QoE-beheer van HTTP-gebaseerde adaptieve videodiensten

QoE Management of HTTP Adaptive Streaming Services

Niels Bouten

Promotoren: prof. dr. ir. F. De Turck, prof. dr. S. Latré
Proefschrift ingediend tot het behalen van de graden van
Doctor in de ingenieurswetenschappen: computerwetenschappen (Universiteit Gent) en
Doctor in de wetenschappen: informatica (Universiteit Antwerpen)

UNIVERSITEIT
GENT

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. D. De Zutter
Faculteit Ingenieurswetenschappen en Architectuur

Universiteit
Antwerpen

Departement Wiskunde en Informatica
Voorzitter: prof. dr. C. Blondia
Faculteit Wetenschappen

Academiejaar 2016 - 2017

ISBN 978-90-8578-933-8
NUR 986, 988
Wettelijk depot: D/2016/10.500/66

UNIVERSITEIT GENT

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Informatietechnologie

Universiteit Antwerpen

Universiteit Antwerpen
Faculteit Wetenschappen
Departement Wiskunde en Informatica

Promotoren: prof. dr. ir. Filip De Turck
prof. dr. Steven Latré

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur

Vakgroep Informatietechnologie
Technologiepark-Zwijnaarde 15, B-9052 Gent, België

Universiteit Antwerpen
Faculteit Wetenschappen

Departement Wiskunde en Informatica
Middelheimlaan 1, B-2020 Antwerpen, België

IWT

Dit werk kwam tot stand in het kader van een specialisatiebeurs van het IWT-Vlaanderen (Instituut voor de aanmoediging van Innovatie door Wetenschap en Technologie in Vlaanderen)

INTEC

Proefschrift tot het behalen van de graden van Doctor in de ingenieurswetenschappen: computerwetenschappen (Universiteit Gent) en Doctor in de wetenschappen: informatica (Universiteit Antwerpen)
Academiejaar 2016-2017

# Dankwoord

Na vijf jaar is eindelijk het moment aangebroken om dit dankwoord neer te pennen. Normaliter zou dit het eenvoudigst te schrijven deel van het boek moeten vormen, maar niets is minder waar. Over de jaren heen heb ik zoveel mensen ontmoet, met zoveel mensen samengewerkt en van zoveel mensen iets geleerd, dat het bijna onmogelijk wordt om alles hier neer te schrijven en niemand over het hoofd te zien. Een doctoraat mag dan wel beschouwd worden als een individueel huzarenstukje, het resultaat zou niet zijn wat het nu is zonder de steun en inbreng van een groot aantal personen.

Zonder mijn promotoren, Filip De Turck en Steven Latré, zou dit doctoraat waarschijnlijk niet tot stand gekomen zijn. Daarom wil ik hen in de eerste plaats bedanken voor de mogelijkheden die ze mij hebben geboden, hun steun en hun vertrouwen door de jaren heen. Filip zou ik in het bijzonder willen bedanken voor het bieden van alternatieve invalshoeken wanneer ik het onbegonnen werk vond om een review commentaar te tackelen. Daarnaast wens ik Filip ook bedanken om mij met lichte dwang aan te zetten om eens een dag te skippen op conferentie om de omgeving te verkennen. De vlucht boven en landing in de Grand Canyon is een ervaring die ik nooit zal vergeten. De eerste ontmoeting met Steven was op mijn eerste werkdag, toen hij met grote haast het bureau kwam binnengestormd, enkele minuten in de lades en kasten zat te rommelen op zoek naar zijn sleutels en zonder boe of ba terug vertrok. Gelukkig lag dit enkel aan een vermoeiende reis en werd dit al vlug rechtgezet met een sympathieke babbel. Bedankt voor de uitmuntende begeleiding tijdens mijn IWT-aanvraag, de balpennen die je hebt leeggeschreven tijdens het reviewen van mijn papers en de plezante off-topic discussies op en naast het werk. Ik heb hier ontzettend veel uit geleerd! Naast mijn promotoren, wil ik ook Jeroen Famaey bedanken voor de vlotte samenwerking die we over de jaren heen hebben gehad, zowel tijdens het begeleiden van thesissen en projecten als tijdens het uitwerken van nieuwe paper- en patentideeën.

Zonder de geschikte omkadering, geboden door de onderzoeksgroep IBCN, was het onmogelijk geweest om zorgeloos mijn doctoraatsonderzoek te verrichten. Vooreerst dien ik dus Piet Demeester te bedanken voor de dagdagelijkse leiding van de toch wel uit de kluiten gewassen onderzoeksgroep. Ook het A-team (Joeri, Bert, Simon, Vicent en Brecht) verdient een pluim voor de IT-ondersteuning en het beschikbaar maken van de nodige infrastructuur. Administratief worden we in de watten gelegd door Martine, Davinia, Bernadette en Joke. Last minute een vluchtwijziging regelen of een conferentieticket en hotel boeken, vormt voor hen nooit een probleem en wordt verzorgd onder een leuke koffieklets. Ook Sabrina

verdient een vermelding voor haar poetswerk en de babbels aan de koffiemachine. Naast de onderzoeksgroep en de Universiteit Gent, dien ik ook het IWT, ondertussen omgedoopt tot VLAIO, te bedanken voor de persoonlijke financiering tijdens mijn doctoraat. Ook iMinds wens ik te bedanken omdat ze mij de mogelijkheid geboden hebben om mee te werken aan verschillende ICON-projecten en zo kennis te maken met heel wat Vlaamse bedrijven.

Tijdens mijn doctoraat ben ik geen enkele dag met tegenzin gaan werken. Dat is voornamelijk te wijten aan de toffe collega's die rondlopen op de onderzoeksgroep. In het bijzonder de bureaugenoten van 2.21 en 200.012: Bram G, Jeroen S, Klaas R, Kristof S, Leandro O, Maxim C, Merlijn S, Niels S, Olivier VL, Philip L, Piet S, Stefano P, Steven B, Steven VC, Thijs W, Thomas V, Tim V en Wim VdM. De interessante gesprekken onder de lunch, de uitstapjes naar het Zuiderpoortparkje, de helaas ten onder gegane Friday drinks, de klucht van het aapje en de banaan, het afscheidsfeest van Olivier met bijhorende gekke Italiaanse dansmoves, het "kindkaartsysteem" dat werd uitgedacht om de klaagzang van papa Philip in te perken, de IBCN-weekends met onze bureau ... zijn dingen die mij altijd zullen bijblijven. Daarnaast wil ik ook enkele niet-bureaugenoten bedanken voor de sympathieke babbels, plezante lunches bij de belastingen en samenwerking tijdens project- of onderwijsactiviteiten: Bruno V, Tim W, de Femkes, Stijn V, Jeroen vdH, Hendrik M, Thomas en Wannes, Pieter B, Jonas A en Cedric DB. Ook Bart Dhoedt wil ik bedanken voor de praatjes onder een koffietje 's ochtends vroeg, hopelijk vind je nieuwe vroege vogels ;).

Een van de punten die een doctoraat aantrekkelijk maken is de mogelijkheid om de wereld rond te reizen om conferenties of projectmeetings bij te wonen. Vooreerst wens ik de collega's van Flamingo te bedanken voor de vele meetings en social events op locatie: van Oktoberfest tot Barceloneta, ik heb ervan genoten! Ook op conferentie werden onvergetelijke avonturen beleefd: de tevergeefse zoektocht naar wildlife die pas laat op de avond werd beloond tijdens een noodstop voor overstekende herten en de ontmoeting in de lift met Doo Doo The Clown in Canada, de stutjes met vet in de zlotti-bar en de ondergrondse discobar met breakdance battle in Krakow, de biertram en road trip in de Yeti in Brno, de cocktailventers en de onvergetelijke zoektocht naar de Bar in Bar-celona, de onleesbare menukaarten en lichtrijke karaoke-bars in Seoul, de Cessna-vlucht vanuit poleposition over de Grand Canyon in Las Vegas ... Bedankt aan de metgezellen op deze onvergetelijke trips: Bram N, Hendrik M, Jeroen F, Jeroen vdH, Maryam B, Matthias S, Maxim C, Pieter-Jan M, Rafael DS, Stefano P, Steven L, Thomas V.

Daarnaast wil ik twee collega's in het bijzonder bedanken. Vooreerst wil ik Steven Van Canneyt bedanken om mij steeds op te beuren wanneer het druk was, door je eigen situatie nog wat pessimistischer voor te stellen. We zijn samen aan dit avontuur begonnen, haalden samen onze IWT-beurs en breien er ook samen een eind aan deze week. Bedankt om me in mei uit mijn sloffen te doen schieten door mij het idee te geven dat je eerder ging gedoctoreerd zijn dan ik! Daarnaast wil ik ook mijn beste collega Maxim Claeys bedanken, of misschien is de term "vriend" ondertussen beter geschikt? Bedankt maatje, om er steeds te zijn met een gepaste kwinkslag om mij op te beuren, voor de zalige tijden op Barceloneta, het

economisch verantwoord blijven plakken in de Bikini Bar, de backup wanneer we werden ondermijnd door een spervuur aan Flamingo-vragen en het vervangen van Paulien haar autoband in de gietende regen!

Om even het doctoraatswerk te vergeten, kon ik steeds terugvallen op de Ieperse vrienden. Bedankt voor de vele overlegmomenten in lokaal d3f00 onder begeleiding van het noodzakelijke natje en droogje en onder toeziend oog van moderator Dries.

Natuurlijk kan ik geen dankwoord schrijven zonder ook mijn ouders te bedanken: bedankt voor de warme thuis, alle kansen en jullie onvoorwaardelijke steun. Ook mijn zus Lies verdient een speciale vermelding, dankzij (of ondanks) haar ervaringen die ze opdeed tijdens haar doctoraat, was ik beter opgewassen tegen de grillen die de wetenschappelijke onderzoekswereld te bieden heeft. Je schoonfamilie kun je kiezen, en gelukkig heb ik dit goed gedaan. Bedankt Kathy en Martin voor jullie steun en voor het mooiste geschenk dat jullie dochter is! Ook omaatje wil ik bedanken voor de vele leute die we samen hebben, we gaan daar zeker nog menig Irish op drinken! Daarnaast wil ik ook mijn grootouders niet vergeten, jullie kunnen er helaas niet meer bij zijn, maar ik weet dat jullie trots zouden zijn op wat jullie klein broskopje heeft bereikt.

Tot slot wil ik de belangrijkste persoon in mijn leven bedanken: Paulien, bedankt om me de voorbije jaren steeds te steunen toen het even tegen zat. Hoewel mijn doctoraat handelt over kwaliteitsadaptatie, ben ik blij dat ik reeds de constante in mijn leven heb gevonden die de kwaliteit van mijn leven alleen maar kan doen toenemen in de toekomst. Liefje, dit boek is geschreven en afgesloten, maar ik hoop dat wij samen nog nog vele mooie bladzijden kunnen vullen.

*Gent, september 2016*
*Niels Bouten*

# Table of Contents

**Dankwoord** **i**

**Samenvatting** **xxix**

**Summary** **xxxiii**

**1 Introduction** **1**
1.1 The Rise of Internet Video Streaming . . . . . 1
1.2 Problem Statement . . . . . 6
1.3 Dissertation Outline . . . . . 8
1.4 Research Contributions . . . . . 12
1.5 Publications . . . . . 14
1.5.1 A1: Journal publications indexed by the ISI Web of Science "Science Citation Index Expanded" . . . . . 14
1.5.2 P1: Proceedings included in the ISI Web of Science "Conference Proceedings Citation Index - Science" . . . . . 15
1.5.3 C1: Other publications in international conferences . . . . 16
1.5.4 European patent applications . . . . . 18
References . . . . . 19

**2 A Multicast-Enabled Delivery Framework for QoE Assurance of Over-The-Top Services in Multimedia Access Networks** **21**
2.1 Introduction . . . . . 22
2.2 Related Work . . . . . 24
2.2.1 Over-The-Top (OTT) video streaming architectures . . . . 24
2.2.2 Management of OTT video services . . . . . 26
2.2.3 Multicast streaming of multimedia services . . . . . 27
2.3 HTTP Adaptive Streaming . . . . . 28
2.4 A Scalable Architecture for Video Delivery . . . . . 30
2.4.1 Architectural overview . . . . . 31
2.4.1.1 Distribution server . . . . . 32
2.4.1.2 Delivery server . . . . . 33
2.4.2 Component interaction details . . . . . 34
2.4.2.1 Handling of packet loss . . . . . 34
2.4.2.2 Managed multicast-enabled video delivery . . . 34

2.5 Autonomic Management of Multicast Streaming . . . . . 35
2.5.1 Autonomic delivery management . . . . . 36
2.5.2 Distribution management . . . . . 37
2.6 Performance Evaluation . . . . . 40
2.6.1 Prototype evaluation . . . . . 40
2.6.1.1 Implementation details . . . . . 40
2.6.1.2 Experimental setup . . . . . 40
2.6.1.3 Results description . . . . . 41
2.6.2 Management algorithm evaluation . . . . . 43
2.6.2.1 Experimental setup . . . . . 45
2.6.2.2 Results description . . . . . 45
2.7 Conclusions . . . . . 50
References . . . . . 52

**3 In-Network Quality Optimization for Adaptive Video Streaming Services 57**
3.1 Introduction . . . . . 58
3.2 Related Work . . . . . 59
3.3 Formal Problem Description . . . . . 63
3.3.1 Definition of variables and assumptions . . . . . 63
3.3.2 Integer Linear Programming (ILP) formulation . . . . . 64
3.4 Algorithms . . . . . 67
3.4.1 Centralized ILP formulation . . . . . 67
3.4.2 Distributed ILP formulation . . . . . 68
3.4.3 Relaxed distributed Linear Programming (LP) formulation 69
3.5 Performance Evaluation . . . . . 71
3.5.1 Experiment setup . . . . . 71
3.5.2 Implementation details . . . . . 73
3.5.3 Evaluation details . . . . . 73
3.5.4 Impact of number of clients . . . . . 74
3.5.5 Impact of number of bottlenecks . . . . . 76
3.5.6 Impact of optimization objective . . . . . 76
3.5.7 Impact of delay . . . . . 79
3.5.8 Impact of multiple servers . . . . . 79
3.6 Conclusion . . . . . 82
References . . . . . 83

**4 QoE-Driven In-Network Optimization for Adaptive Video Streaming Based on Packet Sampling Measurements 87**
4.1 Introduction . . . . . 88
4.2 Related Work . . . . . 91
4.3 QoE-Driven Delivery Architecture for Adaptive Video Streaming . 95
4.4 Algorithm Description . . . . . 98
4.4.1 Definition of variables and assumptions . . . . . 98
4.4.2 Packet-based residual capacity estimation . . . . . 98

4.4.3 QoE-driven quality optimization . . . 101
4.4.4 Distributed QoE-driven quality optimization . . . 103
4.5 Evaluation Results . . . 104
4.5.1 Experiment setup . . . 104
4.5.2 Parameter analysis . . . 109
4.5.2.1 Impact of the decision history . . . 109
4.5.2.2 Impact of the optimization interval . . . 109
4.5.2.3 Impact of the sampling rate . . . 112
4.5.2.4 Impact of the buffer size . . . 113
4.5.2.5 Selected parameter values . . . 114
4.5.3 Impact of the forecasting method . . . 114
4.5.4 Impact of last mile bandwidth fluctuations . . . 116
4.5.5 Overhead of in-network optimization . . . 118
4.5.6 Scalability of the Quality of Experience (QoE)-driven quality optimization . . . 120
4.6 Conclusions . . . 124
References . . . 125

**5 Clustering-Based Adaptation for HAS over Cache Networks 131**
5.1 Introduction . . . 132
5.2 Related Work . . . 134
5.3 Heuristic Description . . . 138
5.3.1 Basic quality selection heuristic . . . 139
5.3.2 Cache-aware quality selection heuristic . . . 140
5.3.3 Cache-assisted quality selection heuristic . . . 141
5.4 Detection of Streaming Origin . . . 143
5.5 Estimation of Cache Content . . . 145
5.6 Evaluation Results . . . 147
5.6.1 Experiment framework . . . 147
5.6.2 Impact of estimated throughput ageing . . . 151
5.6.3 Characterization of the obtained gain of proposed heuristics 152
5.6.4 Impact of detection and estimation . . . 155
5.6.5 Achieved improvement using inferred knowledge . . . 157
5.6.6 Dynamic network scenarios . . . 159
5.7 Conclusions . . . 160
References . . . 162

**6 Semantically Enhanced Mapping Algorithm for Affinity Constrained Service Function Chain Requests 169**
6.1 Introduction . . . 170
6.2 Related Work . . . 172
6.3 Affinity and Anti-Affinity Constraints . . . 176
6.4 Semantic SFC Request Checker . . . 179
6.5 Model . . . 183
6.5.1 Network Function Virtualization (NFV) Infrastructure model 183

6.5.2 Service Function Chain (SFC) model . . . . . 186
6.5.3 Assignment variables . . . . . 187
6.5.4 General constraints . . . . . 187
6.5.5 Affinity and Anti-Affinity constraints . . . . . 189
6.5.6 Objective functions . . . . . 191
6.6 Heuristic Approach . . . . . 192
6.7 Evaluation . . . . . 193
6.7.1 Simulation framework . . . . . 193
6.7.2 Scalability of semantic SFC validation . . . . . 195
6.7.3 Impact of semantic validation on mapping time . . . . . 197
6.7.4 Performance of heuristic approach . . . . . 200
6.7.5 Impact of optimization objective . . . . . 202
6.7.6 Impact of ordering criterion . . . . . 202
6.8 Conclusion and Future Work . . . . . 204
References . . . . . 205

**7 Conclusion** **211**
7.1 Resource-aware management of live HTTP Adaptive Streaming (HAS) services . . . . . 211
7.2 Optimization of Video on Demand (VoD) HAS services . . . . . 212
7.3 Flexible deployment of HAS services . . . . . 214
7.4 Future perspectives . . . . . 214
7.4.1 HAS-aware cache replacement and prefetching strategies . 214
7.4.2 Software Defined Networking (SDN)-enabled delivery optimization of HAS services . . . . . 215
7.4.3 Streaming-aware congestion control . . . . . 216
7.4.4 Mobile HAS delivery . . . . . 216
7.4.5 Automated deployment of HAS SFCs . . . . . 216

**A Minimizing the Impact of Delay on Live SVC-based HTTP Adaptive Streaming Services** **219**
A.1 Introduction . . . . . 220
A.2 Related Work . . . . . 221
A.3 State of the Art Rate Adaptation Heuristics . . . . . 223
A.3.1 Advanced Video Coding (AVC) Microsoft Smooth Streaming (MSS) heuristic . . . . . 223
A.3.2 Scalable Video Coding (SVC) MSS heuristic . . . . . 223
A.3.3 SVC slope heuristic . . . . . 223
A.4 SVC Adaptation Heuristic for Small Buffers . . . . . 224
A.5 Delay-optimized Download Scheduling . . . . . 225
A.5.1 Pipelined scheduling . . . . . 226
A.5.2 Parallel scheduling . . . . . 227
A.6 Experimental Results . . . . . 227
A.6.1 Comparison of adaptation heuristics . . . . . 228
A.6.2 Impact of download scheduling . . . . . 230

A.6.3 Impact of delay . . . 231
A.6.4 Impact of parallel threads . . . 231
A.7 Conclusion . . . 233
References . . . 234

**B Deadline-based Approach for Improving Delivery of SVC-based HTTP Adaptive Streaming Content 237**
B.1 Introduction . . . 238
B.2 Related Work . . . 239
B.3 Priority-Based Delivery of HAS . . . 241
B.3.1 Layer-based prioritization . . . 241
B.3.2 Deadline-based prioritization . . . 242
B.4 Evaluation . . . 243
B.4.1 Prototype implementation . . . 243
B.4.2 Experiment setup . . . 245
B.4.3 Impact of layer-based prioritization . . . 247
B.4.4 Impact of deadline-based prioritization . . . 249
B.5 Conclusion . . . 252
References . . . 253

**C Semantic Validation of Affinity Constrained Service Function Chain Requests 257**
C.1 Introduction . . . 258
C.2 Related Work . . . 259
C.3 Affinity and Anti-Affinity Constraint Model . . . 261
C.4 Semantic SFC Request Checker . . . 264
C.4.1 NFV architecture for SFC request checking . . . 266
C.4.2 Ontology for SFC request modelling . . . 268
C.4.3 Rules . . . 268
C.4.4 Conflict detection . . . 270
C.5 Evaluation . . . 270
C.5.1 Impact of physical network size . . . 272
C.5.2 Impact of virtual network size . . . 273
C.5.3 Impact of number of constraints . . . 273
C.5.4 Combined impact of relevant parameters . . . 274
C.6 Conclusion . . . 275
References . . . 276

# List of Figures


1.1 Forecast of Internet data traffic per month for the next years. Video traffic is expected to exceed $80\%$ of all Internet traffic by 2019 [4]. 3

1.2 HAS traffic exceeded $61\%$ of peak downstream Internet traffic in March 2016 [9]. . . . . 4

1.3 Viewer experience report by Conviva, indicating that buffer starvations and quality degradations are still omnipresent [10]. . . . . 5

1.4 Overview of PhD dissertation. . . . . 9

2.1 Overview of the proposed framework. The framework allows managing OTT services for scalable transportation in the multimedia access network. . . . . 23

2.2 Generic architecture of HAS where a client connects directly to the HAS server over HTTP. . . . . 29

2.3 Overview of a typical consumption pattern of Live TV and Time Shifted TV (TSTV)-based video services. When the content is first streamed live, a peak in viewers occur. Additional waves of service requests can occur depending on the type of the video content. . . 30

2.4 Designed distributed architecture for HAS-enabled content delivery using multicast (MC), allowing a seamless integration with existing HAS-based delivery technologies. The lower part shows an example mapping of the proposed system to a network topology. . 32

2.5 Illustration of the multicast retransmission mechanism at the Delivery Server. . . . . 34

2.6 Illustration of how the management algorithm at the distribution server selects which content to multicast and decides to which channels a delivery server should subscribe. . . . . 35

2.7 Illustration of the subscription algorithm performed by the *Autonomic Delivery Management* component, subscribing to a multicast channel if there exists an $S_r$ where $S_m - S_r < W$. . . . . 37

2.8 Illustration of the mapping algorithm performed by the *Distribution Management* component. . . . . 37

2.9 Emulated network topology modeling a tree-based access network of 1,000+ nodes. . . . . 41

2.10 Impact of the different multicast strategies on the bandwidth. Depending on the multicast strategy, multicasting content is beneficial starting from 2 delivery servers. Confidence levels are shown on the graphs. . . . . 41
2.11 Impact on the average consumed bandwidth of the different multicast strategies. Next to the graph is a table with the average bandwidths and standard deviations. For the multicast scenario no standard deviations are shown since they are all zero. . . . . 42
2.12 Impact on bandwidth of the different retransmission strategies on consumed bandwidth at distribution server. Confidence levels are shown on the graphs. . . . . 43
2.13 Impact on bandwidth of the different retransmission strategies on consumed bandwidth at delivery servers. Confidence levels are shown on the graphs. . . . . 44
2.14 Total average consumed bandwidth on link between distribution server and delivery servers for various loss settings and retransmission methods. Next to the graph is a table with the average bandwidths and standard deviations. . . . . 44
2.15 Impact of multicast strategy on consumed bandwidth on link between distribution and delivery servers, measured with 40 delivery servers, cache sizes of 2,560MB, 20 multicast channels and 20 video channels. . . . . 46
2.16 Impact of number of delivery servers (with 50 clients per delivery server) and multicast management strategy on average consumed bandwidth, measured with cache sizes of 2,560MB, 20 multicast channels and 20 video channels. Confidence intervals are shown on the graphs. . . . . 47
2.17 Impact of number of delivery servers and multicast strategy on average consumed bandwidth, measured with cache sizes of 2,560MB, 20 multicast channels and 20 video channels. . . . . 47
2.18 Impact of different multicasting strategies and cache sizes on average consumed bandwidth, measured with 40 delivery servers, 20 multicast channels and 20 video channels. Confidence intervals are shown on the graphs. . . . . 48
2.19 Impact of different multicasting strategies and cache sizes on average cache hitrate, measured with 40 delivery servers, 20 multicast channels and 20 video channels. Confidence intervals are shown on the graphs. . . . . 49
2.20 Impact of different multicasting strategies and number of multicast channels on average consumed bandwidth, measured with 40 delivery servers, 20 video channels and cache sizes of 2,560MB. Confidence intervals are shown on the graphs. . . . . 49

3.1 Graphical representation of variables and assumptions. . . . . 63

3.2 Network topology, modeling a typical video service delivery network. . . . 72
3.3 Impact of number of clients, using a topology with $k = 2$, $BW_{l-1} = 3Mbps$, $BF = 0.9$. . . . 74
3.4 Impact of number of bottlenecks in the topology on the average decision time, using a topology with $k = 5$, $BW_{l-1} = 2Mbps$, $|\mathcal{C}|= 125$ and $l = 4$. . . . 76
3.5 Impact of the *Switching Alpha* $\alpha_s$, using a topology with $k = 5$, $BW_{l-1} = 4Mbps$, $BF = 0.8$, $|\mathcal{C}|= 125$ and $l = log_k|\mathcal{C}| + 1 = 4$. 77
3.6 Impact of the *Round Trip Time (RTT) (ms)*, using a topology with $k = 5$, $BW_{l-1} = 3Mbps$, $BF = 0.8$, $|\mathcal{C}|= 125$ and $l = log_k|\mathcal{C}| + 1 = 4$. . . . 78
3.7 Network topology, modeling a typical video service delivery network with multiple servers. . . . 79
3.8 Impact of multiple servers with balanced load and the *Number of clients*, using a topology with $BW_{l-1} = 2Mbps$, $BF = 0.8$ and $|\mathcal{C}|= 200$. . . . 80
3.9 Impact of multiple servers with unbalanced load and the *Number of clients*, using a topology with $BW_{l-1} = 2Mbps$, $BF = 0.8$ and $|\mathcal{C}|= 200$. . . . 81

4.1 In-network quality management architecture, showing a centralized component gathering monitoring information, estimating residual bandwidth which serves as input for the QoE-driven quality optimization and quality selection guidance for the clients. Further on in the chapter, also a distributed version is considered. . . . 95
4.2 Distributed optimization process for a node $n$ gathering monitoring information from its upstream link $e_{n^-}$ to estimate $R_{e_{n^-}}$ and downstream restrictions $s_{n^+,c}$ from its successor node set $\mathcal{N}^+$. . . . 103
4.3 Network topology, modeling a typical video service delivery network. . . . 106
4.4 Example cross traffic trace. . . . 106
4.5 Graphical overview of three example sequences that were used to assess the end-user subjective quality perception. Gaps in (c) indicate the occurrence of buffer starvations. . . . 108
4.6 Impact of history size $|\mathcal{H}_c|$ ($RTT = 40ms$, $r = 100$, $\tau = 2s$, $B = 12s$, $N = 32$). These results show a local optimum of $|\mathcal{H}|_c$= 128. . . . 110
4.7 Impact of optimization interval $\tau$ ($RTT = 40ms$, $r = 100$, $|\mathcal{H}_c|$= 128, $B = 12s$, $N = 32$). Setting the optimization interval $\tau$ at the segment length of $2s$ yields the highest QoE at the cost of frequent optimization. . . . 110
4.8 Impact of sampling size $r$ on packet-based throughtput estimation for $r = 100$ showing a high correlation. . . . 111

4.9 Impact of sampling size $r$ ($RTT = 40ms$, $|\mathcal{H}_c|= 128$, $\tau = 2s$, $B = 12s$, $N = 32$). Increasing the sampling size $r$ beyond $10^3$ negatively impacts the QoE due to wrong estimations on the future throughput. Setting $r = 100$ gives good estimations at a limited sampling cost of $1\%$. . . . . 112
4.10 Impact of buffer size $B$ ($RTT = 40ms$, $r = 100$, $|\mathcal{H}_c|= 128$, $\tau = 2s$, $N = 32$). For a buffer of about $4s$, the in-network optimization is able to achieve a similar QoE as the best performing client-side heuristic using a buffer of four times that size. . . . . 113
4.11 Impact of forecasting method ($RTT = 40ms$, $r = 100$, $|\mathcal{H}_c|= 128$, $\tau = 2s$, $B = 12s$, $N = 32$) for multiple values of the sampling rate $r$. . . . . 115
4.12 Impact of access network bandwidth fluctuations ($RTT = 40ms$, $|\mathcal{H}_c|= 128$, $\tau = 2s$, $B = 12s$, $N = 32$). . . . . 117
4.13 Impact of total number of clients $|\mathcal{C}|$ for various combinations of network size parameters ($K = 32$, $M \in [32, 64, 128, 256]$ and $N \in [512, 1024, 2048, 4096]$) on the communication overhead for distributed ((a) and (b)) and centralized ((c) and (d)) in-network QoE optimization respectively. . . . . 119
4.14 Impact of increasing the number of nodes per level $M$ (RTT $= 40ms$, $r = 100$, $|\mathcal{H}_c|= 128$, $\tau = 2s$, $B = 12s$, $N = 16$). Since the calculation delay for the *Centralized* optimization is quite high, the practical results differ significantly from the optimal solution, showing the benefits of the more scalable *Distributed* optimization. 121
4.15 Impact of number of clients $|\mathcal{C}|$ for an uncongested scenario ($RTT = 40ms$, $r = 100$, $|\mathcal{H}_c|= 128$, $\tau = 2s$, $B = 12s$). Even in the absence of cross traffic, the client-side heuristics are not able to achieve a comparable quality as in-networks driven optimization due to the competition between clients. . . . . 122
4.16 Impact of number of homes $N$ for a congested scenario ($RTT = 40ms$, $r = 100$, $|\mathcal{H}_c|= 128$, $\tau = 2s$, $B = 12s$). The in-network optimization also suffers from the congested network ($N = 64$), but is able to maintain an average QoE that is $130\%$ higher compared to the client-side heuristics and is still acceptable since the Mean Opinion Score (MOS) is higher than 3. . . . . 123

5.1 Example run of impact of varying streaming origin on QoE achieved by base heuristic [42]. . . . . 140
5.2 State diagram for cache assisted quality selection heuristic. . . . . 142
5.3 Experimental setup, representing a delivery network with intermediary cache nodes. . . . . 148
5.4 Map showing different streaming locations of the dynamic scenario. 149
5.5 Excerpt of collected mobile delay traces using a 4G smartphone for a moving car (a) and for a stationary environment (b). . . . . 150

5.6 Impact of parameter $\tau$ of aged Exponentially Weighted Moving Average (EWMA) estimation on (a) average MOS, (b) average buffer starvation time, (c) average number of quality switches and (d) average played quality. . . . . 151
5.7 Impact of (a) the bottleneck bandwidth $BW_S$, (b) the delay between server and proxy $d_S$, (c) the cache size $C_P$ and the maximum buffer size $B$ on the average QoE. . . . . 153
5.8 Impact of (a) minimum number of elements in cluster $n_{min}$, (b) minimum distance between clusters $d_{min}$ and (c) maximum standard deviation within a cluster $\sigma^2$ on the clustering accuracy expressed as percentage of correctly classified elements. Impact of cache estimation parameter $\beta$ (d) on average MOS. . . . . 156
5.9 Impact of (a) the bottleneck bandwidth $BW_S$, (b) the delay between server and proxy $d_S$ and (c) the cache size $C_P$ on the average QoE. . . . . 158
5.10 QoE-improvement (a) and reduction of buffer starvations (b) in a dynamic scenario. . . . . 160

6.1 An example SFC. . . . . 178
6.2 An overview of the NFV architecture with support for semantic SFC request checking. . . . . 180
6.3 Graphical representation of ontology. . . . . 181
6.4 Graphical representation of the model. . . . . 185
6.5 Impact of the infrastructure size on the semantic validation. . . . . 195
6.6 Impact of the number of requested Virtual Network Functions (VNFs) on the semantic validation. . . . . 196
6.7 Impact of the number of constraints per SFC on the semantic validation. . . . . 196
6.8 Share of semantic matching and mapping on total execution time for SFC set mapping. . . . . 198
6.9 Share of semantic matching and mapping on total execution time for individual SFC mapping. . . . . 198
6.10 Semantic matching execution time. . . . . 199
6.11 Percentage of SFC requests that are mapped. . . . . 199
6.12 Total mapping time of Set mapping compared to Individual mapping for increasing infrastructure size. . . . . 200
6.13 Objective of Set mapping compared to Individual mapping for increasing infrastructure sizes. . . . . 201
6.14 Percentage of mapped SFCs when applying Set mapping compared to Individual mapping for increasing infrastructure sizes. . . 201
6.15 Total link bandwidth usage for various optimization objectives. . . 202
6.16 Difference between the maximum and minimum link usage for various optimization objectives. . . . . 203
6.17 Percentage of used nodes for various optimization objectives. . . . 203
6.18 Impact of SFC ordering on acceptance rate. . . . . 203

A.1 Illustration how a steeper slope prioritizes backfilling over prefetching. . . . 224
A.2 Illustration of the backfilling operation by SVC Cursor when the quality cursor was improved. . . . 225
A.3 Overview of estimation with pipelining a) accurate estimation b) overestimation c) underestimation. . . . 226
A.4 Illustration of the delay masking behavior of parallel scheduled segment layer downloads. . . . 227
A.5 Experimental setup offering a HAS-based video streaming to $N$ clients. The parameters $B_s$, $B_c$, $P$, $R$ are varied. . . . 228
A.6 Total buffer starvation (s), average played quality level and total number of switches in function of the buffer size $P$ (s) with $B_c = 10Mbps$, $N = 20$, $R = 50ms$ and (a) $B_s = 50Mbps$ (b) $B_s = 100Mbps$. . . . 229
A.7 Average played quality level and total number of switches in function of the buffer size $P$ (s) with $B_c = 10Mbps$, $N = 20$, $R = 50ms$ and (a) $B_s = 50Mbps$ (b) $B_s = 100Mbps$. . . . 230
A.8 Average played quality level in function of the RTT $R$ (s) for $B_s = 100Mbps$, $B_c = 10Mbps$, $N = 20$ and $P = 3.3s$. . . . 231
A.9 Impact of the number of parallel threads with $B_c = 10Mbps$, $N = 20$, $R = 100ms$, $P = 4.4s$ and (a) $B_s = 50Mbps$ (b) $B_s = 100Mbps$, for AVC MSS and SVC Cursor in combination with sequential and parallel scheduling. . . . 232

B.1 Illustration of layer-based prioritization, where lower quality presentations have higher priority than enhancement layers. . . . 242
B.2 Illustration of deadline-based prioritization, where clients decide based on their current buffer filling to request segments with higher priority. . . . 243
B.3 Click-implementation of AF behavior. . . . 244
B.4 Experimental setup showing the Per Hop Behavior (PHB) and introduction of best effort cross traffic on the bottleneck link. . . . 245
B.5 Excerpt of a cross traffic file for a 20Mbps link. . . . 246
B.6 Illustration of the continuous playout under cross traffic when SVC layer-based prioritization is enabled. With segment length of 1s and buffer size 2s (a) and 20s (b) for AVC MSS and buffer size 2s for SVC layer prioritization for a single streaming client. . . . 247
B.7 Impact of the number of parallel DiffServ downloads on the SVC Layer Prioritization for a buffer of 4 seconds and a prioritized channel of 100Mbps. . . . 248
B.8 Impact of the buffer size on average freezing time, estimated MOS and number of switches for a prioritized channel of (a) 150Mbps and (b) 200Mbps respectively. . . . 250

B.9 Impact of the number of allowed qualities for Deadline Based Prioritization on average buffer filling, freezing time, quality rate and number of switches for a prioritized channel of 200Mbps and buffer size of 4 seconds. . . . . 251

C.1 Open Virtualization Format (OVF) specification extension for modelling Affinity node and link constraints. . . . . 263
C.2 An example SFC. . . . . 265
C.3 A simplified constraint specification for the example SFC . . . . . 265
C.4 An overview of the NFV architecture with support for semantic SFC request checking. . . . . 266
C.5 Graphical representation of ontology. . . . . 267
C.6 Impact of physical network size. . . . . 272
C.7 Impact of virtual network size. . . . . 273
C.8 Impact of number of constraints on number of consistent SFC execution time. . . . . 274
C.9 Combined impact of increasing physical network size, requested virtual network size and number of constraints. . . . . 274

# List of Tables


3.1 Variables used for the rate decision. . . . . 64
3.2 Overview of the quality layers for the Big Buck Bunny video. . . . 72

4.1 Overview of the quality layers for the Big Buck Bunny video. . . . 107
4.2 Pearson correlation between the estimated throughput for the next interval $\tau$ and the actual cross traffic rate. A sampling size of $r = 100$ yields a high Pearson correlation of $\rho = 0.965$. . . . . 111
4.3 Pearson correlation and standard deviation for different forecasting techniques for a sampling size of $r = 100$. . . . . 115

5.1 Average one way delays (ms) and variance for fixed connections between the remote locations. . . . . 150
5.2 Probability of predicting a segment to be in the cache for $\beta = 0.75$ as a function of the buffer level $B$. . . . . 157

6.1 VNF placement problem notation. . . . . 184
6.2 Scenario parameters. . . . . 194

B.1 Bitrates of the different video layers. . . . . 246

C.1 Overview of the evaluation parameters. . . . . 271
C.2 Impact of number of constraints on number of consistent SFC requests. . . . . 273

# List of Acronyms

## A

**AF** Assured Forwarding
**AR** Autoregression
**AS** Autonomous System
**AVC** Advanced Video Coding

## B

**BIP** Binary Integer Programming
**BWA** Broadband Wireless Access

## C

**CAPEX** Capital Expenditure
**CDN** Content Delivery Network

## D

**DASH** Dynamic Adaptive Streaming over HTTP
**DC** Datacenter
**DiffServ** Differentiated Services
**DPI** Deep Packet Inspection
**DRM** Digital Rights Management

## E

**EF** Expedited Forwarding
**EPC** Evolved Packet Core
**ET** Exponential Trend
**ETSI** European Telecommunications Standards Institute
**EWMA** Exponentially Weighted Moving Average

## F

**FDD** Frequency Division Duplex
**FDMA** Frequency Division Multiple Access

## G

**GSM** Global System for Mobile Communications

## H

**HAS** HTTP Adaptive Streaming
**HEVC** High Efficiency Video Coding
**HG** Home Gateway
**HW** Holt Winters

## I

**ILP** Integer Linear Programming
**InP** Infrastructure Provider
**IPTV** Internet Protocol television
**ISP** Internet Service Provider
**IX** Internet Exchange

## K

**kbps** kilobit per second

## L

**LP** Linear Programming

**LRU** Least Recently Used

## M

**MINLP** Mixed-Integer Non-Linear Programming

**MIQCP** Mixed Integer Quadratically Constrained Programming

**MLP** Multi Layer Perceptron

**MOS** Mean Opinion Score

**MPEG** Moving Pictures Experts Group

**MPEG-TS** Moving Pictures Experts Group (MPEG) Transport Stream

**MSS** Microsoft Smooth Streaming

## N

**NFV** Network Function Virtualization

**NF** Network Function

## O

**OPEX** Operational Expenditure

**OTT** Over-The-Top

**OVF** Open Virtualization Format

**OWL** Web Ontology Language

# P

**P2P** Peer to Peer
**PCN** Pre-Congestion Notification
**PHB** Per Hop Behavior
**PoP** Point of Presence
**PSNR** Peak Signal-to-noise Ratio
**PUE** Power Usage Efficiency

# Q

**QoE** Quality of Experience
**QoS** Quality of Service
**QP** Quantization Parameter

# R

**RED** Random Early Detection
**RMSE** Root Mean Squared Error
**RTP** Real Time Transport Protocol
**RTSP** Real Time Streaming Protocol
**RTT** Round Trip Time

# S

**SDN** Software Defined Networking
**SFC** Service Function Chain
**SLA** Service Level Agreement
**SMO** Sequential Minimal Optimization
**SNMP** Simple Network Management Protocol
**SP** Service Provider
**SSIM** Structural Similarity

**STB** Set-Top Box
**SVC** Scalable Video Coding
**SVR** Support Vector Regression
**SWRL** Semantic Web Rule Language

## T

**TCO** Total Cost of Ownership
**TE** Traffic Engineering
**TDMA** Time Division Multiple Access
**TSTV** Time Shifted TV

## U

**UDP** User Datagram Protocol
**UMTS** Universal Mobile Telecommunications System

## V

**VBR** Variable Bitrate
**vCDN** virtual Content Delivery Network (CDN)
**vCPE** virtualized Customer Premises Equipment
**VDCNE** Virtual Data Center Network Embedding
**vEPC** virtualized Evolved Packet Core
**VM** Virtual Machine
**VN** Virtual Network
**VNE** Virtual Network Embedding
**VNF** Virtual Network Function
**VNFInP** Virtual Network Function Infrastructure Provider
**VoD** Video on Demand
**VoIP** Voice over IP

# W

**WAN** Wide Area Network

**WWW** World Wide Web

# Samenvatting
# – Summary in Dutch –

In de afgelopen decennia is het internet geëvolueerd van een netwerk dat werd ontworpen voor het verzenden van pakketten tot een aanbieder van een breed scala aan geavanceerde multimediadiensten. Niet alleen de aangeboden diensten zijn geëvolueerd, maar ook de toestellen en technologieën die worden gebruikt om er toegang tot te krijgen zijn drastisch veranderd. Hierdoor kunnen de eindgebruikers hun favoriete services consumeren, gebruikmakend van ieder toestel, geconnecteerd via diverse technologieën en op een locatie van hun keuze. Eén van de diensten die het huidige internetverkeer domineert zijn videodiensten. Het aandeel van videotraffiek in het totale internetverkeer wordt verwacht om de grens van $80\%$ te overschrijden tegen 2019. Een belangrijke factor voor het success van een videodienst is de kwaliteit zoals die wordt ervaren door de eindgebruiker, aangeduid als Quality of Experience (QoE). In tegenstelling tot traditionele Internet Protocol television (IPTV)-diensten, worden de Over-The-Top (OTT) videodiensten (zoals Netflix, YouTube) zonder garanties omtrent de QoE afgeleverd over het internet. In de loop der jaren zijn ook de technologieën voor het afleveren van video over het internet sterk geëvolueerd. Real Time Streaming Protocol (RTSP) en Real Time Transport Protocol (RTP)-gebaseerde video streaming over UDP werd vervangen door HTTP-gebaseerde methoden, afgeleverd via TCP. De gegarandeerde overdracht van de videopakketten, de naadloze integratie en compatibiliteit met firewalls en NAT-technologieën en het hergebruik van de bestaande caching infrastructuur zijn enkele van de voordelen van HTTP-gebaseerde streaming. Om dynamisch en op een schaalbare manier te kunnen reageren op veranderende netwerkcondities, zijn kwaliteitsadaptatietechnieken aan de cliënt-zijde reeds de facto standaard geworden in de commerciële streaming oplossingen.

In HTTP Adaptive Streaming (HAS), wordt de video temporeel opgesplitst in segmenten die worden geëncodeerd volgens verschillende kwaliteitsniveaus. Hierdoor kan de streaming applicatie autonoom beslissen welke kwaliteit wordt aangevraagd op basis van de gemeten bandbreedte, de huidige buffervulling en de specificaties van het toestel. Dit stelt de HAS-applicaties ertoe in staat om te reageren op fluctuaties in het netwerk door het aangevraagde kwaliteitsniveau te wijzigen. Bij niet-adaptieve HTTP-gebaseerde technologieën zou de buffer uitgeput raken en uiteindelijk een onderbreking in de videoweergave veroorzaken. Dankzij de gegarandeerde overdracht via HTTP kan een segment steeds worden afgespeeld aan de gewenste kwaliteit, waardoor pakketverlies geen artefacten kan

veroorzaken in de video. Aangezien echter de kwaliteit wel dynamisch kan worden aangepast, kunnen oscillaties in het kwaliteitsniveau optreden, of erger nog, een onderbreking van de weergave. De QoE van de eindgebruiker zal sterk worden beïnvloed door dergelijke gebeurtenissen. In het verleden werd het netwerk sterk overgedimensioneerd om deze QoE degradatie te voorkomen. Aangezien de kwaliteitseisen van de gebruikers steeds toenemen en ook de populariteit van videodiensten dit zal blijven doen, zal overdimensionering van het netwerk niet langer economisch rendabel zijn. Daarom moeten de bestaande middelen efficiënter worden aangewend door de aflevering van de video intelligenter te beheren. Tegelijkertijd moet ook het netwerk flexibeler worden om te blijven voldoen aan de verhoogde kwaliteitseisen en nieuwe technologieën te blijven ondersteunen. In deze thesis worden verschillende technieken voorgesteld voor het beheren van de QoE bij HAS-diensten, zowel voor live als Video on Demand (VoD) streaming. Bovendien worden uitbreidingen voorgesteld op bestaande Network Function Virtualization (NFV) aanpakken om de flexibele plaatsing van HAS services toe te laten.

Een eerste manier om dit te bewerkstelligen, is om de druk te verminderen die wordt uitgeoefend door live HAS diensten op het onderliggende netwerk, door het toepassen van multicast aflevering. In huidige HAS oplossingen is voor elke gebruiker een unicast overdracht van de segmenten vereist. Gezien in een live en Time Shifted TV (TSTV) scenario heel wat segmenten meerdere keren worden verstuurd langs hetzelfde pad in een korte tijdspanne, lenen multicast technieken zich perfect tot het optimaliseren van de aflevering van dergelijke diensten. Bij binnenkomst in het beheerde netwerk, worden meerdere overlappende sessies gegroepeerd tot een multicast kanaal. Verderop in het netwerk, dichter bij de eindgebruiker, worden deze multicast sessies opgesplitst en worden de oorspronkelijke HAS unicast sessies gereconstrueerd. Door het toepassen van intelligente algoritmes om te bepalen welke sessies worden gegroepeerd en op welke multicast kanalen wordt ingeschreven, kan de druk op het netwerk aanzienlijk worden verminderd. Daarnaast kan de gebruikservaring ook worden verbeterd door de tijd tussen het live moment en de eigenlijke weergave bij de gebruiker drastisch te verminderen. Huidige HAS-technologieën vereisen een grote buffer aan de cliëntzijde om eventuele schommelingen in het netwerk te kunnen opvangen teneinde kwaliteitsoscillaties en onderbrekingen te vermijden. Door het toepassen van prioritisatie in het netwerk, kan de aflevering van bepaalde segmenten worden gegarandeerd. Hierdoor wordt het mogelijk om de buffer en dus ook de verstreken tijd tussen het live moment en de weergave aanzienlijk in te korten.

Het afleveren van HAS VoD-diensten kan worden geoptimaliseerd door de negatieve impact van concurrerende HAS-applicaties te verminderen. Huidige HAS-oplossingen bieden geen mogelijkheid tot coördinatie tussen de verschillende adaptatie algoritmes, waardoor ze sterk onderhevig zijn aan ON/OFF-patronen. Wanneer de buffer van een cliënt voldoende gevuld is, wordt het downloadproces tijdelijk onderbroken, dit wordt de OFF-toestand genoemd. Andere applicaties interpreteren deze tijdelijke toename in beschikbare bandbreedte foutief en trachten de kwaliteit te verhogen. Echter, wanneer de OFF-clients het downloaden her-

vatten, wordt de downloadtijd sterk verhoogd en treden er kwaliteitsoscillaties en zelfs onderbrekingen in de videoweergave op. Door het toevoegen van coördinatie vanuit het netwerk kunnen de negatieve effecten van deze ON/OFF-patronen worden opgelost. Door het aantal beschikbare kwaliteiten per streaming toepassing te beperken, kan de algemene QoE worden verbeterd, terwijl de autonome kwaliteitsadaptatie nog steeds toelaat om te reageren op fluctuaties in het lokale netwerk.

Een derde manier om de QoE te verbeteren, is om de robuustheid te verhogen tegen fluctuaties veroorzaakt door intermediaire caches in het netwerk. Deze zorgen dan wel voor een schaalbare aflevering van de video, maar beïnvloeden tergelijkertijd ook de kwaliteitsaanpassing. Aangezien segmenten zowel van de server als van een intermediaire cache kunnen worden verstuurd, kan dit schommelingen in de gemeten bandbreedte en netwerkvertraging veroorzaken. Deze verkeerde schattingen kunnen op hun beurt dan weer leiden tot foute kwaliteitsselectie en uiteindelijk een reductie van de QoE. Door het includeren van de informatie omtrent de herkomst en de inhoud van de caches, kan een accuratere inschatting worden gemaakt van de haalbare kwaliteit. Op deze manier kunnen oscillaties in de kwaliteit en onderbrekingen in de videoweergave worden vermeden. Aangezien deze informatie niet steeds beschikbaar is, worden ook clustering-gebaseerde technieken voorgesteld om de herkomst en inhoud van de caches autonoom te detecteren.

Om de flexibiliteit waarmee nieuwe diensten kunnen worden aangeboden te verhogen, kunnen de principes van NFV worden toegepast. De verschillende Network Functions (NFs) (bv. caches, prioritisatie functies) kunnen worden gevirtualiseerd en dynamisch geplaatst worden op de gevirtualiseerde infrastructuur. Dit zorgt ervoor dat de beschikbare middelen kunnen worden verdeeld over verschillende diensten heen. Verder wordt de sterke koppeling tussen de functionaliteit en de onderliggende apparatuur aanzienlijk verminderd. Hierdoor kunnen dynamisch nieuwe en innovatieve diensten worden opgezet. Echter, de huidige NFV technologieën bieden slechts beperkte ondersteuning voor het toevoegen van locatierestricties voor deze services. Voor multimediadiensten is dit echter gewenst omwille van efficiëntie (bv. caches dicht bij eindgebruikers), wettelijke (bv. data mag enkel via bepaalde regio's worden verstuurd) of economische redenen (bv. overeenkomsten met providers). Daarom werd een set van affiniteits en anti-affiniteits restricties gedefinieerd die de plaatsing van dergelijke diensten kan beïnvloeden. Een semantisch raamwerk werd ontwikkeld voor het valideren van groepen van deze restricties en het optimaliseren van de plaatsing ervan.

De verschillende oplossingen voor het QoE-beheer van HAS-diensten in combinatie met de toegenomen flexibiliteit voor het plaatsen van dergelijke diensen die tijdens dit proefschrift werden ontwikkeld, richten zich op enkele belangrijke uitdagingen in het veld van HAS. Toekomstig onderzoek kan hier verder op bouwen door bijvoorbeeld intelligente caching technieken te ontwikkelen die de informatie inzake de temporele structuur en kwaliteitsaanpassingen aanwenden om de inhoud van de cache samen te stellen. Ook de inzet van dynamische multimediadiensten kan nog verder worden onderzocht. De verschillende effecten van QoE-beheersmethodes dienen te worden gemodelleerd teneinde een geautomatiseerde aanpassing van de diensten toe te laten.

# Summary

Over the past decades, the Internet has evolved from a network designed for transferring packets to a provider of a wide range of advanced multimedia services. Not only the services offered through the Internet have evolved, but also the devices and technologies that are used to access them have changed. This allows end users to access their favorite services through the Internet using any device via different access technologies and at a location of their choice. One of the services that is dominating the Internet traffic are video streaming services. Key towards a successful streaming solution is the quality experienced by the end-user, commonly denoted as Quality of Experience (QoE). In contrast to Internet Protocol television (IPTV) services, Over-The-Top (OTT) video streaming services (such as Netflix, YouTube) are provided over the best effort Internet, and as such unable to provide any guarantees on the delivered QoE. Over the years, the streaming technologies that support the delivery of OTT video have also evolved. Real Time Streaming Protocol (RTSP) and Real Time Transport Protocol (RTP)-based streaming over UDP were replaced by HTTP-based streaming methods over TCP. The reliable transmission of the video, the seamless integration and compatibility with firewalls and NATs and the reuse of existing Internet caching infrastructure are some of the advantages offered by HTTP streaming. To be able to adapt in a scalable way to dynamic network conditions, client-side rate adaptation schemes are becoming the de facto standard in commercial streaming solutions.

In HTTP Adaptive Streaming (HAS), the video is temporally split into segments which are encoded at different quality rates and stored at the server. This allows the client side heuristic to switch between quality representations based on measured network statistics, buffer filling level and device characteristics. This enables the HAS streaming clients to respond to throughput fluctuations by reducing the quality and providing a continuous video playout, whereas non-adaptive HTTP-based streaming methods would have suffered from buffer starvations. Due to the reliable transmission over HTTP, a segment is always displayed as it was encoded, since no video artefacts can be caused by packet loss. However, since the quality can be adapted, quality oscillations can occur and if a wrong decision is made by the adaptation heuristic, a buffer starvation can lead to a frame freeze. The QoE perceived by the end user can be heavily impacted by such events. Historically, network providers have been over-provisioning the network resources to avoid QoE degradation. However, since the quality expectations of the users are increasing rapidly and also the popularity of streaming services is growing, over-provisioning the network is no longer economically viable. Therefore, the exist-

ing resources need to be used more efficiently by managing the delivery of video streaming services. At the same time, to meet the demands and requirements of new technologies, the network should be made more flexible to be able to support new and innovative services with short time to market. In this dissertation a number of QoE management techniques for HAS are proposed, both for live and Video on Demand (VoD) streaming. Furthermore, extensions to current Network Function Virtualization (NFV) approaches are proposed to allow the flexible and location-aware deployment of HAS services.

A first way to reduce the strain of live HAS services on the underlying network, is to deploy a multicast-enabled delivery framework. In state-of-the-art HAS solutions, each client application requires a unicast transmission of every segment. Since in live and Time Shifted TV (TSTV) streaming, a lot of segments are transferred multiple times across the same path, applying multicast techniques can reduce the network resource usage. At the ingress, multiple overlapping streaming sessions are grouped onto a multicast channel. Further down the network, an egress node splits the multicast session and reconstructs the original HAS unicast sessions. By deploying intelligent algorithms to select which sessions to aggregate and to which multicast channels the network nodes should subscribe, the resource usage can be significantly reduced. Additionally, the user experience can be improved for live HAS services by reducing the buffer size and thus shortening the camera-to-display delay. Current HAS adaptation schemes require a sufficiently large client-side buffer to protect them from quality oscillations and buffer starvations. By deploying in-network prioritization of segments, the assured delivery of these segments allows to shrink the buffer size to several seconds. Furthermore, adapting the request scheduling to take into account high network Round Trip Times (RTTs) can also benefit the overall QoE of a live streaming session.

Another way to optimize the delivery of HAS services involves reducing the negative impact of competing HAS clients on the overall QoE. In state-of-the-art HAS solutions, there is no coordination between the different client applications and they suffer from instabilities caused by ON-OFF patterns. When the buffer of a client is sufficiently filled, the download process is paused and identified as the OFF-state. Other clients can misinterpret the temporary increase in throughput as an increase in available resources, causing them to increase their requested quality. However, when the OFF-clients resume their download process, the download times are increased, causing the buffer to deplete faster than anticipated and leading to quality oscillations and even buffer starvations. By adding in-network coordination, the negative effects of these ON-OFF patterns can be mitigated. The in-network quality optimization nodes measure the available throughput and allocate the resources in order to optimize the QoE. By limiting the set of available qualities for each client, the overall QoE can be improved, while the autonomous adaptation at the clients can still react to sudden fluctuations in the local network resources.

A third way to improve the QoE of HAS services is to increase the robustness of client adaptation heuristics against throughput fluctuations caused by in-network caching nodes. These nodes reduce the resource usage of HAS VoD ser-

vices, but at the same time interfere with the quality adaptation. Since segments served from the cache are typically served with a lower network delay and higher throughput, they can cause the throughput estimations to be too high. These optimistic estimations lead to increased quality oscillations, thus reducing the QoE. By considering the information on the streaming origin, the accuracy of the different throughput estimations per origin can be improved. Furthermore, taking into account the contents of the intermediary caches can reduce the risk of quality oscillations and improve the overall stream quality. Since this information is not always available or intermediary caches cannot be adapted, clustering-based techniques are proposed to autonomously detect the streaming origin at the client. Furthermore, by deploying probability-based cache content estimation techniques, the number of buffer starvations and quality oscillations can be reduced significantly.

To increase the flexibility with which new services can be deployed, the NFV paradigm can be applied to streaming service chains. The different Network Functions (NFs) (e.g., caches, prioritization functions) are virtualized and deployed on top of a virtualized infrastructure. This allows to share the network resources across different services and reduces the tight coupling of the functionality of NFs with the physical resources upon which they are deployed. To deploy new services, the Virtual Network Functions (VNFs) can be deployed onto the NFV infrastructure and linked using programmable network technologies to support the required Service Function Chain (SFC). However, current NFV technologies offer only limited support for adding location constraints to SFC requests. For multimedia services however, the Service Provider (SP) would want to add location constraints for efficiency reasons (e.g., caches close to end users), legislative reasons (e.g., certain content can only be routed through specific regions) or economic reasons (e.g., partner agreements with certain providers). Therefore a set of affinity and anti-affinity constraints are proposed that can steer the deployment of such services. A semantic framework is proposed to check the validity of such constrained SFCs in order to improve the performance of mapping the VNFs and interconnections to the physical network resources.

The different QoE-management extensions to HAS streaming services, together with the increased flexibility when mapping these services onto the virtualized infrastructure, address some of the major challenges for flexible HAS streaming that were identified during the dissertation. Future research can extend this work both in the field of QoE-management of HAS services and by improving the flexible deployment of multimedia services. Other HAS aspects that could be improved are the development of HAS-aware caching techniques that take into account both the temporal and adaptation characteristics of HAS streaming during the cache replacement. Increased cooperation between the client adaptation schemes and the caching mechanisms could potentially improve the QoE even further. Also, the dynamic deployment of multimedia services can be further investigated. The various aspects of the SFCs and their respective impact on QoE could be modeled in order to achieve an automated adaptation of the proposed SFCs to the current resource availability or service usage.

# 1 Introduction

*"The measure of intelligence is the ability to change."*

–Albert Einstein (1879 - 1955)

## 1.1 The Rise of Internet Video Streaming

In October 1969, the first ARPANET link was established between the University of California, Los Angeles and the Stanford Research Institute [1]. Albeit the system crashed after sending the first two characters of the word *login*, the Internet revolution was started that day. A few weeks later, the network had grown to a 4-node network and by 1981, the number of hosts had grown to 213, with new hosts being added approximately every 20 days. In 1989, Tim Berners-Lee introduced the World Wide Web (WWW) and the first web browser in 1990, laying the foundations of the Internet as it is known today [2, 3]. Since then, the Internet became a platform for a multitude of services, such as online shopping and auction sites (e.g., Amazon, eBay), file sharing systems (e.g., Napster, BitTorrent), social networks (e.g., LinkedIn, Facebook), Internet communication services (e.g., Hotmail, Skype), etc. In 2015, the ITU reported 3.2 billion of Internet users, which are interconnected via the Internet through roughly 15 billion devices. Not only the services offered via the Internet have evolved, but also the devices and technologies that are used to access them have changed. In 1969, a desktop terminal was used to transfer the first bytes over a fixed line. Today, many devices such

as laptops, game consoles, media centers, smartphones and tablets can be used to access the Internet using wireless connections, both at home and outside.

The first video streaming event took place in 1993, when a live performance of the band *Severe Tire Damage* was broadcasted to test Internet based live streaming developed at Xerox PARC. In 1995, the Moving Pictures Experts Group (MPEG) released MPEG Transport Stream (MPEG-TS), a standardized container format for the transmission and storage of audio, video and metadata which is used in broadcast systems such as Internet Protocol television (IPTV). This allowed video to be broadcasted over packet networks. Early 1996, Microsoft released the ActiveMovie media player which allowed to stream video using a proprietary streaming format. In 1997, RealPlayer released their answer to Microsoft's application and in 1999 Apple created their own streaming format and player called Quicktime 4. All formats were adopted by different websites and each required to download the proprietary applications, causing the users to end up with multiple streaming applications on their devices. The lack of a unified format led to the wide adoption of the Adobe Flash streaming technology in 2002. Despite the available applications, the popularity of video streaming was quite limited due to the low Internet connection speeds at that time. It remained this way until 2005, when Internet video streaming really took off with the launch of Youtube. This service allowed users to generate their personal video content and share it globally, but also offered support for the live streaming of events. The increased popularity of YouTube attracted other players, such as content providers and broadcasting companies, to hop on the Internet streaming train. Netflix, which originally started as a DVD-rental service, launched their Video on Demand (VoD) streaming service in February 2007 and has since grown to be a global provider of streaming services with over 81 million subscribers worldwide. While YouTube typically offers small clips of user-generated content, services such as Netflix and Hulu offer full movies and series in high quality, gradually replacing linear TV. Between 2009 and 2011, innovation in handheld technologies has led to the development of mobile video applications by some of the major players (e.g., Netflix, Hulu, YouTube). It is clear that over the past decades, the importance of video streaming services increased considerably and continues to increase. Figure 1.1 shows the Cisco Visual Networking Index forecast of May 2015. The growth of video streaming traffic is projected to exceed $80\%$ of the Internet traffic by 2019, demonstrating that video streaming will dominate the Internet [4].

The streaming technologies that support the delivery of Over-The-Top (OTT) video have also evolved. In the past, the Real Time Streaming Protocol (RTSP) and Real Time Transport Protocol (RTP) were used to deliver video over IP networks through UDP. Since these protocols require server-side bit-rate adaptation schemes, they are not suited to cope with highly heterogeneous and dynamically changing network conditions and lack the scalability that is required for such sys-

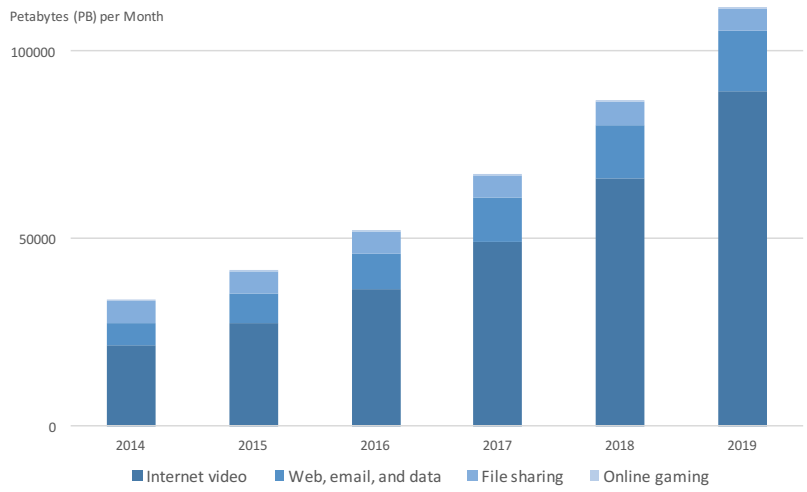



*Figure 1.1: Forecast of Internet data traffic per month for the next years. Video traffic is expected to exceed* 80% *of all Internet traffic by 2019 [4].*

tems. Today, the majority of the video streaming traffic is delivered using HTTP over TCP. The popularity of these HTTP-based streaming services was mainly induced by the advantages offered by HTTP streaming: the reuse of the Internet caching infrastructure, the reliable transmission over HTTP and the compatibility with firewalls and NAT-traversal. Furthermore, to increase the scalability of streaming services and to cope with dynamic network conditions, research and academia began shifting towards client-side adaptation schemes. HTTP Adaptive Streaming (HAS) is now becoming the de facto standard for video streaming delivery. In HAS, the video content is split temporally into segments which are encoded at different quality rates. The client-side intelligence decides at which quality rate each segment should be downloaded, based on measured network statistics, buffer filling level and device characteristics (e.g., screen resolution, computational abilities). This allows HAS adaptation schemes to respond to throughput fluctuations by reducing the quality and continuing video playout, whereas non-adaptive HTTP-based streaming techniques would have run into a buffer starvation. Furthermore, it allows the client to independently choose its playback quality and eliminates the server-side rate adaptation, which is a major advantage in large-scale OTT scenarios. The increased popularity of video consumption over the Internet has led to the development of a range of protocols that allow adaptive video streaming over HTTP. Some of the major industrial players have introduced

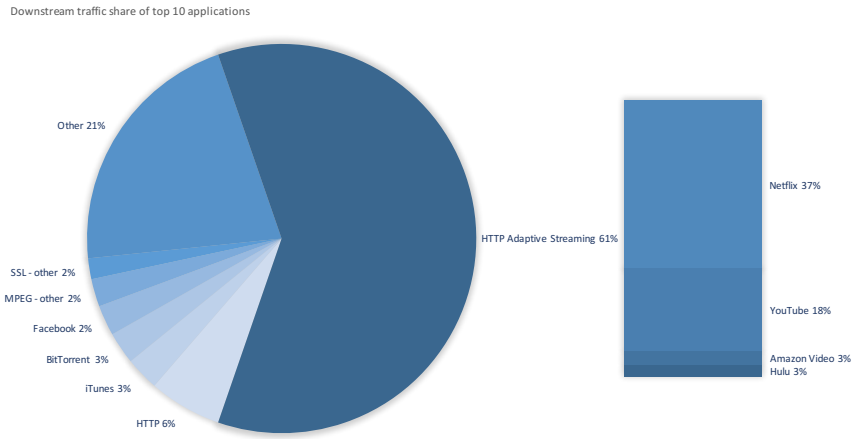



*Figure 1.2: HAS traffic exceeded* $61\%$ *of peak downstream Internet traffic in March 2016 [9].*

their proprietary protocols such as Microsoft's Silverlight Smooth Streaming [5], Apple's HTTP Live Streaming [6] and Adobe's HTTP Dynamic Streaming [7]. Furthermore, a standardized solution has been proposed by MPEG, called Dynamic Adaptive Streaming over HTTP (DASH) [8]. Sandvine reported the HAS share to exceed $61\%$ of the total peak downstream Internet traffic in March 2016 as shown in Figure 1.2 [9].

The aforementioned Internet video streaming services can be categorized as OTT services, as they are delivered using the best-effort Internet. Another category of video services are IPTV services. IPTV services are offered as part of the Triple Play service (telephony, Internet access and IPTV) by a provider over their managed network. In contrast to IPTV services however, OTT providers offer their services over the best effort Internet and therefore do not provide any delivery guarantees to their end-users. IPTV services typically deliver the broadcast channels over IP based networks and in surplus offer a limited catalogue of VoD content to their subscribers. IPTV services are typically offered through the set-top-box at a fixed location in the user's home. Compared to IPTV services, OTT streaming services offer a higher flexibility to their customers, as well as a larger content catalogue. Users can stream any video at any time and at any location using a device of their choosing. As such, the functionality offered by OTT services is more and more evolving into an interesting alternative for managed IPTV services. Therefore, network providers are interested in deploying OTT technologies to offer additional services that can be consumed from various devices outside the customer premises (e.g., YeloTV, TV Overal) and even collaborating with OTT

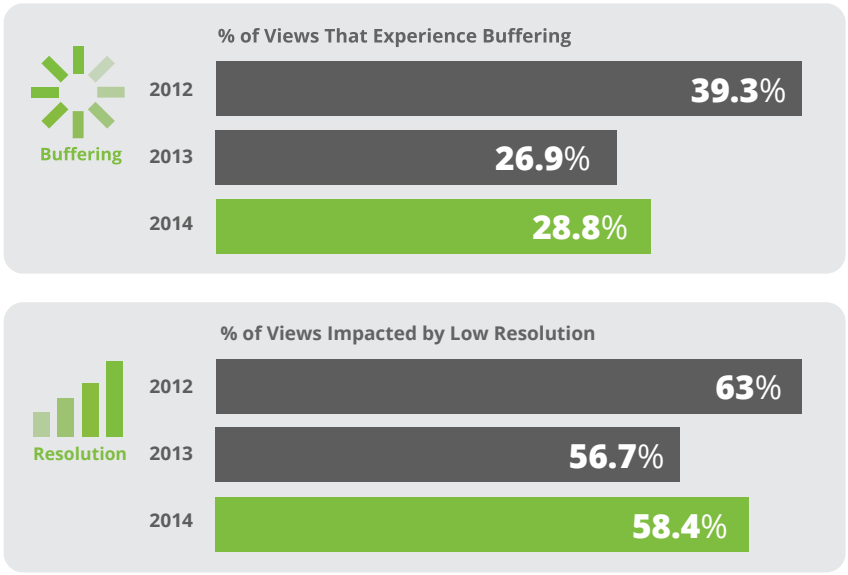



*Figure 1.3: Viewer experience report by Conviva, indicating that buffer starvations and quality degradations are still omnipresent [10].*

providers to optimize the delivery of their services (e.g., Netflix OpenConnect). Managing the delivery of these OTT-services can potentially increase their quality to comparable levels as traditional managed video services. However, a recent consumer survey by Conviva indicated that this is not yet the case, since buffer starvations and quality degradations are still omnipresent for current OTT HAS services, as is shown in Figure 1.3.

A video delivery service can be considered as a chain of Network Functions (NFs) (e.g., caches, load balancers) that are interconnected via the network and provide the streaming service. In traditional telecommunications networks, network functionality is strongly tied to the physical network device it runs on. To create or adapt network services, the network operator needs to deploy a dedicated network appliance for each NF. Furthermore, the placement of the NFs has to adhere to a strict chaining order, which increases the tight coupling of the service with the underlaying network topology. Together with the ever increasing requirements for high quality and stability, this has led to long product cycles, limited service agility and substantial dependence on specialized hardware. In order to compete with highly agile OTT services, which typically have much shorter product development cycles, and at the same time reduce the Capital Expenditures (CAPEX) and Operational Expenditures (OPEX) involved with physical network expansions, network operators needed to devise novel and less expensive

ways to meet the increased capacity requirements and at the same time reduce the time to market of newly developed services. The Network Function Virtualization (NFV)-paradigm [11, 12] has been introduced to alleviate the aforementioned issues by leveraging IT virtualization technology to decouple the network functionality from the physical infrastructure. It allows Virtual Network Functions (VNFs) to be deployed on standard high volume servers, storage devices and switches. The advantages are manyfold. First, there potentially is a significant cost reduction through more efficient maintenance, which can be performed remotely. In addition, thanks to the increased flexibility offered by virtualization, resources can be shared and used more efficiently. Finally, NFV allows network operators to deploy novel services cheaper and faster with higher service agility.

## 1.2 Problem Statement

Key towards a successful streaming solution is the quality experienced by the end-user, commonly denoted as Quality of Experience (QoE). ITU-T defines QoE as *"The overall acceptability of an application or service, as perceived subjectively by the end-user"* and indicates that it *"includes the complete end-to-end system effects (client, terminal, network, services infrastructure, etc.)"* [13]. Since imaging techniques are continuously improving (e.g., 4K encoding) and user devices are rapidly following (e.g., 4K smartphone screens), the quality expectations of the users are also increasing. However, the network in between the content producer and consumer, supporting the OTT video streaming solutions, is not evolving at such a rapid pace and, as discussed in the previous section, even supports less quality reservations than in the past with managed IPTV. As streaming services gain in popularity, the strain they exert on the underlying network resources grows rapidly. With its release in Belgium, Netflix already accounted for $10\%$ of the Internet traffic during the first weekend[1]. To provide streaming services at acceptable QoE, network providers have grossly over-provisioned the current Internet. But with the significant growth in popularity of video services and the rapid developments in multimedia technologies and bandwidth requirements, over-provisioning the network will no longer be economically viable in the future. Therefore, the existing resources need to be used more efficiently by managing the delivery of video streaming services, while at the same time, taking into account the delivered QoE. Furthermore, to be able to meet the demands of new technologies, the network should be made more flexible to be able to support new and innovative services with a short time to market. More specifically the following problems are tackled in this dissertation:

[1] Knack - http://datanews.knack.be/ict/nieuws/telenet-zag-dit-weekend-10-procent-van-zijn-internetverkeer-naar-netflix-gaan/article-normal-431203.html

**Existing HAS technologies for delivering live video streaming services need to be improved.** To cope with fluctuations in the network, HAS solutions apply large client-side buffers. This reduces the risk of running into a buffer starvation and having a forceful interruption of the video playout. However, at the same time, this introduces large camera-to-display delays for the end users. These should be as small as possible for a live streaming service if it is expected to achieve comparable QoE to traditional broadcasting services. The client-side buffers should thus be reduced to shrink the camera-to-display delay without losing in QoE. Furthermore, since in HAS, a dedicated connection is set up for each client, live streaming of large events can exert a lot of strain on the network resources, reducing the scalability. Since for live and Time Shifted TV (TSTV) streaming there is an overlap in the streamed data for each user, this is highly inefficient. Another problem with existing HAS solutions, is the susceptibility to high network delays. Since the segments are requested in a sequential order, high Round Trip Times (RTTs) lead to high idle times between consecutive downloads, reducing the efficiency. In HAS, video segments are delivered over TCP and thus adopt the fair share paradigm. However, suppose a client is close to buffer starvation and competes for bandwidth with another client having a full buffer. Avoiding a frame freeze for the first client, at the cost of lower throughput for second client can greatly improve the QoE of the first client, while having no or limited impact on the QoE of the second client. This example shows that fairness in terms of perceived QoE does not always correspond to fairness in terms of bandwidth consumption.

