Adaptive streaming using Peer-to-Peer and HTTP

Hiba Yousef

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAT017

Thèse de doctorat

Adaptive streaming using Peer-to-Peer and HTTP

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctoral Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Signal, Images, Automatique et Robotique

Thèse présentée et soutenue à Paris, le 09/07/2021, par

HIBA YOUSEF

Composition du Jury :

Véronique VEQUE Professor, Université Paris Saclay	Présidente
Christian TIMMERER Associate Professor, Alpen-Adria-Universität Klagenfurt	Rapporteur
Lucile SASSATELLI Associate Professor, Université Côte d'Azur	Rapporteure
Anissa MOKRAOUI Professor, Université Paris 13 Sorbonne Paris Cité	Examinatrice
Jean-Claude DUFOURD Professeur, Télécom Paris	Directeur de thèse
Jean LE FEUVRE Associate Professor, Télécom Paris	Co-directeur de thèse

Invités :

Alexandre Storelli Lead R&D Efficiency, Lumen	Encadrant Industriel

Résumé

La croissance du trafic vidéo liée à l'offre et au nombre d'utilisateurs, ainsi que les progrès des technologies vidéo et des appareils, ont augmenté les attentes des utilisateurs en termes de Qualité d'Expérience (QoE). Aujourd'hui, le trafic vidéo représente 79% du trafic Internet mondial, et ce pourcentage devrait atteindre 82% d'ici 2022 avec les services Over The Top (OTT) qui représentent plus de 50% du trafic de pointe dans le monde. Les solutions HTTP Adaptive Streaming (HAS) se sont révélées être l'une des techniques essentielles pour faire face à ce trafic vidéo en constante augmentation, grâce à leur logique d'adaptation en débit (ABR) intégrée côté client, qui permet de s'adapter aux conditions d'utilisation (oscillations de bande passante, ressources matérielles...) afin de maximiser la QoE de l'utilisateur. En parallèle, la distribution vidéo sur les réseaux Pair-à-Pair (P2P) et sur les réseaux de diffusion de contenu (CDN) devient cruciale pour permettre au réseau de faire face à l'explosion du nombre de consommateurs vidéo. Suite aux récentes améliorations des technologies P2P et HAS, de nombreux efforts ont été déployés pour rapprocher ces deux techniques. Cependant, le déploiement HAS sur les réseaux P2P pose de nombreux défis. Le réseau P2P est problématique pour les techniques HAS en raison de l'hétérogénéité des ressources et de la fréquence des arrivées/départs des clients. Une grande partie des implémentations repose sur un modèle en couches où les piles HAS et P2P sont isolées l'une de l'autre; dans ce modèle, les techniques de pré-chargement P2P sont in-

dépendantes de la logique ABR utilisée, ce qui conduit à une utilisation inefficace des ressources réseau lors des changements de qualité. Cette thèse se concentre sur les implémentations de piles HAS et P2P en couches et vise à analyser les problèmes mentionnés ci-dessus et à proposer des méthodes pour les résoudre, tout en améliorant l'efficacité de la distribution P2P. Pour y parvenir, nous construisons un environnement de simulation pour tester les solutions HAS dans les systèmes hybrides CDN/P2P et analyser les problèmes associés. Nous proposons «Response-Delay», une méthode permettant l'utilisation d'algorithmes HAS existants dans le contexte de réseaux P2P basés sur le pré-chargement; cette méthode module le délai de réponse des requêtes en amont du lecteur HAS et ne nécessite aucune modification de l'algorithme ABR implémenté. Nous proposons par ailleurs des modèles d'apprentissage pour prédire les décisions de qualité des algorithmes HAS, en utilisant un ensemble de métriques d'entrée que l'ABR utilise pour prendre une décision de débit. Enfin, nous combinons «Response-Delay» et les modèles d'apprentissage ABR pour définir une méthode de pré-charegment et de contrôle de QoE plus efficace. Cette technique utilise la décision ABR prédite dans le processus de pré-chargement et contrôle l'ABR en amont pour prendre des décisions favorables au P2P.

Abstract

The increasing growth of video traffic and the number of Internet users, besides the progressing video technologies and device capabilities, have surged the demand for improving the user Quality of Experience (QoE). Today, video traffic accounts for 79% of the global Internet traffic, and this percentage is projected to strike 82% by 2022 [1], with Over The Top (OTT) services accounting for more than 50% of the peak download traffic globally [2].

HTTP Adaptive Streaming (HAS) solutions have shown to be one of the essential techniques to cope with this ever-increasing video traffic, thanks to their embedded Adaptive BitRate (ABR) logic at the client-side which allows adaptation to the bandwidth oscillations and maximizing QoE.

In parallel, video distribution over Peer-to-Peer (P2P) networks, along with Content Delivery Networks (CDN), is becoming more important to handle the explosion in the number of video consumers.

As a result of P2P and HAS recent improvements, there have been many efforts to bring these two approaches together. However, the deployment of HAS streaming over P2P networks raises many challenges. The P2P nature is problematic due to the heterogeneity of resources and the dynamicity of peers. The layered implementations where HAS and P2P stack are isolated from each other. The P2P prefetching techniques are not aware of the used ABR logic, which leads to inefficient usage of the network resources.

This thesis focuses on the layered HAS and P2P stack implementations and aims to analyze the above-mentioned issues and propose methods to solve them, enhancing QoE and P2P efficiency. To achieve this, we build a simulation environment to test HAS solutions in hybrid CDN/P2P systems and analyze the related issues. We propose Response-Delay, a method enabling usage of existing HAS algorithms in the context of prefetching-based P2P networks; Response-Delay is external to the video player and does not require any modification to the implemented ABR algorithm. Besides, we propose ML-based models to predict the quality decisions of HAS algorithms, using only a set of input metrics that the ABR can use to make a bitrate decision. Finally, we combine Response-Delay and the ML-based ABR models towards an ABR-aware prefetching and quality control technique. This technique uses the predicted ABR decision in the prefetching process and controls the ABR externally to make P2P-friendly decisions.

Contents

Acronyms

ABR Adaptive BitRate

CDN Content Delivery Networks

DASH Dynamic Adaptive Streaming over HTTP

HAS HTTP Adaptive Streaming

MPEG Moving Picture Expert Group

OTT Over-The-Top

P2P Peer-to-Peer

QoE Quality of Experience

QoS Quality of Service

TCP Transmission Control Protocol

UDP User Datagram Protocol

Chapter 1

Introduction

1.1 Motivation

Video streaming has become an essential part of our everyday life due to the advances in video coding technologies, device capabilities, and networking technologies. Broadcasting is the dominant supplier of electronic content to the public; regarding that, Nielsen has published that US adults spend 5 hours and 43 minutes a day on video content across TV, TV-connected devices, computer, smartphone and tablets [3]. Most broadcast services are shifting from traditional terrestrial, satellite services towards streaming over the Internet. The two primary media streaming services are IPTV and Over-The-Top (OTT) applications, and they both rely on the IP protocol. IPTV content is streamed to users through dedicated and managed networks, guaranteeing a high end-to-end Quality Of Service (QoS). On the other hand, OTT services like Netflix, Amazon Prime, Hulu, and Youtube deliver the content over open, unmanaged networks, utilizing the publicly accessible Internet connection, hence the terminology Over the Top. Yet, OTT services often choose a hybrid solution involving managed networks, such as Content Delivery Network (CDN), to guarantee QoS.

Today, video traffic accounts for 79% of the global Internet traffic, up from 75% in 2016, and this percentage is projected to strike 82% by 2022 [1], with OTT services accounting for more than 50% of the peak download traffic globally [2]. In addition to that, the number of Internet users is also ever-increasing. Forecasts refer that nearly two-thirds of the global population will have Internet access by 2023, which means around 5.3 billion total Internet users (66% of the global population), up from 3.9 billion (51% percent of the worldwide population) in 2018 [4].

From these numbers, we can imagine how the future of video streaming, especially for OTT services may become too overwhelming for Internet Service Providers (ISPs), who have to handle and serve all these users and meet their high expectations for a good Quality of Experience (QoE).

As we mentioned, OTT services often choose to guarantee QoS by delivering their content with the help of CDN providers, such as Akamai and Limelight. CDN providers host the contents on a set of servers placed in the network. Thus, instead of downloading the content from the origin multimedia server, a user normally is redirected to download the video from the nearby server (proxies), which reduces the traffic in the core network. However, the major challenge of CDN is scalability, as the bandwidth provisioning at the edge servers has to grow proportionally with the number of clients, making CDNs an expensive solution for large client populations [5].

Peer-to-Peer (P2P) networks help solving the edge scalability issue by leveraging the resources of the participating peers. In P2P streaming, each peer can upload the video segments to other peers while downloading its own segments, contributing to the overall available bandwidth. Contrary to CDN, P2P networks do not require any fixed infrastructures, but they utilize the user's upload resources to achieve video streaming, making P2P a cost-efficient and scalable solution. Thanks to their self-scaling and cost-efficient properties, Peer-to-Peer (P2P) networks have

become a popular alternative for delivering the video contents. However, the P2P nature itself poses some challenges due to the resource instability, in the sense that users can join or leave at any moment, and to the resource heterogeneity as peers have heterogeneous upload and download capacities. This may lead to slow downloads, which in turn degrades QoE [6] [7]. Thus, having a hybrid CDN/P2P approach would combine the advantages of P2P scalability and cost-efficiency and CDN reliability and manageability.

In parallel to the advances in delivery networks, streaming protocols have also evolved in the past decades. Many streaming protocols have been proposed, such as Real-Time Messaging Protocol (RTMP), Real-Time Streaming Protocol (RTSP), and Real-Time Transport Protocol (RTP). Recently, HTTP Adaptive Streaming (HAS) has proven to be a critical feature for delivering the video over the existing web service infrastructures. HAS uses the Hyper Text Transfer Protocol (HTTP) protocol as an application layer protocol that runs over Transmission Control Protocol (TCP) as a transport protocol. HAS partitions the video stream into smaller parts called segments encoded into different qualities and stored on an HTTP server. HAS enables the clients to pull the appropriate quality by running what is called Adaptive BitRate (ABR) logic that makes sequential per-segment bitrate decisions by measuring some system inputs and selecting the best quality that would improve the QoE. Today, HAS has become a part of the video streaming industry, and many big companies have used it and proposed their own implementations. Microsoft developed MSS (Microsoft Smooth Streaming) [8], Apple HLS (HTTP Live Streaming) [9], and Adobe OSMF (Open Source Media Framework) [10]. In 2012, the Moving Picture Expert Group (MPEG) came up with DASH [11] standard, which soon became the key HAS development as it is an open format which is compatible with any codec and container.

Caching and prefetching are also techniques that have been investigated to

improve QoE and reduce the access latency. Caching has been widely known as an effective way to reduce the access latency by keeping popular content at proxies that are closer to users [12] [13]. Prefetching is different from caching in that the cached content is stored for future reuse (typically by other clients), whereas in prefetching, the requested content is expected to be needed in the near future. Prefetching aims to maximize the cache hit ratio by predicting future requests for data before users actually need it. Many P2P systems, such as [14] [15], and [16], have used prefetching techniques to prefetch video segments and store them into local caches before the video player requests them. Prefetching exploits the available bandwidth of peers and decreases the risk of failing to download it in time when requested.

In a nutshell, CDN and P2P streaming networks have to deliver the stream respecting the quality requested by the adaptation logic implemented at the client side. For CDN servers, the process is a simple server-client data exchange where the client pulls the segments from the closest server, and the ABR is designed to adapt to the server-client link capacity. However, achieving the same with P2P is not easy for several reasons.

Some reasons relate to the nature of the P2P itself, as most peers have limited upload capacity since their access links are mostly asymmetric (greater download than upload bandwidth). Also, most P2P streaming schemes work by further dividing the video segment into smaller chunks, so they are easy to get from peers with limited bandwidth and so that one segment can be downloaded from different peers. However, the unexpected joining and leaving of peers may lead to losing the chunks providers or re-assigning the chunks to the newcomers, and thus chunks/segments may not arrive in their playback order, hereby increasing delay/re-buffering.

Other reasons are related to the layered implementations of P2P stack and HAS. In this implementation, the video player is usually integrated on top of the

P2P stack, which replaces the HTTP stack as a transport layer. The player's ABR, whether web-based or native, and the P2P stack are typically unaware of each other, making the P2P-ABR integration more problematic.

In such designs, the existence of P2P local caching is an important issue that directly affects the ABR behavior. Usually, the ABR expects to receive the segments from CDN servers after the download time. Then it measures this download time and updates the throughput and other measurements to select the next bitrate. Clearly, this is problematic when the ABR receives the segment directly from the P2P cache. In this case, the ABR measures a short download time and an infinite bandwidth for cached P2P segments and measures a longer download time for CDN segments which drastically changes the ABR behavior. In that direction, in Chapter 4, we tackle the cache existence problems with the ABR algorithms, and we propose a methodology to solve this issue.

Prefetching is another open issue with P2P/ABR streaming as it usually follows the last ABR decision to get P2P segments in advance. When the ABR switches to another quality, this leads to discarding the cached segments while trying to get the new quality from new peers, even if the cached segments are of a higher quality than the requested one. In that direction, Chapter 5 investigates the ability to predict the ABR decision using machine learning models. Then Chapter 6 combines the propositions of Chapter 4 and 5 to enable an ABR-aware prefetching. Besides, it proposes a method to control the ABR to select the highest prefetched quality, preventing it from switching down when the same segment is prefetched on better quality.

Overall, this thesis treats the adaptive streaming from the perspective of P2P networks. The main motivation is to make the P2P streaming compatible with the already existing HAS work, without any modification to the HAS algorithms, improve the QoE and the P2P efficiency, and improve the global HAS performance in

prefetching-based environments.

In summary, the objectives of this dissertation are to:

- provide tools that would enable testing adaptive streaming in hybrid CDN/P2P environments.
- address the current challenges of HTTP adaptive streaming in hybrid CDN/P2P networks.
- propose methodologies to enable using the existing HAS algorithms in the hybrid CDN/P2P systems without any change on the HAS side, staying compatible with the layered P2P and HAS implementation.
- improve the overall QoE and the P2P performance in prefetching-based environments

Towards achieving these objectives, we enlist in Section 1.2 the main contributions of this dissertation in terms of published work.

1.2 Contributions

In Chapter 2, we provide a comprehensive survey of the state of the art in ABR design, P2P streaming schemes and QoE.

In Chapter 3, we present an initial testbed to enable hybrid CDN/P2P streaming, specifically for WebRTC on web browsers. Although this testbed enables a good traffic shaping, it does not allow reproducible results. Therefore, we also present a full MATLAB-based simulation for adaptive streaming over hybrid CDN/P2P networks, which we made open for research purposes[1].

In Chapter 4, we analyze the main problems raised by using the existing HTTP adaptive streaming algorithms in the context of P2P networks. We mainly treat the

[1] Arxiv paper to be published

problem of the local P2P cache existence on the ABR behavior. We propose two methodologies to solve this issue and make these algorithms more efficient in P2P networks, regardless of the ABR algorithm used. The work presented in Chapter 4 is associated with the following publication:

- H. Yousef, J. L. Feuvre, P.L. Ageneau, and A. Storelli. 2020. Enabling adaptive bitrate algorithms in hybrid CDN/P2P networks. In Proceedings of the 11th ACM Multimedia Systems Conference (MMSys '20). Association for Computing Machinery, New York, NY, USA, 54–65.

In Chapter 5, we present a generic ML-based approach to predict the bitrate decision of any ABR algorithm. This model does not require any knowledge about the player ABR algorithm itself, but assumes that it will use a common set of input features, whatever the logic behind it. The work presented in Chapter 5 is associated with the following publication:

- H. Yousef, J. L. Feuvre and A. Storelli, "ABR prediction using supervised learning algorithms," 2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP), Tampere, Finland, 2020, pp. 1-6.

In Chapter 6, we introduce an innovative ABR-aware prefetching technique that merges the outcomes of Chapters 4 and 5 to predict, while delivering the P2P segments to the player, the next qualities and try to fetch them from P2P. Besides, we propose an innovative ABR-controlling technique to take the cached qualities into account; especially when the ABR decides to down switch the quality while having the segments cached on a higher quality.

In Chapter 7, we summarize this work, and outline potential perspectives that could extend this work for future research.

Chapter 2

State of the art

In this chapter, we introduce the adaptive bitrate algorithms and provide an overview of the P2P streaming schemes. We start with a short recall of the existing adaptive bitrate algorithms, classified broadly into client-side, server-side, and network-assisted classes. Then we present a brief overview of the adaptive streaming systems in the P2P environments, in addition to Quality of Experience (QoE) and P2P related metrics.

Similar surveys are independently provided on ABR schemes by Kua et al. [17], Sani et al. [18] and Bentaleb et al. [19]. These surveys look at the adaptation bitrate algorithms and categorize them according to the location of the adaptation logic in the system and the feedback signal used.

Other surveys look at adaptive streaming via P2P networks, such as Shen et al. [5], Gheorghe et al. [20] and Theotokis et al. [21]. These surveys categorize the systems according to the P2P topology into tree-based, mesh-based, and hybrid tree-mesh-based P2P systems. We will follow a similar classification in our overview, and we focus on the mesh-based topology as it is more relevant to the work presented in the next chapters.

2.1 Adaptive bitrate schemes

Most state-of-the-art surveys categorize the adaptive bitrate schemes based on the adaptation module's location in the global streaming system. In a client-server architecture, the adaptation logic module can be implemented by either the client or the server entities, giving us the first two main client-side and server-side classes of algorithms. Besides, some works have insisted on providing the delivery network an active role in the adaptation logic, and hence, the third class is network-assisted ABR algorithms. A fourth class can be defined as a hybrid of these classes, and we will introduce these families of different algorithms in our review.

2.1.1 Client-side rate adaptation

Most of the ABR solutions reside at the client-side, which gives the client complete control on the bitrate selection process by measuring some metrics at the application or the network level, or even the cross-layer level (see Figure 2.1). The client-side ABR algorithms are classified broadly according to different criteria, but the most common classification follows the input metrics used by these algorithms. Hence, based on their input parameters, the client-side ABR algorithms can be classified into three main classes: Throughput-based, buffer-based, and hybrid adaptation algorithms. Client-side ABR solutions are privileged over other solutions due to their ease of implementation. They directly access the application-layer metrics, like the buffer occupancy, without any need for feedback from the server.

However, the main drawback of the client-driven nature of HAS solutions is that service providers may not have control over the client's behavior. There are many cases where the QoE can be affected [22]. For example, the HAS client may start the stream with unnecessarily low qualities until it gets bandwidth information after receiving the initial segments. Another example is when multiple HAS clients may

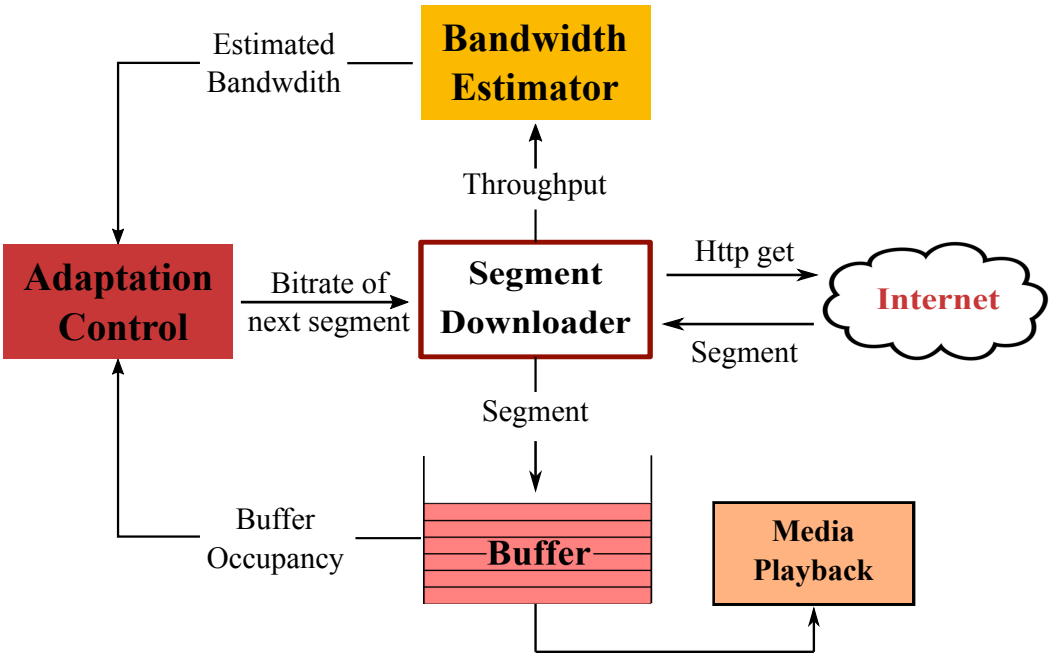


Figure 2.1: Client-side bitrate adaptation

compete for shared bandwidth, leading to undesired oscillations. Consequently, service providers may not be able to guarantee a premium QoE.

2.1.1.1 Throughput-based rate adaptation

This class of ABR algorithms relies on the estimated throughput as seen by the application layer in the adaption logic. Usually, it is measured by the size of the downloaded data over the time it took to be downloaded completely. Thus, it switches up the quality when it estimates higher throughput, and reversely it switches down the quality with decreased estimated throughput.

Liu et al. [23] presented a receiver-driven rate adaptation method for HTTP/TCP streaming that deploys a step-wise increase / aggressive decrease method to switch between bitrates. This algorithm aims to detect the bandwidth fluctuations using a smoothed network throughput based on the Segment Fetch Time (SFT), which measures the time starting from sending the HTTP GET request to receiving

the last byte of the segment. Later, Liu et al. [24] extended this work to sequential and parallel segment fetching methods in content distributed networks. They proposed a new metric which is the ratio of the Expected Segment Fetch Time (ESFT) and SFT, to detect network congestion and spare network capacity as fast as possible.

Conventional [25] is a throughput-based adaptation algorithm; it only uses the TCP throughput measurements over enough probes to decide on the next segment bitrate. It uses a four-step adaptation model starting with estimating, then smoothing, and then quantizing the bandwidth, and lastly scheduling the next segment.

Li et al. [25] proposed PANDA, which relies on the "Probe AND Adapt" principle. This significant algorithm uses only the TCP throughput measurement as long as it accurately indicates the fair-share bandwidth. PANDA determines a target average data rate and adapts the requested segment bitrate according to it. It constantly probes the bandwidth by increasing the target average data rate until it observes congestion, where it starts to back off. Similar to CONVENTIONAL, PANDA uses a four-step adaptation model starting with estimating, then smoothing, and then quantizing the bandwidth, and lastly, scheduling the next segment. Moreover, Panda uses a probing method similar to the TCP congestion control. It has an additive-increase-multiplicative-decrease (AIMD) [26], which makes it more effective in terms of network utilization and fairness as users will compete less aggressively for network resources.

Talking about fairness, Jiang et al. [27] introduced Festive (Fair, Efficient and Stable adapTIVE). Festive is a robust adaptive bitrate mechanism for chunk scheduling, bandwidth estimation, and bitrate selection in order to achieve the best trade-off between stability, fairness, and efficiency.

In another direction, as the HTTP-based video streaming is starting its transition from HTTP/1.1 to HTTP/2, many works have investigated HTTP/2 for video

streaming. Xiao et al. [28] proposed DASH2M (Dynamic Adaptive Streaming over HTTP/2 to Mobile Devices); an algorithm to address mobile users in the first place, aiming to optimize the mobile user's QoE while minimizing the resource consumption. DASH2M uses the most prominent two features for HTTP/2: Server push and Stream termination; it dynamically determines the number of segments for pushing based on the predicted available network resources and the impact of the user's early termination. DASH2M allows quality degradation to deal with network fluctuations in the middle of the push cycle. Besides, it utilizes the stream termination to start a new push cycle once the throughput prediction shows to be inaccurate.

In general, throughput-based rate adaptation lacks a reliable bandwidth estimation method, which makes the ABR select inappropriate qualities that lead to frequent buffer stalls.

2.1.1.2 Buffer-based rate adaptation

The family of buffer-based algorithms uses the playout buffer information to select the appropriate next video bitrate, avoiding the bandwidth estimation's inaccuracy at the application level.

By mapping the amount of buffered video to a specific bitrate, Huang et al. [29] proposed a pure buffer-based algorithm, called BBA-0, to maximize the received video bitrate and minimize the rebuffering events. Then they introduced BBA-1, which generalizes the design to mapping the buffer occupancy to the segment size since the buffer dynamics are more chunk size-dependent instead of the video rate. In addition, they proposed another version called BBA-2 that fills the buffer with a much higher rate than what the map suggests for the startup phase.

Similarly, Abuteir et al. [30] used a buffer occupancy model referred to as DAVS, where the buffer is divided into two areas (risky and safe) and the boundary between the two areas is determined through a dynamic threshold. In addition, DAVS

aims at minimizing quality switches by delaying the bitrate change decision for a window of time, which is also determined dynamically.

The buffer-based concept has also inspired Spiteri et al. [31], who came up with another well-known and significant algorithm called BOLA (Buffer Occupancy based Lyapunov Algorithm). BOLA is designed as an online learning problem based on Lyapunov optimization [32]. It formulates the bitrate adaptation as a utility maximization problem; the utility increases by increasing the average bitrate, whereas rebuffering decreases it.

Recently, the queuing theory came to light to design innovative adaptive streaming solutions. Yadav et al. [33] proposed QUETRA, a queuing theory-based approach that models the DASH client as an $M/D/1/K$ queue. QUETRA estimates the buffer occupancy to which the buffer would converge given a segment bitrate and the network throughput. Then, it selects the bitrate of the next segment to download so that the buffer occupancy converges to an ideal value; the ideal value of the buffer is determined by the closest value to the current buffer occupancy.

Later on, QUETRA has been extended to cover distributed video streaming use cases by a so-called DQ-DASH algorithm. BENTALEB et al. [34] used a $M^x/D/1/K$ queuing theory-based bitrate selection logic to achieve parallel downloading and benefit from the aggregated bandwidth from diverse multiple servers' resource. DQ-DASH checks the buffer occupancy at each scheduling batch and runs QUETRA to select the bitrate and requests one segment from each server, if possible.

In another work, Burger et al. [35] also used queuing theory to model the video buffer, but as a GI/GI/1 queue this time. With pq-policy and discrete-time analysis, the authors accurately evaluated the impact of network and video bitrate dynamics on the video playback quality.

In general, buffer-based rate adaptation suffers from instability, especially with

bandwidth fluctuations.

2.1.1.3 Hybrid rate adaptation

This class of rate adaptation schemes includes all the other algorithms that may use a mix of input metrics from different layers without being limited to network or application layers only.

SARA (Segment-Aware Rate Adaptation) [36] is another ABR algorithm that considers a mix of different input information. It uses the varying segment size, the estimated bandwidth, and the current buffer occupancy to predict the time needed to download the next segment. SARA divides the buffer into different ranges; each applies a specific decision among Fast Start, Additive increase, Aggressive switching, and Delayed Download.

Miller et al. [37], proposed a hybrid bitrate algorithm that combines the current buffer occupancy level, estimated throughput, and average bitrate of the different bitrate levels available at the MPD, as inputs to be used in the video rate selection. This algorithm changes its behavior dynamically based on the current buffer level. It aims to estimate the throughput accurately to improve QoE by avoiding throughput overestimation, quality oscillations, and playback interruptions.

Still in the direction of hybrid adaptation, with ABMA+, a lightweight adaptation algorithm proposed by Beben et al. [38]. ABMA+ uses the Segment Download Time (SDT) instead of measuring the throughput because they considered the download time a higher-level metric, reflecting the throughput variations as a segment size function. In addition to SDT, ABMA+ uses a buffer map to define the buffer capacity needed to satisfy the rebuffering threshold.

2.1.1.4 Control-based rate adaptation

Recently, the research in adaptive streaming has witnessed a shift towards machine learning and optimization control, where many works propose to formulate the bitrate selection as a learning problem.

Yin et al. [81] developed one of the most significant control-theoretic frameworks for bitrate adaptation, FastMPC, a Model Predictive Controller that benefits from both bandwidth and buffer size predictions to maximize QoE. FastMPC formulates the rate adaptation problem as a stochastic optimal control problem.

Other notable works, such as Pensieve [39] and Deep Q-Learning DASH (D-DASH) [40], use a mix of Deep Reinforcement Learning (Deep RL) techniques to enhance the user QoE, combining different input metrics like the throughput estimation and the buffer occupancy.

Online Convex Optimization (OCO) has also joined the family of these ABR schemes, where Karagkioules et al. introduced L2A (Learn to Adapt) [41]. L2A is model-free in the sense that it does not require any parameter tuning, modifications according to application type, or statistical assumptions for the channel. L2A models the HAS client by a learning agent, whose objective is to maximize the average video bitrate of a streaming session, subject to scheduling constraints of the buffer queue.

De Cicco et al. proposed ELASTIC [42], a client-side controller for dynamic adaptive streaming over HTTP designed using feedback control theory. ELASTIC uses one controller that uses a feedback linearization technique to select the appropriate bitrate that drives the buffer level to a certain set-point.

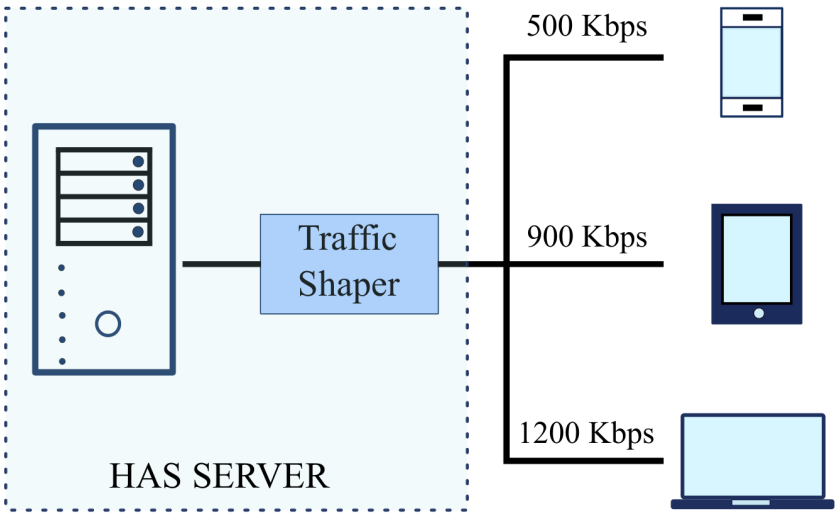


Figure 2.2: Server-side bitrate adaptation

2.1.2 Server-side rate adaptation

Server-side rate adaptation schemes use bitrate shaping methods at the server without any client's cooperation (see Figure 2.2), who may still make decisions. Yet, in this case, all the decisions are driven by the server's shaping methods.

Akhshabi et al. [43] deployed a traffic shaper to improve fairness and stability between clients competing on the same link. It eliminates the bandwidth overestimation that happens when some clients are downloading segments (ON phase) and others are in the steady-state, waiting to have some space in the buffer to request a new segment (off-phase). The server activates the bandwidth shaper when detecting that a video player is oscillating between different video bitrates. Then, the traffic shaper limits the throughput for each segment to the encoding rate of that segment. Thus, as long as the available bandwidth is higher than the shaping rate, the download duration will be roughly equal to the chunk duration. Consequently, the player will remain ON even when it operates in a steady state.

De cicco et al. [44] also contributed to the live streaming with a server-side

adaptation algorithm called Quality Adaptation Controller (QAC). QAC proposes a feedback control theory-based approach to control the buffer size. It aims at selecting the most appropriate bitrate for each client by keeping the playback buffer occupancy as stable as possible while matching bitrate level decisions with the available bandwidth.

Additionally, some recent works study the server-side rate adaptation problem for streaming tile-based adaptive 360-degree videos to multiple users when competing for transmission resources at the network bottleneck. For example, Zou et al. [45] and Liu et al. [46], formulated a fine-grained rate adaptation problem as Nonlinear Integer Programming (NIP) problem, which aims at maximizing the video quality and navigation smoothness for multiple users.

Overall, the main issue with most server-based rate adaptation schemes is scalability. The server needs to store and maintain the information for each client to perform bitrate adaptation, which means a higher overhead on the server-side and increased complexity, particularly when the number of clients increases. Furthermore, in most cases, they require modifications to the manifest or a custom server software to implement the rate adaptation logic, which may be inconsistent with the principle of DASH standard.

2.1.3 Network assisted rate adaptation

As mentioned, the main drawback of client-side HAS solutions is that service providers may not have control over the client's behavior. The network-assisted adaptive bitrate schemes solve this issue by allowing the clients to collect some metrics about the network conditions to improve the bitrate selection. To this end, an auxiliary unit (e.g., agent/ proxy) needs to be plugged into the network to efficiently monitor the network conditions and provide the network-level information to use network resources.

Houdaille and Gouache. [47] deployed some traffic shaping methods at the home gateway aiming to achieve stability and fairness between the multiple clients competing for the same bandwidth link. The home gateway is able to see and control all traffic coming into the home, and bandwidth can be allocated according to device roles and characteristics. Thus, the deployment of the traffic shaper at the home gateway can determine the desired bitrates for each client by each stream, then constrain the clients to stay within their limits.

Mok et al. presented QDASH [48], a system where the available bandwidth measurements are integrated into the video data probes with a measurement proxy architecture. QDASH includes two main modules QDASH-abw and QDASH-qoe to measure the bandwidth and help the client choose a suitable bitrate, respectively. QDASH avoids video oscillations by ensuring a gradual change in bitrate levels using integrated intermediate levels. Still, it may lead to network congestion as it generates significant overhead in the network, especially with increasing client numbers.

Similar work is presented by Bouten et al. [49] who introduced a QoE-driven in-network optimization system for adaptive video streaming to tackle the problem of multiple DASH clients competing for the available bandwidth. This proposed system deploys a set of proxy-like agents between the clients to measure and monitor the available bandwidth, determine the best bitrate for the following segments, and send all this information to the clients.

Sun et al. [50] proposed CS2P, a data-driven throughput prediction to improve the bitrate selection process. It uses clusters of similar sessions, an initial throughput predictor, and a Hidden-Markov-Model-based midstream predictor to model the stateful evolution of throughput. CS2P uses cross-session stateful prediction models that can be implemented into the bitrate selection logic of client- and server-side adaptation algorithms.

SAND[22] is another important contribution that proposes a control plane that offers asynchronous client-to-network, network-to-client, and network-to-network communications. SAND allows to collect metrics from different modules in the system and send them to the clients, servers, caches, and other network entities along the media path. A further study proposed by Jmal et al. [51] shows how the SAND architecture solves the problems raised by the existence of cache proxies in the delivery network, proposing a novel delivery scheme that combines both SAND and Content Centric Networks (CCN).

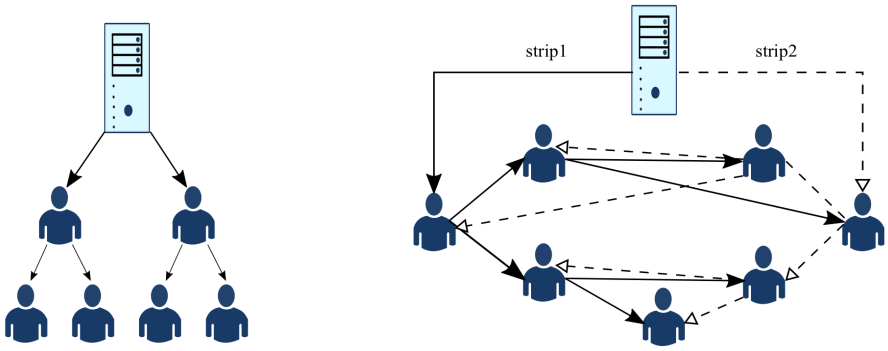
2.2 Adaptive streaming in P2P networks

2.2.1 P2P system architecture

P2P networks enable the peers/clients/end-users to exchange data without the need to fall to central servers, as long as the content is available internally. In a P2P system, peers act as both clients and servers. They participate in the system by sharing their resources with other peers to increase content availability and overall performance, making a P2P system a good candidate for scaling.

As stated in Chapter 1, P2P systems still have many weaknesses, such as peer-churn, which is the effect of peers uncontrolled dynamicity due to the sudden behavior of leaving or joining the system for reasons like network failures or user decisions, and heterogeneous resources among participating peers that can result in poor performance [52] [6] [7].

Before diving into the P2P video delivery details, we will quickly survey the existing P2P schemes. In the following survey, we refer to peers that upload video segments to other peers as seeders and the peers who receive those segments as leechers.



(a) Single Tree (b) Multiple trees

Figure 2.3: Tree-based schemes

2.2.1.1 Tree-based schemes

In tree-based architectures (see Figure 2.3a), such as ESM[53], the peers are organized in tree-like layers, where each layer has one seeder and multiple leechers. The seeders usually flow the data to the leechers in the tree's consecutive layers and organize their leechers' join/leave processes. Although the tree-based approach is simple and easy to control, it can be extremely affected by peer-churn, especially when intermediate seeders leave; streaming disruptions may happen due to a slow recovery of the streaming tree [5]. In addition, the received content quality is limited by the minimum upload bandwidth of the intermediate seeders since each leecher is connected to the seeder through a single tree branch only. Multiple tree architectures, such as in SplitStream[54], were proposed to tackle these issues by allowing each peer to be a seeder and a leecher in different trees. A simple example illustrating the basic approach of multiple trees is presented in Figure 2.3b. The original content is split into two stripes. An independent tree is constructed for each stripe such that a peer is a seeder in one tree and a leecher in the other. The tree-based schemes generate a lot of overhead in the network paths and require a lot of maintenance to minimize the tree depth.

