Low latency video streaming solutions based on HTTP/2

Mariem Ben Yahia

▶ **To cite this version:**

Mariem Ben Yahia. Low latency video streaming solutions based on HTTP/2. Graphics [cs.GR]. Ecole nationale supérieure Mines-Télécom Atlantique, 2019. English. NNT : 2019IMTA0136 . tel-02179028

HAL Id: tel-02179028
https://theses.hal.science/tel-02179028

Submitted on 10 Jul 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THESE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPERIEURE MINES-TELECOM ATLANTIQUE
BRETAGNE PAYS DE LA LOIRE - IMT ATLANTIQUE
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

Mariem BEN YAHIA

Low Latency Video Streaming Solutions Based on HTTP/2

Solutions de transmission vidéo avec faible latence basées sur HTTP/2

Thèse présentée et soutenue à Rennes, le 10/05/2019
Unité de recherche : IRISA
Thèse N°: 2019IMTA0136

Rapporteurs avant soutenance :

Chadi BARAKAT	Directeur de recherche	Sophia-Antipolis Research Center
Daniel NEGRU	Maître de conférences (HDR)	LaBRI - Université de Bordeaux

Composition du Jury :

Président :	Bernard COUSIN	Professeur, Université de Rennes 1
Rapporteurs :	Chadi BARAKAT Daniel NEGRU	Directeur de recherche, Sophia-Antipolis Research Center Maître de conférences (HDR), LaBRI - Université de Bordeaux
Examinateurs :	Lucile SASSATELLI Marco CAGNAZZO	Maître de conférences, Université de Nice Sophia Antipolis Professeur, Telecom ParisTech
Directeur de thèse :	Loutfi NUAYMI	Professeur, IMT Atlantique
Co-directeur de thèse :	Yannick LE LOUEDEC	Chef de projet R&D, Orange Labs
Invité :	Gwendal SIMON	Professeur - *en disponibilité* -, IMT Atlantique (1er Directeur de Thèse)

DEDICATION

For all those who have instilled in me the love of learning,
who have guided me through the darkness to the light,
who have pushed me to be a better version of myself,
for my parents, my teachers, and my husband,
I dedicate this dissertation.

ACKNOWLEDGEMENTS

Many people have accompanied and supported me during the last three years. I would never have been able to finish my Ph.D thesis without their support. I would like to express my gratitude for all of them.

First of all, I would like to thank my thesis defense committee for their helpful comments and invaluable questions.

Second, I would like to thank my advisors, Prof. Gwendal Simon and Prof. Loutfi Nuaymi. They gave me the opportunity to pursue my Ph.D. They believed in my capacities to successfully carry out this professional goal, and they have gradually guided me to be an independent researcher. I appreciate their advises and their continuous presence for my research work. These last three years haven't always been easy but knowing my advisors care helps a lot with getting by. I feel grateful and lucky to be their student.

Besides, I am deeply thankful to my co-advisor Mr. Yannick Le Louedec. I am very grateful to him for helping me overcome the difficult moments of my research work and for his constant enthusiasm and encouragement. Taking the time to talk with me during hours at Orange Labs and helping me develop my ideas means the world to me.

Also, I would like to thank my colleagues at Orange Labs and at IMT Atlantique for their technical help and their daily exchange. Particularly, my sincere thanks to Xavier Corbillon for the great collaboration. He has never hesitated to share the findings of his Ph.D work with me, and he was always available for great discussions.

My warmer thanks to my parents, my sister, my brothers and, my nieces and nephews. They are an endless dose of love, energy, and optimism despite the geographical distance between us.

I am also deeply grateful to my husband who has always encouraged me to make it till the end ♡. My sincere thanks to my friends, especially Malek and Farah, and all the people who have made this journey exceptional.

LIST OF PUBLICATIONS

JOURNALS

2019

- Mariem Ben Yahia, Yannick Le Louédec, Gwendal Simon, Loutfi Nuaymi and Xavier Corbillon: "HTTP/2-Based Frame Discarding for Low-Latency Adaptive Video Streaming," In: ACM Transaction on Multimedia Computing, Communications and Applications, 15, 1, Article 18 (February 2019), 23 pages.

PATENTS

2018

- Mariem Ben Yahia, Yannick Le Louédec, Gwendal Simon and Loutfi Nuaymi: "Méthode et dispositif de diffusion de vidéo à 360 degrés", (Submitted on August 2018).

INTERNATIONAL CONFERENCES

2018

- Mariem Ben Yahia, Yannick Le Louédec, Gwendal Simon and Loutfi Nuaymi: "HTTP/2-Based Streaming Solutions for Tiled Omnidirectional Videos," In: proceedings of 2018 IEEE International Symposium on Multimedia (ISM), December 2018.

2017

- Mariem Ben Yahia, Yannick Le Louédec, Loutfi Nuaymi and Gwendal Simon: "When HTTP/2 Rescues DASH: Video Frame Multiplexing," In: proceedings of IEEE Conference on Computer Communications Workshops (Infocom Workshops), May 2017.

Résume de la thèse en français

Le streaming vidéo est devenu la principale source de trafic dans les réseaux mobiles selon Cisco [17] et devrait représenter plus de $80\,\%$ du trafic mobile d'ici 2021. De nouveaux services comme la TV en ligne et les vidéos immersives contribuent à cette demande croissante. A titre d'exemple, $59\,\%$ des utilisateurs en 2017 préfèrent regarder des vidéos en direct en ligne que sur une télévision traditionnelle [79]. D'autres plateformes de streaming en direct telles que YouTube et Facebook permettent aux utilisateurs d'Internet de visionner un contenu vidéo en temps réel et d'intéragir en postant des commentaires. Afin que les spectateurs soient à l'abri des spoilers, typiquement lors des événements sportifs, un très faible délai de latence est alors indispensable.

Le service de diffusion en direct doit fournir un délai très faible entre le moment où une vidéo est générée et l'instant auquel elle est affichée sur l'écran du client. Ces services sont de plus en plus basés sur le protocole HTTP. Des technologies de streaming adaptatif ont été mises en place afin d'ajuster la qualité de la vidéo transmise aux clients aux ressources réseaux disponibles. Le concept se présente de la manière suivante : Premièrement, la vidéo originale est encodée en plusieurs niveaux de qualité également appelés représentations. Puis, chaque représentation est divisée en segments temporels dont la durée est typiquement de $2\,\mathrm{s}$ à $10\,\mathrm{s}$. Enfin, au début de chaque segment vidéo, le client prédit la bande passante disponible et exécute un algorithme d'adaptation de débit vidéo. Ce qui devrait lui permettre d'ajuster la qualité de la vidéo affichée aux conditions réseaux.

Pourtant, une mauvaise prédiction de la bande passante peut entrainer un mauvais choix de représentation ce qui augmente les délais de livraison du segment en cas de surestimation de la bande passante. C'est notamment le cas dans les réseaux mobiles où la prédiction du débit est

difficile [64]. Pour absorber les instabilités du réseau survenant au cours de la lecture d'un segment, les implémentations actuelles intègrent une mémoire tampon dans le lecteur vidéo dont la taille recommandée est de trois segments vidéo, soit de $6\,\mathrm{s}$ à $30\,\mathrm{s}$ dans le contexte de la vidéo à la demande. Par contre, cette taille doit être maintenue aussi petite que possible dans le contexte de la diffusion en direct. Si la mémoire tampon se vide avant la fin du segment en cours, le client est contraint au rebuffering, c'est le fait d'interrompre la lecture de la vidéo en attendant de recevoir plus de données. Des mesures récentes ont montré que $29\,\%$ des sessions de streaming vidéo présentent au moins un rebuffering ce qui cause l'accumulation des retards, un dépassement de la latence tolérée pour les services de streaming en direct et impacte très négativement la qualité d'expérience des utilisateurs. D'autres mécanismes sont alors nécessaires afin de pouvoir adapter la qualité vidéo au réseau et éviter le rebuffing en attendant un choix de représentation moins conséquent pour le segment suivant.

Le rejet de certaines trames vidéo est une des solutions permettant d'alléger la taille des données d'un flux vidéo transmis. Elle consiste à rejeter volontairement des images vidéo sur le réseau afin d'éviter l'accumulation des retards. Côté client, le lecteur vidéo ne met pas la vidéo en pause. il répète généralement la dernière image affichée. Cependant, HTTP/1 ne convient pas à la mise en œuvre de ces solutions car il ne permet pas d'annuler facilement une partie du flux vidéo qui a déjà été demandé au serveur sans établir une nouvelle session TCP, ce qui induit trop de délais. Pour résoudre ce problème, nous proposons dans la première partie de cette thèse des stratégies de rejet de trames vidéo basées sur le protocole HTTP/2 publié en 2015.

HTTP/2 est désormais implémenté par la plupart des navigateurs principaux comme chrome, Edge, Firefox et Safari. Bien que HTTP/2 ait été principalement conçu pour les services Web, les chercheurs ont aussi étudié les fonctionnalités HTTP/2 pour la diffusion vidéo, mais les efforts ont été particulièrement axés sur la fonctionnalité qui permet au serveur de pousser un contenu au client avant qu'il l'ait demandé. Parmi ses autres nouvelles fonctionnalités, HTTP/2 considère chaque requête/réponse comme un flux et multiplexe tous les flux dans une seule session TCP. Il permet au client de réinitialiser un flux (donc une demande en cours) sans établir une nouvelle session

TCP. Le client peut aussi donner des priorités à ses requêtes d'une façon statique ou dynamique. Nous explorons dans cette thèse des solutions basées sur les fonctionnalités de multiplexage et de réinitialisation des flux HTTP/2 dans un cadre de streaming vidéo en faible latence.

Dans la première partie de cette thèse, nous suggérons d'utiliser le protocole HTTP/2 pour concevoir une stratégie de rejet d'images vidéo. Nous abordons au chapitre 3 les fonctionnalités du HTTP/2 qui conviennent à notre proposition. Notre idée principale consiste à ce que le client demande chaque image vidéo dans un flux HTTP/2 dédié. Il devient alors possible de contrôler la livraison des images vidéo par appel aux fonctionnalités HTTP/2 au niveau des flux transmettant les images concernées. Nous utilisons la fonctionnalité de réinitialisation de flux HTTP/2 pour rejeter des images vidéo. Ensuite, nous proposons un modèle optimal déterminant l'ensemble des trames vidéo à rejetter qui impacte le moins la qualité. Nous développons aussi d'autres algorithmes qui peuvent être implémentés en pratique. Nous évaluons nos algorithmes en utilisant des vidéos dynamiques et statiques et des traces de réseaux cellulaires et WiFi. Nous évaluons nos résultats avec diverses métriques spatiales et temporelles. Les algorithmes proposés évitent l'accumulation de retard tout en présentant une qualité acceptable de la vidéo affichée. Ils réduisent la distorsion de la qualité, en particulier pour les vidéos dynamiques en les comparant avec des statégies du rejet d'image vidéos uniquement au niveau de la mémoire tampon du client. Nous concluons également que la fonction de pondération de HTTP/2 n'est pas nécessaire pour nos algorithmes car il n'est pas nécessaire d'entrelacer les images vidéos mais il vaut mieux les envoyer dans l'ordre de décodage vidéo en configurant la fonctionnalité de dépendance entre flux HTTP/2.

D'autres services contribuent à la hausse du traffic vidéo mobile comme les vidéos immersives qui sont devenu un élément clé du domaine de la réalité virtuelle. En 2017, son adoption s'accroît principalement en raison de l'émergence des casques commerciaux comme Google Daydream et Samsung Gear VR pour des usages avec smartphones, et l'Oculus Rift et le HTC vive, pour des usages avec ordinateurs. Une vidéo 360° permet à son utilisateur à chaque instant de regarder une partie seulement de la vidéo sphérique complète. Cette partie, appelée fenêtre d'affichage, dépend de l'orientation de la tête de l'utilisateur et de la taille de l'écran dans son terminal, appelé

visiocasque, ou casque HMD. Pour assurer une bonne qualité d'expérience, aussi appelée qualité d'immersion virtuelle, et pour éviter la cinétose, le terminal de l'utilisateur doit adapter à chaque instant le contenu audiovisuel dans la fenêtre d'affichage en fonction des mouvements de tête de l'utilisateur.

Une première technique de diffusion de vidéo à 360° consiste à transmettre au terminal de l'utilisateur l'intégralité du contenu de la sphère vidéo. Le client se charge alors localement d'extraire de la sphère vidéo la partie à insérer dans la fenêtre d'affichage du casque de l'utilisateur. Cette technique présente l'inconvénient de transporter une quantité de données très supérieure à celle qui est réellement exploitée par le terminal de l'utilisateur puisque la fenêtre d'affichage représente environ 15% seulement de la sphère vidéo complète. Une deuxième technique de diffusion a pour objectif de résoudre ce problème, à savoir réduire la quantité de données transportées. La sphère vidéo est d'abord projetée sur un plan à deux dimensions (2D). Puis elle est découpée spatialement en plusieurs parties appelées tuiles, formant par exemple un quadrillage du plan. Ensuite chaque tuile est encodée indépendamment des autres tuiles composant la vidéo. Chaque tuile peut ainsi être décodée indépendamment des autres tuiles sur le terminal de l'utilisateur. Plus précisément, chaque tuile est encodée en plusieurs versions correspondant à différents niveaux de qualités ou à différentes résolutions, par exemple, au moins une version à qualité haute et au moins une version à qualité basse. La vidéo est découpée temporellement en segments, par intervalle de temps. La durée des intervalles de temps (et donc la durée des segments) est fixe pour toute la vidéo, et est typiquement de l'ordre d'une ou plusieurs secondes. Puis, pour chacun des segments de la sphère vidéo, et au cours de l'intervalle de temps précédant l'affichage du segment, le client prédit la fenêtre d'affichage pour le segment suivant (c'est-à-dire l'orientation donnée par la tête de l'utilisateur au visiocasque, au cours du prochain intervalle de temps). Le client doit ainsi demander et recevoir en qualité haute les tuiles qui recouvrent la fenêtre d'affichage prédite pour le prochain intervalle de temps et aussi demander et recevoir en qualité basse les autres tuiles en dehors de la fenêtre d'affichage prédite. Ces tuiles en qualité basse permettent de maintenir l'affichage, si nécessaire en qualité basse, de la vidéo lorsque l'utilisateur effectue des mouve-

ments de tête très marqués et imprévus (i.e. en dehors de la fenêtre d'affichage prédite). En effet, afficher une qualité basse dans toute ou une partie de la fenêtre d'affichage provoque certes une dégradation de la qualité d'expérience pour l'utilisateur, mais elle est préférable à l'affichage d'une image fixe ou d'un écran noir. L'inconvénient de cette seconde technique est une dégradation de la qualité d'expérience ressentie par l'utilisateur lorsque la prédiction n'est pas parfaite. Entre deux instants d'affichage, plus on attend pour prédire l'orientation du visiocasque susceptible d'être prise au prochain instant d'affichage, plus cette prédiction est précise, mais moins il reste de temps au procédé pour émettre les requêtes et recevoir les tuiles nécessaires en réponse.

Nous proposons dans la deuxième partie de cette thèse de remédier à ce problème et de permettre au client de demander les tuiles vidéos suffisamment tôt pour bénéficier de la totalité de la bande passante disponible tout en tenant compte de l'évolution de la précision de la prédiction de la fenêtre d'affichage dans le temps. Nous améliorons l'estimation de la fenêtre d'affichage à l'aide d'au moins une seconde estimation, tout en garantissant la réception de toutes les tuiles nécessaires. Nous proposons un modèle de diffusion vidéo à 360° dans lequel le client demande toutes les tuiles vidéo en avance d'une seconde et ajuste ensuite ses décisions inappropriées $50\,\mathrm{ms}$ plus tard. Notre proposition s'appuie sur HTTP/2. Le client demande et reçoit chaque tuile dans un flux HTTP/2 différent et utilise les fonctions de priorité, de pondération et de réinitialisation des flux HTTP/2 pour ajuster la livraison des tuiles au fil du temps. Si après la seconde estimation il ne reste plus suffisamment de temps pour requérir à nouveau toutes les tuiles nécessaires, la première salve de requêtes garantit la réception des tuiles manquantes, même si elles ne sont pas forcément toutes du niveau de qualité correspondant à la seconde estimation. Nous détaillons dans le chapitre 5 le modèle proposé ainsi que des algorithmes pratiques qui utilisent des variantes de la fonctionnalité de poids de HTTP/2. Nous les comparons dans le chapitre 6 avec d'autres algorithmes utilisant une seule occurance de de prédiction de fenêtre d'affichage ou utilisant HTTP/1. L'évaluation des performances en terme de bande passante consommée et de qualité des pixels de la fenêtre d'affichage montrent que notre proposition s'adapte automatiquement aux ressources réseau disponibles et donne la priorité à la livraison des tuiles de la fenêtre d'affichage.

Nous démontrons également que le choix de la fonction de pondération HTTP/2 est critique afin de hiérarchiser les tuiles de la fenêtre d'affichage car la fonction de pesage détermine la stratégie d'entrelacement entre les flux HTTP/2 transportant les tuiles vidéo.

Dans cette thèse, nous avons montré que les systèmes de livraison des contenus pour des services à faible latence basés sur HTTP/1 ne peuvent toujours s'adapter aux contraintes du réseau. Par conséquent, nous avons proposé de nouveaux systèmes s'appuyant sur le protocole HTTP/2 en tirant profit des fonctionnalités de réinitialisation du flux et d'établissement de dépendance et de poids de priorité entre les flux HTTP/2. Des efforts d'implémentation de ces fonctionalités au niveau du lecteur vidéo sont nécessaire afin que le le serveur et le client puissent communiquer en HTTP/2, annuler des demandes en cours et attribuer des pondérations aux flux. HTTP/2 peut être exploré dans le futur pour concevoir des solutions de livraison des contenus audivisuels en faible latence. A titre d'exemple, il est intéressant de modéliser des solutions basées sur HTTP/2 et l'encodage en plusieurs niveau (SVC) afin d'ajuster la livraison des niveaux supplémentaires de la qualité en fonction des ressources réseaux disponibles.

Les travaux de cette thèse ont donné lieu à la publication d'un papier journal et de deux articles scientifiques dans des conférences internationales. Un brevet concernant la partie 360° a aussi été déposé.

ABSTRACT

The new version of the Hypertext Transfer Protocol (HTTP) standard, namely HTTP/2, is now supported by most of the leading web browsers. HTTP/2 was developed to make best use of network resources and, efficiently send and receive data. Among the novelties released in HTTP/2, only the push feature, that enables the server to push content before the client requests it, has received the attention of researchers in the multimedia community. However, HTTP/2 also features a novel mechanism to multiplex the delivery of structured data known as HTTP/2 streams. Its also enables the client to set priorities among the requests and to reset an ongoing stream.

In this dissertation, we examine the benefits of using the HTTP/2 reset stream and priority features in the context of low latency streaming services. Today, HTTP Adaptive Streaming (HAS) is widely used by Over The Top (OTT) platforms. The video content is encoded at different quality levels, called representations. The client is equipped with a rate adaptation algorithm that dynamically decides the best representation to request based on a prediction of the bandwidth. However, inaccurate predictions can happen and may lead to a decrease of the Quality of Experience (QoE). We propose throughout this study HTTP/2-based delivery strategies in order to deal with the prediction errors and insure a continuous playout for different services.

The first service concerns live streaming for traditional/non-immersive videos in constrained networks where HAS throughput prediction is prone to errors. In this case, the video playout may suffer from rebuffering events and an increasing delay between the original video flow and the displayed one, which decreases the QoE. We propose to use HTTP/2 priority and reset stream features to insure the delivery of the most important components of the video stream while stopping the delivery of the least important ones in order to optimize the bandwidth consumption. We design an optimal algorithm, which computes *a posteriori* the best scheduling scheme of video frames that minimizes the distortion, given the awareness of the network variations. It provides an upper bound of the best achievable displayed video quality, which is useful for performance evaluation. We also presents heuristic algorithms that can be implemented in practice in any video client. The results

show that our proposed heuristics behave closely to the optimal algorithm. Our proposed strategy avoids the accumulation of delay while presenting an acceptable displayed video quality.

The second service is about streaming 360° videos in accordance with the viewer head movements. A 360° video allows the viewer to move around the camera giving him control of what he sees, also called viewport. However, it presents two major technical challenges: network resource consumption and the quality of the viewport. Dynamically adapting the content delivery process to the user behavior is a promising approach to ensure both important network resource savings and satisfying experiences. However, predicting the user behavior is challenging. In this thesis, we propose to leverage HAS, tiled-based 360° video encoding and HTTP/2 features to implement a dynamic content delivery process. The 360° video stream is spatially encoded into tiles and temporally divided into segments. The client executes two viewport predictions for each segment, one before and one during its delivery. Upon every prediction, the client decides on a priority and a quality level for each tile; tiles overlapping with the predicted viewport get higher priorities and quality levels. Then, we exploit the reset stream feature of the HTTP/2 protocol to save network resources. The results show that our strategy provides a high quality on the viewport pixels, a low ratio of the delayed viewport pixels in bandwidth-constrained networks, and a reduction of the bandwidth consumption, up to 12% compared to the alternative schemes exploiting 2 viewport predictions per video segment.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

B Bidirectional-predicted.

CDN Content Delivery Network.

CMAF Common Media Application Format.

CP Content Provider.

DASH Dynamic Adaptive Streaming over HTTP.

DNS Domain Name System.

FCFS First Come First Served.

FIFO First In First Out.

FoV Field of View.

fps frames per second.

GLPK GNU Linear Programming Kit.

GoP Group of Pictures.

H264 MPEG-4 Part 10 Advanced Video Coding.

HAS HTTP Adaptive Streaming.

HD High Definition.

HDS HTTP Dynamic Streaming.

HEVC High Efficiency Video Coding.

HLS HTTP Live Streaming.

HMD Head Mounted Display.

HTTP Hypertext Transfer Protocol.

HTTP/1 First version of the standardized hypertext transfer protocol.

HTTP/1.1 Version 1.1 of the standardized hypertext transfer protocol.

HTTP/2 Second version of the standardized hypertext transfer protocol.

I Intra-predicted.

ILP Integer Linear Programming.

IQR Interquartile Range.

ISOBMFF ISO Base Media File Format.

ISP Internet Services Provider.

MOS Mean Opinion Score.

MPD Media Presentation Description.

MS-SSIM Multi-Scale Structural Similarity for IMage.

MSS Microsoft Smooth Streaming.

NAT Network Address Translation.

OTT Over The Top.

P Inter-predicted.

PSNR Peak Signal to Noise Ratio.

QoE Quality of Experience.

RFC Request for Comments.

RoI Region of Interest.

RR Round Robin.

RTSP Real Time Streaming Protocol.

RTT Round Trip Time.

SD Standard Definition.

SDR Spatial Representation Description.

SVC Scalable Video Codec.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

VOD Video On Demand.

VR Virtual Reality.

Chapter 1

Introduction

“Prediction is very difficult,

especially if it’s about the future."

- Niels Bohr -

Contents

Video streaming has become the major source of traffic in mobile networks and is expected to represent more than 80% of mobile traffic by 2021 according to Cisco [17]. In 2016, Youtube lists more than 4 billion videos watched a day and Facebook more than 8 billion. New services like Over The Top (OTT) live streaming and 360° video streaming (also called panoramic or omnidirectional videos) are recently contributing to this increasing demand.

Live streaming should provide a very low delay between the time a video is generated and the time it is displayed at the client side. OTT live streaming systems through the public internet network are based on the Hypertext Transfer Protocol (HTTP) protocol. They are gradually replacing the traditional live TV. According to Koeppel Direct Blog [53], 59% of users prefer watching live video online than on a traditional live TV in 2017. Besides, live streaming platforms such as

Youtube and Facebook allow Internet users to view a video content in real time and interact with the video by asking questions or writing comments [100]. In 2017, over 360 million users regularly watch Facebook live sessions and about 200 million regularly watch Instagram live sessions [66]. Sport and competition events represent 29% of the online consumed content [53] and they require a low latency delays so that viewers are safe from online spoilers.

Video streaming is an ongoing challenge in the way videos are produced, consumed, and distributed via the network. The principal actors of the video streaming market are the Content Providers (CPs), the Internet Services Providers (ISPs) and the Content Delivery Networks (CDNs). CPs such as media companies and e-commerce vendors ask CDN Providers to deliver their content to their end-users. CDNs are widely deployed to improve the performance and scalability of video streaming. They were originally used to reduce the delivery latency experienced by the end users by redirecting their requests to surrogate servers, which also lighten the load on the origin ones. 39% of the Internet traffic was carried by CDNs in 2014. This percentage should increase to 62% in 2019 [17]. In turn, a CDN can host its servers in data centers of ISPs.

Several steps are needed for video streaming as described in Figure 1.1. First, the video is captured and sent to an origin server. This server encodes the video which compresses its volume and facilitates its delivery. Second, a client player requests the video content. In order to enable a scalable video delivery, this client request is redirected to an intermediate CDN server (using Domain Name System (DNS) or HTTP based redirection methods). Third, the CDN ingests the content from the remote origin server. Finally, the CDN server sends the content as a response to its request to the client player which decodes the video before displaying it. It is important to give users a real-time service despite the complexity of the end-to-end media pipeline and the variations of the network resources [25, 72].