**Current HAS VoD streaming techniques still present some flaws. When multiple clients compete for the same bottleneck bandwidth, the adaptation heuristics suffer from instabilities due to ON-OFF patterns**. When the buffer of a client is sufficiently filled, the download is paused, freeing the bottleneck link. Other clients measure an increased throughput during this OFF-period and try to increase their quality. However, when the OFF-clients resume downloading, the download time is increased, causing the buffer to deplete and thus leading to quality fluctuations and even buffer starvations. This significantly reduces the QoE of the video service. The lack of coordination between the client heuristics renders it impossible to guarantee the QoE and thus limits the applicability of HAS solutions in managed networks. Furthermore, caching solutions that are deployed to reduce the strain of HAS services on the network can also intervene with the quality selection heuristics at the clients. Segments can be served both from the cache and from the origin server, without the client being able to distinguish between both. This again can lead to incorrect estimations on the available throughput, causing optimistic quality decisions that are not sustainable in case of cache misses, leading to an increased number of quality oscillations and buffer starvations. Without coordination between the clients and the intermediary caching nodes, the quality decisions are either suboptimal or unsustainable.

**The static nature in which current streaming services are deployed, prevents the flexibility that is required of the network to rapidly offer new and innovative services.** Most of the intermediary NFs (e.g., caches provided by Netflix OpenConnect) that are used in the delivery of streaming services are implemented on physical devices. As such, their functionality is tightly coupled with the devices they run on. To deploy new services, these NFs need to be physically deployed in the network and wired in the correct order. This reduces the flexibility and increases the deployment costs, maintenance costs and time to market. To allow flexible deployment of new and innovative streaming services, these NFs could be virtualized and deployed on the virtualized NFV infrastructure. These VNFs should be deployed in a particular order and at specific locations to bring about the benefits of the streaming Service Function Chains (SFCs). However, current NFV-solutions do not take into account location restrictions, reducing the applicability of NFV techniques for multimedia delivery chains.

## 1.3 Dissertation Outline

This dissertation consists of a number of selected publications that were written in the scope of this PhD research. Together, they provide a consistent overview of the performed work. The various research contributions are detailed in Section 1.4 and the complete list of publications that resulted from this work is presented in Section 1.5. Figure 1.4 shows a schematic overview of the different contributions that are presented in each chapter (Ch.) and appendix (App.). The various SFCs that were optimized in this PhD dissertation are shown at the service layer on top of the virtualized and physical infrastructure layer. Chapters 2-5 and Appendices A-B focus on the various techniques that were developed to optimize the QoE and delivery of both live and VoD streaming services, while Chapter 6 and Appendix C tackle the location-constrained deployment of such SFCs onto the virtualized physical substrate.

Chapter 2 presents the multicast-enabled delivery framework for live HAS services. In standard HAS, each user sets up a unicast connection to the streaming origin, which for live streaming, leads to many unicast sessions transporting the same content. By leveraging multicast and caching techniques, the resource usage of such live streaming services in the network can be reduced. The proposed framework automatically transforms the unicast HAS streaming sessions into multicast sessions at the ingress (Di in Figure 1.4) of the access network. Further down the network (De in Figure 1.4), the multicast sessions are split to reconstruct the original HAS sessions. This approach reduces the strain of OTT streaming services on the access network. An algorithm is presented to autonomously select which video streams are grouped into a multicast channel, based on the popularity in a certain timeframe. Furthermore, another algorithm is presented to select

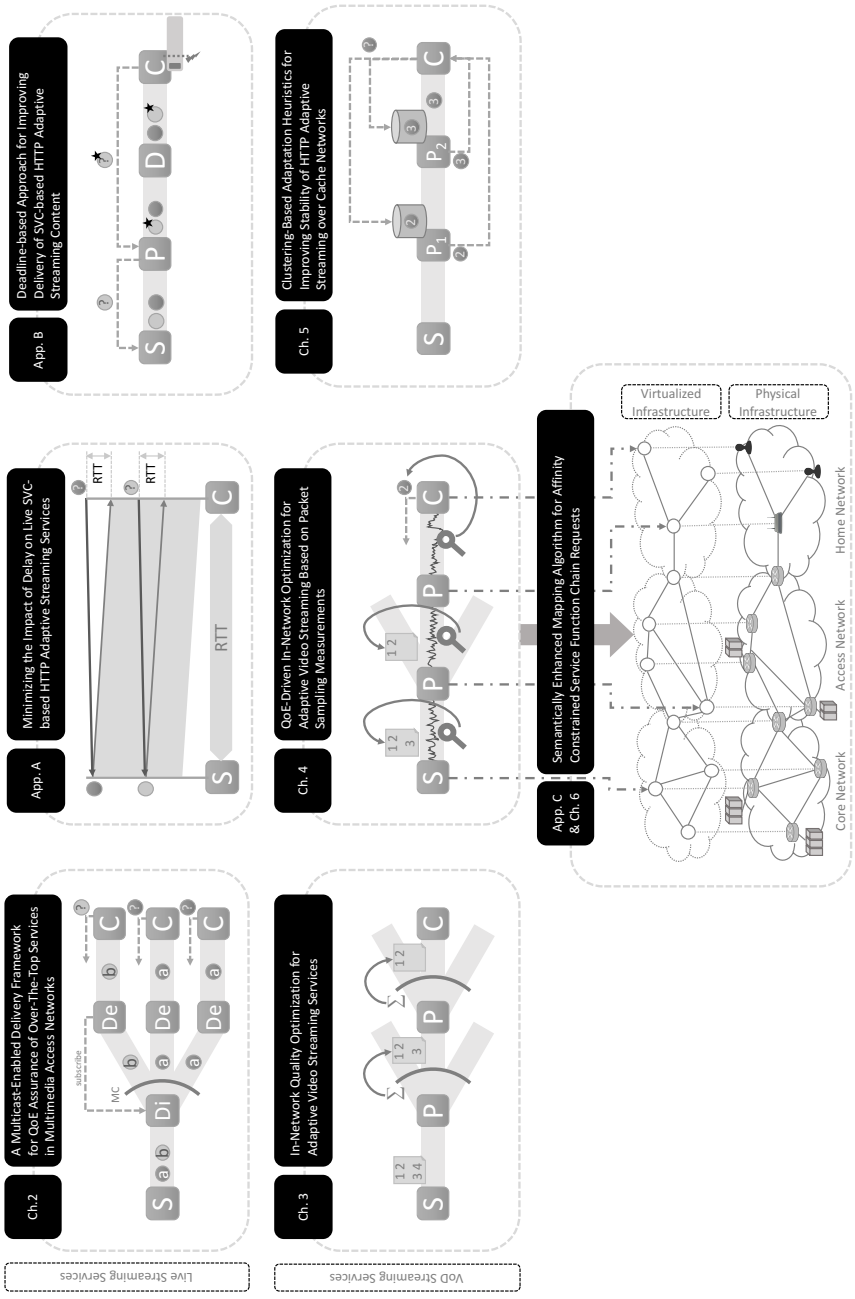



*Figure 1.4: Overview of PhD dissertation.*

which multicast channels to subscribe to at each location in the network, based on the history of requests at that location. Since the popularity of content can vary across geographical locations [14], this avoids having the caches filled with content that will never be requested by the end-users. For example, in Figure 1.4, the top location has not received any requests for *video a*, so it does not subscribe to the multicast channel for that video. The chapter presents the architecture of the framework and the required retransmission schemes to ensure that the multicasting is robust against packet loss. A prototype was deployed on a large-scale testbed to characterize the gain of the approach and the possible overhead in terms of response time. Additionally, a simulation-based framework was implemented to evaluate the performance of the management algorithms for various scenarios.

Appendix A and B present additional management approaches for QoE-improvement of live HAS services. Appendix A tackles the negative impact of high RTTs on live HAS streaming services. By measuring the RTT and pipelining the request one RTT before the estimated finish time of the previous download, the idle time between two consecutive downloads can be removed, while the decision on the quality selection can be delayed as long as possible to account for any variations in the network conditions. Appendix B presents a framework to guarantee a continuous playback for live HAS services without requiring a large play-out buffer. By leveraging the benefits of scalable video coding and introducing an intelligent network proxy, the delivery of the base layer can be guaranteed. A dynamic deadline-based approach is proposed in which the client can signal which segments need to be prioritized (indicated with a star in Figure 1.4) based on the risk of running into a buffer starvation.

Chapter 3 focuses on the managed delivery of HAS services. One of the major obstacles for the adoption of HAS services by network providers is the purely client-driven design, which leads to excessive quality oscillations, suboptimal resource usage and the inability to enforce management policies in managed networks. Furthermore, when a provider wants to offer paid HAS services to its customers, they need to have control over the quality that is provided. This chapter tackles these challenges by proposing a set of centralized and distributed optimization algorithms which allow network nodes, distributed across the network, to monitor the requests and steer the client's quality selection process in order to optimize the QoE and resource usage. These algorithms allow the provider to enforce management policies by limiting the set of available qualities for specific clients as shown in Figure 1.4. Furthermore, a distributed heuristic approach is proposed to allow scalable in-network management of HAS services. By applying the proposed approach the number of quality oscillations can be reduced significantly while improving the overall QoE of the streaming sessions. This work was extended in Chapter 4 to allow the application of in-network QoE-management in dynamic networks as well. By introducing sampling-based measurement probes

in the network, the current resource usage can be monitored in a scalable way and used to estimated the future throughput. Various estimation techniques are proposed and evaluated to make this prediction. Taking this predicted throughput as input, algorithms are proposed to optimize the global QoE of the various streaming sessions. The proposed hybrid approach allows the client to take into account the in-network decisions, while still remaining able to react to sudden bandwidth fluctuations in the local network.

Intermediary caching nodes in HAS delivery networks can compromise the accuracy of the throughput estimations that are used by the adaptation heuristics. Chapter 5 mitigates the impact of incorrect estimations by including additional information on the streaming origin into the adaptation heuristic. Furthermore, cache content information of intermediary caching nodes is exchanged with the clients to improve the quality selection. The proposed approaches require an additional communication channel to transfer the information to the client applications. Since adaptations to intermediary network nodes are required, autonomous approximation methods are proposed as well in case these adaptations are not possible. By leveraging clustering-based techniques, the different streaming origins can be detected based on the measured RTT. By using buffer-based probabilistic prediction techniques, the cache content of intermediary nodes can be predicted as well. The proposed approaches were evaluated for a wide range of scenarios, including mobile scenarios with real RTT traces that were collected.

Chapter 2-5 and Appendices A-B present various SFCs and NFs that optimize the QoE for the delivery of HAS services. Traditionally, these NFs would be implemented on proprietary physical network appliances that need to be placed in a strict chaining order. To increase the service agility and reduce the time to market of new services, these NFs can be virtualized into VNFs and by leveraging NFV technologies, deployed on the fly onto the virtualized substrate. However, many multimedia service chains have some assumptions on the location of the VNFs to achieve the expected performance gain (e.g., a cache should be deployed close to the end-user). Therefore, Appendix C defines a set of affinity and anti-affinity constraints which allow the Service Provider (SP) to define locality restrictions on VNFs and interconnecting virtual edges. Since SFCs can be generated automatically or by human operators, a semantic validation framework is proposed to filter out conflicting constraints. To this end, the substrate and SFCs are modeled using ontologies and by defining a set of inference rules, the semantic reasoner can detect inconsistent constraint sets. Chapter 6 extends this approach by defining aditional algorithms and heuristics to map the SFC sets onto the substrate resources, subject to the capacity and affinity constraints. Furthermore, the benefits of filtering out the SFC requests containing conflicting constraint sets prior to embedding can significantly reduce the total processing time of said embedding requests.

## 1.4 Research Contributions

This dissertation aims to combine various service management techniques to optimize the QoE of HAS streaming services. The ultimate goal is to improve the end-user QoE, while at the same time, reducing the strain multimedia services exert on the network resources and increasing the flexibility of such services. This translates into the following contributions:

1. A framework leveraging multicast techniques to reduce the load of HAS live streaming services on the delivery network. (Chapter 2)
    - Implementation of an emulation framework for the multicast-enabled HAS services, which includes the implementation and evaluation of retransmission schemes to cope with packet loss.
    - Implementation of an NS-3 simulator that supports the simulation of HAS streaming services for large network topologies.
    - Mechanisms to group the various overlapping unicast transmissions that exist for HAS live and TSTV streaming and deliver them via multicast to locations closer to the end-user, reducing the strain on the delivery network. Mechanisms to transform the multicast sessions into unicast sessions which realise the video delivery over the last mile.
    - Algorithms that autonomously select which video segments are multicasted and to which multicast channels the geographically dispersed delivery servers subscribe. These decisions are based on the popularity of the videos for various groups of service users in the network.
    - An evaluation of both the emulation and simulation frameworks for smaller topologies. An extensive evaluation using simulations for various topology configurations and network settings, demonstrating the reduction of the load for the multicast-enabled delivery of HAS live streaming services.
2. An in-network optimization framework to improve the QoE of VoD HAS services. The proposed approach mitigates the quality degradation that arises when multiple autonomous HAS clients compete for bandwidth. (Chapters 3 and 4)
    - Extension of the NS-3 simulator with support for VoD streaming and various alternative HAS adaptation heuristic proposed in literature.
    - A formal description of the optimization algorithm which assigns the most appropriate quality to each client, subject to the available resources and connected users. Next to the optimal algorithms, a set of heuristics was developed to provide suboptimal solutions within a reasonable period of time.

- The implementation and evaluation of multiple bandwidth prediction methods that provide input to the aforementioned optimization algorithm. These methods allow the QoE-optimization in a dynamic environment. By using sampling-based measurement techniques, these network probes can function in a scalable way.
- Quantitative analysis of the QoE-improvement that can be achieved by the in-network optimization. First, the performance gain that can be achieved using the optimal algorithms is evaluated. Second, these results are compared to the solutions provided by the heuristic approaches.

3. Novel adaptation heuristics to improve the performance of HAS in the presence of in-network caches. These heuristics are able to handle varying throughput estimations caused by different origin locations. (Chapter 5)

   - A set of heuristics that take advantage of the knowledge of the streaming origin and upstream cache contents to optimize the quality selection for HAS services.
   - An evaluation of the potential QoE-gain when including perfect knowledge of the streaming origin and cache contents into the quality adaptation heuristic and a comparison with state-of-the art adaptation heuristics for a variety of scenarios.
   - A clustering-based origin detection technique which uses features, such as measured RTT, to differentiate between streaming origins. A probability-based heuristic to estimate the upstream cache contents when this information is not provided by the network nodes.
   - An evaluation of QoE-gain in the absence of perfect knowledge, using the proposed approximation techniques for various scenarios. This includes dynamic scenarios using real-life RTT and throughput traces.

4. Extension of the SFC specification with affinity and anti-affinity constraints to support restrictions on the placement of VNFs or virtual edges interconnecting them for efficiency, economic, legislative, privacy and security reasons. (Chapter 6)

   - Formal definition of the affinity and anti-affinity constraints which allow the SP to enforce or restrict the colocation of specific VNF instances or all instances of a VNF type to a specific location or a group of locations with a certain granularity. Furthermore, constraints are defined that enforce or restrict the interconnecting edges to be routed through specific locations or to be colocated with other virtual edges at certain locations.

- A semantic framework that supports the validation of sets of affinity and anti-affinity constraints contained in an SFC request. The semantic framework models the SFC and virtualized substrate using an ontology and defines a set of rules that can be used by the semantic reasoner to infer knowledge. Applying semantic reasoning allows to check the consistency of the constraint set and filter out inconsistent SFCs.
- Embedding algorithms for affinity-constrained SFC sets to determine the optimal SFC mapping to the virtualized resources, subject to capacity constraints and optimizing multiple objectives (e.g., minimizing or balancing the load on the network).
- A quantitative evaluation to identify the impact of filtering out inconsistent SFC requests before these are served to the embedding algorithms. The performance of the semantically enhanced embedding techniques was evaluated for various scenarios and compared to standard embedding techniques.

## 1.5 Publications

The research results obtained during this PhD research have been published in scientific journals and presented at a series of international conferences. Furthermore, for some of the contributions in the field of NFV, patent applications were submitted. The following list provides an overview of the publications and patent applications during the PhD research.

### 1.5.1 A1: Journal publications indexed by the ISI Web of Science "Science Citation Index Expanded"

1. **Niels Bouten**, Steven Latré, Wim Van de Meerssche, Bart De Vleeschauwer, Koen De Schepper, Werner Van Leekwijck, Filip De Turck. *A Multicast-Enabled Delivery Framework for QoE Assurance of Over-The-Top Services in Multimedia Access Networks*. Published in Journal of Network and Systems Management, vol. 21, pp. 677-706, December 2013.

2. **Niels Bouten**, Steven Latré, Jeroen Famaey, Werner Van Leekwijck, Filip De Turck. *In-Network Quality Optimization for Adaptive Video Streaming Services*. Published in IEEE Transactions on Multimedia (TMM), vol. 16, no. 8, pp. 2281-2293, December 2014.

3. **Niels Bouten**, Ricardo de O. Schmidt, Jeroen Famaey, Steven Latré, Aiko Pras, Filip De Turck. *QoE-Driven In-Network Optimization for Adaptive Video Streaming Based on Packet Sampling Measurements*. Published in Computer Networks (COMNET), vol. 81, pp. 96-115, April 2015.

4. Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, **Niels Bouten**, Filip De Turck, Raouf Boutaba. *Network Function Virtualization: State-of-the-art and Research Challenges*. Published in IEEE Communications Surveys and Tutorials, vol. 18, no. 1, pp. 236-262, January 2016.

5. Maxim Claeys, **Niels Bouten**, Danny De Vleeschauwer, Werner Van Leekwijck, Steven Latré, Filip De Turck. *Cooperative Announcement-based Caching for Video-on-Demand Streaming*. Published online in IEEE Transactions on Network and Service Management (TNSM), March 2016.

6. **Niels Bouten**, Rashid Mijumbi, Joan Serrat, Jeroen Famaey, Steven Latré Filip De Turck. *Semantically Enhanced Mapping Algorithm for Affinity Constrained Service Function Chain Requests*. IEEE Transactions on Network and Service Management (TNSM), Submitted, 2016.

7. **Niels Bouten**, Danny De Vleeschauwer, Werner Van Leekwijck, Steven Latré, Filip De Turck. *Clustering-based Quality Selection Heuristics for HTTP Adaptive Streaming over Cache Networks*. International Journal of Network Management (IJNM), Submitted, 2016.

### 1.5.2 P1: Proceedings included in the ISI Web of Science "Conference Proceedings Citation Index - Science"

1. **Niels Bouten**, Steven Latré, Wim Van de Meerssche, Koen De Schepper, Bart De Vleeschauwer,Werner Van Leekwijck, Filip De Turck. *An Autonomic Delivery Framework for HTTP Adaptive Streaming in Multicast-enabled Multimedia Access Networks*. In proceedings of the IEEE Network Operations and Management Symposium (NOMS 2012), pp. 1248-1253, April 2012.

2. **Niels Bouten**, Anna Hristoskova, Femke Ongenae, Jelle Nelis, Filip De Turck. *Ontology-driven Dynamic Discovery and Distributed Coordination of a Robot Swarm*. In proceedings of the International Conference on Autonomous Infrastructure, Management and Security (AIMS 2012), LNCS, vol. 7279, pp. 2-13, June 2012.

3. **Niels Bouten**, Steven Latré, Jeroen Famaey, Werner Van Leekwijck, Filip De Turck. *Minimizing the Impact of Delay on Live SVC-based HTTP Adaptive Streaming Services*. In proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 1399-1404, May 2013.

4. Jeroen Famaey, Steven Latré, **Niels Bouten**, Wim Van de Meerssche, Bart De Vleeschauwer, Werner Van Leekwijck, Filip De Turck. *On the merits*

*of SVC-based HTTP Adaptive Streaming*. In proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 419-426, May 2013.

5. **Niels Bouten**, Maxim Claeys, Robin Bailleul, Jin Li, Jeroen Famaey, Steven Latré, Jan De Cock, David Lou, Werner Van Leekwijck, Filip De Turck. *Improved Delivery of Live SVC-based HTTP Adaptive Streaming Content*. In Proceedings of the IEEE Network Operations and Management Symposium (NOMS 2014), pp. 1-2, May 2014.

6. Sebastiaan Laga, Thomas Van Cleemput, Filip Van Raemdonck, Felix Vanhoutte, **Niels Bouten**, Maxim Claeys, Filip De Turck. *Optimizing Scalable Video Delivery Through OpenFlow Layer-based Routing*. In Proceedings of the IEEE Network Operations and Management Symposium (NOMS 2014), pp. 1-4, May 2014.

7. **Niels Bouten**, Maxim Claeys, Steven Latré, Jeroen Famaey, Werner Van Leekwijck, Filip De Turck. *Deadline-based Approach for Improving Delivery of SVC-based HTTP Adaptive Streaming Content*. In Proceedings of the IEEE Network Operations and Management Symposium (NOMS 2014), pp. 1-7, May 2014.

### 1.5.3 C1: Other publications in international conferences

1. **Niels Bouten**, Jeroen Famaey, Steven Latré, Rafael Huysegems, Bart De Vleeschauwer, Werner Van Leekwijck, Filip De Turck. *QoE Optimization Through In-Network Quality Adaptation for HTTP Adaptive Streaming*. In proceedings of the 8th IFIP/IEEE International Conference on Network and Service Management (CNSM 2012), pp. 336-342, October 2012.

2. **Niels Bouten**, Steven Latré, Filip De Turck. *QoE-centric Management of Multimedia Networks through Cooperative Control Loops*. In proceedings of the International Conference on Autonomous Infrastructure, Management, and Security (AIMS 2013), LNCS, vol. 7943, pp. 69-99, June 2013.

3. Rashid Mijumbi, Joan Serrat, Javier Rubio-Loyola, **Niels Bouten**, Filip De Turck, Steven Latré. *Dynamic Resource Management in SDN-based Virtualized Networks*. In Proceedings of the International Conference on Network and Service Management (CNSM) and Workshops (CNSM 2014), pp. 412-417, November 2014.

4. Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, **Niels Bouten**, Filip De Turck, Steven Davy. *Design and Evaluation of Algorithms for Mapping and Scheduling of Virtual Network Functions*. In Proceedings of the IEEE Conference on Network Softwarization (NetSoft 2015), pp. 1-9, April 2015.

5. **Niels Bouten**, Jeroen Famaey, Rashid Mijumbi, Bram Naudts, Joan Serrat, Steven Latré, Filip De Turck. *Towards NFV-based Multimedia Delivery*. In Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management 2015 (IM 2015), pp. 738 - 741, May 2015.

6. Stefano Petrangeli, **Niels Bouten**, Jeroen Famaey, Filip De Turck, Philip Leroux. *Design and Evaluation of a DASH-compliant Second Screen Video Player for Live Events in Mobile Scenarios*. In Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management 2015 (IM 2015), pp. 894 - 897, May 2015.

7. Stefano Petrangeli, **Niels Bouten**, Maxim Claeys, Filip De Turck. *Towards SVC-based Adaptive Streaming in Information Centric Networks*. In Proceedings of the IEEE International Conference on Multimedia and Expo Workshops 2015 (ICMEW 2015), pp. 1-6, July 2015.

8. Maxim Claeys, **Niels Bouten**, Danny De Vleeschauwer, Werner Van Leekwijck, Steven Latré, Filip De Turck. *An Announcement-based Caching Approach for Video-on-Demand Streaming*. In Proceedings of the International Conference on Network and Service Management (CNSM 2015), pp. 310-317, November 2015.

9. Jeroen van der Hooft, Stefano Petrangeli, **Niels Bouten**, Tim Wauters, Rafael Huysegems, Tom Bostoen, Filip De Turck. *An HTTP/2 Push-Based Approach for SVC Adaptive Streaming*. In Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2016), pp. 1-8, April 2016.

10. **Niels Bouten**, Maxim Claeys, Rashid Mijumbi, Jeroen Famaey, Steven Latré, Joan Serrat. *Semantic Validation of Affinity Constrained Service Function Chain Requests*. In Proceedings of the IEEE Conference on Network Softwarization (NetSoft 2016), pp. 202-210, June 2016.

11. Maxim Claeys, **Niels Bouten**, Danny De Vleeschauwer, Koen De Schepper, Werner Van Leekwijck, Steven Latré, Filip De Turck. *Deadline-aware TCP Congestion Control for Video Streaming Services*. Accepted as full paper at CNSM 2016.

12. **Niels Bouten**, Maxim Claeys, Bert Van Poecke, Steven Latré, Joan Serrat. *Dynamic Server Selection Strategy for Multi-server HTTP Adaptive Streaming Services*. Accepted as full paper at CNSM 2016.

### 1.5.4 European patent applications

1. **Niels Bouten**, Jeroen Famaey, Rudolf Strijkers, Shuang Zhang. Method for routing data packets to an instance of a network function. European patent application EP14200305.2, Submitted December 2014.

2. **Niels Bouten**, Jeroen Famaey, Rudolf Strijkers, Shuang Zhang. Method for controlling on-demand service provisioning. European patent application EP14200309.4, Submitted December 2014.

3. **Niels Bouten**, Jeroen Famaey. Service provisioning in a communication network. European patent application, Submitted July 2015.

# References

[1] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff. *A brief history of the Internet*. ACM SIGCOMM Computer Communication Review, 39(5):22–31, 2009.

[2] T. Berners-Lee. *Information Management: A Proposal*, 1989. Available from: http://www.w3.org/History/1989/proposal.html.

[3] R. C. Tim Berners-Lee. *WorldWideWeb: Proposal for a HyperText Project*, 1990. Available from: http://www.iis.net/downloads/microsoft/smooth-streaming.

[4] Forecast, Cisco VNI. *Cisco Visual Networking Index: Forecast and Methodology, 2014-2019 White Paper*. Technical report, Cisco Public Information, May 2015.

[5] Microsoft. *Microsoft Smooth Streaming*, 2008. Available from: http://www.w3.org/Proposal.html.

[6] Apple. *Apple HTTP Live Streaming*, 2009. Available from: https://tools.ietf.org/html/draft-pantos-http-live-streaming-19.

[7] Adobe. *Adobe HTTP Dynamic Streaming*, 2009. Available from: http://www.adobe.com/products/hds-dynamic-streaming.html.

[8] T. Stockhammer. *Dynamic adaptive streaming over HTTP: standards and design principles*. In Proceedings of the second annual ACM conference on Multimedia systems, MMSys '11, pages 133–144, New York, NY, USA, 2011. ACM.

[9] Sandvine. *Global Internet Phenomena*. Technical report, Sandvine, Intelligent Broadband Networks, May 2016.

[10] Conviva. *2015 Viewer Experience Report*. Technical report, Conviva, May 2015.

[11] ETSI. *Network Functions Virtualization: An Introduction, Benefits, Enablers, Challenges and Call for Action*. ETSI Document, October 2012. Available from: http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[12] ETSI. *Network Functions Virtualization: Network Operator Perspectives on Industry Progress*. ETSI Document, October 2013. Available from: http://portal.etsi.org/NFV/NFV_White_Paper2.pdf.

[13] ITU-T Recommendation P.10/G.100 (2006) Amendment 4 (06/15). *Vocabulary for performance and quality of service*, 2015. Available from: https://www.itu.int/rec/T-REC-P.10/en.

[14] D. Tuncer, M. Charalambides, R. Landa, and G. Pavlou. *More control over network resources: An ISP caching perspective*. In Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013), pages 26–33, Oct 2013.

# 2

# A Multicast-Enabled Delivery Framework for QoE Assurance of Over-The-Top Services in Multimedia Access Networks

**N. Bouten, S. Latré, W. Van de Meerssche, B. De Vleeschauwer, K. De Schepper, W. Van Leekwijck, F. De Turck.**

**Published in Journal of Network and Systems Management, December 2013.**

*This chapter focusses on the management of live HTTP Adaptive Streaming (HAS) services. A loosely coupled architecture is presented that can be seamlessly integrated into an existing HAS based video delivery architecture. The proposed approach groups the existing HAS based video connections to be multicasted over a network's bottleneck and then splits them again to reconstruct the original HAS sessions. A prototype of this architecture is presented, which includes the caching of videos and incorporates retransmission schemes to ensure robust transmission. Furthermore, an autonomic algorithm is presented that allows to intelligently select which videos need to be multicasted by making a remote assessment of the cache state to predict the future availability of content. The proposed approach can reduce the bandwidth consumption in the network with* $25\%$ *compared to a HAS with proxies approach. Other approaches to increase the Quality of Experience (QoE) of live HAS services will be addressed in Appendices A and B.*

## 2.1 Introduction

The consumption of video services over the best effort Internet, often called Over-The-Top (OTT) video services, has recently gained considerable attention. Typical OTT providers such as BBC iPlayer [1], YouTube [2], Hulu [3] and Netflix [4] are still observing a constant growth. Moreover, several reports [5, 6] mention a double digit growth of OTT video services in recent years.

Compared to traditional managed Internet Protocol television (IPTV) services, OTT services offer higher flexibility to the customers for video consumption and often have a larger content catalogue available. While the original OTT video services typically offered a Video on Demand (VoD) type of service, other service types such as live video streaming are more and more being offered as well. For example, the BBC iPlayer service streams all BBC programs live and YouTube has recently allowed to stream live events through YouTube as well [7]. Moreover, OTT services are already evolving towards adaptive bitrate streaming models, where the bitrate of the video is dynamically adapted as a function of the network load. For example, both BBC iPlayer and Hulu support the Adobe HTTP Dynamic Streaming protocol, which is an implementation of HTTP Adaptive Streaming (HAS). As such, the functionality offered by OTT services is more and more evolving into an interesting alternative for managed IPTV services.

For network and service providers, OTT services however introduce important new revenue opportunities. The increased popularity, greater consumption flexibility and larger content catalogue of OTT services poses a potential threat to the traditional IPTV services. Network and service providers are therefore showing interest in integrating OTT services into their current offerings. An example of this are the paid OTT models that are emerging such as Amazon Instant Video [8].

Although OTT services have several advantages, they differ from managed IPTV services when concerning scalability and robustness. As OTT services are delivered over the best effort Internet, they cannot use the managed infrastructure (e.g., streaming with resource reservation, multicasting) that is used for traditional IPTV services. This results in a number of issues with today's OTT services. First, it is not possible to provide any Quality of Service (QoS) or Quality of Experience (QoE) guarantees on the consumption of the video: the video is accessed over the best effort Internet and no resource reservation is made for this. Second, OTT services are far less scalable than IPTV services: the use of an OTT service by the customer results in a new unicast connection between the OTT server and the customer. As such, the amount of bandwidth that is required for an OTT service is proportional to the number of consumers of that service. Unlike linear TV in a managed IPTV environment, it is not possible to multicast content over the public Internet to cope with peaks in the consumption of the service.

In this chapter, a framework is presented that allows providing QoE assur-

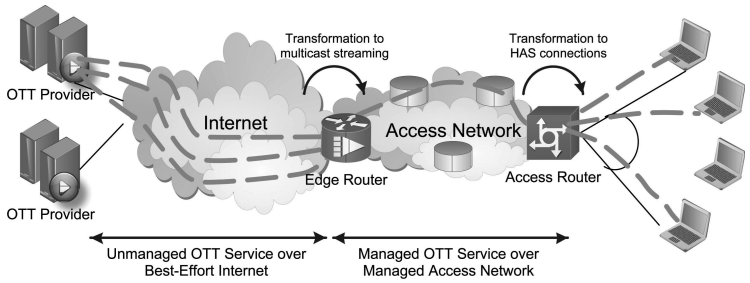



*Figure 2.1: Overview of the proposed framework. The framework allows managing OTT services for scalable transportation in the multimedia access network.*

ance of OTT services in multimedia access networks. As illustrated in Figure 2.1, the framework can be deployed in an access network by a network and service provider and allows transporting OTT services, consumed from the public Internet, over a managed IPTV environment. To address this, a combination of caching and multicasting is used. The framework can be seamlessly integrated into an existing HTTP-based delivery framework: this chapter focuses on a HTTP Adaptive Streaming based delivery framework. In the proposed approach, existing HAS-based connections that stream the same content in a given time window are grouped into one multicast connection, which is transported over the managed network. Afterwards, the received multicast content is split into the original HAS-based connections to ensure that the consumers of the service do not notice any change. The result is a lower network load in the access network to provide the same QoS and QoE levels. This approach allows managing the OTT services and decreasing the amount of bandwidth that is required for its consumption considerably. As such, the network and service provider can optimize the QoE of the OTT services and resolve the scalability issues that arise, by using techniques that are available in a managed environment (i.e. caching, multicasting).

The contributions of this chapter are two-fold. First, an overview is presented of the architecture for providing QoE assurance of OTT services. The architecture features two component types, a distribution server and a delivery server, that cooperate with each other to enable the deployment over a managed network environment. The proposed architecture contains retransmission mechanisms to ensure that the multicasting is robust against packet loss. Second, a management algorithm is presented that decides which content is most suited for multicasting and which connections can be left as unicast HAS-based connections. The algorithm is deployed on the distribution server and combines an identification of popular content with an assessment of the remote caches to select the content that will be responsible for the biggest reduction in peak bandwidth. The proposed framework is evaluated extensively: a prototype has been designed and deployed on a large

scale testbed that characterizes the gain of the approach and its overhead in terms of response time of the service consumption. Additionally, simulation-based experiments evaluate the performance of the management algorithm as a function of the system parameters (i.e. cache sizes, number of delivery servers, number of multicast channels).

The remainder of this chapter is structured as follows. Section 2.2 discusses relevant work in the area of the management of OTT services and the use of multicasting for video streaming. In Section 2.3, the HAS technique, which is used as transport mechanism in the delivery framework, is explained. The proposed framework is presented in Section 2.4 and Section 2.5. Section 2.4 presents the overall architecture, while Section 2.5 describes the management algorithm. Finally, the framework is evaluated both through emulations and simulations in Section 2.6.

## 2.2 Related Work

### 2.2.1 OTT video streaming architectures

The consumption of video services over the Internet has undergone an important evolution over the last few years. In the past, video was often streamed through streaming protocols such as Real-time Transport Protocol (RTP) [9] and Real-time Streaming Protocol (RTSP) [10]. However, more recently, the popularity of OTT services has led to a shift in technologies. Therefore, many streaming technologies today are based on HTTP. These new streaming technologies offer important advantages to the consumption of video. First, they provide a reliable data-transfer of the video which eliminates visual artifacts such as blockiness [11] (i.e. caused by the loss of a packet containing information concerning that encoded block). Second, they are more suited for traversal through firewalls as they use standard HTTP connections. And third, they do not require to maintain a state on the server, which increases their scalability. In [12], Saxena et al. investigate the content delivery frameworks of popular OTT services such as YouTube, Dailymotion and MetaCafe. Besides characterizing the service delay of these services, they investigate the content delivery methods and server locations used by these OTT services. They show that most content is served from central servers, while only a few percents are served by Content Delivery Networks (CDNs).

Initially, HTTP-based delivery of video required the download of the complete video before the video could be played. Afterwards, progressive download techniques allowed to commence playback after only a fraction of the video was downloaded. When using progressive download, the video is stored in a play-out buffer at the client and the playback is started once the play-out buffer is sufficiently filled (typically a few seconds worth of video playback). While visual artifacts such as blockiness are avoided through progressive downloads, other visual artifacts such

as stuttered playback (i.e. caused by frame freezes) remain. This type of visual artifacts often occur on lossy or narrowband links: if the throughput achieved by the HTTP connection is less than the throughput required to stream the video, the play-out buffer can occasionally deplete. This leads to a halted video playback, as the buffer is refilled, resulting in a stuttered playback.

HAS [13–17] is the third evolution in HTTP streaming technologies and allows coping with narrowband and lossy links by downloading and playing a lower bitrate of the video. The HAS technique supports dynamic adaptive streaming of the video. The content is split in small segments of a few seconds long and encoded into multiple qualities. HAS assumes the presence of an intelligent video client that decides which quality level to download next based on performance characteristics such as the experienced network throughput, available computing power etc. A more in depth discussion of HAS is provided in Section 2.3.

In [18] the authors present a proxy agent called Transmission-Rate Adapted Streaming Server (TRASS) to allow users to obtain adaptive video streams according to their varying receiving capabilities in a wireless circumstance. The multi-rate controller converts the original video stream from the video server to several different rates according to the feedback of each user concerning network layer characteristics such as packet receiving rate and packet loss rate. This approach differs from the proposed architecture since the offered quality rates are dependent on the users context while in the proposed approach users can select the quality best suiting their connection characteristics from a list of offered qualities. Furthermore an algorithm minimizes the consumed bandwidth in the managed network, by grouping sessions requesting the same content into a single multicast session.

Several Peer to Peer (P2P) streaming systems have been deployed to provide live and on-demand video streaming services on top of the best-effort Internet at low server cost. As described in [19] there are several fundamental limitations of existing P2P streaming solutions. First of all, user QoE in current streaming systems are not comparable to traditional TV services since users experience longer channel startup times, channel delays and playback lags among users. Secondly, the increased popularity of P2P streaming systems pose a big challenge on Internet Service Provider (ISP)'s network capacity, caused by the high traffic volume without any profit for ISPs. The proposed approach allows higher QoE in terms of low delay streaming and fast startup times as well as higher quality streaming by employing HAS techniques. Furthermore, the system allows service and network providers to control streaming traffic and minimizing it by incorporating caching and multicasting techniques.

### 2.2.2 Management of OTT video services

The increasing popularity of OTT video services has resulted in much recent research attention. On one hand, important research has been performed in characterizing these OTT video services. By monitoring at the edge of the network, several studies [20, 21] have characterized the traffic patterns of services such as YouTube. This work focuses on the management of these important OTT video services. In [16, 17], Begen et al. provide an overview of the current standardization efforts and applications in watching video through OTT services. They argue that the use of adaptive streaming technologies such as HAS is an important driver for OTT video services. Moreover, they identify a few important future research directions. First, they explicitly mention the scalability bottleneck that arises if clients access OTT services concurrently. Second, they state that it is important for providers to introduce network elements that improve the performance of OTT video services. They argue that, for an OTT service, the network should not be regarded as a black box. Instead, an OTT service should exchange information with the network and act accordingly if failures arise. The solution proposed in this chapter makes use of such intelligent network elements: transforming OTT services for transport in a managed environment in order to achieve higher performance.

Such a cooperation between an ISP, representing a managed network, and an OTT system is studied in [22]. There, the traditional network-aware overlays, which exploit locality in the network for the underlying routing strategy, are extended with managed IPTV service delivery mechanisms. The proposed collaboration framework allows exchanging QoS requirements such as jitter, delay and throughput, which can be used to select the appropriate peers in the P2P network that support those QoS requirements. The proposed approach is similar to [22] as it targets a closer collaboration between OTT services and managed environments. However, the proposed approach focuses on OTT services that are delivered directly from centralized OTT servers or CDNs, while the work in [22] focuses on the problem of P2P peer selection.

In [23], an architecture is proposed to optimize Live TV services for OTT video. Moshe et al. propose a multi-layered cache algorithm that aims at reducing the load in the network by effectively caching the Live TV content. Similar to [23], the framework proposed in this chapter also incorporates caches to temporarily store streamed content of an OTT service. However, this work differs from the work by Moshe et al. in two ways. First, next to caching, the proposed approach also uses multicasting techniques to further reduce the load in the network. An autonomic management algorithm is proposed that interacts with the caches to decide which content needs to be multicasted. Second, the approach proposed in this chapter does not only focus on Live TV services. Besides Live TV, Time Shifted TV (TSTV) services are also taken into account in the algorithmic design

and performance evaluation.

This chapter builds further on previous work. In [24], a first preliminary version of the architecture is proposed, focusing solely on Live TV services. The contributions of this chapter extend the work presented in [24] in several ways. First, the scope of supported services is extended to other services besides Live TV such as TSTV and VoD services. The autonomic management algorithm presented in Section 2.5 provides a way to select which content to multicast, therefore supporting also peaks in user requests for these novel services (i.e. by multicasting other content than the live moment). Second, the architecture described in Section 2.4 was also modified to support these services. Third, a more extensive performance evaluation study was carried out, extending both the performance study of the prototype (i.e. by focusing more on the performance of the retransmission schemes) and introducing extensive simulation-based experiments to evaluate the management algorithm.

### 2.2.3 Multicast streaming of multimedia services

In this chapter, it is proposed to transform several OTT video connections into a single multicast signal. In a typical managed multimedia architecture, multicast techniques are used for streaming live video as a broadcast signal to multiple video clients (e.g., for live streaming as part of linear TV). Zhang et. al [25] provide a survey of VoD streaming techniques. They identify four types of VoD streaming techniques: (i) CDNs, which reduce the load through caching, (ii) IP multicast-based schemes, which are mainly used in managed environments such as IPTV, (iii) multicast-based P2P networks, where the P2P network forms an application-based multicast tree and (iv) swarming-based P2P networks, where local P2P neighbors exchange video data. The proposed solution combines the CDN approach with the IP-based multicast approach as it multicasts content from CDNs or regular OTT servers to reduce the load in access networks.

The combination of P2P video multicast streaming with adaptive streaming technologies has been investigated in [26]. The solution presented there uses a meshed P2P network where peers are connected to each other by the User Datagram Protocol (UDP) connection to stream video. By using Scalable Video Coding (SVC) [27], different clients receive different video qualities depending on their link capacity. While their solution focuses on the rate adaptation decision, the proposed approach uses a provider's perspective. An autonomic management algorithm is presented that decides on which content to multicast in order to decrease the load on the network and thus to solve the scalability bottleneck that can arise in a multimedia access network [28].

Also for VoD services (both managed and OTT), P2P-based [29] and multicast streaming [30] solutions have been proposed. Here, the described transfer

mechanism uses additional intelligence in the router in order to tackle two problems: synchronisation of video delivery requests and buffering of video packets. If a requested video packet is received at a router, it is stored into the video request synchronisation buffer and only released when enough video data is available in the buffer. While this approach successfully employs multicast for VoD services it results in long delays when requesting a video. P2PVR [31] proposes a P2P-based VoD architecture where peers are organised into a playback offset aware tree-based overlay. On-demand streaming data is shared among other peers with similar playback offset. Also a directory assists peers, which are searching for nodes that contain the expected streaming data. In the proposed approach, a similar tree-based overlay is used with a fixed number of proxies, called delivery servers in the architecture, closer to the video clients.

The proposed approach differs from existing work by (i) addressing the scalability bottleneck that arises in the multimedia access network by using a combination of multicast and caching techniques, (ii) not only considering Live TV settings but Time-Shifted TV as well, while offering support for Video On Demand services simultaneously and (iii) allowing management of OTT video streaming by providers while offering seamless integration with existing HAS deployments.

## 2.3 HTTP Adaptive Streaming

HAS refers to the family of techniques that allow streaming video over HTTP, while providing the ability to the video client to dynamically alter the received video bitrate. Several HTTP Adaptive Streaming protocols have been proposed by industrial players, such as Microsoft's ISS Smooth Streaming (Microsoft) [13], Apple's HTTP Live Streaming [14] and Adobe's HTTP Dynamic Streaming [15]. Moreover, the Moving Picture Expert Group (MPEG) is currently standardizing the Dynamic Adaptive Streaming over HTTP (DASH) [32] technique, which is similar to the industrial proposals. Although differences exist in the details of the different HAS techniques, all techniques exhibit the same set of architectural components. First, HAS requires a media encoder and stream segmenter at the server side, which are responsible for encoding the content in multiple bitrates, and preparing the segments to be transported over HTTP. Second, a delivery infrastructure is required that consists of standard HTTP web servers and optional proxies. Finally, the video client must support the HAS technique: an intelligent client is responsible for selecting which video qualities to download.

Figure 2.2 provides an overview of the common operation of HAS. At the encoding side, the media encoder encodes the video in several video quality levels, corresponding with different video bitrates. Next, the stream segmenter splits the video into different chunks, called segments. Depending on the used HAS technique, each chunk is between 2 and 10 seconds. Each segment can be downloaded

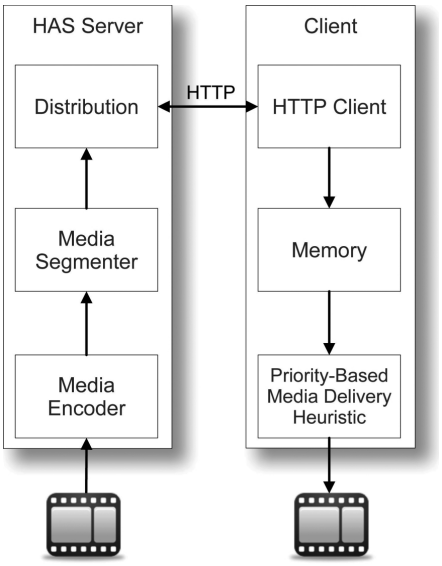



*Figure 2.2: Generic architecture of HAS where a client connects directly to the HAS server over HTTP.*

independently and switching to a different bitrate is easily achieved by downloading the next segment in another bitrate than the previous one. To link the different segments and bitrates with each other, a manifest file is generated as well. This manifest file can be seen as a semantic description of the high level characteristics of the video: it describes which segments are available, where they can be downloaded and the associated bitrate. Differences in the currently available HAS techniques arise by the syntactic difference of these manifest files.

The video clients use the information contained in the manifest file(s) to discover which segments are available and to which bitrates can be switched. The clients request the appropriate media segments through HTTP. After downloading the files, the segments are reassembled so that the media can be represented as a continuous stream. As manifest files also contain information about the supported quality levels, the client can decide to download higher or lower quality segments to ensure a seamless rate adaptation. A video selection heuristic contained in the video client is responsible for deciding which quality levels are to be downloaded. Different video selection heuristics exist depending on the implementation of the video client. Several algorithmic options are available in designing a video selection heuristic: one heuristic can take only network characteristics (e.g., a too low network throughput) into account when making the decision, while another can also take device characteristics such as the amount of battery and CPU power available into account. Although HAS techniques are similar in many ways, differ-

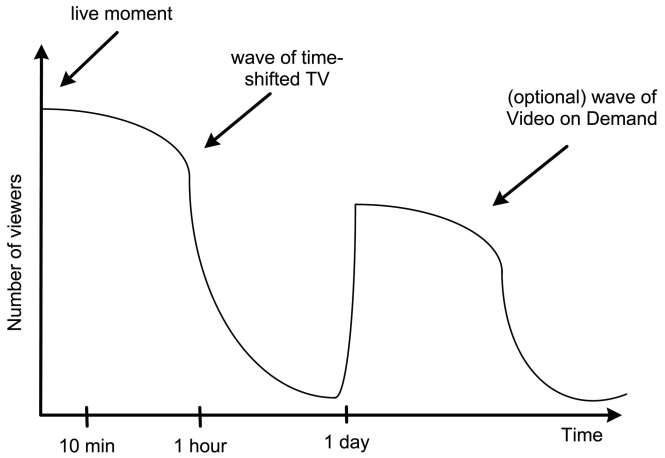



*Figure 2.3: Overview of a typical consumption pattern of Live TV and TSTV-based video services. When the content is first streamed live, a peak in viewers occur. Additional waves of service requests can occur depending on the type of the video content.*

ences can be found in the type of manifest files, segment duration, media container type, supported video codecs, file types on server and end-to-end latency.

## 2.4 A Scalable Architecture for Video Delivery

The goal of the proposed architecture is to decrease the required peak bandwidth of an OTT service between HAS server and client through a combination of multicast streaming and content caching. The architecture supports both Live TV and VoD services: it is assumed that the OTT video service starts offering the content at the same time the live broadcasting channel of linear TV is made available. After the start of the video, consumers can request the video as part of a TSTV or VoD service as well. For TSTV, the video is again streamed from the start but with an offset to the broadcasting moment. Hence, the consumption of TSTV and VoD videos is personalized and there is no synchronisation between the videos.

Figure 2.3 illustrates the typical consumption pattern that can be derived from such an OTT video service. As shown, for a Live TV service, a first peak of viewers typically occurs when the video is made available for the first time, possibly in line with the regular broadcasting of linear TV. This moment in time is referred to as the live moment. In the next minutes and further, the number of consumers of the video service decreases through an exponential distribution. This results in a peak of requests performed by users that were too late to tune in for the live moment but still want to watch the video from the beginning. This type of service requests corresponds with a TSTV service: the users consume the video just as

regular Live TV but watch it with a given delay. Depending on the type of video content, a second and more peaks can occur. For example, the popularity of a video can suddenly increase drastically because of a discussion on the Internet or because of a subsequent episode of the same television program, this type of requests corresponds with a VoD service.

In a standard HAS delivery framework, each of these peaks of requests result in a peak in the number of HAS connections that are being set up. For each request for an OTT video, an additional unicast HTTP connection needs to be set up. This approach is not scalable: although OTT services can have the same request patterns as regular linear TV, they are not suited for such delivery. However, as the users typically request the same video with only a marginal difference in the playback, it is more advisable to try to avoid these peaks by finding commonalities in the delivery of the video. The approach presented in this chapter detects these peaks and solves the scalability issues they introduce. For Live TV, all viewers watch the video in a synchronized way leading to a high number of unicast sessions that can be combined into a single multicast session. With TSTV on the other hand, viewers tune in at different offsets to the original broadcasting time, leading to a less synchronized viewing pattern where sessions can be combined withing a certain time window. A VoD service will lead to even less synchronized viewing and far less sessions sharing the same content. This will cause the proposed architecture to leave most of the unicast HTTP sessions unaltered, delivering the video in the same way as a direct downloading scheme. The proposed approach thus supports Live TV, TSTV and VoD sessions with decreasing bandwidth consumption benefit: for Live TV, almost all sessions share the same content (depending on the number of channels and channel popularity), where for VoD, only few of the sessions share the same content (popular content that is shared among many viewers). In the remainder of this section, a description is given of the components of the proposed architecture, followed by the details on their interactions.

### 2.4.1 Architectural overview

In the proposed framework, two component types are added to the original HAS delivery architecture: a distribution server that could be deployed at the edge router and multiple delivery servers which could be deployed at the access routers. The proposed architecture and a possible mapping to a network topology is shown in Figure 2.4. To the HAS server, the distribution server acts as a video client: it downloads manifest files and segments when new content becomes available and streams it to the connected delivery servers. Depending on the management algorithm, this streaming is done through standard HTTP or through multicast. The delivery servers act as proxies for the original HAS server: they store the content they receive from the distribution server and make it available to the video

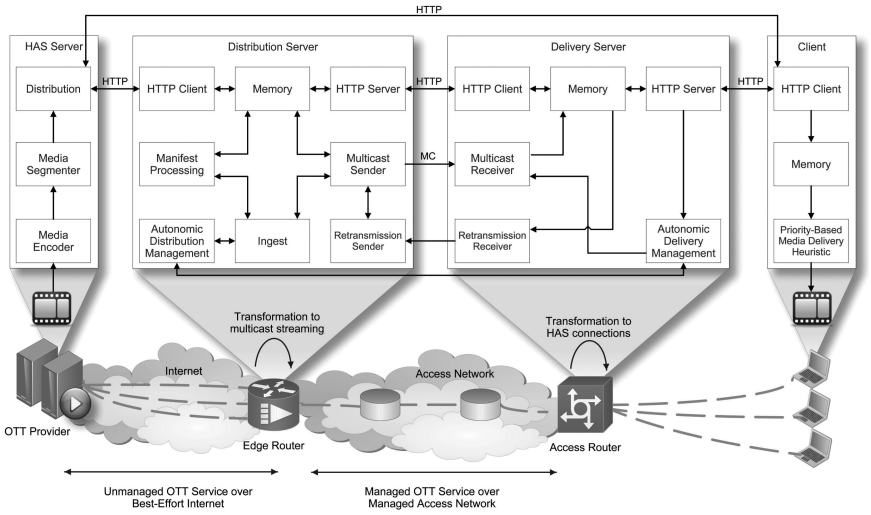



*Figure 2.4: Designed distributed architecture for HAS-enabled content delivery using multicast (MC), allowing a seamless integration with existing HAS-based delivery technologies. The lower part shows an example mapping of the proposed system to a network topology.*

clients. When clients now request a video, it can be downloaded from the caches of the delivery server, resulting in a considerable decrease in bandwidth consumption. In the subsequent sections, the two novel component types are discussed in more detail.

#### 2.4.1.1 Distribution server

The distribution server is responsible for distributing the video content pro-actively to the connected delivery servers over multicast. The distribution server connects to the HAS server through the *HTTP Client* and serves incoming requests via the *HTTP Server* component. Video data is provided by the *Memory* component, which acts as a cache. The *Manifest Processing* continuously polls the HAS server for new manifests at a configurable rate (e.g., every second). Another approach would be to deploy a push-based manifest update mechanism at the HAS Server informing the *Manifest Processing* when new manifests become available. This, however, requires adaptations to the HAS Server which would prevent the architecture to be loosely coupled with existing HAS deployments. When new segments are available and the channel is currently multicasted, the *Ingest* component will order the *Multicast Sender* to start multicasting the content in the selected qualities. Within one multicast tree, segments are pushed onto the tree in a chronological order as they appear in the video sequence. Whenever a manifest becomes available, it is pushed upon its associated multicast tree, taking into account the

condition that the first packets of all selected qualities of a segment have to be sent before sending a manifest first mentioning that segment. This ensures that every segment is available at the cache of the delivery server before the content can be requested by a video client. Since multicast over UDP is an unreliable transport mechanism, resilience measures need to be taken. The *Retransmission Sender* is responsible for handling retransmission requests from the delivery servers. The type of retransmission mechanisms are described later in this section. Each packet sent by the *Multicast Sender* is forwarded to the *Retransmission Sender* history, allowing the retransmission of a single packet.

The distribution server does not multicast all content. If this would be the case, this would lead to an explosion of the number of multicast trees that are being set up and hence a significant overhead. Moreover, multicasting content that is not being reused by other delivery servers is useless. For such content, it is more advisable to transfer it over HTTP. Instead, the distribution server only multicasts the most popular content to achieve the biggest decrease in the network load between distribution server and delivery servers. The *Distribution Management* and *Autonomic Delivery Management* components in the distribution server and delivery servers are responsible for this. The *Autonomic Delivery Management* components in the delivery servers send regular reports on the requested content, which is in turn used by the *Distribution Management* component in the distribution server. This component runs an algorithm to predict which segments will be requested in the future, based on the time dependency between segments from the same video: this algorithm is described in detail in Section 2.5.

#### 2.4.1.2 Delivery server

At the delivery server, the content received over multicast from the distribution server is stored locally in its cache by the *Memory* component. From then on the delivery server fulfills the role of local proxy for the HAS server towards the clients. Requests from the connected clients are received through the *HTTP Server* component. If a client request can be served from the cache, the segments are served from the delivery server. If not, the *HTTP Client* of the delivery server requests them from the distribution server or HAS server over HTTP.

When packet loss is detected (i.e. a gap in sequence numbers between two consecutive packets), an error message concerning the packet loss is reported. The *Memory* then contacts the *Retransmission Receiver* to request the retransmission of the reported packets. An in-depth description of the retransmission mechanism is provided in the next section.

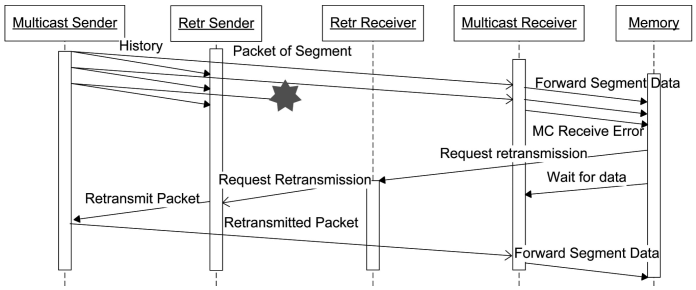



*Figure 2.5: Illustration of the multicast retransmission mechanism at the Delivery Server.*

## 2.4.2 Component interaction details

### 2.4.2.1 Handling of packet loss

As part of the data is multicasted over unreliable UDP-connections, it is crucial to implement resilience measures to account for packet loss. To address this, a retransmission mechanism was implemented supporting three different modes: HTTP fallback, UDP unicast and UDP multicast retransmissions. The HTTP fallback mode consists of requesting the complete video segment from the HAS server over HTTP. Although this guarantees a reliable delivery of the segment, the downside of this mechanism is the large overhead it introduces. If only one packet of the segment is lost, the complete segment needs to be downloaded again. Therefore, the use of this mechanism will only be benificial in absence of a UDP retransmission scheme. The UDP unicast retransmission mode implements a unicast-based retransmission mechanism, where the delivery server requests the retransmission of a single packet from the distribution server. This mechanism is triggered when the distribution server receives only a single request for retransmission from the delivery servers. Figure 2.5 details the handling of packet loss by the multicast retransmission mechanism. If the distribution server receives requests from multiple delivery servers, the UDP multicast mode of retransmission is triggered. In this case, the lost segments are multicasted to the different delivery servers by the *Retransmission Sender*. The advantage is of course that each delivery server will then receive the packets that were marked as lost.

### 2.4.2.2 Managed multicast-enabled video delivery

Figure 2.6 illustrates how the multicast channel management is introduced into the framework. Each time a delivery server receives a segment request from a client via the *HTTP Server*, this request is forwarded to the *Autonomic Delivery Management* component. Periodically, the *Multicast Sender* will request from the *Distribution Management* component which content it needs to multicast during the

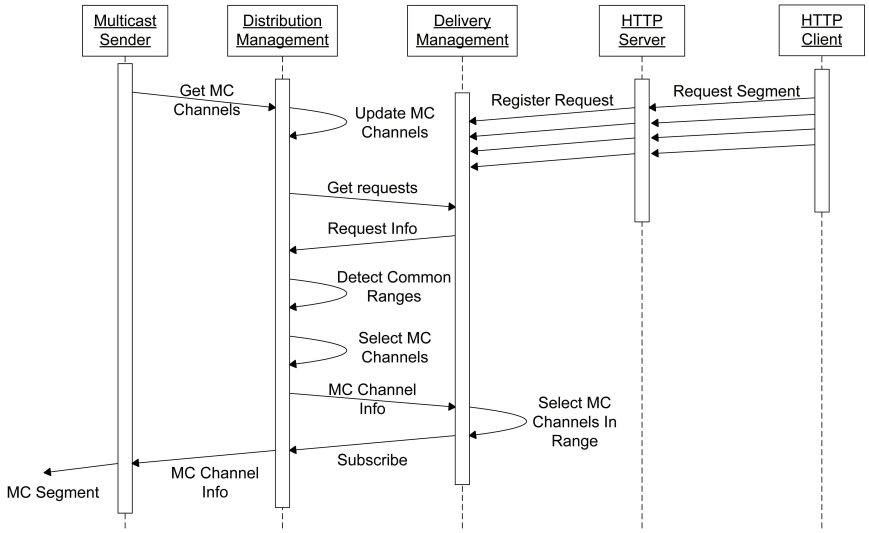



*Figure 2.6: Illustration of how the management algorithm at the distribution server selects which content to multicast and decides to which channels a delivery server should subscribe.*

next time slot. Each delivery server will be asked to forward the received requests during the last slot to the *Distribution Management* component. The management algorithm will detect the common ranges of these request sequences and select the most profitable multicast channels (as explained in Section 2.5). After this, the information concerning the multicasted content is forwarded to the *Autonomic Delivery Management* component. The selection algorithm at this component will then decide to which multicast channels it will subscribe, taking in mind the requests received during the last slot. Subscribing to too many channels will pollute the cache with segments that will never be requested, while too few subscriptions will lead to an increase in redundant HTTP-traffic. A more detailed description of the distribution management is discussed in the next section.

## 2.5 Autonomic Management of Multicast Streaming

In order to achieve a decrease of the consumed bandwidth between distribution and delivery servers, it is important to only multicast content that is actually required by the clients. For a Live TV service, this choice is evident: multicasting the live signal of the content will result in the targeted load decrease. However, in a realistic scenario, a mixture of TSTV and VoD will occur besides Live TV services. In this mix, several unicast TSTV or VoD connections with similar offset to the live moment can potentially also be mapped onto a single multicast connec-

tion if they would also result in important peaks in service consumption. Which unicast connections are suited for such a multicast grouping depends on the status of the caches and the future requests at the delivery servers. As such, the mapping of unicast connections is far less trivial than that of Live TV. In this section, a novel management algorithm is proposed to autonomously decide which unicast flows need to be mapped to a multicast connection and to select to which multicast groups a delivery server should subscribe. This scenario requires management of the multicasted content in order to benefit from a reduced bandwidth consumption in a HAS-based TSTV and VoD scenario since the clients no longer request the video segments in a synchronized way, as is the case with Live TV.

### 2.5.1 Autonomic delivery management

The caches located at the delivery servers allow temporal storage of the video streams and thus relax the need for the video flows to be fully synchronized. Hence, the requests for video content do not need to occur simultaneously in order to be mapped to a multicast connection, but can occur within a predefined time window of $W$ segments. The size of $W$ depends on the cache size and replacement algorithm used at the delivery servers. In order to find the best value for $W$, an autonomous selection algorithm is deployed on the delivery servers. The *Distribution Management* component periodically announces which content is to be multicasted. The *Autonomic Delivery Management* component is responsible for deciding which channels it subscribes to. Taking into account the limited cache size at the delivery server, subscribing to multicasted content which is not requested by clients is not beneficial. These segments substitute other popular segments that will be requested in the future, deteriorating cache hitrates. The *Autonomic Delivery Management* component tackles these problems by making a prediction of future requests and cache contents based on current requests and cache assessment.

Figure 2.7 shows how the subscription algorithm compares recent requests for segments with respect to the multicasted content and the current caching window $W$ at a certain time $t$. For video 1, the segment numbers to be multicasted next are $18$ and $28$, the current caching window is $8$ segment durations long and recently, there were requests for the segments $13$, $15$ and $23$ for video 1. Since the multicasted segments will still be available when the clients will request these, this delivery server subscribes to both multicast channels. For video 4, this is however not the case, the $23^{th}$ segment will be multicasted next and a request was received for segment $12$. This client will request the $23^{th}$ segment within $11$ segment durations, which exceeds the current caching window, hence the delivery server will not subscribe to this channel. If $S_r$ are the segment numbers of the request, $S_m$ the number of the multicasted segment and $W$ the number of segment durations a seg-

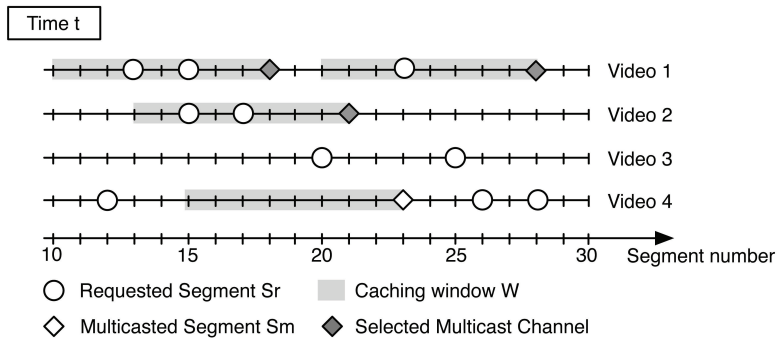



*Figure 2.7: Illustration of the subscription algorithm performed by the Autonomic Delivery Management component, subscribing to a multicast channel if there exists an $S_r$ where $S_m - S_r < W$.*

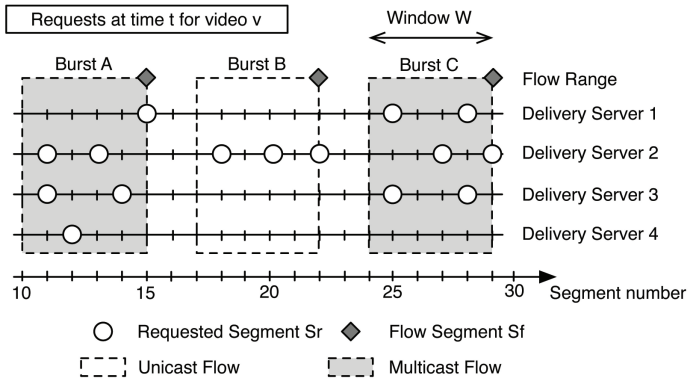



*Figure 2.8: Illustration of the mapping algorithm performed by the Distribution Management component.*

ment is predicted to reside in the cache, the delivery server subscribes to a specific multicast channel if there exists a $S_r$ for which $S_m - S_r < W$ holds. This decision is justified since it is predicted that the clients currently requesting segment $S_r$, will probably request segment $S_m$ within $S_m - S_r$ segment durations. When the current caching window $W$ is larger than this offset to the multicasted content, the segment $S_m$ will still be available in the cache. The size of $W$ is calculated as the weighted moving average of the recent cache residing times.

### 2.5.2 Distribution management

As multicasting video from the distribution server to the delivery servers is only useful when the video is served by multiple delivery servers, the *Distribution Man-*

*agement* component requires regular reports from the various delivery servers, to assess the popularity of the video channels. Figure 2.8 illustrates how this mapping is performed for each individual video. The requested segments within the assessment period are depicted here per delivery server. As shown, the algorithm maps multiple unicast video flows to a single multicast channel if (i) these video flows are requested in a burst (i.e. in the same time window $W$) and (ii) these requests are served by multiple delivery servers. As such, two request bursts (A and C) are mapped onto two distinct multicast channels since the requested segments (15 from delivery server 1, 11 and 13 from delivery server 2, 11 and 14 from delivery server 3 and 12 from delivery server 4) all lay within the caching window W of size 5 and the number of requesting delivery servers is 4, thus exeeding 2, fulfilling both condition (i) and (ii). But the burst in requests for segments (B) is not, since all requested segments (18, 20 and 22) within the window $W$ are requested by only one delivery server, thus condition (ii) does not apply. By mapping to multicast channels, a bandwidth reduction of $D - 1$ is achieved between the distribution server and delivery servers, where $D$ is the number of delivery servers that serve video flows in the time window $W$. A description of the algorithm is provided below in Algorithm 2.1. Each time the multicasted channels are reassessed, the recent clients request information is gathered from the delivery servers (line 3-5). Next to this request information, the caching window $W$ of each delivery server is retrieved. Using these, the *Distribution Management* can update the mapping of unicast connections to multicast channels, by multicasting the flows serving the highest number of delivery servers. For each requested video content, the requested segment numbers are ordered decreasingly in flows $\mathcal{F}$ (line 12). These flows are then grouped into flow ranges $\mathcal{F}_{range}$ with segment number $S_f$ based on the current caching window $W$, i.e. flows in $\mathcal{F}$ where the requested segment number $S_r$ satisfies the condition $S_r - S_f \leq W$ (line 25-36). For each range $\mathcal{F}_{range}$, the set delivery servers sharing the common flow range is stored in $\mathcal{D}_{range}$ (line 33). When the cardinality of this set does not exceed two, the range is added to the unicast flows, otherwise they are grouped into a multicast channel (line 15-19). The multicast trees that were created this way are communicated to the *Autonomic Delivery Management* component, which can now autonomously decide which multicast trees to join or leave depending on the number of clients requesting that content. Each update interval, the created multicast trees are re-evaluated, so to optimize the number of served delivery servers.

```
function UPDATEMULTICASTMAPPING(R)
    R ← φ
    for d ∈ D do
        R ← R ∪ {d → getRecentRequests()}
    end for
    (U, M) ← updateMulticastMapping(R)
end function
function UPDATEMULTICASTMAPPING(R, U, M)
    M ← φ
    U ← φ
    for v ∈ R → getVideos() do
        F ← getOrderedOnSegment(R_v)
        for f ∈ F do
            (F_range, D_range) ← getFlowsInRange(f, F)
            if sizeof(D_range) ≥ 2 then
                M ← M ∪ f
            else
                U ← U ∪ f
            end if
            F ← F\f
        end for
    end for
    return U, M
end function
function GETFLOWSINRANGE(flow, F, F_range, D_range)
    F_range ← φ
    D_range ← φ
    Sf = flow → nextSegment
    for f ∈ F do
        Sr = f → nextSegment
        d = f → deliveryServer
        if Sr − Sr ≤ W then
            F_range ← F_range ∪ f
            D_range ← D_range ∪ d
        end if
    end for
    return F_range, D_range
end function
```

*Algorithm 2.1: Popularity-based mapping from unicast to multicast channels, based on video popularities and cache assessment at the delivery servers.*

## 2.6 Performance Evaluation

This section is structured as follows: Section 2.6.1 discusses the implemented prototype and the obtained emulation results in a HAS-based Live TV scenario in a network topology of 1,000+ nodes. To be able to carry out larger scale experiments, a packet-based simulator was implemented for the extensive evaluation of the management algorithm which is presented in Section 2.6.2. All of the following results are averaged over 10 iterations, with the graphs showing the 95% confidence levels. Furthermore, ANOVA-confidence intervals are provided when the averages are close together. ANOVA (analysis of variance) provides a statistical test of whether or not the means of several groups are all equal.

### 2.6.1 Prototype evaluation

#### 2.6.1.1 Implementation details

A prototype of the architecture was implemented using the Apple Live Streaming protocol as an underlying HAS technique. The quality level selection heuristic used in the prototype is based upon an existing heuristic called Priority-Based Media Delivery [33] and decides which quality to download by considering several previously downloaded fragments through a weighted moving average. In the proposed architecture, caches are used in the *Memory* components of both distribution and delivery server. Least Recently Used (LRU) was used as cache replacement strategy.

The emulations were performed on the iLab.t Virtual Wall infrastructure[1], which consists of 100 servers interconnected by a non-blocking switch. The prototype was deployed on these Linux-based servers with following specifications: 2GHz AMD Dual Core CPU, 4GB RAM and Gbit network interfaces.

#### 2.6.1.2 Experimental setup

The network model illustrated in Figure 2.9 depicts a typical tree based access network of 1,022 nodes consisting of 1 HAS server, 1 distribution server, 20 delivery servers and 1,000 virtual video clients (mapped onto 20 physical clients) connected by gigabit links. The distribution server offers 5 live channels, each available in three qualities: 4Mbit/s, 2Mbit/s and 1Mbit/s. Each delivery server has a cache size of 1,280MB. A limit is put on the shared client links of 160Mbit/s, so clients are likely to switch qualities and to make sure all qualities are requested. The distribution of viewers over the five channels is set according to values measured on an actual broadcast TV [34] and follows a Zipf distribution with parameter $\beta$ equal to 1.7. The distribution of viewers over the different delivery servers is uniformly

[1] iLab.t Virtual Wall - http://ilabt.iminds.be/iminds-virtualwall-overview

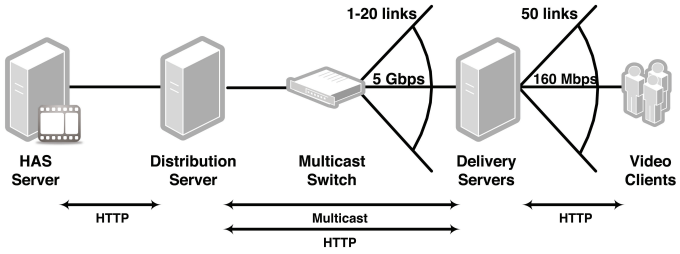



*Figure 2.9: Emulated network topology modeling a tree-based access network of 1,000+ nodes.*

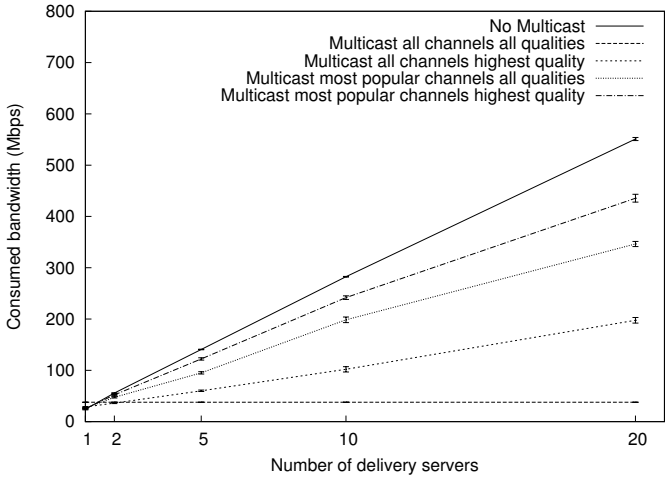
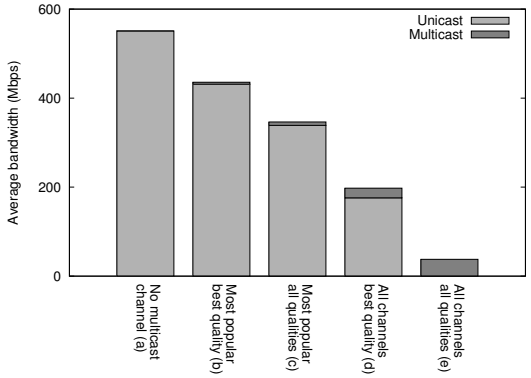



*Figure 2.10: Impact of the different multicast strategies on the bandwidth. Depending on the multicast strategy, multicasting content is beneficial starting from 2 delivery servers. Confidence levels are shown on the graphs.*

random, which implies there are no significant local popularity differences. For these experiments it is investigated what the impact is of the number of delivery servers on the average consumed bandwidth for the different multicast strategies without packet loss. Furthermore, the impact is evaluated of the retransmission strategy on the average retransmission and total consumed bandwidth when the links are subject to packet loss.