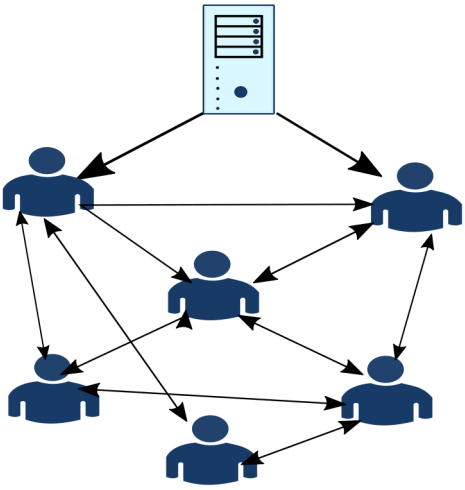


Figure 2.4: Mesh

2.2.1.2 Mesh-based schemes

In mesh-based topologies [55] [56], each peer may have information about all other peers in the system. Yet, even if there is a large size of peers, it only connects and exchanges data with a small number of peers, known as peer-pool. In this topology, peers are self-organized. They can be seeders and leechers simultaneously, facilitating peer-pool management and resiliency against random peers joining or leaving. However, the tradeoff is a higher network overhead due to exchanging more control messages within the peer pool.

Narada [57] is one of the pioneer mesh-based streaming systems. It presents end-system multicast to stream audio and video conferencing, from multiple senders to multiple receivers. It runs a distance-vector algorithm extended with path information on top of the mesh. Narada proposes a mechanism for group membership management that facilitates the self-organization of mesh in the case when any peer leaves the system.

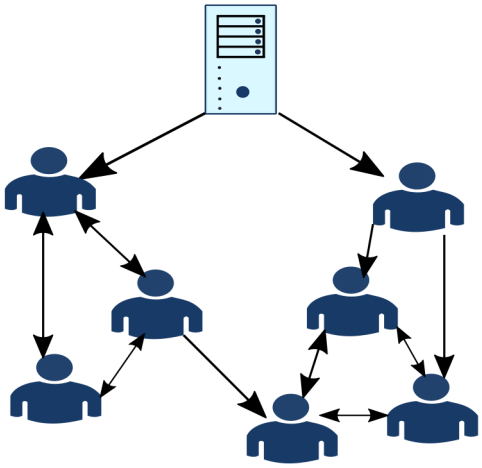


Figure 2.5: Hybrid tree-mesh

2.2.1.3 Hybrid tree-mesh based schemes

The hybrid tree-mesh-based schemes, like MultiPeerCast [58] and LayeredCast [59], inherit the benefits of both tree and mesh architectures. In this scheme, some children can get data from different parents (mesh structure), whereas other children can get the data from only one parent (tree structure). Usually, the peer activates the mesh connection when it can not get the data from the node parent in the above layer. Thus, each node periodically evaluates its sender peers, and in case of detecting some new best peers, it switches to the new sender peer. These schemes are more efficient in terms of tree-management, mesh membership-management, and peer-selection mechanism [60], although most of the hybrid schemes still face the tradeoff's complexity between stability and scalability [61].

2.2.2 Hybrid CDN/P2P systems

Both CDN and P2P delivery networks have their strengths and weaknesses. CDNs provide good service to end-users as long as the traffic load is within their provisioning limits. Thus, CDNs mainly suffer from scalbility, even popular sites and

providers can be overwhelmed by unexpected surges in demand and thus have to deny service to end-users. This scaling limitation becomes more relevant when users and content providers request higher qualitys video, implying higher costs for the CDNs [62]. P2P networks solve the edge scalability issue by leveraging the ressources to the participating users, while keeping the server requirements low. However, this good scalbility comes with the cost of other issues such as resource instability that users can join or leave at any moment, and resource heterogeneity in the sense that peers have heterogeneous upload and download capacities. This may lead to slow downloads, which in turn degrades QoE [6] [7].

Due to the complementary advantages of CDN being reliable and P2P being cheap and scalable, a system that combines both technologies can be highly beneficial [63] [62]. Such a system includes three main components:

1. The actual media server that distributes the video content to the clients.
2. A set of clients who are watching the same video content.
3. A tracker who finds the best peers by matching the clients who are watching the same video content, on the same quality level, if possible, and in adjacent time windows.

Xuening Liu et al. [62] presented LiveSky, a hybrid CDN-P2P system for effectively achieving the best scaling-reliability trade-off. They addressed the key challenges of integrating the P2P into the CDN and design an adaptive scaling mechanism to guide the hybrid CDN-P2P operation.

Ha et al. [64] proposed such a hybrid system for live video streaming over the Web to reduce the CDN usage as much as possible. In short, the video segments are portioned further into small chunks of roughly equal size. Besides the video player module, which consecutively requests video segments to fill the play-out buffer, clients also have an additional P2P module that prefetches the video

chunks ahead of time. However, the prefetching process may be incomplete due to the peers' heterogeneity. Thus, a cost-effective solution is to download the missing data from CDN. A similar architecture will be used in this work, as provided later in Chapter 3.

2.2.3 Adaptive bitrate in P2P networks

A lot of effort has been made to enable adaptive bitrate techniques in the presence of P2P networks. In this way, Mansy et al. [65] analyzed the challenges of the hybrid CDN-P2P adaptive streaming and proposed a stochastic fluid model to the hybrid streaming system with a single video bitrate, then extended the analysis to the adaptive streaming case with multiple video bitrates. They modeled the adaptive streaming as a linear optimization problem to obtain the best bitrate adaptation strategy.

Natali et al [66], designed another platform out of several swarms, where each of them streams a different bitrate. The clients in this platform are not entirely autonomous in selecting which version to download according to current network conditions and its device resources. Instead, a fourth new rate control strategy is implemented at the client-side to maintain a good viewing quality not to the client only but the P2P swarms as well.

Multiple adaptive streaming techniques have been proposed in P2P streaming systems by Jurca et al. [67]. Layered video encoding has been used to deliver different video layers to the clients adaptively. Multiple Description Coding (MDC) and network coding have also been used to propose adaptive streaming systems that support a large number of users [55].

Other works treated the field of pull-based P2P scalable Video Coding (SVC) streaming systems to deliver the best possible video quality by optimizing overlay structuring and data scheduling [68] [69].

Later in Chapter [70], we tackle the problem of adaptive streaming in prefetching based P2P environments. We show the effects of the local cache storage on the adaptation process, and we propose a solution to settle down the behavior.

2.3 QoE and P2P evaluation

2.3.1 Quality of Experience (QoE) Metrics

Quality of Experience (QoE) is a measure of the user's satisfaction with the video service. Recently, the advances of HAS-based technologies have led to a shift from the traditional QoE video measurements (e.g., Peak Signal-to-Noise Ratio) and user experience (e.g., subjective mean opinion scores) to more complex quality metrics (e.g., startup-delay, re-buffering time and frequency, average video rate, rate switching frequency, rate switching amplitude) and engagement-centric metrics (e.g., views per video, viewing duration) [17]. These metrics are often interdependent and have complex relationships, which makes optimizing HAS QoE a challenging task [71].

QoE is reported in [72] and [73] to be highly affected by the perceived video quality, the frequency of quality switches, and the video playback interruptions. Therefore, ABR algorithms aim to improve the QoE by trying to achieve the following objectives [74]:

1. Avoiding playback interruptions caused by buffer stalls.
2. Maximizing the video quality.
3. Minimizing the number of video quality switches.
4. Minimizing the start-up delay which is the time it takes the video to actually start playing, after the user requests it.

However, these objectives are contradicted and achieving them is a trade-off. For example, objective 2 contradicts 1, that it is possible to avoid the buffer under-runs by downloading the lowest quality. Also, objective 3 contradicts 2, that the ABR could adapt to the minor variations in the bandwidth to select the highest quality, which means more quality switches. Even objective 4 is a trade-off with objective 2, in the sense that the ABR could always select the lowest bitrate for the start-up phase to speed the playout start.

Thereby to meet the QoE specifications and achieve the best possible trade-off, many metrics were proposed in different related works [25] [38] [41] [75], to assess the performance of the ABR algorithms in terms of 1) average bitrate for the streaming session, 2) the stability of the session which represents how often the streaming quality changes, 3) the smoothness of the bitrate switches over the session, 4) the time spent on rebuffering and 5) the frequency of rebuffering. In the following chapters, we will use the metrics proposed in [41]; these metrics are referred to as 1) Average bitrate, 2) Stability, 3) Smoothness, 4) Consistency, and 5) Continuity.

2.3.2 P2P evaluation metrics

The performance of the P2P system highly depends on the use case and the design choice. For example, according to the P2P different typologies (tree-based, mesh-based, or hybrid), some works evaluated the performance using metrics such as propagation tree properties, the time a child peer needs to find a new parent, the efficiency of error control mechanism, peer joining complexity, maintenance overhead, and the peer workload represent the peer upload speed and how many requests he can handle from other peers [76].

Other works [76] [77] are more interested in metrics to measure Delivery Rate, which represents the segments that successfully arrive before their respective play-

back deadlines, and Request Window State (RWS), which represents the number of the successfully arrived segments per a scheduling window.

For hybrid CDN/P2P streaming solutions, approaches such as [64] [78] and [79] evaluate the load on the CDN by measuring the number of requests to CDN servers and the end-to-end delay; which is the time between sending a segment from the source node and playing it in the destination node. Also, these works measured the control overhead that represents the ratio between controlling messages, requesting messages, information exchange messages.

Data Overhead is another important metric in prefetching-based streaming systems as it expresses the resource waste in the network. In this sense, Bruneau-Queyreix et al. [80] defined bandwidth overhead as the percentage of data transiting on the network, which does not take part in the displayed content. In addition, Roverso et al. [14] used cache *hit* and cache *miss* ratios to measure the overhead. Their main metric, *traffic savings*, reports the percentage of the amount of data served from the P2P network from the total amount of data consumed by the peers.

The work of the following chapters is more relevant to hybrid CDN/P2P networks, in particular P2P systems that deploy prefetching techniques and deliver adaptive bitrate streams. Therefore, we introduce some metrics to evaluate the savings in the CDN requests in favor of P2P, besides reducing the P2P overhead related to ABR decisions, as we will see in Chapter 4.

2.4 Conclusion

This chapter presented an overview of the current ABR solutions, the P2P solutions for video delivery, and QoE and P2P evaluations. Based on their location in the system, ABR solutions are classified into client-side, server-side, and network-assisted

ABR algorithms. HAS solutions use different metrics from the application and/or the network level, such as buffer occupancy, throughput, and download time. P2P solutions work with CDN solutions to deliver the content to as many users at the best QoE, stability, scalability, and cost trade-off. The P2P solutions are classified based on the topology of peers into tree-based, mesh-based, and hybrid tree-mesh-based classes. The work of the following chapters is more relevant to client-side ABR solutions, as we try to make using these algorithms with hybrid CDN/mesh-based P2P solutions, and we use some of the presented QoE and P2P evaluation metrics to validate the following works.

Chapter 3

Methodologies for performance evaluation

3.1 Introduction

Experimental models, either simulations or real-time deployments, are fundamental to evaluate and validate any system's performance when designing any new feature before the deployment into the end products.

With the emergence of adaptive bitrate techniques, an abundance of ABR algorithms has been proposed. Later on, P2P networks have driven considerable attention to deliver the video content besides regular CDNs and having test tools to evaluate these systems soon became a persistent need.

The evaluation of video streaming sessions can be conducted through physical test benches. Although offering a high degree of real-life challenges, test benches are not time- or cost-efficient; besides, they are usually more challenging to maintain and do not provide a controlled environment in which experiments can be confidently re-conducted.

To cope with these issues, researchers started digging into lighter solutions

based on network simulators and emulators. One of the most critical requirements to validate a simulator is deploying the same conditions and reproducing the same results. Real-time emulators help in full testing the features under real-life conditions; however, it is still hard to get reproducible results precisely. In contrast, theoretical simulators provide complete control on the whole model offering reproducible results on the cost of testing under some real-life challenges.

Some ABR algorithms are already implemented in open-source projects. BOLA [31] and BBA [29] are implemented in dash.js[1] and GPAC[2] , ABMA+ [38] is implemented in VLC player [3] and GPAC, PANDA is implemented in GPAC... However, all these projects are used for real-life streaming, and they do not allow reproducible results for research purposes.

On the other hand, we found that most P2P simulators focus on simulating the P2P network only, without considering the application level, as is the case with ABR streaming.

Hence, the work of this chapter is divided into two main parts. We first start by attempting to build a realistic NS3-based platform to test adaptive streaming in the context of P2P networks. Then we demonstrate the challenges and the consequences of these deployments for research-oriented studies.

In the second part, we propose a simulation framework for adaptive streaming over hybrid CDN/P2P networks. This model provides reproducible experiments, comparable scenarios, and a scalable environment. It allows reusing existing real-life bandwidth traces for the down-links and the up-links. We also provide extensive logging functionality (statistics and visuals), necessary to analyze the behavior of adaptation algorithms and evaluate their performance.

[1] https://github.com/Dash-Industry-Forum/dash.js
[2] https://gpac.wp.imt.fr/
[3] https://www.videolan.org

3.1.1 Prior work

Video streaming technologies are evolving alongside different network capacities. Different platforms have been proposed to evaluate the video streaming technologies under different real-life and simulated networks.

OPNET[4] modeler provides different wireless networks which can be suitable for any real-time application, including video streaming. However, its usage in the literature is limited due to its high cost and design complexity.

OMNeT++[5] is a discrete event simulator that uses C++ and NED language. This simulator enables a large set of different network topologies and has an easy visualization tool. It also provides very good user support.

NS3[6], like OMNeT++, is a discrete event simulator that uses C++ and sometimes python. It enables different real-life implementations taken directly from the Linux kernel. Besides, it enables analyses with other network tools like Wireshark and provides good user support; however, it has a poor visualization tool.

In literature, many works have used NS3 for real-time network emulation. In [81] NS3 was used to study the real-time handover in the 3GPP LTE network environments. Also, authors in [82] used NS3 but in the context of Software Defined Networking (SDN) schemes and interference management. Besides, NS3 has also been used for video streaming applications. Authors in [83] presented an application of real-time video streaming over the NS3 based simulated LTE networks.

In the scope of HTTP adaptive streaming, NS3 was favored again for different applications. In [84], an HTTP Adaptive Streaming traffic generator framework was presented for mobile networks using NS3. Another traffic model was proposed in [85], where users built their traffic generator based on discrete events simulation through NS3. Ott et al. [86] proposed another simulation framework for HTTP-

[4] http://opnetprojects.com/opnet-network-simulator/
[5] https://omnetpp.org/
[6] https://www.nsnam.org/

based adaptive streaming applications. This framework has three main modules: a client module, adaptation algorithm module, and server module. It simulates a simple playback session by downloading the manifest file, requesting the next segments, running an adaptation algorithm module to get the next quality. All of these modules are integrated into the NS3 simulator, which supports many wireless and wired networks. This work was extended by Lyko et al. [87], who used NS3 to model live delivery of DASH and CMAF to enable traffic shaping between the client and the server.

Still in the scope of adaptive streaming, but for simulated models where few works proposed theoretical simulations to test ABR algorithms with different network traces. Sabre [88] is an open-source simulation environment for ABR algorithms. It is a Python tool that facilitates initial development and quick evaluation of algorithms in an environment similar to real production players. Sabre takes a video description, network trace, and an ABR algorithm as inputs and gives a collection of QoE metrics as output. Similarly, Pensieve[7], which is a system that generates ABR decisions using reinforcement learning, also provides, besides a real-time simulator, simulated environments to test and compare different ABR algorithms under different viewing conditions.

3.1.2 Contributions

The contributions of this work are the following:

- We introduce an initial NS3-based P2P platform to simulate hybrid CDN/P2P streaming, specifically for WebRTC on web browsers, using virtual LXCs as peers having differently shaped bandwidth profiles using NS3.
- We present a full Matlab-based simulation for hybrid CDN/P2P streaming,

[7] Available at https://github.com/hongzimao/pensieve

which we make open for research purposes[8].

The rest of this chapter is organized as follows. Section 3.2 presents the realistic NS3-based platform to test the adaptive streaming in a hybrid CDN/P2P environment. The simulation platform is detailed in Section 3.3 with a description of the different components. Then in Section 3.4, we show the experimental setup and the model evaluation with the different statistics and visualization tools.

3.2 NS3-based network platform

The performance of ABR logic is strongly dependent on the bandwidth conditions, and it would be useful to study the behavior of these algorithms through running a real-time video streaming session using different real bandwidth datasets. Interestingly, we found that a generic platform to test real-time P2P video streaming, specifically for WebRTC on web browsers, using real bandwidth datasets is missing. In this part, we focus on building a testbed that implements the following scenario: different users with different viewing conditions request video content from a CDN. In practice, Linux containers (LXC) are used to represent users. Each LXC runs a headless browser simulating a video session over hybrid P2P. In addition, each LXC runs the NS3 network simulator in real-time emulation mode; this mode changes the bandwidth of each user over time following the real recorded one in the used datasets. For the P2P network, we use the Mesh Delivery provided by STREAMROOT[9] technology which allows both live and VOD streaming, offering flexible CDN/P2P scaling for any audience.

[8] Arxiv paper to be published
[9] https://streamroot.io/cdn-mesh-delivery/

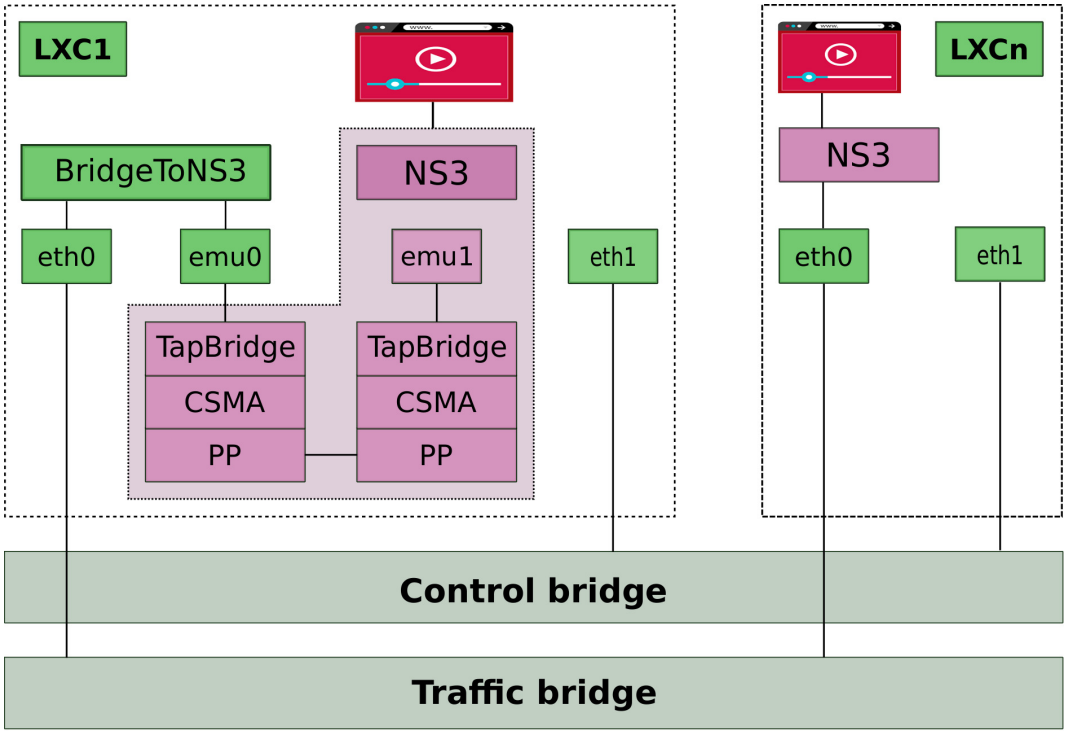


Figure 3.1: NS3-based testbed

3.2.1 Main components

3.2.1.1 Linux containers

LXC[10] is a userspace interface for the Linux kernel containment features. It lets Linux users easily create and manage system or application containers through a powerful API and simple tools. We created some containers, each running a video streaming session as if they were on real devices. The containers on their own can not send or receive any traffic without being linked to a real host. Therefore, each of these containers has its own network stack, and it talks to a network device named "eth0". They are connected to the host and other containers via these network devices to Linux bridges [89]. Linux bridges are virtual network bridges containing ports (interfaces) and a MAC learning database to decide on which ports the traffic

[10]https://linuxcontainers.org/

should be forwarded based on MAC addresses. These bridges create local area networks by combining network interface ports of a computer under a single bridge which forwards the packets from/to the containers. In this testbed, we use two Linux bridges: one called Traffic Bridge, used to forward the traffic in/out the NS3, and the other called Control Bridge, used to control the containers from outside and display the video content on the host GPU.

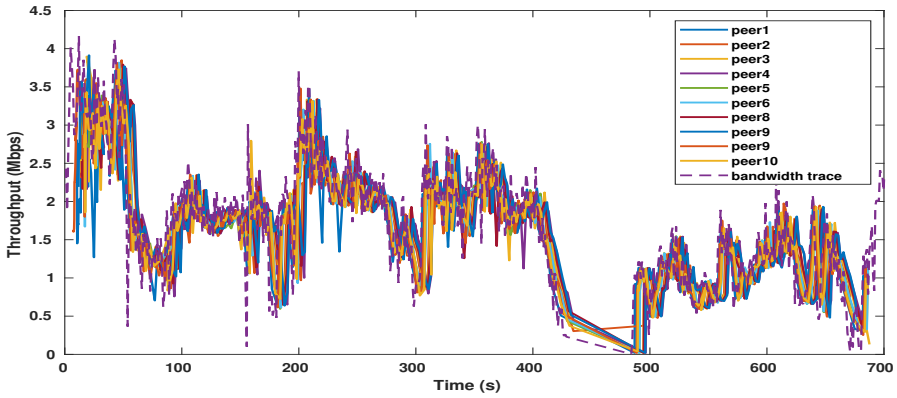
3.2.1.2 NS3 network simulator

NS3 is an open-source discrete-event network simulator available under the GNU GPLv2 license for research and development uses. We used NS3 to build the platform shown in Figure 3.1 which is composed of CSMA and point-to-point(PP) nodes, TapBridges, and virtual interfaces (emu). Tap bridges act as software bridges to connect NS3 nodes to the virtual interfaces. In Figure 3.1 we created two virtual interfaces: emu0 in the container and emu1 in NS3. These interfaces are bridged to NS3 nodes, so the traffic is forwarded inside NS3. We used CSMA nodes because the TapBridge can only be installed on the NS3 CSMA and Wi-Fi devices. The real traffic shaping is done in the PP channel, where it is possible to change the DataRate and Delay properties over time, simulating real varying bandwidth conditions.

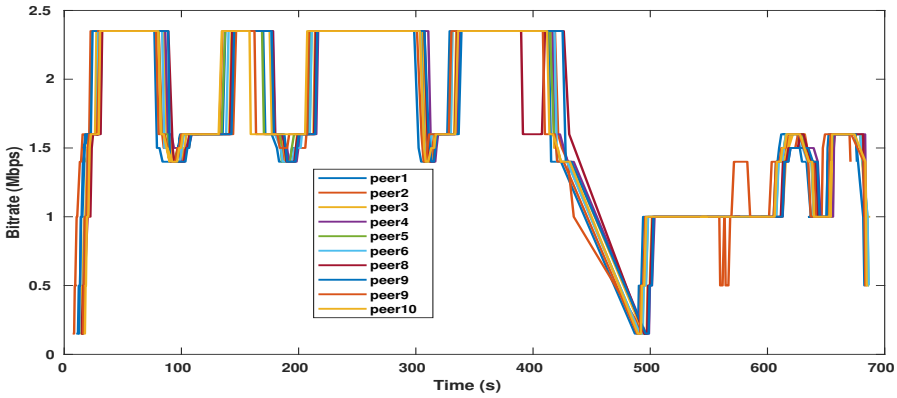
3.2.2 NS3 platform performance evaluation

For the tetbed evaluation, we tested different ABR algorithms BBA [29], BOLA [31], CONVENTIONAL [25] and PANDA [25] in a separate run. Each run involves 10 users, watching the same video content Big Buck Bunny with the bitrates 0.15, 0.5, 1, 1.4, 1.5, 1.6 and 2.45 (Mbps), and having the same bandwidth profile.

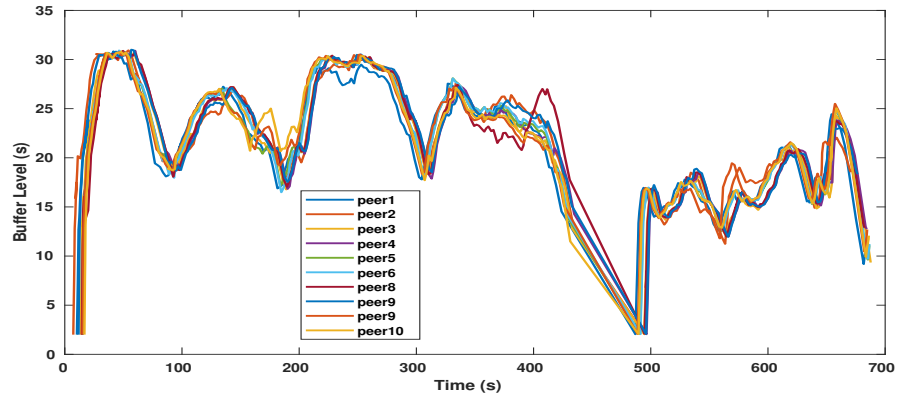
Figures 3.2a, 3.3a, 3.4a and 3.5a show that the testbed can successfully shape



(a) Measured bandwidth

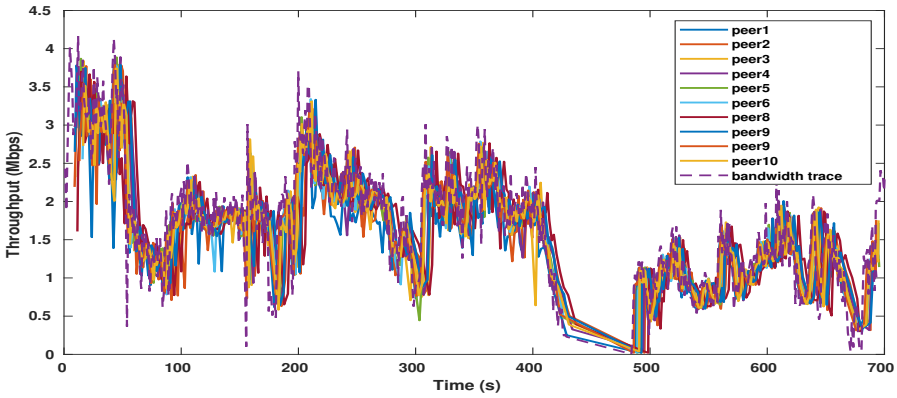


(b) Bitrate selection

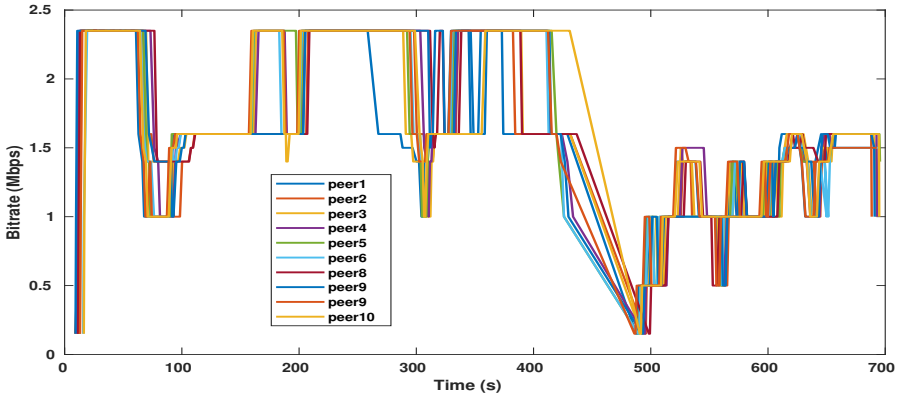


(c) Buffer level variations

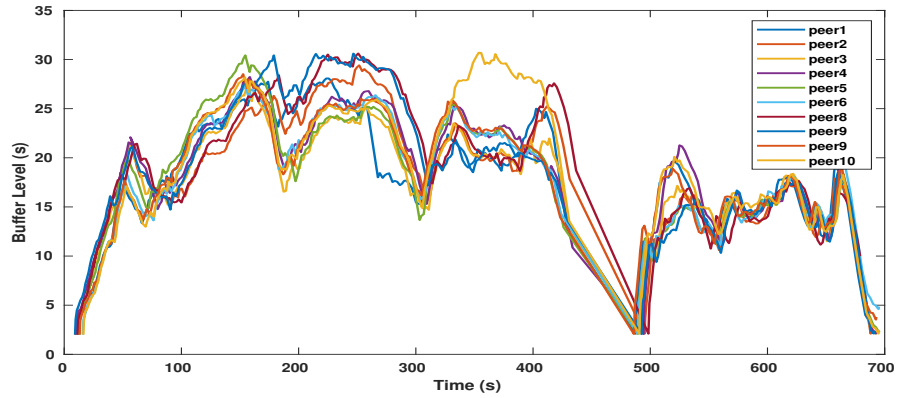
Figure 3.2: Testbed results for 10 peers running BBA algorithm



(a) Measured bandwidth

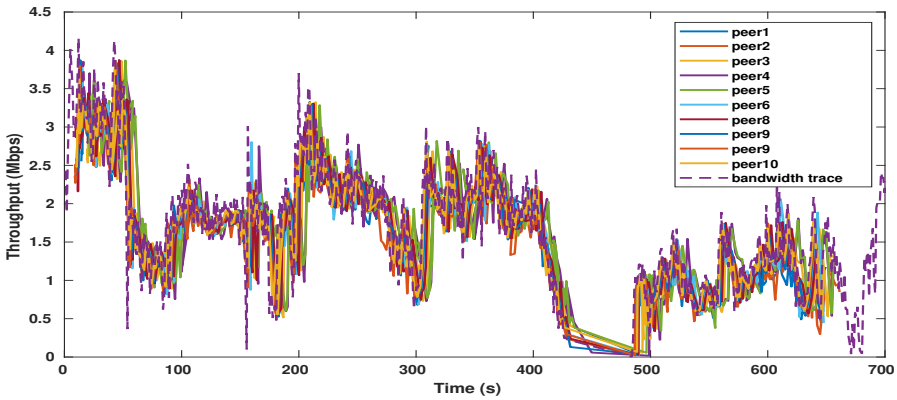


(b) Bitrate selection

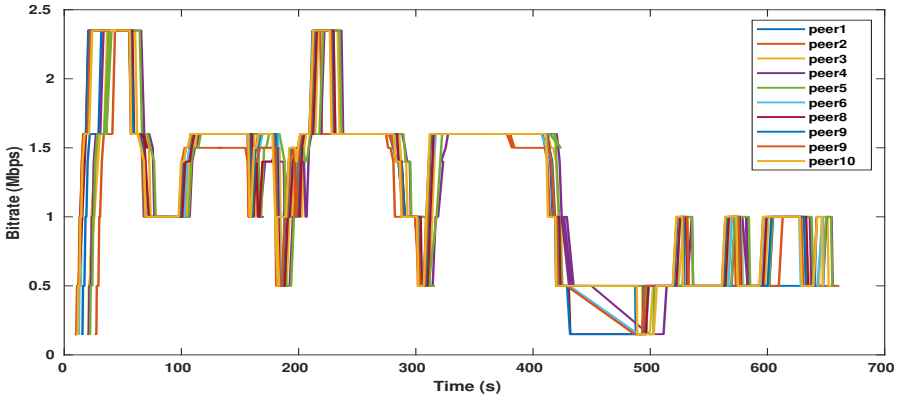


(c) Buffer level variations

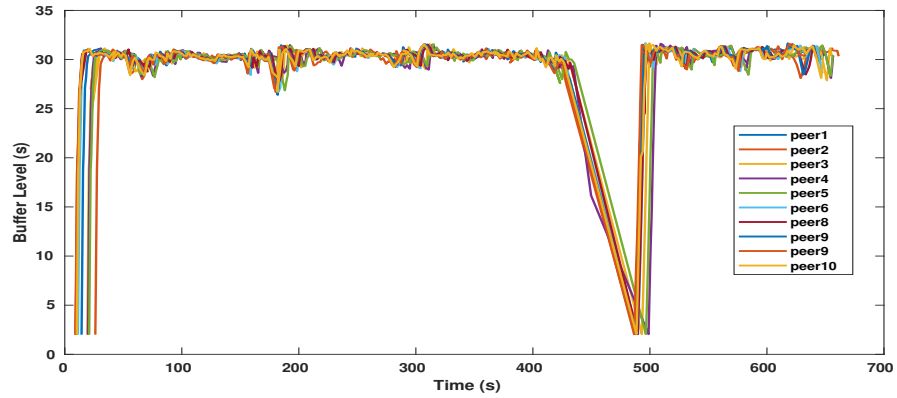
Figure 3.3: Testbed results for 10 peers running BOLA algorithm



(a) Measured bandwidth

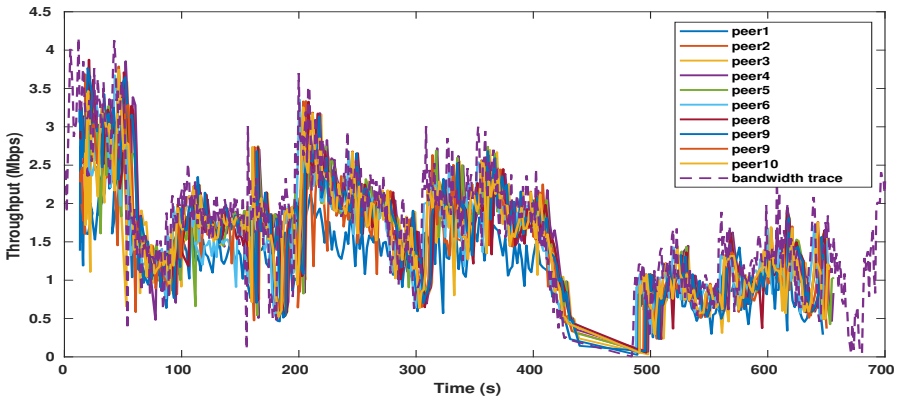


(b) Bitrate selection

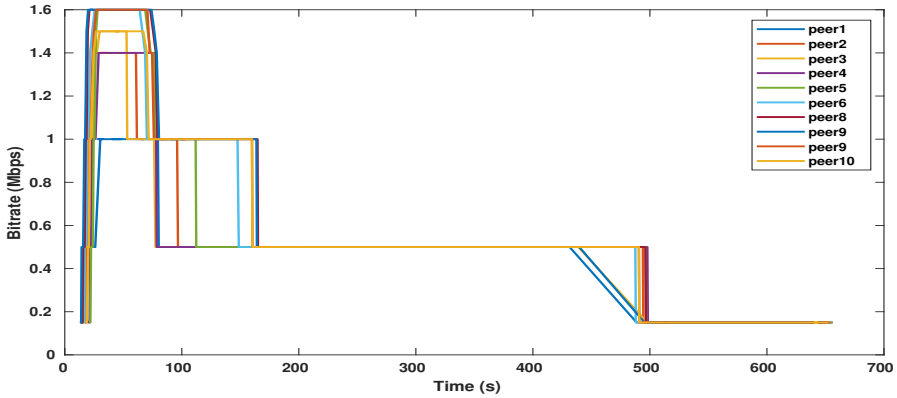


(c) Buffer level variations

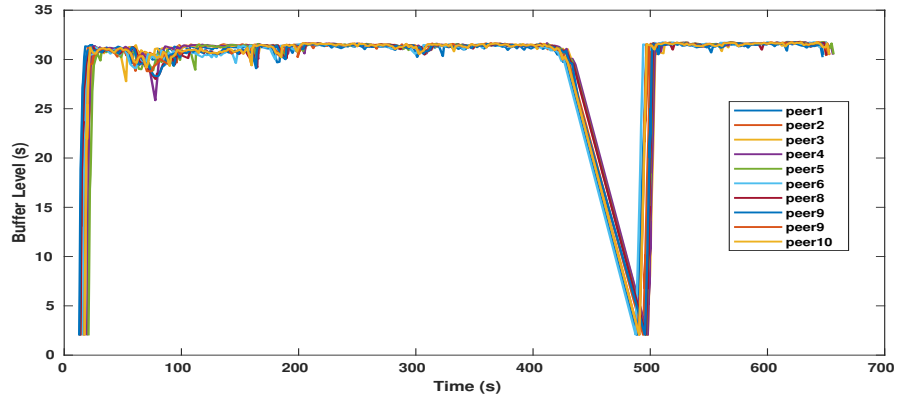
Figure 3.4: Testbed results for 10 peers running CONV algorithm



(a) Measured bandwidth



(b) Bitrate selection



(c) Buffer level variations

Figure 3.5: Testbed results for 10 peers running PANDA algorithm

the bandwidth of each peer over time to follow the realistic bandwidth profile; which makes it useful for traffic shaping and real-time video streaming scenarios.

However, when it comes to reproducing the exact results for comparing different scenarios, the testbed does not seem to be a good solution.