Several techniques aim to adapt the video content to the network performances such as HTTP Adaptive Streaming (HAS) [18, 100]. HAS introduces a client-controlled delivery of the video in order to dynamically adapt it to varying bandwidth and to the viewing device characteristics. HAS presents the advantages of being CDN friendly which drastically reduces the load on source servers

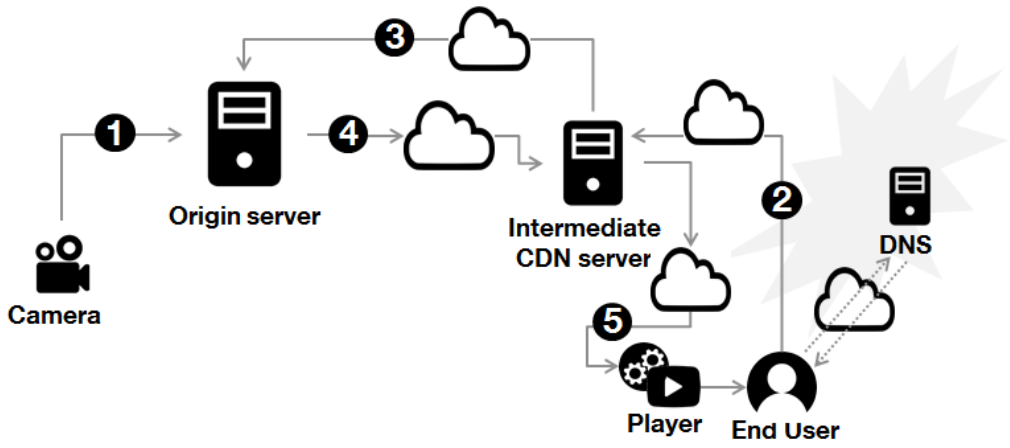


Figure 1.1: A general call flow of a video streaming service including a CDN

and increases scalability. There are several existing HAS implementations, such as the proprietary Adobe HTTP Dynamic Streaming (HDS), Apple HTTP Live Streaming (HLS), Microsoft Smooth Streaming (MSS), and the only international standardized solution Dynamic Adaptive Streaming over HTTP (DASH). All of them follow nearly the same principle that we briefly describe in Figure 1.2.

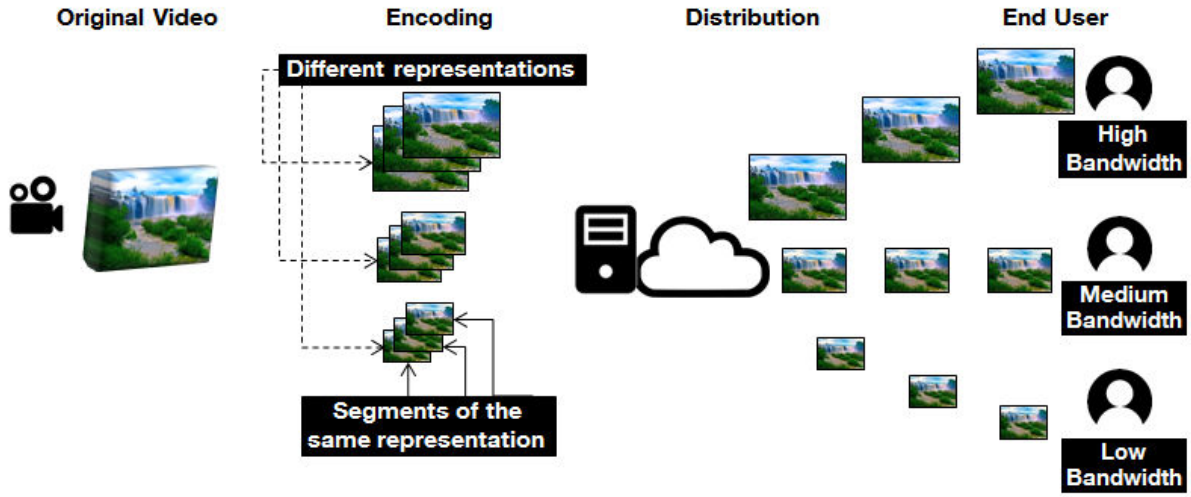


Figure 1.2: Simplified concept of HAS

First, the original video is encoded into several quality levels also called representations. Then, each representation is divided into temporal segments which duration is typically of 2 s to 10 s [59]. Finally, based on *a bandwidth prediction mechanism*, a HAS client executes a rate adaptation

algorithm to request each video segment so that the quality of the video matches the network performance. In this thesis, we analyze the impact of inaccurate bandwidth prediction on the video display and we propose solutions to reduce the streaming latency.

360° video streaming is another service that has emerged recently and has become a key component of the Virtual Reality (VR) services. Since 2017, its popularity is increasing primarily due to commercial headsets like the Oculus Rift and HTC Vive, designed for computers [93] and, Google Daydream and Samsung Gear VR, designed for mobile phones [24, 82].

A 360° video allows the viewer to look in all the directions of a captured sphere across a multitude of yaw, pitch, and roll angles as described in Figure 1.3. The viewer can control his viewport by moving his head, which offers him a more immersive experience compared to traditional videos. The portion of the video watched by the viewer at any given time is known as the *viewport* and must be immediately available at a good quality when the viewer moves his head in order to ensure a good Quality of Experience (QoE).

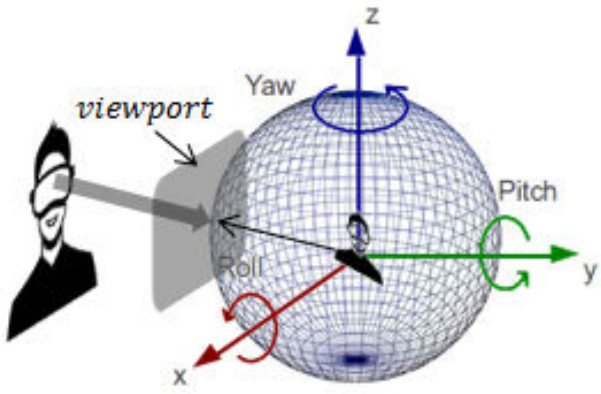


Figure 1.3: Head movement orientation of a 360° video viewer

360° video streaming presents two major technical challenges: network resource consumption and QoE [35]. In fact, streaming the entire video at the same quality, while the viewer only watches a spatial portion of it, leads to limit the video quality on the viewport and is also considered as a waste of bandwidth due to the delivery of the unwatched portions. 360° video streaming brings an additional adaptation challenge: the video content should also match the viewer's head movements.

Encoding techniques based on *tiling* refer to a spatial partitioning of a video where tiles cor-

respond to independently decodable video streams [32]. A HAS client is then able to select and retrieve spatial video portions that are relevant to the user experience. When tiling is combined with *viewport prediction algorithms*, the client may request the viewport of a $360°$ video at a high quality and the rest of it at a low quality. In this case, HAS client can adapt to both the viewer behavior and the network performances, which aim to maximize the QoE and reduce the bandwidth consumption.

1.1 Motivation and problem description

The proper functioning of adaptive streaming solutions depends on its accuracy to predict external factors such as the bandwidth and also the viewport in the specific case of immersive video streaming. However, these predictions are not perfect and are prone to errors which penalize the entire process of adaptation [42].

Poor bandwidth prediction, notably in mobile networks where throughput prediction is hard [64, 87], may increase segment delivery delays [65]. If a HAS client notices that the video quality overloads the network, it opts for a lower representation for the next segment. To absorb network instabilities occurring in a segment duration, the current implementations integrate a *buffer* in the video player which recommended size is conservatively set to the duration of three video segments, so $6\,\mathrm{s}$ to $30\,\mathrm{s}$ [69, 97]. If the buffer gets drained before the end of the current segment, the client is forced to rebuffering. While it is identified as the most impacting factor of QoE degradation [55, 62, 89], recent measures showed that $29\,\%$ of the video streaming sessions exhibit at least one video rebuffering [19]. This issue is aggravated in the case of live streaming, where the client buffer has to be kept as small as possible as advocated by Swaminathan and Wei [95] and Bouzakaria et al. [12]. Rebuffering causes the accumulation of delays, which then exceed the maximum tolerated latency threshold for live streaming services and negatively impacts the QoE [29].

In the case of $360°$ videos, predicting the viewport should enable the client to request in advance the right portions of the video sphere (which are in the viewport) at a high quality. However, the viewport prediction is also challenging since it depends on several factors such as the head ori-

entation, the movement speed, and the audio-visual content [7, 84, 108, 110]. Viewport prediction is prone to errors and its accuracy depends on the time scale of the prediction. According to Bao et al. [7], an accurate viewport position can be predicted in a time scale ranging from $100\,\mathrm{ms}$ to $500\,\mathrm{ms}$ in advance before the start of the segment display. However, a HAS client needs to request the next segment to be displayed no later than a segment duration in advance which is at least equal to $1\,\mathrm{s}$ typically. So, the client cannot wait until its viewport prediction becomes accurate to request the next video segment since the requested data may exceed the available network resources. Therefore, the client shall request the next segment to display before the availability of an accurate viewport prediction which may lead to incorrect quality distribution among the requested tiles and may badly impact the QoE.

For both the bandwidth and the viewport predictions, the client may realize it has made bad choices a few milliseconds later. However, HTTP/1 does not enable the client to stop an ongoing request without setting a new TCP/HTTP session, which causes too much delay. The HTTP/1 client must thus complete all requests before being able to issue another requests. Fortunately, since 2015, a new version of the HTTP standard, called HTTP/2 was published [10] and it is now supported by most leading browsers (Chrome, Edge, Firefox and Safari) [13]. Among the novel features released in HTTP/2, the reset stream feature enables the client to cancel an ongoing request and the priority feature enable a client to specify a priority guideline of its requests to the server. Although HTTP/2 has been primarily designed for web services, researchers have studied HTTP/2 features for video delivery but the efforts were particularly focused on the feature that enables the server to push content [16, 104, 107].

This thesis addresses open questions in the area of meeting the requirements of low latency streaming services by using HTTP/2 priority and reset stream features. We focus on the DASH implementation of HAS. However, the findings of our work are valuable for all HAS implementations.

1.2 Contributions

The main contributions of this thesis are in the field of efficient and adaptive techniques to deliver low latency streaming videos in constrained networks. We contribute to improve the state-of-the-art of traditional and immersive video streaming over HAS by using HTTP/2 protocol. In the following we list the two major contributions made in this thesis.

HTTP/2-based solutions for traditional video streaming with low latency requirement: The main question of this part is about how to avoid rebuffering events during a video display and keep the delay between an original and a displayed video streams small in constrained networks.

Our main idea take profit from HTTP/2 features in order to prioritize the delivery of important video frames in case of bandwidth fluctuation. We propose a video frame discarding model by leveraging the concept of HTTP/2 *stream*, and the associated features of reset stream, dependency, and priority. HTTP/2 streams enable the client to act on the delivery of structured data. We propose to *send each encoded video frame in a different HTTP/2 stream*. We develop an optimal model which decides the best set of video frames to drop taking into account their importance. We also propose practical algorithms that can be implemented in practice and we evaluate our proposals by extensive tests based on real network and video traces.

HTTP/2-based solutions for viewport-adaptive immersive video streaming: The main question of this part is about how to enable the client to request the $360°$ video portions early enough to benefit from the totality of the available bandwidth while taking into account the accuracy evolution of its viewport prediction over time.

We propose a $360°$ video delivery model where the client requests all video tiles $1\,\mathrm{s}$ in advance and then adjusts its inappropriate decisions $500\,\mathrm{ms}$ later. Our proposal leverages the HTTP/2 priority, weight and reset stream features to handle the tiles delivery over time. We propose to *send each tile in a different HTTP/2 stream*. We develop practical algorithms that use different weight strategy in order to schedule the video delivery of tiles. We evaluate these algorithms with intensive

simulation tests. We demonstrate that the choice of the HTTP/2 *weight* function is a critical factor since it determines the interleaving strategy between appropriate and inappropriate tiles.

1.3 Thesis outline

The following parts of this dissertation are organized as follows. In Chapter 2, we familiarize the reader with the research fields of this thesis. Then, we detail our contribution in the context of live streaming in Part I and in the context of immersive video streaming in Part II.

In Part I, we present the main principles of our proposed HTTP/2-based frame discarding model in Chapter 3 and we propose optimal and practical algorithms. In Chapter 4, we evaluate our proposals with intensive tests. We describe our simulation environment and the datasets we use as well as the results we obtain. In Part II, we describe in Chapter 5 the main principles of our proposal to enhance the viewport-dependent 360° video streaming. In Chapter 6, we describe our simulation environments and we conduct a performance evaluation. Finally, we summarize this dissertation and discuss future works in Chapter 7.

Chapter 2

Background

"Science never solves a problem without creating ten more."

- George Bernard Shaw -

Contents

The contributions of this thesis exploit specific features from modern video coding techniques, HAS, and HTTP/2. We present them in the following in order to underline the main aspects of these techniques.

2.1 Dynamic adaptive streaming

Adaptive streaming technologies, and DASH in particular, enable dynamical adaptation of the video quality to external conditions. The MPEG-DASH standard was published as ISO/IEC 23 009-1:2012 in April 2012 [43]. Its core principles are the following. The audiovisual content is encoded into several versions called representations, each with a specific video quality level and resolution. Each representation then goes through a segmentation process to make the content available as a set of web objects.

Two key types of files are generated by the server and used by the client in DASH-based content delivery: (i) the Media Presentation Description (MPD), which provides a set of meta data information about the video content; and (ii) the video segments, which contain the media data that is fetched by clients as web objects (with HTTP GET requests).

2.1.1 Bandwidth adaptive streaming

A DASH client executes a rate-adaptation algorithm at each new request to make the video representation bitrate match the network bandwidth. The rate adaptation algorithms are not part of the standard and depend on the vendor implementations [4, 56]. Rate adaptation algorithms can take into account several input information such as throughput prediction [47, 61, 113], the buffer fullness state [40, 78, 91] and the network characteristics [8, 90].

Karagkioules et al. [49] conduct a comparison study of rate-adaptation algorithm. They show that buffer-based approaches may lack in stability for small buffers, which is common in live streaming services. In this case, buffer-based algorithms are more probable to experience a rebuffering event. Spiteri et al. [92] conclude similar results. The buffer in low latency streaming must be smaller than the targeted latency bound. They show that throughput-based adaptation algorithms are the most appropriate for low latency scenarios.

Throughput-based adaptation algorithms differ in how they predict the throughput [56]. According to Spiteri et al. [92], the current implemented rate-adaptation algorithm in dash.js and in hls.js for low latency streaming predicts the average throughput, noted b, based on the history of

the past successfully downloaded segments and selects for the next video segment the representation (quality level) with the highest available bitrate yet lower than $0.9 * b$. However, Karagkioules et al. [49] have also mentioned that throughput-based algorithms present insufficiency to match the representation to the available throughput due to the significant throughput variation, particularly in mobile networks.

2.1.2 A reality check of the throughput-based rate adaptation mechanism

As an introduction to our research studies, we have conducted an experiment with state-of-the-art DASH algorithms and standard mobile network conditions to check whether it is common in mobile networks to experience situations where the bitrate of the requested video representation is higher than the network throughput. We call *throughput deficit* the difference between the requested representation bitrate and the network throughput, and *throughput deficit event* such a situation. The experimentation is based on the following configuration:

- **Rate-adaptation algorithm:** In this simulation, we use a throughput-based adaptation algorithm. It estimates the average throughput, noted b, from the previous video segment delivery and selects for the next video segment the representation with the highest bitrate yet lower than $0.9 * b$.

- **Network traffic traces:** We use publicly available network traces captured by terminal devices on 4G network [103]. These traces provide the average network throughput every second. The average throughput is $30\,\mathrm{Mbps}$ and it varies from $1\,\mathrm{Mbps}$ to $70\,\mathrm{Mbps}$.

- **Video representations**: We consider six video representations, with bitrates ranging from $5\,\mathrm{Mbps}$ to $30\,\mathrm{Mbps}$.

We apply the selected throughput-based adaptation algorithm on $60\,\mathrm{s}$-long period of the network traces. We check every second whether the network throughput is lower than the selected representation bitrate, which corresponds to what we call a (one-second long) throughput deficit

event. Then, for each trace period, we calculate the mean and the median of the throughput deficit in these events. We show results in Table 2.1.

Segment duration	% of seconds with deficit	% of deficit amplitude	
		Mean	Median
2 s	23.5	23.8	17.7
4 s	23.1	24.8	18.9
6 s	19.0	25.0	22.7
8 s	19.0	25.0	21.2
10 s	15.4	26.3	18.2

Table 2.1: Throughput deficit on DASH experimentation

These results show that the video player experiences one-second long throughput deficit events one fifth of the time. A standard video player does not necessarily pause the video display to re-fill its buffer on each throughput deficit event, but they may jeopardize the video experience in low-latency scenarios with small buffer sizes. Furthermore, the throughput deficit represents a significant proportion of the video bitrate (around one quarter $25\,\%$) during these events.

Houze et al., Swaminathan and Wei [38, 95] propose to set shorter segments to switch faster between representations and avoid the accumulation of delays. However, short segments also lead to an excessive rate of switching between representations. Karagkioules et al. [49] and Duanmu et al. [30] conclude that frequent switching between representations decreases the client satisfaction. We conclude that traditional throughput-based adaptation algorithms can be insufficient to mitigate throughput deficit events.

2.2 Video encoding

Without video compression, the video streaming that revolutionized the Internet would not have been possible [45]. The advancement of video compression enables to minimize redundant digital information from a raw video sequence by making it a coded video stream. Video compression algorithms must specify an encoder for compressing the video, and a decoder for reconstructing the original one [96]. The encoder and the decoder together constitute a codec.

Video compression reduces the memory consumption and the cost of delivery. A codec should encode a series of images in the lowest possible number of bits while insuring a trade-off between compression rate and reconstruction error. The number of video frames (encoded images) composing a $1\,\mathrm{s}$-video segment is known as the frames per second (fps). Usually, it ranges from $25\,\mathrm{fps}$ to $30\,\mathrm{fps}$ for traditional videos and it is recommended to range from $60\,\mathrm{fps}$ to $90\,\mathrm{fps}$ for high quality immersive videos [57].

Video compression uses modern coding techniques, such as MPEG-4 Part 10 Advanced Video Coding (H264)/AVC [105] and H265/High Efficiency Video Coding (HEVC) [44], to reduce redundancy in video data. However, HEVC exhibits efficient video compression by reducing the video file size up to 50% as compared to H264. It also enables tile-based streaming [68, 94, 112], which is particularly interesting for efficient delivery of immersive videos.

2.2.1 Temporal compression

Since there are small changes and so redundant information between successive video frames, encoders use temporal redundancy to compress the video. An encoded video is composed of Groups of Pictures (GoPs), each containing Intra-predicted (I) frames, Inter-predicted (P) frames, and Bidirectional-predicted (B) frames. The I frame can be decoded independently of any other frame. In contrast, the P frame depend on data from previous frames - I or P frames - to be decoded and, the B-frame can use data from both preceding and future frames [96]. A GoP can be represented as a binary tree, as in Figure 2.1. Its display order is $I_0B_1B_2B_3P_4B_5B_6B_7P_8B_9B_{10}B_{11}P_{12}B_{13}B_{14}B_{15}$ while its decoding order is $I_0P_4B_2B_1B_3P_8B_6B_5B_7P_{12}B_{10}B_9B_{11}B_{14}B_{13}B_{15}$. When a video is delivered over a network to a given client, each video frame has a display time, as well as a decoding deadline, which is the minimum display time of the video frames depending on it.

A missing video frame induces spatial and temporal video quality distortion. The distortion depends not only on the missing frame (especially its type) but also on its position in the GoP and the implementation of the decoder. When a video frame is missing, a player may replace it with the last displayed one or with a black screen. The decoding process of video stream presenting a

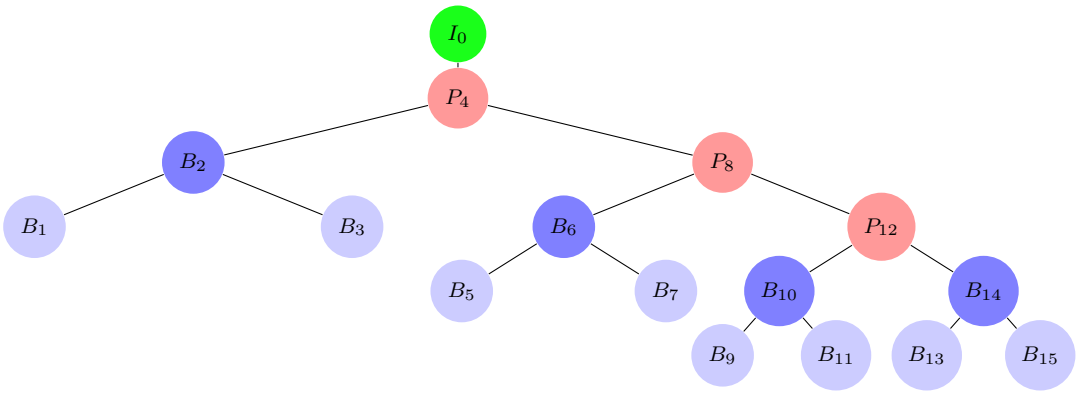


Figure 2.1: Structure representation of video frames inside a displayed GoP in the decoding order with a binary tree. B_2 is a children of P_4 means that B_2 cannot be decoded before the decoding of P_4

missing video frame depends on the player implementation. This process may result on a presence of artifacts on the displayed video since a part of the frame information is lost as we describe in Figure 2.2. When the missing video frame is an I frame, the player drops all the GoP.

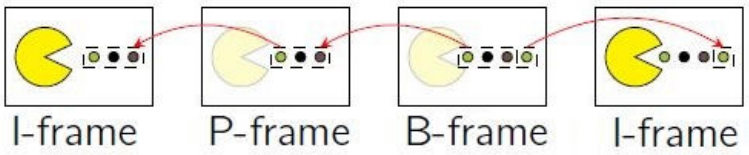


Figure 2.2: Illustration of coding dependencies between video frames

Modeling the distortion function is complex as it is not an additive function of the frames. The distortion caused by a set of missing frames close to each other is greater than the sum of the distortion induced by the isolated loss of the same frames [14]. Previous studies investigated simple ways to assign to each video frame several indicators of their relative importance on the display quality. Corbillon et al. [22] proposed to model the importance of video frames as a multi-criteria linear function taking into account three indicators: frame type, dependent frames, and frame size. We consider in Part I such a function as an input of our optimal scheduler to decide which video frames have to be discarded upon adverse network conditions.

Previous studies have used the Peak Signal to Noise Ratio (PSNR) and the Multi-Scale Structural Similarity for IMage (MS-SSIM) [21, 27, 33] to measure the spatial quality distortion due

to missing video frames. Assessing the temporal distortion (discontinuity) (known as video *jitters*) [41, 51, 79] is more challenging since it depends on the video content motion, the encoding structure and the frame rate. Previous studies showed that, for a large number of missing frames, the output video can be similar to a series of snapshots instead of a continuous stream.

Kester et al. [51] showed that a jitter of two consecutive frames is enough to be perceived by humans, while Huynh-thu and Ghanbari [41] observed that $20\,\%$ of users do not perceive a jitter of three missing consecutive video frames. The QoE degradation is also a function of the video bitrate and the video content type. Kester et al. [51] estimate that the Mean Opinion Score (MOS) decreases from 5 to 4 (respectively from 4 to 3 and from 3 to 2.5) for a jitter duration up to $150\,\mathrm{ms}$ (respectively $600\,\mathrm{ms}$ and $1400\,\mathrm{ms}$).

We refer to these studies in Part I to evaluate our results in terms of PSNR, MS-SSIM as spatial perceptual metrics and in terms of frequency and duration of jitters as temporal perceptual metrics.

2.3 Application layer protocols

Low latency streaming on the Internet primarily used muticast and Real Time Streaming Protocol (RTSP) with a limited number of available video qualities (one Standard Definition (SD) and one High Definition (HD)). Then, in the last decade, HAS has become the main technology to stream live and Video On Demand (VOD) contents on Internet. Actually, HTTP-based streaming allows to use CDNs to improve the client-server communications. Furthermore, since the streaming technology is built on top of HTTP, the packets easily traverse potential obstacles such as firewalls and Network Address Translation (NAT) devices. Besides, the client can maximize the video quality in accordance with the available bandwidth by choosing the adequate representation.

2.3.1 The limitations of HTTP/1

Despite the popularity of HTTP, the HTTP/1 version suffers from inefficiencies. An HTTP/1.0 client, should wait for the response of its request before being able to send the next one, which leads to an accumulation of Round Trip Times (RTTs) and increases the end to end delay as shown

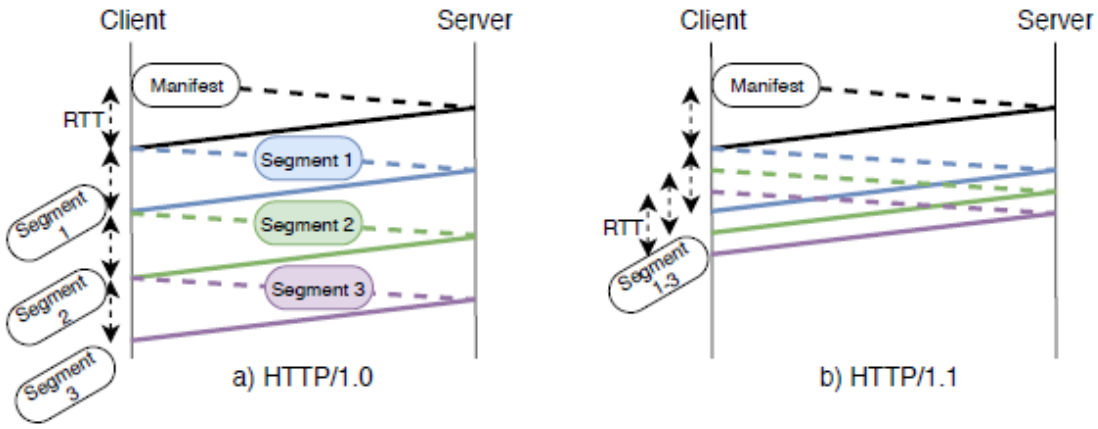


Figure 2.3: An illustration of the HTTP/1.0 and the HTTP/1.1 request/response mechanisms between a client and a server

in Figure 2.3. HTTP/1.1 has been designed in 1990. It enables the pipelining feature; the client can send a new request before receiving the response of the previous one which reduces the end to end latency. However, even with HTTP/1.1 pipelining, the server should response to a request n before processing the request $n + 1$. As illustrated in Figure 2.4, if a response of a request is blocked, the responses for all the next requests are also delayed/blocked, which is known as the HTTP head of line blocking problem [20].

