Aflevering van adaptieve videodiensten met beperkte vertraging

Low-Latency Delivery of Adaptive Video Streaming Services

Jeroen van der Hooft

Promotoren: prof. dr. ir. F. De Turck, dr. ir. T. Wauters
Proefschrift ingediend tot het behalen van de graad van
Doctor in de ingenieurswetenschappen: computerwetenschappen

UNIVERSITEIT
GENT

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. B. Dhoedt
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2018 - 2019

ISBN 978-94-6355-243-1
NUR 986, 988
Wettelijk depot: D/2019/10.500/51

Ghent University
Faculty of Engineering and Architecture
Department of Information Technology

imec
Internet Technology and Data Science Lab

Examination Board:

prof. dr. ir. Filip De Turck	(advisor)
dr. ir. Tim Wauters	(advisor)
prof. dr. ir. Ronny Verhoeven	(chair)
prof. dr. Peter Lambert	(secretary)
prof. dr. ir. Danny De Vleeschauwer	
prof. dr. Jeroen Famaey	
prof. dr. Christian Timmerer	
dr. ir. Jürgen Slowack	

Supported by grant of Flanders
Innovation & Entrepreneurship

Dissertation for acquiring the degree of
Doctor of Computer Science Engineering

Dankwoord

"One, two, three four, five, hunt the hare and turn her down the rocky road, all the way to Dublin, whack follol de rah!"

—D. K. Gavan, as performed by The Dubliners, 1976

"De tijd gaat snel, gebruik haar wel", vertelde Rachelle mijn zus en mij steevast toen we klein waren. Niks is minder waar gebleken: bijna vijf jaar geleden is het alweer dat ik voor het eerst kennis mocht maken met het leven van een doctoraatsstudent. Hoewel het niet altijd even makkelijk was - getuige daarvan de elfendertig deadlines in de aanloop naar het schrijven van dit dankwoord - houd ik vooral positieve herinneringen over aan deze ervaring. Ik kan terugkijken op een fantastisch avontuur, waarin ik zowel op wetenschappelijk als op persoonlijk vlak gegroeid ben. Dit heb ik te danken aan de vele mensen met wie ik me de afgelopen jaren omringd zag, die ik langs deze weg even in de bloemen wil zetten.

In de eerste plaats wil ik mijn promotoren bedanken. Filip, meer dan zeven jaar geleden kwam ik voor de eerste keer met je in contact om een zomerstage te regelen. Sindsdien kon ik steeds bij je terecht voor het nodige advies of een leuk gesprek, en zelfs om hulp te vragen toen ik in het verre Oostenrijk mijn treinticket (weer eens) kwijtgespeeld had. Bedankt voor de aangename samenwerking doorheen de jaren, en om me een voorbeeld te geven om naar op te kijken. Tim, waar ik bij anderen op de deur kwam kloppen, bleef die van jou - letterlijk en figuurlijk - altijd openstaan. Waar een berichtje via Mattermost vaak volstaan had, heb ik hier meer dan eens van geprofiteerd om je om raad te vragen of om even te klagen over zaken waar weinig over te klagen valt. Bedankt voor je geduld en je nuchtere kijk op de zaken, en voor de vele tafeltennisgerelateerde gesprekken die we doorheen de jaren gevoerd hebben.

Mijn Italiaanse compagnon en favoriete co-auteur moet ook bij naam genoemd worden. Stefano, man, I don't know where to begin. Maria once told me that "wherever you go, there's always an Italian". I'm certain there is truth in that, and I'm glad you turned out to be the one I met in Ghent. Thanks for your insights and advise, for the many conversations we had throughout the years and for the awesome trips we shared. I've said it five years ago and I'll say it again today: *grazie mille!*

Ook Jeroen en Maria ben ik veel dank verschuldigd. Jeroen, bedankt om me te begeleiden tijdens mijn masterproef en tijdens het voorbereiden van mijn beursaanvraag. Na je vertrek was het even aanpassen, maar voor deze zaken blijf ik je enorm dankbaar. Maria, thanks for the amazing work together in the last year. I really appreciated your contributions and suggestions, not in the least the ones related to the dissertation you're reading now. *¡Muchas gracias!*

Doorheen mijn doctoraat mocht ik bureaus 2.22 en 200.026 delen met verschillende toffe collega's: Bert, Cedric, Elias, Harald, Jacob, Joachim, Jolien, Lander, Matthias, Ozan, Pieter, Pieter-Jan, Prashant, Rein, Sam, Steven B., Steven V. C., Tim en Xander. Een lunchpauze in 't Blauw Kotje of de Frietketel was nooit ver weg, en ook de TGIF's (met of zonder toastje met zalm) zijn en blijven onvergetelijk. Ook andere collega's wil ik graag bij naam noemen, omwille van de papers waar we samen aan werkten of de verschillende practica die we samen begeleidden (maar stiekem toch vooral omwille van de talloze gesprekken die we 's middags voerden over onze favoriete series en guilty pleasures): Andy, Bart, Bram, Bruno, Chris, Christof, Dries, Eric, Femke D. B., Femke O., Gilles, Gregory, Helen, Hemanth, Hendrik, Ilja, Jelle, Johannes, Jonas, Joris, José, Laurens, Leandro, Mario, Mathias, Maxim, Merlijn, Niels, Philip, Pieter B., Pieter S., Rafael, Stijn, Thilo, Thomas D., Thomas V., Wannes en Wim. Zonder jullie zouden mijn werkdagen niet geweest zijn wat ze waren, bedankt daarvoor!

IDLab zou IDLab niet zijn zonder de uitstekende omkadering waarvan ze geniet. Piet weet onze onderzoeksgroep op uitstekende wijze te leiden, en ik zal nooit vergeten dat hij me toestond met de taxi naar het werk te komen toen ik zes weken lang onbeholpen op krukken rondliep. Voor administratieve zaken kon ik steeds terecht bij Martine en Davinia, die me met een lach op het gezicht met raad en daad bijstonden. Bedankt voor alle hulp doorheen de jaren, en voor de leuke gesprekken die hier vaak het gevolg van waren. Ook bedankt aan Bernadette, Joke en Karen, die me op organisatorisch en financieel vlak steeds verderhielpen waar nodig. Sabrina wil ik graag bedanken om onze werkomgeving zorgvuldig proper te houden, en om iedereen 's ochtends zo warm en enthousiast te onthalen. Ook een shout-out naar ons A-team kan niet ontbreken, aangezien ik steeds op Bert, Brecht, Joeri, Simon en Vincent kon rekenen wanneer ik mijn laptop weer eens om zeep geholpen had.

Naast de collega's bij IDLab wil ik ook de medewerkers van het Agentschap voor Innoveren en Ondernemen (VLAIO) bedanken voor de onderzoeksbeurs die ik mocht ontvangen. Deze liet me toe om gedurende vier jaar mijn onderzoek voort te zetten, zonder me zorgen te moeten maken over contractverlengingen en andere financiële zaken. Many thanks also go to Christian and Hermann, for receiving me so well at the Alpen-Adria Universität in Klagenfurt. I really enjoyed working together, and hope we can continue to do so in the future. Let me know when you're visiting Ghent, and I'll be glad to show you around.

Naast de ondersteuning op het werk, kon ik door de jaren heen steeds terugvallen op mijn familie. Het leven is niet altijd makkelijk geweest, mama, maar dat weerhield je er niet van om er steeds voor Erinke en mij te zijn. Terugblikkend op de voorbije tien jaar, denk ik dat ik mag zeggen dat je het er fantastisch vanaf gebracht hebt. Een niet onbelangrijk aandeel gaat hierbij naar jou, Jurgen, voor de zorgen die je mama ontnomen (en af en toe wel eens bezorgd) hebt. Ik ben blij dat jullie in Assenede jullie stekje gevonden hebben. Ook jij bent er altijd voor mij geweest, zus, en hebt het inmiddels uitstekend gedaan. De eerste stappen naar een nieuwe dierentuin in Sint-Lievens-Houtem zijn gezet, en ik kijk ernaar uit om de groei ervan in de toekomst mee te mogen maken. Of Quentin hier even enthousiast over is, zal nog moeten blijken, maar ik heb er vertrouwen in dat mijn schoonbroer je zal ondersteunen in alles wat je doet.

Familie kan je zijn, maar je kan het ook worden. Getuige hiervan zijn Nancy en Geert, die me de afgelopen jaren bijgestaan hebben in al mijn ondernemingen. Zonder jullie had ik niet gestaan waar ik nu sta, en ik ben jullie daar dan ook oprecht dankbaar voor. Een hoogtepunt voor mij was toch het huwelijk van Jolien en Pieter, waarbij ik met Nadine, Camila en Fu het mooie weer mocht maken achter het stuur van jullie wagen. Het was mooi dat ik deze rol op mij kon nemen, aan het begin van de mooie toekomst die het jonge bruidspaar ongetwijfeld wacht. Jona, onze film- en (niet-)pizzamomenten zal ik nooit vergeten. Ruziën met jou is een voorrecht dat niemand ooit zal begrijpen, maar ik ben blij dat we het nog niet verleerd zijn. Take good care of her, Sabya, and don't forget the kittens!

De voorbije jaren heb ik ook beter kennis mogen maken met mijn familie uit het verre Nederland. Het was mooi om de laatste jaren van opa's leven met hem te kunnen delen, ook al blijft er sinds januari een gevoel van gemis achter. Ik ben er zeker van dat hij trots geweest zou zijn, en dat hij - jury of niet - op de eerste rij had willen zitten. Ik kijk uit naar de vele momenten met mijn neefjes en nichtjes die nog mogen komen, en naar de volgende wedstrijd in de Johan Cruijff ArenA. Ook de andere tak van de familie heeft veel voor me betekend: de kerstfeestjes in Veurne waren telkens iets om naar uit te kijken en vormen een blijvende herinnering aan wie oma en opa waren. De familiefeesten met de Meulenaeres en de Pittillions zijn steeds een ervaring, temeer omdat ik zo mijn tante, nonkel en de resterende kwartjes even terug kan zien. Afspraak op 15 augustus?

Inspanning kan niet bestaan zonder ontspanning. Op dat vlak ben ik de voorbije jaren verwend geweest, met een hele resem aan activiteiten die ik heb kunnen ondernemen. Ik blik met plezier terug op de voetbalwedstrijden met Marieke en Axelle, op de fiets- en trektochten met Isaac, op de optredens met Joren en op de vele Xbox-avonden met Achim en Verelst. Ook de weekends met de CW en de etentjes met onze klas uit Strobrugge, die beide inmiddels een echte traditie geworden zijn, kunnen in dit lijstje niet ontbreken. Het zijn alle activiteiten waar ik heel wat plezier aan beleefd heb, en ik hoop dit in de toekomst ook te mogen blijven doen.

Om inspanning en ontspanning te combineren, kon ik steeds terecht in onze tafeltennisclub. Door de jaren heen heb ik zoveel mensen mogen leren kennen: de jonge garde die nu onze A-ploeg vormt, de iets minder jonge garde die reeds lang mijn ploegmaten zijn in de B- en C-ploeg, mijn eigen persoonlijke trainer en telkens weer nieuwe, jonge spelers in onze jeugdreeksen. Het hele tafeltennisgebeuren boeit me inmiddels al achttien jaar, en ik lijk het maar niet beu te worden. Dat hoeft niet te verwonderen, gezien de vele spannende wedstrijden en toernooien, de leuke kaartavonden en de (soms iets minder leuke) bestuursvergaderingen. Een speciaal woord van dank gaat uit naar Jan, van wie ik zoveel geleerd heb en aan wie ik tot op vandaag een voorbeeld neem.

Het leven zou niet hetzelfde geweest zijn zonder mijn twee fantastische kotgenoten, met wie ik de voorbije vijf jaar een appartement heb mogen delen. Samen barbecueën, poolen of gewoon een serie kijken zijn herinneringen die ik mijn leven lang zal koesteren. Bert en Sigy, ik wens jullie het allerbeste toe in jullie nieuwe appartement en hoop dat we elkaar nog regelmatig mogen zien (er komt een derde seizoen van Derry Girls!). Klaas, met jou kijk ik voorlopig uit naar het moment waarop we de duurste fles champagne kraken die we kunnen vinden, om te vieren dat je overwonnen hebt wat niemand ooit zou moeten overwinnen. Ik wens je een spoedig herstel en een prachtige toekomst toe.

"Save the best for last", zeggen ze wel eens, en in dit geval zit daar zeker waarheid in. Ik had je naam hierboven al verschillende keren kunnen laten vallen, Thomas, want jij zat naast me tijdens die voetbalwedstrijden, hebt meegedeeld in de reizen en trektochten die ik ondernam, en bent en blijft mijn favoriete Xbox-partner. Je kent me na al die jaren door en door, en ik ben blij dat ik jou mijn maat mag noemen. Dat ik door jou mensen als je ouders en je zus heb leren kennen, maakt die band alleen maar sterker. Ik kijk uit naar wat de toekomst mag brengen, en niet het minst naar het moment waarop je met Laura eindelijk die volgende stap mag zetten.

Gent, juni 2019
Jeroen van der Hooft

Table of Contents

List of Figures

List of Tables

List of Acronyms

0-9

3G	3rd Generation
4G	4th Generation
6DoF	Six Dimensions of Freedom

A

AVC	Advanced Video Coding

B

BOLA	Buffer Occupancy-based Lyapunov Algorithm

C

CSS	Cascading Style Sheets
CDN	Content Delivery Network
CMAF	Common Media Application Format
CRF	Constant Rate Factor
CTF	Center Tile First

D

DANE	DASH-Aware Network Elements
DASH	Dynamic Adaptive Streaming over HTTP
DiffServ	Differentiated Services

E

EMWA	Exponentially Weighted Moving Average

F

FDSP	Flexible Dual TCP-UDP Streaming Protocol
FEC	Forward Error Correction
FESTIVE	Fair Efficient and Stable adapTIVE
FINEAS	Fair In-Network Enhanced Adaptive Streaming
FPS	Frames per Second

G

GOP	Group of Pictures

H

HAS	HTTP Adaptive Streaming
HEVC	High Efficiency Video Coding
HM	HEVC Test Model
HMD	Head-Mounted Display
HSDPA	High Speed Downlink Packet Access
HSPA	High Speed Packet Access
HSPA+	Evolved High Speed Packet Access
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol

I

IDR	Instantaneous Decoder Refresh
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IPTV	Integer Linear Program
IPTV	Internet Protocol Television
ISO	International Organization for Standardization

L

LOLYPOP	LOw-LatencY PredictiOn-based adaPtation
LTE	Long-Term Evolution

M

MANE	Media Aware Network Elements
MPD	Media Presentation Description
MPEG	Moving Picture Experts Group
MSS	Microsoft Smooth Streaming

N

NAT	Network Address Translation
NLP	Natural Language Processing

P

PCN	Pre-Congestion Notification
PSNR	Peak Signal-to-Noise Ratio

Q

QUIC	Quick UDP Internet Connections
QoE	Quality of Experience
QoS	Quality of Service
QP	Quantization Parameter

R

RAM	Random Access Memory
RFC	Request for Comments
RGB	Red, Green, Blue

RTP	Real-Time Protocol
RTSP	Real-Time Streaming Protocol
RTT	Round-Trip Time

S

SAND	Server and Network Assisted DASH
SARSA	State-Action-Reward-State-Action
SDN	Software-Defined Networking
SHVC	Scalable High Efficiency Video Coding
SSIM	Structural SIMilarity index
SVC	Scalable Video Coding

T

TCP	Transmission Control Protocol

U

UDP	User Datagram Protocol
URL	Uniform Resource Locator
UVP	Uniform Viewport

V

VoD	Video on Demand
VR	Virtual Reality
VRT	Vlaamse Radio en Televisie
V-PCC	Video Point Cloud Compression

W

WebRTC	Web Real-Time Communication

Samenvatting
– Summary in Dutch –

Sinds de verzending van het eerste bericht via een pakketgebaseerd netwerk, in oktober 1969, is efficiënte aflevering van pakketten, media en diensten steeds belangrijker geworden. Hoewel de onderzoeksinspanningen zich aanvankelijk vooral richtten op het delen van berichten en tekstgebaseerde bestanden, is video streaming in snel tempo de meest dominante toepassing op het internet geworden. Momenteel genereren service providers zoals Netflix en YouTube meer dan 75% van het downstreamverkeer in de Verenigde Staten, een cijfer dat de komende jaren naar verwachting tot 82% zal groeien. Om de media aan de eindgebruiker te bezorgen, wordt over het algemeen HTTP adaptieve streaming (HAS) gebruikt. In HAS wordt de inhoud gecodeerd met behulp van verschillende kwaliteitsrepresentaties, tijdelijk gesegmenteerd en opgeslagen op een of meerdere servers binnen een content delivery network. De duur van het segment ligt over het algemeen tussen een tot tien seconden, afhankelijk van de beschouwde use case. De HAS-server beheert een manifestbestand dat metadata bevat, die onder meer verschillende aanpassingssets beschrijft waaraan een type media is toegewezen, zoals video, audio en ondertitels, en de beschikbare kwaliteitsrepresentaties bijhoudt. Op basis van de waargenomen netwerkomstandigheden, de apparaatkenmerken en de voorkeuren van de gebruiker, kan de client vervolgens beslissen over de kwaliteit van elk van deze segmenten. Door de mogelijkheid om de videokwaliteit aan te passen, vermijdt deze methode op actieve wijze buffer starvation en resulteert daarom in een soepelere weergave en een hogere Quality of Experience (QoE) voor de eindgebruiker.

Ondanks het vermogen van HAS-clients om zich aan hun omgeving aan te passen, blijven sommige uitdagingen onopgelost. In de eerste plaats richten oplossingen zich vaak op de Quality of Service, terwijl de ware zorg bij de QoE van de gebruiker hoort te liggen: de prestaties van een systeem moeten worden gemeten op basis van hoe de gebruiker de streamingsessie ervaart, wat afhangt van factoren zoals de videokwaliteit, het optreden van playout freezes, de opstarttijd, etc. Ten tweede introduceren videostreamingoplossingen vertraging door de opname van beelden, videocodering, opslag op de server, netwerkvertraging en buffering aan de

clientzijde. In de afgelopen jaren is interactiviteit met beperkte vertraging evenwel steeds belangrijker geworden: met een steeds groter aanbod willen aanbieders de gebruiker in staat stellen snel tussen video's te bladeren, video's op te starten en te stoppen wanneer dat nodig is. Verder worden steeds interactievere applicaties en use cases overwogen voor video streaming, zoals 360° video. Dergelijke use cases vereisen snelle reactie van de client op veranderingen in het netwerk, of op de positie van de gebruiker binnen de video: bij een plotselinge beweging van het hoofd kan de gebruiker geconfronteerd worden met lage kwaliteit, omdat de client een dergelijke actie niet geanticipeerd had. Ten derde moeten services worden afgestemd op de eindgebruiker. Vele service providers gebruiken aanbevelingssystemen om relevante inhoud te suggereren, gebaseerd op de geschiedenis van de gebruiker. Hierdoor kan de provider relevante suggesties doen voor nieuwe inhoud, wat resulteert in lagere zoektijden en een hogere klanttevredenheid. Verder is personalisatie vereist om diensten in staat te stellen zich aan te passen aan de manier waarop gebruikers omgaan met de video. Sommige gebruikers kunnen bijvoorbeeld op hoge snelheid bewegen in een immersieve video, terwijl andere weinig bewegen en zich richten op een specifieke regio in de video. Ten slotte moeten diensten voldoende aanpassing bieden, niet alleen aan het netwerk maar ook aan beweging van de gebruiker, de beschouwde video, etc. Vooral in het geval van immersieve video, waarbij ruimtelijke segmentatie gebruikt wordt om de kwaliteit van de video in de zichtbare regio te verhogen, moeten heuristieken voor het aanpassen van de kwaliteit rekening houden met een hele reeks factoren, en aanpassingen kunnen doen op een tijdsschaal in de orde van honderden milliseconden. In dit proefschrift zijn vier onderzoekshoofdstukken opgenomen die deze uitdagingen behandelen voor verschillende use cases.

In Hoofdstuk 2 proberen we de vertraging voor traditionele video streaming te beperken. We stellen een methode voor die een nieuwe HTTP feature gebruikt om de server toe te laten gegevens achtereenvolgens te verzenden, zonder dat de client een verzoek voor elk afzonderlijk segment hoeft te sturen. Door deze methode te combineren met segmenten met een duur lager dan één seconde (*superkorte segmenten*), is het mogelijk om de opstarttijd en de end-to-end vertraging in HAS live streaming aanzienlijk te verlagen. We evalueren eerst de coderingsoverhead voor verschillende segmentduren en laten zien dat de resulterende waarden afhangen van de beschouwde video en frame rate. Vervolgens gebruiken we de meest geschikte segmentduur en evalueren we de pushgebaseerde methode in een multi-clientscenario met variabele bandbreedte en vertraging, en laten we zien dat de opstarttijd kan worden verminderd met 31.2% in vergelijking met traditionele oplossingen via HTTP/1.1 in mobiele netwerken met hoge vertraging. Bovendien kan de end-to-end vertraging in live streaming aanzienlijk worden gereduceerd, terwijl de video aan dezelfde kwaliteit wordt aangeboden.

In Hoofdstuk 3 concentreren we ons op op nieuwsgebaseerde portals, die grote hoeveelheden video aanbieden om nieuwsartikels te begeleiden. Om de betrokkenheid van gebruikers tijdens het zoeken door een video of browsen tussen video's te stimuleren, is het verminderen van de opstarttijd steeds belangrijker geworden: terwijl de huidige laadtijd typisch in de orde van seconden ligt, heeft onderzoek aangetoond dat gebruikers maximaal twee seconden mogen wachten om een acceptabele QoE te bereiken. In dit hoofdstuk stellen we vier complementaire componenten voor die geoptimaliseerd en geïntegreerd werden in een uitgebreid raamwerk voor de aflevering van nieuwsgerelateerde video met beperkte vertraging: *(i)* serverside codering met korte videosegmenten; *(ii)* HTTP/2 server push op de applicatielaag; *(iii)* server-side gebruikersprofilering om relevante inhoud voor een bepaalde gebruiker te identificeren; en *(iv)* opslag aan clientzijde om proactief geleverde inhoud te bewaren. Aan de hand van een grote dataset van een Belgische nieuwsaanbieder, die miljoenen requests voor tekst- en videogebaseerde artikels bevat, laten we zien dat het voorgestelde raamwerk de opstarttijd van de video's in verschillende scenario's met meer dan de helft vermindert, waardoor de gebruikersinteractie verbetert en er sneller door de beschikbare video's gebladerd kan worden.

In Hoofdstuk 4 verleggen we onze focus naar immersieve 360° video. De meeste providers bieden diensten voor virtual reality aan door middel van een tweedimensionale weergave van de video, gecombineerd met traditionele technieken voor video streaming. Omdat slechts een beperkt deel van de video (i.e., de viewport) door de gebruiker wordt bekeken, wordt de beschikbare bandbreedte niet optimaal gebruikt. In dit hoofdstuk bespreken we daarom de voordelen van ruimtelijke segmentatie van de video, wat resulteert in meerdere regio's binnen de video, die ook wel tegels genoemd worden. Met behulp van deze methode kan de kwaliteit van elk van deze tegels worden aangepast aan de netwerkkenmerken en de focus van de gebruiker, wat leidt tot een hogere QoE. Om zijn volledige potentieel te benutten, stellen we een intuïtieve methode voor om de beweging van het hofd van de gebruiker te voorspellen, en bespreken we twee heuristieken voor het aanpassen van de kwaliteit die rekening houden met de geïntroduceerde ruimtelijke dimensie. Verder introduceren we een nieuwe feedback loop aan clientzijde, waardoor kwaliteitsbeslissingen gewijzigd kunnen worden tijdens het downloaden van de vereiste tegels voor een bepaald videosegment, en bespreken we de voordelen van HTTP/2 server push voor de aflevering van deze bestanden. Aan de hand van een uitgebreide evaluatie tonen we aan dat de voorgestelde methoden resulteren in een aanzienlijke verbetering van de videokwaliteit, in vergelijking met state-of-the-art oplossingen.

In Hoofdstuk 5 concentreren we ons op de aflevering van volumetrische media voor immersieve video met zes vrijheidsgraden. Een manier om dit te realiseren is door objecten vast te leggen via een aantal camera's die in verschillende hoeken zijn geplaatst, en op die manier toelaten om

een een zogeheten point cloud te genereren die bestaat uit de locatie van een groot aantal punten in de driedimensionale ruimte, en hun overeenkomstige kleur. Het resulterende object kan vervolgens vanuit elke hoek worden gereconstrueerd, zodat de gebruiker vrij kan bewegen in de immersieve video. In dit hoofdstuk stellen we PCC-DASH voor, een methode voor het streamen van scènes met meerdere, dynamische point cloud objecten. We presenteren verschillende heuristieken voor het aanpassen van de kwaliteit, die informatie over de positie en focus van de gebruiker, de beschikbare bandbreedte en de bufferstatus van de client gebruiken om te beslissen over de meest geschikte kwaliteitsrepresentatie van elk object. Aan de hand van een geëmuleerde evaluatie bespreken we de voor- en nadelen van elke oplossing en belichten we potentiële uitbreidingen voor toekomstig werk.

De voorgestelde benaderingen pakken belangrijke uitdagingen aan bij het leveren van adaptieve videodiensten, wat resulteert in een beperkte vertraging en een verbeterde aanpassing van de kwaliteit. Dit proefschrift is evenwel slechts een eerste stap naar een effectieve aflevering van de diensten van morgen. Hoewel de ontwikkelingen op het vlak van video streaming de afgelopen jaren aanzienlijk zijn toegenomen, voorspellen we dat de hoeveelheid onderzoek naar steeds veeleisendere toepassingen ook in de toekomst zal blijven groeien. Toepassingen voor immersieve video en de intuïtieve bediening van apparaten op afstand zullen de technologie verder stimuleren en nieuwe uitdagingen voor service providers introduceren. We kunnen dus stellen dat ons werk nog lang niet af is.

Summary

Ever since the transmission of the first message over a packet-based network, back in October 1969, efficient delivery of packets, content and services has been of crucial importance. While research efforts initially mainly focused on sharing messages and text-based files, video streaming has quickly become the most dominant application on the Internet. Currently, content delivery providers such as Netflix and YouTube generate more than 75% of downstream traffic in the United States, a number which is expected to grow to 82% in the next few years. To deliver the content to the end user, HTTP adaptive streaming (HAS) is generally used. In HAS, the content is encoded using several quality representations, temporally segmented and stored on one or multiple servers within a content delivery network. The segment duration is generally between one to ten seconds, depending on the considered use case. A manifest file is maintained by the HAS server, which contains metadata describing, among others, different adaptation sets (each of which is attributed a type of media, such as video, audio and subtitles), and the available quality representations. Based on the perceived network conditions, the device characteristics, and the user's preferences, the client can then decide on the quality of each of these segments. Having the ability to adapt the video quality, this approach actively avoids buffer starvation, and therefore results in smoother playback and a higher Quality of Experience (QoE) for the end user.

Despite the ability of HAS clients to adapt to their environment, some challenges remain unsolved. First, solutions often focus on the Quality of Service (QoS), while the true concern should be with the user's QoE: the performance of a system should be measured based on how the user experiences the video streaming session, which depends on factors such as the video quality, the occurrence of playout freezes, the startup time, etc. Second, video streaming solutions introduce latency through camera capture, video encoding, server-side storage, network latency and client-side buffering. Over the last years, however, low-latency interaction has however become more and more important: with an ever-increasing supply of content, providers want to enable the user to quickly browse between videos, starting and stopping playout whenever required. Furthermore, more interactive applications and use cases are now considered for video streaming, such as 360° video. Such use cases requires fast reaction of the

client to changes in the network, or to the user's focus within the video: when making a sudden head movement, the user might be confronted with low-quality content because the client had not anticipated this action. Third, services need to be fine-tuned and personalized towards the user. Many service providers use recommendation systems to suggest relevant content, based on the user's history. This allows the provider to make relevant suggestions for new content, which results in lower search times and higher customer satisfaction. Furthermore, personalization is required to allow services to adapt to the way users interact with the content. Some users might, for instance, move their head at high speeds within an immersive video, while others focus on a specific region. Finally, services need to provide sufficient adaptation, not only to the network but also to user movement, considered video content etc. Especially in the case of immersive video, where spatial segmentation is often considered to increase the video quality of the visible region, rate adaptation heuristics need to take into account a whole range of factors, adapting to changes on a time scale of hundreds of milliseconds. In this dissertation, four research chapters are included which address these challenges for different use cases.

In Chapter 2, we attempt to reduce the latency for traditional video streaming solutions. We propose to use a novel HTTP feature that is able to send data back-to-back, without a need for the client to send a request for each individual segment. Combining this approach with segments of sub-second duration, referred to as *super-short* segments, it is possible to significantly reduce the startup time and end-to-end delay in HAS live streaming. We first evaluate the encoding overhead for different segment durations, and show that resulting values depend on the considered video content and frame rate. Then, using the most appropriate segment duration, we evaluate the push-based approach in a multi-client scenario with highly variable bandwidth and latency, and show that the startup time can be reduced with 31.2% compared to traditional solutions over HTTP/1.1 in mobile, high-latency networks. Furthermore, the end-to-end delay in live streaming scenarios can be reduced significantly, while providing the content at similar video quality.

In Chapter 3, we focus on news-based portals, which provide significant amounts of multimedia content to accompany news stories and articles. To stimulate user engagement with the provided content, such as searching through a video or browsing between videos, reducing the startup time has become more and more important: while the current median load time is in the order of seconds, research has shown that user waiting times must remain below two seconds to achieve an acceptable QoE. In this chapter, we propose four complementary components, which are optimized and integrated into a comprehensive framework for low-latency delivery of news-related video content: *(i)* server-side encoding with short video segments, *(ii)* HTTP/2 server push on the application layer, *(iii)* server-side user profiling to identify relevant content for a given

user, and *(iv)* client-side storage to hold proactively delivered content. Using a large dataset of a major Belgian news provider, containing millions of text- and video-based article requests, we show that the proposed framework reduces the videos' startup time in different mobile network scenarios by more than half, improving user interaction and allowing the user to quickly skim through available content.

In Chapter 4, we shift our focus to immersive media, and 360° video specifically. Most providers offer virtual reality streaming services through a two-dimensional representation of the immersive content, combined with traditional streaming techniques. However, since only a limited part of the video (i.e., the field-of-view or viewport) is watched by the user, the available bandwidth is not optimally used. In this chapter, we therefore discuss the advantages of spatial segmentation of the video, resulting in multiple regions or tiles. Using such approach, the quality of each of the resulting tiles can be adapted to the network characteristics and user movement, resulting in a higher QoE. To unlock its full potential, we propose an intuitive viewport prediction scheme, and two rate adaptation heuristics which take into account the newly introduced spatial dimension. Furthermore, we introduce a novel feedback loop at the client-side, which allows us to change quality decisions whilst downloading the required tiles for a given video segment, and discuss the advantages of HTTP/2 server push for content delivery. Through an extensive evaluation, we show that the proposed approaches result in a significant improvement in terms of video quality, compared to state-of-the-art solutions.

In Chapter 5, we focus on delivery of volumetric media for immersive video with six degrees of freedom. One way to realize this is by capturing objects through a number of cameras positioned in different angles, and creating a so-called point cloud which consists of the location and color of a significant number of points in the three-dimensional space. The corresponding object can then be reconstructed from every given angle, allowing the user to freely move around within the immersive video. In this chapter, we propose PCC-DASH, a standards-compliant means to adaptively stream scenes comprising multiple, dynamic point cloud objects. We present different rate adaptation heuristics which use information on the user's position and focus, the available bandwidth, and the client's buffer status to decide upon the most appropriate quality representation of each object. Through an extensive evaluation, we discuss the advantages and drawbacks of each solution, and highlight interesting paths for future work.

The proposed approaches address important challenges in the delivery of adaptive video streaming services, resulting in lower latency and improved quality adaptation. This dissertation is, however, only a first step towards effective delivery of tomorrow's video streaming solutions. We predict that, although the developments in the field of video streaming have increased considerably in the past few years, research for more de-

manding use cases will continue to grow. Immersive media and remote control, especially, will further push technology forward and continue to introduce new challenges for service providers. One can only conclude that our work is far from finished.

1 Introduction

"Remember who you are."

–The Lion King, 1994

1.1 A Brief History of Video Streaming

In 1994, Disney's *The Lion King* was first released. Young and old could go see the adventure of Simba in theater, and about one year later, buy a copy on a Video Home System (VHS) cassette. Now, about twenty five years later, VHS has mostly been replaced by the more recent digital versatile disc (DVD) and Blu-ray disc alternatives. Over the last decade, however, even their sales have gone down significantly [1]. The decline of traditional video media can mostly be attributed to the ever-increasing supply of content delivery services over the Internet. One notorious example is Netflix [1], the number one video service provider in the United States. This company, founded in 1997 by Reed Hastings and Marc Randolph, initially focused on renting and selling DVDs online, delivering all ordered goods by mail. By the year 2005, the company offered a total of 35 000 different films and shipped around one million DVDs per day. By this time, however, bandwidth speeds had improved significantly, opening up the possibility of delivering video over the Internet. While Netflix initially planned

[1] https://www.netflix.com

to deliver video using a so-called *Netflix box*, a piece of hardware which would download a movie overnight, the popularity of YouTube [2] (founded in 2005 and acquired by Google in 2006) and other streaming services made the company reconsider, and adopt a streaming concept instead. As of today, the company has 139 million paid subscriptions worldwide, and reports a netto profit of about $1.6 billion per year [2].

How did we get so far? To answer this question, we have to go back to 1969. In October of this year, Charley Kline, a student at the University of California in Los Angeles (UCLA), manages to send the first message over a packet-based network [3]. Although the system crashed during the attempt, the message "LO" (the first two letters of the intended "LOGIN") was successfully received on the other end. Over the next decade, more and more hosts would connect to the Advanced Research Projects Agency Network (ARPANET), until the network grew to a total of 213 connected computers. By 1982, the Internet protocol suite (TCP/IP) had been introduced as the standard networking protocol, which is still used today. By 1987, more than 20 000 hosts were connected to the Internet, and by 1991, the World Wide Web was introduced to the public. The first ever live video was broadcast in 1993, showing footage of *Severe Tire Damage*, a band consisting of employees of the DEC Systems Research Center, Xerox PARC and Apple Computer. A few months later, the first webcam was introduced, efficiently monitoring the status of the Trojan Room coffee pot in the University of Cambridge. As of 2018, more than 23 billion devices are connected to the Internet, generating an estimated total of 107 EB (107 million million million bytes) of data each month [4].

Throughout the years, bandwidth speeds increased significantly: while the first modems, developed by AT&T in 1962, offered a throughput of 300 b/s, speeds increased to 56 kb/s by 1986. Today, most developed countries report connection speeds well over 18 Mb/s. As these values increased over time, so did the bit rates of streamable videos. The first ever YouTube video, *Me at the Zoo*[3], came with a frame rate of 15 FPS and a resolution of 144p and 240p only, resulting in bit rates of 61 and 137 kb/s, respectively. Fourteen years later, 360° immersive videos – videos in which the user can look around by moving her head – can be streamed at 8K resolution, with bit rates well over 50 Mb/s.

How can we provide services that efficiently deal with these types of videos and bit rates? Generally, a distinction is made between Internet Protocol Television (IPTV) on the one hand, and over-the-top video streaming on the other. In IPTV, services are offered and managed by a network

[2] https://www.youtube.com
[3] https://www.youtube.com/watch?v=jNQXAC9IVRw

provider over a dedicated and managed network. This allows the optimization of services to suit network and end-device capabilities, so that the Quality of Service (QoS) can be guaranteed. Examples in Belgium include Telenet[4] and Proximus[5], which offer digital television by means of a set-top box, a video decoder in the resident's home. Over-the-top streaming services, however, use the best-effort Internet to deliver the content to the user. Although the QoS is not guaranteed, this approach is cheaper because no dedicated network elements are required. Furthermore, no hardware other than the playout device is needed, further reducing the cost and increasing the ease of use.

Throughout the years, multiple protocols and solutions for video streaming have been proposed. In the early days, most solutions focused on real time delivery, using e.g., the real time transport protocol (RTP) and the Real Time Streaming Protocol (RTSP) to transfer the content to the client. These technologies rely on the User Datagram Protocol (UDP), which achieves lower latency than the Transmission Control Protocol (TCP), because no acknowledgments are required and no retransmissions occur. Over time, however, most providers shifted to approaches based on the Hypertext Transfer Protocol (HTTP) combined with TCP. Using HTTP allows to reuse the existing optimized and scalable network infrastructure of the Internet, while firewall and network address translation (NAT) traversal is guaranteed. Generally, three generations of HTTP-based content delivery methods are distinguished: *(i)* traditional streaming, *(ii)* progressive download and *(iii)* HTTP adaptive streaming (HAS). In traditional streaming methods, data packets are sent at real-time rates only. The server sends enough data to fill the buffer at the client-side, but no additional content is transferred when the video is paused or stopped. A well-known example is the stateless Windows Media HTTP Streaming Protocol (MS-WMSP), which was introduced in 2006 and is still maintained today [5]. In progressive download methods, a simple file is downloaded from the server. The approach is progressive, in that the client can play the video while the download is still in progress. Web servers keep sending data until the download is complete, contrary to streaming servers that stop sending data when a certain amount of video is available to the client. A progressive download allows the client to play the video more smoothly, yet is not always bandwidth-effective: when discarding a video after a mere few seconds, a lot more data might have been downloaded already.

HAS has been introduced about one decade ago, and has since then become the de facto standard for over-the-top video streaming. An overview

[4]https://www.telenet.be/
[5]https://www.proximus.be/

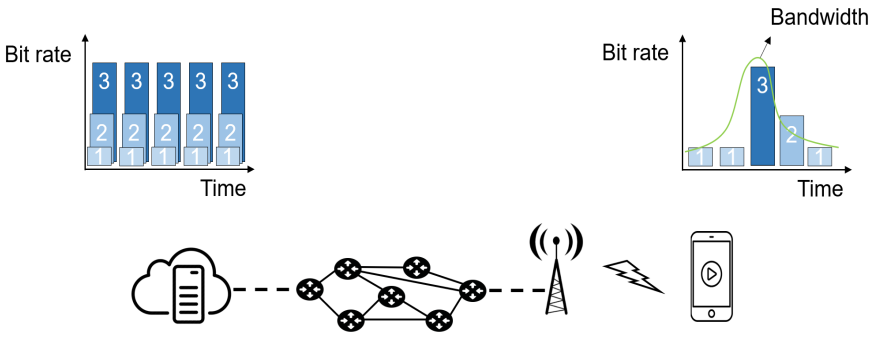


Figure 1.1: The concept of HTTP adaptive streaming.

of the general concept is shown in Figure 1.1. In HAS, the content is encoded using several quality representations, temporally segmented and stored on one or multiple servers within a content delivery network. The segment duration is generally between one to ten seconds, depending on the considered use case. A manifest file is maintained by the HAS server, which contains metadata describing, among others, different adaptation sets (each of which is attributed a type of media, such as video, audio and subtitles), and the available quality representations. Based on the perceived network conditions, the device characteristics, and the user's preferences, the client can then decide on the quality of each of these segments [6]. Having the ability to adapt the video quality, this approach actively avoids buffer starvation, and therefore results in smoother playback of the requested content and a higher Quality of Experience (QoE) for the end user [7]. Protocols and interfaces for HAS are defined by the Dynamic Adaptive Streaming over HTTP (DASH) standard, which defines, among others, the content of the manifest file or Media Presentation Description (MPD) [8, 9]. This standard allows compliant players to request and play content from any HTTP server, increasing reusability, scalability and reach.

1.2 Challenges for HTTP Adaptive Streaming

In 2018, 75% of downstream traffic in the United States consisted of data related to multimedia delivery. Expectations are that this number will increase to 82% by 2022, further straining the network [4]. At the same time, the demands and constraints set by new applications and technologies are increasing fast: while traditional video-on-demand streaming at 1080p comes with bit rates of around 5 Mb/s, high-quality 360° video at 8K resolution requires bit rates of 50 Mb/s and more.

Early evaluations of video streaming services generally reported QoS parameters to define their performance. Effective delivery, however, rather depends on the user's QoE. This concept has been defined by Qualinet, a Network of Excellence, as "the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and/or enjoyment of the application or service in the light of the user's personality and current state", a description which has later been adopted by the International Telecommunication Union [10]. This definition indicates that the performance of the service strongly depends on how the user experiences the service, which depends on multiple factors, including the video quality, the occurrence of playout freezes, the startup time, etc.

Measuring the user's satisfaction, however, requires subjective evaluations. These are expensive and time-consuming, and should be repeated for every considered scenario and parameter configuration. For this reason, a large number of models have been proposed throughout the years, which estimate the subjective QoE based on objective measurements. These models not only allow service providers to evaluate and compare different configurations in a scalable way, but also allow to identify key indicators for the QoE. Nevertheless, these models are scenario-dependent and generally rely on certain assumptions. In this work, we will focus on the optimization of individual indicators, rather than a specific model. Below, we list three important challenges for HAS, each of which is related to one or more of these indicators.

Challenge #1: Allow low-latency interaction with the provided content, both before and during video streaming.

Over the last years, low-latency interaction has become more important. With an ever-increasing supply of content, providers want to enable the user to quickly browse between videos, starting and stopping playout whenever required. Furthermore, more interactive use cases are now considered for video streaming, such as 360° video. When immersive media is considered, the user has the freedom to move her head and focus on a specific region, which is referred to as the field-of-view. While current solutions typically download the whole video at the same quality, some solutions distinguish between different regions. Using appropriate encoding schemes, which will be discussed later in this dissertation, the quality of relevant regions can be increased significantly. Such an approach, however, requires fast reaction to changes in the position of the user's head: when the user makes a sudden movement, she might be confronted with low-quality content because the client had not anticipated this action.

Challenge #2: Personalize the provided content and services to the targeted end user.

Most content providers deal with a huge amount of content. While Netflix has a limited content catalog (e.g., around 5600 titles in the US), YouTube currently deals with more than seven billion videos. While some content might be relevant to a given user, most of it is not. Therefore, content providers spend a lot of effort on personalizing the platform to the targeted user. Both Netflix and YouTube, and many other providers with them, use recommendation systems, which attempt to build a profile based on the user's viewing history. This allows the provider to make relevant suggestions for new content, which results in lower search times and higher customer satisfaction. Aside from recommending the right content, personalization is also required to allow services to adapt to the way users interact with the content. In an immersive video, for instance, users might look around at high speeds, while others stay still and focus on a specific region within the video. A different optimization strategy might be required for each of these users, maximizing the user's satisfaction in entirely different ways.

Challenge #3: Adapt the quality of the requested content to network characteristics, user movement and the considered video content.

One of the key challenges to video streaming is adapting to changes in the network and the user's movement, in order to provide the content at the most relevant video quality. To this end, an adept rate adaptation heuristic should be used at the client-side, which makes decisions on the quality representation for each of the requested media objects. In traditional video streaming, rate adaptation occurs on the temporal level only, defining the quality for each of the different segments, which are typically requested back-to-back. When tile-based solutions are considered, or when multiple objects within an immersive scene can be distinguished, an additional spatial dimension is added to the decision-making process. Taking into account this new dimension is not a straightforward task, since it requires precise monitoring of the perceived bandwidth, the user's movement, the buffer status, the file size of the requested content, etc.

1.3 Dissertation Outline

This doctoral dissertation is comprised of four research-focused chapters and one additional appendix. Each research-focused chapter deals with a unique use case, tackling one or more challenges related to HAS (see Table 1.1). The content of each chapter is briefly discussed below.

Table 1.1: The four different use cases discussed in this dissertation, and the challenges they address.

	Low-Latency	**Personalization**	**Adaptation**
Traditional video	×		
News-based video	×	×	
360° video	×	×	×
Volumetric media	×	×	×

Chapter 2 – Traditional Video

Despite the ability of HAS to deal with changing network conditions, a low average quality and a large camera-to-display delay are often observed in live streaming scenarios. This is a consequence of intermediate steps, such as video capture and processing, video encoding and server-side storage. This can be detrimental for the QoE, which suffers when low-quality content is presented to the user, or when events are spoiled by other parties (e.g., a goal is scored and other viewers are celebrating through chat).

In Chapter 2, we propose to use a novel HTTP feature that is able to deliver data back-to-back, without a need for the client to send a request for each individual segment. Combining this approach with segments of sub-second duration, referred to as *super-short* segments, it is possible to significantly reduce the startup time and end-to-end delay in HAS live streaming. In this chapter, we first evaluate the encoding overhead for different segment durations, and show that resulting values depend on the considered video content and frame rate. Then, using the most appropriate segment duration, we evaluate the push-based approach in a multi-client scenario with highly variable bandwidth and latency, and show that the startup time can be reduced with 31.2% compared to traditional solutions over HTTP/1.1 in mobile, high-latency networks. Furthermore, the end-to-end delay in live streaming scenarios can be significantly reduced, while providing the content at similar video quality.

Chapter 3 – News-Based Video

News-based websites and portals provide significant amounts of multimedia content to accompany news stories and articles. In this context, HAS is generally used to deliver video over the best-effort Internet. To stimulate user engagement with the provided content, such as searching through a video or browsing between videos, reducing the startup delay has become more and more important: while the current median load time is in the order of seconds, research has shown that user waiting times must remain

below two seconds to achieve an acceptable QoE. Low-latency solutions are thus required, and the principles presented in Chapter 2 can be applied again. On top of that, personalization can help define content preferred by the user, so that (the first part of) relevant videos can be prefetched before the user requests the actual content.

In Chapter 3, four complementary components are integrated into a comprehensive framework for low-latency delivery of news-related video content: *(i)* server-side encoding with short video segments, *(ii)* HTTP/2 server push on the application layer, *(iii)* server-side user profiling to identify relevant content for a given user, and *(iv)* client-side storage to hold proactively delivered content. Using a large dataset of a major Belgian news provider, containing millions of text- and video-based article requests, we show that the proposed framework reduces the videos' startup time in different mobile network scenarios by more than half, improving user interaction and allowing the user to quickly skim through available content.

Chapter 4 – 360° Video

The increasing popularity of head-mounted devices and 360° video cameras allows content providers to provide virtual reality video streaming over the Internet, using a two-dimensional representation of the immersive content combined with traditional streaming techniques. However, since only a limited part of the video (i.e., the viewport) is watched by the user, the available bandwidth is not optimally used. For this purpose, tile-based video can be used, which allows us to adapt the quality of each of the resulting tiles to the network characteristics and user's focus. This requires not only appropriate quality adaptation which takes into account the introduced spatial dimension, but also adept prediction of the movement of the user's head. Since this movement can be volatile, changes should be detected on a short time scale, resulting in a need for low-latency solutions.

In Chapter 4, we address the above challenges by means of a content-agnostic viewport prediction scheme (i.e., a scheme which does not take into account the content of the video, but only the movement of the user's head), and two rate adaptation heuristics which take into account the additonal spatial dimension. Furthermore, we introduce a novel feedback loop within the client's viewport prediction and rate adaptation schemes, which allows us to change quality decisions whilst downloading the required tiles for a given video segment, and discuss the advantages of HTTP/2 server push for content delivery. Through an extensive evaluation, we show that the proposed approaches result in a significant improvement in terms of video quality, compared to state-of-the-art solutions.

Chapter 5 – Volumetric Media

While the above solution allows the user to freely move her head, her location is fixed by the camera's position within the scene. Recently, however, an increased interest has been shown for free movement within immersive scenes, referred to as six degrees of freedom (6DoF), in which the user can both walk and look around. One way to realize this is by capturing objects through a number of cameras positioned in different angles, and creating a so-called point cloud. In its simplest form, a point cloud consists of a significant amount of six-tuples, defining each point's location in the three-dimensional space and its corresponding red, green and blue (RGB) color. The resulting output is referred to as volumetric media. Although capturing a human object requires around 5 Gb/s of data for a frame rate of 30 FPS, MPEG's reference encoder now allows to compress dynamic point cloud objects to bit rates in the order of 3 to 55 Mb/s, allowing feasible delivery over today's mobile networks.

