.3.6 *Content Recommender*

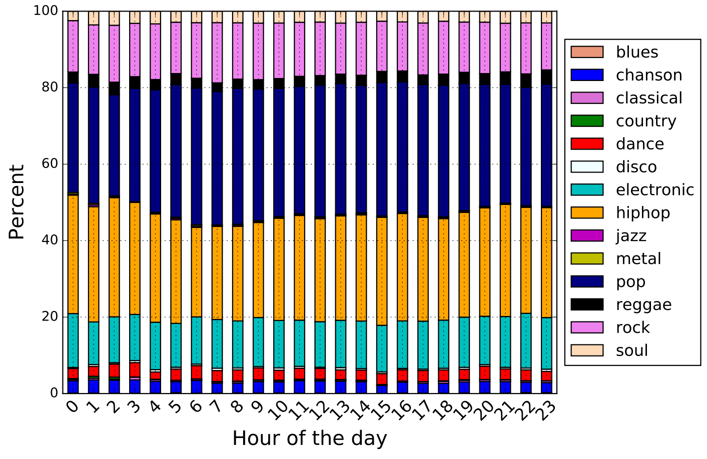
The *Content Recommender*'s task is to estimate content that is going to be popular in the near future. This information is provided to the *Storage Manager* upon request. To do so, the *Content Recommender* gets input from the popularity monitor and the *Content Classifier*. This includes genre and mood from the *Content Classifier* as well as content popularity time series. Therefore, we choose the popularity monitor classes C1: all videos, C2: genre, and C3: mood. Depending on the *Recommendation Policy* used, this information is used to conduct a popularity estimation on a per-video basis. A *Recommendation Policy* defines the actual mechanism used for the recommendation. The recommendation policies which we propose in this section determine which videos are emphasized in caching. Next, we present and discuss a set of recommendation policies suitable for music video popularity estimation.

Two main types of recommendation approaches can be distinguished: content-based and user group-based also known as Collaborative Filtering (CF). The content-based approaches rely on content-related information. The user group-based approaches require a detailed history of many users to work properly. Therefore, user behavior-based recommenders are more resource-demanding and require detailed user history information from many users. Hence, they are, in contrast to content-based approaches, not privacy-preserving. Independent of which of both approaches is used, the output is an ordered list of music video IDs, that are likely to be interesting for the users and, hence are suitable for proactive caching.

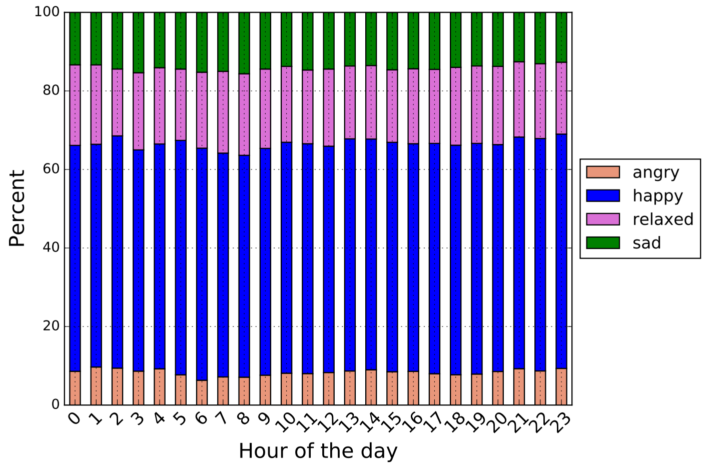
In the following, we propose three content-based recommendation policies: *Popularity*, *Genre*, and *Mood* as well as one user behavior-based *Recommendation Policy*: *Aggregated Similarity Measure*

Popularity

The popularity policy uses the popularity monitors observation for C1, i.e., the popularity of all video content. It provides a simple policy by recommending the most popular videos from the recent past. This approach can be implemented easily and is



(a) Genre



(b) Mood

Figure 38: Genre and mood popularity per hour of the day

likely to result in a reasonable performance for the near future. However, the content recommended by this policy can soon become stale depending on the dynamics of new videos being added to the content catalog. Deriving popular video candidates from the past might not be representative for the future, e.g., because the watching behavior correlates with a certain time or hour of the day.

Genre and Mood

Recommending potential stall content is the main drawback of the *popularity* policy. This can be countered by leveraging seasonal or reoccurring popularity patterns. To this end, we propose time-aware caching. The underlying assumption is that specific music features, e.g., mood and genre correlate with time of the day, as shown in Section 5.3.3.5. Thus, a *Recommendation Policy* that emphasizes music videos matching the most popular genre or mood of the current hour of the day is likely to be more accurate. This policy class the available storage for proactive caching as an input. Based on this, it provides a list of video IDs whose genre or mood match the popularity distribution of the near future, e.g., the next hour proportionally. For the contained mood or genre classes, the most popular content is recommended. We propose two specific recommendation policies that follow the procedure mentioned above: i) *genre* and ii) *mood* depending on which music features the policy's time-awareness orients.

Aggregated Similarity Measure

The *Aggregated Similarity Measure* policy uses a user-item-matrix containing an entry for every video a user has watched. The *Popularity Monitor* provides this information. If a video has been requested at least twice, the number of requests for this user is added to the matrix. Using this matrix makes it possible to determine similar users for a given user that share the user's interest. In the following, we refer to these similar users as neighbors. The Jaccard similarity coefficient defines the similarity between a user and its neighbors. For user a and b, the Jaccard index is determined by Equation 4. In the following, we refer to A and B as the videos watched by the users a and b respectively.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cup B|} \tag{4}$$

We identify the neighbors of a user by a high Jaccard index since both have watched a large share of the same videos. In a next step, the videos that have been watched by the neighbors but not yet by the user are determined. Their watch count is weighted by the similarity between the user and the neighbor that has watched this video. Finally, this value is taken as a *score* and the videos with the highest *score* are recommended to a user.

To adapt the *similarity measure* policy for our in-network caching scenario, we propose the *aggregated* user similarity measure as a policy. Therefore, we take for each user a list of tuples, i.e., (video ID, *score*). For all users, these lists are concatenated and in case an entry occurs multiple times in the resulting list, their *scores* are summed. The resulting list can be perceived as an aggregated similarity measure over all users. Thereby, this metric is suitable to recommend videos for a group of users.

5.4 EVALUATION

This section presents the evaluation of ProCache and discusses the results of our experiments. We present the evaluation methodology in Section 5.4.1. Section 5.4.2 provides an analysis of key aspects of the trace, addressing content properties and user behavior. In Section 5.4.3, we introduce the evaluation metrics used to assess performance and costs of ProCache. Section 5.4.4 presents performance measurements on traditional caching strategies and identifies a benchmark used in the rest of the evaluation. In Section 5.4.5 we evaluate ProCache on a mixed-category workload using dynamic storage allocation for different cache divisions. Additionally, we evaluate ProCache's music-specific work-flow in Section 5.4.6.

5.4.1 *Methodology*

We evaluate ProCache using a large real-world dataset described in the next section. Thereby, we avoid weaknesses of synthetically generated traces such as neglecting Inter-Request Time correlations and heterogeneous video popularity life cycles.

Due to the large number of data requests and content catalog size, we decide to evaluate ProCache by a set of simulation experiments. Thereby, proactive caching is either used for all requests or only the requests towards the content of a specific category, i.e., music. In the proactively managed storage area, videos are placed on a regular basis. In the case that not the entire *Cache Storage* is managed proactively, the remainder is managed traditionally by LRU since it is the most popular caching policy and, therefore ensures high comparability. We conduct experiments for caches of different size to study its influence on the ProCache's performance. Furthermore, the music-specific content recommendation policies proposed in Section 5.3.3.6 are evaluated.

To consider typical cache hierarchies as used by CDNs, we study ProCache's performance on a set of hierarchies that we introduced in Section 2.2.2. In these experiments, we split the overall storage equally among the caches comprised by the hierarchy. Furthermore, for cache hierarchies, we model the delay in our simulation in a way that the Round-Trip Time for the longest path does not exceed 50 ms as observed for real-world CDNs[6]. To this end, we equally distribute the 50 ms delay among the links between the caches. For the example hierarchies in Figure 5, this results in an 8 ms latency for hierarchy 1 and a 12 ms latency for hierarchy 3 per inter-cache connection. In addition, we assign video streaming clients uniform at random to one of the leaf caches to introduce variance into our simulations and to avoid being biased by an unintended beneficial association of clients to caches.

5.4.2 *Dataset Analysis*

We use a real-world trace that consists of over 10 million video requests to YouTube videos to evaluate ProCache. Its content catalog comprises more than 1.6 million videos. The trace was recorded in 2014 by a large European ISP containing only anonymized request information from unencrypted HTTP GET requests. At this time, it is estimated that this captured about 50% of all YouTube requests within this ISP 's network. In a later step, we retrieved the corresponding video category from the YouTube Data API[7] by using the video ID. The video creator chooses this category from a limited set of 15 available categories [CLD13] during the video upload process. Table 12 depicts an artificial but exemplary trace record.

Furthermore, we retrieved[8] the average bit rate of the videos for a resolution of 1080p and 30 Hz leading to a workload that sums up to 2.11 PB. It is a common approach to consider just one video quality aiming to not bias the results by an arbitrary number of video qualities, that can easily reach more than a hundred representations [KZS15].

5.4.2.1 *Content Analysis*

The trace contains YouTube video IDs which are used to enrich the dataset by metadata provided by the YouTube Data API. This metadata provides information about the video's release date, the category assigned by the uploader, as well as the video title. In a first step, we determine the YouTube category for all requests. While ana-

[6]https://www.cdnperf.com/ [Accessed: November 19, 2018]
[7]https://developers.google.com/youtube/v3/ [Accessed: November 19, 2018]
[8]https://youtube-dl.org/ [Accessed: November 19, 2018]

Property	Example	Description
index	1	Unique identifier for each record
requestTime	1397426400.07611	Request time, as Unix timestamp
userId	752210	Requesting user
videoId	WCt17N9fEOM	The YouTube video identifier
sessionDuration	17.639936	The duration of the TCP session
duration	273	Video duration in seconds
uploaded	2016-02-11T09:23:17.000Z	The video's publishing date
uploader	etit darmstadt	YouTube user who uploaded the video
category	Education	The category of the requested video

Table 12: Example record of the trace used for the evaluation

lyzing we the category distribution, we found music to be the most popular category measured in terms of requests in the trace. It causes about 42% of all requests. In a previous analysis from Li et al. [Li+13], they found 37% of the YouTube requests belonging to music. One explanation for this is an increasing trend towards music video watching on YouTube. Figure 39 exhibits the results of our analysis, depicting the relative share of requests belonging to the ten most popular YouTube categories. Categories of minor popularity that exhibit a request share smaller $<1\%$ are summarized in the category *Others*, which comprises: Movies, Trailers, Shows, Nonprofit, Animals, Travel, Tech, and Education. In the figure, we find that Music is more than four times larger than the second largest category: Entertainment, which shows just 10.03%. Another interesting finding is that about 35% of the YouTube channels appearing in the dataset have uploaded videos belonging to the category music. Thus, we deduce a dominant role of music content on YouTube.

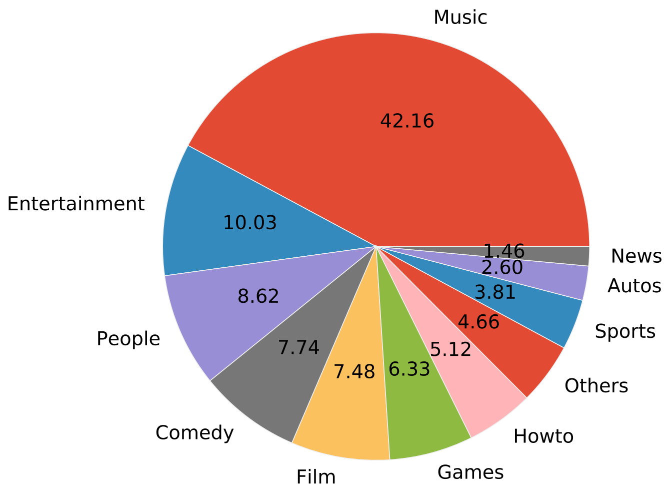


Figure 39: Relative request shares to different YouTube categories; categories showing less than 1% of all requests are aggregated to the category *Others*.

5.4.2.2 *User Analysis*

The network load caused by YouTube videos served to the users varies over the course of the day. Figure 40 exhibits the network load for the first week covered by our trace dataset. For the sake of clarity, we depict the relative number of requests in non-overlapping 10-minutes intervals. We see, that on weekdays, the traffic peeks short after noon, probably because of people watching videos during their lunch break and afterward return to work. At 5 pm, the workload shows a second peak, probably because work has ended and people are watching videos while commuting back to their homes. The request load stays high but slowly decreases until around 5 am. Weekend days show a distinctly different pattern compared with weekdays. On weekends, the users get active later during the day and request more videos overall. We cannot observe any dedicated peaks. Instead, the traffic stays high between 11 am and midnight. Overall, the user activity pattern shows a shift in time about 2-3 hours, as they start requesting later and stay active for later hours.

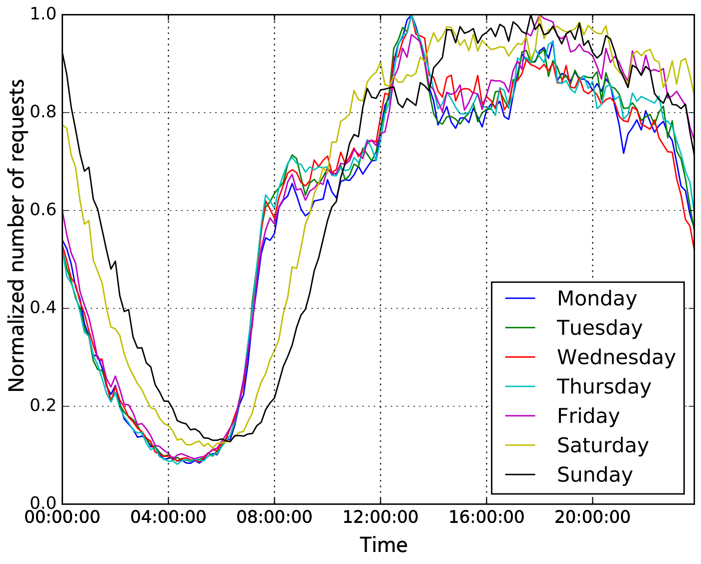


Figure 40: Normalized request count observed for each weekday per hour of the day

5.4.3 *Evaluation Metrics*

To evaluate the performance and cost of ProCache and other caching strategies, we use the following performance and cost metrics.

5.4.3.1 *Cache Hit Rate*

The Cache Hit Rate (CHR) is the ratio of cache hits of a caching system, which can consist of a single cache or multiple caches that form a cache hierarchy. Thereby, the CHR defines the ratio of the number of contents delivered by the cache to the overall number of requests N to the cache. Equation 5 depicts the formula of the CHR. Here, $\mathbb{1}_{h_i}$ evaluates either to 1 in case of a hit or to 0 in case of a miss. This metric is the most established metric and ensures for an easy comparison of the results. The CHR serves as a measure of content source offload since a higher hit rate implies fewer requests to the origin server and, hence less network load.

$$\mathrm{CHR} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{h_i} \tag{5}$$

5.4.3.2 *Startup Delay*

A drawback of considering the CHR for the entire hierarchy is that it obfuscates where the hit occurs. However, this is important since it influences the video's QoE (cf. Section 2.3.1). The further away the cache is that delivers the first segments of a video, the larger is the video's initial stalling time. In the following experiments, we measure the average latency A. It can be estimated as the average of the path latency up to the serving cache weighted by the hit rate at that cache. We denote the average latency between clients and caches of level m as L_m where level $m = 1$ corresponds to the content origin and level $m = M$ to the leaf nodes. Figure 41 depicts a simple example of a cache chain. H_m is the hit rate of level m. Therefore, the average latency A can be estimated as follows:

$$\begin{aligned} A &= L_3H_3 + L_2(1-H_3)H_2 + L_1(1-H_3)(1-H_2) \\ &\text{where } H_3 + (1-H_3) + (1-H_3)(1-H_2) = 1 \end{aligned}$$

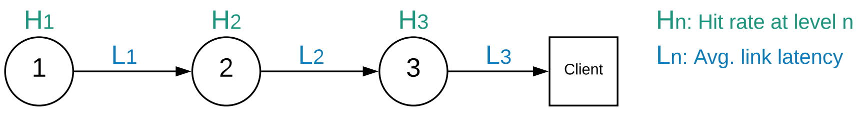


Figure 41: Cache delay model, on the example of a simple cache chain

5.4.3.3 *Write Operations*

The disk utilization if often a major performance bottleneck during the hours when most videos are requested, i.e., peak hours. Especially insert and delete operations take much time compared with read operations and, furthermore, lead to deterioration of Hard Disk Drives (HDDs) as well as Solid-State-Drives (SSDs). During the last years, the price gap between SSDs and HDDs has closed [Hac15]. This makes SSDs attractive as storage for caches. However, one drawback when using SSDs is the wear out. SSDs as NAND flash devices consist of storage cells, which store bits by assigning two different voltage levels, one representing a *1* and the other one representing a *0*. These cells are limited in the number of times this value can change [Bas15]. Thus, the number of write operations is an important cost metric considering the lifetime of SSDs. Read operations are far less costly and strongly correlate with the CHR. Hence, we focus on write operations as the sum of insert operations i and delete operations d in a cache hierarchy. In our evaluation, we determine this cost W as given by the equation below. Due to the large magnitude of this metric, we decide to compute the logarithm to the base 10 of the actual values.

Category	Music	Entertainment	People	Comedy	Misc
Content Catalog Size	354,991	111,859	116,198	47,444	338,400
Demand in TB	1.008,19	192,82	185,57	153,10	569,99
Demand in #Requests	$2.1 * 10^8$	$4.1 * 10^7$	$3.9 * 10^7$	$3.2 * 10^7$	$1.2 * 10^8$

Table 13: Content catalog size and number of requested video segments for the four most popular categories and misc

$$W = \log_{10} \sum_{m}^{M} \sum_{k}^{K_m} (i_{m,k} + d_{m,k}) \text{ for caches } k \in [1, K_m] \text{ on levels } m \in [1, M]$$

5.4.4 *Best Caching Strategy per Content Category*

In a first set of experiments, we aim to figure out what the best state-of-the-art strategy for content caching is. Thereby, we evaluate each the four largest YouTube categories separately. To this end, we use a distinct division covering the entire *Cache Storage* and is only exposed to requests addressing this content category. For this experiment, we consider a single cache hierarchy with a 1 TB large cache. Figure 42

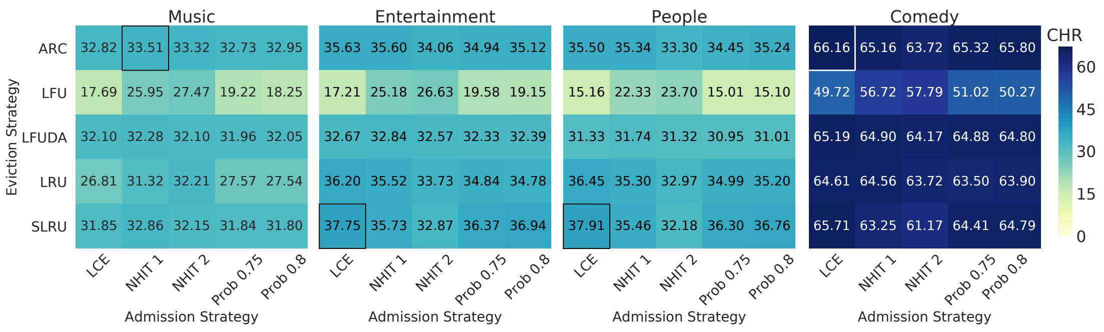


Figure 42: CHR for individual categories using a single cache hierarchy with 1 TB storage

depicts the results for the four most popular YouTube categories. For Music, ARC in combination with NHIT1 (cf. Section 2.2.1.1) performs best with only small differences to ARC in combination with NHIT2. For both categories, Entertainment and People&Blogs, the combination: SLRU and LCE show clearly superior CHRs. For the Comedy category, we observe higher CHRs than the other top four categories. One explanation for this is that Comedy has a much smaller content catalog size. Table 13 depicts the number of videos in the categories content catalog as well as the number of requested video segments. We see that the Comedy category shows a similar number of requests to a content catalog that is less than half the size of People&Blogs and Entertainment. In general, we observe that Least Frequently Used (LFU) shows a significantly lower CHR than all other strategies. As shown in Figure 34, the IRT for the chosen categories is small. This indicates a high peakiness of the content popularity in a short time span to which LFU is not able to adapt timely if new content gains in popularity as it still prefers outdated content that was popular in the past. Remarkably, SLRU and ARC outperform LRU independent of the admission strategy used. This demonstrates that LRU is in many cases a non-optimal choice, though for some admission strategies the performance differences are only marginal. The superiority of SLRU and ARC can be explained by the probationary cache divisions implemented in, both, SLRU and ARC as introduced in Figure 2.2.1.2. We deduce that ARC, SLRU, and LFU with Dynamic Aging (LFUDA) are most likely to show the best CHR for many video categories. Accordingly, we choose them as a benchmark for ProCache in our further analysis for experiments comprising multiple categories.

5.4.5 *Multiple Cache Divisions for Mixed-Content Workloads*

In the following, we evaluate the performance of ProCache by comparing its performance metrics in different scenarios with the empirical performance of the best caching strategy among the strategies introduced in the related work. To this end, we conducted a series of simulations and compute the average CHR, the number of write operations (#Writes), and the average latency (ØLatency) metrics as described in Section 5.4.3. We show our results in Table 14 for a set scenarios with different topologies and varying *Cache Storage* size. In the table, we show the performance metrics of ProCache as well as the best alternative caching strategy, i.e., the combination of admission and eviction policy achieving the highest CHR. Figure 14 shows, both, the performance and cost metrics of the best strategy as well as for ProCache. Additionally, we show the relative improvement achieved by ProCache for the ease of comparability. Note that positive values (green) denote a superiority of ProCache while negative values (red) indicate ProCache's inferiority. We observe that for cache sizes up to 1 TB, the best eviction policy turns out to be ARC combined with NHIT2 or Leave a Copy Down (LCD). For larger cache sizes, SLRU and LFUDA in combination with Leave Copy Everywhere (LCE) result in the highest CHR. Furthermore, we observe that ProCache performs poorly at small cache sizes of 1 GB for all hierarchies with a CHR degradation between 53% and 0.53%. However, larger or equal 10 GB, ProCache consistently increases the CHR up to 18%. Interestingly, we observe a hierarchy-dependent optimum cache size of either 10 GB (H1), 100 GB (H2), or 1 TB (H3, H4). In addition, ProCache decreases the latency to the clients and decreases the number of write operations at the caches slightly. Hence, we deduce that ProCache moves popular content closer to the clients than state-of-the-art caching strategies. We see the highest reduction in latency and write operations for cache sizes larger than the one with the highest CHR, though not the largest cache size considered. ProCache shows the largest performance increasing effect for small and mid-size caches when used in combination with NHIT2. Therefore, we conclude that ProCache is able to increase the CHR and simultaneously lowers the latency and the number of disk write operations in most of the considered scenarios.

5.4.5.1 *Cache Layer Performance*

In the next set of experiments, we provide more insights on the performance in different cache layers over time. For the sake of simplicity, we chose hierarchy 3 (tree) for these experiments. The reason for this is that ProCache exhibits for this hierarchy the highest CHR improvement and, additionally, decreases latency, and

		ProCache as Eviction Policy				Best Caching Strategy w/o ProCache					Improvement by ProCache (%)		
Hierarchy	Storage Size	Admission	CHR	#Writes	ØLatency	Admission	Eviction	CHR	#Writes	ØLatency	CHR	Latency	#Writes
1	1 GB	NHIT1	0.10	5.29	38.79	NHIT2	ARC	0.22	5.03	38.73	-52.83	-0.15	-5.24
1	10 GB	NHIT2	0.91	5.02	38.09	LCD	ARC	0.77	3.30	38.47	18.39	0.99	-52.03
1	100 GB	LCD	2.95	3.69	36.85	LCD	ARC	2.93	3.98	37.30	0.38	1.19	7.13
1	1 TB	NHIT2	8.04	4.96	34.08	NHIT2	ARC	7.9	4.95	34.60	1.92	1.51	-0.05
1	10 TB	NHIT1	19.49	5.11	28.14	NHIT1	SLRU	19.22	5.21	28.92	1.36	2.68	1.79
1	100 TB	LCE	32.67	5.26	20.91	LCE	LFUDA	32.62	5.39	22.72	0.17	9.98	2.41
1	1 PB	LCE	41.19	4.70	16.80	LCE	LFU	40.51	5.04	19.00	1.65	11.54	6.81
2	1 GB	NHIT1	0.25	5.10	47.88	NHIT2	ARC	0.43	4.83	47.79	-42.92	-0.19	-5.61
2	10 GB	NHIT1	1.63	5.09	47.05	NHIT2	ARC RC	1.46	4.82	47.80	11.48	0.74	-5.64
2	100 GB	NHIT2	6.40	4.75	44.44	NHIT2	ARC	5.64	4.80	45.30	13.36	1.90	0.87
2	1 TB	NHIT2	15.25	4.77	40.34	NHIT2	SLRU	14.80	4.87	40.90	3.07	1.38	2.02
2	10 TB	NHIT1	39.62	4.94	28.33	LCE	SLRU	36.44	5.50	30.54	8.74	7.25	10.25
2	100 TB	LCE	60.00	5.01	18.88	LCE	LFUDA	56.10	5.16	21.15	6.94	10.71	2.89
2	1 PB	LCE	63.33	4.52	17.86	LCE	LRU	62.25	4.84	18.26	1.74	2.19	6.55
3	1 GB	LCD	0.17	2.22	43.12	NHIT2	ARC	0.33	4.97	43.03	-48.66	-0.21	55.29
3	10 GB	NHIT2	1.24	4.96	42.26	LCD	ARC	1.23	3.59	42.59	0.88	0.77	-38.33
3	100 GB	LCD	4.81	3.86	40.49	LCD	ARC	4.42	4.13	40.97	8.80	1.19	6.49
3	1 TB	NHIT2	13.06	4.90	36.75	NHIT2	ARC	11.52	4.90	37.27	13.38	1.39	0.10
3	10 TB	NHIT1	30.15	5.07	27.63	LCE	SLRU	28.33	5.60	29.06	6.41	4.93	9.59
3	100 TB	LCE	47.87	5.20	20.54	LCE	LFUDA	45.65	5.31	20.95	4.86	1.94	2.02
3	1 PB	LCE	54.83	4.90	16.34	LCE	LFUDA	54.46	4.97	16.67	0.67	1.99	1.32
4	1 GB	PROB0.75	0.96	5.42	47.18	NHIT2	ARC	0.98	5.51	47.54	-0.53	0.73	1.54
4	10 GB	NHIT2	3.76	5.45	45.33	NHIT2	ARC	3.64	5.49	46.27	3.30	2.00	0.85
4	100 GB	NHIT1	10.85	5.26	41.47	NHIT2	ARC	10.38	5.45	43.02	4.50	3.61	3.50
4	1 TB	NHIT2	24.98	5.41	36.84	NHIT2	ARC	22.37	5.41	37.28	11.68	1.19	0.03
4	10 TB	NHIT1	47.43	5.52	25.16	LCE	SLRU	46.60	6.05	25.68	1.76	2.02	8.87
4	100 TB	LCE	67.41	5.56	15.68	LCE	LFUDA	67.09	5.62	15.92	0.48	1.55	1.05
4	1 PB	LCE	71.56	5.56	13.84	LCE	LFUDA	71.56	5.56	13.84	0.00	0.01	0.08

Table 14: Performance comparison between ProCache and the best performing caching strategy in terms of CHR without ProCache. The size is the sum of all cache sizes of a given hierarchy. The number of write operations is denoted logarithmically, i.e., $5 \triangleq 10^5$

disk write operations. Figure 43 shows two CHR time series, one for ProCache and one for ARC (bottom). The two graphs at the figure's top show the relative gain achieved by ProProCache in comparison to ARC at the edge cache and the leaf caches. We compare ProCache's best caching strategy to the best traditional strategy, i.e., ARC for this scenario. In the figure, the edge cache is denoted as E1 and the leaf caches are denoted as L1 — L4. Note that the shade per line indicates the variance of our measurements over repeated simulations, which is generally low. In general, we observe a gain, i.e., $\frac{\text{CHR of ProCache}}{\text{CHR of ARC}} - 1$ for the entire observed time interval. This gain is more than 30% for leaf caches, observed on Sunday noon. Also for non-peak hours, the gain ranges around 8%, except for the night hours when the gain drops to 1%. ProCache tends to have a higher CHR on average for all days and is able to exploit the traffic peaks exceptionally well. Additionally, serving more content from the leaf caches is beneficial for the user-perceived latency, as they can serve video segments faster due to their smaller geographical distance to users. Overall, ProCache outperforms ARC which is especially visible at the peak-hours at noon.