In Figures 3.2b, 3.3b, 3.4b and 3.5b we present the bitrate selection for the different peers for each ABR algorithm. These figures show that even when peers closely follow the same bandwidth profile, they do not behave the same in terms of ABR decisions. For example, for BBA and BOLA (buffer-based algorithms), when looking at the buffer variations graphs in Figures 3.2c and 3.3c we notice that peers have different buffer behavior, which leads to different bitrate decisions. CONVENTIONAL and PANDA do not guarantee the same bitrate decisions for all peers, as these algorithms follow the bandwidth variation, which is not exactly the same for all peers. In fact, the testbed's reproducibility is limited due to real-world constraints. Although it can throttle the bandwidth and follow a specific bandwidth trace, this throttling precision is likely subject to run-time conditions (memory, CPU usage, LXC and NS3 overhead), resulting in slightly varying throughput as observed. These slight variations have an impact on the HAS at short and middle term resulting in different HAS behaviors. Also, we have no control on the browser execution times, so some slight mismatches between trace start time and effective load time of web ABR client may further impact the HAS performance. For these reasons and the sake of reproducibility and comparison results, we choose to simulate the whole scenarios using MATLAB first and keep the testbed for future works.

3.3 Matlab-based simulator

The need for having a theoretical model is three-fold. First, it provides a full control on the environment and allows performing different scenarios changing only some

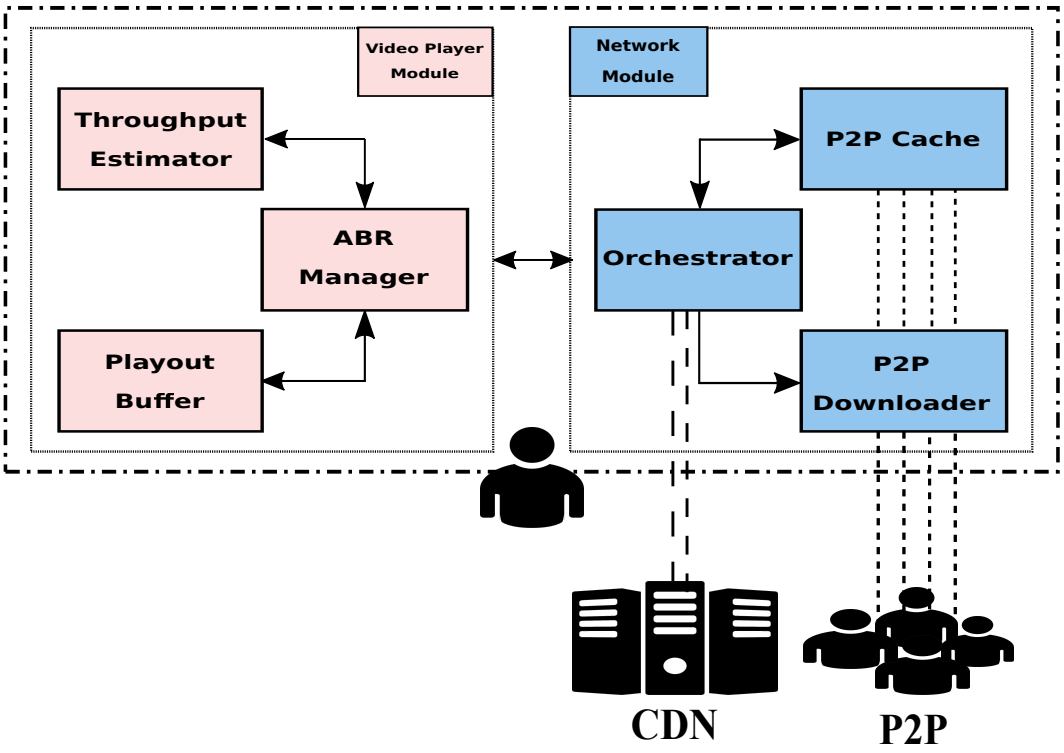


Figure 3.6: Hybrid CDN-P2P HTTP adaptive streaming

parameters. Second, it is completely reproducible, which is something crucial for comparison studies in research domains. Lastly, the simplicity of the implementation itself, which makes it flexible and easier to manipulate and maintain.

3.3.1 System Architecture

In this section, we provide an overview of how the model works, describing the workflow of each individual entity.

As shown in Figure 3.6, all peers use the same model. It includes two main modules: the video player module denoted as Media Engine in the rest of this chapter, and the network module, which is responsible for sending and receiving segments from CDN and/or P2P networks. In this model, we deliberately isolate both modules to be able to test any player logic over P2P network, without modifying

the ABR or player logic, to stay compatible with the layered HAS and P2P stack implementation as explained in Chapter 1. The model allows using different upload and download links; in this work, we simulate symmetrical connections, where both up-link and down-link follow the same bandwidth profile.

3.3.1.1 Media Engine

The video player requests the video segments in their playback order and fills the playout buffer, a simple pseudo-code of the used player logic is shown in Algorithm 3.1. It runs an ABR algorithm to decide on the bitrate r_n and the request scheduling time t_{rqst} of the next segment to download (line 3 of Algorithm 3.1). The request is then forwarded to the Orchestrator which delivers back the segment S_n after the download time t_{dw_n} (line 6 of Algorithm 3.1). Simply, the bandwidth is measured as the downloaded data over its download time and the simulation time t increases by this download time (lines 7 and 8 of Algorithm 3.1). The playout buffer Q_n consumes some data while downloading the segment, then grows by one segment duration once the segment is buffered (line 9 of Algorithm 3.1).

Algorithm 3.1 Video player logic

1: $initialize(Q_n, t_{dw_n}) \leftarrow 0$ for n =1
2: **for** *n* in [1, *N*] **do**
3: $(r[n]\ ,\ t_{rqst}) = ABR(Q_{n-1},\ t_{dw_{n-1}},\ bw_{n-1})$
4: $t = t + t_{rqst}$
5: $Q_n = Q_{n-1} - t_{rqst}$
6: $S_n = PeerAgent(n, r_n)$
7: $bw_n = S_n / t_{dw_n}$
8: $t = t + t_{dw_n}$
9: $Q_n = Q_n + \tau - t_{dw_n}$
10: **end for**

3.3.1.2 ABR Controller

The ABR algorithms take place in the ABR-CONTROLLER module. We implemented six client-side ABR algorithms (described in the Chapter2). However, the user may also add any other ABR algorithms as Matlab modules. Our model contains the following algorithms: BBA [29], BOLA[31] (with all the versions: bola-basic, bola-finite, bola-o and bola-e), PANDA and CONVENTIONAL [25] (denoted as CONV), ABMA [38] and Festive [27]. Usually, each ABR algorithm has two main outputs: the selected quality for the next segment and the segment scheduling time. Some of these algorithms require tuning parameters; we use the default recommended parameters in their documentation.

3.3.1.3 Network Module

The segment requests are forwarded to the network module, which is responsible for managing both CDN and P2P connections. This module performs a regular single client-server ABR streaming scenario where a client pulls all the video segments from only one server. It also enables the P2P mode, where clients can participate in the session. For the P2P mode, we deploy a prefetching-based technique as most of the current P2P networks do (as stated in 1). The Network Module is composed of three main sub-modules: Orchestrator, P2P Downloader, and P2P Cache.

3.3.1.3.1 Orchestrator

The Orchestrator, as indicated in Algorithm 3.2, receives the requests for the video segments at specific qualities. It then checks the segment availability in the P2P cache. These segments have three possible states: available and completed (AC), available but not completed (A$\overline{C}$) or not available ($\overline{A}$). If the requested segment is available and completed, the Orchestrator delivers it back in a very short time δ (line 4 of Algorithm 3.2), which is the time needed to fetch the segment from the

P2P cache. Otherwise, the Orchestrator sends CDN requests either to download only the missing range of data if the segment is not completed yet (lines 6,7 of Algorithm 3.2), or to download the whole segment if it is not available in the P2P cache (line 10 of Algorithm 3.2). It then delivers the segment to the video player right after time t_{cdn} which is the time it took to download the segment (or parts of it) from CDN (lines 8, 11 of Algorithm 3.2). In the case of using the model for a single client-server scenario, all the requested segments will have $\overline{A}$ state and will be fetched directly from CDN.

Algorithm 3.2 Orchestrator logic

1: $n \leftarrow$ the player requested segment
2: $st[n] \in \mathbb{S} : \mathbb{S} = \{AC, A\overline{C}, \overline{A}\} \leftarrow$ cached segment state
3: **if** $st[n] = AC$ **then**
4: $t_{dw} = \delta$
5: **else if** $st[n] = A\overline{C}$ **then**
6: $dataRange = S_n - downloadedData_n$
7: $sendCdnRequest(dataRange)$
8: $t_{dw} = \delta + t_{cdn}$
9: **else if** $st[n] = \overline{A}$ **then**
10: $sendCdnRequest(S[n])$
11: $t_{dw} = t_{cdn}$
12: **end if**
13: wait for t_{dw} to get and deliver the segment
14: **return** S_n

3.3.1.3.2 P2P Downloader

The P2P Downloader keeps downloading data from peers and filling the P2P cache. It divides the work into two processes: scheduling and fetching. In the scheduling process, for every time window w, the P2P Downloader handles three main tasks:

- Choosing segments to schedule first.
- Selecting seeders for these segments.
- Assigning chunks of data to each seeder.

To handle the first task, line 1 of algorithm 3.3, the P2P Downloader schedules a list of consecutive segments. These segments have the same quality level as the last requested segment by the video player, starting from the next segment right after the last requested segment. The scheduled segments should be either not scheduled before (so not available in the P2P cache $\overline{A}$), or already scheduled but partially downloaded during the previous scheduling round (A$\overline{C}$). The second task, which is peers selection (line 2 of algorithm 3.3), is handled by checking the peers who have the required segments and whose estimated upload capacity is high enough to download the segments. The last step in the scheduling phase, line3 of algorithm3.3, is assigning chunks of the segments to be played first to the best seeders.

Algorithm 3.3 P2P downloader logic

```
1: segmentsToFetch = updateSegmentsFetchingList(n,r[n]).
2: seeders = updateSeeders(segmentsToFetch).
3: chunks = assignDataToSeeders(seeders, segmentsToFetch)
4: sendP2PChunksRequests(chunks, seeders)
5: saveFetchedDataInP2PCache(FetchedData)
```

Furthermore, this module organizes the P2P mesh connections, like tracking the peers who have the segments then choosing the best seeders based on their upload capacity. It also keeps a list of the leechers that are connected to this peer at the same time. In this model, the peer upload link is shared equally between all the connected leechers, and when one leecher finishes its download earlier, we redistribute this free bandwidth to the remaining leechers.

3.3.1.3.3 Cache-manager

The last entity in the network model is the P2P cache memory. The peers assign this memory to store all the video segments they already downloaded, no

matter whether downloaded from P2P or CDN, or useful or overhead. Thus, for P2P connections, peers can exchange the content they already have in their local cache memories. The model sets this cache to store up to 200 MB of data, so it matches (somehow) the maximum cache size for web browsers. In addition, we use a FIFO (first in first out) list to organize the storing and cleaning of this local memory.

3.4 Experimental setup and evaluation

3.4.1 Input Data

Simulating any video streaming sessions requires two main inputs: the video content that will be watched by users and the bandwidth profiles to simulate some realistic viewing conditions.

3.4.1.1 Streaming Content

In this model, the video content is provided as a list of video segments encoded into different bitrates. The video content file has the syntax shown in Figure 3.7; the first row shows the provided average encoding bitrates, and every column represents the segment sizes (in Bytes) for each specific bitrate from the first row. Taking the first column in Figure 3.7 as example, the minimum bitrate is 45652 (bps), the first segment size is 13000 (B), the second segment size is 8200 (B) and so on. For the later evaluations, we will use some well-known open video sequences[11] that are recommended in the measurement guidelines of the DASH Industry Forum [90].

[11] available online at https://dash.itec.aau.at/download/

```
45652,  89283,  131087,  178351,  221600
13000,  25000,  36000,   48000,   59000
8200,   18000,  28000,   35000,   45000
11000,  22000,  34000,   47000,   60000
11000,  22000,  32000,   48000,   60000
12000,  24000,  35000,   49000,   60000
13000,  25000,  36000,   48000,   61000
9700,   20000,  31000,   40000,   56000
```

Figure 3.7: Sample of segments sizes

time interval (s)	bandwidth (bps)
0	4700000
55	4700000
55	3500000
55	2500000
55	1900000
55	1500000
55	900000

Figure 3.8: Sample of bandwidth profile

3.4.1.2 Bandwidth Profiles

The model is designed to easily use any bandwidth dataset that provides the time series and the according bandwidth measurements. Some publicly available 3G sets of real bandwidth traces are provided in [91]. We can also use any controlled trace for testing the performance under predicted conditions. An example of the used bandwidth trace is shown in Figure 3.8, where the first column is the measured time intervals, and the second column is the measured throughput. In this example, the bandwidth stays at 4.7(Kbps) for 55 seconds, then it switches to 3.5 (Kbps) and keeps this value for another 55 seconds, then it switches to 2.5 Kbps, and so on.

3.4.2 Statistics

We also keep a log for the whole session to ease the data processing. When the simulation completes, we keep the record of statistics for all the downloaded segments for each participating peer. Figure 3.9 provides the data we collected from each peer, where columns are:

- #1 idx: the segment index.
- #2 size: the segment size (bits).
- #3 duration: the segment duration (s).
- #4 bitrate: the average segment bitrate (bps).
- #5 the bitrate level, as provided by the HAS manifest.
- #6 req_time: the segment request time by the player (s).
- #7 res_time: the time at which the player received the segment (s).
- #8 req_buffer: the buffer level when the request was sent (s).
- #9 res_buffer: the buffer level when the segment was added to the buffer (s).
- #10 download_time: the segment download time (s).
- #11 downloaded: the amount of data downloaded for this segment (bits).
- #12 inter_request_time: the player waiting time before requesting this segment (s).
- #13 measured_bw: the measured bandwidth seen by the player (bps).
- #14 estimated_bw: the ABR estimated bandwidth (used in Panda and conventional) (bps).

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16
1	1208000	2	595491	6	0	0.6385	0	0	0.63857	1208000	0	1891702.590	0	0	1x1 struct
2	2960000	2	1546902	4	0.6385	1.8555	2	2	1.21695	2960000	0.6385	2432308.330	1891702.59	1891702.59	1x1 struct
3	3848000	2	1546902	4	1.8555	3.0283	4	2.8271	1.17280	3848000	1.2169	3281036.495	1933702.59	1900102.59	1x1 struct
4	3608000	2	1546902	4	3.0283	3.9267	4.8271	3.9287	0.89843	3608000	1.1728	4015872.509	1975702.59	1915222.59	1x1 struct
5	3936000	2	1546902	4	3.9267	4.9636	5.9287	4.8919	1.03684	3936000	0.8984	3796133.411	2017702.59	1935718.59	1x1 struct
6	3480000	2	1546902	4	4.9636	5.9531	6.8919	5.9024	0.98949	3480000	1.0368	3516953.790	2059702.59	1960515.39	1x2 struct
7	3360000	2	1546902	4	5.9531	7.1022	7.9024	6.7532	1.14918	3360000	0.9894	2923803.792	2101702.59	1988752.83	1x2 struct
8	4072000	2	1546902	4	7.1022	8.4581	8.7532	7.3973	1.35584	4072000	1.1491	3003293.933	2143702.59	2019742.78	1x2 struct

Figure 3.9: Example of the logs data

- #15 smoothed_bw: the ABR smoothed bandwidth (used in Panda and conventional) (bps).
- #16 cache: the cache state according to segment index, saved as an array of structures. Each structure represents statistics about the prefeteched segment version of that index (segment might be prefetched in different bitrates); Figures 3.10 details this information.

#1	#2	#3	#4	#5	#17	#18	#19	#20	#21	#22	#23	#24	#25	#26	#27
216	7480000	2	3078587	2	1076074	2.0827	0	0	false	1x1struct	0	406.59	408.6827	0	'p2p'
216	3792000	2	1546902	4	3425582	6.4998	0	0	false	1x1struct	0	413.99	420.4998	0	'p2p'
216	2248000	2	1032682	5	1727336	2.2466	520664	1.167	true	1x2struct	1	420.99	423.2465	419.617	'p2p&cdn'

Figure 3.10: Example of P2P logs data

Moreover, we record some vital P2P statistics during the session to enrich the performance evaluation of the P2P network. These statistics, which are recorded for every segment and saved in the P2P cache, give, in addition to the previous information, the following:

- #17 : p2p_downloaded: the data downloaded from P2P (bits).
- #18 p2p_duration: the fetch time from P2P (s).
- #19 cdn_downloaded: the data downloaded from CDN (s).
- #20 cdn_duration: the fetch time from CDN (s).
- #21 is_complete: the segment state in P2P cache (whether the download is complete or not).

- #22 seeders: contains a list of the seeders identifiers with the amount of data downloaded from each in bits.
- #23 is_useful: indicates if the media engine requested the segment or not.
- #24 req_fetch_time: the time at which the segment fetch request from peers was made (s).
- #25 fetched_time: the time the data is fetched from P2P (s).
- #26 req_time: the time at which the player requested the segment (s).
- #27 origin: indicates the source of the segment (CDN, P2P, or both)

An example of the stats is shown in Figure 3.10.

3.4.3 Visualisation

To better evaluate the system performance, simulators should provide some visualization tools that allow researchers to debug and validate their works.

The ABR-related works usually need some visuals to read the QoE metrics. The classic evaluation ways include graphs to follow the bitrate adaptation to the varying bandwidth conditions, in other words, the gain in terms of average bitrate and quality switches. Also, figures for the buffer evolution are valuable to follow the rebuffering events.

For P2P streaming, other graphs are needed to show the ABR's behavior when it receives P2P segments. Besides, some representatives of the P2P cache situation are also required to track the efficiency of the P2P usage.

Consequently, our model provides many scripts to visualize the generic ABR behavior for both CDN and hybrid CDN/P2P networks, besides the P2P evaluation.

First, we visualize the bitrate adaptation process of every ABR algorithm under the same network conditions using the same video content. For example, Figure

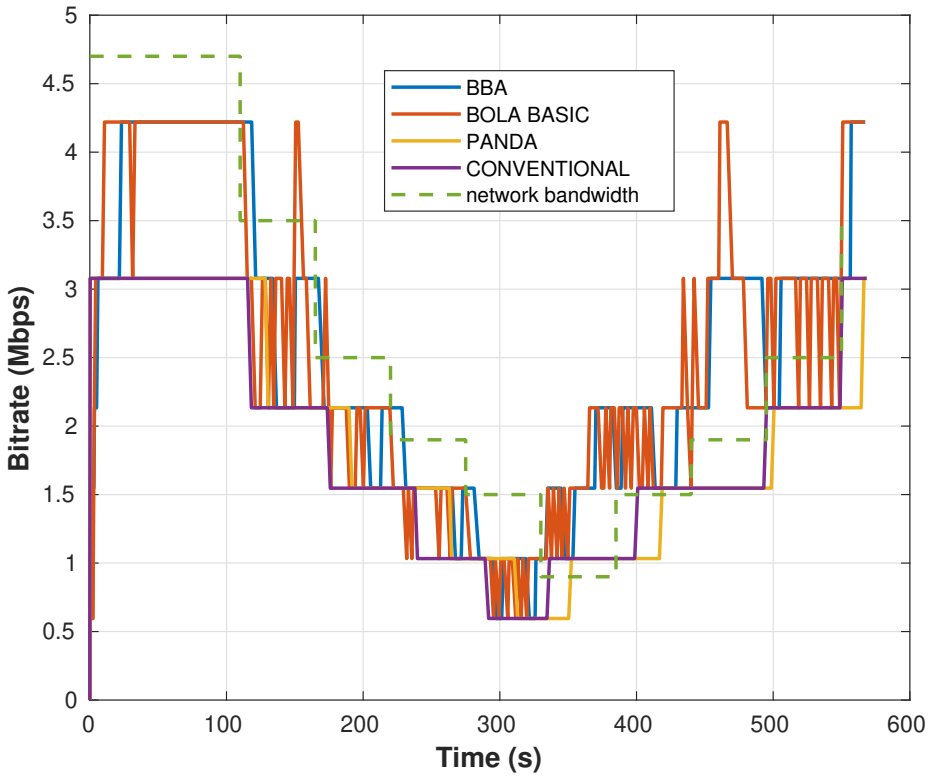


Figure 3.11: Example of the bitrate selection for the implemented ABR algorithms

3.11 allows comparing the behavior of four implemented algorithms (BBA, BOLA, PANDA and CONVENTIONAL) in terms of adaptability to the available bandwidth and the frequency of quality switches.

Furthermore, we provide details on the individual behavior of each ABR algorithm in the hybrid CDN/P2P networks, as can be seen in Figures 3.12a and 3.12b. These figures provide information about PANDA's behavior over the simulation time. In Figure 3.12a, we show on the same plot each of:

- the estimated bandwidth by PANDA,
- the measured bandwidth as seen by the player,
- the real CDN bandwidth as provided from the bandwidth trace,

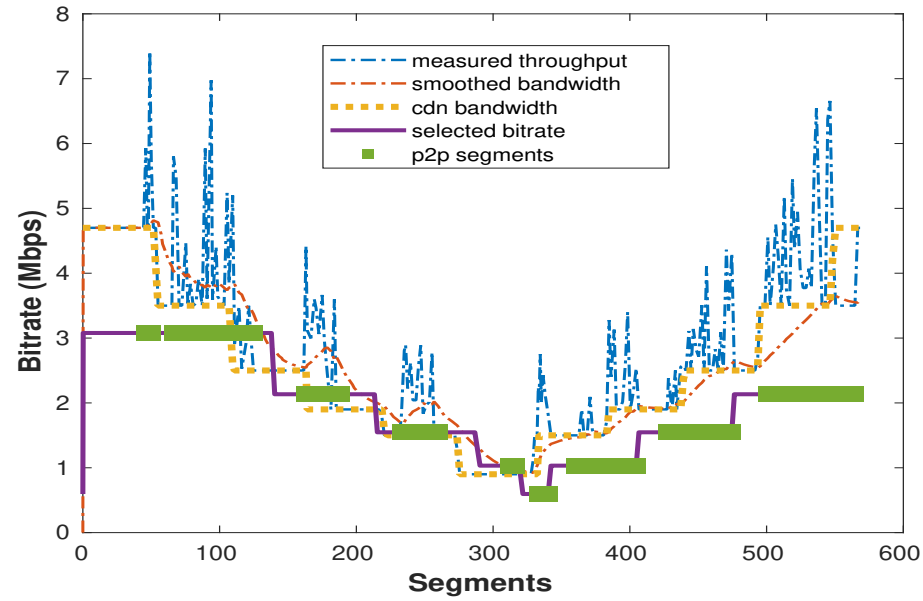
- the selected bitrates,
- the segments origin either from P2P or CDN.

In addition to the bandwidth information, we capture the buffer level variation over the simulation time, as seen in Figure 3.12b, and thus we cover most of the critical information to evaluate the ABR behavior.

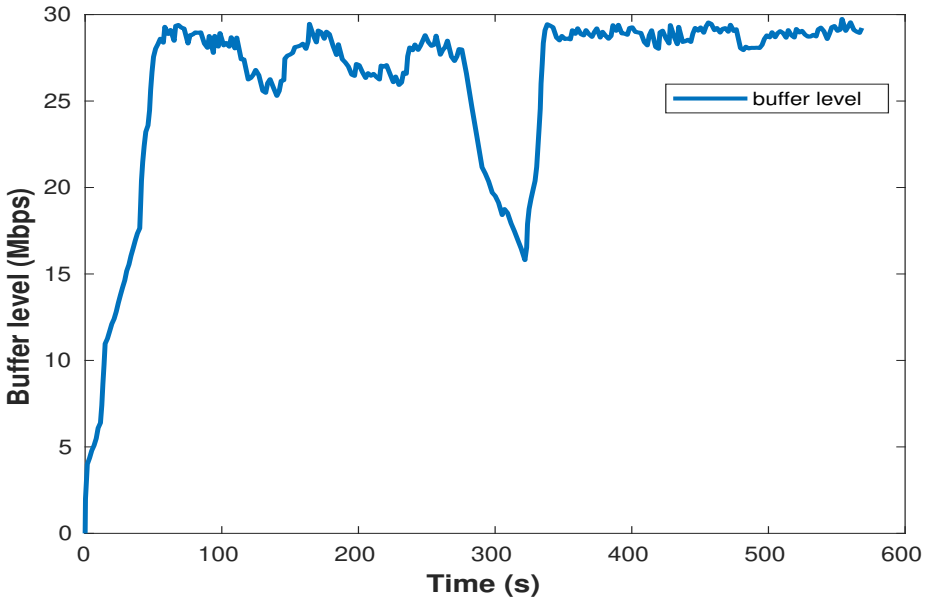
Next, we visualize the status of the P2P cache by providing two different graphs; one shows the overall data in the P2P cache, and the other shows per-segment status.

An example of the first plot is shown in Figure 3.13, where data for each cached segment is presented as a useful download from P2P or CDN, or overhead P2P data that has been prefetched in the P2P cache but not consumed by the video player.

To further evaluate the overall P2P efficiency, we introduce the second graph, shown in Figure 3.14, which details the status of segments in the P2P cache. Here, we show the different segments downloaded for each track/quality/level. Then for each segment, we state its origin, whether it is P2P, CDN, or both. The overhead is also shown on this plot but split into a useful and unuseful overhead. The useful overhead represents the P2P segments that the video player itself does not use but shares with other peers. In contrast, the unuseful overhead represents the P2P segments that are not used by either the peer or any other peers. The first overhead is labeled as unuseful P2P/CDN data and the second is labeled as useful for peers. Additionally, this figure serves in understanding the evaluation metrics that will be discussed in Chapter 4.



(a)



(b)

Figure 3.12: An example of stats visualisation: a) bitrates selection and bandwidth measurements, b) buffer levels for one peer running PANDA algorithm

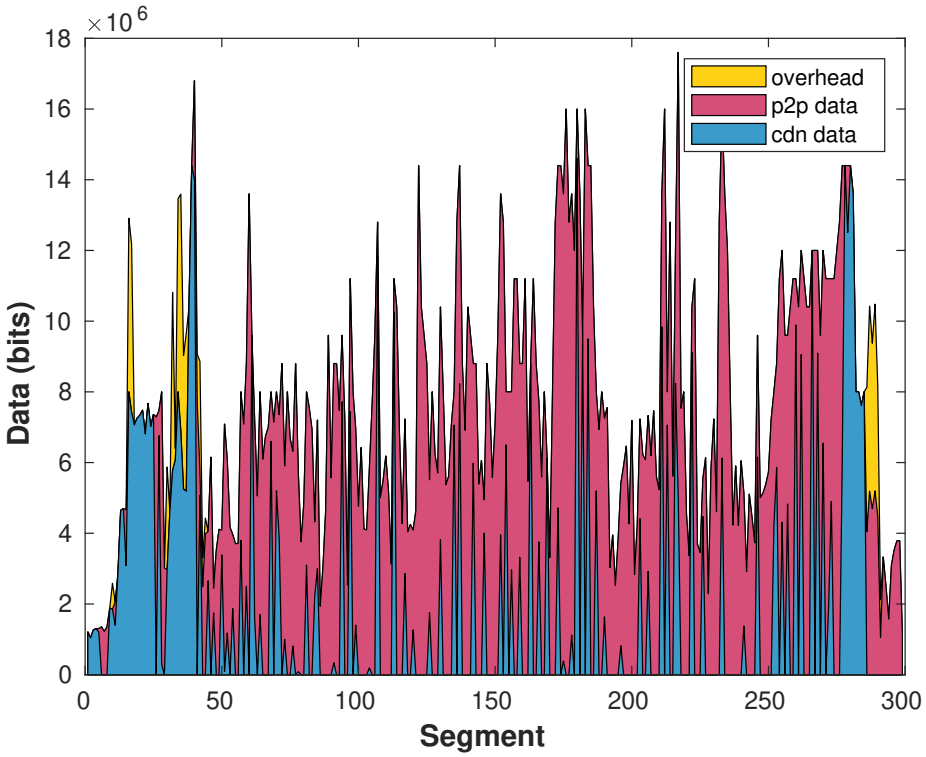


Figure 3.13: Example of the downloaded data from P2P and CDN

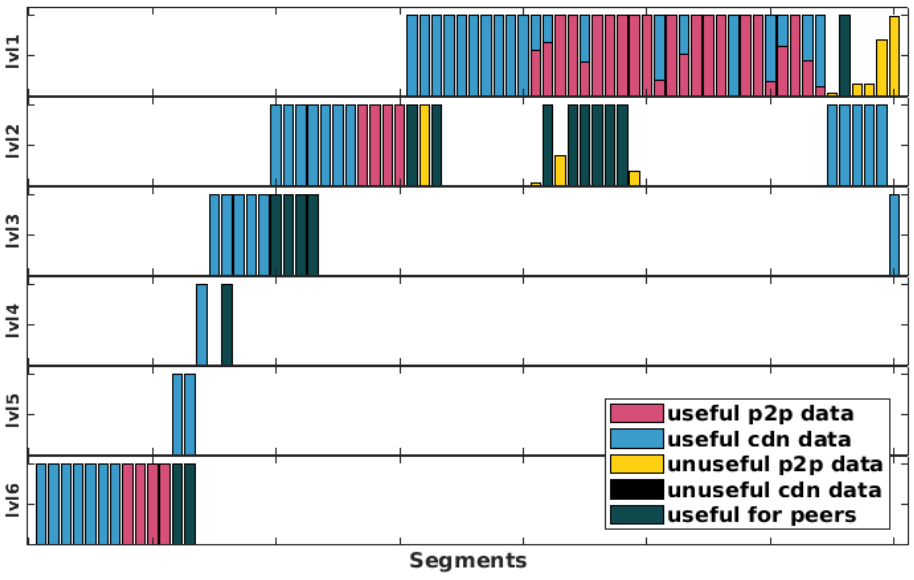


Figure 3.14: Status of P2P cache memory

3.5 Conclusion

We dedicated this chapter to discuss the challenges of testing the hybrid CDN/P2P environments for video streaming, especially for research purposes. In this light, we presented both realistic and simulated platforms to conduct the video streaming experiments. The real-time platform contains Linux containers LXCs, where each container represents a peer participating in the video session through a web browser. NS3 network emulator is then used to shape the traffic for each peer, while the connections to CDN and P2P are controlled using STREAMROOT mesh technology. Although this platform has shown some good results for traffic shaping on the CDN link, it could not precisely reproduce the same results over different experiments. Besides, this platform reuses an existing mesh P2P protocol and does not detail the implementation, which might be a requirement for researchers who desire to work on this platform to achieve P2P streaming.

For this reason, in the second part of the chapter, we provided a model to simulate adaptive video streaming scenarios over hybrid CDN/P2P networks. This model is useful for research as it provides: 1) accurate and reproducible results, 2) the ability to simulate a single client-server ABR session. 3) different ABR algorithms, 4) the P2P protocol, which allows flexible testing scenarios, and 5) many visualizing and statistical logs for better evaluations. We will use this model in the following chapters to validate our proposals.

Chapter 4

Enabling adaptive bitrate algorithms in hybrid CDN/P2P networks

4.1 Introduction

As a result of P2P and ABR key improvements, there have been many efforts to bring these two approaches together, which is where problems have popped up. Since in ABR, each client chooses each segment based on its current viewing conditions, different clients end up watching the video in different qualities, which makes the segments exchanging between users (peers) more challenging.

Interestingly, most of the existing ABR algorithms were designed to work in a server-based scenario, where the client requests segments directly from the CDN. However, when running these algorithms in P2P networks, all the estimated input information (throughput, segment download time, buffer value) will be dependent on the P2P connections of the client.

In this context, some related works implemented adaptive streaming in P2P

networks for both live [66] [14] and VoD [92] media streaming. However, as we mentioned in Chapter 1, no prior work focused on reusing the already existing HAS algorithms in P2P systems, nor focused on treating the local cache problems regarding the ABR.

While in Chapter 2 we classified the ABR solutions according to the module that implements the rate adaptation logic, we focus on the client-side schemes only in this chapter, as it is mostly the case with HAS solutions.

On the P2P part, though in Chapter 2 we presented an overview of the global architecture of the P2P-dependent streaming system, here in this chapter, we focus on the P2P mesh topology, which is easier for management and resilient against the random peers joining and leaving.

We study the behavior of four state-of-the-art ABR algorithms in P2P prefetching environment and comment on the main problems encountered. In particular, we analyze the performance of two different classes: BBA [29] and BOLA [31] from the buffer-based class and CONVENTIONAL and PANDA [25] from the throughput class. We also propose a new solution, called Response Delay, to make these algorithms compatible with the presence of the P2P local cache.

For the overall P2P evaluation, it should be noted that ABR streaming, in a prefetching-based hybrid CDN/P2P system, brings some overhead due to the inefficient usage of the resources. Here we differentiate between three types of overhead. The first is related to the ABR's instability; it happens when there are frequent quality switches for the next segments while having these segments already prefetched but into a different quality, hence not used by the player. The second overhead happens when a partial P2P segment is completed from CDN request, but P2P chunks for this segment are still received (no or late canceling of scheduled chunks). The third overhead is also related to the partial P2P segment, but in the case where CDN replies to the completion byte-range request with a larger

byte-range. This behavior can happen in realistic sessions, but we ignore it in our simulations since it is an artifact to deal with CDN rather than a consequence of the hybrid CDN/P2P system.

4.1.1 Challenges of ABR algorithms in P2P networks

Before putting the existing ABR algorithms and P2P networks together, we should first understand the possible consequences of such a merge. ABR algorithms have different natures according to their input parameters and how they process these parameters to make a bitrate decision. As discussed in section 2.1.1, ABR algorithms use many parameters, but the two leading ones are the available throughput and the buffer occupancy. With hybrid CDN/P2P systems, these two parameters are not dependent on the single-server connection only, but also on the P2P connection.

- In throughput-based ABR algorithms, the next download decisions are taken based on the most recent bandwidth estimation. However, the P2P network conditions vary a lot during the streaming session due to the high dynamics and heterogeneity of the peers. Also, cached (prefetched) P2P segments are delivered almost instantaneously, resulting in a very high bandwidth estimation by the ABR algorithm. As a result, such algorithms will end up selecting the highest bitrate for the next segments, which in turn makes the ABR more fragile to any upcoming bandwidth fluctuation, resulting in quality oscillations.
- The buffer-based ABR algorithms rely on the buffer occupancy to decide on the next bitrate to download. In P2P, the buffer filling rate will vary a lot, depending on the segment source speed (prefetched, from peers only, from CDN only, or from peers and CDN); this induces more frequent changes in buffer level, and thus, more quality switches.

- Finally, to the best of our knowledge, the prefetching-based P2P systems download the next segments on the same quality as the last requested one. With an ABR algorithm changing the current quality too often, the prefetched segments are more likely to be unused because not in the desired quality, thereby diminishing the P2P efficiency.

4.1.2 Contributions

In this chapter, we provide the following contributions:

- We address and analyze the main problems raised by using the existing HAS algorithms in the context of prefetching-based P2P networks.
- We design two methodologies to make HAS algorithms more efficient in P2P networks regardless of the ABR algorithm used, and without any change to the ABR logic.
- Additionally, we introduce two new metrics to quantify the P2P efficiency for ABR delivery over P2P. These two metrics are designed for the P2P overhead in the first place.

The remaining of this chapter is organized as follows. Section 4.2 provides the proposed methodologies to enable the work of ABR algorithms in P2P networks. Section 4.3 presents the experimental evaluation and the new metrics. The methodologies are evaluated in Section 4.4, under simulation and realistic scenarios.

4.2 Proposed solution: Response-Delay

In prefetching-based P2P systems, most of the ABR algorithms get confused by the fast delivery of the video segments when they are downloaded and saved locally

ahead of time. This issue can be addressed by making the ABR believe that such segments were downloaded from CDN. Unfortunately, this requires tracking the CDN bandwidth variation by the actual downloading of the video segments from CDN, which is not the case when the P2P network is active. In this chapter, we show that this issue, regardless of the type of ABR algorithm, can be solved by adding the right delay to the responses before delivering the prefetched segments.

4.2.1 Principle

As stated in the Chapter 1, the main goal of our design is to make the video player agnostic of the P2P network without touching the ABR or the video player logic itself. To this end, we only influence the algorithm by adding a delay before returning the requested segment. From the player point of view, this additional delay will be interpreted as a longer download time of the segment (and by modulating it, we change the effective download time as seen by the ABR logic). Acting as a replacement for the HTTP stack of the player, Response-Delay has the advantage of being generic to work with any video player and most ABR algorithms.

Response-Delay aims at preventing the undesired network variation feedback, which leads to uncontrolled ABR decisions. Our goal then is to add a delay that should:

1. Improve the playout continuity by eliminating the re-buffering events resulting from the wrong P2P speed estimation.
2. Enhance the stability and the playback smoothness through reducing the number and the amplitude of the quality switches when P2P is applied.
3. Keep a reasonably high average quality.
4. Use the P2P resources efficiently by increasing the P2P percentage and reducing the number of unused P2P downloads due to prefetching.

However, choosing the appropriate response delay is a question of compromise. On one hand, too short delays make the ABR switch to high qualities that the available bandwidth cannot support, leading to further rebuffering events. On the other hand, too long delays can starve the player buffer and lead to undesirable rebuffering events. It also makes the ABR falsely believe the available bandwidth is low and switch to lower content qualities.

4.2.2 Response-Delay proposals

In this section, we present our two methodologies for response delay: a buffer-based and a network-based, inspired by the two main classes of ABR algorithms.