While preliminary work exists on the delivery of single point cloud objects, no work has been done on quality adaptation for scenes consisting of multiple objects. Similar to traditional 360° video, considering multiple objects introduces a new spatial dimension which should be taken into account. In Chapter 5, we propose PCC-DASH, a standards-compliant means to adaptively stream scenes comprising multiple, dynamic point cloud objects. We present a number of rate adaptation heuristics which use information on the user's position and focus, the available bandwidth, and the client's buffer status to decide upon the most appropriate quality representation of each object. Through an extensive evaluation, we discuss the advantages and drawbacks of each solution, and highlight interesting paths for future work.

Appendix A – Adaptive Streaming over 4G/LTE Networks

Evaluations of the proposed approaches should include relevant scenarios with high variability and significant network latency. In Appendix A, we first discuss the merits of HTTP/2 server push for HAS, and analyze the induced bit rate overhead for HEVC-encoded video segments with a subsecond duration. We show that the proposed approach results in a higher video quality and lower freeze time, and allows to reduce the live delay compared to traditional solutions over HTTP/1.1. More importantly, however, we then present the details of a measurement study on the available bandwidth in 4G/LTE networks within the city of Ghent, Belgium. The results of this study are used in the evaluations of Chapters 3, 4 and 5.

1.4 Publications

The results obtained during this PhD research have been published in scientific journals and presented at a series of international conferences and workshops. The following list provides an overview of these publications.

1.4.1 Publications in International Journals

[1] **J. van der Hooft**, S. Petrangeli, T. Wauters, R. Huysegems, P. Rondao Alface, T. Bostoen, and F. De Turck. *HTTP/2-Based Adaptive Streaming of HEVC Video over 4G/LTE Networks*. Published in IEEE Communications Letters, vol. 20, no. 11, p. 2177-2180, 2016.

[2] **J. van der Hooft**, M. Claeys, N. Bouten, T. Wauters, J. Schönwälder, A. Pras, B. Stiller, M. Charalambides, R. Badonnel, J. Serrat, C. R. Paula dos Santos, and F. De Turck. *Updated Taxonomy for the Network and Service Management Research Field*. Published in the Journal of Networks and Systems Management, vol. 26, no. 3, p. 790-808, 2018.

[3] **J. van der Hooft**, S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. *An HTTP/2 Push-Based Approach for Low-Latency Live Streaming with Super-Short Segments*. Published in the Journal of Networks and Systems Management, vol. 26, no. 1, p. 51-78, 2018.

[4] S. Petrangeli, **J. van der Hooft**, T. Wauters, and F. De Turck. *Quality of Experience-Centric Management of Adaptive Video Streaming Services: Status and Challenges*. Published in Transactions on Multimedia Computing Communications and Applications, vol. 14, no. 2, p. 31:1-31:29, 2018.

[5] S. Petrangeli, D. Pauwels, **J. van der Hooft**, M. Žiak, J. Slowack, T. Wauters, and F. De Turck. *A Scalable WebRTC-Based Framework for Remote Video Collaboration Applications*. Published in Multimedia Tools and Applications, p. 1-34, 2018.

[6] **J. van der Hooft**, C De Boom, S. Petrangeli, T. Wauters, and F. De Turck. *Performance Characterization of Low-Latency Adaptive Streaming from Video Portals*. Published in IEEE Access, vol. 6, no. 1, p. 43039-43055, 2018.

[7] **J. van der Hooft**, N. Bouten, D. De Vleeschauwer, W. Van Leekwijck, T. Wauters, S. Latré, and F. De Turck. *Clustering-Based Quality Selection Heuristics for HTTP Adaptive Streaming over Cache Networks*. Pub-

lished in the International Journal on Network Management, vol. 28, no. 6, 2018.

[8] **J. van der Hooft**, M. Torres Vega, S. Petrangeli, T. Wauters, and F. De Turck. *Tile-Based Adaptive Streaming for Virtual Reality Video*. In revision for publication in ACM Transactions on Multimedia Computing, Communications, and Applications, 2019.

[9] R. I. Tavares da Costa Filho, M. Caggiani Luizelli, S. Petrangeli, M. Torres Vega, **J. van der Hooft**, T. Wauters, F. De Turck, and L. Paschoal Gaspary. *Dissecting the Performance of VR Video Streaming Through the VR-EXP Experimentation Platform*. Submitted to ACM Transactions on Multimedia Computing, Communications, and Applications, 2019.

1.4.2 Publications in International Conferences

[1] **J. van der Hooft**, S. Petrangeli, M. Claeys, J. Famaey, F. De Turck, *A Learning-Based Algorithm for Improved Bandwidth-Awareness of Adaptive Streaming Clients*. Published in Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, Ottawa, Canada, 2015.

[2] R. Huysegems, **J. van der Hooft**, T. Bostoen, P. Rondao Alface, S. Petrangeli, T. Wauters, F. De Turck, *HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming*. Published in Proceedings of the ACM Multimedia Conference, Brisbane, Australia, 2015.

[3] **J. van der Hooft**, S. Petrangeli, N. Bouten, T. Wauters, R. Huysegems, T. Bostoen, F. De Turck, *An HTTP/2 Push-Based Approach for SVC Adaptive Streaming*. Published in Proceedings of the IEEE/IFIP Network Operations and Management Symposium, Istanbul, Turkey, 2016.

[4] S. Petrangeli, **J. van der Hooft**, T. Wauters, R. Huysegems, P. Rondao Alface, T. Bostoen, F. De Turck, *Live Streaming of 4K Video over the Internet*. Published in Proceedings of the ACM Multimedia Systems Conference, Klagenfurt, Austria, 2016.

[5] D. Pauwels, **J. van der Hooft**, S Petrangeli, T. Wauters, D. De Vleeschauwer, F. De Turck, *A Web-Based Framework for Fast Synchronization of Live Video Players*. Published in Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, Lisboa, Portugal, 2017.

[6] **J. van der Hooft**, S Petrangeli, T. Wauters, R. Rahman, N. Verzijp, R. Huysegems, T. Bostoen, F. De Turck, *Analysis of a Large Multimedia-Rich Web Portal for the Validation of Personal Delivery Networks*. Published in Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, Lisboa, Portugal, 2017.

[7] S. Petrangeli, D. Pauwels, **J. van der Hooft**, J. Slowack, T. Wauters, F. De Turck, *Dynamic Video Bitrate Adaptation for WebRTC-Based Remote Teaching Applications*. Published in Proceedings of the IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 2018.

[8] **J. van der Hooft**, C. De Boom, S. Petrangeli, T. Wauters, F. De Turck, *An HTTP/2 Push-Based Framework for Low-Latency Adaptive Streaming Through User Profiling*. Published in Proceedings of the IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 2018.

[9] R. I. T. da Costa Filho, M. Torres Vega, M. Caggiani Luizelli, **J. van der Hooft**, S. Petrangeli, T. Wauters, F. De Turck, L. Paschoal Gaspary, *Predicting the Performance of Virtual Reality Video Streaming in Mobile Networks*. Published in Proceedings of the ACM Multimedia Systems Conference, Amsterdam, The Netherlands, 2018.

[10] S. Petrangeli, D. Pauwels, **J. van der Hooft**, J. Slowack, T. Wauters, F. De Turck, *Improving Quality and Scalability of WebRTC Video Collaboration Applications*. Published in Proceedings of the ACM Multimedia Systems Conference, Amsterdam, The Netherlands, 2018.

[11] **J. van der Hooft**, D. Pauwels, C. De Boom, S. Petrangeli, T. Wauters, F. De Turck, *Low-Latency Delivery of News-Based Video Content*. Published in Proceedings of the ACM Multimedia Systems Conference, Amsterdam, The Netherlands, 2018.

[12] M. Torres Vega, T. Mehmli, **J. van der Hooft**, T. Wauters, F. De Turck, *Enabling Virtual Reality for the Tactile Internet: Hurdles and Opportunities*. Published in Proceedings of the International Workshop on High-Precision Networks Operations and Control, Rome, Italy, 2018.

[13] **J. van der Hooft**, M. Torres Vega, S. Petrangeli, T. Wauters, F. De Turck, *Quality Assessment for Adaptive Virtual Reality Video Streaming: A Probabilistic Approach on the User's Gaze*. Published in Proceedings of the International Workshop on Quality of Experience Management, Paris, France, 2019.

[14] **J. van der Hooft**, M. Torres Vega, S. Petrangeli, T. Wauters, F. De Turck, *Optimizing Adaptive Tile-Based Virtual Reality Video Streaming*. Published in Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, Washington DC, USA, 2019.

[15] M. Torres Vega, **J. van der Hooft**, J. Heyse, S. Petrangeli, F. De Backere, T. Wauters, F. De Turck, *Exploring New York in 8K - An Adaptive Tile-based Virtual Reality Video Streaming Experience*. Accepted for publication in Proceedings of the ACM Multimedia Systems Conference, Amherst, USA, 2019.

[16] **J. van der Hooft**, T. Wauters, F. De Turck, C. Timmerer, H. Hellwagner, *Towards 6DoF HTTP Adaptive Streaming Through Point Cloud Compression*. Submitted to the ACM Multimedia Conference, Nice, France, 2019.

References

[1] Statista. *Electronic Video Sales in the United States 2014-2018, by Type*, 2019. Available from: https://www.statista.com/statistics/788096/video-sales-revenue/.

[2] Netflix. *Shareholder Letter - Q4 2018*, 2019. Available from: https://s22.q4cdn.com/959853165/files/doc_financials/quarterly_reports/2018/q4/01/FINAL-Q4-18-Shareholder-Letter.pdf.

[3] K. A. Zimmermann and J. Emspak. *Internet History Timeline: ARPANET to the World Wide Web*, 2017. Available from: https://www.livescience.com/20727-internet-history.html.

[4] Cisco. *Cisco Visual Networking Index: Forecast and Trends, 2017–2022*, 2017. Available from: https://www.livescience.com/20727-internet-history.html.

[5] Windows. *Windows Media HTTP Streaming Protocol*, 2019. Available from: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-wmsp/8f34d1ff-237d-4acd-939d-63552c97422c.

[6] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann. *A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP*. IEEE Communications Surveys Tutorials, 21(1):562–585, 2019.

[7] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia. *A Survey on Quality of Experience of HTTP Adaptive Streaming*. IEEE Communications Surveys Tutorials, 17(1):469–492, 2015.

[8] T. Stockhammer. *Dynamic Adaptive Streaming over HTTP: Standards and Design Principles*. In Proceedings of the 2nd ACM Conference on Multimedia Systems, pages 133–144, New York, 2011. ACM.

[9] I. Sodagar. *The MPEG-DASH Standard for Multimedia Streaming over the Internet*. IEEE MultiMedia, 18(4):62–67, 2011.

[10] K. Brunnström, S. A. Beker, K. De Moor, A. Dooms, S. Egger, M. Garcia, T. Hoßfeld, S. Jumisko-Pyykkö, C. Keimel, M. L, et al. *Qualinet White Paper on Definitions of Quality of Experience*. Technical report, Qualinet, 2013. Available from: https://hal.archives-ouvertes.fr/hal-00977812/document.

2

An HTTP/2 Push-Based Approach for Low-Latency Live Streaming with Super-Short Segments

"Now you're looking for the secret. But you won't find it because of course, you're not really looking. You don't really want to work it out. You want to be fooled."

–The Prestige, 2006

As discussed in Chapter 1, a low video quality and a large camera-to-display delay are often observed in live streaming scenarios. In this chapter, we propose a push-based approach for HAS, in which HTTP/2's push feature is used to actively push segments from server to client. This allows the server to send data back-to-back, without the client requesting each individual segment. Combining this approach with segments of sub-second duration, it is possible to significantly reduce the startup time and end-to-end delay. First, we evaluate the encoding overhead for different segment durations, and show that resulting values depend on the considered video content and frame rate. Then, using the most appropriate segment duration, we show that the startup time can be reduced with 31.2% compared to traditional solutions over HTTP/1.1 in mobile, high-latency networks. Furthermore, the end-to-end delay in live streaming scenarios can be reduced with 4 s*, while providing the content at similar video quality.*

⋆⋆⋆

J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck

Published in the Journal of Networks and Systems Management, vol. 26, no. 1, p. 51-78, 2018

vspace0.38cm

Abstract Over the last years, streaming of multimedia content has become more prominent than ever. To meet increasing user requirements, the concept of HTTP adaptive streaming (HAS) has recently been introduced. In HAS, video content is temporally divided into multiple segments, each encoded at several quality levels. A rate adaptation heuristic selects the quality level for every segment, allowing the client to take into account the observed available bandwidth and the buffer filling level when deciding the most appropriate quality level for every new video segment. Despite the ability of HAS to deal with changing network conditions, a low average quality and a large camera-to-display delay are often observed in live streaming scenarios. In the meantime, the HTTP/2 protocol was standardized in February 2015, providing new features which target a reduction of the page loading time in web browsing. In this chapter, we propose a novel push-based approach for HAS, in which HTTP/2's push feature is used to actively push segments from server to client. Using this approach with video segments with a sub-second duration, referred to as *super-short* segments, it is possible to reduce the startup time and end-to-end delay in HAS live streaming. Evaluation of the proposed approach, through emulation of a multi-client scenario with highly variable bandwidth and latency, shows that the startup time can be reduced with 31.2% compared to traditional solutions over HTTP/1.1 in mobile, high-latency networks. Furthermore, the end-to-end delay in live streaming scenarios can be reduced with 4 s, while providing the content at similar video quality.

2.1 Introduction

Over the last years, delivery of multimedia content has become more prominent than ever. Particularly, video streaming applications are currently responsible for more than half of the Internet traffic [1]. To improve video streaming over the best-effort Internet, HTTP adaptive streaming (HAS) has recently been introduced. As shown in Figure 2.1, video content is temporally divided into segments with a typical length of 1 to 10 seconds, each encoded at multiple quality levels. Video segments are requested by the HAS client, equipped with a rate adaptation heuristic to select the best quality level based on criteria such as the perceived bandwidth and the

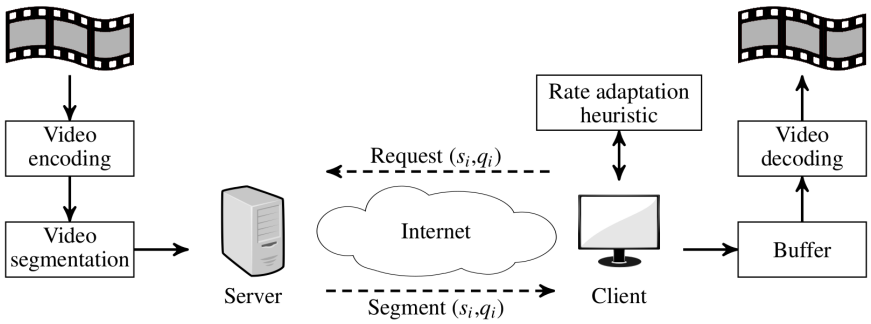


Figure 2.1: The concept of HTTP adaptive streaming.

video player's buffer filling level. The ultimate objective of this heuristic is to optimize the Quality of Experience (QoE) perceived by the user, which depends among others on the average video quality, the frequency of quality changes and the occurrence of playout freezes [2]. The client decodes all incoming segments and plays them out in linear order.

This approach comes with a number of advantages. For the over-the-top provider, video delivery is cheaper because the existing HTTP infrastructure for web surfing can be reused. Better scalability is guaranteed, since quality selection is performed by clients in a distributed way. The user generally perceives a smoother playback experience, because the client adapts the requested bit rate to network conditions; when the perceived bandwidth drops for instance, a lower quality level can be selected to prevent buffer starvation. Because of these advantages, major players such as Microsoft, Apple, Adobe and Netflix massively adopted the adaptive streaming paradigm and proposed their own rate adaptation heuristics. As most HAS solutions use the same architecture, the Motion Picture Expert Group (MPEG) proposed Dynamic Adaptive Streaming over HTTP (DASH), a standard which defines the interfaces and protocol data for adaptive streaming [3].

Despite the many advantages brought by HAS, several inefficiencies still have to be solved in order to improve the QoE perceived by the user in live video streaming scenarios. For instance, the camera-to-display delay, which is the delay between capturing an event and its playout on the client's display, is very important in a live video scenario. In current HAS deployments however, this delay is in the order of tens of seconds. This is because encoding video in real-time is a computationally expensive and time consuming process, a large buffer at the client is generally used to prevent playout freezes and the server only sends a new video segment once a request is issued by the client. One possible way to lower this de-

lay is to use segments with a sub-second duration, henceforth referred to as *super-short* segments: less time is required for encoding at the head-end and transport over the network [4], while a lower temporal buffer size can be used. The rate adaptation heuristic can also adapt faster to changing network conditions, preventing buffer starvation when the available bandwidth drops. Unfortunately, several disadvantages to this approach exist. First, as every segment must start with an Instantaneous Decoder Refresh (IDR) frame to make it independent of other segments, a significant encoding overhead can be introduced [5]. Second, a larger number of HTTP GET requests are required to retrieve the segments, resulting in a larger network and server overhead. Moreover, at least one extra round-trip time (RTT) cycle is lost between subsequent requests, significantly reducing bandwidth utilization when the RTT is relatively high compared to the segment duration. This problem mainly arises in mobile networks, where the RTT can vary significantly, depending on the network carrier and the type of connection [6]. Furthermore, factors such as the type of WiFi network and the distance between client and server, have a possible impact on the RTT in the order of 100 ms [7]. As a consequence, it is typically infeasible to use segment durations below the one second range, and the resulting camera-to-display delay is in the order of tens of seconds. In this chapter however, we introduce a novel technique to use super-short segments, using the push feature of the recently standardized HTTP/2 protocol.

Early 2012, the Internet Engineering Task Force (IETF) httpbis working group started the standardization of HTTP/2 to address a number of deficiencies in HTTP/1.1 [8, 9]. The new HTTP/2 standard was published in February 2015, and is now supported by major browsers such as Google Chrome, Firefox and Internet Explorer [10, 11]. The main focus of this standard is to reduce the latency in web delivery, using three new features that provide the possibility to terminate the transmission of certain content, prioritize more important content and push content from server to client. In earlier work, we proposed a number of HTTP/2-based techniques that can improve the QoE in HAS, and particularly in live video streaming [12]. We mainly focused on a push-based approach, in which video segments are actively pushed from server to client in order to avoid idle RTT cycles. An overview of first results was provided for a fixed bandwidth scenario, showing that the approach is capable of significantly reducing the startup and end-to-end delay in high-RTT networks. In this work, however, we focus on the application of these techniques in mobile networks, where high variability in the available bandwidth is generally perceived. The adoption of our initially proposed approach is not straightforward in this case, because it assumes that pushed segments are delivered within a certain time

interval. This assumption no longer holds when the available bandwidth is highly variable and sudden bandwidth drops occur. We therefore propose an improved technique which allows the deployment of a push-based approach in mobile networks, effectively using HTTP/2's newest features.

The contributions of this work are three-fold. First, we introduce a novel push-based approach, discussing its advantages for live video streaming in HAS. We show that in high-RTT networks, the startup and end-to-end delay can be significantly reduced, while a higher bandwidth utilization can be achieved. Second, we perform a detailed analysis on the encoding overhead for super-short segments, with relevant factors including the frame rate, the GOP length and the type of content. The trade-off between high responsiveness and a higher overhead is discussed, and a suitable segment duration is selected. Third, we present results from an extensive evaluation to characterize the gain of the proposed approach compared to state-of-the-art HAS over HTTP/1.1. Results are reported for a multi-client setup with variable bandwidth and latency, illustrating possible gains in mobile, high-RTT networks.

The remainder of this chapter is structured as follows. Section 2.2 gives an overview of related work, focusing both on HAS and HTTP/2. The proposed push-based solution is presented in Section 2.3, followed by a discussion on the encoding overhead for super-short segments in Section 2.4. Detailed evaluation results on the push-based approach are provided in Section 2.5, before coming to final conclusions in Section 2.6.

2.2 Related Work

2.2.1 HTTP Adaptive Streaming

Techniques to improve the QoE perceived by the user for HAS services can be divided in client-based, server-based and network-based solutions [13]. A number of client-based solutions have recently been proposed in literature. Benno et al. propose a more robust rate adaptation heuristic for wireless live streaming [14]. By averaging the measured bandwidth over a sliding window, fluctuations are smoothed out, allowing the client to select a quality level that is sustainable and avoids oscillations. Petrangeli et al. propose a rate adaptation heuristic that strongly focuses on a new model for the QoE, taking into account the average video quality, its standard deviation and the impact of freezes in the decision taking process [15]. By expanding the heuristic to take into account a hierarchical fairness signal, fairness among clients can be induced without explicit communication between peers. Menkovski et al. propose the use of the SARSA(λ) tech-

nique at the client-side to select the most appropriate quality level, based on the estimated bandwidth, the buffer filling and the position in the video stream [16]. Claeys et al. propose the use of a Q-learning algorithm in the rate adaptation heuristic, based on the estimated bandwidth and the buffer filling [17]. Results show that the client outperforms other deterministic algorithms in several network environments. Many other rate adaptation heuristics exist, but we refer to a survey by Seufert et al. for a more extensive view on the matter [13]. The above research attempts to increase the QoE through more intelligent client-side decisions, either increasing the average video quality, reducing the amount of switches or the occurrence of playout freezes for video on demand (VoD). In contrast, this research focuses on a reduction of the startup time and end-to-end delay for live streaming scenarios, taking into account the average video quality and the duration of playout freezes. Furthermore, the proposed approach is more general in nature: it can be extended to work on top of any existing rate adaptation heuristic.

Server-side solutions typically focus on encoding schemes for HAS [13]. The H.264/AVC codec is most widely used for video streaming, although it requires the server and intermediate caches to store multiple representations of the same video. This generally results in a storage overhead, increased bandwidth requirements and reduced caching efficiency. As suggested by Sánchez et al., the adoption of scalable video coding (SVC) in HAS offers a solution to this problem [18]. In SVC, redundancy is reduced because every quality level is constructed as an enhancement of the lower quality level. In the encoding process, lower quality layers are retrieved from a high-quality video bitstream by lowering the spatial or temporal resolution, the video quality signal or a combination thereof. Starting from the lowest quality level, called the base layer, the client can decode higher quality levels in combination with the lower layers. Although SVC provides an effective means to reduce content redundancy, it does introduce an encoding overhead of about 10% per layer. Since multiple requests are required to retrieve a single video segment, SVC-based solutions are even more susceptible to high RTTs. In addition, the commercial availability of SVC encoders and decoders often poses a problem. For these reasons, the application of SVC-encoded content in HAS is often questioned. In this chapter, we focus on AVC-encoded content only.

Other server-side solutions exist as well. Akhshabi at al. propose server-based traffic shaping to reduce video quality oscillations at the client [19]. Their evaluations show a major reduction in terms of instability, practically stabilizing competing clients without a significant utilization loss. De Cicco et al. propose an approach based on feedback control, providing a

Quality Adaptation Controller centralized at the server-side [20]. Based on a predefined target level, this controller attempts to keep the TCP send buffer at a bit rate close to the available bandwidth. Although the authors show that results are promising, the continuous bit rate control leads to a large, non-scalable workload at the server-side and does not take into account the client's buffer filling. In our proposed approach, quality decisions are located at the client, resulting in a distributed, scalable solution.

Network-based approaches often target IPTV solutions, where a dedicated network is used to provide the content to the user. Network elements are used to either optimize the QoE of single users, or to optimize the global QoE by providing fairness among clients. Bouten et al. avoid suboptimal distribution by introducing intelligence in the network that can override the client's local quality decisions [21]. In more recent work, the same authors use Differentiated Services (DiffServ) to give priority to the base layer segments in SVC [22]. As a result, the SVC-based client is more robust to video freezes, even when the total buffer size is decreased significantly to reduce the end-to-end delay for live video streaming. Petrangeli et al. suggest an approach in which each client learns to select the most appropriate quality level, maximizing a reward based both on its own QoE and on the QoE perceived by other clients [23]. To this end, a coordination proxy estimates all perceived rewards and generates a global signal that is sent periodically to all clients. Without explicit communication among agents, the algorithm is able to outperform both Microsoft Smooth Streaming (MSS) and the algorithm proposed by Claeys et al. in a multi-client scenario [17]. More recently, the same authors propose priority-based delivery of HAS video [24]. In their approach, an OpenFlow controller is used to prioritize segments for clients which are close to buffer starvation. An extensive evaluation shows that their approach allows to reduce the total freeze time and frequency with up to 75%. Schierl et al. apply SVC-based streaming in a network environment with support for graceful degradation of the video quality when the network load increases [25]. The authors argue the need for Media Aware Network Elements (MANEs), capable of adjusting the SVC stream based on a set of policies specified by the network provider. More recently, MPEG proposed Server and Network Assisted DASH (SAND) [26]. In the suggested approach, a bi-directional messaging plane is used between the clients and other so-called DASH-Aware Network Elements (DANEs), in order to carry both operational and assistance information. In this way, the client can request the network to provide guaranteed bit rates for the video streaming session. Finally, Latré et al. propose an in-network rate adaptation heuristic, responsible for determining which SVC quality layers should be dropped, in combination

with a Pre-Congestion Notification (PCN) based admission control mechanism [27]. In contrast to these works, this chapter considers over-the-top video streaming only. Therefore, no extra middlebox functionality is required for the proposed approach.

2.2.2 The HTTP/2 Protocol

The new HTTP/2 standard was published as an IETF RFC in February 2015, mainly focusing on the reduction of latency in web delivery. The first draft for this protocol was based on SPDY, an open networking protocol developed primarily by Google [28]. Using this protocol, an average reduction of up to 64% was observed for the page load time. Other studies show that the mere replacement of HTTP by SPDY helps only marginally. Cardaci et al. evaluate SPDY over high latency satellite channels [29]. On average, SPDY only slightly outperforms HTTP. Erman et al. provide a detailed measurement study to understand the benefits of using SPDY over cellular networks [30]. They report that SPDY does not clearly outperform HTTP due to cross-layer dependencies between TCP and the cellular network technology. Similar results are reported by Elkhatib et al., who conclude that SPDY may both decrease or increase the page load time [31]. This is a consequence of the fact that SPDY's multiplexed connections last much longer than HTTP's, making SPDY more susceptible to packet loss. Similarly, Wang et al. show that SPDY can either reduce or increase the load time of web pages [32]. They conclude that next to a careful configuration of the TCP protocol, a reduction of the page load time may be achieved by changing the page load process using SPDY's server push. With respect to energy consumption, a recent study by Chowdhury et al. shows that HTTP/2 is more energy efficient than HTTP/1.1 when RTTs in the order of 30 ms or more are observed [33]. Since high RTTs are often observed in mobile networks, this is of significant importance for mobile devices with constrained battery life.

2.2.3 HTTP/2 for Multimedia Delivery

Mueller et al. are the first to evaluate the performance of DASH-based adaptive streaming over SPDY [34]. The existing HTTP/1.0 or HTTP/1.1 layer is replaced by SPDY, without any modifications to the HAS client or server. The authors show that if SPDY is used over SSL, the gains obtained by using header compression, a persistent connection and pipelining are almost completely cancelled out by the losses due the SSL and framing overhead. Wei et al. first explore how HTTP/2's features can be used to improve HAS [4]. By reducing the segment duration from five seconds

to one second, they manage to reduce the camera-to-display delay with about ten seconds. An increased number of GET requests is avoided by pushing k segments after each request, using HTTP/2's server push. One disadvantage to this approach is that when a client switches to another quality level, the push stream for the old quality level is in competition with the segments downloaded at the new quality level. This results in an increased switching delay for the client and bandwidth overhead in the network. In later work, the authors show that the induced switching delay is about two segment durations and is independent of the value of k, while the introduced bandwidth overhead heavily depends on this value [35]. Cherif et al. propose DASH fast start, in which HTTP/2's server push is used to reduce the startup delay in a DASH streaming session [36]. Additionally, the authors propose a new approach for video adaptation, in which WebSocket over HTTP/2 is used to estimate the available bandwidth. In previous work, we proposed a number of techniques to improve the QoE in HAS, using on HTTP/2's stream prioritization, stream termination and server push [12]. Every technique has its own advantages, such as a higher bandwidth utilization and a gain of one or multiple RTT cycles in the client's startup phase. We thus proposed a *full-push* approach, in which several techniques are combined together. A first evaluation showed promising results, such as a lower startup time and end-to-end delay in high-RTT networks. In other work, we showed that the occurrence of playout freezes can be reduced through SVC-based streaming over HTTP/2, limiting the quality loss in high-RTT networks by pushing base layer segments to the client [37]. Although results are promising, the average quality is lower than for AVC-based HAS, and only a limited amount of quality layers can be used because of SVC's encoding overhead.

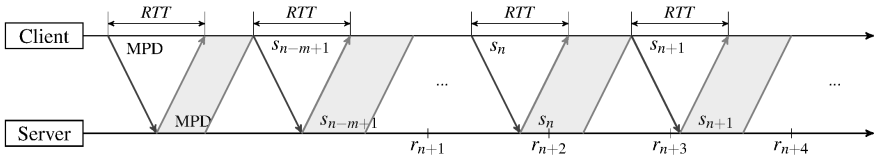
Except for our work on SVC, focus in the above research is mainly on reducing the live latency and the number of GET requests issued by the client. Results are shown for scenarios in which the available bandwidth is constant, or only changes every 20 seconds. Although significant improvements are obtained, it is not clear how the proposed approaches impact other aspects of the QoE, such as the average quality or the amount of playout freezes. Furthermore, there is no analysis of the encoding overhead introduced by using shorter video segments. In this chapter, a new push-based approach is proposed to provide the server with more explicit feedback from the client, in order to avoid network congestion and buffer starvation when sudden bandwidth drops occur. The encoding overhead for shorter video segments is analyzed in detail, and evaluations are extended to a multi-client mobile network scenario with high variations in the available bandwidth and network latency.

2.3 Push-Based Approach

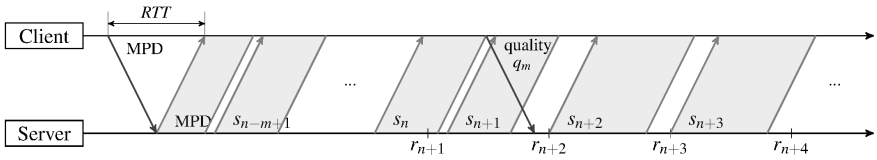
In this section, we first elaborate on the initially proposed push-based approach, in which video segments are actively pushed from server to client [12]. Its advantages and possible applications are discussed in detail, along with new challenges that arise in mobile, bandwidth-variable networks. We then propose an extension to the suggested approach, in which explicit feedback from the client is used to improve performance.

2.3.1 Full-Push Approach

In HAS, a video streaming session starts with the client sending a request for the video's media presentation description (MPD). This file contains information regarding the video segments, such as the duration, resolution and available bit rates. In live video, it also contains information regarding the timing of the video streaming session and the segments available on the server. Based on the contents of the MPD, the client then starts requesting video segments one by one, ramping up the buffer by downloading segments at the lowest quality. Once the buffer filling is sufficiently high - typically when a certain panic threshold is exceeded - further decisions regarding the video quality are made by the rate adaptation heuristic. The main drawback of this approach is that one RTT cycle is lost to download each segment, which has a significant impact on the startup time in high-RTT networks. An illustration of this behavior is presented in Figure 2.2a, where the first phase of a live streaming session is shown. In the full-push approach, the server pushes m segments to the client as soon as the MPD request is received, where m corresponds to the number of segments that fit into a preferred buffer size defined by the client. This of course requires that the server is aware of the relationship between the different HAS objects, and that the client can indicate its maximum buffer size in the MPD request. Since state-of-the-art rate adaptation heuristics ramp up the buffer by downloading segments at the lowest quality, it makes sense to push these at low quality as well. Note that the client cannot select another quality anyway, since the MPD defining all quality levels has not yet been received. As illustrated in Figure 2.2b, at least one RTT cycle is gained in the reception of the first video segment, and multiple RTT cycles are gained during the buffer rampup phase. Once the MPD and the first m segments are sent, the server either periodically pushes a new segment to the client at the specified quality level (VoD), or as soon as a new segment is available (live video). Every time a segment is completely received, the client estimates the perceived bandwidth and checks the buffer filling. Based on these parameters, the rate adaptation heuristic determines the most suit-



(a) Adaptive streaming using traditional HTTP/1.1



(b) Adaptive streaming using HTTP/2's server push

Figure 2.2: An example live scenario for HTTP/1.1 and HTTP/2, where the client requests m available segments to ramp up the buffer [12]. If the last released segment has index n, the first segment to play is $n - m + 1$. Note that r_i denotes the release of segment i at the server-side, while s_i denotes its request/download by the client. Furthermore, *quality* q_j indicates that the server should change the quality of pushed segments to j.

able video quality and if required, a request is sent to change the bit rate of pushed segments. Figure 2.3 shows a possible application of the full-push approach in a content delivery network (CDN) environment, both for VoD and for live video. The client controls the session by sending start, stop, pause, or resume messages under the form of an HTTP GET request to the server. In VoD, the client informs the server about the state of its buffer, indicating that segments must either be sent back-to-back or in a periodic way. In case of live streaming, the server ramps up the buffer by pushing the last m segments and from then on, pushes new segments to the client as soon as they are available.

The proposed approach has a number of advantages. Since the first m segments are pushed back-to-back when the manifest is requested, the client's startup delay can be significantly reduced in high-RTT networks. Since no RTT cycles are lost to request the video segments, video segments with a sub-second duration can be used, further reducing this delay. Using a smaller temporal buffer, the approach can effectively reduce the overall end-to-end delay as well. However, there are two disadvantages to a full-push approach for super-short segments.

First, while a segment n is being pushed from server to client, it is possible that new segments $n + 1, ..., n + l$ are released. Since the rate adapta-

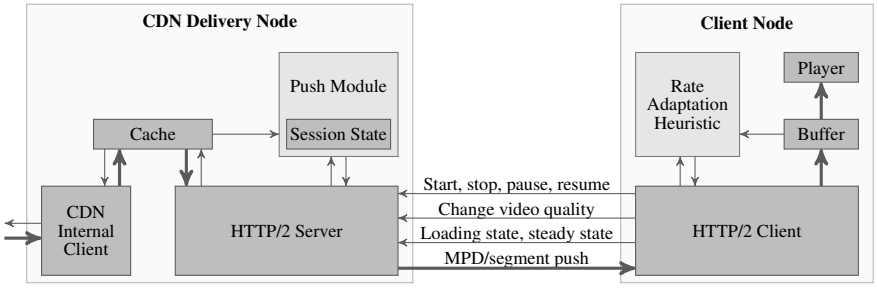


Figure 2.3: Application of the full-push approach in a CDN environment. The client controls the session by sending request messages under the form of HTTP GET requests to the server.

tion heuristic of the client is only activated when a segment is completely received, these segments will all be pushed to the client at the same video quality as segment n. This entails that, when sudden bandwidth drops occur, the server will not immediately lower the quality level to match the available bandwidth within the network. This will generally result in more packet queuing in the network buffers, leading to a higher risk of buffer starvation and thus to a lower QoE. To overcome this issue, Section 2.3.2 covers a way to limit the maximum number of segments in flight. In this way, network congestion is avoided, as the server can proactively adjust the pushed video quality to the available bandwidth.

Second, since every segment has to start with an IDR frame, a higher bit rate is required to achieve the same video quality. Therefore, an encoding overhead is required to achieve the same visual quality for content provided with a smaller Group of Pictures (GOP). In Section 2.4, we investigate the encoding overhead for eight different videos and select an appropriate minimal segment duration to evaluate the proposed push-based approach.

2.3.2 Full-Push Approach with Acknowledgments

In the full-push approach, the server can theoretically push an indefinite number of (high-quality) segments to the client. When the available bandwidth suddenly drops, a large number of packets will thus be queuing in the network. However, the intention of the HTTP/2-based approach is merely to push segments in order to bridge the idle RTT cycles that occur when segments are pulled by the client. To achieve this, the server does not need to send an indefinite number of segments; it is sufficient to push a maximum of k segments consecutively, where k depends on the network's

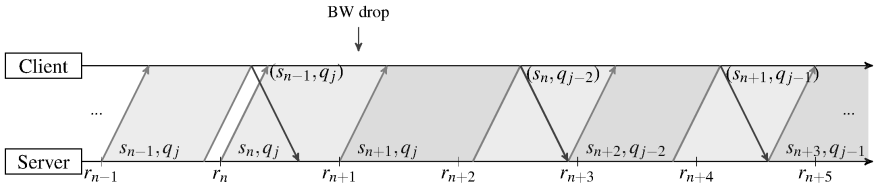


Figure 2.4: Illustration of HTTP/2's server push with explicit acknowledgments. A value of $k = 2$ is used in this scenario, so that either 0, 1 or 2 segments are in flight (indicated in white, light gray and dark gray respectively).

RTT and the segment duration of the video. For this reason, we propose an improved approach in which the maximum number of segments in flight is limited to a certain value k. Note that this is no standalone solution; it represents a possible, complimentary dimension of the initially proposed push-based approach.

The suggested approach is illustrated in Figure 2.4, for a value of $k = 2$. Similar to the full-push approach, the client first sends a GET request to fetch the MPD, indicating its preferred buffer size. The server pushes the first m segments at the lowest quality, where m corresponds to the number of segments that fit in the buffer size defined by the client. Then, segments are periodically pushed as soon as they become available. The main difference is, however, that the server stops pushing new segments until the reception of the kth last segment is acknowledged by the client. To this end, a sliding window is used on the server's application layer, defining which segments can and cannot yet be pushed. At the client-side, the rate adaptation heuristic determines its preferred quality level for future segments once a new segment has completely arrived. This quality level and the received segment's index are then passed along to the server, which moves the sliding window one segment to the right and pushes the next segment as soon as it is available (if not already). With respect to the parameter k, the following rules of thumb apply:

1. For low RTTs and high segment durations, high bandwidth utilization is achieved even without the presence of a push-based approach. Low values of k should be used in order to minimize network congestion;

2. For high RTTs and low segment durations, a large value of k should be applied in order to maximize bandwidth utilization. Unfortunately, this is expected to have a negative impact on the freeze duration and frequency as well.

The value of k should thus be finetuned based on the state of the network and the video's segment duration. In section 2.5, we will show that the following rule of thumb can be used to calculate the optimal value for k, given the RTT and the segment duration seg:

$$k = \begin{cases} ceil\left(\frac{RTT}{seg}\right) + 1, & \text{if } \frac{RTT}{seg} > a. \\ 1, & \text{otherwise.} \end{cases} \tag{2.1}$$

In this equation, the value of k is directly proportional to the ratio of the RTT and the segment duration, unless this ratio is lower than a threshold value a. The reasoning behind this is as follows. When the ratio of the RTT and the segment duration is small, little improvement of the average video quality is expected when a push-based approach is used. Since pushing multiple segments increases the chances of buffer starvation when a sudden bandwidth drop occurs, it is more appropriate to only pull the segments in this case. Using $k = 1$, one segment is pushed for every request/acknowledgment, which indeed corresponds to a pull-based approach. If the ratio of the RTT and the segment duration is large, however, a significant gain can be achieved for the average video quality. In this case, idle RTT cycles should be avoided by actively pushing new video segments. In Section 2.5, we will show that $a = 0.2$ is an appropriate value for the ratio's threshold.

2.4 Impact of the Segment Duration on the Encoding Overhead

Since our proposal is to use video segments with a sub-second duration, it is important to analyze the induced encoding overhead. To perform this analysis, eight different videos are considered: Big Buck Bunny[1] (BBB), Earth from Space[2] (EFS), Netflix' El Fuente[3] (NEF), Sintel[4] (STL), Tears of Steel[5] (ToS), Elephant's Dream[6] (ED), a benchmark performing testing video on Forza Motorsport 6 Apex[7] (FZA) and a 10-minute part of a soccer game from the 2014 World Championship[8] (SOC). All content is available in Full HD (1920 $\times$ 1080 px), and comes with a minimal frame rate of

[1] https://peach.blender.org/
[2] https://www.youtube.com/watch?v=n4IhCSMkADc
[3] https://www.netflix.com/title/70297450
[4] https://durian.blender.org/
[5] https://mango.blender.org/
[6] https://orange.blender.org/
[7] https://www.youtube.com/watch?v=d-Jgf6rtEg8
[8] https://www.youtube.com/watch?v=qOHd20F0e9k

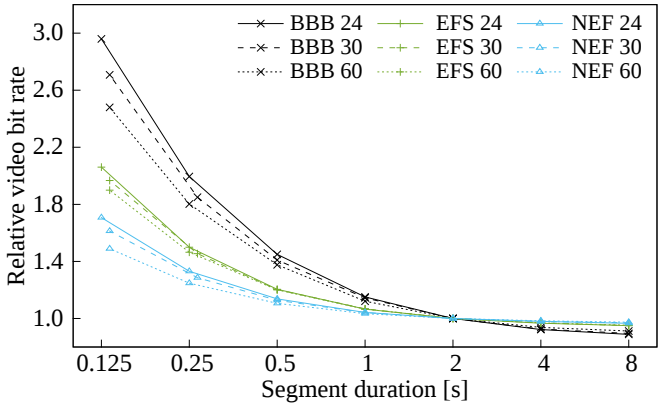


Figure 2.5: Encoding bit rate relative to a segment duration of 2 s as a function of the segment duration, for frame rates of 24, 30 and 60 FPS.

24 FPS. In our analysis, segment durations from 125 ms up to 8 s are considered, using the FFmpeg library[9] to segment the video. To allow each segment to be decoded independently, every segment starts with an IDR frame and the Group of Pictures (GOP) length is set accordingly. To assess the impact of shorter GOP lengths on the compression performance, the encodings for different segment durations are set to target the same visual quality and allow a subsequent overhead in the achieved nominal bit rate. To realize this, the Constant Rate Factor (CRF) rate control implemented in the x264 encoder[10] is used. The obtained encodings for the same nominal rates but different segment durations, have the same visual quality, measured in terms of Peak Signal-to-Noise Ratio (PSNR), with deviations smaller than 0.205 dB for all content bar BBB (1.045 dB).

Figure 2.5 shows the encoding bit rate relative to a segment duration of 2 s, for the BBB, EFS and NEF content provided in Full HD with frame rates of 24, 30 and 60 FPS. The encoding overhead is significantly lower for higher frame rates, which is a consequence of the higher GOP length: for a segment duration of 500 ms, a GOP length of 12, 15 and 30 is used for 24, 30 and 60 FPS respectively, resulting in relatively less IDR frames. The obtained results are also strongly content-dependent: an overhead of 70.8% is obtained for NEF at 24 FPS, while an overhead of 195.9% is obtained for BBB. This is because the former is a video showing actually captured footage with high-detail shots, whereas the latter is synthetically produced 3D video.

[9] https://ffmpeg.org
[10] http://www.videolan.org/developers/x264.html

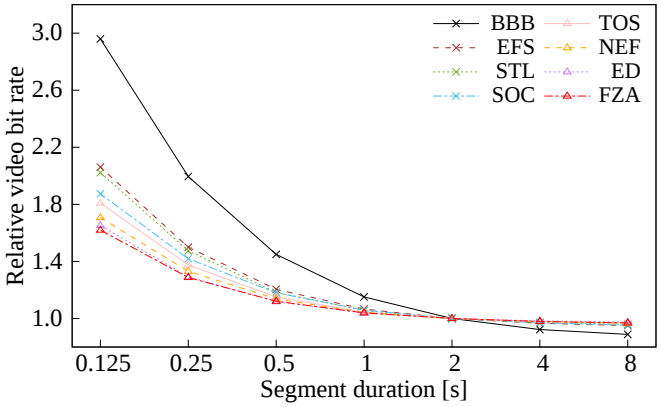


Figure 2.6: Encoding bit rate relative to a segment duration of 2 s as a function of the segment duration, for a frame rate of 24 FPS.

Figure 2.6 shows the relative encoding bit rate for all considered videos, provided in Full HD at 24 FPS. For all content bar BBB, the encoding overhead is between 62.0% and 106.1% for a segment duration of 125 ms, between 28.7% and 49.9% for 250 ms, between 12.1% and 20.4% 500 ms and between 3.9% and 6.6% for 1 s. Based on these values, we conclude that a minimal segment duration of 500 ms should be used in order to limit the encoding overhead and achieve a similar video quality when the available bandwidth is constrained. Evaluation results in the next section will confirm this conclusion, yet also show that lower segment durations can result in an even lower video startup time and end-to-end delay. Since our focus is on live streaming scenarios, the soccer video will be used to evaluate results for the push-based approach. For this video, average overhead values of 87.4%, 41.9%, 18.0% and 5.8% are obtained for a segment duration of 125 ms, 250 ms, 500 ms and 1 s respectively. A frame rate of 24 FPS is used to show the minimal gains of the approach; even better results can be achieved with a frame rate of 30 or 60 FPS, currently supported by most live stream providers .

2.5 Evaluation and Discussion

To illustrate the possible gains of the push-based approach presented in Section 2.3, results are evaluated for different network conditions and video segment durations. The experimental setup is presented below, followed by a discussion on the evaluated rate adaptation heuristics and a detailed overview of the obtained results.

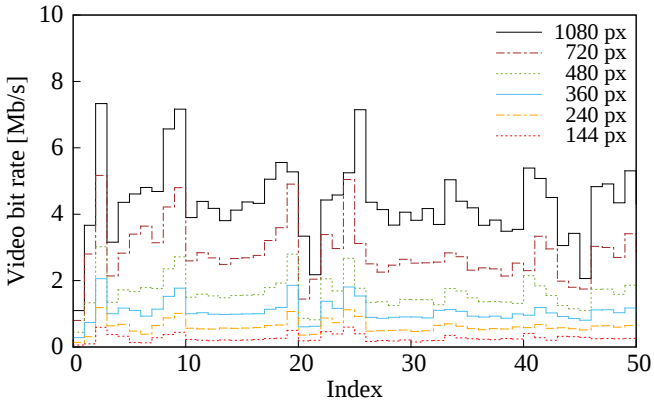


Figure 2.7: Video encoding bit rates for the first 50 segments of the Soccer video, for a segment duration of 2 s.

2.5.1 Experimental Setup

The considered video in our evaluation is a part of a soccer game, which has a total length of 596 seconds and comes with a frame rate of 24 FPS. Similar as in the previous section, the video is segmented using segment durations ranging from 125 ms to 8 s, corresponding to GOP lengths ranging from 3 to 192. The video is encoded at six quality levels, with resolutions of 256×144, 428×240, 640×360, 856×480, 1280×720 and 1920×1080 px, using x264 with a CRF of 23. Note that there's a lot of variability among bit rates for the different segments, as illustrated in Figure 2.7 for a segment duration of 2 s.