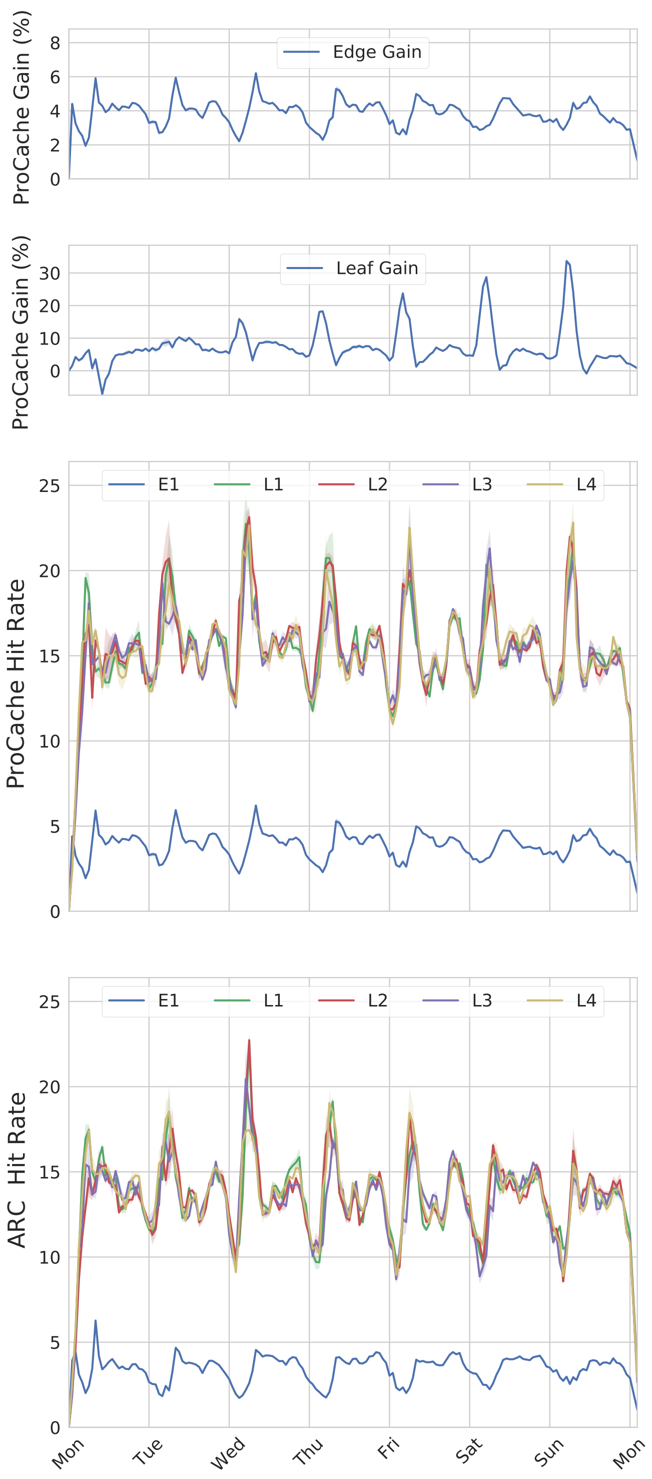
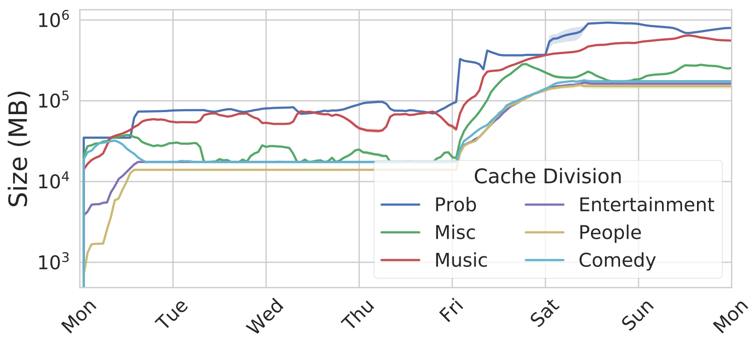


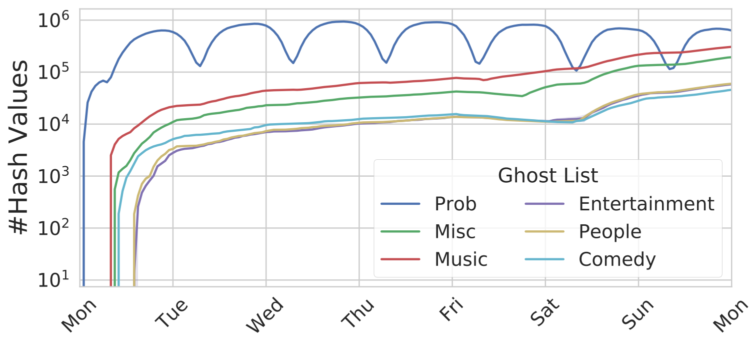
Figure 43: Top: Gain of ProCache compared to ARC, Bottom: CHR for edge (E) and leaf (L) caches of the tree hierarchy, 1 TB storage

5.4.5.2 *Adaptivity over Time*

One of ProCache's features is its adaptability to category-specific cache divisions over time. In the following, we investigate ProCache's resource allocation over the course of the two weeks captured by our request trace. Furthermore, we show how ProCache adapts to an increase of the entire *Cache Storage* size. To this end, we conduct an experiment where we increase the cache size from initially 1 TB to 10 TB.



(a) Cache division size evolution



(b) Ghost list size evolution

Figure 44: Cache division and corresponding ghost list size for a leaf cache in hierarchy 3 (tree) with 1 TB storage; *Cache Storage* is increased from 1 TB to 10 TB on Saturday

Cache Division Size Variation

Figure 44 a) shows a time series of the cache division sizes for each content category of a leaf cache from the tree hierarchy (i.e., hierarchy 3 in Figure 5) in a scenario with 1 TB storage being allocated to the entire cache hierarchy. We observe an initial warm-up phase until the cache division sizes converge to a stable configuration with low variation after half a day. The cache size allocation per category exhibits a diurnal oscillation pattern which shows that ProCache's flexible storage allocation is indeed exploited.

In the given scenario, ProCache's advantage is twofold: First, it discovers the appropriate division size upon a cold start. Second, it adapts to cache size changes, e.g., in combination with a pay-per-use storage cost model, energy saving algorithms for caches [SBI+11], or because of a permanent increase of the available resources. In our experiment, we perform such a cache size increase on Friday. After just one day after, ProCache reaches a stable configuration again with a small variation in cache division size on Saturday and Sunday. Note, the probationary cache division fills most quickly with content that is only requested once. However, the music and misc division immediately increase as well. On Sunday noon, ProCache's DSAS decreases the probationary division's size in favor of other category divisions that are still allowed to grow as they increasingly contribute to ProCache's performance. We can see this on the example of the music division and the misc division which grow on Sunday morning while the probationary division shrinks simultaneously.

Ghost List Size Variation

Besides the cache storage, also the ghost list varies in size. In the following, we investigate their behavior. Figure 44 b) depicts the ghost list size for every cache division in the same scenario as before. We see that the ghost list size ramps up and starts converging quickly. Remember that each content that is being evicted from a cache division contributes to the growth of its respective ghost list. This happens soon for the probationary division and later for the other divisions, e.g., on Monday morning for the music ghost list. Furthermore, we observe the probationary ghost list shrinking overnight because many video segments exceed the maximum TTL value and, thus are evicted. In case the *Cache Storage* is increased, the ghost lists do not increase anymore soon after the increase but shrink slightly due to the TTL-based eviction (cf. Section 5.3.2.1). However, as soon as the entire *Cache Storage* is filled again, content starts being evicted and the ghost lists start growing. For example, the music ghost list begins to increase as soon as the music division size converges, as shown in Figure 44 b). We deduce that ProCache converges to a stable configuration in terms of *Cache Storage* division sizes and also ghost list sizes with low variation after a short consolidation time.

5.4.5.3 *Eviction Policies*

In the previous experiments depicted in Table 14, we have seen that SLRU, ARC, or ProCache always achieve the highest CHR among all conducted experiments. Thus, these eviction policies seem to have a much higher impact on the CHR than the admission policy. To this end, we provide a full overview of the performance of these three eviction policies. Therefore, we evaluate them on all four hierarchies and evaluate their CHRs. In this analysis, we omit to show the belonging admission strategy, for the sake of comprehensibility, and, instead, emphasize the differences in CHR between the three eviction policies.

Table 15 depicts our results. We see that for hierarchy 1, SLRU, ARC, and ProCache show a similar performance while ProCache outperforms SLRU and ARC slightly for all cache sizes except for the case of the smallest cache size of 1 GB. When only comparing SLRU and ARC, ARC tends to outperform SLRU slightly up to 100 GB of storage. The main difference when comparing SLRU and ARC is that when using ARC, the probationary cache segment can adapt its size. Therefore, we deduce that this adaptability is responsible for the different performance in case of different cache sizes. For increasing cache sizes, ProCache's cache division size adaptation contributes less to the overall performance and seems to show diminishing returns. Hence, for 100 TB and 1 PB, the CHRs of all three eviction policies are similar. Overall, ProCache shows similarly or higher CHRs compared with SLRU

Hierarchy	Total Size	SLRU	ARC	ProCache
1	1 GB	0.20	0.22	0.10
1	10 GB	0.66	0.77	0.91
1	100 GB	2.35	2.93	2.95
1	1 TB	7.73	7.89	8.04
1	10 TB	19.22	18.60	19.49
1	100 TB	32.08	32.16	32.67
1	1 PB	39.35	39.38	41.18
2	1 GB	0.40	0.43	0.25
2	10 GB	1.22	1.46	1.63
2	100 GB	4.54	5.64	6.40
2	1 TB	14.80	14.78	15.25
2	10 TB	36.44	34.03	39.62
2	100 TB	55.48	55.57	59.99
2	1 PB	62.22	62.21	63.33
3	1 GB	0.31	0.33	0.17
3	10 GB	1.00	1.23	1.24
3	100 GB	3.47	4.42	4.81
3	1 TB	11.42	11.52	13.06
3	10 TB	28.33	26.88	30.15
3	100 TB	45.38	45.48	47.87
3	1 PB	53.80	53.76	54.83
4	1 GB	0.81	0.97	0.96
4	10 GB	2.65	3.64	3.76
4	100 GB	8.60	10.38	10.85
4	1 TB	21.65	22.37	24.98
4	10 TB	46.60	44.64	47.43
4	100 TB	66.73	66.82	67.42
4	1 PB	71.56	71.56	71.56

Table 15: CHR comparison between SLRU, ARC, and ProCache

and ARC for all cache sizes larger or equal to 10 GB. ProCache achieves the highest CHRs up to 71.56% for hierarchy 4 and hierarchy 2.

5.4.6 *Music-specific Support by a dedicated Division*

In the previous part of the evaluation, we have seen how ProCache can increase the efficiency by emphasizing multiple YouTube categories. In contrast to this, in this section, we emphasize the evaluation of ProCache for music specifically since this is the dominant source of traffic on YouTube. To this end, we evaluate ProCache and its popularity prediction policies introduced in Section 5.3.3.6. Each policy results in a list of videos from which the top videos are chosen to fill the music division. In our experiments, we vary the size of the entire cache as well as the size of the cache division dedicated to music video content. Traditional LRU reactively manages the remainder of the *Cache Storage*.

In our evaluation, we consider the users that show a constantly high demand. This is determined by the popularity monitor which traces all users' requests. While the evaluation is conducted on the entire trace, the *Content Recommender* and its *Recommendation Policy* are served by the requests of the high-demanding users since their stable and high demand allow for a high predictability[9].

5.4.6.1 *Single Cache Scenario*

In the following, we conduct a series of experiments and, thereby vary the cache size between 100 and 1,500 videos. The cache storage division size is varied between 5% and 25% of the entire *Cache Storage* to investigate the influence of the ratio between the music division and the LRU division responsible for all requests towards non-music videos. We evaluated the prediction policies: *popularity*, *genre*, *mood* and *aggregated similarity* as introduced in Section 5.3.3.6. In our experiments, we found that the *popularity* policy achieves the highest performance. Thus, we detail its performance in the following and provide the evaluation results of the other policies in the Appendix in Section A.4. Note that the popularity policy requires the lowest computational complexity among the policies considered and, therefore is the most reasonable choice.

Figure 45 exhibits the resulting CHR for the popularity policy. The figure's x-axis denotes the cache size variation and the legend indicates the share of the music division. In the figure, the blue line represents a cache that is entirely managed by LRU, i.e., 0% storage assigned to the music division. As depicted in the figure, the greater the music division, the higher is the resulting CHR. Increasing the proactive share in 5% steps shows a small positive effect after each step, converging at a CHR of 28% and a cache size of 1,500 videos.

[9]These users have requested at least two videos per day for seven days within the two weeks' trace are considered by the *Recommendation Policy*. This leaves us with 5,351 users that constitute 1,64% of all users comprised by the trace which are responsible for 15.6% of the total video requests. On average, each of these users watches seven videos per day.

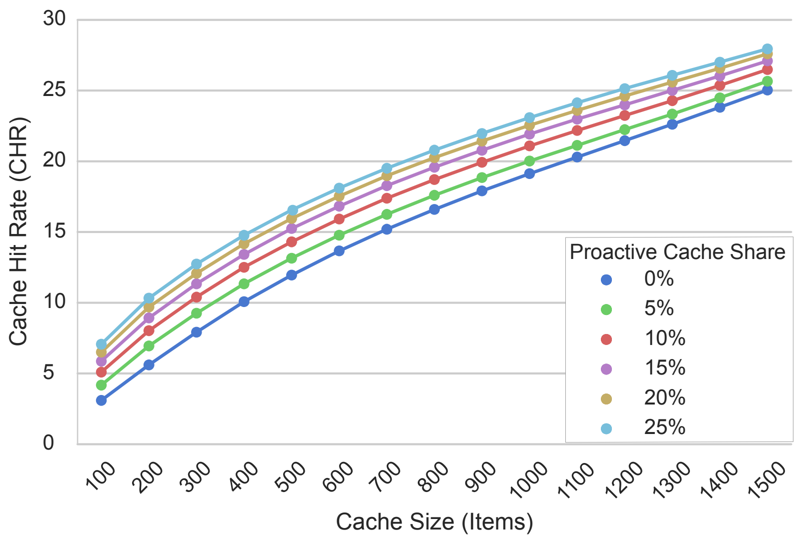


Figure 45: CHR for different cache sizes and proactively managed cache division. A larger proactive cache share increases the CHR.

5.4.6.2 *Multi-Cache Scenario*

In a further experiment, we investigate a setup comprising five distributed caches, e.g., placed in the metropolitan areas of a country. In our experiments, one-fifth of the clients is assigned uniformly at random to each cache to reflect the affinity between clients and cache servers based on geographical closeness. For each of the three policies, the CHR differences compared to a scenario where LRU manages the entire cache are evaluated to investigate the effect of proactive caching using the average CHR of all five caches. Figure 46 depicts the results of the popularity policy

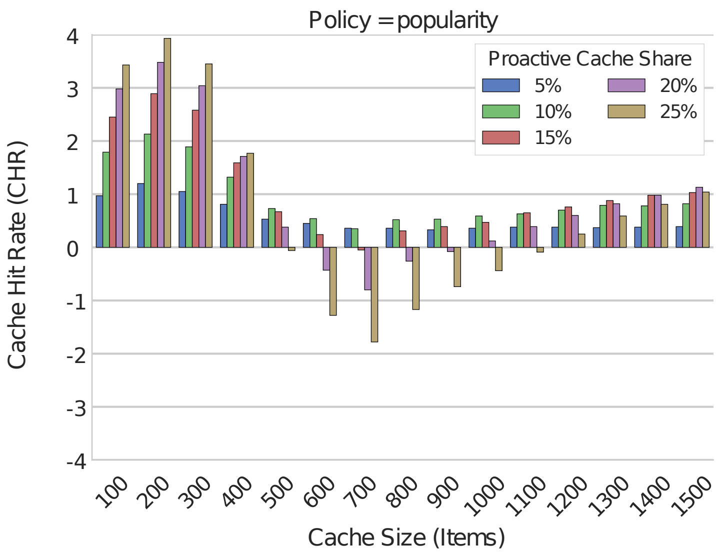


Figure 46: Relative cache performance compared with LRU

which achieved the highest performance. We show the results of the other policies in the appendix in Section A.4. For small cache sizes, ProCache can increase the CHR by up to 4% using the popularity policy. For cache sizes between 600 and 1,100, this policy has a positive effect of up to 0.8% if we consider smaller music division sizes. For mid-sized caches, ProCache is able to decrease the CHR when the proactively managed share of the cache is chosen too large, e.g., 25%. However, a proactive cache share of 5% always increases the performance, even slightly for mid-sized caches. Comparing the three policies, it can be seen that the *popularity* policy is superior to genre and *aggregated similarity*. Overall, the maximum CHR, with 55.1%, is achieved by the *popularity* policy with a proactive cache share of 20% and a cache size of 1,500 items. ProCache's popularity policy achieves the highest gain measured by CHR with a cache size of 200 videos and a music division size of 25%. In case the cache is entirely LRU-managed, it achieves a CHR of just 8.9%. When combined with a music division, it achieves a CHR of 12.8%. Summarizing, the performance of ProCache for music content depends on the cache size, the number of users served by the cache, and the size of the music division relative to the entire storage available.

5.5 SUMMARY AND DISCUSSION

In this section, we designed and evaluated a novel proactive caching mechanism denoted as ProCache. Using a two-week real-world trace from a national ISP, we first studied the performance of traditional combinations of admission and eviction policies. We have seen that SLRU and ARC achieved the highest CHR among the traditional cache eviction policies while the admission policy is less influential. Hence, we used SLRU and ARC as a benchmark for ProCache. In a set of experiments, we provided insights about YouTube category popularity distributions and ProCache's CHR when leveraging those content categories. Furthermore, we discussed the performance of ProCache on different cache hierarchies and cache sizes to provide insights about the individual CHRs. We demonstrated that ProCache results in a superior CHR for cache sizes larger than 10 GB for all cache hierarchies considered. Additionally, in most experiments, ProCache could lower the user-perceived latency and the number of cache insert and delete operations.

Additionally, we proposed and evaluated a music-specific design of ProCache. To this end, we present recommendation policies based on music-specific features using popularity, genre, and mood as well as their time-dependent distributions over the course of the day. We demonstrated ProCache's benefit for large caches with many users being served. However, also caches smaller than 500 can benefit from ProCache. For storage sizes ranging between 500 - 1,000 items, ProCache showed a limited positive effect.

6 SDN-ENABLED MULTICAST FOR VIDEO-ON-DEMAND WITH VODCAST

In our review of the related work (cf. Section 3.3), we identified a lack of multicast mechanisms for Over-the-Top (OTT) Video-on-Demand (VoD) content delivery that efficiently use Software-defined Networking (SDN) as an enabling technology. This chapter strives to counter this lack by presenting an innovative SDN-enabled multicast mechanism for OTT VoD. Hence, the proposed mechanism uses a cross-layering approach that makes it possible to coordinate closely between the control application and the network layer. We denote this mechanism as VoDCast which aims at efficiently reducing the network load such that it consequently reduces transit costs and Quality of Experience (QoE)-decreasing network congestion events (cf. Section 3.3.1.1). VoDCast was first presented in [KHH17]. Furthermore, a number of student theses contributed to the publication mentioned above and the sub-mechanisms presented in the course of this chapter [Hac16; Kli17].

Chapter outline

In Section 6.1, we provide a general overview of multicast applied to the VoD scenario and its benefits. These make multicast of VoD especially valuable in a set of use cases, which we discuss in Section 6.2. VoDCast's system design and major conceptual components are introduced and discussed in Section 6.3. The evaluation of VoDCast is presented in Section 6.4. We evaluate VoDCast using a real-world trace collected by a large, European mobile network operator. This chapter is summarized and discussed in Section 6.5.

6.1 CONCEPTUAL OVERVIEW

Before we detail the envisioned use case scenarios and the mechanism-design of VoDCast, we want to convey a conceptual understanding of the approach. For a popular video, it is likely to observe thousands of video streams, being delivered to customers within a single Internet Service Provider (ISP). Some of these customers watch this popular video with a small difference in their playback positions. In this case, there is a potential for multicast to reduce the link utilization and, therefore network load efficiently as described in the following.

Assuming the playback of a distinct video, the clients are likely to demand different video segments which they need for their current video playback. However, video segments that have not yet been requested by them can be delivered using multicast to all clients that currently playback the video. Figure 47 depicts an exemplary snapshot of three clients and their video buffers. Note that the three clients started their playback consecutively at time 0:00h, 0:05h, and 0:15h respectively. In our example, one exemplary multicast group is triggered by the request of client 1 for a popular video. We decided to exclude the first video segment from being delivered using multicast since many videos are only played for a short time till the user requests the next one as indicated by Figure 48. Thereby, we keep the number of costly system changes low. In Figure 47, we see the current unicast and multicast

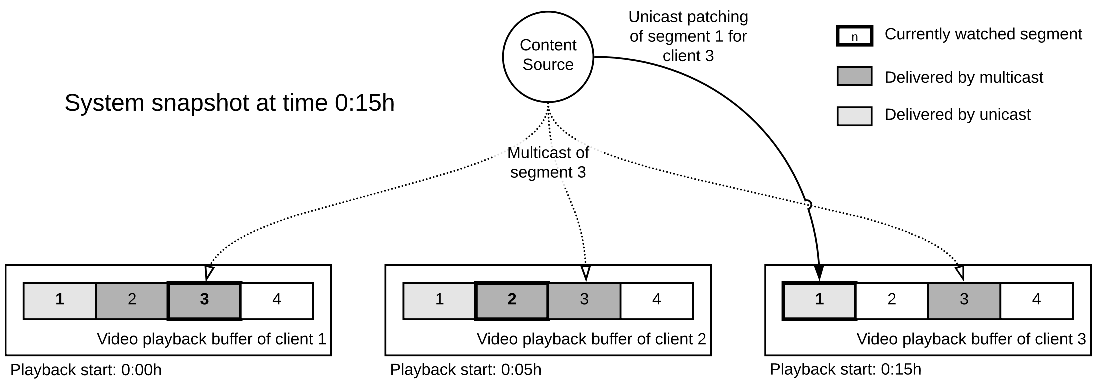

System snapshot at time 0:15h
Content Source
Unicast patching of segment 1 for client 3
Multicast of segment 3
Currently watched segment
Delivered by multicast
Delivered by unicast
1 2 3 4
Video playback buffer of client 1
Playback start: 0:00h
Video playback buffer of client 2
Playback start: 0:05h
Video playback buffer of client 3
Playback start: 0:15h

Figure 47: Conceptual overview of VoDCast's unicast patching

transmissions as well as the clients' buffer states at time 0:15. Since each video segment is assumed to be 5 seconds long, client 1 playbacks video segment 3 already. Client 2 started the video playback at time 0:05. At this time client 2 needed segment 1, but the multicast group delivered segment 2. Therefore, client 2 had to request the first segment using unicast, however, buffers already the second segment delivered by the multicast group. Hence, the content is buffered ahead of time, as we see in the case of client 2 and client 3. All video segments that are not delivered by a multicast group need to be requested using unicast, to stream the video without stalling. This approach is similar to patching (cf. Section 3.3.1.2), but has the advantage of a lower network resource utilization, since not every client stream causes the creation of an additional multicast group in case only one client requests the video at a given time. Instead, we propose an efficient popularity-based approach which we will explain later. Furthermore, stalling events, as possible in batching or Earliest Deadline First (cf. Section 3.3.1.2), are less likely to occur using a unicast patching approach. The reason for this is that the client does not have to wait for the mechanism but can request the missing video segment autonomously as soon as it anticipates that the next segment to playback will not be in its buffer on time.

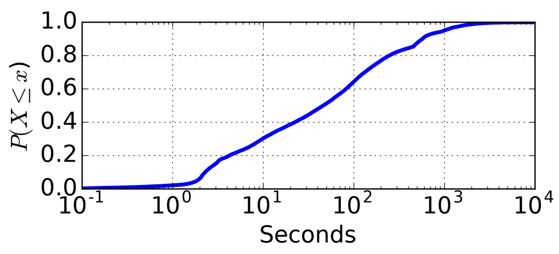


Figure 48: CDF of the video session duration in the YouTube request trace

6.2 USE CASES

Streaming popular VoD content using multicast is beneficial to a set of scenarios where it can be used with one common objective: Decreasing network load and, thereby potential transit costs and network congestion events. Customers of VoD streaming services almost equally blame the video streaming service, the ISP, and the Content Delivery Network (CDN) on poor viewing experience while the ISP is considered to be responsible for a low-quality video stream most often [Con15]. For both, ISPs and CDNs, VoDCast can reduce the network load and, thereby resolve and prevent network congestions that negatively affect the users' QoE. In the following, we present three primary use case scenarios shown in Figure 49 in which VoDCast can contribute by the benefits mentioned before.

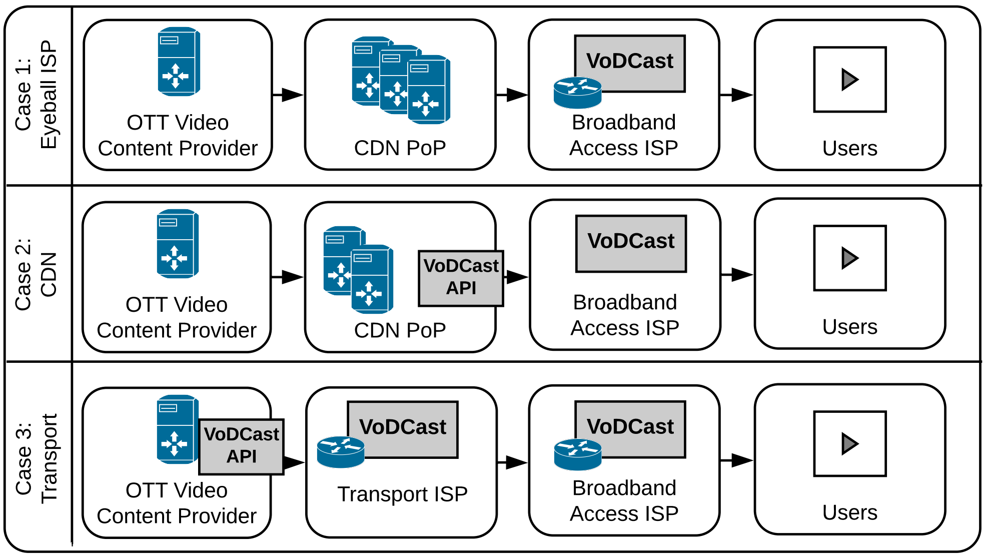


Figure 49: Three major use cases of VoDCast

The first use case is an ISP scenario where IPTV services are often offered together or in combination with the broadband access tariff. This comprises live streams as well as VoD. VoDCast can help ISPs to decrease the video traffic caused by their streaming services and, thereby consume less network capacity. Thereby, ISPs transfer the same amount of video traffic but require fewer network resources which helps to save Capital Expenditure (CAPEX).

The second use case considers a CDN scenario that delivers OTT VoD content to ISP customers. Thus, each content is delivered by a separate unicast transmission per user. By using VoDCast instead, the large amount of traffic caused by, e.g., YouTube can be decreased and, thereby also the required resources of the CDN and the ISP. Since the traffic is streamed from outside of the ISP network, transit costs are likely charged. As VoDCast reduces the network load, it also lowers transit costs for this use case. Furthermore, the QoE is likely to be increased especially during peak hours as observed for video demand [Cis17c] because of a lower risk of network congestion.

The third use case considers a multi-ISP scenario that extends the second use case over more than one ISP since the content transmission often passes multiple ISPs, e.g., a tier-1 ISP that connects the content source's ISP with the ISP whose customers are requesting videos. Here, an inter-ISP collaboration is beneficial for all involved ISPs because of the already introduced reasons for single ISPs. Though, the coordination-effort and also the business models of ISPs of different size and connectivity might lead a cooperation to be more or less likely. A large ISP selling transit to a set of smaller ISPs might be more interested in maximizing its revenue by maximizing the traffic it transports and, hence can charge. However, for smaller ISPs and ISPs of equal size, this motivation does not exist since they are likely to have a peering agreement free of charges.

6.3 SYSTEM DESIGN

This section discusses the design of the VoDCast mechanism. Therefore, we first introduce VoDCast's high-level architecture and its key components. Second, we provide a functional overview of VoDCast's components.

Assumptions and requirements

For the design of VoDCast, we consider only available technologies that are used by the industry to a certain extent already, such as SDN-enabled switches and group encryption schemes. Similar to Section 5.3, we assume videos to be segmented, which is given by today's HTTP Adaptive Streaming (HAS)-approaches, e.g., Dynamic Adaptive Streaming over HTTP (DASH) (cf. Section 2.3.1). Further, we consider only one video quality because this ensures increased comparability in contrast to an arbitrary- or platform-depending set of video qualities, which can reach more than 100 quality levels [KZS15]. The chosen quality is either 1080p with 30 Hz or, if this is not available, the closest lower video quality. A further assumption we make is that ISPs and CDNs have a common interest to collaborate (cf. Section 6.2) to achieve high service quality and to lower their costs. The related work indicates an interest in CDN-ISP collaborations [Wic17; AFA17; WKB+17; AFA16].

6.3.1 *Architectural Overview*

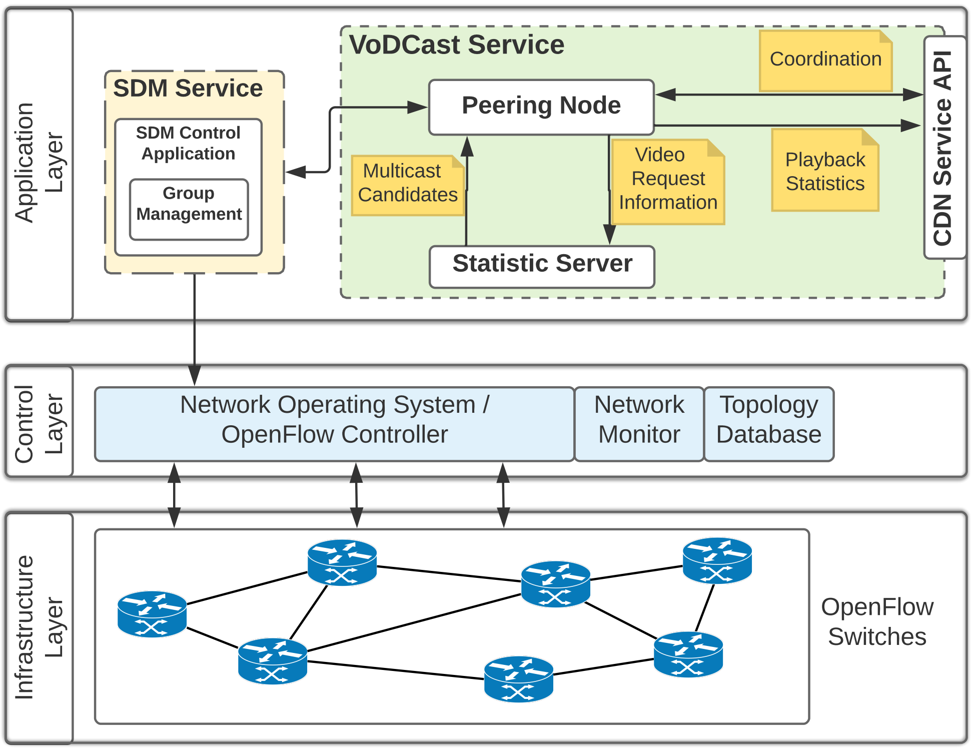


Figure 50: VoDCast Network Architecture based on [ONF14b]

An overview of VoDCast's conceptual architecture and their relationships is given in Figure 50 following the Open Network Foundation (ONF) SDN reference architecture [ONF14b]. On the application layer, we see the VoDCast service and the Software-defined Multicast (SDM) service. The SDM service is responsible for SDN-based multicast group management and the installation of the necessary SDN flow entries at the switches. The VoDCast service offers a *CDN Service Application Programming Interface (API)* to external CDNs. This can be used to coordinate between the ISP and CDNs on which videos to deliver by ISP-internal multicast. This coordination task is performed by the *Peering Node* which checks if enough capacity for SDN flow entries is available at the ISP's switches before allowing a video to be multicasted. In contrast to unicast, when multicast is used, client playback information is no longer available at the CDN. Thus, the *Peering Node* informs the CDNs about client playback statistics. In case VoDCast is used for ISP-internal VoD services or unencrypted OTT VoD traffic, the *Peering Node* does not depend on the CDN. In this case, the *Statistic Server* monitors incoming video streams, identified, e.g., by their Internet Protocol (IP) ranges and, hence can deduce which videos are currently popular. In both cases, a CDN cooperation and a stand-alone-operation, the VoDCast service informs the SDM service of video candidates suitable for being delivered using multicast. The SDM service takes care of multicast group management and implements the necessary multicast flow entry into the SDN switches with the help of the network operating system, e.g., an OpenFlow controller. This controller is also aware of the network topology, the current traffic, and flow rules in its topology. In case the available SDN flow entries are depleted on the hardware switches, the controller would notify the SDM service which would instruct the *Peering Node* not to accept any more videos for being multicasted.