HTTP/1 does not enable the client to prioritize its requests neither to stop an ongoing response without setting a new Transmission Control Protocol (TCP)/HTTP session. Besides, the HTTP/1 server only answers to the user requests and is not able to push appropriate content in advance. The new version of HTTP aims to address these limitations.

2.3.2 HTTP/2 protocol

The specifications of HTTP/2 were published in May 2015 [9]. HTTP/2 breaks down the HTTP protocol communication into an exchange of binary-encoded frames. These HTTP/2 frames convey messages that belong to a particular stream and all streams are multiplexed within a single TCP connection as represented in Figure 2.5.

It is also possible to set priorities among the streams. Priority can be controlled by setting

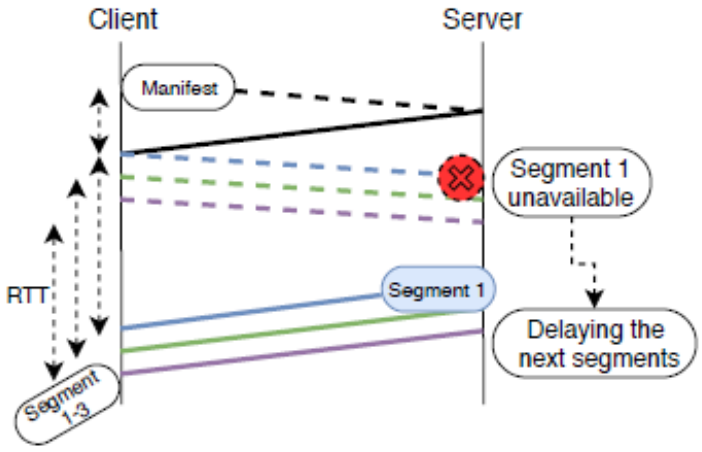


Figure 2.4: An illustration of the HTTP head of line blocking problem with HTTP/1.1

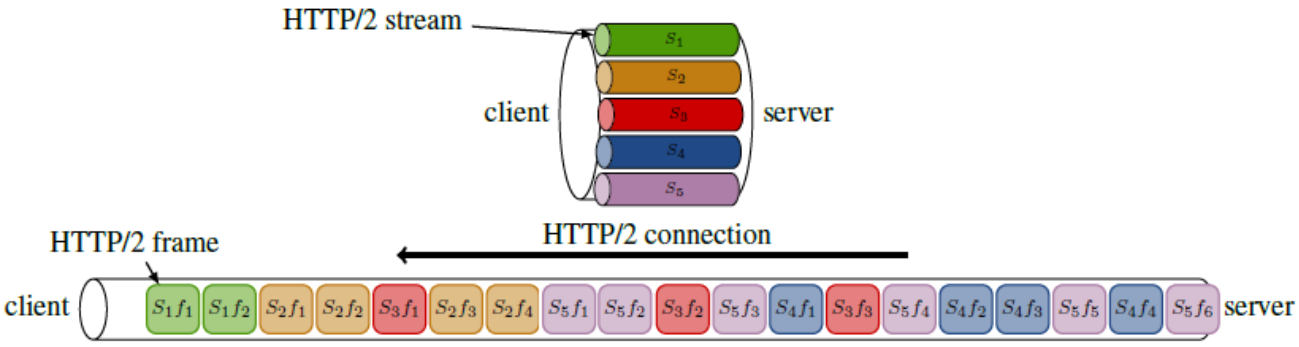


Figure 2.5: An illustration of an HTTP/2 connection with the priority feature activated

explicit dependency and weights to streams. We refer to stream S_i as a parent of stream S_j and to stream S_j as a sibling of stream S_i if the delivery of stream S_j depends on the delivery of stream S_i. The HTTP/2 weights set the relative proportion of resources to allocate to sibling streams sharing the same parent. Each stream has a weight; the higher the weight is, the more network resources are allocated to that stream. Moreover the client can update dependencies and weights at any time, which enables further optimization by reallocating resources in response to user interaction. Besides, the client can terminate an HTTP/2 stream without closing the underlying TCP session.

2.4 Reality check of the implementation of HTTP/2 features

The specifications of the HTTP/2 protocol in the Request for Comments (RFC) document let a part of freedom to the developers in the implementation of the features. Moreover, some of these stream management features are not considered as core features, and thus may not be developed in priority in practice.

To conform our proposal to real HTTP/2 server and client implementations, we developed an HTTP/2 server/client platform and checked the implementation of the main features that we leverage in our proposal. We installed the *H2O* server library,[1] which is presented as compatible with HTTP/2. On the client side, we developed a native C library using the *libcurl* library,[2] which is also supposed to respect the RFC specifications for the HTTP/2 protocol. At the date of Fall 2017, we analyzed different HTTP/2 specific requests between the client and the server and we observe the following:

Cancellation H2O server respects the RFC specification of the reset stream feature. Once a client asks to end a stream, the server cancels its transmission.

Dependency H2O server respects the RFC specification of the dependency feature. A client can set dependencies between HTTP/2 streams. The server sends the streams according to the client recommendations. Exclusive and default dependencies [10] are supported by H2O

[1] H2O — an optimized HTTP server with support for HTTP/1.x and HTTP/2: `https://github.com/h2o/h2o`

[2] libcurl — the multiprotocol file transfer library: `https://curl.haxx.se/libcurl/`

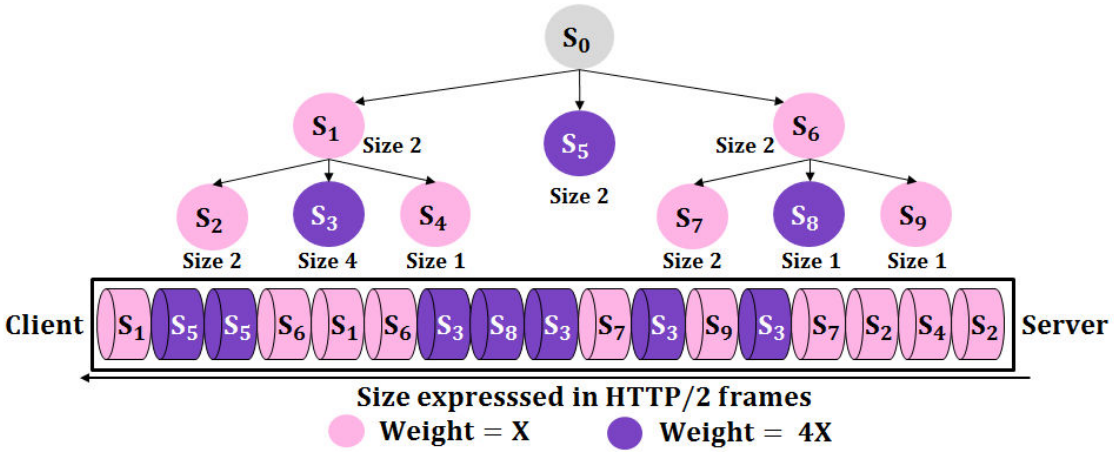


Figure 2.6: HTTP/2 tree-priority implementation. Each HTTP/2 stream $S_{i>0}$ has a parent, siblings, a weight and delivers a specific HTTP/2 frame number.

server and libcurl client. When a content of a stream is entirely sent, the dependent streams share the liberated allocated resources according to their weights. H2O server supports the dependency change. A specific dependency configuration enable the sending of the HTTP/2 streams content in a First In First Out (FIFO) order. For example, we suppose that we have three streams where S_2 depends on S_1 and S_3 depends on S_2. The server entirely delivers the content of S_1 then the content of S_2, and finally the content of S_3 to avoid the interleaving between the different HTTP/2 streams. H2O server chooses the smallest number of HTTP/2 frames to satisfy the resource sharing between HTTP/2 streams with respect to their weights and dependencies when the interleaving is active.

Weight If two HTTP/2 streams share the same parent, H2O server sets the weight of 16 to all HTTP/2 streams and the interleaving is active. If the client attributes a weight to a particular stream, H2O server affects the weight of 16 to all others streams. It is not possible to activate the interleaving for only a set of the HTTP/2 active streams. If the weight is set by the client, all the active streams share the available network resources with respect to the client recommended weights. H2O server supports weight change.

Dependencies and weights allow the client to provide the server with a "prioritization strategy", ensuring optimal delivery of high-priority responses, as shown in Figures 2.6 to 2.8. Wijnants

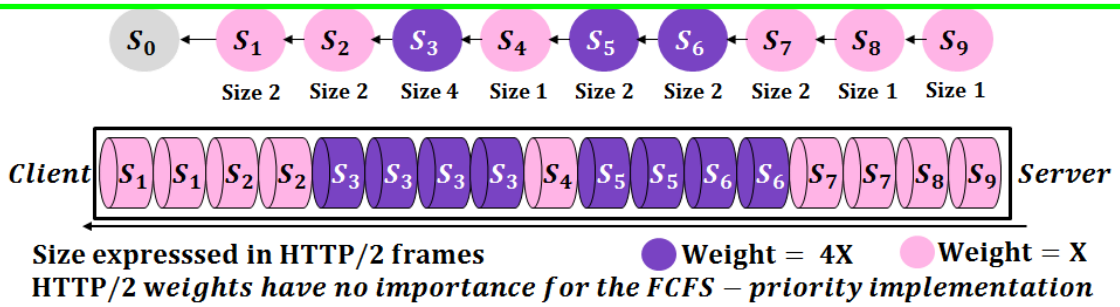


Figure 2.7: HTTP/2 First Come First Served (FCFS)-priority implementation. HTTP/2 weights have no impact on the delivery since streams $S_{i>0}$ have no siblings but only a parent.

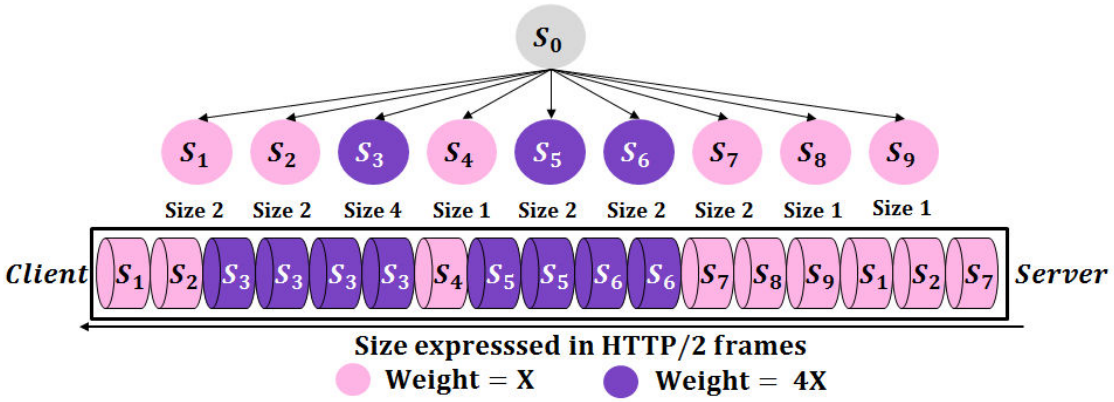


Figure 2.8: HTTP/2 Round Robin (RR)-priority implementation. Each HTTP/2 stream $S_{i>0}$ has only siblings. It is assigned a weight and it delivers a specific HTTP/2 frame number.

et al. [106] describe three current implementations of the HTTP/2 priority features.

Firefox uses the tree priority as shown in Figure 2.6. The sibling parent streams are delivered first, then the resources are allocated to their children in accordance with their weights.

Chrome and Opera use a FCFS priority implementation as shown in Figure 2.7. Each HTTP/2 stream depends on the previously requested one and has no siblings. FCFS is the standard per-TCP-connection behavior of HTTP/1. Yet, with HTTP/2 it is possible to cancel a specific ongoing stream without canceling the whole TCP connection.

Safari and Internet Explorer implement non-exclusive dependencies on the default stream S_0 for all HTTP/2 streams as shown in Figure 2.8. Streams have no children but only siblings, except stream S_0, which is the parent of all the other streams and a default stream dedicated to the HTTP/2 session. All streams concurrently share the available resources in proportion to their

weights. When a uniform default weight of 16 is applied to all streams, this RR model is the default prioritization logic of HTTP/2, and it ensures a fair distribution of the network resources consumption among HTTP/2 streams. The order in which the HTTP/2 frames are interleaved becomes a critical performance consideration.

In this thesis, we explore the advantages of using these dependency, weight and reset stream features in a context of low latency streaming. We focus on live streaming and immersive VOD service.

Part I

HTTP/2-based frame discarding strategies for low latency streaming

“You can’t win unless you learn how to lose."

- Kareem Abdul Jabbar -

Introduction

The DASH adaptation algorithms for low latency streaming can be insufficient to mitigate the varying throughput of mobile networks (refer to Section 2.1.2). Alternative solutions are needed to avoid rebuffering and maintain continuous video display until the end of the throughput deficit event.

Video frame discarding is a solution to lighten the data size of a transmitted video stream. It consists in voluntary dropping video frames on the server, network or player sides in order to avoid the accumulation of delays. At the client side, the video player does not pause the video; it deals with the missing video frames, typically by repeating the last displayed one [41].

In this part, we briefly describe the related works about video frame discarding strategies. Then, we detail an HTTP/2-based frame discarding scheme that we propose which enables the client to avoid rebuffering events.

Related works on video frame discarding

The implementation of video frame discarding strategies has been the focus of several studies in the past. Thang et al. [97] emphasizes the importance of modeling solutions that offer a good trade-off between maximizing the video quality representation, minimizing the buffer size and avoiding the rebuffering events. These solutions include video frame discarding. Kalampogia and Koutsakis [48] propose a Markovian model to predict the size of B-frames and to selectively discard these frames. Mehdian and Liang [67] propose a recursive optimal model to decide which video frames to discard for minimizing the video quality distortion. They do not propose a practical implementation of their scheme. Darabkh et al. [28] studies a policy that controls and maintains the buffer occupancy of network elements. This policy relies on User Datagram Protocol (UDP) and uses the sixth and seventh bits of the IPv4 type-of-service header field to represent the I, P,

B-frames. They reduce the number of B and P-frames to ensure the successful reception of the I-frames. Gangadharan et al. [33] study a trade-off between frame discarding, buffer size and video quality. They design a droppable part at the receiver buffer side where the video frame discarding may occur. This approach does not free bandwidth resources to enable a faster delivery of the next video frames but it ensures a faster decoding process. Darabkh et al. [27] present network and buffer based-models for frame discarding policies. Corbillon et al. [21] and Houze et al. [38] focus on the video frames prioritization delivery in the context of multi-path deliveries. All of these studies [15, 21, 22, 26, 27, 38] have shown that the video frame discarding approach can temporarily reduce the bitrate while maintaining a high QoE.

However, these proposal solutions need a cross layer design between the application and the network layers, which can be hard to implement. Besides, HTTP/1 does not suit the implementation of client-based frame discarding solutions since it does not allow to easily cancel a part of the video stream that has been already requested without establishing a new TCP session.

In this thesis, we suggest to use the HTTP/2 protocol to design a video frame discarding scheme on the application layer [11]. We discuss in Chapter 3 the HTTP/2 features that are suitable for our proposal. Then, we present the video frame discarding model we propose as well as different optimal and practical algorithms. Besides, we evaluate the performances of these algorithms in Chapter 4, by running simulations based on real videos and network traces.

Chapter 3

HTTP/2-Based Solutions for Video Frame Scheduling

Contents

In this chapter, we propose several ways to control the video frames delivery by leveraging the HTTP/2 stream-control features. Our main idea consists on requesting each video frame in a

dedicated HTTP/2 stream. It becomes then possible to control the delivery of the video frames by invoking the stream-level control features defined in the HTTP/2 standard.

3.1 HTTP/2 features for video frame delivery

We describe in the following the advantages of requesting each video frame in an HTTP/2 stream.

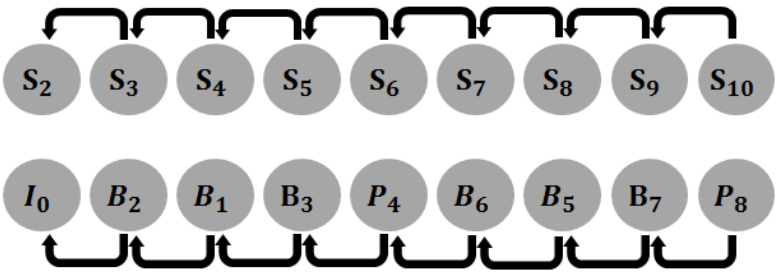
HTTP/2 HPACK compression Requesting separately each video frame results in HTTP headers that often carry the same values. HTTP/2 standard comes along with an algorithm named HPACK [80], which specifies how to compress the HTTP headers and reduce the HTTP overhead.

Cancel HTTP/2 stream The reset stream command enables the client to cancel the delivery of an HTTP/2 stream, which in our case means to cancel the delivery of a video frame. By using this command, the client saves network resources by avoiding the delivery of a video frame with no chance to be downloaded and displayed on time.

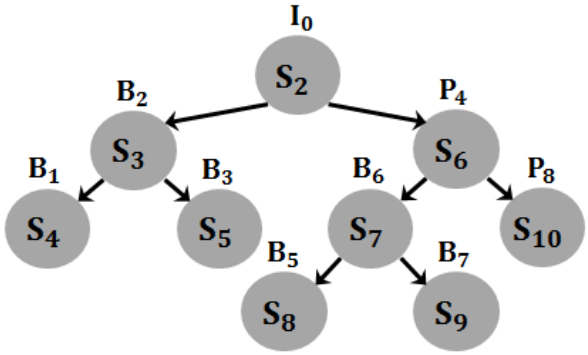
Set HTTP/2 stream priorities The commands controlling the priorities among the HTTP/2 streams enable the client to manage priorities among the transported video frames. These commands set dependencies and weights on the HTTP/2 streams. The server has no obligation to strictly respect the priority guidance of the client but usually the server takes it into consideration unless an object is not available, so the server operates the next requests as to avoid additional latencies due to the head of line blocking. The priority features are described as follow:

Set HTTP/2 stream dependencies The main idea consists in setting the HTTP/2 dependencies among the HTTP/2 streams in accordance with the decoding requirements of video frames. We consider two possible dependency strategies, named FIFO and Interleaving (see Figure 3.1).

The FIFO dependency strategy delivers the video frames in a FIFO mode in accordance with their decoding order. To apply this strategy, the dependency of each HTTP/2



(a) FIFO dependency strategy



(b) Interleaving dependency strategy

Figure 3.1: HTTP/2 streams along with their video frames and dependencies for different dependency strategies

stream delivering the nth encoded video frame is set so that it must depend on the HTTP/2 stream delivering the $(n-1)$th encoded video frame, as shown on Figure 3.1a. The *Interleaving* dependency strategy sets the dependencies on the HTTP/2 streams in accordance with the video frame dependency tree, as shown on Figure 3.1b. In this latter strategy, the interleaving is activated between two streams if they share the same parent, as presented in Figure 2.6. The client must therefore adjust the weights of the HTTP/2 streams carefully to control the order in which video frames are received.

Set HTTP/2 stream weights When the Interleaving dependency strategy is applied, the server automatically attributes a weight of 16 to all the streams, which results in delaying the delivery of video frames, as the weight attributed by the server does not consider the video frame deadline. The priority command enables the client to set and modify the weight of any HTTP/2 stream at any time. The weight of each HTTP/2

stream must be an integer between 1 (lowest priority) and 256 (highest). In case of bandwidth shortage, the client may decide to increase the weight of the most important video frames so that the HTTP/2 multiplexing algorithm increases the fraction of the traffic carrying data of these video frames in an attempt to increase the probability that these important video frames arrive at the decoder on time.

3.2 HTTP/2-based video frame scheduling policies

When the DASH adaptation algorithm miss-estimates the available bandwidth, the delivery of the video segment and the display of the video session is jeopardized. Several triggers enable the client to detect these troubles, such as an abrupt reduction of the amount of data in the client buffer and a reduction of the delay between the reception time of the data composing the concerned video frames and their display times. A client detecting a trouble can decide to apply strategies to prioritize the delivery of a set of video frames by discarding or delaying another set of video frames.

In order to choose an appropriate implementation of the HTTP/2 priority, we analyze and compare two different scheduling policies: i) the **cancel-based** scheduling policy which relies on the FIFO dependency strategy and the HTTP/2 reset stream feature, and ii) the **weight-based** scheduling policy which relies on the Interleaving dependency strategy and the HTTP/2 weight feature.

As shown on Figures 3.2 and 3.3, first the client requests, gets and parses the MPD, via an HTTP/2 stream. When the client requests a video segment, it extracts from the parsed video meta-data the positions (*byte ranges*) of the video frames within each GoP, typically with the technique introduced by Houze et al. [38]. Then the client associates each video frame with an HTTP/2 stream and sets the dependencies of the HTTP/2 streams in accordance with any of the two above-described dependency strategies (FIFO or Interleaving). Then the client starts to request and fetch video frames over their associated HTTP/2 streams: video frame I_0 on HTTP/2 stream S_2, video frame B_2 on HTTP/2 stream S_3, and so on.

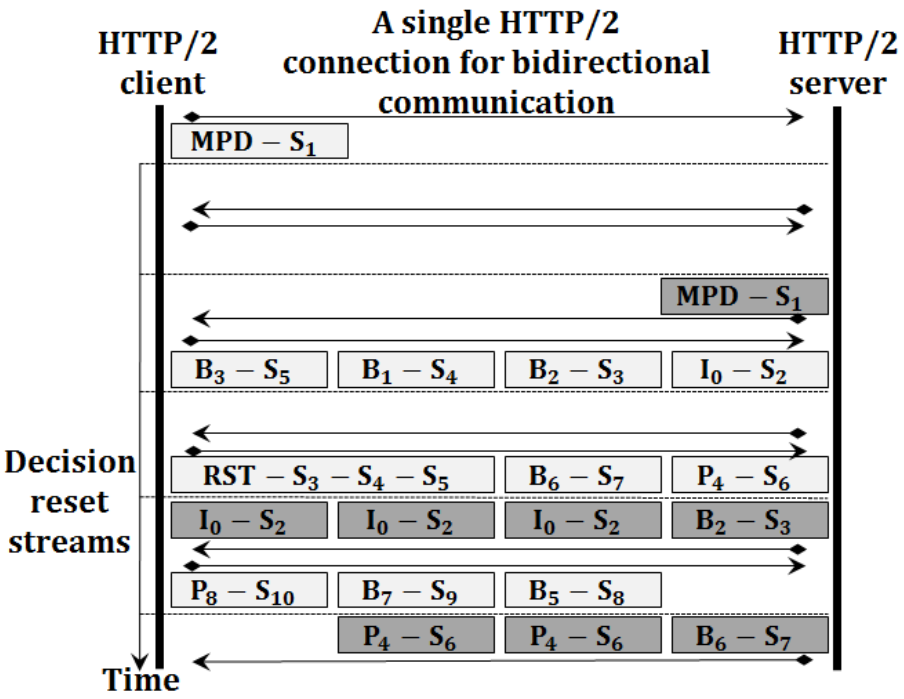


Figure 3.2: Cancel-based scheduling policy

3.2.1 Cancel-based policies

The cancel-based policy relies on the FIFO dependency strategy. The client receives the video frames sequentially in the decoding order. Upon adverse network condition, it takes the decision to completely discard a set of the less important video frames. This scheduling strategy is similar to the traditional video frame discarding policies as the client only asks the server to completely drop a set of video frames.

For example, as described in Figure 3.2, the client decides to cancel the HTTP/2 stream S_3 which transports the video frame B_2. Since video frames B_1 and B_3 depend on B_2 (see Figure 2.1), canceling the delivery of the video frame B_2 jeopardizes the successful decoding of both video frames B_1 and B_3. So the client also asks the server to reset the HTTP/2 streams S_4 and S_5, which carry respectively the video frames B_1 and B_3. The server reallocates the network resources of the canceled HTTP/2 streams S_4 and S_5 to the HTTP/2 stream S_6, which enables an earlier delivery of the video frame P_4 and increases the probability of P_4, as well as the following video frames, to be decoded on time at the client side.

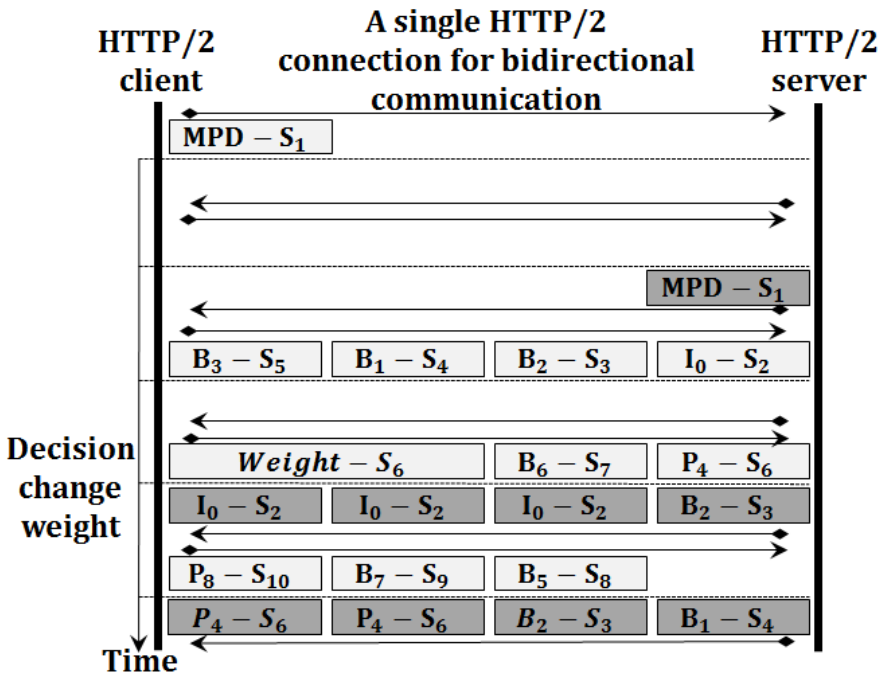


Figure 3.3: Weight-based scheduling policy

3.2.2 Weight-based policies

In a varying network, the bandwidth may decrease and then increase suddenly, which means that the client may drop a set of video frames that could have been received successfully without jeopardizing the delivery of the most important ones. The cancel-based policy does not enable the client to temporarily reduce the network resources allocated to a video frame delivery until the full reception of the more important ones or a bandwidth increase.