4.2.2.1 Buffer-delay map (BufDel)

We first present BufDel, a buffer-based method inspired by the buffer-based ABR, in which the bitrate is selected from lowest to highest as the buffer increases from low level to the maximum level. Similarly, BufDel uses a continuous function to change the delay as a function of the buffer occupancy. It calculates a delay $d = f(Q)$ as shown in Figure 4.1. This delay accelerates the buffer filling with segments when the buffer level is low, whereas it decelerates when the buffer level grows to reach the maximum target. As previously discussed, the delay should be bounded to avoid having too long or too short delays, therefore $d_n = f(Q_n) : Dmin < d_n < Dmax$ and $Qmin < Q_n < Qmax$. The boundaries are chosen to be $Dmin > 0$ and $Dmax$ according to (4.4), where $s_{n_{p2p}}$ is the downloaded data from P2P and r_n is the segment average bitrate. We chose τ, the segment durtion, for the delay upper bound as long delays are undesired and may cause rebuffering, or even lead the

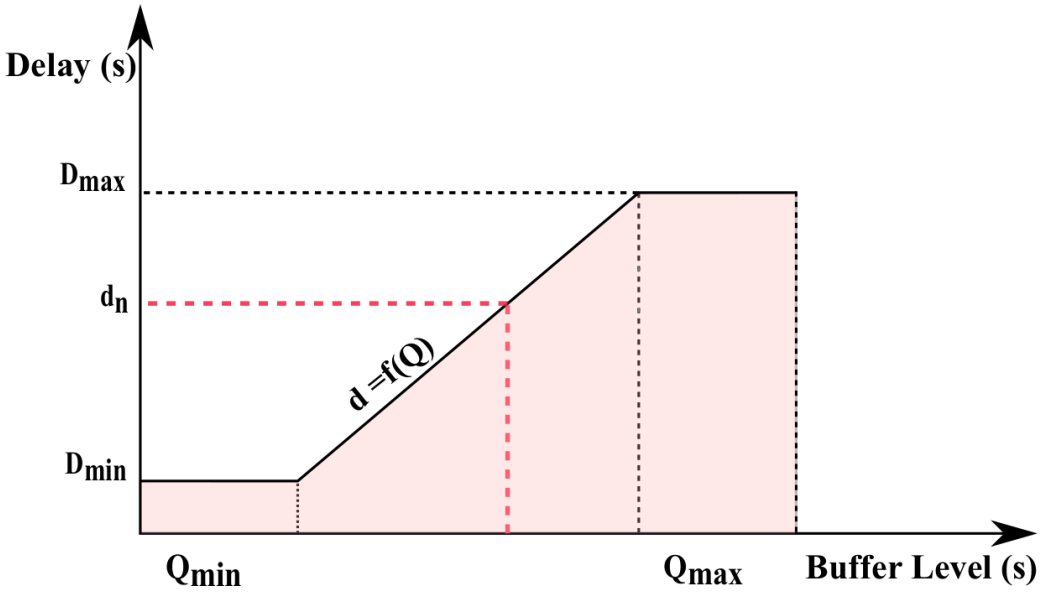


Figure 4.1: The buffer-delay map used for response delay

ABR to quality down switches.

$$D_{max} = \frac{s_{n_{p2p}}}{r_n} \leq \tau \tag{4.1}$$

In addition, BufDel needs to monitor the player's buffer level, which in Web environment is done by monitoring the <video> element without modifying the video player.

4.2.2.2 Network delay (NetDel)

Our second approach, NetDel, is a network-based method calculating the delay as a function of the available measured bandwidth. To do so, NetDel must take both CDN and P2P contributions into account. P2P and CDN measurements cannot be aggregated because CDN and P2P traffic are generally not simultaneous; in our model, CDN is only used when P2P cannot support the selected bitrate. In (4.2), we consider the highest bandwidth between CDN and P2P, which would be

the most appropriate way to determine the available bandwidth for several reasons. First, it has the advantage of being more resilient to obsolete measurements of CDN bandwidth. Second, the measurements of CDN bandwidth are not taken into account in cases of small byte-range requests, as they are usually unstable and inaccurate in that situation. And lastly, this approach works as a CDN/P2P link switcher targeting the highest measured bandwidth; thus, driving the ABR to pick a higher quality that can be sustained via the highest bandwidth.

$$targetBw = \max(bw_{cdn}, bw_{p2p}). \tag{4.2}$$

We show how to calculate the delay in (4.3). The segments with different sizes will be delivered with different delays to ensure that the ABR can detect that, for the same quality, the bigger segments need more time to be downloaded than shorter ones. Furthermore, we upper bound the segment delay to the segment duration; if not doing so, undesired playback pausing or emergency track switches may occur while waiting for the segment to be delivered.

$$d_n = \frac{s_{n_{p2p}}}{targetBw} \leq \tau \tag{4.3}$$

$$d_n = \frac{D_{max} - D_{min}}{B_{max} - Bmin} * (Q_n - Q_{min}) + D_{min} \tag{4.4}$$

There are cases where the last P2P and CDN throughput measurements are lower than the bitrate of the next prefetched segment. In this case, and according to Equation 4.2), the ABR may underestimate the bandwidth and, consequently, down switch the current quality. If the next segment is prefetched in higher quality, it is preferable to prevent the quality down switch and try to keep requesting the same

quality by controlling the response delay to (4.5).

$$d_n = \frac{s_{n_{p2p}}}{\max(targetBw, r_n)} \tag{4.5}$$

4.2.3 Applying Response-Delay

This section discusses when to use the Response-Delay, precisely, which segments should be delayed. We modify the logic presented in Algorithm 3.2 to be compatible with response delay, as shown in Algorithm 4.1.

Algorithm 4.1 Orchestrator logic with response delay

$n \leftarrow$ the player requested segment
$st[n] \in \mathbb{S} : \mathbb{S} = \{AC, A\overline{C}, \overline{A}\} \leftarrow$ cached segment state
if $st[n] = AC$ **then**
  if time needed before delivering the segment **then**
    $t_{dw} = d_n$ ▷ Response-Delay
  else
    $t_{dw} = \delta$
  end if
else if $st[n] = A\overline{C}$ **then**
  $dataRange = S_n - downloadedData_n$
  $sendCdnRequest(dataRange)$
  if time needed before delivering the segment **then**
    $t_{dw} = t_{cdn} + d_n$ ▷ Response-Delay
  else
    $t_{dw} = \delta + t_{cdn}$
  end if
else if $st[n] = \overline{A}$ **then**
  $sendCdnRequest(S_n)$
  $t_{dw} = t_{cdn}$
end if
wait for t_{dw} to get and deliver the segment
return S_n

First, from Chapter 3, we recall some notations about the availability of the segment in the P2P cache. These segments have three possible states: available and completed (AC), available but not completed (A$\overline{C}$) or not available ($\overline{A}$). Obviously,

all the AC should be delayed before being sent to the player (line 5 in Algorithm 4.1). Similarly, CDN segments (nothing prefetched) are already delayed by the speed of CDN connection and will be delivered immediately once they are downloaded (line 11 in Algorithm 4.1). But hybrid segments, which are downloaded from both CDN and P2P, are handled differently (line 13 in Algorithm 4.1). Peer-Agent will wait for a time t_{cdn} to have these segments completed before delaying the P2P part with the time d_n.

4.3 EXPERIMENTAL EVALUATION

4.3.1 Experimental Setup

To evaluate the performance, we use the model described earlier in Section 3.3.1. We simulate one peer pool of 10 peers, which is a good representative of the actual peer pool size for mesh-based systems. The peers exchange the same video content of 300 segments of 2-second length each and encoded in 6 different bitrates: 0.59, 1.032, 1.54, 2.13, 3.078, and 4.219 (Mbps). The client model includes a video player, which consecutively requests the video segments according to Algorithm 3.1. For the player parameters: the maximum buffer capacity is set to 30 seconds, the playback rate is set to one (nominal playback speed). Both of the playback startup and the re-buffering thresholds are set to one video segment. The peers join the video session successively every 15 seconds. The first peer starts the session by connecting to the CDN, then the other participating peers connecting to both CDN and P2P links. To guarantee a fair comparison, we set all peers to experience the same upload and download bandwidth variations and initiate the session from the same starting point (identical network throughput at first segment request). Later in Section 4.4.5, we evaluate our proposal using some publicly available 3G sets of real bandwidth traces [91]. These traces have been used a lot in

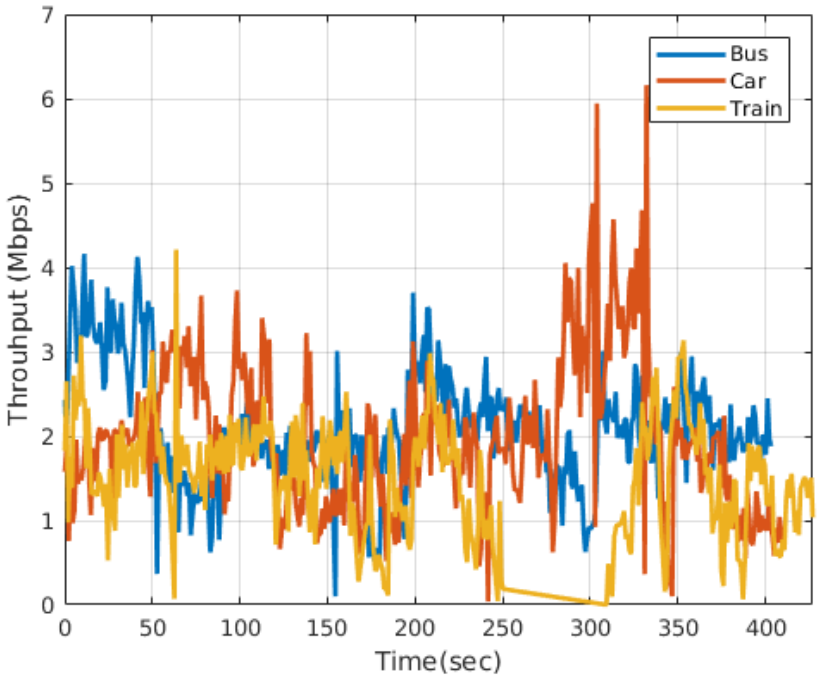


Figure 4.2: Example of the selected bandwidth profiles

the literature to study and compare the used ABR algorithms. Figure 4.2, shows an example of the used traces, selected to show the normal bandwidth variations and fewer outages duration, corresponding to direct throughput measurements from a bus, a train, and a car. Also, to illustrate some different ABR challenges, we use some controlled bandwidth traces, as later shown in sections 4.4.1 to 4.4.4.

4.3.2 Evaluation Metrics

In the following evaluation, we rely on some QoE metrics, presented in Chapter 2. These metrics are average bitrate, stability, smoothness, continuity, and consistency of the streaming.

This whole work pertaining to the P2P systems, we must complement the QoE evaluation with the overall P2P system performance in terms of the efficient usage

of the P2P and CDN resources. Indeed, P2P systems should not be considered good systems unless they can reduce the CDN requests and the P2P overhead as much as possible. In order to measure this, we introduce three new metrics: P2P Offload, Peer Efficiency, and Peer Pool Efficiency. We first introduce the metric *P2P Offload*, as calculated in (4.6), which indicates the cost-effectiveness in terms of the CDN usage; i.e., the less data downloaded from CDN, the better P2P offload.

$$P2POffload = 1 - \frac{1}{N}\sum_{n=1}^{N}\frac{s[n]_{cdn}}{S[n]} \tag{4.6}$$

As previously mentioned, the inaccurate prefetching process results in inefficient usage of P2P resources. P2P segments, denoted as K, may be prefetched in m different qualities. However, only one quality will be useful and requested by the player, while the other $m-1$ qualities are overhead and useless for the peer. Nevertheless, some of these unused segments might be requested by other peers, and the rest is not used by any of the peers. We denote $s[k]_{p2p}$ the P2P data of the useful quality, $s[k]_{\overline{p2p}}$ the P2P data of all the useless qualities for one peer that other peers use and $s[k]_{\overline{\overline{p2p}}}$ the P2P data of all the useless qualities of one peer that are unused by any of the peers. Therefore, the total P2P data per segment k is measured as it is shown in (4.7).

$$P2P[k] = s[k]_{p2p} + s[k]_{\overline{p2p}} + s[k]_{\overline{\overline{p2p}}} \tag{4.7}$$

The Peer Efficiency, as calculated in (4.8), reports the average useful P2P data over the total P2P data of K segments. And the last P2P metric, Peer-Pool-Efficiency, is the average of the reused P2P data over the total P2P data, for K

P2P segments as shown in (4.9).

$$PeerEfficiency = \frac{1}{K}\sum_{k=1}^{K}\frac{s[k]_{p2p}}{P2P[k]} \tag{4.8}$$

$$PeerPoolEfficiency = \frac{1}{K}\sum_{k=1}^{K}\frac{s[k]_{\overline{p2p}}}{P2P[k]} \tag{4.9}$$

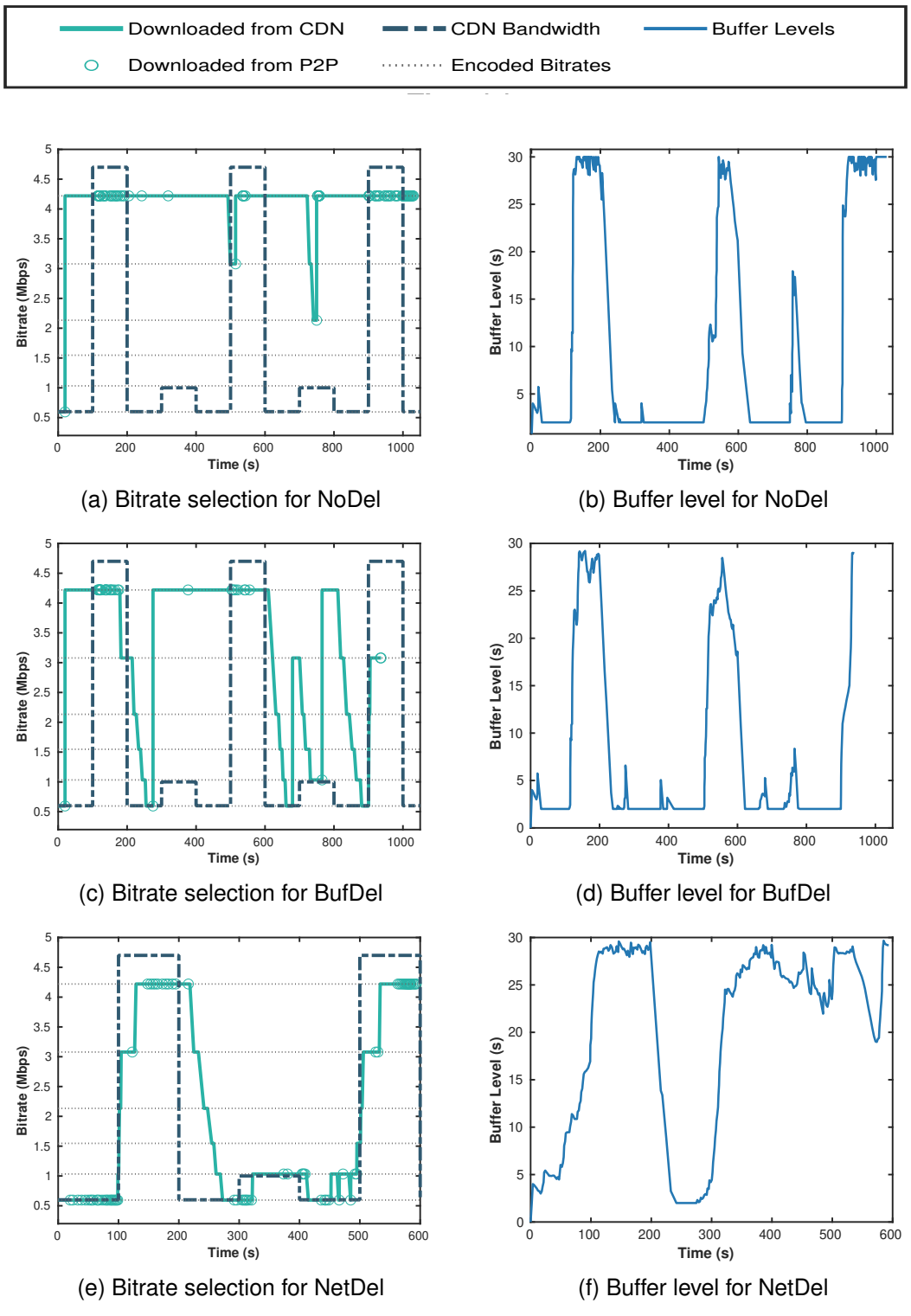
4.4 Results and discussions

We now study the performances of four state of the art ABR algorithms: *BBA*, *BOLA*, *PANDA* and *CONVENTIONAL* (denoted as *CONV*). We consider four scenarios: *CDN-only* is the normal CDN-based streaming with no P2P streaming, *NoDel* is a normal hybrid CDN/P2P streaming without applying any of the Response-Delay methods. *BufDel* is a hybrid CDN/P2P scenario using the BufDel approach described in part 4.2.2.1 and the last scenario *NetDel* is a hybrid CDN/P2P scenario using the NetDel method described in part 4.2.2.2.

4.4.1 Non-conservative throughput-based algorithms

We begin the analysis with the non-conservative throughput-based ABR algorithms, and we pick CONVENTIONAL as an example. The non-conservative algorithms are sensitive to any change in the viewing conditions; they choose the bitrates aggressively by following the measured bandwidth closely.

Figure 4.3 compares, side by side, the bitrate selection and the buffer levels of CONVENTIONAL for the *NoDel*, *BufDel* and *NetDel* scenarios. We are confronted with an important problem: the ABR over-estimating the bandwidth, whenever a P2P segment is fetched directly from P2P cache. In consequence, the ABR keeps requesting the highest quality for some time before re-adapting again. This behav-



(a) Bitrate selection for NoDel

(b) Buffer level for NoDel

(c) Bitrate selection for BufDel

(d) Buffer level for BufDel

(e) Bitrate selection for NetDel

(f) Buffer level for NetDel

Figure 4.3: Conventional's bitrate selection and buffer level for NoDel, BufDel, NetDel scenarios

ior is particularly risky when the current bandwidth can not sustain this quality, as shown in Figure 4.3a, resulting in depleting the playout buffer to a low level, causing re-buffering, as clearly shown in Figure 4.3b.

Moving to Figures 4.3c and 4.3d, it is clearly notable that BufDel does not help that much with this problem. In particular, BufDel works fine until the buffer level drops to a lower value, and some P2P segments are fetched from the P2P cache (looking at 10s, 270s in the same figures). Because these segments are still delivered fast from P2P cache, repeating the same behavior as we just discussed with NoDel case.

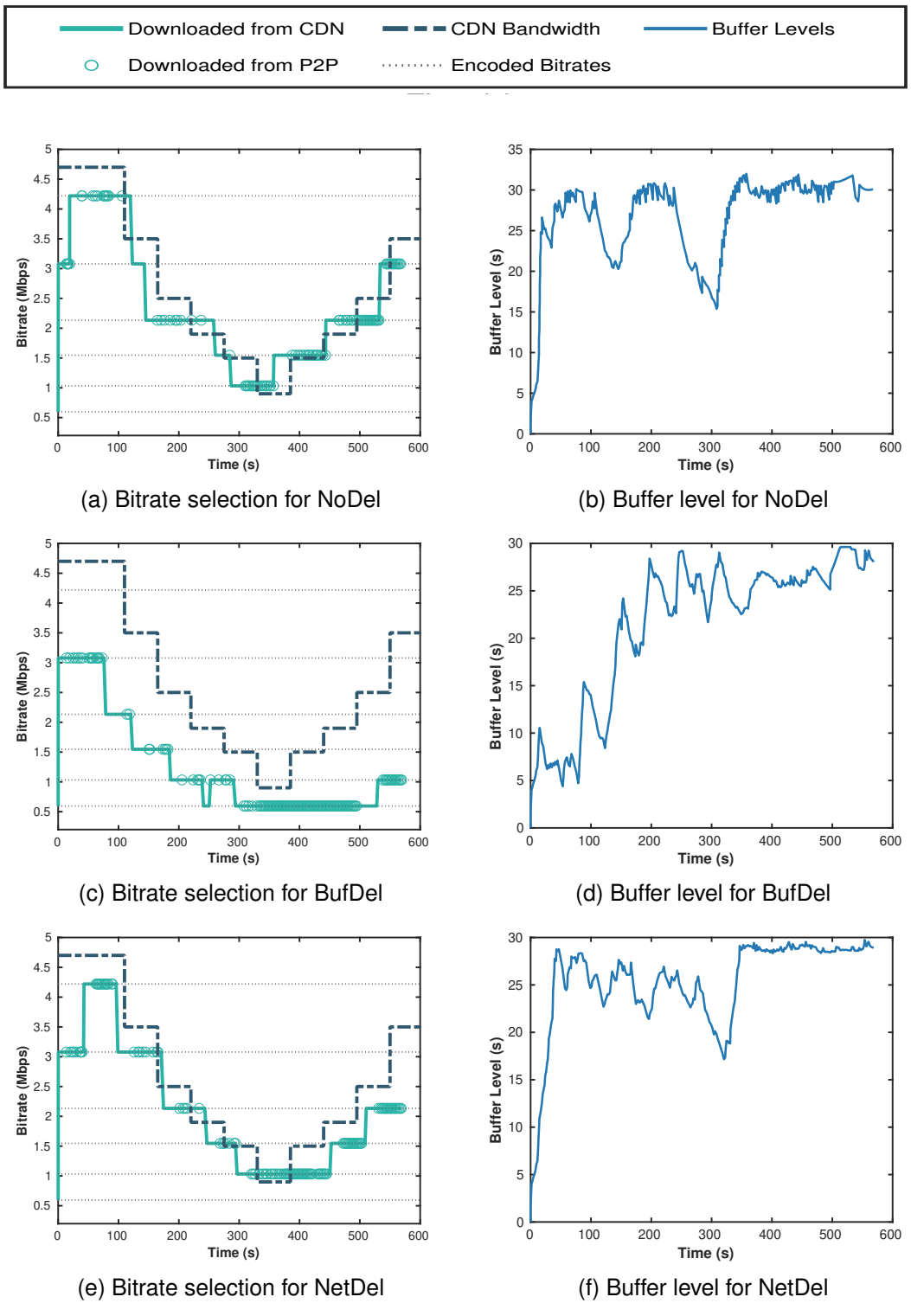
As opposed to NoDel and BufDel, NetDel avoids this issue by adapting the delay to the actual bandwidth, as shown in Figure 4.3e. The same figure shows that NetDel downloads more segments from P2P; it gives the system more time to prefetch data and complete segments from other peers. Whereas with BufDel, segments are delivered rapidly when the buffer level is low, leaving no time to finish the downloading of the next segments from P2P on time.

Also, it should be noted that when the rebuffering occurs more frequently, it causes longer playout time, which explains why the client with NoDel and BufDel finishes the playback after 1000s nearly (Figures 4.3a, 4.3b, 4.3c and 4.3d), while with NetDel the client plays continuously and finishes after 600s (Figures 4.3e and 4.3f).

4.4.2 Conservative throughput-based algorithms

In contrast to non-conservative algorithms, the conservative throughput-based algorithms, such as PANDA, adapt to the estimated bandwidth gradually, i.e., wait for some time before switching the video quality. This improves the playback stability but at the cost of lower quality.

PANDA is known to be resilient against the varying bandwidth, and this is still



(a) Bitrate selection for NoDel

(b) Buffer level for NoDel

(c) Bitrate selection for BufDel

(d) Buffer level for BufDel

(e) Bitrate selection for NetDel

(f) Buffer level for NetDel

Figure 4.4: PANDA's bitrate selection and buffer level for NoDel, BufDel, NetDel scenarios

the case when P2P segments are delivered fast from the P2P cache. Looking at Figures 4.4a and 4.4b, it is obvious that PANDA smooths the bandwidth estimation gradually and adapts to the smoothed bandwidth accordingly.

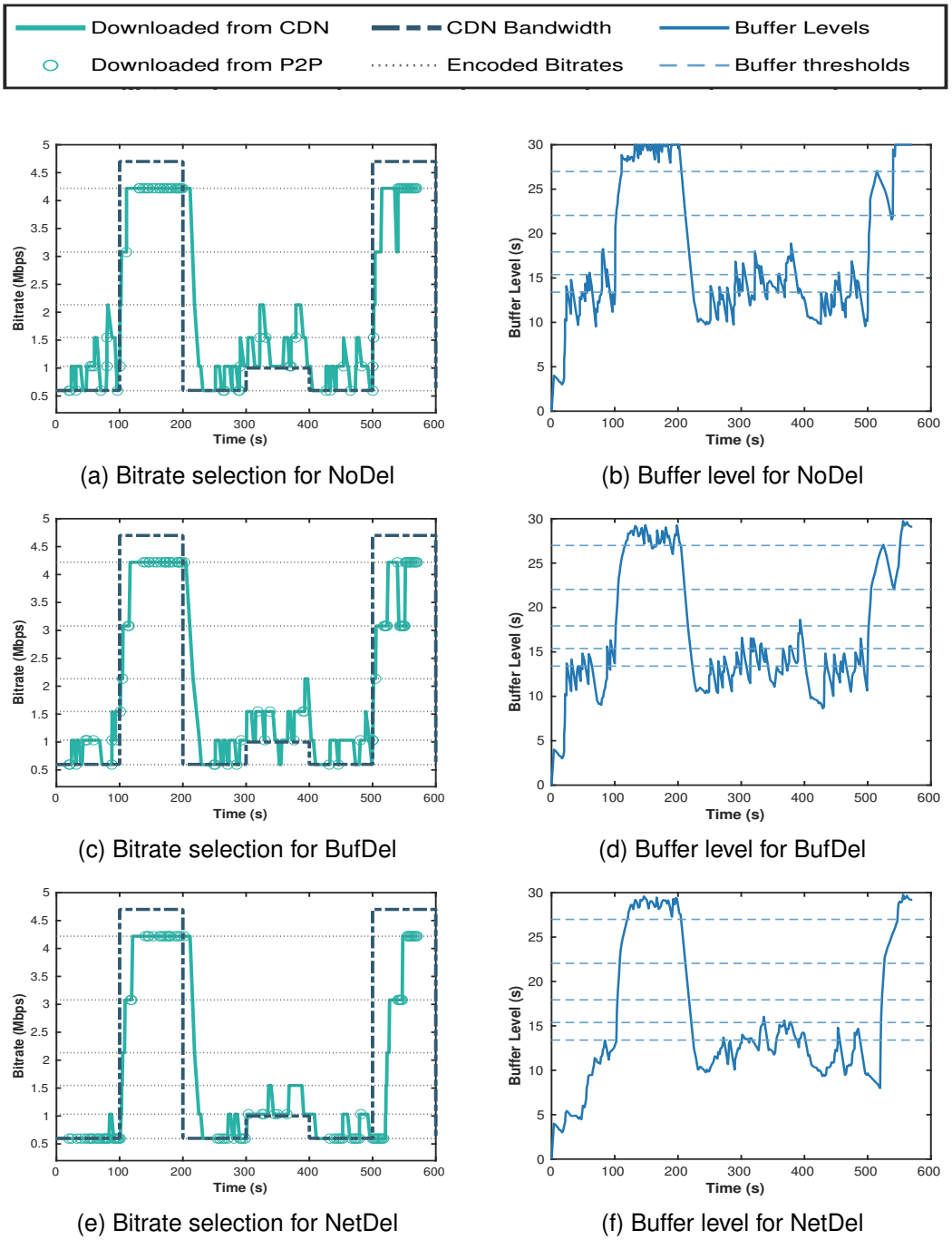
BufDel causes a different problem in terms of bandwidth underestimation. The selected bitrate is clearly lower for BufDel when looking at Figure 4.4c, especially when the buffer level is close to Q_{max}, which is equal to 26 seconds in this example. According to (4.4), these P2P segments are delivered nearly at the average rate, which makes PANDA lose tracking the actual bandwidth and only requests the same quality for a long time; until receiving some CDN segments, when it re-adapts again. Interestingly, this problem may also happen with the non-conservative throughput algorithms; however, they recover faster whenever a new CDN segment is downloaded.

With NetDel, it is notable that this problem does not exist, but another arrives. PANDA uses a mix of P2P and CDN measurements, according to (4.2). When the P2P is activated, no requests will be sent to the CDN for some time, which keeps NetDel with the old measurements of the CDN. This is not a problem when the P2P is good enough to handle the situation, but in a scenario (as the one shown in Figure 4.4e) where the measured P2P bandwidth and the last measured CDN bandwidth are both low, NetDel aims a lower bandwidth, making the ABR degrade the quality as well. This behavior persists as long as the current P2P bandwidth is low and the requested segments are delivered from the P2P cache.

4.4.3 Buffer-based algorithms

We now evaluate buffer based algorithms, illustrated with only BBA algorithm in this section; as BOLA showed the same issues.

In opposition to throughput-based algorithms, buffer-based algorithms are usually more resilient to rebuffering since they make the ABR decision based on the



(a) Bitrate selection for NoDel

(b) Buffer level for NoDel

(c) Bitrate selection for BufDel

(d) Buffer level for BufDel

(e) Bitrate selection for NetDel

(f) Buffer level for NetDel

Figure 4.5: BBA's bitrate selection and buffer level for NoDel, BufDel, NetDel scenarios

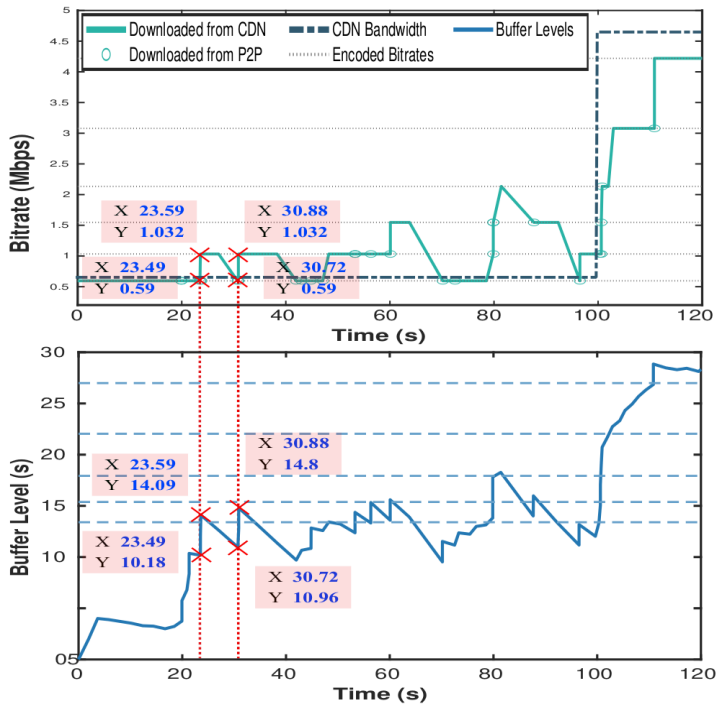


Figure 4.6: Example of quality switches with BBA

buffer occupancy directly. For NoDel, this fact is confirmed again with BBA algorithm, as Figure 4.5b shows.

Nevertheless, this class of algorithms is usually vulnerable to quality switches. We take a sample from Figure 4.5a, to illustrate the quality switching issue as shown in Figure 4.6, The BBA buffer thresholds are 13.4, 15.38, 17.93, 22.04, and 26.99 seconds. At time 23.49s, the buffer level is 10.18s, and the player receives 4 seconds of two consecutive P2P segments of 0.59Mbps. The buffer then grows to 14.09s, crossing the threshold at which the player switches to 1.032 Mbps. The next two segments of 1.032 Mbps are not available in the P2P cache, so they are downloaded from CDN, and it takes almost 7.1s to be completed. Meanwhile, the buffer depletes to 10.96s, crossing the threshold and switching back to 0.59Mbps. Thus, the cycle repeats whenever P2P segments are received fast while the actual bandwidth can not sustain the quality up switches.

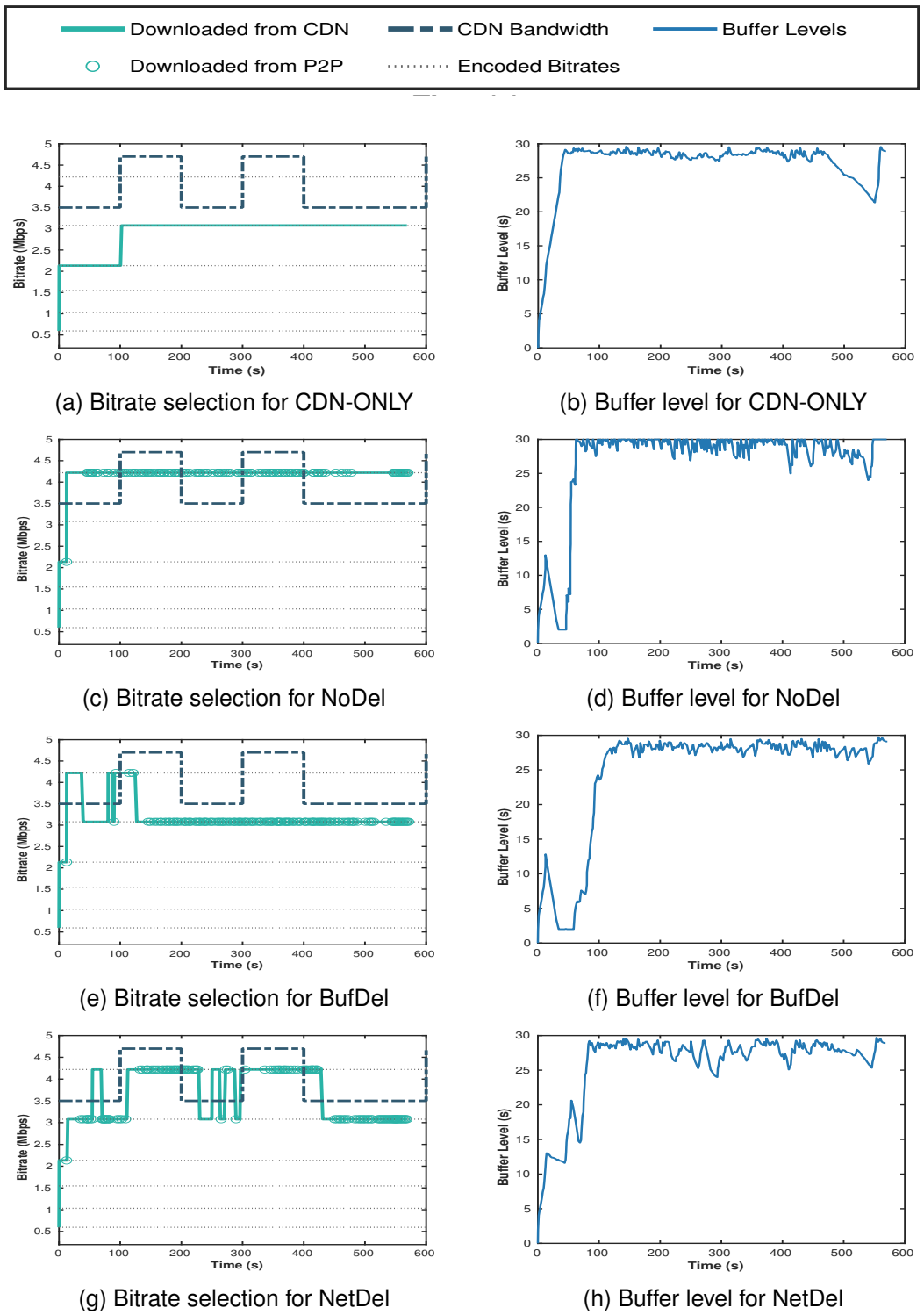
We observe from Figure 4.5c that BufDel does not help avoid these oscillations. Indeed, when BufDel detects a low buffer level, it accelerates the delivery of P2P segments, making the buffer level cross the switching thresholds back and forth again. Contrary to NoDel and BufDel, NetDel avoids this issue by adjusting the delivery rate of P2P segments to the current bandwidth measurements, but at the cost of a lower bitrate (see Figure 4.5e and 4.5f).

4.4.4 Results with normal high network profiles

Besides the varying bandwidth conditions, we herein study the proposed methods under good viewing conditions. We repeat the same scenario for all the used ABR algorithms, replacing only the bandwidth profile with another one where the bandwidth is high enough to sustain the highest two qualities.

Starting with CONV algorithm, Figures 4.7c and 4.7d indicate that fast delivery of P2P segments does not seem to be a problem with NoDel any more. With the high bandwidth, it rather improves both QoE and the P2P efficient usage compared to CDN-only (Figures 4.7a and 4.7b); since all peers have enough bandwidth to watch and exchange higher qualities with other peers. However, BufDel still suffers the same issue of the ABR selecting the same quality for a long time. This issue reappears in Figure 4.7e as long as the P2P segments are prefetched and delivered at nearly the same average bitrate for long time; this stabilizes the ABR feedback, resulting in selecting the same bitrate. With NetDel, looking at Figure 4.7g, we can see that the bitrate selection follows the bandwidth variations closely, sacrificing the QoE in terms of less average rate and higher track switches comparing to NoDel scenario.