6.3.2 *Functional Overview*

In the following, we provide details on VoDCast's key functionalities and the interactions between its major components. In essence, VoDCast is understood as a cross-layer network application that provides managed access to ISP-internal multicast to an external entity, e.g., a CDN or another ISP.

To the best of the author's knowledge, so far, none of the existing works explores the potential of VoD streams with a small playback distance and Zipf-distributed popularity where only a few videos exhibit high popularity. Consequently, one challenge for VoDCast is to determine candidate videos for multicast out of a large number of available videos. This candidate set can be determined in cooperation with the serving CDN that knows which videos are currently popular. For this cooperation, VoDCast exposes a *CDN Service API* to the cooperating CDN to get informed about which videos to multicast. However, VoDCast does not depend on the CDN's cooperation. VoDCast is able to collect popularity information of the videos requested by the ISP's customers, in the case videos are delivered unencrypted. Hence, a content selection mechanism transition (cf. Section 2.6) between a cooperative selection and a stand-alone selection is possible. Though, we assume the stand-alone selection mechanism to be much less efficient than a selection based on a CDN cooperation since there is a tendency to encrypt every transmission.

In the following sections, we first introduce VoDCast's *CDN Service API* (cf. Section 6.3.2.1). Second, we present the two system operation modes, i.e., operation in coordination with the CDN (cf. Section 6.3.2.2) or independently, in the case of unencrypted content transmissions (cf. Section 6.3.2.2). Section 6.3.2.4 details the minor changes that VoDCast requires on the streaming client's side. Since VoD-

Cast is highly configurable, we reason our system parameter value selection in Section 6.3.2.5.

6.3.2.1 *CDN Service API*

CDNs continually monitor which content is requested from which ISPs to optimize system performance for load balancing, caching, and accounting reasons (cf. Section 2.2). Hence, for CDNs, it is transparent if multiple clients from one ISP network watch the same video. In case such a video is frequently requested and many close-in-time video sessions exist, the CDN can signal to VoDCast's *CDN Service API* to prefer this video being delivered via multicast within the ISP network. After the CDN and the ISP have agreed on a set of videos to deliver by VoDCast, they are delivered via Hypertext Transfer Protocol Secure (HTTPS) from the CDN to the ISP. Here, video segments are received and forwarded by the ISP using User Datagram Protocol in the case of multicast or using Transmission Control Protocol (TCP)/Hypertext Transfer Protocol (HTTP) in the traditional case of unicast. Note that to apply multicast, the ISP cannot support an end-to-end encryption scheme between the CDN and the ISP's customers. A naive solution is to break the encryption entirely and rely only on unencrypted communication within the ISP as a trusted party. However, trusting an ISP, in this scenario, is desirable to be avoided by CDNs since this allows not just the users but also the ISP to see the requested content unencrypted. Furthermore, when the transmission leaves the ISP and enters a company or home network, the transmission cannot be considered to be confidential anymore by any means. We propose using a CDN-supporting multicast-compatible group encryption scheme [DC06] as a solution. Here, the CDN loses information about playback behavior of distinct clients as available on unicast transmissions. To compensate for this, VoDCast informs the CDN about users' joins and leaves using a multicast group ID, shared between the CDN and the ISP for each multicast group respectively. Client statistics, only visible to the ISP, are forwarded to the CDN in conjunction with this ID. Thereby the CDN is provided with information about client playbacks, e.g., playback interruptions, pausing, and quitting. Since this information can be provided upon a user event in form of a previously agreed numerical format, we consider the overhead for this feedback information to be low. Ideally, the data is collected at the ISP and sent to the CDN in aggregated and compressed form overnight to minimize its traffic demand, though, this might not be an ideal solution for the CDN when it wants to monitor its users in real-time. Thus, a batch-processing and transmission approach sending the data compressed every minute is likely to be a good compromise.

6.3.2.2 *ISP CDN Cooperation*

The *CDN Service API* is offered by the *Peering Node* which is the key component of VoDCast. Figure 50 depicts one *Peering Node*. In practice, multiple *Peering Nodes* and their associated SDN controllers [HRR+17b] can be placed at the edge of the ISP network, as in reality, an incremental deployment of VoDCast is more likely. For the sake of simplicity, we focus on the case of a single *Peering Node* as the smallest deployment scenario. The *Peering Node*'s task is to coordinate the ISP and the CDN or the content provider about which videos to stream via ISP-internal multicast. Thereby, the CDN and the ISP can agree to multicast not just entire videos but distinct video segments. This is useful to exclude the beginning of a video from multicast since it is likely that a user quits the video playback or jumps to another video. In the YouTube trace used (see Section 5.4.2), this behavior is observed mostly within the first 20 seconds of a video stream. Therefore, it is decided to deliver the first $T_{\text{unicast}} = 20s$ always as unicast which excludes 39% of all video sessions in the trace. This is beneficial since the overhead introduced by frequent multicast group churn, i.e., group creation and modification can be largely reduced. In addition, this helps to keep the initial stalling time small, because the additional delay caused by a group creation and join is avoided. Thereby, the QoE is likely to increase (cf. Section 2.3.1). If a client joins a group later on, the playback is not interrupted, because some segments have been buffered already.

If the CDN and the ISP have negotiated on a video to be selected for multicast, the *Peering Node* triggers the creation of a multicast group by notifying the SDM controller. If the multicast group becomes empty at any time, it is removed. Thereby, situations in which all group members leave before the video ends the group can be removed early and, thereby resources are saved. In addition, to the explicit leave of a client, a keep-alive message[1] ensures that even clients whose system has crashed are removed from their respective multicast groups timely. Each multicast group uses an underlying multicast tree for the content distribution within the ISP network. In case a client joins or leaves, this tree needs to be recomputed. In a next step, the new tree is compared with the established one to determine the switches, i.e., nodes in the graph, which need to be updated by installing, altering, or deleting flow entries.

6.3.2.3 *ISP-based Multicast Streaming*

In the following, we discuss VoDCast in the case of ISP-internal VoD services and situations of non-cooperative CDNs. In these two scenarios, VoDCast can operate solely on popularity statistics of observed video streams. For content that is delivered by a CDN, this is only possible to a certain extent, i.e., for unencrypted content transmissions. VoDCast monitors video requests to collect popularity statistics at the *Statistic Server*. Video popularity distributions are highly dynamic for VoD services like YouTube [CKR+07]. Therefore, the *Statistic Server* maintains a computationally efficient sliding window w that contains the requested videos and their numbers of requests from the last P_t minutes. The top P_k percent of the most popular videos in w are considered for multicast delivery. Thereby, videos from the short tail of the video popularity's Zipf distribution are selected. As shown in Figure 50, the *Peering Node* is connected to the *Statistic Server*. In case it determines a video suitable for multicast delivery, a multicast group that delivers the video content is created as soon as the first client request for this video is observed by the *Peering Node*. In the considered ISP-only scenario, the *Peering Node* takes care of identifying client requests to the same video. This can be implemented by matching CDN destination IP address ranges and analyzing the HTTP GET messages to retrieve the video ID. The decision whether a new group for a popular video is created depends on two conditions: i) if a group G for this video already exists and ii) if one exists, its current playback position $P(G)$, i.e., the segment that the multicast group currently delivers. We provide an example in Figure 51. A video playback P_1 of a popular video is

[1] Similarly to the reports sent when using the Internet Group Management Protocol in traditional IP multicast: https://tools.ietf.org/html/rfc3376 [Accessed: November 19, 2018]

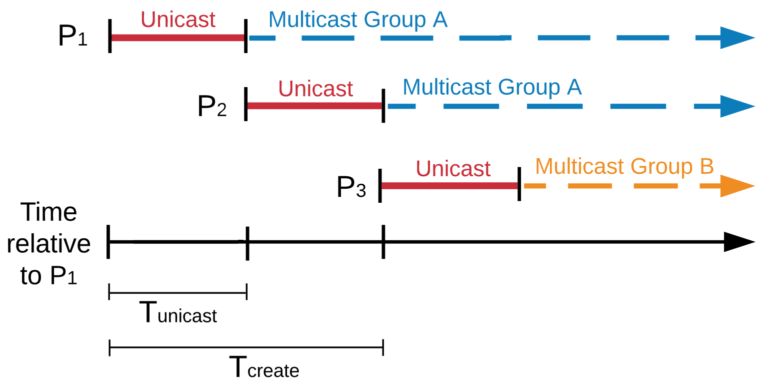


Figure 51: Example of three video sessions P_1, P_2, and P_3

started by a client. While delivering the first $T_{unicast}$ seconds via unicast, the *Peering Node* triggers the creation of multicast group G_A which serves segments for P_1. After $x \leqslant T_{create}$ seconds, a second playback P_2 of the video starts. Since the playback difference between P_2 and $P(G_A)$ is smaller than the minimum time T_{create} (i.e., the minimum playback distance between two multicast groups), P_2 is served by G_A. P_3, however, shows a playback distance between to $P(G_A)$ that is larger than T_{create}. Therefore, VoDCast does not serve P_3 by G_A. In this case, a new multicast group G_B is created serving P_3, which becomes the group owner of G_B. At the one side, this design prevents clients from joining multicast groups that are too far away from their current playback position and, thereby from filling their buffers with segments which are less likely to be watched compared with segments close to the client's playback position. On the other side, several multicast groups per video can exist simultaneously. However, only the most popular videos are considered for multicast and only $\frac{\text{video duration}}{T_{create}}$ groups per video can exist. Hence, the number of multicast groups per video has an upper bound.

6.3.2.4 *Video Streaming Client*

The term client refers to the video player on the user device. This player receives, buffers, and playbacks video segments. To receive video segments using multicast, we assume that the client is able to receive segments using both HTTP/TCP and User Datagram Protocol. Furthermore, the client must signal the *Peering Node* if the player is closed to be removed from multicast groups the client was part of. A client can join a maximal number of $n = \frac{\text{available bandwidth}}{\text{video bitrate}}$ multicast groups simultaneously. In the following, we consider an example where a client joins three groups A, B, and C that deliver streams for the same video. The groups deliver the video's segments starting from different playback positions denoted by the video segment number, i.e., segment 10 from A, 15 from B, and 22 from C. In this example, the client leaves group A after receiving segment 14 since segment 15 is received from group B. Group B is left after segment 21 is received since group C delivers segment 22 and all remaining segments. Finally, the client remains a member of C until the client stops its playback and the buffer is filled or it has received all video segments. In case of a video with multiple groups being available and the client can receive more segments than it can buffer, the client leaves the multicast group that delivers segments farthest from its playback position. Therefore, the client prefers groups which deliver segments close to its playback position. If no group delivers a particular segment, it is requested via unicast immediately before it is needed to avoid a playback interruption. The first video segment of a video is always delivered via unicast to minimize the initial stalling until the playback starts. Thereby, the number of SDM group changes is kept small as short streaming sessions are excluded from multicast delivery. In this work, the clients receive segments of 5 seconds length and adapt the video quality similar to the YouTube Android application, which has been observed to switch to 1080p after playing about 20 seconds in 480p. We do not consider any further quality adaptations.

Using VoDCast, the video buffer is filled quickly by downloading one segment after another till the buffer is at least filled with $T_{unicast}$ segments. This is an important behavior as some related works ignore the impracticality of letting users wait for a playback start more than a few seconds, which decreases the user experience (cf. Section 2.3.1). For the second segment, the video is allowed to be received from multicast groups.

6.3.2.5 *System Parameters*

Symbol	Definition	Default
S_{seg}	Video segment size	5 s
$T_{unicast}$	Video start unicast interval	20 s
T_{create}	Minimum group creation interval	30 s
T_{join}	Maximum join playback interval	120 s
T_{res}	Minimum residence interval	60 s
C_{buff}	Client buffer space	50 MB
C_t	Client throughput	30 Mbps
P_t	Popularity time window	20 m
P_k	Top k% of popular videos	10%

Table 16: Overview of VoDCast's system parameters

In the following, we provide a complete overview of VoDCast's system parameters. Table 16 lists them together with brief descriptions and default values.

Minimum Group Creation Interval

In case a multicast group already exists for a requested video, the minimum group creation interval T_{create} defines if the client joins the existing group or a newly created one. If the existing group's playback position is close to the video start, new clients will join the existing group. We define this closeness by the parameter T_{create}. A high value can cause the client buffer being entirely filled by segments very distant from the user's playback position. As a consequence, the buffer space is occupied for segments closer to the client's current playback position. Therefore, the maximum value also depends on the buffer size and the video bitrate. Ideally, the amount of storage required to store T_{create} seconds of video content should be smaller than the available buffer space C_{buff}, i.e. $T_{create} \leqslant \frac{S_{seg} \times \text{bitrate}(S_{seg})}{C_{buff}}$. Thereby, it is ensured that buffered video segments are consumed before the buffer is full. On the one side, large values are likely to result in a larger share of the video content being delivered via unicast, in the case the buffer is filled with segments not directly consumed by the client, where the currently needed segments have to be requested via unicast. On the other side, small values lead to an increased number of multicast groups and group changes. We consider T_{create} between 10 s and 60 s as a reasonable value since it does not use the entire client buffer but supports efficient multicast of video segments needed later on during the playback. In the following, we use $T_{create} = 30s$ as the default value.

Maximum Join Playback Interval

The maximum join playback distance T_{join} defines the maximum allowed difference between the client's and the group's playback position measured in seconds. It defines whether a client joins an already existing group. This parameter prevents the buffer from being filled with segments too far away from the client's playback position. T_{join} should be chosen depending on the expected average playback duration of the VoD content. By default, we set this parameter equal to the median of the playback session durations observed in our request trace (i.e., $d_{join} = 120s$). This allows a client to join in groups that are at most 2 minutes away from its playback position and prevents downloading segments which are unlikely to be watched. Note that a small T_{join} reduces the number of opportunities a client can join a group.

Minimum Residence Time

The minimum residence time T_{res} defines the time interval a client is able to receive segments from a group. In case the remaining group playback time is shorter than T_{res}, a client will not join. This prevents clients from short group stays and, thereby reduces the multicast management overhead. A larger value reduces group join opportunities. Thus, this parameter defines a trade-off between performance and resource consumption. We consider T_{res} to be set appropriately in a range between 40-120 seconds since the average YouTube video length is reported to be about 4.5 minutes long while the shortest observed video was 40 s and the longest almost 10 minutes long[2]. In the following, we set $T_{res} = 60s$ to exclude videos shorter than 1 minute.

Popularity Time Window

The popularity time window P_t defines the size of the sliding window for which popularity statistics are stored. It is used to continuously estimate which videos are most efficiently delivered using multicast. The reasoning behind this estimation is that video popularity changes over the course of the day and the video's live time [CDL08a]. Accordingly, a time window that is chosen too small results in an unstable set of videos, even for short time intervals. Choosing a too large value for P_t stabilizes the set of videos but also leads to a decision that is more oriented towards the past, giving low emphasis to recent popularity changes. In the following, we consider 20 minutes as a reasonably large time interval to observe a representative sample of multiple requests to popular content [CE17] (cf. Figure 34).

[2] https://www.minimatters.com/youtube-best-video-length/ [Accessed: November 19, 2018]

Hot Stream Ratio

The number of videos traced within P_t is usually still high using our YouTube trace (cf. Section 5.4.2). Hence, we introduce a hot stream ratio that determines the percentage of the most popular videos observed in P_t which are considered for multicast. We denote the hot stream ratio as P_k.

6.4 EVALUATION

This section presents the evaluation of the VoDCast mechanism. We present the methodology used, including workload and network model in Section 6.4.1. The results of VoDCast's SDN flow entry changes and flow entry state are discussed in Section 6.4.2. Section 6.4.3 presents VoDCast's effect on network load.

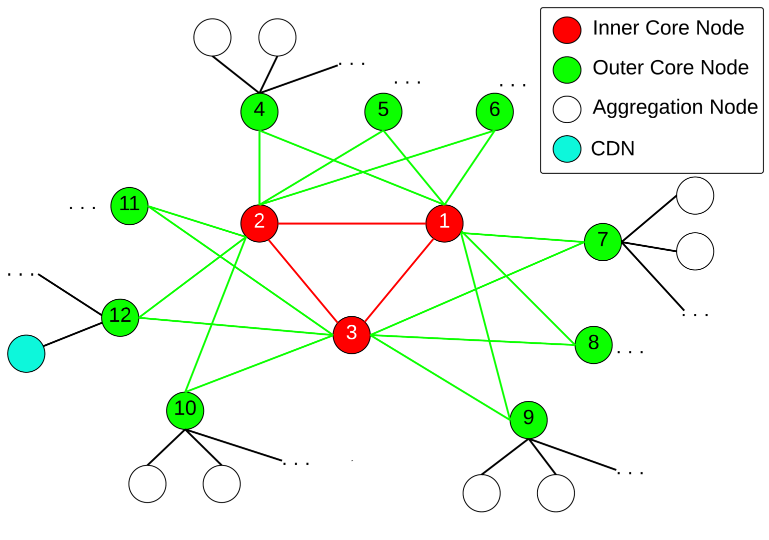


Figure 52: Used ISP core topology, with some aggregation nodes

6.4.1 *Methodology*

For the evaluation, the same YouTube request trace dataset introduced in Section 5.4.2 is used. Since VoDCast's performance depends on the ISP network topology, we chose a simplified version of a representative real-world topology from the Deutsche Telekom as described by Betker et al. [BGK+14]. Figure 52 depicts the inner and outer core nodes of the topology as well as exemplary aggregation nodes. The inner core is shown in red and is fully meshed. The inner core nodes are connected to nine outer core nodes (4-12), depicted in red. Each outer core node is connected to two different inner core nodes. These nodes are distributed over an entire country and connected by fiber-optical links. Overall, 900 aggregation nodes exist in the topology [BGK+14]. They logically connect to one of the outer core nodes. In the scenario considered, these 912 nodes are equipped with SDN-enabled hardware switches. Current standardization efforts show the interest in applying SDN to fixed and mobile networks, as initiatives such as the ONF Wireless and Mobile Working Group (WMWG) [Ope13] indicate. We assume VoDCast to be installed on outer core nodes which are connected to CDNs. The delay between an aggregation node and a client is considered to be between 20-30 ms following the real-world measurement results of to Kaup et al. [KMB+15], where two Round-Trip Time clusters are observed between 20 and 30 ms for a 4G network. Following these results, in the conducted trace-driven simulations, the clients are directly connected to the aggregation nodes with a delay of 30 ms. The delay between aggregation nodes and close-by CDNs is assumed to range between 40-70 ms according to Casas et al. [CFS14] and CDNPerf[3]. Assuming a conservative scenario, the delay between CDNs and clients is modeled uniformly distributed with 115 ms.

In the following, we discuss the simulative evaluation of VoDCast, including our novel two-stage evaluation, the simulation setup, and the multicast tree construction procedure.

Stage 1: Multicast Group Simulation

Figure 53 exhibits our two-stage evaluation methodology. Here, the user request trace serves as an input for the *Multicast Group Simulation*, which is based on the discrete event simulator Omnet++[4]. The simulation produces a trace of multicast group events, i.e., join, leave, remove, and add (cf. Section 6.3.2.2). Therefore, for each of the videos contained in the trace, a multicast group is created on the first client request for this video and all subsequent requests which are at least T_{create} seconds from the previous group creation time away. By coupling the group creation to the client requests and not just creating a new group every T_{create} seconds, multicast groups are only created if there is an actual demand. The *Multicast Group Simulation's* trace comprises group changes denoted by tuples of the format: $(\text{time}, \text{multicast operation}, \text{group ID}, \text{group members})$.

Stage 2: Tree Construction & Flow Entry Inference

The *Tree Construction & Flow Entry Inference* module takes the output of the first stage as an input. Here, in a first step, the users contained in the trace are, uniformly at random, assigned to an aggregation node of the used topology (cf. Figure 52) for each simulation run. Thereby, it is ensured that the measurements are not a result of an ideal assignment of users with similar video consumption to the same aggregation nodes which would introduce a bias leading to unrealistically high bandwidth reduction. During the simulation, a group's multicast tree is recomputed upon each group change. This makes it possible to derive the bandwidth used during the simulation per link using the videos' bitrates. Finally, as each node represents an SDN-enabled hardware switch, statistics about SDN flow entries and their changes can be derived on a per switch basis, which we will detail in the next section.

6.4.2 *Flow Entry State and Changes*

The number of multicast flow entries and their changes state practical limitations of SDN-enabled switches [KRE+15]. The reason for this is that the Ternary Content-Addressable Memory (TCAM) required for accessing these entries is limited in size. Therefore, we measure the rate of flow entry changes and the number of flow entries

[3] `https://www.cdnperf.com/` [Accessed: November 19, 2018]
[4] `https://omnetpp.org/` [Accessed: November 19, 2018]

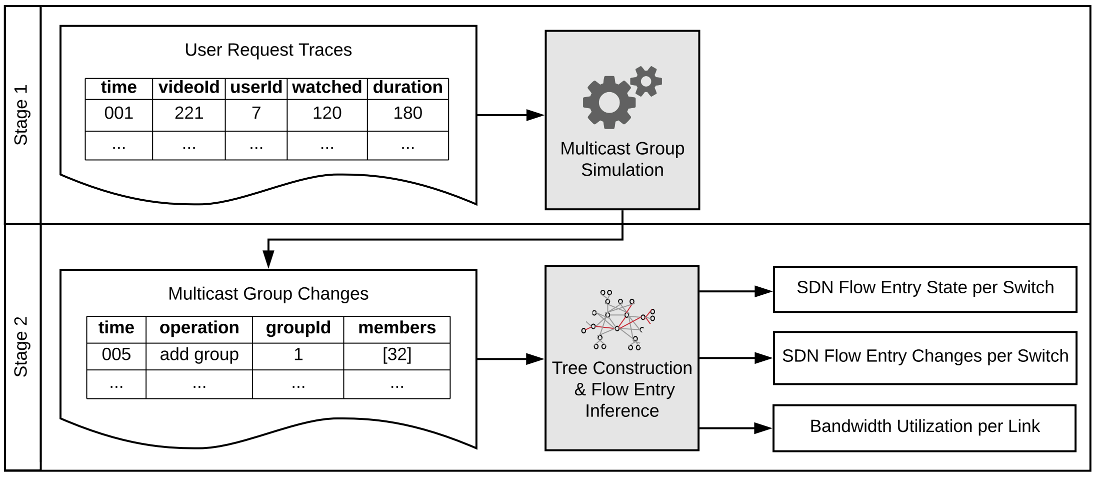

Stage 1
User Request Traces
time | videoId | userId | watched | duration
001 | 221 | 7 | 120 | 180
... | ... | ... | ... | ...
Multicast Group Simulation
Stage 2
Multicast Group Changes
time | operation | groupId | members
005 | add group | 1 | [32]
... | ... | ... | ...
Tree Construction & Flow Entry Inference
SDN Flow Entry State per Switch
SDN Flow Entry Changes per Switch
Bandwidth Utilization per Link

Figure 53: Two-stage simulation architecture using a request trace as an input

stored per switch. By doing so, we show that both metrics are bounded at a low level and, hence, that VoDCast is practicable.

During the simulation, several multicast events occur. On a regular time interval, a snapshot SP_t is taken, containing a batch of these events. For each SP_t, the number of groups, flow entries and flow entry changes per switch, as well as the bandwidth per link, are measured. In each snapshot, the network is modeled as a graph consisting of nodes, e.g., $u, v \in V$ connected by edges $(u, v) \in E$. We distinguish two node types: switches $S \subset V$ and clients $C \subset V$. The number of flow entries F installed during a snapshot is determined as follows: We assume a multicast group $g \in G_{SP_t}$ of a particular snapshot and a set of switches $S_{SP_{t,g}}$ holding flow entries for the multicast tree of this group. These tree nodes consist of clients $c \in C_{SP_{t,g}}$ and switches $s \in S_{SP_{t,g}}$. Each used switch s needs exactly one SDN flow entry per group to determine where the incoming multicast packets have to be forwarded to. We define the number of installed flow entries $F(SP_t)$ with Equation 6. Relying on this equation, we define the flow entry changes $FC(SP_t)$ by comparing the installed flow entries at SP_{t-1} and the current snapshot SP_t with Equation 7.

$$F(SP_t) = \sum_{\forall g \in G_{SP_t}} |S_{SP_{t,g}}| \tag{6}$$

$$FC_{SP} = |(f \in F_{SP_t}, f \notin F_{SP_{t-1}}) \cup (f \in F_{SP_{t-1}}, f \notin F_{SP_t})| \tag{7}$$

Figure 54 and Figure 55 depict the number of rules and their changes per minute for three different classes of switches:

- switch 12 which is an intermediate node between the CDN and all clients,
- switches 2 and 3 which are directly connected to switch 12,
- and all other switches 4 - 11.

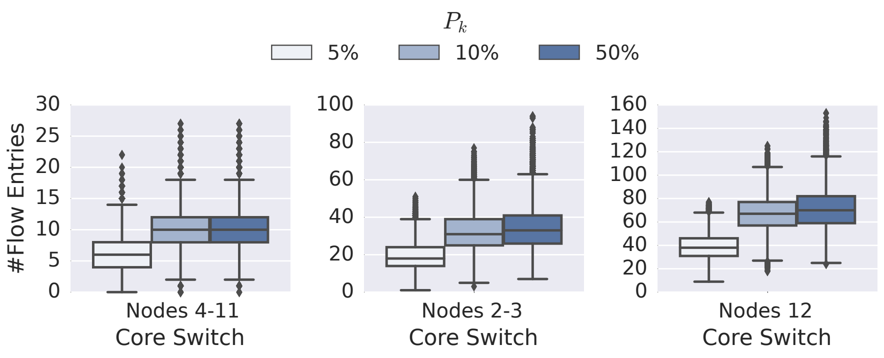


Figure 54: Maximum number of flow entries per minute

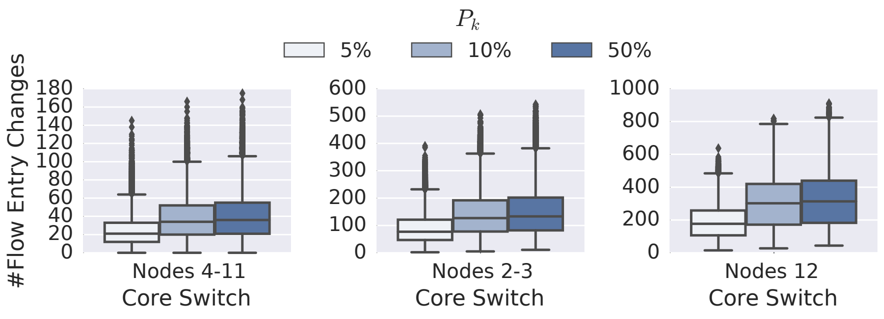


Figure 55: Maximum number of flow entry changes per minute

Switch 1 is not considered in the evaluation because none of the shortest paths from the CDN to the clients relies on it, as only one *Peering Node* is considered. An additional *Peering Node* placed on any outer core node except the nodes 11, 12, and 13 would also involve inner core node 1, and consequently improve load distribution. Figure 54 shows the maximal number of SDN flow entries, e.g., OpenFlow rules, forwarding network traffic for all core switches. By using VoDCast, the median of the maximum number of flows installed on the switch per minute is below 30 for all switches and different configurations of P_k. Here, 2, 3, and 12 are an exception since we observe less than 100 installed flow entries during the two weeks' evaluation. Even with the highest hot stream ratio $P_k = 50$, the 75-percentile is 84 for switch 12 and, hence bound at a low level. We see the highest number of flow entries at switch 12 with 160 entries. This is about two magnitudes lower than the number of flow entries supported by current hardware switches, offering space for up to 60k flow entries [KRE+15]. Note that the nodes in our topologies refer to the point of presence where typically multiple switches are installed. Figure 55 depicts the maximum numbers of flow entry changes per minute, e.g., OpenFlow rule modifications and removals sent to a single switch. The figure shows that the flow entry changes caused by VoDCast are well within the capabilities of nowadays hardware switches which supports up to 32k flow entry changes [KRE+15] per minute. The mean of all core switches, except 2, 3, and 12 is about 10 changes per minute. For the highest configuration $P_k = 50$, no values larger than 912 are observed. A deployment with more than one *Peering Node* would further reduce the load on switch 12, 2, and 3.

6.4.3 *Bandwidth Utilization*

In the following, we explain how we determine the bandwidth utilization of VoDCast. Thus we first compute the bandwidth demand of one video segment $s \in S_{(u,v)}$ sent over an edge $(u, v) \in E$. The consumed bandwidth depends on their duration S_{seg} and the bitrate $r_{s,q}$ of the used quality level q. Equation 8 defines how the bandwidth of all segments on one link (u, v) is calculated.

$$B(S_{(u,v)}) = \sum_{s \in S} (S_{seg} * r_{s,q}) \tag{8}$$

To calculate the bandwidth of group G_j, we sum the bandwidth over all used edges $(u, v) \in E_{G_j}$. The segments delivered to this group are defined as $s \in S_{G_j}$. Consequently, the bandwidth consumed by group G_j is defined by Equation 9.

$$B_{G_j} = \sum_{\forall (u,v) \in E_{G_j}} B(u, v) = \sum_{s \in S_{G_j}} (S_{seg} * r_{s,q} * |E_{G_j}|) \tag{9}$$

Following this methodology, we compute the bandwidth consumed over all groups for each 30-minute interval between 9 am and 11:30 pm for 14 days. We left out the night hours intentionally because there is only few network traffic and the benefit of using VoDCast is limited. Compared with unicast, VoDCast has the potential to reduce costs for ISPs and CDNs as the traffic load variance, as well as the overall network load, can be efficiently reduced. To do so, only a low number of flow entries and flow entry changes is required. The share of unicast and multicast caused by VoDCast compared to the highest unicast load per hour of the day, for different configurations, is depicted in Table 17. Here, we depict the hot stream ratio P_k, the bandwidth reduction $\Delta\mu$, and the reduction of unicast traffic's standard deviation $\Delta\sigma$. $\Delta\mu$ shows the link utilization reduction achieved by VoDCast. It is defined as the difference between the link utilization using VoDCast and in a pure unicast scenario without VoDCast. In the table, $\Delta\sigma$ represents the reduction in standard deviation for the unicast achieved by VoDCast. We see that the bandwidth reduction and variance decrease with increasing size of P_k, up to 3.9% and 6.1% reduction of the standard deviation. However, the increase flattens when we increase P_k from 10% to 50%.