#### 2.6.1.3 Results description

Figure 2.10 shows the impact of the number of delivery servers and the multicasted channels on the consumed bandwidth between distribution and delivery servers. The effect of multicasting all channels and all qualities is a reduced bandwidth




| | MC $\bar{x}$ $Mbps$ | UC $\bar{x}$ $Mbps$ | UC $\sigma$ $Mbps$ |
|---|---|---|---|
| a | 0 | 550.95 | 4.79 |
| b | 4.33 | 431.28 | 12.17 |
| c | 7.57 | 338.78 | 8.04 |
| d | 21.63 | 176.01 | 8.76 |
| e | 37.85 | 0 | 0 |

*Figure 2.11: Impact on the average consumed bandwidth of the different multicast strategies. Next to the graph is a table with the average bandwidths and standard deviations. For the multicast scenario no standard deviations are shown since they are all zero.*

when more than two delivery servers request the video content (one-way ANOVA showed a significant difference between both strategies: $\rho = 0$, $F = 4.327e^3$). When only the most popular channel is multicasted, with the other channels still being served through unicast, the bandwidth reduction is obviously less significant (one-way ANOVA: $\rho = 1.624e^{-5}$, $F = 36.68$). Figure 2.11 shows the average consumed bandwidth for a variety of multicasting strategies in a network with 20 delivery servers and the standard deviations for these configurations. All differences in average bandwidth are significant with all $\rho_{i,j} < 4.85e^{-11}$ (where $i$ and $j$ denote different multicasting strategies). Multicasting all qualities of each channel uses only 7% of the consumed bandwidth when nothing multicasted. Multicasting the best quality of the most popular channel results in a 20% bandwidth reduction.

For the following experiments, both multicast and unicast loss is introduced, respectively for the loss introduced on the link between the distribution server and the multicast switch and on the links between the multicast switch and the delivery servers. Multicast loss will cause all delivery servers to experience the same amount of packet loss, while unicast loss will affect a specific delivery server. The amount of loss that is introduced in these experiments is set at 1%. Both types of loss are combined with UDP packet retransmissions over unicast and multicast into 4 test scenarios. The consumed bandwidth on the link between distribution server and multicast switch is displayed in Figure 2.12. For a single delivery server, all 4 scenarios are similar, as there is no difference between introducing loss on both links and between retransmission strategies. Unicast retransmits and multicast retransmits for unicast loss show a linear correlation between the increase in consumed bandwidth and the number of delivery servers. Multicast retransmits for multicast loss use a constant bandwidth, independent of the number of deliv-

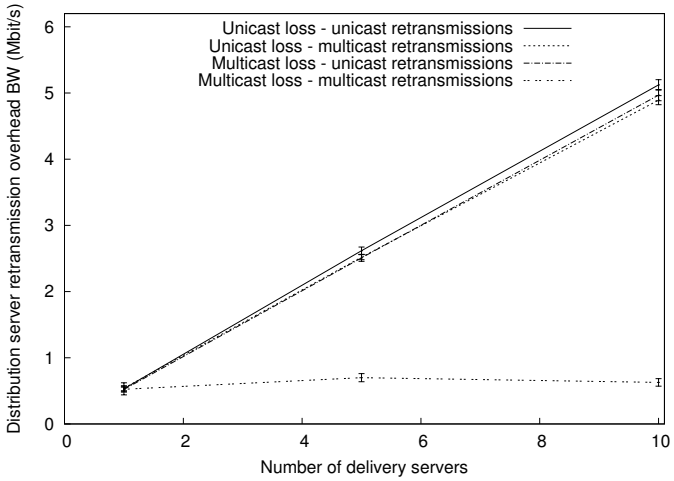



*Figure 2.12: Impact on bandwidth of the different retransmission strategies on consumed bandwidth at distribution server. Confidence levels are shown on the graphs.*

ery servers, as duplicate requests are ignored. The bandwidth used for receiving retransmissions on the link of the first delivery server is shown in Figure 2.13. In this case, multicast retransmits for unicast loss result in an additional bandwidth use on this link, as retransmits requested by other delivery servers are received here as well. These results show that it would be beneficial to adapt the retransmission strategy according to the type of loss, in order to optimise the consumed bandwidth.

The percentage of loss introduced was changed for some experiments (with 5 delivery servers). Loss can also be handled by discarding the received file, and not requesting any retransmits, but instead using the HTTP fallback mechanism to request the entire segment/file again over HTTP. This mechanism was evaluated for the scenario with 5 delivery servers and all content multicasted. Figure 2.14 compares the total outgoing bandwidth on the distribution server for these HTTP full-file retransmits with the UDP retransmission methods (both with unicast and multicast-enabled retransmissions). The bandwidth usage is almost doubled for the HTTP full-file retransmissions, as, regardless of the loss percentage, almost all multicasted files have at least one lost packet, which causes the full file to be downloaded again.

### 2.6.2 Management algorithm evaluation

In order to evaluate the proposed architecture and management algorithms in a realistic TSTV-scenario, a large amount of nodes need to be emulated. Since the

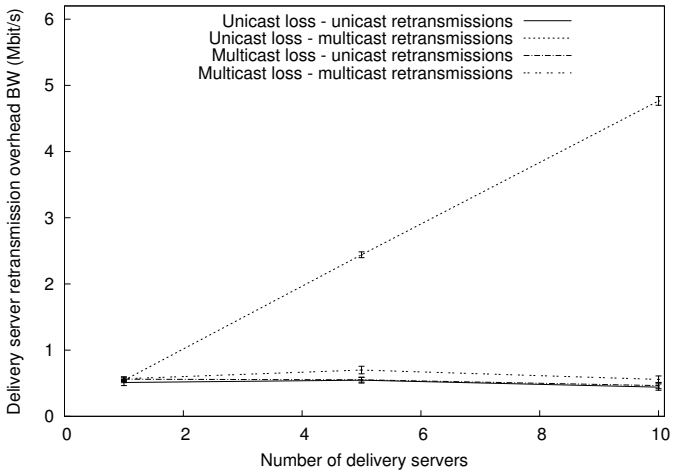
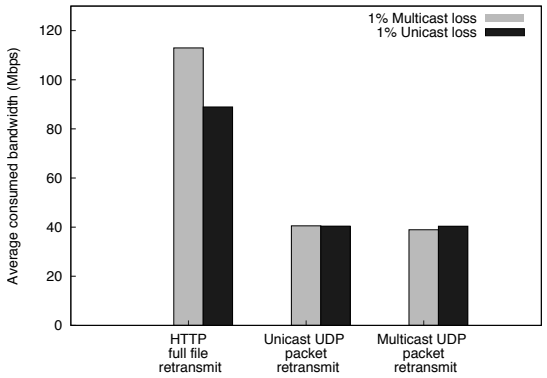



*Figure 2.13: Impact on bandwidth of the different retransmission strategies on consumed bandwidth at delivery servers. Confidence levels are shown on the graphs.*




| | $\bar{x}$ $Mbps$ | $\sigma$ $Mbps$ |
|---|---|---|
| HTTP retr. MC loss | 88.92 | 2.91 |
| HTTP retr. UC loss | 112.97 | 4.35 |
| UC retr. MC loss | 40.39 | 0.50 |
| UC retr. UC loss | 40.52 | 0.49 |
| MC retr. MC loss | 40.38 | 0.51 |
| MC retr. UC loss | 38.94 | 0.51 |

*Figure 2.14: Total average consumed bandwidth on link between distribution server and delivery servers for various loss settings and retransmission methods. Next to the graph is a table with the average bandwidths and standard deviations.*

employed testbed does not support such large size topologies, a simulator was implemented using NS-3 [35]. A packet-based simulator has been developed, which implements all aforementioned functionality of the emulated framework with additional support for multicast management, both at the distribution and delivery server.

#### 2.6.2.1 Experimental setup

Using several studies modelling the behaviour of TSTV-viewers [36, 37] and real measurements by DVR vendors [5, 38], a client request model was constructed. The most important assumptions here are: (i) the channel popularity distribution is Zipf-like, where the top 10% of channels accounts for nearly 80% of the viewers, (ii) only 20-30% of viewers watch the programs live in prime-time and (iii) more than 50% of the viewers watches the program the same day. Similar to the experiments carried out in Section 2.6.1, the distribution of viewers over the channels follows a Zipf distribution with parameter $\beta$ equal to 1.7 [39], and no local popularity differences are modeled among delivery servers. The simulation time was set to 3 hours: this entails the simulation of the consumption of 3 consecutive television programs of 1 hour each. The first hour of the simulation was ignored and solely used to fill the caches with realistic content. Several parameters were varied during these simulations: cache sizes, number of delivery servers and the number of available multicast groups. These tests were performed using 20 videos of different qualities (i.e. 4Mbps, 2Mbps and 1Mbps), 50 clients per delivery server and a bandwidth limit on the link between clients and delivery servers of 160Mbps, causing congestion and clients switching quality levels, leading to a decrease in perceived QoE [40]. Four configurations were tested: (i) proxy-enabled HAS, where all segments are sent to the delivery servers over HTTP-connections (ii) unmanaged delivery, where the live moment of the most popular video channels are multicasted and all delivery servers subscribe to each of these multicast channels (iii) unmanaged distribution, where each delivery server autonomously decides to which multicast channel it subscribes, based on the recent requests and (iv) managed multicast, where the most popular content (Live TV or TSTV) is multicasted and each delivery server autonomously subscribes to them. For these experiments it is evaluated what the impact is of the number of delivery servers, the cache sizes and the number of multicast channels on the average consumed bandwidth for the different multicast strategies.

#### 2.6.2.2 Results description

Figure 2.15 shows the combined (unicast and multicast traffic) consumed bandwidth on the link between the distribution server and the delivery servers and how this bandwidth consumption can be reduced by taking management measures at

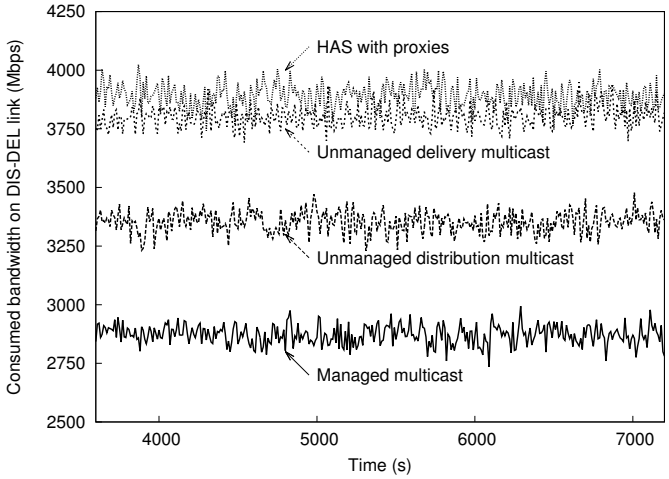



*Figure 2.15: Impact of multicast strategy on consumed bandwidth on link between distribution and delivery servers, measured with 40 delivery servers, cache sizes of 2,560MB, 20 multicast channels and 20 video channels.*

both the delivery and distribution server. The proposed architecture is also compared with a direct downloading scheme where no intermediate servers are present. *Unmanaged Distribution Multicast* reduces the bandwidth between distribution and delivery servers by letting the delivery servers autonomously decide to which multicast channels they will subscribe, in contrast to *Unmanaged Delivery Multicast*, where each delivery server subscribes to all available multicast channels. For 2,000 clients, the management of delivery subscriptions leads to a bandwidth reduction of approximately 560Mbps compared to the setup without multicasting. Not only managing delivery subscription, but also proactively deciding which content to multicast by the *Distribution Management*, leads to an additional reduction of approximately 400Mbps. This is an improvement of 25% with respect to the HAS architecture with proxies and of 70% compared to a direct downloading scheme.

Figure 2.16 shows the impact of the number of delivery servers in the setup on the average consumed bandwidth between distribution and delivery servers. When the number of delivery servers is equal to 2, the original HAS setup with proxies leads to lower bandwidth consumption than the multicast enhanced solution (one-way ANOVA: $\rho = 2.364e^{-8}$, $F = 88.01$). With only one delivery server, the managed multicast solution consumes almost equal bandwidth (one-way ANOVA: $\rho = 0.922$, $F = 9.84e^{-1}$), since content is only multicasted when it is requested by at least 2 delivery servers. But when the number of delivery servers exceeds 5, the managed multicast solution yields lower bandwidth consumption (one-way

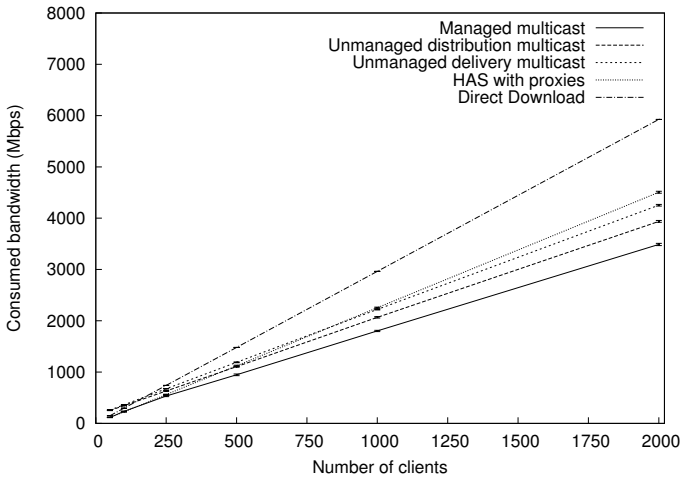



*Figure 2.16: Impact of number of delivery servers (with 50 clients per delivery server) and multicast management strategy on average consumed bandwidth, measured with cache sizes of 2,560MB, 20 multicast channels and 20 video channels. Confidence intervals are shown on the graphs.*

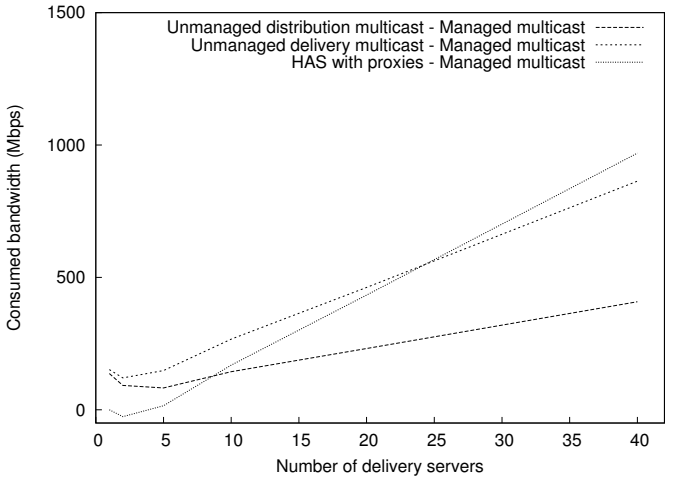



*Figure 2.17: Impact of number of delivery servers and multicast strategy on average consumed bandwidth, measured with cache sizes of 2,560MB, 20 multicast channels and 20 video channels.*

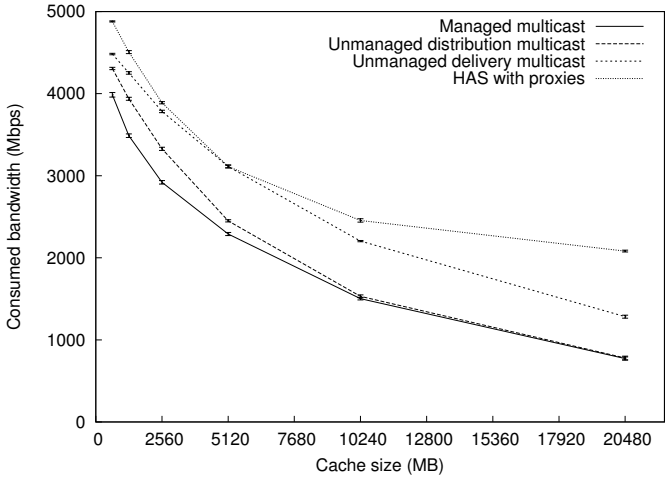



*Figure 2.18: Impact of different multicasting strategies and cache sizes on average consumed bandwidth, measured with 40 delivery servers, 20 multicast channels and 20 video channels. Confidence intervals are shown on the graphs.*

ANOVA: $\rho = 6.51e^{-2}$, $F = 3.86$ for 5 delivery servers and $\rho = 2.36e^{-12}$, $F = 355.61$ for 10 delivery servers). The difference in consumed bandwidth increases when the number of delivery servers increases as is illustrated in Figure 2.17.

Figure 2.18 shows the impact of the cache sizes at the delivery servers on the consumed bandwidth. With small caches, the difference in consumed bandwidth between the managed multicast scenario and both the unmanaged scenarios is considerably large (one-way ANOVA: all $\rho < 7.32e^{-13}$ for 640MB caches), but when the cache size increases the difference between the managed multicast and unmanaged distribution (where all content is multicasted but the delivery servers autonomously subscribe to certain channels) becomes less significant (one-way ANOVA: $\rho = 5.17e^{-2}$, $F = 4.42$ for 10,240MB caches and $\rho = 6.94e^{-1}$, $F = 0.16$ for 20,480MB caches). This can be explained by the fact that the caches are now large enough to store a considerable part of the video content (approximately 1,024 segments in the highest quality). Since the delivery servers only subscribe to the content that will be requested, the usage of the available cache size is optimized. This leads a low number of HTTP requests, comparable to the number of requests in the managed multicast scenario. Figure 2.19 illustrates the impact of the cache size on the average cache hitrate for the different management scenarios. As the cache sizes grow, the average hitrate at the delivery servers increases, where the increase is fairly uniform across the different configurations.

Figure 2.20 displays the impact of the number of available multicast channels on the consumed bandwidth. When no multicast channels are available, all

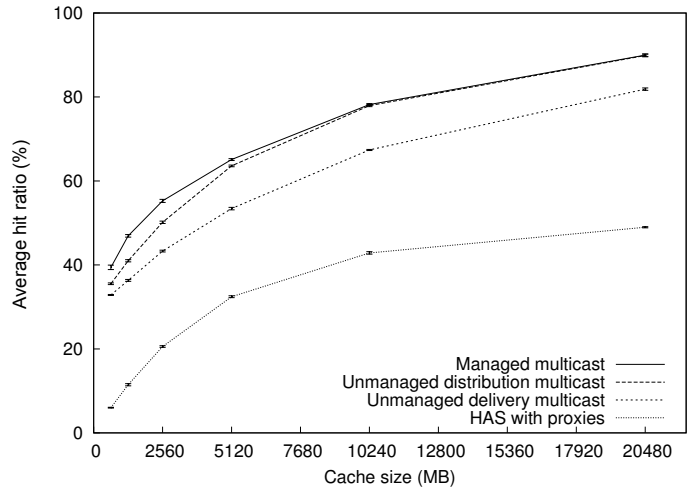



*Figure 2.19: Impact of different multicasting strategies and cache sizes on average cache hitrate, measured with 40 delivery servers, 20 multicast channels and 20 video channels. Confidence intervals are shown on the graphs.*

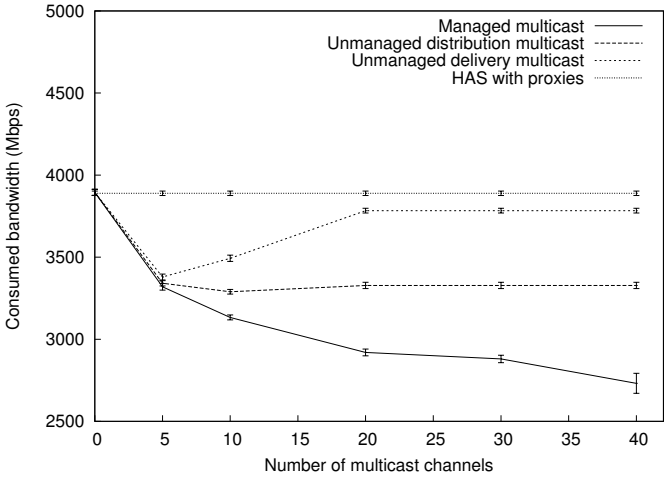



*Figure 2.20: Impact of different multicasting strategies and number of multicast channels on average consumed bandwidth, measured with 40 delivery servers, 20 video channels and cache sizes of 2,560MB. Confidence intervals are shown on the graphs.*

managed scenarios converge to the proxy-enabled HAS solution. With increasing number of multicast channels, the bandwidth consumption keeps decreasing in the managed multicast solution. With only 5 multicast channels available, all 3 multicast scenarios yield almost equal bandwidth consumption (one-way ANOVA: $\rho = 1.47e^{-1}$, $F = 2.31$ when comparing managed multicast with unmanaged distribution multicast). But when the number of multicast channels exceeds 5, they diverge, since non-popular content is now multicasted in the unmanaged distribution scenario, polluting the caches with unrequested content at the delivery servers (one-way ANOVA: $\rho = 6.15e^{-9}$, $F = 153.69$ for 10 multicast channels). When the number of multicast channels is equal to the number of video channels both unmanaged distribution and unmanaged delivery stabilize, since additional multicast channels are not used, while bandwidth consumption in the managed multicast solution keeps decreasing.

## 2.7 Conclusions

In this chapter, the merits are characterized of a novel HTTP Adaptive Streaming (HAS)-based multimedia architecture, allowing a decrease of the consumed bandwidth, through a combination of caching and multicast streaming. Two additional component types were added to a traditional HAS-based architecture: a distribution server and multiple delivery servers. The proposed multicast-enabled architecture is compared with a traditional HAS set-up, which uses only unicast connections. The experiments show that the obtained bandwidth reduction factor, when using multicasting and caching, is proportional to the number of connected delivery servers, even when multiple HAS qualities are multicasted. For example, the proposed architecture requires four times less bandwidth than traditional HAS-based approaches for a moderate network size consisting of eight delivery servers. Additionally, retransmission experiments showed that the implemented multicast retransmission mechanism also provides a retransmission bandwidth reduction compared to unicast retransmission, both for multicast and unicast loss. As such, it is demonstrated how the proposed multicast-enabled architecture is more scalable without losing robustness in delivering HAS-based Live TV. Furthermore, the architecture was extended with autonomic multicast management in order to support Time-Shifted TV and Video on Demand services. The algorithm deployed at the distribution servers intelligently selects which content needs to be multicasted by making a remote assessment of the cache states at the delivery servers. The autonomic subscription algorithm at each delivery server makes predictions on future cache state and expected requests. Using these predictions they then autonomically decide to which multicast channels they should subscribe. The experiments show that the obtained bandwidth reduction in a Time Shifted TV (TSTV)-scenario is higher than in the unmanaged scenario. For example, the managed, multicast-

enabled delivery of TSTV-services requires approximately 23% less bandwidth compared to the unmanaged multicast-enabled delivery and 25% less when compared to proxy-enabled HTTP HAS delivery, for a network size of 40 delivery servers, cache sizes of 2,560MB and 20 multicast channels. These differences even increase when the number of delivery servers grows. Increasing the number of available multicast channels and cache sizes at the delivery servers also decreases the bandwidth consumption. Thus, the results show that the proposed multicast-enabled HTTP Adaptive Streaming framework is able to successfully eliminate the bottleneck between the video servers and the HTTP proxies.

# References

[1] *BBC iPlayer*. www.bbc.co.uk/iplayer - Last Accessed on 24 February 2012.

[2] *YouTube - Broadcast Yourself*. www.youtube.com - Last Accessed on 24 February 2012.

[3] *Hulu*. www.hulu.com - Last Accessed on 24 February 2012.

[4] *Netflix*. www.netflix.com - Last Accessed on 24 February 2012.

[5] Nielsen. *State of the Media: Mobile Media Report Q3 2011*. http://www.nielsen.com/us/en/insights/reports-downloads/2011/state-of-the-media–mobile-media-report-q3-2011.html - Last Accessed on 24 February 2012.

[6] Knowledge Networks. *Over-The-Top (OTT) Video viewing surges*. http://www.knowledgenetworks.com/news/releases/2011/090811_ott-video.html - Last Accessed on 24 February 2012.

[7] *Live - YouTube*. www.youtube.com/live - Last Accessed on 24 February 2012.

[8] *Amazon.com Instant Video*. www.amazon.com/gp/video/ontv/ontv - Last Accessed on 24 February 2012.

[9] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550 (Standard), July 2003. Updated by RFCs 5506, 5761, 6051, 6222. Available from: http://www.ietf.org/rfc/rfc3550.txt.

[10] H. Schulzrinne, A. Rao, and R. Lanphier. *Real Time Streaming Protocol (RTSP)*. RFC 2326 (Proposed Standard), April 1998. Available from: http://www.ietf.org/rfc/rfc2326.txt.

[11] S. Winkler, A. Sharma, and D. McNally. *Perceptual video quality and blockiness metrics for multimedia streaming applications*. In Proceedings of the International Symposium on Wireless Personal Multimedia Communications, pages 547–552, 2001.

[12] M. Saxena, U. Sharan, and S. Fahmy. *Analyzing video services in Web 2.0: a global perspective*. In Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV '08, pages 39–44, 2008.

[13] Microsoft. *Microsoft Smooth Streaming: The Official Microsoft IIS Site*. Available from: http://www.iis.net/download/SmoothStreaming.

[14] E. R. Pantos and W. May. *Internet Draft - HTTP Live Streaming*, 2011. Available from: http://tools.ietf.org/html/draft-pantos-http-live-streaming-07.

[15] Adobe. *HTTP Dynamic Streaming: Flexible Delivery of on-demand and live video streaming*. Available from: http://www.adobe.com/products/httpdynamicstreaming/.

[16] A. Begen, T. Akgul, and M. Baugher. *Watching Video over the Web: Part 1: Streaming Protocols*. Internet Computing, IEEE, 15(2):54 –63, 2011.

[17] A. Begen, T. Akgul, and M. Baugher. *Watching Video over the Web: Part 2: Applications, Standardization, and Open Issues*. Internet Computing, IEEE, 15(3):59 –63, 2011.

[18] J.-S. Leu and S.-F. Chen. *TRASS: A transmission rate-adapted streaming server in a wireless environment*. International Journal of Communication Systems, 24(7):852–871, 2011.

[19] Y. Liu, Y. Guo, and C. Liang. *A survey on peer-to-peer video streaming systems*. Peer-to-peer Networking and Applications, 1(1):18–28, 2008.

[20] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. *Youtube traffic characterization: a view from the edge*. In Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, IMC '07, pages 15–28, 2007.

[21] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. *I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system*. In Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, pages 1–14, 2007.

[22] M. Mushtaq and T. Ahmed. *Enabling Cooperation between ISPs and P2P Systems toward IPTV Service Delivery*. In Proceedings of the 7th IEEE Consumer Communications and Networking Conference (CCNC), pages 1–6, jan. 2010.

[23] B. Ben Moshe, A. Dvir, and A. Solomon. *Analysis and optimization of live streaming for over the top video*. In In proceedings of the IEEE Consumer Communications and Networking Conference (CCNC), pages 60–64, jan. 2011.

[24] N. Bouten, S. Latré, W. Van de Meerssche, K. De Schepper, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. *An Autonomic Delivery Framework for HTTP Adaptive Streaming in Multicast-enabled Multimedia Access Networks*. In Proceedings of the 5th IEEE/IFIP Workshop on Distributed Autonomous Network Management Systems (DANMS 2012), pages 1248–1253, april 2012.

[25] X. Zhang and H. Hassanein. *Video on-demand streaming on the Internet x2014; A survey*. In Communications (QBSC), 2010 25th Biennial Symposium on, pages 88–91, may 2010.

[26] F. de Asís López-Fuentes. *P2P video streaming combining SVC and MDC*. Applied Mathematics and Computer Science, 21(2):295–306, 2011.

[27] H. Schwarz, D. Marpe, and T. Wiegand. *Overview of the scalable video coding extension of the H.264/AVC standard*. In IEEE Transactions on Circuits and Systems for Video Technology In Circuits and Systems for Video Technology, pages 1103–1120, 2007.

[28] A. M. Al-Naamany and H. Bourdoucen. *TCP Congestion Control Approach for Improving Network Services*. Journal of Network and Systems Management, 13:1–6, 2005.

[29] T. Ahmed and M. Mushtaq. *P2P Object-based adaptivE Multimedia Streaming (POEMS)*. Journal of Network and Systems Management, 15:289–310, 2007.

[30] T. Miyoshi and K. Sekiya. *Efficient Transfer Method for On-demand Video Delivery Based on Streaming Packet Analysis*. In Proceedings of the First ACIS/JNU International Conference on Computers, Networks, Systems and Industrial Engineering, pages 141–146, may 2011.

[31] Y.-S. Yu, C.-K. Shieh, C.-H. Lin, and S.-Y. Wang. *P2PVR: A playback offset aware multicast tree for on-demand video streaming with VCR functions*. Journal of Systems Architecture, 57:392–403, April 2011.

[32] Information technology MPEG systems technologies. *Part 6: Dynamic adaptive streaming over HTTP (DASH)*. ISO/IEC DIS 23001-6, 2011.

[33] T. Schierl, Y. Sanchez de la Fuente, R. Globisch, C. Hellge, and T. Wiegand. *Priority-based Media Delivery using SVC with RTP and HTTP streaming*. Multimedia Tools and Applications, 55:227–246, 2011.

[34] T. Wauters, J. De Bruyne, L. Martens, D. Colle, B. Dhoedt, P. Demeester, and K. Haelvoet. *HFC Access Network Design for Switched Broadcast TV Services*. IEEE Transactions on Broadcasting, 53(2):588–594, june 2007.

[35] *NS-3*. http://www.nsnam.org/ - Last Accessed on 24 February 2012.

[36] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatriain. *Watching television over an IP network*. In Proceedings of the 8th ACM SIGCOMM conference on Internet measurement, pages 71–84, 2008.

[37] Y. Liu, Liu and G. Simon, Simon. *Distributed delivery system for time-shifted streaming systems*. In Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks, pages 276–279, 2010.

[38] Nielsen. *How DVRs Are Changing the Television Landscape*. http://blog.nielsen.com/nielsenwire/media_entertainment/how-dvrs-are-changing-the-television-landscape/ - Last Accessed on 24 February 2012.

[39] L. A. Adamic and B. A. Huberman. *Zipf's law and the Internet*. Glottometrics, 3(1):143–150, 2002.

[40] S. Fernandes, J. Kelner, and D. Sadok. *An adaptive-predictive architecture for video streaming servers*. Journal of Network and Computer Applications, 34(5):1683 – 1694, 2011.

# 3

# In-Network Quality Optimization for Adaptive Video Streaming Services

**N. Bouten, S. Latré, J. Famaey, W. Van Leekwijck, F. De Turck.**

**Published in IEEE Transactions on Multimedia, December 2014.**

*While the previous chapter focussed on the resource optimization for live HTTP Adaptive Streaming (HAS) delivery, this chapter tackles the quality management for managed HAS Video on Demand (VoD) delivery. A major obstacle for the adoption of HAS technologies in managed networks is the purely client-driven design of current HAS approaches, which leads to excessive quality oscillations, suboptimal quality selection, and the inability to enforce management policies. Moreover, the provider has no control over the quality that is provided, which is essential when offering a managed service. This chapter tackles these challenges and facilitates the adoption of HAS in managed networks. Specifically, several centralized and distributed algorithms and heuristics are proposed that allow nodes inside the network to steer the HAS client's quality selection process. The algorithms are able to enforce management policies by limiting the set of available qualities for specific clients. Additionally, simulation results show that by coordinating the quality selection process across multiple clients, the proposed algorithms significantly reduce quality oscillations with a factor 5 and increase the average delivered video quality with at least 14%.*

## 3.1 Introduction

The increasing popularity of Over-The-Top (OTT) multimedia services has led to the widespread adoption of HTTP-based streaming protocols. Such protocols have many advantages compared to traditional streaming methods, such as reuse of existing HTTP infrastructure (e.g., servers, proxies and caches), reliable transmission and firewall compatibility. Originally, progressive download techniques were used, allowing the user to start consuming the content after a few seconds of buffering. However, progressive download methods cannot cope with congestion, the highly fluctuating throughput of mobile networks or diverging characteristics of devices and networks. To overcome said problems, a new generation of HTTP-based streaming protocols, collectively referred to as HTTP Adaptive Streaming (HAS), was introduced. The offered content is split into a set of temporal segments, which are encoded at multiple bit rates. In traditional HAS, a rate adaptation algorithm, deployed at the client, is then used to select the bit rate of each segment, based on the current network conditions, buffer status and device capabilities.

State-of-the-art HAS solutions embed the rate adaptation algorithm inside the client application. This allows the client to independently choose its playback quality and prevents the need for intelligent components inside the network, which are the main reasons HAS is used in OTT scenarios. However, academia and industry are showing a growing interest in the use of HAS in managed networks [1][1,2], for example by optimizing the delivery by applying in-network bitrate adaptation[3] or by deploying IP multicasting to ease the distribution of linear TV HAS services [2][4]. The extensive content catalogue and increased flexibility in terms of supported devices of these OTT-services (e.g., YouTube, Hulu, Netflix) but delivered over the managed network, could greatly benefit both the provider and the end-user. However, in such environments, a purely client-driven approach has several significant disadvantages. First, the lack of coordination among clients leads to competing behavior among those clients, resulting in incorrect throughput estimations, causing excessive quality oscillations and suboptimal decisions [3, 4], negatively impacting QoE [5]. Second, management policies, such as user subscription constraints and guarantees on the delivered quality, cannot be easily enforced [6, 7]. In order to facilitate adoption of HAS for the delivery of multimedia services in a managed environment, these challenges should be tackled.

A straightforward solution to the resource scarcity affecting streaming services could be to increase the physical capacity of the delivery network. These up-

[1] http://www.juniper.net/us/en/local/pdf/solutionbriefs/3510463-en.pdf

[2] http://www.rgbnetworks.com/pdfs/RGB-Velocix_Adaptive_Streaming_CDN_White_Paper_0911-01.pdf

[3] http://www.cachelogic.com/vx-portfolio/solutions/velocixeve

[4] http://www.velocix.com/vx-portfolio/solutions/video-optimised-architecture

dates are however associated with high costs for the service provider, while an in-network optimization based solution does not affect these infrastructure costs. Since technologies (e.g. the advent of Ultra High Definition Television streaming) are constantly evolving, frequent infrastructure updates are required to cope with the ever increasing traffic demands. Physical infrastructure upgrades are time-consuming. Therefore, there should be a coexistence of both approaches to deal with future demand by intelligently managing resources in attendance of physical capacity updates.

This chapter proposes a hybrid approach where the rate adaptation algorithm is steered by an in-network component to address the aforementioned challenges. It is deployed on intermediary proxies and supports client-side rate adaptation algorithms by dynamically limiting the possible set of bit rates to select from. Currently an operator's multimedia delivery network typically contains several transparent caches and QoE measurement platforms which interpret HTTP headers and reconstruct HTTP adaptive streaming sessions in order to evaluate the end-to-end QoE. These platforms can be extended to not only measure the QoE, but also optimize the QoE by performing in-network quality optimization, thus requiring only limited extensions to the already available infrastructure. The proposed hybrid approach allows clients to still react upon sudden network changes or scarcity in device resources, while increasing the overall quality and stability. Moreover, it can enforce a wide range of management policies, allowing providers to specify priorities when allocating resources to a certain group of users. This translates into several concrete contributions. First, the in-network rate adaptation problem is formally defined. Second, an optimal centralized algorithm is proposed that solves the problem as an Integer Linear Programming (ILP). Third, a scalable variant of the algorithm is introduced that can be distributed across multiple logically hierarchical intermediary proxies. Finally, a heuristic with significantly lower computational complexity is proposed.

The remainder of this chapter is structured as follows. Section 3.2 lists and discusses state of the art research on client-based and in-network HAS rate adaptation. Subsequently, the in-network rate adaptation problem is formally defined in Section 3.3. Sections 3.4 and 3.5 describe and evaluate the three algorithms proposed to solve this problem respectively. Finally, Section 3.6 concludes the chapter.

## 3.2 Related Work

The increased popularity of video consumption over the Internet has led to the development of a range of protocols that allow adaptive video streaming over HTTP. Some of the major players have introduced their proprietary protocols such

as Microsoft's Silverlight Smooth Streaming[5], Apple's HTTP Live Streaming[6] and Adobe's HTTP Dynamic Streaming[7]. More recently, a standardized solution has been proposed by MPEG, called Dynamic Adaptive Streaming over HTTP (DASH) [8]. Although differences exist between these implementations they are based on the same basic principles: a video is split up into temporal segments which are encoded at different quality rates, the autonomic video client heuristic then dynamically adapts the quality, based on metrics such as average throughput, delay and jitter. The drawback of this approach is of course that all control lays in the hands of the clients which strive to maximize their individual quality. From the providers perspective however, other factors such as minimization of costs and prioritization of users with higher subscription levels are of equal importance. Current HAS approaches do not support intervention during the quality assignment process which is fully dominated by the clients. The approach presented in this chapter therefore focuses on managing the quality for HAS by the service provider.

The performance of HAS-based services can be improved by applying changes both at the client and the delivery network. Each commercial HAS implementation comes with a proprietary client heuristic. *Akhshabi et al.* compare several commercial and open source HAS players and indicate significant inefficiencies in each of them, such as frequent oscillations and unfairness when the number of competing clients increases [3]. Several heuristics have been proposed in literature as well, each focussing on a specific deployment. *Liu et al.* discuss a video client heuristic that is suited for a Content Delivery Network (CDN) by comparing the expected segment fetch time with the experienced segment fetch time to ensure a response to bandwidth fluctuations in the network [9]. *Andelin et al.* provide a heuristic which was specifically designed for Scalable Video Coding (SVC) and using a slope to define the trade-off between downloading the next segment and upgrading a previously downloaded segment [10]. In previous work [11] [12], the authors evaluated different client heuristics both for Advanced Video Coding (AVC) and SVC, applying optimizations such as pipelined and parallel download scheduling. Several of the aforementioned authors indicate the impact of competing HAS clients on the quality oscillations, which are known to have a negative impact on Quality of Experience (QoE) [5]. Furthermore, most of the commercial client heuristics require a considerably large buffer to be able to react to network changes. This chapter therefore aims at controlling the quality by introducing global QoE management, reducing drastically the number of quality oscillations and allowing to reduce the required buffer size. The presented approach is applicable to both AVC and SVC.

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

[6]Apple HTTP Live Streaming - http://tools.ietf.org/html/draft-pantos-http-live-streaming-13

[7]Adobe HTTP Dynamic Streaming - http://www.adobe.com/products/hds-dynamic-streaming.html

An autonomic delivery framework for HAS-based Live TV and Time Shifted TV (TSTV) was presented in previous work [13] [2] which allows to reduce the consumed bandwidth by grouping unicast HAS sessions sharing the same content into a single multicast session. However, for Video on Demand (VoD) HAS sessions, the content is more diverse and only few sessions are potentially shared among multiple users. This prevents them to be grouped into a shared multicast session and therefore prevents them from being delivered in a scalable manner. In [14], an overview of interesting use cases for applying SVC in a network environment are presented, among which the graceful degradation of videos when the network load increases. 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. Similar to this approach, *Latré et al.* proposes an in-network rate adaptation algorithm, responsible for determining which SVC quality layers should be dropped in combination with a Pre-Congestion Notification (PCN) based admission control mechanism [15]. In [16], a prototype of an intermediary adaptation node is proposed, where the media gateway estimates the available bandwidth on the client link and extracts the supported SVC-streams. Similar to this, the WiDASH proxy is responsible for in-network video adaptation and is able to perform global optimization over multiple concurrent HAS flows by prioritizing clients which have poor channel quality [17]. *Wirth et al.* discuss the optimization of multi-user resource assignment for DASH video transmission over the LTE downlink [18]. By deploying a cross-layer technique for allocating the resources at the base station and taking into account the specific information of the video sessions, the number of playout starvations can be considerably reduced. In *Parakh et al.*, the authors propose a game theoretic approach towards decentralized bandwidth allocation for video streams in wireless systems, where users are charged for bandwidth resources proportionally to the requested bit-rate [19]. *Situnen et al.* propose dropping video layers based on their priority when network congestion arises for scalable video streaming over wireless networks [20]. Most of the aforementioned research focuses on the dropping of quality layers when congestion arises, meaning the quality is limited in the same way for all users. The proposed approach limits the maximum quality in a per client manner, allowing the service provider to differentiate the delivered video services based on the clients subscription. This allows the service provider to control the QoE on a per subscriber level, and thus offering different subscription types for the VoD HAS services.

*Lee et al.* describe a three-tier streaming service where multiple clients are connected through multiple intermediate proxies to a multimedia server [21]. The authors only consider live streaming, if however VoD streaming would be targeted, the streaming service can no longer be delivered in an efficient way using multicast streaming, since a lot of requests are on unpopular content which is infrequently re-

quested. This causes the content to be delivered using unicast transport from origin to regional servers and thus having the risk of running into bandwidth bottlenecks on these links as well, which is not addressed within the cited paper. Furthermore, videos need to be transcoded in the intermediary proxies, in standard HAS however, the quality levels are discrete and fixed, causing the objective function in the proposed solution to change drastically and leading to the inability to use the max-min composition. Unlike Real Time Streaming Protocol (RTSP) en Real Time Transport Protocol (RTP) based streaming protocols, there is no server-side bitrate adaptation required, the client decides autonomously which quality it will select, based on the current state of the network and from a list of permitted qualities, selected from within the network. This also implies that if a client struggles to achieve its assigned quality level, for example due to poor wireless connection quality or limited CPU resources, it can still decide to switch to a lower quality.

In [22] [23], the authors focus on optimizing the allocation of bits of video sequences among multiple senders to stream to a single client. Peer-to-peer streaming and multi-server distributed streaming are the main use-cases of this approach, there is no simple extension of the work when multiple clients need to share the same server side bottleneck. Furthermore, this requires fine-grained scalable video streaming to support the allocation of non-overlapping bit ranges to multiple servers, while for HAS, fixed bitrate representations are available, encoded using advanced video coding, leading to video segments of which the quality cannot be improved in a straightforward way by downloading additional bit ranges. This work however, could also be extended to support scalable video in a straightforward way. *Akhshabi et al.* propose server-side rate adaptation to cope with unstable streaming players due to ON-OFF patterns when they compete for bandwidth [24]. The systems detect sudden rate fluctuations in the client playout and try to solve them by shaping the sending rate at the server to resemble the bitrate of the stream. These systems are able to restore the streaming session when oscillation or freezing occurs and then remove the shaping when the client has stabilized. The proposed approach is not only able to solve the problems of oscillation or freezes when they occur, but actively tries to prevent them. This is because the proposed approach can use more detailed in-network information. This chapter is an extension to previous work on in-network quality management for HAS [25]. However, the problem formulation is generalized and the approach is significantly extended with a centralized, distributed and relaxed optimization. Furthermore, the previous work only considered simple topologies with a single bottleneck where multiple clients directly connect to the server. Whereas this chapter supports more complex topologies with multiple levels, multiple bottlenecks and intermediary proxies, as well as asymmetric topologies.

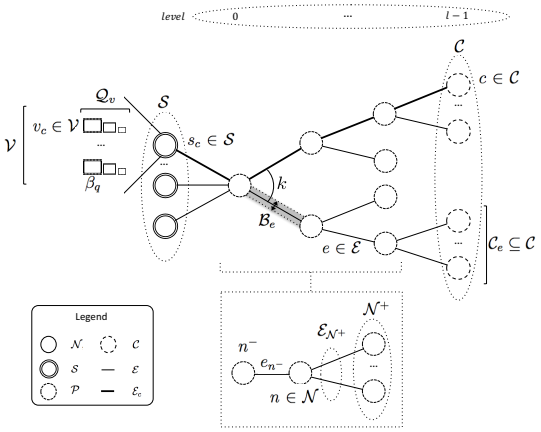



*Figure 3.1: Graphical representation of variables and assumptions.*

## 3.3 Formal Problem Description

Providers are exploring how they can offer VoD HAS services next to traditional TV services over their managed network environment. HAS services offer the same content at multiple qualities, each at their corresponding rate. This allows providers to perform QoE management by adjusting each sessions quality level, based on the current network utilization. At peak times, the consequences of an inadequate amount of resources in the network, can thus be anticipated by reducing the quality of individual streaming sessions, while still allowing admittance of all users.

### 3.3.1 Definition of variables and assumptions

Figure 3.1 gives an overview of the problem variables and assumptions. Let us consider an access network topology modeled as a graph, consisting of a set of nodes $\mathcal{N}$, which encompasses servers $\mathcal{S} \subset \mathcal{N}$, proxies $\mathcal{P} \subset \mathcal{N}$, and clients $\mathcal{C} \subset \mathcal{N}$. A set of edges $\mathcal{E}$ connects the nodes in a logical tree topology which is typically used for video delivery networks[8]. Note that typical access networks are using a logical tree for their delivery, although the underlying physical network is not a tree due to replication concerns. Every node $n \in \mathcal{N}$ has an incoming edge $e_{n^-} \in \mathcal{E}$ connecting it to its predecessor $n^- \in \mathcal{N}$ and a set of outgoing edges $\mathcal{E}_{\mathcal{N}^+} \subseteq \mathcal{E}$ connecting it to its successors $\mathcal{N}^+ \subset \mathcal{N}$. Every edge $e \in \mathcal{E}$ has an associated

[8]An example is the Triple Play Service Delivery Architecture from Alcatel-Lucent (http://goo.gl/4aZVvf), which is used by over 50 operators worldwide (http://goo.gl/kHMY1b)

*Table 3.1: Variables used for the rate decision.*

| | |
|---|---|
| $\alpha_s$ | Weighing factor to model the tradeoff between quality and quality switches |
| $\mathcal{B}_e$ | Bandwidth reserved for HAS traffic for edge $e \in \mathcal{E}$ |
| $\beta_q$ | Bitrate associated with layer $q \in \mathcal{Q}$ |
| $\beta_{max}$ | Highest bitrate in $\mathcal{Q}$ |
| $\mathcal{C} \subset \mathcal{N}$ | The set of HAS VoD clients |
| $\mathcal{C}_e \subseteq \mathcal{C}$ | The set of clients in the service delivery tree for which the VoD traffic traverses edge $e \in \mathcal{E}$ |
| $\mathcal{C}_n \subseteq \mathcal{C}$ | The set of clients in the service delivery tree for which the VoD traffic traverses node $n \in \mathcal{N}$ |
| $\mathcal{E}$ | The set of edges in the service delivery tree |
| $\mathcal{E}_c \subseteq \mathcal{E}$ | The unique delivery path from server $s_c$ to client $c \in \mathcal{C}$ |
| $e_{n^-}$ | The edge connecting node $n$ to its predecessor $n^-$ |
| $\mathcal{E}_{n^+} \subseteq \mathcal{E}$ | The set of edges connecting node $n$ to its successors $\mathcal{N}_{n^+}$ |
| $\mathcal{H}_c$ | The history of previous quality decisions for client $c \in \mathcal{C}$ |
| $h_{c,q,t} \in \mathcal{H}_c$ | Binary variable indicating wether client $c \in \mathcal{C}$ was assigned quality $q \in \mathcal{Q}$ at time $t$ |
| $\mathcal{N}$ | The set of nodes in the service delivery topology |
| $\mathcal{N}_{n^+} \subset \mathcal{N}$ | The set of successors of node $n$ |
| $n^-$ | The predecessor of node $n$ |
| $\mathcal{P} \subset \mathcal{N}$ | The set of proxies in the delivery tree |
| $\mathcal{Q}$ | Available quality rates for video |
| $\mathcal{Q}_v \subseteq \mathcal{Q}$ | Available quality rates for video $v \in \mathcal{V}$ |
| $\mathcal{S} \subset \mathcal{N}$ | The VoD access server |
| $s_c \in \mathcal{S}$ | The VoD access server for client $c$ |
| $\mathcal{V}$ | The set of available videos via VoD server $\mathcal{S}$ |
| $v_c \in \mathcal{V}$ | The video $v \in \mathcal{V}$ for which client $c$ is requesting access |

bandwidth capacity $\mathcal{B}_e$ reserved for HAS traffic.

The servers host a set of videos $\mathcal{V}$. Every video $v \in \mathcal{V}$ has an associated set of quality representations $\mathcal{Q}_v \subseteq \mathcal{Q}$. Moreover, every quality representation $q \in \mathcal{Q}$ has a bit rate $\beta_q$. Every client $c \in \mathcal{C}$ has an associated origin server $s_c \in \mathcal{S}$, a unique delivery path $\mathcal{E}_c \subseteq \mathcal{E}$ from that server, and a video $v_c \in \mathcal{V}$. The set of clients that have an edge $e \in \mathcal{E}$ as part of their delivery path $\mathcal{E}_c$, is represented by $\mathcal{C}_e \subseteq \mathcal{C}$. In summary, Table 3.1 lists the symbols introduced throughout this section.

### 3.3.2 ILP formulation

The problem consists of maximizing the QoE over all clients $c \in \mathcal{C}$, while adhering to the edge bandwidth constraints. The solution is characterised by a boolean

decision matrix $A$. The element $a_{c,q} \in A$ is equal to 1 if quality $q \in \mathcal{Q}_{v_c}$ is selected for client $c \in \mathcal{C}$, and 0 otherwise. The decision variables are subject to the following two constraints:

$$\forall c \in \mathcal{C}, \forall q \in \mathcal{Q}_{v_c} : a_{c,q} \in 0,1 \tag{3.1}$$

$$\forall c \in \mathcal{C} : \sum_{q \in \mathcal{Q}_{v_c}} a_{c,q} = 1 \tag{3.2}$$

The above constraints state that the decision variables are boolean values and that only one quality representation can be selected per client.

According to *Padhye et al.*, the maximum achievable throughput $B$ for a TCP connection subject to a round trip time $RTT$ and maximum window size $W_{max}$, probability of packet loss $p$, delayed ACK number of $b$ and average retransmission timeout $T_0$ can be approximated by the following [26]:

$$B(p) \approx \min \left( \frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)} \right) \tag{3.3}$$

The maximum achievable TCP throughput for client $c$ is thus limited by its window size $W_{max,c}$ and its $RTT_c$. Both parameters can be estimated or measured at the client and forwarded to the in-network control proxy. When $p$ values are low, which is the case in fixed networks, the achievable throughput is primarily limited by the first term. To limit the overhead of acquiring packet loss probabilities for all clients, only the first part of the TCP estimator is considered. Therefore, the following constraint is added, limiting the end-to-end achievable throughput for each client:

$$\forall c \in \mathcal{C} : \sum_{q \in \mathcal{Q}_c} a_{c,q} \times \beta_q \leq \frac{W_{max,c}}{RTT_c} \tag{3.4}$$

When $N$ TCP-connections use the same bottleneck link, *Altman et al.* state that the maximum aggregated achievable throughput that can be obtained is a factor of the link capacity $B_e$ [27]:

$$B_{max} \approx \left(1 - \frac{1}{1+cN}\right) B_e \tag{3.5}$$

Where $c = \frac{1+d}{1-d}$ with $d$ the fraction with which the send rate is decreased when congestion arises. This approximation of the maximum achievable throughput is used to limit the aggregated allocated rate of the different clients:

$$\forall e \in \mathcal{E} : \sum_{c \in \mathcal{C}_e} \sum_{q \in \mathcal{Q}_c} a_{c,q} \times \beta_q \leq \left(1 - \frac{1}{1 + c\,|C|}\right) B_e \tag{3.6}$$

As stated, the objective aims to maximize the global QoE. This is obviously a broad term that can be interpreted in a multitude of ways. As such, a generic objective function is proposed that can be adapted to the service provider's optimization policy, represented by the function $F(\cdot)$:

$$\max \sum_{c \in \mathcal{C}} \sum_{q \in \mathcal{Q}_{v_c}} F(a_{c,q}) \tag{3.7}$$

For example, the provider could aim to maximize the total delivered bit rate, which can be translated into the following objective function:

$$\max \sum_{c \in \mathcal{C}} \sum_{q \in \mathcal{Q}_{v_c}} a_{c,q} \times \beta_q \tag{3.8}$$

The operator could also decide to optimize the fairness among the connected clients. This can be achieved by adopting proportional fairness, as proposed by *Kelly et al.* [28] [29]. A vector of rates $A_c = (\sum_{q \in \mathcal{Q}_{v_c}} a_{c,q} \times \beta_q, c \in \mathcal{C})$ is proportionally fair if it is feasible, according to Equation (3.4) and (3.6) respectively, and if for any other feasible vector $A_c^*$, the aggregate of the proportional changes is zero or negative:

$$\sum_{c \in \mathcal{C}} \frac{A_c^* - A_c}{A_c} \leq 0 \tag{3.9}$$

According to *Wei et al.*, the fair bandwidth allocation can be represented by a local maximum of the logarithmic utility function [30]. Since this function is differentiable and strictly concave, it has only one maximum, which is therefore also the global maximum. However, since it is not linear, the model is not longer an ILP. The objective of a proportionally fair bandwidth allocation can thus be expressed by:

$$\max \sum_{c \in \mathcal{C}} \log \Big( \sum_{q \in \mathcal{Q}_{v_c}} a_{c,q} \times \beta_q \Big) \tag{3.10}$$

Another objective could be to minimize the number of switches since they have a negative impact on overall QoE. This requires maintaining a history $\mathcal{H}_c$ of previous quality decisions for each client $c \in \mathcal{C}$ where $h_{c,q,t} = a_{c,q}$ at time $t$. For quality switches, not only the frequency of switching is important, but also the

distance between quality selections affects the overall quality [5]. Therefore, to assess the impact of distance in quality, the variation in quality over the history $\mathcal{H}_c$ is taken into account. The following weighted sum is used to model the impact on switching behavior, where $\mu$ represents the average quality, $\sigma$ introduces a penalty for quality switching and $\alpha_s$ represents a weighing factor used to emphasize either the impact of quality or the switching behavior:

$$\max \alpha_s \times \mu - (1 - \alpha_s) \times \sigma \tag{3.11}$$

Since the decision variables $a_{c,q}$ are binary variables, the calculation of the objective function can be simplified by calculating $\mu_{c,q}$ and $\sigma_{c,q}$ for each client $c$ and it's associated quality range $\mathcal{Q}_c$. The quality rates are normalized with respect to the highest quality rate $\beta_{max}$.

$$\forall c \in \mathcal{C}, \forall q \in \mathcal{Q}_v : \mu_{c,q} = \frac{1}{|\mathcal{H}_c| + 1} \left( \frac{\beta_q}{\beta_{max}} + \sum_{h_{c,t} \in \mathcal{H}_c} \sum_{q \in \mathcal{Q}_c} \frac{h_{c,q,t} \times \beta_q}{\beta_{max}} \right) \tag{3.12}$$

In this way, the use of quadratic terms in the objective function is avoided. The penalty $\sigma$ for switching between qualities can be calculated as follows:

$$\forall c \in \mathcal{C}, \forall q \in \mathcal{Q}_v : \sigma_{c,q} = \sqrt{\frac{1}{|\mathcal{H}_c| + 1} \left( \left( \frac{q \times \beta_q}{\beta_{max}} - \mu_{c,q} \right)^2 + \sum_{h_{c,q,t} \in \mathcal{H}_c} \sum_{q \in \mathcal{Q}_c} \left( \frac{h_{c,q,t} \times \beta_q}{\beta_{max}} - \mu_{c,q} \right)^2 \right)} \tag{3.13}$$

The total objective can then be expressed as:

$$\max \sum_{c \in \mathcal{C}} \sum_{q \in \mathcal{Q}_v} a_{c,q} \times (\alpha_s \times \mu_{c,q} + (1 - \alpha_s) \times \sigma_{c,q}) \tag{3.14}$$

## 3.4 Algorithms

### 3.4.1 Centralized ILP formulation

The ILP formulation described in Section 3.3.2 can be used to optimize the quality assignments using a centralized controller. It requires as input the knowledge of the delivery network topology $(\mathcal{N}, \mathcal{E})$, link constraints $\mathcal{B}_e$, the set of clients $\mathcal{C}_e$

for which the VoD traffic traverses an edge $e \in \mathcal{E}$ and the characteristics of these clients ($W_{max,c}, RTT_c$). Solving said ILP formulation will yield a set of optimal quality assignments $a_{c,q}$ for each client $c$ and quality level $q$. These assignments are optimal in the sense that they maximize the objective $\sum_{c \in \mathcal{C}} \sum_{q \in \mathcal{Q}} F(a_{c,q})$ subject to the constraints described in Equations (3.1), (3.2), (3.4) and (3.6). As it is assumed that there is a constant bitrate reserved for HAS traffic on each edge, the *Centralized* optimization is executed each time a newly joined client requests a manifest file or if a client becomes inactive by leaving the delivery network.

### 3.4.2 Distributed ILP formulation

The number of constraints for the centralized ILP grows with an increasing depth of the service delivery topology tree. Consider a topology tree with $k$ child nodes per node and $l$ levels (thus $l = log_k|C| + 1$), the total number of edge constraints is then equal to $\sum_{i=0}^{l-1} k^i$ which can be written as $\frac{1-k^l}{1-k}$. This leads to an exponentially increasing model size with the number of levels in the delivery tree, affecting the calculation time. Since the proposed approach would be deployed in an operational setting, the decision process should be able to determine quality allocations in real-time. Therefore, this chapter also proposes a distributed approach, where each proxy locally determines the optimal allocation constrained by the local edge capacities and where the global solution is an aggregation of these local solutions. The advantages of this approach are threefold. First, each node only needs to have local information on the properties of the upstream edge $e_{n^-}$ and the video flows for clients $\mathcal{C}_n$ traversing this node. Second, since each proxy locally solves the optimization problem, the number of constraints does not increase with the tree size, leading to small local ILP models. Third, proxies residing at the same level in the topology tree can optimize their local ILP models in parallel, leading to faster global optimization.

The distributed ILP algorithm uses a bottom-up approach for the distributed solution where each node $n$ locally optimizes the allocation problem and forwards the solution to its predecessor $n^-$ in the delivery tree. The solution at node $n$ is constrained by the limitations of it's successors $\mathcal{N}_{n^+}$. These limitations are determined by the combination of the optimal solutions $a^{n^+}_{c,q}$ of each node $n^+ \in \mathcal{N}_{n^+}$. For each client $c \in \mathcal{C}_n$, $sq_{n,c}$ determines the maximum quality a client is able to receive according to the successors of $n$. The local ILP formulation is then constrained by (3.1) and (3.2) as before but only for $c \in \mathcal{C}_n$, while the following determines that the selected quality for a client $c$ is not allowed to violate the limitations determined by the successors in $sq_{n,c}$:

$$\forall c \in \mathcal{C}_n : \sum_{q \in \mathcal{Q}_{v_c} : q > sq_{n,c}} a_{c,q} = 0 \tag{3.15}$$

Furthermore, the total consumed bandwidth for the allocation is not allowed to exceed the HAS capacity $\mathcal{B}_{e_{n^-}}$ for the edge $e_{n^-}$, connecting $n$ to its predecessor $n^-$.

$$\sum_{c \in \mathcal{C}_n} \sum_{q \in \mathcal{Q}_{v_c}} a_{c,q} \times \beta_q \leq \mathcal{B}_{e_{n^-}} \tag{3.16}$$

The calculation time can be limited by taking advantage of some specific properties of the problem. First, since the distributed optimization is only performed when a client joins or leaves the service delivery network, only the proxies $p \in \mathcal{P}_c$ on the delivery path for client $c$ are required to perform local optimization. For other proxies, the previous optimal solutions remain valid since there are no changes in (3.15) and (3.16). This limits the number of local optimizations during the execution of the distributed algorithm to the number of levels $l = \log_k |C| + 1$. Second, the solutions determined by the predecessors of $n$ are optimal and since these solutions are independent, their combination is optimal. This means that if the combination of the solutions $a^{n^+}_{c,q}$ is feasible, there is no need to perform local optimization. In the best case, where there is only one bottleneck in the network, the number of local optimization steps is thus reduced to 1. For the worst case scenario, where the upstream bottleneck becomes tighter at every node, the maximum number of local optimizations is limited by $l = \log_k |C| + 1$ as before.

There is a communication overhead when using the distributed optimization, which requires sending a list of clients for which the video traffic traverses the node and the decision on the quality level per client. Since this list (or any combination made of it by any node) contains at most $n$ entries, with $n$ the number of connected clients, the number of required bits per client entry is $\lceil \log_2 n \rceil$ to uniquely identify the clients and $\lceil \log_2 q \rceil$ to identify their selected quality with $q$ the highest number of available qualities. For each level in the topology, the information exchanged will thus be equal to $n\,(\lceil \log_2 n \rceil + \lceil \log_2 q \rceil)$ bits and since $log_k n + 1$ is the number of levels, the total information exchange is $n(\lceil \log_2 n \rceil + \lceil \log_2 q \rceil) \log_k n$. If there are 1000000 clients in the network and k is equal to 10, then the total communication overhead is 16.5 MB per optimization. If the lowest quality representation is 1Mbps, the total amount of traffic flowing through the network per second is in the order of Tbps, leading to a negligible communication overhead for the optimization.

### 3.4.3 Relaxed distributed Linear Programming (LP) formulation

Solving the distributed ILP optimally in a single node can however lead to execution times in the order of seconds when the number of VoD flows crossing that node becomes large. The execution speed can be increased at the expense of a

sub-optimal solution by moving from an Integer LP formulation to a Relaxed LP formulation by relaxing the boolean constraints on the variables $a_{c,q}$ in (3.1) by only requiring $a_{c,q}$ to belong to the interval $[0, 1]$:

$$\forall c \in \mathcal{C}, \forall q \in \mathcal{Q}_{v_c} : 0 \leq a_{c,q} \leq 1 \tag{3.17}$$

1: **for all** $c \in \mathcal{C}$ **do**
2: $Proximity_c \leftarrow \min_{q \in \mathcal{Q}_{v_c}}(1 - a_{c,q})$
3: $Contribution_c \leftarrow \sum_{q \in \mathcal{Q}_{v_c}} F(a_{c,q})$
4: $mq_c \leftarrow \max_{q \in \mathcal{Q}_{v_c}}(a_{c,q} > 0)$
5: **end for**
6: $Sort(a_{c,q}, Proximity_c, Contribution_c)$
7: $sol \leftarrow 0$
8: **for all** $c \in \mathcal{C}$ **do**
9: $B \leftarrow B + \sum_{q \in \mathcal{Q}_{v_c}} a_{c,q} \times \beta_q$
10: $MaxObj \leftarrow 0$
11: $setq \leftarrow 0$
12: **for all** $q \in \{0, mq_c\}$ **do**
13: **if** $B \geq \sum_{q \in \mathcal{Q}_{v_c}} a_{c,q} \times \beta_q$ **and** $\sum_{q \in \mathcal{Q}_{v_c}} F(a_{c,q}) > MaxObj$ **then**
14: $MaxObj \leftarrow \sum_{q \in \mathcal{Q}_{v_c}} F(a_{c,q})$
15: $setq \leftarrow q$
16: **end if**
17: **end for**
18: $sol_{c,setq} \leftarrow 1$
19: $B \leftarrow B - \beta_{setq}$
20: **end for**

*Algorithm 3.1: Overview of the heuristic to transform a floating-point solution to an integer solution.*

This relaxation can be solved in polynomial time but at the cost of optimality. The variables $a_{c,q}$ do not longer unambiguously define which quality each client is allowed to download, therefore a heuristic is required to transform the optimal floating point solution into an integer solution. Algorithm 3.1 shows an overview of the heuristic procedure. First, the clients of the solution matrix $A$ are sorted according to two criteria: first on the proximity of the floating point solution to the integer solution and subsequently on the contribution to the objective (line6). The proximity of a client solution is defined as $P_c = \min_{q \in \mathcal{Q}_{v_c}}(1 - a_{c,q})$, where $P_c = 0$ indicates that the floating point client solution can be immediately transformed into an integer client solution (line 2). The contribution to the objective is calculated as $\sum_{q \in \mathcal{Q}_{v_c}} F(a_{c,q})$ and gives an indication of the weight of a single client in the global objective (line 3). Second, the maximum quality each client is allowed to download determined by the solution $a_{c,q}$, is set to $mq_c = \max_{q \in \mathcal{Q}_{v_c}}(a_{c,q} > 0)$ (line 4). This assures that the limitations of the successors $\mathcal{N}_{n^+}$ are not violated which could lead to an infeasible solution further down the delivery tree. Third, for each client $c \in \mathcal{C}_n$ their contribution to the

constraint is calculated as $\sum_{q \in \mathcal{Q}_{v_c}} a_{c,q} \times \beta_q$ and added to the budget $B$ (line 9). Then, the following problem is optimized for client $c$ (line 12 to 19):

$$\max \quad \sum_{q \in \mathcal{Q}_{v_c}} F(a_{c,q}) \tag{3.18}$$

$$\text{subject to} \quad \sum_{q \in \mathcal{Q}_{v_c}} a_{c,q} \times \beta_q < B \tag{3.19}$$

$$a_{c,q} \in \{0,1\} \forall q \in \mathcal{Q}_{v_c}. \tag{3.20}$$

## 3.5 Performance Evaluation

### 3.5.1 Experiment setup

A VoD HAS scenario was implemented by using an NS3 based simulation framework, capable of the transmission of HAS video [25]. This framework has been extended with support for QoE management at both the servers and the proxies. For the HAS Clients, the *AVC MSS* algorithm is used, which is based on the implementation of an open source version of the algorithm of the Microsoft Smooth Streaming (MSS) video player[9] and is extensively described by *Famaey et al.* [11]. The server-based traffic shaping method proposed by *Akhshabi et al.* was also implemented in combination with *AVC MSS* and is referred to as *AVC MSS Rate Controlled* [24]. This approach tries to reduce quality oscillations when multiple clients compete for bandwidth, by dynamically adjusting the shaping rate when oscillations are detected. For further implementation details, the reader is referred to the paper by *Akhshabi et al.* [24]. Furthermore, an additional client heuristic was implemented, which downloads each segment using the QoE management quality decision and checks if these decisions are feasible, given the measured throughput at the client. If the measurements indicate that the proposed quality is not achievable, the proposed client heuristic will select the highest sustainable quality based on the local throughput measurements. The aforementioned heuristic is referred to as *AVC Steered*. Both client heuristics dispose of a buffer of 10 second, thus accommodating space for 5 segments. Both client implementations use persistent connections to avoid the impact incurred by the setup and teardown of TCP-connections. The congestion window at the server was limited to avoid unfair sharing of the bottleneck bandwidth [31]. The configured congestion window allows transmitting segments at a rate that is two times bigger than the maximum bitrate of the stream.

As discussed before, the delivery network is modeled as a tree-based topology, where a video server $\mathcal{S}$ streams videos to a set of clients $\mathcal{C}$. The number of

[9] Source available from https://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming

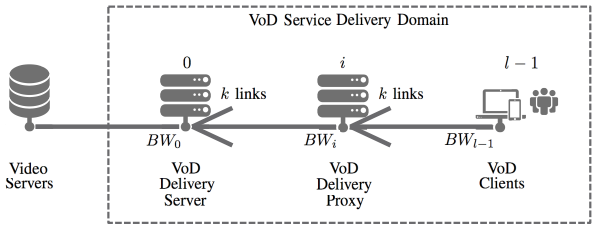



*Figure 3.2: Network topology, modeling a typical video service delivery network.*

*Table 3.2: Overview of the quality layers for the Big Buck Bunny video.*

| Quality Layer Index | Average Bitrate (kbps) | Average PSNR (dB) | Average SSIM |
|---|---|---|---|
| 0 | 300 | 32.04 | 0.8746 |
| 1 | 427 | 32.72 | 0.8861 |
| 2 | 608 | 34.41 | 0.9108 |
| 3 | 866 | 35.71 | 0.9249 |
| 4 | 1233 | 36.88 | 0.9387 |
| 5 | 1636 | 37.64 | 0.9469 |
| 6 | 2436 | 40.07 | 0.9608 |

branches on each level was set to $k$, leading to $l = log_k|\mathcal{C}| + 1$ levels in the tree. For each level the available HAS bandwidth was varied depending on the bottleneck factor $BF$. For each level the bandwidth $BW_i = (k * BF)^{l-i-1} * BW_{l-1}$, meaning that if $BF = 0.8, l = 4, k = 5, BW_{l-1} = 4Mbps$, the bandwidth for a edge at level 1 is $BW_1 = 64Mbps$. Figure 3.2 gives a graphical overview of the service delivery network topology. Clients are started using a Weibull startup process with shape 2.5 and mean of $300s$. The average RTT for each client $c$ is set to $RTT_c = 40ms$ unless stated otherwise [32]. The Big Buck Bunny video[10] was encoded at 7 different quality rates and divided into 200 segments with an average duration of 2 seconds. Table 3.2 gives an overview of the different quality layers, their associated bitrates, average Peak Signal-to-noise Ratio (PSNR) and Structural Similarity (SSIM) values. During the evaluations the SSIM metric introduced by *Wang et al.* is used, which is motivated by the assumption that human visual perception is highly adapted for extracting structural information. It has been shown to have a high correlation with image [33] and video quality [34].

[10]Big Buck Bunny available from http://www.bigbuckbunny.org/

### 3.5.2 Implementation details

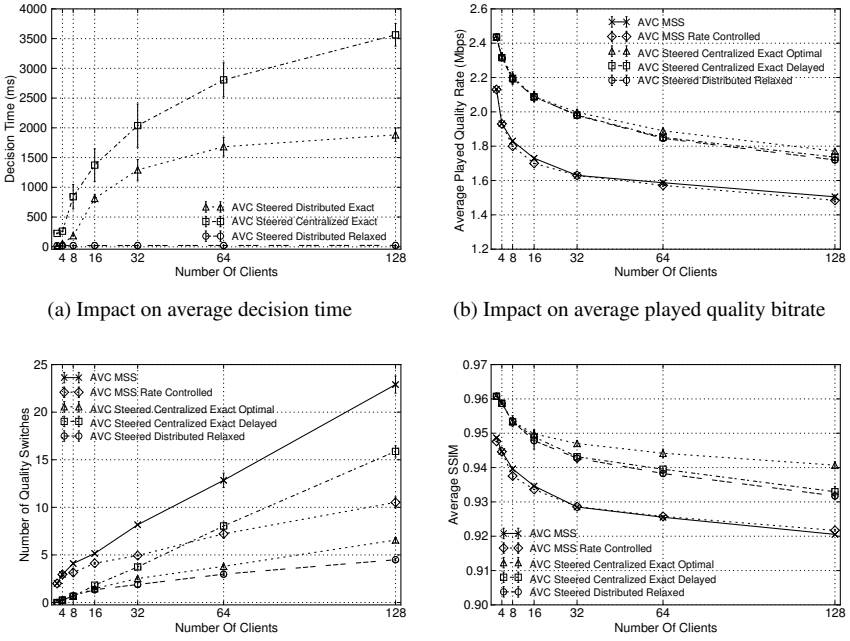
The IBM CPLEX[11] solver was used to implement and solve the proposed binary ILP-problems for both the centralized and distributed algorithm, as well as the relaxed distributed LP-problem. Since NS3 is an event-based simulation framework, the execution times of the optimizations were measured as $t_{exec}$ and used to schedule the release of the calculated configuration at $t_{current} + t_{exec}$. The different experiments are executed using two modes: *Delayed* ensuring that the configurations only become available when optimization is finished and *Optimal* which is agnostic to execution times and installs the configuration immediately.

However, since executions can take several seconds, clients can join during this period, leading to two implications for the system: first, since there is no immediate quality configuration, the *Steered Client* heuristic will not be able to select the suited quality and second, new optimizations, which take into account more recent network configurations are delayed by the execution of the previous one. Both problems can be solved by performing a heuristic optimization first, allowing to quickly install a suboptimal quality configuration and replace it with the optimal configuration when it becomes available. This also allows us to preempt a QoE optimization when additional requests lead to a changed environment and the optimal solution would be outdated. The heuristic optimization checks if the previous limitations in combination with the additional client are feasible for each edge $e$, if not the client qualities for $c \in \mathcal{C}_e$ are lowered by one level until the solution is feasible again.

### 3.5.3 Evaluation details

The performance of the centralized ILP, distributed ILP and relaxed distributed LP was evaluated in terms of service assurance, quality delivery and oscillations. Also the impact of the different approaches on the decision time was quantified. The network size, the number of bottlenecks, the optimization objective, Round Trip Time (RTT) and number of servers were varied. During the evaluations, the maximization of the video bitrates, defined by (3.8), was used as a network provider's objective. The centralized and distributed ILP optimization are referred to as *Centralized Exact* and *Distributed Exact* respectively, while the relaxed optimization is indicated as *Distributed Relaxed*. All experiments use the quick optimization heuristic, providing a preliminary decision on the selected qualities, which is updated when the optimization calculation is finished. Therefore these results are installed with a delay and are referred to as *Delayed* decisions. When it is explicitly stated that the configurations are installed based on the *Optimal* selection, the graphs show the results that could be achieved with the optimal configurations

[11] IBM ILOG CPLEX Optimizer: http://www.ibm.com/software/integration/optimization/cplex-optimizer/




(a) Impact on average decision time

(b) Impact on average played quality bitrate

(c) Impact on average number of quality switches

(d) Impact on average SSIM

*Figure 3.3: Impact of number of clients, using a topology with $k = 2$, $BW_{l-1} = 3Mbps$, $BF = 0.9$.*

in the presence of network variations and buffering. All of the following results are averaged over $n = 10$ iterations, with the graphs showing the $95\%$ confidence intervals $[\bar{X} - 1.96\frac{\sigma}{\sqrt{n}}, \bar{X} + 1.96\frac{\sigma}{\sqrt{n}}]$ [35].