For PANDA, the bitrate selection for both BufDel and NetDel shows that the response delay does not have a significant effect on the conservative behavior of this algorithm (Figures 4.8e and 4.8g). In both cases, PANDA estimates and smooths



(a) Bitrate selection for CDN-ONLY

(b) Buffer level for CDN-ONLY

(c) Bitrate selection for NoDel

(d) Buffer level for NoDel

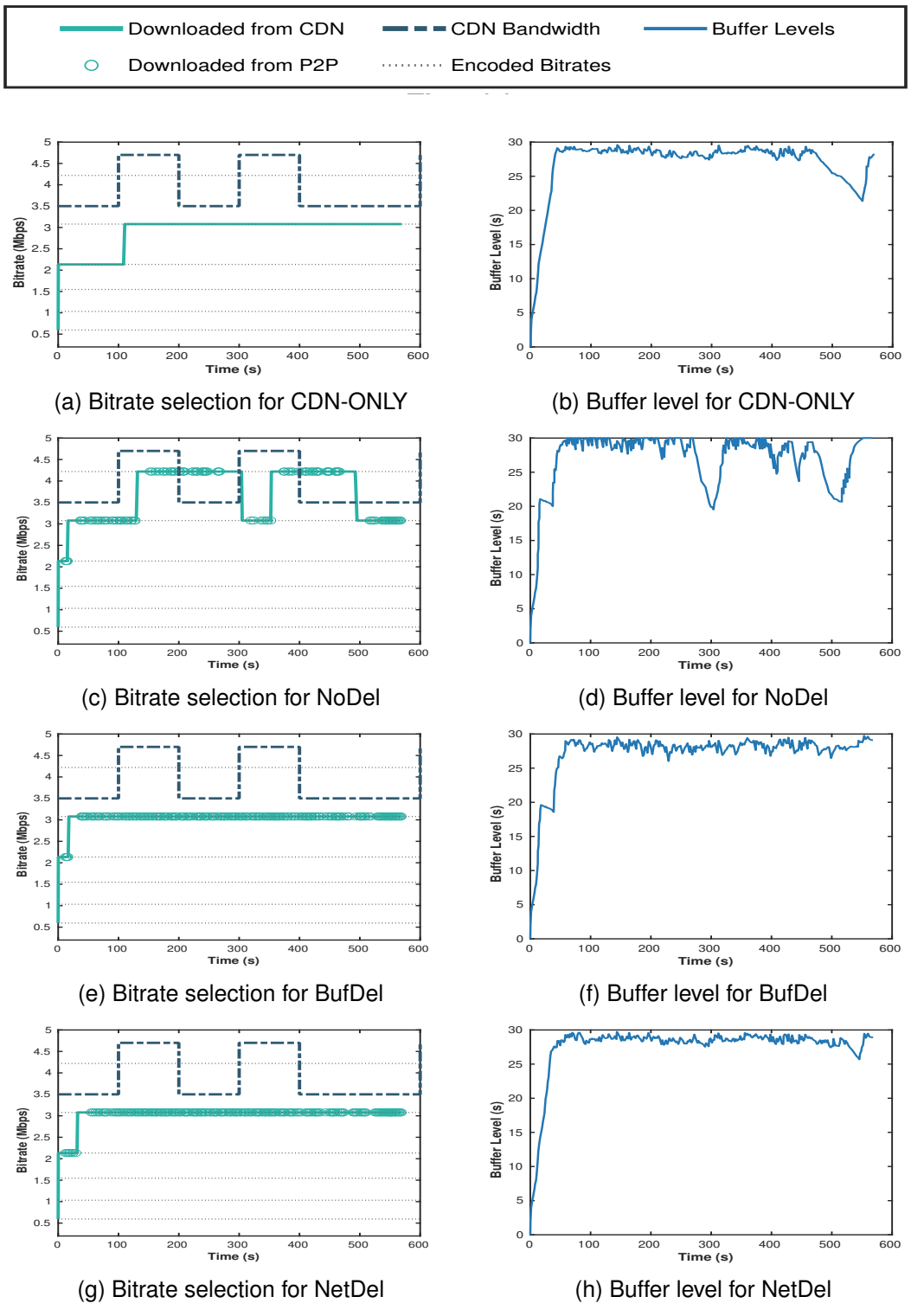
(e) Bitrate selection for BufDel

(f) Buffer level for BufDel

(g) Bitrate selection for NetDel

(h) Buffer level for NetDel

Figure 4.7: CONV's bitrate selection and buffer level for NoDel, BufDel, NetDel scenarios with a normal high trace



(a) Bitrate selection for CDN-ONLY

(b) Buffer level for CDN-ONLY

(c) Bitrate selection for NoDel

(d) Buffer level for NoDel

(e) Bitrate selection for BufDel

(f) Buffer level for BufDel

(g) Bitrate selection for NetDel

(h) Buffer level for NetDel

Figure 4.8: PANDA's bitrate selection and buffer level for NoDel, BufDel, NetDel scenarios with a normal high trace

the bandwidth to pick a fair quality. It stays patient and does not switch up immediately, it rather waits until it guarantees the bandwidth stability, then it switches up to the highest track. With NoDel (Figure 4.8c), we observe that PANDA stays on the highest bitrate for a longer time that CDN-only (Figure 4.8a), and we see that most of the video segments are requested and downloaded from P2P network. Although this behavior is fine from QoE perspectives, it breaks the conservative design of PANDA behavior; by over-estimating the bandwidth. Here, all the P2P segments are delivered directly from the P2P cache leading to an almost infinite bandwidth estimation. PANDA sees this estimation high and fair enough to select the highest quality and keep it for the whole P2P segments.

Finally, the buffer-based algorithms, represented with BBA, show good performances under good bandwidth conditions. With NoDel, the P2P link is good enough to keep filling the P2P cache with the required segments, so the video player receives segments from P2P cache as soon as it requests them. Consequently, the ABR can always aim at the highest quality as long as the buffer level is above the threshold at which it decides to request the highest quality (see Figures 4.9c and 4.9d). Whereas in CDN-only, it takes some time to download the video segments form CDN which leads to buffer level and bitrate fluctuations (Figures 4.9a and 4.9b). Unfortunately, in the case of good viewing conditions, using response delay does not seem to be a good choice. Logically, creating some delay before returning the segments has the drawback of consuming the buffered content for some time. As we mentioned earlier, the buffer level algorithms are sensitive to small changes in the buffer levels since it works by dividing the buffer into areas for each bitrate. Thus, using response delay may risk crossing down the high threshold and switching down to a lower quality. This explains the few quality switches for both BufDel and NetDel in Figures 4.9e and 4.9g, due to the dropping in the buffer levels as shown in the buffer graphs for the proposed methods (Figures 4.9f and

4.9h).

4.4.5 All metrics evaluation

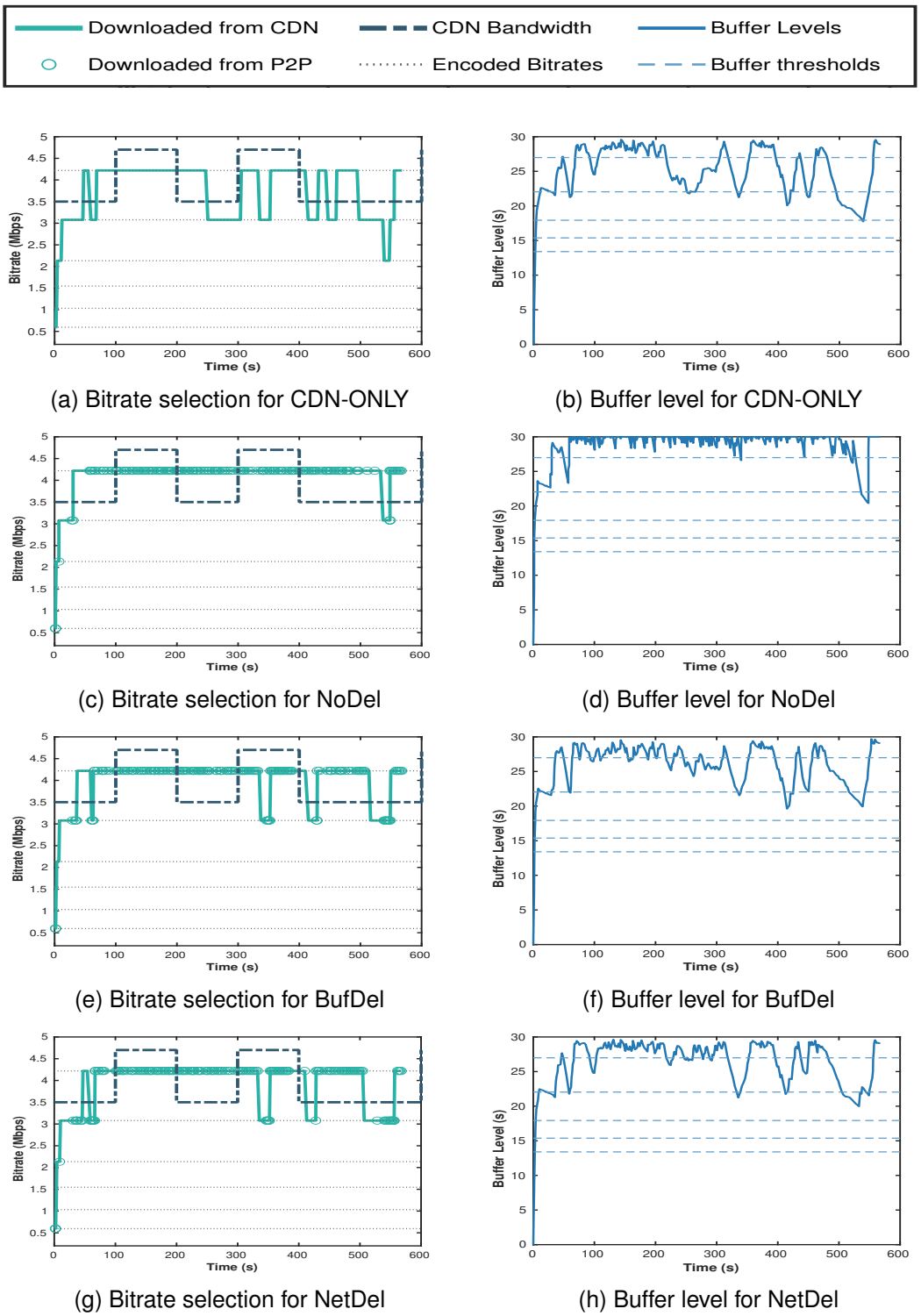
In this section, we provide a comparison between the different scenarios by evaluating the performance in terms of the metrics presented in Chapter 2. Starting with classic QoE metrics, then metrics of interest for P2P performance.

All results are averaged over all peers and all different 3G traces except for P2P metrics, for which the first peer, which is connected to CDN link only, is excluded.

4.4.5.1 QoE metrics

Starting with the average rate metric, shown in Figure 4.10, both BufDel and NetDel show improvements on the average quality compared to the CDN-only scenario. For buffer-based (BBA and BOLA) and non-conservative throughput algorithms (CONVENTIONAL), BufDel achieves higher average bitrate compared to NetDel, which is foreseen as discussed in sections 4.4.1 and 4.4.3). In fact, BufDel works negatively with the low buffer levels by accelerating the delivery from P2P cache, leading to over-estimating the bandwidth and affecting the responsiveness of these algorithms. For PANDA, we see the discussion of Section 4.4.2 in numbers; NoDel has a higher average rate compared to both BufDel and NetDel. Again, because of the bandwidth over-estimation, which breaks the conservativeness of PANDA.

The stability is a major issue for buffer-based algorithms, as stated in Chapter 2. We see in Figure 4.11 that NetDel manages to further stabilize the ABR algorithms, especially BBA and BOLA. It achieves better than BufDel, and as good as the CDN-only case for these algorithms. CONVENTIONAL is a special case: it is more sensitive to the bandwidth changes, and since we have the problem of over-estimating the bandwidth, we can see that NoDel selects higher bitrates compared



(a) Bitrate selection for CDN-ONLY

(b) Buffer level for CDN-ONLY

(c) Bitrate selection for NoDel

(d) Buffer level for NoDel

(e) Bitrate selection for BufDel

(f) Buffer level for BufDel

(g) Bitrate selection for NetDel

(h) Buffer level for NetDel

Figure 4.9: BBA's bitrate selection and buffer level for NoDel, BufDel, NetDel scenarios with a normal high trace

to BufDel and NetDel.

We observe the same result for smoothness (the amplitude between the quality switches), as Figure 4.12 shows: the transitions between the video qualities are almost as smooth as those using CDN-only, and NetDel is slightly better than BufDel since the latter triggers the fast delivery and bandwidth over-estimation issues, in particular when the buffer level is low. The most notable improvement is gained for CONVENTIONAL, which mostly switches between the highest and the lowest bitrates when it is confused by P2P segments (in NoDel), and NetDel correct this behaviour by ajusting the bandwidth estimation.

For consistency (Figure 4.13) and continuity (Figure 4.14), all algorithms for all scenarios gain the same score except for CONVENTIONAL, where NetDel registers a significant improvement (up to 55% for continuity and 30% and consistency) to the normal P2P scenario; this is a direct consequence of resolving the bandwidth over-estimation issue as already discussed.

4.4.5.2 P2P metrics

Regarding the P2P metrics, NetDel shows better improvement on the P2P Offload, as shown in Figure 4.15, compared to NoDel and BufDel. Looking at Figure 4.16, we can see that both BufDel and NetDel have better results on the peer efficiency compared to NoDel for BBA, BOLA, and PANDA, with NetDel being the best. However, for CONVENTIONAL, this metric is lower than NoDel; since with NoDel, CONVENTIONAL does not often switch between qualities but rather stays longer at the highest quality leading to less quality switches-related overhead, which in turn means a higher peer efficiency. The same result is observed for Peer Pool Efficiency metric as seen by Figure 4.17. NetDel shows a better sharing of the overhead segments to other peers for all the ABRs except CONVENTIONAL (again, for the same reason of less overhead due to quality switches).

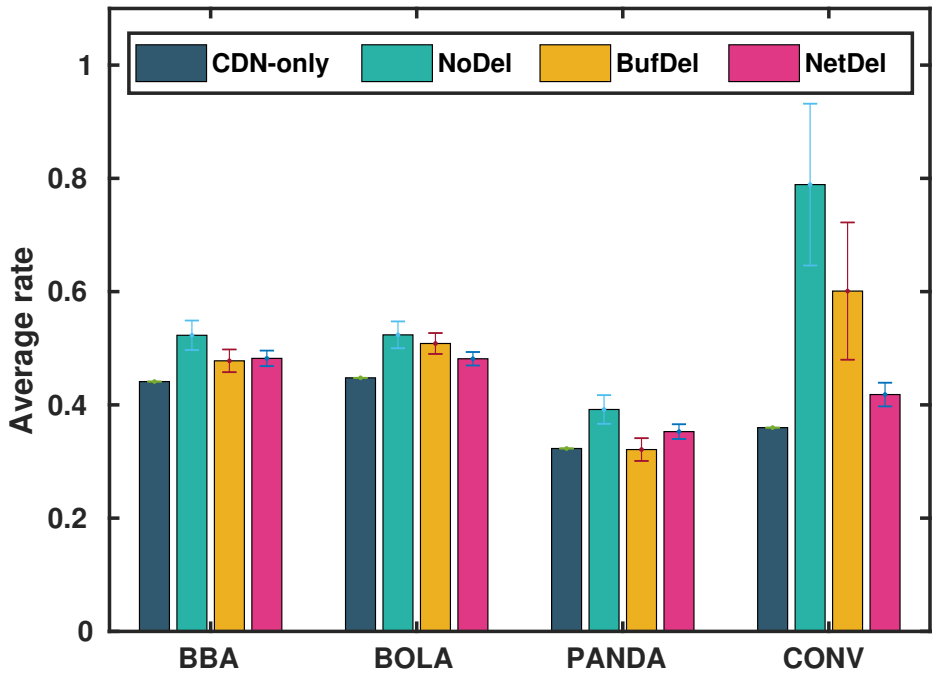


Figure 4.10: QoE metrics: average rate

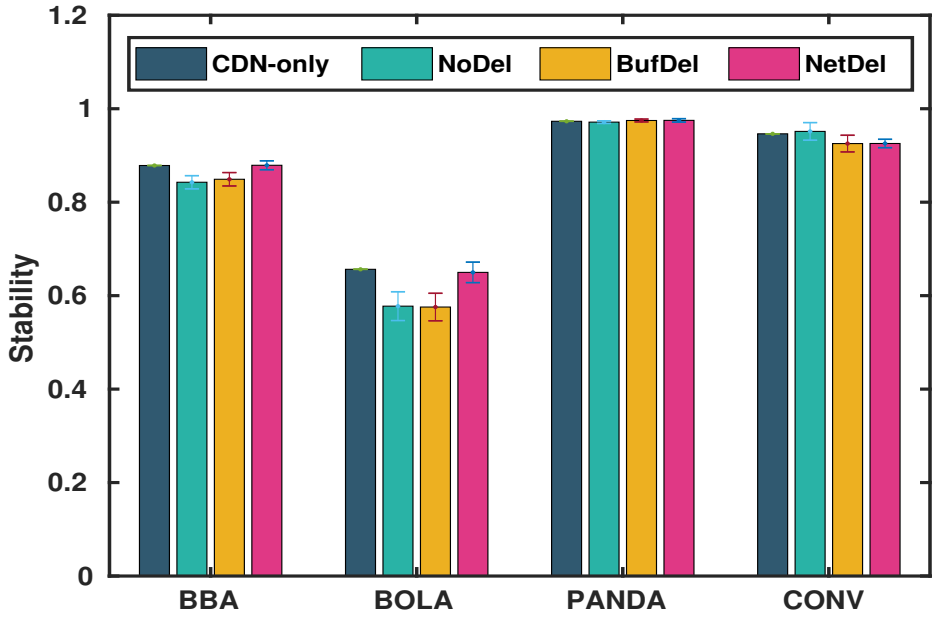


Figure 4.11: QoE metrics: stability

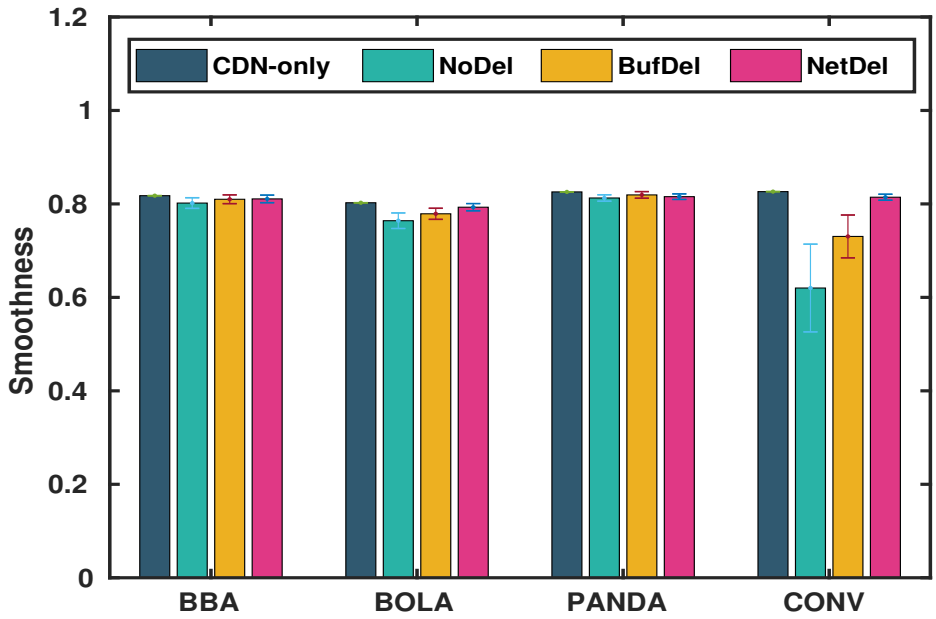


Figure 4.12: QoE metrics: smoothness

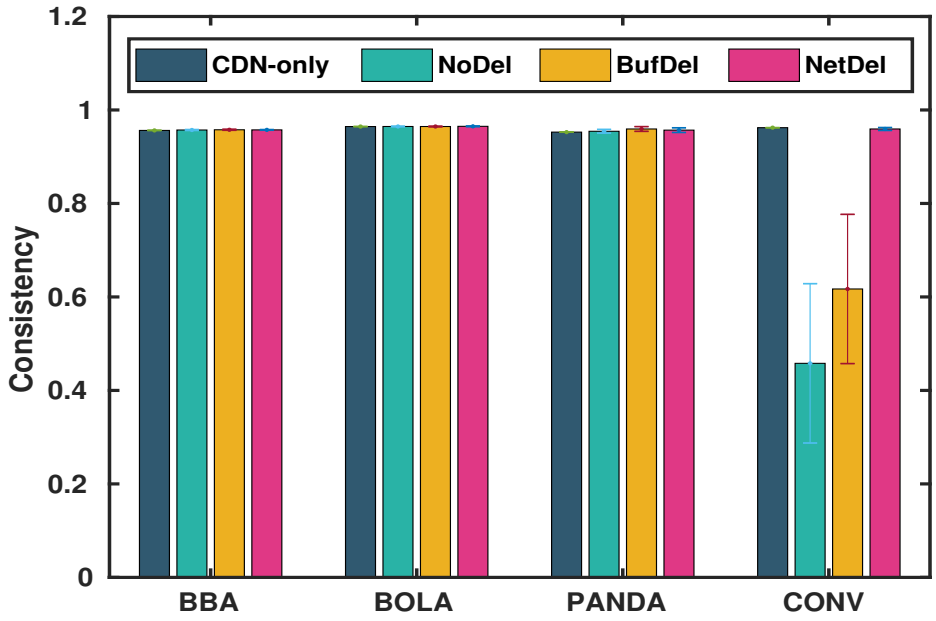


Figure 4.13: QoE metrics: consistency

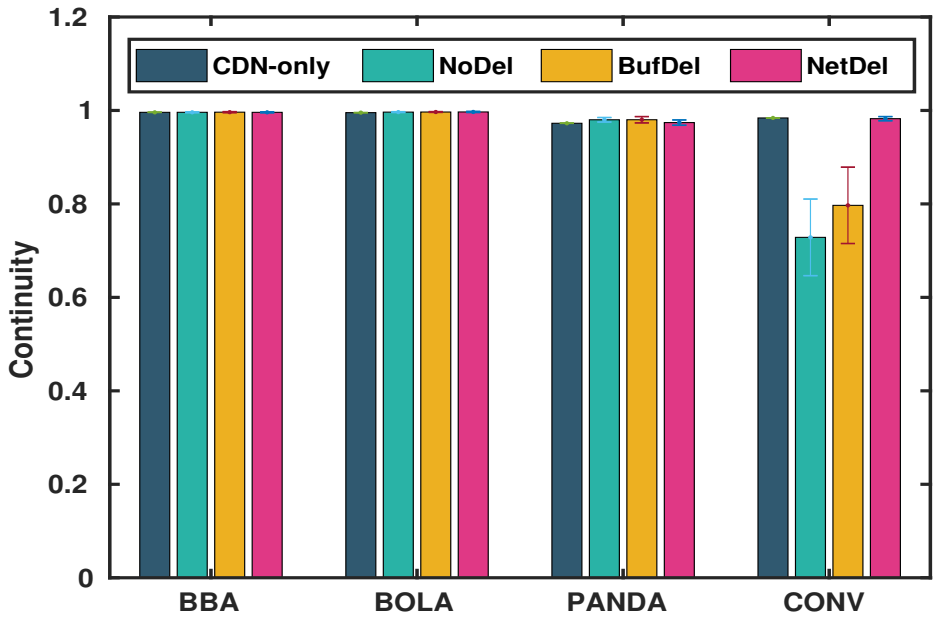


Figure 4.14: QoE metrics: continuity

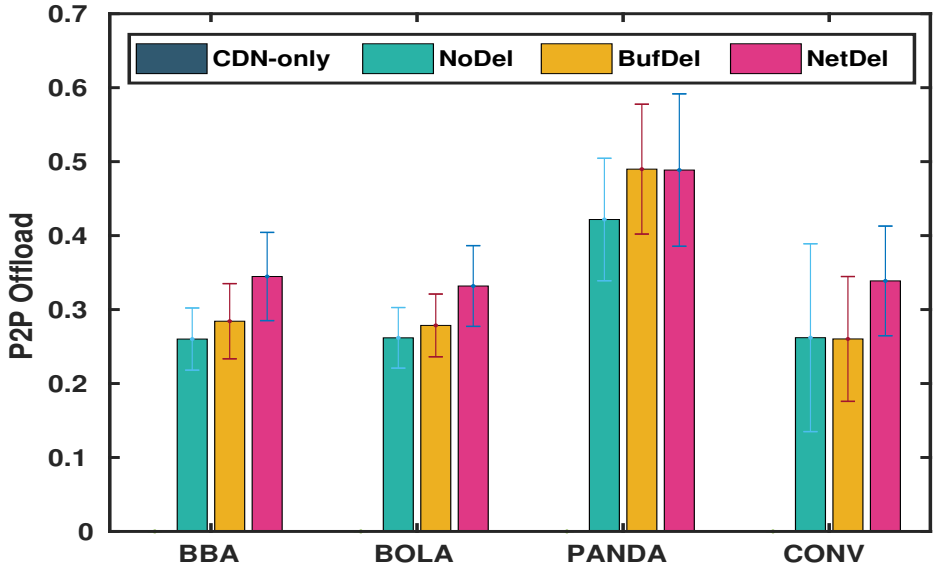


Figure 4.15: P2P metrics: offload

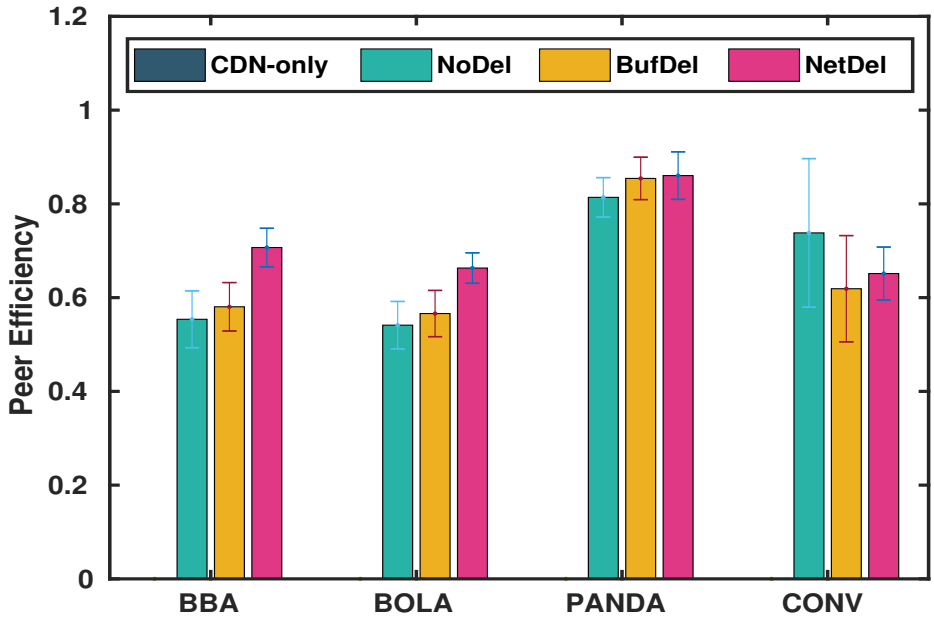


Figure 4.16: P2P metrics: peer efficiency

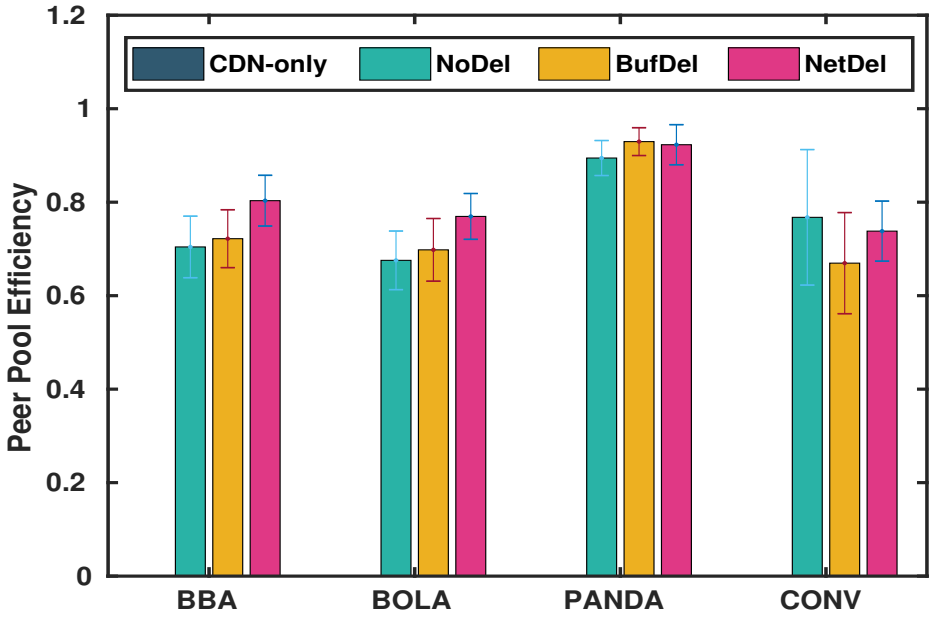


Figure 4.17: P2P metrics: peer pool efficiency

4.4.6 Commercial Service Trials

To verify our work with unknown ABR algorithms, we tested the two proposed methodologies in existing commercial streaming services, thanks to STREAMROOT technology providing the P2P backend. The trials were conducted with various HTML5 video players implementing their own, different and unknown ABR algorithms. The scenarios CDN-only and NoDel were omitted in these trials because of their bad P2P efficiency and QoE drawbacks on the users and the customers, they are not suitable for commercial services. Therefore, we could not do the tests for CDN-ONLY and NoDel, we only compare the BufDel and NetDel with what we observed in our simulations. We launched the test for one day, with an overall range of 5k to 15k concurrent peers participating. The service provider used 3 different live streams, segmented into 10-second segments, and encoded in a different set of bitrates: [2.2, 1.2, 0.94, 0.446], [1, 0.796, 0.446] and [1.248, 0.698, 0.348] (Mbps).

Table 4.1: Commercial Service Trials

Metric	Method	
	BufDel	NetDel
Avg rate (Mbps)	1.183	1.173
Avg track switches (CPM)	0.117	0.115
Avg rebuffering events (CPM)	0.088	0.094
Rebuffering Duration (s)	6.75	6.43
P2P offload %	46.50	46.55

In this comparison, and to ease the data collection, we used some common production metrics, collected per 2 minutes intervals, and averaged over the whole session. These metrics are the average rate in Mbps, the average number of quality switches per minute (CPM), the average number of experienced re-buffering events per minute (CPM), the average re-buffering duration, and the ratio of the useful P2P data over the total downloaded data. Table 4.1 shows the comparison of BufDel and NetDel regarding these metrics. The overall results show that NetDel is slightly better in terms of quality switches and P2P efficiency. It also has a lower re-buffering duration but with a slightly higher number of re-buffering events. On the other hand, BufDel increases the average bitrate (per minute), and it stays longer on the highest quality; this is consistent with our simulation results.

4.5 Conclusion

In this chapter, we studied and discussed the main problems related to the usage of existing ABR algorithms in hybrid CDN/P2P networks, a study which, to the best of our knowledge, was never done before. We also introduced Response-Delay, a novel algorithm ensuring the compatibility of existing ABR algorithms with prefetching-based P2P networks, without any changing to the ABR algorithms. Our

results show that Response-Delay enables using the different ABR algorithms in P2P networks while keeping a good QoE and P2P efficiency. Both simulation and realistic tests show that choosing between the two approaches of Response-Delay is a trade-off: NetDel is recommended for a cost-efficient (more P2P), stable and smooth streaming, whereas BufDel is recommended when the average quality is more important than the cost-efficiency and the stability of the streaming; though we encourage using NetDel as it achieves a better QoE-P2P efficiency trade-off. Yet, both Bufel and NetDel treat only the fast delivery issue of P2P segments, but do not check if this delay leads the ABR to switch to another quality than the cached one. Chapter 6 will discuss using Response-Delay approach in a smarter way, aiming to improve prefetching and drive the player ABR towards selecting the already cached segments. Towards this end, Chapter 5 proposes to learn the ABR algorithm's behavior to be able to predict the bitrate decision given the response delay. Then Chapter 6 uses the bitrate prediction model to optimize the response delay.

Chapter 5

Adaptive BitRate prediction using supervised learning algorithms

5.1 Introduction

Video delivery networks, such as CDN and P2P, have adopted prefetching techniques to better use available resources. These techniques consist of loading data into either local or remote caches before the user requests it. With ABR streaming, these techniques may face some obstacles, especially with the client-based adaptation logic that leaves the bitrate decision to the end-user (as explained in Chapter 2). In this design, the clients usually run an ABR algorithm to sequentially select the best quality according to the available bandwidth. The prefetching techniques, which download video segments ahead of time, may end up wasting resources by downloading segments of a certain quality that is different from what the ABR will request later. Furthermore, for P2P streaming, it takes some time to find peers who have the new segments and may cost downloading these segments from CDN. Thus, knowing the ABR qualities at the prefetching level will be valuable as it saves time and resources.

In Chapter 4, we proposed Response-Delay, a method to solve the issue of the instant delivery of P2P segments from the local cache. Response-Delay suggests creating a delay before delivering these segments to the player. Although this method improves the QoE and the P2P performance, it does not take the P2P cache status into account. The created segment delay will lead the ABR to select a certain quality. It would be preferable to check if this quality exists in the P2P cache or another cached quality for the next segment, which is higher than what the ABR would select. Hence, the first step towards this goal is to predict the quality that will be selected due to the segment delay.

In the layered HAS and P2P stack implementation, the video player is integrated on top of the P2P stack, replacing the HTTP stack as a transport layer (as mentioned in Chapter 1). The ABR logic is completely closed source and unknown to the P2P stack, which means the prefetching technique can not directly know the ABR decisions in advance.

For the reason, in this chapter, we try to learn the ABR behavior and predict its decision without any knowledge about the ABR itself.

The first thing that comes to mind when mentioning learning any system's behavior is Machine Learning (ML) techniques. Indeed, ML and other control techniques have shown to be promising in many research fields, and video streaming is not an exception. In the context of enhancing the video streaming, ML algorithms have been used in different approaches like CDN caching[93], ML-based ABR solutions [94], video traffic classification [95] and of course the wide application of ML-based video coding [96]. Learning the ABR decision fits more into Supervised Learning (SL). SL enables learning from past observations to predict future events, which is the main task of ABR. Therefore, mixing these two approaches is a logical choice.

In light of this debate, in this chapter, we build an ABR agnostic model that pre-

dicts the ABR decisions giving only well-defined measured variables (e.g., bandwidth, buffer level, etc). This model is the first step towards designing an ABR-aware prefetching and P2P-friendly bitrate selection, which is the main objective of Chapter 5. We choose some of the well-known supervised classifiers to predict the bitrate decision of ABR algorithm. To validate that our work is ABR agnostic, we first test the models on six well-known state-of-the-art ABR algorithms. Then, we predict the bitrate decision of three commercial, completely unknown, and closed-source algorithms using real datasets collected from both VoD and Live sessions.

5.1.1 Prior work

In Chapter 2, we classified the ABR algorithms into four main classes: buffer-based, throughput-based, hybrid-buffer-throughput-based and control-based class.

Even if they differ in their deep decision logic, most of the state-of-the-art ABR algorithms rely on heuristic observations as inputs to optimize the bitrate selection of the next segments. These inputs are usually the bandwidth measurements (e.g., TCP throughput and the download time as seen by the application-layer), the buffer dynamics (e.g., the buffer occupancy and the maximum buffer size mainly), the segment characteristics (e.g., size, duration, and encoding bitrate) and in some cases the device capabilities (e.g., CPU usage, memory, playback speed). Authors in [19] provide an extensive survey on different ABR schemes. Interestingly, they show that the majority of the client-based adaptation schemes rely on bandwidth and/or buffer heuristics. Besides, few algorithms use information about the segment characteristics and device capabilities.

As previously mentioned, the research in adaptive streaming is shifting towards machine learning and optimization control. ML and ABR intersect at the point of learning from heuristic observations and predicting future decisions. In this context, some prior works tackled the adaptation problem from another point of view

by not proposing a new algorithm from scratch but improving the existing ones using ML algorithms. In their work [97], authors proposed a model, called MLASH, that uses an additional set of features to train a random-forest classifier to improve the prediction accuracy of some ABR algorithms. However, MLASH uses some information, *True information* as referred, from the ABR algorithms. Besides, we are not aware of any work that implements MLASH on the control-based ABR or any experiments with realistic VoD and Live datasets. A similar work [98] uses Long Short Term Memory (LSTM) to predict a client-side Scalable Video Coding (SVC)-based bit-rate adaptation using a set of heuristic attributes (e.g., buffer and throughput variables.). Although they compare different ML algorithms and perform good trace-based evaluation, the proposed model tackles only the SVC-based algorithms, and we are not aware of any work that extends to the classic heuristic and control-based ABR algorithms.

5.1.2 Contributions

The contributions of this chapter are the following:

1. We present a simple SL-based approach to predict the bitrate decision of any ABR algorithm, without any prior knowledge about the ABR itself. This model helps designing an ABR-aware prefetching and making P2P-friendly ABR decisions (as we will see in Chapter 5).

The rest of this chapter is organized as follows: the bitrate selection is formulated as a classification problem in Section 5.2. The experimental setup is described in Section 5.3. Section 5.4 discusses the performance of the different classifiers.

5.2 Bitrate selection classification problem

ABR bitrate prediction can be formulated as a multiclass classification problem, where the different classes represent the different encoding bitrates. The model then tries to map the set of the input features to the possible bitrate. The first step of solving this ML problem is to define the features needed to predict the ABR bitrate. As stated earlier, most ABR algorithms use a set or a subset of network, buffer, or segment variables. In our study, and to keep the model generic, we will use the following set of features:

1. Buffer Level (s): the buffer occupancy when requesting the segment, which is usually available on most video players' public APIs.
2. Bandwidth (bps): the TCP throughput as seen by the application layer after downloading the segment, which is simply measured by computing the data downloaded (segment size here) over the download time.
3. Previous Bandwidth (bps): the throughput measured when downloading the previous segment; this information is used in different algorithms for smoothing the bandwidth estimation; or when the current estimation is not available as is the case with some buffer-based algorithms.
4. Download Time (s): the segment download time.
5. Previous Bitrate (bps): the bitrate of the previously selected segment, as used by the majority of ABR algorithms to take the bitrate smoothness into account.