Weight-based policy enables a simultaneous delivery of a set of video frames as it relies on the Interleaving dependency strategy. The client sets the dependencies between the HTTP/2 streams with regards to the video frame dependency tree represented in Figure 2.1 and manage the resources distribution among the simultaneous delivery of video frames by changing the HTTP/2 weights. The client sets and updates the weight of each HTTP/2 stream during the HTTP/2 session.

In Figure 3.3, we suppose that adverse network conditions have delayed the delivery of the video frame B_2. The video frame B_2 will *probably* arrive after its display deadline, which also

jeopardizes the timely delivery and display of its dependent video frames B_1 and B_3. Moreover the useless delivery of the video frame B_2 delays the delivery of the more important video frames P_4 and P_8. In this case, updating the HTTP/2 stream weights — decreasing the weight of stream S_3 and increasing the weight of stream S_6 — allows a faster delivery of video frames P_4 and P_8 and a slower delivery of the less important video frames B_2, B_1 and B_3 without ending their transmissions. If the network throughput increases again, this will trigger another update of the weights, causing a faster delivery of all the video frames B_1, B_2, B_3, P_4 and P_8.

3.2.3 Challenges of the weight-based policies

Weight-based policies are in practice highly sensitive to the weights and hard to control. If the client does not attribute a weight to a particular stream, the server sets the weight of 16 to all others stream by default. However, the weight should depend on an importance function that the client computes for each video frame. The importance function should take into account factors such as the dependency, the total size, the received size of the video frame and the delay until its display deadline. Besides, current HTTP/2 implementations do not allow to activate the interleaving for a specific set of HTTP/2 streams without updating the dependency tree. The client cannot attribute a weight of 0 to an HTTP/2 stream: the weight must be an integer between 1 and 256.

In the above-described illustration (Figure 3.3), even in good network conditions, the delivery of video frames B_1, B_2, B_3 gets delayed by the delivery of portions of video frames P_4 and P_8, which increases the risk of dropping video frames B_1, B_2, B_3 while the network is able to successfully deliver all of video frames in a FIFO order.

The weight scheduling policies are complex to implement and manage. Wijnants et al. [106] show that weight-based policies may lead to an excessive sharing of the network resources among HTTP/2 streams and cause delays in the completion of dependent tasks. Therefore, we only focus on the cancel-based policies, based on the well-implemented reset stream feature in the remaining of this dissertation.

Input	Meaning
H	Set of HTTP/2 frames successfully received by the client
V	Set of video frames
P^v	Set of video frames on which the video frame v depends
s^v	Number of HTTP/2 frames needed to transmit video frame v
$t^v_{display}$	Display time of video frame v
g^v	Relative importance of video frame v on the entire video quality
y^v	State of the video frame v, 1 displayed, 0 not displayed
h_{size}	HTTP/2 frame size
v_{size}	Video frame v size
t^h_r	Arrival time of HTTP/2 frame h at the client side

Decision	Meaning
x^v_h	equal to 1 if HTTP/2 frame h contains data relative to video frame v and 0 otherwise
y^v	equal to 1 if video frame v is displayed and 0 otherwise

Table 3.1: Notations for inputs and decision variables

3.3 Cancel-based policies: model and algorithms

Our proposal is entirely client-based. The client decides to drop a set of video frames to avoid the video rebuffering while impacting the less the video quality. To describe the different algorithms in the following, we use the notations listed in Table 3.1. Current video players discard a video frame if it is not displayed at time. They do not keep it in order to decode its dependent video frames. Hence, to be displayed at the client side, a video frame v must meet the following requirements:

(a) v must be entirely received before its display time $t^v_{display}$

(b) All of its parent video frames (*i.e.*, $v_p \in P^v$) must be entirely received and decoded before v display time $t^v_{display}$

(c) Each of its parent video frames (*i.e.*, $v_p \in P^v$), if any, must be displayed on their display time $t^{v_p}_{display}$.

In this section, we introduce the following main categories of our proposed algorithms:

1. The first category is **the optimal scheduler**, modeled with an Integer Linear Programming (ILP). The optimal scheduler provides an upper bound on the achievable displayed video quality. It determines the optimal set of video frames to discard. It takes as inputs the records of a complete HTTP/2 traffic session, which includes the times at which the HTTP/2 frames are successfully received at the client side, and the meta-data of a selected video sequence. The meta-data includes the size, type, dependencies and display time of each of its video frames.

2. The second category is **the basic FIFO scheduler**, where the server deliver all the video frames one after the other in the decoding order. The sever does not discard any frame. At the client side, no rebuffering is allowed. When a video frame is not received on time, the last displayed video frame replaces it. A delayed video frame is discarded at the client buffer. Only, successfully delivered video frames are displayed.

3. The third category consists on **heuristics of the optimal scheduler**. The client does not have a prior knowledge on the network resources but it decides to discard a set of video frames depending on several parameters. We denote these heuristics as the **practical R-S-T algorithms**, which we detail in Section 3.3.2.

3.3.1 Optimal scheduler

The optimal scheduler determines the best set of video frames to discard depending on their importances as well as a perfect knowledge of the network resources. We model the optimal scheduler as an ILP as described in Algorithm 3.1.

Inputs

We denote by H the set of HTTP/2 frames that are successfully received by the client for the whole transmission. Each HTTP/2 frame $h \in H$ is characterized by its reception time at the client side, denoted by t_r^h, which takes into account the initial buffering time. We denote by V the set of video frames that have to be streamed from the server to the client. Each video frame $v \in V$ should be

displayed by the application at the client side at a given time, denoted by $t^v_{display}$. The transmission of a video frame v requires the transmission of s^v different HTTP/2 frames. We consider packet padding if the size of the video frame is not a multiple of the maximum HTTP/2 frame size denoted by $h_{size}, h \in H$. We assume that the size of the HTTP/2 frames is chosen such that the the padding is minimum. We denote by P^v the set of video frames, parents of a video frame v. We denote by g^v the mathematical relative importance of the video frame v, computed with the method presented by Corbillon et al. [22].

Optimization objectives

We aim to maximize the importance of the successfully displayed video frames in order to maximize the video quality, formally:

$$\max \sum_{v \in V} g^v y^v$$

Decision variables

The main decision variable of our model is x^v_h, which indicates whether HTTP/2 frame h transports data for video frame v. The decision variable $y^v, v \in V$, is equal to 1 if the video frame v is displayed on-time and is equal to 0 if the video frame v is discarded.

Integer Poblem Modeling

The formulation of the ILP is described in Algorithm 3.1.

It can be interpreted as follows. Each HTTP/2 frame transports data related to a given video frame. A video frame is received if it is transported by s^v HTTP/2 frames. A video frame v is successfully received, decoded and displayed if (i) s_v HTTP/2 frames carrying data related to v are sent; (ii) The latest of these frames arrives before $t^v_{display}$; And (iii) all video frames in P^v are displayed on-time.

Equation (3.1) means that every successfully sent and displayed video frame v must use exactly s^v HTTP/2 frames. Equation (3.2) means that every HTTP/2 frame h delivers data related to a

Algorithm 3.1 Optimal scheduler Modeling

$$\max \quad \sum_{v \in V} g^v y^v$$

$$\text{subject to} \quad \sum_{h \in H} x_h^v = s^v y^v \qquad \forall v \in V \qquad (3.1)$$

$$\sum_{v \in V} x_h^v \leq 1 \qquad \forall h \in H \qquad (3.2)$$

$$y^v \leq y^{v_p}, \ \forall v \in V \qquad \forall v_p \in P^v \qquad (3.3)$$

$$\text{if } t^v_{display} \leq t^{v_p}_{display}$$

$$\text{then } t^h_r x^{v_p}_h \leq t^{v_p}_{display}(1 - y^v) + y^v t^v_{display} \qquad (3.4)$$

$$\forall v \in V, \ \forall v_p \in P^v \qquad \forall h \in H$$

$$\text{if } t^h_r \geq t^v_{display}$$

$$\text{then } x^v_h = 0 \qquad \forall v \in V, \ \forall h \in H \qquad (3.5)$$

unique video frame v. Equation (3.3) means that all the video frames on which v depends must be also successfully sent and displayed on their display times. Equation (3.4) means that all the parent video frames $v_p \in P^v$ relative to a video frame $v \in V$ must be received before the display time of v if v is successful. Equation (3.5) means that data of v is sent in an HTTP/2 frame h only if h is received before the video frame v display time.

3.3.2 Practical algorithms

We present now practical algorithms to discard video frames that can hardly be decoded on time. The re-allocation of the freed network resources gives more chance to the other video frames to meet their decoding deadlines. We design several variants of these algorithms by considering the following two parameters.

- **The minimum temporal margin for a video frame to be received TM_{min}:** When the client estimates that a video frame will be received late, it discards it as well as all its children. A video frame is considered as late when the margin between its display time and the time the client takes the decision is less than a minimum bound denoted by TM_{min}. We consider two types of algorithms: the *Reset-Reactive* algorithms ($R - R$), which sets TM_{min} to zero,

and the *Reset-Predictive* algorithms ($R-P$), which requires an estimation of the network throughput to compare with positive values of TM_{min}. Various approaches are possible to predict the network bandwidth (for instance the average throughput for the latest TCP packets).

- **The decision time interval $T^{decision}$:** In theory, the client can apply discarding decisions after the reception of each HTTP/2 frame. However, it generates more processing, which may burden the client. In our algorithm, we let the client check the status of frame delivery and take discarding decisions in a periodic manner, every $T^{decision}$. In the evaluation, we study the impact of varying this time interval $T^{decision}$ from $0\,\mathrm{ms}$ to $200\,\mathrm{ms}$.

For a generic description of our practical algorithms in the following parts, we denote them as $R-S-T$ algorithms, where $S \in \{R, P\}$ and $T = T^{decision}$. We denote by $R-R$ the $R-R$ algorithm where the decision time is taken at the reception of each HTTP/2 frame. We summarize in Table 3.2 the different parameters described above.

Parameters	**Meaning**	**Possible values for different $R-S-T$ algorithms**		
		R-R	**R-R-T**	**R-P-T**
$T^{decision}$	period between consecutive discarding decisions	at the reception of every HTTP/2 frame	a fixed T (chosen between $40\,\mathrm{ms}$ and $200\,\mathrm{ms}$)	a fixed T (chosen between $40\,\mathrm{ms}$ and $200\,\mathrm{ms}$)
TM_{min}	A video frame v is late when: $T^{decision} - t^{v}_{display} < TM_{min}$	$0\,\mathrm{s}$	$0\,\mathrm{s}$	estimation of the time needed to fetch video frame v

Table 3.2: Parameters $T^{decision}$ and TM_{min} for $R-S-T$ algorithms

We evaluate the performance of our proposed frame discarding algorithms in Chapter 4 using intensive simulations that combine real video and network traces.

Chapter 4

Evaluation of Video Frame Discarding Algorithms

Contents

We developed two simulators to evaluate the performance of the proposed scheduler; a simulator based on the Java GNU Linear Programming Kit (GLPK) solver[1] for the optimal algorithm 3.3.1 and a Java simulator for the heuristic algorithms listed in Table 3.2 to combine network traces and video datasets.

[1] GLPK — GNU Linear Programming Kit: https://www.gnu.org/software/glpk/

4.1 Datasets

We describe in this section the network dataset and video dataset prepared for the simulations, which results are reported in Section 4.2. We collected cellular and WiFi network traces using our own platform. We used six representative videos (publicly available at [109]) to test our proposal. Our simulations considered 6480 delivery scenarios using 12 network traces classified in 2 types, 6 videos encoded in 12 different bitrates (6 for each network type), and 15 algorithms. Then, we used a publicly available code [21] to measure performance and evaluate the video quality of the obtained video segments.

4.1.1 Network datasets

We collected WiFi and cellular network traces, which we have made publicly available on our website [2] to enable further researches. We used in Section 2.1.2 some network traces [103]. However, these traces do not suit for our simulations. Indeed these traces give information about the network bandwidth only. In contrast, our simulations need the sequence number, the reception timestamp, and the size of each TCP packet in order to deduce the order, the reception timestamp and the size of each HTTP/2 frame. Besides we aim to assess the performance of our video frame discarding algorithms on two types of wireless networks: WiFi and cellular networks.

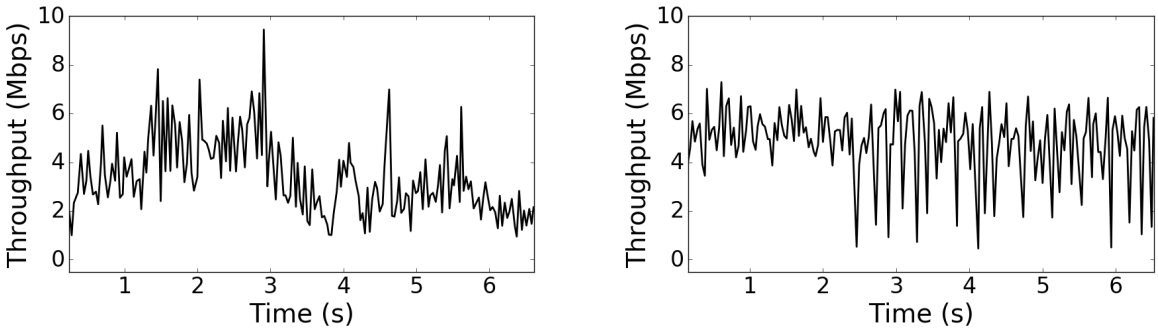
To collect the real network traces of TCP traffic, we set up a platform composed of a CDN edge server from a French telecom operator and a client running on an Ubuntu 16.04 machine. We achieved two series of 65 s long network captures. Then we extracted network captures lasting 6 s (which is the HAS segment duration of the videos considered in our simulation). In the first series the client is connected to a 4G cellular network, and in the second series to the WiFi interface of an xDSL residential gateway. The captures are done with the Wireshark tool while the client downloads a file from the CDN server.

Throughput variations are significant in mobile networks [87], where various factors (*e.g.* the fading, the number of the simultaneous connected users, the density of the deployed cellular cells

[2] WiFi and cellular network traces — `http://dash.ipv6.enstb.fr/acmtomm2018/`

in an area and the user speed) may result in abrupt changes in network conditions. WiFi networks present a steadier throughput than cellular networks. However, WiFi networks may suffer from instantaneous throughput drops that are difficult to detect when only the average throughput over the duration of segments is analyzed. The cellular throughput varies from $0.5\,\mathrm{Mbps}$ to $7\,\mathrm{Mbps}$. The WiFi throughput varies from $3\,\mathrm{Mbps}$ to $6\,\mathrm{Mbps}$. Our proposal is meaningful when there is a deficit between the video representation selected by DASH client and the real available bandwidth at each video frame reception time.

We present in Figure 4.1 the average throughput variation of two different network traces sampled at the display time of every video frame (ie. $1/30\,\mathrm{s}$). In our simulation, we extracted 6 different traces from each network capture with average throughput values of $5\,\mathrm{Mbps}$ for WiFi networks and $3.3\,\mathrm{Mbps}$ for cellular networks.



(a) for cellular network

(b) for WiFi network

Figure 4.1: Average throughput sampled every $30\,\mathrm{ms}$ during $6\,\mathrm{s}$.

4.1.2 Video datasets

Our video dataset comes from a public repository [109]. The six selected video sequences are listed in Table 4.1; they range from small moving regions of interest on static background to fast moving sport clips. We achieved a first set of preparatory operations on these video sequences to generate a set of $6\,\mathrm{s}$ long video segments and to classify them. Using H264, we encoded the video clips at a fixed frame-rate of $30\,\mathrm{fps}$. Then, we extracted from each video clip a $6\,\mathrm{s}$ long video segment, which corresponds to the considered HAS segment duration in our simulations. The applied time

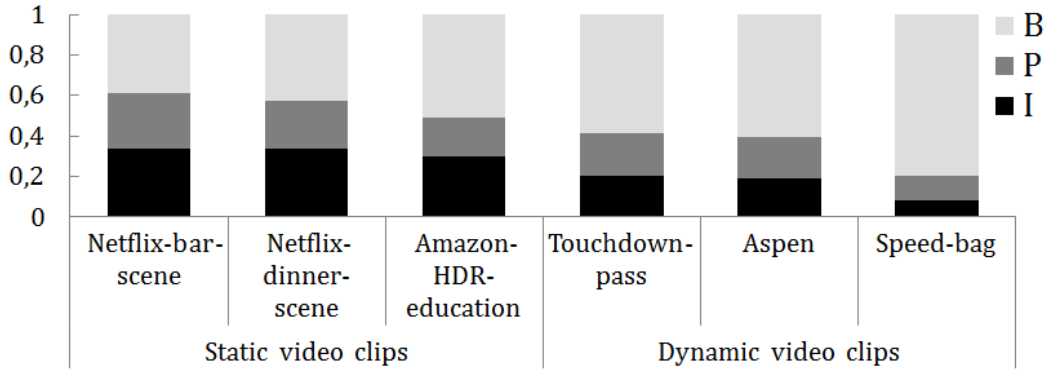


Figure 4.2: Ratio of the relative data size of frames I, P and B

offsets are also reported in Table 4.1. We encoded the extracted video segments at a resolution of 1920×1080 pixels and with a GoP structure MN, with $M = 7$ (distance between two successive I or P frames) and $N = 25$ (number of frames in a GoP).

Figure 4.2 represents the average ratio of data contained in each encoded bit stream dedicated to frames I, P and B for video segments encoded at $3\,\mathrm{Mbps}$. Encoded bit streams dedicated to frames B in dynamic video segments are larger than in static video segments. We encoded the video segments at different bitrates to generate several deficits (up to $1\,\mathrm{Mbps}$). The chosen video bitrates create a relative deficit amplitude varying from $0\,\%$ to $27\,\%$ for the cellular networks and from $0\,\%$ to $16\,\%$ for the WiFi networks. For cellular networks, we encoded the 6 videos at bitrates ranging between $3.2\,\mathrm{Mbps}$ and $4.2\,\mathrm{Mbps}$. For the WiFi networks, the bitrate ranges from $4.8\,\mathrm{Mbps}$ to $5.8\,\mathrm{Mbps}$. Our goal is to evaluate the behavior of our proposed algorithms for different deficits for low-latency scenarios. In order to respect the $1\,\mathrm{s}$ targeted latency, we set the initial buffering to $1\,\mathrm{s}$.

Name	Offset	Category
Netflix-bar-scene	$0\,\mathrm{s}$	Static
Netflix-dinner-scene	$9\,\mathrm{s}$	Static
Amazon-HDR-education	$171\,\mathrm{s}$	Static
Touchdown-pass	$6\,\mathrm{s}$	Dynamic
Aspen	$3\,\mathrm{s}$	Dynamic
Speed-bag	$9\,\mathrm{s}$	Dynamic

Table 4.1: Parameters summary

4.2 Performance evaluation

We first evaluate the HTTP/2 overhead generated by requesting every video frame in an HTTP/2 stream. Then, we evaluate the performance of our proposed algorithms using three different type of metrics.

Video Flow Importance Ratio The frame *importance* (introduced by Corbillon et al. [22]) is computed from the features of each frame (size, type, and dependencies). We used the ratio of the sum of the importance of successfully displayed frames over the total importance of all frames. This metric is based on the received and original encoded videos; it is thus independent of the decoder strategy for the missing frame replacement.

Spatial Quality Degradation We computed the PSNR and MS-SSIM metrics to get the spatial quality of the videos. We rebuilt the encoded received streams to compute these metrics. To replace missing frames, the decoder displays the last decoded frame.

Jitter and Temporal Degradation We evaluate the temporal distortion by analyzing the number, type and duration of the jitters in the received encoded video stream.

4.2.1 HTTP/2 multi-stream overhead

We estimate the protocol overhead resulting from requesting and delivering a $6\,\mathrm{s}$-long video segment. We consider two cases: (i) the mono-stream case: the video segment is requested and delivered over a single HTTP/2 stream; (ii) the multi-stream case: each video frame of the video segment is requested and delivered over a dedicated HTTP/2 stream. From the reference software described in Section 2.4, we extracted the size of the HTTP/2 protocol headers. The size of the payload downstream (the video segment) is $18\,\mathrm{Mbits}$. The estimations of the downlink and uplink protocol overheads (expressed as the ratio of the HTTP/2 overhead to the total amount of delivered data) are reported in Table 4.2.

The overhead in the HTTP/2 mono-stream case is consistent with the state-of-the-art [70, 99]. Our video frame discarding algorithms implement the HTTP/2 multi-stream case. This strategy

	Download				Upload			
	multi-stream		mono-stream		multi-stream		mono-stream	
	Size	**Number**	**Size**	**Number**	**Size**	**Number**	**Size**	**Number**
Uncompressed header	114	1	114	1	31	1	31	1
Compressed headers	15	179	0	0	16	179	0	0
Data Frame headers	8	180	8	138	5	10	0	0
Settings	2	3	6	3	18	3	18	3
Windows_update	4	2	4	2	4	32	4	100
Total overhead	5.76 %		4.34 %		2.34 %		0.95 %	

Table 4.2: HTTP/2 protocol overheads for video streaming

slightly increases the total overhead on the downlink; however, it always remains below $6\,\%$ thanks to the HTTP/2 HPACK compression mechanism and the variability of the HTTP/2 frame size. The overhead increase is more significant on the uplink, but the client is barely constrained on the uplink bandwidth. We conclude that the overhead of our multi-stream strategy offers more control on the delivery at the expense of a moderate and affordable overhead.

4.2.2 Video flow importance

We define and simulate a set of video delivery scenarios to assess the performance of our video frame discarding strategy with the Video Flow Importance Ratio metric. Each video delivery scenario is characterized by a triplet: a video frame discarding algorithm, a network trace and a video segment. The results of each simulation are the list of the video frames successfully displayed on the client and the computed value of the video flow importance ratio metric.

The simulation results are reported in Figure 4.3, Figure 4.4, and Figure 4.5. On each box-plot the colored surface represents the Interquartile Range (IQR): it encompasses all the values of the video flow importance ratio metric that ranges from the first quartile to the third quartile. Inside this surface, the horizontal line and the point represent, respectively, the median and the average values of the video flow importance ratio metric. The whiskers extending from either side of the box represent the ranges of the bottom $15\,\%$ and top $15\,\%$ values of the video flow importance ratio metric.

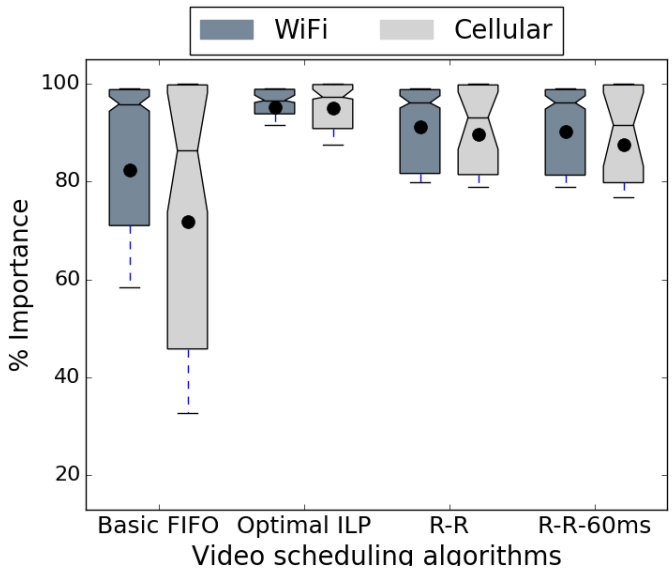


Figure 4.3: Flow importance ratio of the basic FIFO, optimal, R-R, and R-R-60 ms algorithms for cellular and WiFi networks

Figure 4.3 presents the results of the simulations on each network (WiFi and cellular) for the following video scheduling algorithms: Basic FIFO, optimal, R-R, and R-R-60 ms. Each box-plot on Figure 4.3 corresponds to a network (WiFi or cellular) and a video scheduling algorithm; it presents the distribution of the video flow importance ratio obtained by simulating all the video delivery scenarios (*i.e.* the whole video dataset and all the network traces). The box-plots show a wide distribution on this metric, even if the median values are at the same level on the WiFi network, regardless of the algorithm. The median and average values are always higher on the WiFi network than on the cellular networks. The optimal algorithm provides an upper bound to evaluate the other algorithms. The performance of our video scheduling algorithms is much closer to the optimal than to the basic FIFO. Typically, the R−R and R−R−60 ms algorithms manage to reach at least 80 % of the video flow importance ratio in all scenarios, while the optimal algorithm reaches around 90 % on both networks, and the basic FIFO 60 % on the WiFi network and only 40 % on the cellular network. Moreover, the optimal, R-R and R-R-60 ms algorithms show more stable performance results than the basic FIFO, as their box-plots have smaller IQR. The basic FIFO does not react as well as our video scheduling algorithms to network changes.