To stream the content, the network topology in Figure 2.8 is emulated using the Mininet framework[11]. It consists of 30 clients, streaming video from a dedicated HAS server. To evaluate the proposed approaches in a realistic scenario, bandwidth and latency patterns are applied for every client. Patterns for the available bandwidth are extracted from an open-source dataset, collected by Riiser et al. on a real 3G/HSDPA network [38]. The average available bandwidth in the 30 traces is 2358 kb/s, with a standard deviation of 1387 kb/s. The minimum available bandwidth is set to 300 kb/s, in order to provide the minimal video streaming service. An example trace is shown in Figure 2.9a, illustrating the high bandwidth variability over time. As for latency, first evaluations are performed with a statically defined value ranging between 0 and 300 ms in order to assess its impact on the performance of a number of rate adaptation heuristics and the optimality of the number of segments in flight k. In a final evaluation,

[11] http://mininet.org/

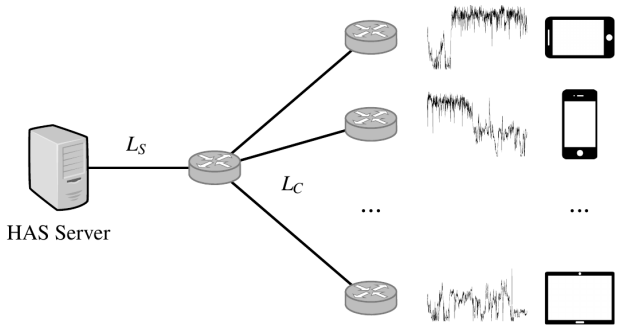
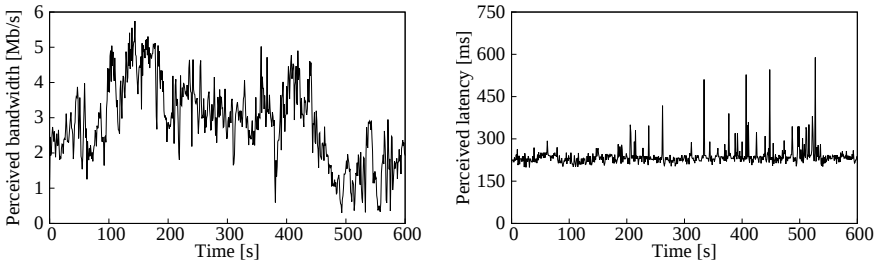


Figure 2.8: Experimental Mininet setup.



(a) Perceived bandwidth (b) Perceived latency

Figure 2.9: Perceived bandwidth in one of the considered 3G/HSDPA traces [38], along with the perceived latency collected in a 3G/HSPA network.

the latency is dynamically changed according to measurements performed in a real 3G/HSPA network, illustrated in Figure 2.9b. To collect a trace for the perceived latency, we hosted a dedicated server in iLab.t's Virtual Wall infrastructure[12]. Sending a ping request every second from a smartphone connected over 3G/HSPA, the perceived latency was measured and logged for a total of 17 minutes. For each client, the trace is looped and latency values are set according to a random starting point. The average latency is 232.2 ms, with a standard deviation of 12.5 ms.

Values for variable bandwidth and latency are introduced through traffic shaping with *tc* on the client-side switches. The bandwidth on links L_C is set to 10 Mb/s, while the bandwidth on the link L_S is set to $30 \times L_C = 300$ Mb/s. Since the maximum bandwidth in the traces is 6.2 Mb/s, the bottleneck thus resides with the clients and the actual bandwidth corresponds to the bandwidth perceived at the time of trace collection.

[12]https://doc.ilabt.imec.be/ilabt-documentation/

2.5.1.1 Server-Side Implementation

The HAS server implementation is based on the Jetty web server[13], which was recently expanded to provide support for HTTP/2. Jetty's HTTP/2 component allows to define a push-based strategy, which defines all resources that need to be pushed along with the requested resource. Such a strategy is ideal for web-based content, where the required JavaScript and CSS files, images and other content can immediately be pushed. However, since we target a live stream scenario, not all segments are available when the initial request is issued. Therefore, we defined a new request handler that processes GET requests issued by the client. This handler allows a client to issue a live stream request, passing along parameters such as the temporal buffer size and quality level. When this request corresponds to a new session, the server starts a push thread that pushes the m last released video segments at the lowest quality, where m corresponds to the number of segments that fit in the indicated buffer. In order to simulate a live stream scenario, a release thread makes new segments available every segment duration. As soon as a new segment is available, the push thread is notified and the segment is pushed to the corresponding client. When the client wants to change the quality level at which the segments are pushed, a new request is issued and the quality level is updated at the server-side accordingly.

2.5.1.2 Client-Side Implementation

The HAS client is implemented on top of the libdash library[14], the official reference software of the ISO/IEC MPEG-DASH standard. To make use of the server push provided by HTTP/2, a number of changes are made. First, an HTTP/2-based connection is added to enable the reception of pushed segments. The nghttp2 library[15] is used to set up an HTTP/2 connection over SSL. Second, the rate adaptation heuristic is modified to recalculate the quality level every time the push stream of a pushed segment is closed. While a GET request is required for every segment in HTTP/1.1, no request is sent in the HTTP/2-based scheme if no quality change is required. Third, the perceived bandwidth is estimated based on the elapsed time between the reception of the push promise and the time the segment is available. In HTTP/1.1, this estimation is based on the total download time, which naturally includes the time to send the GET request. Note that by default, the available bandwidth is estimated by the Exponentially Weighted Moving

[13] https://webtide.com/
[14] https://github.com/bitmovin/libdash/
[15] https://nghttp2.org/

Average (EWMA) over the observed download rates. It is worth noting that major browsers such as Google Chrome, Moxilla Firefox and Internet Explorer all provide support for HTTP/2 [11]. Although a standalone client is used in this chapter, a browser-based dash.js reference player has been released by libdash as well[16]. Provided that some changes to the implementation are made, the proposed approach can thus be applied in any browser with support for HTTP/2.

2.5.2 Rate Adaptation Heuristics

To achieve a baseline for adaptive streaming over HTTP/1.1, a number of state-of-the-art rate adaptation heuristics were embedded at the client-side. Evaluated heuristics are the MSS heuristic developed by Microsoft [39], the FESTIVE heuristic developed by Jiang et al. [40] and the FINEAS heuristic[17] developed by Petrangeli et al. [15].

In the MSS heuristic, the next quality level is selected based on the buffer filling and the perceived bandwidth. The most important parameters are the buffer size and the panic, lower and upper thresholds, which actively steer the buffer filling towards a value between the lower and upper threshold. A lower quality level is selected when the buffer filling drops below the lower threshold, while a higher quality level is selected when the buffer filling exceeds the upper threshold. When the buffer filling is lower than the panic threshold, the rate adaptation heuristic immediately selects the lowest quality level, in an attempt to avoid buffer starvation. In accordance with conclusions drawn by Famaey et al., panic, lower and upper thresholds of respectively 25%, 40% and 80% of the total buffer size were selected [41].

In the FESTIVE heuristic, the first 20 segments are downloaded at the lowest quality, because initially there is not enough information on the available bandwidth [40]. From segment 21 on, the rate adaptation heuristic computes a reference quality level q_{ref} based on the available bandwidth (defined as the harmonic mean of the download rates for the last 20 segments) and the last selected quality level q_{cur}. A gradual switching strategy is applied, so that the reference quality can only switch to the next lower or higher quality layer. Furthermore, a switch from quality level m to $m+1$ can only occur once at least m segments have been downloaded at quality m. Given q_{cur} and q_{ref}, the heuristic then calculates a certain metric

[16] http://dashif.org/reference/players/javascript/1.4.0/samples/dash-if-reference-player/

[17] The in-network computation performed by the authors has not been implemented in this work.

score for the session's stability and efficiency, and finally selects the most appropriate quality level.

In the FINEAS heuristic, the goal is to maximize the user's QoE [15]. This is achieved by intelligently selecting the next quality level, pursuing a high average quality level while limiting the number of quality switches and avoiding play-out freezes. Quality selection is based on a utility function, designed to be a metric for the QoE. Used parameter values for this evaluation are a panic threshold of two segments, a targeted buffer filling of 80% and a quality window of 70 seconds, selected after tweaking the heuristic for a buffer size of 10 s. Note that, in order to limit the number of switches for super-short segments, upward switching is only allowed every 2 seconds.

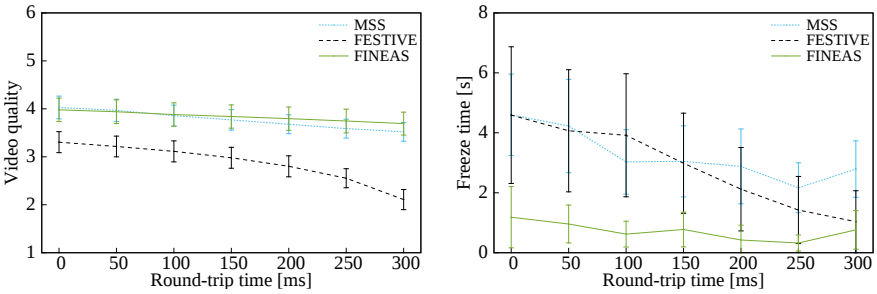
2.5.3 Evaluation Metrics

The following evaluation metrics are considered. First, the average video quality, expressed as the average quality level (1-7) perceived during a streaming session. Second, buffer starvation, defined by the frequency and total duration of playout freezes. Third, the initial startup delay, which is defined as the time between requesting the live video at the client-side and the playout of the first video segment. Fourth, the server-to-display delay, defined as the time between the release of a segment at the server-side and its playout at the client-side, plus the segment duration. This is not the same as the camera-to-display delay, as the content is not captured in real-time. Note that segments are not skipped during playout, so every freeze results in a larger buffer and total server-to-display delay. When no freezes occur, the initial and final server-to-display delay are the same.

To evaluate the impact of the proposed approach on these metrics, the considered video is streamed by 30 different clients; results are therefore shown using the observed averages and the corresponding 95% confidence intervals.

2.5.4 Evaluation Results

In this Section, we first show results for HTTP/1.1, highlighting the advantages that the proposed push-based approach can bring. Second, a parameter analysis is performed to find the optimal value for the parameter k, based on the segment duration and the network's RTT. Third, the impact of the buffer size on the QoE is evaluated, showing results for the average video quality, freeze time, startup time and server-to-display delay. Finally, a detailed comparison of results is made between the proposed HTTP/2-based approach and traditional HAS over HTTP/1.1.

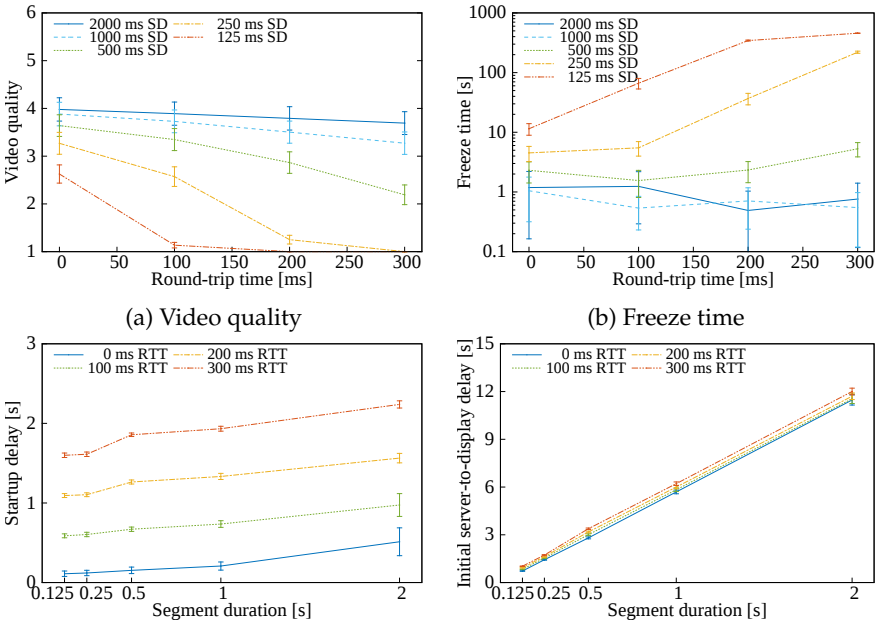


(a) Video quality (b) Freeze time

Figure 2.10: Average video quality and freeze time for the MSS, FESTIVE and FINEAS heuristics, as a function of the RTT for a segment duration of 2 s and a buffer size of 10 s.

2.5.4.1 HAS over HTTP/1.1

With respect to the considered rate adaptation heuristics, Figure 2.10 shows the average video quality and freeze time for MSS, FESTIVE and FINEAS as a function of the latency. The MSS heuristic decreases the video quality in a conservative way, only allowing a decrease of one level at a time as long as the panic threshold is not exceeded. Because of this, reaction to changes in the available bandwidth are generally slow, resulting in a relatively large average freeze time. In the FESTIVE heuristic, the video quality is decreased one level at a time as well. Furthermore, the estimated bandwidth is based on the harmonic mean over the last 20 samples, instead of the more aggressive EWMA. Therefore, the average freeze time is similar to the one for MSS. The average video quality is however significantly lower, because even with sufficient bandwidth it takes a long time for the heuristic to reach the highest quality: 20 segments are first requested at the lowest quality, another 20 are requested at the intermediary levels (2-6). The FINEAS heuristic allows to change the quality level with more than one level at a time, resulting in a more aggressive reaction to changes in the available bandwidth. Furthermore, buffer starvation is actively prevented by taking into account both the current buffer filling and the estimated download times of future segments. Comparing results with MSS, a similar average video quality is observed, yet with a significantly lower average freeze time. FINEAS thus leads to the best results, and will be used to evaluate the proposed HTTP/2-based approach in the following sections.



(a) Video quality

(b) Freeze time

(c) Startup delay

(d) Initial server-to-display delay

Figure 2.11: Impact of the RTT on the average video quality and freeze time for the FINEAS heuristic (top), and the impact of the segment duration (SD) on the startup delay and server-to-display delay (bottom). While super-short segments can result in a lower startup and end-to-end delay, they result in a low video quality and high freeze time for high RTTs.

Figure 2.11 shows results for the FINEAS heuristic over HTTP/1.1, for an increasing RTT and for all considered segment durations. From Figure 2.11a, it is clear that super-short segments suffer significantly from large RTTs: for a segment duration of 125 ms, the lowest quality is always selected for RTTs above 200 ms. For a segment duration of 2 s however, the average video quality is only reduced with 7.2% for an RTT of 300 ms. Looking at the total freeze time in Figure 2.11b, a significant increase is observed for a segment duration of 125 ms and RTTs above 200 ms: in this case, a freeze inevitably occurs for every downloaded video segment. Furthermore, since the buffer size is five times the segment duration (and thus only 625 ms for a segment duration of 125 ms), the freeze time for super-short segments is relatively high, even when a negligible RTT is considered. From these results, we conclude that a segment duration of 2 s should be used in order to achieve an appropriate video quality and freeze time in

Table 2.1: Evaluated configurations for the parameter k.

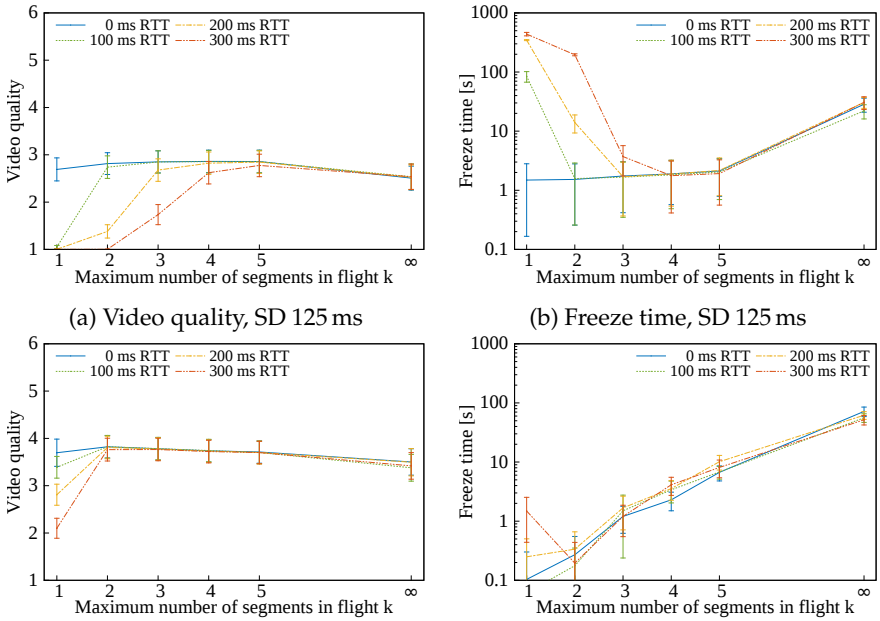
Parameter	Evaluated values
Segment duration [s]	0.125, 0.25, 0.5, 1, 2
Latency [ms]	0, 100, 200, 300
k	1, 2, 3, 4, 5, ∞

mobile, high-RTT networks. Note that similar results are achieved for the MSS and FESTIVE heuristics, omitted here due to space constraints.

Despite these disadvantages, there are clear advantages as well. For one, Figure 2.11c shows that the startup delay increases significantly for larger segment durations. This is because more data needs to be put on the wire for the first video segment, and because the underlying TCP slow start requires multiple RTT cycles to transfer the required files from server to client. More importantly, Figure 2.11d shows that the initial server-to-display delay increases linearly with the buffer size. Since both the startup and the end-to-end delay are important factors in live video streaming, our goal is to realize the advantages of super-short segments, while still providing the content at an acceptable video quality. The following sections will therefore focus on the applicability of super-short segments, using the proposed push-based approach to avoid idle RTT cyles.

2.5.4.2 Impact of the Parameter k

In Section 2.3.2, Equation 2.1 was proposed as a rule of thumb for the maximum number of segments in flight k. To test the validity of this equation, all combinations of the parameter configurations in Table 2.1 were evaluated, applying latency to the links L_C in the experimental setup. Figures 2.12a and 2.12c show the impact of the value of k on the average quality, for a segment duration of 125 and 500 ms respectively. In the former case, a total of two, three and four segments must be pushed for an RTT of 100, 200 and 300 ms respectively, in order to achieve a video quality similar as for a negligible RTT. In the latter case, however, it is sufficient to push a maximum of two segments at the same time. With respect to the average freeze time, shown in Figures 2.12b and 2.12d, two observations can be made. First, for super-short segments, the average freeze time is significantly reduced when multiple segments are pushed. This is because bandwidth is used more efficiently, so that no freeze occurs for every single segment. For a segment duration of 500 ms, however, a reduction is less apparent. Second, once a certain threshold for k is exceeded, the average freeze time increases significantly. This is because more segments



(a) Video quality, SD 125 ms

(b) Freeze time, SD 125 ms

(c) Video quality, SD 500 ms

(d) Freeze time, SD 500 ms

Figure 2.12: Impact of the RTT and impact of the parameter k on the average video quality and freeze time, for the FINEAS heuristic with a segment duration of 125 ms (top) and 500 ms (bottom). The optimal value for k depends both on the segment duration and the network's RTT, with higher values for lower segment durations and high RTTs.

are allowed to be in flight, resulting in a slower quality reduction when sudden bandwidth drops occur. This is in line with the reasoning in Section 2.3.2, where the maximum number of segments in flight is based on the segment duration and the network's RTT, and shows that the initial approach ($k = \infty$) is not suitable in mobile, highly variable networks.

For all combinations of the evaluated segment durations and RTTs in Table 2.1, the lowest value for k was selected that resulted in a comparable video quality compared to a scenario with negligible RTT. Figure 2.13 shows the obtained values for these configurations, along with the output of Equation 2.1. A nearly perfect match is obtained for $a = 0.2$, showing the suitability of the proposed rule of thumb. In the remainder of this chapter, a value of $k = 2$ will be used to push video segments with a segment duration of 500 ms.

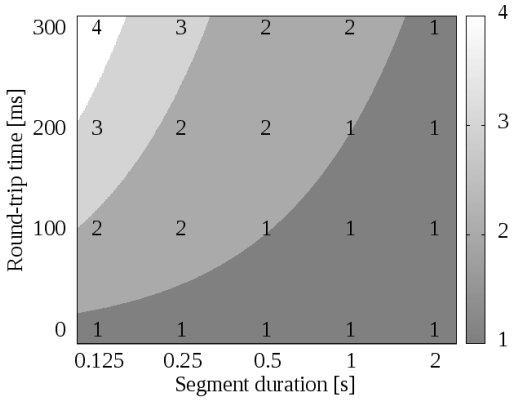


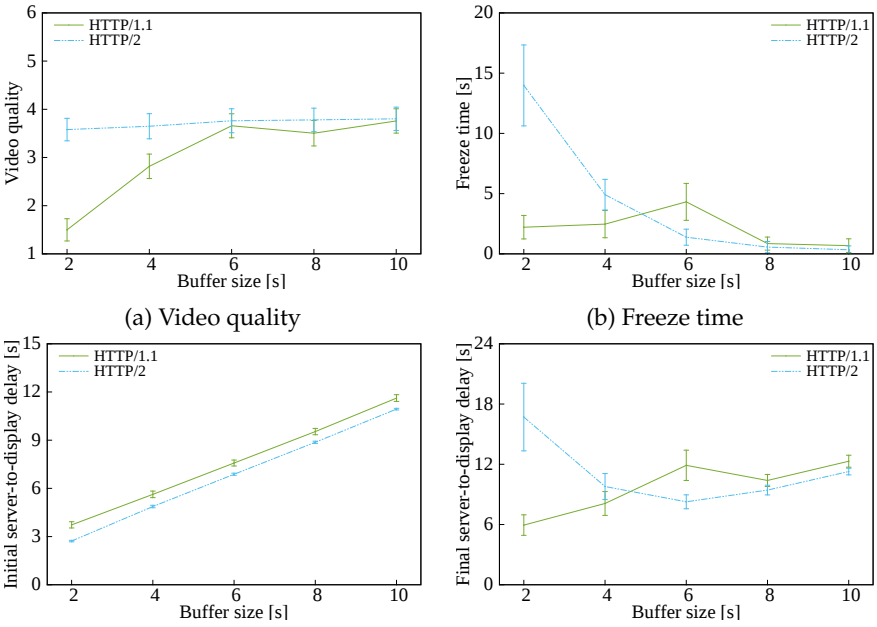
Figure 2.13: Optimal values for the parameter k in the configurations of Table 2.1, along with the output of Equation 2.1 for a value of $a = 0.2$ in colors from dark grey ($k = 1$) to white ($k = 4$).

2.5.4.3 Impact of the Buffer Size

Having analyzed different heuristics and optimal values for the parameter k as a function of the latency, we illustrate the gains of the proposed approach below. In this experiment, we analyze the impact of the buffer size on the different evaluation metrics, and show that the proposed approach allows to reduce the buffer size and the end-to-end delay significantly. To this end, temporal buffer sizes ranging from 2 s to 10 s are used. A segment duration of 2 s is used for HTTP/1.1, while a segment duration of 500 ms is used for HTTP/2 ($k = 2$). In the former case, panic thresholds and targeted buffer filling are selected based on observations by Petrangeli et al., with values presented in Table 2.2 [15]. In the latter case, multiples of 500 ms are selected for the panic threshold and buffer target respectively. Note again that the maximum buffer size can change over time: no segments are

Table 2.2: Evaluated panic thresholds (PT) and targeted buffer filling (BF) for the FINEAS heuristic, both for HTTP/1.1 (1) and HTTP/2 (2) with different values of the buffer size.

Buffer size [s]	PT-1 [s]	BF-1 [s]	PT-2 [s]	BF-2 [s]
2	2	2	1	1.5
4	2	2	2	3
6	4	4	2.5	4.5
8	4	6	3.5	6
10	4	8	4	8



(a) Video quality

(b) Freeze time

(c) Initial server-to-display delay

(d) Final server-to-display delay

Figure 2.14: Impact of the buffer size on the video quality, the freeze time and the server-to-display delay for the FINEAS heuristic, for an RTT of 300 ms. A segment duration of 2 s is used for HTTP/1.1, while a segment duration of 500 ms is used for HTTP/2 ($k = 2$).

skipped when playout freezes occur, thus increasing the end-to-end delay and the amount of content the server can push to the client.

Figure 2.14a shows the average video quality in terms of the initial buffer size. For HTTP/1.1, the lowest quality is most frequently selected for a buffer size of 2 s, since the panic threshold is almost always exceeded. For higher buffer sizes, the average quality gradually increases. For HTTP/2, however, the average video quality is almost unaffected by the buffer size. Figure 2.14b shows the average freeze time per episode, indicating a decreasing trend for larger buffer sizes. When a buffer size of 2 or 4 s is used, a large number of playout freezes occur for HTTP/2. This is because the rate adaptation heuristic can download the video at high quality (the panic threshold is often not exceeded), yet a buffer this small cannot cope with the high variability of the perceived bandwidth and the video segment's size. This is less of a problem for HTTP/1.1, simply because the heuristic most often downloads the video at the lowest video quality. Once the buffer threshold exceeds 6 s, higher quality rep-

resentations can be downloaded both for HTTP/1.1 and HTTP/2. Since the latter allows to react faster to changes in the available bandwidth, a lower playout freeze time is observed. The initial server-to-display delay is shown in Figure 2.14c, increasing linearly with the selected buffer size. Figure 2.14d, on the other hand, shows the server-to-display delay at the end of the video streaming session. Since no segments are skipped in our setup, every playout freeze results in an increase of the end-to-end delay. Therefore, the total delay corresponds to the sum of the initial delay and the episode's total freeze time. Even though a buffer size in the order of a few seconds can be used to reduce the initial delay, it cannot cope with the highly variable bandwidth and segment sizes. As a result, the total delay is significantly higher than for a buffer size of 6 s.

From these results, we conclude that a minimal buffer size of 10 s is required for HTTP/1.1, in order to obtain an acceptable video quality with a relatively low amount of playout freezes. Because of the shorter segment duration, a buffer size of 6 s can be used for HTTP/2, reducing the end-to-end delay while limiting the total freeze duration.

2.5.4.4 Discussion

Table 2.3 summarizes results for traditional HAS over HTTP/1.1 and our proposed push-based approach. The average quality for MSS and FESTIVE is significantly lower than for the FINEAS heuristic (-3.9% and -32.4% respectively), while the average quality for the proposed push-based approach is more or less the same. Furthermore, while MSS results in an average freeze time close to 3 s, the FINEAS heuristics results in a freeze time well below 2 s both for HTTP/1.1 and HTTP/2. With respect to the video quality and freeze time, we thus conclude that both FINEAS approaches perform well. For the startup delay, however, a reduction of 0.57 s (-31.2%) is obtained. This is because the first segment is immediately pushed along with the MPD, and because TCP slow startup requires fewer RTT cycles to transfer the first, super-short segment. As for the server-to-display delay, a reduction of 4.04 s (-32.9%) is obtained, which is beneficial in a live streaming scenario. Given that significantly better results are obtained for the startup and server-to-display delay, and similar results are obtained for the video quality and freeze time, we conclude that the proposed push-based approach with super-short segments can significantly improve the QoE for live streaming in this scenario.

The reported results specifically target a mobile 3G/HSPA network, with relatively high values for the perceived latency. Compared to these networks, 4G/LTE networks allow to significantly reduce latency. For Belgian providers such as Base, Proximus and Mobistar, OpenSignal reports

Table 2.3: Performance summary for the different heuristics and the proposed push-based approach. Average values are reported, along with the 95% confidence intervals. Note that a segment duration of 2 s and a buffer size of 10 s are used for HTTP/1.1, while a segment duration of 500 ms and a buffer size of 6 s are used for HTTP/2. The term "server-to-display" is abbreviated to S2D.

Approach	Heuristic	Video quality	Freeze time [s]	Startup time [s]	S2D delay [s]
HTTP/1.1	MSS	3.61 ± 0.20	2.68 ± 1.00	1.82 ± 0.13	14.31 ± 1.04
HTTP/1.1	FESTIVE	2.54 ± 0.21	1.54 ± 1.09	1.82 ± 0.12	13.32 ± 1.14
HTTP/1.1	FINEAS	3.76 ± 0.25	0.68 ± 0.57	1.82 ± 0.13	12.30 ± 0.60
HTTP/2	FINEAS	3.76 ± 0.25	1.38 ± 0.67	1.25 ± 0.04	8.26 ± 0.69

a reduction from 214 to 94 ms, from 241 to 64 ms and from 229 to 77 ms respectively [6]. Although the proposed approach should still result in lower startup times and end-to-end delay, the gains will be significantly lower. Highly congested networks, in contrast, introduce additional latency through network queueing. This can have a significant impact on the overall perceived latency, and depending on the variability of the network load, result in packet delay variation. In such a scenario, the parameter k could be changed dynamically, similar to the approach proposed by Wei et al. [35].

It should be noted that results are presented for H.264-encoded video with a frame rate of 24 FPS. As shown in Section 2.4, using higher frame rates results in a lower relative encoding overhead for the same segment duration, thus reducing its impact on shorter video segments. The encoding bit rate however increases, which might be unsuitable under certain network conditions. An alternative can be provided through H.265, which in some cases allows to reduce the encoding bit rate by half compared to H.264 [42]. Numerous other possibilities exist within adaptive streaming, but cannot all be considered in this evaluation.

2.6 Conclusions and Future Work

In this chapter, focus is on improving the user's Quality of Experience (QoE) in HTTP adaptive streaming (HAS) in mobile, high round-trip time (RTT) networks. To this end, we proposed a novel push-based approach for HAS over the recently standardized HTTP/2 protocol. Using this approach, available video segments are actively pushed from server to client, effectively eliminating idle RTT cycles. This enables the use of segments with a sub-second duration, referred to as *super-short* segments, which not only allow the client to start the video playout faster, but also allow to sig-

nificantly reduce the buffer size and thus the end-to-end delay. In contrast with previous work, we proposed a means to effectively limit the amount of segments in flight, and performed an analysis of its optimal value for different segment durations and network latencies. Furthermore, we discussed in detail the encoding overhead introduced by shorter segments, and evaluated the proposed approach using highly variable bandwidth and latency traces collected in real 3G networks. Using a segment duration of 500 ms and a buffer of a mere 6 s, reductions of 31.2% and 32.9% are obtained for the startup time and the server-to-display delay respectively, compared to traditional HAS over HTTP/1.1 with a segment duration of 2 s and a buffer size of 10 s.

Future work includes a real-life study of the overall approach using subjective measurements, such as the Mean Opinion Score, which could reveal interesting insights. An interesting future research path includes maximizing the average QoE and fairness for competing HAS clients in congested networks, using a dynamic approach to change the parameter k based on changing network conditions.

References

[1] Sandvine Incorporated. *Global Internet Phenomena Report*. https://www.sandvine.com/downloads/general/global-internet-phenomena/2016/global-internet-phenomena-report-latin-america-and-north-america.pdf, 2016. Accessed 21 February 2017.

[2] R. Mok, E. Chan, and R. Chang. *Measuring the Quality of Experience of HTTP Video Streaming*. In IFIP/IEEE International Symposium on Integrated Network Management, pages 485–492, 2011.

[3] T. Stockhammer. *Dynamic Adaptive Streaming over HTTP: Standards and Design Principles*. In Proceedings of the 2nd Annual ACM Conference on Multimedia Systems, pages 133–144, 2011.

[4] S. Wei and V. Swaminathan. *Low Latency Live Video Streaming over HTTP 2.0*. In Proceedings of the Network and Operating System Support on Digital Audio and Video Workshop, pages 37:37–37:42. ACM, 2014.

[5] G. Van Wallendael, W. Van Lancker, J. De Cock, P. Lambert, J.-F. Macq, and R. Van de Walle. *Fast Channel Switching Based on SVC in IPTV Environments*. Broadcasting, IEEE Transactions on, 58(1):57–65, 2012.

[6] OpenSignal. *IConnect 4G Coverage Maps*. http://opensignal.com/networks/usa/iconnect-4g-coverage/, 2016. Accessed 21 February 2017.

[7] Igvita. *High Performance Browser Networking*. https://www.igvita.com/2014/03/26/why-is-my-cdn-slow-for-mobile-clients/, 2014. Accessed 21 February 2017.

[8] M. Belshe, R. Peon, M. Thomson, and A. Melnikov. *SPDY Protocol*. https://tools.ietf.org/html/draft-ietf-httpbis-http2-00/, 2012. Accessed 21 February 2017.

[9] IETF. *Hypertext Transfer Protocol (httpbis)*. https://datatracker.ietf.org/wg/httpbis/charter/, 2012. Accessed 21 February 2017.

[10] M. Belshe, R. Peon, and M. Thomson. *Hypertext Transfer Protocol Version 2*. https://datatracker.ietf.org/doc/rfc7540/, 2015. Accessed 21 February 2017.

[11] A. Deveria. *Can I Use HTTP/2?* http://caniuse.com/#search=HTTP%2F2, 2016. Accessed 21 February 2017.

[12] R. Huysegems, J. van der Hooft, T. Bostoen, P. Alface, S. Petrangeli, T. Wauters, and F. De Turck. *HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming*. In Proceedings of the 23rd ACM Multimedia Conference. ACM, 2015.

[13] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia. *A Survey on Quality of Experience of HTTP Adaptive Streaming*. IEEE Communications Surveys Tutorials, 17(1):469–492, 2015.

[14] S. Benno, A. Beck, J. Esteban, L. Wu, and R. Miller. *WiLo: A Rate Determination Algorithm for HAS Video in Wireless Networks and Low-Delay Applications*. In IEEE Globecom Workshops, pages 512–518, 2013.

[15] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turck. *QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming*. ACM Transactions on Multimedia Computing, Communications and Applications, 12(2):28:1–28:24, 2015.

[16] V. Menkovski and A. Liotta. *Intelligent Control for Adaptive Video Streaming*. In IEEE International Conference on Consumer Electronics, pages 127–128, 2013.

[17] M. Claeys, S. Latré, J. Famaey, and F. De Turck. *Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client*. Communications Letters, IEEE, 18(4):716–719, 2014.

[18] Y. Sánchez de la Fuente, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, and Y. Le Louédec. *iDASH: Improved Dynamic Adaptive Streaming over HTTP Using Scalable Video Coding*. In Proceedings of the 2nd Annual ACM Conference on Multimedia Systems, pages 257–264. ACM, 2011.

[19] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. Begen. *Server-Based Traffic Shaping for Stabilizing Oscillating Adaptive Streaming Players*. In Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, pages 19–24. ACM, 2013.

[20] L. De Cicco, S. Mascolo, and V. Palmisano. *Feedback Control for Adaptive Live Video Streaming*. In Proceedings of the 2nd Annual ACM Conference on Multimedia Systems, pages 145–156. ACM, 2011.

[21] N. Bouten, J. Famaey, S. Latré, R. Huysegems, B. Vleeschauwer, W. Leekwijck, and F. Turck. *QoE Optimization Through In-Network Quality Adaptation for HTTP Adaptive Streaming*. In 8th International Conference on Network and Service Management, pages 336–342, 2012.

[22] N. Bouten, M. Claeys, S. Latré, J. Famaey, W. Van Leekwijck, and F. De Turck. *Deadline-Based Approach for Improving Delivery of SVC-Based HTTP Adaptive Streaming Content*. In IEEE Network Operations and Management Symposium, pages 1–7, 2014.

[23] S. Petrangeli, M. Claeys, S. Latré, J. Famaey, and F. De Turck. *A Multi-Agent Q-Learning-Based Framework for Achieving Fairness in HTTP Adaptive Streaming*. In IEEE Network Operations and Management Symposium, pages 1–9, 2014.

[24] S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. *Network-Based Dynamic Prioritization of HTTP Adaptive Streams to Avoid Video Freezes*. In IFIP/IEEE International Symposium on Integrated Network Management, pages 1242–1248, 2015.

[25] T. Schierl, C. Hellge, S. Mirta, K. Grüneberg, and T. Wiegand. *Using H.264/AVC-based Scalable Video Coding (SVC) for Real Time Streaming in Wireless IP Networks*. In IEEE International Symposium on Circuits and Systems, pages 3455–3458, 2007.

[26] E. Thomas, M. van Deventer, T. Stockhammer, A. Begen, and J. Famaey. *Enhancing MPEG DASH Performance via Server and Network Assistance*. In Proceedings of the IBC 2015 Conference, 2015.

[27] S. Latré and F. De Turck. *Joint In-Network Video Rate Adaptation and Measurement-Based Admission Control: Algorithm Design and Evaluation*. Journal on Network and Systems Management, 21(4):588–622, 2013.

[28] S. Badukale and W. P. *SPDY: An Experimental Protocol for a Faster Web*. http://www.chromium.org/spdy/spdy-whitepaper/, 2009. Accessed 21 February 2017.

[29] A. Cardaci, L. Caviglione, A. Gotta, and N. Tonellotto. *Performance Evaluation of SPDY over High Latency Satellite Channels*. In Personal Satellite Services, volume 123, pages 123–134. Springer International Publishing, 2013.

[30] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan. *Towards a SPDY'Ier Mobile Web?* In Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies, pages 303–314. ACM, 2013.

[31] Y. Elkhatib, G. Tyson, and M. Welzl. *Can SPDY Really Make the Web Faster?* In IFIP Networking Conference, pages 1–9. IEEE, 2014.

[32] X. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. *How Speedy is SPDY?* In Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, pages 387–399. USENIX Association, 2014.

[33] S. Chowdhury, V. Sapra, and A. Hindle. *Is HTTP/2 More Energy Efficient than HTTP/1.1 for Mobile Users?* PeerJ PrePrints, 3:e1571, 2015.

[34] C. Müller, S. Lederer, C. Timmerer, and H. Hellwagner. *Dynamic Adaptive Streaming over HTTP/2.0*. In IEEE International Conference on Multimedia and Expo, pages 1–6, 2013.

[35] S. Wei and V. Swaminathan. *Cost Effective Video Streaming Using Server Push over HTTP 2.0*. In IEEE 16th International Workshop on Multimedia Signal Processing, pages 1–5, 2014.

[36] W. Cherif, Y. Fablet, E. Nassor, J. Taquet, and Y. Fujimori. *DASH Fast Start Using HTTP/2*. In Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, pages 25–30. ACM, 2015.

[37] J. van der Hooft, S. Petrangeli, N. Bouten, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. *An HTTP/2 Push-Based Approach for SVC Adaptive Streaming*. In IEEE/IFIP Network Operations and Management Symposium, pages 104–111, 2016.

[38] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen. *Video Streaming Using a Location-Based Bandwidth-Lookup Service for Bitrate Planning*. ACM Transactions on Multimedia Computing, Communications and Applications, 8(3):24:1–24:19, 2012.

[39] A. Zambelli. *IIS Smooth Streaming Technical Overview*. https://www.iis.net/learn/media/on-demand-smooth-streaming/smooth-streaming-technical-overview/, 2009. Accessed 21 February 2017.

[40] J. Jiang, V. Sekar, and H. Zhang. *Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming with Festive*. IEEE/ACM Transactions on Networking, 22(1):326–340, 2014.

[41] J. Famaey, S. Latré, N. Bouten, W. Van de Meerssche, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. *On the Merits of SVC-based HTTP Adaptive Streaming*. In IFIP/IEEE International Symposium on Integrated Network Management, pages 419–426, 2013.

[42] G. Sullivan, J. Ohm, H. W., and T. Wiegand. *Overview of the High Efficiency Video Coding (HEVC) Standard*. IEEE Transactions on Circuits and Systems for Video Technology, 22(12):1649–1668, 2012.

Addendum

Since the start of this PhD research, a significant amount of contributions have been made with respect to fast video startup and low end-to-end delay. In this regard, it is worth mentioning the Common Media Application Format (CMAF), which was proposed by Apple and Microsoft in 2016 and was standardized in 2018. One of the advantages of this format is that it allows chunked encoding: the video can be broken into smaller chunks of a set duration, which are immediately published upon encoding. This way, near-real-time delivery can take place while later chunks are still processing. An additional advantage compared to the proposed approach with super-short segments, is that the encoding overhead is less significant: although chunked encoding requires additional headers, a single IDR-frame per (longer) segment suffices.

Additionally, the publication above does not define the characteristics of the videos used to evaluate the encoding overhead in Figures 2.5and 2.6. To illustrate the considered types of content, Figure 2.15 shows a screenshot of each of the eight videos.

(a) Big Buck Bunny

(b) Earth from Space

(c) Netflix' El Fuente

(d) Sintel

(e) Tears of Steel

(f) Elephant's Dream

(g) Forza Motorsport 6 Apex

(h) World Championship Soccer

Figure 2.15: Screenshots of the considered videos.

3

Performance Characterization of Low-Latency Adaptive Streaming from Video Portals

"There is a crack, a crack in everything. That's how the light gets in."

–Leonard Cohen, 1992

*In Chapter 2, a set of eight videos was used to evaluate the encoding overhead of super-short segments and determine the impact of the proposed push-based approach on the startup time, video quality and end-to-end delay. In this chapter, however, we consider video portals with hundreds or thousands of videos, either standing on their own or accompanying news stories and articles. To stimulate user engagement with the provided content, such as browsing between videos, we propose a comprehensive framework for low-latency delivery of news-related video content. Using a large dataset of a major Belgian news provider, containing millions of text- and video-based article requests, we show that the proposed framework reduces the videos' startup time in different mobile network scenarios by more than 50%, thereby improving user interaction and browsing through available content.*s

⋆⋆⋆

J. van der Hooft, C. De Boom, S. Petrangeli, T. Wauters, and F. De Turck

Published in IEEE Access, vol. 6, p. 43039-43055, 2018

Abstract News-based websites and portals provide significant amounts of multimedia content to accompany news stories and articles. In this context, HTTP adaptive streaming is generally used to deliver video over the best-effort Internet, allowing smooth video playback and an acceptable Quality of Experience (QoE). To stimulate user engagement with the provided content, such as browsing between videos, reducing the videos' startup time has become more and more important: while the current median load time is in the order of seconds, research has shown that user waiting times must remain below two seconds to achieve an acceptable QoE. In this chapter, four complementary components are optimized and integrated into a comprehensive framework for low-latency delivery of news-related video content: *(i)* server-side encoding with short video segments, *(ii)* HTTP/2 server push at the application layer, *(iii)* server-side user profiling to identify relevant content for a given user, and *(iv)* client-side storage to hold proactively delivered content. Using a large dataset of a major Belgian news provider, containing millions of text- and video-based article requests, we show that the proposed framework reduces the videos' startup time in different mobile network scenarios by more than 50%, thereby improving user interaction and skimming available content.

3.1 Introduction

In recent years, news providers have started to produce significant amounts of multimedia content to accompany news stories and articles. News providers such as the New York Times[1] and the Washington Post[2] now provide a large number of video-based news articles, containing individual topics or full news broadcasts. To encourage consumers to use the provided services, facile user interaction while browsing new content and skimming videos is of the utmost importance. In this context, reducing the videos' startup time has become more and more important: while videos generally take in the order of seconds to load, research has shown that user waiting times must remain below two seconds to achieve acceptable Quality of Experience (QoE) [1].

[1] https://www.nytimes.com/
[2] https://www.washingtonpost.com/

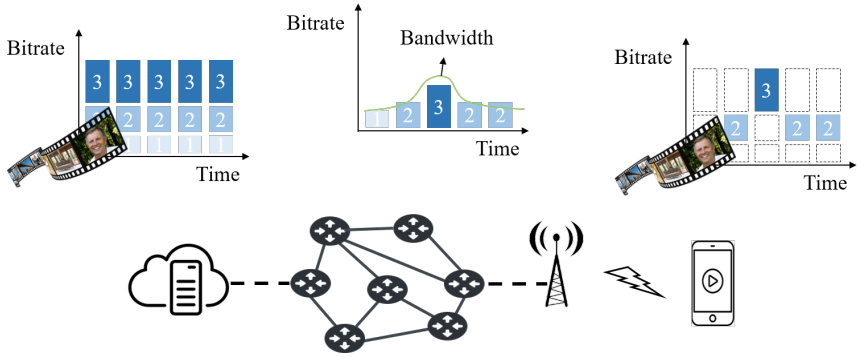


Figure 3.1: The concept of HAS. At the server-side, the video is temporally segmented and encoded at different quality levels. The client requests the video segments at the most appropriate quality level, and plays them out in linear order.

Nowadays, news content providers generally use HTTP adaptive streaming (HAS) to deliver video content over the best-effort Internet. In HAS, video is encoded at different quality levels and temporally divided into multiple segments with a typical length of 2 to 30 seconds [2]. As illustrated in Figure 3.1, an HAS client requests these video segments at the most appropriate quality level, based on e.g., the available bandwidth and the amount of buffered content. To this end, a client-based heuristic is used which attempts to optimize the QoE perceived by the user, which depends among others on the average video quality, the frequency of quality changes and the occurrence of playout freezes [3]. The client stores the incoming segments in a buffer, before decoding the sequence in linear order and playing out the video on the user's device.

This approach generally enables smooth video playout, and therefore results in a higher QoE than traditional video streaming techniques. Because of this, major players such as Microsoft, Apple and Adobe adopted the adaptive streaming paradigm and proposed their own rate adaptation heuristics. As most HAS solutions use the same architecture, the Motion Picture Expert Group (MPEG) proposed Dynamic Adaptive Streaming over HTTP (DASH), a standard which defines the interfaces and protocol data for HAS [4].

HAS is well-suited for video on demand (VoD) scenarios, and is therefore put to good use by content providers such as Netflix[3] and YouTube[4]. The startup time in these types of scenarios is however in the order of sec-

[3]https://netflix.com/
[4]https://youtube.com/

onds, with variations depending on the type of network connection. One of the reasons for this is that a significant number of resources need to be delivered before the video can start to play: the web page, the video player, the video's media presentation description (MPD) file, the video's (optional) initialization segment and the different video segments. Especially in mobile networks, where the available bandwidth is limited and the network latency is relatively high, this will have a significant impact on the video's startup time. A second reason is found in the segment duration, which is typically in the order of one to ten seconds: longer segments simply take longer to deliver, and thus result in higher video startup times.

Research has shown that reducing the startup delay in HAS is relevant, although it cannot occur at the cost of playout freezes or a reduced video quality: as users are used to some delay before the start of the playback, they usually tolerate it if they intend to watch the video [5]. However, when browsing through videos, i.e., when users start a larger number of videos but only watch parts of it, initial delays should be low for optimal acceptance [6]. To address this specific use case, a framework is presented for low-latency delivery of news-related HAS content, in a VoD scenario. This framework integrates four complementary optimizations in the content delivery chain:

1) Server-side encoding, to provide shorter video segments during the video's startup phase;

2) Changing the application layer protocol, using HTTP/2's server push to deliver resources back-to-back;

3a) Server-side user profiling to identify relevant content for each user;

3b) Client-side storage to hold proactively delivered content.

Each of these optimizations can be used separately, although they are very complementary. HTTP/2 server push, for example, can be used to deliver short video segments back-to-back, eliminating the need for individual requests for each of the segments. Therefore, content delivery and buffer rampup can happen more quickly, thus reducing the video's startup time. Prefetching news content can be done based on article recency (i.e., prefetch the n newest articles only), but also based on user profiling (i.e., prefetch based on determined user preferences), measured on an entirely different timescale. Having the right content available allows to start the video locally, thus eliminating the time needed to deliver the content over the best-effort network.

Preliminary evaluations showed that the proposed framework is able to significantly reduce the video startup time, albeit at the cost of limited

network overhead and additional complexity at the server- and client-side [7]. In this chapter, we elaborate on each of the proposed optimizations in detail, and present a large-scale and in-depth evaluation (i.e., thousands of users, videos and video streaming sessions) on a dataset of deredactie.be[5], an important Belgian news provider.

The remainder of this chapter is structured as follows. In Section 3.2, related work on low-latency video content delivery is discussed. The proposed framework is presented in Section 3.3, elaborating on the advantages of each of the optimizations. The experimental setup and results are presented in Section 3.4, before coming to final conclusions in Section 3.5.

3.2 Related Work

A large number of techniques have been proposed in literature to improve the QoE of video streaming services. These techniques can be divided in multiple ways (e.g., in client-based, server-based and network-based solutions [2]). While a lot of research has recently be done on client-side rate adaptation (e.g., BOLA and Pensieve [8, 9]), related work below mainly focuses on relevant research on low-latency end-to-end delivery on the one hand, and client-side prefetching on the other.

3.2.1 Low-Latency End-to-End Delivery

Since a certain amount of data must be transferred before decoding and playback can begin, startup delay is always present in HAS [2]. The minimal achievable initial delay strongly depends on the available transmission data rate and the encoder settings. Depending on the use case, the client can start playout as soon as content is available, or wait until a minimum amount of content is present. The advantage of the latter is that the buffer filling is higher when playout starts, therefore reducing the risk of buffer starvation. However, since most players start the stream at the lowest quality level, rebuffering events in the early stages of the video stream are less likely to occur. The dash.js reference player uses a stalling threshold of 0.5 seconds by default, which is generally low enough to start playout as soon as the first segment arrives.