Some other features can be included in the bitrate selection problem, such as RTT, moving average bandwidth over n segments, rewards when ML-based or control-based ABR are used, processing power, packet loss, some network-assisted information, etc. Nonetheless, some of these parameters require some information about the ABR algorithm beforehand (e.g., the value of n and the $reward$),

which conflicts with the core idea of our proposal of being ABR agnostic. Moreover, other metrics strongly depend on the device capabilities, and generalizing these metrics over different users is of our interest in future work.

As our features have a wide range of values, we rescale the above-mentioned features, using Min-Max scaler, where all features will be transformed into the range [0,1]. Then, we build our dataset matrix of pairs of M inputs (features) and output (label) over N instances (video segments in our context). For training and testing, we perform stratified K-Folds Cross-Validation with $K = 10$, where the data is further split into K different subsets (or folds). The folds are made by preserving the percentage of samples for each class. Then, $K - k$ folds are used to train the model, whereas the subset k is left as test data. The model results are then averaged against each of the folds and tested after against the Test set.

5.3 EXPERIMENTAL EVALUATION

To validate our model's performance, in this section, we use a wide set of datasets from simulation and real-life cases.

Starting with simulation, we used six classic ABR algorithms: BBA [29], BOLA [31], CONVENTIONAL [25], PANDA [25], FESTIVE [27] and Robust MPC [99]. These algorithms are quite famous in the adaptive video streaming domain, and they are frequently used in the literature. We implemented all these algorithms using the MATLAB-based model presented in Chapter 3, except for Robust MPC, for which we used an existing Python-based implementation available as part of Pensieve [1]. The statistics were collected from 10 users who were watching the same content and experiencing different bandwidth profiles. The bandwidth profiles were chosen from some publicly available sets of real 3G [91], and 4G [100] bandwidth

[1] Available at https://github.com/hongzimao/pensieve

traces so that the performance could be evaluated in highly dynamic adaptive scenarios. For the video content, we chose Red Bull Play Streets video from DASH Dataset[2], one-hour duration, segmented into 2-second segments and encoded at six different bitrates: 0.3, 0.7, 1.19, 1.99, 2.99 and 4.981 Mbps.

For realistic datasets, and to validate our work with unknown ABR algorithms, we collected the stats from commercial streaming services using STREAMROOT technology providing the P2P backend. The trials were conducted with various HTML5 video players implementing their own, different ABR algorithms, where some information like the buffer level can be accessed through DOM independently from the ABR implementation. Three commercial service providers, referred as $S1, S2, S3$, were chosen randomly, and for each, the data was collected from an overall range of 5k to 10k concurrent peers. The service provider used different live and VoD streams, so we built two datasets, one for VoD and one for Live sessions, each with up to 40000 instances.

We performed our experiments with the following ML algorithms: Logistic Regression (LGS), Support Vector Machine (SVM) [101], Random Forest (RF) [102], Decision Tree (DT) [103], Ada Boost (AdBst)[104], Gradient Boost (GrdBst)[105], Naive Bayes (NB) [106], K-Nearest Neighbors (KNN)[107]. For their implementation and measurement, we used Scikit-Learn package [108] on a Linux machine (Ubuntu 20.04 LTS), with processor Intel® Core™ i7-8665U, running 8 cores at 1.90GHz with 16 GB RAM.

Solving the machine learning problem usually starts with an important step: figuring how the used features are relevant to the model's output. Every feature in the dataset is represented by a column in the feature matrix which is used in training and testing the model. Thus, all features will have an impact on the output, and irrelevant features will have a negative impact on the output. In our ABR problem,

[2]Available at: https://dash.itec.aau.at/download/

feature importance is even more important since it reveals the nature of the used ABR algorithm, or the class to which the ABR belongs; more information is provided in Sections 5.4.1.1 and 5.4.2.1.

For the model evaluation, it is common in classification problems to use some known metrics (e.g., accuracy, precision, recall) [109] to evaluate models. The prediction accuracy metric reports how many classes are predicted correctly out of the total prediction samples. The precision looks at the proportion of samples that actually belong to a class out of the total samples that are predicted as belonging to this class. Lastly, Recall describes the proportion of correctly assigned samples to a class out of all the samples that are actually belonging to this class.

Many works recommend that using accuracy only is not enough. Depending on the application, one metric may be favored over the other. Taking our bitrate selection problem, if the purpose is to check, for any given bitrate, how often it was truly predicted out of all the total predictions of this class, then precision is the metric that should be taken into account. On the other hand, if the goal is to assure that the actual classes are truly predicted then recall is the preferred metric.

5.4 Results and discussion

In this section, we present and discuss our results, starting with the simulated ABR algorithms over the simulation datasets. Then we move to the realistic datasets, to show the predictability of the unknown commercial ABR algorithms.

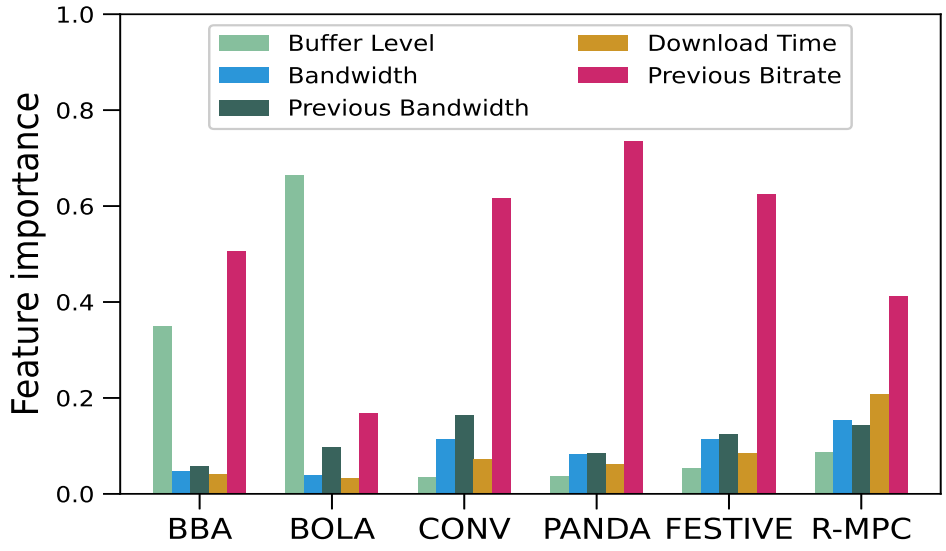


Figure 5.1: Feature importance for the selected ABR algorithms

5.4.1 Simulation-based datasets

5.4.1.1 Feature importance

We start with the feature importance using Random Forest Mean Decrease in Impurity (MDI), presented in Fig. 5.1. The previous bitrate feature is noticeably dominating the decision of the selected ABR algorithms; this acknowledges how much smoothness is important in the bitrate decision for these algorithms. Additionally, with feature importance, we can infer the nature of the ABR: the buffer-based algorithms (BBA and BOLA) show high importance of the buffer level feature over the network features, while the throughput-based algorithms (CONVENTIONAL, PANDA, and FESTIVE) give higher importance to the bandwidth and the download time features. Finally, we can see that R-MPC, which is a hybrid control-based algorithm, relies on all the selected features (buffer and network features) with different, but close, percentages.

Table 5.1: Classification accuracy (%) of ML classifiers on different ABR datasets

	LGS	SVM	RF	DT	GrdBst	AdBst	NB	KNN
BBA	46.97	87.71	99.63	99.60	99.82	41.36	87.55	85.84
BOLA	53.34	71.23	99.88	99.969	99.97	61.13	63.91	70.01
CONV	44.86	93.29	96.38	94.83	96.23	67.63	93.01	91.79
PANDA	41.05	97.22	98.37	95.79	97.37	65.95	97.13	95.42
FESTIVE	36.61	93.04	96.54	94.34	96.23	63.03	92.46	91.79
R-MPC	39.59	32.35	96.97	94.05	98.01	70.64	80.11	52.47

Table 5.2: Precision (%) of ML classifiers for simulation datasets

	LGS	SVM	RF	DT	GrdBst	AdBst	NB	KNN
BBA	90.95	91.43	99.84	99.84	99.84	32.00	92.72	96.46
BOLA	74.22	84.07	99.86	99.00	99.98	49.64	91.53	94.81
CONV	90.83	92.18	96.83	91.77	96.28	56.92	91.56	92.08
PANDA	90.08	97.88	98.18	96.05	97.72	56.34	97.59	97.28
FESTIVE	91.39	93.91	96.15	92.48	96.51	52.95	92.73	93.65
R-MPC	83.69	96.05	97.57	95.31	98.46	67.90	77.23	93.92

5.4.1.2 Metrics evaluation

We first evaluate the different classifiers based on the prediction accuracy as shown in TABLE 5.1. We highlight RF and GrdBst as the best classifiers among the selected ML classifiers, achieving the highest prediction accuracy for all the selected ABR algorithms. For BBA and BOLA, this accuracy is more than 99%, with GrdBst scoring slightly higher than RF (around 99.9%). With the throughput-based algorithms, we notice a small decline in the overall accuracy for both RF and GrdBst, which still show close scores (96-97%). Lastly, for control-based algorithms, represented by *R-MPC*, GrdBst scores the best accuracy, slightly higher than 98%.

From TABLE 5.2 and TABLE 5.3, which respectively present the precision and recall metrics achieved with all the ML classifiers, we see that RF and GrdBst are still achieving the best performance for most of the ABR algorithms. The best

Table 5.3: Recall (%) of ML classifiers for simulation datasets

	LGS	SVM	RF	DT	GrdBst	AdBst	NB	KNN
BBA	86.50	94.63	99.85	99.69	99.85	55.21	91.41	96.93
BOLA	73.31	87.12	99.84	94.05	99.97	68.25	87.73	93.25
CONV	84.05	94.02	96.68	92.18	96.14	73.77	93.71	93.56
PANDA	86.35	98.01	98.17	97.24	97.71	84.05	98.01	98.16
FESTIVE	88.04	93.56	96.12	92.33	96.55	83.74	93.40	94.33
R-MPC	80.35	83.65	97.56	76.73	98.45	65.04	81.18	82.36

scores are obtained for buffer-based algorithms, with RF reaching a bit higher than 99.8% (for both BBA and BOLA), while GrdBst reaches the same percentage with BBA and even better with 99.98% for BOLA. Unfortunately, this good behavior drops a bit with throughput-based algorithms: both RF and GrdBst score around 96% for CONVENTIONAL and FESTIVE. For PANDA, RF achieves 1% more than GrdBst (98% to 97%). Oppositely, with Robust MPC, GrdBst is better than RF with scores 98.4 to 97.5% to each respectively.

5.4.2 Realistic commercial-based datasets

5.4.2.1 Feature importance

As we mentioned earlier, the feature importance is noteworthy in this study since it helps understand the type or the class of the used ABR algorithm. As previously stated, the realistic datasets are collected from unknown ABR algorithms, and here, we only try to guess how they behave without any prior ABR knowledge. The feature importance information, shown in Fig. 5.2, shows that all these commercial ABR algorithms are more sensitive to the previous bitrate feature in the first place, which reveals their smoothness preference. Also, apart from the VoD scenario for $S1$, all the ABR algorithms rely on the throughput measurements for the current and the last downloaded segment, which tells that these algorithms are more

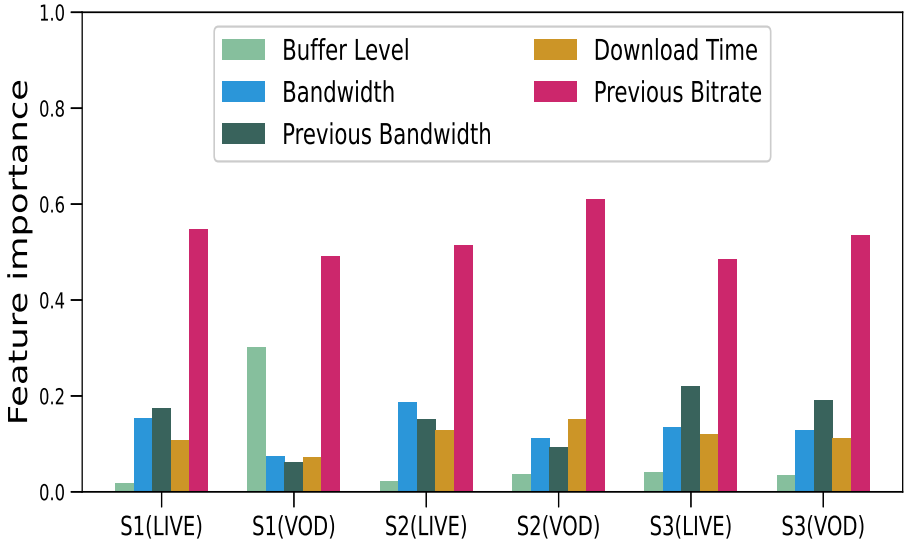


Figure 5.2: Feature importance for the unknown ABR gained from three commercial services

throughput-based. Interestingly, the ABR of $S1$ seems to give higher importance to the buffer level feature in VoD scenario. In contrary to LIVE cases, the buffer size is usually bigger for VoD cases, hence the buffer thresholds are larger and the client might consider a hybrid or a buffer-based ABR; this is for example the case for DASH reference player, where the default ABR, DYNAMIC as referred [88], uses a simple throughput-based algorithm called THROUGHPUT when the buffer levels are low, and uses BOLA when the buffer levels are high.

5.4.2.2 Metrics evaluation

The classification accuracy of each ML algorithm for the different datasets is presented in TABLE 5.4. This table shows that RF, DT, and GrdBst perform the best among the tested ML algorithms. For $S1$ (both Live and VoD), these three algorithms achieve very high accuracy, slightly higher than 98%. For $S2$, RF and GrdBst

Table 5.4: Classification accuracy (%) of ML classifiers on LIVE and VoD commercial services datasets

	LGS	SVM	RF	DT	GrdBst	AdBst	NB	KNN
S1(LIVE)	83.37	93.57	98.30	98.06	98.24	61.69	97.86	98.06
S1(VoD)	84.64	96.43	98.29	97.99	97.26	98.31	98.83	98.67
S2(LIVE)	65.07	86.16	96.89	96.19	97.05	92.85	92.48	92.61
S2(VoD)	85.11	96.66	98.25	97.75	98.37	97.01	93.44	96.00
S3(LIVE)	88.90	88.93	96.26	94.75	96.84	87.17	90.84	95.42
S3(VoD)	84.27	84.28	96.57	95.86	95.04	87.79	95.12	96.35

perform even better than DT with an accuracy around 97% and 98% for Live and VoD, respectively. Finally, $S3$, which has the highest number of quality levels (seven qualities), for Live, we notice that RF and GrdBst achieve high accuracy again, with GrdBst achieving slightly higher accuracy. With VoD, RF performs the best with an accuracy of 96.5%, which is nearly 1% better than DT and GrdBst.

Before moving to precision and recall metrics, we present each of the three best classifiers' confusion matrix, in tables 5.5, 5.6 and 5.7 for each commercial dataset. The first column of each confusion matrix shows the True labels of the classes. In our context, this column indicates the actual ABR decisions for the quality levels (with class 0 representing the minimum bitrate). The first row presents the predicted classes, which again represent the predicted ABR qualities.

Looking at these tables, we see that predicting the highest bitrate is almost accurate. However, the medium bitrates are predicted with lower accuracy. This behavior is seen with the third dataset S3, looking at 5.7, which has seven qualities. This table shows that the highest bitrate is almost always predicted correctly, but the classifiers suffer from falsely classifying the other bitrates. Even though, it is interesting to see that RF does not fall below 90% for per-class correct prediction, compared to 83% and 80% for DT and GrdBst, respectively.

Finally, we present precision and recall metrics in TABLE 5.8, which shows that

Table 5.5: Confusion matrixes of RF, DT and GrdBst for the first commercial dataset

(a) RF for S1 (LIVE)

True	Predicted Class				
%	0	1	2	3	4
0	96.9	2.2	0.3	0.0	0.6
1	0.3	99.4	0.0	0.1	0.2
2	0.2	0.8	96.8	2.0	0.2
3	0.3	0.2	2.3	94.8	2.3
4	0.0	0.0	0.0	0.1	99.9

(b) RF for S1 (VoD)

True	Predicted Class			
%	0	1	2	3
0	99.1	0.6	0.0	0.3
1	1.6	94.6	3.1	0.7
2	0.0	1.2	95.9	2.9
3	0.0	0.0	0.2	99.8

(c) DT for S1 (LIVE)

True	Predicted Class				
%	0	1	2	3	4
0	96.3	2.5	0.3	0.3	0.6
1	0.3	98.3	0.7	0.5	0.2
2	0.0	1.0	94.5	4.5	0.0
3	0.1	0.3	3.1	94.0	2.4
4	0.0	0.0	0.0	0.2	99.8

(d) DT for S1 (VoD)

True	Predicted Class			
%	0	1	2	3
0	98.7	1.0	0.2	0.1
1	2.7	92.9	3.4	0.9
2	0.5	4.5	90.2	4.8
3	0.0	0.1	0.3	99.7

(e) GrdBst for S1 (LIVE)

True	Predicted Class				
%	0	1	2	3	4
0	96.3	3.1	0.0	0.3	0.3
1	0.3	98.6	0.8	0.1	0.2
2	0.0	1.2	96.0	2.7	0.2
3	0.2	0.6	2.8	94.4	2.0
4	0.0	0.0	0.0	0.1	99.9

(f) GrdBst for S1 (VoD)

True	Predicted Class			
%	0	1	2	3
0	99.2	0.6	0.1	0.2
1	2.6	93.4	3.3	0.7
2	0.0	3.9	93.2	2.9
3	0.0	0.0	0.2	99.8

Table 5.6: Confusion matrixes of RF, DT and GrdBst for the second commercial dataset

(a) RF for S2 (LIVE)

True	Predicted Class			
%	0	1	2	3
0	99.7	0.2	0.0	0.1
1	1.3	97.9	0.1	0.6
2	0.5	2.3	93.6	3.5
3	0.2	0.3	0.5	99.0

(b) RF for S2 (VoD)

True	Predicted Class		
%	0	1	2
0	99.4	0.5	0.1
1	0.7	97.0	2.2
2	0.0	0.4	99.6

(c) DT for S2 (LIVE)

True	Predicted Class			
%	0	1	2	3
0	99.0	0.7	0.1	0.2
1	1.2	97.2	0.9	0.6
2	0.7	0.6	94.3	4.4
3	0.2	0.4	1.0	98.4

(d) DT for S2 (VoD)

True	Predicted Class		
%	0	1	2
0	98.8	1.1	0.1
1	1.0	96.4	2.5
2	0.0	0.6	99.4

(e) GrdBst for S2 (LIVE)

True	Predicted Class			
%	0	1	2	3
0	99.5	0.4	0.1	0.1
1	1.3	97.5	0.6	0.7
2	0.6	4.3	89.5	5.6
3	0.2	0.4	0.7	98.7

(f) GrdBst for S2 (VoD)

True	Predicted Class		
%	0	1	2
0	99.5	0.4	0.1
1	0.6	97.9	1.5
2	0.0	0.4	99.6

Table 5.7: Confusion matrixes of RF, DT and GrdBst for the third commercial dataset

(a) RF for S3 (LIVE)

True	Predicted Class						
%	0	1	2	3	4	5	6
0	90.2	4.3	0.5	1.0	0.8	0.8	2.5
1	2.5	93.5	1.2	1.1	0.7	0.3	0.7
2	0.7	3.6	90.7	2.1	0.9	0.9	1.1
3	0.1	0.5	2.5	93.4	2.8	0.3	0.5
4	0.1	0.2	0.1	3.3	93.3	2.2	0.7
5	0.1	0.0	0.2	0.5	5.6	90.3	3.3
6	0.0	0.0	0.0	0.0	0.0	0.3	99.6

(b) RF for S3 (VoD)

True	Predicted Class						
%	0	1	2	3	4	5	6
0	90.2	4.3	0.5	1.0	0.8	0.8	2.5
1	2.5	91.5	1.2	3.1	0.7	0.3	0.7
2	0.7	1.6	90.7	4.1	0.9	0.9	1.1
3	0.1	0.5	2.5	93.4	2.8	0.3	0.5
4	0.1	0.2	0.1	3.3	93.3	2.2	0.7
5	0.1	0.0	0.2	0.5	5.6	90.3	3.3
6	0.0	0.0	0.0	0.0	0.0	0.3	99.6

(c) DT for S3 (LIVE)

True	Predicted Class						
%	0	1	2	3	4	5	6
0	90.0	4.3	1.5	2.3	0.5	0.5	1.0
1	4.7	87.9	3.4	2.8	0.7	0.4	0.1
2	2.0	6.1	83.3	5.2	2.2	0.7	0.4
3	0.6	1.0	3.6	89.6	4.3	0.6	0.4
4	0.4	0.4	0.6	4.7	89.5	3.8	0.5
5	0.1	0.0	0.4	0.8	7.7	87.6	3.3
6	0.1	0.0	0.1	0.0	0.2	0.6	98.9

(d) DT for S3 (VoD)

True	Predicted Class						
%	0	1	2	3	4	5	6
0	88.0	6.8	0.8	1.3	0.5	0.5	2.3
1	3.9	88.2	2.3	3.5	1.0	0.4	0.6
2	0.7	5.6	87.0	3.3	1.6	0.9	0.9
3	0.1	0.6	2.5	92.8	3.5	0.1	0.5
4	0.0	0.2	0.9	5.8	89.5	2.9	0.6
5	0.2	0.0	0.0	0.4	7.2	88.5	3.7
6	0.1	0.1	0.1	0.0	0.1	0.4	99.3

(e) GrdBst for S3 (LIVE)

True	Predicted Class						
%	0	1	2	3	4	5	6
0	90.4	3.2	0.0	3.8	0.0	2.6	0.0
1	2.7	90.1	2.3	4.1	0.4	0.4	0.0
2	0.0	2.3	87.3	5.9	2.7	0.9	0.9
3	0.0	0.7	2.6	93.5	2.9	0.2	0.2
4	0.1	0.1	1.0	3.5	91.1	3.8	0.3
5	0.0	0.1	0.0	0.5	4.3	92.7	2.4
6	0.1	0.0	0.0	0.0	0.0	0.2	99.8

(f) GrdBst for S3 (VoD)

True	Predicted Class						
%	0	1	2	3	4	5	6
0	86.3	2.1	2.9	0.2	2.6	2.6	3.3
1	3.3	88.8	3.6	4.3	0.0	0.0	0.0
2	0.5	2.9	80.6	14.9	0.6	0.3	0.3
3	0.2	0.2	4.4	92.5	2.5	0.1	0.3
4	0.0	0.0	0.2	3.7	92.5	3.4	0.1
5	0.0	0.0	0.1	0.3	4.2	92.4	3.0
6	0.0	0.0	0.0	0.0	0.0	0.3	99.6

Table 5.8: Precision and Recall(%) using RF, DT and GrdBst classifiers on LIVE and VoD commercial services datasets

	RF		DT		GrdBst	
	Precision	Recall	Precision	Recall	Precision	Recall
S1(LIVE)	99.16	99.17	98.87	98.86	99.16	99.17
S1(VoD)	99.21	99.19	99.12	99.11	99.20	99.16
S2(LIVE)	98.98	98.99	98.44	98.43	98.71	98.71
S2(VoD)	99.32	99.33	98.96	98.95	99.33	99.33
S3(LIVE)	97.65	97.83	96.87	96.83	97.89	97.18
S3(VoD)	97.99	97.99	96.08	96.06	96.95	96.94

each RF, DT, and GrdBst achieves high precision and recall, with RF and GrdBst slightly better than DT in some scenarios.

In addition, processing latency is another important metric, especially for online learning. It is even more important for video streaming when the download decisions rely on bitrate prediction. We present in TABLE 5.9 the training and prediction times for the different ML algorithms to guarantee the latency requirements. The training time is computed over the whole dataset, and the prediction time is averaged over 20% of the datasets. This table shows that DT is faster than both RF and GrdBst, for training and prediction phases. GrdBst is computationally the most demanding algorithm for the training phase, though it performs well and slightly better than RF, for the prediction phase.

5.5 Conclusion

In this chapter, we showed the possibility of learning the behavior of the ABR logic thanks to machine learning techniques and supervised learning in particular. We performed our study over different ABR algorithms, including both classic ABR algorithms and real-world closed-source deployments. From the gained results, we found that Random Forest and Gradient boost algorithms are able to achieve a very

Table 5.9: Processing time of ML classifiers on Live and VoD commercial services datasets

ML algo	Training		Prediction	
	mean (s)	std(s)	mean (ms)	std (ms)
LGS	1.69	0.99	0.19	0.05
SVM	0.99	0.46	56.69	38.93
RF	1.55	0.38	10.74	1.77
DT	0.04	0.016	0.09	0.02
GrdBst	14.64	6.71	7.51	3.47
AdBst	1.61	0.48	33.80	27.45
NB	0.02	0.003	0.21	0.03
KNN	0.41	0.015	41.68	4.82

high prediction accuracy, using only the basic information provided as input to the application layer. This work serves the prefetching-based P2P streaming, as it is compatible with our target use case of P2P and ABR, in which the ABR and P2P stack do not know about each other. In the next chapter, we will use this work to make ABR predictions to improve the prefetching and caching efficiency.

Chapter 6

Adaptive BitRate-aware prefetching methods in P2P

6.1 Introduction

Most of the P2P video delivery networks rely on prefetching techniques to exploit the available resources efficiently and get as much data as possible from other peers, then keep it in a local memory either to serve the peer itself, or other peers in the network. In chapter 4, we discussed the challenges of prefetching-based environments when delivering adaptive streaming contents, especially the compatibility with the client-side bitrate adaptation logics. We showed how the player ABR gets confused when it receives P2P segments immediately from the local cache, consequently mistakenly over-estimating the bandwidth, leading to unsuitable bitrate decisions. We proposed Response-Delay, a solution to smooth the delivery of the local cached P2P segments, respecting the video player logic by waiting some time before returning the cached segments to the video player. Interestingly, the idea of delaying the segments by adjusting the download time has opened another door for further improvements. On one hand, delaying the response gives more

time to get more data from other peers; hence it affects the prefetching. On the other hand, adjusting the segments responses leads the ABR to select different decisions, and hence, it controls the player ABR in one way or the other. To turn these two effects into good behavior, we try to use the approach of response delay wisely to enhance the prefetching efficiency.

However, bringing prefetching together with ABR has some challenges, since with ABR, the video segments are encoded into different qualities. Therefore, the prefetching logic requires further knowledge of the user's qualities of interest, to avoid wasting resources on downloading undesired segments. Furthermore, the ABR may take a decision to switch down the quality while a higher quality may already exist in the P2P cache, and a wise decision would be to use the higher quality and avoid wasting bandwidth on downloading a lower quality.

In both scenarios, the prefetching logic needs to be aware of the player ABR, which may be closed source and completely unknown to the P2P stack. Hence, we recall the proposed idea in Chapter 5, of learning the ABR algorithm from its input features only and using the learned model afterward to help designing an ABR-aware prefetching environment.

Following this debate, this chapter binds together the ideas proposed in Chapter 4 and Chapter 5 to investigate the possibility of enhancing the prefetching technique. The main idea is to anticipate the quality switches using the ABR ML model, and from then, investigate two possible actions to be taken. The first action is trying to download the anticipated quality from other peers, then downloading it prior to the player's request. The second action is to keep the prefetched quality for QoE reasons, and further step requires finding the optimal delay for which the ABR responds by selecting the cached quality. For the second action, we prefer the ABR to select the prefetched quality only when it is higher than what the ABR would request, so we do not degrade the average quality.

6.1.1 Contributions

The contributions of this chapter are:

- introduce an innovative ABR-aware prefetching technique that predicts, while delaying the responses, the next qualities and tries to fetch them from peers in time which improves both QoE and P2P efficiency.
- propose an innovative ABR-controlling technique to force the ABR to select different qualities, by adjusting the segments response times to make the desired quality switch.
- efficiently utilize the P2P cache, by forcing the ABR to select the cached qualities, especially when the ABR decides to down-switch the quality while having the segments cached in a higher quality.

6.2 Proposed solution

6.2.1 ML-based prefetching

As stated earlier, our first goal is to be able to predict the ABR quality switches and try to get the new qualities in advance. With response delay, the player waits for some time before receiving the cached segments, and so we can use this spare time to fetch more P2P data for the next segments. Hence we try to predict what the ABR will select for the next segment, after receiving the current delayed one.

In chapter 5, we discussed how to formulate the ABR prediction as a classification problem (see section 5.2), using a set of inputs only.

In this work, this set of input features is represented by the vector X_{n+1} (Line 10 in Algorithm 6.1). This means that to predict the ABR decision for segment $n+1$, the model uses the buffer level b_{n+1} as measured when requesting the segment $n+1$,

the bandwidth bw_n as measured after downloading the segment n, the previous bandwidth bw_{n-1} registered for segment $n-1$, the download time for segment n represented by the response delay d_n, and the last bitrate decision r_n

When delaying the segment n, the model tries to predict the quality of the next segment $\hat{r}_{n+1}$ using X_{n+1}, which includes some delay-dependent features such as the buffer level, the measured bandwidth and, of course, the download time.

To simplify, Lines 8 and 9 in Algorithm 6.1 show the relation between the response delay with each of the buffer and the bandwidth estimation. Since the ABR sees the response delay d_n as the time it took the segment to be downloaded, the buffer drains for amount of d_n, then it increases by one segment duration τ when the segment is received. Meanwhile, the bandwidth is estimated by measuring the segment size over d_n.

Hereafter, we pass these inputs to the ML predictor, Random Forest (RF) in our work, to predict the next bitrate decision $\hat{r}_{n+1}$ (Line 11 in Algorithm 6.1). The predicted quality is then used by the P2P downloader (presented in Algorithm 3.3 in Chapter 3) to update the prefetching bitrate for the next segments.

6.2.2 ABR controlling with Response Delay

The second goal of our proposal is to control the player's ABR algorithm by forcing it to request another quality level since modifying the segment responses has the effect of changing the ABR decision externally, without even modifying the ABR logic itself. In this work, we try to prevent the ABR from down switching the quality, especially when we have the segments already prefetched at a higher quality, thus saving the resources and improving the average quality. To this end, we try to find the optimal value for response delay that leads the ABR to pick the desired quality r^*, which is the maximum cached quality for this segment in our study. To do so, we scan for values of delay d'_n such that $0 < d'_n < \tau$; we chose τ for the delay

upper bound as long delays are undesired and may cause rebuffering, or even lead the ABR to switch down the quality. Then, we update X_{n+1} according to d'_n by re-estimating the buffer level and the bandwidth according to Lines 8 and 9 in Algorithm 6.1. It must be noted that the optimal value is not necessarily one unique value; rather, there might exist a range of values that drive the ABR to the same decision.

Algorithm 6.1 Logic of ML-based prefetching and quality control

1: $n \leftarrow$ the player requested segment
2: $st[n] \in \mathbb{S} : \mathbb{S} = \{AC, A\overline{C}, \overline{A}\} \leftarrow$ cached segment state
3: $R^*[n] \leftarrow$ the bitrates of segment n in P2P cache
4: $\hat{r}_n \leftarrow$ prediction of the ABR bitrate decision for segment n
5: $S_n, r_n, t_{dw_n} \leftarrow$ Size and bitrate and download time for segment n.
6: $b_n, bw_n \leftarrow$ buffer level and measured bandwidth for segment n.
7: $d_n \leftarrow$ response delay for segment n, received from Algorithm 6.2
8: $b_{n+1} = b_n + \tau - d_n$
9: $bw_n = S_n / d_n$
10: $X_{n+1} = [b_{n+1}, bw_n, bw_{n-1}, d_n, r_n]$
11: $\hat{r}_{n+1} = RF(X_{n+1})$
12: **if** $\hat{r}_{n+1} < max(R^*[n+1])$ **then** ▷ Undesired down-switch
13: $D_n = \emptyset$
14: **for** d'_n in $0 : \tau$ **do**
15: *Repeat steps 8, 9, 10 and 11* for $d_n = d'_n$
16: **if** $\hat{r}_{n+1} = max(R^*[n+1])$ **then**
17: $D_n = \{..., d'_n\}$
18: **end if**
19: **end for**
20: **return** $avg(D_n)$
21: **end if**
22: **if** $\hat{r}_{n+1} \notin R^*[n+1]$ **then** ▷ ABR-aware prefetching
23: *Try to prefetch new quality* $\hat{r}_{n+1}$
24: **end if**

One possibility is to select the minimum or the maximum delay values of this range. In fact, the ABR ML model is not 100% accurate, and it might mispredict the next quality. We tested both options, and we found that having the minimum or too-short delay may lead to the bandwidth over-estimation issue, while having

the maximum delay may lead the ABR to switch down to a lower quality than the desired one. The best performance we had is with the average value of all the delays that lead to $\hat{r}_{n+1} = r^*$ (Lines 16, 17 and 20 in Algorithm 6.1).

Algorithm 6.2 Orchestrator logic with response delay and ABR prediction

1: $n \leftarrow$ the player requested segment
2: $st[n] \in \mathbb{S} : \mathbb{S} = \{AC, A\overline{C}, \overline{A}\} \leftarrow$ cached segment state
3: **if** $st[n] = AC$ **then**
4: **if** time needed before delivering the segment **then**
5: $t_{dw} = d_n$ ▷ Response-Delay
6: **Call Algorithm 6.1 for segment delay =** d_n
7: **else**
8: $t_{dw} = \delta$
9: **end if**
10: **else if** $st[n] = A\overline{C}$ **then**
11: $dataRange = S[n] - downloadedData[n]$
12: $sendCdnRequest(dataRange)$
13: **if** time needed before delivering the segment **then**
14: $t_{dw} = t_{cdn} + d_n$ ▷ Response-Delay
15: **Call Algorithm 6.1 for segment delay** $d_n = t_{cdn} + d_n$
16: **else**
17: $t_{dw} = \delta + t_{cdn}$
18: **end if**
19: **else if** $st[n] = \overline{A}$ **then**
20: $sendCdnRequest(S[n])$
21: $t_{dw} = t_{cdn}$
22: **end if**
23: wait for t_{dw} to get and deliver the segment
24: **return** $S[n]$

6.2.3 Applying ML-based prefetching and quality control with Response Delay

In this section, we show how to integrate the proposed methods into the orchestrator unit, which handles the P2P connection part, including scheduling, caching, and delaying the responses (Algorithm 4.1 from Chapter 4).

We update the Orchestrator logic by adapting to the ML-based prefetching and

quality control part, as shown In Algorithm 6.2.

We have to deal with two states of cached P2P segments: available and completed (AC) and available but uncompleted ($A\overline{C}$). Clearly, the delay for AC segments will be seen as a download time by the ABR algorithm; thus, it is safe to use this delay to estimate the buffer and the bandwidth states to predict the next segment quality (according to Lines 8-11 in Algorithm 6.1). However, for $A\overline{C}$ segments, the ABR measures a download time t_{dw} which is equal to the time it takes to finish the segment from CDN (t_{cdn}) plus the delay for the P2P part d_n. Consequently, for $A\overline{C}$, we need to consider $d_n = t_{dw}$ in Algorithm 6.1 to keep the logic correct.

6.3 Experimental setup

We investigate the performance of our proposals on the five classic ABR algorithms: BBA [29], BOLA [31], CONVENTIONAL [25], PANDA [25], FESTIVE [27], to stay consistent with the presented work in Chapters 4 and 5. We implemented all these algorithms using the MATLAB-based model presented in Chapter 3. In Chapter 5, each ABR algorithm has a unique trained ML model, that we can reuse in this work to make predictions. It must be noted that these models are trained on *Red Bull Play Streets* video content from DASH Dataset Sequences[1]. This content is one-hour duration, segmented into 2-second segments, and encoded at six different bitrates: 0.3, 0.7, 1.19, 1.99, 2.99, and 4.981 Mbps. Thus, we will use the same video content in the following tests to be able to reuse the trained models from Chapter 5.

The stats were collected from 10 users who were watching the same content and experiencing different bandwidth profiles. The bandwidth profiles were chosen from some publicly available sets of real 3G [91] and 4G [100] bandwidth traces;

[1] Available at https://dash.itec.aau.at/download/

excluding the bandwidth traces that were used for training the ML models.

For the delay method, we used NetDel algorithm from Chapter 4, as this method showed a better overall QoE and P2P behavior. However, the proposals of this chapter are independent of the Response-Delay algorithm itself; they use the value of delay as an input only. Finally, to illustrate the behavior of our proposal, we use a controlled high-low-high trace, as later shown in section 6.4.1.

6.4 Results and discussion

To show the benefits of our proposals, we first explain the behavior over two examples for one throughput-based and one buffer-based algorithm. We consider four different scenarios for each delay method: NetDel is a normal hybrid CDN/P2P streaming applying Response-Delay without any ML-based prefetching or quality control. MLQF is a hybrid CDN/P2P streaming applying Response-Delay with ML-based prefetching for the P2P segments. MLQC represents a hybrid CDN/P2P streaming applying Response-Delay with ML-based quality control that aims to force the ABR to select the highest cached quality. Lastly, we merge the two ML approaches in MLQFC, which performs ML-based prefetching with quality control to select the highest cached quality.