We report in Figure 4.4 the results per category of video segments (static vs. dynamic video

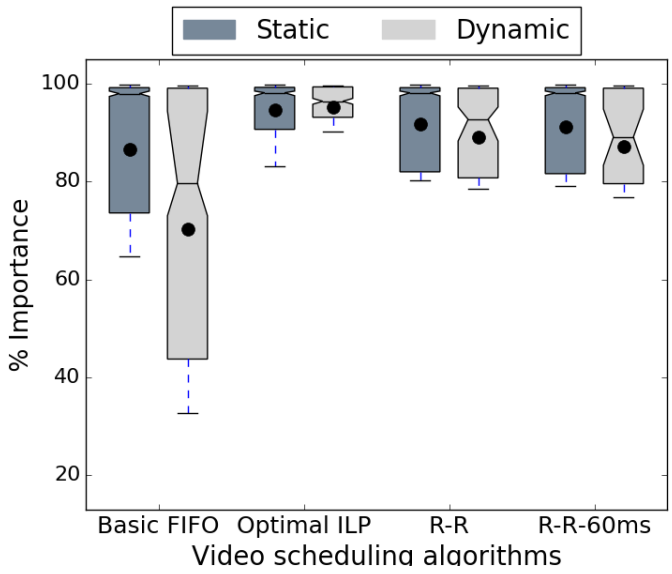


Figure 4.4: Flow importance ratio of the basic FIFO, optimal, R-R, and R-R-60 ms algorithms for static and dynamic videos

segments categories). The basic FIFO presents better performances on the static videos than on the dynamic ones, while we observe the opposite behavior with the optimal algorithm. This can be explained by the relative cumulative size of the B video frames, which is more important in the dynamic videos. The delivery performance on dynamic videos are more sensitive to the network resource variations and to the frame scheduling and discarding policies. In contrast, the B frames are smaller in the static videos; so discarding them does not always suffice to free enough network resources for on-time delivery of the most important video frames. The R-R and R-R-60 ms algorithms provide better performance than the basic FIFO on both static and dynamic video categories, and have a similar size of IQR for both categories.

Finally we analyze the impact of the parameter $T^{decision}$ on the performance of the R-S-T algorithms (both the reactive R-R-T and the predictive R-P-T algorithms) for the flow importance ratio. We considered only the simulations on the dynamic videos, since they are the most sensitive to the network resource variations and to the frame scheduling and discarding policies. The results reported in Figure 4.5 show that both algorithms are not sensitive to the parameter $T^{decision}$, at least for values below 200 ms. Indeed the bottom whisker of the R-S-T box-plot decreases by less than 5 % when the parameter $T^{decision}$ increases from 40 ms to 200 ms. We conclude that the proposed

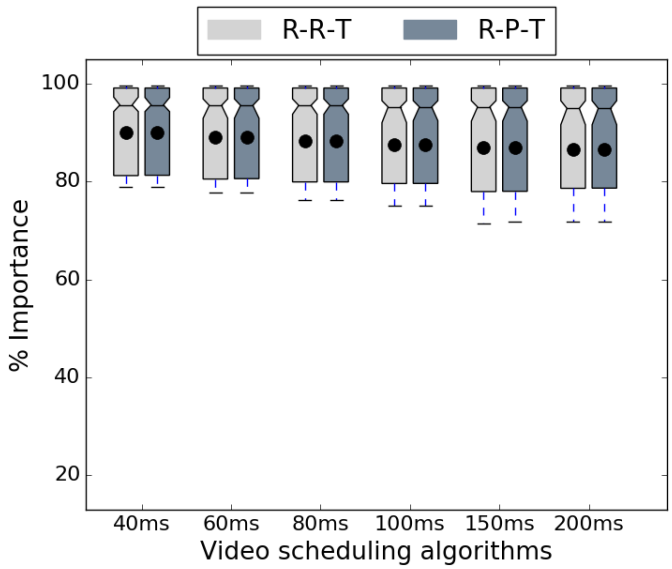
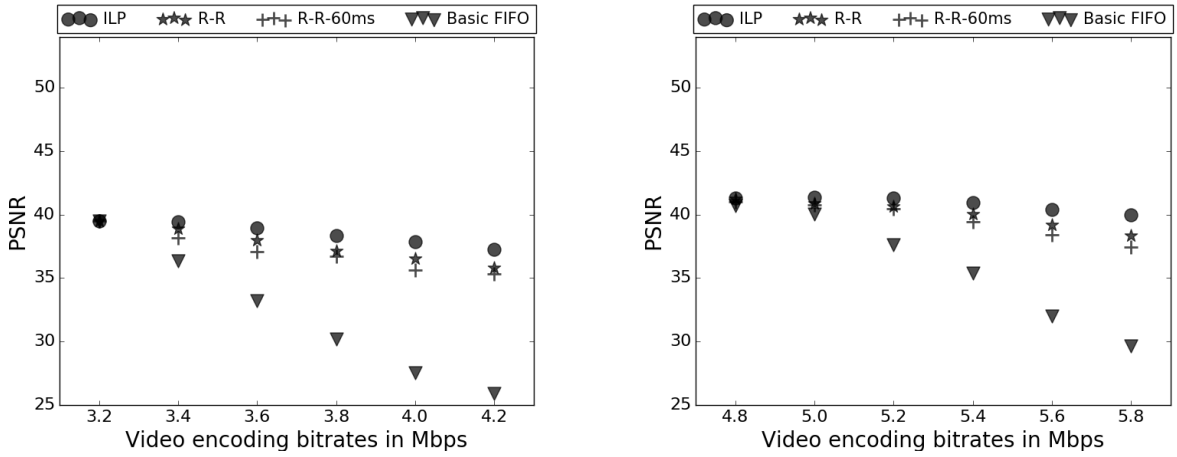


Figure 4.5: Impact of $T^{decision}$ on the flow importance ratio, for R-S-T algorithms and dynamic videos

algorithms can tolerate a delay to discard a late video frame of up to $200\,\mathrm{ms}$, without major impact on the system performances. Network delays may contribute to increase delays in the frame discarding decisions, but RTT are rarely higher than $150\,\mathrm{ms}$, even in cellular networks. Besides, both algorithms behave the same way. It means that estimating the network throughput and taking account of this estimation, as done in the R-P-T algorithms, do not improve the performances of the video scheduling algorithms. So the reactive R-R-T algorithms, being easier to implement, are more attractive.

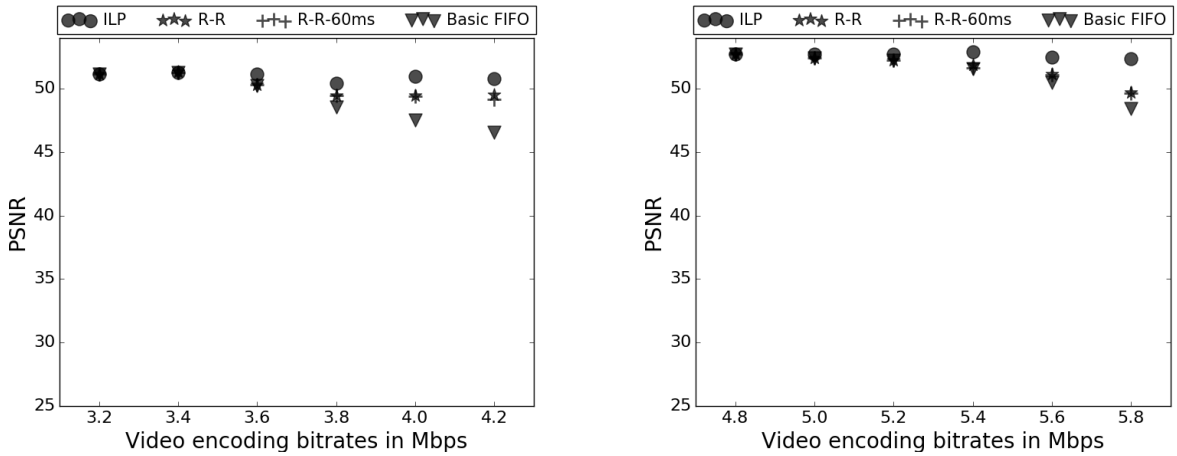
4.2.3 Spatial quality distortion

We exploit the simulations of 6480 video delivery scenarios, especially the list of video frames that are decoded on time at the client side to rebuild the videos as they should be displayed, *i.e.* without the missing video frames. Then we compare the corresponding rebuilt and original videos with two objective metrics: the PSNR and the MS-SSIM.



(a) dynamic videos with cellular networks

(c) dynamic videos with WiFi networks



(b) static videos with cellular networks

(d) static videos with WiFi networks

Figure 4.6: Cloud of points representing the average PSNR of dynamic or static videos combined to cellular or WiFi networks

PSNR

Figure 4.6 shows the average PSNR of each algorithm for different combinations of video content (dynamic or static) and network type (WiFi or cellular). The x-axis represents the bitrate of the considered video segments. The PSNR is computed with regards to the original video with bitrate $7\,\mathrm{Mbps}$. The video encoding bitrates are $3.2\,\mathrm{Mbps}$, $3.4\,\mathrm{Mbps}$, $3.6\,\mathrm{Mbps}$, $3.8\,\mathrm{Mbps}$, $4.0\,\mathrm{Mbps}$ and $4.2\,\mathrm{Mbps}$ for the cellular network, and $4.8\,\mathrm{Mbps}$, $5.0\,\mathrm{Mbps}$, $5.2\,\mathrm{Mbps}$, $5.4\,\mathrm{Mbps}$, $5.6\,\mathrm{Mbps}$ and $5.8\,\mathrm{Mbps}$ for the WiFi network. We show here the average value of the PSNR for each video scheduling algorithm, each category of videos (static and dynamic), and each of the 6 video encoding bit rates.

Figure 4.6 shows that the average value of the PSNR is a decreasing function of the video encoding bitrate, since the bi-rate is related to the throughput deficit. Indeed, there is no throughput deficit on the cellular (respectively WiFi) network in the video delivery scenarios in which the video encoding bite rate is $3.2\,\mathrm{Mbps}$ (respectively $4.8\,\mathrm{Mbps}$), so the PSNR is here only due to video encoding. The larger is the video bitrate, the higher is the throughput deficit. The frame discarding scheme then applies, with a higher impact on dynamic videos, because, as already reported in Section 4.2.2, the dynamic videos are more sensitive to video scheduling than static, and also because two consecutive frames in the static videos present smaller differences. The average PSNR of the optimal algorithm decreases slightly (by $2\,\mathrm{dB}$ for the dynamic videos and by $1\,\mathrm{dB}$ for the static videos). This demonstrates that our strategy has the potential to guarantee a smooth video playback until the end of the segment even in the case of severe mis-prediction of the bitrate (more than $25\,\%$ throughput deficit). On the contrary, applying the basic delivery scheme (basic FIFO algorithm) results in severe quality drops on both the cellular and the WiFi networks, and especially for the dynamic videos (up to $11\,\mathrm{dB}$). In contrast, the R-S-T algorithms perform closely to the optimal ILP.

We evaluate the parameter $T^{decision}$ set in the R-S-T algorithms on the PSNR metric. We show in Figure 4.7 the box-plot of the PSNR over all dynamic video delivery scenarios. All box-plots show similar distributions, which confirms the conclusion in Section 4.2.2 about the limited impact

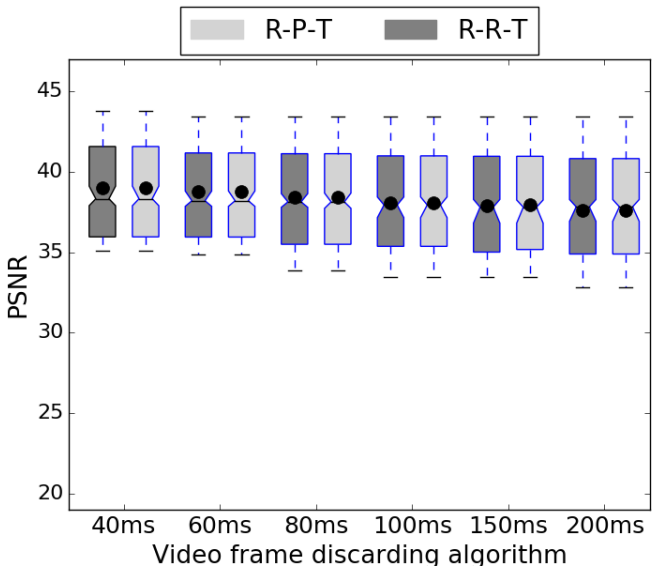
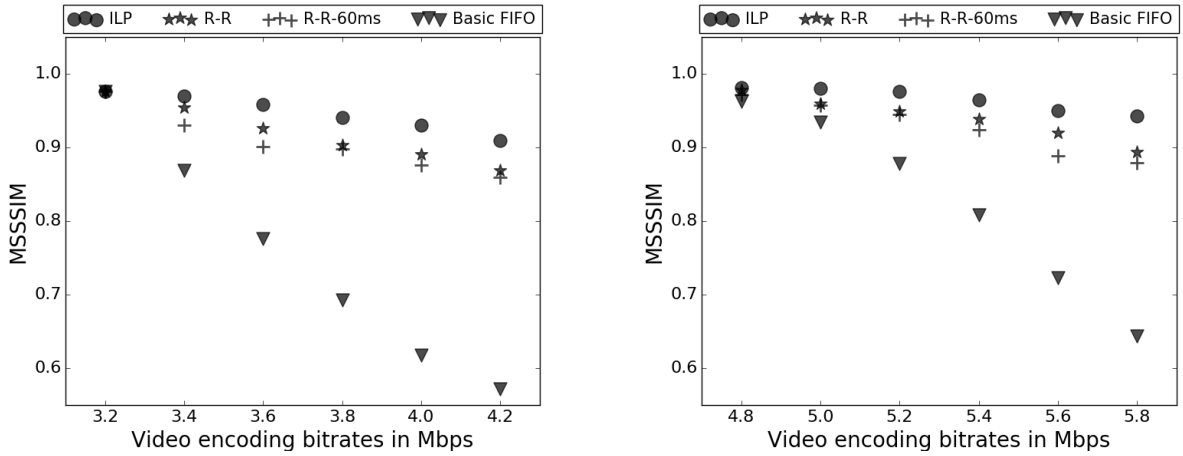


Figure 4.7: PSNR comparison of R-S-T algorithms for dynamic videos combined with all network traces

of $T^{decision}$ on the performance of the scheduling algorithms.

MS-SSIM

We apply the same process to compute the MS-SSIM metric. Figure 4.8 presents the average values of the MS-SSIM obtained with the dynamic videos and the different video scheduling algorithms over, respectively, the cellular network (Figure 4.8a), and the WiFi network (Figure 4.8b). The MS-SSIM ranges between 0 and 1, and humans barely notice qualiyt difference for MS-SSIM above 0.9. The optimal algorithm presents values of MS-SSIM higher than 0.9 in all cases, although the basic video delivery (basic FIFO algorithm) can have video quality as low as 0.5. The R$-$R and R-R-$60\,\mathrm{ms}$ algorithms perform closely to the optimal algorithm with MS-SSIM higher than 0.86 on the cellular network and 0.88 on the WiFi network. We conclude that the proposed HTTP/2-based video scheduling algorithms significantly enhance the spatial video quality compared to basic video delivery algorithm.



(a) dynamic videos with cellular networks

(b) dynamic videos with WiFi networks

Figure 4.8: Cloud of points representing the average MS-SSIM of dynamic videos combined with cellular and WiFi networks

4.2.4 Jitters and temporal degradation

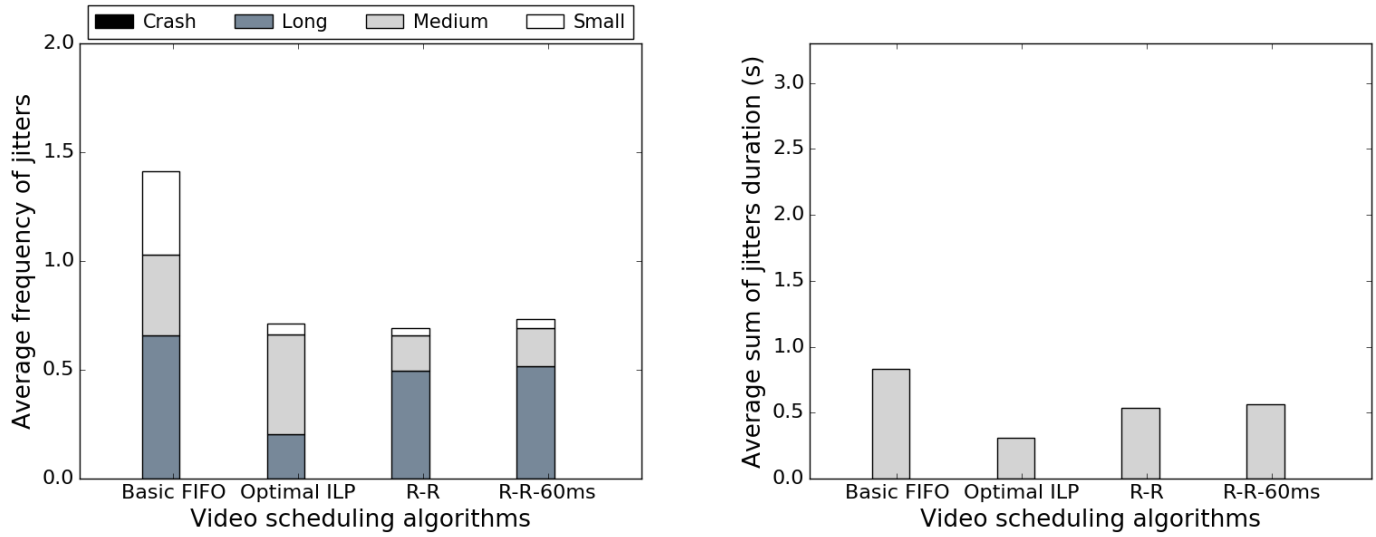
We now analyze the temporal degradation caused by the adverse network conditions and the video frame discarding process: jitters and rebuffering. Each video segment contains $6\,\text{seconds}$ of video content and is encoded at $30\,\text{fps}$.

A traditional video player displays all video frames of a video segment; it pauses the video display when a video frame is missing, and resumes playback when this video frame has been received (*rebuffering*). We observed that rebuffering may result in an additional delay of up to $2\,\text{s}$ between the original video stream and the displayed video stream (excluding the initial buffer). In contrast, our video frame discarding algorithms always meet the $1\,\text{s}$ low-latency requirement, at the price of some *jitters*, *i.e.* temporal discontinuities on the video display. We evaluate in the remaining the severity, the frequency and the duration of the jitters. We distinguish jitters regarding their duration as follows:

- **Small jitter**: if 3 to 5 consecutive video frames are missing
- **Medium jitter**: if 6 to 15 consecutive video frames are missing
- **Long jitter**: if 15 to 40 consecutive video frames are missing

	Classification of the perceived video quality		
Algorithm	**Entirely received**	**With jitters and no crash**	**With crash**
Basic FIFO	34 %	44 %	23 %
ILP	34 %	66 %	0 %
R-R	34 %	65 %	1 %
R-R-60 ms	34 %	63 %	3 %

Table 4.3: Ratio of videos with different anomalies per algorithm



(a) Jitter frequency for no-crash video

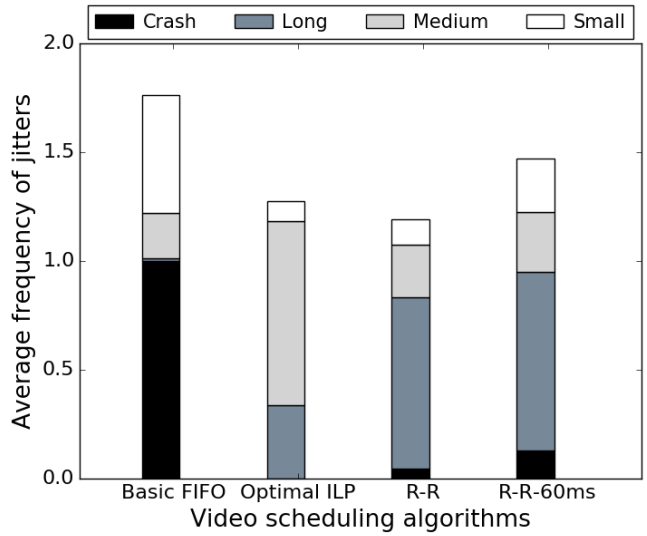
(b) Sum of jitters duration for no-crash videos

Figure 4.9: Temporal distortion per algorithm for video/network combinations which we classify into two categories, a first category for whom the basic FIFO does crash

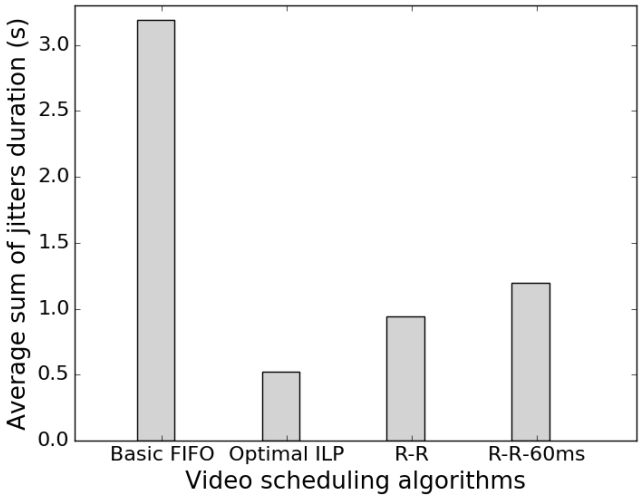
- **Crash**: if more than 40 consecutive video frames are missing

We also distinguish the videos into categories based on the worst type of QoE anomaly: (i) entirely received videos, (ii) videos with small-medium-long jitters, and $(iiii)$ videos with at least one crash. For each video scheduling algorithm we compute the percentage of videos owning to each category. A video crash corresponds to a jitter longer than 1370 ms which decreases the MOS below 2 according to Kester et al. [51]. In light of this observation, the results reported in Table 4.3 indicate that our proposed algorithms significantly reduces the video experiencing crashes, from 23 % for the traditional delivery to only 3 % for the R−R−60 ms algorithm. The optimal ILP completely eliminates crashes.

Second, we compare the the frequency and the duration of the jitters, with the basic FIFO



(a) Jitter frequency for crash videos



(b) Sum of jitters duration for crash videos

Figure 4.10: Temporal distortion per algorithm for video/network combinations which we classify into two categories, a second category for whom the basic FIFO does not crash

algorithm. We classify again the videos into categories: (i) the videos that do not experience any crash with the basic FIFO algorithm, and (ii) the others. We analyze then the jitters for both categories by determining the average frequency and the sum of the duration of the jitters. The results are presented in Figures 4.9 and 4.10. Figures 4.9a and 4.9b focuses on the no-crash videos while Figure 4.9b shows that these videos present on average a cumulative duration of jitters higher than $500\,\mathrm{ms}$, which decreases the video quality to poor according to Kester et al. [51]. The proposed algorithms decrease both the cumulative duration and the frequency of the jitters of the videos owning to this first category. The optimal ILP shows that a theoretical optimal implementation manages to have jitters below $500\,\mathrm{ms}$, which is considered as a good-fair quality by Kester et al. [51]. Figures 4.10a and 4.10b focuses on crash videos. The proposed algorithms almost eliminates crashes without increasing the average frequency of jitters, at the expense of longer jitters.

It is also worth mentioning that our proposed algorithms always decrease the frequency and the duration of the jitters on the dynamic videos. In contrast, for static videos, our algorithms decrease the frequency and the duration of jitters on the no-crash videos, but only decrease the

duration of jitters on the crash videos. These results are still not sufficient to fully evaluate the QoE. They show that our proposed algorithms enhance the video quality with lower frequency and cumulative duration of jitters. The results motivate future works on conducting subjective tests and on evaluating the QoE.

Conclusion

Our proposal aims to avoid DASH video delivery rebuffering events while ensuring a low latency delay between the displayed and the original video flows. We propose in this part to use a novel approach to deal with bandwidth shortage occurring during a video segment delivery: the implementation of video frame discarding policies with HTTP/2 until the end of segment. We leverage the HTTP/2 reset stream feature to discard video frames. We evaluate our algorithms by combining dynamic and static videos with cellular and WiFi network traces. We propose an optimal ILP and practical algorithms. We evaluate our results with various spatial and temporal metrics. The proposed algorithms avoid the accumulation of delay while presenting an acceptable displayed video quality. They reduce the quality distortion especially for dynamic videos. The proposed algorithms behave closely to the optimal solution. We also conclude that the HTTP/2 *weights* feature is not necessary for our algorithms since there is no need for interleaving frames.

For future work, it is interesting to conduct subjective video quality tests because the QoE measured with spatial and temporal objective metrics is not sufficient to assess how humans react to jitters in addition to HAS rate-adaptation. It is also important to study the best trade-off between the chunk size selection, the representation switching frequency and the video frame discarding scheme to maximize the QoE. We showed in our paper [11] that investing on the capacity of decoders to deal with missing frames brings advantages. The performance of our algorithms depends on the video encoding (number and size of frames B) and also on the decoder strategy. Besides, HTTP/2 have other features such as *server push*. The *server push* can be used to fetch the video frames from the server to the client and avoid the frequent client requests of video frames. However, this optimization needs implementation efforts on the server side and a complementary

communication mechanism between the client and the server.

This part opens various perspectives. In particular, interleaving may be useful for immersive videos, where *tiles* (spatial video parts) can be decoded independently of each others and have different importance with respect to the client point of view. We study this perspective is part II of this dissertation.

Part II

HTTP/2-based viewport adaptive strategies for 360° streaming

“You can have it all. Just not all at once.”

- Oprah Winfrey -

Introduction

An efficient $360°$ video streaming not only requires to adapt the content to the network resources but it also takes into account the viewer viewport in order to maximize the QoE. A $360°$ video is first captured by multiple video cameras. Then, the contents are stitched together by using various methods to determine the overlapping regions [39] to provide a $360° \times 180°$ degrees of field of view [83]. Today's coders are not able to process three-dimensional structured content. To cope with this issue, the $360°$ video is then projected into a two-dimensional plan to enable its processing by traditional 2D codecs [111]. Several works address efficient $360°$ video 2D-projections following different projection schemes such as equirectangular [24, 50, 85, 101, 102] and cubemap projections [36, 60].

Once the video is projected, it goes through traditional compression techniques such as temporal and spatial compression. There is usually a high correlation between neighboring pixels of an image. Encoders uses intra-prediction techniques to reduce the spatial redundancy inside each of the images composing the video which enables efficient spatial compression and reduce the overall size of the video. The most popular technique, which is adopted by many standards, is the transform technique where each image is split into blocks and the transform is applied to each block [5, 63].