### 3.5.4 Impact of number of clients

In this section, the deployment of in-network quality adaptation algorithms is motivated for HAS delivery networks and the impact of the delivery tree size on the in-network adaptation performance is quantified. For the following experiments the number of child nodes is set to $k$ for each node to 2 and the number of clients $|\mathcal{C}|$ is varied in the interval $[2, 128]$, the bandwidth on the first link $BW_{l-1}$ is set to $3Mbps$, while the bottleneck factor is set to $BF = 0.9$, leading to a bottleneck at the server $\mathcal{S}$ of approximately $184Mbps$.

Figure 3.3(a) illustrates the impact of the number of clients on the QoE optimization execution times. For both the *Centralized Exact* and *Distributed Exact* optimization, the execution times show a logarithmic increase with the number of clients and thus a linear increase with respect to the number of levels in the

topology tree. For 7 levels, the average optimization time for the *Centralized Exact* algorithm is $3563.34ms$, for the *Distributed Exact* algorithm it is $1881.75ms$, while the *Distributed Relaxed* algorithm only requires $23.52ms$ (including an average one-way delay of $20ms$ for forwarding local solutions). These results indicate that only the distributed heuristic approach is feasible for medium to large size problems.

The impact of the in-network quality management on the average bitrate, the number of switches and average quality in terms of SSIM is shown in Figure 3.3(b), 3.3(c) and 3.3(d) respectively. The results show a significant improvement on the average played bitrate over traditional client-based heuristics ranging from 14% to 23%, while the number of switches can be reduced with a factor of $1.5$ to $5$. Since client-based heuristics have only a local view on the network situation, they require safety measures to avoid buffer starvations and quality oscillations, resulting in underestimations of the available throughput and thus underutilization of the available bandwidth. Combining a client-driven approach with server-based traffic shaping allows maintaining the same quality as with a client-driven approach, while reducing the number of switches up to a factor $2.5$. In-network quality adaptation is able to react more quickly to changing network environments and allows to fully utilize the available bandwidth. The *Distributed Relaxed* optimization is able to reduce the number of switches with a factor $5$ compared to the traditional client-based heuristic and with a factor $2.5$ when traditional client-based approaches are combined with server-based rate shaping. The *Centralized Exact Delayed* optimization is able to achieve a slightly higher quality than the *Distributed Relaxed* optimization, but shows an increased number of switches which can be accounted to the longer execution times causing the clients to choose suboptimal quality configurations while waiting for the optimal configuration. The *Centralized Exact Optimal* optimization shows the theoretical configuration that could be achieved in absence of the long execution times. These results show a penalty of about $3\%$ in terms of average quality rate when using the *Distributed Relaxed* optimization due to suboptimal solutions attained by the rounding heuristic. As shown in Figure 3.3(d), also the achieved average SSIM, is slightly lower for the *Distributed Relaxed* optimization. The average number of switches is slightly higher for the *Centralized Exact Optimal* optimization when compared to the *Distributed Relaxed* heuristic. The *Bandwidth* optimization objective does not take into account the switching penalty explicitly, while the *Distributed Relaxed* optimization reuses local solutions which are still feasible to calculate the next optimal solution, inherently minimizing the difference between two subsequent solutions. For further discussion on how this switching penalty could be included during the optimization, the reader is referred to Section 3.5.6.

The aforementioned results confirm the advantages of adopting in-network quality adaptation: First, the average played quality can be improved compared

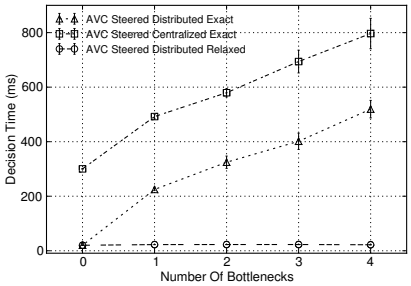



*Figure 3.4: Impact of number of bottlenecks in the topology on the average decision time, using a topology with $k = 5$, $BW_{l-1} = 2Mbps$, $|\mathcal{C}| = 125$ and $l = 4$.*

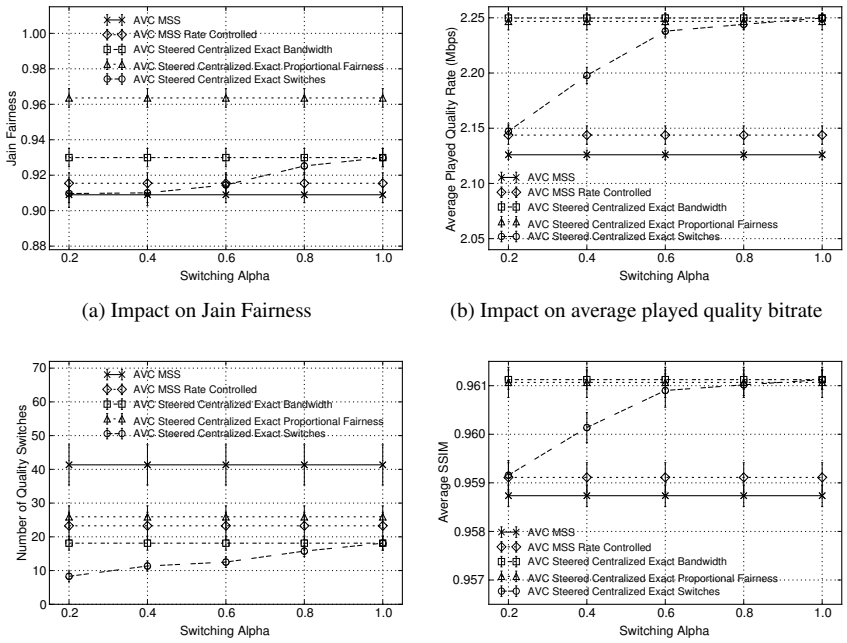
to the quality delivered by traditional client heuristics. Second, also the number of quality switches can be significantly reduced. Furthermore, the *Distributed Relaxed* heuristic is able to calculate a suboptimal configuration at low execution cost, making the approach viable for real-time delivery systems. The fast calculation of the optimal configuration also leads to fewer quality switches, since the configurations can be installed quickly and there is no need to install suboptimal temporary configurations as is the case with both *Centralized Exact Delayed* optimization.

### 3.5.5 Impact of number of bottlenecks

As indicated in Section 3.4.2, the number of bottlenecks has an impact on the behavior of the *Distributed Exact* optimization. The topologies for these experiments were created by introducing bottlenecks in a top-down manner and setting the bottleneck factor $BF_l$ to 1 if there is no bottleneck at level $l$ and $BF_l = 0.8$ otherwise. Figure 3.4 confirms a linear increase in execution time for both the *Exact* and *Relaxed* optimization with an increasing number of bottlenecks. The *Centralized Exact* optimization however, takes $300ms$ to execute, even in the absence of a bottleneck, while the *Distributed* optimization is only performed when the configuration assigning maximum quality to each client becomes infeasible, leading to an execution time of on average $20ms$, consisting solely out of the delay introduced by forwarding the local solutions.

### 3.5.6 Impact of optimization objective

An operator can optimize different policies when offering a HAS streaming service such as maximizing the total bitrate over all streams (Equation (3.8)), maximizing the proportional fairness across the streaming sessions (Equation (3.10)) or maximizing the QoE as a weighted sum of the total bitrate and bitrate variations (Equation (3.14)). Decreasing the factor $\alpha_s$ puts the emphasis on decreasing the impact




(a) Impact on Jain Fairness

(b) Impact on average played quality bitrate

(c) Impact on average number of quality switches

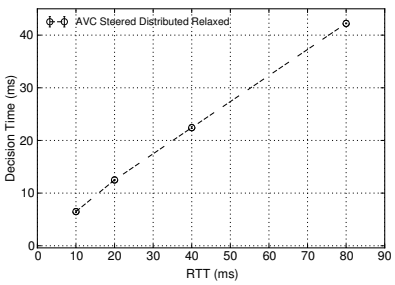
(d) Impact on average SSIM

*Figure 3.5: Impact of the Switching Alpha $\alpha_s$, using a topology with $k = 5$, $BW_{l-1} = 4Mbps$, $BF = 0.8$, $|\mathcal{C}| = 125$ and $l = log_k|\mathcal{C}| + 1 = 4$.*

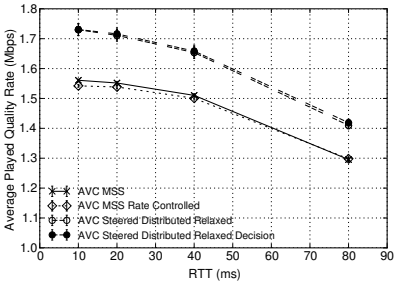
of switches, while increasing $\alpha_s$ increases the impact of bitrate optimization. To quantitatively evaluate the fairness degree of the different optimization schemes, the Jain's Fairness Index is used [36]. This index states that if a system allocates resources to $|\mathcal{C}|$ users, where user $c$ receives a rate allocation $b_c$, the fairness index of the system is defined to be:

$$J = \frac{(\sum_{c \in \mathcal{C}} b_c)^2}{|\mathcal{C}| \sum_{c \in \mathcal{C}} {b_c}^2} \tag{3.21}$$

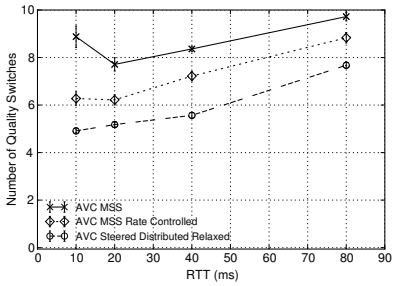
Figure 3.5(a) shows the impact of these different optimization goals on the average Jain Fairness in function of the switching factor $\alpha_s$. Optimizing the total bitrate allocation is able to achieve a fairness index closer to 1 then *AVC MSS*, indicating a fairer distribution of the available throughput among the clients. Adding rate shaping at the server allow increasing the fairness for *AVC MSS* at a slightly increased quality and a reduction with a factor 1.7 in terms of number of switches. When optimizing for proportional fairness, the in-network optimization is able




(a) Impact on average buffer starvations




(b) Impact on average played quality bitrate




(c) Impact on average number of quality switches

*Figure 3.6: Impact of the RTT (ms), using a topology with* $k = 5$*,* $BW_{l-1} = 3Mbps$*,* $BF = 0.8$*,* $|\mathcal{C}| = 125$ *and* $l = log_k|\mathcal{C}| + 1 = 4$*.*

to increase the fairness index at the cost of slightly decreasing the average quality as indicated in Figure 3.5(b) and 3.5(d) and increasing the average number of switches from 18 to 26. This indicates the trade-off between maximizing fairness and total bitrate allocation.

When the impact of the number of switches increases (by decreasing $\alpha_s$), the fairness index drops. This can be attributed to the fact that the in-network optimization will prefer to retain the same quality level for each client at any moment in time, leading to a number of clients downloading the highest quality, while newly arrived streaming sessions are assigned a lower quality, decreasing the total fairness. As Figure 3.5(c) shows, the in-network adaptation is able to reduce the number of quality switches compared to *AVC MSS* for all optimization goals. Explicitly minimizing the number of switches allows further reduction of the number of switches from $43\%$ to $30\%$ at the cost of decreasing quality.

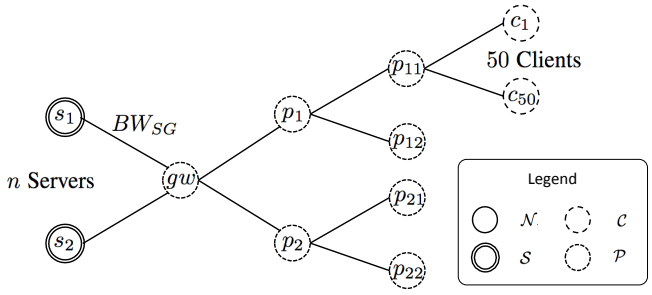


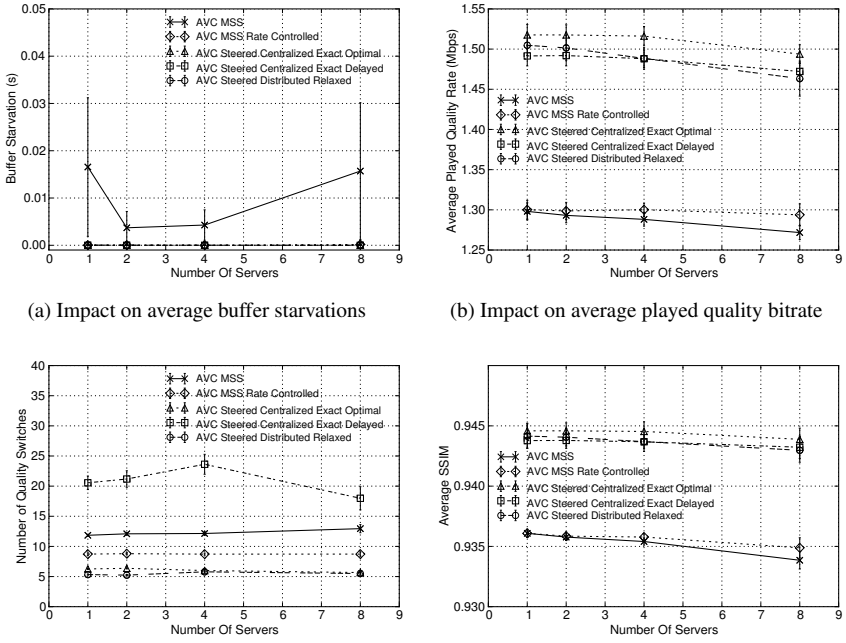

*Figure 3.7: Network topology, modeling a typical video service delivery network with multiple servers.*

### 3.5.7 Impact of delay

The in-network optimization uses an approximation of the achievable throughput as an upper bound for each link. Since RTT has the biggest impact on the available throughput, the RTT was varied and the impact on streaming performance was evaluated. Figure 3.6(b) shows how both the client-side and in-network assisted quality decision based clients suffer from a quality degradation when delay increases. For low RTT (10 ms) the in-network approach is able to increase the quality by $10.8\%$ by deploying in-network quality decisions, while for higher RTTs (80ms), the gain decreases to $8.9\%$. This slight performance decrease can be attributed to the fact that an approximation of the achievable throughput is used. As discussed in previous work [12], HAS quality decreases quickly when RTTs increase due to the subsequent download-request cycles. Using HTTP pipelining can reduce the negative impact of the RTT on quality, by eliminating the idle time between two successive downloads. Figure 3.6(a) illustrates the impact of RTT on the decision times. Since the *Distributed* optimization requires a bottom up propagation of intermediary solutions, the decision times are also impacted by increasing delays. Figure 3.6(b) shows that for increasing delay, the performance of the in-network decisions slightly decreases when compared to the optimal decision, due to the network delay increasing the decision time.

### 3.5.8 Impact of multiple servers

To analyze the impact of multiple servers on the in-network optimization, a typical video service delivery network is modeled with multiple servers as illustrated in Figure 3.7. The *Distributed optimization* was modified to also support this type of topologies by first performing two types of bottom-up optimizations, one taking no limitations as input and a second optimization taking into account the limitations of performing a local optimization between *Gateway* $gw$ and servers first. Both




(a) Impact on average buffer starvations

(b) Impact on average played quality bitrate

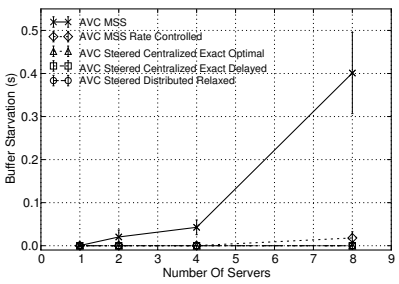
(c) Impact on average number of quality switches

(d) Impact on average SSIM

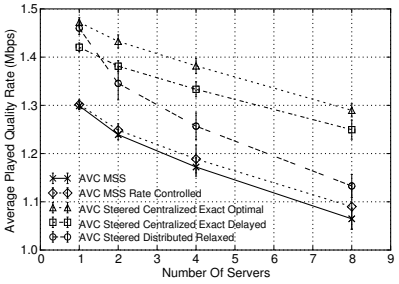
*Figure 3.8: Impact of multiple servers with balanced load and the Number of clients, using a topology with $BW_{l-1} = 2Mbps$, $BF = 0.8$ and $|\mathcal{C}| = 200$.*

approaches yield feasible solutions, out of which the optimal one is selected. The number of servers $n$ was varied and the link bandwidths ($BW_{SG}$) were scaled to reflect this situation. 20 content items were created and the clients were distributed over these content items using a Zipf distribution with $\alpha = 0.81$ to mimic the Internet popularity [37]. The content items were then assigned to the different server instances to evenly distribute the load among them. Figure 3.8(a) illustrates the average buffer starvation in seconds, showing how the in-network optimization is able to deliver the video stream without buffer starvations, whereas *AVC MSS* suffers some minor frame freezes due to competing behavior.

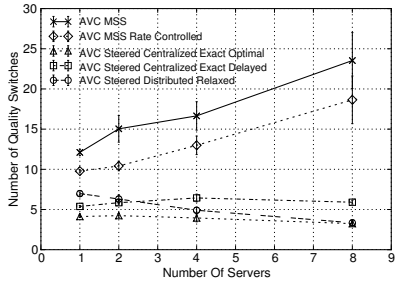
The average quality is shown in Figure 3.8(b) and 3.8(d) which indicates that the in-network optimization is able to yield a higher average quality than *AVC MSS*. Adding rate shaping at the server, allows increasing the quality for *AVC MSS* when the number of servers increases. Up to 4 servers, the *Distributed Relaxed* optimization is able to outperform the *Centralized Exact Delayed* optimization due to the installation delay of the former approach, which was discussed earlier. The *Distributed Relaxed* decision however is suboptimal compared to the *Centralized Exact Optimal* decision. This is caused by the rounding heuristic and the distributed



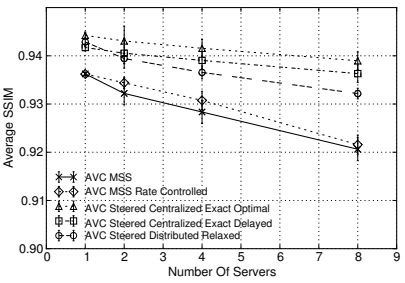

(a) Impact on average buffer starvations




(b) Impact on average played quality bitrate




(c) Impact on average number of quality switches




(d) Impact on average SSIM

*Figure 3.9: Impact of multiple servers with unbalanced load and the Number of clients, using a topology with $BW_{l-1} = 2Mbps$, $BF = 0.8$ and $|\mathcal{C}| = 200$.*

approach, which does not propagate multiple identical solutions which could yield more optimal solutions upwards the stream. Propagating multiple similar solutions, could benefit the distributed approach, at the cost of increased complexity and network communication. With respect to the number of switches, the *Distributed Relaxed* approach outperforms the *AVC MSS* algorithm, *AVC MSS* with rate shaping and *Centralized Delayed* optimization with approximately a factor between $[2.1, 2.3]$, $[1.5, 1.7]$ and $[3.3, 4.1]$ respectively.

During the previous experiments the load was evenly distributed across the different servers. During the following experiments, the content items were assigned to the different server instances following the pareto principle: $80\%$ of the content items are stored on $20\%$ of the servers. This results in an unevenly distributed request pattern across the servers. The *Distributed approach* is not able to achieve the same performance as the *Centralized optimization*. This can be attributed to the fact that an optimal resource allocation upstream of the gateway does not necessarily translate into a feasible and optimal solution downstream of the gateway and vice versa. Figure 3.9(b) shows the impact of adding multiple servers with uneven load distribution on the average achieved quality rate. This shows how the

performance of the *Distributed optimization* degrades when the number of servers increases due to the suboptimal quality decisions. For $8$ servers, the performance of the *Distributed optimization* drops to about $88\%$ of the optimal solution. Figure 3.9(a) and (c) show the impact of increasing the number of servers on the buffer starvations and quality switches respectively. Both in-network approaches are able to maintain a fluent playout, while for *AVC MSS*, the average freeze time is about $0.4s$. For $8$ servers, the average number of switches for *AVC MSS* amounts to $23$, which can be reduced to $19$ when applying server-based rate shaping. Enabling the *Centralized* in-network approach allows reducing the number of switches to $6$. The *Distributed* approach even further decreases the average number of switches to $3.3$, but at the cost of reduced quality compared to the *Centralized* optimization, as was mentioned before. This shows that an unbalanced spread across the content servers impacts the performance of the *Distributed* approach compared to the *Centralized optimization*.

## 3.6 Conclusion

In this chapter, an in-network Quality of Experience (QoE) management framework is proposed for Video on Demand (VoD) HTTP Adaptive Streaming (HAS) in a managed network environment. The adaptation algorithms enable the network providers to control the quality selection at the client. This allows them to increase the average played quality with at least $14\%$ compared to traditional client-based heuristics. Moreover, due to the in-network management, the number of quality oscillations can be reduced with a factor $5$ and with a factor $2.5$ when traditional client-based approaches are combined with server-based rate shaping. This chapter also discussed different variants of the in-network QoE management: an optimal Centralized ILP, a Distributed ILP and a relaxation of the Distributed algorithm. The impact of the number of clients, the Absolute Gap for the integer optimization and the number of bottlenecks were quantified. The impact of the delayed installation of the configurations showed that, even though the *Distributed Relaxed* optimization yields suboptimal configurations, the immediate installation of these configurations allows them to yield higher average quality at a significantly lower number of switches compared to the *Exact* optimization algorithms.

# References

[1] X. Wang. *Network-Assistance and Server Management in Adaptive Streaming on the Internet*. In Proceedings of the Fourth W3C Web and TV Workshop, 2014.

[2] N. Bouten, S. Latré, W. Van de Meerssche, B. De Vleeschauwer, K. De Schepper, W. Van Leekwijck, and F. De Turck. *A Multicast-Enabled Delivery Framework for QoE Assurance of Over-The-Top Services in Multimedia Access Networks*. Journal of Network and Systems Management, 21(4):677–706, 2013.

[3] S. Akhshabi, A. C. Begen, and C. Dovrolis. *An Experimental Evaluation of Rate-adaptation Algorithms in Adaptive Streaming over HTTP*. In Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSys), pages 157–168, 2011.

[4] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. *What Happens when HTTP Adaptive Streaming Players Compete for Bandwidth?* In Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), pages 9–14, 2012.

[5] D. C. Robinson, Y. Jutras, and V. Craciun. *Subjective Video Quality Assessment of HTTP Adaptive Streaming Technologies*. Bell Labs Technical Journal, 16(4):5–23, 2012.

[6] S. Benno, J. O. Esteban, and I. Rimac. *Adaptive streaming: The network HAS to help*. Bell Labs Technical Journal, 16(2):101–114, 2011.

[7] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. *Helping Hand or Hidden Hurdle: Proxy-Assisted HTTP-Based Adaptive Streaming Performance*. In Proceedings of the IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), pages 182–191, 2013.

[8] T. Stockhammer. *Dynamic Adaptive Streaming over HTTP – Standards and Design Principles*. In Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSys), pages 133–144, 2011.

[9] C. Liu, I. Bouazizi, M. M. Hannuksela, and M. Gabbouj. *Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network*. Signal Processing: Image Communication, 27(4):288 – 311, 2012.

[10] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala. *Quality Selection for Dynamic Adaptive Streaming over HTTP with Scalable Video*

*Coding*. In Proceedings of the 3rd Multimedia Systems Conference (MMSys), pages 149–154, 2012.

[11] 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 Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 419–426, 2013.

[12] N. Bouten, S. Latré, J. Famaey, F. De Turck, and W. Van Leekwijck. *Minimizing the impact of delay on live SVC-based HTTP adaptive streaming services*. In Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 1399–1404, 2013.

[13] N. Bouten, S. Latré, W. Van de Meerssche, K. De Schepper, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. *An autonomic delivery framework for HTTP Adaptive Streaming in multicast-enabled multimedia access networks*. In Proceedings of the IEEE Network Operations and Management Symposium (NOMS), pages 1248–1253, 2012.

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

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

[16] Y.-M. Hsiao, S.-W. Yeh, J.-S. Chen, and Y.-S. Chu. *A design of bandwidth adaptive multimedia gateway for scalable video coding*. In Proceedings of IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), pages 160–163, 2010.

[17] W. Pu, Z. Zou, and C. W. Chen. *Video adaptation proxy for wireless Dynamic Adaptive Streaming over HTTP*. In Proceedings of the International Packet Video Workshop (PV), pages 65–70, 2012.

[18] T. Wirth, Y. Sánchez, B. Holfeld, and T. Schierl. *Advanced Downlink LTE Radio Resource Management for HTTP-streaming*. In Proceedings of the ACM International Conference on Multimedia (MM), pages 1037–1040, 2012.

[19] S. Parakh and A. Jagannatham. *Game theory based dynamic bit-rate adaptation for H.264 scalable video transmission in 4G wireless systems*. In Proceedings of the International Conference on Signal Processing and Communications (SPCOM), pages 1–5, 2012.

[20] T. Sutinen, J. Vehkaperä, E. Piri, and M. Uitto. *Towards ubiquitous video services through scalable video coding and cross-layer optimization*. EURASIP Journal on Wireless Communications and Networking, 2012(1), 2012.

[21] H.-C. Lee and S.-M. Guu. *On the Optimal Three-tier Multimedia Streaming Services*. Fuzzy Optimization and Decision Making, 2(1):31–39, 2003.

[22] C. Hsu and M. Hefeeda. *Optimal bit allocation for fine-grained scalable video sequences in distributed streaming environments*. In Proceedings of ACM/SPIE Multimedia Computing and Networking Conference (MMCN), 2007.

[23] M. Hefeeda and C.-H. Hsu. *Rate-distortion Optimized Streaming of Fine-grained Scalable Video Sequences*. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 4(1):2:1–2:28, 2008.

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

[25] 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 Proceedings of the International Conference on Network and Service Management (CNSM), pages 336–342, 2012.

[26] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. *Modeling TCP Throughput: A Simple Model and Its Empirical Validation*. In Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), pages 303–314, 1998.

[27] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic. *Parallel TCP Sockets: Simple Model, Throughput and Validation*. In Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), pages 1–12, 2006.

[28] F. P. Kelly. *Charging and rate control for elastic traffic*. European Transactions on Telecommunications, 8(1):33–37, 1997.

[29] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. *Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability*. The Journal of the Operational Research Society, 49(3):237–252, 1998.

[30] W. Li, S. Wang, Y. Cui, X. Cheng, R. Xin, M. Al-Rodhaan, and A. Al-Dhelaan. *AP Association for Proportional Fairness in Multirate WLANs*. IEEE/ACM Transactions on Networking, 22(1):191–202, 2014.

[31] T. Kupka. *On the HTTP Segment Streaming Potentials and Performance Improvements.* PhD thesis, University of Oslo, 2013.

[32] L. Plissonneau and E. Biersack. *A Longitudinal View of HTTP Video Streaming Performance*. In Proceedings of the Multimedia Systems Conference (MMSys), pages 203–214, 2012.

[33] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. *Image quality assessment: from error visibility to structural similarity*. IEEE Transactions on Image Processing, 13(4):600–612, 2004.

[34] Z. Wang, L. Lu, and A. C. Bovik. *Video quality assessment based on structural distortion measurement*. Signal Processing: Image Communication, 19(2):121–132, 2004.

[35] M. Hazewinkel. *Confidence Estimation*. In Encyclopedia of Mathematics. Springer, 2001.

[36] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe. *A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems*. Technical report, Digital Equipment Corporation, 1984.

[37] L. A. Adamic and B. A. Huberman. *Zipf's law and the Internet*. Glottometrics, 3(1):143–150, 2002.

# 4

# QoE-Driven In-Network Optimization for Adaptive Video Streaming Based on Packet Sampling Measurements

**N. Bouten, R. de O. Schmidt, J. Famaey, S. Latré, A. Pras, F. De Turck.**

**Published in Elsevier Computer Networks (COMNET), April 2015.**

*The previous chapter focussed on the optimization of Video on Demand (VoD) HTTP Adaptive Streaming (HAS) delivery in managed networks. This chapter extends this with support for other networks as well by incorporating measurement and estimation techniques into the in-network optimization. Furthermore, next to the pure resource optimization, the current chapter optimizes the global Quality of Experience (QoE). In-network quality optimization agents are deployed, which monitor the available throughput using sampling-based measurement techniques. The in-network QoE optimization is achieved by solving a linear optimization problem both using centralized as well as distributed algorithms. The proposed hybrid QoE-driven approach allows the client to take into account the in-network decisions during the rate adaptation process, while still keeping the ability to react to sudden bandwidth fluctuations in the local network. The proposed approach allows improving existing autonomous quality selection heuristics by at least* $30\%$*, while outperforming an in-network approach using purely bitrate-driven optimization by up to* $19\%$*.*

## 4.1 Introduction

The increased popularity of video consumption over the Internet has led to the development of a range of protocols that allow adaptive video streaming over HTTP. Some of the major industrial players have introduced their proprietary protocols such as Microsoft's Silverlight Smooth Streaming[1], Apple's HTTP Live Streaming[2] and Adobe's HTTP Dynamic Streaming[3]. More recently, a standardized solution has been proposed by MPEG, called Dynamic Adaptive Streaming over HTTP (DASH) [1]. Although differences exist between these implementations, they are all based on the same basic principles: a video is split into temporal segments that are encoded at different quality rates, the client rate adaptation algorithm then dynamically adapts the quality, based on metrics such as average throughput and current buffer filling. The popularity of these adaptive HTTP-based streaming techniques was mainly induced by the advantages offered by HTTP streaming: the reuse of caching infrastructure, the reliable transmission over HTTP and the compatibility with firewalls.

State-of-the-art HTTP Adaptive Streaming (HAS) solutions embed the rate adaptation algorithm inside the client application. This allows the client to independently choose its playback quality and prevents the need for intelligent components inside the network, which are the main reasons HAS is used in Over-The-Top (OTT) scenarios. However, academia and industry are showing a growing interest in the use of HAS in managed networks [2][4,5], for example by optimizing the delivery by applying in-network bitrate adaptation[6] or by deploying IP multicasting to ease the distribution of linear TV HAS services [3][7]. In a recent survey, *Seufert et al.* argue that a centralized control unit or client-proxy based communication can enhance the quality and establish a fair Quality of Experience (QoE) distribution amongst competing clients [4]. The extensive content catalogue and increased flexibility in terms of supported devices of these OTT-services (e.g., YouTube, Hulu, Netflix) but delivered over the managed network, could greatly benefit both the provider and the end-user. However, in such environments, a purely client-driven approach has several significant disadvantages. First, the lack of coordination among clients leads to frequent quality oscillations and subop-

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

[2] Apple HTTP Live Streaming - http://tools.ietf.org/html/draft-pantos-http-live-streaming-12

[3] Adobe HTTP Dynamic Streaming - http://www.adobe.com/products/hds-dynamic-streaming.html

[4] Juniper Broadband TV Solution - http://www.juniper.net/us/en/local/pdf/solutionbriefs/3510463-en.pdf

[5] Velocix Live Streaming - http://www.rgbnetworks.com/pdfs/RGB-VelocixAdaptiveStreamingCDNWhitePaper0911-01.pdf

[6] Velocix Enhanced Video Experience - http://www.cachelogic.com/vx-portfolio/solutions/velocixeve

[7] Velocix Optimized Architectures - http://www.velocix.com/vx-portfolio/solutions/video-optimised-architecture

timal decisions [5–8]. Second, management policies, such as user subscription constraints (e.g., gold, silver and bronze) and guarantees on the delivered quality, cannot be enforced [9, 10]. In order to facilitate adoption of HAS for the delivery of multimedia services in a managed environment, these challenges should be tackled.

The first drawback of a purely client-driven approach is of course that control is distributed over the various clients and each client strives to optimize its individual quality. Several clients for which the video flows traverse the same path in the network therefore compete for the available bandwidth. This competition leads to instability in the quality decisions, causing frequent oscillations among different quality representations, bandwidth underutilization and unfairness between players. Most of the client heuristics try to maintain a buffer filling between the configured thresholds and use the download time of segments to estimate the available throughput. If a client has a sufficient buffer filling, the download process is paused until one of the thresholds is reached. During this pause, other clients can benefit from the released resources and increase their download rate. This causes their adaptation heuristic to overestimate the available throughput and wrongfully increase the quality for the next download. When the buffer filling of paused clients reaches the threshold again, their download process is resumed at the same quality level as before. This overestimation of available throughput could lead to congestion in the network, causing segments to arrive late, which in turn can lead to quality oscillations and eventually cause buffer depletion.

A second drawback is the inability for providers to exploit the merits of HAS in a managed environment, since the quality adaptation component is entirely controlled by the end-user. This prevents them from offering any type of QoE guarantees to their subscribers, possibly leading to low QoE and unfair quality distribution among clients. There is also no opportunity to offer service differentiation among the clients, delivering a higher QoE for a higher subscription fee. Moreover, since each client independently takes its quality decisions, there is no opportunity for coordinated management, optimizing the service provider's goals globally. The proposed approach allows service providers to impose specific management policies. In this way, guaranteeing QoE, service differentiation and global optimization can successfully be deployed.

To alleviate the aforementioned problems caused by wrong estimation of the available throughput, this chapter proposes a hybrid quality decision process. Several in-network decision agents are deployed along the delivery path, which monitor the available throughput and based on this, inform the clients on the optimal quality selection. This optimal quality selection is achieved by solving a linear optimization taking as input the monitoring information and the connected subscribers. Clients then take this quality selection as an input to their quality decision heuristic, together with the current buffer filling and throughput estimations.

In this way, the clients can still react to sudden throughput changes in their local delivery path, while they can react more dynamically and accurately to throughput changes in the network. To monitor the available throughput, the framework uses sampling techniques, allowing to make a distinction between HAS and cross traffic flows in a scalable way. The in-network quality selection can then take this available HAS throughput as input to its optimization and calculate the optimal quality assignation. This in-network optimization can take several objectives into account, based on the management policy of the provider. This chapter proposes a QoE-driven optimization, where the quality is maximized, while minimizing the impact of quality oscillations and buffer starvations. As will be shown in this chapter, this approach outperforms the state-of-the-art HAS rate adaptation in terms of QoE.

The benefits of the proposed approach are threefold. First, since the in-network optimization has a global view on the connected video clients, it is able to optimally divide the available resources. This reduces the number of quality oscillations, increases the network utilization and reduces unfairness between players, benefitting the overall QoE of the HAS video delivery service. Second, the sampled monitoring framework allows to accurately forecast future in-network available bandwidth at low capturing costs. Using these estimations as input to the optimization process allows the in-network approach to cope with dynamic networks. Third, since providers are now able to guide clients to certain quality decisions, they can create new business opportunities when adding subscription based management policies.

The proposed approach requires the in-network monitoring and QoE-optimization components to be deployed in the Internet Service Provider (ISP) network. There exist several scenarios in which this could be achieved, depending on which stakeholder deploys the components and offers the HAS streaming service. In a first scenario, the ISP wants to improve its HAS-based streaming service that is offered as part of its triple-play service. In this case, the ISP deploys the in-network components and offers the HAS streaming service as well, allowing subscribers to stream the content offered on the set-top-box, to their own devices (e.g., tablets, smartphones) as well. The second scenario pertains to an ISP that resells third party OTT services with better quality as part of a subscription plan. In this case, the ISP deploys both the in-network components and offers the service, or leases these components to the OTT or Content Delivery Network (CDN)[8] [11] service providers. ISPs such as Proximus[9] and Virgin Media[10] are currently offering Netflix as an additional service without affecting the home broadband connection. In a

[8]Akamai Aura Managed CDN - http://www.akamai.com/html/solutions/managed-content-delivery-network.html

[9]Netflix on Proximus TV - http://www.proximus.be/en/id_cr_netflix/personal/our-products/television/series-movies/proximus-and-netflix.html

[10]Virgin Media, Netflix on TiVo - http://store.virginmedia.com/digital-tv/channels/netflix.html

third scenario, an OTT provider[11] or CDN[12] deploys its own hardware components in the ISP network, which allows them to perform in-network optimization.

The remainder of this chapter is structured as follows. Section 4.2 discusses the related research on client-based and in-network HAS rate adaptation. Subsequently, the delivery architecture is described in Section 4.3. In Section 4.4 the in-network rate adaptation problem is formally defined, as well as the sampling-based monitoring. Section 4.5 evaluates the proposed delivery architecture and optimization algorithms and compares the approach with a client-based solution. Finally, Section 4.6 concludes the chapter.

## 4.2 Related Work

At the client side, each commercial HAS implementation comes with a proprietary client heuristic as discussed in Section 4.1. Several heuristics have been proposed in literature as well, each focussing on a specific deployment. *Miller et al.* propose a receiver-driven adaptation heuristic for DASH that takes into account a history of available throughput and the buffer level [12]. The quality is adjusted to attain a buffer level between certain target thresholds, this improves the stability of the quality and avoids frequent switching as a consequence of short-term throughput variations. *Jiang et al.* identified the problems that arise when multiple clients share a link [13]. The authors propose a variety of techniques that can help avoid said undesirable behavior, such as harmonic bandwidth estimation, stateful and delayed bitrate update and randomized scheduling of requests, which are grouped in the FESTIVE adaptation algorithm. *Tian et al.* show that there is a trade-off between responsiveness and smoothness for client-side DASH adaptations [14]. The proposed rate-switching logic provides a dynamic control of this trade-off according to the trend of the buffer growth. The approach uses machine-learning based TCP throughput prediction to support multiple servers simultaneously. In previous work [15] [16], the authors evaluated different client heuristics both for Advanced Video Coding (AVC) and Scalable Video Coding (SVC), applying optimizations such as pipelined and parallel download scheduling. The approach presented in this chapter is applicable to both AVC and SVC. *Liu et al.* discuss a video client heuristic that is suited for a CDN by comparing the expected segment fetch time with the experienced segment fetch time to ensure a response to bandwidth fluctuations in the network [17], while *Adzic et al.* present a client heuristic which is tailored for mobile environments [18].

Among others [5, 8], *Akhshabi et al.* compare several commercial and open source HAS players and indicate significant inefficiencies in each of them, such

[11] Netflix Open Connect - https://openconnect.itp.netflix.com

[12] Akamai Accelerated Network Partner - http://www.akamai.com/html/solutions/akamai-accelerated-network-partner.html

as frequent oscillations and unfairness when the number of competing clients increases [6, 7]. Said quality oscillations are known to have a negative impact on QoE [19] and cause inefficient resource utilization within the bottleneck network [7, 20]. In a recent survey, *Seufert et al.* argue that a centralized control unit or client-proxy based communication can enhance the quality and establish a fair QoE distribution amongst competing clients [4]. This chapter aims at controlling the quality by introducing global QoE management, reducing the number of quality oscillations and allowing to reduce the required buffer size.

By altering the behavior of the streaming server, stability and bandwidth efficiency can be increased. *Akhshabi et al.* propose server-side rate adaptation to cope with unstable streaming players due to ON-OFF patterns when they compete for bandwidth [21]. The system detects sudden rate fluctuations in the client playout and tries to solve them by shaping the sending rate at the server to resemble the bitrate of the stream. *Liu et al.* follow a comparable approach where the rate is shaped according to QoE maximization metrics [22]. These systems are able to restore the streaming session when oscillation or freezing occurs and then remove the shaping when the client has stabilized. The proposed approach is not only able to solve the problems of oscillation or freezes when they occur, but actively tries to prevent them, while at the same time optimizing the QoE. In previous work, it was shown that, although the proposed server-side rate shaping can increase stability, they are not able to achieve the stability offered by in-network quality optimization due to a reactive, rather than proactive behavior [23]. The current approach is also able to handle multiple origin servers and intermediate caching nodes.

*Li et al.* propose a collaboration scheme between CDNs and ISPs and peer-assisted CDNs to reduce the load on both peering links and internal ISP links [24]. Distributed CDN servers alter the manifests to associate chunks with regional storage servers or by changing or increasing the available video quality levels by transcoding the video. The approach proposed in this chapter could be complementary to the distributed CDN case, where intermediary storage servers exist and client sessions consuming their content are terminated in these nodes. The proposed approach further increases the efficient use of the ISP's network by minimizing the negative impact of competing sessions on QoE. *Famaey et al.* assess the impact of increased latency on QoE caused by the redirection of HAS requests in CDNs and propose updated request routing schemes in order to reduce the number of redirects [25].

Network level adaptations allow a more efficient use of the underlying network resources for HAS. *Essaili et al.* propose a TCP rate shaping mechanism on a per flow level to enhance the QoE by rewriting client requests and offering control to the network operator which has better information on the load and radio conditions in the cell [26]. *Houdaille et al.* propose to use traffic shaping in the residential gateway to implement bandwidth arbitration between competing

clients [8]. Others propose to exploit the advantages of Software Defined Networking (SDN) in a HAS environment to monitor the streaming sessions and, in conjunction with a client control plane, dynamically adjust video flow characteristics to ensure QoE [27]. Recently, *Mueller et al.* proposed to use HTTP/2.0 when deploying adaptive streaming and evaluate the overhead and impact on link utilization when using HTTP/2.0 for HAS [28]. Using new features in HTTP/2.0, such as server push, persistent connections and pipelining, HAS services can be improved. *Wei et al.* show that the increasing protocol overhead that is caused by decreasing the segment length can be overcome by automatically pushing a number of segments, allowing them to reduce the live latency [29, 30]. Using the server push feature proposed in HTTP/2.0 would allow the in-network optimization to force certain quality decisions onto the clients, increasing the overall QoE.

In previous work, it was proposed to use Differentiated Services (DiffServ) to guarantee the delivery of certain segments in a live streaming scenario [31]. This requires altering the relevant priorization fields for these specific segments and implementing DiffServ routers in the ISP network. An autonomic delivery framework for HAS-based Live TV and Time Shifted TV (TSTV) was presented in previous work [3, 32] which allows to reduce the consumed bandwidth by grouping unicast HAS sessions sharing the same content into a single multicast session. However, for Video on Demand (VoD) HAS sessions, the content is more diverse and only few sessions are potentially shared among multiple users. This prevents them to be grouped into a shared multicast session and therefore prevents them from being delivered in a scalable manner.

*Petrangeli et al.* use in-network proxies to determine the overall median chunk level requested by the clients and disseminates this information towards each client [33]. A reinforcement learning-based quality selection allows then to achieve fairness among the clients. *Krishnamoorthi et al.* use a set of intermediary proxies to evaluate the impact of caching policies on HAS [10]. Furthermore, they argue that client and proxy should cooperate and exchange information on buffer filling and cache contents, in order to optimize the delivery of cached segments. The proposed approach could be extended to incorporate the intelligent caching mechanism, as it is a mere relocation of the content server. Since the optimization is performed regularly, the caching dynamics and their impact on link utilization could be included during the optimization. The current approach extends the related work by furthermore alleviating instability and inefficiency problems that arise when multiple HAS clients compete. QDASH is a QoE-aware DASH system where an in-network measurement proxy is deployed to provide accurate bandwidth measurements to the client [34]. This measurement proxy only performs measurements on a per-client level, the proposed approach supports multi-client scenarios and estimates upstream available throughput to divide amongst them in order to maximize QoE.

Other approaches limit the available video quality by altering the data transported over the network. In [35] a graceful degradation of video quality is proposed when the network load increases. 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. Similar to this approach, *Latré et al.* proposed an in-network rate adaptation algorithm, responsible for determining which SVC quality layers should be dropped in combination with a PCN based admission control mechanism [36]. In [37], a prototype of an intermediary adaptation node is proposed, where the media gateway estimates the available bandwidth on the client link and extracts the supported SVC-streams. *Situnen et al.* propose dropping video layers based on their priority when network congestion arises for scalable video streaming over wireless networks [38]. Most of the aforementioned research focuses on the dropping of quality layers when congestion arises, meaning the quality is limited in the same way for all users. The proposed approach limits the maximum quality in a per client manner, allowing the service provider to differentiate the delivered video services based on the client's subscription. This allows the service provider to control the QoE on a subscriber level, and thus offering different subscription types for the VoD HAS services.

*Lee et al.* describe a three-tier streaming service where clients are connected through multiple intermediate proxies to a multimedia server [39]. The authors only consider live streaming, whereas the proposed system also supports VoD streaming. Furthermore, videos need to be transcoded in the intermediary proxies, in standard HAS however, the quality levels are discrete and fixed, causing the objective function in the proposed solution to change drastically and leading to the inability to use the max-min composition. In [40] [41], the authors focus on optimizing the allocation of bits of video sequences among multiple senders to stream to a single client. Peer-to-peer streaming and multi-server distributed streaming are the main use-cases of this approach, there is no simple extension of the work when multiple clients need to share the same server side bottleneck. Furthermore, this requires fine-grained scalable video streaming to support the allocation of non-overlapping bit ranges to multiple servers, while for HAS, fixed bitrate representations are available, encoded using advanced video coding, leading to video segments of which the quality cannot be improved in a straightforward way by downloading additional bit ranges. The current work however, could also be extended to support scalable video in a straightforward way.

This chapter is an extension to previous work on in-network quality management for HAS [23, 42]. This previous work only considered managed environments, where a fixed capacity is available. This approach was extended to allow dynamic scenarios where non-HAS traffic is measured to estimate the available residual capacity. To this end, several throughput forecasting techniques are presented and evaluated. Additionally, other client-side rate adaptation heuristics

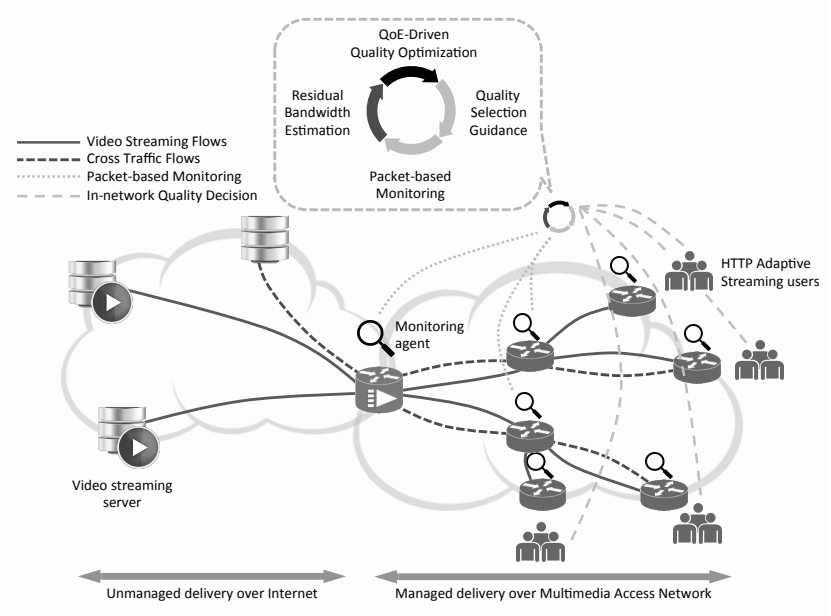



*Figure 4.1: In-network quality management architecture, showing a centralized component gathering monitoring information, estimating residual bandwidth which serves as input for the QoE-driven quality optimization and quality selection guidance for the clients. Further on in the chapter, also a distributed version is considered.*

were implemented and evaluated. Furthermore, both the model and evaluations were extended to take into account the impact of freezes, switches and playout quality on perceived QoE. These improvements over the previous approach allow a network-wide QoE optimization in a dynamic network environment.

## 4.3 QoE-Driven Delivery Architecture for Adaptive Video Streaming

This chapter proposes a hybrid approach, steering the rate adaptation algorithm at the client by an in-network component. It is deployed on one or multiple intermediary proxies and supports client-side rate adaptation algorithms by dynamically limiting the possible set of bit rates to select from. Figure 4.1 shows an overview of the in-network quality decision architecture and the different flows of data and information for a centralized deployment. Using a hybrid approach, where client heuristics take the in-network quality decisions as a maximum bound, allows clients to still react upon sudden network changes or scarcity in device resources, while increasing the overall quality and stability. One or more adaptation

agents can be deployed to optimize the HAS delivery, depending on the size of the delivery network. Using distributed optimization algorithms, several cooperating agents can be deployed along the delivery path to achieve a scalable in-network quality adaptation. The in-network rate adaptation agents are modeled to resemble an autonomic control loop and consists out of the following steps:

**Packet-based Monitoring** Controlling the streaming quality from within the network, requires measuring the current traffic traversing the streaming paths links via monitoring agents to reconstruct the cross traffic on the delivery path. Operators estimate this available bandwidth by reading interface counters over specific time periods (e.g., every 5 to 15 minutes), which could be accomplished by using Simple Network Management Protocol (SNMP)[13]. However, these measurements are of coarse granularity and might lead to very inaccurate estimations of bandwidth usage. More accurate approaches are available, but they often demand continuous packet capturing to calculate traffic statistics [43]. However, due to traffic rates observed nowadays, continuous packet capturing does not scale and requires expensive software/hardware. To avoid these issues, packet sampling is deployed within the monitoring process. Packet sampling provides a trade-off between accuracy and ease-of-use of bandwidth estimation approaches. That is, although providing only a summary of actual traffic, as compared to continuous packet capturing, sampled packet capturing provides much more granular data than interface counters at a limited capturing overhead. *Schmidt et al.* propose a monitoring technology using packet filtering and sampling to provide scalable packet-based monitoring in high-speed networks and use it for link dimensioning [44]. The monitoring architecture consists of agents embedded in switches and routers and a centralized controller. Several monitoring agents can be grouped to monitor the traffic along a fraction of the media delivery path. This allows both a centralized and distributed approach towards packet-based monitoring and in-network quality decision.

**Residual Bandwidth Estimation** To achieve dynamic quality adaptation, several packet-based monitoring agents are deployed along the HAS delivery paths. The monitoring information of these monitoring agents is then processed to estimate the bandwidth taken by the different flows. Using these estimations on each link's cross traffic, the residual bandwidth available for HAS along the path can be predicted. This provides the in-network quality control with a prediction of the future residual bandwidth available for HAS traffic. The packet-based residual capacity estimation will be discussed in Section 4.4.2. The HAS sessions that are

[13]Cisco Systems Inc. (2005) - http://www.cisco.com/image/gif/paws/8141/calculate_bandwidth_-snmp.pdf

monitored along the delivery path and the predicted residual bandwidth then serve as input towards the in-network quality decision algorithm.

**QoE-Driven Quality Optimization** This step takes the predicted residual bandwidth along the delivery paths as input and optimizes the selected quality for each individual HAS flow. In Figure 4.1, a centralized approach is shown, where the monitoring information of all the monitoring agents is forwarded to a centralized controller. This controller then has a full view on the delivery network and is able to optimize the QoE as will be discussed in Section 4.4.3. To provide a more scalable solution, multiple intermediate proxies could be deployed within the delivery network. They have a local view of the network and only require input of a subset of monitoring agents, limiting the communication overhead of the monitoring data. These intermediate proxies then cooperate to achieve a global policy as will be discussed in Section 4.4.4. Based on the management policy of the operator, the objective of this optimization can be altered.

**Quality Selection Guidance** The result of the aforementioned step is a list of quality limitations for each individual client. The clients are then guided towards selecting the optimal quality in order to optimize the objective. This could be done in several ways. A first method is to throttle the throughput of the client's flows [21] at the server which allows to impact the actions of the client decision heuristic in a transparent way. Since the client will measure a lower or higher throughput, depending on the configured rate at the server, the decision will be steered towards the optimal quality. A disadvantage of this approach is that the quality guidance process is indirect and it could possibly take a while for the client to converge to the optimal quality. A second approach is to rewrite the manifest files in a per client way, by leaving out quality representations that are not feasible under the current circumstances. This allows the client to still autonomously select the best quality, based on its measurements, while enforcing a certain maximum quality. Since the manifest is regularly updated, this approach requires the client to download the manifest frequently. A third way, is to extend HAS protocol to allow specification of optimal quality guidelines, which are then processed by the clients. Quality selection information could be embedded in the HAS header of the segments, causing limited overhead of only a few bytes without requiring the introduction of additional protocol messages. In this chapter, it was chosen to add additional information to the segment headers which is then processed by the proposed client, since the guidance is direct and causes limited overhead.

## 4.4 Algorithm Description

### 4.4.1 Definition of variables and assumptions

A network topology is considered that is modeled as a graph, consisting of a set of nodes $\mathcal{N}$, which encompasses servers $\mathcal{S} \subset \mathcal{N}$, proxies $\mathcal{P} \subset \mathcal{N}$, and clients $\mathcal{C} \subset \mathcal{N}$. A set of edges $\mathcal{E}$ connects the nodes in a logical tree topology which is typically used for video delivery networks, although the underlying physical network might not be a tree due to replication concerns. Each node $n \in \mathcal{N}$ has an incoming edge $e_{n^-} \in \mathcal{E}$ connecting the node to its predecessor $n^- \in \mathcal{N}$ as well as a set of outgoing edges $\mathcal{E}_{\mathcal{N}^+} \subset \mathcal{E}$ connecting to its successors $\mathcal{N}_n^+ \subset \mathcal{N}$ in the logical tree topology. Every edge $e \in \mathcal{E}$ has an associated capacity $B_e$ as well as an estimation for the cross traffic $T_{e,t}$ for the next timeslot $t$. This results in an estimated residual capacity $R_{e,t}$ which is available to assign to HAS traffic. Each video $v \in \mathcal{V}$ has an associated set of quality representations $\mathcal{Q}_v$, for which every quality representation $q \in \mathcal{Q}_v$ has a bit rate $\beta_q$. Each client $c \in \mathcal{C}$ has a unique delivery path $\mathcal{E}_c \subseteq \mathcal{E}$ from server to client. The set of clients that have an edge $e \in \mathcal{E}$ as part of their delivery path $\mathcal{E}_c$, is represented by $\mathcal{C}_e \subseteq \mathcal{C}$. Correspondingly, the set of clients for which the delivery path crosses a node $n \in \mathcal{N}$ is represented by $\mathcal{C}_n \subseteq \mathcal{C}$.

### 4.4.2 Packet-based residual capacity estimation

To estimate the future traffic on an edge $e$, sampled traffic measurements are used. The HAS traffic needs to be distinguished from unrelated traffic generated by other services, to estimate the impact of this cross traffic on the edge $e$. This can be obtained by intercepting packets and inspecting the TCP and HTTP packet headers. However, due to increasing traffic rates, full packet capturing is often not advised due to scalability issues. Packet sampling is an attractive alternative to continuous packet capturing, while still providing highly granular traffic measurements (as compared to, e.g., interface counters). If the sampled traffic amount $L_{e,t}$ was obtained at time $t$ at a rate $1/r$ (e.g., $r = 100$ for $1:100$ sampling) every $\tau$ seconds, the original amount of traffic $A_{e,t}$ can be estimated by Equation (4.1). This value gives an estimate of the actual amount of cross traffic during the interval $\tau$.

$$A_{e,t} = r \times L_{e,t} \tag{4.1}$$

Tailoring the decisions of the HAS clients during the next interval, requires a prediction of the future load. Several forecasting techniques can be used to predict the future load on $e$:

- **Exponential Weighted Moving Average** Exponentially Weighted Moving Average (EWMA) attempts to forecast the future load $T_{e,t}$ on edge $e$ based

on the packet-based traffic estimation $A_{e,t}$ using a smoothing factor $\alpha$ as shown in Equation (4.2) [45].

$$T_{e,t} = \alpha \times A_{e,t} + (1 - \alpha) \times T_{e,t-1} \tag{4.2}$$

- **Holt Winters** The non-seasonal Holt Winters (HW) predictor is a variation of EWMA, attempting to capture trends in time series [46]. A separate smoothing component $T^s_{e,t}$ and a trend component $T^t_{e,t}$ are defined in Equation (4.4) and (4.5) respectively and depend on the parameters $\alpha$ and $\beta$.

$$T_{e,t} = T^s_{e,t} + T^t_{e,t} \tag{4.3}$$
$$T^s_{e,t} = \alpha \times A_{e,t} + (1 - \alpha) \times T_{e,t-1} \tag{4.4}$$
$$T^t_{e,t} = \beta \times (T^s_{e,t} - T^s_{e,t-1}) + (1 - \beta) \times T^t_{e,t-1} \tag{4.5}$$

- **Exponential trend method** A variation of Holt's linear trend method is generated by using multiplicative adjustments to the level and slope, rather than additive. This leads to an Exponential Trend (ET) $T^t_{e,t}$, rather than a linear trend. These adjustments are shown in Equation (4.6), (4.7) and (4.8).

$$T_{e,t} = T^s_{e,t} \times T^t_{e,t} \tag{4.6}$$
$$T^s_{e,t} = \alpha \times A_{e,t} + (1 - \alpha) \times T_{e,t-1} \tag{4.7}$$
$$T^t_{e,t} = \beta \times (T^s_{e,t}/T^s_{e,t-1}) + (1 - \beta) \times T^t_{e,t-1} \tag{4.8}$$

- **Autoregressive model** In Autoregression (AR), a history of past values of the measured cross traffic are used to forecast the future load. Using a AR model of order $h$, the prediction can be performed as shown in Equation (4.9), where $c$ is a constant and the random variable $\epsilon_t$ is white noise. To determine the order $h$, the Akaike information criterion can be used [47], while least median of squares estimation can be used to find the parameters $\phi_i : i = 1..h$ [48].

$$T_{e,t} = c + \sum_{i=1}^{h} (\phi_i \times A_{e,t-i}) + \epsilon_t \tag{4.9}$$

- **Support Vector Regression** Time series can also be predicted using a lagged vector of previous measurements. The basic principle of Support Vector

Regression (SVR) is to estimate the output variable $T_{e,t}$), from $\phi(\mathcal{T}_{e,t})$. $\mathcal{T}_{e,t} = (A_{e,t-h}, .., A_{e,t-1})^\intercal$ is an input vector containing $h$ previous measurements [49]. Using a non-linear mapping $\phi(.)$, the vector $\mathcal{T}_{e,t}$ is projected to a higher dimensional space. The regression model is then shown in Equation (4.10), where $\omega$ is the weight vector and $b$ the bias term. Using Sequential Minimal Optimization (SMO), the regression problem can be solved [50].

$$T_{e,t} = \omega^\intercal \times \phi(\mathcal{T}_{e,t}) + b \tag{4.10}$$

- **Multi Layer Perceptron** Also a two-layer Multi Layer Perceptron (MLP) can be used to predict the future load. The network consists of a single linear output activation ($n_0$) and $m$ hidden sigmoid activations ($n_1, ..., n_m$) and takes as inputs a history of $h$ measurements, normalized with respect to the available capacity in the interval $\tau$: $\tau \times B_e$. Each node takes as input the output of the preceding nodes in the network, a weight vector $\omega^n$ and a bias value $b_n$. The linear output and hidden sigmoid activations are shown in Equation (4.11) and (4.12) respectively, where $\mathcal{O}_i$ are intermediary outputs of the hidden layer nodes. The logistic sigmoid function $\varphi(x) = \frac{1}{1+\exp(-x)}$ is used as activation function for the hidden layer.

$$T_{e,t} = \tau \times B_e \times (b_0 + \sum_{i=1}^{m} \omega_i^0 * \mathcal{O}_i) \tag{4.11}$$

$$\mathcal{O}_i = \varphi\left(b_i + \sum_{j=0}^{h} \omega_j^i * \frac{A_{e,t-j}}{\tau \times B_e}\right) \tag{4.12}$$

The residual capacity at edge $e$ can then be estimated as defined in Equation (4.13). According to *Padhye et al.*, the maximum achievable throughput $B$ for a TCP connection subject to a round trip time $RTT$ and maximum window size $W_{max}$, is limited by $\frac{W_{max}}{RTT}$ [51]. This adds an additional constraint on the per flow achievable throughput a shown in Equation (4.14).

$$R_e = B_e - \frac{T_{e,t}}{\tau} \tag{4.13}$$

$$\forall c \in \mathcal{C} : \sum_{q \in \mathcal{Q}_c} a_{c,q} \times \beta_q \leq \frac{W_{max,c}}{RTT_c} \tag{4.14}$$

### 4.4.3 QoE-driven quality optimization

The problem is modeled as an Integer Linear Programming (ILP) and consists of maximizing the QoE over all clients $c \in \mathcal{C}$, while adhering to the edge bandwidth constraints. The solution is characterised by a boolean decision matrix $A$. The element $a_{c,q} \in A$ is equal to 1 if quality $q \in \mathcal{Q}_v$ is selected for client $c \in \mathcal{C}$, and 0 otherwise. The constraints in Equation (4.15) and (4.16), state that the decision variables are boolean values and that only one quality representation can be selected per client. The total consumed bandwidth of HAS-traffic on every edge $e \in \mathcal{E}$ caused by all clients $c \in \mathcal{C}_e$ should not exceed the estimated residual bandwidth for HAS traffic $R_{e,t}$ as defined in Equation (4.17).

$$\forall c \in \mathcal{C}, \forall q \in \mathcal{Q}_v : a_{c,q} \in [0,1] \tag{4.15}$$

$$\forall c \in \mathcal{C} : \sum_{q \in \mathcal{Q}_v} a_{c,q} = 1 \tag{4.16}$$

$$\forall e \in \mathcal{E} : \sum_{c \in \mathcal{C}_e} \sum_{q \in \mathcal{Q}_v} a_{c,q} \times \beta_q \leq R_e \tag{4.17}$$

The provider can choose to optimize video delivery in several ways using a different objective function. One possibility is to maximize the total video bitrate over all clients using the following optimization function:

$$\max \sum_{c \in \mathcal{C}} \sum_{q \in \mathcal{Q}_v} a_{c,q} \times \beta_q \tag{4.18}$$

A major drawback of the aforementioned approach is that it neglects the impact of quality switches on QoE [19]. Therefore, a QoE-metric for HAS was adapted to include both quality and switching information during the in-network optimization. The QoE-driven objective aims to optimize the global QoE over all clients. For HAS services, the QoE as shown in Equation (4.19) is a weighted combination of the average delivered quality ($\mu$), the standard deviation of quality ($\sigma$) [52, 53] and a correction factor to incorporate the impact of frame freezes ($\phi$) [54]. The formulas for calculating the values $\mu$, $\sigma$ and $\phi$ are defined in Equation (4.20) to (4.24), with $K$ the number of played segments, $N$ the number of quality levels for video, $Q_k$ the quality played for segment $k \in [1, K]$ and $F$ the set of frame freezes.

$$eMOS = \alpha \times \mu - \beta \times \sigma - \gamma \times \phi + \delta \tag{4.19}$$

$$\mu = \frac{\sum_{k=1}^{K} \frac{Q_k}{N}}{K} \tag{4.20}$$

$$\sigma = \sqrt{\frac{\sum_{k=1}^{K} (\frac{Q_k}{N} - \mu)^2}{K}} \tag{4.21}$$

$$F_{freq} = \frac{|F|}{K} \tag{4.22}$$

$$F_{avg} = \frac{\sum_{f \in F} duration(f)}{|F|} \tag{4.23}$$

$$\phi = \frac{7}{8} \max\left(\frac{\ln(F_{freq})}{6} + 1, 0\right) + \frac{1}{8} \min\left(\frac{F_{avg}}{15}, 1\right) \tag{4.24}$$

The approximation for MOS in Equation (4.19) was used to model the objective function. The term $\phi$, quantifying the impact of freezes was omitted, since the constraints of the proposed approach guarantee the delivery of the lowest quality, thus avoiding freezes. The average quality over time $\mu$ includes the decision variables for each client. The formula to calculate the switching term $\sigma$ is dependent on $\mu$ and is quadratic in this term. Including the estimated MOS as is, would yield a non-linear objective function. In order to avoid this, the calculation of $\mu$ and $\sigma$ values is split into several calculations $\mu_{c,q}$ and $\sigma_{c,q}$ per client $c$ and per quality $q$. Since the decision variables $a_{c,q}$ are modeled as binary values, multiplying this decision variable with the corresponding $\mu_{c,q}$ and $\sigma_{c,q}$ values allows a linear objective function. To this end, a history $\mathcal{H}_c$ of previous quality decisions is required for every client $c \in \mathcal{C}$. This history allows calculating the average quality $\mu_{c,q}$ for every client $c \in \mathcal{C}$ and every possible quality decision $q \in \mathcal{Q}_v$ for the next timeslot in Equation (4.25). The variation in quality $\sigma_{c,q}$ can be calculated for each client using the set of averages $\mu_{c,q}$ for each quality decision $q \in \mathcal{Q}_v$ as shown in Equation (4.26).

$$\forall c \in \mathcal{C}, \forall q \in \mathcal{Q}_v : \mu_{c,q} = \frac{1}{|\mathcal{H}_c| + 1} \left( \frac{q}{|\mathcal{Q}_v|} + \sum_{h_{c,t} \in \mathcal{H}_c} \frac{h_{c,t}}{|\mathcal{Q}_v|} \right) \tag{4.25}$$

$$\forall c \in \mathcal{C}, \forall q \in \mathcal{Q}_v : \sigma_{c,q} = \sqrt{\frac{1}{|\mathcal{H}_c| + 1} \left( \left( \frac{q}{|\mathcal{Q}_v|} - \mu_{c,q} \right)^2 + \sum_{h_{c,t} \in \mathcal{H}_c} \left( \frac{h_{c,t}}{|\mathcal{Q}_v|} - \mu_{c,q} \right)^2 \right)} \tag{4.26}$$

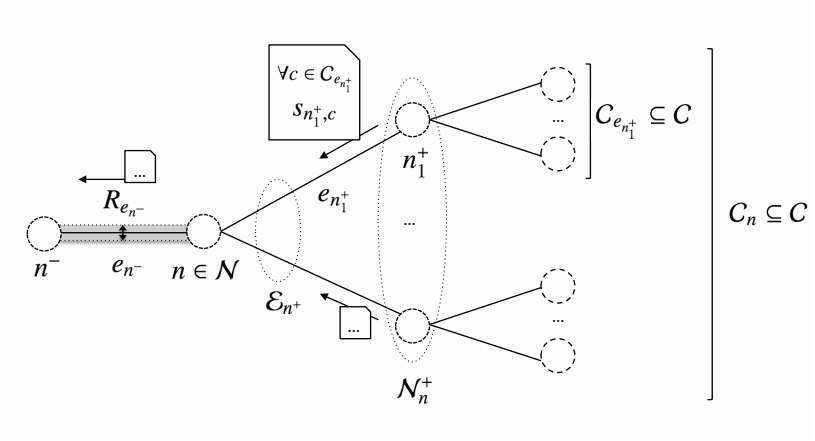



*Figure 4.2: Distributed optimization process for a node $n$ gathering monitoring information from its upstream link $e_{n^-}$ to estimate $R_{e_{n^-}}$ and downstream restrictions $s_{n^+,c}$ from its successor node set $\mathcal{N}^+$.*

Calculating the specific averages $\mu_{c,q}$ and deviations $\sigma_{c,q}$ for each possible decision allows to formulate the objective function without introducing quadratic terms for the decision variables. Using the decision variable $a_{c,q}$, the estimated MOS can then be maximized as follows:

$$\max \sum_{c \in \mathcal{C}} \sum_{q \in \mathcal{Q}_v} a_{c,q} \times (\alpha \times \mu_{c,q} - \beta \times \sigma_{c,q} + \delta) \tag{4.27}$$

### 4.4.4 Distributed QoE-driven quality optimization

The number of constraints for the QoE-driven quality optimization grows significantly with the number of proxies and clients in the service delivery tree. Solving the optimization problem using only one agent will increase the execution times. Prolonged optimization times endanger the ability to react adequately to sudden throughput changes along the edges of the delivery network. The problem can be distributed over the different nodes along the delivery network to obtain a scalable solution. The proposed approach uses a bottom-up technique, that was described in the previous chapter.

Figure 4.2 shows a graphical overview of how the distributed optimization process is performed. Each node $n \in \mathcal{N}$ monitors the upstream edge $e_{n^-}$ and estimates the residual bandwidth $R_{e_{n^-}}$ for HAS traffic along that edge. Using this information the local optimization objective for node $n$ can be formulated as shown in Equation (4.28). This optimization is constrained by the available HAS capacity of the upstream edge as detailed in Equation (4.29).

$$\max \sum_{c \in \mathcal{C}_n} \sum_{q \in \mathcal{Q}_v} a_{c,q} \times (\alpha \times \mu_{c,q} - \beta \times \sigma_{c,q} + \delta) \tag{4.28}$$

$$\sum_{c \in \mathcal{C}_n} \sum_{q \in \mathcal{Q}_v} a_{c,q} \times \beta_q \leq R_{e_{n^-}} \tag{4.29}$$

$$\forall c \in \mathcal{C}_n, \sum_{q \in \mathcal{Q}_v : q > s_{n^+,c}} a_{c,q} = 0 \tag{4.30}$$

The local optimization is then constrained by (4.15) and (4.16) as before but only for the subset $\mathcal{C}_n$ of clients for which the traffic traverses node $n$. In order not to violate any bitrate limitations further downstream the topology, the local optimization is constrained by the solutions obtained by its set of successor nodes $\mathcal{N}_n^+$. For each client $c \in \mathcal{C}_n$, $s_{n^+,c}$ determines the maximum quality a client is allowed to download according to the local optimizations downstream. Equation (4.30) puts an additional constraint on the optimization determining that the selected quality for client $c$ is not allowed to violate the downstream limitations. The aforementioned distributed approach has several advantages. Each node only requires local information on the upstream edge, while the number of edge constraints per local optimization process is equal to one. Furthermore, since the optimization for different subtrees is independent of subtrees at the same level, the processes can be executed in parallel.

The complexity of the optimization can be reduced at the expense of optimality by moving from an ILP formulation towards a Relaxed Linear Programming (LP) formulation. This can be achieved by removing the boolean constraints on the decision variables $a_{c,q}$ as defined in Equation (4.15) and replacing them by the following floating point decision variables as shown in Equation (4.31). The reader is referred to the previous chapter on how this floating point solution can be transformed into an integer solution.

$$\forall c \in \mathcal{C}_n, \forall q \in \mathcal{Q}_v : 0 \leq a_{c,q} \leq 1 \tag{4.31}$$

## 4.5 Evaluation Results

### 4.5.1 Experiment setup

A VoD HAS scenario was implemented using the discrete-event network simulator NS3[14], simulating the transmission of HAS-based video [42]. The framework has been extended with support for packet-based measurements in network routers and switches. The HAS servers and proxies have been adapted to incorporate

[14] ns-3 - https://www.nsnam.org

these measurements during the QoE-driven network optimization. For the autonomic HAS Clients, several heuristics found in literature were implemented. A first implementation uses the *Microsoft Smooth Streaming (MSS)* algorithm, which is based on the implementation of an open source version of the algorithm of the *MSS* video player[15] and is extensively described by *Famaey et al.* [15]. A second implementation is based on the heuristic proposed by *Miller et al.* [12], which is a receiver-driven adaptation algorithm based on buffer filling level and throughput estimations. This heuristic is referred to as *Miller*. A third heuristic, called *Festive*, is based on the implementation described by *Jiang et al.* using randomized scheduling and stateful bitrate selection [13]. The parameters of the aforementioned heuristics were optimized to attain the best QoE for a variety of scenarios.

An additional client heuristic was implemented, which downloads each segment using the QoE management quality decision is proposed in this chapter. This approach is referred to as *AVC Steered*. This heuristic takes the signalled quality decision as an input and decides wether the assigned quality level is feasible under the current network circumstances. This allows the *AVC Steered* client to address local network congestion, which the QoE-driven in-network optimization is unable to take into account.

The optimization algorithms were implemented using the IBM CPLEX[16] solver. Two versions of the optimization algorithm are used: an *Exact* calculation, modelled as an ILP and the relaxation of the problem denoted as *Relaxed*. Both the *QoE-driven optimization* and the simpler *bitrate optimization* were implemented, which only optimizes the average quality, disregarding quality oscillations [42]. Next to a *Centralized*, a *Distributed* heuristic approach was implemented to address scalability issues. This *Distributed* approach requires upstream information exchange between the different cascading proxies. This exchange is modelled as network communication to take into account network delays when exchanging solutions and installing the optimal configuration at the clients. Also the delay introduced by forwarding monitoring information and processing this data to predict the future bandwidth are taken into account during the simulations.