Hot stream ratio (P_k):		0.01%	1%	5%	10%	50%
V-MC	mean	0	1.9	4.4	8.1	8.7
	median	0	2	4	8	9
	std. dev.	0	0.7	1.3	1.6	1.9
V-UC	mean	68.7	67.6	64.5	59.9	59.1
	median	76.5	76	71	65	65
	std. dev.	22.5	21.4	19.3	16.6	15.9
Bandwidth Reduction ($\Delta\mu$)		1.9	2.2	2.8	3.7	3.9
Std. Dev. Reduction ($\Delta\sigma$)		0	0.6	2.7	5.4	6.1

Table 17: VoDCast's Multicast (V-MC) and Unicast (V-UC) traffic and the achieved bandwidth reduction measured by volume ($\Delta\mu$) and std. dev. ($\Delta\sigma$)

6.5 SUMMARY AND DISCUSSION

In this chapter, we designed and evaluated a novel multicast mechanism for OTT VoD delivery, denoted as VoDCast. It is highly configurable and delivers streams with small playback differences partially via multicast. Thereby, it considers efficient client buffer management by preventing video segments too far from the client's playback position from being buffered. This acknowledges that most video streams on YouTube are only watched partially. Furthermore, VoDCast supports the cooperation with CDNs by offering a *CDN Service API* but can also be operated by the ISP only in case of ISP-internal VoD services or unencrypted OTT VoD traffic. The results presented rely on a real-world YouTube workload over two weeks and show the impact of realistically chosen key parameter settings on the achievable bandwidth reduction. By using VoDCast, the ISP's network load is reduced by 3.9% and its standard deviation decreased about 6% compared to unicast. This indicates VoDCast's potential to lower transit costs and reduce network congestions that can negatively impact the QoE. While 3.9% might sound like a minor improvement, VoDCast's third use case presented in Section 6.2 is likely to increase the saved traffic among multiple ISPs even more than for the single ISP case considered in this chapter's evaluation. This is because the most popular videos tend to be popular in many ISP networks within a country and the typical market share of one ISP is typically not larger than 40%, e.g., for mobile networks in Germany[5] or in the US[6]. Hence, using VoDCast results in a considerable large amount of saved traffic considering the large forecasted growth of video traffic. Assuming CDN transit costs between 0.02-0.13$[7] per GB, depending on the geographic region, and a VoD data traffic volume of 509 Exabytes [Cis17a], the overall traffic costs for VoD traffic are expected to sum up to 10.2-66.2 billion$ by the year 2021. Hence, VoDCast can save between 0.4-2.6 billion dollars while introducing only a low resource footprint as shown in Section 6.4.2. We demonstrate VoDCast's practicability by showing the resource utilization in terms of SDN flow entry state and changes that stay well within the limits of current SDN-enabled hardware switches and at a low level.

[5] https://www.smartweb.de/mobilfunk-report-deutschland-q1-2017 [Accessed: November 19, 2018]

[6] https://www.statista.com/statistics/199359 [Accessed: November 19, 2018]

[7] https://www.rackspace.com/cloud/cdn-content-delivery-network [Accessed: November 19, 2018]

7 SUMMARY, CONCLUSIONS, AND OUTLOOK

In the following, we provide a summary of the thesis, conclude each of our contributions by addressing the research questions separately and highlight promising directions for future work.

7.1 SUMMARY

Video-on-Demand content started to dominate the Internet traffic during the last decade. This strong growth trend has not stopped yet since additional services and use cases, as well as higher qualities, become available at a rapid pace. For example, the advent of 4k and 360° videos are just beginning to become mainstream. Hence, the amount of Video-on-Demand (VoD) traffic is expected to increase further. In parallel to this rapidly growing amount of video transmissions, the required network resources and costs rise. As a result, the delivery of Over-the-Top (OTT) VoD is, already today, challenging for network operators and content providers. The strong traffic growth that is forecasted requires novel and efficient approaches that either enhance the performance of the existing caching infrastructure of Content Delivery Networks (CDNs) or involve the user device as an active contributor in the content delivery process. We identified three key research areas in this thesis which are considered essential to achieve this goal. Within these research areas, we found proactivity and efficiency to be essential requirements.

The first area considers individual user interests and how they can be used to prefetch videos on their mobile terminals in advance. Thereby, the negative consequences for a user's Quality of Experience (QoE), when watching videos using a potentially low-quality mobile Internet connection, are mitigated. Furthermore, this saves energy at the mobile terminals since Wi-Fi offloading is about 20-times less energy demanding than streaming a video over a 4G Internet connection [HQG+12]. Determining video content for prefetching is challenging since on User-generated Content (UGC) VoD platforms, such as YouTube, a vast amount of content exists and intuitive approaches like prefetching from a user's subscribed channels [KLR+17], taking videos recommended by YouTube's landing page [WSS+16], and selecting videos that are popular [WRT+15] results in a poor performance. Therefore, in the first area, the research goal is to support prefetching for individual users on their mobile terminals by learning the user's interests locally on the user's device.

The second area focuses on a network-centric perspective that takes in-network caching and different content popularity distributions into consideration. At the moment, caching by CDNs makes video streaming globally scalable since it delivers content from servers close to the users and, thereby keeps traffic local and provides low latency for the users. The importance of efficient caching mechanisms will gain in importance driven by an increasing amount of video streaming [Cis17c] and the advent of edge caching. Proactive caching has shown to outperform reactive approaches in general [Gou+15; HNH14; KWR+18]. Hence, the goal for the second research area is to support proactive caching for VoD to lower the Internet Service Provider (ISP) and CDN network traffic by considering the distinct popularity distributions of video content categories.

The third area addresses using ISP-internal multicast for popular VoD content in cooperation with the CDN. While multicast is already used for ISP-owned streaming services [LLW+11], it is not yet available for OTT content. The reason for this is that traditional IP multicast requires multicast routers on the delivery path [DLL+00] which are costly and limited in the number of supported multicast groups. This makes IP multicast unscalable for OTT content. However, the broad adoption of Software-defined Networking (SDN) enables scalable multicast, next to other network services. SDN-based multicast has been proposed for OTT live video streaming already [Rüc16] but has not yet adapted for the OTT VoD case. Here, popular content is typically watched thousands of times per day. Hence, many people watch the same video with only minor differences in their playback positions which offers a potential for multicasting parts of these videos to multiple clients. Thus, in the third area, the research goal is to support multicast for popular VoD content to lower the ISP's network traffic while adapting to a dynamic Zipf-distributed content popularity [GHM13; ACG+09; GAL+07] and keeping costs, regarding SDN flow entries and flow entry changes, low.

By reviewing the related work, we identified a lack of specific efficient and proactive delivery mechanisms for OTT VoD. Thereby, we derived a set of research area-specific research questions. These research questions are the focal points in the design and the evaluation of the three contributions towards this research questions as shown in the course of this thesis.

7.2 CONTRIBUTIONS

This thesis presented three contributions which address the common design goal of proactive and efficient VoD content delivery. The design and evaluation of our proposed mechanisms were guided by the research questions identified in Section 1.2. In cases where we could not deduce a direct answer to the research questions, trade-offs were identified that allow us to configure the proposed mechanisms to meet the specific requirements of the application scenario.

7.2.1 *vFetch*

The first contribution is the design of vFETCH, a privacy-preserving prefetching mechanism for mobile devices that we presented in Chapter 4. It focuses explicitly on OTT VoD services such as YouTube. vFETCH learns the user interest towards different content channels and uses these insights to prefetch content on a user terminal. To do so, it continually monitors the user behavior and the mobile connectivity pattern of the device to allow for efficient and resource-saving download scheduling. Thereby, vFETCH illustrates how personalized prefetching saves the mobile data volume and alleviates mobile networks by offloading mobile networks from peak-hour traffic. In the following, we will briefly present how vFETCH answered the research questions identified in Chapter 1.2 in the course of its design and evaluation:

RQ 1.1: *How accurate can video requests of different users be predicted, considering user interests?*

To answer this questions, we analyzed users' request behavior over several months. We found that the channel on which a video is presented to the user has a significant impact on a video's likelihood to be watched. To this end, we modeled the user interests as the ratio of videos watched in the past to get a prediction of the likelihood that the user watches content from this channel in the future.

In the course of the user behavior analysis, we observed that only a minority of video views is caused by videos published on channels that the user has subscribed. Thus, we proposed the concept of pseudo subscriptions, i.e., channels from which a user watches many videos. Pseudo subscriptions are a key contributor to vFetch's superior performance. In our experiments, we investigate the influence of the mobile device's storage capacity on prefetching accuracy measured in precision, recall, and F1-measure.

In the assessment of vFetch, we found that precision and recall were on the same level which indicates a robust and accurate detection of prefetching candidates. However, the level of accuracy was found to depend on the storage size available and is quite heterogeneous for different users. On the largest storage considered, the mean slightly exceeded 60% while for storage larger than 5 GB, some users exhibit a very high F1-measure close to 1. The reason for this is the difficulty of predicting when precisely a user watches a video. Therefore, a larger storage helps to compensate for this uncertainty and stabilized vFetch's performance.

RQ 1.2: *Which Cache Hit Rate (CHR) gain can be achieved by considering content properties for content access prediction?*

In addition to vFetch's accuracy, we evaluated its Byte Hit Rate (BHR) as a more precise measure of the CHR that considers different video file sizes. In our experiments, we observe hit rates of about 10% for cache sizes between 50 MB and 500 MB. For larger cache sizes, the BHR increases almost parallel to the available storage and reaches up to 60% among our users. Similar to the accuracy measurements, we observe distinct users with BHRs close to 100%.

RQ 1.3: *How much of a user's mobile video traffic can be saved using a predictive prefetching model?*

To answer this question thoroughly, we enhanced vFetch's prefetching with request-based caching on the same storage area managed by Least Recently Used (LRU). We observed that request-based caching can further increase vFetch's performance for all cache sizes. Since many YouTube video requests are caused by ephemeral interests that are generally hard to predict, we distinguished two cases. The first case considers only videos from predictable sources, e.g., known channels, the watch later list, and playlists of the user. Here, we see that the BHR cannot benefit from additional request-based caching and there is no significant performance change. Thus, we deduce that vFetch efficiently prefetches content when considering the content sources mentioned above. In the second case, we considered all videos watched by the user, i.e., also random views, e.g., from ephemeral interests. In this case, request-based caching increases the BHR and also the F1-measure significantly. In the first case, caching in combination with prefetching leads to a maximum BHR of about 60% while in the second case we achieve a 15% BHR. Therefore, prefetching in combination with request-based caching is desirable since videos from sources that are not predictable are, in many cases, requested multiple times. Overall, vFetch's performance is considerably larger than for related approaches evaluated on YouTube which achieve a BHR lower than 0.03% as demonstrated by Wilk et al. [WSS+16].

7.2.2 *ProCache*

The second contribution focuses on proactive in-network caching. To this end, we present the design of the ProCache mechanism in Chapter 5. The mechanism splits the cache storage by considering separate content categories. Here, the available storage is allocated to these category-based cache divisions based on their contribution to the overall cache efficiency. We propose a general work-flow that emphasizes multiple categories of a mixed content workload in addition to a work-flow tailored for music video content which is the dominant traffic source on YouTube. Thereby, ProCache shows how content-awareness can contribute to efficient in-network caching. In the following, we will briefly present how ProCache answered the research questions identified in Chapter 1.2 in the course of its design and evaluation:

RQ 2.1: *How can proactivity be used to enhance caching performance?*

We answered this question in the course of the design of ProCache by investigating proactivity in two different components, i.e., the *Storage Manager* and the *Content Recommender*. The *Storage Manager* proactively manages the cache storage by allocating storage to category-specific divisions. To this end, we proposed a set of different Division Size Adaptation Strategy (DSAS). In our experiments, we evaluated these DSASs and found Relative Largest Ghost List (RLGL) to achieve the highest CHR. In case the entire cache storage is occupied and new content has to be cached, RLGL reduces the storage used by the cache division that exhibits the largest ghost list relative to its currently used storage to allow storing the new item in the cache division that corresponds to its YouTube category. Thereby, ProCache continuously adapts to changing content popularity dynamics and content catalog sizes proactively. In our design, the *Storage Manager* can also use the *Request Monitor*'s *Content Recommender* to get a content popularity estimate. This is a further proactive sub-mechanism that we designed for the category of music videos as they are the largest source of requests on YouTube. The *Content Recommender* relies on a *Recommendation Policy* that allows estimating which content is going to be or to stay popular. The resulting set of contents is used to place them proactively on the cache and exclude them from the eviction policy. Accordingly, a particular part of the cache storage is filled proactively instead of reactively as common in traditional caching approaches.

RQ 2.2: *Which prediction policies show the highest performance regarding the CHR?*

We presented a set of prediction policies to answer this question for the work-flow of ProCache that emphasizes music videos. We focused on music as the largest source of requests on YouTube and to design and evaluate a music-specific solution instead of a generic approach. To this end, we trained a genre and a mood classifier to provide the prediction mechanism with characteristic music features. We used these features to investigate and exploit their dynamic popularity distributions over the course of the day. The respective prediction policies were denoted as *Genre* and *Mood*. Furthermore, we evaluated a policy based on user similarity recommendation and a popularity policy that uses the currently popular content as an estimate for the near future. Among these policies, the popularity-based one achieved the highest performance measured in CHR. Here, the maximum achieved CHR was 55.1% in case 20% of the cache storage is managed proactively and the remainder of the cache is managed reactively. When comparing ProCache's music-specific work-flow with LRU, we observe that ProCache can increase the CHR of small caches with up to 4% improvement. Considering the vast amount of forecasted video traffic, this may result in a multi-billion dollars benefit if deployed widely as shown in the exemplary calculation in Section 6.5.

RQ 2.3: *How does cache storage size contribute to the CHR gain of proactive caching?*

To answer this questions, we conducted experiments with cache sizes up to 1 PB of storage. We observed diminishing returns on increasing cache sizes for ProCache as well as for other caching strategies. For cache sizes larger or equal 10 GB, ProCache results in a larger or equal CHR than Segmented Least Recently Used (SLRU) and Adaptive Replacement Cache (ARC), which we used as benchmarks because of their high performance. However, the performance gain of ProCache decreases for large cache sizes while it can still reduce the number of write operations on the cache and decrease the transmission latency. Depending on the cache hierarchy used, ProCache achieved 12%–18% higher CHRs than the next-best caching strategy. Hence, mid-size caches, of 1 GB - 1 TB, contribute the most to ProCache's performance depending on the cache hierarchy.

7.2.3 *VoDCast*

The third contribution targets the application of multicast for VoD scenarios. Many users request popular VoD content with only small differences in their request time which offers a potential for multicast. To this end, we presented the design of VoDCast in Chapter 6, which uses this potential to multicast parts of popular videos. Thereby, VoDCast illustrates how ISPs can collaborate with CDNs to coordinate on popular content to be delivered using VoDCast. Furthermore, we addressed Software-defined Networking as the underlying technology. In the following, we briefly present how VoDCast answered the research questions identified in Chapter 1.2 in the course of its design and evaluation:

RQ 3.1: *How much does multicast lower the traffic volume caused by VoD content within ISP networks?*

We conducted a series of experiments to answer this question. VoDCast's most influential parameter regarding traffic reduction is the hot stream ratio which defines how much of the most popular and most recent content is considered for multicast. We found that the traffic reduction potential has an upper bound at 3.9% with a reduction of the standard deviation of 6.1% when compared to unicast. Given the vast amount of traffic that video streaming causes in today's network, this is a large number that allows reducing transit costs and load within ISP networks. Considering recent transit costs and forecasted VoD traffic volume, VoDCast may reduce the overall traffic costs by up to 66.2 billion dollars by the year 2021 (cf. Section 6.5).

RQ 3.2: *How can VoD multicast be realized using SDN?*

SDN is an enabling technology for multicasting OTT content. To this end, we proposed the design of the CDN Service Application Programming Interface (API) that allows ISPs and CDNs to negotiate on videos to deliver by ISP-internal multicast. In our experiments, we rely on a simple version of Software-defined Multicast (SDM), which was originally proposed for live video streaming. However, to adopt this approach to be useful for VoD content, suitable videos for multicast need to be determined. VoDCast supports two ways to achieve this. First, in a collaborative scenario between CDN and ISP, the CDN can propose content using VoDCast's service API. Second, in case of an ISP-internal VoD service, VoDCast's *Statistics Server* continuously traces the content popularity. Thereby, suitable content is identified that is transferred using SDM to the demanding clients.

RQ 3.3: *Which VoD content should be selected for being delivered by multicast to decrease the data traffic volume?*

Multicast for VoD content is only reasonable for popular content to leverage simultaneous streams with small playback position differences. We propose to rely on the CDN's recommendation for this task since it knows best which content is demanded by multiple users of the respective ISP. Alternatively, VoDCast can use its *Statistics Server* to determine popular content. We determine this content by tracing video requests by a sliding window approach over time. From the content observed, the most popular share is considered for multicast as defined by the hot stream ratio. We evaluated different values for the hot stream ratio and found 10% of the most popular content within a 30 minutes' time interval to result in a reasonable performance while larger values showed diminishing returns. Thereby, a significant traffic and cost reduction, considering burstable billing, is achieved.

7.3 OUTLOOK

In the following, we name a set of research directions which can further complement the mechanisms proposed in the course of this thesis. We designed and evaluated vFetch, ProCache, and VoDCast by separate software architectures. However, the reproducibility and integrability of all three mechanisms can benefit by porting them into a common network evaluation architecture, e.g., the one proposed by Frömmgen et al. [FSK+18]. Furthermore, we designed and studied the three mechanisms on the relevant example of YouTube. As an extension, it is desirable to have an analytical model which supports estimating a mechanism's performance from traffic monitoring [BRB+16; KHR+18] given parameters that characterize the workload, the content catalog, and the popularity distributions. Thereby, the parameter configuration of the presented mechanisms can be adapted automatically to different application scenarios but also for highly dynamic environments that require a timely mechanism adaptation.

vFetch

Concerning privacy-preserving mobile prefetching, it is promising to evaluate how machine learning methods such as Natural Language Processing and face recognition could contribute to vFetch's performance. Statistical analysis and rule-based decision logic are important to explain how behavioral threads can be leveraged regarding prefetching. Though, machine learning is a promising approach to identify and learn such rules and patterns automatically. However, considering the advent of deep-learning-supporting chips on smartphones[1], this is likely to become computationally efficient on smartphones, e.g., overnight when charging. Furthermore, textual features such as the video title can be used to determine a user's topic interests and, thus can further increase the performance of vFetch. Therefore, Natural Language Processing (NLP) techniques can be used to determine video topics, e.g., by its title or audio transcript to conduct an estimate if the video's content is within the user's interests. Face recognition makes it possible to determine which persons appear in a video. This information is promising considering vFetch's *Interest Model* as it is likely not solely to depend on the video's topic but also on the user's interest towards distinct persons. Furthermore, the number of YouTube actors depicted in a video has been shown to make it more attractive for many users and, hence popular as shown in a previous work by the author [KLS+18]. Regarding vFetch's scheduling mechanism, it is a promising direction to include the mobile operator in this process. By signaling the availability of network capacities, vFetch can contribute even more in offloading mobile networks. Additionally, high download rates can be achieved which saves energy. Users can be additionally motivated to cooperate, e.g., by an increased high-speed data volume. A detailed concept on how vFetch can be extended for this purpose is given in one of the author's publications [KBR+15] and vFetch's extended architecture is presented in the appendix in Section A.3.

ProCache

ProCache can be extended in a way that supports cache strategy transitions for different workloads than the YouTube trace used in this thesis. Furthermore, this extension would enable ProCache to adapt to highly dynamic and rapidly changing workloads by adapting its parameters, e.g., the cache division's ghost list Time To Live (TTL) values or the maximum size of the probationary cache division. In addition, workload-aware mechanism transitions (cf. Section 2.6) of the admission and eviction policies for distinct cache divisions, based on content popularity distribution and available storage, are a further direction of research. In this thesis, ProCache has been evaluated regarding its CHR, transmission delay, and the number of write operations. This set of metrics can benefit from being extended by a QoE estimate. To this end, an application-specific set of different video qualities has to be considered in conjunction with quality adaption strategies as they have shown to significantly influence the QoE [SFR+17], depending on scenario and workload. In addition to the content category or music specific features, other modalities are interesting to be evaluated for their potential to identify suitable categories for cache division. By creating an embedding with the help of a Deep Neural Network (DNN), a set of clusters that refer to content with similar properties can be derived and guide the cache division. Therefore, multiple modalities come into consideration in the case of music for which DNNs have been proposed already, but not yet applied for this

[1] https://www.androidpit.com/machine-learning-and-ai-on-smartphones

use case. Guidance on the training of DNNs can be found in works focusing on audio [VDS13], text [ESH15; MCC+13], as well as multi-modal approaches that consider image data [SS12; NKK+11].

VoDCast

For SDN-based streaming of OTT VoD content, we considered the first step in an incremental deployment with just one VoDCast-supporting node in the network. It is promising to evaluate how multiple VoDCast nodes can contribute to the mechanism's performance. Furthermore, the trade-off between SDN flow entries used by VoDCast in the presence of other SDN applications can be modeled as a reinforcement learning problem, similar to KhudaBukhsh et al. [KRF+17], that tries to balance cost and benefit of multiple concurrent SDN applications. This is promising to efficiently manage VoDCast's usage in highly dynamic scenarios where the cost and the benefit of distinct SDN applications and their number vary dynamically. Besides, VoDCast can benefit from extensions that support SDN flow entry load balancing among the switches within the ISP network [BRV+15; RBH+16] to distribute the state caused by the mechanism and load-balance traffic. In addition, it is promising to extend VoDCast by a soft-transition meta-mechanism that determines the optimal system parameters during runtime [PKW+17]. Thereby, the system can be flexibly adapted for workloads of different content catalog size and popularity dynamics as well as network environments with varying loss and failure rates [HRR+17a]. Furthermore, an analysis of predictive methods for identification of videos which will be most popular in the near future and, therefore suitable to be delivered via multicast, will further decrease the bandwidth utilization achievable by using VoDCast.

BIBLIOGRAPHY

[ACG+09] V. Aggarwal, R. Caldebank, V. Gopalakrishnan, R. Jana, K. K. Ramakrishnan, and F. Yu. “The Effectiveness of Intelligent Scheduling for Multicast Video-on-Demand.” In: *ACM International Conference on Multimedia (MM)*. 2009 (cit. on pp. 6, 57–59, 133).

[ACS03] I. Arsovski, T. Chandler, and A. Sheikholeslami. “A Ternary Content-addressable Memory (TCAM) based on 4T Static Storage and including a Current-race Sensing Scheme.” In: *IEEE Journal of Solid-State Circuits* 38.1 (2003), pp. 155–158 (cit. on p. 32).

[ADA+13] A. Arvidsson, M. Du, A. Aurelius, and M. Kihl. “Analysis of User Demand Patterns and Locality for YouTube Traffic.” In: *IEEE International Teletraffic Congress (ITC)*. 2013, pp. 1–9 (cit. on p. 18).

[AFA16] A. Ahmad, A. Floris, and L. Atzori. “QoE-aware Service Delivery: A Joint-venture Approach for Content and Network Providers.” In: *IEEE International Conference on Quality of Multimedia Experience (QoMEX)*. 2016, pp. 1–6 (cit. on pp. 21, 119).

[AFA17] A. Ahmad, A. Floris, and L. Atzori. “OTT-ISP Joint Service Management: A Customer Lifetime Value based Approach.” In: *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2017, pp. 1017–1022 (cit. on p. 119).

[AJC+12] V. K. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang. “Vivisecting YouTube: An Active Measurement Study.” In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2012, pp. 2521–2525 (cit. on pp. 2, 43).

[AMM+03] K. Andreev, B. Maggs, A. Meyerson, and R. Sitaraman. “Designing Overlay Multicast Networks for Streaming.” In: *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. 2003, pp. 149–158 (cit. on p. 12).

[AS10] A. Abhari and M. Soraya. “Workload Generation for YouTube.” In: *Multimedia Tools and Applications* 46 (2010), pp. 91–118 (cit. on pp. 18, 182, 183).

[ASK+10] B. Ager, F. Schneider, J. Kim, and A. Feldmann. “Revisiting Cacheability in Times of User Generated Content.” In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2010, pp. 1–6 (cit. on p. 44).

[AV07] D. Arthur and S. Vassilvitskii. “k-means++: The Advantages of Careful Seeding.” In: *ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics. 2007, pp. 1027–1035 (cit. on p. 26).

[AWY96a] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. "A Permutation-based Pyramid Broadcasting Scheme for Video-on-Demand Systems." In: *IEEE International Conference on Multimedia Computing and Systems (ICMCS)*. 1996, pp. 118–126 (cit. on p. 54).

[AWY96b] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. "On optimal Batching Policies for Video-on-Demand Storage Servers." In: *IEEE International Conference on Multimedia Computing and Systems (ICMCS)*. 1996, pp. 253–258 (cit. on pp. 53, 56).

[Bas15] D. K. Base. *Hard Drive - Why Do Solid State Devices (SSD) Wear Out*. 2015. URL: https://goo.gl/VMqkkZ (cit. on pp. 45, 104).

[BBD+13] K. Brunnström, S. A. Beker, K. De Moor, A. Dooms, S. Egger, M.-N. Garcia, T. Hossfeld, S. Jumisko-Pyykkö, C. Keimel, M.-C. Larabi, et al. "Qualinet White Paper on Definitions of Quality of Experience." In: (2013) (cit. on p. 20).

[BBD14] E. Bastug, M. Bennis, and M. Debbah. "Living on the Edge: The Role of Proactive Caching in 5G Wireless Networks." In: *IEEE Communications Magazine* 52.8 (2014), pp. 82–89 (cit. on pp. 4, 43, 48, 50, 51).

[BBD16] E. Baştuğ, M. Bennis, and M. Debbah. "Proactive Caching in 5G Small Cell Networks." In: *Towards 5G: Applications, Requirements and Candidate Technologies* (2016), pp. 78–98 (cit. on pp. 48, 50).

[BBK+15] E. Baştuğ, M. Bennis, M. Kountouris, and M. Debbah. "Cache-enabled Small Cell Networks: Modeling and Tradeoffs." In: *EURASIP Journal on Wireless Communications and Networking* 2015.1 (2015), p. 41 (cit. on p. 48).

[BBS14] Z. S. Bischof, F. E. Bustamante, and R. Stanojevic. "Need, Want, Can afford: Broadband Markets and the Behavior of Users." In: *ACM SIGCOMM Internet Measurement Conference (IMC)*. 2014, pp. 73–86 (cit. on p. 2).

[BFS+84] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. CRC press, 1984 (cit. on p. 24).

[BGK+14] A. Betker, I. Gamrath, D. Kosiankowski, C. Lange, H. Lehmann, F. Pfeuffer, F. Simon, and A. Werner. "Comprehensive Topology and Traffic Model of a Nationwide Telecommunication Network." In: *Journal of Optical Communications and Networking* 6.11 (2014), pp. 1038–1047 (cit. on pp. 10, 11, 126).

[BGS+14] D. S. Berger, P. Gland, S. Singla, and F. Ciucu. "Exact Analysis of TTL Cache Networks." In: *Elsevier Performance Evaluation* 79 (2014), pp. 2–23 (cit. on p. 46).

[BHC+15] D. S. Berger, S. Henningsen, F. Ciucu, and J. B. Schmitt. "Maximizing Cache Hit Ratios by Variance Reduction." In: *ACM SIGMETRICS Performance Evaluation Review* 43.2 (2015), pp. 57–59 (cit. on p. 46).

[BIV+13] T. Broxton, Y. Interian, J. Vaver, and M. Wattenhofer. "Catching a Viral Video." In: *Journal of Intelligent Information Systems* 40.2 (2013), pp. 241–259 (cit. on pp. 38, 40).

[BJS13] X. Bai, F. Junqueira, and A. Silberstein. "Cache Refreshing for Online Social News Feeds." In: *ACM International Conference on Information and Knowledge Management (CIKM)*. 2013 (cit. on p. 47).

[BNP+15] N. Baranasuriya, V. Navda, V. N. Padmanabhan, and S. Gilbert. "QProbe: Locating the Bottleneck in Cellular Communication." In: *ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. 2015, p. 33 (cit. on p. 45).

[BPV08] R. Buyya, M. Pathan, and A. Vakali. *Content Delivery Networks*. Vol. 9. Springer Science & Business Media, 2008 (cit. on p. 13).

[BRB+16] Z. Bozakov, A. Rizk, D. Bhat, and M. Zink. "Measurement-based Flow Characterization in Centrally Controlled Networks." In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2016, pp. 1–9 (cit. on p. 137).

[BRV+15] J. Blendin, J. Rückert, T. Volk, and D. Hausheer. "Adaptive Software Defined Multicast." In: *IEEE Conference on Network Softwarization (NetSoft)*. 2015, pp. 1–9 (cit. on pp. 58, 59, 139).

[BRZ+17] D. Bhat, A. Rizk, M. Zink, and R. Steinmetz. "Network Assisted Content Distribution for Adaptive Bitrate Video Streaming." In: *ACM Multimedia Systems Conference (MMSys)*. 2017, pp. 62–75 (cit. on pp. 44, 47, 49, 51).

[BSR+06] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz. "Heuristics for QoS-aware web service composition." In: *IEEE International Conference on Web Services (ICWS)*. 2006, pp. 72–82 (cit. on p. 20).

[CBV+15] M. Claeys, N. Bouten, D. D. Vleeschauwer, W. V. Leekwijck, S. Latré, and F. D. Turck. "An Announcement-based Caching Approach for Video-on-Demand Streaming." In: *ACM International Conference on Network and Service Management (CNSM)*. 2015, pp. 310–317 (cit. on pp. 47, 48, 51).