Quite often, the initial delay and rebuffering time are trade-off factors: reducing the startup delay at the expense of content buffering, may result in playback freezes. Hoßfeld et al. showed that in a VoD scenario for YouTube, most users tolerate an initial delay in the order of seconds, if

[5]http://deredactie.be/

they inted to watch the video [5]. When browsing through videos, however, initial delays should be low for optimal acceptance [6]. Especially in the case of volatile and user-generated content, the startup delay should be low in order to maximize user engagement and acceptance. Although the focus in our evaluations is primarily on the observed video startup time, final results will also be reported in terms of buffer starvation.

A straightforward approach to reduce the startup time in HAS, is limiting the amount of data needed to start video playout. Therefore, the adopted encoding scheme can play an important role in the QoE of video streaming services. One possibility is to adopt the principle of scalable video coding (SVC) in HAS. This reduces the encoding and storage overhead, since each quality representation is constructed as an enhancement of the lowest quality level [10]. Because an SVC-based client has an increased number of decision points, it can cope better with highly variable bandwidth. Although SVC reduces the footprint for storage, caching and transport compared to a complete simulcast H.264/AVC system, it does introduce an encoding overhead of about 10% per layer [11]. Furthermore, since the client initially has no knowledge of the available bandwidth, most players start playout at the lowest quality level; in this case, the adoption of SVC does not impact the startup time. More recently, a number of standalone HAS players have adopted H.265/HEVC, a video compression standard was developed to provide twice the compression efficiency of the previous standard, H.264/AVC [12]. In HEVC, coding units of up to 64×64 pixels are used instead of 16×16, and more intra-picture directions, finer fractional motion vectors and larger transform blocks are used to achieve this improvement in compression performance. Although its application is increasing, most browsers offer no support for HEVC at the time of writing [13]. It is worth noting that a scalable version of the standard, SHVC, exists as well.

In live streaming scenarios, advanced client-side rate adaptation heuristics can be used in order to achieve an acceptable QoE when the buffer size is small. Recently, Shuai et al. proposed a heuristic which allows the client to stream video with stable buffer filling [14]. In the model for the QoE, rebuffering events – including the one at startup – are penalized by applying a suitable weight factor. In this way, high startup times and stalling events are actively avoided. In the evaluation of the proposed approach, results are however only shown in terms of the estimated QoE; therefore, it is unclear how the video startup time is affected. Miller et al. propose LOLYPOP, a rate adaptation heuristic for low-latency prediction-based adaptation designed to operate with a transport latency of a few seconds [15]. In their evaluation, a video streaming scenario is considered

in which the total delay (i.e., segment duration plus upper bound on transport latency) equals merely 5 seconds. To achieve such a low value, LOLYPOP leverages TCP throughput predictions on multiple time scales, from 1 to 10 seconds, along with estimations of the relative prediction error distributions. Using this approach, the authors are able to improve the mean video quality by a factor of 3 compared to FESTIVE, a well-known rate adaptation heuristic to improve fairness and stability in HAS [16]. Results for the video startup time are however not considered in this chapter.

Although real-time streaming protocols are not applicable in the targeted use case, we can use different protocols at the application layer. In this context, the new HTTP/2 standard was published as an IETF RFC in February 2015, mainly focusing on the reduction of latency in web delivery. Since then, research has shown the application of HTTP/2 can either reduce or increase the load time [17–19]. In the context of video content delivery, however, significant improvements can be achieved. Wei et al. first explore how HTTP/2's features can be used to improve HAS [20]. By reducing the segment duration from 5 to 1 second, they manage to reduce the camera-to-display delay with about 10 seconds. An increased number of GET requests is avoided by pushing k segments after each request, using HTTP/2's server push. Cherif et al. propose DASH fast start, in which HTTP/2's server push is used to reduce the startup delay in a DASH streaming session [21]. The authors also propose a new approach for video adaptation, in which WebSocket over HTTP/2 is used to estimate the available bandwidth. In previous work, we proposed a full-push feature, in which segments are pushed from server to client until the client specifies otherwise, or the connection is terminated. In contrast to these works, we propose to combine HTTP/2's server push with a hybrid segment duration scheme, resulting in smoother buffering and faster startup.

Optimizations can also be performed on the transport layer. In recent work, Gatimu et al. showed that the Flexible Dual TCP-UDP Streaming Protocol (FDSP), which combines the reliability of TCP with the low-latency characteristics of UDP, can be used to reduce the video startup time [22]. In the considered setup, FDSP delivers the more critical parts of the video data via TCP and the rest via UDP. The authors show that this approach results in significantly less rebuffering than TCP-based streaming and a lower packet loss rate than UDP-based streaming. Although results are promising, the evaluated approach does not map to traditional HAS, in which segments are retrieved back-to-back and have to be fully downloaded before playout can start.

When it comes to encoding, new real-time technologies such as Web Real-Time Communication (WebRTC) have recently been introduced [23].

Where HAS is generally used in VoD scenarios or live streaming scenarios where the delay can be in the order of tens of seconds, such as sports events, these technologies focus on collaborative real-time video streaming communication where the delay should be in the order of a few hundreds of milliseconds. Furthermore, they have been developed with a peer-to-peer architecture in mind, where a small group of clients can directly communicate with each other. Since each sender needs to encode a separate stream for each of the receivers, this approach suffers from scalability issues when many participants are present at the same time. Although useful in real-time communication, these technologies do not envision traditional VoD scenarios and are thus unsuitable to provide the required low-latency aspects in HAS.

Recently, further improvements to HAS have been made, such as Server and Network Assisted DASH (SAND) [24]. While the focus of this chapter is on over-the-top video delivery only (i.e., delivery without control over the network), SAND aims to further improve performance by enabling in-network decisions. In the suggested approach, a bi-directional messaging plane is used between the clients and other so-called DASH-Aware Network Elements (DANEs), in order to carry both operational and assistance information. This allows to trigger control mechanisms such as flow prioritization, bandwidth reservation and video quality adaptation based on the network's and client's current state. A large number of studies has been conducted, showing that the SAND principle can significantly improve the QoE in HAS [25, 26]. This concept however moves away from over-the-top solutions, and is therefore not considered in this chapter.

3.2.2 Prefetching of Multimedia Content

To improve the QoE in video streaming, content can be brought closer to the end user. Streaming providers have massively adopted the use of Content Delivery Networks (CDN), reducing the load on the origin server and serve the video with lower latency and increased bandwidth. General caching strategies can be applied, taking into account characteristics such as content popularity, user preferences, the client's location, etc. In this regard, a number of studies have shown significant improvements. As an example, Krishnappa et al. exploit particular user behavior to improve the caching efficiency of YouTube videos [27]. By rearranging the related video list to give preference to cached videos, the cache hit rate is improved by a factor of 5. Caching strategies can also be improved when future user requests are known in advance. For example, binge-watching has become a well-known phenomenon for video streaming services, meaning that users

tend to watch multiple episodes of the same TV show consecutively. As shown by Claeys et al., this information can be used to estimate future video segment requests and improve caching efficiency [28]. Apart from being cached in a CDN, content can also be prefetched by the client. Krishnamoorthi et al., for instance, propose a scheme in which three policy classes are applied to preload content for HAS [29]. The authors however assume that the content provider has already established a list of relevant content which should be prefetched. In this work, we also focus on how to select this content at the server-side.

For this purpose, we will use techniques that have been proposed in the context of recommender systems, where users and their consumed items are projected in a low-dimensional vector space [30]. Users with similar consumer characteristics will typically have vector representations that lie close to each other in this space, while dissimilar user vectors are located far apart. Matrix factorization, for instance, is an often used technique to arrive at a vector space with such characteristics. It is typically applied to a user-item rating matrix, while imposing that the scalar product between a user and item vector is a good rating predictor. The problem with this approach, however, is that the user vectors are considered static, which is not always ideal in dynamic scenarios in which many items are consumed one after the other, such as songs, videos and news content. It has recently been shown in literature, and real-life scenarios at Netflix and Spotify, that it is often beneficial to explicitly consider the time aspect by modeling users in a dynamic fashion [31–33]. In these systems, a user is typically represented by aggregating item vectors across time. For example, in the work by Hidasi et al. a recurrent neural network is used to process items one after the other [34]; the output of this neural network is a set of recommended items after processing a new item. The downside of this approach is that users are not modeled explicitly in the same space as the items, and therefore does not allow for direct user-item comparisons. In order to achieve this, we can simply sum or average the consumed item vectors, through which we remain in the same item space. We will further elaborate upon this in the next section.

3.3 Proposed Framework

The proposed framework integrates four complementary optimizations in the content delivery chain, as illustrated in Figure 3.2. First, we consider the aspect of video encoding, using a shorter video segment duration to reduce the playout delay. This optimization requires sufficient resources at the server-side, in order to encode provided content both for multiple

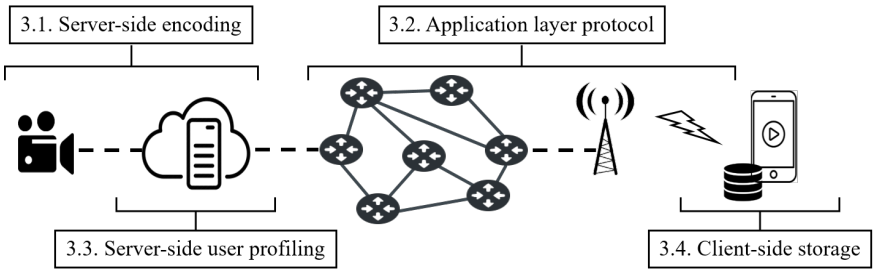
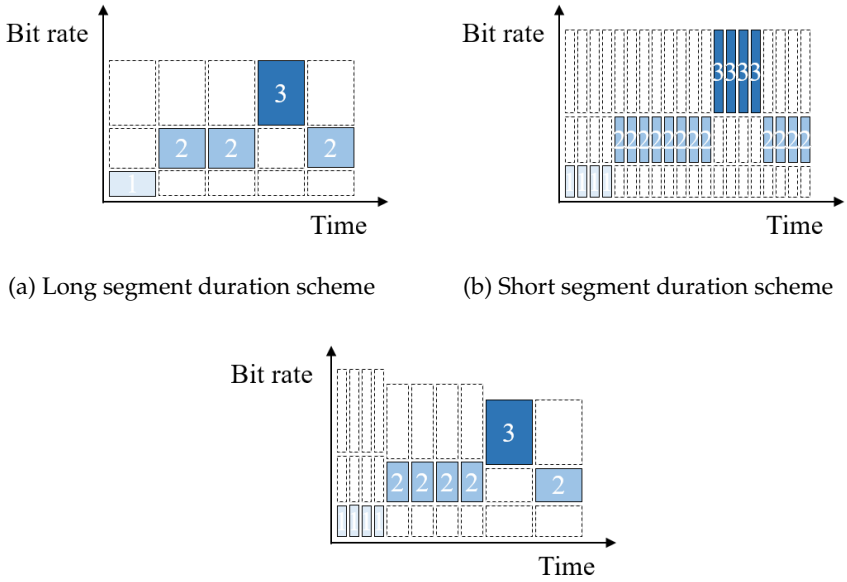


Figure 3.2: The proposed HAS delivery framework for media-rich content from news providers [7].

quality representations and for different segment durations. Second, we focus on the applied application layer protocol, discussing the possibilities of HTTP/2's server push feature. This requires an HTTP/2-enabled server, equipped with a custom request handle to push required resources from server to client. Since most browsers nowadays have full support for HTTP/2, no changes to the client are required. Third, we consider user profiling as a way to predict user interest and interaction. To this end, the server needs to monitor all incoming requests and keep track of several content- and user-based characteristics. Fourth, client-side storage is considered to store content which is proactively delivered to the user, once it is deemed of interest by the profiling component. This requires additional complexity at the client-side, and is prone to bandwidth overhead when the wrong content is prefetched. Below, we elaborate on each of these optimizations in detail.

3.3.1 Server-Side Encoding Using Hybrid Segment Duration

The first part of the proposed framework consists of server-side encoding, and more specifically on the segment duration of the provided content. As found in previous work, reducing the duration of video segments comes with a number of advantages [35]. Most importantly, the short segments require a lower download time, resulting in a reduced delivery time and thus in faster startup. However, since every segment has to start with an Instantaneous Decoder Refresh (IDR) frame, a higher bit rate is required to achieve the same visual quality compared to segments of higher length. This encoding overhead was analyzed for seven videos at multiple frames per seconds (FPS) in previous work, showing that a segment duration of 1 second results in a bit rate overhead between 12.1 and 22.4% compared to a segment duration of 8 seconds, and in an overhead between 28.7 and



(a) Long segment duration scheme

(b) Short segment duration scheme

(c) Hybrid segment duration scheme

Figure 3.3: Possible segment duration schemes in HAS. While traditional schemes use a fixed segment duration, the hybrid scheme changes the segment duration over time, smoothly ramping up the video player's buffer.

49.9% for a segment duration of 250 ms [35]. Moreover, since a unique request is required to retrieve each single video segment, solutions with low segment duration are susceptible to high round-trip times (RTT). This problem mainly arises in mobile networks, where the RTT is in the order of 100 ms, depending on the network carrier and the type of connection.

While traditional streaming solutions use a fixed segment duration in the order of 2 to 30 seconds, we propose to use different segment durations for the startup and steady-state phase of the video streaming session. This allows us to both reduce the video startup time by using short video segments in the startup phase, and overcome the aforementioned issues by switching to longer segment durations once the video is steadily playing. Two approaches are possible: *(i)* initially start at the lowest segment duration d_1, switching to the highest segment duration d_n once a significant amount of segments has been downloaded, and *(ii)* initially start at the lowest segment duration d_1, switching to d_2, d_3, ... until a segment duration of d_n is reached. The advantage of the latter is that the buffer is ramped up smoothly, preventing possible freezes when switching from the lowest to the highest segment duration when the buffer level is relatively low. A

disadvantage to this approach is that multiple versions of the content need to be available, each containing a different segment duration. However, since the segment duration is changed only during the startup phase, only the first part of the content needs to be encoded multiple times. Furthermore, since most players generally start playout at the lowest video bit rate, it is sufficient to provide multiple segment durations for the lowest quality representation only.

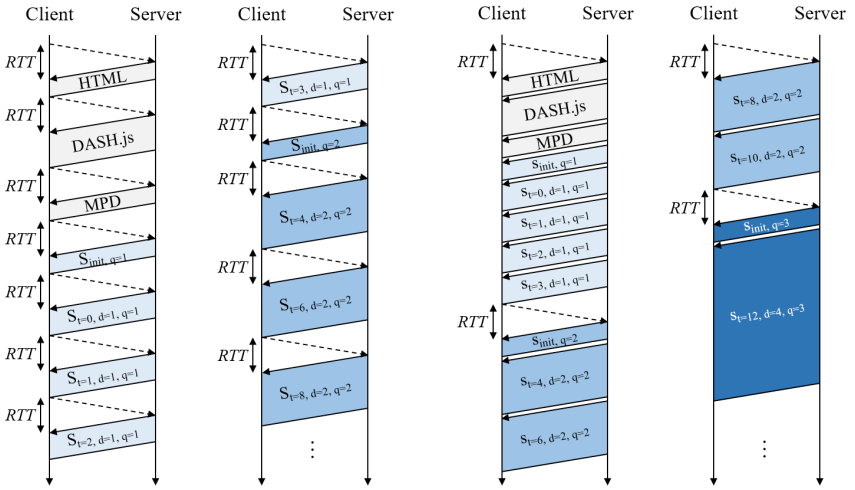
There are multiple ways to apply the proposed segment duration scheme. One possible approach is to generate the MPD, which contains relevant information on the available content (e.g., the video's duration and available quality representations), in such way that several parts of the video are distinguished [36]. It is then possible to define different segment durations for different quality representations and time intervals, making the approach completely DASH-compliant. A second approach is to limit the segment duration in the video player itself, for instance by tracking the video playout progress and available quality representations.

It is worth noting that the proposed scheme can not only be applied at startup, but each time the buffer filling drops below a certain threshold: the player can recover more quickly, and thus reduce the total time of stalling. In this case, however, the full video needs to be available in different segment durations. This scheme is not adopted in this chapter.

3.3.2 Application Layer Optimizations Using HTTP/2's Server Push

At the start of an HAS video streaming session, a large number of files need to be downloaded. In a stand-alone client, a request is first sent for the video's MPD file. Based on the contents of this file, the client proceeds to download the initialization segment (if any) and from then on, requests video segments one by one. In a web-based context, the HTML page and its required resources need to be fetched as well, including the HAS player, JavaScript sources, CSS files, images, etc. All these resources are requested over HTTP, which among others, allows to traverse firewall and NAT devices, and reuse the existing delivery infrastructure. Retrieving these resources one by one takes time, given RTT is lost for every request. Especially in mobile networks, where the latency is in the order of tens to hundreds of milliseconds, this can have a significant impact on the startup time.

When it comes to browser page loading, the total time can be reduced by using up to six parallel HTTP/1.1 connections. This allows to retrieve resources faster, since idle RTTs can be omitted. It is however infeasible to



(a) HAS over HTTP/1.1 (b) HAS over HTTP/2

Figure 3.4: Sequence diagrams of the proposed hybrid segment duration schemes. Using HTTP/1.1, each video segment requires a separate GET request. Using HTTP/2, one request allows to deliver multiple resources, thereby reducing the video's buffering and startup time.

use this approach in the case of HAS: during the startup phase, the client's browser does not know in advance which resources will be required next (e.g., the download of the dash.js player cannot start before the HTML source code is parsed), and once the video session has started, segments are retrieved one by one to guarantee in-order delivery, and thus the (timely) arrival of the most crucial segments first. An alternative way to deliver required resources more quickly, would be to forward these without the client requesting them. To this end, HTTP/2's server push can be used.

In Figure 3.4a, the example scenario from Figure 3.3c is illustrated for HTTP/1.1. Here, the HTML source code is downloaded first, and is parsed by the browser. This code includes the dash.js script, which is in its turn requested. Once the player is present, it initiates the download of the video session's MPD and parses the document. It finds out that an initialization segment is required for each of the quality representations, and issues a request for the one corresponding to the lowest quality. From then on, segments are requested one by one, evaluating the available bandwidth once each of the segments has been downloaded.

In 2015, the HTTP/2 standard was published as an IETF RFC. Its main purpose is to reduce the latency in web delivery, using request/response multiplexing, stream prioritization and server push. The latter can be used

to push video segments from server to client, without the client sending a GET request for the required resources. Pushing the content back-to-back allows to eliminate idle RTT cycles, reducing buffering time and improving bandwidth utilization.

In previous work, a stand-alone client was considered to evaluate the use of HTTP/2's server push feature [35]. In this context, we propose to push resources related to the video streaming session only: once the manifest is requested, video segments are continuously pushed to the client, until an explicit stop request is sent or the connection with the client is terminated. In this way, idle RTTs during content download are eliminated, resulting in reduced delivery times and thus faster video startup.

In this work, we consider the browser-based dash.js reference player, with the hybrid segment duration scheme presented above. As illustrated in Figure 3.4b, HTTP/2 server push is now used to deliver the HTML source code of a sample web page, the dash.js reference player embedded within this page, the MPD, the initialization segment and the first k video segments, corresponding to the first x seconds of the the video stream. From then on, one GET request can be used to retrieve each x seconds of video. Depending on the selected segment duration, the server can push a different amount of segments, while the client can specify the desired quality representation based on its rate adaptation heuristic. Similarly, this approach results in the removal of idle RTTs, reducing the video's startup time and increasing the total throughput.

Although not applicable in the evaluation setup in Section 3.4, it is worth noting that additional sources, such as images, scripts and CSS, can be retrieved using HTTP/1.1 (potentially with multiple connections) or by HTTP/2 (potentially with server push), or from the browser's cache if present.

3.3.3 Server-Side User and Content Profiling

A third optimization consists of server-side user and content profiling. Its purpose is to build a profile for all platform users, determining their preferences towards certain news content. Generally speaking, the purpose of the profiling component is to select a subset of relevant, recent video content for any given user. This content can then be prefetched by the client, effectively reducing the video startup delay at the time of request. In the proposed framework, we thus want a means to determine such a subset. To this end, for each of the users, we compare the performance of each of the following recommendation strategies over a recent period (i.e., the last n number of requests):

- **Likely to consume most popular content** - Certain video articles have a higher probability of being requested than others. Research has shown that this depends, among others, on the type of subject, the location of the event and the objectivity of the title. Popular articles are generally highlighted on (top of) the home page, and are more often shared on social networks such as Facebook and Twitter. If we want to consider a relevant subset of videos to prefetch, it thus would make sense to consider the most popular content only. When this profile is assigned, the number of requests issued within a certain time interval is considered to rank the available content. From this list, the n most popular articles are considered of interest to the user;

- **Likely to consume most recent content** - News content is typically short-lived, i.e., its relevance decreases quickly over time [37]. News articles and videos are only featured on (top of) the home page for a limited amount of time only, and are quickly replaced by new data and news topics. Furthermore, a significant amount of users prefer to stay up-to-date by visiting news web sites multiple times a day, making news older than a couple of hours less relevant. Therefore, when this profile is assigned, all content is ranked according to the time of publication. From this list, the n most recent articles are considered of interest to the user;

- **Likely to consume personalized content** - Some users are interested in particular news subjects and articles, and therefore consume specific videos only. For this type of users, we will, as traditionally done in recommender systems, represent each user and video article by a low-dimensional vector. To this end, we assume that every video has associated textual metadata, and apply a natural language processing model to represent each of the articles. This metadata can include, but is not limited to, the author(s), title, summary and text-based content of the news article. Based on previous work, word2vec is selected to accomplish this [7, 38]. Word2vec learns low-dimensional word vectors, also called word embeddings, by training a two-layer neural networks to reconstruct linguisitic contexts of words in a large text corpus. For each word in this corpus, a word embedding is learned that is located in a low-dimensional space such that words with a common context are positioned close to one another [38]. Since word2vec operates on word level, we will represent each article by the sum of the word vectors it contains. A user is represented by a vector as well, which is initially an all-zeros vector. Each time a new

article is requested by a user, the corresponding vector is updated by summation of the user and article vector in an online fashion. This approach allows us to create a unique vector for each user, building a user profile over time. The relevance of an article **a** to a user **u** can then be determined using the cosine similarity:

$$\cos(\mathbf{u}, \mathbf{a}) = \frac{\mathbf{u} \cdot \mathbf{a}}{\|\mathbf{u}\|_2 \|\mathbf{a}\|_2}. \tag{3.1}$$

The higher this similarity, the higher the user's preference towards an article is assumed. When this profile is assigned, recent content is considered and ranked according to the cosine similarity between the user and article vectors. From this list, the n most similar articles are considered of interest to the user. Note that this approach is prone to the cold-start problem, since the user vector is initially set to zero and is slowly built up for each user.

We propose to assign each user one of the aforementioned profiles, but do it in such way that performance does not suffer from the cold-start problem and that a user's preferences can change over time. To this end, each user is initially assigned the popularity profile, where the most requested articles are considered. Once a sufficient amount of requests has been issued by the user, we allow the assigned profile to be changed throughout time, using online evaluations of the user behavior and preferences towards certain content. To determine the most appropriate category for a given user, and therefore the best approach to rank the available content, a sliding window is used over the user's past n requests. Within this window, the position of each of the user's requests is evaluated in the sorted list generated by each of the three profiles. The user is then be mapped to the profile which results in the lowest average position of the requests.

Using the average position for the last n requests as a metric, differences between the three profiles can be small. This can result in a large number of switches in the assigned profile, which should preferably be avoided. For this reason, hysteresis is applied to only change the assigned profile when the metric of a certain profile outperforms the currently assigned profile by at least a fraction α.

The result of the proposed approach is a shortlist containing the most relevant video articles for each of the users. As explained below, the content of this shortlist can be used to proactively deliver the content to the user, anticipating future article requests.

3.3.4 Client-Side Storage

A final component of the proposed framework consists of client-side storage, which is used to enable proactive delivery of relevant video content. If the right content is sent, using such approach allows to significantly reduce the video session's startup time. Depending on the use case scenario, multiple options for content delivery and client-side storage are possible. In a stand-alone application, a dedicated cache on the local device can be used. Based on server recommendations, the application can retrieve content in the background. Measure needs to be taken as not to use prefetching too aggressively, as it introduces a risk of increasing battery drain and bandwidth use. In web-based applications, control over client-side storage is less evident. Recent versions of browsers such as Google Chrome allow to prefetch web pages which are referred to in the current page. Pages are prerendered in a hidden tab, and moved to the foreground upon request. This can be extended as to provide support for HAS, retrieving the first x seconds of embedded video content. Most browsers now also support HTTP/2, storing pushed resources in the browser's cache. This allows the server to push additional resources upon a client's incoming request, yet care needs to be taken that prefetching does not interfere with the transfer of more urgent resources (e.g., the current video being played out). For this reason, the best approach is to only deliver additional resources once the requested article page has been retrieved completely.

Regardless of how the content is delivered, it is important that the right content is sent. Indeed, proactive delivery of non-relevant content results in network overhead, since bandwidth is wasted on content which may never be consumed. Therefore, we can apply the profiling detailed in Section 3.3.3 to determine a shortlist of articles, and consider the top k articles for prefetching. In our evaluations in Section 3.4, we investigate the impact of this parameter (which corresponds to the *cache size*) on the startup time (no requests need to be sent to the server in case of a cache hit) and the network overhead. Note that only the first d_n seconds of video, corresponding to the largest segment duration, are stored: as soon as video playout has started, the remaining content is fetched from the server as in a traditional streaming scenario.

3.4 Evaluation

In this section, we evaluate the impact of the proposed framework on the video startup time. We first discuss the considered use case of a major Belgian news provider, and present the experimental setup used to evaluate

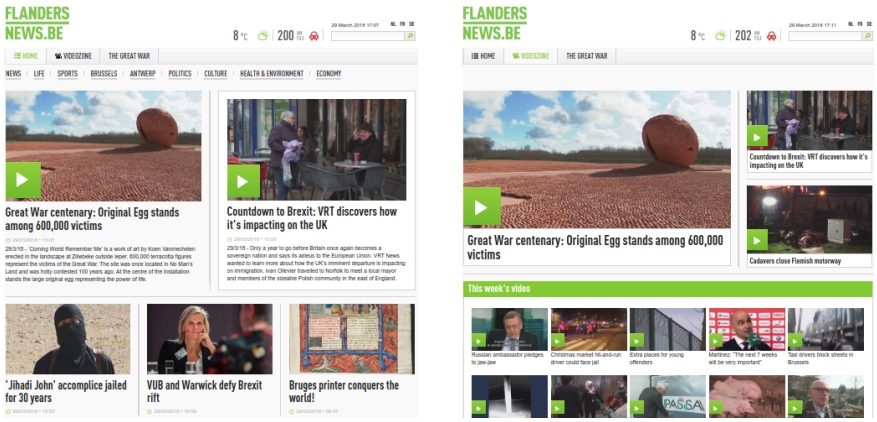


Figure 3.5: Screenshots of deredactie.be, homepage (left) and video zone (right).

results. We then present the most important results, discussing the advantages and shortcomings of each of the proposed optimizations.

3.4.1 Use Case: deredactie.be

Deredactie.be is one of the major news websites in Belgium, hosted by the Flemish Radio and Television Broadcasting Organization (VRT). In recent years, its focus has shifted largely from simple text-based articles towards multimedia-rich news reports. Because of this, the website is an excellent use case for the proposed delivery framework. On the home page, users are presented an overview of recently published content, containing a relevant poster, a title and a short introduction on each of the news topics (Figure 3.5). In the "video zone", a separate page, all videos published in the last week are presented in order of publication (i.e., most recent first). The website also contains references to other (news) sources, such as Sporza (sports) and Canvas (culture).

In collaboration with VRT, Van Canneyt et al. were able to collect a data set containing approximately 300 million website requests, issued between April 2015 and January 2016 [37]. For every request to the website, among others the requested URL, the referrer URL, the server's and client's local time, and the client's hashed IP and cookie ID were logged.

From the given dataset, all users and article requests were extracted. The HTML and XML sources of the requested articles were retrieved, and relevant information such as the title, summary and content was extracted. All embedded video was retrieved as well, resulting in a total of 19 437 videos. All data was used by the proposed framework, as detailed below.

Figure 3.6: Experimental setup. Mininet is used to host a virtual network within a Docker container. The dash.js player is used in the Google Chrome browser, starting and playing different video streaming sessions from the HTTP/2-enabled Jetty server.

3.4.2 Experimental Setup

In the remainder of this section, we evaluate the proposed framework under realistic network conditions. To this end, a network setup is emulated using Mininet[6], where a client is connected to an HTTP/1.1- and HTTP/2-enabled Jetty server (Figure 3.6). As for the mobile network scenario, both 3G and 4G network scenarios are considered; while 4G coverage in Belgium is excellent, clients are still often forced onto 3G in areas with a lower population density [39]. To emulate network conditions, traffic control is used to set the network latency to 120 and 60 ms for 3G and 4G respectively, and to shape the available bandwidth of the client according to bandwidth traces provided by Riiser et al. and van der Hooft et al. (see Appendix A) [40, 41]. The client uses the Google Chrome browser in headless mode to start a video streaming session, using the reference dash.js player. The open-source code of the Jetty server is slightly modified, allowing it to push the required HTML and JavaScript resources, the MPD, the initialization segment and the first 10 seconds of a given video upon request. To allow seamless connection over HTTP/2, a Node.js proxy is provided for each client. This proxy can retrieve content from the local filesystem as well, and can therefore be used to measure startup times for prefetched video. The complete setup has been wrapped in a docker container, increasing portability and allowing parallel execution of video streaming sessions. Experiments were carried out on five physical nodes on the Virtual Wall[7], with twelve docker containers running simultaneously on a hexa-core Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz with 24 GB of RAM. Below, results are shown for different scenarios, including an additional optimization in each experiment.

[6] http://mininet.org
[7] https://doc.ilabt.imec.be/ilabt-documentation/

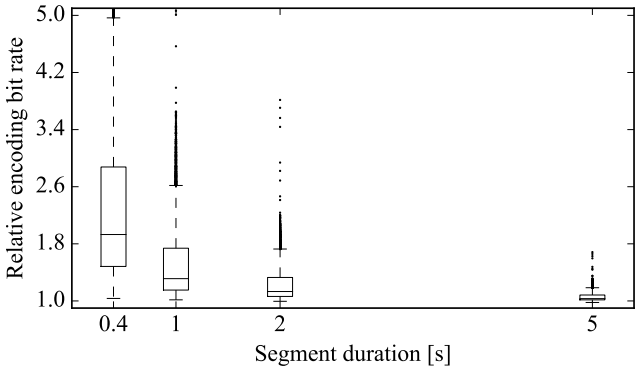


Figure 3.7: Encoding bit rate as a function of the segment duration, relative to the bit rate for a segment duration of 10 seconds. Outliers for a segment duration of 0.4 seconds go as high as 14.065, but are omitted in favor of readability.

3.4.3 Short Segment Duration

Since shorter video segments require less data to be transferred from server to client, the client should be able to start playout faster. As mentioned above, this approach introduces an encoding overhead. In previous work, we evaluated this overhead for shorter video segments on seven different videos [41]. Given the extent of the presented dataset, we decided to evaluate the encoding overhead for all 19 437 video articles published within the time of logging. By default, deredactie.be provides its video content at a frame rate of 25 FPS, a spatial resolution of 640×360 and a segment duration of ten seconds. This content was re-encoded using AVC/H.264 with the same frame rate and resolution, but with a segment duration ranging from 400 ms to 10 seconds. To allow each segment to be decoded independently, every segment starts with an IDR frame, and the Group of Pictures (GOP) length is set to values ranging from 10 to 250 respectively. To realize the same visual quality, the Constant Rate Factor (CRF) rate control in the x264 encoder is enabled, with a CRF value of 25. This results in average video bit rates of 423 and 231 kb/s for a segment duration of 400 ms and 10 seconds respectively.

Figure 3.7 shows a boxplot of the encoding overhead for different segment durations, relative to the bit rate of a segment duration of 10 seconds. The obtained average video bit rates equal 423, 300, 261, 238 and 231 kb/s for a segment duration of 0.4, 1, 2, 5 and 10 seconds respectively, or a relative overhead of 83.3, 30.4, 13.2 and 3.3% compared to a segment duration of 10 seconds. Values for the overhead range from 1.035 (low overhead)

to 14.065 (significant overhead) for a segment duration of 0.4 seconds. The former stems from videos with a lot of movement and scene switches, for which the insertion of additional IDR frames has almost no impact on the overall video bit rate (e.g., a report on the world record pillow fight), while the latter stems from videos with hardly any movement at all (e.g., still photos during a news broadcast). This shows that the encoding overhead can be significant, and should be avoided whenever possible. Therefore, applying a hybrid approach in which short segments are used at startup and long segments in steady-state, is beneficial in terms of consumed bandwidth and video quality.

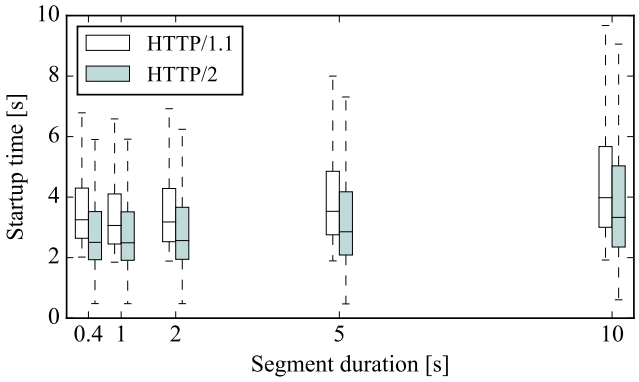
In a first video streaming experiment, we evaluate the startup time for different segment durations d_1, where the required content is requested over HTTP/1.1. The segment duration ranges from 0.4 to 10 seconds, at a frame rate of 25 FPS. Figure 3.8 shows the boxplots for the startup time of 19 437 video streaming sessions, one for each video in the dataset. Outliers, corresponding to at most 7% of the data, are omitted in favor of readability: some outliers reach values up to 38.4 seconds because the throughput traces contain different periods of low throughput, where connection is bad to non-existing.

In a 3G scenario, the observed gains are significant: the median startup time is reduced from 4.0 to 3.1 seconds for a segment duration of 10 and 1 seconds respectively. Furthermore, variability is significantly lower, reducing high startup times which generally impede the QoE the most: the 90% and 99% quantiles, for instance, are reduced from 8.4 to 5.9 seconds and from 15.7 to 11.3 seconds. Results also show that the segment duration cannot be reduced indefinitely: for a segment duration of 0.4 seconds, the median startup time increases again to 3.3 seconds, and the 90% and 99% quantiles to 6.1 and 11.4 seconds respectively.

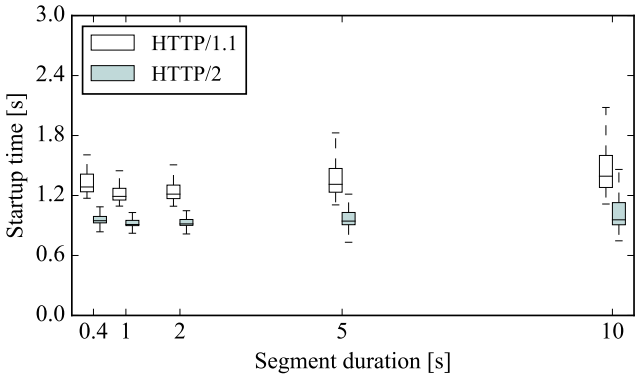
In a 4G scenario, the gains are less outspoken: the median startup time is reduced from 1.4 to 1.2 seconds for a segment duration of 10 and 1 seconds respectively, while the 90% and 99% quantiles are reduced from 1.9 to 1.5 seconds and from 4.2 to 3.4 seconds. Because the throughput is significantly higher, the importance of the video file size is simply reduced. Relatively speaking, however, the observed gains are still significant: a reduction of the median startup time of 14.5% is achieved for 4G, compared to 23.1% for 3G.

3.4.4 Short Segment Duration and Server Push

As explained in Section 3.3, the application of HTTP/2 server push allows to avoid idle RTT cycles in the buffer rampup phase. In Figure 3.8,



(a) 3G network



(b) 4G network

Figure 3.8: Startup time as a function of the segment duration. Outliers, corresponding to at most 7% of the data, are omitted in favor of readability.

the startup time of all video streaming sessions is compared between HTTP/1.1 and HTTP/2. In a 3G scenario, where the latency is equal to 120 ms, the median startup time is reduced from 4.0 to 3.3 seconds (-17.5%) and from 3.1 to 2.5 seconds (-19.3%) for a segment duration of 10 and 1 seconds respectively. In a 4G scenario, where the latency is equal to 60 ms, the median startup time is reduced from 1.4 to 1.0 seconds (-28.6%) and from 1.2 to 0.9 seconds (-25.0%) for a segment duration of 10 and 1seconds respectively. These gains are a direct consequence of back-to-back delivery of the required video resources. It is worth mentioning that the application of HTTP/2 server push has no impact on variability: the reduction of the startup time is linearly correlated to the network delay only.

3.4.5 User and Content Profiling

The application of user profiling is situated on a different timescale than the actual video streaming. For this reason, we first show results for the proposed user profiling strategies, before applying them in a video streaming scenario. To enable user profiling, the title, summary and text content from each article published within the time of logging was extracted, Dutch stopwords were eliminated and the resulting lower-case text was used as input to train a word2vec-based model with 100 dimensions. Note that all articles containing video, include at least a title and a brief summary; therefore, each article can be represented by the sum of its word vectors.

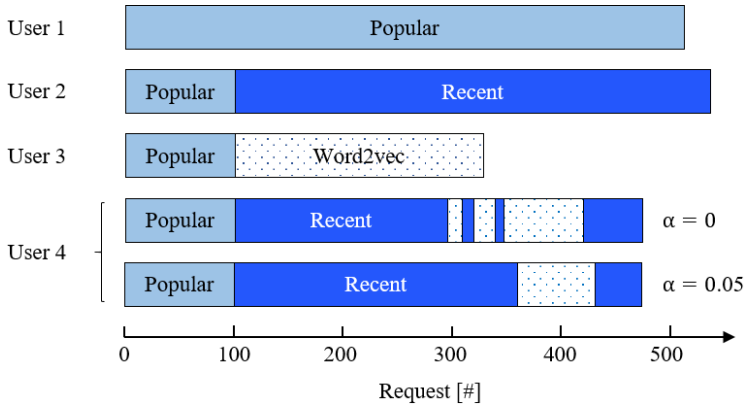
To evaluate the accuracy of the resulting models, we replayed all article requests issued by users who were active during at least half of the period of logging (i.e., five months) and who, on average, requested at least one video article per day. Eliminating page crawlers, this resulted in a pool of 5835 users, who together request a total of 1 848 319 video articles. Similar to articles, each user is represented by a 100-dimensional vector, which is updated each time an article is requested.

Before each request, a sliding window over recently issued requests is updated. Within this window, each of the three strategies (i.e., popular, recency and word2vec) is evaluated based on the rank of the articles in the sorted list, and a decision is taken to either stay on the current strategy, or switch to an alternative one if this has shown to result in a better performance. We performed an analysis of different parameter settings, including the optimal window size and hysteresis fraction thereof.

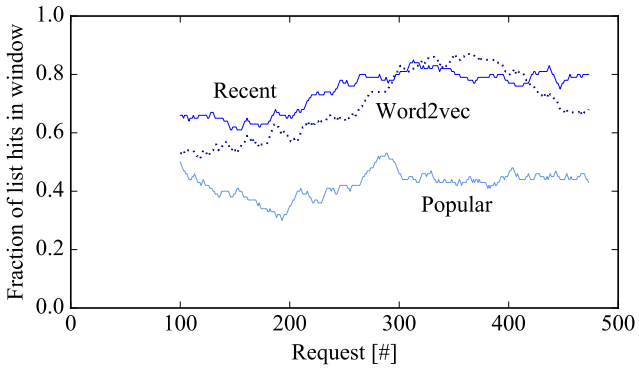
A higher window size entails more information, and thus a better decision regarding which category to assign to the user. However, using a window size which is too large, users which are likely to switch to a different category remain at the (initially assigned) popular strategy too long, resulting in lower performance. After careful consideration, a window size of 100 requests was selected.

When changing the assigned profile based on the number of list hits within the sliding window, differences in accuracy can be small. This can result in a large number of switches in the assigned recommendation strategy. To avoid this from happening, hysteresis is applied to only change the strategy when the cache hit ratio of a certain approach outperforms the currently assigned approach by at least a fraction α of the window size. As illustrated in Figures 3.9a and 3.9b for an example user, using a value of $\alpha = 0.05$ is sufficient to reduce the number of otiose strategy switches.

Figure 3.10 shows results for the three strategies and for the combined approach. On the x-axis the considered article list size (i.e., the number of



(a) Switching behavior for four example users



(b) Accuracy within the sliding window for user 4

Figure 3.9: Category preference and switching for four different users (top) and the accuracy in the sliding window for user 4 (bottom), for a list size of 16 articles and a window size of 100 requests. Because results for the recent and word2vec profile overlap, a large number of switches can occur.

articles that the recommendation algorithm is able to select) is presented, on the y-axis the relative number of requested articles which were present in this list. As an example, using a list containing the sixteen most recent articles at each point in time, 43.3% of requested articles were present in the list. Compared to a static popularity strategy, applying the proposed approach results in higher accuracy, although improvements are limited; as an example, the accuracy increases from 0.4781 to 0.4794 (0.3%) and from 0.6629 to 0.6664 (0.5%) for a list size of eight and sixteen respectively. For most users in this dataset, simply considering the most popular articles at

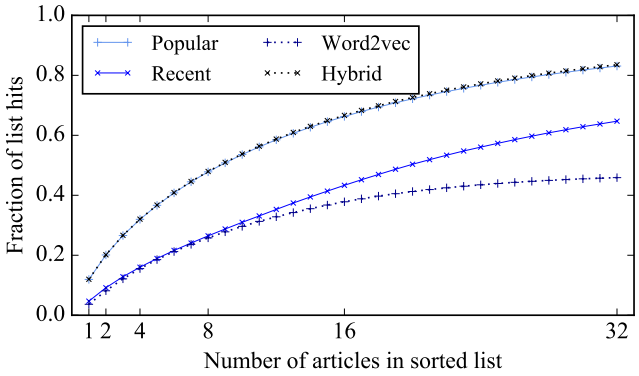


Figure 3.10: Relative number of list hits for the three different user profiles, and the proposed hybrid approach with a window size of 100 requests and $\alpha = 0.05$.

each point in time, results in the best performance. This is because most users tend to consume content which is readily available and presented on (top of) the home page. For this reason, it is more difficult to build accurate user profiles, or predict consumption behavior based on previous requests.

Figure 3.11 shows the relative number of users for which the profile is changed over time (total), and the relative number of users for which either the popular, recent or word2vec-based profile is selected most often when switching. As can be observed, at most 10% of users is ever assigned a different profile than the most popular one. The recency profile is preferred by at most 6% of users, and the word2vec-based profile by 2%. As illustrated in Figure 3.12, however, changing the recommendation strategy for these users does result in significant improvements.

As an example, for a list size of eight, the performance relative to a static profile based on popularity, is improved by 12.5% and 11.6% for users who are most often assigned the recency and word2vec-based strategy respectively. These users have an outspoken preference towards certain topics and TV programs, which, when taken into account, in some cases improve performance by a factor of 2 and more. For these users, the proposed hybrid profiling approach can thus be adopted to improve accuracy, and therefore improve the list of content considered for prefetching when client-side storage is enabled. It is worth noting that for lower list sizes, switching the assigned profile can in some cases result in lower performance: given the low number of considered articles, there is little certainty about potential list hits.

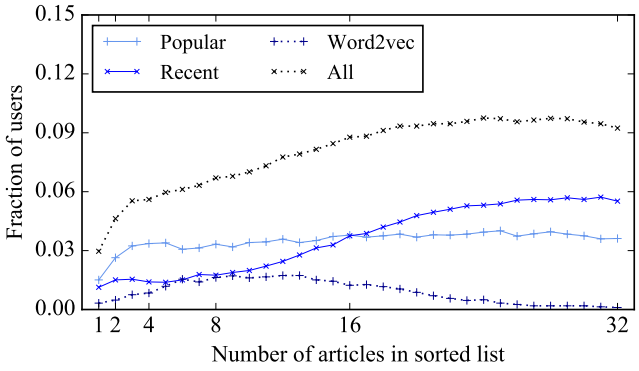


Figure 3.11: Relative number of users whose user strategy is changed over time. Users are categorized based on the dominant user strategy (either popular, recent or word2vec-based).

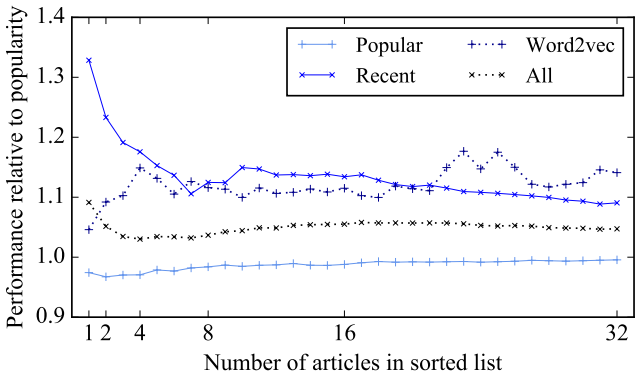


Figure 3.12: Performance for users whose recommendation strategy is changed over time, relative to a static popularity strategy. Values lower than 1 can occur, since switching is not always beneficial (although one strategy outperforms the others in the sliding window, it is not guaranteed that this will also be the case in the requests to come).

3.4.6 Short Segment Duration, Server Push and Proactive Prefetching

In a final set of experiments, we assume that client-side prefetching and storage is possible as well. To this end, the proposed recommendation scheme is adopted to rank the available content for the 5835 users in the user pool. It is worth noting that any recommendation scheme could be used here: the proposed optimizations are complementary, and can thus

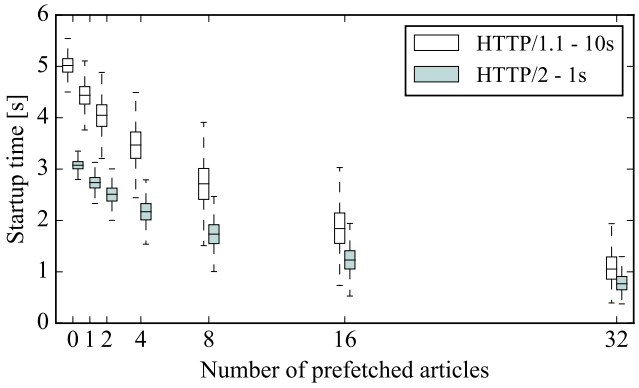
be omitted or replaced by valid alternatives. Then, the full request log is replayed for each user.

In practice, the client would prefetch relevant content during off-times. In the case of a stand-alone application on a mobile device, prefetching can be done in the background. In the web browser, this can be done based on server suggestions, either when the current page has finished loading or when the playing video has been retrieved. In our experimental setup, it is however practically infeasible to recreate the full user session: only requested article URLs are available, yet not the information on the played out quality of video, the length of the session, the available throughput, etc. For this reason, prefetching is not directly implemented: we simply assume that the client has, at each point in time, stored the first $d_n = 10$ seconds of content for each of the top n video-based articles. When the client issues a request, the content is retrieved either from local storage (i.e., through the proxy) if the content is available in this list, or from the server if not.