6.4.1 Explaining MLQF and MLQC over examples

We first present a sample run to illustrate how MLQC and MLQF work with NetDel algorithm (presented in Section 4.2.2.2). For this example, we use a high-low-high bandwidth trace, where the bandwidth changes every 20 seconds to 6, 4.4, 2.5, 1.5, 1, 0.6 Mbps, respectively. This run involves 10 peers, the first 9 peers apply the regular NetDel without modifications, and the last 10th peer, who joins the session the last, apply one of NetDel, MLQF, and MLQC per simulation run.

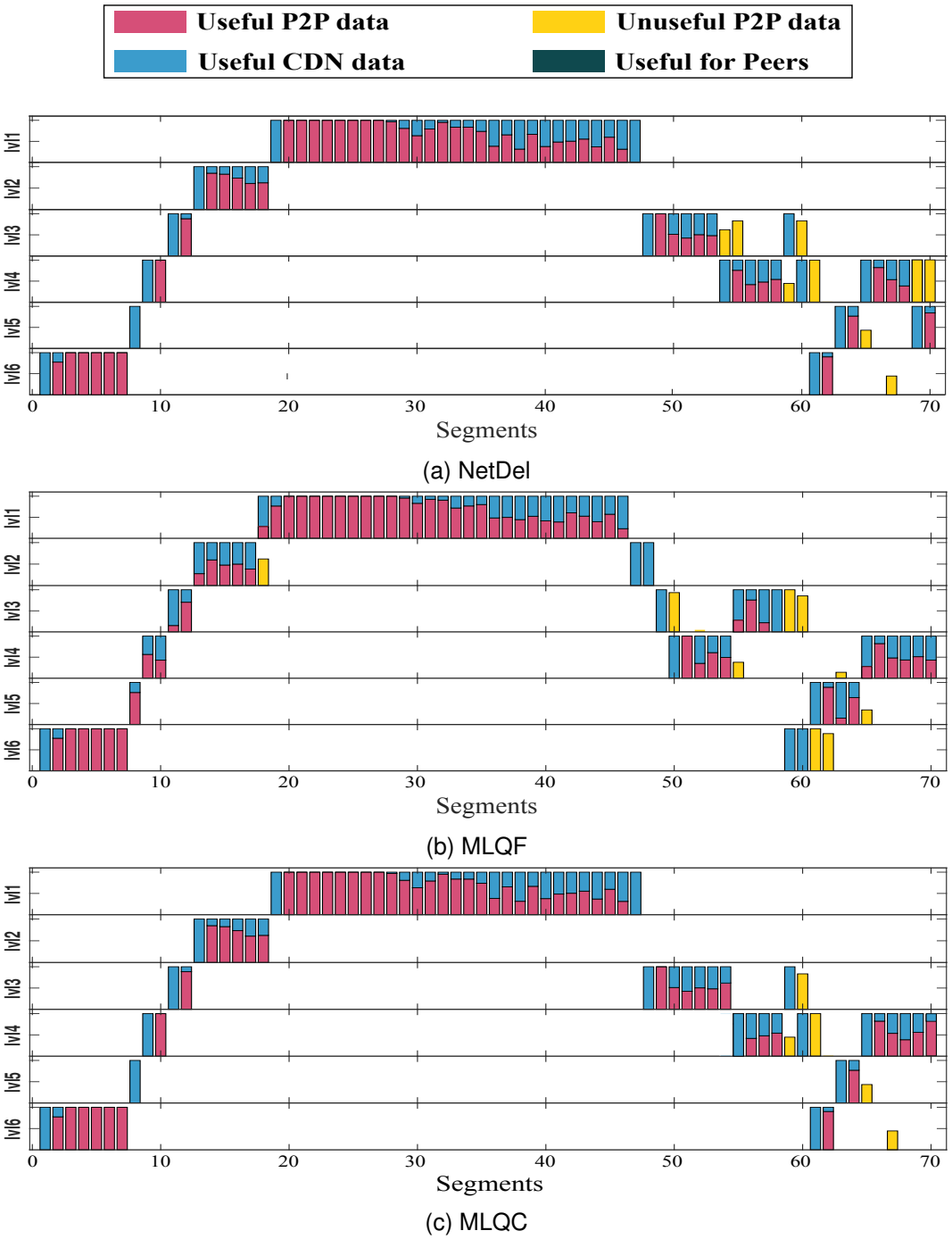
Then, we discuss the behavior of the 10th peer for each configuration using the cache status visualization presented in Chapter3. In this example, we pick BBA as a buffer-based ABR, CONVENTIONAL as a non-conservative throughput-based ABR, and PANDA as a conservative throughput-based ABR on the sample run for 70 segments.

Starting with BBA, we see some room for improvement with NetDel scenario (Figure 6.1a). First, when the ABR decides to switch the quality, the new segment will be requested from CDN immediately if it does not exist in the P2P cache, such as segments 8, 9, 11, 13 .., etc. Second, it happens that the ABR down switches the quality of the next segment while having this segment (partially or fully) cached, as we can see for segments 54, 55, 69, 70.

We expect MLQF to help in resolving the first issue and MLQC to deal with the second one. Indeed, Figure 6.1b shows that when applying MLQF, some P2P data appears on segments 8, 9, 11, 13, 18, 55, and 65, which are the requested segments on the new quality. Hence, MLQF allows predicting the track switches in advance, which in turn serves the P2P prefetching system to fetch the new quality ahead of the ABR request.

MLQC manages to force the ABR to select the cached quality when it is already fetched in the P2P cache. Going back to NetDel scenario, we see that segments 54, 55 of quality 3 and segments 69, 70 of quality 4 are already downloaded in the P2P cache, and yet the ABR selects a lower quality for these segments as a side effect of NetDel. MLQC is designed particularly to deal with this case, and Figure 6.1c shows that the ABR does not down switch the quality for these segments anymore. It is rather forced to stay at the prefetched quality by applying the delay suggested by MLQC.

For CONVENTIONAL algorithm, NetDel scenario (Figure 6.2a) shows that segments 5, 6, 7, 8, 16, and 17 are fetched on quality 1, then the ABR switches down



(a) NetDel

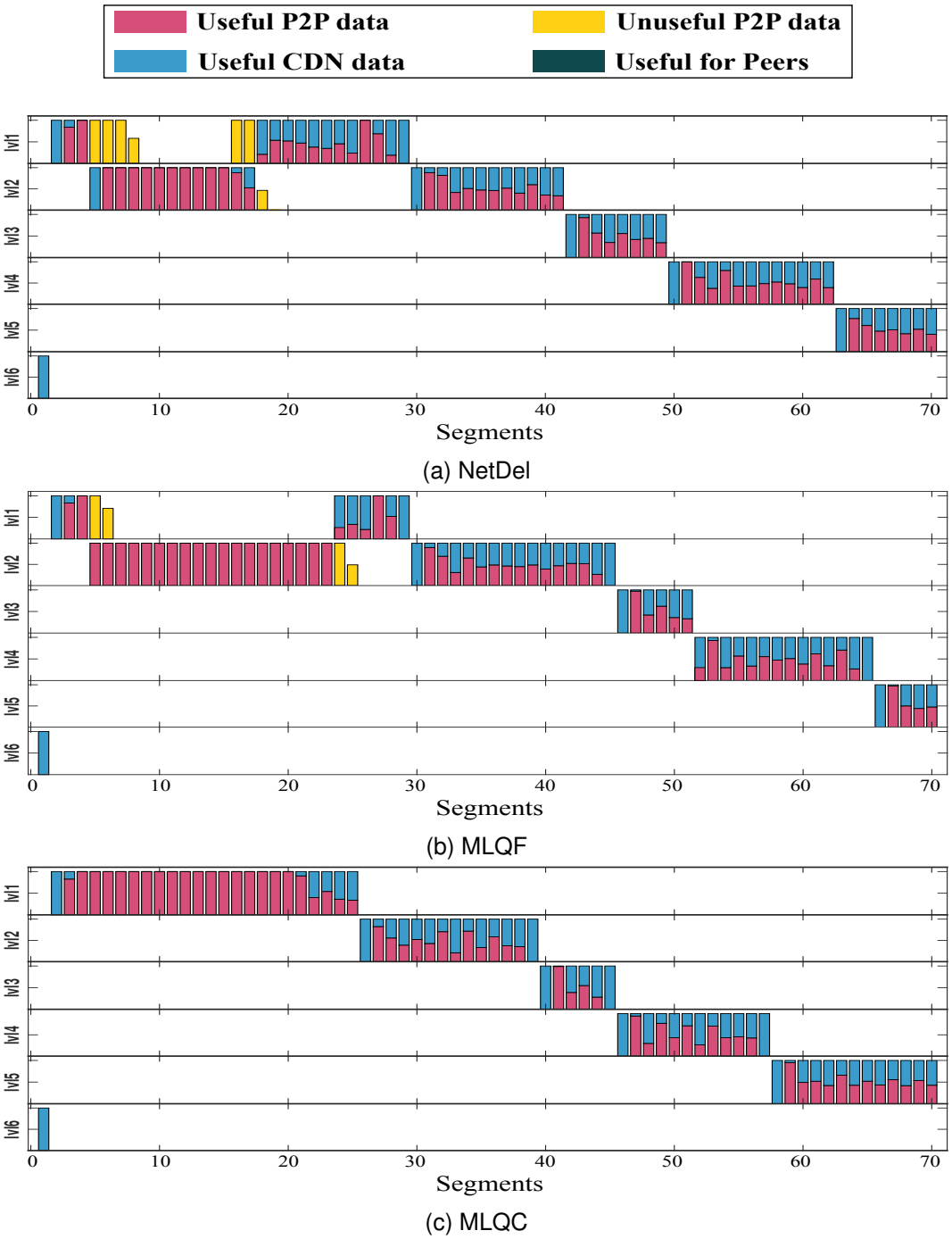
(b) MLQF

(c) MLQC

Figure 6.1: Example of cache state for one peer running BBA and applying NetDel with (a) NetDel, (b) MLQF and (c) MLQC

to quality 2 when receiving segment 4 from P2P cache. With MLQF (Figure 6.2b), the scheduler predicts the down switch at segment 4, while the ABR is waiting for this segment to be delivered from P2P. Accordingly, the scheduler stops asking for quality 1 and switches to quality 2. As we can see, quality 2 appears in the P2P cache starting from segment 4, and less overhead of quality 1. Then at segment 23, it predicts a quality up switch and tries to fetch part of segment 24 at quality 1 and so on. MLQC, avoids the quality down switch and manages to keep the ABR at the highest fetched quality as long as possible. In this example (Figure 6.2c), the P2P module detects the down switch at segment 4, it then checks that segment 5 is fetched at a higher quality, and it optimizes the response delay of segment 4, so that it prevents the ABR from requesting quality 2, and force it select quality 1.

PANDA, which is a conservative throughput-based algorithm, shows less quality switches compared to BBA and CONVENTIONAL for NetDel scenario (Figure 6.3a). Although it has segments 10, 11, 12, 13, and 14 prefetched at quality 1, it still down switches to quality 2 at segment 10. MLQF (Figure 6.3b) allows predicting this down switch when the ABR is waiting for segment 9 to be delivered from P2P cache. The scheduler then no longer requests any more data from P2P at quality 1, and it switches to quality 2 starting from segment 10. Unfortunately, MLQC does not manage to keep PANDA at a higher quality when it detects the quality down switch at segment 10. The main reason behind this behavior is that the conservative approach prevents PANDA from adapting to the instant variations in bandwidth. Instead, it waits for some time/segments before it changes its decision; this explains why we could not force PANDA's decision by changing the response delay for only one segment.

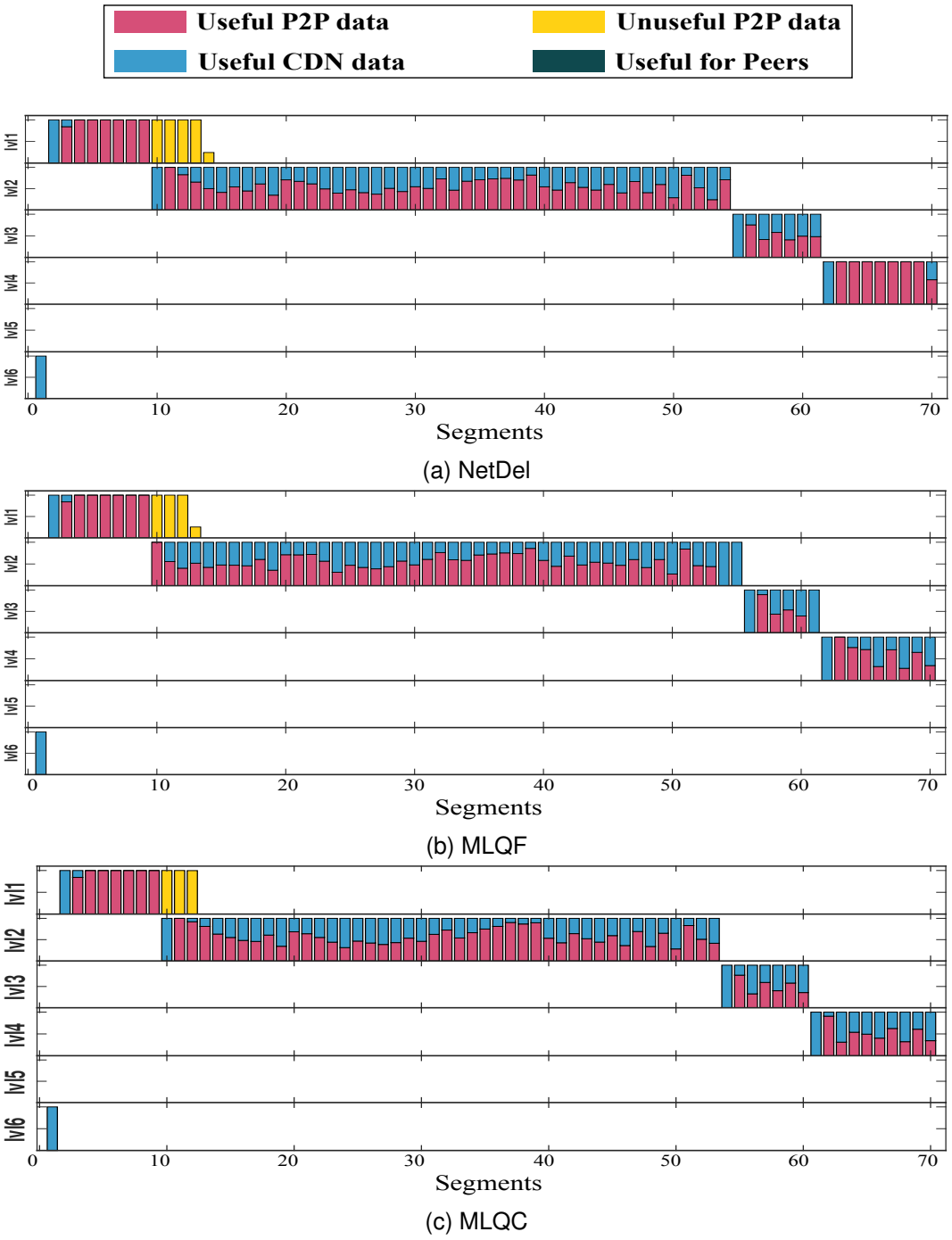


(a) NetDel

(b) MLQF

(c) MLQC

Figure 6.2: Example of cache state for one peer running CONVENTIONAL and applying NetDel with (a) NetDel, (b) MLQF and (c) MLQC



(a) NetDel

(b) MLQF

(c) MLQC

Figure 6.3: Example of cache state for one peer running PANDA and applying NetDel with (a) NetDel, (b) MLQF and (c) MLQC

6.4.2 Metrics evaluations

We now evaluate the performances of MLQF, MLQC and MLQFC in terms of QoE and P2P metrics (presented in Chapter 4 Section 4.3.2). In each of these scenarios, all peers use the according method (MLQF, MLQC or MLQFC) except the first peer, who is connected to CDN link only. All results are averaged over all peers and all different 3G and 4G traces, except for P2P metrics, for which the first peer, which is connected to CDN link only, is excluded.

Starting with QoE metrics, Figure 6.4 shows that MLQC manages to improve the average rate metric compared to NetDel and MLQF, for all algorithms but PANDA. As we explained, MLQC tries to find the response delay of the P2P segment that would lead to a certain ABR decision; the highest prefetched quality in our study. This explains the improvement in the average rate for most ABR algorithms. For PANDA, as we mentioned in Section 6.4.1, MLQC does not have a lot of impact modifying only one segment since PANDA is conservative enough to not react immediately. MLQFC also improves the average quality compared to NetDel and MLQF, and slightly more than MLQC as it combines the benefits of both MLQC and MLQF.

Stability, as mentioned in Chapter 2, is a major issue for buffer-based algorithms. Interestingly Figure 6.5 shows that MLQF, MLQC and MLQFC improve the stability of BOLA algorithm, with slight improvements on the other ABRs. With MLQC and MLQFC, the ABR does not down-switch when the next segment is already prefetched at a higher quality, which explains why we have this improvement in stability.

For the rest of QoE metrics, MLQC, MLQF and MLQFC do not affect the overall performance. They all perform similar to NetDel for Smoothness, Continuity, and Consistency (Figures 6.6, 6.7 and 6.8 respectively).

Moving to P2P metrics, Figure 6.9 shows that MLQC, MLQF, and MLQFC improve the P2P offload, with MLQF, MLQFC slightly higher than MLQC. As both MLQC and MLQF aim at improving the P2P usage: MLQC tries to use what is already downloaded from P2P, while MLQF tries to prefetch what the ABR would request next, with MLQFC combining the merits of both. For PANDA, MLQC does not change the performance for the same reason we mentioned when explaining the average rate metric.

For the Peer Efficiency metric, shown in Figure 6.10, both MLQC and MLQF reduce the P2P overhead with different but close scores. MLQC reduces the overhead by leading the ABR to use the already prefetched segments at a higher quality, whereas MLQF tries to prefetch segments at the predicted qualities and stops downloading the old ones, which leads to reducing the overhead. MLQFC achieves the best efficiency as it reduces the overhead both ways.

Lastly, Figure 6.11 shows that MLQF, MLQC and MLQFC slightly improve Peer Pool Efficiency metric. In fact, this metric depends on the overhead segments downloaded by each peer, which might be requested later by other peers. We saw that MLQC and MLQF improve the Peer Efficiency by reducing the peer overhead, which explains the slight improvement with MLQF, MLQC and MLQFC compared to NetDel, as less unused data left to share with other peers.

6.5 Conclusion

In this chapter, we discussed the challenges of prefetching-based P2P networks when working with adaptive streaming. We proposed to improve the Response-Delay technique, that we introduced in Chapter 4, in order to take the P2P cache status into consideration. Thanks to Response-Delay, and the proposed ML-based ABR prediction model in Chapter 5, we showed that we are able to predict what the

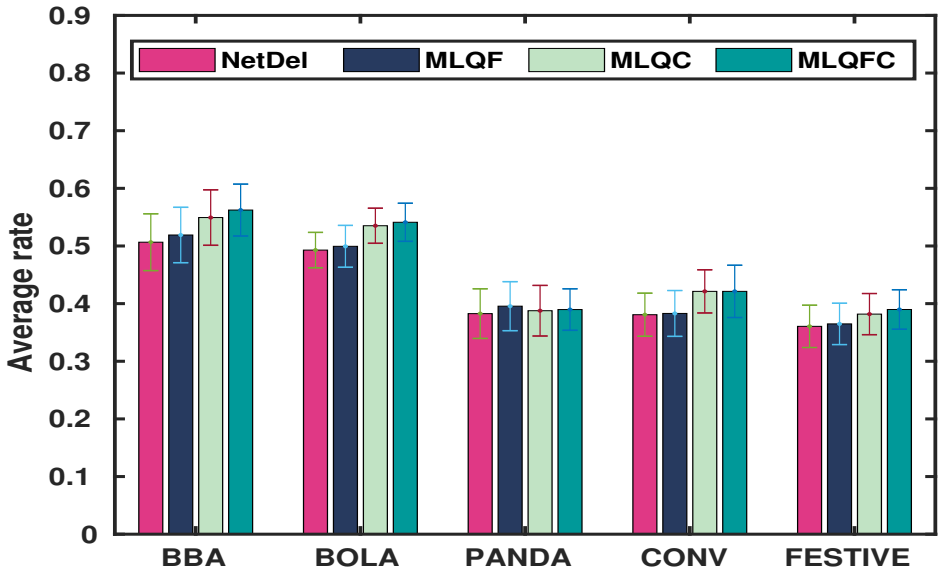


Figure 6.4: Average rate for NetDel

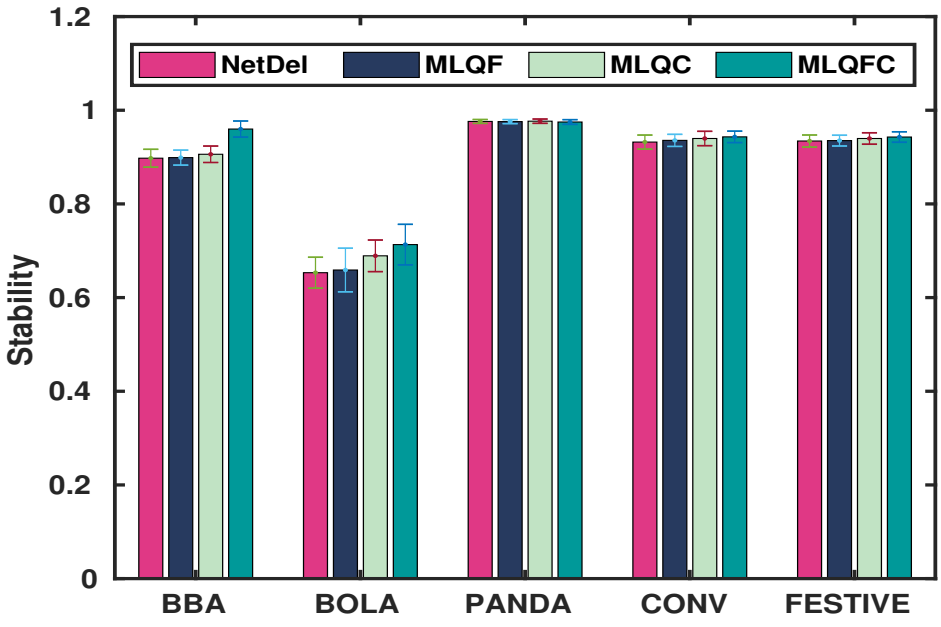


Figure 6.5: P2P metrics: stability

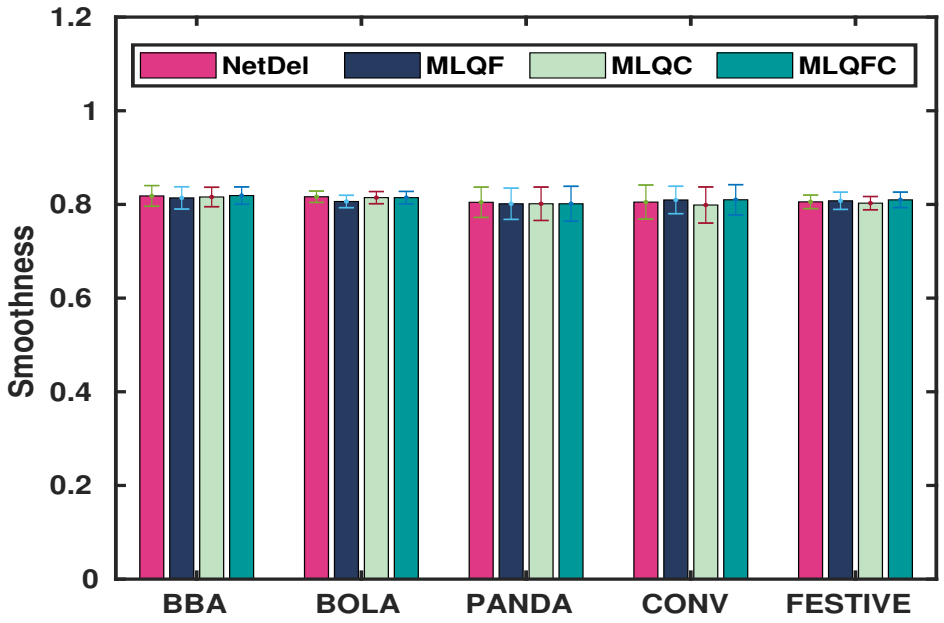


Figure 6.6: P2P metrics: smoothness

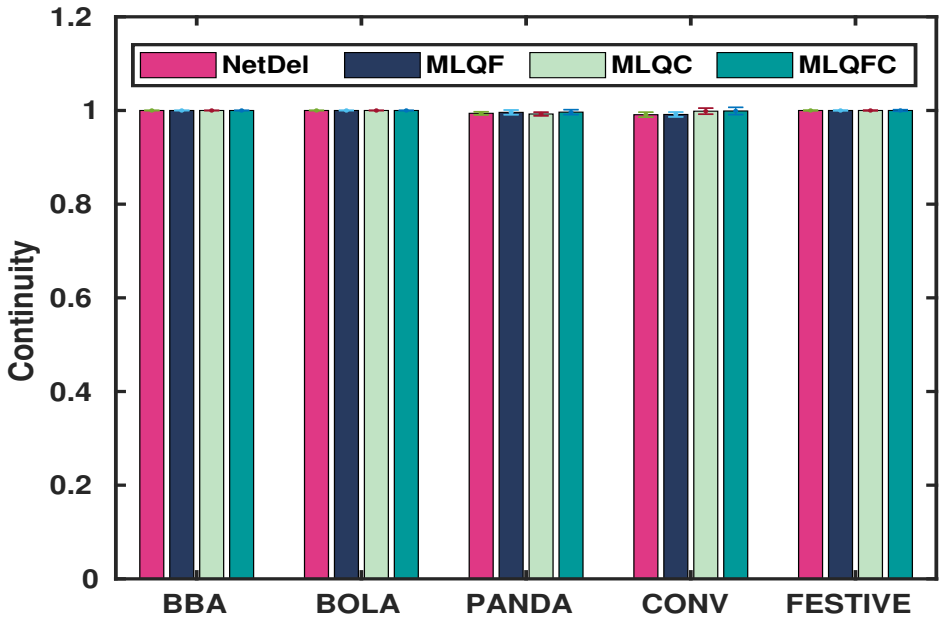


Figure 6.7: P2P metrics: continuity

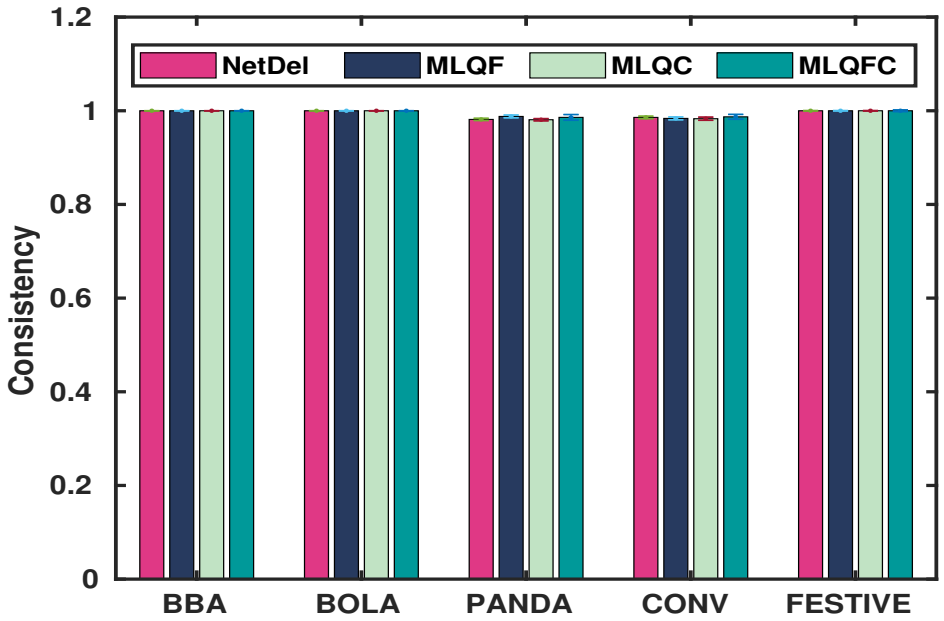


Figure 6.8: P2P metrics: consistency

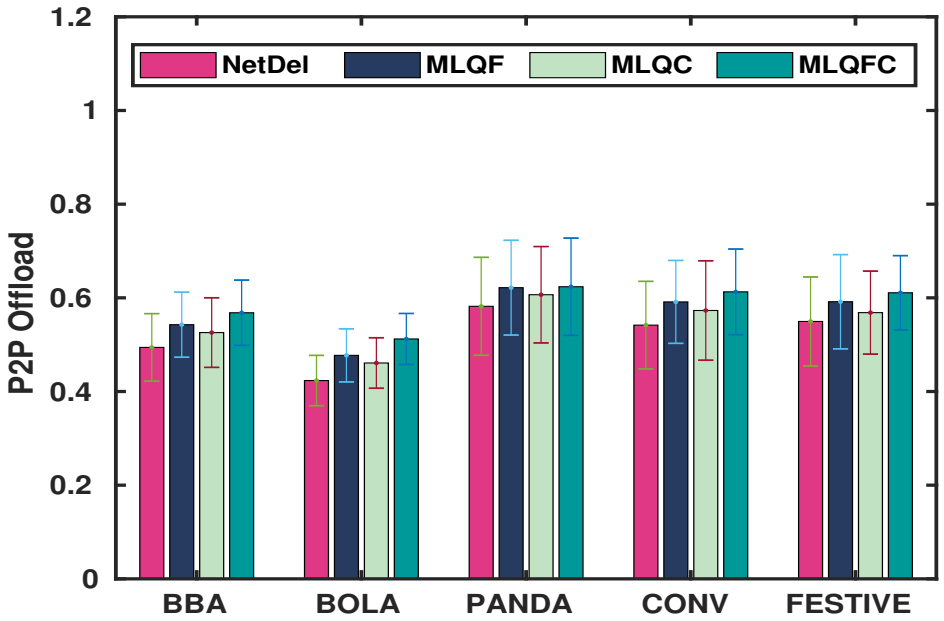


Figure 6.9: P2P metrics: offload

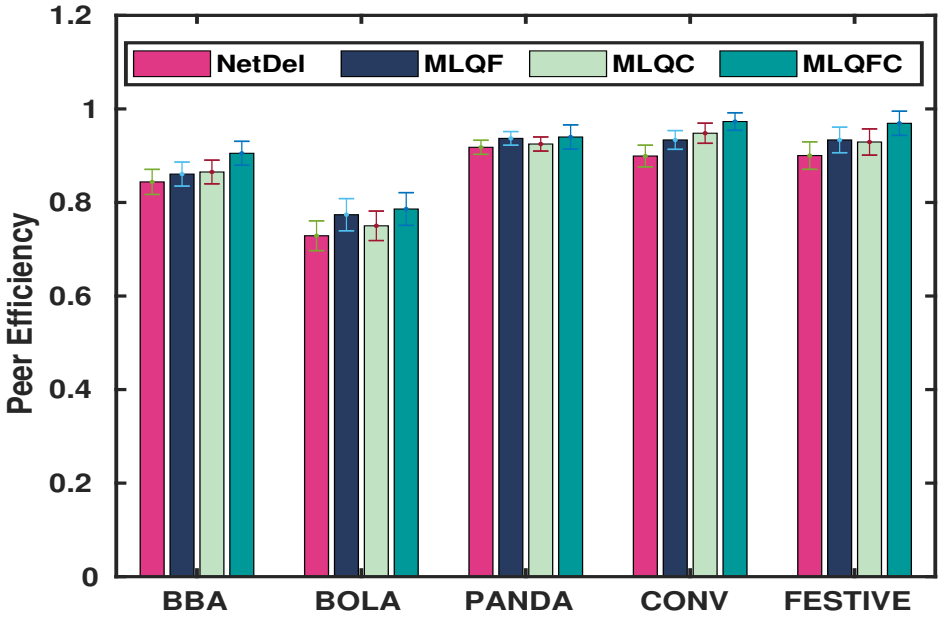


Figure 6.10: P2P metrics: peer efficiency

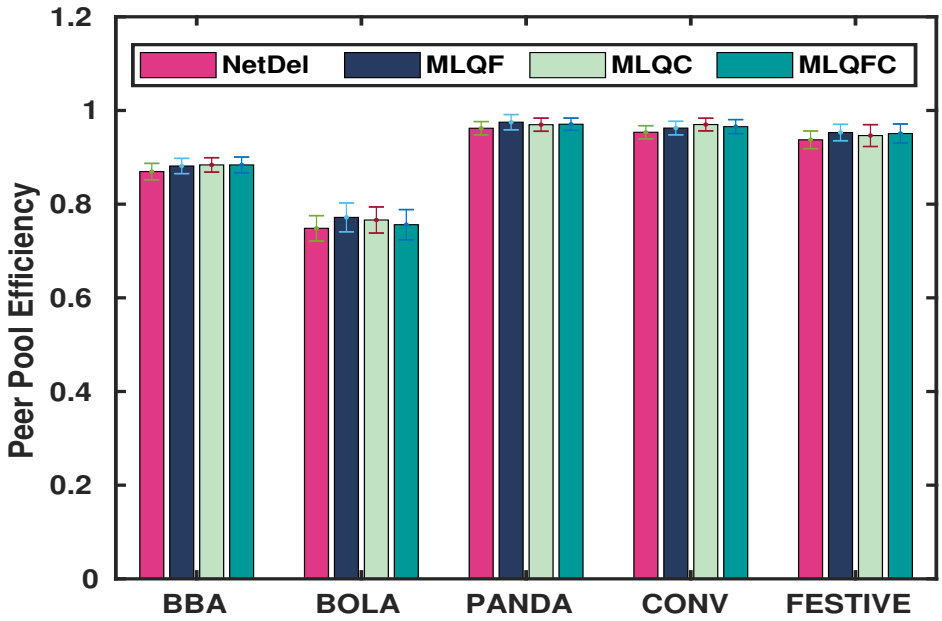


Figure 6.11: P2P metrics: peer pool efficiency

ABR would request for the next segment upon receiving the current one. With this prediction, we introduced MLQF and MLQC, two different proposals for enhancing the prefetching efficiency. MLQF uses the predicted quality at the P2P scheduler level to update the prefetching quality and stop fetching the current one. MLQC uses this quality prediction to catch the quality down switches and prevent them when a higher quality exists in the P2P cache. We merged both MLQF and MLQC in one logic called MLQFC, which uses MLQF to fetch the predicted quality as long as this quality is higher or equal to what is already fetched. Otherwise, MLQFC uses MLQC to prevent the quality down-switch. Results show that MLQC achieves a better performance in terms of the average bitrate. Both MLQF and MLQC manage to improve the P2P usage efficiency by increasing the P2P offload and reducing the P2P overhead while keeping the same QoE. MLQFC combines the merits of both proposals, and improves the average quality by 5-7% and the stability by nearly 6% while gaining better P2P offload by 6-10% and less P2P overhead by 5-8 % compared to NetDel.

Chapter 7

Conclusions

7.1 Summary

In this thesis, we focused on HTTP adaptive streaming in hybrid CDN / prefetching-based P2P networks. Our main concern was to stay compatible with the layered implementations of HAS and P2P stack, where these two are unknown to each other. We stated in Chapter 1, that our main objectives are to enable the existing HAS algorithms in prefetching-based P2P networks without any modification to the video player's ABR algorithm while improving QoE and P2P efficiency in the prefetching environments. We achieved these objectives as discussed below.

In Chapter 2, we provided a review of the state-of-the-art in ABR design, P2P networks, and QoE and P2P evaluations. ABR algorithms are classified, according to their location in the system, into client-side, server-side and network-assisted algorithms. Client-side adaptation algorithms are the most popular, and in turn, they are classified based on their input metrics into buffer-based, throughput-based, and hybrid class. P2P networks are a scalable and cost-efficient delivery solution; users are distributed into two main architectures: mesh-based and tree-based. P2P networks work with CDNs to ensure the best delivery of the video content in terms

of QoE and scalability.

In Chapter 3, we discussed the challenges related to testing the hybrid CDN / P2P environments for video streaming, for research purposes in particular. We introduced both realistic and simulated platforms to conduct video streaming experiments. The real-time platform performs a good traffic shaping for the participating peers; however, it can not precisely reproduce the same results over different experiments. The simulation model enables reproducible and flexible adaptive video streaming scenarios using different ABR algorithms. In addition, it allows both CDN only and hybrid CDN/P2P networks. We used the simulation model in the other chapters to evaluate our contributions and compare them to the existing works.

In Chapter 4, we discussed the main problems facing the usage of existing ABR algorithms in hybrid CDN/P2P networks, especially the existence of the local cache at the client-side. To treat these problems, we designed Response-Delay, a novel method that modifies the segment responses to make HAS algorithms compatible with P2P; without changing the ABR logic which is our main concern. Response-Delay rectifies the fast delivery issue of P2P segments and shows good performance in terms of QoE and P2P efficiency, which is one of our main objectives. However, it does not take the exact reactions of the ABR logic or the status of the P2P cache into consideration, which we investigated in Chapters 5 and 6.

In Chapter 5, we investigated the possibility of learning the behavior of the ABR logic using machine learning techniques; supervised learning in particular. We tested different ML models to learn the behavior of different ABR algorithms from state-of-the-art and real-world closed-source deployments. Random Forest and Gradient boost algorithms achieved a very high prediction accuracy, using only the basic information provided as input to the application layer. This work is compatible with the layered integration of P2P and ABR, as it is agnostic of the ABR algorithm.