Encoding techniques based on tiling enable to encode an image into independently decodable parts. Tiling presents the advantages of dividing the picture into rectangular and regular size parts known as tiles. It enables efficient parallel processing and provides entry points for local access as explained by Misra et al. [68]. It is important to find the appropriate size of tiles that cover the Region of Interest (RoI) without significantly exceeding it. Large tiles enable a better compression rate but requires a higher quality since the data in the tiles covering the RoIs should be encoded with the same bitrate [88]. Figure 4.11 shows an example of a 6×4 grid tiled-picture.

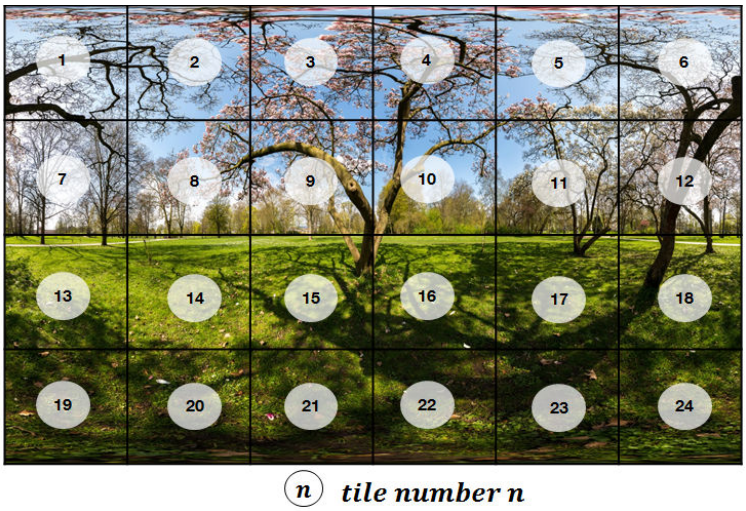


Figure 4.11: A 6×4 tiles grid encoded picture

Several studies propose to use tiling to stream 360° videos [32, 34, 37, 73]. Tiled media enables the client to retrieve and render the portion watched by the client (the viewport) in a faster way compared to untiled 360° media [52]. Without tiling, the entire content must be decoded before extracting and rendering the viewport [58]. Then, the portion of the video watched by the viewer is extracted and rendered on the client device according to its Field of View (FoV). A FoV dimension depends on the headset but it is usually about $110° \times 90°$.

Modern encoders such as HEVC are able to compress the video as a set of tiles [44, 58, 77, 86, 112]. The resulting tiles (spatially independent sub-videos of each segment) are then encoded into multiple quality levels. The client can select the proper quality distribution over tiles in accordance with his viewport. The client predicts the viewer head orientation for the next segment and requests it in advance to meet the latency requirements. The tiles overlapping with the predicted viewport are transmitted at high bitrate, and others are transmitted at low bitrate.

Several techniques allow a HAS client to request a set of tiles at relevant resolutions in accordance with the viewer head orientation. Le Feuvre and Concolato [58] propose to add additional packaging information on the ISO Base Media File Format (ISOBMFF) to enable a tile-based encapsulation and byte range-based delivery of tiles. Niamut et al. [77] present the Spatial Representation Description (SDR) feature of the MPEG DASH standard [1]. It extends the MPD of

DASH by describing spatial relationships between associated pieces of video content. The challenging question remains to decide the set of tiles the client should request in advance as well as their qualities which depends on the viewport prediction.

Due to the presence of the response delay, the client should estimate the viewport in the future in order to make decisions. Viewport prediction should enable the client to estimate the viewer head orientation for the next video segment. Jeong et al. [46] predict the user's viewport by utilizing the location information of the sound sources in the 360° video. Aladagli et al. [2] have analyzed the correlation between real viewer head movements and sequential fixations predicted from a saliency model of the video. They show that predicting the viewport based only on the saliency map is insufficient. Ban et al. [6] consider user's personalized information and the behaviors of other users watching the same video to predict future viewport. Fan et al. [31] combine the history of the user headset orientations and the saliency of the video content to predict the viewport for the n next video frames. They have reached $89\,\%$ of accuracy in the best cases.

Viewport prediction is challenging due to its random nature and it is prone to errors. Nguyen and Yun [76] show that the error levels get higher as the prediction interval increases. According to Bao et al. [7], a viewer position can be accurately predicted in a maximum time scale of $500\,\mathrm{ms}$ in advance. However, without an accurate viewport prediction, viewers may watch lower resolution pixels when they change their motion which decreases the QoE. Nguyen et al. [74] show that a viewport-independent streaming may outperform all considered viewport-adaptive methods in case of long buffer (exceeding $2\,\mathrm{s}$) and inaccurate viewport predictions since HTTP/1 suffers from long response delays to the client urgent requests of viewport tiles.

Viewport prediction in large time scale is prone to errors. The client should predict the viewport late enough to obtain a high accuracy, but also early enough to request and receive the tiles of the video segment in time. In case of erroneous predictions, the client should correct its decision, but with HTTP/1, it needs to receive the inappropriate quality levels of the video parts before requesting the appropriate ones. For example, the client can receive video tiles in wrong quality levels, which leads to a useless over-consumption of the network resources or a video frame after

its deadline, which enforces rebuffering and decreases the QoE.

In this part, we propose to enable the client to benefit from the totality of the available network resources as well as of the enhancement of its viewport prediction accuracy over time. For each video segment, the client selects the quality level of tiles according to a first viewport prediction, and it adjusts its decisions later on time based on HTTP/2 priority and reset stream features and by running a second viewport prediction. We briefly describe the related works about viewport-dependent streaming on the following.

Related works on immersive streaming

Viewport-adaptive solutions have recently captivated the research community [24, 34, 37, 73, 84]. Corbillon et al. [24] propose to create seven different copies of the same 360° video content based on pre-defined positions of the viewport. For each copy, they decrease the quality level of the background and maximize the quality level of a specific viewport. Then, each copy is encoded into several quality levels so as to match the bandwidth. Authors in [37, 73] propose tile-based viewport-dependent adaptive streaming approaches. The 360° video is divided into *tiles*, each tile being encoded at multiple different quality levels. Each video segment is therefore composed of different spatial video tiles. Only the tiles that overlap the viewport are streamed at the highest quality while tiles of the background are delivered at a lower quality. Graf et al. [34] describe an implementation of tiles as specified within modern video codecs such HEVC/H.265 and VP9 so as to enable the tile-based viewport-dependent adaptive streaming solutions.

Recent studies [75, 81] propose to use HTTP/2 protocol for 360° video streaming. Nguyen et al. [75] propose to send each 360° video tile in an HTTP/2 stream and assign higher weights to streams that deliver viewport tiles in order to ensure that they are received and decoded first by the client. They propose an RR-policy-based implementation of the HTTP/2 priority since tiles are completely independent. However, this solution brings benefits only in the case of bandwidth insufficiency, where some tiles may arrive late. This solution may also suffer from inaccurate viewport predictions which accidentally leads to prioritize tiles of the background and increases

the risk of delaying the delivery of real viewport tiles. Nguyen et al. propose to use the HTTP/2 stream termination feature to cancel the delivery of delayed tiles before requesting tiles of the next video segment which reduces rebuffering events and saves bandwidth. Petrangeli et al. [81] focus on the k-push HTTP/2 feature. In order to reduce the impact of high RTT on the video delivery over constrained networks, only one request is sent from the client to the server, specifying the quality levels of tiles. Tiles are consequently pushed by the server to the client using the FCFS-policy-based priority implementation.

Qian et al., Xie et al. [84, 108] show that the viewport accuracy gets higher with shorter prediction intervals. According to Bao et al. [7], an accurate viewport position can be predicted in a time scale lower than $500\,\mathrm{ms}$ in advance. All the aforementioned works, dealing with the viewport-dependent adaptive streaming, consider only one viewport prediction per video segment. Since they do not exploit the enhancement of the viewport prediction accuracy over time, their proposed systems may suffer from the prediction's errors.

In this part, we propose to run two predictions at the client side to request a video segment. By using the HTTP/2 priority and reset stream features at the second prediction, the client adapts the delivery of the ongoing tiles in accordance with the most recent predicted viewport and bandwidth. We develop adaptive streaming algorithms based on 1-viewport-prediction and 2-viewport-predictions with different HTTP/2 weight functions. Before concluding, we compare the algorithms and evaluate their performance in chapter 6.

Chapter 5

A viewport-dependent streaming solution based on 2-predictions and a HTTP/2 viewport-weight function

Contents

In this chapter, we propose a new adaptive streaming system, which enables the client to control the video tiles delivery by leveraging the HTTP/2 stream-control features. Our main idea consists on requesting each video tile in a dedicated HTTP/2 stream, running two viewport predictions, and then control the delivery of the video tiles by invoking the stream-level control features defined in the HTTP/2 standard.

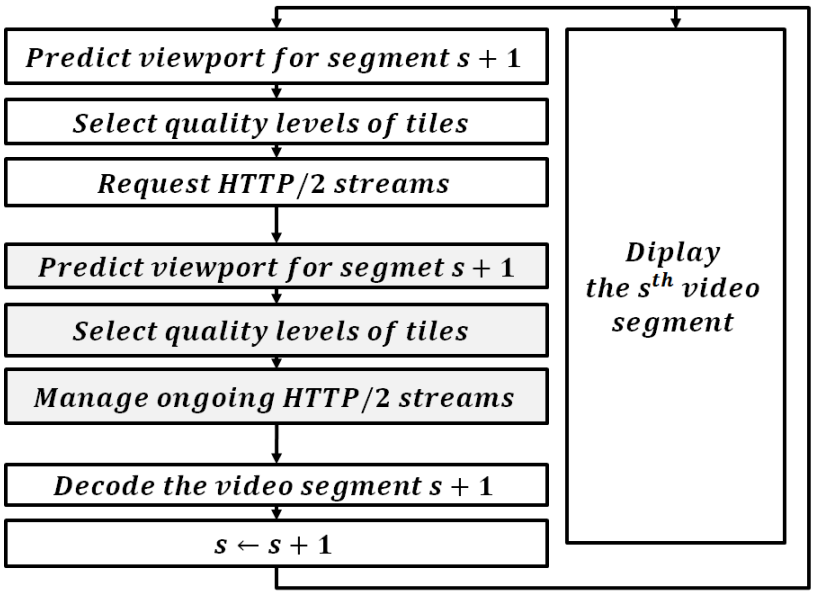


Figure 5.1: General principles of our proposed scheme. During the display of the s^{th} video segment, the client manages the delivery of the $(s+1)^{th}$ video segment using two consecutive viewport predictions

5.1 System model for immersive video delivery

Our solution is client-based. Figure 5.1 shows the general scheme of our proposed system. At the display time of a video segment s, the client executes a first viewport prediction for the next video segment $s + 1$. It selects the quality levels of future tiles in accordance with the first viewport prediction. Then, it requests each video tile in an HTTP/2 stream. Later, the client runs a second viewport prediction and, based on it, the client adapts the ongoing delivery of video tiles by using the HTTP/2 stream features. Finally, the client decodes the received tiles, reconstructs the $360°$ video content, extracts and displays the viewport corresponding to the user viewing direction. The main keys of our proposal are the following:

- Our proposed scheme is generic: it works with any HAS technology, with any number of tiles, and it is orthogonal to the viewport prediction model.
- The client runs two viewport predictions.
- At each prediction, the client divides the video sphere into regions depending on the pre-

dicted viewport, as we describe in section 5.1.1.

- We describe the tile quality selection mechanism in section 5.1.2. The selected video quality of tiles depends on the viewport prediction occurrence and also on the state of the network.
- At both predictions, the client prioritizes the viewport tiles delivery (tiles overlapping with the viewport) by assigning dynamic HTTP/2 weights as we describe in section 5.1.3.
- At the second prediction, the client can stop the delivery of a set of tiles and also request new ones according to the relative tiles position to the predicted viewport as we detail later in Table 5.3.

5.1.1 Viewport-dependent regions division

The probability of tiles to be in the real viewport varies in accordance with the distance to the predicted viewport center and the prediction accuracy. We propose to divide the video into four regions at each viewport prediction. Figure 5.2 shows an illustration of the four regions and a grid of 6×4 tiles overlapping. We denote by NbT the number of the resulting tiles (for example, 24 tiles in the grid shown in Figure 5.2). We define the regions as follow:

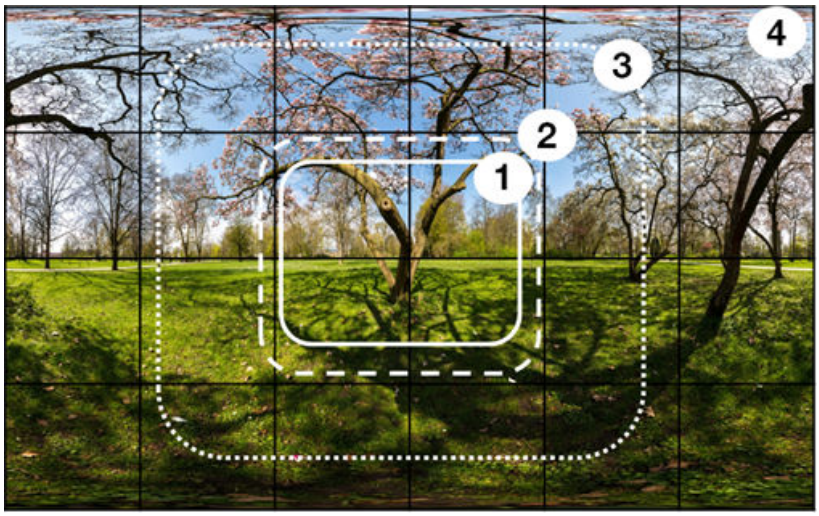


Figure 5.2: Overlapping between video regions and a 6×4 tiles grid for an equirectangular projection

1. The **viewport area**: In this dissertation, usually we refer to the predicted viewport as the viewport, unless we state that we refer to the real displayed one. We define as the *viewport tiles*, the tiles overlapping entirely or partially with the real viewport.

2. The **extension area**. It corresponds to small spatial region of the video frames surrounding the viewport. The extension area has high chances to overlap with the future viewport within a short time due to natural small head movements of the viewer. Almquist et al. [3] show that using a small extra safety margin protect against fast back-and-forth directional changes. We define then the **extended area** which is the set of the viewport and the extension ones.

3. The **immediate background area**. When the prediction is accurate, this part of the video is not expected to be displayed in the Head Mounted Display (HMD) in a near future, but since prediction is prone to errors, it should nevertheless be delivered to the client in case of an unexpected sudden head movement.

4. The **far background area**. This region is the opposite of the viewport in the sphere. During one segment length, the probability that an abrupt head movement makes that the viewport overlaps these tiles is low. If the prediction is accurate, this region has no chances to be required by the viewer.

We denote by the extended area the set of region 1 and 2, while the background area refers to regions 3 and 4 in the following. We define as the *extended viewport tiles*, the tiles overlapping entirely or partially with the extension area and by the *background tiles* the tiles overlapping with the background area.

At each prediction, the client exploits its viewport to calculate the ratio of the extended area pixels of each video frame overlapping with each tile t. Then, it computes its average value over all the frames of segment s, as described in Equation (5.1). We denote by $p^k_{i,s,t}$ the ratio of the predicted extended area pixels of the i^{th} frame of the video segment s overlapping with tile t. The

integer k represents the viewport prediction occurrence on segment s.

$$p^k_{s,t} = \frac{1}{n} \sum_{i \in [1,n]} p^k_{i,s,t} \tag{5.1}$$

Secondly, the client classifies the tiles into extended viewport tiles if $p^k_{s,t} > 0$, and to background tiles otherwise. In order to compute the far background tiles, the client replaces the normal dimension of the viewport by very large dimensions (for example 180×180 degrees instead of 110×90 degrees). In this case, $p^k_{s,t} = 0$ represents a tile of the far background.

5.1.2 Tile quality level selection

At the first prediction, the client first attributes the lowest possible quality level to the background tiles. Then, it predicts the remaining available network resources and assigns the highest quality level to the extended viewport tiles (tiles of viewport + extension areas) as a margin of security.

At the second prediction, the viewport prediction is more accurate, however, the client can be limited by the network resources. When the network resources are sufficient enough to receive tiles of regions (1, 2 and 3) at time, the client requests the new extended viewport tiles (ie. tiles overlapping with the viewport at the second prediction but not at the first prediction) at a high quality, otherwise it keeps the delivery of these tiles at a low quality.

5.1.3 Tiled media delivery with HTTP/2

One of our main contributions is our proposal to deliver each video tile in an HTTP/2 stream and set priorities between streams. Figure 5.3 illustrates an HTTP/2 communication between a server and a client. First, the client requests the MPD file. It extracts information about the video and then requests each video tile in an HTTP/2 stream.

In order to optimize the viewport tiles delivery, the client not only assigns a high quality level to the extended viewport tiles, but it also assigns high HTTP/2 weights to streams transmitting extended viewport tiles so as to prioritize their delivery. We use an RR-implementation for HTTP/2-

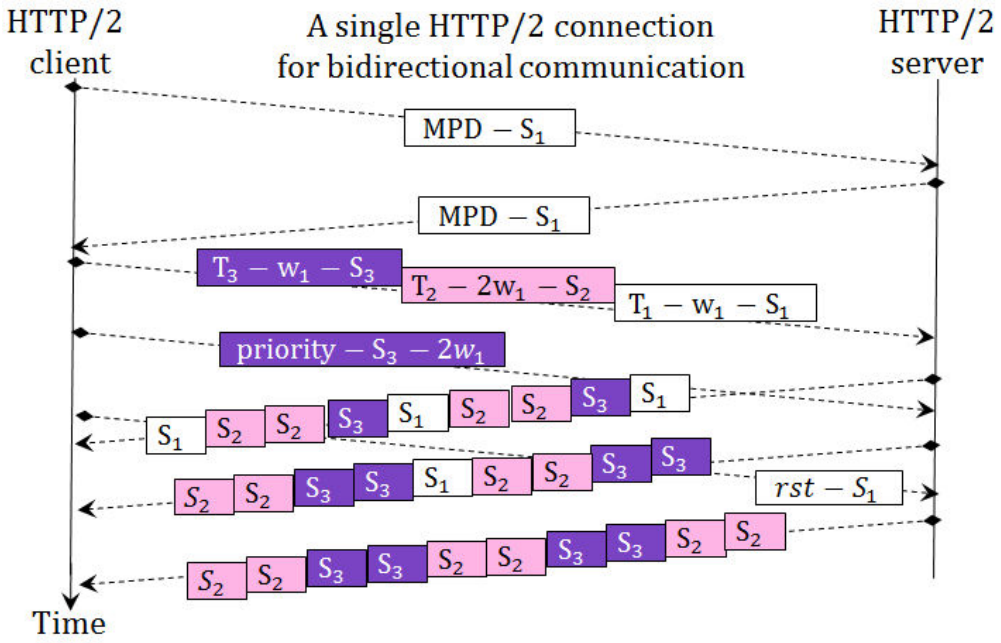


Figure 5.3: HTTP/2 client to server communication example. Each HTTP/2 S_i has a weight w_i and contains a video tile T_i

priority as described in Figure 2.8 since the video tiles are completely independent. The server multiplexes the streams with respect to the weights set by the client. At any time, the client can manage the resource allocation between streams.

At the first viewport prediction for segment s, the client assigns for each video tile t a weight equal to either $256 \times p^1_{s,t}$ if it is in the extended area, or 1 if it is in the background area. At the second viewport prediction for the same segment s, the client updates the stream weights depending on the values of the viewport pixels ratio per tile $p^2_{s,t}, t \in \{1, \ldots, NbT\}$. Other weight functions can be attributed to the stream, which we investigate in chapter 6.

Our HTTP/2-based proposal allows the client to upgrade or degrade the quality level of a tile while being delivered, by first canceling the current delivery, and then requesting this tile with another quality level.

In Figure 5.3, the client decides to double the weight of stream S_3 so the server adjusts the resources distribution by assigning $\frac{2}{5}$ of network resources to S_3, $\frac{2}{5}$ to S_2 ,and $\frac{1}{5}$ to S_1. Then, the client decides to cancel stream S_1, so the server stops S_1 delivery and adjusts the resource sharing with $\frac{1}{2}$ to S_3 and $\frac{1}{2}$ to S_2. We detail the different possible delivery scenarios as well as the client

decisions in Section 5.2.

5.2 Practical HTTP/2-based Algorithms

5.2.1 Viewport Prediction Accuracy Model

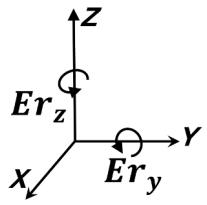


Figure 5.4: Axis of the sphere

Category	Er_z^1 values	Er_y^1 values
Small class	[15,35]	[10,20]
Medium class	[45,65]	[25,35]
Big class	[75,95]	[40,50]

Table 5.1: Viewport prediction errors per error class in degrees

Our scheme is compatible with any viewport prediction system that enhances its accuracy over time. In this work, we do not propose a particular viewport prediction model. We assume that a video segment is composed of a set of n frames. For each frame $i \in [1, n]$ and a prediction $p \in [1, 2]$,[1] we define the deviation errors Er_i^p as a couple of (Er_z, Er_y) values. Er_z and Er_y denote respectively the deviation around the z-axis and the y-axis between the center of the real and predicted viewports, as shown in Figure 5.4. The deviation errors decrease between the first and the second viewport predictions. We classify the deviation errors into small, medium and big classes as described in Table 5.1

[1]We set only two prediction updates for the sake of simplicity but our proposal can run with any number of consecutive predictions.

Accuracy	**Accuracy assumptions**	
model name	1 s **before**	500 ms **before**
First model: **segment-based prediction**	All video frames of a segment present the same viewport prediction error class	Perfect
Second model: **frame-based prediction**	Viewport prediction error class per frame depends on the video frame ID (order) in the segment	Perfect

Table 5.2: Two viewport accuracy models: 1) per video segment and 2) per video frame.

We distinguish between two models of the accuracy enhancement of viewport prediction over time as shown in Table 5.2. For the first model, we consider that all the predicted viewports per frame in the segment have the same error class. For the second model, we consider that frame i has a higher viewport prediction accuracy than frame $i+1$ since video frame i is displayed before video frame $i+1$, which is more realistic. Viewport errors only concerns the first prediction. We suppose that the second prediction is accurate.

5.2.2 Decisions of the HTTP/2 client

We present in the following the algorithms that we propose for an HTTP/2 viewport-based scheme with two consecutive viewport predictions per video segment. We denote by $t_s^{display}$ the display time of the video segment s. In this paper, without loss of generality, we consider that the first prediction of the viewport for segment s is done one second before $t_s^{display}$ and the second prediction is made 500 ms later. Let $t_s^p, p \in \{1, 2\}$ be both prediction times for segment s. In future work, we will explore different number of consecutive predictions and various time differences between these predictions.

Since the viewport prediction improves over the time, the second prediction is more accurate than the first one. In constrained networks, our proposal allows the client to cancel the ongoing

delivery of the far background area tiles of segment s if it does not completely receive these tiles at t_s^2. Besides, at t_s^2 the client may upgrade or degrade some video tiles according to the evolution of their viewport ratio $p_{s,t}^p$ from prediction $p = 1$ to prediction $p = 2$. We summarize in Table 5.3 these different possible scenarios.

Tile position		**Completely received at t_s^2**	
Prediction 1	**Prediction 2**	**Yes**	**No**
Background area	Extended area	**Upgrade Algorithm**	
Extended area	Extended area	No actions	**Update weight**
Background area	Background area	No actions	**Update weight**
Any area	Far background	No actions	**Cancel stream**
Extended area	Immediate Background	No actions	**Degrade Algorithm**

Table 5.3: Actions of the HTTP/2 client at the second prediction t_s^2 for segment s

We detail in Algorithms 5.1 and 5.2 the different network conditions considered by the client while taking a decision of upgrading or degrading a tile quality level. We denote by S_t^{Low} and S_t^{High} the total size of tile t at respectively the high and the low quality levels. We denote by $S_t^{Q,r}$ the remaining amount of data for tile t that the client should still receive at quality $Q \in \{\text{low}, \text{high}\}$. We denote by B^r the expected remaining available bandwidth between t_s^2 and $t_s^{display}$, by B^{up} the remaining bandwidth available to request viewport tiles in a high quality level and by $Pool^{up}$ the set of the viewport tiles that need to be upgraded.

Algorithm 5.1 Degrade Algorithm:

1: **if** $S_t^{High,r} > S_t^{Low}$ **then**
2: Cancel tile t delivery
3: Request tile t with a Low quality
4: **else**
5: Keep the ongoing delivery of tile t
6: Update the remaining size $S_t^{Q,r}$ of the requested tile t

In Algorithm 5.1, the client cancels a tile delivery to request it with a low quality only when this action enables a bandwidth saving. It is possible that a video tile of the background is already entirely or almost entirely received with a high quality due to an erroneous first viewport prediction.

In this case, the HTTP/2 reset stream feature does not bring any bandwidth saving; therefore the client does not use it.

Algorithm 5.2 Upgrade Algorithm

1: $B^{up} = B^r - \sum_{t \notin \{Pool^{up} \cup OppositeArea\}} S_t^{Q,r}$ {The client first ensures the correct delivery of tiles that do not need to be upgraded}
2: **while** $\sum_{t \in Pool^{up}} S_t^{High} > B^{up}$ and $B^{up} > 0$ **do**
3: Find $t_{min} \in Pool^{up} / \{p_{s,t_{min}}^2 \leqslant p_{s,t}^2 \forall t \in Pool^{up}\}$
4: Remove tile t_{min} from $Pool^{up}$
5: $B^{up} = B^{up} - S_{t_{min}}^{Low,r}$ {The client checks if it is possible to upgrade all tiles of $Pool^{up}$. If not, it keeps the delivery of the least important video tiles at the low quality level and try to upgrade the quality of the remaining tiles.}
6: Cancel the delivery of all tiles of $Pool^{up}$
7: Request all tiles of $Pool^{up}$ with high quality level

In Algorithm 5.2, the client checks the available bandwidth resources before deciding whether to upgrade tiles' quality levels. First, it prioritize the complete delivery of all video tiles (except for the far background area tiles). Then, it upgrades the quality of the most important video tiles with respect to bandwidth constraints.