[CDF+14a] P. Casas, A. D'Alconzo, P. Fiadino, A. Bär, and A. Finamore. "On the Analysis of QoE-based Performance Degradation in YouTube Traffic." In: *IEEE International Conference on Network and Service Management (CNSM)*. 2014, pp. 1–9 (cit. on p. 2).

[CDF+14b] P. Casas, A. D'Alconzo, P. Fiadino, A. Bär, A. Finamore, and T. Zseby. "When YouTube does not work—Analysis of QoE-relevant Degradation in Google CDN Traffic." In: *IEEE Transactions on Network and Service Management (TNSM)* 11.4 (2014), pp. 441–457 (cit. on p. 2).

[CDL08a] X. Cheng, C. Dale, and J. Liu. "Statistics and Social Network of YouTube Videos." In: *IEEE International Workshop on Quality of Service (IWQoS)*. 2008 (cit. on p. 125).

[CDL08b] X. Cheng, C. Dale, and J. Liu. "Statistics and Social Network of YouTube Videos." In: *International Workshop on Quality of Service (IWQoS)*. 2008, pp. 229–238 (cit. on pp. 83, 86, 88).

[CE17] N. Carlsson and D. Eager. "Ephemeral Content Popularity at the Edge and Implications for on-demand Caching." In: *IEEE Transactions on Parallel and Distributed Systems* 28.6 (2017), pp. 1621–1634 (cit. on p. 125).

[CFS14] P. Casas, P. Fiadino, and A. Sackl. "YouTube in the Move: Understanding the Performance of YouTube in Cellular Networks." In: *IFIP Wireless Days (WD)*. 2014 (cit. on p. 127).

[Cho+17] K. Choi et al. "Transfer Learning for Music Classification and Regression Tasks." In: *International Society of Music Information Retrieval (ISMIR)*. 2017 (cit. on p. 27).

[Cis17a] Cisco. *Cisco Visual Networking Index: Forecast and Methodology, 2016-2021*. Tech. rep. Cisco, 2017 (cit. on p. 131).

[Cis17b] Cisco. *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update*. Tech. rep. Cisco, 2017 (cit. on p. 14).

[Cis17c] Cisco Systems, Inc. *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021*. Tech. rep. 2017 (cit. on pp. 1, 19, 118, 132).

[Cis17d] Cisco Systems, Inc. *The Zettabyte Era: Trends and Analysis*. Tech. rep. 2017 (cit. on pp. 1, 13).

[CKK17] L. E. Chatzieleftheriou, M. Karaliopoulos, and I. Koutsopoulos. "Caching-aware Recommendations: Nudging User Preferences towards Better Caching Performance." In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2017, pp. 1–9 (cit. on pp. 47–49, 51).

[CKR+07] 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: *ACM SIGCOMM Internet Measurement Conference (IMC)*. 2007, pp. 1–14 (cit. on pp. 6, 54, 122).

[CKS02] I. Cidon, S. Kutten, and R. Soffer. "Optimal Allocation of Electronic Content." In: *Computer Networks* 40.2 (2002), pp. 205–218 (cit. on p. 43).

[CL09] X. Cheng and J. Liu. "NetTube: Exploring Social Networks for Peer-to-Peer Short Video Sharing." In: *IEEE International Conference on Computer Communications (INFOCOM)*. Apr. 2009 (cit. on pp. 39, 42, 79).

[Cla16] D. D. Clark. "The Contingent Internet." In: *Daedalus* 145.1 (2016). MIT Press, pp. 9–17 (cit. on p. 2).

[CLD13] X. Cheng, J. Liu, and C. Dale. "Understanding the Characteristics of Internet short Video Sharing: A YouTube-based Measurement Study." In: *IEEE Transactions on Multimedia* 15.5 (2013), pp. 1184–1194 (cit. on p. 101).

[CLN04] Y. Cui, B. Li, and K. Nahrstedt. "oStream: Asynchronous Streaming Multicast in Application-layer Overlay Networks." In: *IEEE Journal on Selected Areas in Communications (JSAC)* 22.1 (2004), pp. 91–106 (cit. on p. 57).

[CM13] S. A. Chowdhury and D. J. Makaroff. "Popularity Growth Patterns of YouTube Videos-A Category-based Study." In: *International Conference on Web Information Systems and Technologies (WEBIST)*. 2013, pp. 233–242 (cit. on pp. 46, 48, 49, 83, 183, 184).

[CMG09] M. Cha, A. Mislove, and K. P. Gummadi. "A Measurement-driven Analysis of Information Propagation in the Flickr Social Network." In: *ACM International Conference on World Wide Web (WWW)*. 2009, pp. 721–730 (cit. on p. 38).

[CMM+14] X. Cheng, F. Mehrdad, X. Ma, C. Zhang, and J. Liu. "Understanding the YouTube Partners and their Data: Measurement and Analysis." In: *IEEE China Communications* 11.12 (2014), pp. 26–34 (cit. on pp. 18, 182, 183).

[Coh03] B. Cohen. "Incentives build Robustness in BitTorrent." In: *Workshop on Economics of Peer-to-Peer Systems*. Vol. 6. 2003, pp. 68–72 (cit. on p. 39).

[Con15] Conviva Inc. *How Consumers Judge their Viewing Experience - The Business Implications of Sub-par OTT Service*. 2015 (cit. on p. 118).

[Con16] Conviva Inc. *The Secret Life of Streamers: Devices, Content, Location, and Quality*. 2016 (cit. on pp. 1, 2).

[Cos+17] Y. M. Costa et al. "An Evaluation of Convolutional Neural Networks for Music Classification using Spectrograms." In: *Applied Soft Computing* 52 (2017), pp. 28–38 (cit. on p. 27).

[CSZ10] O. Chapelle, B. Scholkopf, and A. Zien. *Semi-supervised Learning*. Vol. 1. The MIT Press, 2010 (cit. on p. 23).

[CTW02] H. Che, Y. Tung, and Z. Wang. "Hierarchical Web Caching Systems: Modeling, Design and Experimental Results." In: *IEEE Journal on Selected Areas in Communications (JSAC)* 20.7 (2002), pp. 1305–1314 (cit. on pp. 15, 17, 18, 43).

[CV95] C. Cortes and V. Vapnik. "Support-vector networks." In: *Machine learning* 20.3 (1995), pp. 273–297 (cit. on p. 24).

[DA99] J. Dilley and M. Arlitt. "Improving Proxy Cache Performance: Analysis of Three Replacement Policies." In: *IEEE Internet Computing* 3.6 (1999), pp. 44–50 (cit. on p. 16).

[DC06] Y. Duan and J. Canny. "How to Construct Multicast Cryptosystems Provably Secure Against Adaptive Chosen Ciphertext Attack." In: *Cryptographers' Track at the RSA Conference*. 2006, pp. 244–261 (cit. on pp. 55, 121).

[Dee89] S. Deering. *Host Extensions for IP Multicasting*. RFC1112. 1989. URL: https://www.ietf.org/rfc/rfc1112.txt (cit. on p. 12).

[DF02] N. L. Da Fonseca and R. D. A. Façanha. "The Look-ahead-maximize-batch Batching Policy." In: *IEEE Transactions on Multimedia* 4.1 (2002), pp. 114–120 (cit. on p. 53).

[DHL+12] J. Dai, Z. Hu, B. Li, J. Liu, and B. Li. "Collaborative Hierarchical Caching with Dynamic Request Routing for Massive Content Distribution." In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2012, pp. 2444–2452 (cit. on p. 17).

[DKA+15] M. Du, M. Kihl, A. Arvidsson, C. Lagerstedt, and A. Gawler. “Analysis of Prefetching Schemes for TV-on-demand Service.” In: *International Conference on Digital Telecommunications (ICDT)*. 2015, pp. 310–315 (cit. on p. 4).

[DLL+00] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen. “Deployment Issues for the IP Multicast Service and Architecture.” In: *IEEE Network* 14.1 (2000), pp. 78–88 (cit. on pp. 5, 12, 52, 133).

[Dom12] P. Domingos. “A Few Useful Things to Know About Machine Learning.” In: *Communications of the ACM* 55.10 (2012), pp. 78–87 (cit. on p. 23).

[DRC10] R. Doverspike, K. Ramakrishnan, and C. Chase. “Structural Overview of ISP Networks.” In: *Guide to Reliable Internet Services and Applications*. Ed. by C. R. Kalmanek, S. Misra, and Y. R. Yang. Computer Communications and Networks. Springer, 2010, pp. 19–93 (cit. on pp. 2, 10).

[DSA+11] 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: *ACM SIGCOMM Computer Communication Review* 41.4 (2011), pp. 362–373 (cit. on p. 19).

[DSS94] A. Dan, D. Sitaram, and P. Shahabuddin. “Scheduling Policies for an On-demand Video Server with Batching.” In: *ACM International Conference on Multimedia (MM)*. 1994, pp. 15–23 (cit. on p. 53).

[DSS96] A. Dan, D. Sitaram, and P. Shahabuddin. “Dynamic Batching Policies for an On-Demand Video Server.” In: *Springer International Publishing, Multimedia Systems* (1996), pp. 112–121 (cit. on pp. 6, 53, 56, 58, 59).

[DTK+16] S. Dernbach, N. Taft, J. Kurose, U. Weinsberg, C. Diot, and A. Ashkan. “Cache Content-selection Policies for Streaming Video Services.” In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2016, pp. 1–9 (cit. on pp. 44, 47, 48, 50, 51).

[DZW+14] N. Do, Y. Zhao, S.-T. Wang, C.-H. Hsu, and N. Venkatasubramanian. “Optimizing Offline Access to Social Network Content on Mobile Devices.” In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2014, pp. 1950–1958 (cit. on pp. 40, 42, 64).

[EGH+11] J. Erman, A. Gerber, M. Hajiaghayi, D. Pei, S. Sen, and O. Spatscheck. “To Cache or Not to Cache: The 3G Case.” In: *IEEE Internet Computing* 15.2 (2011), pp. 27–34 (cit. on p. 43).

[EKS+96] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. “A Density-based Algorithm for Discovering Clusters in Large Spatial Databases With Noise.” In: *KDD*. Vol. 96. 34. 1996, pp. 226–231 (cit. on p. 75).

[ELT09] T. Eerola, O. Lartillot, and P. Toiviainen. “Prediction of Multidimensional Emotional Ratings in Music from Audio Using Multivariate Regression Models.” In: *International Society for Music Information Retrieval Conference (ISMIR)*. 2009 (cit. on p. 29).

[ER15] S.-E. Elayoubi and J. Roberts. “Performance and Cost Effectiveness of Caching in Mobile Access Networks.” In: *ACM Conference on Information-Centric Networking (ICN)*. 2015, pp. 79–88 (cit. on pp. 43, 44, 82).

[Eri13] Ericsson ConsumerLab. *TV and Media - Identifying the Needs of Tomorrow's Video Consumers*. Tech. rep. 2013 (cit. on p. 1).

[Eri14] Ericsson ConsumerLab. *TV and Media 2014 - Changing Consumer Needs are Creating a New Media Landscape*. Tech. rep. 2014 (cit. on p. 20).

[Eri15] Ericsson ConsumerLab. *TV and Media 2015 - The Empowered TV and Media Consumer's Influence*. Tech. rep. 2015 (cit. on p. 1).

[Eri17] Ericsson AB. *Ericsson Mobility Report*. June 2017 (cit. on p. 1).

[ES98] W. Effelsberg and R. Steinmetz. *Video Compression Techniques: An Introduction*. dpunkt-Verlag, 1998 (cit. on p. 1).

[ESH15] A. M. Elkahky, Y. Song, and X. He. "A Multi-view Deep Learning Approach or Cross Domain User Modeling in Recommendation Systems." In: *ACM International Conference on World Wide Web (WWW)*. 2015, pp. 278–288 (cit. on p. 139).

[EV98] D. L. Eager and M. K. Vernon. "Dynamic Skyscraper Broadcasts for Video-on-Demand." In: *International Workshop on Multimedia Information Systems (MIS)*. Springer. 1998, pp. 18–32 (cit. on p. 54).

[Eva14] D. Evans. *Flash! (Modern File Systems), CS4414*. 2014. URL: http://www.rust-class.org/class-17-flash.html (cit. on pp. 17, 47).

[EVZ01] D. Eager, M. Vernon, and J. Zahorjan. "Minimizing Bandwidth Requirements for on-demand Data Delivery." In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 13.5 (2001), pp. 742–757 (cit. on pp. 56, 58).

[FAK09] S. Farhad, M. M. Akbar, and M. H. Kabir. "Multicast Video-on-Demand Service in an Enterprise Network with Client-assisted Patching." In: *Multimedia Tools and Applications* 43.1 (2009), pp. 63–90 (cit. on p. 57).

[FBA11] F. Figueiredo, F. Benevenuto, and J. M. Almeida. "The Tube over Time: Characterizing Popularity Growth of Youtube Videos." In: *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*. 2011 (cit. on p. 83).

[FCL17] H. Feng, Z. Chen, and H. Liu. "Design and Optimization for VoD Services with Adaptive Multicast and Client Caching." In: *IEEE Communications Letters* 21.7 (2017), pp. 1621–1624 (cit. on p. 8).

[FHK+16] A. Frömmgen, M. Hassan, R. Kluge, M. Mousavi, M. Mühlhäuser, S. Müller, M. Schnee, M. Stein, and M. Weckesser. *Mechanism Transitions: A New Paradigm for a Highly Adaptive Internet*. 2016 (cit. on p. 32).

[FMG+13] A. Finamore, M. Mellia, Z. Gilani, K. Papagiannaki, V. Erramilli, and Y. Grunenberger. "Is there a Case for Mobile Phone Content Pre-staging?" In: *ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. 2013, pp. 321–326 (cit. on pp. 40–43).

[FPL+13] B. Frank, I. Poese, Y. Lin, G. Smaragdakis, A. Feldmann, B. Maggs, J. Rake, S. Uhlig, and R. Weber. "Pushing CDN-ISP Collaboration to the Limit." In: *ACM SIGCOMM Computer Communication Review* 43.3 (2013), pp. 34–44 (cit. on p. 19).

[FSK+18] A. Frömmgen, D. Stohr, B. Koldehofe, and A. Rizk. “Don’t Repeat Yourself: Seamless Execution and Analysis of Extensive Network Experiments.” In: *arXiv preprint arXiv:1802.03455* (2018) (cit. on p. 137).

[Fur98] B. Furht. *Handbook of Multimedia Computing*. Vol. 5. CRC Press, 1998 (cit. on p. 13).

[GAL+07] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. “YouTube Traffic Characterization: A View from the Edge.” In: *ACM SIGCOMM Internet Measurement Conference (IMC)*. 2007, pp. 15–28 (cit. on pp. 6, 18, 58, 133, 182, 183).

[GEF16] S. Grover, R. Ensafi, and N. Feamster. “A Case Study of Traffic Demand Response to Broadband Service-Plan Upgrades.” In: *International Conference on Passive and Active Network Measurement (PAM)*. Springer. 2016, pp. 124–135 (cit. on p. 2).

[GHM13] F. Guillemin, T. Houdoin, and S. Moteau. “Volatility of YouTube Content in Orange Networks and Consequences.” In: *IEEE International Conference on Communications (ICC)*. 2013, pp. 2381–2385 (cit. on pp. 6, 58, 133, 182, 183).

[GKT02] L. Gao, J. Kurose, and D. Towsley. “Efficient Schemes for Broadcasting Popular Videos.” In: *Multimedia Systems* 8.4 (2002), pp. 284–294 (cit. on p. 54).

[GLM16] M. Garetto, E. Leonardi, and V. Martina. “A Unified Approach to the Performance Analysis of Caching Systems.” In: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (ToMPECS)* 1.3 (2016), p. 12 (cit. on p. 18).

[GLZ+00] C. Griwodz, M. Liepert, M. Zink, and R. Steinmetz. “Tune to Lambda Patching.” In: *ACM SIGMETRICS Performance Evaluation Review* 27.4 (2000), pp. 20–26 (cit. on p. 57).

[GMM+13] D. Grois, D. Marpe, A. Mulayoff, B. Itzhaky, and O. Hadar. “Performance Comparison of H.265/MPEG-HEVC, VP9, and H.264/MPEG-AVC Encoders.” In: *IEEE Picture Coding Symposium (PCS)*. 2013, pp. 394–397 (cit. on p. 1).

[Gou+15] A. Gouta et al. “CPSys: A System for Mobile Video Prefetching.” In: *IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 2015, pp. 188–197 (cit. on pp. 5, 37, 40–42, 72, 80, 82, 132).

[GPN13] N. Gautam, H. Petander, and J. Noel. “A Comparison of the Cost and Energy Efficiency of Prefetching and Streaming of Mobile Video.” In: *ACM Workshop on Mobile Video (MoVid)*. 2013, pp. 7–12 (cit. on pp. 35, 36, 39, 41–43).

[Gri00] C. Griwodz. “Wide-area True Video-on-Demand by a Decentralized Cache-based Distribution Infrastructure.” PhD thesis. Technische Universität, 2000 (cit. on p. 57).

[Gri17] S. Grizzle. *Three Streaming Video Challenges We Still Face Today, and How to Solve Them*. http://StreamingMedia.com. Oct. 2017. URL: `https://goo.gl/BrZxmg` (cit. on p. 2).

[Gro+13] C. Gross et al. “EnerSim: An Energy Consumption Model for Large-scale Overlay Simulators.” In: *IEEE Conference on Local Computer Networks (LCN)*. 2013 (cit. on pp. 35, 62).

[GS15] M. Gillhofer and M. Schedl. “Iron Maiden while Jogging, Debussy for Dinner?” In: *International Conference on Multimedia Modeling (MMM)*. Springer LNCS, 2015 (cit. on pp. 27, 29, 97).

[GSD+12] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire. “Femtocaching: Wireless Video Content Delivery through Distributed Caching Helpers.” In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2012, pp. 1107–1115 (cit. on pp. 4, 5, 43, 47, 49, 51).

[GT99] L. Gao and D. Towsley. “Supplying Instantaneous Video-on-Demand Services using Controlled Multicast.” In: *IEEE International Conference on Multimedia Computing and Systems (ICMCS)*. Vol. 2. 1999, pp. 117–121 (cit. on p. 54).

[GZL+00] C. Griwodz, M. Zink, M. Liepert, G. On, and R. Steinmetz. “Multicast for Savings in Cache-based Video Distribution.” In: *Multimedia Computing and Networking*. Vol. 3969. International Society for Optics and Photonics. 2000, pp. 26–36 (cit. on p. 57).

[GZT99] L. Gao, Z.-L. Zhang, and D. Towsley. “Catching and Selective Catching: Efficient Latency Reduction Techniques for Delivering Continuous Multimedia Streams.” In: *ACM International Conference on Multimedia (MM)*. 1999, pp. 203–206 (cit. on p. 54).

[Hac15] M. Hachman. *Notebook Hard Drives are Dead: How SSDs will Dominate Mobile PC Storage by 2018*. 2015. URL: https://goo.gl/4FmLCi (cit. on pp. 45, 104).

[Hac16] S. Hacker. “Simultaneous Partial Delivery of Video-on-Demand Streams.” In: Master Thesis, Technische Universität Darmstadt, PS-D-0028. May 2016 (cit. on p. 116).

[HB96] J. Hawkinson and T. Bates. *Guidelines for Creation, Selection, and Registration of an Autonomous System (AS)*. RFC 1930. Mar. 1996. URL: https://tools.ietf.org/html/rfc1930 (cit. on p. 10).

[HBY04] M. M. Hefeeda, B. K. Bhargava, and D. K. Yau. “A Hybrid Architecture for Cost-effective on-demand Media Streaming.” In: *Elsevier Computer Networks* 44.3 (2004), pp. 353–382 (cit. on p. 57).

[HC16] J.-P. Hong and W. Choi. “User Prefix Caching for Average Playback Delay Reduction in Wireless Video Streaming.” In: *IEEE Transactions on Wireless Communications* 15.1 (2016), pp. 377–388 (cit. on p. 57).

[HC99] H. Holbrook and D. Cheriton. “IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications.” In: *ACM SIGCOMM Computer Communication Review* 29.4 (1999), pp. 65–78 (cit. on p. 12).

[HCS98] K. A. Hua, Y. Cai, and S. Sheu. “Patching: A Multicast Technique for True Video-on-Demand Services.” In: *ACM International Conference on Multimedia (MM)*. 6. 1998 (cit. on pp. 6, 53, 56–59).

[HFG+12] B. D. Higgins, J. Flinn, T. J. Giuli, B. Noble, C. Peplin, and D. Watson. "Informed Mobile Prefetching." In: *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2012, pp. 155–168 (cit. on pp. 35, 37, 43).

[HG08] Q. Huynh-Thu and M. Ghanbari. "Temporal Aspect of Perceived Quality in Mobile Video Broadcasting." In: *IEEE Transactions on Broadcasting* 54.3 (2008), pp. 641–651 (cit. on pp. 19, 21).

[HGC+15] X. Hu, J. Gong, G. Cheng, and C. Fan. "Enhancing in-network Caching by Coupling Cache Placement, Replacement and Location." In: *IEEE International Conference on Communications (ICC)*. 2015, pp. 5672–5678 (cit. on p. 18).

[HH11] G. Haßlinger and F. Hartleb. "Content Delivery and Caching from a Network Provider's Perspective." In: *Computer Networks* 55.18 (2011), pp. 3991–4006 (cit. on pp. 5, 12, 14, 43, 55, 85).

[HHK+12] B. Han, P. Hui, V. A. Kumar, M. V. Marathe, J. Shao, and A. Srinivasan. "Mobile Data Offloading through Opportunistic Communications and Social Participation." In: *IEEE Transactions on Mobile Computing* 11.5 (2012), pp. 821–834 (cit. on p. 43).

[HHS+17] T. Hoßfeld, P. E. Heegaard, L. Skorin-Kapov, and M. Varela. "No Silver Bullet: QoE Metrics, QoE Fairness, and User Diversity in the Context of QoE Management." In: *IEEE International Conference on Quality of Multimedia Experience (QoMEX)*. 2017, pp. 1–6 (cit. on p. 21).

[HK07] M. J. Halvey and M. T. Keane. "Exploring Social Dynamics in Online Media Sharing." In: *ACM International Conference on World Wide Web (WWW)*. 2007, pp. 1273–1274 (cit. on p. 18).

[HLL+07] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. "A Measurement Study of a Large-scale P2P IPTV System." In: *IEEE Transactions on Multimedia* 9.8 (2007), pp. 1672–1687 (cit. on p. 43).

[HLW+14] Y. Huang, S. Lin, H. Wu, and Y. Li. "Music Genre Classification based on Local Feature Selection using a Self-adaptive Harmony Search Algorithm." In: *Data & Knowledge Engineering* 92 (2014), pp. 60–76 (cit. on pp. 29, 94).

[HMS66] E. B. Hunt, J. Marin, and P. J. Stone. *Experiments in Induction*. 1966 (cit. on p. 24).

[HNH14] G. Hasslinger, K. Ntougias, and F. Hasslinger. "A New Class of Web Caching Strategies for Content Delivery." In: *IEEE International Telecommunications Network Strategy and Planning Symposium (Networks)*. 2014, pp. 1–7 (cit. on pp. 5, 48, 49, 51, 132).

[HQG+12] J. Huang, F. Qian, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. "A Close Examination of Performance and Power Characteristics of 4G LTE Networks." In: *ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2012 (cit. on pp. 4, 35, 60, 80, 132).

[HRJ+10] B.-J. Han, S. Rho, S. Jun, and E. Hwang. "Music Emotion Classification and Context-based Music Recommendation." In: *Springer Multimedia Tools and Applications* 47.3 (2010), pp. 433–460 (cit. on pp. 28, 29).

[HRR+17a] R. Hark, N. Richerzhagen, B. Richerzhagen, A. Rizk, and R. Steinmetz. “Towards an Adaptive Selection of Loss Estimation Techniques in Software-defined Networks.” In: *IEEE/IFIP Networking Conference (IFIP Networking)*. 2017, pp. 1–9 (cit. on p. 139).

[HRR+17b] R. Hark, A. Rizk, N. Richerzhagen, B. Richerzhagen, and R. Steinmetz. “Isolated in-band communication for distributed SDN controllers.” In: *IEEE/IFIP Networking Conference (IFIP Networking)*. 2017, pp. 1–2 (cit. on p. 121).

[HS97] K. A. Hua and S. Sheu. “Skyscraper Broadcasting: A new Broadcasting Scheme for Metropolitan Video-on-Demand Systems.” In: *ACM SIGCOMM Computer Communication Review*. Vol. 27. 4. 1997, pp. 89–100 (cit. on p. 54).

[HSB+13] T. Hoßfeld, R. Schatz, E. Biersack, and L. Plissonneau. “Internet Video Delivery in YouTube: From Traffic Measurements to Quality of Experience.” In: *Data Traffic Monitoring and Analysis*. Ed. by E. Biersack, C. Callegari, and M. Matijasevic. Vol. 7754. Lecture Notes in Computer Science. Springer, 2013, pp. 264–301 (cit. on pp. 2, 19, 20, 53).

[HSH+11] T. Hoßfeld, M. Seufert, M. Hirth, T. Zinner, P. Tran-Gia, and R. Schatz. “Quantification of YouTube QoE via Crowdsourcing.” In: *IEEE International Symposium on Multimedia (ISM)*. 2011, pp. 494–499 (cit. on p. 21).

[HSK14] T. Hoßfeld, R. Schatz, and U. R. Krieger. “QoE of YouTube Video Streaming for Current Internet Transport Protocols.” In: *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*. Vol. 8376. Lecture Notes in Computer Science. Springer, 2014, pp. 136–150 (cit. on p. 39).

[HSS+15] T. Hossfeld, M. Seufert, C. Sieber, T. Zinner, and P. Tran-Gia. “Identifying QoE Optimal Adaptation of HTTP Adaptive Streaming based on Subjective Studies.” In: *Elsevier Computer Networks* 81 (2015), pp. 320–332 (cit. on p. 22).

[Hu01] A. Hu. “Video-on-Demand Broadcasting Protocols: A Comprehensive Study.” In: *IEEE International Conference on Computer Communications (INFOCOM)*. Vol. 1. 2001, pp. 508–517 (cit. on p. 57).

[Hud17] T. Huddleston, Jr. *Netflix Has More U.S. Subscribers Than Cable TV*. Ed. by Time Inc. https://goo.gl/Hg6oP6. June 2017 (cit. on p. 1).

[Ine16] IneoQuest Technologies, Inc. *Tech's Newest Epidemic: Buffer Rage*. 2016. URL: https://goo.gl/goH39t (cit. on p. 2).

[JG13] C. Jayasundara and V. Gopalakrishnan. “Facilitating Multicast in VoD Systems by Content Pre-placement and Multistage Batching.” In: *IEEE International Conference on Communication Systems and Networks (COMSNETS)*. 2013, pp. 1–10 (cit. on p. 57).

[JKM+13] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, and M. Zhu. “B4: Experience with a Globally-deployed Software Defined WAN.” In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 3–14 (cit. on pp. 5, 52).

[JL17] S.-y. Jang and C. Y. Lee. "Batching with Reneging and AMC for VoD Streaming Service over Wireless Networks." In: *Wireless Personal Communications* 97.3 (2017), pp. 4211–4227 (cit. on p. 57).

[JLT+16] F. Jiang, Z. Liu, K. Thilakarathna, Z. Li, Y. Ji, and A. Seneviratne. "Transfetch: A Viewing Behavior Driven Video Distribution Framework in Public Transport." In: *IEEE Conference on Local Computer Networks (LCN)*. IEEE. 2016, pp. 147–155 (cit. on p. 62).

[JT98] L.-S. Juhn and L.-M. Tseng. "Fast Data Broadcasting and Receiving Scheme for Popular Video Service." In: *IEEE Transactions on Broadcasting* 44.1 (1998), pp. 100–105 (cit. on p. 57).

[KBD13] M. Kaafar, S. Berkovsky, and B. Donnet. "On the Potential of Recommendation Technologies for Efficient Content Delivery Networks." In: *ACM SIGCOMM Comput. Commun. Rev.* 43.3 (2013), pp. 74–77 (cit. on p. 47).

[KBR+15] C. Koch, N. Bui, J. Rückert, G. Fioravantti, F. Michelinakis, S. Wilk, J. Widmer, and D. Hausheer. "Media Download Optimization through Prefetching and Resource Allocation in Mobile Networks." In: *ACM Multimedia Systems Conference (MMSys)*. 2015, pp. 85–88 (cit. on pp. 7, 60, 138).

[KC10] H. J. Kim and S. G. Choi. "A study on a QoS/QoE Correlation Model for QoE Evaluation on IPTV Service." In: *IEEE International Conference on Advanced Communication Technology (ICACT)*. Vol. 2. 2010, pp. 1377–1382 (cit. on p. 20).

[KE93] D. Kotz and C. Ellis. "Practical Prefetching Techniques for Multiprocessor File Systems." In: *Distributed and Parallel Databases* 1.1 (1993), pp. 33–51 (cit. on p. 34).

[Kel13] D. Kelleher. "95.6% of Commuters in the US put Company Data at Risk over Free Public Wi-Fi." In: *Tech. Rep.* (2013) (cit. on p. 62).

[KFH17] F. Kaup, F. Fischer, and D. Hausheer. "Measuring and Predicting Cellular Network Quality on Trains." In: *IEEE International Conference on Networked Systems (NetSys)*. 2017, pp. 1–8 (cit. on p. 61).

[KH14] C. Koch and D. Hausheer. "Optimizing Mobile Prefetching by Leveraging Usage Patterns and Social Information." In: *IEEE International Conference on Network Protocols (ICNP)*. 2014, pp. 293–295 (cit. on pp. 15, 48, 55, 58, 60, 65).

[KHH17] C. Koch, S. Hacker, and D. Hausheer. "VoDCast: Efficient SDN-based Multicast for Video on Demand." In: *IEEE International Symposium on a World of Wireless and Multimedia Networks (WoWMoM)*. 2017, pp. 1–6 (cit. on pp. 59, 116).

[KHR+18] S. Kar, R. Hark, A. Rizk, and R. Steinmetz. "Towards Optimal Placement of Monitoring Units in Time-Varying Networks Under Centralized Control." In: *Springer International Conference on Measurement, Modelling and Evaluation of Computing Systems (MMB)*. 2018, pp. 99–112 (cit. on p. 137).

[Kim15] E. Kim. *Everything You Wanted to Know about the Kernel Trick (But Were Too Afraid to Ask)*. 2015. URL: http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick_blog_ekim_12_20_2017.pdf (cit. on p. 25).

[Kis17] Kissmetrics. *How Loading Time Affects Your Bottom Line*. 2017. URL: https://blog.%20kissmetrics.com/loading-time/ (cit. on pp. 4, 13).