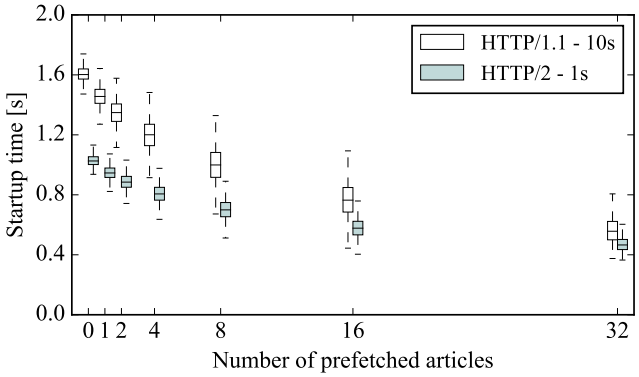
We assume storage capacity is limited, so that previously stored content is removed when the list of articles is updated (e.g., when new articles have been published in the recency strategy, when a certain article generated more requests in the last hour in the popularity strategy, or when the user switches from one strategy to another, possibly resulting in a significantly different set of articles). For instance, the client might be able to store the top 8 video-based articles only, but not more.

The 5835 users have sent a total of 1.8 million requests to video-based articles during the time of logging. Because of the time complexity, it is infeasible to replay all these requests for each evaluated parameter configuration. For this reason, the startup times for each of the videos in Figure 3.8 are used to represent the average startup time of each user. Of course, startup times are strongly dependent on the available bandwidth at the time of buffering, and can thus differ severely between sessions. Since the trace is started at a random point in time, however, the resulting values should be a close approximation of the expected startup time under realistic network conditions.

Figure 3.13 shows the average startup time as a function of the number of articles the client is able to prefetch. Naturally, the more articles the client can prefetch, the lower the average video startup time will be. As an example, the median startup time in a 3G network is reduced from 5.0 to 2.7 seconds (-45.9%) when the eight most relevant videos for each user are prefetched with a segment duration of 10 seconds over HTTP/1.1. When this number is increased to 32, the median startup time can be reduced even further to 1.1 seconds (-78.0%). When a lower segment duration of



(a) 3G network

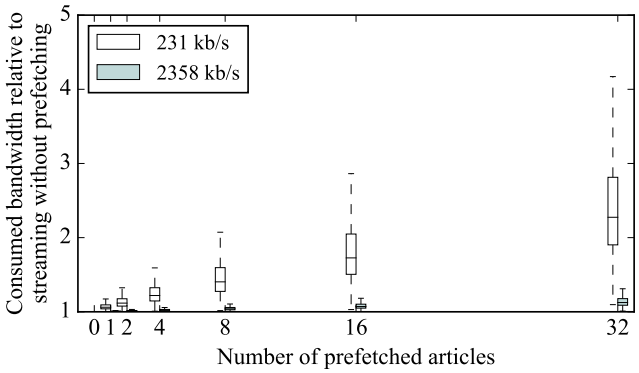


(b) 4G network

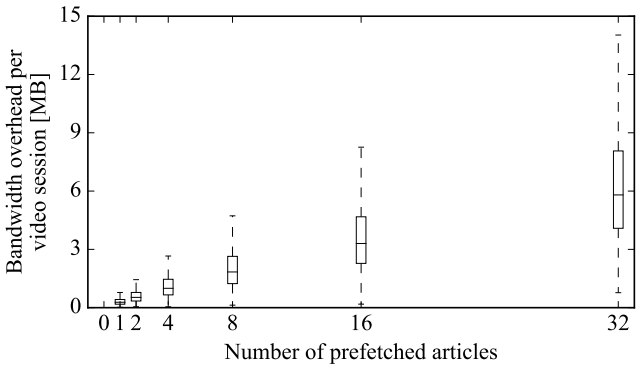
Figure 3.13: Startup time as a function of the number of prefetched articles, using a segment duration of 10 seconds over HTTP/1.1 or a segment duration of 1 second over HTTP/2. Boxplots include the average startup time for all 5835 considered users.

1 second is used and HTTP/2's server push is enabled, the startup time can be reduced from 3.1 to 1.7 (-43.5%) and 0.8 seconds (-74.1%) for eight and 32 prefetched videos respectively. In a 4G scenario, the startup time for a segment duration of 10 seconds over HTTP/1.1 is reduced from 1.6 to 1.0 (-37.7%) and 0.8 seconds (-64.6%) for eight and 32 prefetched videos respectively, and from 1.0 to 0.7 (-31.8%) and 0.5 seconds (-54.0%) for a segment duration of 1 second over HTTP/2.

Naturally, prefetching the content comes with a drawback: articles which are never requested, inevitably result in bandwidth overhead. De-



(a) Overhead per video session



(b) Estimation of the relative overhead

Figure 3.14: Bandwidth overhead as a function of the number of prefetched articles.

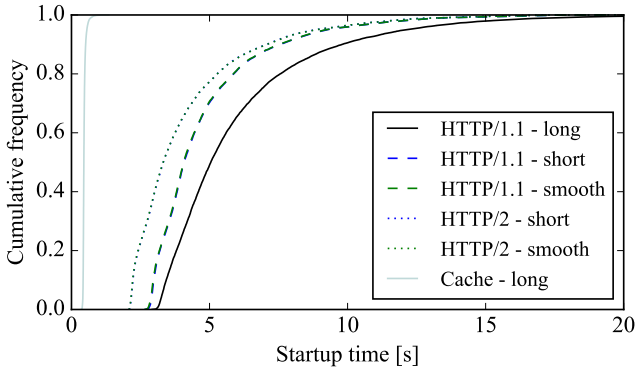
pending on the network carrier and the type of subscription, this can be detrimental to the user. In this context, Figure 3.14a shows the total overhead per video streaming session as a function of the number of videos the client is able to prefetch. When the eight most relevant videos are prefetched, for instance, the median overhead amongst all users is 1.8 MB per video session, while the 90% percentile equals 3.6 MB. As a reference, downloading the home page or a general single text-based article with JavaScript, CSS and images included, requires 3.2 and 1.9 MB respectively. These numbers are in the same order of magnitude, and therefore, we conclude that a value of eight is an ideal number of videos to prefetch by the client. Note that the required storage capacity is limited: the client only needs to be able to store the MPDs and initialization segments of eight

videos, along with $8d_n = 8 \cdot 10$ seconds $= 80$ seconds of content at the lowest video quality.

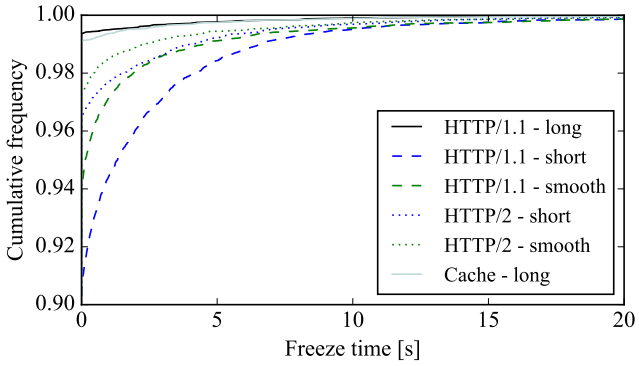
Next to the absolute amount of wasted bandwidth, it would be interesting to determine the bandwidth overhead relative to the bandwidth consumed without prefetching. However, since the dataset does not include requests for single video segments, it is not possible to extract the exact time each user spends watching the content, nor what the total bandwidth consumption was at the time. Furthermore, the total bandwidth usage depends on all issued requests, including requests for the home page and text-based article requests. As a rough estimation, we can however assume that the user finishes each video, and that the user either streams the whole session at the average bit rate of the lowest quality (i.e., 231 kb/s), or at the average bandwidth in the provided 3G traces (i.e., 2358 kb/s). For each user, we keep track of the requested video, and determine its length based on the provided MPD files. Multiplying the total video length with the average bit rate results in the total bandwidth consumption for video content - discarding other resources such as JS, CSS and images. As illustrated in Figure 3.14b, the relative overhead differs significantly for the two examples. When the eight most relevant videos are prefetched, for instance, the median overhead is 40.4% in the former case, and 4.0% in the latter. In the end, the relative overhead strongly depends on the viewing behavior of the user, the videos considered and the type of network conditions during the video streaming session.

3.4.7 Impact on Buffer Starvation

Reducing the number of bytes to transfer is an efficient way to reduce the video startup time. However, as mentioned previously in Section 3.3.1, reducing the segment duration also results in lower buffer filling at the start of the video playout. When the available bandwidth is insufficient to provide the video at the lowest quality, or when the segment duration is changed too abruptly, this can result in buffer starvation and therefore, in playout freezes. To evaluate the impact of the proposed optimizations on buffer starvation, the 13.636 videos in the dataset with a minimum length of one minute, have been used to evaluate six different configurations. Similar as in the previous evaluations, each video is started at a random point in the provided throughput traces (but at the same time for all configurations) and the first minute of content is played out completely. Because we are interested in the impact of the configurations only, the content is played out at the lowest quality: this way, the applied rate adaptation heuristics in the dash.js player do not come into play. Note that



(a) Startup time



(b) Freeze time

Figure 3.15: Cumulative distribution of the measured startup time and total freeze time for different configurations for the considered 13 636 videos.

the stalling threshold in the player, defined as the amount of content (in seconds) which should be left in the buffer in order to continue playout, is set to 0 (i.e., play out as soon as content is available, as long as content is available).

Figure 3.15 shows the cumulative distribution of the gains in measured startup time (top) and the total freeze time observed during playout of the first one minute of video (bottom), for six different configurations:

- HTTP/1.1 - long: a segment duration of 10 seconds;
- HTTP/1.1 - short: a segment duration of 1 second during the first 10 seconds, 10 seconds during the remainder of the stream;

- HTTP/1.1 - smooth: a segment duration smoothly changing from 1 to 2, 5 and eventually 10 seconds (a change occurs after each 10 seconds);

- HTTP/2 - short: idem as for HTTP/1.1, but now with HTTP/2 server push for shorter segments ($k = 10$);

- HTTP/2 - smooth: idem as for HTTP/1.1, but now with HTTP/2 server push for shorter segments ($k = 10, 5, 2$ respectively);

- Cache - HTTP/1.1: the first 10 seconds are retrieved from local storage, the remainder of the session from the server.

Figure 3.15a shows the results which were previously obtained: the startup time can be reduced significantly when the segment duration is lowered, when HTTP/2 server push is applied and when caching is used. Note that the "short" and "smooth" approaches result in the same startup time, since both require the same resources to start video playout (including a single one-second video segment).

Figure 3.15b shows the total freeze time observed during playout of the first minute of content. When the default configuration is used, only 0.6% of video streaming sessions suffers from playout freezes. When switching from the cache to the local server, a new TCP connection has to be started, which implies that the available bandwidth cannot immediately be used yet. For this reason, it can take a while before the second segment arrives, in cases of low bandwidth sometimes resulting in a playout freeze. When a segment duration of 1 second is used for the first 10 seconds of content, results show that the client is indeed more prone to buffer starvation: when using the "short" segment scheme, 9.5% of video streaming sessions results in playout freezes. Adopting the "smooth" scheme, this number is reduced to 6.1%. Applying HTTP/2 server push on top of that, a further reduction to 2.7% is achieved. It is worth noting that most rebuffering events do not last longer than 500 ms, which is significantly smaller than the gains in terms of startup time (a median reduction of 2.1 seconds, with outliers higher than 15 seconds). Higher values occur only in cases where the available throughput is significantly lower than the bit rate for the lowest quality level, in which case no approach can achieve a desirable result: one can argue that in the given use case, it can be better to start the stream under 10 seconds while temporarily suffering from rebuffering events, than starting the stream in 10 seconds or more and risking abandonment by the user.

3.4.8 Summary

In the evaluations above, we showed that the proposed optimizations can result in significantly shorter video startup times, which is beneficial when browsing news content in a video web portal. In summary, the main reductions are achieved by:

1. Using a shorter video segment duration: when the available bandwidth is limited, this can result in reductions in the order of 10 to 25%. This optimization is straightforward to implement, as it requires minor adaptations to the MPD and limited additional storage at the server-side;

2. Using HTTP/2 on the application layer, making use of HTTP/2 server push: even when the network delay is limited to 120 ms, this results in an additional reduction in the order of 15 to 25%. At the server-side, this optimization requires a separate request handler to push required resources. Since most browsers nowadays have full support for HTTP/2, no changes are required at the client-side;

3. Using server-side user profiling and client-side storage to prefetch (the first part of) possibly relevant videos: storing the eight most relevant video articles, for instance, additional reductions in the order of 45% can be achieved. This however comes at the cost of increased bandwidth usage by the client. This optimization requires server-side logging and analytics, content prefetching and client-side storage, making it less straightforward to deploy.

All aforementioned optimizations are complementary: one can, for instance, consider a scenario where a segment duration of 1 second is used, HTTP/2 server push is enabled and up to eight articles are prefetched on a per-user bases. Comparing results with the reference scenario for deredactie.be, i.e., a segment duration of 10 seconds over HTTP/1.1 without prefetching, the median startup time can be reduced from 5.0 to 1.7 seconds (-66.0%) in a 3G network scenario, and from 1.6 to 0.7 seconds (-56.3%) in a 4G network scenario.

3.5 Conclusions

In this work, a novel framework for low-latency delivery of news-related video content is presented. Its main components include server-side encoding, HTTP/2's server push, user profiling and client-side storage for

proactive content delivery. Through a relevant use case of a major Belgian news provider, we showed that each of the proposed optimizations can significantly reduce the median startup time of video streaming sessions, by 10 to 25% using a shorter video segment duration, by 15 to 25% using HTTP/2 server push and by 45% when the first 10 seconds of the eight most relevant videos are stored at the client-side. Combining these optimizations, the median startup time can be reduced by more than 50% in both 3G and 4G mobile networks. These reductions allow the news provider to improve the user's Quality of Experience, encouraging low-latency user interaction with the provided video content. In future work, we will characterize the performance of the considered optimizations in other use case scenarios, such as 360° video delivery for virtual reality.

References

[1] S. Egger, T. Hoßfeld, R. Schatz, and M. Fiedler. *Waiting Times in Quality of Experience for Web-Based Services*. In Proceedings of the International Workshop on Quality of Multimedia Experience, 2012.

[2] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia. *A Survey on Quality of Experience of HTTP Adaptive Streaming*. IEEE Communications Surveys Tutorials, 17(1):469–492, 2015.

[3] R. Mok, E. Chan, and R. Chang. *Measuring the Quality of Experience of HTTP Video Streaming*. In Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, pages 485–492, 2011.

[4] T. Stockhammer. *Dynamic Adaptive Streaming over HTTP: Standards and Design Principles*. In Proceedings of the ACM Conference on Multimedia Systems, pages 133–144, 2011.

[5] T. Hoßfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch, and C. Lorentzen. *Initial Delay vs. Interruptions: Between the Devil and the Deep Blue Sea*. In Proceedings of the International Workshop on Quality of Multimedia Experience, pages 1–6, 2012.

[6] L. Chen, Y. Zhou, and D. M. Chiu. *Video Browsing - A Study of User Behavior in Online VoD Services*. In Proceedings of the International Conference on Computer Communication and Networks, pages 1–7, 2013.

[7] J. van der Hooft, C. De Boom, S. Petrangeli, T. Wauters, and F. De Turck. *An HTTP/2 Push-Based Framework for Low-Latency Adaptive Streaming Through User Profiling*. In Proceedings of the IEEE/IFIP Network Operations and Management Symposium, 2018. Accepted for publication.

[8] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. *BOLA: Near-Optimal Bitrate Adaptation for Online Videos*. In Proceedings of the IEEE International Conference on Computer Communications, pages 1–9, 2016.

[9] H. Mao, R. Netravali, and M. Alizadeh. *Neural Adaptive Video Streaming with Pensieve*. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pages 197–210. ACM, 2017.

[10] Y. Sánchez de la Fuente, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, and Y. Le Louédec. *iDASH: Improved Dynamic Adaptive Streaming over HTTP Using Scalable Video Coding*. In Proceedings of the ACM Conference on Multimedia Systems, pages 257–264. ACM, 2011.

[11] H. Schwarz, D. Marpe, and T. Wiegand. *Overview of the Scalable Video Coding Extension of the H.264/AVC Standard*. IEEE Transactions on Circuits and Systems for Video Technology, 17(9):1103–1120, 2007.

[12] G. J. Sullivan et al. *Overview of the High Efficiency Video Coding (HEVC) Standard*. IEEE Trans. on Circuits and Systems for Video Technology, 22(12):1649–1668, 2012.

[13] A. Deveria. *Can I use HEVC?*, 2018. Available from: https://caniuse.com/#search=HEVC.

[14] Y. Shuai and T. Herfet. *On Stabilizing Buffer Dynamics for Adaptive Video Streaming with a Small Buffering Delay*. In Proceedings of the IEEE Consumer Communications Networking Conference, pages 435–440, 2017.

[15] K. Miller, A. Al-Tamimi, and A. Wolisz. *QoE-Based Low-Delay Live Streaming Using Throughput Predictions*. ACM Transactions on Multimedia Computing, Communications, and Applications, 13:4:1–4:24, 2016.

[16] J. Jiang, V. Sekar, and H. Zhang. *Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming with Festive*. IEEE/ACM Transactions on Networking, 22(1):326–340, 2014.

[17] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan. *Towards a SPDY'Ier Mobile Web?* In Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies, pages 303–314. ACM, 2013.

[18] Y. Elkhatib, G. Tyson, and M. Welzl. *Can SPDY Really Make the Web Faster?* In IFIP Networking Conference, pages 1–9. IEEE, 2014.

[19] X. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. *How Speedy is SPDY?* In Proceedings of the USENIX Conference on Networked Systems Design and Implementation, pages 387–399. USENIX Association, 2014.

[20] S. Wei and V. Swaminathan. *Low Latency Live Video Streaming over HTTP 2.0*. In Proceedings of the Network and Operating System Support on Digital Audio and Video Workshop, pages 37:37–37:42. ACM, 2014.

[21] W. Cherif, 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, pages 25–30. ACM, 2015.

[22] K. Gatimu, A. Dhamodaran, T. Johnson, and B. Lee. *Experimental Study of Low-Latency HD VoD Streaming Using Flexible Dual TCP-UDP Streaming Protocol*. In Proceedings of the Consumer Communications Networking Conference, pages 1–6, 2018.

[23] W3C/IETF. *Web Real-Time Communication (WebRTC)*, 2018. Available from: https://www.webrtc.org.

[24] ISO/ICE. *Dynamic Sdaptive Streaming over HTTP (DASH) - Part 5: Server and Network Assisted DASH (SAND)*, 2017.

[25] A. Bentaleb, A. C. Begen, and R. Zimmermann. *SDNDASH: Improving QoE of HTTP Adaptive Streaming Using Software Defined Networking*. In Proceedings of the ACM Multimedia Conference, pages 1296–1305. ACM, 2016.

[26] S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. *Software-Defined Network-Based Prioritization to Avoid Video Freezes in HTTP Adaptive Streaming*. International Journal of Network Management, 26(4):248–268, 2016.

[27] D. K. Krishnappa, M. Zink, C. Griwodz, and P. Halvorsen. *Cache-Centric Video Recommendation: An Approach to Improve the Efficiency of YouTube Caches*. ACM Transactions on Multimedia Computing, Communications and Applications, 11(4):48:1–48:20, 2015.

[28] M. Claeys, N. Bouten, D. De Vleeschauwer, W. Van Leekwijck, S. Latré, and F. De Turck. *Cooperative Announcement-Based Caching for Video-on-Demand Streaming*. IEEE Transactions on Network and Service Management, 13(2):308–321, 2016.

[29] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. *Bandwidth-aware Prefetching for Proactive Multi-video Preloading and Improved HAS Performance*. In Proceedings of the ACM Multimedia Conference, pages 551–560. ACM, 2015.

[30] Y. Koren, R. Bell, and C. Volonsky. *Matrix Factorization Techniques for Recommender Systems*. Computer, 42(8):30–37, 2009.

[31] C. De Boom, R. Agrawal, S. Hansen, E. Kumar, R. Yon, C. Chen, T. Demeester, and B. Dhoedt. *Large-Scale User Modeling with Recurrent Neural Networks for Music Discovery on Multiple Time Scales*. Multimedia Tools and Applications, 2017.

[32] J. Basilico and Y. Raimond. *DéJà Vu: The Importance of Time and Causality in Recommender Systems*. In Proceedings of the Conference on Recommender Systems, pages 342–342. ACM, 2017.

[33] T. Donkers, B. Loepp, and J. Ziegler. *Sequential User-based Recurrent Neural Network Recommendations*. In Proceedings of the Conference on Recommender Systems, pages 152–160. ACM, 2017.

[34] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. *Session-based Recommendations with Recurrent Neural Networks*. Computing Research Repository, abs/1511.06939, 2015.

[35] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. *An HTTP/2 Push-Based Approach for Low-Latency Live Streaming with Super-Short Segments*. Journal of Network and Systems Management, 26(1):51–78, 2018.

[36] I. Sodagar. *The MPEG-DASH Standard for Multimedia Streaming Over the Internet*. IEEE Multimedia, 18(4), 2011.

[37] S. Van Canneyt, B. Dhoedt, S. Schockaert, and T. Demeester. *Knowledge Extraction and Popularity Modeling Using Social Media*. 2016.

[38] T. Mikolov, K. Chen, G. Corrado, and J. Dean. *Efficient Estimation of Word Representations in Vector Space*. In Proceedings of the International Conference on Learning Representations Workshop, volume 2013, 2013.

[39] nperf. *Cellular Data Networks in Belgium*, 2018. Available from: https://www.nperf.com/en/map/BE/-/-/signal/.

[40] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen. *Video Streaming Using a Location-Based Bandwidth-Lookup Service for Bitrate Planning*. ACM Transactions on Multimedia Computing, Communications and Applications, 8(3):24:1–24:19, 2012.

[41] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck. *HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks*. IEEE Communications Letters, 20(11):2177–2180, 2016.

4

Tile-Based Adaptive Streaming for Virtual Reality Video

"Are you real?" - "If you can't tell, does it matter?"

–Westworld, 2016

In Chapters 2 and 3, traditional video content is considered. Over the last years, however, the increasing popularity of head-mounted devices and 360° video cameras has allowed content providers to provide virtual reality video streaming over the Internet. In this regard, a two-dimensional representation of the immersive content is typically used, combined with traditional streaming techniques. Since only a limited part of the video (i.e., the viewport) is watched by the user, such approach does not optimally use the available bandwidth. In this chapter, we discuss the advantages of tile-based video, adapting the quality of each of the resulting tiles to the network characteristics and user movement. We propose a content-agnostic viewport prediction scheme based on unidirectional spherical trajectories, and present two rate adaptation heuristics which take into account the spatial dimension. Furthermore, we introduce a novel feedback loop within the client's viewport prediction and rate adaptation schemes, which allows us to change quality decisions whilst downloading the required tiles for a given video segment, and discuss the advantages of HTTP/2 server push for content delivery.

⋆⋆⋆

J. van der Hooft, M. Torres Vega, S. Petrangeli, T. Wauters, and F. De Turck

In revision for publication in ACM Transactions on Multimedia Computing, Communications, and Applications, 2019

Abstract The increasing popularity of head-mounted devices and 360° video cameras allows content providers to provide virtual reality (VR) video streaming over the Internet, using a two-dimensional representation of the immersive content combined with traditional HTTP adaptive streaming (HAS) techniques. However, since only a limited part of the video (i.e., the viewport) is watched by the user, the available bandwidth is not optimally used. Recent studies have shown the benefits of adaptive tile-based video streaming; rather than sending the whole 360° video at once, the video is cut into temporal segments and spatial tiles, each of which can be requested at a different quality level. This allows prioritization of viewable video content, and thus results in an increased bandwidth utilization. Given the early stages of research, there are still a number of open challenges to unlock the full potential of adaptive tile-based VR streaming. The aim of this work is to provide an answer to several of these open research questions. Among others, we propose two tile-based rate adaptation heuristics for equirectangular VR video, which use the great-circle distance between the viewport center and the center of each of the tiles to decide upon the most appropriate quality representation. We also introduce a feedback loop in the quality decision process, which allows the client to revise prior decisions based on more recent information on the viewport location. Furthermore, we investigate the benefits of parallel TCP connections and the use of HTTP/2 as an application layer optimization. Through an extensive evaluation, we show that the proposed optimizations result in a significant improvement in terms of video quality (more than twice the time spent on the highest quality layer), compared to non-tiled HAS solutions.

4.1 Introduction

Over the last years, the popularity of virtual reality (VR) has increased significantly. This is partly due to recent advancements in consumer electronics, which allow the user to enjoy a fully immersive experience using low-cost head-mounted displays (HMD). Furthermore, well-known content providers such as YouTube[1] and Facebook[2] now allow to stream VR

[1] https://www.youtube.com
[2] https://www.facebook.com

video content. However, given today's network capacity, streaming immersive video at high resolution and a suitable frame rate is often not a straightforward task. For this reason, the principles of HTTP adaptive streaming (HAS) are often applied to deliver the immersive video over the best-effort Internet.

In HAS, video is encoded at different quality levels and temporally divided into multiple segments with a typical length of 2 to 30 seconds [1]. An HAS client requests these video segments at the most appropriate quality, based on e.g., the available bandwidth and the buffer size. To this end, a rate adaptation heuristic attempts to optimize the Quality of Experience (QoE) perceived by the user. This QoE depends, among others, on the average video quality, the frequency of quality changes and the occurrence of playout freezes [2]. The client stores incoming segments in a buffer before decoding the sequence in linear order and playing the video out on the user's device. These principles were standardized as the Dynamic Adaptive Streaming over HTTP (DASH) [3].

Nowadays, content providers often employ HAS to deliver 360° content in a similar manner as for traditional non-immersive video. To this end, the content is first mapped on a two-dimensional representation, using e.g., equirectangular projection. The resulting video is then encoded at multiple resolutions, temporally divided and made available for the client to download during the video streaming session. Fixing the quality/resolution for the whole video, however, results in suboptimal use of the available bandwidth: since the user has a limited view on the video when wearing an HMD (referred to as the viewport) a significant part of the network traffic is wasted on content which is not consumed. To overcome this issue, an additional dimension can be inserted by also considering spatial segmentation of the video. Using the HEVC/H.265 standard, for instance, equirectangular content can be split into $m \times n$ tiles of the same resolution. The client is able to request each tile at a different quality level, prioritizing tiles within the viewport by assigning a higher video quality.

Although beneficial, this approach poses a number of new challenges. First, the introduction of tiles induces an encoding overhead: a 16×16 tiling scheme, for instance, requires a higher bit rate to achieve the same visual quality than a 4×4 tiling scheme. Careful consideration must thus be given to the trade-off between the video's granularity on the one hand, and the required encoding/network resources on the other. Second, since a buffer is used to store incoming segments, tile-based video streaming is prone to user movements. Even when the considered buffer contains a mere 2 seconds of video, the user could temporally perceive a lower quality when moving around within the video scene. Therefore, accurate prediction of

future viewport coordinates is of the utmost importance. Third, the client-side rate adaptation heuristic needs to take into account the new spatial dimension in the decision-taking process. This increases the heuristic's complexity, as it should take into account the perceived video quality and avoid buffer starvation. Fourth, since multiple HTTP GET requests are required, the approach is affected by network latency. This is especially true for mobile networks, where round-trip times are in the order of 30-70 and 50-200 ms for 4G/LTE and 3G/HSPA(+) respectively [4, 5]. Finally, the introduction of tiles makes it harder to evaluate the resulting video quality within the viewport, impeding accurate quality assessment.

etIn this work, we aim to address the above-mentioned challenges of tile-based HAS for VR video, building further on preliminary research efforts [6, 7]. The main contributions of this paper are threefold. First, two rate adaptation heuristics for tile-based video quality assignment are presented, prioritizing tiles according to their great-circle distance to the viewport center. Using the proposed heuristics, the available bandwidth is mostly assigned to tiles within the visible region of the user, resulting in a higher overall video quality. Second, we introduce a novel feedback loop which allows to change quality decisions whilst downloading the required tiles for a given video segment. This enables the client to revise earlier decisions, given newer information on the user's focus. Third, we evaluate the above components in a comprehensive framework for tile-based 360° video, which includes viewport prediction and application layer optimizations for end-to-end delivery. This allows us not only to evaluate results for the proposed heuristics and feedback mechanism, but also to assess the impact of the different components on the overall video quality.

The remainder of this chapter is structured as follows. In Section 4.2, the general HAS architecture for VR streaming is presented and state-of-the-art solutions are discussed. Suggested approaches are detailed in Section 4.3, elaborating on the advantages of each of our proposed optimizations. The experimental setup and evaluation results are presented in Section 4.4, along with a discussion on lessons learned. Finally, conclusions and future work are presented in Section 4.5.

4.2 State-of-the-Art and Challenges

As illustrated in Figure 4.1, the HAS principle can be adopted for tile-based VR video streaming. To this end, content is captured by a 360° camera, encoded at different qualities, temporally and spatially segmented and made available on the server. At the client-side, a head-mounted device is used to consume the immersive content, registering the user's movement and

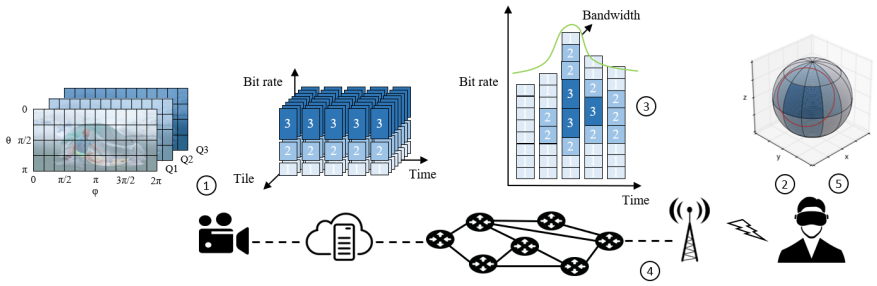


Figure 4.1: The HAS principle applied to VR video streaming.

regions of interest within the video. Based on the available bandwidth and current or future viewport coordinates, a rate adaptation heuristic decides at which video quality to download each of the tiles of the next video segment. The content is requested from the server, buffered in a client-side buffer and finally played out on the user's device. Below, we elaborate on the five most relevant components for over-the-top video streaming: i) (live) video capture and encoding; ii) viewport prediction; iii) tile-based rate adaptation; iv) application layer optimizations; and v) video quality evaluation.

4.2.1 Video Capture and Encoding

Tile-based encoding for VR video is often achieved through the HEVC standard, implemented by encoders such as HEVC Test Model (HM[3]). Many mapping schemes exist, including cubic, pyramid and dodecahedron mappings [8]. As an example, Skupin et al. propose to use a cubic projection consisting of 24 tiles, each of which is provided at two different resolutions [9]. Based on the user's focus, the client can then decide upon the most appropriate resolution for each of these tiles. Nevertheless, equirectangular mapping is most commonly used, in which the sphere is mapped onto a rectangle. This results in stretching at the poles of the sphere, which reduces the encoding efficiency and can result in an increased bandwidth consumption. To overcome this, Budagavi et al. propose to gradually smoothen the quality of the polar parts of the video, reducing the bit rate by 20% [10]. Using a similar starting point, Hosseini et al. propose to use a different tiling structure altogether, using four central and two polar tiles [11].

[3]https://hevc.hhi.fraunhofer.de/HM-doc/

Other approaches for VR content encoding have been considered as well. Team Pixvana, for instance, provides commercial solutions for field-of-view adaptive streaming, in which thirty different versions of the video, each focusing on a unique viewport, are created [12]. Similarly, Kuzyakov et al. propose a pyramid mapping with thirty unique viewports, allowing bit rate reductions of up to 80% compared to non-tiled video [13]. Zare et al. consider twelve unique viewports, using a 12×4 tiling scheme which focuses on equatorial tiles only (i.e., polar tiles are always retrieved at the lowest quality) [14]. The advantage of these encoding techniques is that video segments can be requested as a whole, reducing the number of GET requests and the impact of latency, while still allowing to allocate bandwidth to the most important regions. However, they require significant encoding efforts and additional storage space - up to five quality representations are provided for each viewport configuration - compared to tile-based encoding. Although there are benefits to these approaches, they limit the granularity of the quality decision-making. In this work, we will therefore allow the client to request the most appropriate quality representation on a per-tile basis.

Regarding video encoding, it should be noted that MPEG has recently standardized motion-constrained tile sets (MCTS) for viewport-based transmission of 360° HEVC-encoded video [15]. Implementations such as the one by Son et al. [16] now allow to extract and decode a limited subset of tiles from a given video bitstream, rather than having to decode the full video. Finally, it is worth mentioning that a scalable variant of HEVC, SHVC, exists. By combining a scalable approach with tiling, yet another dimension is created. As discussed by Taghavi Nasrabadi et al., a layered approach to tile-based encoding reduces storage and bandwidth requirements, and results in a lower amount of buffer starvations [17]. Scalable approaches are not commonly used, however, and given the large amount of GET requests needed to retrieve the content, suffer from latency in the network. Furthermore, an encoding overhead of about 10% is introduced per quality layer. In this paper, we will therefore focus on non-scalable encoding only.

In our evaluations, we will use the HEVC standard to tile the considered equirectangular video using an $m \times n$ tiling scheme. This approach allows for more finegrained decision-making by the client, using the available throughput where it is needed. Using different tiling schemes, we will evaluate the encoding overhead and the resulting video quality under similar network conditions. Similar to related work, the Constant Rate Factor (CRF) rate control will be used to differ between visual quality, and thus resulting bit rates for different quality representations.

4.2.2 Viewport Prediction

In a video streaming use case, content is requested from the server through the best-effort Internet. In order to avoid playout freezes, the client is thus equipped with a playout buffer. In practice, this buffer is kept as small as possible, so that the video quality can be updated as fast as possible when the user moves to another region within the immersive video. Still, when a one to five seconds buffer is used, priority can potentially be given to the wrong tiles if the user keeps on moving within the video scene. For this reason, viewport prediction is an important means to tile-based HAS: if the future position is accurately predicted, the client can compensate for the user's movements and download relevant parts of the video at higher quality. In related work, a distinction can be made between content-agnostic and content-aware viewport prediction.

Content-agnostic approaches take into account information on the viewport only, in order to predict future movement. Petrangeli et al. propose to use linear extrapolation of the user's recent trajectory on the equirectangular projection of the video [6]. Given the user's most recent position P_c and a position P_p Δt_p seconds ago, the user's position Δt seconds from now is estimated by $P_c + \frac{\Delta t}{\Delta t_p}(P_c - P_p)$. Qian et al. apply (weighted) linear regression on the yaw, pitch and roll angles [18]. The authors show that the prediction accuracy decreases for a higher Δt. Similarly, Xu et al. use linear regression to prove that the variation of the prediction error increases for a higher Δt [19].

Content-aware solutions take into account not only the location of the viewport, but also information on the content itself, defining regions of interest, moving objects and key frames within the video. Focusing on tile-based VR streaming, Fan et al. use neural networks that concurrently leverage sensor-related features (i.e., the viewport position) and content-related features (i.e., image saliency and object motion maps) to predict the future viewport position [20]. Sitzmann et al. observe a fixation bias in immersive video, which is used to apply existing saliency predictors for two-dimensional video to VR [21]. Hu et al. take the viewport prediction process one step further, making the video move automatically according to previous user behavior [22]. To this end, a deep learning-based agent shifts the current viewing angle based on main object extraction. Similar approaches have been used to automatically align video cuts, perform panorama synopsis or even saliency-based video compression.

Although there are benefits to content-aware solutions, a significant amount of user data and processing efforts is required for new video content. Furthermore, these solutions introduce processing delays in live

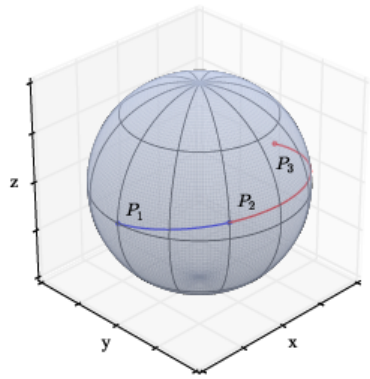


Figure 4.2: Viewport prediction using spherical walks. When the user moves from point P_1 to P_2 in a certain amount of time (e.g., 100 ms), point P_3 is predicted by extending the current trajectory unidirectionally.

streaming scenarios. In this paper, we will use a content-agnostic approach which has been proposed in previous work [7]. The considered scheme considers the user's movement in a brief interval only, taking note of the current position and the position 100 ms in the past. Based on this information, the current path on the surface of the sphere is extended unidirectionally to predict the future user position (see Figure 4.2). In previous work, a parameter sweep showed that following this trajectory for 400 ms resulted in the highest prediction accuracy when a buffer size of 2 seconds is used [7]. Since this buffer size is considered in this paper as well, the same configurations will be used in our evaluations.

4.2.3 Tile-Based Rate Adaptation

Adding a spatial dimension increases the complexity of the rate adaptation heuristic. Recently, a number of solutions have been proposed to address this extension. Ghosh et al. present an Integer Linear Program (ILP) to decide upon the most relevant quality representation of tiles, optimizing a model for the QoE [23]. This approach however rests on a number of assumptions, and, since ILPs are computationally expensive, is expected not to run in real-time when a large number of tiles and/or quality representations is considered. Zare et al. propose tile-based video streaming for 360° video, using two representations only: the original video, and a low-quality representation generated by lowering the resolution of the original video by half [24]. The client's logic does not take into account the per-

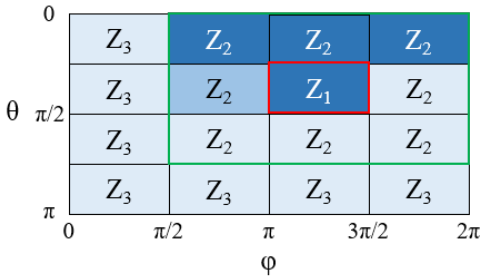


Figure 4.3: Illustration of the rate adaptation heuristic proposed by Hosseini and Swaminathan [11].

ceived bandwidth, but rather adapts to the considered viewport only: tiles within the viewport are requested at the highest quality, while all others are requested at the lowest quality. Le Feuvre et al. propose a heuristic based on fixed tile priorities, including uniform and center-based priority schemes [25]. The authors do not consider viewport changes, however, and evaluate results for a non-interactive environment only (segment duration 10 s). Hosseini et al. suggest a rate adaptation heuristic which is based on three regions of priority: Z_1, containing the center tile; Z_2, containing the surrounding tiles; and Z_3, all others (see Figure 4.3) [11]. Initially, all tiles are assigned the lowest video quality. Then, zone per zone, the quality of each tile is increased from the lowest to the highest level, as long as the available bandwidth is not exceeded. Finally, the quality of the last considered tile is increased to the highest quality supported by the remaining bandwidth budget. While it is indeed recommended to assign higher weights to tiles close to the current viewport location, the proposed heuristic does not take into account the fact that some tiles contribute more to the viewport than others. Furthermore, using a 3×3 bounding box in an equirectangular projection is not always representative of the user's viewport (e.g., when the user is looking to the zenit). Petrangeli et al. propose a scheme based on center and polar zones [6]. Starting from a 4×4 tiling scheme, the top and bottom row tiles are concatenated, and the tiles on the second and third row are concatenated column-wise (see Figure 4.4). The resulting six tiles are divided on a per-zone level, depending on the current and predicted viewport position. Then, qualities to tiles are assigned zone by zone. Using this approach allows to reduce the number of GET requests, which is important in high-latency environments, but strongly reduces the granularity of the video. Heuristics for other types of VR applications exist as well, focusing for instance on rate adaptation for zoomable video [26]. In this paper, however, we will focus on regular VR video only.

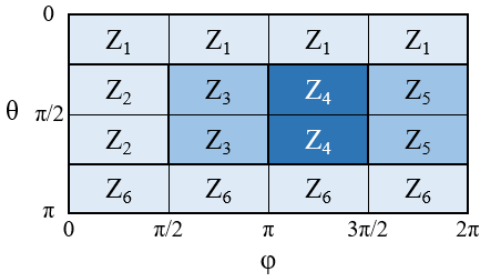
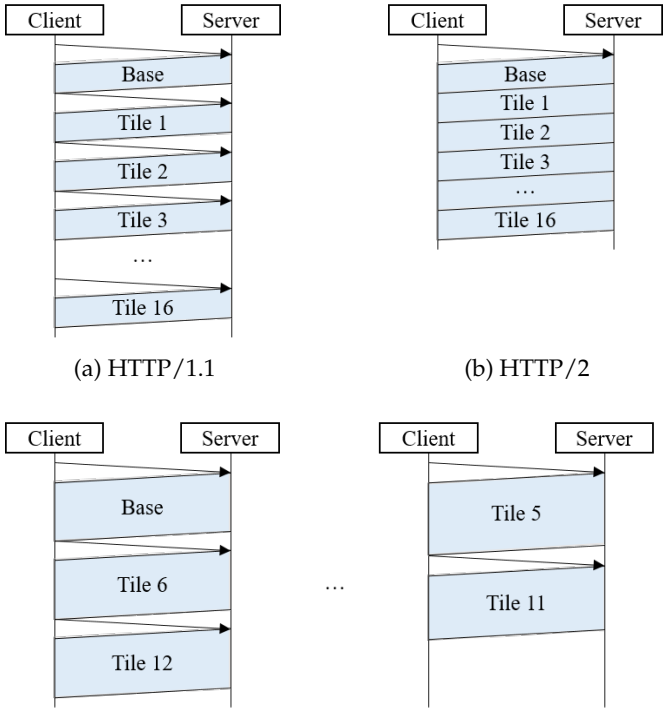


Figure 4.4: Illustration of the rate adaptation heuristic proposed by Petrangeli et al. [6].

4.2.4 Application Layer Optimization

The HTTP/1.1 protocol has been around since 1997, and is implemented in all major Internet browsers [27]. Most HAS solutions use this protocol with request-response transactions to retrieve the required resources, buffering fetched video segments and playing them out in linear order. In traditional HAS, only one GET request is needed in order to retrieve the next temporal part of the video; therefore, this approach is feasible as long as the duration of the video segments is of a higher order than the latency within the network. In VR-based HAS, however, multiple resources have to be retrieved in order to play out one temporal video segment: a GET request is required both for the base layer tile and for each of the spatial video tiles. When a 4×4 tiling scheme is used, for instance, no less than 17 requests need to be issued by the client. Using a segment duration in the order of 1 s, even a minor network latency of 20 ms will significantly impede the overall throughput between client and server, and will therefore result in a lower video quality.

There are two ways to deal with this issue. First, an approach based on multiple persistent TCP connections can be used (Figure 4.5c). Although the HTTP/1.1 RFC originally specified that no more than two persistent TCP connections should be used per server, this requirement was later lifted [27]. Most browsers today (including Chrome, Firefox and Safari) use up to six parallel TCP connections in order to reduce the page load time, fetching required resources in parallel. This allows to increase the overall throughput, and partly eliminates idle RTT cycles introduced by network latency. Similarly, a VR-based HAS client can use multiple TCP connections to download the different tiles in parallel, resulting in these very same advantages.



(a) HTTP/1.1

(b) HTTP/2

(c) HTTP/1.1 with six persistent TCP connections

Figure 4.5: Using multiple parallel TCP connections allows to increase the observed throughput and reduce the impact of latency. HTTP/2 server push allows to send resources back-to-back, eliminating the need for additional GET requests. Note that push promise and window increase frames are not shown for HTTP/2.

Second, the need for an excessive number of GET requests can be eliminated by using the server push feature of the HTTP/2 protocol (Figure 4.5b). HTTP/2 was standardized in 2015, introducing multiplexing of requests and responses to partly avoid the head-of-line blocking in HTTP/1.x[4], header compression and prioritization of requests [28]. Because of its interesting features and its ability to reduce page load times, most major browsers now have support for HTTP/2. According to W3Techs, 32.7% of the top 10 million websites support HTTP/2 as of January 2019 [29]. Wei et al. have shown that pushing shorter video seg-

[4]It is worth noting that not all head-of-line blocking can be avoided, since a single TCP connection is used in HTTP/2.

ments back-to-back, can reduce the video's startup time and end-to-end delay in HAS [30]. More recently, Petrangeli et al. showed that server push can also be applied in the case of tile-based VR streaming, where the client can request the server to push all tiles belonging to a single segment simultaneously [6]. In this work, we will use a custom request handler at the server-side, allowing the client to define a list of quality levels for each of the tiles to retrieve. The decisions of the applied rate adaptation heuristic can thus be communicated to the server, which allows the client to retrieve all tiles at the desired quality level.

4.2.5 Quality Evaluation

The ultimate goal of any video streaming optimization is to improve the user experience or QoE. In regular HAS, many factors have shown to affect the QoE: the video quality, the occurrence of playout freezes, the video's startup time, the end-to-end latency in live streaming, etc. [1]. When VR content is considered, new factors appear. For instance, frequent switching between different quality representations can result in lower QoE. This is true not only on a temporal level, such as in regular HAS, but also on a spatial level in tile-based solutions. Furthermore, the speed with which the quality is adjusted when moving around plays a crucial role as well: if the user has to wait several seconds before the quality is adjusted, the QoE will be strongly affected. With regard to the video quality perceived within the viewport, a number of evaluation metrics are being used in related work. Some works report the average PSNR values for the obtained video [19, 31], or show results in terms of video bit rate [19, 31, 32]. Other works consider the quality of the tile in the center of the viewport only, either to average this quality over all segments, or to measure the time spent on each layer [6]. Although the latter has shown to be an excellent evaluation metric for regular HAS, it is uncertain whether it is directly applicable to VR: the (potentially lower) video quality of surrounding tiles can have a significant impact on the perceived quality in the center of the viewport. Some works consider the video quality for a subset of tiles, where tiles are weighted according to a predefined zone they are located in [33].

The above metrics do not take into account the location of the user's gaze within the viewport. As shown in results obtained by Rai et al., presented in Figure 4.6, the user's eyes are rarely fixed to the center of the viewport; rather, a peak is observed for angles between 12 and 20 degrees [34]. In previous work, we therefore proposed to weigh the quality of each of the tiles within the viewport, taking into account the distribution of the user's gaze [35]. To this end, a density function is constructed based on

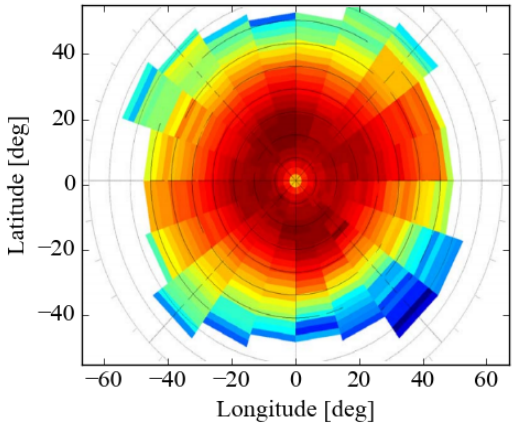


Figure 4.6: Heatmap of the users' gaze, relative to the center of the viewport [34]. Blue and red colors indicate a low and high frequency respectively.

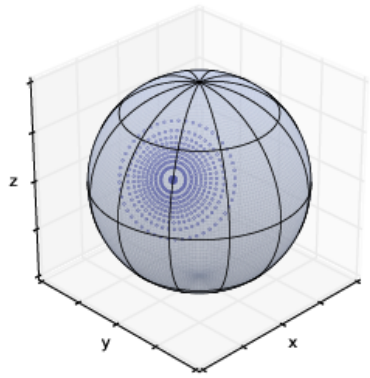


Figure 4.7: Viewport sampling for $n_1 = 10$ and $n_2 = 50$, based on the distribution of the distance between the viewport center and the user's gaze.

the heatmap in Figure 4.6 and a frequency histogram. Using uniform sampling on the cumulative distribution, n_1 circles of latitude are constructed, each of which is uniformly sampled n_2 times (see Figure 4.7). The overall video quality can then be expressed as the average video quality over all $n_1 \cdot n_2$ points, where the latter is defined as the quality level (ranging from 1 (lowest) to q_{max} (highest)), the average bit rate, the peak signal-to-noise ratio (PSNR) or the structural similarity index (SSIM) of the tile to which the considered point belongs. This removes the bias of focusing on

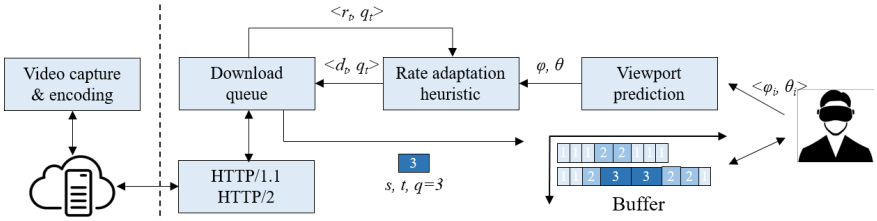


Figure 4.8: Required components for VR-based HAS.

the quality of the tile at the center of the viewport only, and therefore reflects more accurately the quality observed by the user. It is worth noting, however, that this metric does not take into account switches in quality between adjoining spatial regions, and does not take into account the fact that the speed of the head movement has an influence on the perception of the user. Such QoE model requires extensive subjective studies, which are not considered in this work.