Figure 4.3 shows a typical tree-structured video service delivery network overlay. The first level has $K$ branches, while the second level has $M$ branches, both with a default value of $4$. To each of these branches, a number of connected clients is assigned randomly within the interval $[0, N]$, representing the number of active clients out of the number of connected homes $N$. The links are dimensioned proportionally to the maximum number of clients $N$ per branch. The average Round Trip Time (RTT) for each client $c$ is set to $RTT = 40ms$ [55]. Clients are started using a Weibull startup process with shape $2.5$ and mean of $300s$. The *Big Buck*

[15]Source available from https://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming

[16]IBM ILOG CPLEX Optimizer: http://www.ibm.com/software/integration/optimization/cplex-optimizer/

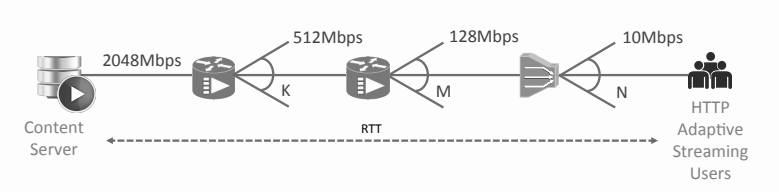



*Figure 4.3: Network topology, modeling a typical video service delivery network.*

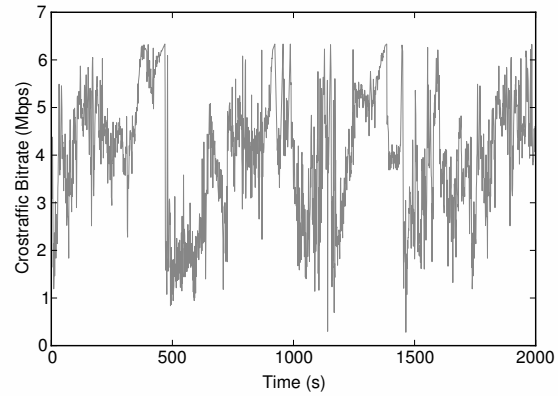



*Figure 4.4: Example cross traffic trace.*

*Bunny* video[17] was encoded at 7 different quality rates and divided into 200 segments with an average duration of 2 seconds. Table 4.1 gives an overview of the different quality layers and their associated bitrates.

To introduce cross-traffic in the network with a realistic degree of variability, the cross traffic was modelled based on a set of bandwidth traces described by *Riiser et al.* [56]. The traces[18] provide a highly variable throughput, which makes the prediction of the future bandwidth challenging. They have a total duration of about 220 minutes. The available bandwidth fluctuates between $202bps$ and $6,335kbps$ with an average of $2,192kbps$ and a standard deviation of $1,317kbps$. The bandwidth traces were cut in $2,000$ second parts which were used to generate traffic on different paths during the evaluations. Figure 4.4 shows an example of such a bandwidth trace excerpt of $2,000$ seconds. As discussed in Section 4.4.2, several forecasting techniques can be used to predict the future load on an edge. The parameters of each of the presented prediction techniques were estimated using a training set of the bandwidth data. For the implementation of AR, SVR and

[17] Big Buck Bunny available from http://www.bigbuckbunny.org/

[18] Dataset available from: http://home.ifi.uio.no/paalh/dataset/hsdpa-tcp-logs/bus.ljansbakken-oslo/

*Table 4.1: Overview of the quality layers for the Big Buck Bunny video.*

| Quality Layer Index | Average Bitrate (kbps) | Average PSNR (dB) |
|---|---|---|
| 0 | 300 | 32.04 |
| 1 | 427 | 32.72 |
| 2 | 608 | 34.41 |
| 3 | 866 | 35.71 |
| 4 | 1,233 | 36.88 |
| 5 | 1,636 | 37.64 |
| 6 | 2,436 | 40.07 |

MLP prediction, WEKA 3[19] was used to perform training and prediction. Unless otherwise stated MLP prediction is used during the evaluations.

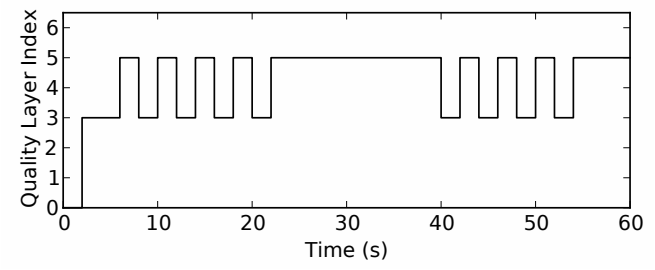
The estimated average Mean Opinion Score (MOS) defined by Equation (4.19) was used to evaluate the performance of both the purely client-based approaches and the in-network assisted approach [52–54]. To tune the parameters ($\alpha$, $\beta$, $\gamma$ and $\delta$), a set of 15 adaptive streaming scenarios was generated to assess the end-user perception of bitrate switching and buffer starvations using the ***Big Buck Bunny*** video. Figure 4.5 shows a graphical example of three such test scenarios, where it is tried to capture the impact of frequent switching (a), gradual switching (b) and buffer starvations (c). During a subjective screening test with 10 experts in the field of video streaming, the MOS values for each of the test sequences ($n \in [1, N]$) were measured [57]. By minimizing the Root Mean Squared Error (RMSE) between the predicted values $MOS_{pred,n}$ and the actual measured values $MOS_{subj,n}$, the parameters were tuned:

$$\min \sum_{n \in [1,N]} \frac{(MOS_{pred,n} - MOS_{subj,n})^2}{N} \tag{4.32}$$

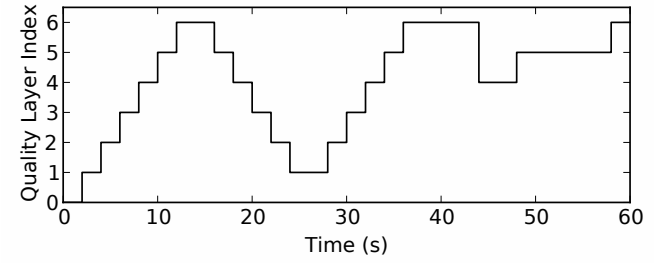
The following values were obtained: $\alpha = 5.67$, $\beta = 6.72$, $\gamma = 4.95$ and $\delta = 0.17$. The MOS estimation assumes that there is a linear relationship between the relative quality level of each video representation and the MOS score for that quality level.

The individual values of each of these QoE-terms are used during the discussion to indicate how the behavior of the evaluated approaches differs. Since frame freezes have a negative impact on QoE, the total buffer starvation time of each client was measured and is indicated as the total buffer starvation time in seconds. Also the total number of quality oscillations each client experiences, as well as the average played quality bitrate in Mbps was examined. To indicate the buffering behavior of the different approaches, the average buffer filling is expressed as a percentage of the maximum allowed buffer. The in-network optimization uses the

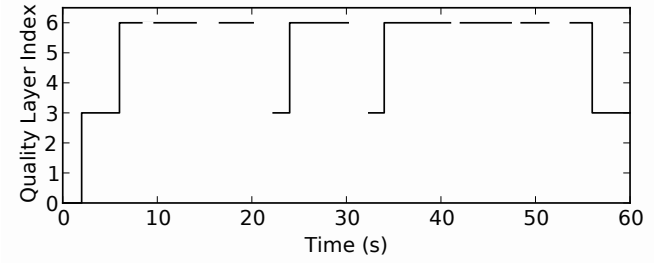
[19]WEKA: Data Mining Software in Java - http://www.cs.waikato.ac.nz/ml/weka/




(a)




(b)




(c)

*Figure 4.5: Graphical overview of three example sequences that were used to assess the end-user subjective quality perception. Gaps in (c) indicate the occurrence of buffer starvations.*

prediction methods discussed in Section 4.4.2 to forecast the available throughput for HAS during the next timeslot. To indicate the correlation between the actual values and the estimated values, the Pearson Correlation Coefficient is used. All of the following results are averaged over $n = 20$ iterations, using different cross traffic traces, with the graphs showing the $95\%$ confidence intervals.

To assess the impact of the different parameters on the in-network QoE optimization, first, a parameter analysis is carried out where simulations are performed for different values of the *history size* $|\mathcal{H}_c|$, the *optimization interval* $\tau$, the *sampling rate* $r$ and the *buffer size* $B$. The optimal values (as discussed in Section 4.5.2.5) for the respective parameters obtained during the analysis are used further on during the evaluations. Next, the different forecasting techniques are evaluated in terms of precision and their impact on QoE. The interference between the in-network optimization and client-side adaptation are discussed afterwards. Furthermore, the overheads of the proposed approach introduced by exchanging partial solutions and measurement data are quantified and the scalability of the proposed approach are evaluated for increasing *network size* $M$ and *number of homes* $N$.

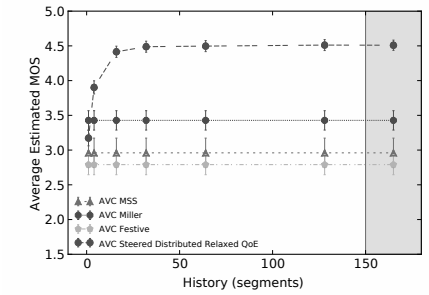
## 4.5.2 Parameter analysis

### 4.5.2.1 Impact of the decision history

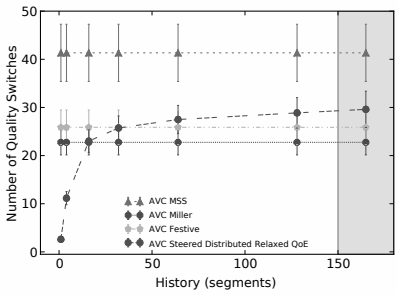
In order to optimize the QoE in terms of average quality and quality switches, the in-network quality optimization maintains a list of previous quality decisions on a per client basis. The size of this history $|\mathcal{H}_c|$ has an impact on the actual quality perceived by the client in terms of MOS. This is illustrated in Figure 4.6(a) where the history size $|\mathcal{H}_c|$ is varied within $[1, 2, 4, 8, 16, 32, 64, 128, \infty]$, with $|\mathcal{H}_c|= \infty$ corresponding to the situation where the proposed approach keeps a history of all decisions for each client $c \in \mathcal{C}$. Increasing the size of the maintained history has a positive effect on the average estimated MOS. A history of only 1 quality decision will cause the optimization to severely penalize quality switches, leading to a low number of quality switches as indicated in Figure 4.6(b) but at the same time preventing the decision algorithm to increase the quality, negatively impacting the average quality as indicated in Figure 4.6(c). With a history of only 1 decision, the impact of the current decision will be much higher than with a larger history, leading to a high variance which heavily increases the switching penalty and thus prevents the optimization to switch to another quality. There is a local optimum for a history size $|\mathcal{H}_c|$ of 128 previous decisions.

### 4.5.2.2 Impact of the optimization interval

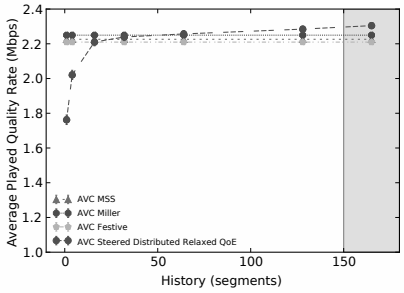
The in-network optimization calculates the optimal quality for each client when clients join or leave the network. Additionally, to adapt to the fluctuating cross




(a) Impact on average estimated MOS

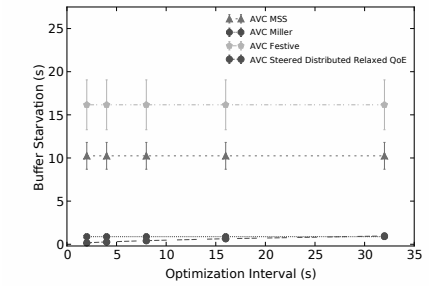



(b) Impact on average number of switches

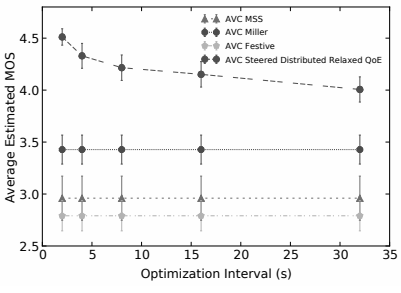



(c) Impact on average bitrate (Mbps)

*Figure 4.6: Impact of history size $|\mathcal{H}_c|$ ($RTT = 40ms$, $r = 100$, $\tau = 2s$, $B = 12s$, $N = 32$). These results show a local optimum of $|\mathcal{H}|_c = 128$.*




(a) Impact on buffer starvations (s)




(b) Impact on average estimated MOS

*Figure 4.7: Impact of optimization interval $\tau$ ($RTT = 40ms$, $r = 100$, $|\mathcal{H}_c| = 128$, $B = 12s$, $N = 32$). Setting the optimization interval $\tau$ at the segment length of $2s$ yields the highest QoE at the cost of frequent optimization.*

*Table 4.2: Pearson correlation between the estimated throughput for the next interval $\tau$ and the actual cross traffic rate. A sampling size of $r = 100$ yields a high Pearson correlation of $\rho = 0.965$.*

| Sampling size ($r$) | Pearson Correlation ($\rho$) |
|---|---|
| 10 | 0.970 |
| 100 | 0.965 |
| 1,000 | 0.939 |
| 10,000 | 0.827 |

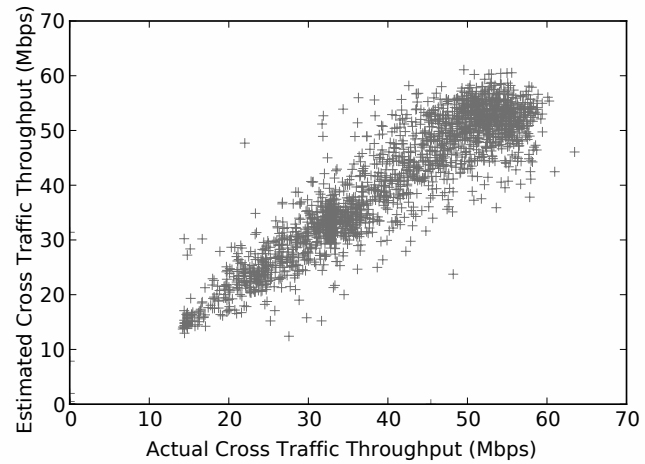
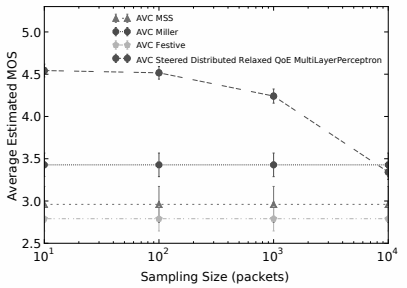
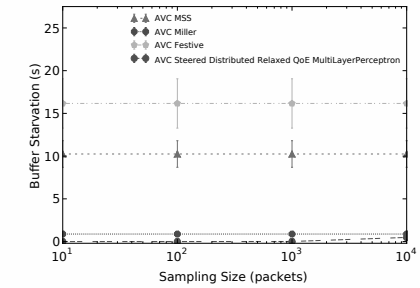



*Figure 4.8: Impact of sampling size $r$ on packet-based throughtput estimation for $r = 100$ showing a high correlation.*

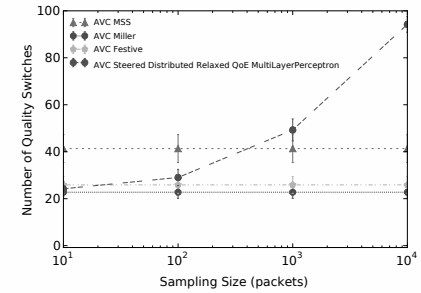
traffic, the optimization is performed periodically. Figure 4.7(a) shows the increased risk of running into buffer starvations when the optimization interval $\tau$ is increased. This can be attributed to the larger timespan between two consecutive bandwidth estimations, leading to less accurate cross traffic estimations and quality decisions when traffic fluctuates heavily during that period of time. An interval of $2s$ allows the in-network QoE-optimization to completely avoid frame freezes, positively impacting the QoE, while purely client-driven approaches are suffering from frequent frame freezing due to wrong estimations of the available throughput. Figure 4.7(b) shows the impact on the average MOS of these frame freezes. This shows the tradeoff between optimization frequency and overal QoE. Under highly fluctuating cross traffic, optimizing the quality decisions every $2s$ yields a $12\%$ improvement compared to optimizing every $32s$. The $2s$ interval corresponds to the segment length of the video, indicating that optimizing the quality for each segment produces optimal results.




(a) Impact on average estimated MOS




(b) Impact on buffer starvations (s)

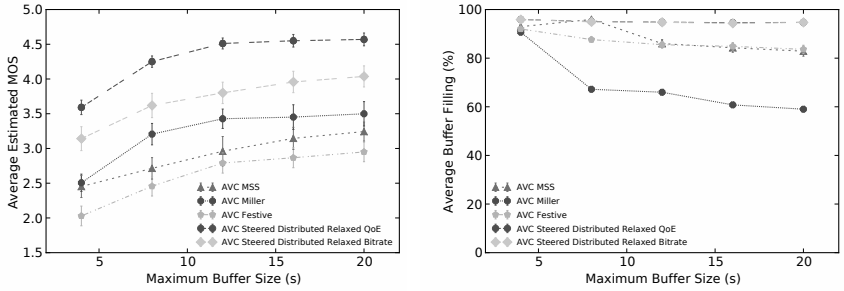



(c) Impact on number of quality switches

*Figure 4.9: Impact of sampling size $r$ ($RTT = 40ms$, $|\mathcal{H}_c|= 128$, $\tau = 2s$, $B = 12s$, $N = 32$). Increasing the sampling size $r$ beyond $10^3$ negatively impacts the QoE due to wrong estimations on the future throughput. Setting $r = 100$ gives good estimations at a limited sampling cost of $1\%$.*

#### 4.5.2.3 Impact of the sampling rate

When sampling network traffic at a rate $1/r$, there is a tradeoff between the scalability of the monitoring and the precision of the estimated bandwidth as illustrated in Figure 4.8 and Table 4.2. Randomly sampling the traffic with a sampling size $r = 100$ as shown in Figure 4.8 yields precise estimations using MLP with very few outliers and a high correlation of $\rho = 0.965$. Increasing the sampling size $r$ has a negative impact on estimation precision and as a result, a negative impact on the in-network optimization process as illustrated in Figure 4.9(a). Making incorrect predictions on the estimated throughput of the cross traffic can yield wrong decisions during the optimization, leading to an increased number of frame freezes and more frequent switching as illustrated in Figure 4.9(b) and Figure 4.9(c) respectively. A sampling size of $r = 100$ allows accurate predictions of the throughput, while limiting the overhead to sampling only $1\%$ of the packets.




(a) Impact on average estimated MOS

(b) Impact on average buffer filling (%))

*Figure 4.10: Impact of buffer size $B$ ($RTT = 40ms$, $r = 100$, $|\mathcal{H}_c| = 128$, $\tau = 2s$, $N = 32$). For a buffer of about $4s$, the in-network optimization is able to achieve a similar QoE as the best performing client-side heuristic using a buffer of four times that size.*

#### 4.5.2.4 Impact of the buffer size

Figure 4.10(a) shows the impact of the buffer size on average estimated MOS for *MSS*, *Miller*, *Festive*, bitrate-based and QoE-based optimization. Depending on the buffer size, the in-network QoE-driven optimization is able to achieve an improvement between $30\%$ and $43\%$ in terms of estimated MOS compared to the best client-side adaptation heuristic (*Miller*). The bitrate-based optimization allows increasing the estimated MOS with $10\%$ to $25\%$. These results show a significant increase of $13\%$ to $19\%$ in average QoE when also quality oscillations are included during the optimization process, instead of only optimizing the quality in terms of average bitrate per client. The maximum improvement over traditional autonomic adaptation is achieved for a buffer of $8s$. Starting from $12s$, increasing the buffer size for the in-network optimization has only a limited effect on the average MOS, while the client-side algorithms continue to improve. This can be attributed to the increased quality stability and larger safety margin to cope with bandwidth changes when the buffer is increased.

The higher quality stability can be achieved by the QoE-driven optimization for smaller buffer sizes as there is additional knowledge on estimated throughput and optimal quality decisions from within the network. As shown in Figure 4.10(b), the in-network optimization almost completely fills the buffer for any size, while client-side heuristics try to improve the quality when the buffer filling is sufficiently high and thus trade in a higher buffer filling for a quality increase. Especially when considering live streaming, small buffers are important since they allow reducing the latency with respect to the live event. Furthermore, maintaining only a limited buffer reduces the memory costs of the video application. For a buffer of about $4s$, the in-network optimization is able to achieve a similar QoE as

the best performing client-side heuristic using a buffer of four times that size. Using in-network optimization allows shrinking the client-side buffer without compromising the QoE.

#### 4.5.2.5 Selected parameter values

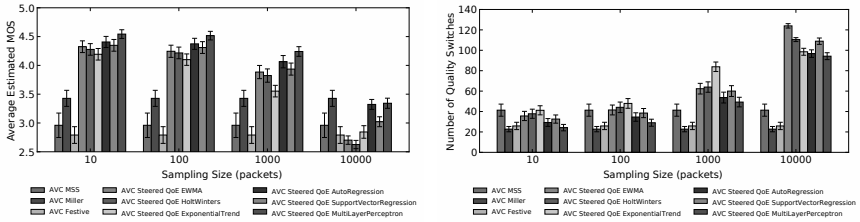
Based on the above parameter analysis, the best configuration is determined to be $|\mathcal{H}_c|= 128$, $\tau = 2s$, $r = 100$ and $B = 12s$. These parameter configurations allow increasing the overall QoE with about $30\%$ compared to pure client-side state-of-the-art adaptation heuristics. In the previous sections, it was shown that keeping a history of previous decisions allows the in-network optimization to reduce the number of quality oscillations, which benefits the overall QoE. Choosing a history that is too small, will prevent quality switching, but may impact the quality rate. Setting the history too large, increases storage overhead. Therefore, it is suggested to keep a history $|\mathcal{H}_c|$ of $128$ previous decisions. In a highly dynamic network environment, it is important to frequently reassess the optimal quality allocation. Not only clients joining or leaving the network can cause suboptimal or infeasible allocations, also the fluctuating available network throughput can impact the decisions. Therefore, the optimization interval $\tau$ should be sufficiently small in order to respond to these changes in a timely fashion. Since clients need to make a decision for every segment they download, it is advised to select $\tau$ equal to the segment length of $2s$. Continuously capturing packets is not scalable and increases the complexity of capturing equipment and thus the overall network cost. Therefore, packet sampling is deployed within the monitoring process. Sampling packets at a rate $r = 100$, allows a high correlation $\rho = 0.965$, while reducing the capturing to $1\%$ of total data packets. A buffer storing segments deployed at the client side, allows resolving temporal throughput fluctuations. A large buffer increases storage requirements at the client device and increases latency with respect to the live signal. In case of channel switching the prefetched content of this buffer is cleared, which means that the buffered content was not useful. Therefore, the size of the buffer should be chosen large enough so that temporal fluctuations can be resolved, but at the same time sufficiently small to limit the storage, useless prefetching and live latency. A buffer size of $12s$ is proposed, since it requires limited storage and greatly improves QoE over client-side heuristics. The optimal parameter values deduced during the analysis are used to evaluate the proposed system further on.

### 4.5.3 Impact of the forecasting method

In Section 4.4.2, several forecasting techniques were proposed to estimate the future available capacity for the HAS traffic on each edge $e$. Table 4.3 gives an overview of the average Pearson correlation for the different forecasting techniques when using a sampling rate $r = 100$. More complex techniques, such

*Table 4.3: Pearson correlation and standard deviation for different forecasting techniques for a sampling size of $r = 100$.*

| Forecasting Technique | Pearson Correlation ($\rho$) | Standard Deviation ($\delta_\rho$) |
|---|---|---|
| Exponential Weighted MA (EWMA) | 0.932 | 0.036 |
| Holt Winters (HW) | 0.926 | 0.037 |
| Exponential Trend (ET) | 0.909 | 0.042 |
| Auto Regression (AR) | 0.949 | 0.034 |
| Support Vector Regression (SVR) | 0.939 | 0.034 |
| Multi Layer Perceptron (MLP) | 0.966 | 0.035 |




(a) Impact on average estimated MOS

(b) Impact on number of quality switches

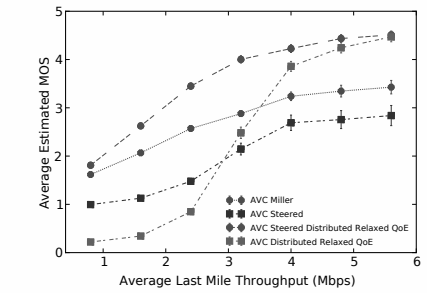
*Figure 4.11: Impact of forecasting method ($RTT = 40ms$, $r = 100$, $|\mathcal{H}_c| = 128$, $\tau = 2s$, $B = 12s$, $N = 32$) for multiple values of the sampling rate $r$.*

as MLP, SVR and AR yield slightly higher correlation then more straightforward estimation techniques, such as EWMA, HW and ET. The latter only require setting the weighting parameters $\alpha$ and/or $\beta$, while the more complex techniques require a training step, yielding more accurate forecasts. Improved estimation of the future available bandwidth for HAS also translates in more accurate QoE optimization. Figure 4.11(a) shows the impact of the sampling size $r$ for the various forecasting techniques on QoE. These results show, that although EWMA, HW and ET have lower correlation values, they still allow an improvement over state-of-the-art adaptation heuristics in terms of QoE. However, when the sampling size $r$ increases, these forecasting techniques are more prone to misestimations, leading to even lower prediction accuracy and eventually lower QoE. Deploying more complex forecasting techniques allows improving overall QoE, even if the sampling size $r$ increases. When comparing MLP to EWMA, better estimations allow an increase of $9\%$ in QoE for a sampling size of $r = 100$ and $28\%$ if the sampling size is further increased up to $r = 10,000$. MLP is even able to outperform EWMA for a higher sampling size $r$: QoE of $4.24$ at $r = 1,000$ for MLP compared to a QoE of $4.14$ at $r = 100$ for EWMA. Even though there is a slight increase in complexity due to the more sophisticated forecasting, MLP is able to reduce the sampling overhead with a factor $10$ compared to EWMA.

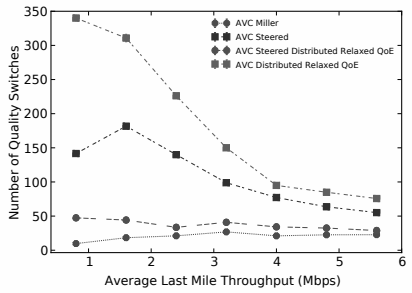
### 4.5.4 Impact of last mile bandwidth fluctuations

In this section, the impact of last mile throughput fluctuations is assessed and how the in-network optimization and client adaptation heuristic interact in such scenarios. To this end, additional cross-traffic is introduced on the last mile connection to simulate the network behavior of mobile devices. Several sets of traces were generated leading to an average available throughput of $0.8Mbps$, $1.6Mbps$, $2.4Mbps$, $3.2Mbps$, $4Mbps$, $4.8Mbps$ and $5.6Mbps$ respectively. Two additional adaptation schemes are also evaluated. For the first scheme, the *AVC Steered* client side adaptation is used, without the in-network control. This comes down to a purely client-driven adaptation scheme that only takes into account the estimated throughput at the client side to select the next quality. In the second scheme (*AVC Distributed Relaxed QoE*), only the in-network adaptation is performed, without taking into account possible throughput fluctuations due to the mobile network. Here, the client downloads the quality that is suggested by the in-network optimization without checking the feasibility in view of the current network conditions. Furthermore, the combination of both schemes is evaluated (*AVC Steered Distributed Relaxed QoE*) together with *Miller*, which yielded the best results in the previous sections. This allows us to compare a purely client-driven adaptation, a purely network-driven adaptation and the combination of both techniques.

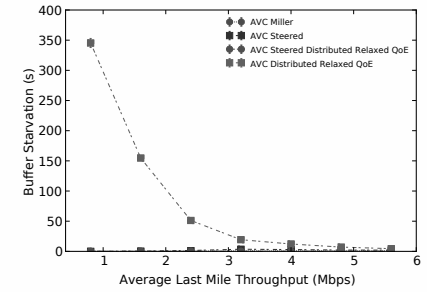
Figure 4.12(a) shows the impact on QoE for the different scenarios. If the average throughput at the last mile is low, the purely network-driven approach is not able to detect this and overestimates the available throughput, negatively impacting the QoE. As the last mile throughput increases, the bottleneck shifts to the shared part of the network and the network-driven adaptation is able to increase the QoE thanks to additional monitoring information and global view of the subscribers. The purely client-driven approach shows a similar course as *Miller*, but about $20\%$ lower due to the basic adaptation heuristic. Figure 4.12(c) shows the impact of the different scenarios on buffer starvations. Since the purely network-driven adaptation constantly overestimates the available throughput, most of the segments arrive late, leading to almost $350s$ of frame freezing, plunging the QoE as shown before. In Figure 4.12(d) the results for the purely network-driven approach are left out for presentation reasons. These results show that the simple client-driven adaptation is not able to cope with the ON-OFF behavior that occurs when multiple HAS clients are connected. Also the adaptation heuristic proposed by *Miller et al.* shows an increasing number of frame freezes, when the last mile throughput increases. This indicates that it is able to cope with throughput fluctuations of the mobile network, but suffers from the congestion that is induced by other connected clients in the network. The combination of in-network and client-driven adaptation is able to overcome both the throughput fluctuations caused by the local network and those caused by competition between connected clients. When comparing the number of switches in Figure 4.12(b), a similar behavior is shown, where the ba-



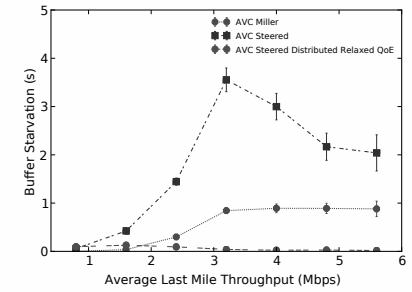

(a) Impact on average estimated MOS




(b) Impact on number of quality switches




(c) Impact on buffer starvations (s)




(d) Impact on buffer starvations (s), leaving out results for the purely network-driven approach

*Figure 4.12: Impact of access network bandwidth fluctuations ($RTT = 40ms$, $|\mathcal{H}_c|= 128$, $\tau = 2s$, $B = 12s$, $N = 32$).*

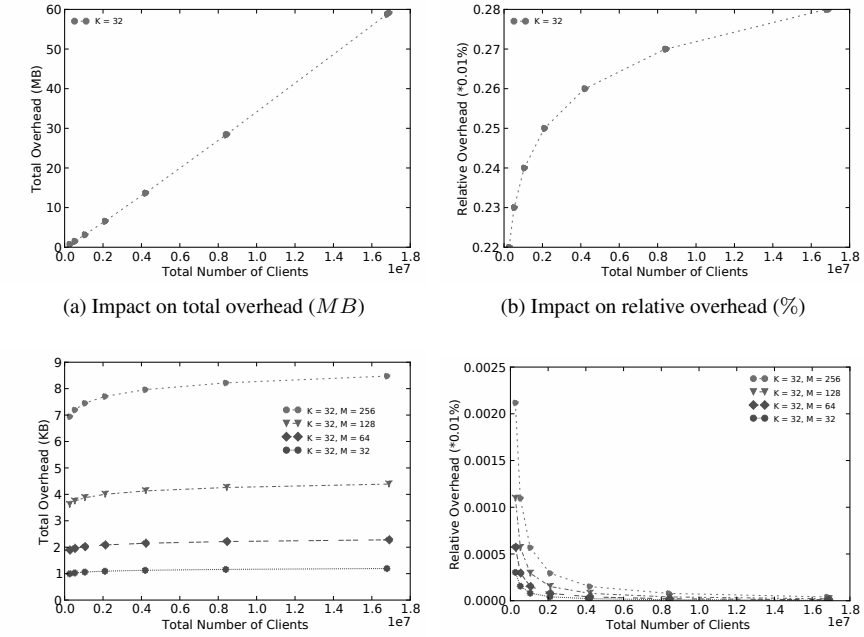
sic client-driven approach is not able to reach a stable quality level and the purely network-based adaptation makes wrong estimations on the available throughput, causing frame freezes, which are interpreted as quality switches. These results show that a combination of in-network and client-side adaptation is required to cope with both the competition of HAS clients and the local network fluctuations, which are not monitored by the in-network adaptation.

### 4.5.5 Overhead of in-network optimization

When deploying in-network QoE-optimization for HAS, several overheads are introduced, ranging from monitoring data transfers to partial solution exchanges. The overheads that are incurred differ between the centralized and distributed approach. To allow a distributed optimization, local subsolutions need to be propagated to their parent nodes. If $|\mathcal{C}|$ is the total number of subscriptions, this list of local solutions, or any combination made by any node, contains at most $|\mathcal{C}|$ entries. To be able to uniquely identify each client, the list should contain an ID composed out of at least $\lceil \log_2 |\mathcal{C}| \rceil$ bits. If the number of available quality rates is $|\mathcal{Q}|$, there are $\lceil \log_2 |\mathcal{Q}| \rceil$ bits required to encode the local decision for each client. Figure 4.13(a) shows how the total network size impacts the communication overhead incurred by distributed optimization. This graph shows a linear relation ($\mathcal{O}(|\mathcal{C}|)$) between the total number of clients (for various configurations of $K$, $M$ and $N$) and the total incurred overhead in $MB$ during each execution of the in-network optimization. Figure 4.13(b) shows the overhead relatively to the total network usage when assuming an average load of $1Mbps$ induced by each connected client. With an impact of around $0.0028\%$ on total network usage, the communication overhead introduced by the distributed optimization can be considered negligible.

The centralized approach does not require the exchange of subsolutions. However, since a global view of the network is required, the throughput measurements performed at several locations in the network need to be forwarded to the node performing the centralized optimization. For each edge $e$, the estimated amount of traffic (in bits) $A_{e,t}$ is forwarded to this node every $\tau$ seconds. Since only one value has to be transferred per link every $\tau$ seconds, the total overhead ($\sum_{e \in \mathcal{E}} \lceil \log_2 A_{e,t} \rceil$) involved with exchanging these measurements is negligible compared to the total streaming traffic. Figure 4.13(c) shows the total overhead in function of the number of connected clients for different values of $M$. This shows the ($\mathcal{O}(\log(|\mathcal{C}|))$) relationship between the measurement data and the total number of connected clients per interval $\tau$ for a specific value of $M$.

Increasing $K$ or $M$ in the topology shown in Figure 4.3, increases the number of links that need to be monitored, while increasing $N$, only increases the number of connected clients, leaving the number of monitored links unchanged. This shows that the structure of the topology has an impact on the communication




(a) Impact on total overhead ($MB$)

(b) Impact on relative overhead (%)

(c) Impact on total overhead ($MB$)

(d) Impact on relative overhead (%)

*Figure 4.13: Impact of total number of clients $|\mathcal{C}|$ for various combinations of network size parameters ($K = 32$, $M \in [32, 64, 128, 256]$ and $N \in [512, 1024, 2048, 4096]$) on the communication overhead for distributed ((a) and (b)) and centralized ((c) and (d)) in-network QoE optimization respectively.*

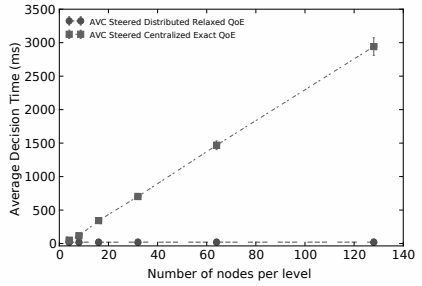
overhead for the centralized optimization. Figure 4.13(d) shows an inverse logarithmic behavior for the relative impact of measurement communication overhead when the number of clients increases. Increasing $N$ leads to more clients and thus higher traffic rates, decreasing the relative impact of the communication overhead. Increasing $K$ or $M$, increases the number of links that need to be monitored, so for the same number of clients $|\mathcal{C}|$, the relative impact is higher when $K$ or $M$ are increased. Increasing the sampling interval $\tau$ can further reduce the impact of the measurement communication overhead, but at the cost of lower prediction precision and hence QoE as was shown in Section 4.5.2.2.

To optimize the QoE, a history of prior decisions $\mathcal{H}_c$ needs to be maintained for each client $c \in \mathcal{C}_n$. In Section 4.5.2.1, it was established that $|\mathcal{H}_c| = 128$ yields the highest QoE. However, maintaining a history of previous decisions for each client also comes at the cost of increased memory requirements. The amount of information that needs to be maintained by each node is in the order of $|\mathcal{C}_n| * |\mathcal{H}_c| * \lceil \log_2 |\mathcal{Q}| \rceil$, where $\lceil \log_2 |\mathcal{Q}| \rceil$ is the number of bits required to uniquely represent $|\mathcal{Q}|$ quality levels. Concretely, for a client set $\mathcal{C}_n$ of $1,000,000$, a history size $|\mathcal{H}_c|$ of 128 and videos with a quality set $|\mathcal{Q}_v|$ of maximum 7 representations, the required storage is $48MB$. Using compression techniques, these storage requirements could be further reduced thanks to repetitive character of the quality values, caused by the avoidance of quality oscillations.

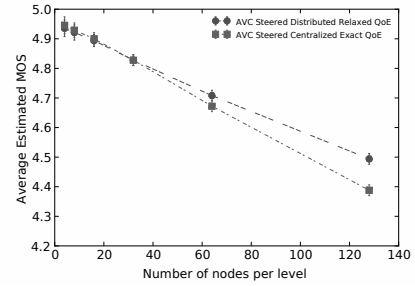
The delay incurred by the network affects the exchange of these partial solutions for the distributed optimization. Since each local optimization requires input of the previous ones, the communication delay incurred by these partial solutions, also affects the execution time of the global optimization as was discussed in previous work [23]. Also the centralized optimization is affected by network delay, since the global optimization requires input of all link states in the network.

### 4.5.6 Scalability of the QoE-driven quality optimization

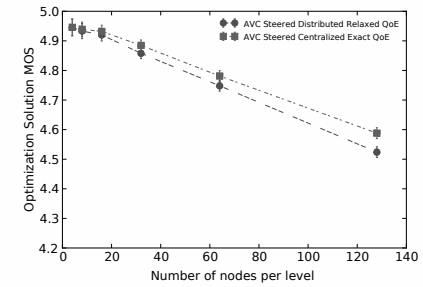
Figure 4.14(a) shows the impact of increasing the number of nodes per level $M$ and the associated link dimensions on the average execution time of the in-network optimization. The network delay impacts both the *Centralized* and *Distributed* approach for the distribution of cross traffic information and quality decisions, respectively. The *Centralized* optimization times linearly increase with the number of nodes per level, since the number of connected clients and edge constraints increase. The *Distributed* optimization experiences only limited impact thanks to the parallel optimization, leading to execution times of about $21ms$, mostly caused by the network delay. Increasing the number of nodes per level does not increase the execution times at that level since the optimizations can be executed in parallel. For small problem sizes, the *Centralized* optimization outperforms the *Distributed* optimization in terms of optimality, thanks to the complete knowledge




(a) Impact on the average decision time (ms)




(b) Impact on average estimated MOS

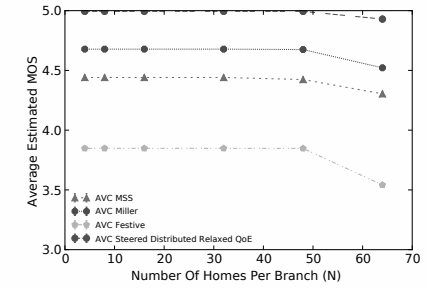



(c) Impact on average optimization solution

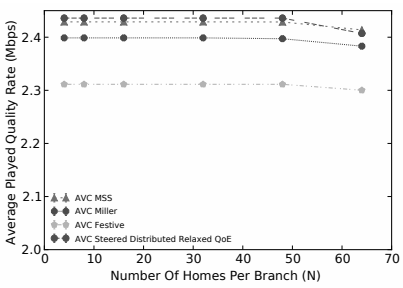
*Figure 4.14: Impact of increasing the number of nodes per level $M$ (RTT $= 40ms$, $r = 100$, $|\mathcal{H}_c| = 128$, $\tau = 2s$, $B = 12s$, $N = 16$). Since the calculation delay for the Centralized optimization is quite high, the practical results differ significantly from the optimal solution, showing the benefits of the more scalable Distributed optimization.*

of the problem. As can be seen from Figure 4.14(b), this optimality is lost when the number of nodes per level $M$ exceeds 32. Due to the longer solution times (up to $2.9s$), suboptimal solutions are installed, leading to occasional buffer starvations and more frequent switching, negatively impacting the overall QoE. In a real-life setting, these execution times heavily impact the optimality of the calculated solution due to delayed availability of the optimal solution. Figure 4.14(c) shows the resulting QoE for the solutions obtained by the in-network optimization. This is the theoretical solution that could be achieved by the optimization in absence of calculation delays and buffering at the client.

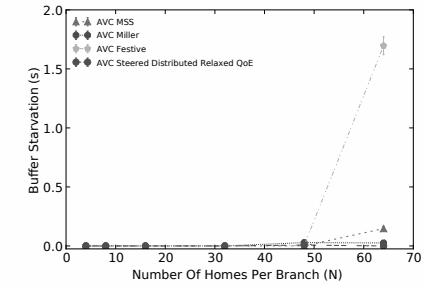
The optimal solution for the *Distributed* relaxation is almost identical to the results obtained in Figure 4.14(b), since there is little impact of the calculation delay. For the *Centralized* optimization however, the results are quite different. Since the calculation delay is quite high, the practical results differ significantly from the optimal solution. The *Centralized* ILP optimization outperforms the *Distributed* relaxation by $1.4\%$ in terms of theoretical optimality, while in a practical setting,




(a) Impact on average estimated MOS




(b) Impact on average quality (Mbps)



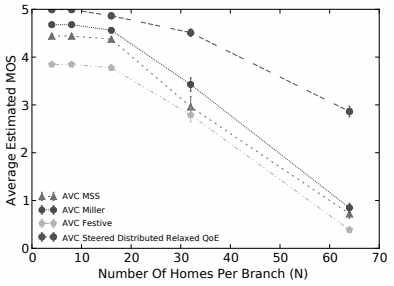

(c) Impact on buffer starvations (s)

*Figure 4.15: Impact of number of clients $|\mathcal{C}|$ for an uncongested scenario ($RTT = 40ms$, $r = 100$, $|\mathcal{H}_c| = 128$, $\tau = 2s$, $B = 12s$). Even in the absence of cross traffic, the client-side heuristics are not able to achieve a comparable quality as in-networks driven optimization due to the competition between clients.*

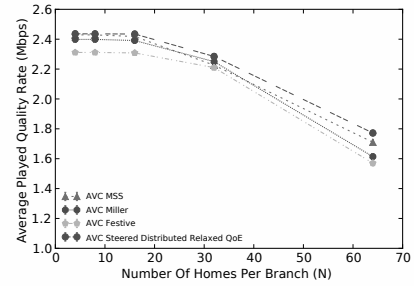
the *Centralized* optimization is outperformed by $2.4\%$.

To measure the impact of increasing the number of homes $N$ per branch, two scenarios were evaluated. In the first scenario, no additional traffic was sent over the links in the network. This ensures that the competition for throughput among clients is the sole source of quality adaptations. The first scenario is referred to as *uncongested*. For the second scenario, additional cross traffic was introduced that competes with the video traffic, causing additional quality adaptations. This additional traffic causes the network to be congested and is referred to as *congested* scenario.

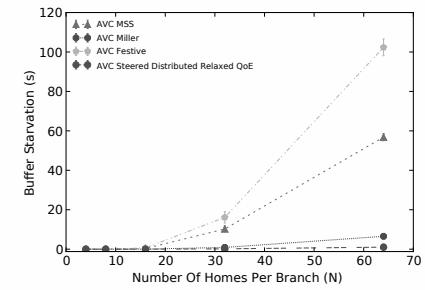
In absence of cross traffic, the QoE-driven optimization achieves an average QoE that is about $7\%$ higher than for *Miller* as illustrated in Figure 4.15(a). This can be attributed to the faster startup quality of in-network based quality optimization, whereas *Miller* gradually increases the quality, leading to quality switches. Increasing the number of connected clients per branch ($N$), introduces congestion by creating additional competition between the different clients. This decreases




(a) Impact on average estimated MOS




(b) Impact on average quality (Mbps)




(c) Impact on buffer starvations (s)

*Figure 4.16: Impact of number of homes $N$ for a congested scenario ($RTT = 40ms$, $r = 100$, $|\mathcal{H}_c| = 128$, $\tau = 2s$, $B = 12s$). The in-network optimization also suffers from the congested network ($N = 64$), but is able to maintain an average QoE that is $130\%$ higher compared to the client-side heuristics and is still acceptable since the MOS is higher than 3.*

the average quality in terms of bitrate as shown in Figure 4.15(b) and can even lead to freezes for the client-side heuristics as shown in Figure 4.15(c). Even in the absence of cross traffic, the client-side heuristics are not able to achieve a comparable quality as in-networks driven optimization due to the competition between clients and over-estimation of available throughput, potentially leading to quality oscillations and buffer starvations.

Increasing the number of connected homes $N$ and introducing additional cross traffic further demonstrates the benefits of in-network quality adaptation. Additional cross traffic from other sources than video traffic leads to wrong estimations for autonomic adaptation heuristics such as *Miller* as is shown in Figure 4.16(a). This causes some of the clients to switch to a higher quality as is shown in Figure 4.16(b), but unlike the wrong estimation predicted, this quality cannot be maintained and ultimately leads to an increasing number of buffer starvations to as is shown in Figure 4.16(c). The in-network optimization also suffers from the con-

gested network ($N = 64$), but is able to maintain an average QoE that is 3 times as high compared to the client-side heuristics and is still acceptable since the MOS is higher than 3. These results indicate the benefits of the additional knowledge for the *AVC Steered* adaptation heuristic provided by the in-network quality adaptation. Measuring the cross traffic along the delivery paths and estimating the achievable quality for each HAS session benefits QoE both in an uncongested scenario ($7\%$ gain) and a heavily congested scenario ($340\%$ gain) compared to traditional adaptation heuristics. This increase can mainly be attributed to a higher quality stability and the avoidance of buffer starvations. Figure 4.16(c) shows how the QoE optimization has almost no freezes in the congested scenario, while for purely client-driven approaches the buffer starvations range from $6s$ to $102s$, seriously degrading the QoE.

## 4.6 Conclusions

In HTTP Adaptive Streaming (HAS), competing clients impact the behavior of each other, causing wrong estimations of the available throughput. This leads to frequent quality oscillations and frame freezes, heavily impacting Quality of Experience (QoE). This chapter therefore proposes a hybrid alternative, where in-network proxies monitor the available throughput using packet-based sampling and estimate the optimal quality selection for each client. The in-network optimization is driven by QoE by maintaining a history of previous decisions and maximizing the QoE in terms of both quality, quality oscillations and buffer starvations. This allows the QoE-driven in-network optimization to outperform standard autonomic quality heuristics by $30\%$ to $43\%$ and bitrate-based in-network optimization by $13\%$ to $19\%$. Both a *Centralized* optimization and a scalable *Distributed* heuristic approach are presented. The *Centralized* optimization allows an optimal solution of the optimization problem, which is about $1.4\%$ higher than the solutions obtained by the *Distributed* optimization. However, due to the longer execution times, the actual output in terms of QoE for the *Centralized* optimization is outperformed by the *Distributed* heuristic when the network size grows. The impact of the historic information as well as the optimization interval were evaluated, showing that a sufficiently large history ($\geq 100$) and optimizing with an interval equal to the average segment length yields significantly better results when compared to an autonomic quality selection heuristic with the same buffer size. Sampling the cross traffic with a sampling size of $100$ yields a high correlation with the actual traffic, while limiting the overhead of sampling $1\%$ of the packets. The proposed solution is able to achieve comparable QoE as a purely client-based quality selection with a buffer that is four times as small.

# References

[1] T. Stockhammer. *Dynamic adaptive streaming over HTTP: standards and design principles*. In Proceedings of the second annual ACM conference on Multimedia systems, MMSys '11, pages 133–144, New York, NY, USA, 2011. ACM.

[2] X. Wang. *Network-Assistance and Server Management in Adaptive Streaming on the Internet*. In Proceedings of the Fourth W3C Web and TV Workshop, 2014.

[3] N. Bouten, S. Latré, W. Van de Meerssche, B. De Vleeschauwer, K. De Schepper, W. Van Leekwijck, and F. De Turck. *A Multicast-Enabled Delivery Framework for QoE Assurance of Over-The-Top Services in Multimedia Access Networks*. Journal of Network and Systems Management, 21(4):677–706, 2013.

[4] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. Tran-Gia. *A Survey on Quality of Experience of HTTP Adaptive Streaming*. Communications Surveys Tutorials, IEEE, PP(99):1–1, 2014.

[5] R. Kuschnig, I. Kofler, and H. Hellwagner. *An Evaluation of TCP-based Rate-control Algorithms for Adaptive Internet Streaming of H.264/SVC*. In Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems, MMSys '10, pages 157–168, 2010.

[6] S. Akhshabi, A. C. Begen, and C. Dovrolis. *An Experimental Evaluation of Rate-adaptation Algorithms in Adaptive Streaming over HTTP*. In Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSys), pages 157–168, 2011.

[7] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. *What Happens when HTTP Adaptive Streaming Players Compete for Bandwidth?* In Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), pages 9–14, 2012.

[8] R. Houdaille and S. Gouache. *Shaping HTTP Adaptive Streams for a Better User Experience*. In Proceedings of the 3rd Multimedia Systems Conference, MMSys '12, pages 1–9, 2012.

[9] S. Benno, J. O. Esteban, and I. Rimac. *Adaptive streaming: The network HAS to help*. Bell Labs Technical Journal, 16(2):101–114, 2011.

[10] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. *Helping Hand or Hidden Hurdle: Proxy-Assisted HTTP-Based Adaptive*

*Streaming Performance*. In Proceedings of the IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), pages 182–191, 2013.

[11] B. Frank, I. Poese, Y. Lin, G. Smaragdakis, A. Feldmann, B. Maggs, J. Rake, S. Uhlig, and R. Weber. *Pushing CDN-ISP Collaboration to the Limit*. SIGCOMM Comput. Commun. Rev., 43(3):34–44, July 2013.

[12] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz. *Adaptation algorithm for adaptive streaming over HTTP*. In Proceedings of the 19th International Packet Video Workshop (PV), pages 173–178. IEEE, 2012.

[13] J. Jiang, V. Sekar, and H. Zhang. *Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE*. In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12, pages 97–108. ACM, 2012.

[14] G. Tian and Y. Liu. *Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming*. In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12, pages 109–120, 2012.

[15] 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 Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 419–426, 2013.

[16] N. Bouten, S. Latré, J. Famaey, F. De Turck, and W. Van Leekwijck. *Minimizing the impact of delay on live SVC-based HTTP adaptive streaming services*. In Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 1399–1404, 2013.

[17] C. Liu, I. Bouazizi, M. M. Hannuksela, and M. Gabbouj. *Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network*. Signal Processing: Image Communication, 27(4):288 – 311, 2012.

[18] V. Adzic, H. Kalva, and B. Furht. *Optimized Adaptive HTTP Streaming for Mobile Devices*. In SPIE Optical Engineering+ Applications, pages 81350T–81350T. International Society for Optics and Photonics, 2011.

[19] D. C. Robinson, Y. Jutras, and V. Craciun. *Subjective Video Quality Assessment of HTTP Adaptive Streaming Technologies*. Bell Labs Technical Journal, 16(4):5–23, 2012.

[20] J. Esteban, S. A. Benno, A. Beck, Y. Guo, V. Hilt, and I. Rimac. *Interactions Between HTTP Adaptive Streaming and TCP*. In Proceedings of the 22Nd International Workshop on Network and Operating System Support for Digital Audio and Video, NOSSDAV '12, pages 21–26, 2012.

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

[22] X. Liu and A. Men. *QoE-aware Traffic Shaping for HTTP Adaptive Streaming*. International Journal of Multimedia & Ubiquitous Engineering, 9(2), 2014.

[23] N. Bouten, S. Latré, J. Famaey, W. Van Leekwijck, and F. De Turck. *In-Network Quality Optimization for Adaptive Video Streaming Services*. Multimedia, IEEE Transactions on, 16(8):2281–2293, Dec 2014.

[24] Z. Li, M. Sbai, Y. Hadjadj-Aoul, A. Gravey, D. Alliez, J. Garnier, G. Madec, G. Simon, and K. Singh. *Network friendly video distribution*. In Network of the Future (NOF), 2012 Third International Conference on the, pages 1–8, Nov 2012.

[25] J. Famaey, S. Latré, R. van Brandenburg, M. O. van Deventer, and F. De Turck. *On the Impact of Redirection on HTTP Adaptive Streaming Services in Federated CDNs*. In Proceedings of the 7th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security Volume 7943, AIMS'13, pages 13–24, 2013.

[26] A. El Essaili, D. Schroeder, D. Staehle, M. Shehada, W. Kellerer, and E. Steinbach. *Quality-of-experience driven adaptive HTTP media delivery*. In Communications (ICC), 2013 IEEE International Conference on, pages 2480–2485, June 2013.

[27] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. *Towards Network-wide QoE Fairness using OpenFlow-assisted Adaptive Video Streaming*. In Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking, pages 15–20. ACM, 2013.

[28] C. Mueller, S. Lederer, C. Timmerer, and H. Hellwagner. *Dynamic Adaptive Streaming over HTTP/2.0*. In Multimedia and Expo (ICME), 2013 IEEE International Conference on, pages 1–6, July 2013.

[29] S. Wei and V. Swaminathan. *Low Latency Live Video Streaming over HTTP 2.0*. In Proceedings of Network and Operating System Support on Digital Audio and Video Workshop, NOSSDAV '14, pages 37:37–37:42, 2014.

[30] S. Wei and V. Swaminathan. *Cost effective video streaming using server push over HTTP 2.0*. In Multimedia Signal Processing (MMSP), 2014 IEEE 16th International Workshop on, pages 1–5, Sept 2014.

[31] N. Bouten, M. Claeys, S. Latre, J. Famaey, W. Van Leekwijck, and F. De Turck. *Deadline-based approach for improving delivery of SVC-based HTTP Adaptive Streaming content*. In Network Operations and Management Symposium (NOMS), 2014 IEEE, pages 1–7, May 2014.

[32] N. Bouten, S. Latré, W. Van de Meerssche, K. De Schepper, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. *An autonomic delivery framework for HTTP Adaptive Streaming in multicast-enabled multimedia access networks*. In Proceedings of the IEEE Network Operations and Management Symposium (NOMS), pages 1248–1253, 2012.

[33] S. Petrangeli, M. Claeys, S. Latre, J. Famaey, and F. De Turck. *A multi-agent Q-Learning-based framework for achieving fairness in HTTP Adaptive Streaming*. In Network Operations and Management Symposium (NOMS), 2014 IEEE, pages 1–9, May 2014.

[34] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang. *QDASH: A QoE-aware DASH System*. In Proceedings of the 3rd Multimedia Systems Conference, MMSys '12, pages 11–22, 2012.

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

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

[37] Y.-M. Hsiao, S.-W. Yeh, J.-S. Chen, and Y.-S. Chu. *A design of bandwidth adaptive multimedia gateway for scalable video coding*. In Proceedings of IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), pages 160–163, 2010.

[38] T. Sutinen, J. Vehkaperä, E. Piri, and M. Uitto. *Towards ubiquitous video services through scalable video coding and cross-layer optimization*. EURASIP Journal on Wireless Communications and Networking, 2012(1), 2012.

[39] H.-C. Lee and S.-M. Guu. *On the Optimal Three-tier Multimedia Streaming Services*. Fuzzy Optimization and Decision Making, 2(1):31–39, 2003.

[40] C. Hsu and M. Hefeeda. *Optimal bit allocation for fine-grained scalable video sequences in distributed streaming environments*. In Electronic Imaging 2007, pages 650402–650402. International Society for Optics and Photonics, 2007.

[41] M. Hefeeda and C.-H. Hsu. *Rate-distortion Optimized Streaming of Fine-grained Scalable Video Sequences*. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 4(1):2:1–2:28, 2008.

[42] N. Bouten, J. Famaey, S. Latré, R. Huysegems, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. *QoE optimization through in-network quality adaptation for HTTP Adaptive Streaming*. In Proceedings of the International Conference on Network and Service Management (CNSM), pages 336–342, 2012.

[43] A. Pras, L. Nieuwenhuis, R. van de Meent, and M. Mandjes. *Dimensioning network links: a new look at equivalent bandwidth*. Network, IEEE, 23(2):5–10, 2009.

[44] R. d. O. Schmidt, R. Sadre, A. Sperotto, and A. Pras. *Lightweight Link Dimensioning using sFlow Sampling*. In Proceedings of the 9th International Conference on Network and Service Management (CNSM), pages 152–155. IEEE, 2013.

[45] R. G. Brown. *Exponential Smoothing for predicting demand*. In Operations Research, volume 5, pages 145–145, 1957.

[46] C. Chatfield. *The Holt-Winters forecasting procedure*. Applied Statistics, 27(3):264–279, 1978.

[47] R. Shibata. *Selection of the order of an autoregressive model by Akaike's information criterion*. Biometrika, 63(1):117–126, 1976.

[48] P. J. Rousseeuw. *Least Median of Squares Regression*. Journal of the American statistical association, 79(388):871–880, 1984.

[49] A. J. Smola and B. Schölkopf. *A tutorial on Support Vector Regression*. Statistics and computing, 14(3):199–222, 2004.

[50] S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy. *Improvements to the SMO algorithm for SVM regression*. Neural Networks, IEEE Transactions on, 11(5):1188–1193, 2000.

[51] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. *Modeling TCP Throughput: A Simple Model and Its Empirical Validation*. In Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures,

and Protocols for Computer Communication (SIGCOMM), pages 303–314, 1998.

[52] J. De Vriendt, D. De Vleeschauwer, and D. Robinson. *Model for estimating QoE of video delivered using HTTP adaptive streaming*. In Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, pages 1288–1293. IEEE, 2013.

[53] J. De Vriendt, D. De Vleeschauwer, and D. C. Robinson. *QoE Model for Video Delivered Over an LTE Network Using HTTP Adaptive Streaming*. Bell Labs Technical Journal, 18(4):45–62, 2014.

[54] 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, April 2014.

[55] L. Plissonneau and E. Biersack. *A Longitudinal View of HTTP Video Streaming Performance*. In Proceedings of the Multimedia Systems Conference (MMSys), pages 203–214, 2012.

[56] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. *Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications*. In Proceedings of the 4th ACM Multimedia Systems Conference, pages 114–118, February 2013.

[57] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck. *Design and Optimisation of a FAQ-learning-based HTTP Adaptive Streaming Client*. Connection Science, 26(1):25–43, January 2014.

# 5

# Clustering-based Quality Selection Heuristics for HTTP Adaptive Streaming over Cache Networks

**N. Bouten, D. De Vleeschauwer, W. Van Leekwijck, S. Latré, F. De Turck.**

**Submitted to International Journal of Network Management, August 2016.**

*Chapter 3 and 4 considered the Quality of Experience (QoE)-management of Video on Demand (VoD) HTTP Adaptive Streaming (HAS) services. The impact of intermediary caches was not considered. However, introducing intermediary caching nodes on the delivery path can impact the QoE perceived by the end user. In cache-assisted HAS, segments can be served from different origins based on the content of the caches, causing highly fluctuating throughput and Round Trip Time (RTT) measurements, negatively impacting the stability and optimality of the quality decisions. In this chapter, heuristics are proposed that are able to use information on the streaming origin and intermediary cache contents to optimize the quality selection process. Furthermore, approximation techniques based on unsupervised incremental clustering are proposed to detect the streaming origin in the absence of an external information channel. Similarly, a probability-based heuristic is proposed to predict the cache content of the expected delivery location when this information is not provided.*

## 5.1 Introduction

Over the past decades, the importance of multimedia services such as video streaming has increased considerably. This growth is projected to exceed 75 percent of the mobile data traffic by 2020, causing video streaming to dominate the Internet [1]. Today, popular Over-The-Top (OTT)-services such as YouTube and Netflix are offering large catalogues of user generated and professionally created video content. In the past, the Real Time Streaming Protocol (RTSP) and Real Time Transport Protocol (RTP) were used to deliver video over IP networks through UDP. Today, the majority of the video streaming traffic is delivered using HTTP over TCP. The popularity of these HTTP-based streaming services was mainly induced by the advantages offered by HTTP streaming: the reuse of caching infrastructure, the reliable transmission over HTTP and the compatibility with firewalls. Furthermore, to increase the scalability of streaming services and to cope with dynamic network conditions, research and academia shifted towards client-side adaptation schemes. Therefore, HTTP Adaptive Streaming (HAS) is now becoming the de facto standard for video streaming delivery.

In HAS, the video content is split temporally into segments which are encoded at different quality rates. The client side heuristic decides at which quality rate each segment should be downloaded, based on measured network statistics, buffer filling level and device characteristics. This allows HAS to respond to throughput fluctuations by reducing the quality and continuing video playout, whereas non-adaptive HTTP-based streaming techniques would run into a buffer starvation. This allows the client to independently choose its playback quality and prevents the need for server-side rate adaptation, which is a major advantage in large-scale OTT scenarios.

The increased popularity of video consumption over the Internet has led to the development of a range of protocols that allow adaptive video streaming over HTTP. Some of the major industrial players have introduced their proprietary protocols such as Microsoft's Silverlight Smooth Streaming[1], Apple's HTTP Live Streaming[2] and Adobe's HTTP Dynamic Streaming[3]. Furthermore, a standardized solution has been proposed by MPEG, called Dynamic Adaptive Streaming over HTTP (DASH) [2].

The goal of the rate adaptation heuristics is to optimize the quality that is perceived by the end-user of the streaming service. For UDP-based streaming services, network impairments such as packet loss can lead to incorrectly decoded frames resulting in blockiness artefacts. For HAS services, a reliable transport protocol is used which, by design, prevents the appearance of video artefacts. How-

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

[2] Apple HTTP Live Streaming - http://tools.ietf.org/html/draft-pantos-http-live-streaming-19

[3] Adobe HTTP Dynamic Streaming - http://www.adobe.com/products/hds-dynamic-streaming.html

ever, due to the fact that the quality can be dynamically adapted, the Quality of Experience (QoE) is impacted by the quality of the selected representation, as well as the switching between different quality representations. Furthermore, since the segment can only be played when it has been received completely and error-free, buffer starvations can lead to video stallings, negatively impacting the user experience [3, 4]. Therefore, the aim of the rate adaptation heuristic is to achieve a fluent video playout at a stable and sufficiently high quality representation with minimal number of switches.

State-of-the-art HAS solutions use a buffer containing several segments to cope with dynamic network conditions. Selection of the quality level of the next segment is done by evaluating the current buffer filling level and the measured throughput. The achieved throughput is impacted by the perceived delay and bottleneck bandwidth of the end-to-end delivery path. Previous research indicated that delay and delay variations can have a significant impact on the quality that can be streamed and eventually on the QoE perceived by the end user [5]. Furthermore, since the bottleneck bandwidth can demonstrate fluctuations over time (either caused by the the channel quality of the access network and varying congestion further upstream in the network) the throughput estimation is based on a weighted history of previous measurements.

One of the advantages of HAS streaming is the reuse of existing HTTP caching infrastructure. Furthermore, with the deployment of Content Delivery Network (CDN) infrastructure (e.g., Akamai) and collaboration programs between Internet Service Providers (ISPs) and content providers (e.g., Netflix OpenConnect), intermediary caches are becoming omnipresent. However, the current adaptation heuristics are not able to cope with the differences in throughput and delay that exist when segments are downloaded from different streaming origins (e.g., content server or intermediary caches). Since the throughput estimations are averaged over time, measurements for segments received from different origins get mixed up, leading to inaccurate estimations of the achievable throughput. If a client is served by an intermediary caching proxy with lower delay and possibly higher throughput, the estimated throughput can be higher than the achievable throughput to the server, resulting in the possibility that a cache miss leads to a buffer starvation.

In the current chapter, the aforementioned problems are mitigated by performing throughput estimations on a per origin basis. In this way, the adaptation heuristic can base its decisions on a more accurate estimation per origin. Furthermore, the adapation heuristic can use information provided by the different origins on the availability of certain segments and quality representations at these locations. A solution is proposed in which the intermediary nodes provide this information using an additional channel or signalling protocol in resemblance to techniques

proposed by Server and Network Assisted DASH[4].

However, if such information is not available, this chapter also proposes techniques to estimate the origin based on delay measurements and to predict the content of intermediary caching nodes. Using unsupervised incremental clustering techniques, the different throughput measurements are clustered based on the measured Round Trip Time (RTT). In this way, the client heuristic is able to differentiate segment deliveries from multiple origins and can make more accurate decisions on the quality representation to request. Furthermore, using a history of origin locations for certain qualities, the proposed approach is able to predict whether the next segment could also be served by the cache. Using the proposed estimation techniques, the approach is able to reduce the number and duration of buffer starvations, while improving the overall quality of the video streaming session. To evaluate the impact of the proposed approach in realistic scenarios, a set of 4G mobile traces was collected and used during the experiments.

The remainder of this chapter is structured as follows. Section 5.2 gives an overview of relevant research in the field of client-based and network-assisted HAS techniques. The proposed heuristics using the additional origin and caching information are presented in Section 5.3. Section 5.4 and Section 5.5 discuss the clustering-based origin detection and cached content prediction techniques respectively. The evaluation framework and experiments are presented in Section 5.6. Section 5.7 concludes the chapter and summarizes the most important findings.

## 5.2 Related Work

At the client side, each commercial HAS implementation comes with a proprietary client heuristic as discussed in Section 5.1. Several heuristics have been proposed in literature as well, each focussing on a specific deployment. *Miller et al.* propose a receiver-driven adaptation heuristic for DASH that takes into account a history of available throughput and the buffer level [6]. The quality is adjusted to attain a buffer level between certain target thresholds, this improves the stability of the quality and avoids frequent switching as a consequence of short-term throughput variations. *Jiang et al.* identified the problems that arise when multiple clients share a link [7]. The authors propose a variety of techniques that can help avoid such undesirable behavior, such as harmonic bandwidth estimation, stateful and delayed bitrate update and randomized scheduling of requests, which are grouped in the FESTIVE adaptation algorithm. *Tian et al.* show that there is a trade-off between responsiveness and smoothness for client-side DASH adaptations [8]. The proposed rate-switching logic provides a dynamic control of this trade-off according to the trend of the buffer growth. The approach uses machine-learning based

[4]SAND - https://tools.ietf.org/html/draft-begen-webpush-dash-reqs-00

TCP throughput prediction to support multiple servers simultaneously. In previous work [9] [5], the authors evaluated different client heuristics both for Advanced Video Coding (AVC) and Scalable Video Coding (SVC), applying optimizations such as pipelined and parallel download scheduling. The approach presented in this chapter is applicable to both AVC and SVC. *Liu et al.* discuss a video client heuristic that is suited for a CDN by comparing the expected segment fetch time with the experienced segment fetch time to ensure a response to bandwidth fluctuations in the network [10], while *Adzic et al.* present a client heuristic which is tailored for mobile environments [11].

Among others [12, 13], *Akhshabi et al.* compare several commercial and open source HAS players and indicate significant inefficiencies in each of them, such as frequent oscillations and unfairness when the number of competing clients increases [14, 15]. Those quality oscillations are known to have a negative impact on QoE [16] and cause inefficient resource utilization within the bottleneck network [15, 17]. In a recent survey, *Seufert et al.* argue that a centralized control unit or client-proxy based communication can enhance the quality and establish a fair QoE distribution amongst competing clients [18]. The present chapter aims at improving the QoE by providing the client adaptation heuristic with additional in-network information or by estimating it.

By altering the behavior of the streaming server, stability and bandwidth efficiency can be increased. *Akhshabi et al.* propose server-side rate adaptation to cope with unstable streaming players due to ON-OFF patterns when they compete for bandwidth [19]. The system detects sudden rate fluctuations in the client playout and tries to solve them by shaping the sending rate at the server to resemble the bitrate of the stream. *Liu et al.* follow a comparable approach where the rate is shaped according to QoE maximization metrics [20]. These systems are able to restore the streaming session when oscillation or freezing occurs and then remove the shaping when the client has stabilized. The proposed approach is not only able to solve the problems of oscillation or freezes when they occur, but actively tries to prevent them, while at the same time optimizing the QoE. In previous work, it was shown that, although the proposed server-side rate shaping can increase stability, these techniques are not able to achieve the stability offered by in-network quality optimization due to a reactive, rather than proactive behavior [21]. Furthermore, the proposed approach is also able to handle multiple intermediate caching nodes.

*Li et al.* propose a collaboration scheme between CDNs and ISPs and peer-assisted CDNs to reduce the load on both peering links and internal ISP links [22]. Distributed CDN servers alter the manifests to associate chunks with regional storage servers or by changing or increasing the available video quality levels by transcoding the video. The approach proposed in this chapter could be complementary to the distributed CDN case, where intermediary storage servers exist and client sessions consuming their content are terminated in these nodes. The current

approach further increases the efficient use of the ISP's network by minimizing the negative impact of competing sessions on QoE. *Famaey et al.* assess the impact of increased latency on QoE caused by the redirection of HAS requests in CDNs and propose updated request routing schemes in order to reduce the number of redirects [23].

Network level adaptations allow a more efficient use of the underlying network resources for HAS. *Essaili et al.* propose a TCP rate shaping mechanism on a per flow level to enhance the QoE by rewriting client requests and offering control to the network operator which has better information on the load and radio conditions in the cell [24]. *Houdaille et al.* propose to use traffic shaping in the residential gateway to implement bandwidth arbitration between competing clients [13]. Others propose to exploit the advantages of Software Defined Networking (SDN) in a HAS environment to monitor the streaming sessions and, in conjunction with a client control plane, dynamically adjust video flow characteristics to ensure QoE [25]. Recently, *Mueller et al.* proposed to use HTTP/2.0 when deploying adaptive streaming and evaluate the overhead and impact on link utilization when using HTTP/2.0 for HAS [26]. Using new features in HTTP/2.0, such as server push, persistent connections and pipelining, HAS services can be improved. *Wei et al.* show that the increasing protocol overhead that is caused by decreasing the segment length can be overcome by automatically pushing a number of segments, allowing them to reduce the live latency [27, 28].

In previous work, a Differentiated Services (DiffServ) approach was proposed to guarantee the delivery of certain segments in a live streaming scenario [29]. This requires altering the relevant priorization fields for these specific segments and implementing DiffServ routers in the ISP network. An autonomic delivery framework for HAS-based Live TV and Time Shifted TV (TSTV) was presented in previous work [30, 31] which allows to reduce the consumed bandwidth by grouping unicast HAS sessions sharing the same content into a single multicast session. However, for Video on Demand (VoD) HAS sessions, the content is more diverse and only few sessions are potentially shared among multiple users. This prevents them to be grouped into a shared multicast session and therefore prevents them from being delivered in a scalable manner. *Petrangeli et al.* use in-network proxies to determine the overall median chunk level requested by the clients and disseminates this information towards each client [32].

The effects of intermediary caching proxies on HAS have received limited attention in the research community. *Mueller et al.* first discussed the negative impact intermediary caches can have on QoE in HAS [33]. The autonomous quality adaptation in the clients can lead to uncontrolled distributions of the different quality representations in the caches which can falsify the throughput estimations at the clients. *Krishnamoorthi et al.* evaluate the effects of intermediary proxies and propose a number of prefetching strategies to aid HAS streaming when there

is limited bandwidth available on the path between the server and the proxies [34]. By prefetching segments that will be likely requested in the future, the bottleneck bandwidth can be better utilized and yield substantial benefits. However, prefetching too many segments clogs the server proxy bottleneck and increases the stalling time. To aid the prefetching strategies, a client-proxy cooperation is proposed to align the quality selections with the prefetching. To avoid the impact of bitrate oscillations caused by intermediary caching proxies, *Lee et al.* propose to shape the outgoing download rate for the clients to avoid quality oscillations due to wrong estimations on the available throughput [35]. In contrast to the proposed prefetching method, next to the client-proxy cooperation, the present chapter also proposes methods to allow the client to autonomously detect the streaming origin and estimate the cache contents to avoid the impact of throughput oscillations. The discussed prefetching method could further improve the proposed approach by prefetching one segment ahead if required. Shaping the outgoing rate at the server yields suboptimal quality decisions, since the client will request quality representations that are supported by the bottleneck bandwidth between the server and the proxy. However, if a large part of the video is cached at higher quality at the intermediary proxy, the client is still able to stream at higher quality while avoiding quality oscillations. The proposed approach does not utilize shaping, but avoids quality oscillations by detecting the streaming origin and estimating the cache contents.