We evaluate our proposed scheme in terms of quality of viewport pixels and network consumption in Chapter 6 and we compare it to different HTTP/2 and HTTP/1-based algorithms.

Chapter 6

Evaluation of viewport-dependent streaming algorithms

Contents

In this chapter, we aim to answer two main questions; the first question concerns the effects of considering two successive viewport predictions for a video segment instead of only one prediction. We analyze its impact on the video quality and on the network resources consumption. The second question is about the effects of using HTTP/2 protocol to stream each video tile in an HTTP/2 stream compared to HTTP/1-based streaming solutions. We evaluate the benefits and challenges of the HTTP/2 reset stream and the priority features. In order to answer these questions, we compare our proposed solution, named *2-predictions HTTP/2 viewport weight*-based algorithm, to four different algorithms described in Section 6.1.

6.1 Reference Algorithms

We compare the performance of our proposal with the following algorithms:

1-prediction HTTP/1 The client runs only one viewport prediction at $t_s^1 = 1\,\mathrm{s}$ before the segment display. It selects the quality level of the tiles and requests them in the FCFS order.

2-predictions HTTP/1 The client runs a first viewport prediction for segment s at $t_s^1 = 1\,\mathrm{s}$ and second one at $t_s^2 = 0.5\,\mathrm{s}$. It cannot cancel any ongoing delivery, nor prioritize the delivery. However, it can request at higher quality a tile in the extended area that was mistakenly predicted in the background and so previously requested at low quality.

2-predictions HTTP/2 default weight The client behaves as in our proposed solution. However, it does not attribute weights to the HTTP/2 streams. The server automatically assigns weight equal to 16 to all streams, which enables a fair network resource sharing [9].

2-predictions HTTP/2 sized weight The client behaves as in our proposed scheme. However, it assigns weights to the HTTP/2 streams so that all video tiles are received at the same time. We denote by w_i the weight of stream S_i delivering a video tile t and by T the set of all video tiles. The weight function is given by eq. (6.1).

$$W_{i,t} = \frac{S_t^{Q,r}}{\min_{k \in T}(S_k^{Q,r})} \tag{6.1}$$

6.2 Datasets

We developed a Python 3 simulator to evaluate the performance of our proposal and to combine the video datasets, the traces of the users' viewport behavior, the prediction errors, and the network resources.

We consider three publicly available 360° videos [23]. The first one is the *RollerCoaster* video which was captured in front of a moving roller-coaster car. Corbillon et al. [23] observes that viewers aim to have a quite fixed head orientation while watching the RollerCoaster video, which usually makes the viewport prediction accurate. The second video is a virtual reconstruction of Venice with a flying camera, named *Venice*. Viewers usually move their head softly as to look at the different parts of the city. The third one is the *Timelapse* video which presents a series of scene

cuts including many fast moving objects. Viewers usually moves fast their head orientations while watching this video, which makes an accurate viewport prediction challenging.

Real viewers' viewport traces are publicly available [23]. We randomly select 9 users (3 users per video). For the purpose of assessing our proposal, we generate erroneous viewport predictions the following way. For a specific $360°$ video content and a specific user, the dataset provides the user head orientation on each video frame. We transform the real viewport by introducing random errors to generate inaccurate viewport predictions in accordance with fig. 5.4 as to simulate of the results of a real viewport prediction mechanism. Changing the user's head orientation very often during a video segment time scale imposes to display more tiles at the highest quality level. The viewer head movements frequency and speed depend on both the video content and the viewer.

For each video, we consider the first $60\,\mathrm{s}$. We used the open-source Kvazaar [54] software to encode each video with an average bitrate target of $5\,\mathrm{Mbps}$ and $18\,\mathrm{Mbps}$ and a resolution of 3840×1920. Then, we split the video into 6×4 tiles and into one-second video segments. We extracted the size of each encoded tile and used it as input of our simulator. For each scenario, we consider a constant bandwidth ranging from $8\,\mathrm{Mbps}$ to $18\,\mathrm{Mbps}$.

6.3 Performance evaluation

For each algorithm and each prediction accuracy model (as described in Table 5.2) we evaluate the network resource consumption as well as the status of each pixel of the viewport. Three status are possible:

- Viewport pixels received at a high quality
- Viewport pixels received at a low quality
- Unreceived viewport pixels

When a set of pixels of the viewport is not received because tiles containing these pixels are delayed, the player displays a green bloc instead of it, which badly impacts the QoE. It is important

to highlight that we only consider pixels of the real viewport in our performance evaluation and not pixels of the extended viewport.

We represent in Figure 6.1 the impact of different prediction errors levels on the viewport pixels status of each of the studied algorithms. We consider the *Timelapse* video, a mean of the results of the three viewers we randomly selected and the segment-based viewport prediction accuracy model (Table 5.2). Results show that the more important prediction errors are, the higher the ratio of the unreceived viewport pixels (black blocs) is. We observe that running two predictions with HTTP/1 only enhances the performances for bandwidth values higher than $13\,\mathrm{Mbps}$. However, our proposed *2 predictions HTTP/2 viewport weight* scheme always decreases the ratio of unreceived viewport pixels for all errors levels as well as the ratio of low quality viewport pixels. We also observe that the HTTP/2 default weight and sized weight functions significantly decrease the video quality especially for small prediction errors and should be avoided.

Besides, Figure 6.2 shows that running two predictions with HTTP/1 always consumes more network resources that running only one prediction. The additional network consumption for bandwidth values lower than $13\,\mathrm{Mbps}$ is a total waste since the video quality is not enhanced as described in Figure 6.1. However, our proposed *2 predictions HTTP/2 viewport weight* scheme always decreases the network resource consumption comparing to the *2-predictions HTTP/1* algorithm and also saves bandwidth resources compared to the *1-predictions HTTP/1* algorithm for bandwidth values lower than $13\,\mathrm{Mbps}$. Besides it consumes only $13\,\%$ more network resources than the *1-prediction HTTP/1* in the worst cases. In fact, our proposed *2 predictions HTTP/2 viewport weight* automatically adapts the tiles delivery and quality levels to the bandwidth resources due to the weight functions-based algorithms we propose (ref to section 5.2).

Figure 6.3 show that all *2 predictions HTTP/2 based algorithms* consume the same network resources. We conclude that considering two consecutive predictions in the tiles' request and delivery process, as enforced in our proposal smartly consumes the network resources. However, Figure 6.1 also shows the importance of considering an appropriate HTTP/2-weight function based on viewport predictions in order to enhance the status of the viewport pixels and the overall video

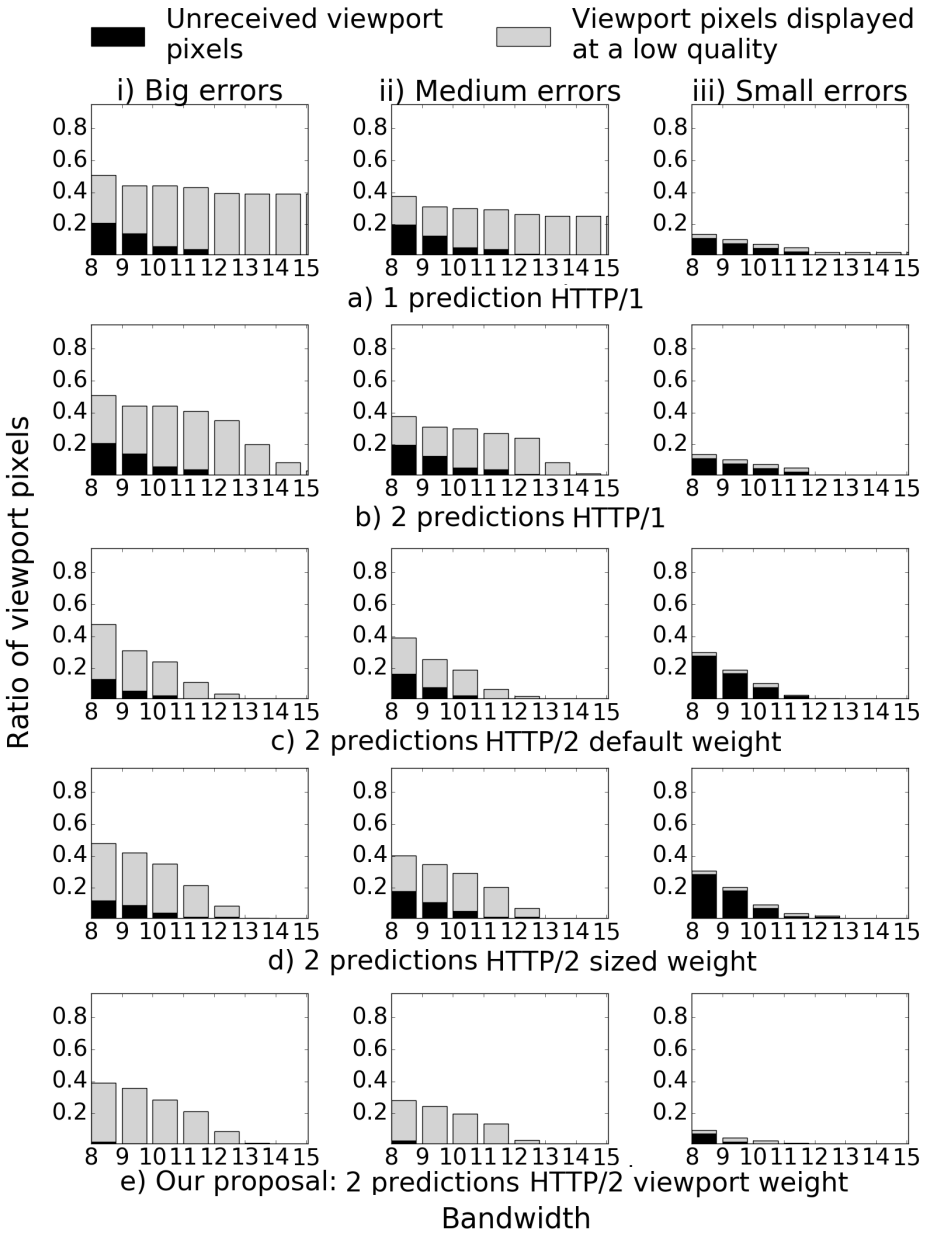


Figure 6.1: Ratio of viewport pixels incorrectly or badly received per algorithm and per prediction error level, for the *Timelapse* video. The viewport ratio of correctly received pixels is the complement until 1.

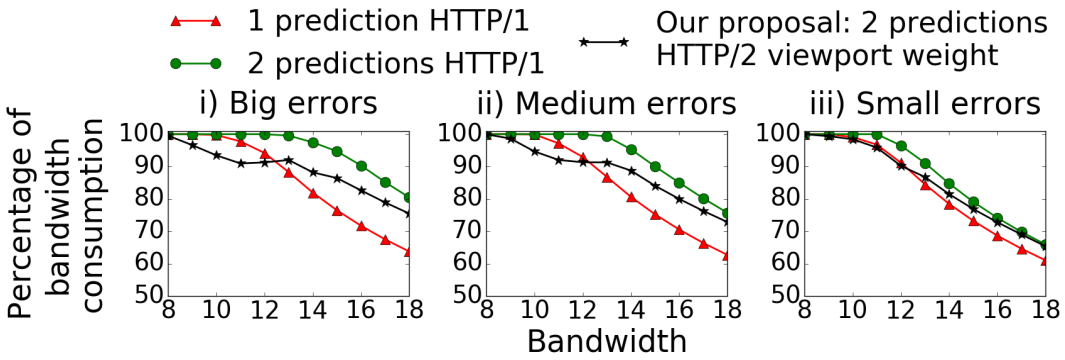


Figure 6.2: Bandwidth consumption for the *Timelapse* video, using the segment-based viewport prediction accuracy model

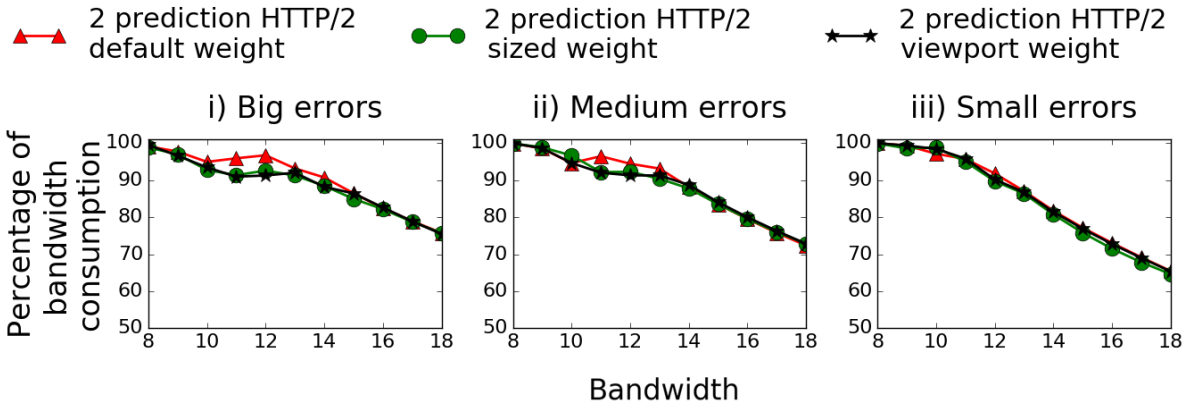


Figure 6.3: HTTP/2-based algorithms bandwidth consumption for the *Timelapse* video, using the segment-based viewport prediction accuracy model

quality.

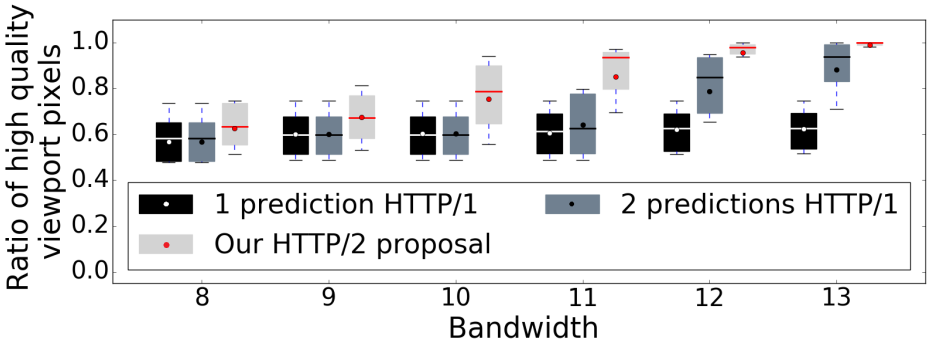


Figure 6.4: Viewport pixels received with the highest quality

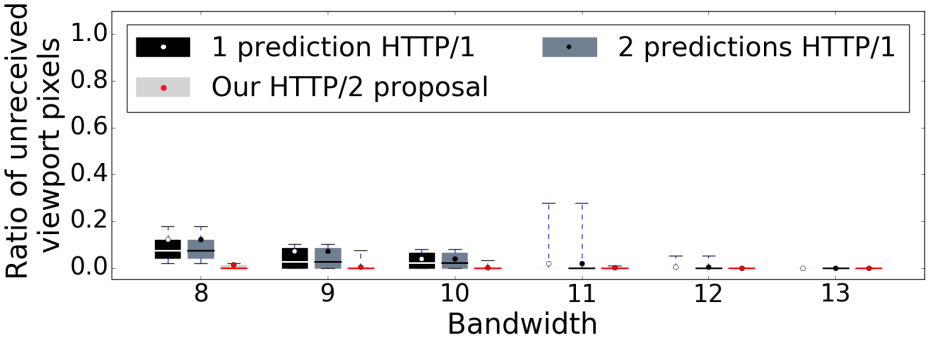


Figure 6.5: Representation with box-plots of the viewport pixels status of all the video segments watched by all users and for all viewport error classes

We use box-plots in Figure 6.5 to represent the results for all the viewers and the videos of our datasets. On each box-plot the colored surface represents the distribution of the viewport pixels' quality status for all video segments, users, videos and errors taking into account the segment-based viewport prediction accuracy model. Inside this surface, the horizontal line and the point represent, respectively, the median and the average values. The whiskers extending from either side of the box represent the ranges of the bottom $15\,\%$ and top $15\,\%$ values. Our proposal shows stable performance results. It always presents a higher ratio of high quality viewport pixels and significantly decreases the ratio of unreceived ones.

We also evaluate the performance of our proposal when the frame-based accuracy model is applied, as introduced in Table 5.2. We consider that at the first prediction video frames displayed

in the first 330 ms have small error predictions, then frames of the next 330 ms have medium error predictions and the rest ones have big error predictions.

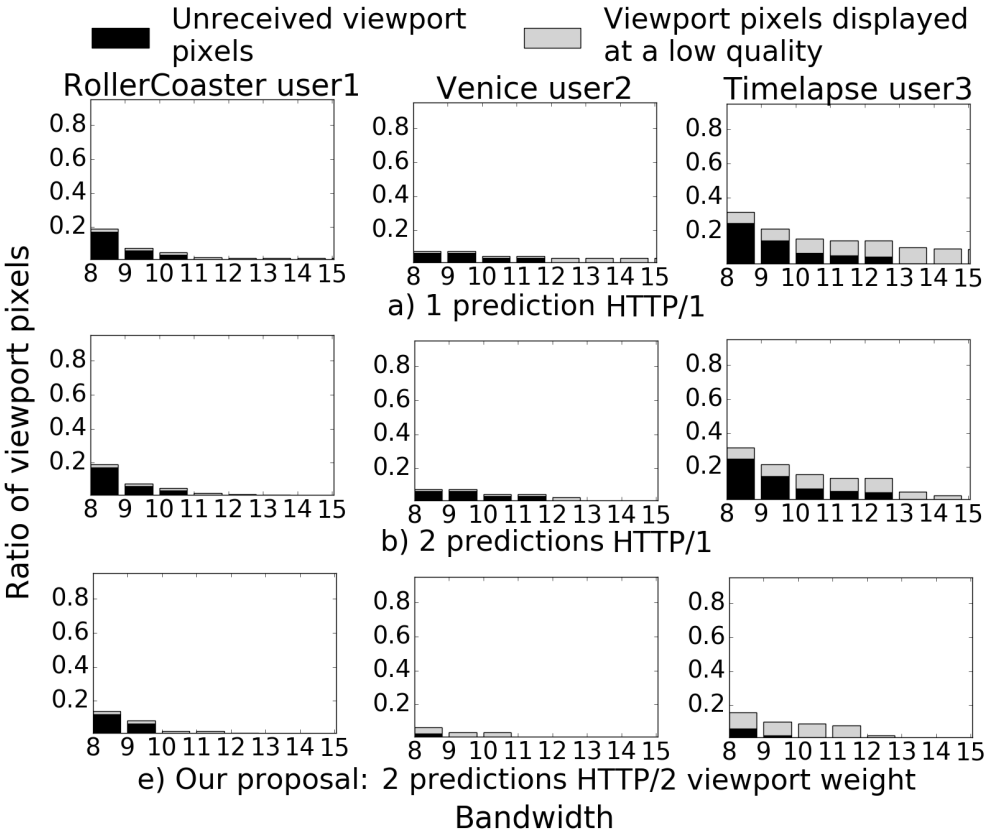


Figure 6.6: Unreceived and low quality received pixels of the real viewport area for the frame-based viewport accuracy scheme and for different videos and users

Figures 6.6 and 6.7 show the results for different videos and viewers. Our proposed scheme always increases the overall quality of the viewport pixels. It also decreases the bandwidth consumption, up to 12 % in the best cases compared to the HTTP/1-viewport adaptive schemes, and up to 35 % compared to delivering the entire video segment with the highest quality level (18 Mbps in the case of our study).

Besides, the performance related to the *Timelapse* video show the advantages of our proposal. Figure 6.7 shows an increase of the consumed bandwidth for the Timelapse video compares to the Venice and the RollerCoaster ones. Indeed, our proposal is remarkably interesting for dynamic videos where the user often moves his head and a high number of tiles is required for display. The RollerCoaster is easy to predict but in our study we applied the same rules for all the videos in

order to understand the functioning of all the algorithms in different scenarios, so introduced the same errors than the other contents, which justifies the high unreceived viewport pixels observed for the RollerCoaster video.

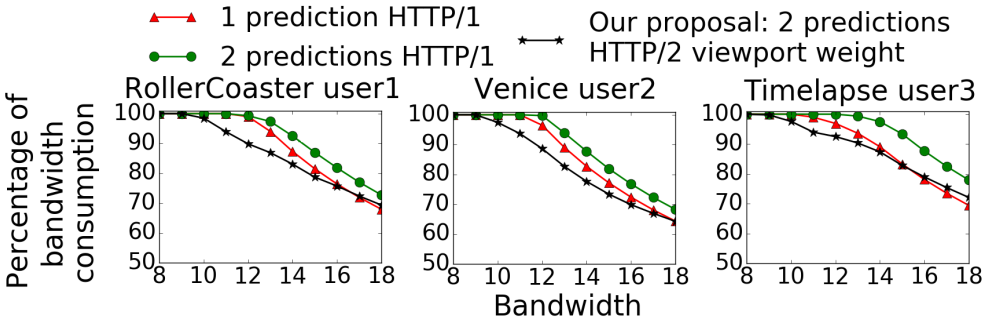


Figure 6.7: Bandwidth consumption for the frame-based viewport prediction accuracy scheme and for different videos and users

We also observed that the HTTP/2 default and sized weight-based algorithms reduce the bandwidth consumption but can decrease the video quality. The results are similar to those of segment-based viewport prediction accuracy model. Since the overall quality of the viewport pixels is not always enhanced with the sized and the weight-based algorithms, the choice of an appropriate HTTP/2 weight function is a crucial parameter of our proposal.

Conclusion

In this part, we described our proposed scheme to increase the QoE on $360°$ video streaming by using a novel approach that exploits the enhancement of the viewport prediction over the time. We propose to request each tile in HTTP/2 stream and we leverage the HTTP/2 and priority features to handle the tiles delivery. We evaluate our scheme in terms of quality of viewport pixels and network consumption and we compare it to different HTTP/2 and HTTP/1-based algorithms. Results show that our proposal automatically adapts to the available network resources and prioritizes the delivery of viewport tiles. We also demonstrate that the choice of the HTTP/2 *weight* function is critical in order to prioritize the viewport tiles since the weigh function determines the interleaving scheme between the HTTP/2 streams delivering the video tiles.

In order to benefit from the reset stream and priority features, implementation efforts should

be done on the client side so that it can communicate with the server, cancel ongoing requests and attribute weights to the streams. In the best of our knowledge, there is no commercial players supporting the HTTP/2 weight feature in 2018. However, *TiledMedia*, a company that was founded in 2016, has developed a player supporting the reset stream feature of HTTP/2. This work was lead in partnership with Akamai and Harmonic companies in the context of optimizing 360° video streaming but they do not explore the benefits of the HTTP/2 weight features [98].

Future studies can focus on studying the best trade-offs on the available network resources and the time the second viewport prediction occurs. Indeed, short prediction period makes the system more sensitive to bandwidth variations while a long prediction period results in a lower viewport quality. Besides, the client can run several predictions, it would be interesting to study the optimal number of viewport predictions to run separately in order to achieve the best performance. Subjective video quality tests to assess how human viewers experience the unreceived and the low quality viewport pixels are also of a great value and should be conducted.

Chapter 7

Future Directions

“Design is a way of life, a point of view."

- Paul Rand -

Contents

The emergence of the low latency services, like live OTT or immersive videos, has created new challenges for the HTTP-based streaming systems. These services not only require fast data delivery, but they also require good video quality leading to important data volumes. However, existing systems have mechanisms that use aggressively the available network resources which usually leads to rebuffering events and a decrease of the QoE. Since 2015, HTTP/2 offers interesting features that can be used to better utilize the network resources and reduce the end to end latency.

In this thesis, we have showed that the HTTP/1-based low latency streaming systems cannot always adapt to the network constraints. Hence, we have proposed new systems to enhance the video delivery by leveraging the HTTP/2 reset stream and priority features. We summarize the main parts of this thesis in section 7.1. Then in section 7.2, we discuss possible future works.

7.1 Achievements

The aim of this dissertation is the description of possible ways to use HTTP/2 features to enhance the performances of low latency streaming services.

In Chapter 1, we generally introduced the subject and described the motivations of our work. In Chapter 2, we detailed the key technical elements of the video streaming system and we introduced the HTTP/2 protocol. Then, we divided our main contributions in two parts.

In Part 1, we focused on improving the QoE of live streaming sessions where the DASH adaptation mechanism miss-estimates the available bandwidth. We started the part with a brief description of related works about video frame discarding strategies. Then, we proposed HTTP/2-based frame discarding strategies in order to avoid the accumulation of rebuffering delays in Chapter 3. We presented the main concept of our proposal and we detailed optimal and heuristics algorithms of video frame discarding. In Chapter 4, we described the simulation environment and we presented the performance evaluation of our proposal. The results show that our proposed scheme avoid the accumulation of delay between the displayed and the original video flows while presenting an acceptable displayed video quality.

In Part 2, we focused on the service of 360° video streaming. We proposed to improve the viewport pixels quality of immersive videos by refining the tiles delivery in accordance with the available bandwidth and the viewer head orientation. The part begins with a description of the related works about the viewport-dependent adaptive streaming. In Chapter 5, we presented the main element of our proposal which enables the client to control the 360° video tiles delivery by leveraging the HTTP/2 stream-control features. In Chapter 6, we described the simulation environment and the findings of the performance evaluation. Our proposal automatically adapts to the available network resources and prioritizes the delivery of viewport tiles when the HTTP/2 weight function is set in a dependent way of the viewport prediction.