4.3 Proposed Framework

In the previous section, state-of-the-art for VR-based HAS was discussed, along with new challenges and possible ways to tackle these. Below, we present a number of novel approaches and optimizations for VR video streaming. To make our contributions more clear, Figure 4.8 shows a component-based version of the illustration in Figure 4.1. The following components can be distinguished:

1. At the server-side, the video is captured, encoded and made available for the client. In this paper, we consider the traditional equirectangular projection for tile-based encoding, investigating the trade-off between granularity and encoding overhead;

2. At the client-side, the HMD monitors the user's viewport coordinates, which are used by a viewport prediction algorithm to predict future coordinates. Based on previous research by Petrangeli et al. [6], we propose a straightforward extension to the three-dimensional space;

3. A rate adaptation heuristic decides upon the quality assigned to all tiles belonging to the next video segment. We propose two new heuristics based on the great-circle distance between the viewport center and the center of each tile;

4. Decisions on the tiles' quality are forwarded to a download queue, which requests all files from the server. While the rate adaptation heuristic is typically consulted once for each segment, we introduce a feedback loop which allows the client to partly update its decisions once new estimations of the viewport coordinates are available;

5. Traditional HAS solutions typically use a single persistent TCP connection between client and server, requesting files one by one. Introducing tile-based encoding however results in a significant amount of GET requests for each video segment. We discuss the advantages of multiple persistent connections and HTTP/2's server push.

4.3.1 Tile-Based Rate Adaptation

In this paper, we propose two rate adaptation heuristics. A first heuristic, based on uniform viewport quality (UVP), focuses strongly on the (predicted) viewport location, uniformly increasing the quality of each tile whose center is within the viewport. A second heuristic, using a center tile first (CTF) approach, increases the quality of the tiles to the highest representation, in the order of decreasing distance between the center of the tile and the viewport location.

4.3.1.1 Uniform Viewport Quality

The UVP heuristic attempts to increase the quality of all tiles within the predicted viewport, one quality representation at a time. The choice for a homogeneous quality is a deliberate one: the human eye is sensitive to changes in the perceived quality, especially when changes occur within the spatial domain (contrary to regular HAS, where changes in video quality only occur in the temporal domain). In addition, this approach reduces the sensitivity to viewport prediction errors: even when relatively high errors are made, the resulting video quality in the center of the viewport should still be similar to the quality in the previously predicted point.

The rate adaptation heuristic is presented in Algorithm 4.1. Given the segment index s and the perceived bandwidth BW, this heuristic determines the most appropriate quality level for each of the video tiles. To this end, it uses a *Size*-function, which determines the file size corresponding to a given segment, tile and quality representation, and a *Distance*-function, which calculates the great-circle distance between the center of the tile and the given spherical coordinates. The algorithm is as follows. If the client has just started buffering, all tiles are downloaded at the lowest quality (lines 1-3). When enough segments have been retrieved, the total budget

is calculated, based on the available bandwidth and the segment duration dur (line 4). If the total file size of all tiles at the lowest quality exceeds this budget, the lowest quality is selected for each tile (lines 5-7). Similarly, if the total file size of all tiles at the highest quality does not exceed this budget, the highest quality is selected for each tile (lines 8-9). If none of these conditions is met, the heuristic will proceed to allocate the tile qualities. First, the great-circle distance between the center of the viewport, indicated by ϕ, θ, and the center of each of the tiles is calculated (lines 10). Then, tiles are divided according to whether or not the center is within the viewport (i.e., the distance is lower than half the viewport size vp), and the total number of bits for the current configuration (at the lowest quality) is calculated (lines 11-13). Next, the heuristic attempts to increase the quality of each tile within the viewport until either all tiles have the same quality, or the budget has been exceeded (lines 14-21). This process is repeated until either the budget is exceeded or all the tiles within the viewport are assigned the highest video quality. If there is still budget left, a similar process starts for each of the remaining tiles. The computational complexity of the considered rate adaptation heuristics is $O(n \log n + mn)$, where n is the number of tiles and m is the number of available quality representations: the tiles are sorted according to their distance to the center of the viewport, and for each tile, up to $m - 1$ quality increases are considered.

Although we envision an equirectangular projection in this paper, it is worth noting that the proposed rate adaptation heuristics can also be used for other projections: as long as a distance metric for each of the tiles can be defined, either by determining the distance to a single point (the center, if it is defined), or to multiple points (e.g., on the edge of the tile), tiles can be ranked and given priority accordingly. Furthermore, it is worth noting that the file size of the tth tile for the sth segment at quality q can be determined in a number of ways, depending on the considered use case. In a live streaming scenario, the size of the tiles and segments is not known in advance. One possibility is then to use the average bit rate of the tth tile over all previously encoded segments. Alternatively, one can use the average bit rate of the whole video, and estimate the file size of the tth tile by using an appropriate weight indicator that takes into account the applied tiling scheme. In a VoD scenario, the file size of each of the tiles is known by the server. In such a scenario, the file size can be explicitly stated in the MPD (an approach also used by Juluri et al. for regular HAS [36]).

4.3.1.2 Center Tile First

The CTF heuristic uses a more greedy approach, increasing the quality of relevant tiles to the highest representation. A similar approach was pro-

Algorithm 4.1: First proposed rate adaptation heuristic, uniform viewport quality (UVQ).

Input: m, n, the number of rows/columns
BW, the available bandwidth $[b/s]$
dur, the segment duration $[s]$
$buffer$, the buffer size $[s]$
n_{qual}, the number of quality representations
vp, the width of the viewport $[deg]$
ϕ, θ, the viewport coordinates

Output: *qualities*, specifying the quality representation of each tile

$qualities \leftarrow ones(m \cdot n)$
if $s \leqslant buffer/dur$ **then**
  return *qualities*
$budget \leftarrow BW \cdot dur$
$tiles \leftarrow [1; m \cdot n]$
if $\sum_{t \in tiles} Size(s, t, 1) \geqslant budget$ **then**
  return *qualities*
if $\sum_{t \in tiles} Size(s, t, n_{qual}) \leqslant budget$ **then**
  return $qualities \cdot n_{qual}$
$dists \leftarrow [Distance(t, \phi, \theta), \forall t \in tiles]$
$tiles_{in} \leftarrow [t, \forall t \in tiles : dists[t] \leqslant vp/2]$
$tiles_{out} \leftarrow tiles \setminus tiles_{in}$
$n_{bits} \leftarrow \sum_t Size(s, t, 1)$
for $tiles \in [tiles_{in}, tiles_{out}]$ **do**
  for $q \in [2; n_{qual}]$ **do**
    for $t \in Sort(tiles, distances)$ **do**
      $cost \leftarrow Size(s, t, q) - Size(s, t, q-1)$
      if $n_{bits} + cost > budget$ **then**
        return *qualities*
      $n_{bits} \leftarrow n_{bits} + cost$
      $qualities[t] \leftarrow q$

posed by Hosseini and Swaminathan [11], but with a key difference: rather than defining three priority zones and iterating over tiles zone by zone, we

Algorithm 4.2: Second proposed rate adaptation heuristic, center tile first (CTF). Only lines 11-21 are changed compared to Algorithm 4.1.

```
11 n_bits ← Σ_t Size(s, t, 1)
12 for t ∈ Sort(tiles, dists) do
13     for q ∈ [2; n_qual] do
14         cost ← Size(s, t, q) − Size(s, t, q − 1)
15         if n_bits + cost > budget then
16             return qualities
17         n_bits ← n_bits + cost
18         qualities[t] ← q
```

propose to sort the tiles according to the great-circle distance between the viewport coordinates and the center of each tile. Then, the quality of the tiles is increased to the highest representation, until no more bandwidth is available. In Algorithm 4.2, we therefore calculate these distances and sort the tiles (lines 10-12), and increase the video quality starting with the closest tile. Each tile is assigned the highest quality, until no more bandwidth budget is available. In this case, the quality of the last tile is increased to the highest quality which still fits within the remaining budget. Compared to UVP, this algorithm is more aggressive towards the center of the viewport: the highest quality is immediately assigned, without taking into account the remainder of the viewable footage.

4.3.2 Feedback Loop for Quality Reassignment

As discussed above, each of the required components for VR-based HAS provide input to one another. Indeed, the client detects the coordinates of the user's viewport in the HMD; a viewport prediction algorithm predicts the coordinates of the user at the time of playout; the rate adaptation heuristic makes a decision on the quality representation for each of the tiles; a network module requests all content to the server. Although this approach has significant advantages, it is prone to changes on the initial conditions: the user might change its trajectory, or the download of certain tiles might take longer than expected. Furthermore, the lower the time between the viewport prediction and the playout of the segment, the more accurate the prediction is expected to be. For these reasons, we propose to incorporate an additional feedback loop in the VR-based HAS scheme.

Algorithm 4.3: Pseudocode for buffering with a feedback loop in the tile-based VR player.

Input: m, n, the number of rows/columns
n_{seg}, the number of video segments
predictor, the viewport prediction algorithm
adapter, the rate adaptation heuristic
queue, the download queue

```
1 tiles_all ← [1; m · n]
2 for s ∈ [1; n_seg] do
3     tiles ← tiles_all
4     while len(tiles) > 1 do
5         φ, θ ← predictor.predict(s)
6         qualities, distances ← adapter.adapt(s, φ, θ, tiles)
7         queue.update(s, qualities, distances)
8         tiles ← tiles_all \ queue.requested()
```

Algorithm 4.3 shows the proposed logic in its simplest form. As soon as a segment is removed from the buffer and playout is started, the client can start buffering the next segment. To this end, the viewport prediction algorithm predicts the expected viewport coordinates ϕ, θ at the time of playout of this segment, based on the client's location history (line 5, $<\phi_i, \theta_i>$ in Figure 4.8). This prediction is then fed to the rate adaptation heuristic, which makes a decision on the quality of each of the tiles based on the viewport coordinates, the observed throughput and the available quality representations. Its calculations and decisions (i.e., the quality q_t and the distance d_t for each tile t) are forwarded to a download queue, which starts retrieving the tiles that are furthest away from the currently predicted viewport location (lines 6-7). This particular ordering is chosen because it allows the client to first retrieve the tiles which are expected to have the least impact on the final viewport, and it does not affect the chances of freezing (all tiles need to be downloaded before playout of the next segments can start). Whilst the downloading is taking place, the viewport coordinates of the HMD are regularly updated, so that new predictions can be made. Using the new viewport location, along with information of the download queue (r_t, whether or not tile t has already been requested) (line 8), the heuristic reassigns the reduced bandwidth budget among all tiles which have not yet been downloaded, and forces an update of the data in the download queue. Initial decisions can thus be overruled by the client, in favor of tiles

which are expected to be of higher importance than previously estimated. It is important to note that tiles are only downloaded once: the quality of previously requested tiles thus remains unchanged.

An example of the proposed solution is presented in Figure 4.9, where the CTF heuristic is used with a 4×4 tiling scheme. Based on recent user movement, the client predicts the viewport center two seconds in the future. It then sorts the tiles according to the distance of the center of each of the tiles to these viewport coordinates, and assigns each tile a given quality level (Figure 4.9a). Subsequently, the client proceeds to request the tiles in order, starting with the tiles which are furthest away. Whilst the second tile is being downloaded, the user changes its trajectory, forcing the viewport prediction algorithm to update its predicted coordinates. The rate adaptation heuristic recalculates the assigned quality levels, keeping the quality of the tiles which have already been requested unchanged (Figure 4.9b). This procedure is repeated throughout the download process, as long as there are at least two tiles which have not yet been requested by the client.

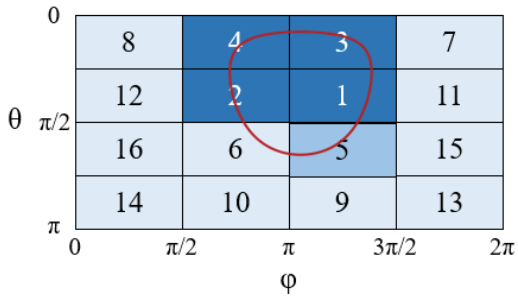
4.4 Evaluation and Discussion

To evaluate the proposed optimizations, a dataset provided by Wu et al. was chosen [37]. This dataset contains traces for 48 unique users and 9 different VR videos, specifying the coordinates of the viewport center throughout all video sessions with an average sampling rate of 47 Hz. For reasons of time complexity, we selected three videos which show significantly different features: Sandwich (an indoor performance of length 164.22 s), Spotlight (an action movie of length 293.28 s) and Surf (an ensemble of surf clips made using a GoPro camera of length 205.72 s). Below, we first discuss the experimental setup, the considered evaluation metrics and the evaluation space. Then, we present results for each of the proposed optimizations.

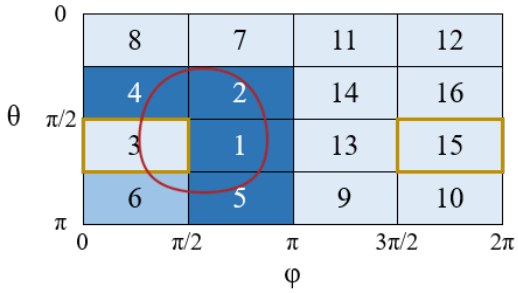
4.4.1 Experimental Setup

Each of the considered videos is encoded using the HM encoder[5], applying a relevant and diverse set of tiling schemes at 4K resolution and 30 FPS. The Group of Pictures (GOP) length is set to 32, resulting in a segment duration of around 1.067 s. The CRF factor for the different quality representations is set to 15, 20, 25, 30 and 35 respectively, resulting in the average bit rates specified in Table 4.1. Note that the Sandwich and Spotlight video come at an original resolution of 3840×2160 (ratio 16:9), while the

[5]https://hevc.hhi.fraunhofer.de/HM-doc/



(a) Initial quality assignment



(b) Updated quality assignment

Figure 4.9: An example of the presented quality reassignment scheme, with quality representations ranging from lowest (light blue) to highest (dark blue). Tile numbers indicate the tile's index when sorting according to the great-arc distance to the viewport center (lower is closer). The client starts requesting the two tiles which are furthest away, until a new viewport prediction is available. These tiles are stored in the buffer at the previously requested quality (indicated in orange), while the quality of the remaining tiles is changed according to the new viewport position and the considered rate adaptation, using a revised bandwidth budget.

Surf video is provided at 3840×1920 (ratio 2:1). Both types of video were rescaled to 4096×2304 and 4096×2048 respectively, so that tiles, whose width and height is required to be a multitude of 64 by the HM encoder, can be uniformly defined. Given the amount of movement in the GoPro video, we observe significantly higher bit rates for the lowest video quality (e.g., 2.3 Mb/s versus 1.1 and 1.3 Mb/s when no tiling is used). We also observe that the impact of the tiling overhead is limited, up until the number of tiles reaches 16×12 or 16×16: comparing these schemes with no tiling, an average encoding overhead of 1.3, 1.4, and 1.9 Mb/s, or 127.4%,

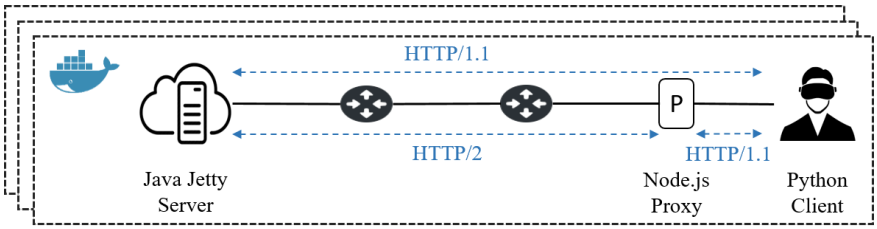


Figure 4.10: Experimental setup. Mininet is used to host a virtual network within a Docker container. A Python VR player requests content from a Jetty server or through a Node.js proxy with built-in support for HTTP/2.

112.5% and 85.3%, is observed for the lowest quality representation of the Sandwich, Spotlight and Surf video respectively.

To evaluate the impact of the available bandwidth and network latency, a network setup is emulated using Mininet[6], where a client is connected to an HTTP/2-enabled Jetty server[7] (Figure 4.10). The open-source code of the Java-based server is slightly modified, providing a custom request handler for the pushing of tile-based video segments. The client is a headless Python-based implementation. This client provides support for different viewport prediction schemes, rate adaptation heuristics and quality metrics. A buffer size of two segments, or 2.133 s, is used in our evaluations. The viewport size is set to 110°, similar to the VIVE head-mounted display used by Wu et al. during data collection [37]. To allow seamless connection over HTTP/2, a Node.js proxy is provided which serves as a client to the HTTP/2-enabled server, transparently forwarding the required push request and handling incoming files as an intermediate for the VR client. The complete setup is wrapped in a docker container, increasing portability and allowing parallel execution of video streaming sessions. Experiments are carried out on imec's Virtual Wall[8], with at most six docker containers running simultaneously on a hexacore Intel(R) Xeon(R) CPU E5645 @ 2.40GHz with 24 GB of RAM.

4.4.2 Evaluation Metrics

A number of evaluation metrics are considered in this paper. The viewport prediction accuracy, on the one hand, is evaluated by considering the great-circle distance between the predicted coordinates and the coordinates actually visited by the user. The video quality, on the other hand, is evaluated

[6]http://mininet.org
[7]https://www.eclipse.org/jetty/
[8]https://doc.ilabt.imec.be/ilabt-documentation/

Table 4.1: Obtained bit rates [Mb/s] for the Sandwich, Spotlight and Surf videos. Sandwich and Spotlight are assigned an 8×6 and a 16×12 tiling scheme, while Surf is assigned an 8×8 and a 16×16 tiling scheme.

Video	CRF	1×1	2×2	4×2	4×4	8×4	8×6/8	16×12/16
Sandwich	15	21.8 ± 6.5	21.8 ± 6.5	21.9 ± 6.5	21.9 ± 6.6	22.1 ± 6.6	22.2 ± 6.6	23.4 ± 6.6
	20	10.1 ± 3.2	10.2 ± 3.2	10.2 ± 3.2	10.3 ± 3.2	10.4 ± 3.2	10.5 ± 3.2	11.6 ± 3.2
	25	4.4 ± 1.4	4.4 ± 1.4	4.5 ± 1.4	4.5 ± 1.4	4.7 ± 1.4	4.8 ± 1.4	5.9 ± 1.5
	30	2.1 ± 0.7	2.1 ± 0.7	2.1 ± 0.7	2.2 ± 0.7	2.3 ± 0.7	2.4 ± 0.7	3.4 ± 0.7
	35	1.1 ± 0.3	1.1 ± 0.3	1.1 ± 0.3	1.2 ± 0.3	1.3 ± 0.4	1.4 ± 0.4	2.4 ± 0.4
Spotlight	15	20.6 ± 13.9	20.7 ± 13.9	20.7 ± 13.9	20.8 ± 13.9	21.0 ± 14.0	21.2 ± 14.0	22.6 ± 14.2
	20	10.4 ± 8.9	10.5 ± 8.9	10.5 ± 8.9	10.6 ± 8.9	10.8 ± 9.0	10.9 ± 9.0	12.2 ± 9.1
	25	5.1 ± 5.1	5.1 ± 5.1	5.1 ± 5.1	5.2 ± 5.1	5.4 ± 5.1	5.5 ± 5.2	6.7 ± 5.2
	30	2.5 ± 2.7	2.5 ± 2.7	2.6 ± 2.7	2.6 ± 2.7	2.8 ± 2.7	2.9 ± 2.7	4.0 ± 2.8
	35	1.3 ± 1.3	1.3 ± 1.3	1.3 ± 1.3	1.4 ± 1.3	1.5 ± 1.4	1.6 ± 1.4	2.7 ± 1.4
Surf	15	26.2 ± 12.6	26.2 ± 12.6	26.3 ± 12.7	26.4 ± 12.7	26.6 ± 12.7	27.0 ± 12.8	29.0 ± 13.0
	20	16.5 ± 8.6	16.5 ± 8.6	16.6 ± 8.7	16.7 ± 8.7	16.9 ± 8.7	17.2 ± 8.7	18.9 ± 8.9
	25	9.4 ± 5.4	9.5 ± 5.4	9.5 ± 5.4	9.6 ± 5.4	9.7 ± 5.4	10.0 ± 5.5	11.6 ± 5.6
	30	4.6 ± 2.8	4.7 ± 2.8	4.7 ± 2.8	4.8 ± 2.8	4.9 ± 2.8	5.2 ± 2.9	6.6 ± 2.9
	35	2.3 ± 1.4	2.3 ± 1.4	2.3 ± 1.4	2.4 ± 1.4	2.5 ± 1.4	2.8 ± 1.4	4.2 ± 1.5

Table 4.2: Overview of parameter configurations.

Parameter	Configurations
Tiling scheme	1×1, 2×2, 4×2, 4×4, 8×4, 8×6/8, 16×12/16
Bandwidth	[2; 24] Mb/s, 4G traces
Latency	[0; 100] ms
Viewport prediction	Current coordinates, spherical walk, perfect prediction
Rate adaptation heuristic	UVP $vp = 110°$, UVP $vp = 360°$, CTF, Petrangeli, Hosseini
Reordering	No reassignment, reassignment (47 Hz)
HTTP version	HTTP/1.1, HTTP/2
Persistent connections	1, min(#tiles, 6)

using three metrics. First, the average quality level of the tile corresponding to the viewport center, between 1 (lowest quality) and 5 (highest quality). Second, the relative time spent on the highest quality layer for this tile. While these two metrics are often used in related work, evaluating the quality of the viewport center is not enough: the user can also move her eyes within the viewport, resulting in a different perceived video quality. For this reason, we also use the weighted quality metric proposed in previous work, with a total of $n_1 \cdot n_2 = 50 \cdot 50 = 2500$ samples drawn according to the distribution of the user's gaze [35]. Initial evaluations are performed in a network with fixed bandwidth, but a final evaluation will focus on highly variable, 4G/LTE bandwidth scenarios; for the latter, we also mention the relative freeze time (i.e., the total freeze time divided by the video length) induced by buffer starvation.

4.4.3 Evaluation Space

Given the number of videos, quality representations, tiling schemes and optimizations proposed in Section 4.3, the evaluation space is significant. Table 4.2 presents an overview of all considered parameter configurations, which can be applied to each of the three videos and each of the 48 users. For reasons of time complexity - each combination of parameters requires at least 164 seconds for a single run - the most relevant combinations have been selected for evaluation. Below, we first discuss simulated results for viewport prediction. Then, emulated results regarding rate adaptation, quality reassignment and application optimization are discussed in detail. It is worth noting that, for comparison reasons, five rate adaptation heuristics are evaluated: UVP with a viewport of 110° (HTC VIVE as used by Wu

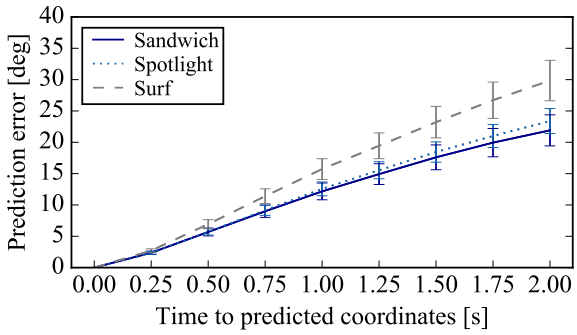


Figure 4.11: Average prediction error for the viewport center among all 48 users, as a function of the time to the predicted coordinates.

et al. [37]), UVP with a viewport of 360° (uniform quality increases, since all tiles fall within the viewport), CTF, the heuristic proposed by Petrangeli et al. [6] and the one by Hosseini and Swaminathan [11].

4.4.4 Viewport Prediction

With regard to the proposed tile reordering and quality refinement scheme, it is worth evaluating the prediction error as a function of the time to the predicted coordinates. For this reason, we first evaluate the performance of the proposed spherical viewport prediction scheme as a function of the time to the predicted coordinates. Using simulation, the results in Figure 4.11 are obtained. As can be observed, the prediction error is, by approximation, linearly dependent on the time to the prediction. Therefore, the longer the client postpones its final decision on the quality of tiles close to the viewport, the better this decision is expected to be. This is exactly what the proposed tile reordering and quality recalculation scheme attempts to do (Section 4.4.6). Below, we first show the impact of the tiling schemes and the proposed rate adaptation heuristics, before showing the impact of this approach.

4.4.5 Tiling and Rate Adaptation

Next, we evaluate the performance of tiling-based solutions and the proposed rate adaptation heuristics. To evaluate the different configurations under distinct network conditions, traffic control is used to fix the available bandwidth between client and server to values between 2 and 24 Mb/s. The former should result in the lowest video quality only, while the latter is expected to provide the user with a reasonably high video quality.

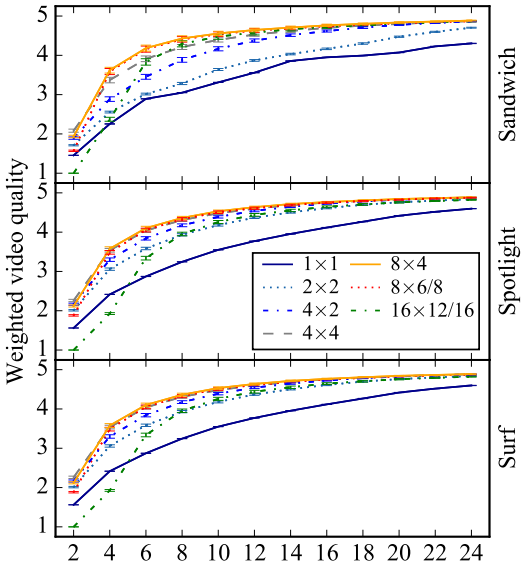


Figure 4.12: Weighted video quality as a function of the available bandwidth, for different tiling schemes and UVP $vp_{deg} = 110°$. The content is requested over HTTP/1.1 with negligible latency.

Figure 4.12 shows the weighted video quality as a function of the available bandwidth, for the three videos and five different tiling schemes, with the UVP $vp = 110°$ heuristic. When no tiling is applied (1×1), the viewport quality is relatively low: valuable bandwidth is wasted to parts of the video which are never consumed by the user. Results improve for the 2×2 and 4×2 tiling schemes, but better results are achieved through the 4×4, 8×4 and 8×6/8 tiling schemes. The 16×12/16 tiling scheme performs significantly worse, especially for lower bandwidths. This can be attributed to the significant encoding overhead on the one hand, and to the large number of GET requests on the other. Overall, results reflect the trade-off between a higher granularity and encoding overhead, with the 4×4, 8×4 and 8×6/8 tiling schemes resulting in the highest viewport quality.

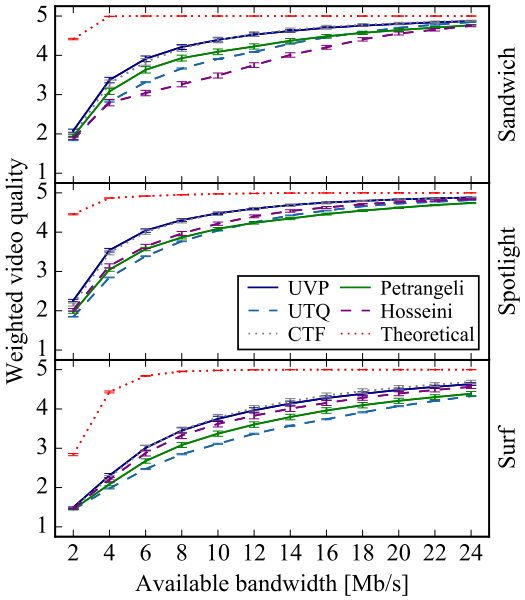
Next, we evaluate the proposed rate adaptation heuristics and compare results both with the state-of-the-art and the theoretical optimum. Two values for the vp parameter in Algorithm 4.1 are used: 110°, referred to as UVP, and 360°, referred to as uniform tile quality (UTQ). Figure 4.13a shows the weighted video quality as a function of the available bandwidth, for the 4×4 tiling scheme. We observe that UVP and CTF, which both take into account the great-circle distance to the viewport center, outperform

the others by a significant margin. First, using UTQ distributes the available bandwidth to the whole video, negating the benefits of tile-based encoding. Second, compared to our approach, the heuristic proposed by Petrangeli et al. is more conservative, since tiles belonging to the same zone are requested at the same quality. Although this approach might reduce the number of GET requests sent by the client - which is especially important in the presence of network latency - when tiles are concatenated, the overall quality is significantly lower. Finally, the heuristic proposed by Hosseini and Swaminathan only takes into account the zone to which each tile belongs, giving the same priority and weight to tiles in the same zone. This approach does not take into account the fact that the surface area of some tiles contribute more to the viewport than others, which explains the lower overall video quality. Results for the system are, however, still far off from the theoretical optimum, which is achieved under perfect viewport prediction and bandwidth estimation, zero delay between client and server and no initial low-quality buffering. In this case, a multiple-choice knapsack problem can be solved with the file size of each tile as the weight, and the gains in terms of video quality as the value [38]. In Section 4.4.7, we will further illustrate the impact of accurate viewport prediction and latency on the resulting video quality.

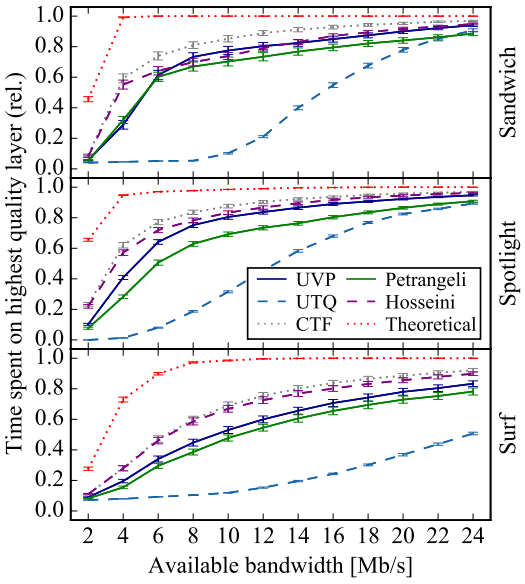
Similar deductions can be made when evaluating the time spent on the highest quality layer, which is reflected in Figure 4.13b. However, since UVP assigns similar quality to tiles within the viewport, and CTF immediately assigns the highest quality to the center tile, the latter leads to better results. As an example, for the Spotlight video with a bandwidth of 8 Mb/s, the relative time spent on the highest quality layer is 83.5% for CTF, while this is 75.2%, 18.7%, 62.9% and 78.3% for UVP, UTQ, Petrangeli and Hosseini, respectively.

4.4.6 Feedback Loop for Quality Reassignment

Having established that the proposed rate adaptation heuristics outperform other approaches, we next evaluate the impact of tile reordering and quality reassignment. To this end, the feedback loop presented in Section 4.3 was implemented in the Python player, with at least 20 ms between succeeding rate adaptation evaluations (same order of magnitude as coordinate sampling, at 47 Hz). Results for the weighted video quality and the time spent on the highest quality layer are presented in Figure 4.14, for three different tiling schemes, both with and without quality reassignment. Evaluating the video quality, minor yet significant differences are observed for the Surf video; as an example, the weighted video quality is increased

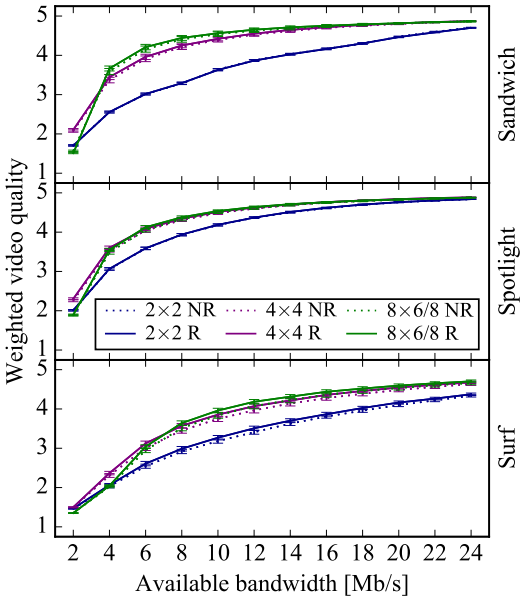


(a) Weighted video quality in the viewport

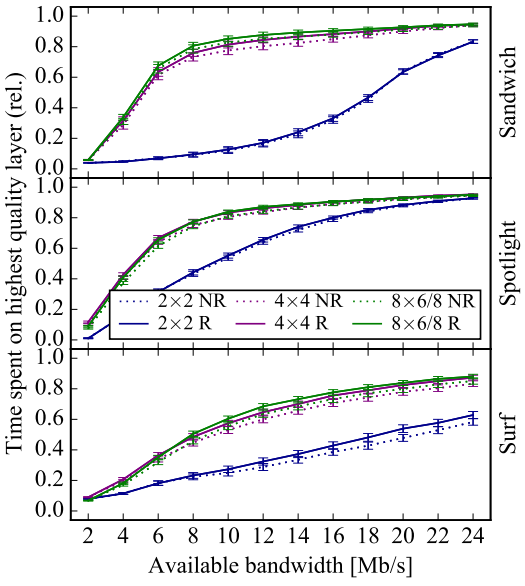


(b) Time spent on the highest quality layer

Figure 4.13: Results as a function of the available bandwidth, for 4×4 tiling. The content is requested over HTTP/1.1 with negligible latency.



(a) Weighted video quality in the viewport



(b) Time spent on the highest quality layer

Figure 4.14: Results for the UVP heuristic as a function of the available bandwidth, both with (R) and without (NR) quality reassignment.

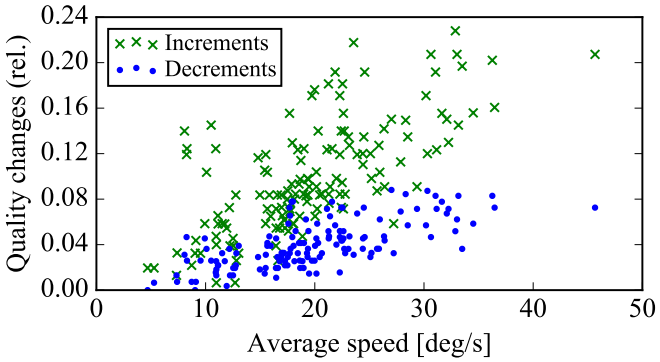


Figure 4.15: The number of segments for which the quality of the central viewport tile is incremented and decremented compared to the first quality assignment, relative to the total number of segments. The UVP heuristic is used with an 8×6/8 tiling scheme for all three videos and all 48 users, streamed at 8 Mb/s.

from 3.97 to 4.07 and from 4.08 to 4.18 at 12 Mb/s, for a 4×4 and an 8×8 tiling scheme respectively. For the other two videos, where the average speed and prediction error are lower (Figure 4.11), differences are less outspoken. Evaluating the time spent on the highest quality layer, however, an increase can be observed for all videos. Considering an 8×6/8 tiling scheme, for instance, the time spent on the highest quality layer increases from 85.1% to 87.8%, from 84.3% to 87.1% and from 63.5% to 68.5% for the Sandwich, Spotlight and Surf video respectively. This is an immediate consequence of postponed quality decisions, based on more accurate viewport predictions.

Evidence of the quality increase relative to the user's speed is presented in Figure 4.15. Here, the number of segments for which the quality of the central viewport tile has changed compared to the first quality assignment are shown, relative to the total number of segments. From these results, two important observations can be made. First, the number of quality changes is linearly dependent on the average speed of movement within the video streaming session. This behavior is expected, since decisions on the quality are only changed by the client when the predicted viewport coordinates change significantly. Second, both beneficial and detrimental changes to the quality are issued: even when predictions are made later in time, it is possible that the original prediction would have resulted in a lower viewport prediction error. The order of magnitude however differs by a factor 2.5, which results in an overall increase of the average video quality.

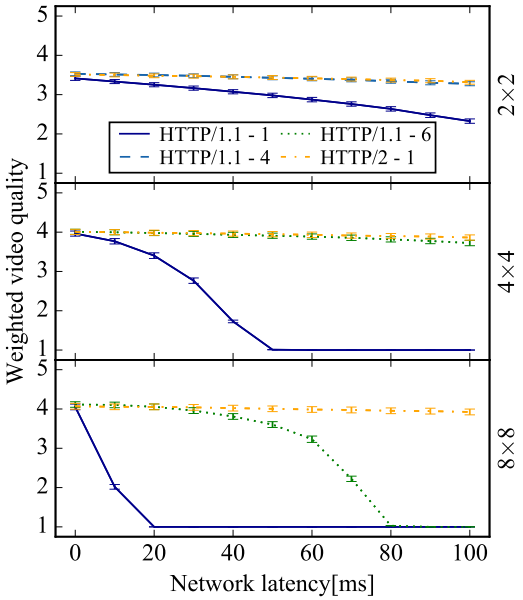
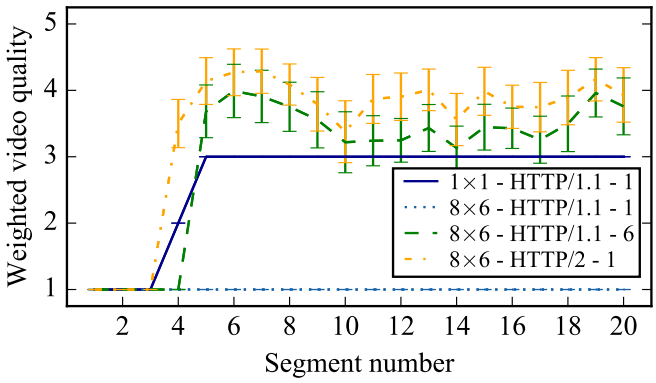


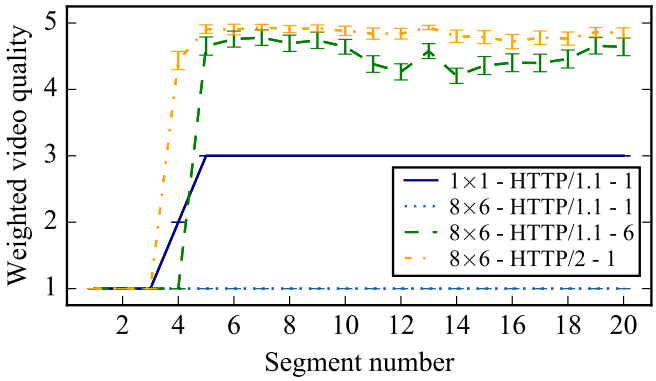
Figure 4.16: Weighted video quality as a function of the network latency, for the Surf video with different tiling schemes and UVP, and a bandwidth of 12 Mb/s.

4.4.7 Application Layer Optimization

Next, both the bandwidth and the latency are changed using traffic control (tc). Figure 4.16 shows the observed video quality as a function of the latency for a bandwidth of 12Mb/s, using UVP and the proposed viewport prediction approach. When a single persistent connection over HTTP/1.1 is used, the quality quickly decreases: this is an immediate consequence of the idle RTT cycles lost to retrieve the tiles for each segment. Indeed, 17 requests are issued for each video segment when a 4×4 tiling scheme is used, resulting in an idle time of more than 500 ms when the network latency is 30 ms. Using multiple parallel persistent connections (one for each tile, with a maximum of six), however, this effect can be partly mitigated. Indeed, when the number of requested resources is limited, the quality is only mildly affected by network latency. For a higher number of tiles, however, the quality quickly degrades as the latency increases. In contrast, HTTP/2's server push allows to deliver segments back-to-back, eliminating the need for multiple GET requests. For this reason, the resulting quality is only slightly impacted by higher network latency.



(a) Proposed viewport prediction



(b) Perfect viewport prediction

Figure 4.17: Weighted video quality over time, for the Sandwich video and UVP $vp = 110°$, with a bandwidth of 8 Mb/s and a network latency of 60 ms. Given the variable bit rate encoding, even with perfect prediction a decreased visual quality can be observed when a scene switch occurs after 15 seconds.

To assess the impact of the applied tiling and application layer optimizations during streaming, we evaluated the performance for all users during playout of the first twenty video segments. Results are shown in Figure 4.17, for the Sandwich video and UVP, with a bandwidth of 8 Mb/s and a network latency of 60 ms (representative for 4G/LTE networks in the USA [4]). When no tiling is applied, the video quality slowly ramps up and stabilizes at the third quality representation. For an 8×6 tiling scheme, the total latency is higher than the segment duration when a sin-

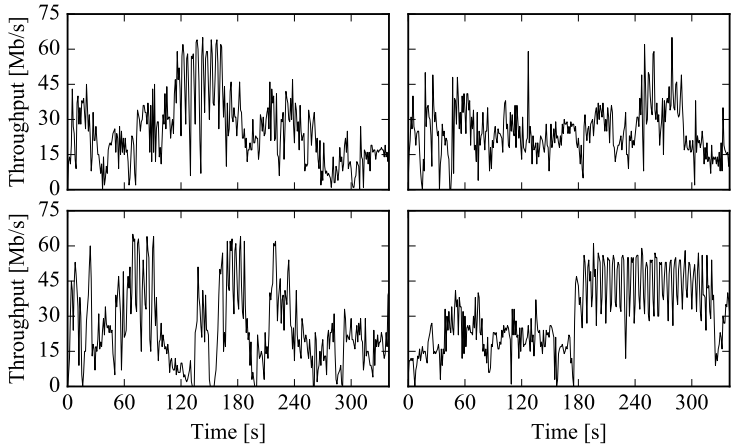


Figure 4.18: Four 4G/LTE bandwidth traces, collected in Ghent, Belgium [39]. Transportation means, from left to right: bus, foot, train and tram.

gle HTTP/1.1 connection is used. This results in regular playout freezes, forcing the client to stay at the lowest quality. Comparing HTTP/1.1 with six persistent connections with HTTP/2, the latter results in a significantly higher video quality. In terms of quality rampup, HTTP/1.1 is outperformed as well; this is because the estimated bandwidth for the former only increases slowly, being more severely impacted by RTT cycles. Overall, we conclude that the use of HTTP/2 has a beneficial impact on the perceived video quality in scenarios with higher network latency.

4.4.8 4G/LTE Scenario

As a final step, we evaluate the performance of the proposed rate adaptation heuristics and the application layer optimizations in a mobile, 4G/LTE network scenario. To this end, we use a set of 4G/LTE throughput traces, collected in Ghent, Belgium between December 2015 and February 2016 (see Appendix A) [39]. Four relevant traces were selected, based on length (at least three minutes) and bandwidth (ranges in the order of the observed video bit rates). Figure 4.18 shows the available bandwidth over time, for each of the traces and corresponding means of transport. In the Mininet setup, tc is used to shape the network throughput between client and server. Based on a recent OpenSignal measurement study in Belgium, the network latency is set to 37 ms [5].

Table 4.3 shows results for a selection of most relevant configurations: no tiling and $8 \times 6/8$ tiling, using both HTTP/1.1 and HTTP/2, with or

Table 4.3: Emulation results for the CTF heuristic over 4G/LTE networks (latency 37 ms), in terms of (sampled) quality in the viewport Q_{vp} (1-5), the quality of the center tile Q_c (1-5), the time spent on the highest layer T_c and the freeze time relative to the video duration FT. Different configurations consider the tiling scheme T, the HTTP version H, the number of persistent connections C and quality recalculation Q.

T	H	C	Q	Q_{vp}	Q_c	T_c	FT
1×1	1.1	1	0	4.11	4.11	0.423	0.016
1×1	2	1	0	4.05	4.05	0.387	0.015
8×6/8	1.1	1	0	1.00	1.00	0.000	0.915
8×6/8	1.1	6	0	4.53	4.57	0.884	**0.014**
8×6/8	1.1	6	1	4.56	4.61	0.896	0.015
8×6/8	2	1	0	**4.61**	**4.64**	**0.904**	0.017

Sandwich

T	H	C	Q	Q_{vp}	Q_c	T_c	FT
1×1	1.1	1	0	4.35	4.35	0.610	0.014
1×1	2	1	0	4.31	4.31	0.585	**0.012**
8×6/8	1.1	1	0	1.00	1.00	0.000	0.939
8×6/8	1.1	6	0	4.62	4.67	0.912	0.014
8×6/8	1.1	6	1	4.65	**4.69**	0.917	0.014
8×6/8	2	1	0	**4.66**	**4.69**	**0.921**	0.015

Spotlight

T	H	C	Q	Q_{vp}	Q_c	T_c	FT
1×1	1.1	1	0	3.78	3.78	0.359	0.014
1×1	2	1	0	3.72	3.72	0.330	**0.012**
8×6/8	1.1	1	0	1.00	1.00	0.000	1.561
8×6/8	1.1	6	0	4.16	4.24	0.802	0.015
8×6/8	1.1	6	1	4.25	4.33	0.826	0.015
8×6/8	2	1	0	**4.36**	**4.42**	**0.850**	0.017

Surf

without quality reassignment in case of the former. When no tiling is used, the weighted video quality over all users ranges between 3.72 (Surf) and 4.35 (Spotlight), while the relative time spent on the highest layer ranges from 33.0% to 61.0%. In this case, only one file is downloaded for each temporal video segment, so that quality reassignments cannot be be considered.

When 8×6 or 8×8 tiling is considered, more interesting results are observed. When a single persistent connection is used, the combined net-

work latency (approximately 49 and 65 times 37 ms, respectively) exceeds the segment duration, resulting in the lowest video quality and the occurrence of playout freezes for each video segment. When six persistent connections are employed, however, the impact of network latency can be partly overcome, and the weighted video quality increases to values between 4.16 (Surf) and 4.62 (Spotlight). As far as quality reassignment goes, both the video quality and the time spent on the highest layer are increased when additional feedback between the viewport prediction module and the download queue is introduced.

Finally, the application of HTTP/2 server push allows to further reduce the impact of latency, by proactively delivering the tiles to the client. As expected, this again results in an increase of the video quality and time spent on the highest quality layer. For the Surf video, for instance, the weighted video quality increases from 4.25 to 4.36, while the time spent on the highest quality layer increases from 82.6% to 85.0%. It is worth noting that even larger increases are observed for higher network latency, omitted here due to space constraints.

Overall, the gains of tile-based HAS for VR video are significant: comparing the Surf video with no tiling over HTTP/1.1 with an 8×8 tiling scheme with HTTP/2, the weighted and center video quality increases from 3.78 to 4.36 and 4.42 respectively, while the relative time spent on the highest quality layer increases from 35.9% to 85.0%.