To determine the streaming origin in absence of an external information channel, a clustering-based approximation technique is proposed based on the measured RTT towards the streaming origin. *Zhang et al.* propose an RTT-based mechanism to detect web proxies and take advantage of HTTPS connections to retrieve a page of which the delivery is not subject o proxy interposition [36]. Also in other application domains RTT measurements are used to detect intermediary nodes. For example, *Trabelsi et al.* propose a detection mechanism for sniffers using differences in RTT measurements [37], while *Tun et al.* propose a RTT-based system to detect wormhole attacks [38]. Next to RTT measurements, other techniques can be used to determine the presence of intermediary caching nodes. *Xu et al.* try to take advantage of the fact that many proxies manipulate the header fields by creating fingerprints based on the header fields to distinguish between origins [39]. *Weaver et al.* discuss a variety of proxy detection techniques among which are HTTP 404 fetches, leveraging the fact that most proxy vendors have custom HTTP 404 messages [40]. Tracebox is an extension to traceroute that is capble of detecting various types of middleboxes over almost any path by sending TCP segments with different TTL values and analyzing the packet encapsulated in the returned ICMP messages [41]. This chapter relies on RTT measurements for the estimation of the streaming origin since this information can be easily gathered without requiring access to lower level packet traces.

```
state ← BUFFERING
while SegmentsToDownload do
    if state ≡ BUFFERING then
        if BufferDecreasing ∨ BufferSlowlyChanging then
            Limit change to 1 quality level
        end if
        q ← Calculate Quality According to Bandwidth
        if B ≥ B_upper + B_lower/2 then
            state ← STEADY
        end if
    else if state ≡ STEADY then
        if B < B_panic then
            q ← 0
            state ← BUFFERING
        else if BufferSlowlyChanging then
            if B < B_lower then
                q ← q − 1
            else if B > B_upper then
                AttemptIncreasingQuality()
            end if
        else if BufferDecreasing ∧ B < B_lower then
            q ← 0
            state ← BUFFERING
        else
            AttemptIncreasingQuality()
        end if
    end if
    Download at quality q
end while
```

*Algorithm 5.1: Basic quality selection heuristic.*

## 5.3 Heuristic Description

In this section an overview of the various selection heuristics is presented. First, a basic quality selection heuristic based on the AVC Microsoft Smooth Streaming (MSS) quality adaptation heuristic is presented. Second, an extension is made to the aforementioned heuristic to include cache-awareness. This entails that the heuristic is made aware of the streaming origin (e.g., intermediary caches or content server) which allows the heuristic to maintain a separate throughput estimator per origin. Finally, a cache-assisted heuristic is proposed, which is able to include additional information on the upstream cache contents during the quality selection.

```
    if TimeSinceLastImprove > ImproveTimeout then
        n ← q + 1
        while BR_n < R_s do
            TimeSinceLastImprove ← 0
5:          q ← n
            n ← n + 1
        end while
    end if
```

*Algorithm 5.2: AttemptIncreasingQuality().*

### 5.3.1 Basic quality selection heuristic

The basic quality selection heuristic, which serves as a starting point, does not take into account additional information that can be deduced or received from the network elements. It is based on the Microsoft's Smooth Streaming heuristic of which an overview is shown in Algorithm 5.1 [9]. The heuristic continuously evaluates the status of the play-out buffer and makes its decision based on the comparison of the buffer's state with 3 thresholds: a lower threshold $B_{lower}$, an upper threshold $B_{upper}$ and a panic threshold $B_{panic}$. The goal of the heuristic is to maintain a steady state of the play-out buffer and download the highest quality possible. Every time a new segment is downloaded, the buffer's state is evaluated to decide on the next segment. The heuristic starts off in *buffering* state. This means that it follows a more aggressive way of increasing the quality. It downloads the highest possible quality according to the performed throughput measurements (line 7). This behavior is overridden if the buffer is showing signs of decrease or slow changes. In that case, the quality is only allowed to increase or decrease with a single step (lines 4-6). This is done to avoid large oscillations in the decision of the heuristic: continuous fluctuations in the experienced quality of a video is known to be annoying to the user. The buffering state continues until the buffer is sufficiently filled (lines 8-10). In that case, the heuristic moves into a *steady state*.

In the *steady state*, the change in quality is more conservative: the quality is only increased or decreased with one level at a time or increased with multiple levels if the buffer is sufficiently full and an improvement timer has timed out. To determine whether to change the quality, the heuristic analyses the buffer state, how the state evolves over time and the download time history of the previous segments. If the buffer drops below the panic threshold $B_{panic}$, the lowest quality is downloaded, and the algorithm goes back into the *buffering* state (lines 12-14). This panic mode is used to avoid the occurrence of play-out buffer starvations due to a sudden buffer underrun.

If the buffer state is sufficiently high (i.e. higher than $B_{panic}$) and no anomalies have occurred, the heuristic evaluates the evolution of the buffer. If it is slowly changing, the heuristic intervenes by reducing the quality level of the next segment

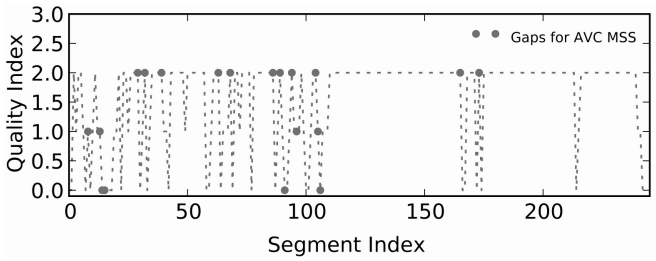



*Figure 5.1: Example run of impact of varying streaming origin on QoE achieved by base heuristic [42].*

when buffer levels are decreasing (lines 16-17) or increasing the quality level otherwise (lines 18-19). Note that, in the latter case, the quality is only increased if bandwidth measurements indicate that there is enough bandwidth to download the next segment at a higher quality as shown in Algorithm 5.2. If the buffer evolution is more rapid, the heuristic switches to a panic mode when the buffer is lower than $B_{lower}$. The result is that the quality of the next segment is set to the lowest possible and the heuristic returns to the *buffering* state (lines 21-23). Note that the algorithm compares with the lower threshold $B_{lower}$ and not with the panic threshold $B_{panic}$ as it detects a rapid change in the buffer state. If the buffer evolution is not slowly changing and not decreasing, a quality increase is again attempted (lines 25). In cases not identified, the quality of the next segment is not modified. Once the quality has been determined, the segment is downloaded (line 28) and the whole procedure is repeated when this segment is received.

### 5.3.2 Cache-aware quality selection heuristic

Figure 5.1 shows an example run performed in previous work [42] where segments are delivered from different streaming origins. The base adaptation heuristic presented in the previous section was used to perform this experiment. The $2s$ segments were encoded at $1Mbps$, $1.4Mbps$ and $2Mbps$, delivered over links ranging from $0.5Mbps$ to $5Mbps$ using a client buffer size of $12s$ and a video duration of $500s$. The dots in the graph are the buffer starvations that occur at the client side. In total, 19 starvation events occurred and the total starvation time was $48.6s$.

As demonstrated in the example, applying the basic quality selection heuristic in a network where caches are present can lead to a number of problems. A simple throughput estimation mechanism which uses the download times and segment sizes to calculate a weighted average is not able to cope with the differences in throughput and delay that are observed when the segments are delivered from different locations. These inaccurate estimations can cause wrong decisions when trying to improve the quality and, even worse, when calculating the achievable quality when the heuristic is in the *buffering* state. Ultimately, the erroneous deci-

sions will lead to buffer starvations, detoriating the QoE. Therefore, this chapter proposes to either deduce the origin location of the segments by inspecting the measured RTT, or by incorporating additional information that was received from network elements along the delivery path.

When the streaming application is able to detect whether segments were delivered from a cache or not, multiple throughput estimation instances can be created for the various delivery locations in the network. In this way, the achievable throughput and delay towards every delivery location can be estimated more accurately, thus increasing the precision of the quality selection heuristic. As proposed in previous research [43], an Exponentially Weighted Moving Average (EWMA) estimator (Equation (5.1)) can be used to calculate the estimated throughput based on the previous estimation and the last perceived throughput, weighed by a factor $\alpha$.

$$BW_t = \alpha * BW_{meas} + (1 - \alpha) * BW_{t-1} \tag{5.1}$$

However, when multiple consecutive segments are served from the same location, the estimations for the other locations are no longer reliable since they reflect the network characteristics in the past. Therefore, it is proposed to replace the simple weight factor $\alpha$ by an ageing factor $\alpha(t_n, t_{n-1})$ which depends on the time that has passed between the current and the previous measurement as shown in Equation (5.2), where $t_n$ is the instant that the $n^{th}$ segment was downloaded from the considered location (i.e., intermediary caches or content server) and $\tau$ a tuneable parameter. This estimation is referred to as A-EWMA.

$$\alpha(t_n, t_{n-1}) = 1 - e^{-\left(\frac{t_n - t_{n-1}}{\tau}\right)} \tag{5.2}$$

Since the streaming application has no knowledge whether the next segment will be served from a cache, the selection heuristic uses the estimation for the end-to-end throughput to the server during the calculations. This reduces the risk of running into buffer starvations, but comes at the cost of a more conservative quality adaptation when segments are served from a cache. In that case the achievable throughput might be higher and would allow a higher quality to be selected.

### 5.3.3 Cache-assisted quality selection heuristic

In the previous section, it was assumed that no additional knowledge of the cached content was available. This additional knowledge could however be provided to the client heuristic by the network nodes. The client heuristic is thus able to combine the accurate throughput estimations per location with cache information when deciding which quality to download next.

Figure 5.2 illustrates the different states and state transitions of the adapted client heuristic. An additional state called *steady state cached* is added. When the

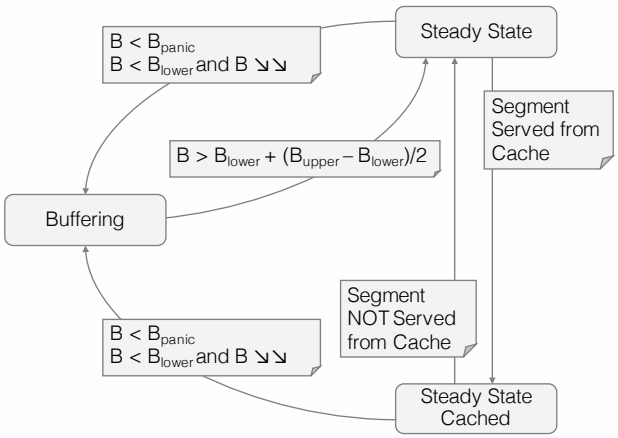



*Figure 5.2: State diagram for cache assisted quality selection heuristic.*

heuristic is in *steady state* and a segment was served by a cache, the heuristic enters the *steady state cached*. Conversely, when a segment was not served from cache although the heuristic is in *steady state cached*, the heuristic returns to the *steady state*. The *steady state cached* cannot be reached directly from the *buffering* state, however, the same state transition conditions hold for returning to the *buffering* state.

Algorithm 5.3 shows the additional selection intelligence that is added to the heuristic discussed in Algorithm 5.1. If the buffer filling level is beneath $B_{lower}$ and the buffer filling is quickly decreasing, then the heuristic selects the lowest quality and switches to the *buffering state*. When the buffer filling level is below $B_{upper}$, the buffer filling is at its target state, it is checked whether the current quality level can be maintained. If the next segment is cached in quality $q$ ($C_q = true$) and the throughput of the connection between the client and the cache ($R_c$) exceeds the required bitrate ($BR_q$) of that quality level, the quality is selected. Otherwise, if the segment is not cached in that quality, but the rate of the connection with the server ($R_s$) allows streaming at rate $BR_q$, the quality is selected as well. If not, the quality is decreased with one step until one of the conditions holds or the lowest quality level is selected. If the buffer filling is greater than $B_{upper}$, the buffer is sufficiently filled and the adaptation heuristic can try to improve the streaming quality. To avoid frequent quality switching, the time since the last quality improvement should be larger than a certain improve timeout. If this is the case, the heuristic tries to improve the quality based on the knowledge of the content of the cache and the achievable throughputs to the server and cache. If the cache contains a higher quality level and the rate $R_c$ is able to support streaming this quality, the quality is improved to that level (lines 19-21). If the quality is not available in the

```
if state ≡ STEADYSTATECACHED then
    if B < B_panic then
        q ← 0
        state ← BUFFERING
    else if B < B_lower ∧ BufferQuicklyDecreasing then
        q ← 0
        state ← BUFFERING
    else if B < B_upper then
        while q > 0 do
            if (C_q ∧ BR_q < R_c) ∨ (BR_q < R_c ∧ BR_q < R_s) then
                break
            end if
            q ← q − 1
        end while
    else
        if TimeSinceLastImprove > ImproveTimeout then
            n ← q + 1
            while BR_n < R_c do
                if C_n ∨ R_s > BR_n then
                    q ← n
                end if
                n ← n + 1
            end while
        end if
    end if
end if
```

*Algorithm 5.3: Cache-assisted quality selection heuristic.*

cache, but the rate $R_s$ is able to support the quality, the quality is improved as well.

## 5.4 Detection of Streaming Origin

The streaming origin could be communicated to the adaptation heuristic using an additional communication channel or by adding header fields which identify the origin. However, in case there is no support for the in-network or out-of-band signalling of such information, caches are managed by multiple independent instances or some of the caches are not supporting this, the client should be able to autonomously detect the streaming origin. Each time a segment is received, several characteristics of the download can be derived such as an estimate of the RTT, the achieved throughput, etc. These can be used as inputs to cluster the segment downloads based on their estimated origin. The problem of clustering segment downloads according to their origin has several characteristics. First, the data to cluster only come available after a segment is downloaded and can thus be considered as a data stream. Second, there is no priori knowledge of the number of

$T = (t_1, ..., t_k) \leftarrow (x_1, ..., x_k)$
$n_1, ..., n_k = 1$
**repeat**
  $x \leftarrow (RTT, Throughput)$
  **if** $(t_i - x) = min(t_i - x : \forall i \in [1, k])$ **then**
    $n_i \leftarrow n_i + 1$
    $t_i \leftarrow t_i + (1/n_i)(x - t_i)$
  **end if**
**until** session closes

*Algorithm 5.4: Sequential k-means.*

streaming origins in the network. Third, network characteristics can change over time, impacting both the clusters themselves as well as the criteria to determine if two clusters are similar/dissimilar.

Since the data is received as a stream and instant classification is required, an incremental clustering algorithm is best suited. One of the most well-known incremental clustering algorithms is probably sequential k-means, which is an incremental variant of Lloyd's algorithm [44]. Algorithm 5.4 shows how the k-means clustering algorithm starts off by setting the cluster centers of the $k$ clusters to the first $k$ samples. After this, each of the samples is added to the cluster $i$ to which the distance to the center $t_i$ is minimal. Afterwards the cluster center $t_i$ is updated to reflect the new clustering.

One of the assumptions required for incremental k-means is the knowledge of the number of clusters $k$. However, since the streaming application has no a priori knowledge of the number of streaming origins in the network, this condition does not hold. Therefore, a dynamic incremental k-means algorithm should be used in which the number of clusters is allowed to change [45]. Clusters with too few elements might be eliminated, clusters of which the distance between the centers is small might be merged and clusters with large dispersion might be split. A possible method for changing the number of clusters can be based on a predetermined quality threshold on the distortion. The ISODATA algorithm allows the number of clusters to be adjusted automatically based on pre-specified parameters: $K_0$ the desired number of clusters, $n_{min}$ the minimum number of samples in each cluster (for discarding clusters), $\sigma^2_{max}$ the maximum variance (for splitting clusters) and $d_{min}$ the minimum pairwise distance (for merging clusters) [46]. Algorithm 5.5 shows the heuristic for discarding, splitting and merging clusters adapted to the incremental k-means clustering procedure.

Network conditions can change over time due to user mobility, flash crowds or dynamic changes in the streaming topology (e.g., deployment of additional intermediary caches). To cope with gradual shifting of cluster centers due to increases and decreases of access network delays, a weight factor $\omega$, based on measurement age is introduced when calculating the cluster centers. The weighing factor $\omega_j$ for

```
Execute sequential k-means
for all c_i do
    if n_i < n_min then
        Discard cluster c_i
        Reassign members of c_i to other clusters
        k ← k − 1
    end if
end for
for all c_i do
    Update centers t_i = 1/n_i Σ_{x∈c_i} x
    Update variance σ_i^2 = 1/n_i Σ_{x∈c_i} (x − t_i)^2
end for
if K ≤ K_0/2 then
    for all c_i do
        if σ_i^2 > σ_max^2 ∧ n_i > 2n_min then
            Split into new clusters and reassign
            t_i^+ ← t_i + σ_i
            t_i^− ← t_i − σ_i
            k ← k + 1
        end if
    end for
else if K > 2K_0 then
    d_{i,j} ← (t_i − t_j)^2
    if d_{i,j} < d_min then
        Merge cluster c_i and c_j into c_i
        t_i ← (n_i t_i + n_j t_j)/(n_i + n_j)
        k ← k − 1
    end if
end if
```

*Algorithm 5.5: ISODATA algorithm.*

each element $j$ in the cluster is calculated as $\omega_j = \frac{\alpha^{t-t_j}}{\sum_i \alpha^{t-t_i}}$, decreasing the impact of older measurements on the center calculation.

## 5.5 Estimation of Cache Content

In Section 5.3, a cache-assisted quality selection heuristic was introduced which uses information on the cache content to support the quality decisions. Similar to the section on the cache-aware quality selection heuristic, this information could be transferred using additional communication channels. However, in case there is no such information channel available, the client can still try to estimate whether the next segment will be in a cache or not. Using the origin detection mechanisms introduced in the previous section, the client is able to determine whether the previous segments were downloaded from the same origin.

The first decision based on the cache content information in the cache-assisted heuristic is taken when switching between the *Steady State* and *Steady State Cached*. Detecting when a segment was not served from the cache can be done by comparing the current origin with the origin of the previous segment. When the RTT associated with the new segment is substantially higher than the RTT of previous downloads, the segment was not served from the cache. Hence the *Steady State Cached* state is exited. On the other hand, when a segment is served from a location that is closer (i.e., its RTT is smaller) to the one from which the previous segment was served, the *Steady State Cached* is entered. Thus, the approach is able to achieve similar behavior as when the cache state information is transferred to the clients.

The second time the caching information is used is when evaluating the conditions to switch to a higher quality representation. To this end, it is verified whether the higher quality representation is available in the cache and the connection towards the cache is able to serve this segment with sufficient throughput. When this is not the case, the connection towards the server is also considered. Since now the information on cached quality representations is not available, an estimation has to be made. If a number of previous segments $n$ was served from the same origin at the same quality level $q$, the probability is high that the next segment will also be served from this origin if the same quality representation is requested. If the segment is requested in a higher quality, there is less evidence supporting the argument that the next segment could be served from the cache as well. However, since the goal is to improve the quality, this decision should be taken once in a while to avoid getting stuck in a local optimum. Therefore, when sufficient segments $n > n_{min}$ were served from the cache at the previous quality $q$, the heuristic will randomly estimate wether the next segment will be served from the same origin in a higher quality. A discrete probability distribution is used with the following probability mass function:

$$Prob(notcached) = 1 - \frac{e^{B\beta}}{e^{\beta}} \tag{5.3}$$

$$Prob(cached) = \frac{e^{B\beta}}{e^{\beta}} \tag{5.4}$$

where $B$ is the current buffer filling and $\beta$ is a parameter to control the balance between exploitation and exploration. This allows the heuristic to improve the selected quality but without the risk of running into a buffer starvation. If $\beta$ is small, there is a higher probability that the segment is estimated to be served from a cache. If $\beta$ is large, the safer decision is favored and the knowledge that the previous $n$ segments were served at quality $q$ from the same location is exploited.

## 5.6 Evaluation Results

This section discusses the experiment framework that was extended to support the different heuristics that are under evaluation. First, the possible improvement of the proposed approach with perfect knowledge of the streaming origin and cache contents is evaluated. Next, the various parameters of the heuristics for detecting the origin and estimating the cache content are optimized. Subsequently, the improvement subject to the estimated knowledge is quantified. Finally, different scenarios, subject to variable network conditions are evaluated.

### 5.6.1 Experiment framework

A VoD HAS scenario was implemented using the discrete-event network simulator NS3[5], simulating the transmission of HAS-based video [47]. For the autonomic HAS Clients, several heuristics found in literature were implemented. A first implementation uses the *MSS* algorithm, which is based on the implementation of an open source version of the algorithm of the *MSS* video player[6] and is extensively described by *Famaey et al.* [9]. A second implementation is based on the heuristic proposed by *Miller et al.* [6], which is a receiver-driven adaptation algorithm based on buffer filling level and throughput estimations. This heuristic is referred to as *Miller*. A third heuristic, called *Festive*, is based on the implementation described by *Jiang et al.* using randomized scheduling and stateful bitrate selection [7]. The parameters of the aforementioned heuristics were optimized to attain the best QoE for a variety of scenarios. The QoE is evaluated by calculating the Mean Opinion Score (MOS) as defined by *Claeys et al.* [48].

Next to the state-of-the-art adaptation heuristics, also the proposed heuristics discussed in Section 5.3 were implemented in the ns-3 simulator. These heuristics are referred to as *MSS (Distinguish Origin)* and *MSS (Caching Info)*. Furthermore, to optimize the different parameters of the unsupervised incremental clustering algorithm offline, a python-based simulation framework was built. This simulation takes as input a trace generated by an ns-3 simulation containing a list of downloads, the corresponding estimated RTT and the actual origin location of the download. This file is then used to perform the clustering algorithm and evaluate the accuracy of the proposed approach under different parameter configurations. The ns-3 simulation platform was then extended to incorporate the clustering-based origin detection (*MSS (Clustered Origin)* as described in Section 5.4) and cache content estimation (*MSS (Cache Estimation)* as described in Section 5.5) as well.

A Least Recently Used (LRU)-cache application was implemented in the ns-3 simulator, which is extended with the functionality to respond to segment avail-

[5] ns-3 - https://www.nsnam.org

[6] Source available from https://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming

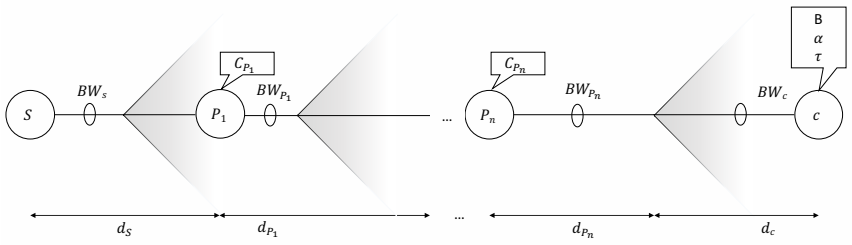



*Figure 5.3: Experimental setup, representing a delivery network with intermediary cache nodes.*

ability queries by the client applications, as well as support for inserting the origin ID into the HAS header. The cache size is a configurable parameter and sizes are based on the content catalogue that is available at the server. For the simulations, a set of 20 videos was used of which the 7 quality representations range between $300kbps$ and $2436kbps$ and contain 200 segments with a $2s$ duration. This leads to a total content catalogue size of about $7.5GB$. For cache sizes of $100MB$, $35\%$ of the videos can be stored in lowest quality, while around $1.3\%$ of the content catalogue can be stored in all qualities. This is comparable to storage capacities reported for VoD services such as Netflix [49]. The distribution of viewers over the content catalogue is set according to a Zipf distribution with parameter $\alpha$ set to $0.81$ [50]. Clients are started randomly using a Weibull startup process with shape of $5$ and mean of $250s$.

Figure 5.3 shows an example of the topology that is used during the experiments. It interconnects the clients $c$, using a hierarchical set of caching proxies $P_1...P_n$, each having their respective cache sizes $C_{P_1}...C_{P_n}$ to a streaming server $S$. The client has several values that can be configured: the maximum buffer size (in seconds) $B$, the EWMA-parameter $\alpha \in [0, 1]$ and the A-EWMA parameter $\tau \in [0, \infty]$.

For the single-cache level experiments the number of cache levels is $n = 1$ and the index for the proxy level is omitted. $4$ proxies are connected to the server and $50$ client applications are connected to each of the caching proxies. The bandwidth and delay between server and proxy is indicated as $BW_S$ and $d_S$, respectively. The size of the cache is indicated by $C_P$, while the bandwidth and delay to the next router are denoted by $BW_P$ and $d_P$. Similarly, $BW_c$ and $d_c$ are used for the client $c$ connection to the router. Unless otherwise stated, the following parameters are used: $BW_S = 250Mbps$, $d_S = 40ms$, $BW_P = 250Mbps$, $d_P = 5ms$, $BW_c = 10Mbps$, $d_c = 5ms$, $C_P = 50MB$ and $B = 20s$.

For the dynamic experiments the number of cache levels is $n = 2$. To assess


*Figure 5.4: Map showing different streaming locations of the dynamic scenario.*

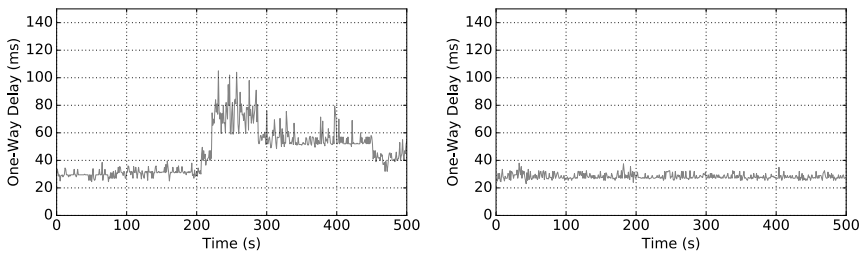
the viability of the proposed approach in dynamic network scenarios, a number of delay traces were gathered, both for fixed and wireless networks. Using the jFed-tool[7] a number of emulab[8] resources across the globe can be accessed from remote locations. Nodes were reserved at Stanford University (S), Northwestern University (N), Ohio Metro Data Center (O), Princeton University (P), Clemson University (C) and Ghent University (G) and used the ping-command to collect the delay traces between every pair of nodes. Table 5.1 gives an overview of the average measured one-way delays between the different static locations. Figure 5.4 gives an overview of the resulting topology that was created interconnecting the streaming server at Ghent University, through a caching proxy at Clemson University to 4 other caching proxies in the US.

Next to the fixed network traces, also 3G/4G traces were collected using a mobile phone that was connected to a node in the iMinds iLab.t testbed[9] located at the Ghent University facilities. These traces were collected for several scenarios: stationary at work and moving in a car during commuting. The commuting traces were collected during a $32km$ drive that was executed multiple times at varying times of the day and following different routes containing highway, city and rural sections. Figure 5.5(a) and 5.5(b) show the collected traces for a moving car and the stationary office environment respectively. These traces were used to configure the latency on the interconnections between the caching proxies and the user devices.

[7] jFed - http://jfed.iminds.be
[8] emulab wiki - https://wiki.emulab.net/wiki
[9] iMinds iLab.t - http://ilabt.iminds.be

*Table 5.1: Average one way delays (ms) and variance for fixed connections between the remote locations.*

| | | | Source | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | Princeton | Clemson | Northwestern | Stanford | Ohio | Ghent |
| Destination | Princeton | d (ms) | 0 | 21.46 | 12.68 | 59.76 | 10.07 | 43.97 |
| | | $\sigma_d$ | 0 | 5.24 | 1.05 | 2.41 | 0.25 | 0.4 |
| | Clemson | d (ms) | 18.33 | 0 | 20 | 66.97 | 19.92 | 60.56 |
| | | $\sigma_d$ | 5.16 | 0 | 6.15 | 7.58 | 5.01 | 7.2 |
| | Northwestern | d (ms) | 12.62 | 21.38 | 0 | 50.92 | 5.74 | 50.38 |
| | | $\sigma_d$ | 1.17 | 4.88 | 0 | 2.61 | 0.4 | 1.16 |
| | Stanford | d (ms) | 60.06 | 58.17 | 50.83 | 0 | 48.54 | 96.54 |
| | | $\sigma_d$ | 2.37 | 2.7 | 2.81 | 0 | 2.47 | 2.43 |
| | Ohio | d (ms) | 10.1 | 22.85 | 5.76 | 48.67 | 0 | 55.39 |
| | | $\sigma_d$ | 0.37 | 5.24 | 0.74 | 2.24 | 0 | 1.05 |
| | Ghent | d (ms) | 44.01 | 60.59 | 50.34 | 96.47 | 55.49 | 0 |
| | | $\sigma_d$ | 0.33 | 5.07 | 1.07 | 2.54 | 1.07 | 0 |




(a) (b)

*Figure 5.5: Excerpt of collected mobile delay traces using a 4G smartphone for a moving car (a) and for a stationary environment (b).*

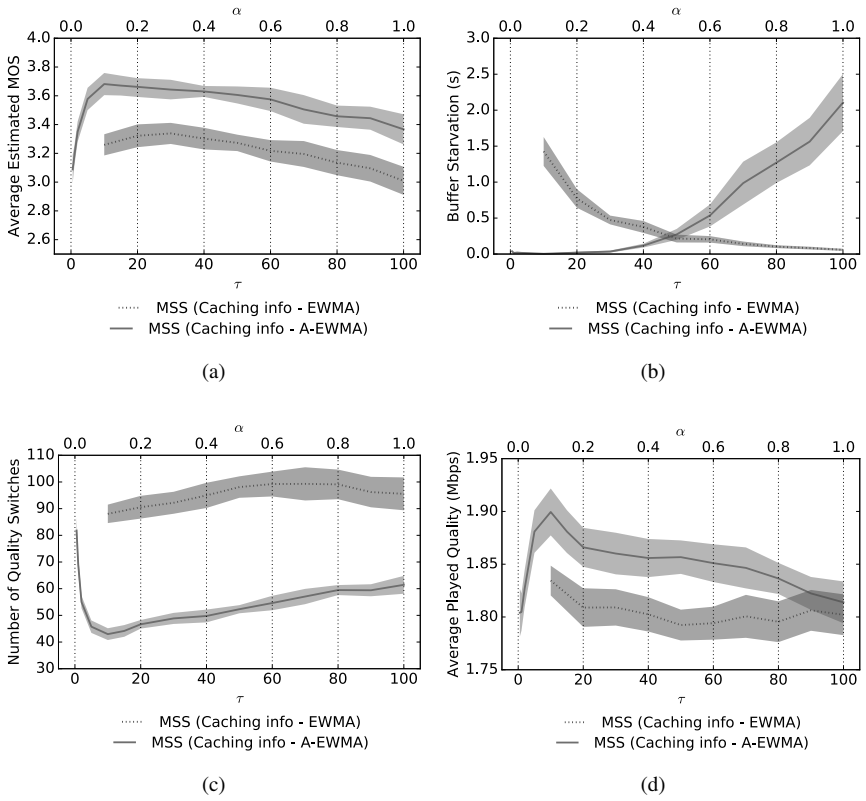



*Figure 5.6: Impact of parameter $\tau$ of aged EWMA estimation on (a) average MOS, (b) average buffer starvation time, (c) average number of quality switches and (d) average played quality.*

### 5.6.2 Impact of estimated throughput ageing

The goal of the proposed heuristics is to be able to cope with streaming over cache networks. To keep the estimated throughput up to date, an ageing factor was added to the EWMA estimator. This allows to balance between the current measurement and previous estimation based on the time that has passed since the previous measurement. Increasing $\tau$ decreases the relative impact of the previous estimations. The $\alpha$-parameter of the regular EWMA estimator shows a similar behavior, but is a static value.

Figure 5.6(a) shows the impact of both the $\alpha$ and the $\tau$-parameter on the average MOS perceived by the end-user for the cache-aware heuristic. Also the results for a client using the regular EWMA estimation, in function of the static $\alpha$-parameter, is shown in these graphs. This client keeps track of the streaming origin

and uses additional cache information to select the next quality, but the throughput estimation does not take into account the age of the collected measurements. Compared to regular EWMA, adding ageing to the estimations allows the heuristic to improve the average MOS with about $11\%$. By letting the weight of the measurements depend on the measurement age, the heuristic is more resistant to fluctuations in throughput that can occur when a streaming origin was not used for a period of time. For regular EWMA estimations, these weights are statically configured and do not fit the wide range of scenarios that can occur during the entire video streaming session.

Increasing $\tau$ beyond $30$, puts too much weight on the historic data which may no longer be up to date due to changes in the network conditions, leading to a rapid increase of buffer starvations as shown in 5.6(b). Setting $\tau$ too high causes the estimations to react too slow to variability in throughput. This can lead to multiple consecutive quality decisions that are not sustainable under the current network conditions and ultimately causes buffer starvations. On the other hand, decreasing $\tau$ puts more weight on the recent measurements, reducing the number of starvations. However, it also causes higher variability in the estimations, leading to an increased number of quality switches, as is demonstrated in Figure 5.6(c). Figure 5.6(d) shows the impact of the parameter $\tau$ on the played bitrate. Also here it is clear that setting $\tau$ too low or too high, leads to a degradation of quality. The parameter sweep shows that the optimal value for $\tau$ is equal to $10$. In the remainder of this section, this value is used during the evaluations.

### 5.6.3 Characterization of the obtained gain of proposed heuristics

In this section, it is assumed that the client heuristic is informed by the streaming origin of its ID and the current cache content. These results show the potential QoE-improvements that can be achieved when the client is able to distinguish the different origins, and additionally, when the client has access to additional information on the current cache content at the different origin locations. To assess the possible improvement, the MOS is used as a metric. When required, the text also refers to the different components that lead to the MOS (i.e. buffer starvations, quality switches and playout quality rate).

The different heuristics are evaluated using a variety of scenarios where the server bandwidth ($BW_S$), the one-way delay of the server link ($d_S$), the cache size of the intermediary proxies ($C_P$) and the client-side buffer size ($B$) are varied. For other variables in the scenarios, the previously mentioned fixed values are used.

Figure 5.7(a) shows the impact of the available bandwidth $BW_S$ on the shared link between the server and the proxies for various heuristics. Also a scenario where no cache is present in the network is shown for the heuristic proposed by

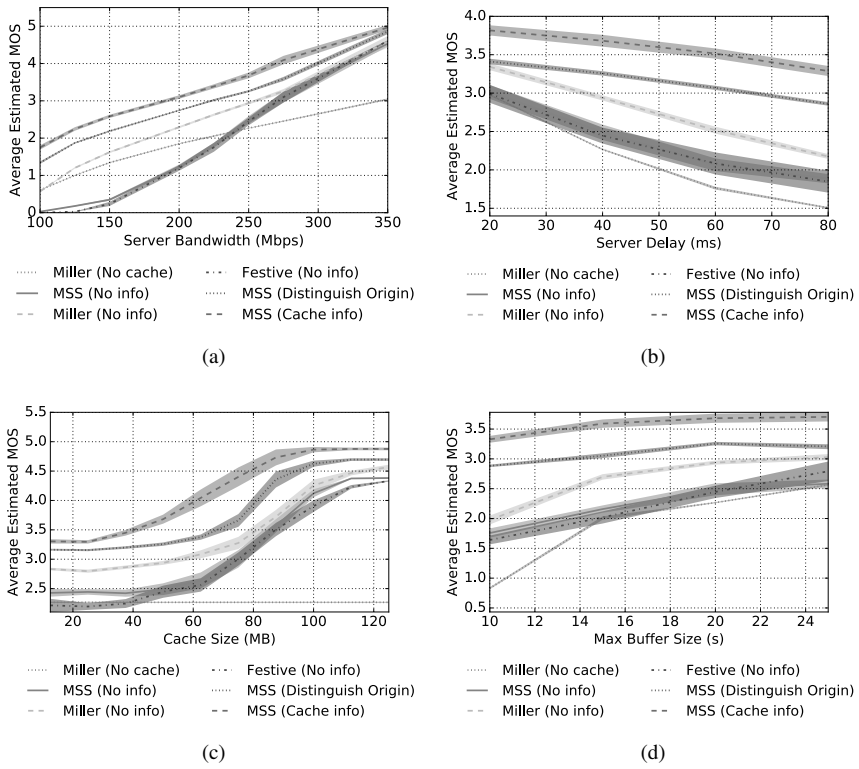



*Figure 5.7: Impact of (a) the bottleneck bandwidth $BW_S$, (b) the delay between server and proxy $d_S$, (c) the cache size $C_P$ and the maximum buffer size $B$ on the average QoE.*

*Miller et al.*, which is referred to as *Miller (No cache)*. For each value of $BW_S$, the cache-aware heuristics are able to outperform the cache-agnostic heuristics. By taking into account the streaming origin, the MOS can be increased by $35\%$ on average, compared to the best performing cache-agnostic heuristic (*Miller*). Also including the information about the upstream cache contents into the decision process can further increase the MOS with $15\%$. *MSS* and *Festive* do not perform well when the bottleneck bandwidth $BW_S$ is below $275Mbps$. This poor performance can be accounted to the high average buffer starvation time ($[4.5s, 76.4s]$ for *MSS* and $[13.1s, 128s]$ for *Festive*) and the high number of switches ($[140.3, 223.8]$ and $[66.5, 192.3]$ respectively). For $BW_S \geq 275Mbps$, all three cache-agnostic heuristics result in a comparable MOS.

By taking into account the streaming origin, the client heuristic is able to estimate the available end-to-end throughput more accurately. Especially when the bottleneck bandwidth is small, making wrong estimations on the available throughput is penalized heavily (e.g., by a buffer starvation or drop in buffer filling followed by a switch to the lowest quality) when a segment is served from the server instead of the caching proxy. By using the worst-case estimation (i.e. the bandwidth to the server) to take the decisions, the *MSS (Distinguish Source)* heuristic is able to significantly increase the overall QoE. If the client is furthermore aware of the upstream cache content, it is able to take into account the throughput estimation for the actual origin which can increase the overal QoE even more (e.g., for $BW_S = 150Mbps$, an increase of $59\%$ compared to *Miller* and $18\%$ compared to *MSS (Distinguish Source)*).

The impact of the one-way delay between the server and the caching proxy ($d_S$) on the QoE is shown in Figure 5.7(b). Previous work has shown that an increasing delay has a negative impact on the QoE for HAS-streaming due to the TCP-based request-reply patterns that are generated [5]. Increasing $d_S$ increases the idle times between two consecutive downloads and thus the efficiency of the streaming session. The QoE decay with increasing delay is much steeper for the cache-agnostic heuristics. For increasing delay, the gain of including the streaming origin and cache information increases from $2\%$ to $32\%$ and $14\%$ to $51\%$ respectively.

Figure 5.7(c) shows the impact of increasing the capacities $C_P$ of the in-network proxy caches on the QoE for the different heuristics. For smaller cache sizes, more segments will be served from the server, clogging the link between the server and the caching proxy and leading to a reduction in played quality. Furthermore, the probability increases that for consecutive segments, the streaming origin will differ, causing wrong estimations of the available throughput and eventually, buffer starvations. For a cache size of $100MB$, most of the segments can be served from the cache for the *MSS (Cache Info)* heuristic since it can exploit the knowledge of which quality levels that are present in the cache, allowing all

clients to quickly converge to the highest quality level. For the other heuristics, the requested quality levels are more dispersed across the available range of representations, causing a higher number of cache misses and eventually leading to lower quality selections based on the throughput that is perceived to the server. The *MSS (Distinguish Source)* heuristic is able to distinguish between segments served from cache and server and allows a more accurate throughput estimation, leading to a higher average MOS. On average, this allows the QoE to increase with $0.34$, while including the cache information allows a further increase of the MOS with $0.33$. Especially for smaller cache sizes ($C_P \in [37.5MB, 87.5MB]$), the additional knowledge of the cache content allows a significant increase in QoE with about $28\%$.

Increasing the buffer size $B$ typically benefits the robustness of the quality adaptation heuristic since they are more forgiving when an unsustainable quality is selected. Figure 5.7(d) shows the effects of increasing the maximum buffer size from $10s$ to $25s$. Smaller buffers allow a shorter time-to-display delay for live streaming but also in VoD scenarios, smaller buffers can be beneficial. For example, the initial startup delay can be decreased when the client starts after a percentage of the buffer is filled. Furthermore, less unnecessary data is transferred when the client prematurely aborts the video streaming session. Since most adaptation heuristics apply some type of panic threshold which is typically $20\%$ of the buffer size in seconds and during the evaluations a segment size of $2s$ is used, the smallest buffer size that was employed during the experiments was $10s$. The results show that the *MSS (Distinguish Source)* heuristic is able to achieve the same QoE of a buffer size of $12s$ as the *Miller* heuristic for a maximum buffer size of $20s$.

### 5.6.4 Impact of detection and estimation

In the previous section, the proposed heuristics are evaluated assuming that they are able to retrieve the required information. However, it is possible that this information is not available (e.g., when optimizing third-party streaming services, no access to the proxy implementations) and that this information should be derived by the client using a combination of clustering techniques to estimate the origin and probabilistic models to estimate the current cache content. It is obvious that in such situations, the heuristic is only able to approximate the proposed optimal heuristics. To optimize the different detection and estimation techniques, traces were generated for the optimal heuristics in which the streaming origin and the RTT for that origin are logged. This data is used as ground truth for the origin clustering optimization which is performed using an offline simulator that is implemented in python. This allows us to quickly evaluate the impact of the different parameters of the clustering algorithm and select the optimal configuration.

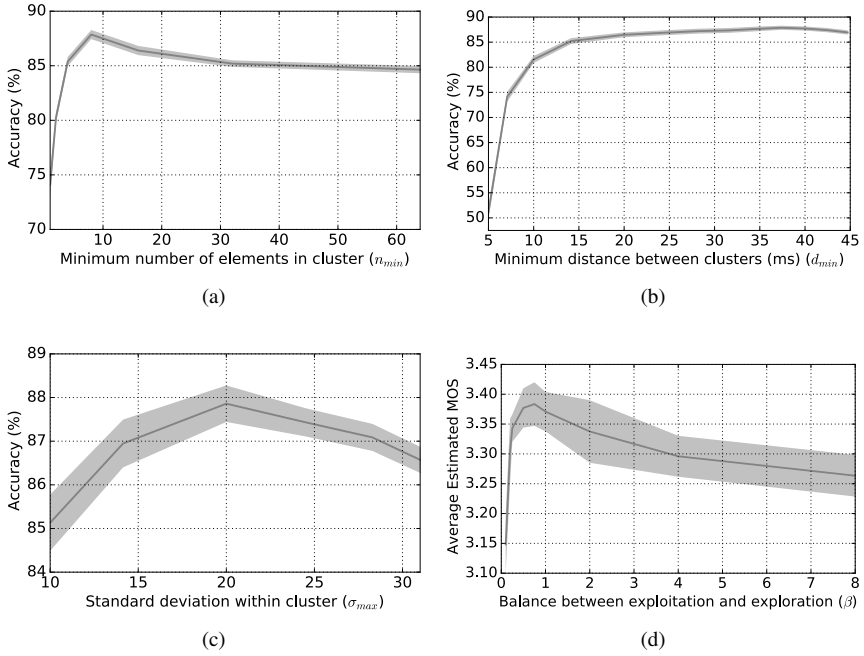



*Figure 5.8: Impact of (a) minimum number of elements in cluster $n_{min}$, (b) minimum distance between clusters $d_{min}$ and (c) maximum standard deviation within a cluster $\sigma^2$ on the clustering accuracy expressed as percentage of correctly classified elements. Impact of cache estimation parameter $\beta$ (d) on average MOS.*

Several parameters should be configured for the clustering-based detection of the streaming origin: $K_0$, $n_{min}$, $\sigma_{max}$ and $d_{min}$. $K_0$ is the desired number of clusters and thus depends on the number of streaming origins in the topology that is used. For a topology including a single cache and a streaming server $K_0 = 2$. Also the cache estimation has a configurable parameter $\beta$, which regulates the balance between exploration and exploitation. The different parameters are optimized using a set of traces that were generated during the previous experiments for a configuration of $d_S = 40ms$, $BW_S = 250Mbps$ and $C_P = 50MB$.

Figure 5.8(a) shows the impact of increasing the number of required elements to form a cluster $n_{min}$, which is used for discarding clusters. Setting this value too low, can lead to the creation of many small clusters, while setting it too high can lead to the wrong classification in case only few segments are downloaded from that specific origin. The optimal value determined by the parameter sweep is $n_{min} = 8$. The impact of the minimum distance between two clusters ($d_{min}$) is shown in Figure 5.8(b). Since the traces for $d_S = 40ms$ were used, the optimal value is around $d_{min} = 37.5ms$, which optimally allows to differentiate between

*Table 5.2: Probability of predicting a segment to be in the cache for $\beta = 0.75$ as a function of the buffer level $B$.*

| Buffer Filling (%) | Prob(not cached) | Prob(cached) |
|---|---|---|
| 0 | 0.527633447 | 0.472366553 |
| 10 | 0.490843579 | 0.509156421 |
| 20 | 0.451188364 | 0.548811636 |
| 30 | 0.408444636 | 0.591555364 |
| 40 | 0.362371848 | 0.637628152 |
| 50 | 0.312710721 | 0.687289279 |
| 60 | 0.259181779 | 0.740818221 |
| 70 | 0.201483781 | 0.798516219 |
| 80 | 0.139292024 | 0.860707976 |
| 90 | 0.072256514 | 0.927743486 |
| 100 | 0 | 1 |

the content server and proxy. It is clear that $d_{min}$ is dependent on $d_S$ and could be configured by using additional information retrieved by the client. However, in absence of such information, the various settings for $d_{min} \geq 10ms$ still allow to determine the origin with an accuracy between $82\%$ to $88\%$. Figure 5.8(c) shows the impact of the standard deviation between the different samples within the cluster $\sigma_{max}$. The optimal value here is $\sigma_{max} = 20$, which again can be related to the value that was set for $d_S$. Also here other settings for $\sigma_{max}$ will still allow the clustering to correctly classify the samples with an accuracy between $85\%$ and $88\%$.

The impact of the parameter $\beta$ on the average estimated QoE is shown in Figure 5.8(d). The $\beta$ parameter regulates the probability that a certain segment is predicted to be present in the cache for different buffer filling levels $B$. Setting $\beta$ too high decreases this probability, leading to a more conservative quality selection and ultimately a lower QoE. Setting $\beta$ too small ($\beta = 0.1$) leads to an $95\%$ chance a segment is predicted to be in the cache, even though the buffer is only half full. This behavior leads to a more aggressive quality selection and thus a higher risk of overshooting the actual sustainable quality level. The optimal value for $\beta$ was determined to be $0.75$, which leads to the probabilities shown in Table 5.2

### 5.6.5 Achieved improvement using inferred knowledge

Section 5.6.3 showed that for practically every scenario, the heuristic proposed by *Miller et al.* outperforms the other cache-agnostic heuristics. In the remainder of this section, we therefore compare the heuristics using inferred knowledge with the adaptation heuristic proposed by *Miller et al.* and the heuristics using perfect knowledge. Using the parameter configurations inferred in the previous section, the performance of the proposed heuristics using inferred knowledge will be evaluated.

Figure 5.9(a) shows the QoE for varying bandwidth $BW_S$. When using the clustering-based approach to detect the streaming origin, the client applications

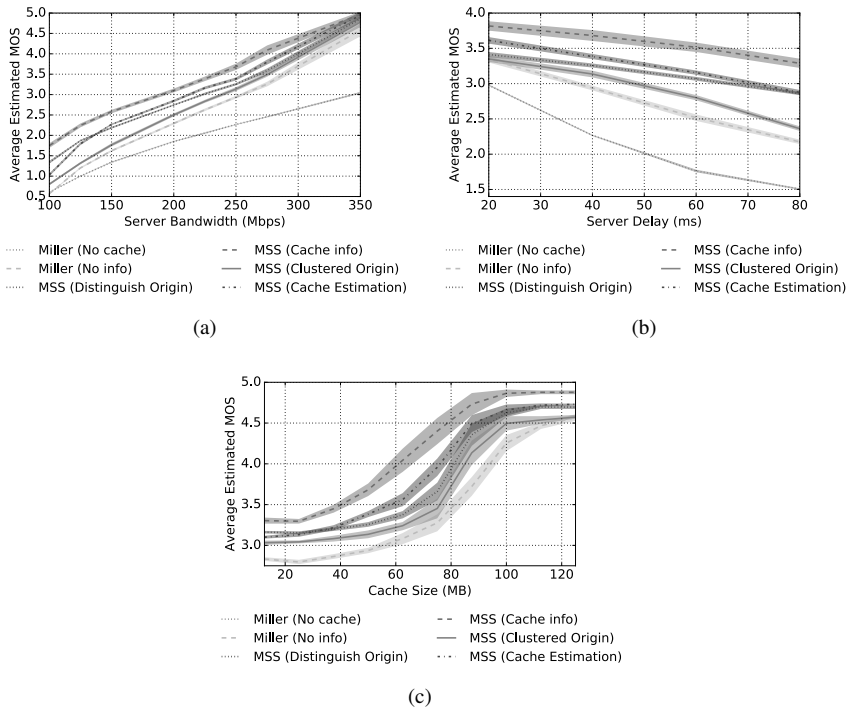



*Figure 5.9: Impact of (a) the bottleneck bandwidth $BW_S$, (b) the delay between server and proxy $d_S$ and (c) the cache size $C_P$ on the average QoE.*

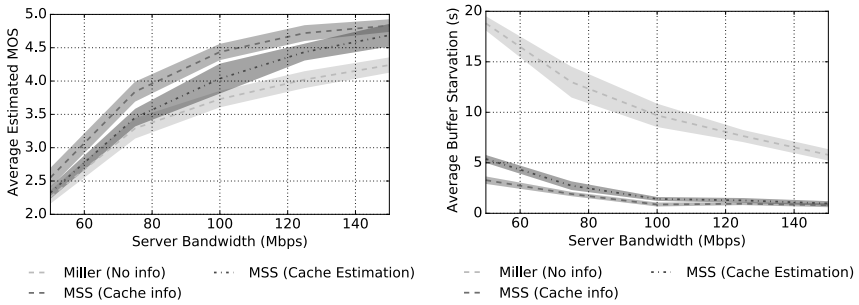
are able to increase the MOS with on average $0.19$ compared to cache-agnostic heuristics. This is only $86\%$ of the QoE achieved by the origin-aware heuristic using perfect knowledge. When also estimating the cache contents in a probabilistic way, the MOS can be improved further with $0.32$, which is $87\%$ of the QoE when using the exact data. For lower $BW_S$ ($BW_S \leq 200Mbps$), the achieved increase in QoE is much smaller, this can be assigned to a higher cost of using a wrong estimation on either the origin or cache content during the decision process. The heuristic using the estimated cache info is cautious when increasing the quality by also taking into account the worst-case estimated server bandwidth, reducing the negative impact of wrong estimations. On average, the QoE can be improved with $32\%$ when estimating the cache contents and $12\%$, when only detecting the origin.

Since the clustering is based on the perceived RTT, the one-way delay $d_S$ between server and caching proxy can have an impact on the performance of the clustering algorithm. Figure 5.9(b) shows indeed that for increasing $d_S$, the QoE decreases much faster than when the actual origin data is received (i.e. from $95\%$ to $87\%$ for *MSS Cache Estimation* and from $98\%$ to $83\%$ for *MSS Origin Clustering*). The different parameters of the clustering algorithm were optimized based on traces that were captured for $d_S = 40ms$, which are not optimal for all other values of $d_S$. This is a drawback of the proposed approximation method since the clustering configuration impacts the performance of the client adaptation heuristic. A possible improvement would be to inform the client of the network delays for different origin locations or to determine them via measurements. These values could then serve as inputs for the clustering algorithm.

Figure 5.9(c) shows the impact of the cache size $C_P$ on the performance of the heuristics using inferred knowledge. Also here, a more modest performance increase can be observed when using imperfect origin data, especially for smaller cache sizes (i.e. $C_P \leq 75MB$). For larger cache sizes, the performance increase compared to cache-agnostic heuristics is between $1\%$ and $11\%$ when using clustering-based origin detection and between $4\%$ and $21\%$ when also estimating the cache contents. The cache-aware heuristic using inferred knowledge using a cache of $80MB$ can achieve the same QoE as *AVC Miller* for a $100MB$, thus leading to a reduction in resources of $20\%$.

### 5.6.6 Dynamic network scenarios

Figure 5.10(a) shows the impact of applying the cache-aware heuristics in a dynamic scenario. Using perfect information, the QoE can be improved with $16\%$ on average, compared to the best performing cache-agnostic heuristic, *Miller*. Using clustering techniques to detect the origin and the probability-based method to estimate the cache contents allows to improve the QoE between $3\%$ and $11\%$ on average. Depending on the configuration of $BW_S$, the achieved QoE is between




(a) (b)

*Figure 5.10: QoE-improvement (a) and reduction of buffer starvations (b) in a dynamic scenario.*

$90\%$ and $97\%$ compared to the heuristics using perfect knowledge. The average buffer starvations are shown in Figure 5.10(b). There is a significant reduction in buffer starvations when using the cache-aware heuristics in the dynamic scenario. The buffer starvations can be reduced with a factor 3 to 7 for the estimation heuristic.

## 5.7 Conclusions

HTTP Adaptive Streaming (HAS) has become the de-facto standard for multimedia streaming over the Internet. One of the advantages of HTTP streaming is the seamless reuse of existing general purpose HTTP caching infrastructure. However, one of the drawbacks of current adaptation heuristics is that the decisions are partially based on throughput estimations. Since the origin for the segment downloads is not considered, these throughput estimations cannot be trusted anymore when multiple consecutive segments are served from different origins. These incorrect estimations ultimately lead to quality switches and even buffer starvations, negatively impacting the Quality of Experience (QoE). This chapter proposes to take advantage of additional information on the streaming origin and use multiple throughput estimations per origin. Using the worst-case throughput, the adaptation heuristic is made more robust against streaming over cache networks, reducing quality switching and buffer starvations. Furthermore, leveraging information on the current cache contents, more informed quality selection decisions can be made, increasing the video bitrate that is played. Using the aforementioned optimizations, the QoE can be improved by up to $60\%$ in the static and with $16\%$ on average in a dynamic scenario. These solutions require adaptations to the streaming origin and intermediary proxies, which is not always possible. Therefore, this chapter

also proposes detection and estimation techniques based on clustering techniques using the perceived Round Trip Time (RTT) as a feature and probability based solutions to estimate the cache contents. Using these approximation techniques, the QoE can be improved between $3\%$ and $11\%$, depending on the scenario.

# References

[1] Forecast, Cisco VNI. *Cisco Visual Networking Index: Global Mobile data Traffic Forecast Update 2012-2017*. Technical report, Cisco Public Information, May 2013.

[2] T. Stockhammer. *Dynamic adaptive streaming over HTTP: standards and design principles*. In Proceedings of the second annual ACM conference on Multimedia systems, MMSys '11, pages 133–144, New York, NY, USA, 2011. ACM.

[3] N. Staelens, J. D. Meulenaere, M. Claeys, G. V. Wallendael, W. V. den Broeck, J. D. Cock, R. V. de Walle, P. Demeester, and F. D. Turck. *Subjective Quality Assessment of Longer Duration Video Sequences Delivered Over HTTP Adaptive Streaming to Tablet Devices*. IEEE Transactions on Broadcasting, 60(4):707–714, Dec 2014.

[4] J. De Vriendt, D. De Vleeschauwer, and D. Robinson. *Model for estimating QoE of video delivered using HTTP adaptive streaming*. In Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, pages 1288–1293. IEEE, 2013.

[5] N. Bouten, S. Latré, J. Famaey, F. De Turck, and W. Van Leekwijck. *Minimizing the impact of delay on live SVC-based HTTP adaptive streaming services*. In Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 1399–1404, 2013.

[6] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz. *Adaptation algorithm for adaptive streaming over HTTP*. In Proceedings of the 19th International Packet Video Workshop (PV), pages 173–178. IEEE, 2012.

[7] J. Jiang, V. Sekar, and H. Zhang. *Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE*. In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12, pages 97–108. ACM, 2012.

[8] G. Tian and Y. Liu. *Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming*. In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12, pages 109–120, 2012.

[9] 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 Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 419–426, 2013.

[10] C. Liu, I. Bouazizi, M. M. Hannuksela, and M. Gabbouj. *Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network*. Signal Processing: Image Communication, 27(4):288 – 311, 2012.

[11] V. Adzic, H. Kalva, and B. Furht. *Optimized Adaptive HTTP Streaming for Mobile Devices*. In SPIE Optical Engineering+ Applications, pages 81350T–81350T. International Society for Optics and Photonics, 2011.

[12] R. Kuschnig, I. Kofler, and H. Hellwagner. *An Evaluation of TCP-based Rate-control Algorithms for Adaptive Internet Streaming of H.264/SVC*. In Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems, MMSys '10, pages 157–168, 2010.

[13] R. Houdaille and S. Gouache. *Shaping HTTP Adaptive Streams for a Better User Experience*. In Proceedings of the 3rd Multimedia Systems Conference, MMSys '12, pages 1–9, 2012.

[14] S. Akhshabi, A. C. Begen, and C. Dovrolis. *An Experimental Evaluation of Rate-adaptation Algorithms in Adaptive Streaming over HTTP*. In Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSys), pages 157–168, 2011.

[15] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. *What Happens when HTTP Adaptive Streaming Players Compete for Bandwidth?* In Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), pages 9–14, 2012.

[16] D. C. Robinson, Y. Jutras, and V. Craciun. *Subjective Video Quality Assessment of HTTP Adaptive Streaming Technologies*. Bell Labs Technical Journal, 16(4):5–23, 2012.

[17] J. Esteban, S. A. Benno, A. Beck, Y. Guo, V. Hilt, and I. Rimac. *Interactions Between HTTP Adaptive Streaming and TCP*. In Proceedings of the 22Nd International Workshop on Network and Operating System Support for Digital Audio and Video, NOSSDAV '12, pages 21–26, 2012.

[18] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. Tran-Gia. *A Survey on Quality of Experience of HTTP Adaptive Streaming*. Communications Surveys Tutorials, IEEE, PP(99):1–1, 2014.

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

[20] X. Liu and A. Men. *QoE-aware Traffic Shaping for HTTP Adaptive Streaming*. International Journal of Multimedia & Ubiquitous Engineering, 9(2), 2014.

[21] N. Bouten, S. Latré, J. Famaey, W. Van Leekwijck, and F. De Turck. *In-Network Quality Optimization for Adaptive Video Streaming Services*. Multimedia, IEEE Transactions on, 16(8):2281–2293, Dec 2014.

[22] Z. Li, M. Sbai, Y. Hadjadj-Aoul, A. Gravey, D. Alliez, J. Garnier, G. Madec, G. Simon, and K. Singh. *Network friendly video distribution*. In Network of the Future (NOF), 2012 Third International Conference on the, pages 1–8, Nov 2012.

[23] J. Famaey, S. Latré, R. van Brandenburg, M. O. van Deventer, and F. De Turck. *On the Impact of Redirection on HTTP Adaptive Streaming Services in Federated CDNs*. In Proceedings of the 7th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security Volume 7943, AIMS'13, pages 13–24, 2013.

[24] A. El Essaili, D. Schroeder, D. Staehle, M. Shehada, W. Kellerer, and E. Steinbach. *Quality-of-experience driven adaptive HTTP media delivery*. In Communications (ICC), 2013 IEEE International Conference on, pages 2480–2485, June 2013.

[25] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. *Towards Network-wide QoE Fairness using OpenFlow-assisted Adaptive Video Streaming*. In Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking, pages 15–20. ACM, 2013.

[26] C. Mueller, S. Lederer, C. Timmerer, and H. Hellwagner. *Dynamic Adaptive Streaming over HTTP/2.0*. In Multimedia and Expo (ICME), 2013 IEEE International Conference on, pages 1–6, July 2013.

[27] S. Wei and V. Swaminathan. *Low Latency Live Video Streaming over HTTP 2.0*. In Proceedings of Network and Operating System Support on Digital Audio and Video Workshop, NOSSDAV '14, pages 37:37–37:42, 2014.

[28] S. Wei and V. Swaminathan. *Cost effective video streaming using server push over HTTP 2.0*. In Multimedia Signal Processing (MMSP), 2014 IEEE 16th International Workshop on, pages 1–5, Sept 2014.

[29] N. Bouten, M. Claeys, S. Latre, J. Famaey, W. Van Leekwijck, and F. De Turck. *Deadline-based approach for improving delivery of SVC-based HTTP Adaptive Streaming content*. In Network Operations and Management Symposium (NOMS), 2014 IEEE, pages 1–7, May 2014.

[30] N. Bouten, S. Latré, W. Van de Meerssche, K. De Schepper, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. *An autonomic delivery framework for HTTP Adaptive Streaming in multicast-enabled multimedia access networks*. In Proceedings of the IEEE Network Operations and Management Symposium (NOMS), pages 1248–1253, 2012.

[31] N. Bouten, S. Latré, W. Van de Meerssche, B. De Vleeschauwer, K. De Schepper, W. Van Leekwijck, and F. De Turck. *A Multicast-Enabled Delivery Framework for QoE Assurance of Over-The-Top Services in Multimedia Access Networks*. Journal of Network and Systems Management, 21(4):677–706, 2013.

[32] S. Petrangeli, M. Claeys, S. Latre, J. Famaey, and F. De Turck. *A multi-agent Q-Learning-based framework for achieving fairness in HTTP Adaptive Streaming*. In Network Operations and Management Symposium (NOMS), 2014 IEEE, pages 1–9, May 2014.

[33] C. Mueller, S. Lederer, and C. Timmerer. *A proxy effect analyis and fair adatpation algorithm for multiple competing Dynamic Adaptive Streaming over HTTP clients*. In Visual Communications and Image Processing (VCIP), 2012 IEEE, pages 1–6, Nov 2012.

[34] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. *Helping Hand or Hidden Hurdle: Proxy-Assisted HTTP-Based Adaptive Streaming Performance*. In 2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, pages 182–191, Aug 2013.

[35] D. H. Lee, C. Dovrolis, and A. C. Begen. *Caching in HTTP Adaptive Streaming: Friend or Foe?* In Proceedings of Network and Operating System Support on Digital Audio and Video Workshop, NOSSDAV '14, pages 31:31–31:36, New York, NY, USA, 2014. ACM.

[36] H. Zhang and D. Choffnes. *Client-Side Web Proxy Detection from Unprivileged Mobile Devices*. arXiv preprint arXiv:1511.04493, 2015.

[37] Z. Trabelsi, H. Rahmani, K. Kaouech, and M. Frikha. *Malicious sniffing systems detection platform*. In Applications and the Internet, 2004. Proceedings. 2004 International Symposium on, pages 201–207. IEEE, 2004.

[38] Z. Tun and N. L. Thein. *Round Trip Time based Wormhole Attacks Detection*. In IEEE Wireless Communications andNetworking Conference-WCNC, 2008.

[39] X. Xu, Y. Jiang, T. Flach, E. Katz-Bassett, D. Choffnes, and R. Govindan. *Investigating transparent web proxies in cellular networks*. In International Conference on Passive and Active Network Measurement, pages 262–276. Springer, 2015.

[40] N. Weaver, C. Kreibich, M. Dam, and V. Paxson. *Here be web proxies*. In International Conference on Passive and Active Network Measurement, pages 183–192. Springer, 2014.

[41] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet. *Revealing middlebox interference with tracebox*. In Proceedings of the 2013 conference on Internet measurement conference, pages 1–8. ACM, 2013.

[42] S. Petrangeli, N. Bouten, M. Claeys, and F. D. Turck. *Towards SVC-based Adaptive Streaming in information centric networks*. In Proceedings of IEEE International Conference on Multimedia Expo Workshops (ICMEW), pages 1–6, June 2015.

[43] N. Bouten, R. de O. Schmidt, J. Famaey, S. Latré, A. Pras, and F. D. Turck. *QoE-driven in-network optimization for Adaptive Video Streaming based on packet sampling measurements*. Computer Networks, 81:96 – 115, 2015.

[44] S. P. Lloyd. *Least squares quantization in PCM*. Information Theory, IEEE Transactions on, 28(2):129–137, 1982.

[45] B. Aaron, D. E. Tamir, N. D. Rishe, and A. Kandel. *Dynamic incremental K-means clustering*. In Computational Science and Computational Intelligence (CSCI), 2014 International Conference on, volume 1, pages 308–313. IEEE, 2014.

[46] N. Memarsadeghi, D. M. Mount, N. S. Netanyahu, and J. Le Moigne. *A fast implementation of the ISODATA clustering algorithm*. International Journal of Computational Geometry & Applications, 17(01):71–103, 2007.

[47] N. Bouten, J. Famaey, S. Latré, R. Huysegems, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. *QoE optimization through in-network quality adaptation for HTTP Adaptive Streaming*. In Proceedings of the International Conference on Network and Service Management (CNSM), pages 336–342, 2012.

[48] M. Claeys, S. Latre, J. Famaey, and F. D. Turck. *Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client*. IEEE Communications Letters, 18(4):716–719, April 2014. doi:10.1109/LCOMM.2014.020414.132649.

[49] W. Bellante, R. Vilardi, and D. Rossi. *On Netflix catalog dynamics and caching performance*. In 2013 IEEE 18th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), pages 89–93, Sept 2013.

[50] C. Fricker, P. Robert, J. Roberts, and N. Sbihi. *Impact of traffic mix on caching performance in a content-centric network*. In Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on, pages 310–315. IEEE, 2012.

# 6

# Semantically Enhanced Mapping Algorithm for Affinity Constrained Service Function Chain Requests

**N. Bouten, R. Mijumbi, J. Serrat, J. Famaey, S. Latré, F. De Turck**

**Submitted to Transactions on Network and Service Management (TNSM), May 2016.**

*Chapter 2-5 and Appendices A-B present various Service Function Chains (SFCs) and Network Functions (NFs) that optimize the Quality of Experience (QoE) for the delivery of HTTP Adaptive Streaming (HAS) services. To increase the service agility and reduce the time to market of new services, these NFs can be virtualized into Virtual Network Functions (VNFs) and by leveraging Network Function Virtualization (NFV) technologies, deployed on the fly onto the virtualized substrate. However, many service chains have some assumptions on the locality of the VNFs to achieve the expected performance gain. This chapter defines a set of affinity and anti-affinity constraints which allow the Service Provider (SP) to define locality restrictions on VNFs and interconnecting virtual edges. Since SFCs can be generated automatically or by human operators, a semantic validation framework is proposed in which the substrate and SFCs are modeled using ontologies. By defining a set of inference rules, the semantic reasoner can detect inconsistent constraint sets. Additional algorithms are proposed to map the SFC sets onto the substrate subject to the capacity and affinity constraints.*

## 6.1 Introduction

In traditional telecommunications networks, network functionality is strongly tied to the physical network device it runs on. To adapt or create network services, the network operator needs to deploy a dedicated network appliance for each Network Function (NF) (e.g., Deep Packet Inspection (DPI), Firewall). Furthermore, the placement of the NFs has to adhere to a strict chaining order, which increases the tight coupling of the service with the underlaying network topology. Together with the ever increasing requirements for high quality and stability, this has led to long product cycles, limited service agility and substantial dependence on specialized hardware. In order to compete with highly agile Over-The-Top (OTT) services, which typically have much shorter product development cycles, and at the same time reduce the Capital Expenditures (CAPEX) and Operational Expenditures (OPEX) involved with physical network expansions, network operators need to devise novel and less expensive ways to meet the increased capacity requirements and at the same time reduce the time to market of newly developed services.

The Network Function Virtualization (NFV)-paradigm [1, 2] has been introduced to alleviate the aforementioned issues by leveraging IT virtualization technology to decouple the network functionality from the physical infrastructure. It allows Virtual Network Functions (VNFs) to be deployed on standard high volume servers, storage devices and switches. The advantages are manyfold. First, there potentially is a significant reduction in total costs through more efficient maintenance, which can be performed remotely. In addition, thanks to the increased flexibility offered by virtualization, resources can be shared and used more efficiently. Finally, NFV allows network operators to deploy novel services cheaper and faster with higher service agility.

The concepts of NFV have introduced new business opportunities for Virtual Network Function Infrastructure Providers (VNFInPs), which act as brokers between Infrastructure Providers (InPs) and Service Providers (SPs) [3]. InPs can profit from opening (part of) their infrastructure and providing virtualized access to remote parties, thereby maximizing resource utilization and optimizing energy usage [4]. The VNFInPs act as brokers between SPs and InPs and couple the different virtualized infrastructures into an end-to-end virtualized infrastructure. The VNFInPs lease the infrastructure provided by different InPs and deploy, orchestrate and interconnect VNFs to create end-to-end Service Function Chains (SFCs) [5], which are operated by SPs to offer value-added services to their customers. SPs benefit from the proposed model since it allows rapid deployment and testing in a real network environment, thus allowing faster time to market of new services. The offered services benefit from the dynamic nature of the network, computing and storage resources offered by the NFV infrastructure, which allows them to dynamically scale based on service requirements and user mobility.

Together with these new opportunities and stakeholders, a set of new interactions arises as well. For example, the SPs need a way to express their SFC requests and requirements to the VNFInP. In traditional network embedding approaches [6], only node and link restrictions can be specified. However, many scenarios can be envisioned where a SP might want to attach more detailed constraints concerning the placement and routing between NFs as well as constraints on their colocation. For example, to increase efficiency, the SP may require the embedding of VNFs at a certain granularity (e.g., within the same datacenter or even on the same host). Other reasons for more detailed affinity and anti-affinity constraints could be resilience, economic, legislative and privacy issues. In this chapter, a set of affinity and anti-affinity constraints is proposed that increases the control of SPs on the embedding of their SFC requests.

With this newfound ability to add custom constraints, the possibility arises that conflicting constraints are introduced by SPs in their requested SFCs. Therefore, the VNFInP requires a method to check the consistency of SFC requests and inform the SP on potential conflicts. For example, if the SP states that two VNFs of certain types need to be embedded on the same host, but also requires that any instances of functions with those VNF types are required to be embedded in distinct datacenters, a conflict arises. Since SFC requests can contain many VNFs, virtual edges and constraints, detecting conflicts within these requests is not a straightforward task, neither for human operators, nor for computer systems. Since conflicts can arise between sets of constraints, pairwise detection will not suffice. Therefore, this chapter proposes to take advantage of semantic modelling to define an ontology and rule set, which can be enriched with individuals based on the specific SFC request. Using a semantic reasoner, the consistency can be determined and subsequently the validity of the SFC request can be assessed. Furthermore, the proposed framework can also be used by SPs to validate the SFC embeddings made by the VNFInP with respect to the imposed affinity requirements.

Most of the VNF and Virtual Network (VN) embedding approaches focus on the resource constraints posed by the SFC requests. These approaches do not take into account other constraints during the embedding. In this chapter, the affinity-constrained SFC placement problem is formalized. First, the resource and flow conservation constraints are modeled for the concurrent mapping of SFCs sets. Second, the affinity and anti-affinity constraints are added to the model. Finally, to allow scalable SFC mapping, a heuristic procedure is proposed. This heuristic approach takes advantage of additional knowledge of the SFC set to sort the individual SFCs in a way that optimizes the objective. Both the optimal and heuristic approaches are evaluated considering different metrics and use cases, such as the number of VNFs in an SFC, the size of the physical infrastructure, the number of SFCs in a set and the impact of applying semantic filtering to the SFC request set prior to embedding.

The contributions of this chapter are fourfold. First, a set of affinity and anti-affinity constraints that can be attached to an SFC request by the SP are defined. Second, this chapter proposes and evaluates a semantic conflict detection mechanism that can be employed by the VNFInP to check the validity of SFC requests. In this way, SFCs containing inconsistent constraints can be filtered out by the VNFInP before the embedding step. Third, a mathematical formulation is proposed of the affinity-constrained SFC embedding problem as well as a set of algorithms to solve them. Finally, a heuristic approach to tackle the computationally expensive task of embedding SFC sets is proposed and evaluated.

The remainder of this chapter is structured as follows: Section 6.2 discusses the relevant related work in the field of embedding and affinity constraint mapping in other research fields. The affinity and anti-affinity constraints are discussed in Section 6.3, while Section 6.4 describes the semantic modelling for affinity-constrained SFC requests. The Integer Linear Programming (ILP) model for affinity-constrained SFC embedding is discussed in Section 6.5. The proposed heuristic approach is discussed in Section 6.6. Section 6.7 discusses the evaluation platform, among which the implementation of the semantic SFC request checking engine and the embedding algorithm, as well as a set of results on the performance of the proposed approaches.

## 6.2 Related Work

NFV has been proposed as a paradigm that allows more flexible service deployment by leveraging IT virtualization technology in combination with programmable networks [7, 8]. A recent survey on NFV by co-authors of this chapter, identifies the decoupling of NFs from hardware, flexible NF deployment and dynamic scaling as the main differences between network service provisioning in NFV compared to current practice [9]. The authors provide an overview of the history of NFV, the NFV architecture proposed by ETSI, the standardization efforts and relevant research and implementation projects. Furthermore, several research challenges are identified, among which the security, privacy and trust concerns, the modelling of resources, functions and services and the efficient allocation of NFV resources [10]. This chapter focuses on defining a set of affinity and anti-affinity constraints that can be used to impose function placement requirements (e.g., for privacy or legislative purposes). Furthermore, a framework is provided to model these constraints and to evaluate their mutual interactions and consistency. Finally, an alternative optimization model is proposed taking into account the resource and flow constraints, as well as the affinity constraints imposed by a set of SFC requests.

Next to the management and orchestration challenges, *Han et al.* identify the network performance of VNFs to be a major obstacle for the adoption of NFV

technologies [8]. The end-to-end networking performance has been shown to be heavily impacted by the sharing of processors, even when the network is not heavily loaded [11]. To alleviate these performance issues, *Hwang et al.* propose the NetVM platform which allows network functions based on the Intel DPDK technology to be executed at line-speed on top of commodity hardware [12]. *Martins et al.* propose a high-performance platform named ClickOS, which allows the execution of hundreds of concurrent VNFs without incurring significant overhead [13]. This chapter does not focus on a specific VNF implementation platform, but provides a general approach to embed affinity-constrained SFCs, independent of the underlying implementations. During the optimization, the packet processing resources required by a certain VNF type are taken into account, as well as the packet processing resources required by intermediary routing nodes.