[KKH17] C. Koch, G. Krupii, and D. Hausheer. "Proactive Caching of Music Videos based on Audio Features, Mood, and Genre." In: *ACM Multimedia Systems Conference (MMSys)*. 2017, pp. 100–111 (cit. on pp. 70, 83, 86, 88, 180).

[Kli17] K. Kliinnglills. "Video-on-Demand Optimization for Internet Service Provider." In: Master Thesis, Technische Universität Darmstadt, KOM-M-0602. July 2017 (cit. on p. 116).

[KLL+08] H. J. Kim, D. H. Lee, J. M. Lee, K. H. Lee, W. Lyu, and S. G. Choi. "The QoE Evaluation Method through the QoS-QoE Correlation Model." In: *IEEE International Conference on Networked Computing and Advanced Information Management (NCM)*. Vol. 2. 2008, pp. 719–725 (cit. on p. 20).

[KLR+17] C. Koch, B. Lins, A. Rizk, R. Steinmetz, and D. Hausheer. "vFetch: Video Prefetching using Pseudo Subscriptions and User Channel Affinity in YouTube." In: *IEEE International Conference on Network and Service Management (CNSM)*. 2017, pp. 1–9 (cit. on pp. 4, 42, 60, 132).

[KLS+18] C. Koch, M. Lode, D. Stohr, A. Rizk, and R. Steinmetz. "Collaborations on YouTube: From Unsupervised Detection to the Impact on Video and Channel Popularity." In: *arXiv preprint arXiv:1805.01887* (2018), pp. 1–28 (cit. on p. 138).

[KLW94] R. Karedla, J. S. Love, and B. G. Wherry. "Caching Strategies to Improve Disk System Performance." In: *Computer* 27.3 (1994), pp. 38–46 (cit. on p. 16).

[KMB+15] F. Kaup, F. Michelinakis, N. Bui, J. Widmer, K. Wac, and D. Hausheer. "Behind the NAT - A Measurement based Evaluation of Cellular Service Quality." In: *IEEE International Conference on Network and Service Management (CNSM)*. 2015, pp. 228–236 (cit. on p. 127).

[KMS+] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao. "Moving beyond end-to-end Path Information to Optimize CDN Performance." In: *ACM SIGCOMM Conference on Internet Measurement (IMC)*, pp. 190–201 (cit. on p. 19).

[KMS05] H.-G. Kim, N. Moreau, and T. Sikora. *MPEG-7 Audio and Beyond: Audio Content Indexing and Retrieval*. John Wiley & Sons, 2005 (cit. on p. 27).

[Kos15] J. Koschier. "Prädiktives Cachen von Videos bereitgestellt auf Sozialen Plattformen." In: Bachelor Thesis, Technische Universität Darmstadt, PS-S-0017. Nov. 2015 (cit. on p. 60).

[KPR+18] C. Koch, J. Pfannmüller, A. Rizk, D. Hausheer, and R. Steinmetz. "Category-aware Hierarchical Caching for Video-on-Demand Content on YouTube." In: *ACM Multimedia Systems Conference (MMSys)*. 2018, pp. 1–12 (cit. on pp. 7, 15, 51, 83).

[KRB+14] C. Koch, J. Rückert, N. Bui, F. Michelinakis, G. Fioravantti, J. Widmer, and D. Hausheer. “Demo: Mobile Social Prefetcher using Social and Network Information.” In: *IEEE International Workshop on Computer-Aided Modeling Analysis and Design of Communication Links and Networks (CAMAD)*. 2014, pp. 1–10 (cit. on pp. 7, 60, 76).

[KRE+15] D. Kreutz, F. M. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. “Software-Defined Networking: A Comprehensive Survey.” In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76 (cit. on pp. 5, 30, 53, 127, 129).

[KRF+17] W. R. KhudaBukhsh, A. Rizk, A. Frömmgen, and H. Koeppl. “Optimizing Stochastic Scheduling in Fork-join Queueing Models: Bounds and Applications.” In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2017, pp. 1–9 (cit. on p. 139).

[Kru16] G. Krupii. “Developing a Proactive Caching Mechanism for Music Content.” In: Master Thesis, Technische Universität Darmstadt, PS-D-0032. May 2016 (cit. on p. 83).

[KS13] S. S. Krishnan and R. K. Sitaraman. “Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-experimental Designs.” In: *IEEE/ACM Transactions on Networking* 21.6 (2013), pp. 2001–2014 (cit. on p. 21).

[KS92] J. J. Kistler and M. Satyanarayanan. “Disconnected Operation in the Coda File System.” In: *ACM Trans. Comput. Syst.* 10.1 (Feb. 1992), pp. 3–25 (cit. on p. 34).

[KWR+18] C. Koch, S. Werner, A. Rizk, and R. Steinmetz. “MIRA: Proactive Music Video Caching using ConvNet-based Classification and Multivariate Popularity Prediction.” In: *IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 2018, pp. 1–7 (cit. on pp. 5, 83, 132).

[KZG13] D. Krishnappa, M. Zink, and C. Griwodz. “What Should you Cache?: A Global Analysis on YouTube Related Video Caching.” In: *ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. 2013, pp. 31–36 (cit. on pp. 36, 43).

[KZK+12] S. Khemmarat, R. Zhou, D. K. Krishnappa, L. Gao, and M. Zink. “Watching User Generated Videos with Prefetching.” In: *Elsevier Image Communication* 27.4 (2012), pp. 343–359 (cit. on pp. 38–40, 42).

[KZS15] D. K. Krishnappa, M. Zink, and R. Sitaraman. “Optimizing the Video Transcoding Workflow in Content Delivery Networks.” In: *ACM Multimedia Systems Conference (MMSys)*. 2015 (cit. on pp. 86, 101, 119).

[LAO+14] J.-P. Laulajainen, Â. Arvidsson, T. Ojala, J. Seppänen, and M. Du. “Study of YouTube Demand Patterns in mixed Public and Campus WiFi Network.” In: *IEEE International Wireless Communications and Mobile Computing Conference (IWCMC)*. 2014 (cit. on p. 83).

[Lau+10] C. Laurier et al. “Indexing Music by Mood: Design and Integration of an Automatic Content-based Annotator.” In: *Multimedia Tools and Applications* 48.1 (2010), pp. 161–184 (cit. on p. 27).

[LBW+09] W. Liang, J. Bi, R. Wu, Z. Li, and C. Li. "On Characterizing PPStream: Measurement and Analysis of P2P IPTV under Large-scale Broadcasting." In: *IEEE Global Communications Conference (GLOBECOM)*. 2009 (cit. on p. 43).

[LC06] J. LeBrun and C.-N. Chuah. "Bluetooth Content Distribution Stations on Public Transit." In: *ACM International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking*. 2006, pp. 63–65 (cit. on p. 43).

[LCS06] N. Laoutaris, H. Che, and I. Stavrakakis. "The LCD Interconnection of LRU Caches and its Analysis." In: *Elsevier Performance Evaluation* 63.7 (2006), pp. 609–634 (cit. on p. 15).

[Led15] S. Lederer. "Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length." In: (2015). URL: https://bitmovin.com/mpeg-dash-hls-segment-length/ (cit. on pp. 21, 22).

[Lei09] T. Leighton. "Improving Performance on the Internet." In: *Communications of the ACM* 52.2 (2009), pp. 44–51 (cit. on p. 19).

[Li+13] J. Li et al. "YouTube Traffic Content Analysis in the Perspective of Clip Category and Duration." In: *IEEE International Conference on the Network of the Future (NOF)*. 2013 (cit. on p. 102).

[Lin16] B. Lins. "Textual and Content based Video Prefetching for Mobile Devices." In: Bachelor Thesis, Technische Universität Darmstadt, PS-S-0023. Dec. 2016 (cit. on p. 60).

[LLW+11] Y. Liu, Z. Liu, X. Wu, J. Wang, and C. C.-Y. Yang. "IPTV System Design: An ISP's Perspective." In: *IEEE International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. 2011, pp. 234–240 (cit. on pp. 2, 5, 12, 14, 43, 52, 133).

[LLX+12] H. Li, J. Liu, K. Xu, and S. Wen. "Understanding Video Propagation in Online Social Networks." In: *IEEE International Workshop on Quality of Service (IWQoS)*. 2012, pp. 1–9 (cit. on pp. 18, 183).

[LMS+10] C. Laurier, O. Meyers, J. Serrà, M. Blech, P. Herrera, and X. Serra. "Indexing Music by Mood: Design and Integration of an Automatic Content-based Annotator." In: *Multimedia Tools and Applications* 48.1 (2010), pp. 161–184 (cit. on pp. 28, 29, 94–96).

[Log00] B. Logan. "Mel Frequency Cepstral Coefficients for Music Modeling." In: *International Society for Music Information Retrieval Conference (ISMIR)*. 2000 (cit. on p. 27).

[LPB+15] A. Lareida, G. Petropoulos, V. Burger, M. Seufert, S. Soursos, and B. Stiller. "Augmenting Home Routers for Socially-aware Traffic Management." In: *IEEE Conference on Local Computer Networks (LCN)*. 2015, pp. 347–355 (cit. on pp. 5, 43).

[LRD+13] G. M. Lee, S. Rallapalli, W. Dong, Y.-C. Chen, L. Qiu, and Y. Zhang. "Mobile video delivery via human movement." In: *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2013 10th Annual IEEE Communications Society Conference on*. 2013, pp. 406–414 (cit. on p. 43).

[LSS04] N. Laoutaris, S. Syntila, and I. Stavrakakis. “Meta Algorithms for Hierarchical Web Caches.” In: *IEEE International Conference on Performance, Computing, and Communications (IPCCC)*. 2004, pp. 445–452 (cit. on pp. 15–17).

[LSW+12] Z. Li, H. Shen, H. Wang, G. Liu, and J. Li. “SocialTube: P2P-assisted Video Sharing in Online Social Networks.” In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2012, pp. 2886–2890 (cit. on pp. 39, 42, 43, 77, 79).

[LTE08] O. Lartillot, P. Toiviainen, and T. Eerola. “A Matlab Toolbox for Music Information Retrieval.” In: *Studies in Classification, Data Analysis, and Knowledge Organization*. Springer, 2008, pp. 261–268 (cit. on pp. 27, 95).

[LWL+13] H. Li, H. Wang, J. Liu, and K. Xu. “Video Requests from Online Social Networks: Characterization, Analysis and Generation.” In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2013 (cit. on pp. 38, 183).

[LWY+12] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian. “Optimizing Cost and Performance for Content Multihoming.” In: *ACM SIGCOMM 2012*. 2012, pp. 371–382 (cit. on p. 19).

[LWY93] A. Leff, J. L. Wolf, and P. S. Yu. “Replication Algorithms in a Remote Caching Architecture.” In: *IEEE Transactions on Parallel and Distributed Systems* 4.11 (1993), pp. 1185–1204 (cit. on p. 43).

[Lyn15] J. Lynch. *New Study Says by 2025, Half of Consumers Under 32 Won’t Pay for Cable - Resistance is futile: The 'cord-nevers' cannot be stopped*. Ed. by Adweek Inc. https://goo.gl/RnY6Yz. Oct. 2015 (cit. on p. 1).

[MAC+14] A. Mansy, M. Ammar, J. Chandrashekar, and A. Sheth. “Characterizing Client Behavior of Commercial Mobile Video Streaming Services.” In: *ACM Workshop on Mobile Video Delivery*. 2014, p. 8 (cit. on p. 36).

[MAS+17] S. Müller, O. Atan, M. van der Schaar, and A. Klein. “Context-aware Proactive Content Caching with Service Differentiation in Wireless Networks.” In: *IEEE Transactions on Wireless Communications* 16.2 (2017), pp. 1024–1036 (cit. on p. 86).

[MCC+13] T. Mikolov, K. Chen, G. Corrado, and J. Dean. “Efficient Estimation of Word Representations in Vector Space.” In: *arXiv preprint arXiv:1301.3781* (2013) (cit. on p. 139).

[MGL14] V. Martina, M. Garetto, and E. Leonardi. “A Unified Approach to the Performance Analysis of Caching Systems.” In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2014 (cit. on p. 18).

[MGP+18] L. Maggi, L. Gkatzikis, G. Paschos, and J. Leguay. “Adapting Caching to Audience Retention Rate.” In: *Computer Communications* 116 (2018), pp. 159–171 (cit. on p. 15).

[MH10] A. J. Mashhadi and P. Hui. “Proactive Caching for Hybrid Urban Mobile Networks.” In: *University College London, Tech. Rep* (2010). URL: https://goo.gl/rtRGGy (cit. on p. 43).

[Mil16] K. Miller. “Adaptation Algorithms for HTTP-based Video Streaming.” PhD thesis. 2016 (cit. on p. 22).

[MM04] N. Megiddo and D. S. Modha. “Outperforming LRU with an Adaptive Replacement Cache Algorithm.” In: *IEEE Computer* 37.4 (2004), pp. 58–65 (cit. on pp. 16, 17, 47, 84).

[MMF+05] D. McEnnis, C. McKay, I. Fujinaga, and P. Depalle. “jAudio: An Feature Extraction Library.” In: *International Society for Music Information Retrieval Conference (ISMIR)*. 2005, pp. 600–603 (cit. on p. 27).

[MS02] H. Ma and K. G. Shin. “Multicast Video-on-Demand Services.” In: *ACM SIGCOMM Computer Communication Review* 32.1 (2002), pp. 31–43 (cit. on p. 57).

[MS15] B. M. Maggs and R. K. Sitaraman. “Algorithmic Nuggets in Content Delivery.” In: *ACM SIGCOMM Computer Communication Review* 45.3 (2015), pp. 52–66 (cit. on pp. 4, 13, 14, 16).

[Ng09] A. Ng. *CS229: Machine Learning, Autumn 2009*. 2009. URL: http://robotics.stanford.edu/~ang/courses.html (cit. on p. 23).

[NKK+11] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. “Multimodal Deep Learning.” In: *International Conference on Machine Learning (ICML)*. 2011, pp. 689–696 (cit. on p. 139).

[NLB12] B. Niven-Jenkins, F. Le Faucheur, and N. Bitar. “Content Distribution Network Interconnection (CDNI) Problem Statement.” In: *RFC 6707* (2012) (cit. on p. 14).

[NLH15] L. Nobach, Y. Le Louédec, and D. Hausheer. “Evaluating Device-to-Device Content Delivery Potential on a Mobile ISP’s Dataset.” In: *IEEE International Conference on Network and Service Management (CNSM)*. 2015, pp. 301–309 (cit. on p. 43).

[NN08] A. J. Nicholson and B. D. Noble. “BreadCrumbs: Forecasting Mobile Connectivity.” In: *ACM International Conference on Mobile Computing and Networking (MobiCom)*. 2008, pp. 46–57 (cit. on p. 37).

[Nor14] W. B. Norton. *The Internet Peering Playbook: Connecting to the Core of the Internet*. 2nd ed. DrPeering Press, 2014 (cit. on p. 12).

[NSS10] E. Nygren, R. Sitaraman, and J. Sun. “The Akamai Network: A Platform for High-performance Internet Applications.” In: *ACM SIGOPS Operating Systems Review* 44.3 (2010), pp. 2–19 (cit. on p. 19).

[ONF14b] *SDN Architecture*. Tech. rep. ONF TR-502. Open Networking Foundation, 2014. URL: https://goo.gl/ubM6U6 (cit. on p. 119).

[Ope09] Open Networking Foundation. *OpenFlow Switch Specification - Version 1.0.0*. Dec. 2009. URL: https://goo.gl/iPzGp1 (cit. on p. 32).

[Ope13] Open Networking Foundation. *Wireless and Mobile Working Group (WMWG)*. 2013. URL: https://goo.gl/6eozk9 (cit. on p. 126).

[Ope15] Open Networking Foundation. *OpenFlow Switch Specification - Version 1.5.1*. Mar. 2015. URL: https://goo.gl/7UELBj (cit. on pp. 31, 32).

[PB03] S. Podlipnig and L. Böszörmenyi. "A Survey of Web Cache Replacement Strategies." In: *ACM Computing Surveys (CSUR)* 35.4 (2003), pp. 374–398 (cit. on p. 43).

[PCO16] C. Pleşca, V. Charvillat, and W. T. Ooi. "Multimedia Prefetching with Optimal Markovian Policies." In: *Elsevier Journal of Network and Computer Applications* 69 (2016), pp. 40–53 (cit. on pp. 39, 40, 42).

[Pfa17] J. Pfannmüller. "Comparison of Caching Strategies for Different Content Types." In: Bachelor Thesis, Technische Universität Darmstadt, KOM-B-0600. June 2017 (cit. on p. 83).

[PHZ12] A. Pathak, Y. C. Hu, and M. Zhang. "Where is the Energy Spent Inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof." In: *ACM European Conference on Computer Systems (EuroSys)*. New York, NY, USA, 2012, pp. 29–42 (cit. on p. 35).

[Pin+13] H. Pinto et al. "Using Early View Patterns to Predict the Popularity of YouTube Videos." In: *ACM International Conference on Web Search and Data Mining*. 2013, pp. 365–374 (cit. on p. 49).

[PKW+17] M. Pfannemüller, C. Krupitzer, M. Weckesser, and C. Becker. "A Dynamic Software Product Line Approach for Adaptation Planning in Autonomic Computing Systems." In: *IEEE International Conference on Autonomic Computing (ICAC)*. 2017, pp. 247–254 (cit. on p. 139).

[PM00] D. Pelleg and A. Moore. "X-means: Extending K-means with Efficient Estimation of the Number of Clusters." In: *International Conference on Machine Learning (ICML)*. Morgan Kaufmann, 2000, pp. 727–734 (cit. on p. 26).

[PPW+15] T. Paul, D. Puscher, S. Wilk, and T. Strufe. "Systematic, Large-scale Analysis on the Feasibility of Media Prefetching in Online Social Networks." In: *IEEE Consumer Communications and Networking Conference (CCNC)*. 2015, pp. 755–760 (cit. on pp. 39, 40).

[QD06] Y. Qi and M. Dai. "The Effect of Frame Freezing and Frame Skipping on Video Quality." In: *IEEE International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP)*. 2006, pp. 423–426 (cit. on p. 19).

[QQH+12] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. "Web Caching on Smartphones: Ideal vs. Reality." In: *ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2012, pp. 127–140 (cit. on p. 43).

[QWB+09] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. "Cutting the Electric Bill for Internet-scale Systems." In: *ACM SIGCOMM Computer Communication Review*. Vol. 39. 4. 2009, pp. 123–134 (cit. on p. 19).

[QWG+10] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. "TOP: Tail Optimization Protocol For Cellular Radio Resource Allocation." In: *IEEE International Conference on Network Protocols (ICNP)*. Oct. 2010, pp. 285–294 (cit. on p. 35).

[RBC+11] T. Rodrigues, F. Benevenuto, M. Cha, K. Gummadi, and V. Almeida. "On Word-of-mouth Based Discovery of the Web." In: *ACM SIGCOMM Internet Measurement Conference (IMC)*. New York, NY, USA, 2011, pp. 381–396 (cit. on p. 38).

[RBH+16] J. Rückert, J. Blendin, R. Hark, and D. Hausheer. "Flexible, Efficient, and Scalable Software-Defined Over-the-Top Multicast for ISP Environments With DynSdm." In: *IEEE Transactions on Network and Service Management* 13.4 (Dec. 2016), pp. 754–767 (cit. on pp. 5, 6, 55, 58, 59, 139).

[RCW16] F. M. Ramos, J. Crowcroft, and I. H. White. "Blending Photons with Electrons to Reduce the Energy Footprint of IPTV Networks." In: *IEEE/IFIP Networking Conference (Networking)*. 2016, pp. 288–296 (cit. on p. 12).

[RDH+13] B. A. Ramanan, L. M. Drabeck, M. Haner, N. Nithi, T. E. Klein, and C. Sawkar. "Cacheability Analysis of HTTP Traffic in an Operational LTE Network." In: *IEEE Wireless Telecommunications Symposium (WTS)*. 2013, pp. 1–8 (cit. on pp. 4, 44).

[Rec08] Recommendation ITU-T P. 10/G. 100. "Vocabulary for Performance and Quality of Service, Amendment 2: New Definitions for Inclusion." In: (2008) (cit. on p. 20).

[RHH09] S. Rho, B.-j. Han, and E. Hwang. "SVR-based Music Mood Classification and Context-based Music Recommendation." In: *ACM International Conference on Multimedia (MM)*. 2009 (cit. on pp. 28, 29, 96).

[Ric17] B. Richerzhagen. "Mechanism Transitions in Publish/Subscribe Systems-Adaptive Event Brokering for Location-based Mobile Social Applications." PhD thesis. Technische Universität, 2017 (cit. on p. 32).

[RRG01] S. Ramesh, I. Rhee, and K. Guo. "Multicast with Cache (Mcache): An Adaptive Zero-delay Video-on-Demand Service." In: *IEEE Transactions on Circuits and Systems for Video Technology* 11.3 (2001), pp. 440–456 (cit. on p. 57).

[RS13] J. Roberts and N. Sbihi. "Exploring the Memory-Bandwidth Tradeoff in an Information-centric Network." In: *IEEE International Teletraffic Congress (ITC)*. 2013, pp. 1–9 (cit. on pp. 14, 43).

[RSB99] P. Rodriguez, C. Spanner, and E. W. Biersack. "Web Caching Architectures: Hierarchical and Distributed Caching." In: *International Workshop on Web Caching and Content Distribution (WCW)*. Vol. 99. 1999 (cit. on p. 17).

[Rüc16] J. Rückert. *Large-scale Live Video Streaming Over the Internet-Efficient and Flexible Content Delivery Using Network and Application-Layer Mechanisms*. Verlag Dr. Hut, 2016 (cit. on pp. 3, 5, 52, 133).

[Rüc17] T. Rückelt. "Connecting Vehicles to the Internet-Strategic Data Transmission for Mobile Nodes using Heterogeneous Wireless Networks." PhD thesis. Technische Universität, 2017 (cit. on p. 62).

[RW13] M. S. Rahman and A. B. Wagner. "Multicasting for Wireless Video-on-Demand." In: *IEEE Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 2013, pp. 690–697 (cit. on p. 57).

[RW16] M. S. Rahman and A. B. Wagner. “A Downlink Scheduler for Multicasting Wireless Video-on-Demand.” In: *IEEE Transactions on Mobile Computing* 15.12 (2016), pp. 2921–2938 (cit. on p. 57).

[RZS17] A. Rizk, M. Zink, and R. Sitaraman. “Model-based Design and Analysis of Cache Hierarchies.” 2017 (cit. on p. 18).

[San15a] Sandvine. *Global Internet Phenomena Report - Asia Pacific and Europe*. 2015 (cit. on p. 1).

[San15b] Sandvine. *Global Internet Phenomena Report - Latin America and North America*. 2015 (cit. on p. 1).

[Sar12] H. Sarkissian. “The Business Case for Caching in 4G LTE Networks.” In: *LSI-Wireless Technical Report* (2012) (cit. on p. 43).

[SBE+16] J. Summers, T. Brecht, D. Eager, and A. Gutarin. “Characterizing the Workload of a Netflix Streaming Video Server.” In: *IEEE International Symposium on Workload Characterization (IISWC)*. 2016, pp. 100–111 (cit. on pp. 5, 49, 50).

[SBH13] M. Seufert, V. Burger, and T. Hoßfeld. “HORST-Home Router Sharing based on Trust.” In: *IEEE International Conference on Network and Service Management (CNSM)*. 2013, pp. 402–405 (cit. on pp. 5, 43).

[SBI+11] N. Sharma, S. K. Barker, D. E. Irwin, and P. J. Shenoy. “Blink: Managing Server Clusters on Intermittent Power.” In: *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2011, pp. 185–198 (cit. on p. 110).

[SBI14] M. Schedl, G. Breitschopf, and B. Ionescu. “Mobile Music Genius: Reggae at the Beach, Metal on a Friday Night?” In: *ACM International Conference on Multimedia Retrieval (ICMR)*. 2014 (cit. on p. 94).

[SC08] I. Steinwart and A. Christmann. *Support Vector Machines*. Springer Science & Business Media, 2008 (cit. on p. 24).

[SC17] H. Song and W. Choi. “Multicast Transmission for Asynchronous Data Requests.” In: *IEEE Transactions on Vehicular Technology* (2017) (cit. on p. 57).

[SCF+03] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. “Real-time Transport Protocol.” In: *RFC1899* (2003) (cit. on p. 21).

[SDP12] Y. Song, S. Dixon, and M. Pearce. “Evaluation of Musical Features for Emotion Classification.” In: *International Society for Music Information Retrieval Conference (ISMIR)*. 2012 (cit. on pp. 28, 29, 95).

[SES+15] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hobfeld, and P. Tran-Gia. “A Survey on Quality of Experience of HTTP Adaptive Streaming.” In: *IEEE Communications Surveys & Tutorials* 17.1 (2015), pp. 469–492 (cit. on pp. 20, 21).

[SFF+16] D. Stohr, A. Frömmgen, J. Fornoff, M. Zink, A. Buchmann, and W. Effelsberg. “QoE Analysis of DASH Cross-Layer Dependencies by Extensive Network Emulation.” In: *ACM SIGCOMM Workshop on QoE-based Analysis and Management of Data Communication Networks (Internet-QoE)*. 2016, pp. 25–30 (cit. on pp. 2, 13, 20, 45).

[SFR+17] D. Stohr, A. Frömmgen, A. Rizk, M. Zink, R. Steinmetz, and W. Effelsberg. “Where are the Sweet Spots?: A Systematic Approach to Reproducible DASH Player Comparisons.” In: *ACM Multimedia Systems Conference (MMSys)*. 2017, pp. 1113–1121 (cit. on p. 138).

[SH00] P. Sarkar and J. H. Hartman. “Hint-based Cooperative Caching.” In: *ACM Transactions on Computer Systems (TOCS)* 18.4 (2000), pp. 387–419 (cit. on p. 18).

[SHK+15] R. Steinmetz, M. Holloway, B. Koldehofe, B. Richerzhagen, and N. Richerzhagen. “Towards Future Internet Communications–Role of Scalable Adaptive Mechanisms.” In: *ACADEMIA EUROPAEA* (2015), p. 59 (cit. on p. 32).

[Sit13] R. Sitaraman. “Network Performance: Does It Really Matter to Users and by How Much?” In: *IEEE International Conference on Communication Systems and Networks (COMSNETS)*. 2013 (cit. on pp. 19, 21).

[SKL+14] R. K. Sitaraman, M. Kasbekar, W. Lichtenstein, and M. Jain. “Overlay Networks: An Akamai Perspective.” In: *Advanced Content Delivery, Streaming, and Cloud Services* 51.4 (2014), pp. 305–328 (cit. on pp. 4, 13, 17).

[SKL16] M. Z. Shafiq, A. R. Khakpour, and A. X. Liu. “Characterizing Caching Workload of a Large Commercial Content Delivery Network.” In: *IEEE International Conference on Computer Communications (INFOCOM)*. 2016, pp. 1–9 (cit. on pp. 46, 47, 49, 51, 83).

[SL91] S. R. Safavian and D. Landgrebe. “A Survey of Decision Tree Classifier Methodology.” In: *IEEE Transactions on Systems, Man, and Cybernetics* 21.3 (1991), pp. 660–674 (cit. on p. 24).

[SME17] Y. Sani, A. Mauthe, and C. Edwards. “On the Trajectory of Video Quality Transition in HTTP Adaptive Video Streaming.” In: *Springer Multimedia Systems* (2017), pp. 1–14 (cit. on p. 20).

[Smi82] A. J. Smith. “Cache Memories.” In: *ACM Comput. Surv.* 14.3 (Sept. 1982), pp. 473–530 (cit. on p. 34).

[SMM+11] S. Scellato, C. Mascolo, M. Musolesi, and J. Crowcroft. “Track Globally, Deliver Locally: Improving Content Delivery Networks by Tracking Geographic Social Cascades.” In: *ACM International Conference on World Wide Web (WWW)*. 2011, pp. 457–466 (cit. on p. 38).

[SRM+09] E. Sun, I. Rosenn, C. Marlow, and T. M. Lento. “Gesundheit! Modeling Contagion through Facebook News Feed.” In: *International AAAI Conference on Web and Social Media (ICWSM)*. 2009 (cit. on p. 38).

[SS12] N. Srivastava and R. Salakhutdinov. “Learning Representations for Multimodal Data with Deep Belief Nets.” In: *International Conference on Machine Learning (ICML)*. Vol. 79. 2012 (cit. on p. 139).

[SSF08] M. Saxena, U. Sharan, and S. Fahmy. “Analyzing Video Services in Web 2.0: A Global Perspective.” In: *ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. 2008, pp. 39–44 (cit. on p. 34).

[SSX+13] S. Sedhain, S. Sanner, L. Xie, R. Kidd, K.-N. Tran, and P. Christen. "Social Affinity Filtering: Recommendation through Fine-grained Analysis of User Interactions and Activities." In: *ACM Conference on Online Social Networks (COSN)*. 2013, pp. 51–62 (cit. on pp. 38, 40).

[Ste12] R. Steinmetz. *Multimedia: Computing Communications & Applications*. Pearson Education India, 2012 (cit. on pp. 2, 14).

[Ste92] J. Steuer. "Defining Virtual Reality: Dimensions Determining Telepresence." In: *Journal of communication* 42.4 (1992), pp. 73–93 (cit. on p. 2).

[Ste96] R. Steinmetz. "Human Perception of Jitter and Media Synchronization." In: *IEEE Journal on Selected Areas in Communications* 14.1 (1996), pp. 61–72 (cit. on pp. 13, 20).

[Stu13] B. L. Sturm. "The GTZAN Dataset: Its Contents, its Faults, their Effects on Evaluation, and its Future Use." In: *arXiv preprint arXiv:1306.1461* (2013) (cit. on p. 27).

[SW05] R. Steinmetz and K. Wehrle. "What Is This "Peer-to-Peer" About?" In: *Peer-to-Peer Systems and Applications*. Springer, 2005, pp. 9–16 (cit. on p. 39).

[SW97] R. Steinmetz and L. Wolf. "Quality of Service: Where are we?" In: *Building QoS into Distributed Systems*. Springer, 1997, pp. 210–221 (cit. on p. 20).

[TC02] G. Tzanetakis and P. Cook. "Musical Genre Classification of Audio Signals." In: *IEEE Transactions on Speech and Audio Processing* 10.5 (July 2002), pp. 293–302 (cit. on p. 27).

[TCZ+13] F. P. Tso, L. Cui, L. Zhang, W. Jia, D. Yao, J. Teng, and D. Xuan. "DragonNet: A Robust Mobile Internet Service System for Long-distance Trains." In: *IEEE Transactions on Mobile Computing* 12.11 (2013), pp. 2206–2218 (cit. on p. 43).