7.2 Perspectives

In this Ph.D dissertation, we proposed some ideas to enhance the QoE of low latency services with HTTP/2 reset stream and priority features.

The results of our work present a motivation to conduct subjective tests in order to assess how viewers perceive the temporal jitters in the case of the traditional live streaming, and the viewport-adaptive video streaming. Besides, the findings of this thesis motivates to develop players supporting the HTTP/2 priority and reset stream features. Still, the effectiveness of canceling an HTTP/2 stream instead of establishing a new TCP connection depends on the implementation and, especially of the configured size of the application layer buffer on the server side. When this buffer is not short enough, a reset stream will not be efficient since a number of HTTP/2 frames of the concerned stream would already have been pushed to the server buffer.

There are still plenty of open questions and existing scientific problems to be solved to provide HTTP-based streaming systems with low latency and enhance the QoE at minimum cost. Further HTTP/2-based ideas can be explored in future works. We briefly describe two of them in the following.

It would be interesting to model HTTP/2-based solutions in order to efficiently deliver Scalable Video Codec (SVC) encoded content. SVC encodes a high-quality video bitstream with one or more subset layers, each enhancing the basic low quality. The client can improve the video quality in the viewport by requesting enhancement layers. Nasrabadi et al. [71] use a SVC-based solution to efficiently stream 360° videos. They show that SVC-based tile delivery helps reducing the cost of updating the quality of already downloaded low quality viewport tiles. However, they do not use the HTTP/2 protocol. The HTTP/2-client can request each SVC layer in an HTTP/2 stream, and then reset the inappropriate streams or prioritize the most important ones in accordance with the evolution of the viewport prediction accuracy and the bandwidth status.

Recently, the Common Media Application Format (CMAF) standard has also captivated the attention of the multimedia research community. However, simply using CMAF segments is not enough to reduce latency. To obtain low end-to-end latency, the CMAF-based systems must be

paired with encoders, CDNs and modified client behaviors so that the overall system enables low latency. The currently proposed systems are based on HTTP/1.1 chunked transfer encoding. The chunked transfer encoding feature was however removed from the HTTP/2 protocol. It would be interesting to conduct a research study to explore the benefit and the inconvenient of building a low latency CMAF system based on the HTTP/2 protocol, typically by taking benefits from the multiplexing and the push server features.

Bibliography

[1] I. .-A. 2:2015. Spatial relationship description, generalized url parameters and other extensions. Technical Report 23009, ISO/IEC.

[2] A. D. Aladagli, E. Ekmekcioglu, D. Jarnikov, and A. Kondoz. Predicting head trajectories in 360° virtual reality videos. In *2017 International Conference on 3D Immersion (IC3D)*, 2017.

[3] M. Almquist, V. Almquist, V. Krishnamoorthi, N. Carlsson, and D. L. Eager. The prefetch aggressiveness tradeoff in 360° video streaming. In *Proceedings of the 9th ACM Multimedia Systems Conference, MMSys, The Netherlands*, pages 258–269, 2018.

[4] I. Ayad, Y. Im, E. Keller, and S. Ha. A practical evaluation of rate adaptation algorithms in http-based adaptive streaming. *Computer Networks*, 133:90–103, 2018.

[5] S. Bae, J. Kim, and M. Kim. Hevc-based perceptually adaptive video coding using a dct-based local distortion detection probability model. *IEEE Trans. Image Processing*, 25(7): 3343–3357, 2016.

[6] Y. Ban, L. Xie, Z. Xu, X. Zhang, Z. Guo, and Y. Wang. Cub360: Exploiting cross-users behaviors for viewport prediction in 360 video adaptive streaming. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, July 2018.

[7] Y. Bao, H. Wu, T. Zhang, A. A. Ramli, and X. Liu. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *IEEE Int. Conf. on Big Data*, 2016.

[8] A. Beben, P. Wisniewski, J. M. Batalla, and P. Krawiec. ABMA+: lightweight and efficient algorithm for HTTP adaptive streaming. In *Proceedings of the 7th International Conference on Multimedia Systems, MMSys, Austria*, pages 2:1–2:11, 2016.

[9] M. Belshe, R. Peon, and M. Thomson. Hypertext transfer protocol version 2 (http/2) standard. Technical Report RFC-7540, IETF, May 2015.

[10] M. Belshe, R. Peon, and M. Thomson. Hypertext transfer protocol version 2 (http/2) standard. RFC 7540, IETF, 2015.

[11] M. Ben-Yahia, Y. L. Louédec, L. Nuaymi, and G. Simon. When HTTP/2 rescues DASH: video frame multiplexing. In *IEEE Conference on Computer Communications Workshops, INFOCOM Workshops*, pages 677–682, 2017.

[12] N. Bouzakaria, C. Concolato, and J. L. Feuvre. Fast DASH bootstrap. In *IEEE Int. Workshop on Multimedia Signal Processing, MMSP*, 2015.

[13] P. Campbell. Why every brand needs to migrate to http/2 and how to do it. Technical report, State of Digital, 2016. http://tinyurl.com/zefhtkm.

[14] J. Chakareski, J. G. Apostolopoulos, W. Tan, S. Wee, and B. Girod. Distortion chains for predicting the video distortion for general packet loss patterns. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP*, pages 1001–1004, 2004.

[15] Y. Chang, T. Lin, and P. C. Cosman. Network-based H.264/AVC whole-frame loss visibility model and frame dropping methods. *IEEE Trans. Image Processing*, 21(8):3353–3363, 2012.

[16] W. Chérif, Y. Fablet, E. Nassor, J. Taquet, and Y. Fujimori. DASH fast start using HTTP/2. In *Proceedings of the ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV*, pages 25–30, 2015.

[17] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021 white paper, December 2017.

[18] Computer Modules, Inc. DVEO Division. Video streaming protocols: A brief history, December 2016. https://dveo.com/pdf/Streaming-Protocols-A-Brief-History.pdf.

[19] Conv. Viewer experience report 2015. Conviva Annual Report, December 2015. http://tinyurl.com/hc5nwdy.

[20] R. Corbel, E. Stephan, and N. Omnes. Http/1.1 pipelining vs http2 in-the-clear: Performance comparison. In *2016 13th International Conference on New Technologies for Distributed Systems (NOTERE)*, 2016.

[21] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon. Cross-layer scheduler for video streaming over MPTCP. In *Proceedings of the International Conference on Multimedia Systems, MMSys*, pages 7:1–7:12, 2016.

[22] X. Corbillon, F. Boyrivent, G. A. D. Williencourt, G. Simon, G. Texier, and J. Chakareski. Efficient lightweight video packet filtering for large-scale video data delivery. In *IEEE International Conference on Multimedia & Expo Workshops, ICME Workshops*, pages 1–6, 2016.

[23] X. Corbillon, F. De Simone, and G. Simon. 360-Degree Video Head Movement Dataset. In *ACM Conf. on Multimedia Systems (MMSys)*, 2017.

[24] X. Corbillon, G. Simon, A. Devlic, and J. Chakareski. Viewport-adaptive navigable 360-degree video delivery. In *IEEE Int. Conf. on Communications (ICC)*, 2017.

[25] M. Dabrowski, R. Kolodynski, and W. Zielinski. Analysis of video delay in internet TV service over adaptive HTTP streaming. In *Position Papers of the Federated Conference on Computer Science and Information Systems, FedCSIS, Poland*, pages 143–150, 2015.

[26] K. A. Darabkh, A. M. Awad, and A. F. Khalifeh. Intelligent and selective video frames discarding policies for improving video quality over wired/wireless networks. In *IEEE International Symposium on Multimedia*, 2013.

[27] K. A. Darabkh, A. M. Awad, and A. F. Khalifeh. Efficient pfd-based networking and buffering models for improving video quality over congested links. *Wireless Personal Communications*, 79(1):293–320, 2014.

[28] K. A. Darabkh, A. M. Awad, and A. F. Khalifeh. New video discarding policies for improving UDP performance over wired/wireless networks. *Int. Journal of Network Management*, 25(3):181–202, 2015.

[29] F. Dobrian, A. Awan, D. A. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang. Understanding the impact of video quality on user engagement. *Commun. ACM*, 56(3): 91–99, 2013.

[30] Z. Duanmu, K. Zeng, K. Ma, A. Rehman, and Z. Wang. A quality-of-experience index for streaming video. *J. Sel. Topics Signal Processing*, 11(1):154–166, 2017.

[31] C.-L. Fan, J. Lee, W.-C. Lo, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu. Fixation prediction for 360° video streaming in head-mounted virtual reality. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV'17, 2017.

[32] V. R. Gaddam, M. Riegler, R. Eg, C. Griwodz, and P. Halvorsen. Tiling in interactive panoramic video: Approaches and evaluation. *IEEE Transactions on Multimedia*, 18(9): 1819–1831, Sept 2016.

[33] D. Gangadharan, L. T. X. Phan, S. Chakraborty, R. Zimmermann, and I. Lee. Video quality driven buffer sizing via frame drops. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA, Volume 1*, pages 319–328, 2011.

[34] M. Graf, C. Timmerer, and C. Müller. Towards Bandwidth Efficient Adaptive Streaming of Omnidirectional Video over HTTP: Design, Implementation, and Evaluation. In *ACM Conf. on Multimedia (MM)*, 2017.

[35] D. He, C. Westphal, and J. J. Garcia-Luna-Aceves. Network support for AR/VR and immersive video application: A survey. In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications, ICETE*, 2018.

[36] Y. He, X. Xiu, P. Hanhart, Y. Ye, F. Duanmu, and Y. Wang. Content-adaptive 360-degree video coding using hybrid cubemap projection. In *2018 Picture Coding Symposium, PCS, USA*, 2018.

[37] M. Hosseini and V. Swaminathan. Adaptive 360 VR Video Streaming: Divide and Conquer. In *IEEE Int. Symp. on Multimedia (ISM)*, 2016.

[38] P. Houze, E. Mory, G. Texier, and G. Simon. Applicative-layer multipath for low-latency adaptive live streaming. In *IEEE International Conference on Communications, ICC*, pages 1–7, 2016.

[39] K. Huang, P. Chien, C. Chien, H. Chang, and J. Guo. A 360-degree panoramic video system design. In *International Symposium on VLSI Design, Automation and Test*, April 2014.

[40] T. 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 *ACM SIGCOMM, USA*, pages 187–198, 2014.

[41] Q. Huynh-thu and M. Ghanbari. Impact of jitter and jerkiness on perceived video quality. In *Workshop Video Process. Quality Metrics*, 2006.

[42] R. Huysegems, T. Bostoen, P. Rondao-Alface, J. van der Hooft, S. Petrangeli, T. Wauters, and F. D. Turck. Http/2-based methods to improve the live experience of adaptive streaming. In *Proceedings of the Annual ACM Conference on Multimedia Conference, MM '15*, 2015.

[43] ISO. Dynamic adaptive streaming over http (dash). Technical Report 23009, ISO/IEC, 2012.

[44] ISO. Information technology – high efficiency coding and media delivery in heterogeneous environments – part 2: High efficiency video coding, iso/iec 23008-2, 2015.

[45] James Stringer. An introduction to video compression, February 2018.

[46] E. Jeong, D. You, C. Hyun, B. Seo, N. Kim, D. H. Kim, and Y. H. Lee. Viewport prediction method of 360 vr video using sound localization information. In *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2018.

[47] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *IEEE/ACM Trans. Netw.*, 22(1):326–340, 2014.

[48] A. Kalampogia and P. Koutsakis. H.264 and H.265 video bandwidth prediction. *IEEE Trans. Multimedia*, 20(1):171–182, 2018.

[49] T. Karagkioules, C. Concolato, D. Tsilimantos, and S. Valentin. A comparative case study of HTTP adaptive streaming algorithms in mobile networks. In *Proceedings of the Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV*, pages 1–6, 2017.

[50] U. Kart, J. Kämäräinen, L. Fan, and M. Gabbouj. Evaluation of visual object trackers on equirectangular panorama. In *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP, Portugal*, 2018.

[51] S. V. Kester, T. Xiao, R. E. Kooij, K. Brunnström, and O. K. Ahmed. Estimating the impact of single and multiple freezes on video quality. In *Human Vision and Electronic Imaging XVI, part of the IS&T-SPIE Electronic Imaging Symposium, Proceedings.*, page 78650O, 2011.

[52] H. Kim, J. Yang, M. Choi, J. Lee, S. Yoon, Y. Kim, and W. Park. Eye tracking based foveated rendering for 360 VR tiled video. In *Proceedings of the 9th ACM Multimedia Systems Conference, MMSys*, 2018.

[53] Koeppel Direct Blog. Peter koeppel, rise of live streaming: Trends and marketing tips, June 2017.

[54] A. Koivula, M. Viitanen, A. Lemmetti, J. Vanne, and T. D. Hamalainen. Performance evaluation of Kvazaar HEVC intra encoder on Xeon Phi many-core processor. In *IEEE Global Conf. on Signal and Information Processing*, 2015.

[55] S. S. Krishnan and R. K. Sitaraman. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. *IEEE/ACM Trans. Netw.*, 21(6):2001–2014, 2013.

[56] J. Kua, G. Armitage, and P. Branch. A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP. *IEEE Communications Surveys and Tutorials*, 19(3):1842–1866, 2017.

[57] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, 2017.

[58] J. Le Feuvre and C. Concolato. Tiled-based adaptive streaming using MPEG-DASH. In *ACM Conf. on Multimedia Systems (MMSys)*, 2016.

[59] S. Lederer, C. Müller, and C. Timmerer. Dynamic adaptive streaming over http dataset. In *Proceedings of the 3rd Multimedia Systems Conference*, 2012.

[60] L. Li, Z. Li, M. Budagavi, and H. Li. Projection based advanced motion model for cubic mapping for 360-degree video. In *IEEE International Conference on Image Processing, ICIP, China*, 2017.

[61] 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*, 32(4):719–733, 2014.

[62] Limelight Networks. The state of online video, December 2016.

[63] C. Lin, W. Yang, and C. Su. FITD: fast intra transcoding from H.264/AVC to high efficiency video coding based on DCT coefficients and prediction modes. *J. Visual Communication and Image Representation*, 38:130–140, 2016.

[64] K. Liu and J. Y. B. Lee. Achieving high throughput and low delay in mobile data networks by accurately predicting queue lengths. In *Proceedings of the ACM International Conference on Computing Frontiers, CF'15*, pages 20:1–20:8, 2015.

[65] L. V. Ma, J. Park, J. Nam, H. Ryu, and J. Kim. A fuzzy-based adaptive streaming algorithm for reducing entropy rate of DASH bitrate fluctuation to improve mobile quality of service. *Entropy*, 19(9):477, 2017.

[66] MarTech Blog. Douglas karr, live streaming trends and statistics, August 2017.

[67] S. Mehdian and B. Liang. Jointly optimal selection and scheduling for lossy transmission of dependent frames with delay constraint. In *IEEE International Symposium of Quality of Service, IWQoS*, pages 168–177, 2014.

[68] K. M. Misra, C. A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou. An overview of tiles in HEVC. *J. Sel. Topics Signal Processing*, 7(6):969–977, 2013.

[69] 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 Annual ACM SIGMM Conference on Multimedia Systems, MMSys*, pages 11–22, 2012.

[70] C. Müller, S. Lederer, C. Timmerer, and H. Hellwagner. Dynamic adaptive streaming over

HTTP/2.0. In *Proceedings of the IEEE International Conference on Multimedia and Expo, ICME*, pages 1–6, 2013.

[71] A. T. Nasrabadi, A. Mahzari, J. D. Beshay, and R. Prakash. Adaptive 360-degree video streaming using layered video coding. In *2017 IEEE Virtual Reality, VR, USA*, pages 347–348, 2017.

[72] Net Insight. How true, synchronized live ott can change the second screen and social tv game, September 2016.

[73] D. V. Nguyen, H. T. T. Tran, A. T. Pham, and T. Cong Thang. A New Adaptation Approach for Viewport-adaptive 360-degree Video Streaming. In *IEEE Int. Symp. on Multimedia (ISM)*, 2017.

[74] D. V. Nguyen, H. T. T. Tran, and T. C. Thang. Impact of delays on 360-degree video communications. In *2017 TRON Symposium (TRONSHOW)*, 2017.

[75] M. Nguyen, D. H. Nguyen, C. T. Pham, N. P. Ngoc, D. V. Nguyen, and T. Cong Thang. An adaptive streaming method of 360 videos over HTTP/2 protocol. In *Int. Conf. on Information and Computer Science*, 2017.

[76] T. C. Nguyen and J. Yun. Predictive tile selection for 360-degree vr video streaming in bandwidth-limited networks. *IEEE Communications Letters*, 22(9):1858–1861, Sept 2018.

[77] O. A. Niamut, E. Thomas, L. D'Acunto, C. Concolato, F. Denoual, and S. Y. Lim. MPEG DASH SRD: spatial relationship description.

[78] J. Park and K. Chung. Client-side rate adaptation scheme for HTTP adaptive streaming based on playout buffer model. In *2016 International Conference on Information Networking, ICOIN Malaysia*, pages 190–194, 2016.

[79] R. Pastrana-Vidal and J. Gicquel. Automatic quality assessment of video fluidity impairment

using a no-reference metric. In *Proceedings of Workshop Video Process. Quality Metrics for Consumer Electron, VPQM*, 2006.

[80] R. Peon and H. Ruellan. HPACK: header compression for HTTP/2. *RFC*, 7541:1–55, 2015.

[81] S. Petrangeli, V. Swaminathan, M. Hosseini, and F. De Turck. An HTTP/2-Based Adaptive Streaming Framework for 360° Virtual Reality Videos. In *ACM Conf. on Multimedia (MM)*, 2017.

[82] B. Petry and J. Huber. Towards effective interaction with omnidirectional videos using immersive virtual reality headsets. In *Proceedings of the 6th Augmented Human International Conference*, 2015.

[83] D. Podborski, E. Thomas, M. Hannuksela, S. Oh, T. Stockhammer, , and S. Pham. Virtual reality and dash. In *Proceedings of the International Broadcasting Convention (IBC) Conference*, 2017.

[84] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *ACM MobiCom Workshop on All Things Cellular*, 2016.

[85] B. Ray, J. Jung, and M. Larabi. A low-complexity video encoder for equirectangular projected 360 video content. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, Canada*, 2018.

[86] G. Ribezzo, G. Samela, V. Palmisano, L. De Cicco, and S. Mascolo. A dash video streaming system for immersive contents. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys)*, 2018.

[87] A. Samba, Y. Busnel, A. Blanc, P. Dooze, and G. Simon. Instantaneous throughput prediction in cellular networks: Which information is needed? In *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 624–627, 2017.

[88] Y. Sanchez, R. Skupin, and T. Schierl. Compressed domain video processing for tile based panoramic streaming using hevc. In *IEEE International Conference on Image Processing (ICIP)*, Sept 2015.

[89] 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 and Tutorials*, 17(1):469–492, 2015.

[90] A. Sobhani, A. Yassine, and S. Shirmohammadi. A video bitrate adaptation and prediction mechanism for http adaptive streaming. *ACM Trans. Multimedia Comput. Commun. Appl.*, 13(2), 2017.

[91] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. BOLA: near-optimal bitrate adaptation for online videos. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM, USA*, pages 1–9, 2016.

[92] K. Spiteri, R. K. 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, MMSys, Amsterdam*, pages 123–137, 2018.

[93] Steam. Steam hardware & software survey, September 2018.

[94] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Trans. Circuits Syst. Video Techn.*, 2012.

[95] V. Swaminathan and S. Wei. Low latency live video streaming using HTTP chunked encoding. In *IEEE International Workshop on Multimedia Signal Processing MMSP*, pages 1–6, 2011.

[96] A. M. Tekalp. *Digital Video Processing*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2nd edition, 2015.

[97] T. C. Thang, Q. Ho, J. W. Kang, and A. T. Pham. Adaptive streaming of audiovisual content using MPEG DASH. *IEEE Trans. Consumer Electronics*, 58(1):78–85, 2012.

[98] TiledMedia. Streaming uhd-quality vr at realistic bitrates: Mission impossible?, May 2017.

[99] C. Timmerer and A. Bertoni. Advanced transport options for the dynamic adaptive streaming over HTTP. *CoRR*, abs/1606.00264, 2016.

[100] Todd Hoff. How facebook live streams to 800,000 simultaneous viewers. High Scalability, June 2016. goo.gl/MhvSfZ.

[101] Y. Wang, Y. Li, D. Yang, and Z. Chen. A fast intra prediction algorithm for 360-degree equirectangular panoramic video. In *2017 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4, Dec 2017.

[102] Y. Wang, Y. Li, D. Yang, and Z. Chen. A fast intra prediction algorithm for 360-degree equirectangular panoramic video. In *IEEE Visual Communications and Image Processing, VCIP, USA*, 2017.

[103] S. Wei, V. Swaminathan, and M. Xiao. Power efficient mobile video streaming using HTTP/2 server push. In *IEEE International Workshop on Multimedia Signal Processing, MMSP*, pages 1–6, 2015.

[104] S. Wei, V. Swaminathan, and M. Xiao. Power efficient mobile video streaming using http/2 server push. In *ACM MMSP*, 2015.

[105] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Trans. Circuits Syst. Video Techn.*, 13(7):560–576, 2003.

[106] M. Wijnants, R. Marx, P. Quax, and W. Lamotte. HTTP/2 Prioritization and Its Impact on Web Performance. In *ACM World Wide Web (WWW) Conf.*, 2018.

[107] M. Xiao, V. Swaminathan, S. Wei, and S. Chen. Evaluating and improving push based video streaming with HTTP/2. In *Proceedings of the International Workshop on Network*

and Operating Systems Support for Digital Audio and Video, NOSSDAV, pages 3:1–3:6, 2016.

[108] L. Xie, X. Zhang, and Z. Guo. CLS: A cross-user learning based system for improving qoe in 360-degree video adaptive streaming. In *2018 ACM Multimedia Conference on Multimedia Conference, MM, Republic of Korea,*, pages 564–572, 2018.

[109] Xiph.org. Xiph.org test media, 1994-2016. https://media.xiph.org/video/derf/.

[110] Y. Xu, Y. Dong, J. Wu, Z. Sun, Z. Shi, J. Yu, and S. Gao. Gaze Prediction in Dynamic 360 Immersive Videos. In *IEEE Conf. on Comp. Vision and Pattern Recog.*, 2018.

[111] M. C. Yu, H. Lakshman, and B. Girod. A framework to evaluate omnidirectional video coding schemes. In *IEEE Int. Symp. on Mixed and Augmented Reality*, 2015.

[112] A. Zare, A. Aminlou, and M. M. Hannuksela. 6k effective resolution with 4k HEVC decoding capability for omaf-compliant 360° video streaming. In *Proceedings of the 23rd Packet Video Workshop*, 2018.

[113] T. Zhou, H. Zhang, M. Xu, and Y. Chen. Throughput prediction-based rate adaptation for real-time video streaming over uavs networks. In *Wireless Algorithms, Systems, and Applications - 9th International Conference, WASA Proceedings*, 2014.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Solutions de Transmission Vidéo avec Faible Latence Basées sur HTTP/2

Mots clés : Diffusion à faible latence, HTTP/2, vidéos immersives

Résumé : Les techniques adaptatives de transmission vidéo s'appuient sur un contenu qui est encodé à différents niveaux de qualité et divisé en segments temporels. Avant de télécharger un segment, le client exécute un algorithme d'adaptation pour décider le meilleur niveau de qualité à considérer. Selon les services, ce niveau de qualité doit correspondre aux ressources réseaux disponibles, mais aussi à d'autres éléments comme le mouvement de tête d'un utilisateur regardant une vidéo immersive (à 360°) afin de maximiser la qualité de la portion de la vidéo qui est regardée. L'efficacité de l'algorithme d'adaptation a un impact direct sur la qualité de l'expérience finale. En cas de mauvaise sélection de segment, un client HTTP/1 doit attendre le téléchargement du prochain segment afin de choisir une qualité appropriée. Dans cette thèse, nous proposons d'utiliser le protocole HTTP/2 pour remédier à ce problème. Tout d'abord, nous nous focalisons sur le service de vidéo en direct. Nous concevons une stratégie de rejet d'images vidéo quand la bande passante est très variable afin d'éviter les arrêts fréquents de la lecture vidéo et l'accumulation des retards. Le client doit demander chaque image vidéo dans un flux HTTP/2 dédié pour contrôler la livraison des images par appel aux fonctionnalités HTTP/2 au niveau des flux concernées. Ensuite, nous optimisons la livraison des vidéos immersives en bénéficiant de l'amélioration de la prédiction des mouvements de têtes de l'utilisateur grâce aux fonctionnalités d'initialisation et de priorité de HTTP/2. Les résultats montrent que HTTP/2 permet d'optimiser l'utilisation des ressources réseaux et de s'adapter aux latences exigées par chaque service.

Title : Low Latency Video Streaming Solutions based on HTTP/2

Keywords : low-latency streaming, HTTP/2, video delivery, immersive video

Abstract : Adaptive video streaming techniques enable the delivery of content that is encoded at various levels of quality and split into temporal segments. Before downloading a segment, the client runs an adaptation algorithm to determine the level of quality that best matches the network resources. For immersive video streaming this adaptation mechanism should also consider the head movement of a user watching the 360° video to maximize the quality of the viewed portion. However, this adaptation may suffer from errors, which impact the end user's quality of experience. In this case, an HTTP/1 client must wait for the download of the next segment to choose a suitable quality. In this thesis, we propose to use the HTTP/2 protocol instead to address this problem. First, we focus live streaming video. We design a strategy to discard video frames when the bandwidth is very variable in order so as to avoid the rebuffering events and the accumulation of delays. The customer requests each video frame in an HTTP/2 stream which allows to control the delivery of frames by leveraging the HTTP/2 features at the level of the dedicated stream. Besides, we use the priority and reset stream features of HTTP/2 to optimize the delivery of immersive videos. We propose a strategy to benefit from the improvement of the user's head movements prediction overtime. The results show that HTTP/2 allows to optimize the use of network resources and to adapt to the latencies required by each service.