In Chapter 6, we discussed the issues of prefetching-based P2P networks with

HAS algorithms. We combined the benefits of Chapters 4 and 5 and designed an ABR-aware prefetching technique that improves the P2P efficiency in the prefetching context. We managed to use the spare delay time to predict the P2P-related quality switches and fetch the new qualities before the player requests. Besides, we introduced a novel method to control the player ABR externally to select a better quality from the P2P cache. This method managed to improve the P2P efficiency by leading the ABR to make P2P-friendly decisions, which is our last main objective in this thesis.

The work presented in Chapter 4 is part of the following patent:

- H. Yousef, P. Ageneau, A. Delmas. "Method for broadcasting streaming contents in a peer-to-peer network". US Patent App. 16/832,088. Nov, 2020.

In addition, we filed a family of patents related to Chapters 5 and 6 and other related topics, which we enlist below:

- H. Yousef, A. Storelli. "Method for playing on a player of a client device a content streamed in a network". Filing date April 20, 2021. Filing number 21305519.7.

- H. Yousef, A. Storelli. "Method for playing on a player of a client device a content streamed in a network". Filing date April 20, 2021. Filing number 21305520.5.

- H. Yousef, A. Storelli, A. Delmas. "Method for playing on a player of a client device a content streamed in a network". Filing date December 11, 2020. Filing number 20306545.3.

- H. Yousef, A. Storelli, A. Delmas. "Method for playing on a player of a client device a content streamed in a network". Filing date March 26, 2020. Filing number 20315054.5.

- H. Yousef, A. Storelli. "Method for playing on a player of a client device a content streamed in a network". Filing date February 28, 2020. Filing number 20305202.2.

7.2 Future research perspectives

7.2.1 ABR controlling in a single client-server architecture

The idea of controlling the ABR can be extended to other scenarios than P2P only. For example, in a single client-server scenario, the service provider may want to drive the HAS algorithm towards a specific decision without any further modifications on the player side. In this case, our proposed method can be applied at the server-side, at a network-assisted entity (ex. proxy), or even the client-side to modify the segment responses before returning them to the video player. This use case is different from P2P use case in that all the segments can be controlled; in P2P we had control on the P2P cached segments only.

In fact, this idea has attracted our attention, so we started to investigate whether it is possible to drive the ABR to a specific behavior when we have a control on all the segments, as the service provider may like to do.

Here, we show the initial investigations on buffer-based and throughput-based algorithms. We run a simple simulation where the client has a good link of 10 Mbps. The ABR controller resides at the client-side and tries to drive the ABR towards selecting the desired bitrate. In the following discussion, we refer to the bitrate we want the ABR to select as the *desired bitrate*, and the actual ABR decision as *selected bitrate*. The converged bitrate is either equal to the desired bitrate when the ABR controller manages to find an optimal response delay that leads the ABR to pick the desired bitrate; otherwise, the converged bitrate is zero.

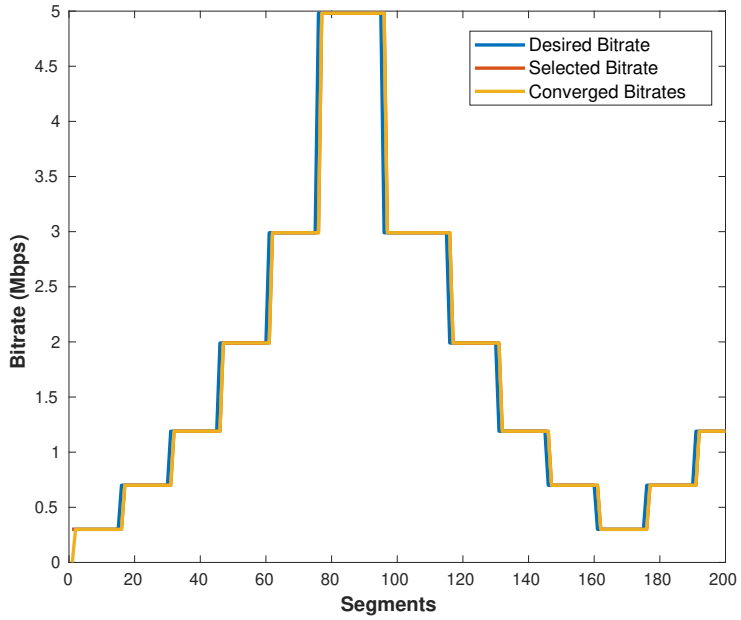
Figure 7.1a, shows promising results for BOLA, where the controller manages

to find the delay value that leads the ABR to select the desired bitrate. However, Figure 7.1b shows different behavior for CONVENTIONAL. In particular, at the quality transitions, the ABR controller fails to force the ABR to select another bitrate. In this example, if the converged bitrate is zero for some time, CONVENTIONAL estimates a bandwidth around 10 Mbps and selects the highest quality for the next segments. We suspect that sometimes it is hard to make the ABR switch the quality by modifying the response of one segment only. For example, if the ABR follows a buffer-based logic, adding one segment may not be enough to increase the buffer to a value where the ABR decides to switch. On the other hand, if the ABR follows a through-put approach, it may wait to measure the bandwidth for few segments before switching the quality. This therefor needs further research to solve this issue, most probably by adding a metric to learn the ABR smoothness nature and control a horizon of segments instead of per-segment control.

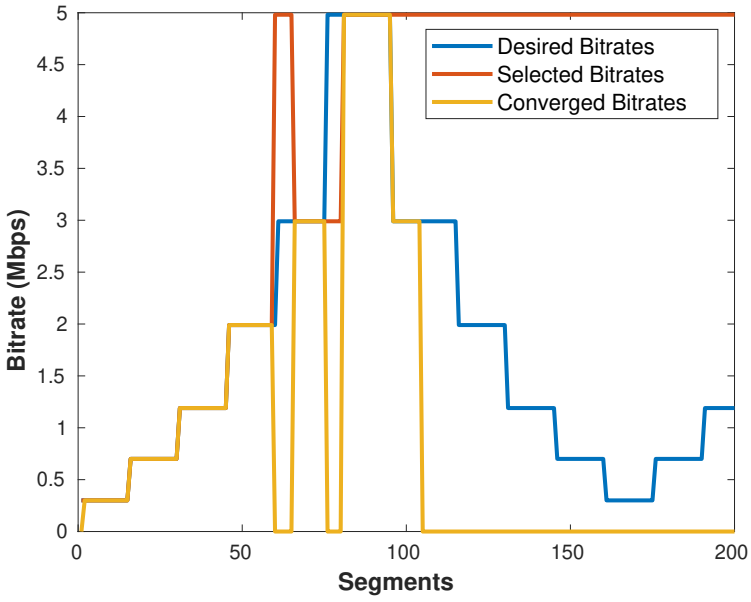
7.2.2 ABR controlling using feedback control theory

Feedback control theory has been used in many works for bitrate adaptation, either at the server side [44] or at the client-side [42]. In this design, the ABR changes the bitrate selection based on network and client feedback assistance. Another direction would be to use feedback control theory to control the already implemented ABR algorithm by modifying the ABR input feedback externally. This direction would extend our idea of modifying the segment response but using a feedback controller and/or ML-based controller. In the previous Section 7.2.1, we found that our ML-based ABR controller may fail to control the ABR logic to switch to another quality at some cases.

To investigate this issue, we explored using a simple feedback controller in the quality transition phases, where we insist the player to select the desired quality. This feedback controller would accelerate the delivery when choosing a higher



(a) BOLA



(b) CONV

Figure 7.1: Bitrate selection of (a) BOLA and (b) CONV algorithm using the ABR ML-based ABR controller

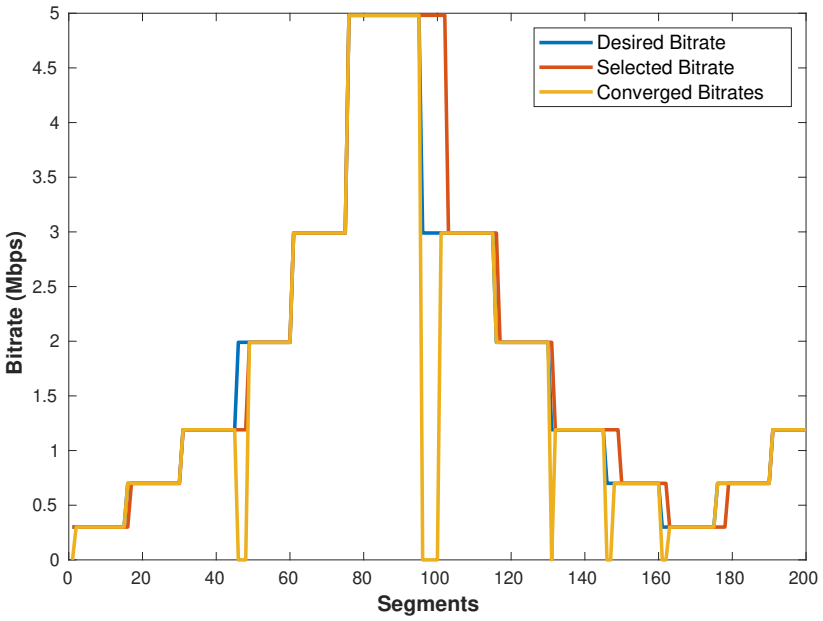


Figure 7.2: Bitrate selection of CONV algorithm using hybrid ML/feedback controller

quality while slowing it down when choosing lower quality. The initial results of this investigation are promising, as shown in Figure 7.2. Here, we notice that the feedback controller manages to take over the ML-based controller at the quality transition phases. For example, for segments 96-100 the ML-based model does not converge and fails to find a delay that switches down the quality to the desired ones. Here, the controller tries to slow down the delivery for these segments until the ML-based controller resumes its work for segment 101 and so on. Further research can continue this work towards merging the ML-based and feedback-based controllers to have full control on the player ABR algorithm.

7.2.3 Lightweight ML model for ABR algorithms

HAS solutions are evolving, and many complex ABR algorithms have emerged to achieve better performance, such as Model Predictive Control (MPC) [99] and the

ABRs that rely on neural networks [39] [110]. However, these complex algorithms require expensive computation processes, which make them too heavy to deploy at the end-user devices such as mobile phones, and they may introduce undesired decision latency. Some algorithms like FastMPC[99] and ABMA+ [38] pre-compute solutions, store them results into a table, and look up the table when running online. However, the performance is not as good as the optimal solution; FastMPC drops the performance of up to 30%. Other ABR algorithms dedicate remote ABR servers to offloading computation. However, by introducing new servers, these solutions increase the expenses for the content providers are not scalable to large-scale deployment. In this context, one potential solution could be to learn these heavy ABR algorithms, then replacing them with their corresponding lightweight ML models (such as the one proposed in Chapter 5).

7.2.4 P2P-friendly ABR

As stated in Chapter 1, our work focuses on using P2P with pre-implemented ABR of the video player that does not know about the P2P existence, which makes the P2P integration with these players more challenging. Some works such as [92] [66] combine chunk, bitrate, and peer selection policies together in one client-based ABR to minimize the occurrence of rebuffering, delivering high-quality video, and improving the efficiency of the P2P networks. However, these works can not be implemented in the layered implementation of the ABR and P2P stack. Further research can be conducted on using these algorithms or even designing a new P2P-friendly ABR algorithm optimized in the isolated layered ABR and P2P stack. One possibility is to implement the P2P-friendly ABR outside the player, then using the proposals of Chapter 6 to control the player's ABR externally, so it behaves like the new P2P-friendly ABR algorithm.

7.3 Conclusion

This thesis is a CIFRE thesis, and the work was carried out between the Information Processing and Communications Laboratory (LTCI) in Télécom Paris and Research and Development department in STREAMROOT (now LUMEN) company. We focused our work on HTTP adaptive streaming in hybrid CDN / prefetching-based P2P networks using layered HAS and P2P stack implementation, which is also the design used by STREAMROOT technology. We introduced methodologies to enable the usage of existing HAS algorithms in the P2P networks and to design an ABR-aware prefetching technique. All our achievements have the advantage of being compatible with the layered HAS and P2P stack implementation, and part of the findings are already implemented in STREAMROOT MESH Technology.

Bibliography

[1] Cisco, “Cisco visual networking index: Forecast and methodology, 2017 - 2022,” *White Paper*, February 2019.

[2] Sandvine, “Global internet phenomena,” *Report*, June 2019.

[3] Nielsen, “The nielsen total audience report,” April 2020.

[4] Cisco, “Cisco annual internet report (2018–2023),” *White Paper*, March 2020.

[5] Z. Shen, J. Luo, R. Zimmermann, and A. V. Vasilakos, “Peer-to-peer media streaming: Insights and new developments,” *Proceedings of the IEEE*, vol. 99, no. 12, pp. 2089–2109, 2011.

[6] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang, “Challenges, design and analysis of a large-scale p2p-vod system,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, p. 375–388, Aug. 2008. [Online]. Available: https://doi.org/10.1145/1402946.1403001

[7] N. Ramzan, H. Park, and E. Izquierdo, “Video streaming over p2p networks: Challenges and opportunities,” *Signal Processing: Image Communication*, vol. 27, no. 5, pp. 401–411, 2012, aDVANCES IN 2D/3D VIDEO STREAMING OVER P2P NETWORKS. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0923596512000331

[8] Microsoft smooth streaming. [Online]. Available: http://www.iis.net/downloads/microsoft/smooth-streaming

[9] Apple http live streaming. [Online]. Available: https://developer.apple.com/resources/http-streaming

[10] (2012) Open source media framework (osmf). CA, USA. [Online]. Available: https://sourceforge.net/adobe/osmf/home/Home/

[11] T. Stockhammer, "Dynamic adaptive streaming over http –: Standards and design principles," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 133–144. [Online]. Available: https://doi.org/10.1145/1943552.1943572

[12] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, no. 4, p. 374–398, Dec. 2003. [Online]. Available: https://doi.org/10.1145/954339.954341

[13] K. Liang, J. Hao, R. Zimmermann, and D. K. Y. Yau, "Integrated prefetching and caching for adaptive video streaming over http: An online approach," in *Proceedings of the 6th ACM Multimedia Systems Conference*, ser. MMSys '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 142–152. [Online]. Available: https://doi.org/10.1145/2713168.2713181

[14] R. Roverso, S. El-Ansary, and S. Haridi, "Smoothcache: Http-live streaming goes peer-to-peer," in *NETWORKING 2012*, R. Bestak, L. Kencl, L. E. Li, J. Widmer, and H. Yin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 29–43.

[15] U. Abbasi and T. Ahmed, "Cooching: Cooperative prefetching strategy for p2p video-on-demand system," in *Proceedings of the 12th IFIP/IEEE International*

Conference on Management of Multimedia and Mobile Networks and Services: Wired-Wireless Multimedia Networks and Services Management, ser. MMNS 2009. Berlin, Heidelberg: Springer-Verlag, 2009, p. 195–200. [Online]. Available: https://doi.org/10.1007/978-3-642-04994-1_18

[16] E. Kim, T. Kim, and C. Lee, "An adaptive buffering scheme for p2p live and time-shifted streaming," *Applied Sciences*, vol. 7, no. 2, 2017. [Online]. Available: https://www.mdpi.com/2076-3417/7/2/204

[17] J. Kua, G. Armitage, and P. Branch, "A survey of rate adaptation techniques for dynamic adaptive streaming over http," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1842–1866, 2017.

[18] Y. Sani, A. Mauthe, and C. Edwards, "Adaptive bitrate selection: A survey," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2985–3014, 2017.

[19] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann, "A survey on bitrate adaptation schemes for streaming media over http," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 562–585, 2019.

[20] G. Gheorghe, R. Lo Cigno, and A. Montresor, "Security and privacy issues in p2p streaming systems: A survey," *Peer-to-Peer Netw. Appl*, pp. 75 –9 1, 2011.

[21] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys (CSUR)*, vol. 36, pp. 335–371, December 2004.

[22] E. Thomas, M. O. van Deventer, T. Stockhammer, A. C. Begen, M. Champel, and O. Oyman, "Application of sand technology in dash-enabled content de-

livery networks and server environments," *SMPTE Motion Imaging Journal*, vol. 127, no. 1, pp. 48–54, 2018.

[23] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive http streaming," ser. MMSys '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 169–174. [Online]. Available: https://doi.org/10.1145/1943552.1943575

[24] C. Liu, I. Bouazizi, M. M. Hannuksela, and M. Gabbouj, "Rate adaptation for dynamic adaptive streaming over http in content distribution network," *Image Commun.*, vol. 27, no. 4, p. 288–311, Apr. 2012. [Online]. Available: https://doi.org/10.1016/j.image.2011.10.001

[25] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, "Probe and adapt: Rate adaptation for http video streaming at scale," *IEEE Journal on Selected Areas in Communications*, vol. 32, April 2014.

[26] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, p. 314–329, Aug. 1988. [Online]. Available: https://doi.org/10.1145/52325.52356

[27] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency and stability in http-based adaptive video streaming with festive," in *IEEE/ACM Transactions on Networking (TON)*, pp. 326–340.

[28] M. Xiao, V. Swaminathan, S. Wei, and S. Chen, "Dash2m: Exploring http/2 for internet streaming to mobile devices," in *Proceedings of the 24th ACM International Conference on Multimedia*, ser. MM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 22–31. [Online]. Available: https://doi.org/10.1145/2964284.2964313

[29] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: evidence from a large video streaming service," in *Proceedings of the 2014 ACM conference on SIGCOMM*, Chicago, Illinois, USA, August 2014.

[30] R. M. Abuteir, A. Fladenmuller, O. Fourmaux, and M. Ammar, "Variable-threshold buffer based adaptation for dash mobile video streaming," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2017, pp. 1–7.

[31] K. Spiteri1, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," *IEEE INFOCOM*, April 2016.

[32] M. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*, 01 2010, vol. 3, no. 1.

[33] P. K. Yadav, A. Shafiei, and W. T. Ooi, "Quetra: A queuing theory approach to dash rate adaptation," in *Proceedings of the 25th ACM International Conference on Multimedia*, ser. MM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1130–1138. [Online]. Available: https://doi.org/10.1145/3123266.3123390

[34] A. Bentaleb, P. K. Yadav, W. T. Ooi, and R. Zimmermann, "Dq-dash: A queuing theory approach to distributed adaptive video streaming," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 16, no. 1, Mar. 2020. [Online]. Available: https://doi.org/10.1145/3371040

[35] V. Burger, T. Zinner, L. Dinh-Xuan, F. Wamser, and P. Tran-Gia, "A generic approach to video buffer modeling using discrete-time analysis," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 14, no. 2s, Apr. 2018. [Online]. Available: https://doi.org/10.1145/3183511

[36] P. Juluri, V. Tamarapalli, and D. Medhi, "Sara: Segment aware rate adaptation algorithm for dynamic adaptive streaming over http," in *2015 IEEE International Conference on Communication Workshop (ICCW)*, 2015, pp. 1765–1770.

[37] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz, "Adaptation algorithm for adaptive streaming over http," in *2012 19th International Packet Video Workshop (PV)*, 2012, pp. 173–178.

[38] A.Beben1, P.Wiśniewski, J. M. Batalla, and P.Krawiec, "Abma+ : lightweight and efficient algorithm for http adaptive streaming," in *Proceedings Int. ACM Conference on Multimedia Systems (MMSys).*

[39] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 197–210. [Online]. Available: https://doi.org/10.1145/3098822.3098843

[40] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella, "D-dash: A deep q-learning framework for dash video streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 703–718, 2017.

[41] N. L. A. F. D. T. M. C. Theodoros Karagkioules, Georgios S. Paschos, "Boptimizing adaptive video streaming in mobile networks via online learning," November 2019.

[42] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo, "Elastic: A client-side controller for dynamic adaptive streaming over http (dash)," in *2013 20th International Packet Video Workshop*, 2013, pp. 1–8.

[43] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen, "Server-based traffic shaping for stabilizing oscillating adaptive streaming players," in *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 19–24. [Online]. Available: https://doi.org/10.1145/2460782.2460786

[44] L. De Cicco, S. Mascolo, and V. Palmisano, "Feedback control for adaptive live video streaming," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 145–156. [Online]. Available: https://doi.org/10.1145/1943552.1943573

[45] J. Zou, C. Li, C. Liu, Q. Yang, H. Xiong, and E. Steinbach, "Probabilistic tile visibility-based server-side rate adaptation for adaptive 360-degree video streaming," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 1, pp. 161–176, 2020.

[46] C. Liu, N. Kan, J. Zou, Q. Yang, and H. Xiong, "Server-side rate adaptation for multi-user 360-degree video streaming," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, 2018, pp. 3264–3268.

[47] R. Houdaille and S. Gouache, "Shaping http adaptive streams for a better user experience," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1–9. [Online]. Available: https://doi.org/10.1145/2155555.2155557

[48] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang, "Qdash: A qoe-aware dash system," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12. New York, NY, USA:

Association for Computing Machinery, 2012, p. 11–22. [Online]. Available: https://doi.org/10.1145/2155555.2155558

[49] N. Bouten, R. de O. Schmidt, J. Famaey, S. Latré, A. Pras, and F. De Turck, "Qoe-driven in-network optimization for adaptive video streaming based on packet sampling measurements," *Computer Networks*, vol. 81, pp. 96 – 115, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615000468

[50] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 272–285. [Online]. Available: https://doi.org/10.1145/2934872.2934898

[51] R. Jmal, G. Simon, and L. Chaari, "Network-assisted strategy for dash over ccn," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, 2017, pp. 13–18.

[52] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 189–202. [Online]. Available: https://doi.org/10.1145/1177080.1177105

[53] Y. hua Chu, S. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, pp. 1456 – 1471, October 2002.

[54] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth content distribution in cooperative environments," in *Peer-to-Peer Systems II*, M. F. Kaashoek and I. Stoica, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 292–303.

[55] X. Zhang, J. Liu, B. Li, and Y.-S. Yum, "Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming," *IEEE INFOCOM*, vol. 3, pp. 2102–2111, March 2005.

[56] N. Magharei and R. Rejaie, "Understanding mesh-based peer-to-peer streaming," Newport, Rhode Island, November 2006.

[57] Y. Chu, S. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay muilticast architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, p. 55–67, Aug. 2001. [Online]. Available: https://doi.org/10.1145/964723.383064

[58] Z. Lu, Y. Li, J. Wu, S. Zhang, and Y. Zhong, "Multipeercast: A tree-mesh-hybrid p2p live streaming scheme design and implementation based on peercast," *2008 10th IEEE International Conference on High Performance Computing and Communications*, September 2008.

[59] M. Moshref, R. Motamedi, H. R. Rabiee, and M. Khansari, "Layeredcast - a hybrid peer-to-peer live layered video streaming protocol," *2008 10th IEEE International Conference on High Performance Computing and Communications*, September 2008.

[60] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree: A comparative study of live p2p streaming approaches," in *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, 2007, pp. 1424–1432.

[61] Q. Huang, H. Jin, and X. Liao, “P2p live streaming with tree-mesh based hybrid overlay,” *2007 International Conference on Parallel Processing Workshops (ICPPW 2007)*, September 2007.

[62] H. Y. Tsinghua, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li, “Design and deployment of a hybrid cdn-p2p system for live video streaming: experiences with livesky,” Beijing, China, October 2009, pp. 25–34.

[63] D. XuEmail, authorSunil Suresh Kulkarni, C. Rosenberg, and H.-K. Chai, “Analysis of a cdn-p2p hybrid architecture for cost effective streaming media distribution,” *Multimedia Systems*, vol. 11, pp. 383–399, 2006.

[64] T. T. T. Ha, J. Kim, and J. Nam, “Design and deployment of low-delay hybrid cdn–p2p architecture for live video streaming over the web,” *Wireless Personal Communications*, vol. 94, no. 3, pp. 513–525, Jun 2017.

[65] A. Mansy and M. Ammar, “Analysis of adaptive streaming for hybrid cdn/p2p live video systems,” in *2011 19th IEEE International Conference on Network Protocols*, 2011, pp. 276–285.

[66] M. L. Merani and L. Natali, “Adaptive streaming in p2p live video systems: A distributed rate control approach,” *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 12, 2016.

[67] D. Jurca, J. Chakareski, J. Wagner, and P. Frossard, “Enabling adaptive video streaming in p2p systems [peer-to-peer multimedia streaming],” *IEEE Communications Magazine*, vol. 45, no. 6, pp. 108–114, 2007.

[68] S. Medjiah, T. Ahmed, and R. Boutaba, “Avoiding quality bottlenecks in p2p adaptive streaming,” *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 734–745, 2014.

[69] Y.-H. Moon, J.-N. Kim, and C.-H. Youn, "Churn-aware optimal layer scheduling scheme for scalable video distribution in super-peer overlay networks," *J. Supercomput.*, vol. 66, no. 2, p. 700–720, Nov. 2013. [Online]. Available: https://doi.org/10.1007/s11227-012-0858-7

[70] H. Yousef, J. Le Feuvre, P.-L. Ageneau, and A. Storelli, "Enabling adaptive bitrate algorithms in hybrid cdn/p2p networks," in *Proceedings of the 11th ACM Multimedia Systems Conference*, ser. MMSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 54–65. [Online]. Available: https://doi.org/10.1145/3339825.3391859

[71] T. Hoßfeld, P. E. Heegaard, L. Skorin-Kapov, and M. Varela, "Fundamental relationships for deriving qoe in systems," in *2019 Eleventh International Conference on Quality of Multimedia Experience (QoMEX)*, 2019, pp. 1–6.

[72] O. Oyman and S. Singh, "Quality of experience for http adaptive streaming services," *IEEE Communications Magazine*, vol. 50, pp. 20–27, April 2012.

[73] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A survey on quality of experience of http adaptive streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, p. 469–492, 2015.

[74] S. Varma, "Flow control for video applications," in *Internet Congestion Control*, S. Varma, Ed. Boston: Morgan Kaufmann, 2015, pp. 173–203. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128035832000062

[75] J. Rückert, O. Abboud, T. Zinner, R. Steinmetz, and D. Hausheer, "Quality adaptation in p2p video streaming based on objective qoe metrics," in *NETWORKING 2012*, R. Bestak, L. Kencl, L. E. Li, J. Widmer, and H. Yin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–14.

[76] X. Zhang and H. Hassanein, "A survey of peer-to-peer live video streaming schemes – an algorithmic perspective," *Computer Networks*, vol. 56, no. 15, pp. 3548–3579, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128612002393

[77] M. L. Merani and L. Natali, "Adaptive streaming in p2p live video systems: A distributed rate control approach," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 12, no. 3, pp. 1–23, June 2016.

[78] S. M. Y. Seyyedi and B. Akbari, "Hybrid cdn-p2p architectures for live video streaming: Comparative study of connected and unconnected meshes," in *2011 International Symposium on Computer Networks and Distributed Systems (CNDS)*, 2011, pp. 175–180.

[79] T. Sanguankotchakorn and N. Krueakampliw, "A hybrid pull-push protocol in hybrid cdn-p2p mesh-based architecture for live video streaming," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2017, pp. 187–192.

[80] J. Bruneau-Queyreix, M. Lacaud, D. Négru, J. M. Batalla, and E. Borcoci, "Adding a new dimension to http adaptive streaming through multiple-source capabilities," *IEEE MultiMedia*, vol. 25, pp. 65–78, 2018.

[81] H. Heo, W. Lee, H. Kim, B.-D. Lee, B. Jeong, and N. Kim, "Analysis of 3gpp lte handover using ns-3 simulator," *Annual Int'l Conference on Intelligent Computing*, pp. 24–25, 2016.

[82] R. Gupta, B. Bachmann, R. Ford, S. Rangan, N. Kundargi, A. Ekbal, K. Rathi, M. Sanchez, A. de la Oliva, and A. Morelli, "ns-3-based real-time emulation

of lte testbed using labview platform for software defined networking (sdn) in crowd project," 05 2015, pp. 91–97.

[83] A. Fouda, A. N. R. A. Esswie, and A. S. Ibrahim, "Real-time video streaming over ns3-based emulated lte networks," *International Journal of Electronics Communication and Computer Technology*, vol. 4, pp. 659–2014, 2014.

[84] W. D. Diego Maza, "A Framework for Generating HTTP Adaptive Streaming Traffic in ns-3," in *SIMUTools - 9th EAI International Conference on Simulation Tools and Techniques - 2016*. Prague, Czech Republic: ACM SIGSIM, Aug. 2016, accepted short paper. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01362445

[85] W. Y. Campo-MuÃ, E. Astaiza-Hoyos, and L. F. MuÃ-Sanabria, "Traffic modelling of the video-on-demand service through NS-3," *DYNA*, vol. 84, pp. 55 – 64, 09 2017. [Online]. Available: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0012-73532017000300055&nrm=iso

[86] H. Ott, K. Miller, and A. Wolisz, "Simulation framework for http-based adaptive streaming applications," in *Proceedings of the Workshop on Ns-3*, ser. WNS3 '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 95–102. [Online]. Available: https://doi.org/10.1145/3067665.3067675

[87] T. Lyko, M. Broadbent, N. Race, M. Nilsson, P. Farrow, and S. Appleby, "Evaluation of cmaf in live streaming scenarios," in *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 21–26. [Online]. Available: https://doi.org/10.1145/3386290.3396932

[88] K. Spiteri, R. Sitaraman, and D. Sparacio, "From theory to practice: Improving bitrate adaptation in the dash reference player," in *Proceedings of the 9th ACM Multimedia Systems Conference*, ser. MMSys '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 123–137. [Online]. Available: https://doi.org/10.1145/3204949.3204953

[89] N. Varis, "Anatomy of a linux bridge," 2012.

[90] K. Spiteri, R. Sitaraman, and D. Sparacio, "From theory to practice: Improving bitrate adaptation in the dash reference player," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 15, no. 2s, Jul. 2019. [Online]. Available: https://doi.org/10.1145/3336497

[91] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3g networks: analysis and applications," in *ACM MMsys*, 2013.

[92] K.-W. Hwang, V. Gopalakrishnan, R. Jana, S. Lee, V. Misra, K. K. Ramakrishnan, and D. S. Rubenstein, "Joint-family: Adaptive bitrate video-on-demand streaming over peer-to-peer networks with realistic abandonment patterns," *Computer Networks: The International Journal of Computer and Telecommunications Networking archive*, vol. 106, pp. 226–244, 2016.

[93] D. S. Berger, "Towards lightweight and robust machine learning for cdn caching," in *Proc of the 17th ACM Workshop on Hot Topics in Networks*, ser. HotNets '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 134–140.

[94] M. Claeys, S. Latré, J. Famaey, and F. De Turck, "Design and evaluation of a self-learning http adaptive video streaming client," *IEEE Communications Letters*, vol. 18, no. 4, pp. 716–719, 2014.

[95] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[96] D. Liu, Y. Li, J. Lin, H. Li, and F. Wu, "Deep learning-based video coding: A review and a case study," *ACM Comput. Surv.*, vol. 53, no. 1, 2020.

[97] Y. Chien, K. C. Lin, and M. Chen, "Machine learning based rate adaptation with elastic feature selection for http-based streaming," in *IEEE International Conference on Multimedia and Expo (ICME)*, 2015, pp. 1–6.

[98] A. Lekharu, S. Kumar, A. Sur, and A. Sarkar, "A qoe aware lstm based bitrate prediction model for dash video," in *2018 10th International Conference on Communication Systems Networks (COMSNETS)*, 2018, pp. 392–395.

[99] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 325–338, Aug. 2015. [Online]. Available: https://%doi.org/10.1145/2829988.2787486

[100] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, "Http/2-based adaptive streaming of hevc video over 4g/lte networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.

[101] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, p. 273–297, Sep. 1995. [Online]. Available: https://%doi.org/10.1023/A:1022627411411

[102] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, October 2001.

[103] S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *Data Mining and Knowledge Discovery*, vol. 2, pp. 345–389, 1997.

[104] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119 – 139, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002200009791504X

[105] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001. [Online]. Available: http://www.jstor.org/stable/2699986

[106] B. Cestnik, "Estimating probabilities: A crucial task in machine learning." 01 1990, pp. 147–149.

[107] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, 1967.

[108] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, p. 2825–2830, Nov. 2011.

[109] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: The MIT Press, 2012.

[110] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han, "Neural adaptive content-aware internet video delivery," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, 2018, p. 645–661.

Titre : Streaming adaptatif sur réseaux pair-à-pair et HTTP

Mots clés : Streaming adaptatif, réseaux pair-à-pair, HTTP, Qualité d'Expérience

Résumé : La croissance du trafic vidéo liée à l'offre et au nombre d'utilisateurs, ainsi que les progrès des technologies vidéo et des appareils, ont augmenté les attentes des utilisateurs en termes de Qualité d'Expérience (QoE). Aujourd'hui, le trafic vidéo représente 79% du trafic Internet mondial, et ce pourcentage devrait atteindre 82% d'ici 2022 avec les services Over The Top (OTT) qui représentent plus de 50% du trafic de pointe dans le monde. Les solutions HTTP Adaptive Streaming (HAS) se sont révélées être l'une des techniques essentielles pour faire face à ce trafic vidéo en constante augmentation, grâce à leur logique d'adaptation en débit (ABR) intégrée côté client, qui permet de s'adapter aux conditions d'utilisation (oscillations de bande passante, ressources matérielles...) afin de maximiser la QoE de l'utilisateur. En parallèle, la distribution vidéo sur les réseaux Pair-à-Pair (P2P) et sur les réseaux de diffusion de contenu (CDN) devient cruciale pour permettre au réseau de faire face à l'explosion du nombre de consommateurs vidéo. Suite aux récentes améliorations des technologies P2P et HAS, de nombreux efforts ont été déployés pour rapprocher ces deux techniques. Cependant, le déploiement HAS sur les réseaux P2P pose de nombreux défis. Le réseau P2P est problématique pour les techniques HAS en raison de l'hétérogénéité des ressources et de la fréquence des arrivées/départs des clients. Une grande partie des implémentations repose sur un modèle en couches où les piles HAS et P2P sont isolées l'une de l'autre; dans ce modèle, les techniques de pré-chargement P2P sont indépendantes de la logique ABR utilisée, ce qui conduit à une utilisation inefficace des ressources réseau lors des changements de qualité. Cette thèse se concentre sur les implémentations de piles HAS et P2P en couches et vise à analyser les problèmes mentionnés ci-dessus et à proposer des méthodes pour les résoudre, tout en améliorant l'efficacité de la distribution P2P. Pour y parvenir, nous construisons un environnement de simulation pour tester les solutions HAS dans les systèmes hybrides CDN/P2P et analyser les problèmes associés. Nous proposons ≪Response-Delay≫, une méthode permettant l'utilisation d'algorithmes HAS existants dans le contexte de réseaux P2P basés sur le pré-chargement; cette méthode module le délai de réponse des requêtes en amont du lecteur HAS et ne nécessite aucune modification de l'algorithme ABR implémenté. Nous proposons par ailleurs des modèles d'apprentissage pour prédire les décisions de qualité des algorithmes HAS, en utilisant un ensemble de métriques d'entrée que l'ABR utilise pour prendre une décision de débit. Enfin, nous combinons ≪Response-Delay≫ et les modèles d'apprentissage ABR pour définir une méthode de précharegment et de contrôle de QoE plus efficace. Cette technique utilise la décision ABR prédite dans le processus de pré-chargement et contrôle l'ABR en amont pour prendre des décisions favorables au P2P.

ECOLE DOCTORALE

Title : Adaptive streaming using Peer-to-Peer and HTTP

Keywords : Adaptive streaming, P2P networks, HTTP, Quality of Experience

Abstract : The increasing growth of video traffic and the number of Internet users, besides the progressing video technologies and device capabilities, have surged the demand for improving the user Quality of Experience (QoE). Today, video traffic accounts for 79% of the global Internet traffic, and this percentage is projected to strike 82% by 2022, with Over The Top (OTT) services accounting for more than 50% of the peak download traffic globally.

HTTP Adaptive Streaming (HAS) solutions have shown to be one of the essential techniques to cope with this ever-increasing video traffic, thanks to their embedded Adaptive BitRate (ABR) logic at the client-side which allows adaptation to the bandwidth oscillations and maximizing QoE.

In parallel, video distribution over Peer-to-Peer (P2P) networks, along with Content Delivery Networks (CDN), is becoming more important to handle the explosion in the number of video consumers.

As a result of P2P and HAS recent improvements, there have been many efforts to bring these two approaches together. However, the deployment of HAS streaming over P2P networks raises many challenges. The P2P nature is problematic due to the heterogeneity of resources and the dynamicity of peers. The layered implementations where HAS and P2P stacks are isolated from each other. The P2P prefetching techniques are not aware of the used ABR logic, which leads to inefficient usage of the network resources.

This thesis focuses on the layered HAS and P2P stack implementations and aims to analyze the above-mentioned issues and propose methods to solve them, enhancing QoE and P2P efficiency. To achieve this, we build a simulation environment to test HAS solutions in hybrid CDN/P2P systems and analyze the related issues. We propose Response-Delay, a method enabling usage of existing HAS algorithms in the context of prefetching-based P2P networks; Response-Delay is external to the video player and does not require any modification to the implemented ABR algorithm. Besides, we propose ML-based models to predict the quality decisions of HAS algorithms, using only a set of input metrics that the ABR can use to make a bitrate decision. Finally, we combine Response-Delay and the ML-based ABR models towards an ABR-aware prefetching and quality control technique. This technique uses the predicted ABR decision in the prefetching process and controls the ABR externally to make P2P-friendly decisions.

Institut Polytechnique de Paris
91120 Palaiseau, France