[Tha90] R. Thayer. *The Biopsychology of Mood and Arousal*. Oxford University Press, 1990 (cit. on pp. 27, 94).

[THD04] M. A. Tantaoui, K. A. Hua, and T. T. Do. "BroadCatch: A Periodic Broadcast Technique for Heterogeneous Video-on-Demand." In: *IEEE Transactions on Broadcasting* 50.3 (2004), pp. 289–301 (cit. on p. 57).

[TMB+13] M. Taghizadeh, K. Micinski, S. Biswas, C. Ofria, and E. Torng. "Distributed Cooperative Caching in Social Wireless Networks." In: *IEEE Transactions on Mobile Computing* 12.6 (2013), pp. 1037–1053 (cit. on p. 43).

[TMF+14] S. Tombaz, P. Monti, F. Farias, M. Fiorani, L. Wosinska, and J. Zander. "Is Backhaul becoming a Bottleneck for Green Wireless Access Networks?" In: *IEEE International Conference on Communications (ICC)*. 2014, pp. 4029–4035 (cit. on p. 45).

[TPS+16] A. G. Tasiopoulos, I. Psaras, V. Sourlas, and G. Pavlou. "Tube Streaming: Modelling Collaborative Media Streaming in Urban Railway Networks." In: *IEEE/IFIP Networking Conference (IFIP Networking)*. 2016, pp. 359–367 (cit. on p. 43).

[TTK+08] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. P. Vlahavas. "Multi-Label Classification of Music into Emotions." In: *International Society for Music Information Retrieval Conference (ISMIR)*. 2008 (cit. on pp. 28, 29, 95).

[TV97] A. K. Tsiolis and M. K. Vernon. "Group-guaranteed Channel Capacity in Multimedia Storage Servers." In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 25. 1. 1997, pp. 285–297 (cit. on p. 53).

[VDS13] A. Van den Oord, S. Dieleman, and B. Schrauwen. "Deep Content-based Music Recommendation." In: *Advances in Neural Information Processing Systems (NIPS)*. 2013, pp. 2643–2651 (cit. on p. 139).

[VI95] S. Viswanathan and T. Imielinski. "Pyramid Broadcasting for Video-on-Demand Service." In: *Multimedia Computing and Networking*. Vol. 2417. International Society for Optics and Photonics. 1995, pp. 66–78 (cit. on p. 54).

[VP03] A. Vakali and G. Pallis. "Content Delivery Networks: Status and Trends." In: *IEEE Internet Computing* 7.6 (2003), pp. 68–74 (cit. on pp. 13, 14).

[Wan+13] Z. Wang et al. "The Analysis and Comparison of Vital Acoustic Features in Content-Based Classification of Music Genre." In: *IEEE International Conference on Information Technology and Applications (ITA)*. 2013, pp. 404–408 (cit. on p. 27).

[Wer17] S. Werner. "Investigating Machine Learning Methods for Proactive Network Caching of Music Content." In: Bachelor Thesis, Technische Universität Darmstadt, KOM-B-0593. Oct. 2017 (cit. on p. 83).

[Wic17] M. Wichtlhuber. *On Collaborative Mechanisms for Content Distribution*. Verlag Dr. Hut, 2017 (cit. on p. 119).

[Wik16] Wikipedia, The Free Encyclopedia. *Transition (computer science)*. 2016. URL: `https://en.wikipedia.org/wiki/Transition_(computer_science)` (cit. on p. 32).

[WKB+17] M. Wichtlhuber, J. Kessler, S. Bücker, I. Poese, J. Blendin, C. Koch, and D. Hausheer. "SoDA: Enabling CDN-ISP Collaboration with Software Defined Anycast." In: *IFIP International Conference on Networking (NETWORKING)*. 2017, pp. 1–9 (cit. on p. 119).

[WRT+15] S. Wilk, J. Rückert, T. Thräm, C. Koch, W. Effelsberg, and D. Hausheer. "The Potential of Social-aware Multimedia Prefetching on Mobile Devices." In: *IEEE International Conference on Networked Systems (NetSys)*. 2015, pp. 1–5 (cit. on pp. 3, 4, 39, 40, 64, 132).

[WSS+16] S. Wilk, D. Schreiber, D. Stohr, and W. Effelsberg. "On the Effectiveness of Video Prefetching Relying on Recommender Systems for Mobile Devices." In: *IEEE Annual Consumer Communications & Networking Conference (CCNC)*. 2016, pp. 429–434 (cit. on pp. 3, 4, 64, 79, 132, 135).

[WSY11] Z. Wang, L. Sun, and S. Yang. "Prefetching Strategy in Peer-Assisted Social Video Streaming." In: *ACM Conference on Multimedia (MM)*. 2011, pp. 1233–1236 (cit. on pp. 38–40, 42).

[YLC06] Y.-H. Yang, C.-C. Liu, and H. H. Chen. “Music Emotion Classification: A Fuzzy Approach.” In: *ACM International Conference on Multimedia (MM)*. 2006 (cit. on pp. 28, 29, 94).

[YLS+08] Y.-H. Yang, Y.-C. Lin, Y.-F. Su, and H. H. Chen. “A Regression Approach to Music Emotion Recognition.” In: *IEEE Transactions on Audio, Speech, and Language Processing* 16.2 (2008), pp. 448–457 (cit. on pp. 28, 29).

[YLZ+09] H. Yin, X. Liu, T. Zhan, V. Sekar, and F. Qiu. “Design and Deployment of a Hybrid CDN-P2P System for Live Video Streaming: Experiences with LiveSky.” In: *ACM International Conference on Multimedia (MM)*. 2009 (cit. on p. 43).

[YWX17] J. Yuan, X. Wang, and L. Xiao. “Hybrid Video Transmission Scheme for Minimizing Maximum Waiting Time in Video-on-Demand (VOD) System.” In: *IEEE International Conference on Multimedia and Image Processing (ICMIP)*. 2017, pp. 225–229 (cit. on p. 57).

[YYR+15] J. Yang, E. Yang, Y. Ran, and S. Chen. “SDM^2 Cast An OpenFlow-Based, Software-Defined Scalable Multimedia Multicast Streaming Framework.” In: *IEEE Internet Computing* 19.4 (2015), pp. 36–44 (cit. on pp. 5, 6, 56, 58, 59).

[ZAC+13] M. Zhao, P. Aditya, A. Chen, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, B. Wishon, and M. Ponec. “Peer-Assisted Content Distribution in Akamai NetSession.” In: *ACM SIGCOMM Internet Measurement Conference (IMC)*. 2013 (cit. on p. 35).

[ZDW+13] Y. Zhao, N. Do, S.-T. Wang, C.-H. Hsu, and N. Venkatasubramanian. “O^{2}SM: Enabling Efficient Offline Access to Online Social Media and Social Networks.” In: *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. 2013, pp. 445–465 (cit. on pp. 38, 40, 42).

[ZHA+10] T. Zinner, O. Hohlfeld, O. Abboud, and T. Hoßfeld. “Impact of Frame Rate and Resolution on Objective QoE Metrics.” In: *IEEE International Conference on Quality of Multimedia Experience (QoMEX)*. 2010 (cit. on p. 21).

[Zha14] S. Zhang. *Optimizing Mobile Backhaul: Breaking the 4G Bottleneck*. 2014. URL: https://www.telecomasia.net/pdf/Huawei_OptimizingMobileBackhaul-0902.pdf (cit. on p. 45).

[ZLL13] G. Zhang, Y. Li, and T. Lin. *Caching in Information Centric Networking: A Survey*. Vol. 57. 16. Elsevier Computer Networks, 2013, pp. 3128–3141 (cit. on pp. 15, 43).

[ZSG+09] M. Zink, K. Suh, Y. Gu, and J. Kurose. “Characteristics of YouTube Network Traffic at a Campus Network–Measurements, Models, and Implications.” In: *Elsevier Computer Networks* 53.4 (2009), pp. 501–514 (cit. on p. 18).

[ZTQ+10] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. “Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones.” In: *IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 2010, pp. 105–114 (cit. on p. 35).

[ZW17] C. Zhan and Z. Wen. “Content Cache Placement for Scalable Video in Heterogeneous Wireless Network.” In: *IEEE Communications Letters* 21.12 (2017), pp. 2714–2717 (cit. on p. 43).

All web pages cited in this thesis have been checked in October 2018. However, due to the dynamic nature of the World Wide Web, the long-term availability of websites cannot be guaranteed.

LIST OF FIGURES

LIST OF TABLES

LIST OF DEFINITIONS

LIST OF ACRONYMS

ABR	Adaptive Bitrate Streaming
ANN	Artificial Neural Network
API	Application Programming Interface
ARC	Adaptive Replacement Cache
ARIMA	Autoregressive Integrated Moving Average
ASes	Autonomous Systems
ASN	Autonomous System Number
BG	Border Gateway
BHR	Byte Hit Rate
BNG	Broadband Network Gateway
CAPEX	Capital Expenditure
CAGR	Cumulative Average Growth Rate
CAM	Content-Addressable Memory
CBR	Constant Bitrate
CCDF	Complementary Cumulative Distribution Function
CDN	Content Delivery Network
CF	Collaborative Filtering
CHR	Cache Hit Rate
DASH	Dynamic Adaptive Streaming over HTTP
DBSCAN	Density-based Spatial Clustering of Applications with Noise
DNN	Deep Neural Network
DSAS	Division Size Adaptation Strategy
DSL	Digital Subscriber Line
EDF	Earliest Deadline First
EGP	Exterior Gateway Protocol
EPC	Evolved Packet Core
FIFO	First In - First Out
GGC	Google Global Cache
HAS	HTTP Adaptive Streaming
HDD	Hard Disk Drive

HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICN	Information-centric Networking
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IPTV	IP Television
IRT	Inter-Request Time
ISP	Internet Service Provider
LCE	Leave Copy Everywhere
LCD	Leave a Copy Down
LER	Label Edge Router
LFU	Least Frequently Used
LFUDA	LFU with Dynamic Aging
LGL	Largest Ghost List
LIFO	Last In - First Out
LRU	Least Recently Used
LTE	Long Term Evolution
LSR	Label Switching Router
MAN	Metropolitan Area Network
MCD	Move Copy Down
MFCC	Mel Frequency Cepstral Coefficients
MMS	Microsoft Media Server Protocol
MPEG	Moving Picture Experts Group
MPLS	Multiprotocol Label Switching
NLP	Natural Language Processing
OLS	Ordinary Least Squares
OSN	Online Social Network
OTT	Over-the-Top
P2P	Peer-to-Peer
PDN	Packet Data Network
PGW	PDN Gateway
PPPoE	Point-to-Point Protocol over Ethernet
QoE	Quality of Experience

QoS	Quality of Service
RMSE	Root Mean Square Error
REST	Representational State Transfer
RLGL	Relative Largest Ghost List
RSGL	Relative Smallest Ghost List
RTP	Real-time Transport Protocol
RTT	Round-Trip Time
SDM	Software-defined Multicast
SDN	Software-defined Networking
SGL	Smallest Ghost List
SGW	Serving Gateway
SLRU	Segmented Least Recently Used
SRAM	Static Random-Access Memory
SSD	Solid-State-Drive
SVC	Scalable Video Coding
SVM	Support Vector Machine
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
TTL	Time To Live
UDP	User Datagram Protocol
UGC	User-generated Content
VoD	Video-on-Demand

A APPENDIX

The appendix provides further information, analysis, and evaluation results on the three contributions of this thesis.

A.1 DETAILS OF THE VIDEO ORIGIN PRESENTED IN THE PREFETCHING USER STUDY

This section provides complementary information to the request sources of distinct users as presented in the cope of the vFetch user study (cf. Section 4.3.1). Therefore, we present Figure 56 which is depicted in a similar style as Figure 21. Additionally, we can see video sources that we consider as less useful for a mobile long-term prefetching mechanism such as vFetch. However, it is interesting to see that videos coming from the *related video list* of an already watched video are responsible for around 10% of all user requests, depending on the individual user. Furthermore, we observe that videos coming from playlists care not visible and, hence unlikely to serve as a valuable source for prefetching candidates. For two users, we observe a large share of videos to come from the watch later list. However, they mostly origin from subscribed channels and are, thus considered in vFetch's evaluation. Overall, subscriptions and related videos are found to be responsible for most of the users' video requests. Still, for most users, about 60% of the videos come from unknown sources as discussed in Section 4.3.1.

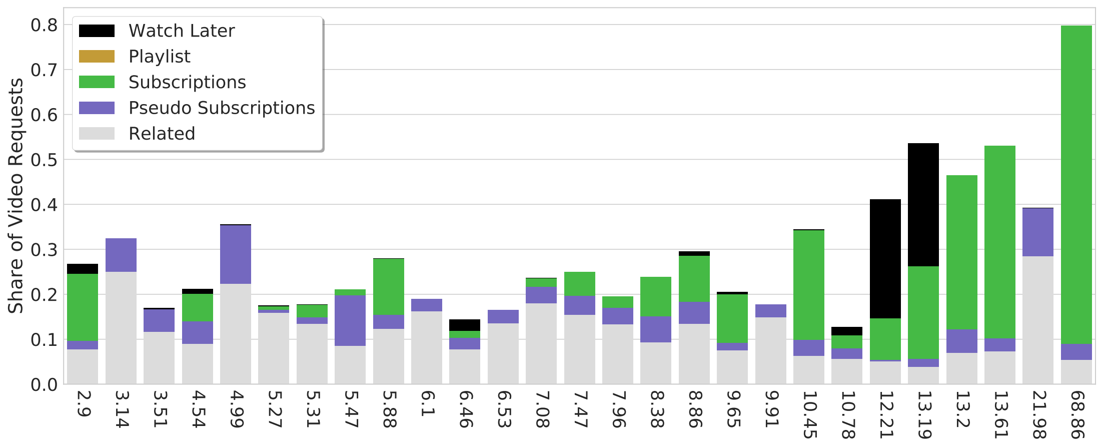


Figure 56: Watched subscriptions(%) by YouTube video category

A.2 USER REQUEST TIME AFFINITY CONSIDERING DAYTIME

In this section, we provide more insights into the user's time affinity concerning multiple categories. In contrast to Figure 24 presented in Section 4.3.1.4, we depict each of the 27 participants. Note that not every participant of the user study presented in Section 4.3 watched videos from all categories.

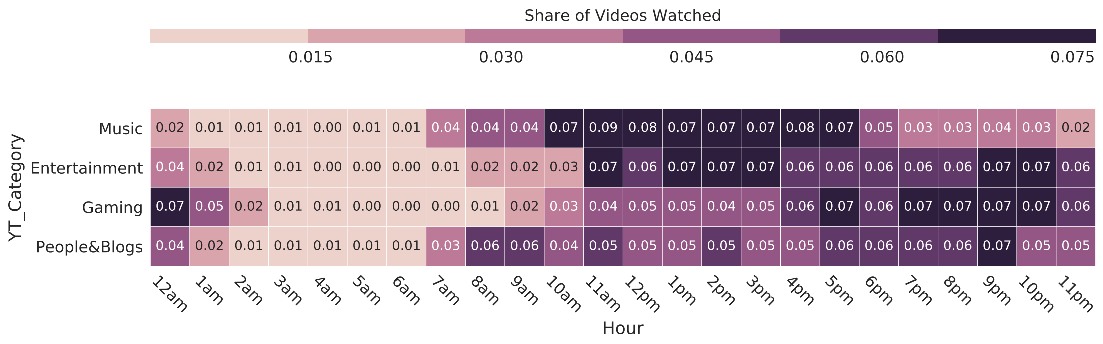


Figure 57: Share of watched videos per hour of the day and the four most popular YouTube video categories

Figure 57 shows the aggregated distribution of videos watched over the course of the day. We limit our analysis to the four most popular YouTube categories for our participants, i.e., Music, Entertainment, Gaming, and People&Blogs. We see that Music videos are requested more during noon compared to Entertainment which is almost constantly popular at noon until the late evening. For our participants, Gaming videos are requested late, mostly in the evening hours and stay popular until midnight. The category Poeple&Blogs is popular all over the day, starting at 7 am and ending at midnight which a peak at 9 pm. Note that the heat maps which we show in this section are robustly colored to make differences more evident. Consequently, high values are aggregated to a common color, i.e., every value larger than 0.075 is colored in the same dark purple. The largest popularity values observed in this figure range about 8% of the overall watched videos per hour of the day and are, thus quite small.

In addition to the previous figure, we depict all users on an individual row and aggregate all categories in Figure 58. We can see that there are clear hotspots that represent up to 20% of the overall watched videos. Though, most of the hours of the day when users watch videos seem to belong together or to two clusters, i.e., the morning and the evening hours. However, some users watch videos during the entire day without a clear pattern.

In a further analysis, we depict the individual request shares per user, concerning the four most popular YouTube categories among the user study participants, individually. Figure 59 depicts the user's video watch events for the category *Music*. Similarly, Figure 60 exhibits *Entertainment*, Figure 61 shows *People&Blogs*, and Figure 62 shows the watch events for the YouTube category *Gaming*. Note that the heat color indicates the share of videos watched per category, which usually results in a quite low number of total views.

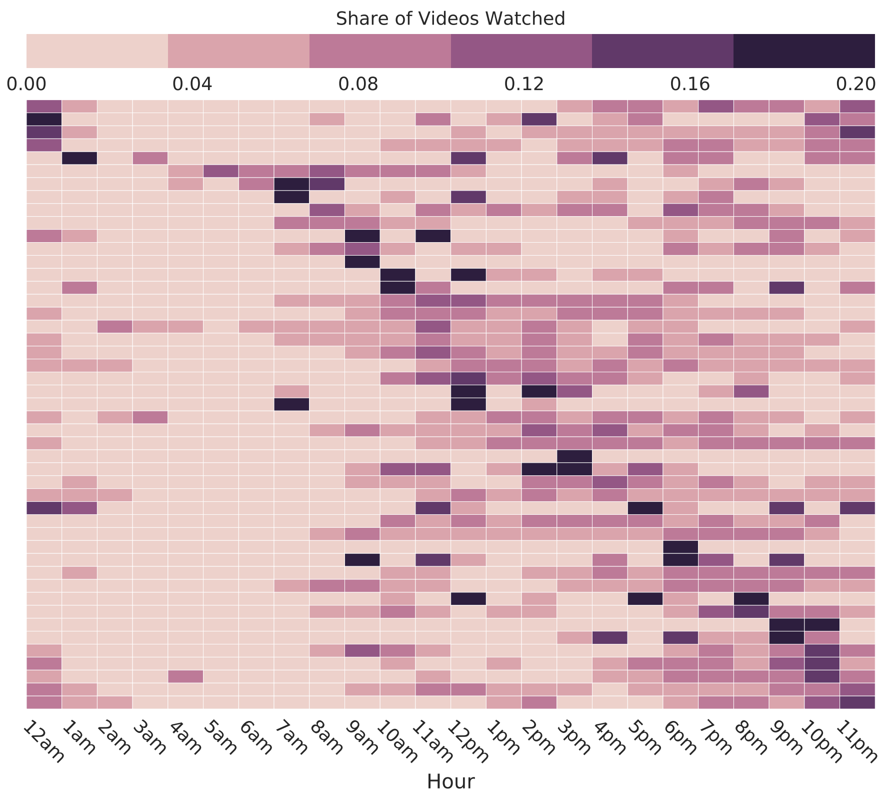


Figure 58: Views over the hours of the day, one line represents one user

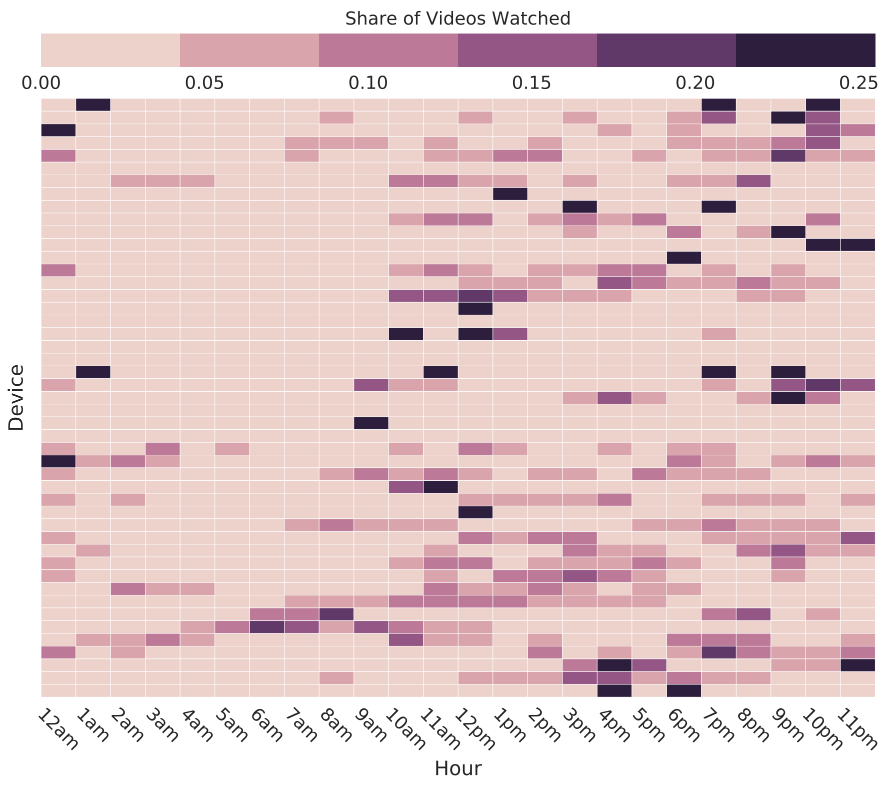


Figure 59: Views over the hours of the day and per participant for the category *Music*

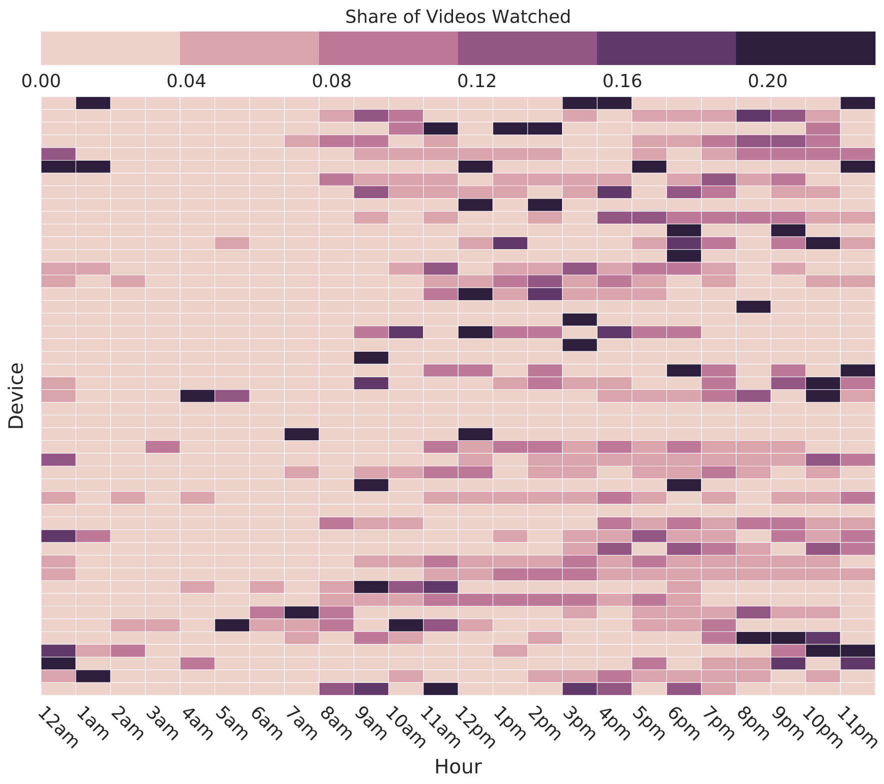


Figure 60: Views over the hours of the day and per participant for the category *Entertainment*

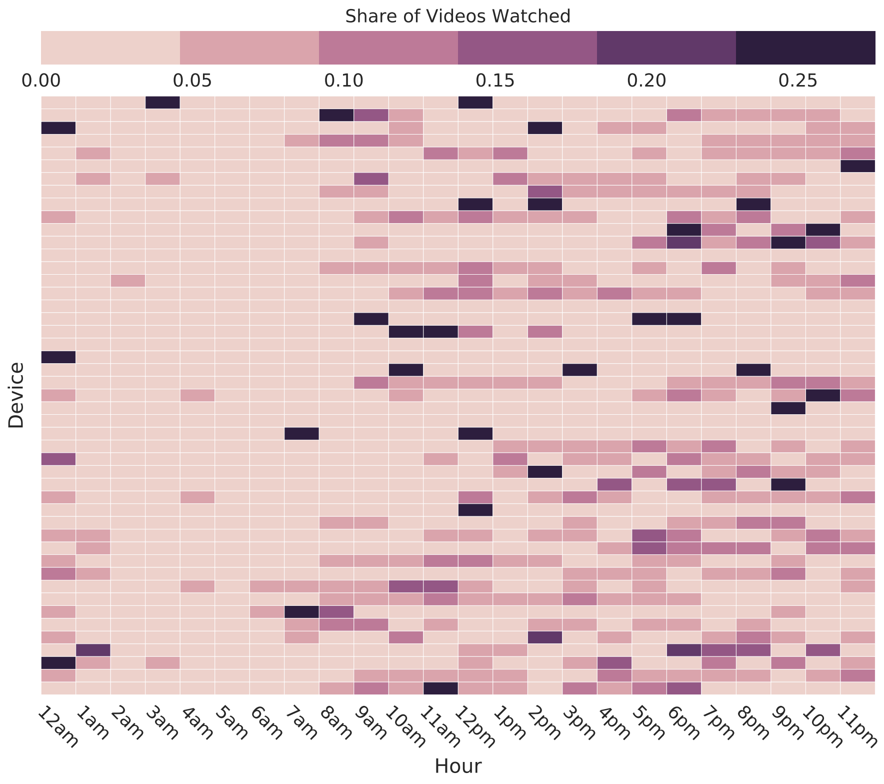


Figure 61: Views over the hours of the day and per participant for the category *People&Blogs*

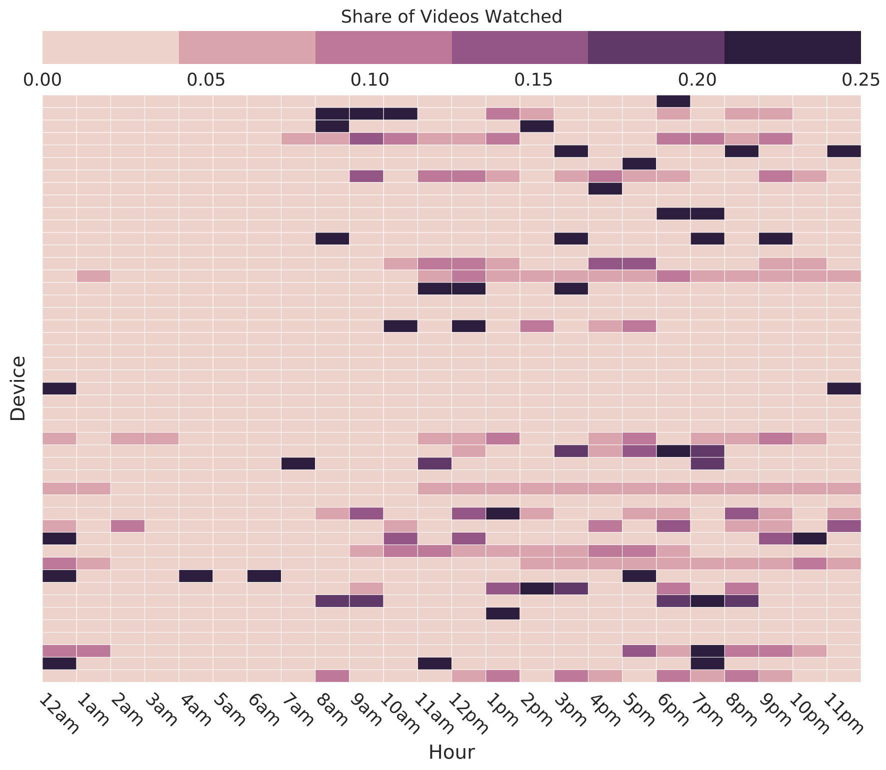


Figure 62: Views over the hours of the day and per participant for the category *Gaming*

In this section, we describe how the design of vFetch as presented in Section 4.5.1 can be extended to cooperate with the mobile network operator to which the mobile user terminal is connected. The extended architecture is depicted in Figure 63.

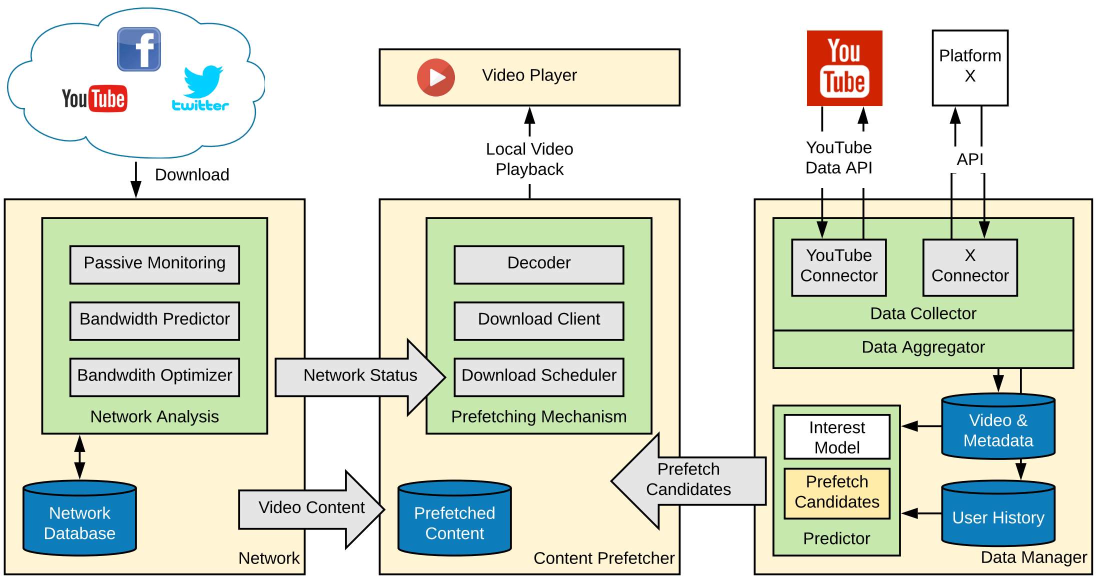


Figure 63: vFetch architecture extended with the *Network* module

The intuition is that the mobile network operator predicts the resource capacities and needs to give guidance on when to prefetch or smoothen the network load caused by video streaming. The *Network* module comprises three sub-mechanisms that we discuss in the following:

1. The **Bandwidth Optimizer** dynamically adapts the bandwidth allocated to a mobile terminal in case a content is not prefetched but streamed using the cellular connection. Thereby, it can compensate for fluctuating transmissions from the content source and provide a stable bandwidth to the user. To this end, the potentially fluctuating content stream from the source is delivered in a steady manner to the terminal, e.g., by a leaky bucket approach. This helps saving network resources since heavily fluctuating traffic demands are smoothened. To this end, it takes the forecasted throughput of the mobile terminal as an input, as well as the content-specific Quality of Experience (QoE) constraints and feedback from the terminal's video player.