Affinity and anti-affinity restrictions have previously been studied in the context of grid and cloud computing. Many argued that the lack of influence on the placement of workflow or service components is a hindrance for the adoption of the technology [14, 15]. Even though performance and economical benefits of cloud computing are clear, potential users hesitate to deploy the technology because legal, privacy, efficiency and resilience aspects are completely out of their control. Also, recent media coverage shows an increased concern by end-users, companies and governments about their data privacy, raising the need for SPs to take into account these issues when deploying and offering their services[1,2]. These concerns also arise for NFV when deploying VNFs at certain locations and transferring data between them over virtual paths. Therefore, it is argued that also in NFV, mechanisms should be designed to allow SPs to add constraints concerning locality and affinity, both to VNFs as well as the interconnecting paths.

The solutions proposed in the affinity and anti-affinity context for cloud computing mostly relate to two aspects: developing models to describe affinity rules and developing service placement algorithms that can work under the constraints of these rules. *Konstanteli et al.* present a set of affinity rules for cloud computing applications which are added to a Mixed-Integer Non-Linear Programming (MINLP) [16]. The authors define constraints that require allocating components and services in the same subnet or physical node, or prevent services from being federated. *Larsson et al.* and *Espling et al.* propose a model for defining Virtual Machine (VM) placement in cloud computing supporting a set of affinity and anti-affinity constraints [17, 18]. This approach is extended by defining affinity and anti-affinity restrictions for SFCs. To this end support was added for the specification of constraints on the path between network functions and furthermore, a more expressive syntax was proposed that allows constraints to apply to

[1]No Safe Harbor: How NSA Spying Undermined U.S. Tech and Europeans' Privacy - https://www.eff.org/nl/deeplinks/2015/10/europes-court-justice-nsa-surveilance

[2]Facebook case may force European firms to change data storage practices - http://www.theguardian.com/us-news/2015/sep/23/us-intelligence-services-surveillance-privacy

specific VNFs, VNF types, locations and location types. A semantic framework is proposed which allows to check the validity of these constraints.

One of the benefits of NFV is that it supports automated deployment and orchestration of services. To achieve this, a number of descriptions are needed for everything that was configured manually in the past, including VNFs and network requirements [3]. Also, Service Level Agreement (SLA)-related parameters such as affinity and anti-affinity rules should be transformed into machine-readable description formats [19]. Huawei mentions the generation of affinity and anti-affinity policies as a mechanism for fault prevention [20]. In the definition of *Service Quality Metrics* by ETSI, special attention is brought to the enforcement of NFV customer anti-affinity rules which can improve the availability mechanisms [21]. The automatically generated affinity rules for VNFs in combination with user-specific affinity requirements could lead to conflicting constraints. In this chapter, a machine-readable format for affinity and anti-affinity constraints is proposed. Furthermore, an automated way is established to detect conflicting constraints based on ontologies. The proposed conflict detection is applicable to both user-generated as well as automatically generated affinity constraint sets. Furthermore, the proposed framework can also be used to validate SFC embeddings with respect to the imposed affinity requirements.

To attain the gains promised by NFV, the VNFs and interconnecting virtual links should be efficiently mapped onto the physical substrate. To achieve this, several placement algorithms have been proposed in the related fields of Virtual Network Embedding (VNE) [6] and Virtual Data Center Network Embedding (VDCNE) [22, 23], as well as for NFV [9]. A placement algorithm can be formulated as an optimization problem with a particular objective such as load balancing, resource utilization, acceptance ratio, etc.

In the context of VNE, *Melo et al.* provide a node-link formulation for optimal VNE and compare their approach to state-of-the-art heuristics in terms of optimality [24]. *Chowdhury et al.* propose PolyViNE, which is a policy-based interdomain VNE framework [25]. The authors define a distributed protocol that coordinates the VNE process across participating InPs. *Dietrich et al.* propose an additional layer of indirection between the SPs and InPs, which is similar to the concept of a VNFInP. The authors study the problem of VNE with limited information disclosure and use a traffic matrix based VNE framework that enables request partitioning with limited information [26]. *Cheng et al.* propose a topology-aware measure on node resources based on random walks and provide an ILP formulation and particle swarm optimization algorithm for the VNE problem [27]. VNE generally involves networks that can be modeled as undirected graphs, which is not the case for SFCs which have strict directions to adhere to. In the field of VDCNE, *Amokrane et al.* propose a holistic resource management framework for embedding virtual data centers across geographically distributed

data centers [28]. *Rabbani et al.* propose an embedding system for data centers that distinguishes multiple resource types on multiple equipments and take into account the relation between switches and links [29]. One of the objectives is to ensure that the infrastructure is as environment-friendly as possible. Most of the data center network virtualization approaches focus on specific network topologies (e.g., clos topology, fat-tree topology) and can therefore not be applied to more general topologies, which is required in a VNF embedding scenario.

*Basta et al.* propose a model for placing virtualized Evolved Packet Core (EPC) functions in a way that minimizes the network overhead introduced by Software Defined Networking (SDN) control plane interactions [30]. *Mehraghdam et al.* apply Mixed Integer Quadratically Constrained Programming (MIQCP) to solve the placement problem and conclude that to obtain efficient use of resources, the placement of functions should be different according to the desired objective [31]. *Beck et al.* propose a coordinated allocation heuristic based on the backtracking concept in which they take into account that the structure of SFCs can be flexible [32]. This allows to reorder some of the VNFs in coordination with the allocation of the SFC. Also *Mehraghdam et al.* take into account such flexible structures [33]. However, they propose a two-step approach without coordination between the SFC composition and embedding. *Moens et al.* propose an ILP-based solution in which hybrid scenarios are considered where part of the functions are provided by dedicated physical hardware and part of them by virtualized instances [34]. *Luizellie et al.* formalize the NF placement and chaining problem and propose an ILP model to solve it [35]. *Xia et al.* propose a Binary Integer Programming (BIP) model for optimal VNF placement subject to minimizing the expensive optical/electronic/optical conversions for NFV chaining in packet/optical data centers [36]. The authors also propose a greedy heuristic which sorts VNFs according to the resource demands and embeds the resource-demanding VNFs with highest priority. Other authors focus on specific NFV use cases such as the virtualized Customer Premises Equipment (vCPE) or virtualized Evolved Packet Core (vEPC). *Addis et al.* define a NFV network model suitable for vCPE operations [37]. The authors take into account two forwarding latency regimes (with and without fastpath), both under Traffic Engineering (TE) and NFV objectives. *Baumgartner et al.* propose an ILP formulation for the cost-optimal embedding of a vCPE which relies on joint embedding of individual core network service chains [38]. *Bouet et al.* focuses on the virtualized DPI placement for network operators [39]. The problem is formulated as a multi-commodity flow problem which is solved as an ILP and takes into account the trade-off between traffic management targets and operational costs. *Sahhaf et al.* propose an algorithm for mapping SFCs to the network infrastructure while taking into account possible decompositions of NFs [40]. The authors view a decomposition as a graph of more refined NFs with the same external interface as the higher-level NF. *Yoshida et al.* propose

a multi-objective resource scheduling algorithm which optimizes simultaneously possibly conflicting objectives with multifaceted constraints [41]. *Vaishnavi et al.* tackle the problem of inter-domain virtual network provisioning by abstracting the resources of a domain to appear as a single node [42]. This allows to re-use existing embedding algorithms with minor modifications. In previous work, the online virtual function mapping and scheduling problem was formulated and a set of algorithms for solving it was proposed [43]. For an extended overview of resource allocation in NFV, the reader is referred to a survey by *Herrera et al.* [44]. None of the aforementioned approaches offers support for attaching affinity or anti-affinity constraints to the SFCs nor do they take into account such constraints when evaluating the embeddings.

This chapter is an extension of previous work [45] in which the affinity and anti-affinity constraints were first proposed. The previous work proposed a first version of the semantic framework for affinity-constrained SFCs validation. In the current chapter, the aforementioned framework was further fine-tuned and extended with embedding algorithms for affinity-constrained SFCs. Furthermore, using the ILP model proposed in this chapter, the semantic filtering is validated. Also the impact of applying semantic filtering prior to the embedding is evaluated in this chapter.

## 6.3 Affinity and Anti-Affinity Constraints

Currently, in an NFV context, SPs have limited control over the mapping of VNFs to physical hosts or SFC edges to physical paths. Nevertheless, many situations can be envisioned where an SP might want to attach constraints to the placement of certain functions or on the routing of traffic, such as:

- **Efficiency:** VNFs that exchange a lot of data may want to be positioned close to one another (e.g., within the same datacenter, or even on the same physical host).

- **Resilience:** The SP might want to spread instances of the same VNF across multiple datacenters in order to improve resilience in case a failure occurs in one of the datacenters.

- **Legislation:** The SP might want to avoid hosting VNFs in certain countries due to legislative restrictions on the location of the data that is processed or transferred.

- **Privacy:** SPs or their customers might not want the traffic to pass through certain domains due to privacy concerns.

- **Economic:** SPs might have economic reasons (e.g., peering agreements) to place their functions in or route their traffic through certain domains.

However, currently there is no way to specify or model such requirements in an SFC template. In this section, the set of affinity and anti-affinity constraints for VNFs and their interconnecting paths are discussed. The affinity constraints apply to a set of physical locations $P$, a set of VNF instances $V$ and a set of edges $E$ interconnecting them. There are different location granularities $g \in G$ that can be considered (e.g., countries, network domains, datacenters), leading to a hierarchical structure of locations. Two hosts in a single datacenter represent different locations at the granularity of hosts, but have the same location at the granularity of datacenters. $P^g \subseteq P$ is the set of locations at a certain granularity $g$. Furthermore, each VNF instance has an associated VNF type (e.g., firewall, DPI), forming subsets $V^t \subseteq V$ of VNFs with type $t \in T$. Finally, each virtual edge $e = (a, b) \in E$ connects two VNFs $a \in V$ and $b \in V$ and maps to a single (or path of) physical network links. This chapter proposes and defines the following affinity and anti-affinity constraints:

- Affinity($p \in P^g, v \in V$ or $t \in T$): A specific instance $v$ or all instances $v \in V^t$ of type $t \in T$ must be located at a specific location $p$ with granularity $g$.
- Anti-Affinity($p \in P^g, v \in V$ or $t \in T$): A specific instance $v$ or all instances $v \in V^t$ of type $t \in T$ must not be located at a specific location $p$ with granularity $g$.
- Affinity($p \in P^g$ or $g \in G, v \in V$ or $s \in T, w \in V$ or $t \in T$): A specific instance $v$ or all instances $v \in V^s$ of type $s \in T$ must be placed together with a specific instance $w$ or all instances $w \in V^t$ of type $t \in T$ at a specific location $p \in P^g$ or at the same location at a specific granularity $g \in G$.
- Anti-Affinity($p \in P^g$ or $g \in G, v \in V$ or $s \in T, w \in V$ or $t \in T$): A specific instance $v$ or any instances $v \in V^s$ of type $s \in T$ must not be placed together with a specific instance $w$ or any instances $w \in V^t$ of type $t \in T$ at a specific location $p \in P^g$ or at the same location at a specific granularity $g \in G$.
- Affinity($p \in P^g, e \in E$): A virtual edge $e \in E$ must be fully embedded at a specific location $p \in P^g$ with a granularity $g \in G$. It does not suffice to model this as location constraints on the endpoints, since physical nodes along the virtual path can still be located in other locations $q \in P^g \setminus \{p\}$.
- Anti-Affinity($p \in P^g, e \in E$): The physical links comprising the virtual edge $e \in E$ must not pass through a specific location $p \in P^g$ with a granularity $g \in G$.

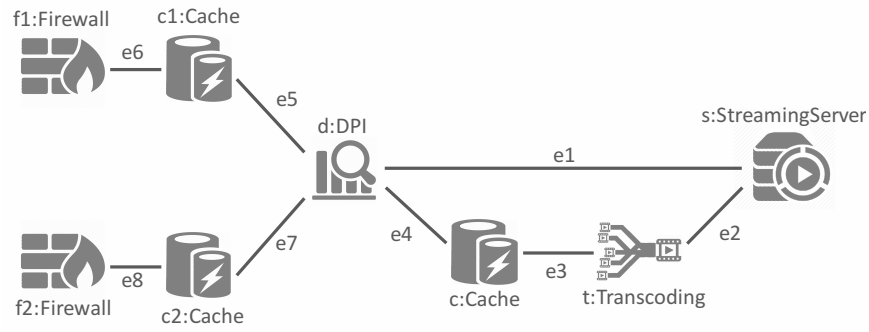



*Figure 6.1: An example SFC.*

- Affinity($e \in E, f \in E$): Two virtual edges $e \in E$ and $f \in E$ must overlap (i.e. all physical links comprising the edge $e$ must be the same as those comprising edge $f$).
- Anti-Affinity($e \in E, f \in E$): Two virtual edges $e \in E$ and $f \in E$ must not overlap (i.e. none of the physical links comprising the virtual edges shall be part of both $e$ and $f$). Modelling this as an anti-affinity constraint for the VNFs of the endpoints of $e$ and $f$ does not achieve this, since the physical paths can still have overlapping links even though the endpoints are mapped to distinct physical nodes.

To further clarify the presented constraint formulations, an example of an SFC request with both affinity and anti-affinity constraints is given here. Given a set of location types {*Autonomous System (AS), Datacenter (DC), Host*} and a set of network function types {*Firewall, DPI, Cache, Transcoding, Streaming Server*}. An example SFC is depicted in Figure 6.1, where a *Streaming Server* is connected via a DPI function (e.g., for tagging data packets) to a content cache. The DPI functions may either directly forward requests to the streaming server or may send them via the *Transcoder* for transcoding and packaging (e.g., for delivery to mobile devices). The transcoding is preceded by a *Cache* in the chain to store the transcoded content. The end-users are connected via a firewall to the service. Suppose a SP wants to offer a Video on Demand (VoD) service in Belgium where two infrastructure providers are active: Telenet (*AS6848*) and Proximus (*AS6774*). Furthermore, the SP wants to deploy part of the service in an Amazon DC identified by *DC_AWS*. Let us consider the following set of affinity and anti-affinity constraints:

- *Affinity(AS6848, c1)*
- *Affinity(AS6774, c2)*

- *Affinity(DC, Transcoder, Streaming Server)*
- *Affinity(DC_AWS, e2)*
- *Affinity(Host, t, c)*
- *Anti-Affinity(e1, e3)*

Specifically, the first two constraints state that the *Caches c1* and *c2* need to be located in the Telenet and Proximus AS respectively (e.g., because they should be close to the end user and limit uplink traffic through other networks). The third constraint states that, for efficiency reasons and for reducing the network traffic, all *Transcoder* functions should be colocated with the *Streaming Server* functions. The next constraint states that the virtual edge *e2* should be fully embedded within the scope of the *DC* identified by *DC_AWS*. The fifth constraint assures that the *Transcoder t* and the *Cache c* should be located at the same *Host*. Finally, the last constraints dictates that none of the physical links that are used for the embedding of the virtual edges *e1* and *e3* are allowed to overlap.

## 6.4 Semantic SFC Request Checker

Since SPs are now free to specify their custom constraints during the SFC request, it is possible that conflicting constraints are introduced. Extending the example from the previous section and adding the constraints *Affinity(Host, c1, f1)* (i.e. specifying that *c1* and *f1* should be colocated in the same Host) and *AntiAffinity(DC, Cache, Firewall)* (i.e. specifying that a VNF of type Cache cannot be colocated with a VNF of type Firewall in the scope of a *DC*) leads to a conflicting constraint set. Also more complex conflicts can appear when multiple constraints are involved in the conflicting set that can only be detected as a conflict when considering the full set. For example, returning to the base example from the previous section and adding the constraints *Affinity(DC, DPI, Cache)* and *Anti-Affinity(DC_AWS, d)* would lead to a conflict set {*Affinity(Host, t, c), Anti-Affinity(DC_AWS, d), Affinity(DC_AWS, e2), Affinity(DC, DPI, Cache)*}. Since *d* and *c* should be colocated in the same *DC* due to the VNF type affinity constraint and *t* and *c* are colocated at the *Host* level, *d* and *t* should be colocated at the *DC* level as well. Furthermore, since *e2* should be fully embedded in *DC_AWS*, *t* should also be located in *DC_AWS*. This means that *d* should be located in *DC_AWS* as well. However, this inferred knowledge conflicts with the defined constraint *Anti-Affinity(DC_AWS, d)*.

When the VNFInP tries to deploy the requested SFC, none of the resulting embedding configurations will lead to a feasible realisation of the SFC request. The VNFInP should however be able to differentiate between a non-acceptance of

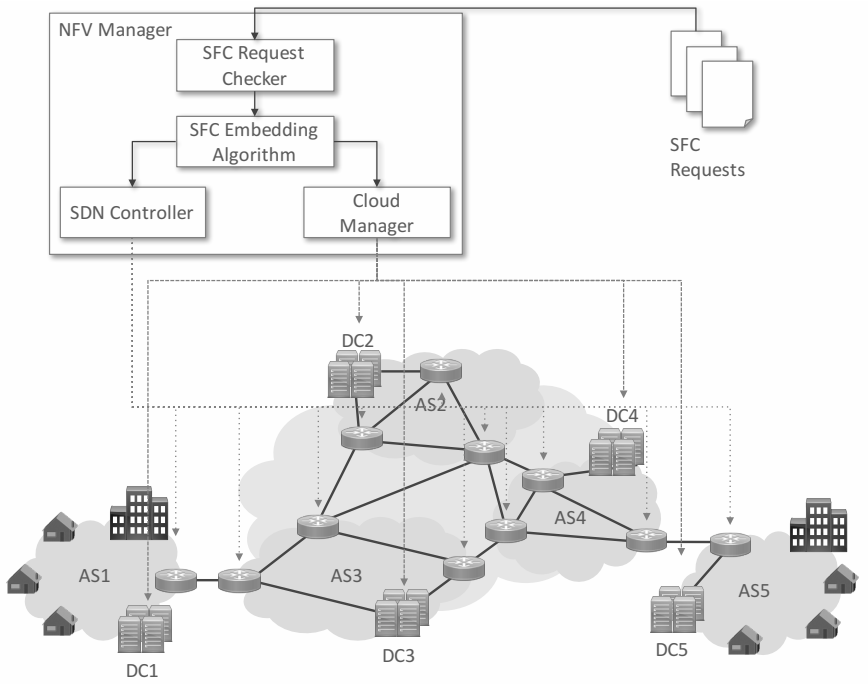



*Figure 6.2: An overview of the NFV architecture with support for semantic SFC request checking.*

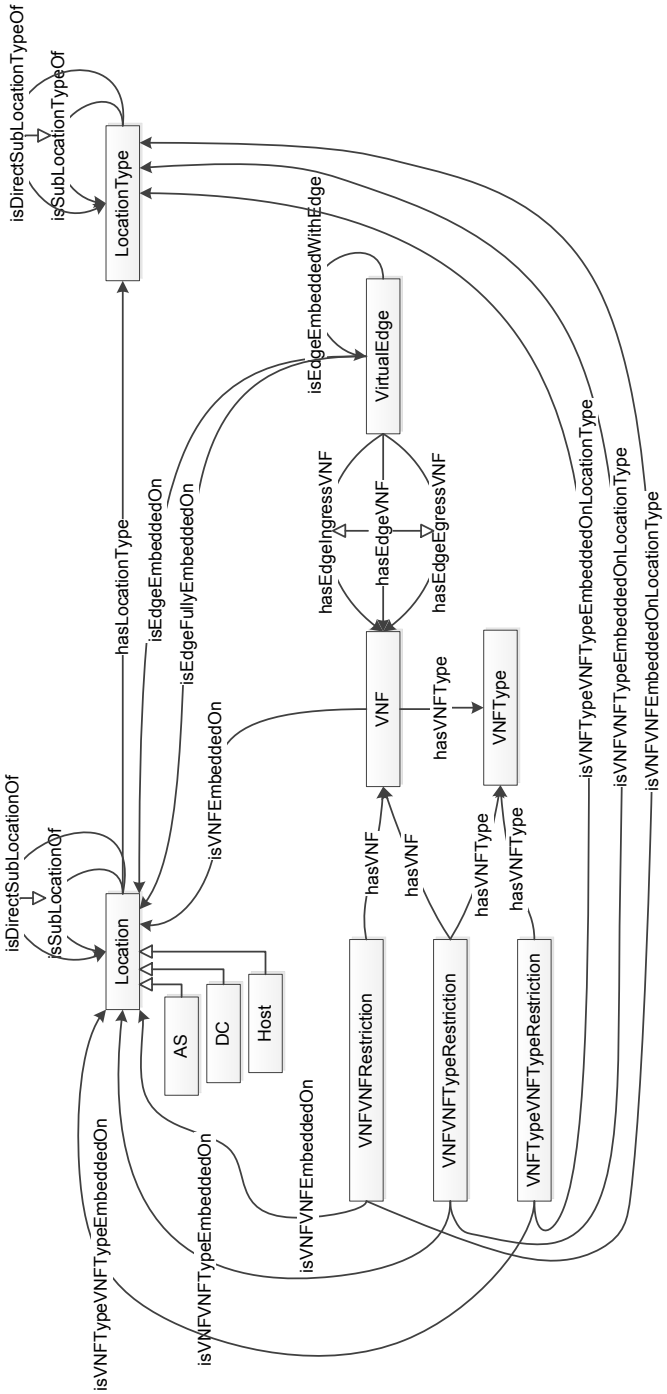



*Figure 6.3: Graphical representation of ontology.*

the SFC request caused by a shortage of appropriate resources and conflicting request constraints, in order to inform the SP on the reason why the SFC deployment failed. The previous example shows the need for the VNFInP to check the validity of an SFC request upon reception in order to exclude any conflicting constraints when trying to provision the requested SFC.

Figure 6.2 depicts how the SFC request checking system could be integrated into the *NFV Manager*. In this architecture, the *SFC Embedding Algorithm* (which will be discussed in the next section) is responsible for assigning resources to the SFC requests. Concretely, it decides on which VNFs should be deployed on which hosts and how many resources should be assigned to them. The *Cloud Manager* performs the management of deployed VNFs and server resources. Moreover, the algorithm selects the forwarding paths interconnecting the VNFs and assigns network resources to them through the *SDN Controller*. Before the SFC request is forwarded to the *SFC Embedding Algorithm* it needs to be checked by the *SFC Request Checker* to confirm the validity.

Appendix C proposes to exploit ontology representations for the purpose of modeling the physical substrate, the SFC requests and defining a set of rules that can be used to infer additional information [45]. Figure 6.3 represents the proposed semantic model. The SFC requests are modeled as a set of *VNFs* with a certain *VNFType* and *VirtualEdges* containing an ingress and egress *VNF*. The physical resources are modeled at the granularity level of *Hosts*, *DCs* and *ASs*. Each of these *Locations* has a certain *LocationType* (i.e., AS, DC or Host). The hierarchical relations between these *Locations* and *LocationTypes* are modeled by *isSubLocationOf* and *isSubLocationTypeOf* respectively. To model affinity (respectively anti-affinity) constraints for single VNFs and edges, positive (respectively negative) object property assertions of the type *isVNFEmbeddedOn*, *isEdgeEmbeddedOn* and *isEdgeFullyEmbeddedOn* are attached to *VNFs* and *VirtualEdges*.

To be able to model more complex affinity and anti-affinity relationships between two *VNFs*, two *VNFTypes* or between a *VNF* and *VNFType*, the additional concepts *VNFVNFRestriction*, *VNFVNFTypeRestriction* and *VNFTypeVNFTypeRestriction* were added to the ontology. By adding the respective positive (respectively negative) property *isVNFVNFEmbeddedOn* or *isVNFVNFEmbeddedOnType*, one is able to model affinity (respectively anti-affinity) restrictions for more complex constraints on the *Location* or *LocationType*. By using the *isEdgeEmbeddedWithEdge* relationship, affinity and anti-affinity constraints between edges can be modeled.

To infer new information out of existing knowledge, a set of rules is defined. For example, Rule (6.1) stipulates that if a certain *VNF x* is embedded on a *Location y* and if *y* is a sublocation of *z*, this *VNF* is also embedded on *Location z*. When a *VNFType y* is embedded on a *Location z*, each *VNF x* of that *VNFType y* needs to be embedded at the *Location z* (Rule (6.2)). If a *VirtualEdge z* contains

a *VNF x* embedded at *Location a*, this *VNF z* is embedded at *Location a* as well (Rule (6.3)). For a full description of the rules, the reader is referred to Appendix C which focuses on the semantic validation of SFC requests.

$$isVNFEmbeddedOn(x, y) \wedge isSubLocOf(y, z) \rightarrow isVNFEmbeddedOn(x, z) \quad (6.1)$$

$$hasVNFType(x, y) \wedge isVNFTypeEmbeddedOn(y, z) \rightarrow isVNFEmbeddedOn(x, z) \quad (6.2)$$

$$hasEdgeVNF(z, x) \wedge isVNFEmbeddedOn(x, a) \rightarrow isEdgeEmbeddedOn(z, a) \quad (6.3)$$

When a new SFC request arrives at the VNFInP, this request is parsed and the set of VNFs and edges are added as individuals to the Web Ontology Language (OWL) ontology. Next, the set of affinity and anti-affinity constraints are also added by either creating new individuals (i.e. *VNFVNFRestriction*), adding property assertions (i.e. *isVNFEmbeddedOn*) or both. For example, the affinity restriction *Affinity(AS6848, c1)* concerning *VNF c1* and *AS AS6848*, is modeled as a positive property assertion *isVNFEmbeddedOn(c1, AS6848)*. On the other hand, the negative restriction *Anti-Affinity(e1, e3)* concerning *VirtualEdge e1* and *VirtualEdge e3*, is modeled as a negative property assertion *isEdgeEmbeddedWithEdge(e1, e3)*.

## 6.5 Model

In this section the affinity-constrained SFC placement problem is described. First, the notations for the inputs and variables used throughout the chapter are presented. Second, the general constraints involved in SFC placement are introduced. Third, the constraints related to affinity and anti-affinity based placement restrictions are explained. Finally, the placement objectives are introduced. Table 6.1 gives an overview of the various notations that are introduced in this section. Figure 6.4 shows a graphical representation of the variables and their relationships.

### 6.5.1 NFV Infrastructure model

The topology of an NFV infrastructure can be described as a weighted directed graph $INF = \{N, L\}$, where $N$ is the set of substrate nodes in the infrastructure and $L$ the set of links interconnecting them. Each substrate node $n \in N$ represents a location where a network function could be deployed. Each substrate node $n$ is characterized by its available processing capacity $C_n$, available memory capacity $M_n$ and its location $p \in P$. This geographic location represents a hierarchical

*Table 6.1: VNF placement problem notation.*

| Symbol | Definition |
|---|---|
| $INF$ | NFV infrastructure |
| $N$ | Set of substrate nodes |
| $n \in N$ | Substrate node |
| $C_n$ | Available processing capacity for node $n$ |
| $M_n$ | Available memory capacity for node $n$ |
| $Q_n$ | Available packet processing capacity for node $n$ |
| $P$ | Set of geographic locations |
| $g \in G$ | Particular granularity for geographic locations |
| $P^g \subseteq P$ | Subset of geographic locations at certain granularity $g$ |
| $N_p^g \subseteq N$ | Subset of nodes at a certain location $p$ with specific granularity $g$ |
| $L$ | Set of substrate links |
| $(m, n) \in L$ | Substrate link where $m, n \in N$ |
| $B_{(m,n)}$ | Available bandwidth of substrate link $(m, n)$ |
| $D_{(m,n)}$ | Network latency of substrate link $(m, n)$ |
| $SFC$ | SFC request |
| $R$ | Set of SFC requests |
| $V_r$ | Set of requested VNFs for SFC request $r \in R$ |
| $v \in V_r$ | VNF |
| $V_r^e \subseteq V_r$ | Set of endpoint-VNFs |
| $T_r$ | Set of VNF types in SFC request $r \in R$ |
| $V_r^t \subseteq V_r$ | Subset of requested VNFs with type $t \in T_r$ |
| $C_v$ | Requested processing capacity for VNF $v$ |
| $M_v$ | Requested memory capacity for VNF $v$ |
| $Q_v$ | Requested packet processing capacity for VNF $v$ |
| $D_v$ | Processing delay incurred by VNF $v$ |
| $E_r$ | Set of edges interconnection the VNFs in SFC request $r \in R$ |
| $(u, v) \in E_r$ | Virtual connection between $u$ and $v$ where $u, v \in V_r$ |
| $B_{(u,v)}$ | Requested bandwidth capacity of virtual link $(u, v)$ |
| $S_r$ | Set of sections in VNF request $r \in R$ |
| $s = \{V_r^s, E_r^s\}$ | Tuple $s \in S_r$ representing a section or path of a VNF request |
| $V_r^s \subseteq V_r$ | Ordered collection of VNFs in a section $s$ |
| $E_r^s \subseteq E_r$ | Ordered collection of virtual links in a section $s$ |
| $D_s$ | Maximum allowed latency associated with a section $s \in S_r$ |
| $A_r$ | Set of affinity and anti-affinity constraints defined in $r \in R$ |
| $A_r^a \subseteq A_r$ | Set of affinity-constraints defined in $r \in R$ |
| $A_r^{aa} \subseteq A_r$ | Set of anti-affinity-constraints defined in $r \in R$ |

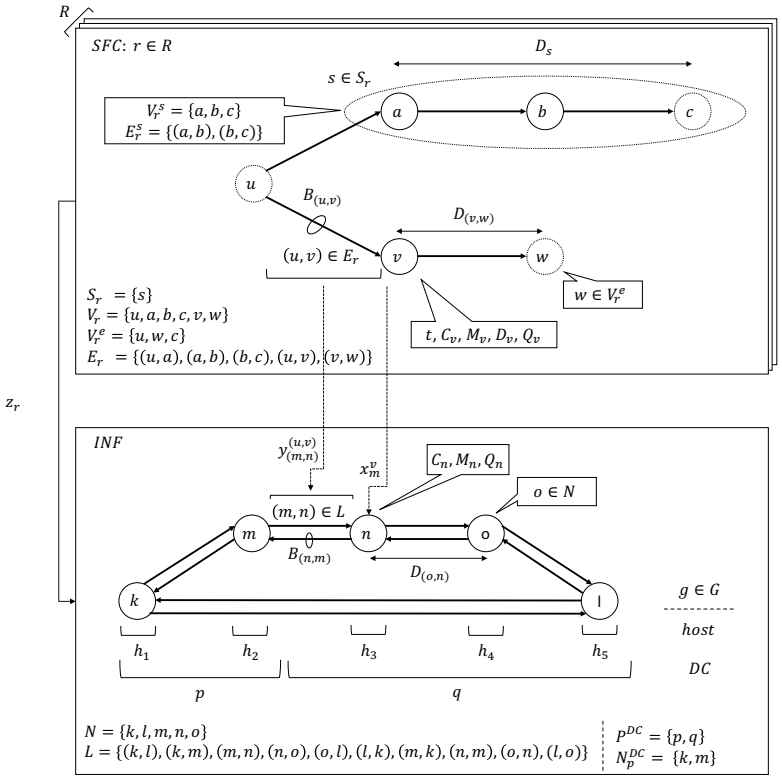



*Figure 6.4: Graphical representation of the model.*

structure where each level of this structure represents a certain geographic granularity (e.g., host, rack, datacenter, network domain, etc.). The subset of geographic locations $P^g \subseteq P$ represents the set of locations at a particular granularity $g \in G$ (e.g., $P^{DC}$ represents the set of all datacenter locations). Similarly, $N_p^g$ represents the set of all substrate nodes at a geographic location $p \in P^g$ at a certain granularity $g$ (e.g., $N_p^{DC}$ represents the set of all substrate nodes located at a datacenter $p$). It is possible that a certain network node $n$ is not involved directly in mapping VNFs for an SFC but is along the path between VNFs and thus participates in forwarding the traffic for the SFC. Therefore, one needs to take into account the packet processing capacity of such a node: $Q_n$ when deploying SFCs on the substrate network.

Each link $(m, n) \in L$ represents a unidirectional link between two network nodes $m, n \in N$. Bidirectional links are represented as a pair of unidirectional links (i.e. $(m, n), (n, m)$). This allows us to model a variety of network links with different up/download characteristics. Each link is characterized by an available bandwidth capacity $B_{(m,n)}$ and latency $D_{(m,n)}$. Similar to network nodes, also the links are characterized by a geographic location which is the set of locations of their endpoints $\{P_m, P_n\}$.

### 6.5.2 SFC model

Similarly to the VNF infrastructure model, an SFC request $r \in R$ can be described as a weighted directed graph $SFC_r = \{V_r, E_r\}$, where $V_r$ is the set of VNFs and $E_r$ the set of edges interconnecting them. An SFC typically has two or multiple endpoint VNFs which terminate the service chain. These are modeled as endpoint-VNFs $V_r^e \subseteq V_r$. $T_r$ represents the set of VNF types (e.g., firewall, gateway, router). Each VNF $v \in V_r$ has an associated VNF type $t \in T_r$. $V_r^t$ forms the set of VNFs in the SFC that have a type $t$ associated with them. Each VNF $v$ requires a certain processing capacity $C_v$ and a specific memory capacity $M_v$. Furthermore, each VNF $v$ has an associated processing delay $D_v$.

Each edge $(u, v) \in E_r$ represents a unidirectional virtual edge between two VNFs $u, v \in V_r$. These are modeled as unidirectional connections since SFCs can have different capacity requirements for up-and downstream respectively (e.g., a video streaming SFC will have higher downstream requirements). An SFC edge can be composed of one or more substrate links or could be an internal link when both $u$ and $v$ are mapped onto the same substrate node $n$. Link aggregation, where a virtual edge is composed out of multiple distinct underlying paths, is not considered here. Each edge has a required capacity $B_{(u,v)}$ between the VNFs, this capacity is also used to account for the packet processing requirements of the forwarding nodes (i.e. $Q_n$). A section or path $s \in S_r$ is a tuple composed of a set of ordered VNFs $V_r^s$ and the ordered set of consecutive virtual edges interconnecting

these VNFs $E_r^s$. An SFC request $r$ can associate a maximum allowed latency $D_s$ for the end-to-end flow processing of such sections. $A_r$ denotes the affinity and anti-affinity constraints that are defined in the SFC request $r$. $A_r^a$ and $A_r^{aa}$ denote the set of affinity constraint and anti-affinity constraints respectively.

### 6.5.3 Assignment variables

For every SFC request $r \in R$, a set of node and link mapping variables is defined. The binary variable $z_r$ defines whether the SFC $r \in R$ is mapped or not as defined in Expression (6.4). The binary variable $x$ is used in the mapping of the VNFs to network nodes and is defined in Expression (6.5), where $x_n^v \rightarrow N \cdot V_r$. Similarly, the binary variable $y$ is used as defined in Expression (6.6), where $y_{(m,n)}^{(u,v)} \rightarrow L \cdot E_r$.

$$z_r = \begin{cases} 1, & \text{if } r \in R \text{ is mapped.} \\ 0, & \text{otherwise.} \end{cases} \tag{6.4}$$

$$\forall r \in R : x_n^v = \begin{cases} 1, & \text{if } v \in V_r \text{ is mapped to } n \in N. \\ 0, & \text{otherwise.} \end{cases} \tag{6.5}$$

$$\forall r \in R : y_{(m,n)}^{(u,v)} = \begin{cases} 1, & \text{if } (u,v) \in E_r \text{ is} \\ & \text{mapped to } (m,n) \in L. \\ 0, & \text{otherwise.} \end{cases} \tag{6.6}$$

### 6.5.4 General constraints

In order to assure a correct mapping of the requested SFC, a set of constraints can be defined. Equations (6.7) and (6.8) prevent that VNFs and virtual edges respectively are mapped when the corresponding SFC to which they belong is not mapped. Equation (6.9) ensures that if an SFC request $r$ is mapped ($z_r = 1$), each VNF is assigned and that it is assigned to exactly one node. On the other hand, if the SFC request $r$ is not mapped ($z_r = 0$), none of the VNFs should be mapped. Equations (6.10) and (6.11) respectively assure that the available CPU and memory capacity of each individual node is not exceeded by the capacity requirements of all VNFs $v \in V$ in the request set $R$ ($V = \bigcup_{r \in R} V_r$).

$$\forall r \in R : \forall v \in V_r : \forall n \in N : x_n^v \leq z_r \tag{6.7}$$

$$\forall r \in R : \forall (u,v) \in E_r : \forall (m,n) \in L : y_{(m,n)}^{(u,v)} \leq z_r \tag{6.8}$$

$$\forall r \in R : \forall v \in V_r : \sum_{n \in N} x_n^v = z_r \tag{6.9}$$

$$\forall n \in N : \sum_{v \in V} (x_n^v \cdot C_v) \leq C_n \tag{6.10}$$

$$\forall n \in N : \sum_{v \in V} (x_n^v \cdot M_v) \leq M_n \tag{6.11}$$

$$\forall (u,v) \in E, m \in N : \sum_{n \in N} y_{(m,n)}^{(u,v)} - \sum_{n \in N} y_{(n,m)}^{(u,v)} = x_m^u - x_m^v \tag{6.12}$$

$$\forall (m,n) \in L : \sum_{(u,v) \in E} \left( y_{(m,n)}^{(u,v)} \cdot B_{(u,v)} \right) \leq B_{(m,n)} \tag{6.13}$$

$$\forall n \in N : \sum_{u \in V} (x_n^u \cdot Q_u) + \sum_{(u,v) \in E} \left(2 \cdot x_n^u \cdot x_n^v \cdot B_{(u,v)}\right) + \sum_{(u,v) \in E} \sum_{\substack{(m,n) \in L \\ (n,o) \in L}} \left( \left( y_{(m,n)}^{(u,v)} + y_{(n,o)}^{(u,v)} \right) \cdot B_{(u,v)} \right) \leq Q_n \tag{6.14}$$

$$\forall r \in R : \forall s \in S_r : \sum_{u \in V_r^s} \sum_{n \in N} (x_n^u \cdot D_u) + \sum_{(u,v) \in E_r^s} \sum_{(m,n) \in L} \left( y_{(m,n)}^{(u,v)} \cdot D_{(m,n)} \right) \leq D_s \tag{6.15}$$

The multi-commodity flow conservation is assured by Equation (6.12). This ensures that there exist virtual paths between the requested VNFs. Modeling that no capacity is required for a certain virtual edge $(u, v)$, while a virtual edge $(v, u)$ exists with non-zero capacity requirements can be achieved by setting $B_{(u,v)} = 0$ and $B_{(v,u)} > 0$ respectively. Using the constraint defined in Equation (6.13), it is ensured that each substrate link has enough capacity to serve all virtual paths $(u, v) \in R$ mapped on it for a request set $R$ ($E = \bigcup_{r \in R} E_r$). Equation (6.14) guarantees that the required processing capacity is available for each of the substrate nodes. The first term sums up the required capacity $B_u$ by each VNF $u \in V$. The second term assures that if a virtual edge is mapped internally within a single substrate node, the required processing capacity for this edge is added. The last term sums up the processing capacity of each ingress and egress links of a substrate node. Finally, Equation (6.15) ensures that the end-to-end latency incurred by the substrate links comprising the virtual paths $s \in S_r$ and the processing delays at the

VNFs adhere to the maximum allowed latency $D_s$ specified by the SFC request $r \in R$.

### 6.5.5 Affinity and Anti-Affinity constraints

To take into account the required geographic and colocation requirements for a specific VNF request $r$, these are modeled as constraints for the optimization problem. Equation (6.16) stipulates that all VNFs of a certain type $t$ need to be located at a location $p$ with granularity $g$, while Equation (6.17) prevents the embedding of any VNF of type $t$ at this location $p$. To model the affinity of certain VNF types $s$ and $t$ at a any location with granularity $g$, Equation (6.18) guarantees that every VNF $v$ of type $s$ is embedded with all VNFs of type $t$ (=$|V^t|$) at a single location $p$ with granularity $g$. Equation (6.19) prevents a VNF of type $s$ to be embedded with a VNF of type $t$ at the same location $p$ with granularity $g$. Similar constraints were added for the other affinity and anti-affinity constraints involving VNFs and VNF types defined in Section 6.3.

Next to VNF affinity constraints, link affinity constraints are also modeled. Equation (6.20) assures that every substrate link $(m, n)$ on which a virtual edge $(v, w)$ is mapped, is located at a location $p \in P^g$. Furthermore, since edges can be mapped internally (i.e. both VNFs $v$ and $w$ are mapped onto the same substrate node), Equation (6.21) ensures that also the VNFs $v$ and $w$ are mapped onto a host at that location $n \in N_p^g$. To prevent the mapping virtual edges $(u, v)$ at a certain location $p \in P^g$, Equation (6.22) ensures that none of the substrate links $(m, n)$ for which one of the endpoints is located at that location ($m \vee n \in N_p^g$) is used. Similarly, Equation (6.23 prevents the internal embedding of the virtual edge $(v, w)$ at that location.

Equation (6.25) stipulates that two virtual edges need to be mapped onto the same virtual path. Equation (6.24) ensures that the corresponding endpoints of the virtual edge are mapped onto the same host. This constraint takes care of the cases where the virtual edges are mapped internally. Equation (6.26) prevents two virtual edge mappings of sharing a link. Furthermore, none of the endpoints of these respective virtual edges should be colocated at the same host (Equation (6.27))

$$\begin{aligned} &\text{Affinity}(p \in P^g, t \in T_r) : \\ &\forall v \in V_r^t : \sum_{n \in N_p^g} x_n^v = z_r \end{aligned} \tag{6.16}$$

$$\begin{aligned} &\text{Anti-Affinity}(p \in P^g, t \in T_r) : \\ &\forall v \in V_r^t : \sum_{n \in N_p^g} x_n^v = 0 \end{aligned} \tag{6.17}$$

$$\text{Affinity}(g \in G, s \in T_r, t \in T_r):$$
$$\forall v \in V_r^s : \forall p \in P^g : \sum_{n \in N_p^g} \left(x_n^v \cdot \left|V_r^t\right|\right) = \sum_{w \in V_r^t} \sum_{n \in N_p^g} x_n^w \tag{6.18}$$

$$\text{Anti-Affinity}(g \in G, s \in T_r, t \in T_r):$$
$$\sum_{p \in P^g} \left( \sum_{n \in N_p^g} \sum_{v \in V_r^s} x_n^v \cdot \left( \sum_{n \in N_p^g} \sum_{w \in V_r^s} x_n^w \right) \right) = 0 \tag{6.19}$$

$$\text{Affinity}(p \in P^g, e = (v, w) \in E_r):$$
$$\sum_{\substack{(m,n) \in L \\ m \wedge n \in N_p^g}} y_{(m,n)}^{(v,w)} \geq \sum_{(m,n) \in L} y_{(m,n)}^{(v,w)} \tag{6.20}$$
$$\left( \sum_{n \in N_p^g} x_n^v = z_r \right) \wedge \left( \sum_{n \in N_p^g} x_n^w = z_r \right) \tag{6.21}$$

$$\text{Anti-Affinity}(p \in P^g, e = (v, w) \in E_r):$$
$$\sum_{\substack{(m,n) \in L \\ m \vee n \in N_p^g}} y_{(m,n)}^{(v,w)} = 0 \tag{6.22}$$
$$\left( \sum_{n \in N_p^g} x_n^v = 0 \right) \wedge \left( \sum_{n \in N_p^g} x_n^w = 0 \right) \tag{6.23}$$

$$\text{Affinity}(e = (v, w) \in E_r, f = (u, z) \in E_r):$$
$$\left( \sum_{n \in N} \left(x_n^v \cdot x_n^u\right) = z_r \right) \wedge \left( \sum_{n \in N} \left(x_n^w \cdot x_n^z\right) = z_r \right) \tag{6.24}$$
$$\begin{gathered} \sum_{(m,n) \in L} \left( y_{(m,n)}^{(v,w)} \cdot y_{(m,n)}^{(u,z)} \right) \geq \sum_{(m,n) \in L} y_{(m,n)}^{(v,w)} \\ \sum_{(m,n) \in L} \left( y_{(m,n)}^{(v,w)} \cdot y_{(m,n)}^{(u,z)} \right) \geq \sum_{(m,n) \in L} y_{(m,n)}^{(u,z)} \end{gathered} \tag{6.25}$$

$$\text{Anti-Affinity}(e = (v, w) \in E_r, f = (u, z) \in E_r) : \sum_{(m,n) \in L} \left( y^{(v,w)}_{(m,n)} \cdot y^{(u,z)}_{(m,n)} \right) = 0 \tag{6.26}$$

$$\begin{gathered} \left( \sum_{n \in N} (x^v_n \cdot x^u_n) = 0 \right) \wedge \left( \sum_{n \in N} (x^w_n \cdot x^z_n) = 0 \right) \\ \left( \sum_{n \in N} (x^v_n \cdot x^z_n) = 0 \right) \wedge \left( \sum_{n \in N} (x^w_n \cdot x^u_n) = 0 \right) \end{gathered} \tag{6.27}$$

### 6.5.6 Objective functions

When embedding SFC request set $R$, there are multiple objectives that could be considered. To maximize the revenue of the VNFInP, the number of accepted SFC requests should be maximized as shown in Equation (6.28). On the other hand, to efficiently manage the infrastructure resources, the InP could have other objectives, such as minimizing the number of substrate nodes and links that are used, minimizing the overall traffic or load balancing the traffic over the full infrastructure. To achieve this, the problem is first solved by using the acceptance maximization as an objective function. Afterwards, the solution $Z_{sol}$, which characterizes the number of embedded SFC requests is added as an additional constraint by Equation (6.29).

$$\max \sum_{r \in R} z_r \tag{6.28}$$

$$\sum_{r \in R} z_r \geq Z_{sol} \tag{6.29}$$

Afterwards the adapted optimization problem is solved again, this time with the InP-specified objective. Equation (6.30) minimizes the number of substrate nodes $n \in N$ that are used for embedding the requests. The total bandwidth consumption is minimized by the objective defined in Equation (6.31). Similar objectives can be defined for other resource types as well.

$$\min \sum_{r \in R} \sum_{v \in V_r} \sum_{n \in N} x^v_n \tag{6.30}$$

$$\min \sum_{r \in R} \sum_{(u,v) \in E_r} \sum_{(m,n) \in L} \left( y^{(u,v)}_{(m,n)} \cdot B_{(u,v)} \right) \tag{6.31}$$

To balance the bandwidth consumption over the full infrastructure, the difference between the maximum and minimum load over all links is minimized

as shown in Equation (6.32). To model this, two additional continuous decision variables $Load_{min} \in [0.0, 1.0]$ and $Load_{max} \in [0.0, 1.0]$ are added, as well as constraints stating that the load of each link should be smaller, respectively larger, than the maximum and minimum load. The objective is then to minimize the difference $Load_{max} - Load_{min}$. The same could be achieved with other resources than bandwidth as well.

$$\begin{aligned} \min \Bigg( & \max_{(m,n) \in L} \sum_{r \in R} \sum_{(u,v) \in E_r} \left( \frac{y^{(u,v)}_{(m,n)} \cdot B_{(u,v)}}{B_{(m,n)}} \right) \\ & - \min_{(m,n) \in L} \sum_{r \in R} \sum_{(u,v) \in E_r} \left( \frac{y^{(u,v)}_{(m,n)} \cdot B_{(u,v)}}{B_{(m,n)}} \right) \Bigg) \end{aligned} \tag{6.32}$$

## 6.6 Heuristic Approach

Solving the SFC embedding problem can be computationally expensive, as will be shown later on during the evaluations. Therefore, a heuristic approach is proposed in which the SFCs are ordered according to certain criteria and the embedding algorithm proceeds to map them individually to the infrastructure taking into account different objective functions. The ILP model that was presented before was adapted to be able to support such individual mapping of ordered SFC sets. To this end, the decision variable $z_r$, indicating if SFC request $r \in R$ should be mapped is left out of the model. This ensures that the optimization process tries to embed the request $r$ if there exists a feasible solution. Similar objectives can be defined as before, however also the resource usage incurred by previous individual embeddings should now be taken into account when evaluating the objective functions.

The ordering of the SFCs in the set can be achieved in many ways. A first ordering criterion could be to order the SFCs based on the number of affinity constraints that they contain, resulting in the set $R_{nc}$ (Equation (6.33)). Another option is to order the SFC requests based on the requested bandwidth resources as shown in Equation (6.34).

$$R_{nc} : \left\{ r \in R, i \leq j : |A_{r_i}| \leq \left|A_{r_j}\right| \right\} \tag{6.33}$$

$$R_{bw} : \left\{ r \in R, i \leq j : \sum_{(u,v) \in E_i} B_{(u,v)} \leq \sum_{(u,v) \in E_j} B_{(u,v)} \right\} \tag{6.34}$$

## 6.7 Evaluation

In this section, the simulation framework that was developed to evaluate the affinity-constrained SFC embedding is discussed. The implementation details of the semantic validation module are discussed, as well as the implementation of the mathematical model. Furthermore, the generation of substrate topologies and SFCs is discussed. The first set of experiments evaluates the impact of the infrastructure size, the requested SFC size and the number of affinity and anti-affinity constraints on the scalability of the semantic validation. Afterwards, the performance gain of the semantic validation on the SFC embedding is discussed.

### 6.7.1 Simulation framework

The simulation framework is implemented in Java 8[3] and allows to generate topologies, generate random SFCs, validate these SFCs and map them onto the substrate. The mathematical models presented in the previous sections are implemented using CPLEX 12.6[4]. The topologies are generated using the BRITE topology generator[5]. The ASs and their interconnections are generated using BRITE after which a number of DCs are added to each of the ASs. The DC topologies that are generated for the evaluations are two-level fat-tree topologies [46].

The SFCs and their respective constraints are generated randomly. First a set of VNFs and interconnecting virtual edges is generated. The respective capacity constraints for VNFs and edges are uniformly distributed between the configured ranges. Furthermore, end-to-end segments are assigned with a maximum delay. Second, random affinity/anti-affinity constraints are added and the required VNFs, virtual edges and VNF types are randomly selected from the ones that are present in the SFC. When generating these constraints, it is checked whether the same restriction or its counterpart (affinity or anti-affinity restriction with the same parameters) is not already present in the set. Finally, the locations are randomly selected from all available locations and added to the constraints. To select these locations, a discrete probability distribution with the following probability mass function is used for affinity restrictions (AS: $0.6$, DC: $0.3$, host: $0.1$) and anti-affinity restrictions (AS: $0.1$, DC: $0.3$, host: $0.6$). The rationale behind this is that affinity constraints apply to more general location restrictions while for anti-affinity constraints, more granular specification of locations apply. The type of the constraints is uniformly distributed among the constraints defined in Section 6.3. Table 6.2 lists the set ($[x, y]$) or ranges ($[x - y]$) for the various parameters that are set during the evaluations.

[3] Java 8 - https://java.com/en/download/faq/java8.xml
[4] IBM CPLEX - http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
[5] BRITE - www.cs.bu.edu/brite/

*Table 6.2: Scenario parameters.*

| Type | Parameter | Range |
|---|---|---|
| BRITE Topology | #AS | [2,3,4,5,8,16,32,64,128,256] |
| | #Neighbour-AS | [2] |
| | #AS-Routers | [8,16,32] |
| | #Neighbour-Routers | [3] |
| | #DC | [1,2,4] |
| | Intra-AS BW | [5Gbps ... 50Gbps] |
| | Inter-AS BW | [1Gbps ... 10Gbps] |
| | Intra-AS delay | [5ms ... 10ms] |
| | Inter-AS delay | [1ms ... 5ms] |
| Fat-tree Topology | #Core switches | [2,3] |
| | #Pods | [1 ... 4] |
| | #Servers per pod | [5 ... 10] |
| | Link BW | [5Mbps ... 10Mbps] |
| | Link delay | [1ms] |
| SFC | #VNF | [2,5,10,20,30,40] |
| | #Affinity constraints | [5,10,20,40,60,80] |
| | Link BW | [50Mbps ... 500Mbps] |
| | Segment delay | [5ms ... 100ms] |
| | Processing delay | [0ms ... 5ms] |

The Protégé editor[6] was used to develop the SFC request modelling ontology using the OWL API[7]. Semantic Web Rule Language (SWRL)[8] was used to express the aforementioned rules using concepts from the ontology defined in Section 6.4. The Protégé editor was also used to define the rules using the Manchester syntax[9]. The HermiT OWL Reasoner[10] was used to check the consistency and the classification of the ontology. HermiT is a semantic reasoner for ontologies written in OWL. It is able to determine whether or not the ontology is consistent, identify subsumption relationships between classes, etc. The reasoner is based on a hypertableau calculus which provides efficient reasoning. The output of the reasoning process allows us to determine whether the SFC request at hand is valid or not. In the case of an invalid request, this is communicated to the requesting SP, otherwise the request is passed on to the embedding engine.

The evaluations were carried out using the STEVIN Supercomputer Infrastructure at Ghent University[11]. The nodes are equipped with 2 Intel Xeon CPU E5-

[6]Protégé - http://protege.stanford.edu/
[7]OWL2 - http://www.w3.org/TR/owl-features/
[8]SWRL - http://www.w3.org/Submission/SWRL/
[9]Manchester Syntax - http://www.w3.org/2007/OWL/wiki/ManchesterSyntax
[10]HermiT OWL Reasoner - http://hermit-reasoner.com
[11]HPC UGent - http://www.ugent.be/hpc

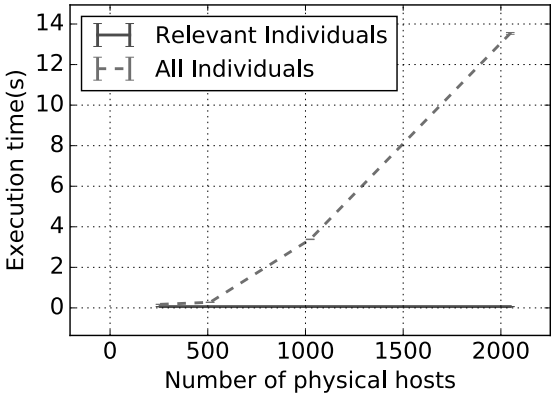



*Figure 6.5: Impact of the infrastructure size on the semantic validation.*

2680 v3 12-core processors and 32GB of physical memory. For the experiments, a single 2.5GHz core and 16GB of memory were requested. All evaluations were repeated 20 times with varying seed values, the graphs show the average values and $95\%$ confidence intervals.

Since none of the existing NFV resource allocation techniques take into account affinity constraints, it is impossible to compare the proposed approach to a baseline solution. Therefore, the mathematical model computing the optimal solution is taken as a benchmark to evaluate the semantic validation and the heuristic approach.

### 6.7.2 Scalability of semantic SFC validation

Semantic reasoning times are known to increase exponentially with the number of individuals in the ontology. In Section 6.4, a semantic SFC request validation framework was discussed which loads both the substrate topology and requested SFC into an ontology. Using semantic reasoning, the consistency of the SFC request can be validated. Figure 6.5 shows the impact of the infrastructure size on the semantic reasoning process and indicates an exponential increase of the reasoning time with the size of the infrastructure. However, as mentioned before, the reasoning times can be significantly reduced when only including the relevant parts of the infrastructure in the ontology. To determine these relevant parts, all physical locations of the substrate topology that are included in the SFC constraints are modeled. Furthermore, also the parent locations are subsequently modeled. This set is typically much smaller than the complete infrastructure and can be considered constant for SFCs of the same size. When only loading relevant individuals, execution times show a constant trend at about $62ms$, even when the infrastructure size increases. These experiments use a fixed number of $10$ VNFs per SFC and $5$

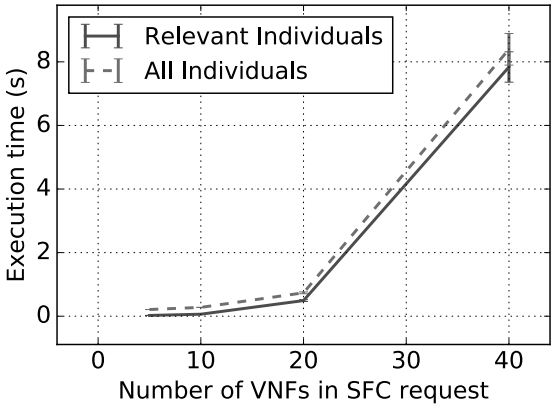



*Figure 6.6: Impact of the number of requested VNFs on the semantic validation.*

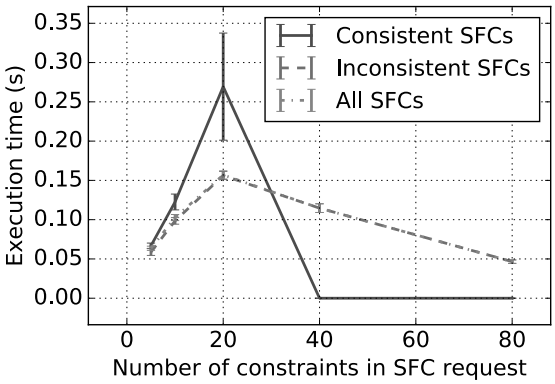



*Figure 6.7: Impact of the number of constraints per SFC on the semantic validation.*

randomly generated affinity and anti-affinity restrictions.

Figure 6.6 shows the impact of increasing the requested SFC size on the semantic validation time. As can be expected, the execution time increases exponentially with an increasing number of VNFs per SFC. Validating SFC requests with 20 VNFs takes about 500 ms. Considering that a standard Hermit reasoner was used without optimizations on standard off-the-shelve hardware, this could be considered a reasonable processing overhead. As can be seen from the graph, only including the relevant substrate nodes has only a limited impact in this configuration, since substrate topologies are considered with 512 nodes and 5 affinity restrictions per SFC.

To evaluate the impact of the number of affinity constraints on the semantic validation time, substrate topologies with 512 nodes were created and SFC requests

with 10 VNFs instances were used. The number of randomly generated affinity constraints was varied from 5 to 80. With an increasing number of constraints, the probability of inconsistencies in the SFC requests increases. Therefore, the average validation times for both groups of inconsistent and consistent SFC requests are shown separately. Figure 6.7 shows that the average validation times increase up to a certain point, after which they drop again. This behavior can be accounted to the fact that the probability of easy-to-detect conflicts (e.g., two contradictory constraints) increases when the number of constraints increases, quickly terminating the reasoning process. When the number of constraints is lower, the variety of constraint types and subjects on which they are posing these constraints is larger, leading to more complex conflicts that only appear when more information is inferred from the ontology, causing the validation to take more time. None of the SFCs with over 40 constraints is consistent, leading to an execution time of $0s$ for this subgroup.

### 6.7.3 Impact of semantic validation on mapping time

In this set of experiments, the impact on the mapping time when performing semantic validation on SFC sets prior to running the embedding algorithm, is evaluated. Furthermore, the share of the semantic validation step on the total execution time is evaluated. To this end, SFC request sets are generated where a certain percentage of SFCs has conflicting affinity constraints. The substrate topologies under consideration contain 512 nodes, while the request sets contain 20 SFCs, of which each SFC contains 10 VNFs and 5 affinity restrictions. During this set of evaluations, the bandwidth minimization objective defined in Equation (6.31) is used.

Figure 6.8 shows the positive impact on the total mapping time per SFC request when semantically filtering the inconsistent SFC requests out of the request set prior to performing the mapping step. The SFC requests are mapped as a set and contain 20 SFC requests per set. By validating the SFC request set prior to mapping it, the total execution time can be reduced by 59% on average. Even when only 10% of the SFC requests are inconsistent, the execution times can be reduced with more than 25%. The relative execution time reduction approximately shows a logarithmic trend between 0% and 100% with an increasing number of inconsistent requests. These results show that it is beneficial to perform the semantic matching although it yields a small additional execution overhead when no inconsistent SFCs are present.

Figure 6.9 shows the same graph, but for the individual mapping of 100 SFC requests. On average, semantically filtering the request set reduces the mapping time with 50%. When no inconsistent SFC requests are present, there is an overhead incurred by the semantic matching step of 0.4%, which can be considered

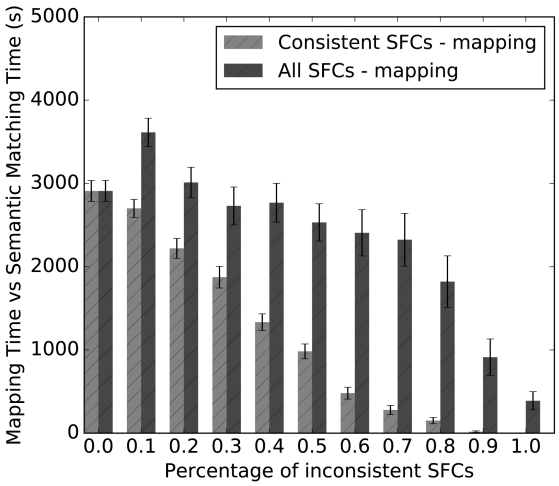



*Figure 6.8: Share of semantic matching and mapping on total execution time for SFC set mapping.*

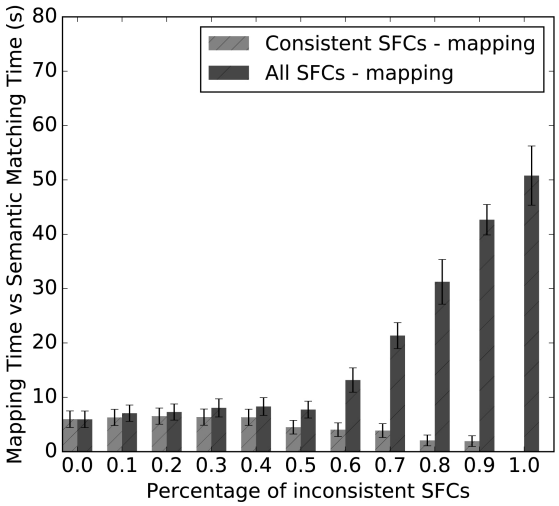



*Figure 6.9: Share of semantic matching and mapping on total execution time for individual SFC mapping.*

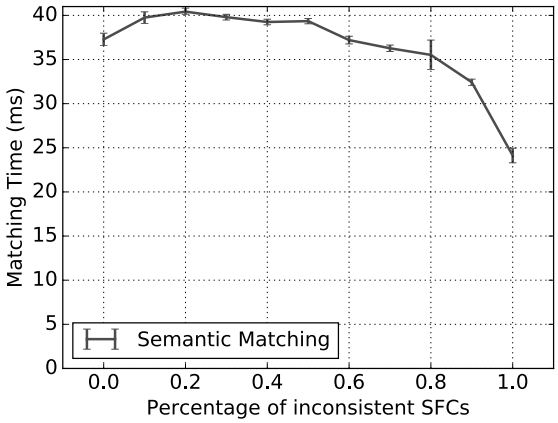



*Figure 6.10: Semantic matching execution time.*

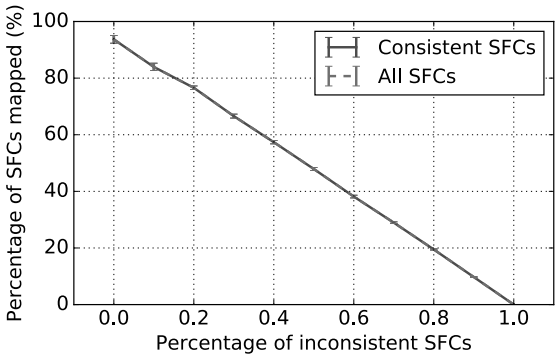



*Figure 6.11: Percentage of SFC requests that are mapped.*

negligible compared to the total mapping time. For a $10\%$ fraction of inconsistent SFCs per request set, the performance gain is about $11\%$.

The previous graphs shows both the mapping and matching times. Figure 6.10 shows the matching time per SFC separately, note that the y-axis of this graph shows the time in $ms$ instead of $s$. As can be seen from the graph, the semantic filtering takes about $40ms$ but yields a benefit which is $28000$ times greater in the case of set mapping and $355$ times greater for individual request mapping.

To assess the validity of the proposed semantic SFC validation framework, also the number of mapped SFC requests was tracked when filtering out inconsistent SFCs, as well as when the complete SFC request set is considered for mapping. Figure 6.11 shows that every inconsistent SFC that was filtered out during the semantic matching process, was indeed impossible to map during the embedding

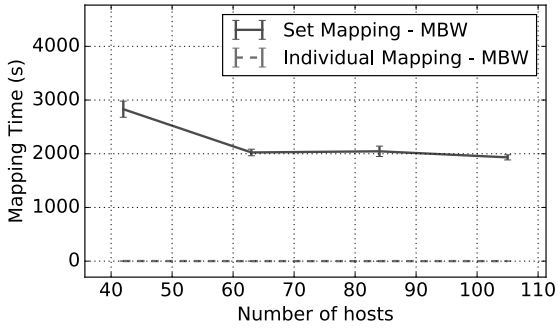



*Figure 6.12: Total mapping time of Set mapping compared to Individual mapping for increasing infrastructure size.*

phase due to inconsistencies in the request.

### 6.7.4 Performance of heuristic approach

Mapping the SFC request sets in a single step yields an increase in total execution time of the mapping process due to an increasing number of decision variables and constraints in the mathematical model. To alleviate these problems, a heuristic mapping procedure is proposed in which the different SFCs in the set are first ordered using a certain criterion, after which they are individually mapped onto the infrastructure. Figure 6.12 shows the impact of performing the heuristic procedure on the total execution time. During the experiments, both approaches are using the bandwidth minimization objective defined in Equation (6.31), the heuristic approach orders the SFCs according to the requested bandwidth criterion defined in Equation (6.34). The total execution time can be reduced with a factor $3378$ on average when performing the heuristic approach. The evaluations were performed using SFC request sets containing $20$ SFCs and each SFC containing on average $5$ VNFs and $3$ affinity constraints. Mapping the SFC requests individually takes on average $0.85s$, including the semantic matching.

The heuristic approach comes at a cost in optimality, since each request is considered individually, the global optimum is not achieved. Figure 6.13 shows the allocated bandwidth as a percentage of the total bandwidth available in the infrastructure when minimizing the bandwidth usage. On average, the heuristic approach allocates $2.6\%$ more bandwidth than the optimal solution. Also in terms of number of mapped SFC requests, the heuristic approach is outperformed by the optimal set mapping approach. Figure 6.14 shows that in some cases only $97.5\%$ of the feasible SFC requests are mapped when applying the heuristic approach. This also implies that the resource consumption for the heuristic approach would be higher if the same amount of requests could be mapped.

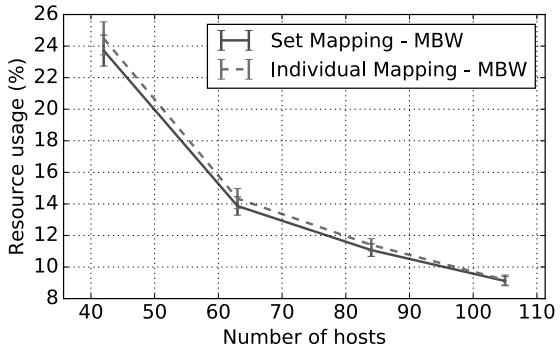



*Figure 6.13: Objective of Set mapping compared to Individual mapping for increasing infrastructure sizes.*

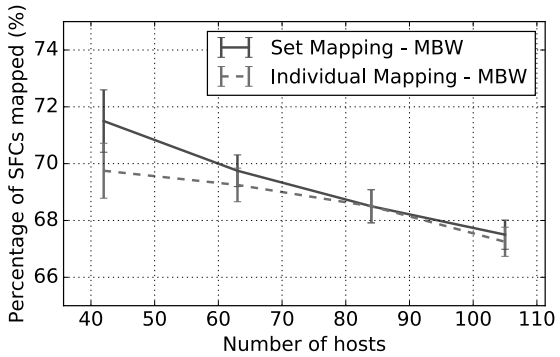



*Figure 6.14: Percentage of mapped SFCs when applying Set mapping compared to Individual mapping for increasing infrastructure sizes.*

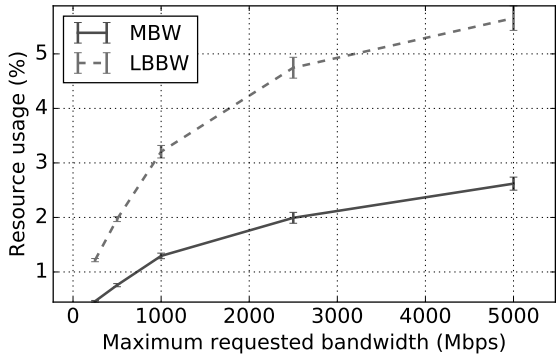



*Figure 6.15: Total link bandwidth usage for various optimization objectives.*

### 6.7.5 Impact of optimization objective

Multiple optimization objectives were proposed in Section 6.5. The bandwidth minimization objective (MBW) proposed in Equation (6.31) is compared with the load balancing objective (LBBW) of Equation (6.32). To compare the objectives, the maximum requested bandwidth for the virtual edges is varied. A set of $40$ consistent SFC requests is generated and mapped onto the infrastructure resources. Figure 6.15 shows the impact of the objectives on the total resource usage. It is obvious that the LBBW optimization is outperformed in terms of total resource usage by MBW. Looking at the difference between the maximum and minimum load, shown in Figure 6.16, it is clear that the proposed balancing objective allows a better spread of the load compared to pure bandwidth optimization. Figure 6.17 shows the percentage of nodes that is used for the mapping, showing a higher node usage for LBBW. This graph also confirms that the load is spread across the infrastructure. The decrease in node usage with increasing bandwidth can be attributed to a reduced number of SFCs that can be mapped on the infrastructure due to capacity constraints.

### 6.7.6 Impact of ordering criterion

To assess the impact of the ordering criteria proposed in Section 6.6, the experiments of the previous section were repeated for different criteria. The objective used is the bandwidth minimization objective. Each of the SFC requests has the same amount of affinity constraints attached to them. Figure 6.18 shows the difference in acceptance rate when ordering based on the requested bandwidth, compared to ordering on the number of constraints. Mapping the SFCs with the lowest bandwidth requirements first, increases the acceptance rate significantly, since the load on the infrastructure is minimized.

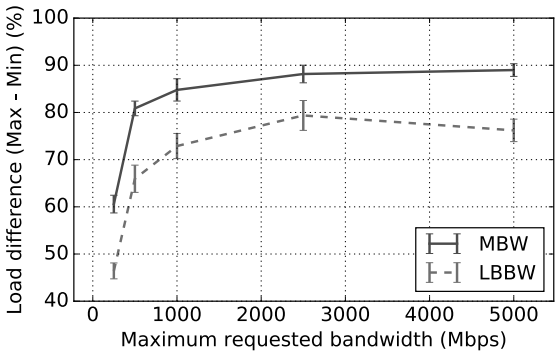



*Figure 6.16: Difference between the maximum and minimum link usage for various optimization objectives.*

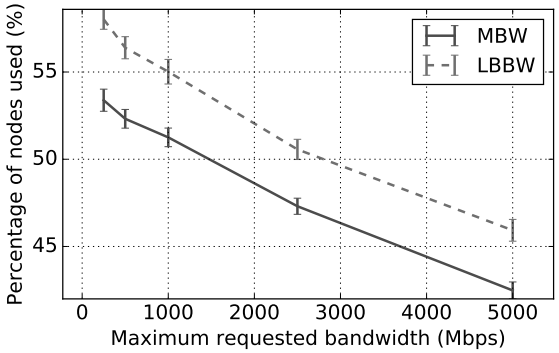



*Figure 6.17: Percentage of used nodes for various optimization objectives.*

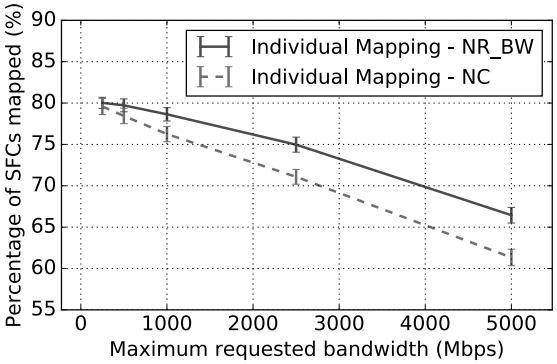



*Figure 6.18: Impact of SFC ordering on acceptance rate.*

## 6.8 Conclusion and Future Work

This chapter proposes a way for Service Providers (SPs) to attach location and colocation constraints to the mapping of Service Function Chains (SFCs) onto the substrate. These affinity constraints can be used to increase efficiency and resilience, to adhere to legislative and privacy restrictions or for economic reasons. First, the different sets of affinity and anti-affinity constraints are formalized. Second, a semantic validation framework is proposed, which allows the Virtual Network Function Infrastructure Provider (VNFInP) to check the consistency of the constraints posed by the SFC requests. To this end, the substrate and the SFC request are modeled using an ontology of which the consistency is checked using a semantic reasoner. Finally, the SFC embedding problem subject to affinity constraints is formalized and an Integer Linear Programming (ILP) formulation for both set-based SFC and individual SFC mapping is proposed. The semantic validation and different mapping algorithms were evaluated thoroughly. By only loading the parts of the infrastructure that are relevant for the SFC request into the ontology, the number of individuals and thus the semantic reasoning time can be significantly reduced. Furthermore, by filtering out inconsistent SFC requests before mapping, the total execution time of the embedding algorithm can be reduced with more than $50\%$ for the considered scenarios. Next to the set embedding formulation, also a heuristic approach is proposed in which the SFCs requests are embedded individually. To this end, the SFC requests are ordered based on certain criteria, after which the embedding is performed by solving the ILP. This heuristic can significantly reduce the calculation time. Depending on the scenario, a time reduction up to a factor $3000$ can be achieved at the cost of a reduction of the number of mapped SFCs with $2.5\%$ and an increased resource consumption of about $2.6\%$.