4.4.9 General Conclusions

The above evaluations show that the application of a tile-based approach for VR video streaming allows to significantly increase the video quality, when the right viewport prediction, rate adaptation and application layer optimizations are used. In summary, these are the lessons learned:

1. **Video capture & encoding:** The application of tile-based video encoding allows to increase the video quality of visible regions within the immersive video. However, a trade-off exists between the granularity on the one hand, and an introduced encoding overhead on the other. Upon evaluation, the best results are obtained for the 4×4 and 8×6/8 tiling schemes: using 1×1 or 2×2 tiling results in inefficient bandwidth usage, while 16×12/16 tiling results in an encoding overhead ranging from 1.3 to 2.8 Mb/s (85.3% to 127.4% for the lowest quality representation) and an increased number of GET requests. Comparing tile-based solutions with traditional encoding, the video quality can be increased by more than 50% in certain scenarios;

2. **Viewport prediction:** By modeling movement as a time-constrained walk on a sphere, rather than using a two-dimensional representation on the equirectangular plane, the viewport prediction error can be significantly reduced. Compared to an approach in which the last-known location is used as prediction, reductions are considerably smaller. Since our results indicate that optimal viewport prediction allows for a considerable increase of the video quality (up to 20% in some cases), viewport prediction is a relevant topic for future research;

3. **Tile-based rate adaptation:** The proposed rate adaptation heuristics strongly focus on the user's viewport, using the great-circle distance to the viewport center as the predominant prioritization metric. Evaluation results show that the proposed heuristics outperform state-of-the-art solutions: with gains up to 20%, UVP results in a higher video quality for the viewport as a whole while CTF results in a higher video quality in the center of the viewport in particular;

4. **Quality reassignment:** Using a more intelligent approach to determine the order in which tiles are downloaded, regular updates on the viewport prediction can be used to redefine the quality assigned to tiles which have not yet been retrieved. Therefore, the overall video quality - and the video quality in the center of the viewport in particular - can be increased by 5 to 10% in some cases.

5. **Application layer optimization:** Network latency has a significant impact on the download time and bandwidth utilization for tile-based solutions. Using multiple persistent TCP connections, reduces this impact and results in a higher overall network throughput. When HTTP/2 server push is applied, a single GET request to the server suffices to retrieve all content, further reducing the impact of latency and resulting in a higher video quality, with increases up to 20% when a network latency of 60 ms is considered.

4.5 Conclusions and Future Work

In this chapter, we have identified important challenges for tile-based 360° video streaming. Using an existing dataset of viewport traces, where the movement of 48 users is tracked throughout immersive video sessions, we have shown that the proposed content-agnostic viewport prediction approach is able to reduce the prediction error compared to other solutions.

Through emulation, we have proven that our novel tile-based rate adaptation heuristics outperform other heuristics in terms of video quality and time spent per quality layer. We have also presented a feedback loop in the quality decision process whilst downloading content, which allows a further increase of the observed video quality. Finally, we have shown that the application of HTTP/2 server push allows to counter negative effects introduced by network latency, especially given the many GET requests required by tile-based solutions. Evaluating emulation results for a 4G/LTE scenario, the time spent on the highest quality layer can be more than doubled compared to non-tiled solutions.

In future work, we will perform a more extensive evaluation of the proposed optimizations under highly variable network conditions. We will also extend the proposed rate adaptation heuristics to take into account probabilistic information on the viewport position, rather than a single predicted value. Finally, we plan to investigate the applicability of SHVC in combination with updated viewport predictions, which could further improve the video quality perceived by the user.

References

[1] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia. *A Survey on Quality of Experience of HTTP Adaptive Streaming*. IEEE Communications Surveys Tutorials, 17(1):469–492, 2015.

[2] R. Mok, E. Chan, and R. Chang. *Measuring the Quality of Experience of HTTP Video Streaming*. In Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, pages 485–492, 2011.

[3] *Information technology - Dynamic Adaptive Streaming over HTTP (DASH) - Part 1: Media Presentation Description and Segment Formats*. Technical report, International Organization for Standardization, 2014.

[4] OpenSignal. *State of Mobile Networks: USA (January 2018)*, 2018. Available from: https://opensignal.com/reports/2018/01/usa/state-of-the-mobile-network/.

[5] OpenSignal. *State of Mobile Networks: Belgium (March 2018)*, 2018. Available from: https://opensignal.com/reports/2018/03/belgium/state-of-the-mobile-network/.

[6] S. Petrangeli, V. Swaminathan, M. Hosseini, and F. De Turck. *An HTTP/2-Based Adaptive Streaming Framework for 360-Degree Virtual Re-*

ality Videos. In Proceedings of the ACM Multimedia Conference, pages 306–314, 2017.

[7] J. van der Hooft, M. Torres Vega, S. Petrangeli, T. Wauters, and F. De Turck. *Optimizing Adaptive Tile-Based Virtual Reality Video Streaming*. In Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, 2019. Accepted for publication.

[8] M. Yu, H. Lakshman, and B. Girod. *A Framework to Evaluate Omnidirectional Video Coding Schemes*. In Proceedings of the IEEE International Symposium on Mixed and Augmented Reality, pages 31–36, 2015.

[9] R. Skupin, Y. Sanchez, D. Podborski, C. Hellge, and T. Schierl. *HEVC Tile-Based Streaming to Head-Mounted Displays*. In Proceedings of the IEEE Annual Consumer Communications Networking Conference, pages 613–615, 2017.

[10] M. Budagavi, J. Furton, G. Jin, A. Saxena, J. Wilkinson, and A. Dickerson. *360 Degrees Video Coding Using Region Adaptive Smoothing*. In Proceedings of the IEEE International Conference on Image Processing, pages 750–754, 2015.

[11] M. Hosseini and V. Swaminathan. *Adaptive 360 VR Video Streaming: Divide and Conquer!* In Proceedings of the IEEE International Symposium on Multimedia, pages 107–110, 2016.

[12] T. Pixvana. *An Intro to FOVAS: Field of View Adaptive Streaming for Virtual Reality*, 2016. Available from: https://pixvana.com/intro-to-field-of-view-adaptive-streaming-for-vr/.

[13] E. Kuzyakov and D. Pio. *Next-Generation Video Encoding Techniques for 360 Video and VR*, 2016. Available from: https://code.fb.com/virtual-reality/next-generation-video-encoding-techniques-for-360-video-and-vr/.

[14] A. Zare, A. Aminlou, and M. M. Hunnuksela. *Virtual Reality Content Streaming: Viewport-Dependent Projection and Tile-Based Techniques*. In Proceedings of the IEEE International Conference on Image Processing, pages 1432–1436, 2017.

[15] *Temporal MCTS Coding Constraints Implementation*. Technical report, Joint Collaborative Team on Video Coding, 2017.

[16] J. Son, D. Jang, and E. Ryu. *Implementing Motion-Constrained Tile and Viewport Extraction for VR Streaming*. In Proceedings of the 28th ACM

SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video, pages 61–66. ACM, 2018.

[17] A. Taghavi Nasrabadi, A. Mahzari, J. D. Beshay, and R. Prakash. *Adaptive 360-Degree Video Streaming Using Layered Video Coding*. In Proceedings of the IEEE Virtual Reality Conference, pages 347–348, 2017.

[18] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan. *Optimizing 360 Video Delivery over Cellular Networks*. In Proceedings of the Workshop on All Things Cellular: Operations, Applications and Challenges, pages 1–6, 2016.

[19] Z. Xu, X. Zhang, K. Zhang, and Z. Guo. *Probabilistic Viewport Adaptive Streaming for 360-Degree Videos*. In Proceedings of the IEEE International Symposium on Circuits and Systems, pages 1–5, 2018.

[20] C. Fan, J. Lee, W. Lo, C. Huang, K. Chen, and C. Hsu. *Fixation Prediction for 360-Degree; Video Streaming in Head-Mounted Virtual Reality*. In Proceedings of the Workshop on Network and Operating Systems Support for Digital Audio and Video, pages 67–72, 2017.

[21] V. Sitzmann, A. Serrano, A. Pavel, M. Agrawala, D. Gutierrez, B. Masia, and G. Wetzstein. *Saliency in VR: How Do People Explore Virtual Environments?* IEEE Transactions on Visualization and Computer Graphics, 24(4):1633–1642, 2018.

[22] H. Hu, Y. Lin, M. Liu, H. Cheng, Y. Chang, and M. Sun. *Deep 360 Pilot: Learning a Deep Agent for Piloting through 360-Degree Sports Video*. Computing Research Repository, abs/1705.01759, 2017.

[23] A. Ghosh, V. Aggarwal, and F. Qian. *A Rate Adaptation Algorithm for Tile-Vased 360-Degree Video Streaming*. Computing Research Repository, abs/1704.08215, 2017.

[24] A. Zare, A. Aminlou, M. M. Hannuksela, and M. Gabbouj. *HEVC-Compliant Tile-Based Streaming of Panoramic Video for Virtual Reality Applications*. In Proceedings of the ACM International Conference on Multimedia, pages 601–605. ACM, 2016.

[25] J. Le Feuvre and C. Concolato. *Tiled-based Adaptive Streaming Using MPEG-DASH*. In Proceedings of the International Conference on Multimedia Systems, pages 41:1–41:3, 2016.

[26] L. D'Acunto, J. van den Berg, E. Thomas, and O. Niamut. *Using MPEG DASH SRD for Zoomable and Navigable Video*. In Proceedings of the International Conference on Multimedia Systems, pages 34:1–34:4, 2016.

[27] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230, RFC Editor, 2014. Available from: https://www.rfc-editor.org/rfc/rfc7230.txt.

[28] M. Belshe, G. Peon, and M. Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540, RFC Editor, 2015. Available from: https://www.rfc-editor.org/rfc/rfc7540.txt.

[29] W3Techs. *Usage of HTTP/2 for Websites*, 2019. Available from: https://w3techs.com/technologies/details/ce-http2/all/all.

[30] S. Wei and V. Swaminathan. *Low Latency Live Video Streaming over HTTP 2.0*. In Proceedings of the Network and Operating System Support on Digital Audio and Video Workshop, pages 37:37–37:42, 2014.

[31] L. Xie, Z. Xu, Y. Ban, X. Zhang, and Z. Guo. *360ProbDASH: Improving QoE of 360 Video Streaming Using Tile-Based HTTP Adaptive Streaming*. In Proceedings of the ACM Multimedia Systems Conference, pages 315–323, 2017.

[32] H. Ahmadi, O. Eltobgy, and M. Hefeeda. *Adaptive Multicast Streaming of Virtual Reality Content to Mobile Users*. In Proceedings of the Thematic Workshops of ACM Multimedia, pages 170–178, 2017.

[33] R. I. T. da Costa Filho, M. C. Luizelli, M. Torres Vega, J. van der Hooft, S. Petrangeli, T. Wauters, F. De Turck, and L. P. Gaspary. *Predicting the Performance of Virtual Reality Video Streaming in Mobile Networks*. In Proceedings of the ACM Multimedia Systems Conference, pages 270–283, 2018.

[34] Y. Rai, P. Le Callet, and P. Guillotel. *Which Saliency Weighting for Omni Directional Image Quality Assessment?* In Proceedings of the International Conference on Quality of Multimedia Experience, pages 1–6, 2017.

[35] J. van der Hooft, M. Torres Vega, S. Petrangeli, T. Wauters, and F. De Turck. *Quality Assessment for Adaptive Virtual Reality Video Streaming: A Probabilistic Approach on the User's Gaze*. In Proceedings of the International Workshop on Quality of Experience Management, 2019. Accepted for publication.

[36] P. Juluri, V. Tamarapalli, and D. Medhi. *SARA: Segment-Aware Rate Adaptation Algorithm for Dynamic Adaptive Streaming over HTTP*. In Proceedings of the IEEE International Conference on Communication Workshop, pages 1765–1770, 2015.

[37] C. Wu, Z. Tan, Z. Wang, and S. Yang. *A Dataset for Exploring User Behaviors in VR Spherical Video Streaming*. In Proceedings of the ACM Multimedia Systems Conference, pages 193–198, 2017.

[38] H. Kellerer, U. Pferschy, and D. Pisinger. *The Multiple-Choice Knapsack Problem*, pages 317–347. Springer Berlin Heidelberg, 2004.

[39] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. Rondao Alface, T. Bostoen, and F. De Turck. *HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks*. IEEE Communications Letters, 20(11):2177–2180, 2016.

Addendum

The publication above does not define the characteristics of the videos used to evaluate proposed framework. To illustrate the considered types of content, Figure 4.19 shows a screenshot of each of the three videos.

(a) Sandwich

(b) Spotlight

(c) Surf

Figure 4.19: Screenshots of the considered videos.

5

Towards 6DoF HTTP Adaptive Streaming Through Point Cloud Compression

"Help me, Obi-Wan Kenobi. You're my only hope."

–Star Wars: Episode IV - A New Hope, 1977

While 360° video solutions, discussed in Chapter 4, allow the user to freely move her head, her location is fixed by the camera's position within the scene. Recent solutions attempt to realize 6DoF by capturing objects through a number of cameras positioned in different angles, and creating a point cloud which consists of the location and RGB color of a significant number of points in the three-dimensional space. In this chapter, we propose PCC-DASH, a standards-compliant means for HTTP adaptive streaming of scenes comprising multiple, dynamic point cloud objects. We present rate adaptation heuristics which use information on the user's position and focus, the available bandwidth, and the client's buffer status to decide upon the most appropriate quality representation of each object. Through an extensive evaluation, we discuss the advantages and drawbacks of each solution. We argue that the optimal solution depends on the considered scene and camera path, which opens interesting possibilities for future work.

⋆⋆⋆

J. van der Hooft, T. Wauters, F. De Turck, C. Timmerer, and H. Hellwagner

Submitted to ACM Multimedia, 2019

Abstract The increasing popularity of head-mounted devices and 360° video cameras allows content providers to offer virtual reality video streaming over the Internet, using a relevant representation of the immersive content combined with traditional streaming techniques. While this approach allows the user to freely move her head, her location is fixed by the camera's position within the scene. Recently, an increased interest has been shown for free movement within immersive scenes, referred to as six degrees of freedom. One way to realize this is by capturing objects through a number of cameras positioned in different angles, and creating a point cloud which consists of the location and RGB color of a significant number of points in the three-dimensional space. Although the concept of point clouds has been around for over two decades, it recently received increased attention by ISO/IEC MPEG, issuing a call for proposals for point cloud compression. As a result, dynamic point cloud objects can now be compressed to bit rates in the order of 3 to 55 Mb/s, allowing feasible delivery over today's mobile networks. In this chapter, we propose `PCC-DASH`, a standards-compliant means for HTTP adaptive streaming of scenes comprising multiple, dynamic point cloud objects. We present a number of rate adaptation heuristics which use information on the user's position and focus, the available bandwidth, and the client's buffer status to decide upon the most appropriate quality representation of each object. Through an extensive evaluation, we discuss the advantages and drawbacks of each solution. We argue that the optimal solution depends on the considered scene and camera path, which opens interesting possibilities for future work.

5.1 Introduction

In recent years, delivery of immersive video has become more prominent than ever. Recent technological advancements have resulted in affordable head-mounted displays, allowing a broad range of users to enjoy virtual reality (VR) content. Service providers such as Facebook[1] and YouTube[2] were among the first to provide 360° video, using the principle of HTTP adaptive streaming (HAS) to deliver the content to the user. In HAS, the content is encoded using several quality representations, temporally seg-

[1] https://www.facebook.com
[2] https://www.youtube.com

mented and stored on one or multiple servers within a content delivery network. Based on the network conditions, the device characteristics, and the user's preferences, the client can then decide on the quality of each of these segments [2]. Having the ability to adapt the video quality, this approach actively avoids buffer starvation, and therefore results in smoother playback of the requested content and a higher Quality of Experience (QoE) for the end user [3]. Protocols and interfaces for HAS are defined by the Dynamic Adaptive Streaming over HTTP (DASH) standard [4, 5].

The introduction of 360° video provides the user with three-dimensional freedom to move within an immersive world, allowing changes in the yaw, roll, and pitch. However, the location of the user remains fixed to the position of the camera within the scene. Recently, a number of efforts have been made to realize six degrees of freedom (6DoF). Two types of solutions are generally considered: *(i)* image-based and *(ii)* volumetric media-based solutions. The former requires a representation of images at every different angle and tilt (e.g., light field video [6]). This results in large storage and bandwidth requirements, since roughly every 0.3° difference in angle requires a new image in order to provide a smooth transition between images. Volumetric media-based solutions, however, store objects as a collection of points. Capturing the geometry (x, y, z tuples) and color (RGB values) of thousands of points, the object can be rendered from any viewing angle [7]. This reduces storage and bandwidth costs, but requires complex preprocessing (i.e., multiple camera angles and depths) and rendering at the client-side. An example point cloud object is shown in Figure 5.1.

With magnitudes of Gigabytes per second, bandwidth requirements for uncompressed point cloud objects are significant. For this reason, a large number of compression techniques have recently been proposed. While prior studies generally focused on compression of static scenes through kd-tree- and octree-based solutions (e.g., [8, 9]), newer efforts have shifted focus to static, dynamic, and dynamically acquired scenarios [7]. The reference video point cloud encoder, which was recently introduced after a call for proposals (CfP) was issued by ISO/IEC MPEG [1], achieves lossy compression factors of 1:100 to 1:500. These compression rates bring the video bit rate for a single point cloud object down to values between 3 and 55 Mb/s, well in the range of the throughput achievable by today's mobile 4G/LTE networks [10].

While the communication demand of compressed point cloud objects meets today's network capacity, the capacity of contemporary devices is not sufficient to decode these compressed objects in real-time. Indeed, preliminary tests on the dynamic objects in the MPEG CfP on a hexacore In-

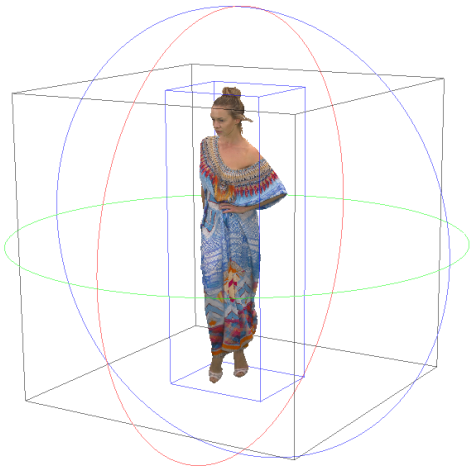


Figure 5.1: Dynamic point cloud object in MPEG's dataset [1], showing the bounding box, the bounding cube, and the yaw, pitch and roll circles corresponding to the user's movement.

tel(R) Xeon(R) CPU E5645 @ 2.40 GHz with 24 GB of RAM showed that the content can only be decoded at 0.1 FPS. However, given the recent technological advancements, it can well be expected that future devices and V-PCC techniques will be able to respect real-time constraints.

In the past, initial work has been done on adaptive streaming for single point cloud objects [11]. However, no work has been done on the evaluation of over-the-top rate adaptation for scenes consisting of multiple point cloud objects. These complex scenes require careful consideration, taking into account, among others, *(i)* the user's location, *(ii)* the user's focus, *(iii)* the position and viewing angle of each object, *(iv)* the available representations, *(v)* the perceived bandwidth and *(vi)* the status of the client's buffer. In this chapter, we propose and evaluate a number of rate adaptation heuristics, collectively referred to as `PCC-DASH`, as one of the first steps towards 6DoF HTTP adaptive streaming.

The contributions of this chapter are twofold. *First*, we propose a means to generate scenes consisting of multiple point cloud objects, and stream these in a standards-compliant way. This guarantees that our solution follows DASH recommendations and can eventually be adopted by existing players such as the *dash.js* reference player [12]. *Second*, we propose a number of rate adaptation heuristics for streaming multi-object scenes, using information on the user's location and focus, the available bandwidth and the buffer status. Through emulation, we show that the selected rate adaptation heuristic has a significant impact on the observed video quality.

The remainder of this chapter is structured as follows. Related work is discussed in Section 5.2. The proposed approach for streaming of multi-object scenes is presented in Section 5.3, focusing on DASH compliance and relevant rate adaptation. The experimental setup and evaluation results are discussed in Section 5.4. Finally, conclusions are drawn and future work is discussed in Section 5.5.

5.2 Related Work

5.2.1 Point Cloud Compression

Image-based solutions require a representation of images at every different angle and tilt. Feasible compression rates can be obtained through spatial reduction (i.e., reducing the resolution of each image) and angular reduction (i.e., reducing the number of images, mostly affecting users with high movement) [6]. For volumetric media-based solutions, however, other approaches are adopted. Most of these focus on static kd-tree- and octree-based solutions. Notable examples of the former include Google's Draco [13] and the work by Devillers and Gandoin [14], while the latter include the work of Schnabel and Klein [9], and Huang et al. [15]. Using a more lightweight approach, Hosseini and Timmerer propose to sample the octree-based representation of the point cloud to limit the amount of data, using three different sampling schemes [11]. Krivokuća et al. propose the adoption of volumetric functions for point cloud compression, using a B-spline wavelet basis to code these functions representing both geometry and attributes [8]. Rather than storing the whole object, an octree of wavelet coefficients of the volumetric functions is generated.

Since MPEG launched its CfP, alternative approaches for dynamic point clouds have been suggested. Following an extensive evaluation of nine submitted proposals, MPEG finally selected a reference encoder for video-based point cloud compression (V-PCC) [7]. This encoder converts point clouds into two separate video sequences, which capture the geometry and texture information, and applies traditional video coding techniques to compress the data. This reference encoder will be used to evaluate the performance of the proposed rate adaptation heuristics in Section 5.3.

5.2.2 6DoF Video Streaming

Regarding image-based solutions, Wijnants et al. propose a DASH-compliant framework for the delivery of light fields [16]. While this approach allows the user to move around and download content in an adaptive way, only static light fields are considered. Furthermore, only single

objects are considered in this chapter. Daniel et al. propose SMFoLD, an open streaming media standard for light field video [17]. This standard allows compliant displays to receive a stream of three-dimensional frame descriptions, and render scenes without the need for specialized head-mounted devices. Kara et al. propose a framework for subjective evaluation of light field scenes, considering different spatial resolutions and angular differences between images [6]. Similar to traditional video, the authors show that quality switching is preferred over long stalling events.

Considering point cloud streaming, Hosseini and Timmerer are the first to propose a DASH-compliant approach for single point cloud streaming [11]. Using point cloud sampling, the authors discuss the merits of a media presentation description (MPD), which contains the required metadata to request the content frame by frame. Using this approach, a number of HTTP GET requests proportional to the frame rate and the number of point cloud objects is required. This can be problematic, especially when network latency is taken into account. In further work, Hosseini presents a rate adaptation heuristic for multiple point cloud objects [18]. This heuristic is however based on an undefined *priority of importance coefficient* for each point cloud object, and its performance has not been evaluated. He et al. consider view-dependent streaming of point cloud objects, using cubic projection to create six two-dimensional images which can then be compressed using traditional compression techniques [19]. The proposed approach relies on a hybrid network (broadband and broadcast) and in-network optimizations such as caching. Park et al. consider a utility-based rate adaptation heuristic for volumetric media [20]. The proposed approach is both throughput- and buffer-aware, and allows for network and user adaptability. The heuristic was evaluated using simulation, reporting considered utility metric values of each object rather than the resulting visual quality. Qian et al. propose Nebula, a volumetric video streaming system for mobile devices [21]. The authors present two rate adaptation mechanisms, adapting the video quality to network conditions between the client and a proxy, and this proxy and the server. However, the proposed heuristics are not evaluated, and no results are reported.

5.3 PCC-DASH Approaches

In this section, we present our `PCC-DASH` approaches. We first discuss how point cloud scenes can be generated from a collection of individual objects in a DASH-compliant way. We then propose a means for rate adaptation, using e.g., the visual area of the point cloud objects within the field of view to differentiate among them.

5.3.1 DASH-Compliant Scene Generation

Similar to traditional video streaming, we propose to use an XML-formatted MPD which contains the required metadata to stream the content. On the highest level, one or multiple periods are defined, each describing a part of the content with a start time and duration. Within a period, adaptation sets are used to describe the different multimedia formats, such as video, audio and subtitles. Each set contains one or multiple representations, each with a unique spatial resolution, bit rate or codec for the same content. The client's rate adaptation heuristic can then choose the most appropriate representation based on the network conditions, the buffer filling, the device characteristics, and the user's preferences.

In this work, we propose to use one adaptation set for each point cloud object. Like in traditional DASH deployments, different representations are generated by means of different quantization values in the point cloud compression phase. For each adaptation set, we propose additional attributes to assign a unique position and rotation within the three-dimensional scene, specifying:

- $xRot$: rotation around the x-axis (rad);
- $yRot$: rotation around the y-axis (rad);
- $zRot$: rotation around the z-axis (rad);
- $xOff$: offset on the x-axis;
- $yOff$: offset on the y-axis;
- $zOff$: offset on the z-axis.

Defining these attributes allows to select and/or position the objects within a scene, and thus to uniquely merge point cloud objects together. An example MPD is presented in Listing 1, which shows two reference point cloud objects available in multiple quality representations. While the first object is left in its original position, the second object is turned 180° and moved on the x-axis, so as to face the first object. The advantage of this generic approach is that an unlimited amount of point cloud objects can efficiently be merged together: point cloud objects can be used in multiple scenes, while only being stored on the server once. Since each object comes with a number of predefined quality representations, the client can differentiate between the different objects, and request the most appropriate representation for each. Furthermore, it is worth noting that the above approach can easily be extended to include static objects and backgrounds, by defining additional adaptation sets. In this chapter, however, we will focus on dynamic objects only.

```
<?xml version='1.0'?>
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011"
 ↪  profiles="urn:mpeg:dash:profile:full:2011" minBufferTime="PT1S">
  <BaseURL>https://www.example.com/</BaseURL>
  <Period duration="PT120S">
    <AdaptationSet id="1" mimeType="application/mpegvpcc" xRot="0"
     ↪  yRot="0" zRot="0" xOff="0" yOff="0" zOff="0">
      <Representation id="1" bandwidth="2400000">
        <SegmentTemplate media="loot/1/segment_$Number%04u$.bin"
         ↪  duration="30" timescale="30" startNumber="1"/>
      </Representation>
      <Representation id="2" bandwidth="3620000">
        <SegmentTemplate media="loot/2/segment_$Number%04u$.bin"
         ↪  duration="30" timescale="30" startNumber="1"/>
      </Representation>
      <!-- further representations -->
    </AdaptationSet>
    <AdaptationSet id="2" mimeType="application/mpegvpcc" xRot="0"
     ↪  yRot="3.1416" zRot="0" xOff="2000" yOff="0" zOff="0">
      <Representation id="1" bandwidth="3500000">
        <SegmentTemplate
         ↪  media="redandblack/1/segment_$Number%04u$.bin"
         ↪  duration="30" timescale="30" startNumber="1"/>
      </Representation>
      <!-- further representations -->
    </AdaptationSet>
    <!-- further adaptation sets -->
  </Period>
  <!-- further periods -->
</MPD>
```

Listing 1: An example DASH-compliant MPD, containing two unique point cloud objects.

5.3.2 Multi Point Cloud Rate Adaptation

The goal of the rate adaptation heuristic is to make a decision on the quality representation for each of the required video components. While traditional video only requires a single decision for each of the temporal video segments – not taking into account other components, such as audio and subtitles – the considered use case requires a decision for each of the point cloud objects within the video scene. Therefore, we propose first to rank the objects according to certain properties, including for instance the visual area of the object within the field of view. Then, the available bandwidth can be allocated to the different objects based on certain heuristics. In

this context, we proposed and formulate three bit rate allocation schemes: *greedy*, *uniform*, and *hybrid*.

An example rate adaptation heuristic is presented in Algorithm 5.1, where the distance between the camera and each object is used to rank the objects. Given the segment index and the perceived bandwidth, measured by dividing the total file size by the total download time for all objects in the last video segment, this heuristic determines the most appropriate quality level for each of the point cloud objects in the next segment. To this end, a *Size*-function is used, which determines the file size corresponding to a given segment, point cloud object, and quality representation, as specified in the MPD. Initially, all objects are assigned the lowest quality 1 (line 1). If the client has just started playing, the quality is not improved in order to fill the buffer as quickly as possible (lines 2-3). When enough segments have been retrieved, the total budget is calculated, based on the available bandwidth and the segment duration (line 4). If the total file size of all objects at the lowest quality exceeds this budget, the lowest quality is selected for each object (lines 5-7). If this condition is not met, the heuristic proceeds with calculating the distance to each of the point cloud objects (line 8). Then, starting with the closest object, the heuristic increases the quality until no more bandwidth is available, or all objects have been assigned the highest quality (lines 9-15).

5.3.2.1 Point Cloud Ranking

Similar to the approach proposed by Hosseini [18], the above rate adaptation heuristic considers the Euclidean distance to each object in order to distinguish between them (lines 8-9). However, this metric does not take into account the true potential impact it has on the user's field of view: a close object in the user's back does not affect the perceived quality, and the size of the objects can differ significantly, making a distant object appear larger than a closer one. To compare results, we consider the following metrics for point cloud ranking:

1. **Distance**: the Euclidean distance between the object and the user's coordinates;
2. A_{vis}: an estimation of the object's visible area. To this end, the bounding box of each object is considered, and the area of the convex hull within the field of view is calculated. The resulting value is 1 if the object covers the whole field of view, while it is 0 if the object is not visible. Note that the metric does not take into account objects blocking one another, and that the estimation of the visible area can be off by a significant margin if the bounding box is sparsely filled.

Algorithm 5.1: Example rate adaptation heuristic. The Euclidean distance between the user and each object is used to rank the objects, after which the available bandwidth is allocated in a greedy way.

Input: s, the segment number, starting at 1
$objects$, the point cloud objects
$bandwidth$, the perceived bandwidth $[b/s]$
$duration$, the segment duration $[s]$
$buffer$, the buffer size $[s]$
n_{qual}, the number of quality representations
x, y, z, the user's coordinates
Output: $qualities$, specifying the selected quality representation

$qualities \leftarrow [1, \forall pc \in objects]$
% Initialize at lowest quality **if** $s \leqslant buffer/duration$ **then**
  return $qualities$
$budget \leftarrow bandwidth \cdot duration$
$n_{bits} \leftarrow \sum_{pc \in objects} Size(s, pc, 1)$
if $n_{bits} \geqslant budget$ **then**
  return $qualities$
$dists \leftarrow [(x - x_{pc})^2 + (y - y_{pc})^2 + (z - z_{pc})^2, \forall pc \in objects]$ **for** $pc \in Sort(objects, dists)$ **do**
  for $q \in [2; n_{qual}]$ **do**
    $cost \leftarrow Size(s, pc, q) - Size(s, pc, q-1)$
    if $n_{bits} + cost > budget$ **then**
      break
    $n_{bits} \leftarrow n_{bits} + cost$
    $qualities[pc] \leftarrow q$
return $qualities$

3. $\boldsymbol{A_{pot}}$: an estimation of the object's potential visible area, if the user would shift her focus to this object without changing position. This metric can be advantageous when the user often moves her head, especially if objects are close by.

4. $\boldsymbol{A_{vis}}$**/ bits**: the visible area of the object, divided by the number of bits required by the highest quality representation. This allows to priori-

tize smaller objects, which require less bandwidth to be streamed at the highest quality.

A combination of metrics is possible as well, considering for instance first the visual and then the potential visual area ($A_{vis-pot}$). Such an approach will be covered in our evaluations in Section 5.5.

5.3.2.2 Bit Rate Allocation

Once the objects have been ranked, the available bandwidth can be allocated to the different objects (lines 9-15). We propose three different schemes:

1. **Greedy bit rate allocation**. As illustrated in Algorithm 5.1, the highest possible quality is given to the highest ranked point cloud object, before moving onto the next. While this approach is useful when a limited number of objects are in scope, it can significantly reduce the perceived video quality when multiple objects are considered by the user.

2. **Uniform bit rate allocation**. Starting with the highest ranked point cloud object, the quality of the different objects is increased one representation at a time. The advantage of this approach is that a similar quality is assigned to all objects, resulting in a smoother field of view. On the downside, a significant amount of bandwidth might be wasted by requesting high-quality objects which are never consumed.

3. **Hybrid bit rate allocation**. In this approach, a distinction is made between objects within and objects outside the field of view. First, the quality of the former is improved uniformly until either the highest quality is assigned, or no more bandwidth remains. Only then is the quality of the latter objects considered, improving the quality representation one by one. This approach combines the advantages of both previous approaches, and should mainly be beneficial when the user targets a specific group of objects.

In the next section, the above approaches will be evaluated for different point cloud scenes and camera paths.

5.4 Evaluation

To evaluate the proposed approaches, MPEG's dataset of dynamic objects [22] was used to create scenes consisting of multiple point cloud ob-

Table 5.1: Observed bit rates [Mb/s] for the four dynamic point cloud objects in the MPEG dataset [22]. The terms "representation", "geometry" and "texture" are abbreviated as "R", "G" and "T", respectively. Raw bit rates [Gb/s] and the number of points per frame are presented as well.

R	GQP	TQP	loot	redandblack	soldier	longdress
1	32	42	2.40 ± 0.01	3.50 ± 0.03	4.48 ± 0.01	5.00 ± 0.03
2	28	37	3.62 ± 0.01	5.03 ± 0.04	7.14 ± 0.01	8.65 ± 0.05
3	24	32	5.81 ± 0.03	7.83 ± 0.05	12.08 ± 0.03	15.86 ± 0.11
4	20	27	10.00 ± 0.07	13.63 ± 0.09	21.95 ± 0.07	30.16 ± 0.21
5	16	22	18.00 ± 0.13	24.72 ± 0.24	40.35 ± 0.14	53.51 ± 0.37
Raw bit rate			4.12 ± 0.10	3.76 ± 0.21	5.72 ± 0.09	4.55 ± 0.20
Points per frame			0.794 M	0.727 M	1.075 M	0.834 M

jects. Below, we first discuss the applied compression and scene generation, the experimental setup, the evaluation metrics and the evaluation space, before presenting the obtained results.

5.4.1 Object Compression and Scene Generation

The four dynamic objects in the MPEG dataset were captured by either 31 or 32 surrounding cameras, at a frame rate of 30 FPS. In our evaluation setup, these objects are used to generate scenes consisting of multiple point cloud objects. To this end, the objects are first compressed using the MPEG reference software[3]. Each object is encoded based on one of five quantization parameter (QP) configurations, resulting in the bit rates presented in Table 5.1. It is worth noting that each of the objects only comes with 300 frames, or ten seconds of video. To obtain a more realistic length for the video streaming sessions, the resulting footage is played out twelve times, alternating between forward and backward movement as to smoothly change the location of the objects. This results in a total video length of 120 seconds, or two minutes.

Using the four dynamic objects, *three different scenes* are created, and a unique camera trace, defining the user's position and focus for each frame, is generated for each of these. In a first scene (*Scene 1*), the objects are placed in a circle, looking outwards. The camera moves around these objects, always focusing on the center of the circle. In a second scene (*Scene 2*), the objects are placed on a single line, all looking straight ahead. The camera moves on a parallel line, focusing on the objects under an angle of 45°. Finally, in a third scene (*Scene 3*), the objects are placed on a 60° arc, looking inward. The camera is initially positioned at the center of the corresponding circle, moving closer to and back farther from a point cloud object,

[3]http://mpegx.int-evry.fr/software/MPEG/PCC/TM/mpeg-pcc-tmc2

Figure 5.2: An example scene and camera path, where the four dynamic objects are placed in a circle (Scene 1).

Figure 5.3: An example field of view corresponding to Scene 1.

before moving left or right towards the next one. As an illustration, Figure 5.2 shows the first scene and camera path, while Figure 5.3 shows an example frame for the field of view.

5.4.2 Experimental Setup

To evaluate the impact of bandwidth and network latency on the resulting video streaming sessions, a network setup is emulated using Mininet[4], where a client is connected to an HTTP/1.1-enabled Jetty server[5] (Figure 5.4). Using traffic control, the available bandwidth can be fixed to certain values, or shaped according to one of three considered 4G/LTE throughput traces (see Appendix A) [23]. An example trace is shown in Figure 5.5. The client is a headless Python-based implementation which provides support for different camera traces, rate adaptation heuristics and quality metrics. The complete setup is wrapped in a Docker container, increasing portability and allowing parallel execution of video streaming

[4]http://mininet.org
[5]https://www.eclipse.org/jetty/

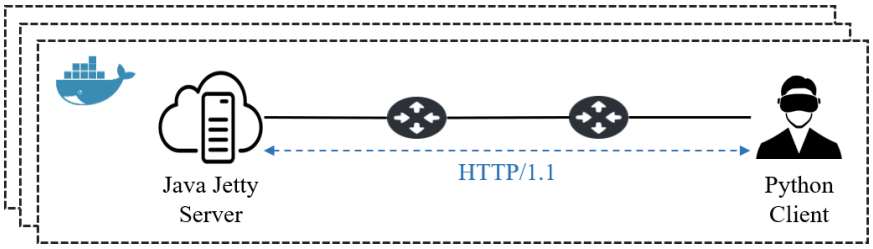


Figure 5.4: Experimental setup, using Mininet to host a virtual network within a Docker container.

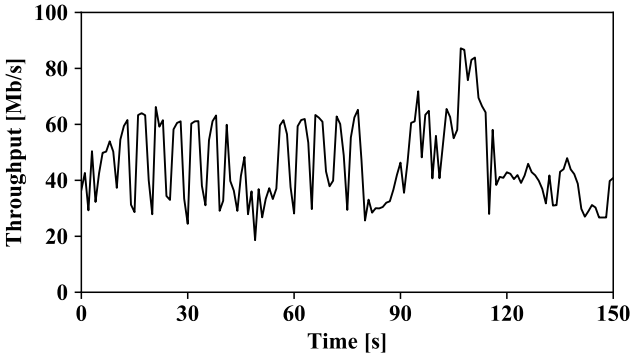


Figure 5.5: An example 4G/LTE bandwidth trace [23].

sessions. Experiments are carried out on a virtualized computing infrastructure with Docker containers running on a hexacore Intel(R) Xeon(R) CPU E5645 @ 2.40GHz with 24 GB of RAM.

Because the current decoder is unable to obtain real-time performance, video streaming is split into an online and an offline phase. During the online phase, the client downloads the encoded point cloud objects, using the information in the MPD. Each time a segment has been downloaded, the client evaluates the current playback time, and determines the camera's position and focus. Based on the available bandwidth and buffer status, the objects in the next segment are requested at the quality chosen by the rate adaptation heuristic. Decisions on the quality representation for each of the point cloud objects are logged, which are used in the offline phase to recreate the scenes using the decoded version of the retrieved objects. To this end, the point cloud renderer by Technicolor[6] is used. This player allows to load a predefined camera path, and record the resulting video in RGB format.

[6]http://mpegx.int-evry.fr/software/MPEG/PCC/mpeg-pcc-renderer

Table 5.2: Overview of parameter configurations.

Parameter	Configurations
Bandwidth	[20; 100] Mb/s, 4G/LTE traces
Latency	0, 37 ms
Scene / camera path	1, 2, 3
Segment duration	1 second
Buffer size	1, 2, 4 seconds
Rate adaptation	distance, A_{vis}, A_{pot}, A_{vis} / bits
Bit rate allocation	uniform, greedy, hybrid
Prediction	most recent, clairvoyant

5.4.3 Evaluation Metrics

In this chapter, we are mainly interested in the impact of the rate adaptation heuristics on the perceived video quality. MPEG's CfP for point cloud compression includes two evaluation metrics for point cloud objects, referred to as point-to-point and point-to-plane distortion metrics [1]. Although useful to assess the quality of point cloud compression, these metrics do not reflect the video quality of the field of view observed by the user. We thus decided to use weighted PSNR on the luminance components, a well-known metric for two-dimensional video [24], to compare the resulting field of view containing the decoded objects, with the same field of view containing the original, raw point cloud objects. To this end, the RGB frames rendered by the player are converted to YUV, and used as input to calculate the weighted PSNR:

$$PSNR_{YUV} = \frac{6 \cdot PSNR_Y + PSNR_U + PSNR_V}{8} \tag{5.1}$$

This allows to take into account the tradeoffs between luma and chroma component fidelity. PSNR values are computed over the whole video, or 3600 frames, allowing us to use Bjøntegaard Delta (BD) for both PSNR and bit rate (BR) [25]: the former reports the expected difference in PSNR for the same bit rate, while the latter reports the increase in bit rate (in percentage) required to achieve the same PSNR. In addition, we also consider the frequency and duration of playout freezes when variable bandwidth patterns are considered.

5.4.4 Evaluation Space

Given the number of scenes and proposed rate adaptation heuristics, the evaluation space is significant. Table 5.2 presents an overview of all considered parameter configurations, which include, among others, the segment

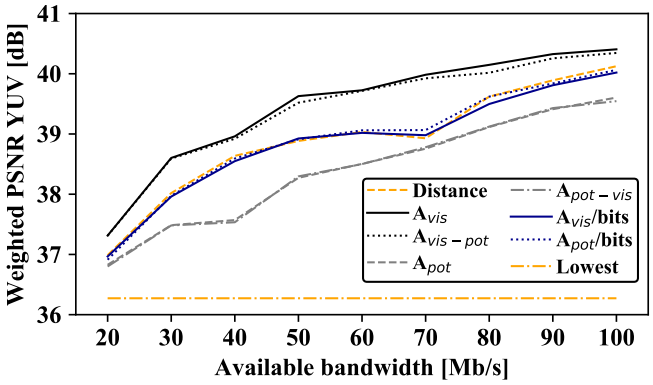


Figure 5.6: Weighted PSNR as a function of the available bandwidth, for scene 3 with different ranking methods, using a buffer size of two seconds, greedy bit rate allocation and most recent prediction.

duration, the buffer size, and the rate adaptation heuristics. It is worth noting that two straightforward approaches for viewport prediction are considered: *(i)* most recent, in which the current user's location and focus (during playout) are used, and *(ii)* clairvoyant, in which perfect prediction is assumed, so that the user's location and focus for the first frame of the segment to request are known. For reasons of time complexity – each combination of parameters requires at least 120 seconds for a single run – the most relevant combinations have been selected for evaluation, resulting in a total of 6750 video streaming sessions. Below, we first discuss results regarding rate adaptation, bit rate allocation, prediction of the user's movement, and the buffer size for constant bandwidth with negligible latency. Then, we discuss results for variable bandwidth scenarios based on collected 4G/LTE traces.

5.4.5 Evaluation Results

First, we investigate the application of different point cloud ranking methods on the resulting weighted PSNR. Figure 5.6 shows results as a function of the available bandwidth, for scene 3 with a buffer size of two seconds, greedy bit rate allocation and the most recent user position. For this particular scene and camera path, considering the visual surface of the different objects results in the best performance. Since the user often zooms in and out on a specific object, allocating the available bandwidth to this object results in the highest visual quality. As indicated by results in Table 5.3, however, this approach is not always recommended: compared to the dis-

Table 5.3: BD-BR for the weighted PSNR for greedy bit rate allocation and different ranking methods, compared to the distance metric. A buffer size of two seconds is considered.

	Scene 1		Scene 2		Scene 3	
Configuration	PSNR	BR	PSNR	BR	PSNR	BR
Distance	**0.00**	**0.0**	0.00	0.0	0.00	0.0
A_{vis}	-0.45	31.2	0.32	-22.3	**0.54**	**-25.5**
$A_{vis-pot}$	-0.48	34.6	**0.34**	**-23.1**	0.50	-24.3
A_{pot}	-0.07	4.2	0.18	-15.0	-0.58	39.0
$A_{pot-vis}$	-0.07	4.0	0.19	-15.5	-0.58	39.8
A_{vis} / bits	-0.06	3.0	-0.06	2.1	-0.04	1.6
A_{pot} / bits	-0.04	2.2	-0.05	0.7	-0.02	-0.1

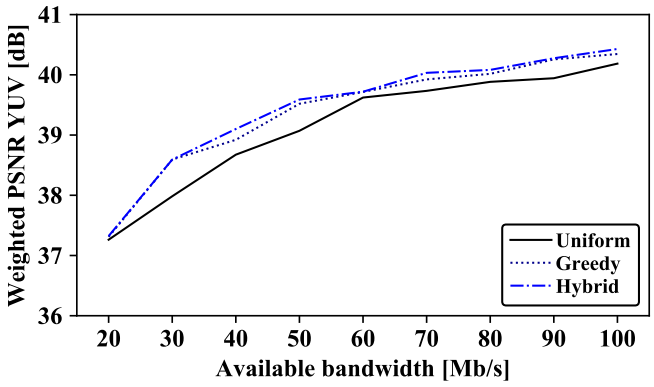


Figure 5.7: Weighted PSNR as a function of the available bandwidth, for scene 3 with different bit rate allocation schemes, using the $A_{vis-pot}$ ranking method with a buffer size of two seconds and most recent prediction.

tance ranking, 31.2% more bandwidth is required in order to achieve a similar visual quality. Indeed, as the camera is turning around the objects, the closest object in the circle is the one which should be given the highest amount of bandwidth. Results further indicate that the potential visual surface is not a good indicator, which could be attributed to the fact that the considered camera paths do not include sudden quick head movements.

Next, Figure 5.7 shows results for the different rate allocation schemes, for scene 3 with a buffer size of two seconds, the $A_{vis-pot}$ ranking method and most recent prediction. For this scene, the hybrid approach outperforms all others: dividing the available bandwidth among all visible objects, the highest PSNR values are observed. As shown in Table 5.4, however, this is not true for all scenes. Indeed, for scenes 1 and 2, uniform bit

Table 5.4: BD-BR for the weighted PSNR for different bit rate allocation schemes compared to uniform allocation, for the $A_{vis-pot}$ ranking method with most recent (MR) and clairvoyant (CV) prediction, for a buffer size of two seconds.

	Scene 1		Scene 2		Scene 3	
Configuration	PSNR	BR	PSNR	BR	PSNR	BR
Uniform - MR	**0.00**	**0.0**	**0.00**	**0.0**	0.00	0.0
Greedy - MR	-0.66	42.9	-0.27	16.9	0.31	-15.8
Hybrid - MR	-0.47	27.5	-0.07	3.1	**0.37**	**-18.9**
Uniform - CV	0.00	0.0	0.00	0.0	0.00	0.0
Greedy - CV	0.08	-5.0	-0.11	6.5	0.64	-28.0
Hybrid - CV	**0.16**	**-8.9**	**0.17**	**-9.6**	**0.66**	**-28.7**

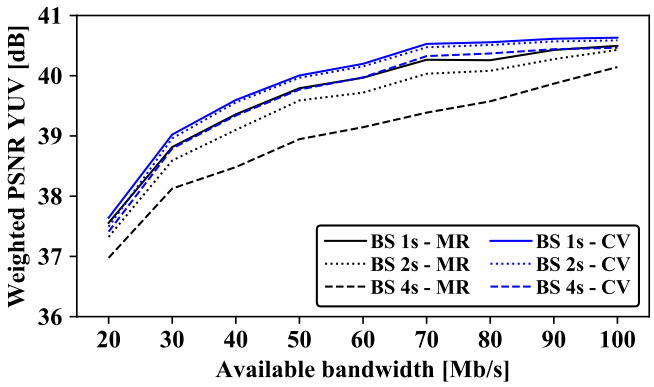


Figure 5.8: Weighted PSNR as a function of the available bandwidth, for scene 3 with different buffer sizes (BS), and most recent (MR) and clairvoyant (CV) prediction. Hybrid bit rate allocation is considered with the $A_{vis-pot}$ ranking method.

rate allocation leads to the best results. This is because the composition of these scenes changes on a time scale lower than the buffer size: as a consequence, the wrong objects are considered in the field of view (e.g., the user might currently be looking at the longdress object in scene 3, but will be focused on the soldier by the time the content at the end of the buffer is played out). Indeed, considering clairvoyant prediction of the user's position and focus, the hybrid approach outperforms all other bit rate allocation schemes. Similar to the 3DoF use case, this illustrates the importance of accurate viewport prediction.

The buffer size also has a significant impact on the video quality. To illustrate this, Figure 5.8 shows results for scene 3, with hybrid bit rate allo-

Table 5.5: BD-BR for the weighted PSNR for different buffer sizes (BS) and most recent (MR) and clairvoyant (CV) location prediction, compared to a buffer size of one second and most recent prediction. Hybrid bit rate allocation is considered with the $A_{vis-pot}$ ranking method.