2. The **Bandwidth Predictor** feeds the *Bandwidth Optimizer* with the throughput forecasts mentioned before. Therefore, it uses statistical information about the terminal's mobility trace and the available bandwidth in a given cellular network cell. We envision two typed of prediction mechanisms. First, short-term predictions are conducted by using time series forecasting such as Autoregressive Integrated Moving Average as proposed by Bui et al. [BW14]. Thereby, the prediction-frequency should be adaptive to the user's movement speed. Second, medium-term predictions are conducted using statistical models as proposed by Bui et al. [BMW14]. These predictions consider the degradation of accuracy of information over time considering the user's movement but also the

network cell congestion which is likely to change over the course of the course of the day. By combining the two prediction mechanisms introduced before, the **Bandwidth Predictor** can give guidance to vFetch's *Download Scheduler* when prefetching on a cellular connection is efficient since enough bandwidth is available to download content quickly and, hence to save energy.

3. By **Passive Monitoring** of the available bandwidths for the network cells that are managed by the mobile network operator, long-term statistics can be collected that support the *Bandwidth Predictor*'s medium-term bandwidth predictions.

A.4 EVALUATION OF FURTHER PROACTIVE CACHING POLICIES

In Section 5.4.6.1 and Section 5.4.6.2, we depicted only the results of the *popularity* policy (cf. Section 5.3.3.6) since it achieved the best results in our evaluation compared with other policies. To detail the differences, we exhibit the results of the policies *genre* and *aggregated similarity* in addition to *popularity* in Figure 64. Note that for the sake of clarity we do not show the results of the *mood* policy since it always performs worse than genre [KKH17] but follows a similar approach. We observe that the performance curves, for the policies *popularity*, *genre* and *aggregated similarity*, behave similar wile popularity is in any cache size scenario the superior policy. *Genre* and *Aggregated Similarity* show more similar performance results than *Popularity*. However, all policies show a clear performance drop for cache sizes for cache sizes between 600 and 1,000 items. In the case of *Aggregated Similarity*, the performance is decreased even for cache sizes up to 1,500 items, depending on the proactive cache share used.

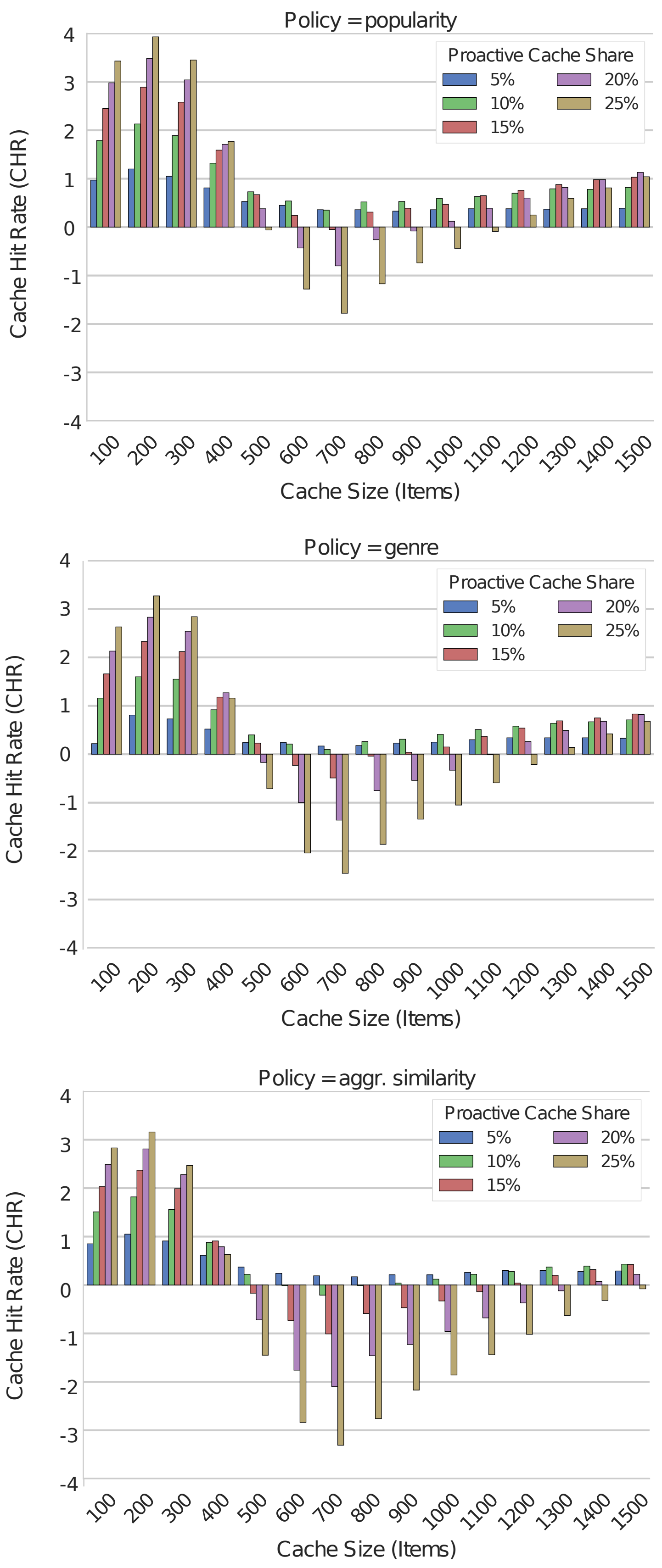


Figure 64: Relative cache performance compared with LRU

A lot of conducted studies try to find a fitting model that accurately describes real-world video popularity characteristics. This section provides a brief overview and compares their outcomes according to the changes over time. A common approach, discovered when analyzing web server loads, is the *Pareto rule* [AW97], which describes that 20% of a server's files are responsible for 80% of all requests. Nowadays, this rule is only partially applicable for Video-on-Demand (VoD) requests [GAL+07], because Pareto is not skewed enough as observed by Guillemin et al. [GKM+13]. Figure 65 depicts their results and indicates that only for specific popularity ranks Pareto is a suitable popularity model.

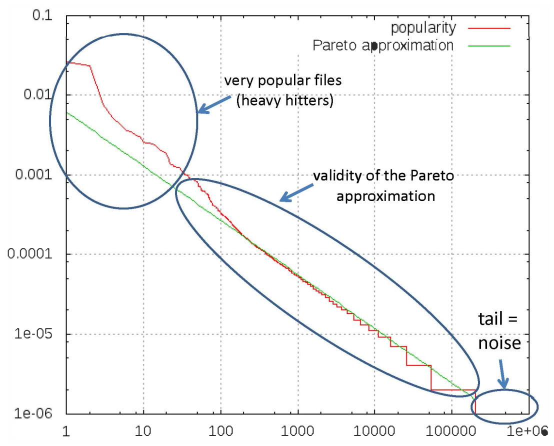


Figure 65: Popularity observed in traces from [GKM+13]

Zipf's Law

Zipf's Law is a popular specialized power-law model which allows assigning decreasing request probabilities $z(r)$ to each video according to their popularity rank $r \in \{1, ..., N\}$, whereas $\beta > 0$ can influence the shape and $\alpha > 0$ is a normalization constant [HNH16]:

$$z(r) = \alpha r^{-\beta}; \ \alpha = z(1) = 1/\sum_{r=1}^{N} z(r) \qquad (10)$$

To determine, if the current popularity is Zipf-distributed, we need to draw the ordered video rank versus the view count on a log-log scale. The primary parameter of the Zipf model is the shaping parameter β. In Table 18 currently used and measured values of β from the literature are presented.

Unfortunately, many papers that use a Zipf distributed content popularity model in their simulations do not mention their source nor provide reasons why they have chosen a specific value for β. By considering the age of the publications in Table 18, the more recent request traces, e.g., [AS10] and [GHM13] tend to have slightly higher values than 0.5, which indicates, that YouTube's video popularity distribution becomes more skewed over time. However, a set of studies [BMA+11; CKR+09; CMM+14] note that the Zipf model by itself is not the best fitting solution

Source	β	R²	Trace-based?	Description
[GAL+07]	0.56	0.97	yes	collected from an edge/campus network
[AS10]	0.63	0.71	yes	collected from recursive crawling of popular videos
[AS10]	0.8	0.98	yes	collected from recursive crawling most popular videos
[Di 15]	0.4-1	-	no	used for simulation
[HGK+15]	1	-	no	used for simulation
[KPH13]	0.7, 2	-	no	used for simulation
[SP06]	0.2	-	no	used for simulation
[CCC12]	0.729, 0.5-5	-	no	used for simulation (variation showed no effect)
[GJM+12]	0.5-1.5	-	no	used for proposing model
[AD12]	0.8	-	no	used for simulation (taken from [CKR+09])
[GHM13]	0.69-1	not given	yes	collected from Orange networks in France
[HNH16]	0.5-1	not given	yes	collected from different web content requests

Table 18: Zipf parameter β in the literature, recursive means that the videos related to a video were also crawled recursively

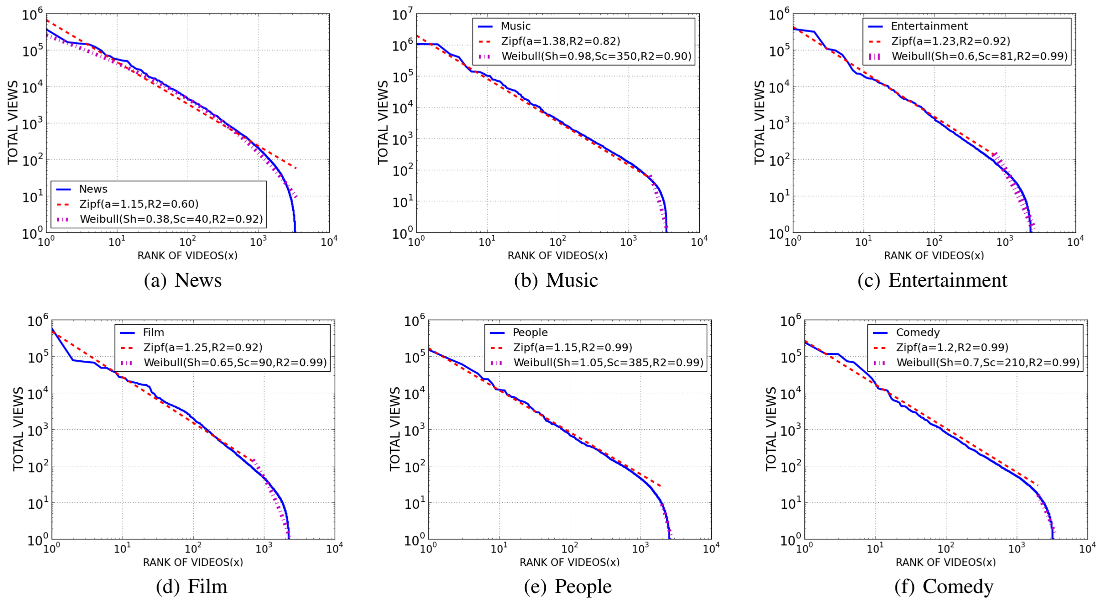
for every case. Generally, many works observed power-law behavior or even Zipf without mentioning further details. For example [LLX+12] and [LWL+13] observe a power-law popularity distribution while monitoring video requests coming from Online Social Networks (OSNs). Cheng et al. [CMM+14] even observe a Zipf distribution on the daily requests measured by a YouTube partner.

Weibull Distribution

Another common and flexible popularity distribution model is named, after its inventor, Weibull. It has a set of parameters that lead to great flexibility in the curve's shape. Below, we exhibit the Weibull distribution function with the scale parameter $\lambda > 0$ and the shape parameter $k > 0$:

$$f(x) = \begin{cases} 1 - e^{-(\lambda x)^k} & x \geqslant 0 \\ 0 & x < 0 \end{cases} \tag{11}$$

As we can see in Figure 66, a Weibull distribution fits better for real-world traces such as from Chowdhury et al. [CM13], because of the heavy long tail which covers unpopular videos. YouTube has 15 different categories, e.g., Music, Entertainment, Sports, or News. The authors discover YouTube category-dependent popularity preferences. A further interesting finding is that videos from the Sports and News category have a higher popularity than others for the first couple of days after publication. In longer observation intervals, Music videos surpass all other categories regarding popularity. Also Cha et al. [CKR+09] stress, that exact popularity distributions appear to be category-dependent. From a YouTube's partners perspective, the observed re-watch rates for entertaining channels, e.g., Music or Games differ from other categories due to the attractiveness of their content [CMM+14]. Further on, the crawled traffic traces from Abhari et al. [AS10], which are related videos of the most popular ones (collected 2-3 times a week), do not fit a Zipf distribution very well, as we can see in Table 18 ($R^2 = 0.71$). Therefore, they computed the Weibull parameters for their data



(a) News (b) Music (c) Entertainment

(d) Film (e) People (f) Comedy

Figure 66: Video ranks and views for different YouTube categories [CM13]

which fit significantly better with low coefficients of variation (COV), a metric which is defined as $\frac{\sigma}{\mu}$ and determines the relative standard deviation.

Other Approaches

Besides Zipf and Weibull distributions, other approaches have been proposed. Borghol et al. [BMA+11] split the content popularity evolution into three conceptual phases: *before*, *at*, and *after* the content popularity peak. Hence, each of the phases can be modeled by its distinct distribution which makes the overall distribution more flexible and agile. The authors use different log-normal or power-law distributions for the tail and a beta distribution for the body. The formula for the density function with the parameters μ and σ is:

$$f(x) = \begin{cases} \frac{1}{\sqrt{2\pi}\sigma x} \exp\left(-\frac{(\ln(x)-\mu)^2}{2\sigma^2}\right) & x < 0 \\ 0 & x \geqslant 0 \end{cases} \tag{12}$$

Note that log-normal and power-law distributions have similar shapes [Mit04], so it is often difficult to differentiate which distribution fits best. To determine this, Borghol et al. applied the Log Likelihood Ratio test (LLR). Their results show that the log-normal distribution fits better for the tails of their empirical data. For the body, they used a beta distribution. The reason for this decision is that it fits best to their empirical data, after trying out a set of distributions. This may lead to an overfitting problem since other trace datasets may have other underlying distributions which better fit their data. Therefore, it is hard to determine a common approach without focusing too much on the given traces.

Cha et al. [CKR+09] study popularity distributions using User-generated Content (UGC) traces. Besides YouTube, they also analyze a similar Korean service called Daum[1]. While testing against the Pareto rule, for both platforms they observed that around 10% of the most popular videos account for nearly 80% of views. Figure 67 depicts the best distribution which most accurately models the video popularity in their traces, i.e., a power-law distribution with an exponential cutoff. The authors

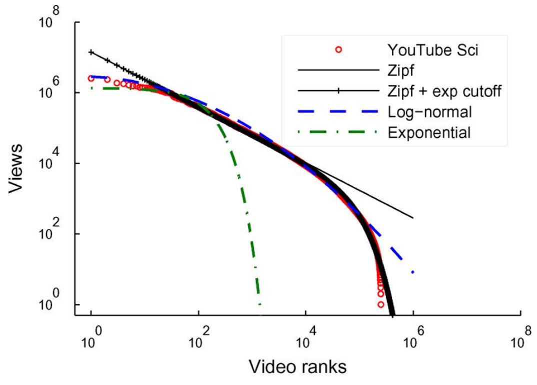


Figure 67: YouTube tail fitting for different distributions [CKR+09]

discuss if the underlying distribution can be modeled as a Zipf distribution with a bottleneck or if it is a curved log-normal distribution. According to the authors, one reason for Zipf is that its distributions are “overwhelmingly prevalent in the real world” [CKR+09]. The authors explain the bottleneck with a sort of information filtering which prevents the users from finding rare niche videos.

[1] http://www.daum.net/ [Accessed: November 19, 2018]

Carlinet et al. [CHK+12] observe a related heavy-tailed popularity distribution with a smaller cutoff while collecting session traces from Dailymotion[2]. Figure 68 exhibits this distribution. Similar as in [BMA+11], they categorized videos, not in different phases, but in patterns, such as bursty or long-span, where bursty videos are exhibit generally a higher ranking than long-span ones.

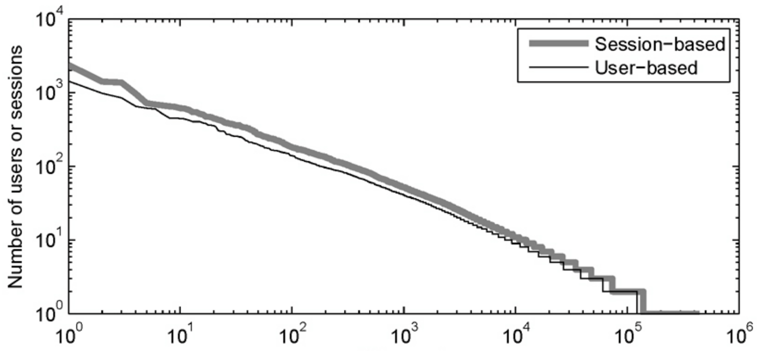


Figure 68: Ranking over video session frequency [CHK+12]

[2] http://www.dailymotion.com [Accessed: November 19, 2018]

B AUTHOR'S PUBLICATIONS

B.1 MAIN PUBLICATIONS

[KBR+15] C. Koch, N. Bui, J. Rückert, G. Fioravantti, F. Michelinakis, S. Wilk, J. Widmer, and D. Hausheer. "Media Download Optimization through Prefetching and Resource Allocation in Mobile Networks." In: *ACM Multimedia Systems Conference (MMSys)*. 2015, pp. 85–88.

[KH14] C. Koch and D. Hausheer. "Optimizing Mobile Prefetching by Leveraging Usage Patterns and Social Information." In: *IEEE International Conference on Network Protocols (ICNP)*. 2014, pp. 293–295.

[KHH17] C. Koch, S. Hacker, and D. Hausheer. "VoDCast: Efficient SDN-based Multicast for Video on Demand." In: *IEEE International Symposium on a World of Wireless and Multimedia Networks (WoWMoM)*. 2017, pp. 1–6.

[KKH17] C. Koch, G. Krupii, and D. Hausheer. "Proactive Caching of Music Videos based on Audio Features, Mood, and Genre." In: *ACM Multimedia Systems Conference (MMSys)*. 2017, pp. 100–111.

[KLR+17] C. Koch, B. Lins, A. Rizk, R. Steinmetz, and D. Hausheer. "vFetch: Video Prefetching using Pseudo Subscriptions and User Channel Affinity in YouTube." In: *IEEE International Conference on Network and Service Management (CNSM)*. 2017, pp. 1–9.

[KLS+18b] C. Koch, M. Lode, D. Stohr, A. Rizk, and R. Steinmetz. "Collaborations on YouTube: From Unsupervised Detection to the Impact on Video and Channel Popularity." In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 14.4 (Oct. 2018), pp. 1–23.

[KPR+18] C. Koch, J. Pfannmüller, A. Rizk, D. Hausheer, and R. Steinmetz. "Category-aware Hierarchical Caching for Video-on-Demand Content on YouTube." In: *ACM Multimedia Systems Conference (MMSys)*. 2018, pp. 1–12.

[KWR+18] C. Koch, S. Werner, A. Rizk, and R. Steinmetz. "MIRA: Proactive Music Video Caching using ConvNet-based Classification and Multivariate Popularity Prediction." In: *IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 2018, pp. 1–7.

B.2 CO-AUTHORED PUBLICATIONS

[CVM+15] C. G. Cordero, E. Vasilomanolakis, N. Milanov, C. Koch, D. Hausheer, and M. Mühlhäuser. "ID2T: A DIY Dataset Creation Toolkit for Intrusion Detection Systems." In: *IEEE Conference on Communications and Network Security (CNS)*. 2015, pp. 739–740.

[Gou+15] A. Gouta et al. "CPSys: A System for Mobile Video Prefetching." In: *IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 2015, pp. 188–197.

[WKB+17] M. Wichtlhuber, J. Kessler, S. Bücker, I. Poese, J. Blendin, C. Koch, and D. Hausheer. "SoDA: Enabling CDN-ISP Collaboration with Software Defined Anycast." In: *IFIP International Conference on Networking (NETWORKING)*. 2017, pp. 1–9.

[WRT+15] S. Wilk, J. Rückert, T. Thräm, C. Koch, W. Effelsberg, and D. Hausheer. "The Potential of Social-aware Multimedia Prefetching on Mobile Devices." In: *IEEE International Conference on Networked Systems (NetSys)*. 2015, pp. 1–5.

B.3 DEMO PAPERS

[KRB+14] C. Koch, J. Rückert, N. Bui, F. Michelinakis, G. Fioravantti, J. Widmer, and D. Hausheer. "Demo: Mobile Social Prefetcher using Social and Network Information." In: *IEEE International Workshop on Computer-Aided Modeling Analysis and Design of Communication Links and Networks (CAMAD)*. 2014, pp. 1–10.

B.4 TECHNICAL REPORTS

[KLS+18a] C. Koch, M. Lode, D. Stohr, A. Rizk, and R. Steinmetz. "Collaborations on YouTube: From Unsupervised Detection to the Impact on Video and Channel Popularity." In: *arXiv preprint arXiv:1805.01887* (2018), pp. 1–28.

[LÖK+18] M. Lode, M. Örtl, C. Koch, A. Rizk, and R. Steinmetz. "Detection and Analysis of Content Creator Collaborations in YouTube Videos using Face-and Speaker-Recognition." In: *arXiv preprint arXiv:1807.02020* (2018), pp. 1–12.

C CURRICULUM VITÆ

PERSONAL INFORMATION

Name	Christian Koch
Date of Birth	April 5, 1988
Place of Birth	Mühlhausen, Germany
Nationality	German

EDUCATION

01/2014–10/2018	Technische Universität Darmstadt Doctoral candidate at the department of Electrical Engineering and Information Technology
10/2008–11/2013	Technische Universität Darmstadt Master of Science in IT Security
10/2008–11/2013	Technische Universität Darmstadt Master of Science in Computer Science

PROFESSIONAL EXPERIENCE

05/2017–09/2018	Technische Universität Darmstadt Department of Electrical Engineering and Information Technology Research assistant at Multimedia Communications Lab (KOM)
01/2014–04/2017	Technische Universität Darmstadt Department of Electrical Engineering and Information Technology Research assistant at Peer-to-Peer Systems Engineering (PS), affiliated with Multimedia Communications Lab (KOM)

AWARDS AND HONORS

05/2017	**Student Travel Grant** from the ACM SIGMM - the Special Interest Group on Multimedia, awarded by ACM Multimedia Systems Conference (MMSys) 2017 Christian Koch, Ganna Kruppi, David Hausheer: *Proactive Caching of Music Videos based on Audio Features, Mood, and Genre*
12/2017	**Student Travel Grant** from the Selection Committee of the IEEE International Conference on Network and Service Management (CNSM) 2017

	Christian Koch, Benedikt Lins, Amr Rizk, Ralf Steinmetz, David Hausheer: *vFetch: Video Prefetching using Pseudo Subscriptions and User Channel Affinity in YouTube*
03/2018	**KuVS Award** from the Selection Committee of the ”KuVS: Fachgruppe “Kommunikation und Verteilte Systeme (KuVS)” 2017
	Moritz Lode: "Detection and Analysis of Content Creator Collaborations in YouTube Videos using Face Recognition", Bachelor Thesis under the supervision of Christian Koch

SCIENTIFIC ACTIVITIES

Reviewer	IFIP Int. Conf. on Networking (NETWORKING): *2014, 2015, 2016, 2017*
	IFIP Int. Conf. on Autonomous Infrastructure, Mgmt. and Security (AIMS): *2014, 2015, 2016, 2017*
	IFIP/IEEE Int. Symp. on Integrated Network Mgmt. (IM): *2015, 2017*
	IEEE Int. Conf.on Computer Communications (INFOCOM): *2019*
	IEEE Conf. on Network Softwarization (NetSoft): *2015, 2017*
	IEEE Conf. on Local Computer Networks (LCN): *2015, 2016*
	IFIP Int. Conf. on Network and Service Mgmt. (CNSM): *2015, 2016, 2017*
	IEEE Conf. on Cloud Comp. (CloudCom): *2015*
	IEEE/IFIP Network Operations and Mgmt. Symp. (NOMS): *2016*
	ACM Int. Symp. on Mob. Ad Hoc Networking & Comp. (MobiHoc): *2016*
	ACM Multimedia Systems Conference (MMSys): *2018*
	ACM Multimedia (MM): *2017, 2018*
	ACM Int. Conf. on Cloud Comp. and Services Science (CLOSER): *2018*

SCIENTIFIC ACTIVITIES

Organization	Student Member of IEEE Communications Society since 2014
	Student Member of ACM Community since 2014

TEACHING ACTIVITIES

Lectures	“P2P Systems and Applications”: Lecture and exercise presentation, general organization (SoSe14, SoSe15, SoSe16)
	“Software Defined Networking”: Lecture and exercise presentation, organization, exam design and coordination (WiSe14/15, WiSe15/16, WiSe16/17)
	“Communication Networks I”: Lecture and exercise presentation, organization, exam design and coordination (SoSe17, SoSe18)

Seminars	“Internet Scale Multimedia Distribution and Monitoring": Supervisor (WiSe14/15)
	“Software Defined Networking”: Supervisor (SoSe14, SoSe15, SoSe16)
Labs	“SmartNets Lab"/“Praktikum Intelligente Netzwerke”: Organization, task definition, supervisor (WiSe15/16)
	“Bachelor Students Traineeship / Bachelor-Praktikum (FB20)”: Supervisor of four-person student group over six months (WiSe16)
	“Advanced Topics in Communication Networks”: Supervisor (SoSe14, WiSe14/15, SoSe15, WiSe15/16, SoSe16, WiSe16/17)
	“Multimedia Communications Lab”: Supervisor (SoSe15, WiSe17/18, SoSe18)

SUPERVISED STUDENT THESES

KOM-B-0626	Arne-Tobias Rak, *Transitions of Quality Adaptation Mechanisms in 360° Video Streaming*. Bachelor thesis, Technische Universität Darmstadt, August 2018.
KOM-M-0640	Joel Koschier, *Designing a Machine Learning based Prefetch System for YouTube Videos on Mobile Device*. Master thesis, Technische Universität Darmstadt, August 2018.
KOM-B-0594	André Daube, *Improvement of Costs and QoE of Composed Live Video Streams by Combined Multicast and Caching Approaches*. Bachelor thesis, Technische Universität Darmstadt, July 2017.
KOM-M-0602	Kliinnglills Kliinnglills, *A Comprehensive Evaluation of Video-on-Demand Multicast on YouTube*. Master thesis, Technische Universität Darmstadt, July 2017.
KOM-B-0593	Stefan Werner, *Investigating Machine Learning Methods for Proactive Network Caching of Music Content*. Bachelor thesis, Technische Universität Darmstadt, December 2017.
KOM-B-0592	Nils Mäser, *Alleviating Mobile Video Streaming from Dead Zones by Preloading Video Segments*. Bachelor thesis, Technische Universität Darmstadt, February 2018.
PS-D-0044	Markus Schanz *Synthetic Workload Generation for Online Video Platforms*. Master thesis, Technische Universität Darmstadt, June 2017.
PS-D-0038	Sudeep Duggal *Recommending Video-on-Demand for Prefetching in Content Delivery Networks*. Master thesis, Technische Universität Darmstadt, January 2017.
PS-D-0036	Simon Schindel *Cache-aided DASH in 5G Networks using HTTP/2 Server Push*. Master thesis, Technische Universität Darmstadt, November 2016.
PS-D-0032	Ganna Krupii *Developing a Proactive Caching Mechanism for Music Content*. Master thesis, Technische Universität Darmstadt, May 2016.

PS-D-0028	Stefan Hacker *Simultaneous Partial Delivery of Video-on-Demand Streams*. Master thesis, Technische Universität Darmstadt, May 2016.
PS-S-0026	Moritz Lode *Detection and Analysis of Content Creator Collaborations in Youtube Videos using Face Recognition*. Bachelor thesis, Technische Universität Darmstadt, June 2016.
PS-S-0024	Johannes Pfannmüller *Vergleichen von Caching-Strategien in Netzen für unterschiedliche Typen von Inhalten*. Bachelor thesis, Technische Universität Darmstadt, June 2016.
PS-S-0023	Benedikt Lins *Textual and Content based Video Prefetching for Mobile Devices*. Bachelor thesis, Technische Universität Darmstadt, December 2016.
PS-S-0017	Joel Koschier *Prädiktives Cachen von Videos bereitgestellt auf Sozialen Plattformen*. Bachelor thesis, Technische Universität Darmstadt, November 2015.
PS-S-0016	Julio Klappich *GPU-basierte Optimierung von Maximum Influencer Algorithmen in sozialen Netzwerken*. Bachelor thesis, Technische Universität Darmstadt, January 2016.

ERKLÄRUNG LAUT PROMOTIONSORDNUNG

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Disserationsthema und Ergebnis dieses Versuchs mitzuteilen.

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 26. Juni 2018

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/wiki/Home

The YouTube symbol used in Section 4.3, Section 4.5, and Section A.3 is provided by Aha-Soft and is licensed under the Attribution 3.0 Unported (CC BY 3.0) license. It is available at:

The YouTube logo used in Section A.3 is provided by YouTube and available at:

https://commons.wikimedia.org/wiki/File:Logo_of_YouTube_(2015-2017).svg

The Facebook logo used in Section A.3 is provided by Facebook and available at:

https://commons.wikimedia.org/wiki/File:F_icon.svg

The Twitter logo used in Section A.3 is provided by Twitter and available at:

https://seeklogo.com/vector-logo/305546/twitter

All plots were generated using a custom plotting platform mainly based on the Python library `matplotlib` (http://matplotlib.org/) and `Seaborn` (https://seaborn.pydata.org/).

Final Version as of November 19, 2018 (`classicthesis` 1.0.0).