	Scene 1		Scene 2		Scene 3	
Configuration	PSNR	BR	PSNR	BR	PSNR	BR
BS 1s - MR	0.00	0.0	0.00	0.0	0.00	0.0
BS 2s - MR	-0.40	26.5	-0.15	9.2	-0.22	11.3
BS 4s - MR	-1.08	100.4	-0.46	31.4	-0.73	48.8
BS 1s - CV	**0.48**	**-22.5**	**0.16**	**-7.7**	**0.21**	**-9.5**
BS 2s - CV	0.44	-20.7	0.12	-6.1	0.15	-6.5
BS 4s - CV	0.36	-17.2	0.04	-1.9	-0.02	1.1

cation and different buffer sizes. When most recent prediction is used, relatively high differences are observed; this is a direct consequence of the fact that a larger buffer introduces additional latency between the movement of the user and the decision made by the rate adaptation heuristic. This proposition is confirmed by the results for clairvoyant prediction, which show that differences for the different buffer sizes are significantly smaller. Still, since the buffer is ramped up with low-quality objects at the start of the video streaming session, a lower buffer size results in higher PSNR values. As shown in Table 5.5, these results are confirmed for all scenes. Increasing the buffer size from one to four seconds, a bandwidth increase between 31.4 and 100.4% is required to achieve the same visual quality, while accurate prediction of the user's position and focus allows to reduce the bandwidth by 7.7 to 22.5%.

In a second set of experiments, three 4G/LTE traces are applied to evaluate the impact of the buffer size on the video quality and playout freezes. Based on recent data provided by OpenSignal, the network latency was set to 37 ms [26]. Results for a relevant subset of experiments are reported in Table 5.6, showing the observed PSNR values, the number of freezes and the total freeze time. Similarly as before, increasing the buffer size results in a lower video quality. At the same time, however, the client's robustness against sudden changes in the available bandwidth increases: while a buffer of one second, or one segment in this case, results in many short playout freezes (e.g., 44 freezes with an average duration of 0.22 s for the first scene and bandwidth trace), a negligible amount of freezes are observed for higher buffer sizes. This illustrates the trade-off between accurate prediction on the one hand, and resilience to playout freezes on the other hand.

Table 5.6: Results for different 4G/LTE traces and buffer sizes. Hybrid bit rate allocation is considered with the $A_{vis-pot}$ ranking method. The resulting PSNR is shown, along with the number of observed playout freezes, and the total freeze time (TFT).

		Scene 1			Scene 2			Scene 3		
Trace	Buffer [s]	PSNR	# Freezes	TFT [s]	PSNR	# Freezes	TFT [s]	PSNR	# Freezes	TFT [s]
1	1	42.999	44	9.5	39.790	42	9.4	39.193	46	9.5
1	2	42.367	0	0.0	39.371	0	0.0	38.787	0	0.0
1	4	41.715	0	0.0	39.035	0	0.0	38.232	0	0.0
2	1	42.983	56	12.5	39.683	60	13.0	39.190	60	13.4
2	2	42.196	1	0.1	39.197	1	0.1	38.526	1	0.2
2	4	41.580	0	0.0	38.878	0	0.0	37.943	0	0.0
3	1	42.971	54	10.6	39.791	49	10.6	39.193	50	10.5
3	2	42.389	0	0.0	39.457	0	0.0	38.784	0	0.0
3	4	41.812	0	0.0	39.119	0	0.0	38.207	0	0.0

5.4.6 Lessons Learned

The above evaluations confirm the impact of the considered rate adaptation heuristics, viewport prediction, and buffer size on the observed video quality and playout freezes for point cloud streaming. In summary, these are the lessons learned:

- While some rate adaptation heuristics outperform others by a significant margin, there is no one-size-fits-all scheme: depending on the considered content and user's movement, different point cloud ranking schemes can be used.
- The best results are obtained when the available bandwidth is uniformly distributed among visible objects. This approach however requires accurate prediction of the user's location and focus, in order to correctly identify these objects.
- Increasing the buffer size results in lower interactivity, prediction accuracy, and video quality. However, a larger buffer results in higher resilience to playout freezes, which is an important factor in over-the-top video streaming solutions.
- Although MPEG's reference encoder allows to significantly compress dynamic point clouds, it cannot be run in real-time on contemporary hardware. Further research efforts are needed to enable timely volumetric media compression.

5.5 Conclusions and Future Work

In this chapter, we focus on over-the-top delivery of volumetric media. We present a DASH-compliant framework for point cloud streaming and propose different rate adaptation heuristics for scenes consisting of multiple, dynamic point cloud objects. In our evaluations, we use MPEG's dataset to generate different scenes and camera paths and use the reference encoder for point cloud compression. Results show that the optimal solution depends on the considered scene and camera path and on the accuracy of the predicted user's location and focus. The impact of the buffer size is significant: a buffer of one segment results in higher accuracy and video quality than longer buffers, but is highly susceptible to playout freezes.

In future work, we will focus on subjective evaluation of the proposed `PCC-DASH` approaches. This will help identify the most important factors contributing to the QoE for volumetric media streaming. We also plan to

extend the current rate adaptation heuristics, taking into account the distance, the size, and the probability of point cloud objects being within the field of view. Finally, we will apply HTTP/2's server push to deliver resources back-to-back.

References

[1] MPEG. *MPEG 3DG and Requirements - Call for Proposals for Point Cloud Compression V2*, 2017.

[2] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann. *A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP*. IEEE Communications Surveys Tutorials, 21(1):562–585, 2019.

[3] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia. *A Survey on Quality of Experience of HTTP Adaptive Streaming*. IEEE Communications Surveys Tutorials, 17(1):469–492, 2015.

[4] T. Stockhammer. *Dynamic Adaptive Streaming over HTTP: Standards and Design Principles*. In Proceedings of the 2nd ACM Conference on Multimedia Systems, pages 133–144, New York, 2011. ACM.

[5] I. Sodagar. *The MPEG-DASH Standard for Multimedia Streaming over the Internet*. IEEE MultiMedia, 18(4):62–67, 2011.

[6] P. A. Kara, A. Cserkaszky, M. G. Martini, A. Barsi, L. Bokor, and T. Balogh. *Evaluation of the Concept of Dynamic Adaptive Streaming of Light Field Video*. IEEE Transactions on Broadcasting, 64(2):407–421, 2018.

[7] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, Z. Li, J. Llach, K. Mammou, R. Mekuria, O. Nakagami, E. Siahaan, A. Tabatabai, A. M. Tourapis, and V. Zakharchenko. *Emerging MPEG Standards for Point Cloud Compression*. Journal on Emerging and Selected Topics in Circuits and Systems, 9(1):133–148, 2018.

[8] M. Krivokuća, M. Koroteev and P. A. Chou. *A Volumetric Approach to Point Cloud Compression*, 2018. arXiv:1810.00484.

[9] R. Schnabel and R. Klein. *Octree-Based Point-Cloud Compression*. In Proceedings of the 3rd Eurographics/IEEE VGTC Conference on Point-Based Graphics, pages 111–121, New York, 2006. ACM.

[10] OpenSignal. *State of Mobile Networks: USA (January 2018)*, 2018. Available from: https://opensignal.com/reports/2018/01/usa/state-of-the-mobile-network/.

[11] M. Hosseini and C. Timmerer. *Dynamic Adaptive Point Cloud Streaming*. In Proceedings of the 23rd Packet Video Workshop, pages 1–7, New York, 2018. ACM.

[12] K. Spiteri, R. Sitaraman, and D. Sparacio. *From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player*. In Proceedings of the 9th ACM Multimedia Systems Conference, pages 123–137, New York, 2018. ACM.

[13] Google. *Google Draco*. https://github.com/google/draco, 2016.

[14] O. Devillers and P. Gandoin. *Geometric Compression for Interactive Transmission*. In Proceedings of the 11th IEEE Visualization Conference, pages 319–326, New York, 2000. IEEE.

[15] Y. Huang, J. Peng, C.-C. J. Kuo, and M. Gopi. *Octree-based Progressive Geometry Coding of Point Clouds*. In Proceedings of the 3rd Eurographics/IEEE VGTC Conference on Point-Based Graphics, pages 103–110, Genève, 2006. Eurographics Association.

[16] M. Wijnants, H. Lievens, N. Michiels, J. Put, P. Quax, and W. Lamotte. *Standards-Compliant HTTP Adaptive Streaming of Static Light Fields*. In Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology, pages 4:1–4:12, New York, 2018. ACM.

[17] J. R. Daniel, B. Hernández, C. E. Thomas, S. L. Kelley, P. G. Jones, and C. Chinnock. *Initial Work on Development of an Open Streaming Media Standard for Field of Light Displays (SMFoLD)*. Electronic Imaging, 2018(4):140–1–140–8, 2018.

[18] M. Hosseini. *Adaptive Rate Allocation for View-Aware Point-Cloud Streaming*. Technical report, University of Illinois, 2017.

[19] L. He, W. Zhu, K. Zhang, and Y. Xu. *View-Dependent Streaming of Dynamic Point Cloud over Hybrid Networks*. In Advances in Multimedia Information Processing, pages 50–58, New York, 2018. Springer.

[20] J. Park, P. A. Chou, and J. Hwang. *Rate-Utility Optimized Streaming of Volumetric Media for Augmented Reality*. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 9(1):149–162, 2019.

[21] F. Qian, B. Han, J. Pair, and V. Gopalakrishnan. *Toward Practical Volumetric Video Streaming on Commodity Smartphones*. In Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications, pages 135–140, New York, NY, USA, 2019. ACM.

[22] E. d'Eon, T. Myers, B. Harrison, and P. A. Chou. *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG1M40059/WG1M74006. 8i Voxelized Full Bodies - A Voxelized Point Cloud Dataset*, 2017.

[23] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. Rondao Alface, T. Bostoen, and F. De Turck. *HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks*. IEEE Communications Letters, 20(11):2177–2180, 2016.

[24] J. Ohm, G. J. Sullivan, H. Scharz, T. K. Tan, and T. Wiegand. *Comparison of the Coding Efficiency of Video Coding Standards - Including High Efficiency Video Coding (HEVC)*. IEEE Transactions on Circuits and Systems for Video Technology, 22(12):1669–1684, 2012.

[25] G. Bjøntegaard. *Calculation of Average PSNR Differences Between RD-Curves, ITU-TVCEG-M33*, 2001. Available from: http://wftp3.itu.int/av-arch/video-site/0104_Aus/VCEG-M33.doc.

[26] OpenSignal. *State of Mobile Networks: Belgium (March 2018)*, 2018. Available from: https://www.opensignal.com/reports/2018/03/belgium/state-of-the-mobile-network/.

Addendum

To illustrate the impact of rate adaptation on the observed video quality, two frames of the field of view are shown below. In Figure 5.9, an illustration of scene 2 is considered. In this case, PSNR values between a scene consisting of raw point point clouds objects, and a scene consisting of low quality objects, are relatively high because of the user's position and focus. In In Figure 5.10, where an illustration of scene 3 is considered, PSNR values for the lowest quality are significantly lower.

(a) Raw video

(b) Lowest quality

Figure 5.9: Example field of view for scene 2.

(a) Raw video

(b) Lowest quality

Figure 5.10: Example field of view for scene 3.

6 Conclusions and Future Perspectives

"You mustn't be afraid to dream a little bigger, darling."

–Inception, 2010

In this dissertation, several approaches have been proposed for low-latency delivery of adaptive video streaming services. In this chapter, we review the challenges addressed in this dissertation and outline several research directions arising in the field of multimedia delivery, which we believe will become increasingly more important in the future.

6.1 Review of the Addressed Challenges

Table 6.1 presents the contributions of this dissertation to the considered use cases. Below, we briefly summarize our most important findings.

Challenge #1: Allow low-latency interaction with the provided content, both before and during video streaming.

Focusing on traditional two-dimensional video, the startup delay can be considerably reduced when the segment duration is decreased. Using either a short segment duration or a hybrid segment duration scheme, reductions between 31.2% (end-to-end delivery) and 66.0% (including

Table 6.1: Contributions to the four use cases considered in this dissertation.

	Low-Latency	**Personalization**	**Adaptation**
Traditional video	Startup delay -31.2%, end-to-end delay -4 s		
News-based video	Startup delay -66.0%, hybrid segments	Prefetching strategies	
360° video	Buffer size of 2 s	Viewport prediction error -25.8%	Quality +6.07%
Volumetric media	Buffer size of 2 s	Importance of viewport prediction	PSNR, basis for future work

prefetching) are obtained. Furthermore, more fine-grained temporal segmentation allows us to use a smaller buffer, since resilience to bandwidth variability is increased. This results in a lower end-to-end delay, with possible reductions of four seconds and more. Focusing on immersive video, a small buffer size of two seconds is considered; this is necessary in order to increase reaction times towards user movement. Evaluation results show that this results in a significantly higher video quality, at the cost of a slightly higher probability of running into playout freezes compared to a buffer of e.g., four seconds, in a variable bandwidth scenario.

Challenge #2: Personalize the provided content and services to the targeted end user.

In the context of news-based video portals, we proposed to use different prefetching strategies for news articles with video content. While more than 90% of people were better of when the most popular content was prefetched (i.e., , the content which is generally highlighted by the news provider), a small part of users was better of prefetching either the most recent content, or content which was redeemed of interest by an approach based on natural language processing of the metadata (i.e., , the title, abstract and content of the corresponding news article). Prefetching results in a reduced startup delay, increasing interactivity. In the context of 360° video, we proposed and applied an intuitive content-agnostic viewport prediction scheme, and showed that it outperformed the scheme proposed by Petrangeli et al. Reductions are lower when the suggested scheme is compared to a prediction scheme in which the user is predicted not to move, showing that further optimization is possible. Focusing on immersive video with six degrees of freedom (6DoF), finally, we showed that accurate viewport prediction has a significant impact on the resulting video quality.

Challenge #3: Adapt the quality of the requested content to network characteristics, user movement and the considered video content.

In terms of rate adaptation, we first focused on immersive media with three degrees of freedom, in which the user can move her head and change the yaw, pitch and roll. To avoid wasting bandwidth on unimportant regions, spatial segmentation of the content was considered, allowing us to prioritize tiles within the user's field of view. To do so, we proposed two rate adaptation heuristics which take into account the estimated bandwidth, the user's (predicted) viewport location and information on the considered content, in order to maximize the perceived quality. Comparing the relative time spent on the highest quality layer for a specific configuration, an increase is observed from 78.3 to 83.5% (+6.07%). Combining these heuristics with a feedback loop or HTTP/2 server push, results were even further improved. We later focused 6DoF video, in which point clouds are used to capture and render three-dimensional objects. In this context, we proposed a number of rate adaptation heuristics for scenes consisting of multiple point cloud objects. Comparing these heuristics for different scenes and camera traces, we found that, although some approaches outperform others in terms of PSNR, there is no one-size-fits-all. Related work in this context did not exist, so our results form a basis for future work.

It is worth noting that, throughout this dissertation, objective metrics for the video quality, freeze time and startup delay have been presented. As mentioned in Chapter 1, however, the system should be evaluated in terms of the QoE rather than the QoS. In this regard, subjective experiments could help define the overall impact of different optimizations on the user experience. This is especially true for the immersive video use cases, where the video quality is both temporally and spatially changed to adapt to network throughput and user movement. Care must be taken, though, that the conditions and configurations of these subjective experiments are well-defined, and that conclusions are statistically valid.

6.2 Future Perspectives of Media Delivery

In recent years, technological advancements have made it possible to consider more challenging use cases for video streaming. This is apparent in this dissertation as well, starting from traditional video and moving on to immersive and volumetric media solutions. We believe that this trend will continue to exist in the near future, which means that research in these domains is of the utmost importance. While this work focuses on visually consuming the content only, it is worth noting that many researchers

and developers envision scenarios in which the user can also interact with the environment. Examples include interactive video-based games such as *Control Robots in Chernobyl*, in which radio controlled cars are used to drive and fight in a real-life setting [1], and remote surgery, in which a specialist is expected to perform operations remotely [2]. The latter task is more critical by nature, however, and thus requires strong guarantees in terms of latency, bandwidth and reliability. This is exactly what the so-called Tactile Internet envisions: an Internet network that combines ultra low latency with extremely high availability, reliability and security [3]. Below, we briefly discuss three important research directions which we believe will become increasingly more important in the near future.

Server- and Client-Side Optimizations

From an encoding point of view, further research is required to develop adept encoding for immersive video with six degrees of freedom (6DoF). Based on our evaluations concerning volumetric media, it is safe to say that technology is not there yet: while point cloud objects can be captured, encoded and rendered on the user's display, these steps are cumbersome and cannot be executed in real-time. More advanced – or rather simplified – encoding schemes could provide a step in the right direction. Furthermore, offloading tasks to a more resourceful cloud or fog infrastructure would increase the computational capacity and could help deliver real-time results.

A number of client-side optimizations can be considered as well. First, only limited work has been done on predicting 6DoF user movement, while we have shown that accurate prediction is required to deliver high-quality video quality to the end user. Further research should determine how users interact with complex scenes consisting of multiple light field or point cloud objects. Second, more elaborate rate adaptation heuristics should be developed for this type of use cases, taking into account not only the available bandwidth and the considered object bit rates, but also the user's history and preferences. Finally, it is worth mentioning that actions such as viewport prediction, rate adaptation and synchronization of kinesthetic feedback can become too intensive for a lightweight client device to cope with. In such cases, being able to partially or totally migrate these tasks to a cloud or fog infrastructure would again increase the computational capacity of these low-cost devices, as well as saving battery [4].

Advanced Network Protocols

Throughout this work, HTTP/TCP solutions have been considered for content delivery. It is, however, possible to adopt other protocols as well.

Some solutions revert to UDP, in order to improve latency at the cost of reliability. As an example, the HTTP/3 protocol will soon be standardized by the Internet Engineering Task Force (IETF) [5]. This protocol is based on the Quick UDP Internet Connections (QUIC) protocol[1], proposed by Google in 2012 [6]. HTTP/3 establishes a number of multiplexed UDP connections, resulting in independent delivery of multiple streams of data. In contrast to HTTP/2, which uses a single TCP connection, this approach avoids head-of-line-blocking if any of the TCP packets are delayed or lost. Applying QUIC to the delivery of YouTube videos, Google reported a reduction of the rebuffering rate of 18.0% and 15.3% for desktop and mobile users, respectively, compared to standard TCP [7]. We believe, however, that the full potential of QUIC, and other network protocols in general, has not yet been reached. More research is required to further improve content delivery, focusing on those aspects which contribute most to the QoE.

Softwarized Networks

In this dissertation, we focused on end-to-end delivery of the provided content and services only. Over the last years, however, in-network optimizations gained a lot of attention. Network virtualization allows to map virtual nodes on top of an existing physical network [8]. These nodes are interconnected through virtual links, which are realized as a path through the underlying network. This allows to define multiple networks on top of an existing infrastructure, complying to possibly entirely different QoS requirements. This principle can be combined with software-defined networks (SDN), in which the data layer is separated from the control layer; while packets are sent on the former, their routing is softwarematically defined by the latter [9]. This not only results in more flexibility, but also allows to dynamically change virtual network topologies.

SDN-based approaches have recently been applied to HAS. As an example, Petrangeli et al. propose to use SDN to prioritize packets when a client is expected to experience buffer starvation in the near future [10]. In doing so, requested segments can be delivered sooner, and playout freezes can be avoided. The importance of these paradigms is further reflected in extensions to the Dynamic Adaptive Streaming over HTTP (DASH), in the form of Server and Network-Assisted DASH (SAND) [11]. SAND defines a list of messages a DASH client and a SAND-enabled server can exchange, allowing in-network optimizations during the video streaming session. We believe that this concept will become increasingly more important for HAS, and therefore, should gain more research attention in the future.

[1] IETF's use of the term QUIC is no longer an acronym, but rather the name of the protocol.

References

[1] R. Games. *Control Robots in Chernobyl*, 2018. Available from: https://www.kickstarter.com/projects/remotegames/game-with-remote-controlled-robots-over-internet.

[2] Independent. *Surgeon Performs World's First Remote Operation Using 5G Surgery on Animal in China*, 2019. Available from: https://www.independent.co.uk/life-style/gadgets-and-tech/news/5g-surgery-china-robotic-operation-a8732861.html.

[3] G. Fettweis, H. Boche, T. Wiegand, E. Zielinski, H. Schotten, P. Merz, et al. *The Tactile Internet*. Technical report, International Telecommunications Union, 2014. Available from: https://www.itu.int/dms_pub/itu-t/oth/23/01/T23010000230001PDFE.pdf.

[4] M. Torres Vega, T. Mehmli, J. van der Hooft, T. Wauters, and F. De Turck. *Enabling Virtual Reality for the Tactile Internet: Hurdles and Opportunities*. In Proceedings of the 14th International Conference on Network and Service Management, pages 378–383, 2018.

[5] Internet Engineering Task Force. *Identifying our deliverables*, 2018. Available from: https://mailarchive.ietf.org/arch/msg/quic/RLRs4nB1lwFCZ_7k0iuz0ZBa35s.

[6] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind. *Innovating Transport with QUIC: Design Approaches and Research Challenges*. IEEE Internet Computing, 21(2):72–76, 2017.

[7] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, et al. *The QUIC Transport Protocol: Design and Internet-Scale Deployment*. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pages 183–196, New York, 2017. ACM.

[8] N. M. Mosharaf Kabir Chowdhury and R. Boutaba. *Network Virtualization: State-of-the-Art and Research Challenges*. IEEE Communications Magazine, 4(7):20–26, 2009.

[9] Open Networking Foundation. *Software-Defined Networking: The New Norm for Networks*. Technical report, Open Networking Foundation, 2012. Available from: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[10] S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. *Software-Defined Network-Based Prioritization to Avoid Video Freezes in HTTP Adaptive Streaming*. Networks, 26(4):248–268, 2016.

[11] *Information Technology - Dynamic Adaptive Streaming over HTTP (DASH) - Part 5: Server and Network Assisted DASH*. Technical report, International Organization for Standardization, 2017.

A

HTTP/2-Based Adaptive Streaming of HEVC Video over 4G/LTE Networks

In HAS, the client can adapt the requested video quality to network changes, generally resulting in a smoother playback. Unfortunately, live streaming solutions still often suffer from playout freezes and a large end-to-end delay. By reducing the segment duration, the client can use a smaller temporal buffer and respond even faster to network changes. However, since segments are requested subsequently, this approach is susceptible to high round-trip times. In this chapter, we discuss again the merits of an HTTP/2 push-based approach, and analyze the induced bit rate overhead for HEVC-encoded video segments with a sub-second duration. Through an extensive evaluation with the generated video content, we show that the proposed approach results in a higher video quality and a lower freeze time, and allows to reduce the live delay compared to traditional solutions over HTTP/1.1. More importantly, however, we present the details of a measurement study on the available bandwidth in real 4G/LTE networks. The results of this study have been used in the evaluations of Chapters 3, 4, and 5.

⋆⋆⋆

J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. Rondao Alface, T. Bostoen, and F. De Turck

Published in IEEE Communications Letters, vol. 20, no. 11, p. 2177-2180, 2016

Abstract In HTTP adaptive streaming (HAS), video content is temporally divided into multiple segments, each encoded at several quality levels. The client can adapt the requested video quality to network changes, generally resulting in a smoother playback. Unfortunately, live streaming solutions still often suffer from playout freezes and a large end-to-end delay. By reducing the segment duration, the client can use a smaller temporal buffer and respond even faster to network changes. However, since segments are requested subsequently, this approach is susceptible to high round-trip times. In this letter, we discuss the merits of an HTTP/2 push-based approach. We present the details of a measurement study on the available bandwidth in real 4G/LTE networks, and analyze the induced bit rate overhead for HEVC-encoded video segments with a sub-second duration. Through an extensive evaluation with the generated video content, we show that the proposed approach results in a higher video quality (+7.5%) and a lower freeze time (-50.4%), and allows to reduce the live delay compared to traditional solutions over HTTP/1.1.

A.1 Introduction

Today, more than half of the Internet traffic is generated by video streaming applications [1]. To meet increasing requirements, the concept of HTTP adaptive streaming (HAS) has recently been introduced. As shown in Figure A.1, content is encoded at different quality levels and temporally divided into segments with a typical length of 2 to 10 seconds. The client uses a rate adaptation heuristic to decide upon the downloaded quality for each segment, based on criteria such as the perceived bandwidth and the buffer filling. The goal of this heuristic is to optimize the user's Quality of Experience (QoE), which depends among others on the average video quality, the frequency of quality changes and the occurrence of video freezes. Many heuristics and solutions have been proposed in literature, but we refer to a survey by Seufert et al. for an elaborate view on the matter [2].

Despite the many advantages of HAS, there are drawbacks as well. First, playout freezes still occur in 27% of video sessions [3]. Especially in environments with rapid bandwidth changes, the client may fail to adapt to new network conditions. Furthermore, video content is generally en-

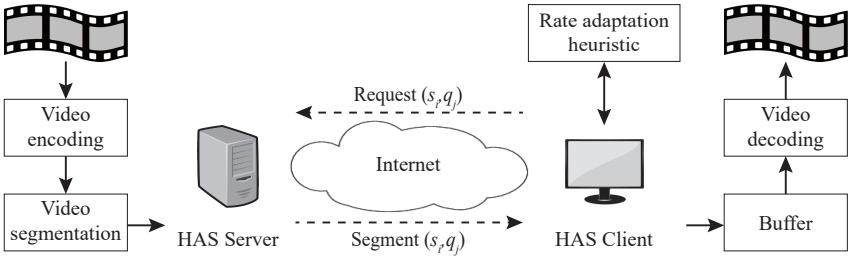


Figure A.1: The concept of HTTP adaptive streaming.

coded at variable bit rate, with more bits assigned to scenes with rapid motion. Therefore, it often takes significantly longer to download a segment than initially estimated, increasing the chances of buffer starvation. Second, since segments of multiple seconds are typically used, the end-to-end delay in current HAS deployments is in the order of tens of seconds. This is detrimental for the QoE in live video streaming, where the delay should be as low as possible [4].

One solution to these issues is the use of H.265/HEVC, a video compression standard which was developed to provide twice the compression efficiency of the previous standard, H.264/AVC [5]. In HEVC, coding units of up to 64x64 pixels are used instead of 16x16, and more intra-picture directions, finer fractional motion vectors and larger transform blocks are used to achieve this improvement in compression performance. Reducing the encoding bit rate has a significant impact on the QoE, as fewer data needs to be transferred from server to client. Another solution is to use segments with a sub-second duration. Shorter segments allow to limit the maximum download time of individual segments and respond faster to sudden changes in the available bandwidth. Furthermore, they allow to use a smaller buffer, which results in a potential decrease of the end-to-end delay in live streaming scenarios. Unfortunately, since every segment has to start with an Instantaneous Decoder Refresh (IDR) frame, a higher bit rate is required to achieve the same visual quality. Moreover, since a unique request is required to retrieve every single video segment, solutions with low segment duration are susceptible to high round-trip times (RTT). This problem mainly arises in mobile networks, where the RTT varies from 33 to 857 ms, depending on the network carrier and the type of connection [6].

The contributions of this letter are threefold. First, we explain an effective means to eliminate RTT cycles in Section A.2, using the server push feature of the recently standardized HTTP/2 protocol [7, 8]. This approach allows to effectively use short video segments, achieving the ad-

vantages described above. Second we present the details of two measurement studies in Section A.3. Particularly, we actively measured the available throughput in real 4G/LTE networks and performed an analysis of the induced bit rate overhead for short, HEVC-encoded video segments. Third, detailed results are presented in Section A.4 to characterize the gain of the proposed push-based approach compared to state-of-the-art HAS over HTTP/1.1. Final conclusions are drawn in Section A.5.

A.2 HTTP/2 Push-Based Approach

In HAS, a video session starts with the client sending a request for the video's media presentation description (MPD). This file contains information regarding the video segments, such as the duration, resolution and available bit rates. Based on the contents of the MPD, the client then requests video segments subsequently, generally ramping up the buffer by downloading segments at the lowest quality. After this startup phase, further decisions regarding the video quality are made by the client. The main drawback of this approach is that one RTT cycle is lost to download each segment, which has a significant impact on the startup time and bandwidth utilization in high-RTT networks. This behavior is illustrated in Figure A.2, for the first phase of a live streaming session.

The HTTP/2 standard was published as an IETF RFC in February 2015, mainly focusing on the reduction of latency in web delivery [7]. Recently, a number of papers were published regarding the use of this new protocol in HAS. Wei et al. proposed a k-push approach, in which k segments are sent per request [9]. In later work, the authors proposed to change the parameter value of k dynamically based on network characteristics [10]. Focus in this research is mainly on reducing the live latency and the number of GET requests issued by the client, without considering the impact of freezes or the encoding overhead introduced by shorter video segments. In previous work, we proposed a scheme in which the base layer segments for Scalable Video Coding (SVC) are pushed by the server, while enhancement layers are pulled by the client [11]. Although a significant reduction of the freeze time is achieved compared to AVC-based solutions, the encoding overhead introduced by inter-layer dependencies makes it unfeasible to provide more than three quality representations.

In the push-based approach [8], the server uses HTTP/2's server push to push m segments to the client as soon as the MPD request is received, where m corresponds to the number of segments that fit into a preferred buffer size defined by the client. Since state-of-the-art heuristics ramp up the buffer by downloading segments at the lowest quality, it makes sense

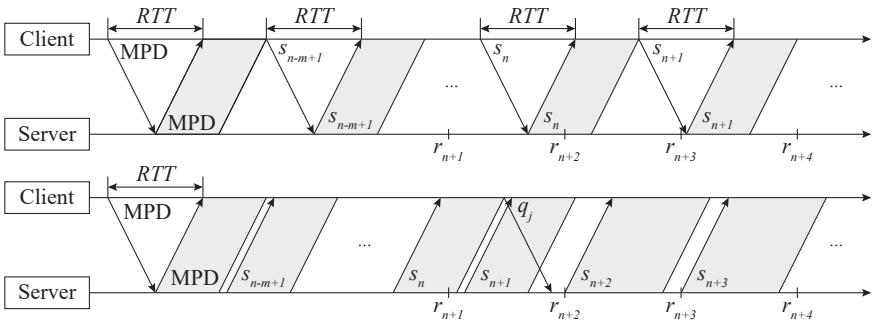


Figure A.2: An example live video scenario for HTTP/1.1 (top) and HTTP/2 (bottom), where the client requests m available segments to ramp up the buffer. If the last released segment has index n, the first segment to play is $n - m + 1$. Note that r_i denotes the release of segment i at the server-side, while s_i denotes its request for download by the client. Furthermore, quality q_j indicates that the server should change the quality of pushed segments to j.

to push segments at this quality as well. As illustrated in Figure A.2, at least one RTT cycle is gained in the reception of the first video segment, and multiple RTT cycles are gained during the buffer rampup phase. Once the MPD and the first m segments are sent, the server periodically pushes a new segment to the client at the specified quality level. Every time a segment is received, the rate adaptation heuristic determines the most suitable video quality and if required, a request is sent to change the bit rate of pushed segments. Since the first m segments are pushed back-to-back when the MPD is requested, the proposed approach can significantly reduce the client's startup delay in high-RTT networks. Short segments can be used, as no RTT cycles are lost, further reducing the startup delay. Additionally, since a smaller buffer can be used, the approach allows to reduce the total end-to-end delay as well.

Preliminary evaluations showed that it is important to limit the maximum number of segments in flight; if a large amount of high-quality segments are queued in the network, e.g. right after a bandwidth drop, buffer starvation at the client-side is likely to occur. An appropriate rule of thumb for the maximum number of segments k in flight is $ceil\left(\frac{RTT}{seg}\right) + 1$, where k is directly proportional to the ratio of the RTT and the segment duration seg. Indeed, the higher this ratio, the more segments should be pushed in order to bridge idle RTT cycles. In our experimental setup, it will be sufficient to use $k = 2$.

A.3 Measurement Study

A.3.1 Available Bandwidth in 4G/LTE Networks

To evaluate the proposed approach, we decided to focus on 4G/LTE networks. In order to provide a realistic evaluation, we collected throughput measurements in 4G networks within the city of Ghent, Belgium, in January and February 2016. We have built a dataset over multiple routes, measuring the available bandwidth while downloading a large file over HTTP. To guarantee appropriate download speeds, we hosted a dedicated server in iLab.t's Virtual Wall infrastructure[1], connected through a 100 Mb/s Ethernet connection. In this way, bandwidth and latency measurements indicate the performance of the wireless 4G connection, with minimal interference from the wired network. As for the client, we developed an Android application which logs all required information, running on a smartphone (Huawei P8 Lite) connected over 4G. Similar to the collection of 3G throughput traces by Riiser et al. [12], several properties are logged, among which the GPS coordinates, the number of bytes received since last datapoint and the number of milliseconds since last datapoint. From these last two entries, the average throughput can be obtained.

We collected throughput logs for six types of transportation: foot, bicycle, bus, tram, train and car[2]. As an example, Figure A.3 shows the selected route in a car and the measured bandwidth over time. Lower throughput values are observed when connectivity is limited, due to tunnels, large buildings and bad coverage in general. Also, the type of transportation and the selected route have a strong impact on the available bandwidth. As an example, the average throughput on a train around the city was 22.8 Mb/s $\pm$ 14.6 Mb/s, while this was 33.9 Mb/s $\pm$ 15.8 Mb/s in a car driving on the ring road. The measured bandwidth ranged from 0 Mb/s (connection interrupted) through 111 Mb/s (higher than 100 Mb/s because of network queuing), with an average of 30.3 Mb/s $\pm$ 16.7 Mb/s. The complete dataset, which consists of 40 traces and covers 5 hours of monitoring, has been made available online [13].

A.3.2 HEVC-Encoded Video

In this research, we decided to focus on HEVC because of its promising compression efficiency. Since our intention is to use video segments with a sub-second duration, it is important to analyze the induced encoding

[1] https://doc.ilabt.imec.be/ilabt-documentation/

[2] The authors would like to thank T. Baele and L. Timperman for their kind assistance during the data collection.

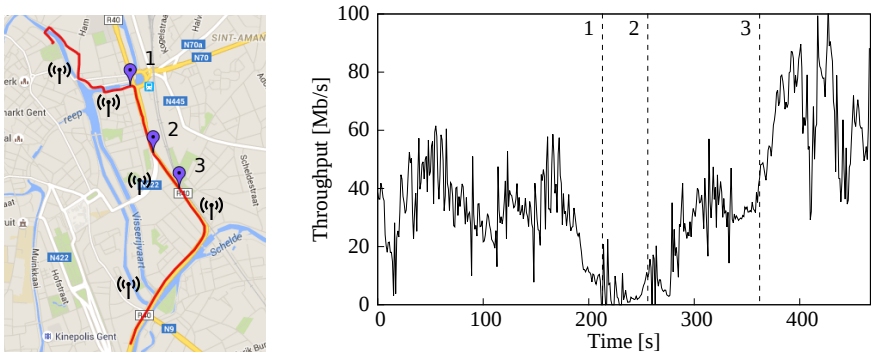


Figure A.3: A car travelling from north to south (left), along with the measured throughput (right). When travelling from (1) to (2), large townhouses on the right side impede the client's connection. Arriving at (2), the client switches to a new antenna with better coverage. Once an open area is reached in (3) and a new antenna is again selected, throughput improves significantly.

overhead. The considered video sequence in our analysis and evaluation is Netflix's El Fuente, which has a total length of 476 seconds and a frame rate of 60 FPS. The video is encoded using HEVC, providing six quality levels at nominal bit rates of 0.3, 1.0, 2.3, 5.2, 10.9 and 21.4 Mb/s, with a spatial resolution ranging from 540p to 2160p video. Using the x265 encoder[3], the video is segmented using five segment durations: 133, 267, 500, 1000 and 2000 ms. To allow each segment to be decoded independently, every segment starts with an IDR frame and the Group of Pictures (GOP) length is set to 8, 16, 30, 60 and 120 frames respectively. To assess the impact of shorter GOP lengths on the compression performance, the encodings for different segment durations have been set to target the same visual quality and allow a subsequent overhead in the achieved nominal bit rate. To realize this, we have selected the Constant Rate Factor (CRF) rate control implemented in the x265 encoder. The obtained encodings for the same nominal rates but different segment durations, have the same visual quality, measured in terms of Peak-Signal-to-Noise-Ratio (PSNR), with deviations smaller than 0.233 dB. Compared to a GOP length of 120 frames, the average over-head is 6.3%, 9.2%, 29.3% and 60.5% for a GOP length of 60, 30, 16 and 8 frames respectively. Figure A.4 shows the obtained bit rates of the six quality representations, with a clear increase for segments with a sub-second duration. In the next section, the proposed approach will be evaluated for a segment duration of 500 ms. This allows

[3]http://x265.org

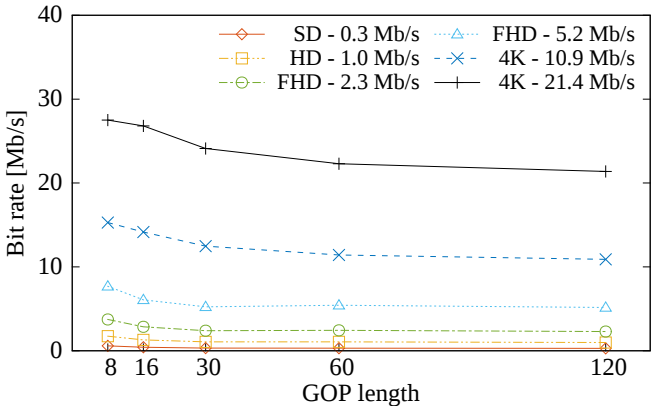


Figure A.4: Obtained video bit rates for the different quality representations and a GOP length of 8, 16, 30, 60 and 120 frames.

to reduce the buffer size to the order of seconds and increase video quality in high-RTT networks, while the overhead is limited to 9.2%.

As for the encoding time, using a multicore platform with Intel Core i7 CPUs and an Nvidia GTX 980 GPU, x265 with OpenCL acceleration was able to encode the FHD content in real-time, with frame rates ranging from 63 FPS (GOP 8) to 68 FPS (GOP 120). For the 4K representations however, frame rates ranged from 21 FPS (GOP 8) to 24 FPS (GOP 120). Faster software HEVC encoders were reported recently to be able to encode 4K in real-time on similar CPU platforms [14].

A.4 Evaluation

A.4.1 Experimental Setup

To allow a fair comparison of the proposed approach with traditional HAS, a network topology is emulated using the Mininet framework[4]. It consists of a single client, streaming the encoded video from a dedicated Jetty web server[5]. A new request handler is defined, which processes the client's GET requests using a specific query to start the pushing of segments at a given quality representation. The client is implemented on top of the libdash library[6], the official reference software of the MPEG-DASH standard. We provided support for HTTP/2 using the nghttp2 library[7], and

[4]http://mininet.org/
[5]http://www.eclipse.org/jetty/
[6]https://github.com/bitmovin/libdash
[7]https://nghttp2.org/

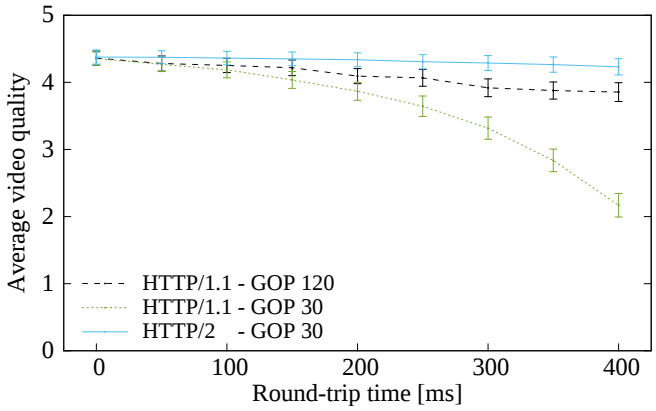


Figure A.5: Impact of the RTT on the video quality, both for HTTP/1.1 and HTTP/2 with an initial buffer size of 10 seconds.

implemented the required logic to asynchronously handle pushed video segments. Client-side rate adaptation is based on the FINEAS heuristic by Petrangeli et al. [15]. This heuristic estimates the segments' download time to achieve a target buffer filling level, resulting both in a higher video quality and a lower amount of playout freezes compared to state-of-the-art solutions. To avoid an excessive amount of quality switches for short segments, the client is only allowed to increase the quality every 2 s. The collected 4G traces for same-type vehicles are merged together, in order to obtain 30 unique bandwidth traces with a minimal length of 494 s and an average bandwidth of 30.3 Mb/s $\pm$ 16.8 Mb/s. Using traffic control command *tc* for traffic shaping, the client can stream 30 episodes of the video with a different bandwidth pattern for every episode. A lower threshold of 50 kb/s is used, in order to guarantee correct packet scheduling with *tc*. The bandwidth at the server-side is fixed at 100 Mb/s, same as in the measurement study.

A.4.2 Obtained Results

First, the performance of traditional HAS and the push-based approach are evaluated for increasing values for the RTT, with an initial buffer size of 10 s. Note that when playout freezes occur, the buffer is expanded as to hold all segments released at the server-side. Figure A.5 shows that for HTTP/1.1, the video quality, averaged out over all segments - 0 for the lowest quality level, 5 for the highest - drops significantly for higher RTTs, regardless whether a segment duration of 2000 or 500 ms is used. The video quality for the proposed approach over HTTP/2 is not impacted however,

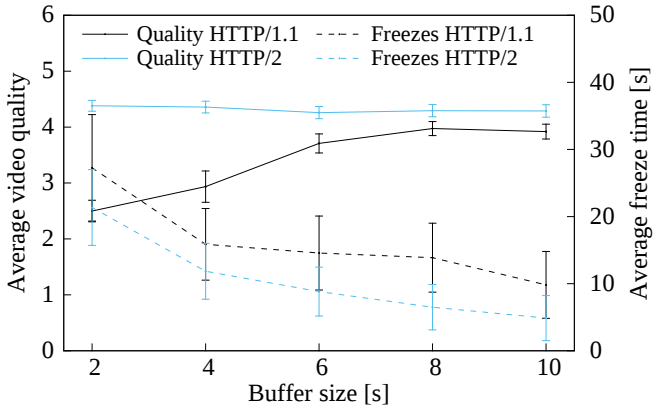


Figure A.6: Impact of the buffer size on the video quality and freeze time, both for HTTP/1.1 (GOP 120) and HTTP/2 (GOP 30) with an RTT of 300 ms.

because bandwidth utilization is maximized by actively pushing segments from server to client. Short segments can thus effectively be used, which is not true for traditional HAS over HTTP/1.1.

In a second set of experiments, performance is evaluated as a function of the initial buffer size, for an RTT of 300 ms. Figure A.6 shows that, while the video quality over HTTP/1.1 increases for larger values of the buffer size, it is more or less constant for the push-based approach. Despite an encoding overhead of 9.2%, the average quality is significantly higher because of better bandwidth utilization. As for the freeze time, a clear decrease is observed for higher buffer sizes, because a playout freeze is less likely if more content can be buffered at the client-side. More importantly however, the freeze time for the proposed approach is always lower than for traditional HAS, because the client can respond faster to changes in the available bandwidth or buffer fulling.

The most relevant results are summarized in Table A.1. For a standard buffer size of 10 s, the proposed approach results in a significantly higher video quality (+7.5%), a lower freeze time (−50.4%) and a lower startup delay (−25.0%) compared to traditional HAS. Focusing on a re-

Table A.1: Performance summary for an RTT of 300 ms. Average values are reported, along with the 95% confidence intervals.

HTTP	Buffer [s]	Video quality	Quality switches	Freeze time [s]	Startup delay [s]
HTTP/1.1	10	4.919 ± 0.132	49.633 ± 6.663	9.817 ± 4.988	2.408 ± 0.052
HTTP/1.1	6	4.754 ± 0.140	64.333 ± 7.254	15.190 ± 5.204	2.405 ± 0.047
HTTP/2	10	5.288 ± 0.111	52.067 ± 9.766	4.867 ± 3.361	1.806 ± 0.085
HTTP/2	6	5.270 ± 0.117	60.233 ± 10.600	8.977 ± 4.363	1.799 ± 0.084

duction of the live delay, a smaller buffer size of 6 s with pull-based HAS results in a significantly lower video quality (-3.4%) and a higher freeze time ($+54.7\%$), compared to a buffer size of 10 s. However, comparing results for the push-based approach and a buffer size of 6 s, with traditional HAS and a buffer size of 10 s, a higher video quality ($+7.1\%$) and a lower startup delay (-25.3%) are obtained, while differences for the freeze time are not statistically significant (two-tailed Wilcoxon signed-rank test, $p = 0.82$). This shows that the proposed approach allows the client to follow the live signal more closely, without losing performance on other metrics.

A.5 Conclusions

In this letter, we discussed an HTTP/2 push-based approach for HTTP adaptive streaming (HAS) which enables the use of video segments with a sub-second duration in mobile, high round-trip time networks. We quantified the encoding overhead for short HEVC-encoded segments, and determined that the segment duration should not be lower than 500 ms to limit the overhead to 9.2%. We also performed measurements for the available bandwidth in real 4G/LTE networks within the city of Ghent, Belgium, and created a dataset which has been made available online. Using the encoded content and collected throughput traces in an extensive evaluation, we showed that the presented approach results in a higher video quality ($+7.5\%$) and a lower freeze time (-50.4%), and allows to reduce the live delay compared to solutions over HTTP/1.1. Future work will focus on further improving the user's QoE through HTTP/2 features such as request/response multiplexing and stream prioritization, on reducing the encoding overhead for short video segments and on adaptively changing the segment duration based on network conditions.

References

[1] Sandvine Incorporated. *Global Internet Phenomena Report*. 2016.

[2] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia. *A Survey on Quality of Experience of HTTP Adaptive Streaming*. IEEE Communications Surveys Tutorials, 17(1):469–492, 2015.

[3] Conviva. *Viewer Experience Report*. 2015.

[4] T. Lohmar, T. Einarsson, P. Fröjdh, F. Gabin, and M. Kampmann. *Dynamic Adaptive HTTP Streaming of Live Content*. In IEEE International

Symposium on a World of Wireless, Mobile and Multimedia Networks, pages 1–8, 2011.

[5] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand. *Overview of the High Efficiency Video Coding (HEVC) Standard*. IEEE Transactions on Circuits and Systems for Video Technology, 22(12):1649–1668, 2012.

[6] OpenSignal. *IConnect 4G Coverage Maps*. 2014. Available from: http://opensignal.com/networks/usa/iconnect-4g-coverage/.

[7] M. Belshe, R. Peon, and M. Thomason. *Hypertext Transfer Protocol Version 2*. Technical Report Internet-Draft, RFC Editor, 2015. Available from: https://datatracker.ietf.org/doc/draft-ietf-httpbis-http2/.

[8] R. Huysegems, J. van der Hooft, T. Bostoen, P. Rondao Alface, S. Petrangeli, T. Wauters, and F. De Turck. *HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming*. In ACM Multimedia Conference, pages 541–550, 2015.

[9] S. Wei and V. Swaminathan. *Low Latency Live Video Streaming over HTTP 2.0*. In ACM Network and Operating System Support on Digital Audio and Video Workshop, pages 37:37–37:42, 2014.

[10] M. Xiao, V. Swaminathan, S. Wei, and S. Chen. *Evaluating and Improving Push-Based Video Streaming with HTTP/2*. In ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video, pages 3:1–3:6, 2016.

[11] J. van der Hooft, S. Petrangeli, N. Bouten, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. *An HTTP/2 Push-Based Approach for SVC Adaptive Streaming*. In IEEE/IFIP Network Operations and Management Symposium, pages 104–111, 2016.

[12] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. *Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications*. In ACM Conference on Multimedia Systems, pages 114–118, 2013.

[13] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. Rondao Alface, T. Bostoen, and F. De Turck. *4G/LTE Bandwidth Logs*. 2016. Available from: https://users.ugent.be/~jvdrhoof/dataset-4g/.

[14] T. K. Heng, W. Asano, T. Itoh, A. Tanizawa, J. Yamaguchi, T. Matsuo, and T. Kodama. *A Highly Parallelized H.265/HEVC Real-Time UHD Software Encoder*. In IEEE International Conference on Image Processing, pages 1213–1217, 2014.

[15] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turck. *QoE-driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming*. ACM Transactions on Multimedia Computing, Communications and Applications, 12(2):28:1–28:24, 2015.