# References

[1] ETSI. *Network Functions Virtualization: An Introduction, Benefits, Enablers, Challenges and Call for Action*. ETSI Document, October 2012. Available from: http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[2] ETSI. *Network Functions Virtualization: Network Operator Perspectives on Industry Progress*. ETSI Document, October 2013. Available from: http://portal.etsi.org/NFV/NFV_White_Paper2.pdf.

[3] S. Latre, J. Famaey, F. De Turck, and P. Demeester. *The fluid internet: service-centric management of a virtualized future internet*. Communications Magazine, IEEE, 52(1):140–148, January 2014.

[4] R. Mijumbi. *On the Energy Efficiency Prospects of Network Function Virtualization*. arXiv preprint arXiv:1512.00215, 2015.

[5] S. Boucadair, D. Lopez, I. Telefonica, D. Guichard, and C. Pignataro. *Service Function Chaining: Framework & Architecture draft-boucadair-sfc-framework-00*. 2014.

[6] A. Fischer, J. Botero, M. Till Beck, H. de Meer, and X. Hesselbach. *Virtual Network Embedding: A Survey*. Communications Surveys Tutorials, IEEE, 15(4):1888–1906, Fourth 2013.

[7] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Denf, et al. *Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action*. In SDN and OpenFlow World Congress, pages 22–24, 2012.

[8] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. *Network function virtualization: Challenges and opportunities for innovations*. Communications Magazine, IEEE, 53(2):90–97, 2015.

[9] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. *Network Function Virtualization: State-of-the-art and Research Challenges*. Communications Surveys Tutorials, IEEE, PP(99):1–1, 2015.

[10] R. Mijumbi, J. Serrat, J.-L. Gorricho, S. Latre, M. Charalambides, and D. Lopez. *Management and orchestration challenges in network functions virtualization*. Communications Magazine, IEEE, 54(1):98–105, January 2016.

[11] G. Wang and T. Ng. *The Impact of Virtualization on Network Performance of Amazon EC2 Data Center*. In INFOCOM, 2010 Proceedings IEEE, pages 1–9, March 2010.

[12] J. Hwang, K. Ramakrishnan, and T. Wood. *NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms*. Network and Service Management, IEEE Transactions on, 12(1):34–47, March 2015.

[13] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. *ClickOS and the Art of Network Function Virtualization*. In Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, NSDI'14, pages 459–473, 2014.

[14] S. Benkner and C. GEMSS. *Report on COTS Security Technologies and Authorisation Services*. Project report, February 2004. Available from: http://eprints.cs.univie.ac.at/3311/.

[15] P. Massonet, S. Naqvi, C. Ponsard, J. Latanicki, B. Rochwerger, and M. Villari. *A Monitoring and Audit Logging Architecture for Data Location Compliance in Federated Cloud Infrastructures*. In Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, pages 1510–1517, May 2011.

[16] K. Konstanteli, T. Cucinotta, K. Psychas, and T. Varvarigou. *Admission Control for Elastic Cloud Services*. In Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, pages 41–48, June 2012.

[17] L. Larsson, D. Henriksson, and E. Elmroth. *Scheduling and monitoring of internally structured services in Cloud federations*. In Computers and Communications (ISCC), 2011 IEEE Symposium on, pages 173–178, June 2011.

[18] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth. *Modeling and Placement of Cloud Services with Internal Structure*. Cloud Computing, IEEE Transactions on, PP(99):1–1, 2014.

[19] H. Basilier, M. Darula, and J. Wilke. *Virtualizing network services–the telecom cloud*. Ericsson Review, 2014. Available from: http://www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2014/er-telecom-cloud.pdf.

[20] H. Technologies. *White Paper - Huawei Observation to NFV*. 2014. Available from: www.huawei.com/ilink/en/download/HW_399662.

[21] E. I. S. G. I. NFV. *Network Functions Virtualisation (NFV); Service Quality Metrics*. 2014. Available from: http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/010/01.01.01_60/gs_nfv-inf010v010101p.pdf.

[22] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang, and M. Zhani. *Data Center Network Virtualization: A Survey*. Communications Surveys Tutorials, IEEE, 15(2):909–928, Second 2013.

[23] E. Correa, L. Fletscher, and J. Botero. *Virtual Data Center Embedding: A Survey*. Latin America Transactions, IEEE (Revista IEEE America Latina), 13(5):1661–1670, May 2015.

[24] M. Melo, S. Sargento, U. Killat, A. Timm-Giel, and J. Carapinha. *Optimal Virtual Network Embedding: Node-Link Formulation*. Network and Service Management, IEEE Transactions on, 10(4):356–368, December 2013.

[25] M. Chowdhury, F. Samuel, and R. Boutaba. *PolyViNE: Policy-based Virtual Network Embedding Across Multiple Domains*. In Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures, VISA '10, pages 49–56, 2010.

[26] D. Dietrich, A. Rizk, and P. Papadimitriou. *Multi-domain virtual network embedding with limited information disclosure*. In IFIP Networking Conference, 2013, pages 1–9, May 2013.

[27] X. Cheng, S. Su, Z. Zhang, K. Shuang, F. Yang, Y. Luo, and J. Wang. *Virtual network embedding through topology awareness and optimization*. Computer Networks, 56(6):1797 – 1813, 2012.

[28] A. Amokrane, M. Zhani, R. Langar, R. Boutaba, and G. Pujolle. *Greenhead: Virtual Data Center Embedding across Distributed Infrastructures*. Cloud Computing, IEEE Transactions on, 1(1):36–49, Jan 2013.

[29] M. Rabbani, R. Pereira Esteves, M. Podlesny, G. Simon, L. Zambenedetti Granville, and R. Boutaba. *On tackling virtual data center embedding problem*. In Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, pages 177–184, May 2013.

[30] A. Basta, W. Kellerer, M. Hoffmann, H. J. Morper, and K. Hoffmann. *Applying NFV and SDN to LTE Mobile Core Gateways, the Functions Placement Problem*. In Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges, AllThingsCellular '14, pages 33–38. ACM, 2014.

[31] S. Mehraghdam, M. Keller, and H. Karl. *Specifying and Placing Chains of Virtual Network Functions*. CoRR, abs/1406.1058, 2014.

[32] M. T. Beck and J. F. Botero. *Coordinated Allocation of Service Function Chains*. In 2015 IEEE Global Communications Conference (GLOBECOM), pages 1–6, Dec 2015.

[33] S. Mehraghdam and H. Karl. *Placement of services with flexible structures specified by a YANG data model*. In 2016 IEEE NetSoft Conference and Workshops (NetSoft), pages 184–192, June 2016.

[34] H. Moens and F. De Turck. *VNF-P: A model for efficient placement of virtualized network functions*. In Network and Service Management (CNSM), 2014 10th International Conference on, pages 418–423, Nov 2014.

[35] M. Luizelli, L. Bays, L. Buriol, M. Barcellos, and L. Gaspary. *Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions*. In Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on, pages 98–106, May 2015.

[36] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs. *Network Function Placement for NFV Chaining in Packet/Optical Datacenters*. J. Lightwave Technol., 33(8):1565–1570, Apr 2015.

[37] B. Addis, D. Belabed, M. Bouet, and S. Secci. *Virtual network functions placement and routing optimization*. In Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on, pages 171–177, Oct 2015.

[38] A. Baumgartner, V. Reddy, and T. Bauschert. *Mobile core network virtualization: A model for combined virtual core network function placement and topology optimization*. In Network Softwarization (NetSoft), 2015 1st IEEE Conference on, pages 1–9, April 2015.

[39] M. Bouet, J. Leguay, T. Combe, and V. Conan. *Cost-based placement of vDPI functions in NFV infrastructures*. International Journal of Network Management, 25(6):490–506, 2015.

[40] S. Sahhaf, W. Tavernier, D. Colle, and M. Pickavet. *Network service chaining with efficient network function mapping based on service decompositions*. In Network Softwarization (NetSoft), 2015 1st IEEE Conference on, pages 1–5, April 2015.

[41] M. Yoshida, W. Shen, T. Kawabata, K. Minato, and W. Imajuku. *MORSA: A multi-objective resource scheduling algorithm for NFV infrastructure*. In Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific, pages 1–6, Sept 2014.

[42] I. Vaishnavi, R. Guerzoni, and R. Trivisonno. *Recursive, hierarchical embedding of virtual infrastructure in multi-domain substrates*. In Network Softwarization (NetSoft), 2015 1st IEEE Conference on, pages 1–9, April 2015.

[43] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and S. Davy. *Design and evaluation of algorithms for mapping and scheduling of virtual network functions*. In Network Softwarization (NetSoft), 2015 1st IEEE Conference on, pages 1–9, April 2015.

[44] J. G. Herrera and J. F. Botero. *Resource Allocation in NFV: A Comprehensive Survey*. IEEE Transactions on Network and Service Management, PP(99):1–1, 2016.

[45] N. Bouten, M. Claeys, R. Mijumbi, J. Serrat, J. Famaey, S. Latré, and F. De Turck. *Semantic Validation of Affinity Constrained Service Function Chain Requests*. In Network Softwarization (NetSoft), 2016 2nd IEEE Conference on, pages 1–9, June 2016.

[46] M. Al-Fares, A. Loukissas, and A. Vahdat. *A scalable, commodity data center network architecture*. ACM SIGCOMM Computer Communication Review, 38(4):63–74, 2008.

# 7 Conclusions and perspectives

*"Success is not final, failure is not fatal: it is the courage to continue that counts."*

–Winston Churchill (1874 - 1965)

In this dissertation, several research challenges were addressed with regard to the flexible deployment and management of both Video on Demand (VoD) and live HTTP Adaptive Streaming (HAS) services. These challenges can be subdivided into three categories: (1) resource-aware management of live HAS services, (2) in-network optimization of VoD HAS services and (3) flexible deployment of HAS services. This dissertation argues that in-network management for VoD and live HAS services is required to meet the Quality of Experience (QoE) requirements expected by the end users, while at the same time reducing the impact of these services on the network resources. Moreover, to allow a flexible deployment of new and innovative streaming services, the different functions that were proposed should be virtualized to allow the on demand deployment of the proposed Service Function Chains (SFCs) onto the virtualized network resources.

## 7.1 Resource-aware management of live HAS services

Over-The-Top (OTT) live streaming services using HAS technologies are widely deployed across the Internet. In HAS, every client requests the content using a unicast connection to the streaming server. This implies that the network resource

usage is directly proportional to the number of connected clients. For large live streaming events (e.g., the FIFA World Cup), HAS streaming exerts a high strain on the underlying network resources. In Chapter 2, a multicast-enabled delivery framework is proposed to reduce the strain of HAS services on the network resources. To this end, intermediary network nodes are deployed at the ingress that group multiple overlapping unicast sessions into a single multicast channel, which is delivered to egress nodes closer to the end-users. These nodes transform the multicast sessions into HAS unicast sessions, significantly reducing the network resource usage. The proposed algorithms for grouping the unicast into multicast sessions and to select the multicast channels to subscribe to are able to improve both live and Time Shifted TV (TSTV) HAS services. The managed multicast-enabled HAS delivery can reduce the bandwidth usage with up to $60\%$ in a realistic TSTV scenario compared to unicast cache-assisted HAS delivery.

Another issue with live HAS delivery is the requirement of large client-side buffers in order to provide an acceptable QoE for the end-user. This increases the camera-to-display delay significantly, negatively impacting the overall user experience for live events. Especially in high Round Trip Time (RTT) networks, large buffers are required to cope with variable network conditions. Appendix A proposes to adopt Scalable Video Coding (SVC) techniques for live HAS streaming in combination with improved request scheduling techniques to reduce the impact of RTT on the QoE. This allows eliminating the idle time between two consecutive downloads and thus more efficient usage of available resources. By applying these techniques, the required client-side buffer size can be shrunk significantly compared to state-of-the-art Advanced Video Coding (AVC)-based heuristics, while still offering the same QoE. Appendix B adds in-network delivery management nodes to further decrease the camera-to-display delay. By adding an intermediary proxy which prioritizes segments destined for clients which are close to a buffer starvation, the delivery of the base layer can be guaranteed using Differentiated Services (DiffServ). By deploying dynamic deadline-based algorithms, the risk of buffer starvations can be further reduced, allowing the buffer to be shrunk from multiple dozens to only a few seconds.

## 7.2 Optimization of VoD HAS services

Multiple autonomous HAS clients competing for bandwidth show frequent quality oscillations and buffer starvations due to uncoordinated adaptation heuristics. When offering a HAS service, such QoE-degradations are unacceptable. Therefore, Chapter 3 mitigates this QoE degradation by proposing an in-network optimization framework for the delivery of HAS VoD services. The in-network proxies along the delivery path manage the resource usage of the connected client applications and dynamically limit the possible set of quality representations for each

client. The proposed hybrid approach allows clients to still react upon sudden network changes or scarcity in device resources, while increasing the overall quality and stability. Moreover, it can enforce a wide range of management policies, allowing providers to specify priorities when allocating resources to a certain group of users. The average quality can be improved with $14\%$, while the number of quality oscillations can be reduced with a factor $5$.

Chapter 4 extends the aforementioned framework to allow it to be deployed under dynamic network conditions as well. To this end a set of monitoring probes are deployed that use sampling-based measurement techniques to monitor the available throughput on the different delivery paths in the network. These measurements are fed to a throughput prediction component of which the output is used during the in-network optimization. The quality of each client is then optimized in terms of QoE, allowing the proposed approach to improve the QoE with $19\%$ compared to the purely resource-driven optimization and with $30\%$ compared to the autonomous quality selection heuristics. The sampling-based estimation techniques allow a reliable prediction of the future available bandwidth at the limited cost of sampling $1\%$ of the packets.

Another issue with HAS streaming issues lies in one of its benefits: the seamless reuse of existing HTTP caching infrastructure. While the deployment of caches can significantly reduce the impact of HAS VoD streaming services on the network resources, at the same time, it increases the quality instability of the adaptation heuristics. Since the quality decisions of HAS adaptation heuristics are based on the estimated throughput, incorrect throughput estimations can cause frequent quality switching and even worse, buffer starvations. In cache-assisted HAS, segments can be served from different origins based on the content of the caches, causing highly fluctuating throughput and RTT measurements, negatively impacting the stability and optimality of the quality decisions.

In Chapter 5, a set of heuristics are proposed which take advantage of additional information on the streaming origin and intermediary cache contents to optimize the quality decisions. Using more accurate and per origin throughput measurements, the QoE can be significantly improved. Since this information is not always available or the required changes of the intermediary nodes are not possible in an OTT scenario, approximation techniques are proposed as well. Using unsupervised incremental clustering techniques the streaming origin can be detected based on the measured RTT. With an accuracy of above $85\%$, the streaming origin can be detected and used to optimize the quality selection, improving the QoE up to $11\%$. Furthermore, by using probability-based estimation techniques to predict the cache contents, the QoE can be improved with up to $21\%$, compared to cache-agnostic adaptation heuristics.

## 7.3 Flexible deployment of HAS services

Static deployments of HAS services where intermediary Network Functions (NFs) (e.g., caches, prioritization proxies) are implemented as middleboxes that need to be physically deployed, reduce the flexibility which is required to rapidly offer new and innovative services. The Network Function Virtualization (NFV) paradigm offers a solution to these problems by decoupling the functionality of the NFs from the physical appliances by virtualizing them and deploying them on high capacity servers, switches and storage equipment. This allows to deploy new SFCs on demand and reduces the time to market of new services. Unfortunately, current NFV approaches offer limited control to the Service Provider (SP) on the deployment locations of the Virtual Network Functions (VNFs) in the SFCs. This may however be required for efficiency, economic, privacy or legislative reasons.

Chapter 6 defines a set of affinity and ant-affinity constraints, which can be used to attach location specific requirements to an SFC request. This allows the SP to attach the desired location requirements (e.g., caching nodes close to end-users) that enable the full potential of the QoE-management solutions that were proposed earlier. To check the consistency of the affinity-constrained SFC requests, a semantic validation framework was proposed in Appendix C. By modelling the SFC requests and virtual resources using an ontology and applying a set of inference rules, the consistency of the SFC request can be checked. By pre-filtering these invalid SFC requests, the mapping times can be reduced with more than $50\%$.

## 7.4 Future perspectives

This dissertation offers several contributions towards QoE-managed delivery of HAS services and the flexible deployment of such services. This thesis has identified further challenges that require the attention of the research community. These are discussed below:

### 7.4.1 HAS-aware cache replacement and prefetching strategies

This dissertation proposed mitigation strategies for the negative impact in-network caches can have on the perceived QoE. However, not only the client-side adaptation can be improved, also the cache replacement and prefetching strategies could be further improved. Current research within the group is focussing on the optimization of cache replacement strategies and cooperative caching mechanisms for segmented HTTP streaming services. The temporally segmented video content has some specific characteristics that can be exploited when selecting which segments to evict from the cache. When a segment was recently requested by a client application, there is a high probability that it will also request the next segment.

Exploiting this knowledge, the replacement strategies can be optimized, both for a single cache as well as a network of caches. Current research only considered segmented streaming without adaptation. In HAS however, the quality adaptation can also have a significant impact on the performance of the caching strategy. After all, although there is a high probability the next segment will also be requested, the quality in which this segment will be requested is not known, meaning that the current caching strategies for segmented streaming cannot be reused without adjustments. Increasing the cooperation between the clients and the proxies to jointly decide which qualities to request based on the current buffer filling and cache content could potentially improve the QoE. By enabling the prefetching of segments, the benefits could even be increased further. However, one should take into account the additional resources that are consumed by the prefetching strategies on the bottleneck links. The impact of prefetching should be minimized as to not interfere with the delivery of regular HAS traffic, in order not to reduce the quality of clients which cannot be served by the intermediary caches.

### 7.4.2 Software Defined Networking (SDN)-enabled delivery optimization of HAS services

In Appendix B, a method using DiffServ technologies was proposed to prioritize the delivery of certain layers or segments of a live SVC HAS streaming service. More recent technology developments, such as SDN, could be exploited to deploy an SDN controller in the network which is responsible for prioritizing the delivery of HAS segments, based on the feedback collected from the network nodes and the clients. Currently, the research group is investigating how the client status can be estimated without any explicit communication between the different network nodes. This would allow to estimate the current buffer status of the connected clients and decide which segments to prioritize. Using an SDN-based approach increases the flexibility of the DiffServ approach that was proposed in Appendix B, since it is not limited by the number of Per Hop Behavior (PHB) that are implemented in intermediary routers. In contrast to the proposed DiffServ approach, an SDN-based solution does not require the deployment of the marking proxies, since the controller is responsible for applying the specific behavior. Also the approach proposed in Chapter 2 can be implemented using SDN technologies. SDN allows to construct and maintain the multicast tree between the distribution and subscribed delivery servers using a control application running on the logically centralized SDN controller. Thanks to the global view, this application is able to select the content that needs to be multicasted, as well as the channels each delivery server needs to subscribe to. The programmable nature of SDN allows for immediate deployability, scalability, adaptability and updatability, traits that are not associated with the more fixed nature of IP multicast solutions.

### 7.4.3 Streaming-aware congestion control

As mentioned in the introduction, bandwidth fairness at the TCP level does not necessarily translate into fairness in terms of QoE. A client can define deadlines for each segment that is requested based on the current buffer filling, i.e. the segment has to be fully delivered before the buffer depletes to assure a fluent playout. These deadlines can be taken into account at the sender side to automatically adapt the aggressiveness of the TCP sessions. By adjusting the level of increase or decrease of the window size, the relative throughput compared to other sessions can be increased or decreased. In this way, sessions with a nearer deadline can be granted more resources than sessions with a deadline further away. This allows to improve the overall fairness in terms of QoE.

### 7.4.4 Mobile HAS delivery

Although many of the proposed approaches can also be applied to mobile HAS streaming, there exist some differences. First, since mobile devices can move between different locations, also the access technologies that are used can vary (e.g., 3G/4G, WiFi). Being able to cope with such mobility, is a major challenge for streaming technologies. The switch in access technology should be seamless and not interrupting the video playout. Not only the characteristics of the network (e.g., throughput, RTT) can vary during the streaming session, also the various VNFs that are included in the SFCs, possibly need to be relocated, reconfigured or replaced. Furthermore, since these devices have limited battery life, which varies with the quality that is streamed, streaming solutions should also take into account the energy consumption of their decisions.

### 7.4.5 Automated deployment of HAS SFCs

Up until now, the SFCs were statically defined and deployed by the SP. However, it would be interesting to let the different characteristics (e.g., cache sizes, link bandwidths) scale according to the number of connected users. If scaling does not suffice, or resources are scarce, the SFC could be adapted by adding additional instances and load balancing the traffic between these functions. Another approach could be not to adapt the SFC, but to deploy a totally different SFC. For example, the prioritization-based SFC could be replaced by a multicast-enabled delivery SFC when the number of users increases. To be able to automate this process, the impact of the different SFCs on the resources subject to the number of users needs to be used as an input for the decision algorithm. By evaluating the potential gain of scaling functions, adjusting SFCs or deploying alternative SFCs, the management algorithm can decide on how to progress. Further research is required on how to model the different SFCs and their impact on QoE so that the process can

be automated. Furthermore, also migration strategies are required when functions are duplicated or alternative SFCs are deployed. Since streaming applications are sensitive to high delays and service interruptions, such migration strategies should be tailored to reduce the impact of the migration events on the continuity of the service.

# A

# Minimizing the Impact of Delay on Live SVC-based HTTP Adaptive Streaming Services

**N. Bouten, S. Latré, J. Famaey, W. Van Leekwijck, F. De Turck.**

**Published in Proceedings of the 2013 IFIP/IEEE International Symposium on Integrated Network Management, May. 2013.**

*Chapter 2 proposes a multicast-enabled framework to optimize the resource consumption of live HTTP Adaptive Streaming (HAS) services. This appendix focuses on mitigating the negative impact on Quality of Experience (QoE) of high Round Trip Time (RTT) networks for live HAS by leveraging the benefits of Scalable Video Coding (SVC). Instead of downloading one file for a certain quality level, scalable video streaming requires downloading several interdependent layers to obtain the same quality. This implies that the base layer is always downloaded and is available for playout, even when throughput fluctuates and enhancement layers cannot be downloaded in time. This layered video approach can help in providing better service quality assurance for video streaming. In HAS, requesting multiple files over HTTP leads to an increased impact of the end-to-end delay on the service provided to the client. This is even worse in a Live TV scenario where the drift on the live signal should be minimized, requiring smaller segment and buffer sizes. In this appendix, several ways are proposed to overcome the high RTT issues, such as parallel and pipelined downloading.*

## A.1 Introduction

HTTP Adaptive Streaming (HAS) is becoming the de-facto standard for Over-The-Top (OTT) video streaming services. This shift was mainly induced by the advantages offered by HTTP-based streaming: reliable transmission over TCP, reuse of existing caching infrastructure and compatibility with NATs and firewalls. Initially, the HTTP-based video protocols required downloading the entire video file before playout. Later, progressive download techniques allowed playout to begin after a sufficient amount of data was stored in the buffer. However, when congestion arised in the network, these protocols were not able to cope with buffer starvations, leading to playout gaps, which have a negative impact on the provided service quality. The third evolution in HTTP-based streaming, being HAS, tackles these shortcomings by splitting the content into segments which are encoded at different quality levels. Client heuristics then decide at which quality rate the next segment should be downloaded, taking into account network statistics, buffer filling and device characteristics. HAS therefore provides service assurance, at a reduced quality, if congestion occurs in the network.

Traditionally, Advanced Video Coding (AVC) is used to encode the different segments, introducing a significant amount of redundancy across quality representations. Scalable Video Coding (SVC) can cope with these issues of content redundancy by creating dependencies between the base and enhancement layers. Adopting SVC in HAS significantly improves caching and bandwidth efficiency at the server side, while reducing the risk of running into frame freezes at the client, since for every segment the base layer is always downloaded. Furthermore, since SVC requires multiple layers to be downloaded, quality rate adaptations can be performed at higher granularity, since throughput fluctuations can now be detected earlier, allowing heuristics to react faster.

However, there are several drawbacks to adopting SVC in HAS [1, 2]. First, separating the video flow into several SVC layers introduces a coding penalty, which leads to an encoding overhead of approximately 10% per enhancement layer. Second, when downloading multiple layers per segment, more requests are generated at the client to download subsequent layers of a single segment. These request-response cycles introduce a wait time between the reception of the last byte of the previous segment layer and the first byte of the next segment layer, which is equal to the Round Trip Time (RTT). Third, when considering Live TV over HAS, the segment sizes should be as small as possible to reduce the latency on the broadcast signal. These smaller segments increase the decision granularity even further, but at the same time increase the idle time between two consecutive downloads, negatively impacting the service quality for the user.

In this appendix several AVC and SVC-based client heuristics are proposed and compared. Furthermore, their behavior in a Live TV setting is evaluated

where client side buffers need to be small to reduce latency on the broadcast signal, while still providing service assurance. The heuristics decide on the best quality to download next, for the video streaming service, based on low level monitoring data such as bandwidth measurements and the status of the client's play-out buffer. Furthermore, this appendix proposes and compares several download scheduling approaches such as sequential, pipelined and parallel HTTP downloading and their ability to reduce the impact of latency on service delivery. Additionally, the impact of high round trip times on SVC-based HAS streaming is investigated in a Live TV setting.

The remainder of this appendix is structured as follows. Section A.2 provides an overview of relevant work, followed by an overview of existing adaptation heuristics in Section A.3 and of a novel SVC-based heuristic, specifically designed for small buffers in Section A.4. Section A.5 discusses how adapting the download scheduling can improve the quality and service assurance. Section A.6 elaborates on the experiment setup and evaluation results, which are summarized in Section A.7.

## A.2 Related Work

The increased popularity of video consumption over the Internet has led to the development of a range of protocols that allow adaptive HTTP-based video streaming. Some of the major players have introduced their own protocols, server and client software such as Microsoft's Silverlight Smooth Streaming [3], Apple's HTTP Live Streaming [4] and Adobe's HTTP Dynamic Streaming [5]. More recently, a standardized solution has been proposed by MPEG, called Dynamic Adaptive Streaming over HTTP (DASH) [6]. Even though differences exist between these implementations, they adopt the same design principles. The video is split into several segments, encoded at different quality rates. These segments are offered by a web server and are transmitted over standard HTTP connections. The intelligent video client uses a selection heuristic to dynamically adapt the quality, based on the current network statistics, buffer filling and other device characteristics.

Optimizations of HAS-based delivery can be performed at the server, the network or the client. At the server side, optimizations are focussed on the encoding scheme. Traditional deployments of HAS use the H.264/AVC codec for creating the different representations of the video. For each representation a separate file needs to be stored at the video server, leading to an increased storage penalty due to the redundant information. Adopting an SVC extension to H.264/AVC [7] or High Efficiency Video Coding (HEVC) [8], allows alleviating the storage issues with AVC at the server, while improving caching efficiency. Huysegems et al. discuss the advantages of using SVC instead of AVC, such as guaranteed playout during

fluctuations since the base layer is always downloaded and a reduction in bandwidth and storage requirements at the server [2]. However, important challenges for AVC are indicated to be the encoding overhead of SVC and the increased vulnerability to high round trip times. This appendix focuses on these high round trip times and how their negative impact on SVC-based HAS can be reduced.

Liu et al. present an in-network optimization of HAS for 3GPP networks. By parallelizing the download and request of HAS segments, a better resource utilization can be achieved [9]. Bouten et al. have discussed how a network provider can manage the quality that is offered to the clients in a HAS environment [10]. The solution takes into account subscription parameters and device parameters to restrict the qualities that are offered to the client. An autonomic delivery framework is presented in previous work [11, 12], which allows to reduce the consumed bandwidth by grouping unicast HAS sessions sharing the same content into a single multicast session. This appendix focuses on measures at the client side rather than the network with a focus on the scheduling of segment requests in networks with high round trip times.

Each commercial HAS implementation comes with an existing video client heuristic of its own. Akhshabi et al. compare several commercial and open source HAS players and indicate significant inefficiencies in each of them [13]. Several heuristics have been proposed in literature as well, each focussing on a specific deployment. Liu et al. discuss a video client heuristic that is suited for Content Delivery Networks (CDNs) by comparing the expected segment fetch time with the experienced segment fetch time to ensure a response to bandwidth fluctuations in the network [14], while Adzic et al. present a client heuristic which is tailored for mobile environments [15]. Jiang et al. aimed to develop an efficient, fair and stable heuristic by randomizing chunk scheduling to avoid synchronization, stateful bitrate selection and delayed update to avoid instability [16], and compared their approach to commercial players. Generic algorithms exist for selecting the next video quality to download, using a priority-based scheme where base layers receive higher priority in download scheduling compared to the enhancement layers [17]. Andelin et al. provide a heuristic which was specifically designed for SVC and using a slope to define the trade-off between downloading the next segment and upgrading a previously downloaded segment [18].

In this appendix, several heuristics are evaluated that are either based on commercial players, algorithms described in literature or designed from scratch. Furthermore, this appendix focuses on how downloads should be scheduled in networks suffering from high round trip times and compares sequential, pipelined and parallel download schedulers.

## A.3 State of the Art Rate Adaptation Heuristics

In this section the state of the art rate adaptation heuristics for AVC and SVC-based HAS are summarized. For an extensive description of these heuristics, the reader is referred to [1].

### A.3.1 AVC Microsoft Smooth Streaming (MSS) heuristic

The heuristic for AVC video streaming is based on an open source version of the algorithm of the MSS video player [1]. The heuristic can be configured using 3 thresholds: the panic (P), lower (L) and upper (U) threshold. There are two states: *buffering* and *steady state*. During the *buffering state*, quality decision is based on measured throughput and can only be increased with one level. When the buffer level is equal to or exceeds $L + (U - L)/2$, the heuristic goes into *steady state*. If during the *steady state*, the buffer filling level is slowly changing but lower than $L$, the quality level is decreased. When the buffer filling level is between $L$ and $U$ and quickly increasing or when the buffer filling level exceeds the upper threshold $U$, the heuristic attempts to improve the quality level if the throughput measurements indicate that the next segment can be downloaded in time. If the buffer filling level drops under the panic threshold $P$ or the buffer filling level is quickly decreasing and lower than $L$, the next segment is downloaded at the lowest quality level and the heuristic returns to *buffering state*.

### A.3.2 SVC MSS heuristic

The AVC MSS heuristic can also be used for SVC-based HAS when the quality decisions are translated into subsequent layer downloads. SVC MSS however, allows adapting the quality decisions in between two consecutive layer downloads. Thanks to the finer granularity, SVC MSS is able to cancel the download of one or more enhancement layers when the measured throughput indicates that they will not arrive in time for playout.

### A.3.3 SVC slope heuristic

The SVC Slope heuristic exploits two main advantages of SVC-based HAS. First, SVC video streaming allows more fine grained decisions as the quality decision can be adapted after every download of a layer rather than every segment. Second, since SVC video layers are interdependent, the heuristics can decide either to download the base layer of a new segment or to increase the quality of a previously downloaded segment by downloading additional enhancement layers. This

[1] Source available fromhttps://slextensions.svn.codeplex.com/svn/trunk/SLExtensions/AdaptiveStreaming

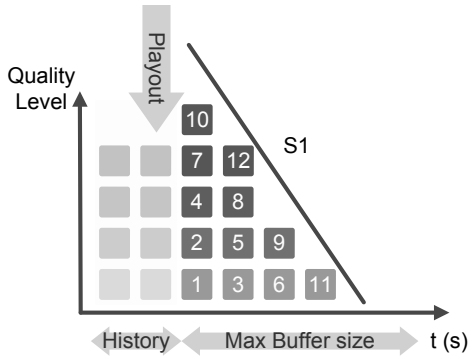



*Figure A.1: Illustration how a steeper slope prioritizes backfilling over prefetching.*

backfilling is also possible with AVC, but since the redundant data is downloaded again, this affects the efficiency of the heuristic drastically.

Andelin et al. proposed a slope-based SVC heuristic [18], where the backfilling is limited by a moving buffer slope. The slope can be configured to give priority to either prefetching (downloading lower quality layers for future segments) or backfilling (downloading additional enhancement layers for buffered segments). This configuration is done by defining a slope in the heuristic: the steeper the slope, the more backfilling will be chosen over prefetching. Similarly, the flatter the slope, the more prefetching will be done for future segments. This filling behavior is illustrated in Figure A.1, which shows the segment and layer download order for a configuration $S_1$ of the slope parameter.

## A.4 SVC Adaptation Heuristic for Small Buffers

A novel heuristic for scalable video was designed that is able to cope with small buffer sizes, which are common in a Live TV scenario. To be able to optimize service quality for the end-user, three factors are important for the client: 1) avoiding frame freezes and gaps in video playout, 2) ensuring quality stability limiting the number of quality switching occurrences and 3) allowing high quality streaming. The SVC cursor based algorithm is designed to avoid gaps and limit quality switches while trying to provide the highest possible quality. This is accomplished by using two distinct cursors: segment cursor and quality cursor, defining which segment is under consideration for the next decision and the goal quality respectively. Limiting the number of switches is further accomplished by using a timeout for the quality improvement decision. The segment cursor advances to the next segment when a) all qualities up to the quality cursor are downloaded for the current segment cursor or b) the layer under consideration cannot be downloaded in

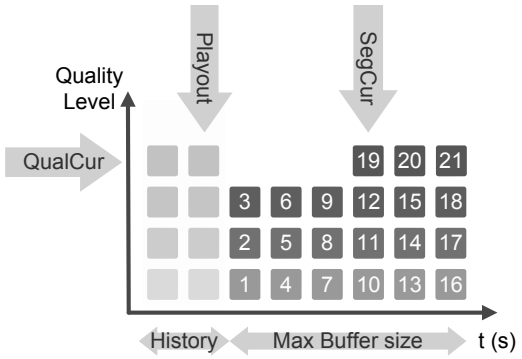



*Figure A.2: Illustration of the backfilling operation by SVC Cursor when the quality cursor was improved.*

time. When a layer will not be downloaded in time for playout, the quality cursor is decreased and the improvement timer is reset. The quality cursor can only be incremented when all lower layers of every segment are downloaded and the improvement timer has timed out. The segment cursor is then moved based on the estimations of the arrival times of these enhancement layers and their playout times, evaluated from right to left. When some of the enhancement layers are estimated to be downloaded before their respective playout time, the quality cursor is incremented and the scheduler starts downloading from left to right as shown in Figure A.2, after which the improvement timer is reset. So when the quality cursor is increased, a backfilling operation updates the buffer to the required quality level.

## A.5 Delay-optimized Download Scheduling

As discussed, adopting layered video coding has several advantages for HAS. But at the same time, there is a major drawback when considering networks with high RTTs. Since multiple layers per segment need to be downloaded, more request/response cycles are induced when downloading the subsequent segment layers. As a result, there is a wait time between receiving the last byte of the previous segment layer and the first byte of the next segment layer. This idle-time is equal to the RTT. When HAS is applied to a Live TV scenario, the drift on the broadcast signal is to be kept as small as possible to enable the streaming for live events and the use of second screen applications. As a consequence, the segment size should be as small as possible. When reducing the segment size however, the impact of the idle-time between downloading two consecutive segments even increases. When the RTT is 50msec and a 5-layer SVC video representation is used with a 1 second segment size, the idle-time accounts for 250msec when downloading the

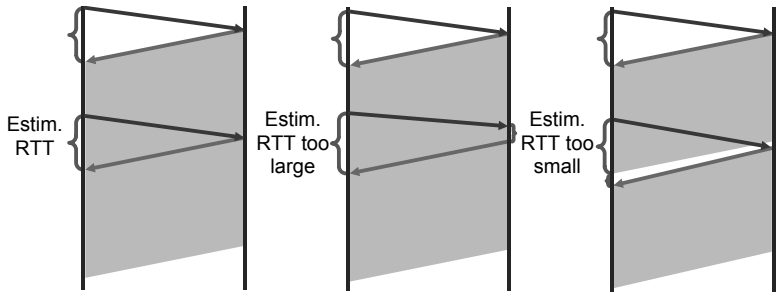



*Figure A.3: Overview of estimation with pipelining a) accurate estimation b) overestimation c) underestimation.*

highest quality representation, which is one fourth of the available download window. This example shows the importance of alleviating the impact of RTT on SVC HAS in Live TV scenarios. This can be achieved by applying pipelined or parallel download scheduling, both of which are able to eliminate the incurred idle times.

### A.5.1 Pipelined scheduling

HTTP pipelining is a technique in which multiple HTTP requests are sent on a single TCP connection without waiting for the corresponding responses. Kaspar et al. proposed Pipelining to improve progressive downloading, the predecessor of HAS [19]. Pipelining all requests at once will of course not yield a viable solution, since then the ability for fast response to network changes is sacrificed. This appendix proposes to estimate the RTT and to schedule the next request RTT seconds before the current download will be finished. This technique allows postponing the decision on which segment layer to download as long as possible, while still eliminating the idle time between two consecutive downloads. With perfect estimations of download time and RTT, the delay could be completely eliminated as shown in Figure A.3(a). However with varying RTT, this will not be the case and RTT will be overestimated or underestimated most of the time. When overestimating the RTT the decision on which segment layer to download is taken too early and conditions could change during this period. But as illustrated in Figure A.3(b), the overestimation of the RTT cannot be noticed at the client side which has no indication of the request being queued at the server. When underestimating the RTT however, there is a gap between the two consecutive downloads, which can be measured at the client side. Adding this measurement to the estimated RTT, yields an accurate estimation for the RTT as shown in Figure A.3(c). The proposed approach for the pipelined scheduling is to linearly decrease the estimated delay until an underestimation is perceived ($t_{first\_byte\_s} - t_{last\_byte\_s-1} > 0$), and an accurate estimation for the delay can be established.

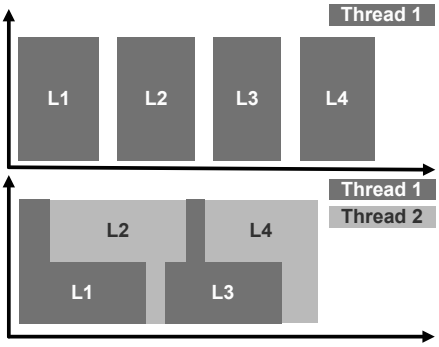



*Figure A.4: Illustration of the delay masking behavior of parallel scheduled segment layer downloads.*

### A.5.2 Parallel scheduling

Another approach to avoid idle time between consecutive downloads is to request several segments at the same time using parallel TCP connections. This allows request/response cycles to interleave with active downloads, reducing the idle time. A simplified example of this masking behavior is shown in Figure A.4. An additional advantage of parallel download scheduling is the improved performance when using parallel TCP connections. However, since now the segment layer downloads are requested over concurrent TCP connections, they compete for the available bandwidth and the download times are proportional to the number of threads. The disadvantage is of course that the base layers take longer to complete and thus an increased risk of buffer starvations. Therefore, the number of parallel threads needs to be limited.

## A.6 Experimental Results

The performance of the video client heuristics and download schedulers was evaluated using the NS-3 network simulator[2] in combination with the Network Simulation Cradle[3]. Figure A.5 illustrates the used network topology with $N$ clients connected to the HAS Server via a router. Each client has a playout buffer of $P$ seconds which is varied during the experiments and has a connection link with bandwidth $B_c$. The shared bandwidth $B_s$ and the total RTT $R$ are varied during the experiments.

The simulations were conducted using the traces of a Variable Bitrate (VBR)

[2]NS-3 Network Simulator - http://www.nsnam.org/

[3]WAND Network Research Group: Network Simulation Cradle - http://research.wand.net.nz/software/nsc.php

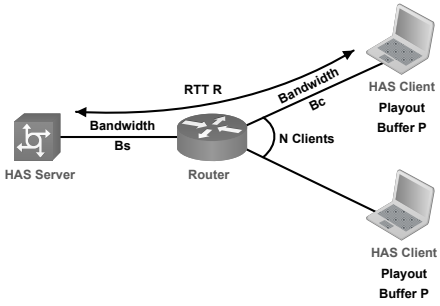



*Figure A.5: Experimental setup offering a HAS-based video streaming to $N$ clients. The parameters $B_s$, $B_c$, $P$, $R$ are varied.*

video file, encoded both for H.264/AVC and H.264/SVC using the JSVM 9.19.15 Encoder. The video has a frame rate of 30fps and GOP size of 32 frames, which leads to a minimum segment size of 1.06667s when using I-frame segmentation. The clients were started using a Weibull startup process with average 900 seconds and shape 2.5.

### A.6.1 Comparison of adaptation heuristics

Figure A.6 shows the total buffer starvation in seconds, the average playout quality level and the total number of switches for an increasing buffer size $P$. All four client heuristics are compared using a sequential download scheduler. For a buffer size of 1 segment ($P = 1.1s$), AVC MSS remains in panic mode, since only one segment fits in the buffer and it needs to be played out completely before the next segment is able to fit in the buffer. This causes AVC MSS to always download the lowest quality, which explains the low number of switches and buffer starvations. For a two segment buffer however, AVC MSS constantly switches between *buffering* and *steady state*, constantly switching between representations, which explains the large increase in the number of switches. One can thus conclude that for AVC MSS, a minimum buffer size of 3 segments is required to obtain acceptable quality while avoiding quality oscillations. For small buffers, SVC Slope is able to yield the highest quality, but at the cost of a high number of switches and buffer starvations, diminishing stability and quality assurance. For buffer sizes of 3 segments and up, AVC MSS yields higher quality than the SVC algorithms. This can be attributed to two reasons. First, the overhead introduced by the SVC encoding causes a higher load on the bottleneck $B_s$. Second, for a larger bottleneck $B_s$, as illustrated in Figure A.6(b), the problems of the SVC-based algorithms persists because of the vulnerability of SVC to high delays. These delays in combination with the increased number of request-response cycles of SVC-based algorithms limit the throughput causing lower efficiency and thus a lower average quality. SVC Cur-

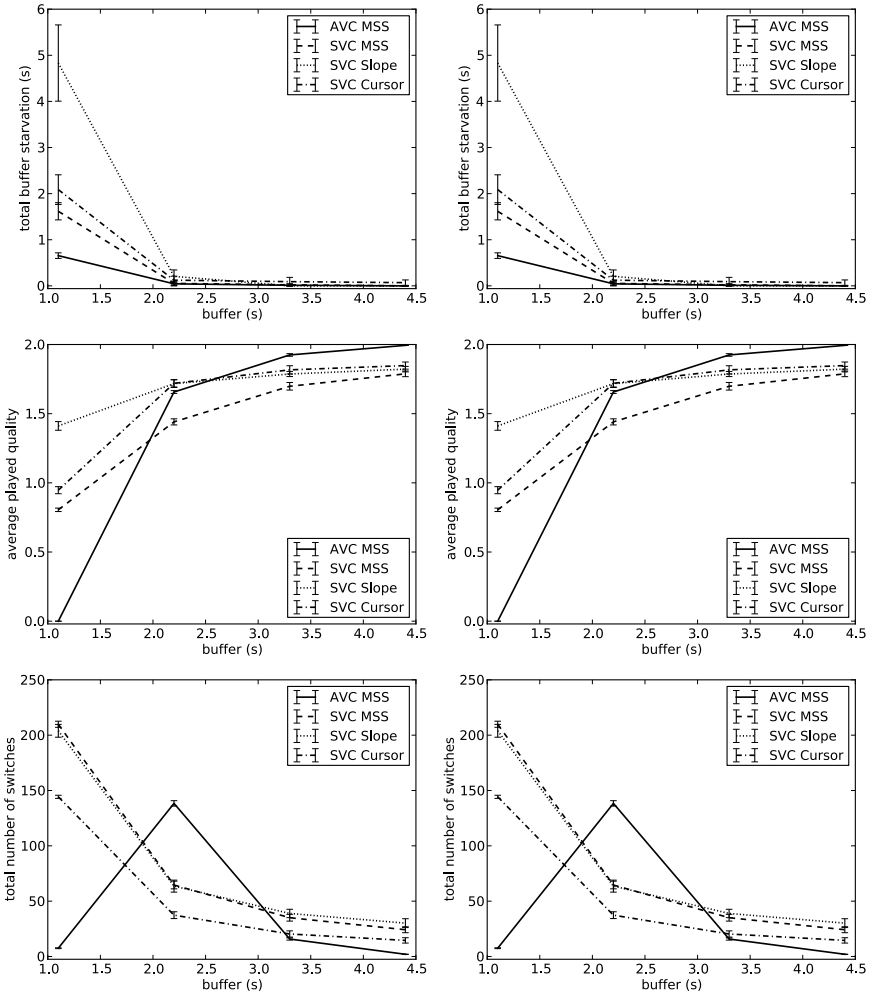



*Figure A.6: Total buffer starvation (s), average played quality level and total number of switches in function of the buffer size $P$ (s) with $B_c = 10Mbps$, $N = 20$, $R = 50ms$ and (a) $B_s = 50Mbps$ (b) $B_s = 100Mbps$.*

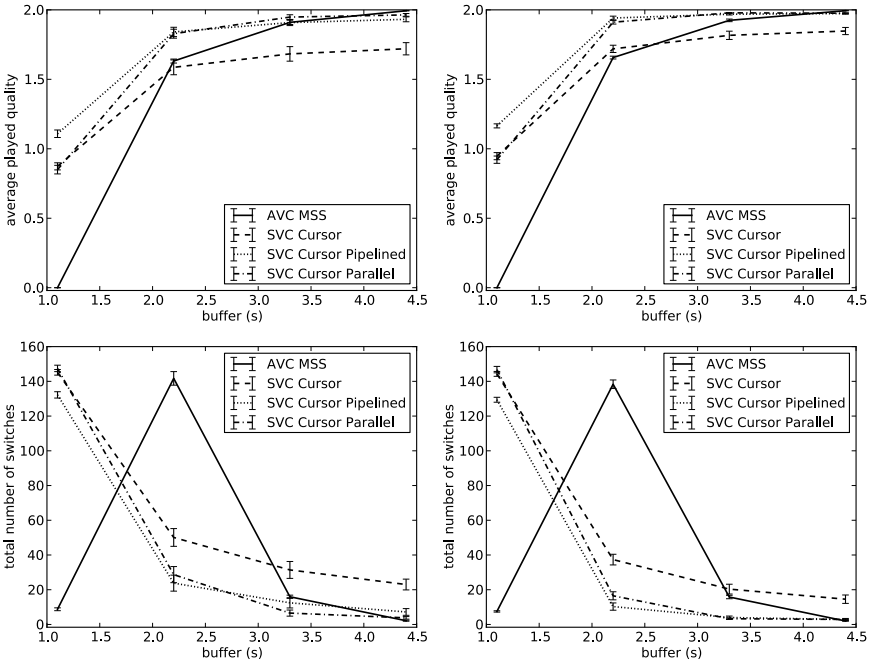
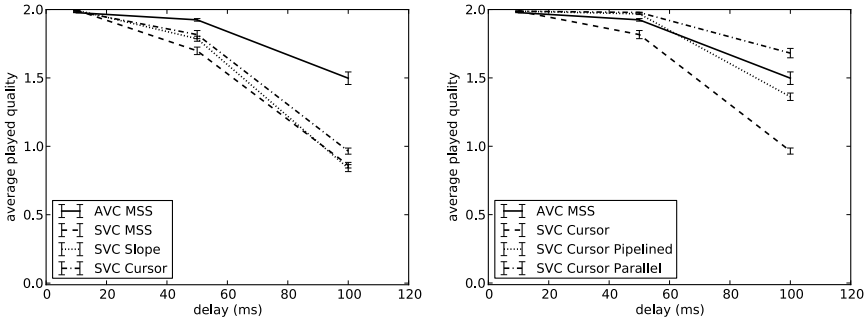



*Figure A.7: Average played quality level and total number of switches in function of the buffer size $P$ (s) with $B_c = 10Mbps$, $N = 20$, $R = 50ms$ and (a) $B_s = 50Mbps$ (b) $B_s = 100Mbps$.*

sor is able to minimize buffer starvation time and the number of switches, while yielding the highest quality for buffer sizes containing 2 or more segments. SVC Slope tends to yield lower quality, more frequent switches and a higher overall gap time. This can be attributed to the difficulty of configuring the slope parameter for the varying situations. Overall, it can be concluded that SVC Cursor outperforms all other SVC-based heuristics in terms of quality, switches and buffer starvations, therefore this heuristic is used for the subsequent results.

### A.6.2 Impact of download scheduling

As shown in the previous results, SVC-based HAS suffers from high RTTs, leading to inefficient use of the available throughput and reducing the benefits of SVC-based HAS in terms of caching efficiency and server bandwidth utilization. Therefore, this appendix suggests more optimized HAS-scheduling by using HTTP Pipelining and parallel downloads. Figure A.7 shows the results for the same configurations as before, but with the pipelined and parallel scheduled variants of SVC




(a) (b)

*Figure A.8: Average played quality level in function of the RTT $R$ (s) for $B_s = 100Mbps$, $B_c = 10Mbps$, $N = 20$ and $P = 3.3s$.*

Cursor. Here, one can clearly see the advantages of using optimized scheduling for SVC-based HAS for high RTTs. SVC Cursor Pipelined and Parallel are able to improve the quality level, while lowering the number of quality switches at the same time. When the bottleneck is tight ($B_s = 50Mbps$), the quality yielded by SVC based HAS is only slightly lower than when using AVC-based HAS, this is caused by the encoding overhead of SVC. But when taking a look at the 100Mbps scenario, SVC-based HAS is able to outperform AVC-based HAS, in terms of switches and average quality. Even when the buffer only contains 2 segments, the quality level and number of switches are at an acceptable level.

### A.6.3 Impact of delay

Figure A.8 illustrates the impact of high RTTs on the different base algorithms and on the SVC Cursor algorithm in combination with pipelined and parallel scheduling for a buffer containing 3 segments. These graphs show that parallel scheduling in combination with SVC Cursor is able to outperform AVC MSS in terms of average quality when delay increases, while yielding comparable buffer starvations and a lower number of switches (graphs omitted due to space limitations). This enables service providers to deploy SVC-based HAS services, benefitting from higher caching efficiency, while avoiding the drawbacks of SVC-based HAS in a high delay setting.

### A.6.4 Impact of parallel threads

The behavior of AVC MSS and SVC Cursor in combination with parallel scheduling is illustrated in Figure A.9. These results show that with increasing number of threads, parallel scheduling yields higher quality and lower switches for both AVC

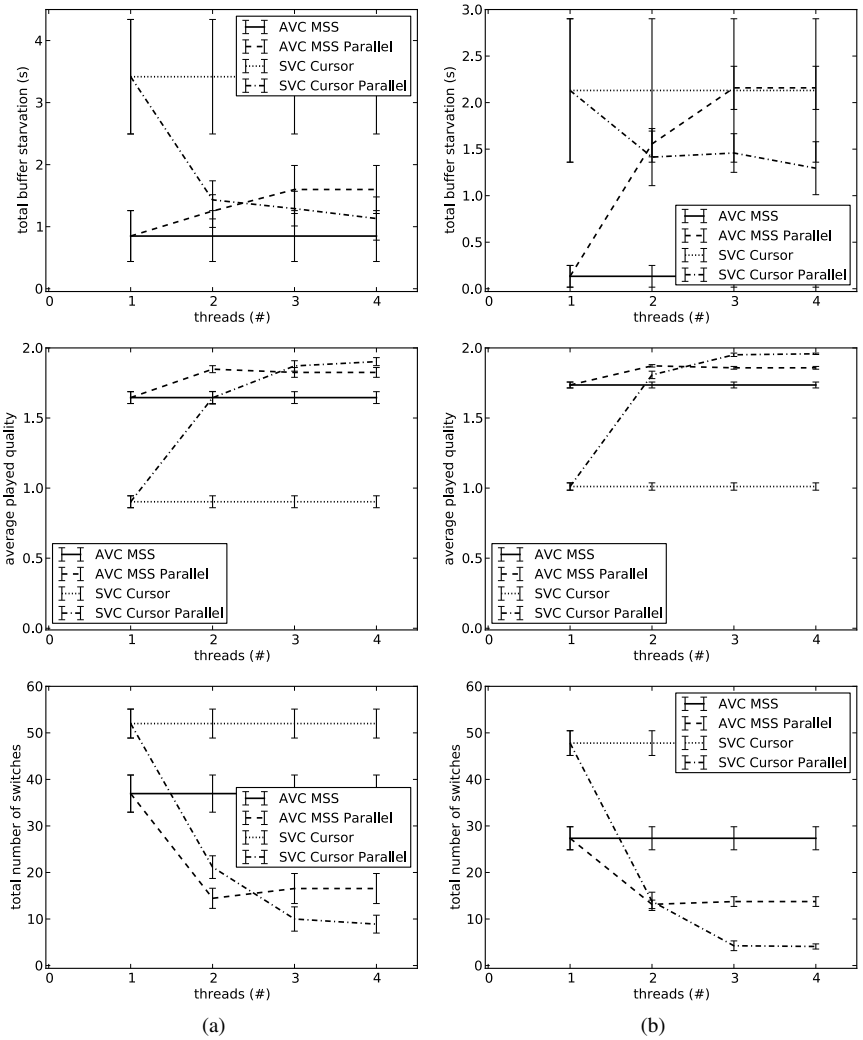



*Figure A.9: Impact of the number of parallel threads with $B_c = 10Mbps$, $N = 20$, $R = 100ms$, $P = 4.4s$ and (a) $B_s = 50Mbps$ (b) $B_s = 100Mbps$, for AVC MSS and SVC Cursor in combination with sequential and parallel scheduling.*

MSS and SVC Cursor. However, AVC MSS suffers from higher gap time with an increasing number of parallel threads, while SVC Cursor is able to lower the total buffer starvation time, while increasing quality and minimizing the number of switches.

It can be concluded that for Live TV, the smallest possible buffer should at least contain 2 segments to attain acceptable service quality. The evaluations have shown that the approach was indeed able to reduce the impact of high RTTs on SVC-based HAS by using parallel downloads. Furthermore, the novel SVC Cursor heuristic is able to outperform existing heuristics when considering a Live TV setting with small buffers.

## A.7 Conclusion

This appendix quantitatively reveals the drawbacks of using Scalable Video Coding (SVC) based HTTP Adaptive Streaming (HAS) Live streaming in high delay networks and proposes techniques to tackle these issues. The root causes of these are 1) the encoding overhead, yielding larger data transfers to attain the same quality as with Advanced Video Coding (AVC) based HAS and 2) the increased request-response rate in SVC-based HAS, which as a result of the high Round Trip Times (RTTs), leads to inefficient use of the available bandwidth. This appendix proposes to overcome the second issue by using HTTP Pipelined and Parallel download schedulers, eliminating the idle time between two consecutive downloads. Next to the schedulers, this appendix also proposes a cursor based SVC client heuristic, which outperforms existing SVC-based heuristics for small buffer sizes. The experimental results have shown that even with small buffer sizes, the combination of SVC Cursor with parallel scheduling is not only able to overcome the issues in high delay networks, but is even capable of achieving higher quality with less frequent switches than AVC-based HAS. For a buffer size of 3 segments, parallel and pipelined scheduling improve the quality with about 14%, while reducing the number of switches with a factor 4 compared to the sequential scheduling. Hence, the combination of the proposed novel SVC Cursor heuristic in combination with parallel download scheduling is able to outperform the state of the art heuristics by alleviating the drawbacks of SVC-based HAS while retaining the higher decision granularity and other advantages.

# References

[1] 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 Proceedings of the seventh IFIP/IEEE International Symposium on Integrated Network Management, pages 419–426, may 2013.

[2] R. Huysegems, B. De Vleeschauwer, T. Wu, and W. Van Leekwijck. *SVC-Based HTTP Adaptive Streaming*. Bell Labs Technical Journal, 16(4):25–41, 2012.

[3] Microsoft. *Microsoft Smooth Streaming: The Official Microsoft IIS Site*. Available from: http://www.iis.net/download/SmoothStreaming.

[4] R. Pantos and W. May. *HTTP Live Streaming*, 2012. Available from: http://tools.ietf.org/html/draft-pantos-http-live-streaming-10.

[5] Adobe. *HTTP Dynamic Streaming: Flexible Delivery of on-demand and live video streaming*. Available from: http://www.adobe.com/products/httpdynamicstreaming/.

[6] T. Stockhammer. *Dynamic adaptive streaming over HTTP: standards and design principles*. In Proceedings of the second annual ACM conference on Multimedia systems, MMSys '11, pages 133–144, 2011.

[7] H. Schwarz, D. Marpe, and T. Wiegand. *Overview of the scalable video coding extension of the H.264/AVC standard*. In IEEE Transactions on Circuits and Systems for Video Technology In Circuits and Systems for Video Technology, pages 1103–1120, 2007.

[8] H. Choi, J. Nam, D. Sim, and I. Bajic. *Scalable video coding based on high efficiency video coding (HEVC)*. In Proceedings of Communications, Computers and Signal Processing (PacRim), 2011, pages 346 –351, aug. 2011.

[9] C. Liu, I. Bouazizi, and M. Gabbouj. *Parallel Adaptive HTTP Media Streaming*. In Proceedings of 20th International Conference on Computer Communications and Networks, pages 1–6, August 2011.

[10] N. Bouten, J. Famaey, S. Latré, R. Huysegems, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. *QoE optimization through in-network quality adaptation for HTTP Adaptive Streaming*. In Proceedings of the 8th International Conference on Network and Service Management (CNSM), pages 336–342, 2012.

[11] N. Bouten, S. Latré, W. Meerssche, B. Vleeschauwer, K. Schepper, W. Leekwijck, and F. Turck. *A Multicast-Enabled Delivery Framework for QoE Assurance of Over-The-Top Services in Multimedia Access Networks*. Journal of Network and Systems Management, 21:677–706, 2013.

[12] N. Bouten, S. Latré, W. Van De Meerssche, K. De Schepper, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. *An autonomic delivery framework for HTTP Adaptive Streaming in multicast-enabled multimedia access networks*. In Proceedings of the Fifth IFIP/IEEE Workshop on Distributed Autonomous Network Management Systems (DANMS), 2012, pages 1248–1253. IEEE, 2012.

[13] S. Akhshabi, A. Begen, and C. Dovrolis. *An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP*. Proceedings of the second annual ACM conference on Multimedia systems, pages 157–168, 2011.

[14] C. Liu, I. Bouazizi, M. M. Hannuksela, and M. Gabbouj. *Rate adaptation for dynamic adaptive streaming over HTTP in content distribution networks*. Signal Processing: Image Communication, 27(4):288 – 311, 2012.

[15] V. Adzic, H. Kalva, and B. Furht. *Optimized adaptive HTTP streaming for mobile devices*. In SPIE Optical Engineering+ Applications. International Society for Optics and Photonics, 2011.

[16] J. Jiang, V. Sekar, and H. Zhang. *Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE*. Technical report, Carnegie Mellon University, 2012.

[17] T. Schierl, Y. Sanchez de la Fuente, R. Globisch, C. Hellge, and T. Wiegand. *Priority-based Media Delivery using SVC with RTP and HTTP streaming*. Multimedia Tools and Applications, 55:227–246, 2011.

[18] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala. *Quality Selection for Dynamic Adaptive Streaming over HTTP with Scalable Video Coding*. In Proceedings of the 3rd Multimedia Systems Conference, pages 149–154. ACM, 2012.

[19] D. Kaspar, K. Evensen, P. Engelstad, and A. Hansen. *Using HTTP pipelining to improve progressive download over multiple heterogeneous interfaces*. In Proceedings of the IEEE International Conference on Communications (ICC), pages 1–5, 2010.

# B

# Deadline-based Approach for Improving Delivery of SVC-based HTTP Adaptive Streaming Content

**N. Bouten, M. Claeys, S. Latré, J. Famaey, W. Van Leekwijck, F. De Turck.**

**Published in Proceedings of 2014 IEEE Network Operations and Management Symposium (NOMS), May. 2014.**

*While Appendix A focussed on minimizing the impact of Round Trip Time (RTT) on live HTTP Adaptive Streaming (HAS) services, this appendix focusses on the optimization of the delivery of live Scalable Video Coding (SVC) HAS. To guarantee continuous playback, current-generation HAS protocols require a large play-out buffer. This makes them ill-suited for live television, as it significantly increases the camera-to-display delay. This appendix proposes a novel HAS solution for live streaming services. A HAS video player was designed that can cope with buffers as small as 2 seconds. To achieve this, an intelligent network proxy was developed that guarantees the delivery of the SVC base quality layer using Differentiated Services (DiffServ). Furthermore, a more dynamic deadline-based approach is proposed which allows the client itself to decide which segments should be prioritized based on the risk of running into a buffer starvation. The combination of these technologies allows the video player to align its quality adaptation decisions to the available bandwidth more efficiently and avoid buffer starvations.*

## B.1 Introduction

Over the past decades, the importance of multimedia services such as video streaming has increased considerably. This growth is projected to exceed 90 percent of the Internet traffic by 2017 [1], causing video traffic to dominate the Internet. In the past, the Real Time Streaming Protocol (RTSP) and Real Time Transport Protocol (RTP) were used commercially to deliver video over IP networks. Since these protocols require server-side bit-rate adaptation schemes, they are not ideally suited to deal with highly heterogeneous and dynamically changing network conditions. Therefore, research and academia began shifting towards client-side adaptation schemes which have the benefit of distributing the rate decision and thus requiring significantly less investments in server-side infrastructure. HTTP Adaptive Streaming (HAS) is now becoming omnipresent in video streaming services due to many advantages offered by HTTP-based streaming: reliable transmission over TCP, reuse of existing caching infrastructure and compatibility with NATs and firewalls.

In HAS, the video content is split temporally into segments which are encoded at different quality rates. The client side heuristic decides at which quality rate each segment should be downloaded, based on measured network statistics, buffer filling level and device characteristics. This allows HAS to respond to throughput fluctuations by reducing the quality and continuing video playout, whereas previous HTTP-based streaming techniques would have run into a buffer starvation. This allows the client to independently choose its playback quality and prevents the need for intelligent components inside the network, which is a major advantage in large-scale Over-The-Top (OTT) scenarios. Traditionally, Advanced Video Coding (AVC) is used to encode the different segments, introducing a significant amount of redundancy across quality representations. Scalable Video Coding (SVC) can cope with these issues of content redundancy by creating dependencies between the base and enhancement layers. Adopting SVC in HAS significantly improves caching and bandwidth efficiency at the server side and allows gradual upgrading of the video quality by downloading additional video layers.

Academia and industry are showing a growing interest in the use of HAS for managed networks. The extensive content catalogue and increased flexibility in terms of supported devices of OTT-services (e.g., YouTube, Hulu, Netflix) but delivered over a managed network, could greatly benefit both provider and end-user. Delivering such paid HAS services, however, requires the ability for the provider to offer guarantees in terms of reliability to the end-users. Since HAS services are originally designed for OTT scenarios, there are some changes required in the network to allow such guarantees.

To guarantee continuous playback, current-generation HAS protocols require a large play-out buffer. This makes them ill-suited for live television, as it signifi-

cantly increases the live signal delay. Shrinking the buffer size has however some implications on the reliability of the streaming service and requires the player to react in a fast and robust way to changes in the network. Therefore, this appendix proposes a managed HAS multimedia framework, using SVC and prioritization in the network, which allows to offer a reliable video streaming service. It takes advantage of the ability to gradually improve quality which is present in SVC-based HAS to deliver the base layer with higher priority to the clients. This way, when congestion arises in the managed network, the continuous playout of the base layer can be guaranteed, allowing the provider to offer a reliable service. This appendix also proposes a client-side heuristic that decides which request should be scheduled with higher priority based on the playout deadline for that particular segment. This approach is more dynamic and allows clients to decide for themselves which segments are more important without introducing additional state in the network.

The remainder of this appendix is structured as follows. Section B.2 provides an overview of relevant work, followed by an illustration of how differentiated services can contribute to the reliability and stability of SVC-based adaptation heuristics in Section B.3. Section B.4 elaborates on the experiment setup and evaluation results, which are summarized in Section B.5.

## B.2 Related Work

The increased popularity of video consumption over the Internet has led to the development of a range of protocols that allow adaptive video streaming over HTTP. Some of the major industrial players have introduced their proprietary protocols such as Microsoft's Silverlight Smooth Streaming[1], Apple's HTTP Live Streaming[2] and Adobe's HTTP Dynamic Streaming[3]. More recently, a standardized solution has been proposed by MPEG, called Dynamic Adaptive Streaming over HTTP (DASH) [2]. Although differences exist between these implementations they are based on the same basic principles: a video is split up into temporal segments which are encoded at different quality rates, the client rate adaptation algorithm then dynamically adapts the quality, based on metrics such as average throughput, delay and jitter.

Optimizations of HAS-based delivery can be performed at the server, the network or the client. At the server side, optimizations are focussed on the encoding scheme. Traditional deployments of HAS use the H.264/AVC codec for creating the different representations of the video. For each representation, a separate file needs to be stored at the video server, leading to an increased storage penalty due

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

[2] Apple HTTP Live Streaming - http://tools.ietf.org/html/draft-pantos-http-live-streaming-12

[3] Adobe HTTP Dynamic Streaming - http://www.adobe.com/products/hds-dynamic-streaming.html

to the redundant information. Adopting a scalable extension to H.264/AVC [3] or High Efficiency Video Coding (HEVC) [4], allows alleviating the storage issues with AVC at the server, while improving caching efficiency. *Huysegems et al.* [5] discuss the advantages of using SVC instead of AVC, such as more stable playout during fluctuations since the base layer is always downloaded first and a reduction in bandwidth and storage requirements at the server. Using SVC in HAS also has some drawbacks caused by the encoding overhead, the need to download multiple layers to increase the quality of a single segment and the vulnerability to high round trip times [6]. These problems can be avoided by using HTTP pipelining or parallel downloads to minimize the impact of high round trip times [7]. In this appendix the reliability of SVC-based HAS systems is further improved by delivering the base layer segments with high priority allowing the continuous playout of at least the base layer.

*Liu et al.* [8] present an in-network optimization of HAS for 3GPP networks. By parallelizing the download and request of HAS segments, a better resource utilization can be achieved. An autonomic delivery framework is presented in previous work [9, 10], which allows to reduce the consumed bandwidth by grouping unicast HAS sessions sharing the same content into a single multicast session. *Schierl et al.* [11] present an overview of interesting use cases for applying SVC in a network environment, among which the graceful degradation of videos when the network load increases. 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. Similar to this approach, *Latré et al.* [12] propose an in-network rate adaptation algorithm, responsible for determining which SVC quality layers should be dropped in combination with a Pre-Congestion Notification (PCN) based admission control mechanism. *Hsiao et al.* [13] propose a prototype of an intermediary adaptation node, where the media gateway estimates the available bandwidth on the client link and extracts the supported SVC-streams. *Akhshabi et al.* [14] propose a server-side traffic shaping approach to avoid ON-OFF behavior when multiple clients compete to minimize oscillations during streaming.

Each commercial HAS implementation comes with an existing video client heuristic of its own. *Akhshabi et al.* [15] compare several commercial and open source HAS players and indicate significant inefficiencies in each of them. Several heuristics have been proposed in literature as well, each focussing on a specific deployment. *Liu et al.* [16] discuss a video client heuristic that is suited for CDNs by comparing the expected segment fetch time with the experienced segment fetch time to ensure a response to bandwidth fluctuations in the network, while *Adzic et al.* [17] present a client heuristic which is tailored for mobile environments. *Jiang et al.* [18] aimed to develop an efficient, fair and stable heuristic by randomizing chunk scheduling to avoid synchronization, stateful bitrate selection and delayed

update to avoid instability, and compared their approach to commercial players. Generic algorithms exist for selecting the next video quality to download, using a priority-based scheme where base layers receive higher priority in download scheduling compared to the enhancement layers [19]. We also use a comparable scheduling technique at the client and go beyond this, by increasing the priority of the base layer segments in the network. *Andelin et al.* [20] provide a heuristic which was specifically designed for SVC and using a slope to define the trade-off between downloading the next segment and upgrading a previously downloaded segment.

## B.3 Priority-Based Delivery of HAS

Several clients for which the video flows traverse the same path in the network are competing with each other, resulting in a fair-share bandwidth between them. An implication of this TCP behavior is that urgent requests from a client risking to run into a buffer starvation are treated in the same way as requests of clients that have already built up a sufficiently large buffer. If congestion arises in the network, both of the clients will experience a delayed arrival of the requested segment. For the first client however, this causes the buffer to deplete, resulting in a frame freeze, which is the main factor responsible for lowering the Quality of Experience (QoE) [21].

For live TV, also the latency with respect to the live signal is an important aspect of QoE and interactivity [22, 23]. If a frame freeze occurs, the latency on the live signal increases with the length of the frame freeze. This emphasizes the importance of avoiding frame freezes at any time to guarantee an acceptable QoE for live video streaming. Most of the client rate adaptation heuristics tackle this problem by requiring a fairly large buffer size (10s-30s). However, at the same time they increase the latency on the live signal with the buffer length, negatively impacting QoE. This indicates the trade-off between minimizing the latency on the live signal and improving reliability that exists in a live streaming framework. There are however several ways to achieve higher reliability while decreasing the buffer size, both from within the network and by the client, when allowing in-network prioritization.

### B.3.1 Layer-based prioritization

A first way to improve reliability in SVC-based HAS systems is to apply prioritization based on the representation level. This can be established by interpreting the requests at the proxy, classifying them into different priority classes and mark the packets based on this priority. If one chooses to deliver the base layer representation of the video with highest priority, clients with limited buffer filling will

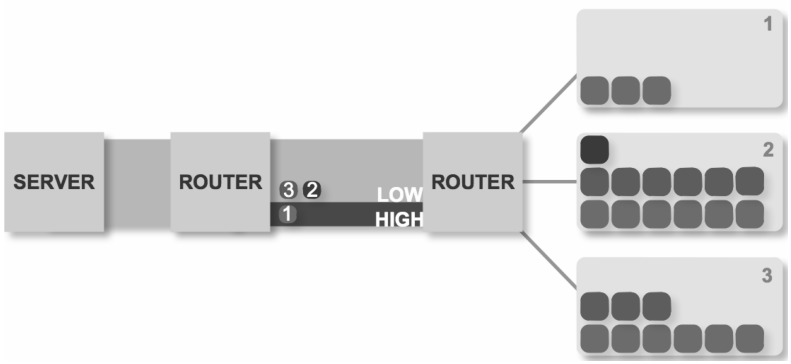



*Figure B.1: Illustration of layer-based prioritization, where lower quality presentations have higher priority than enhancement layers.*

be able to continue playout at the lowest quality when congestion arises. This is illustrated in Figure B.1, where the delivery of the base layer for client 1 will have higher priority than the delivery of the enhancement layers for client 2 and 3 respectively. Although this approach improves reliability, the decision granularity is limited. The layers that are prioritized are chosen in a static way and the approach requires sufficient dimensioning of the high-priority bandwidth share.

The SVC-based client heuristic that was presented in previous work [7] was extended to take advantage of the in-network prioritization. More specifically, the current approach differentiates between the download behavior of the prioritized layers and the best-effort layers by performing them in different download threads. The prioritized download scheduler starts off by filling the buffer, after which it goes into steady state and periodically (every segment length) downloads an additional segment. There are two parameters that can be set for the prioritized deadline scheduler: the number of representations that will be sent with higher priority and whether they are downloaded sequentially or in parallel. The best effort download scheduler uses the SVC Cursor client heuristic to download additional segments over the best effort channel, taking into account the current network conditions and buffer filling [7].

### B.3.2 Deadline-based prioritization

The previous approach requires prioritized representations to be statically configured and implies that the selected segments will always be sent with higher priority. Although this allows to provide a reliable SVC-based streaming service, segments are often sent with high priority, while this was not necessary due to a high buffer filling. Allowing the client to decide for itself which segments should be downloaded with higher priority can alleviate this problem and guarantee that

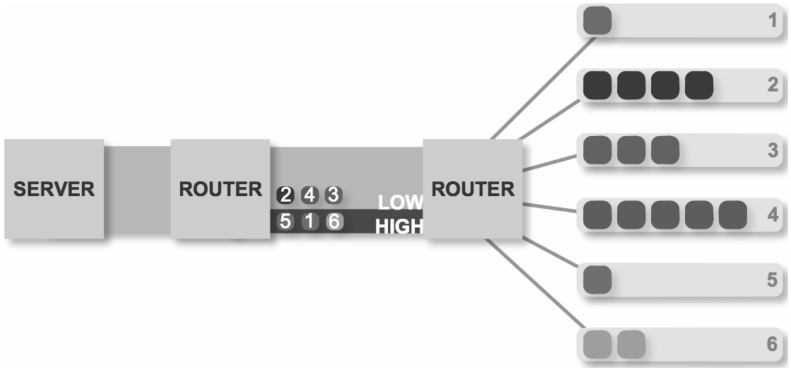



*Figure B.2: Illustration of deadline-based prioritization, where clients decide based on their current buffer filling to request segments with higher priority.*

the prioritized channel is only used when it is required. This allows more efficient use of prioritization while still improving the reliability of the streaming service.

Using deadline-based prioritization, the client decides which segments are to be delivered with higher priority. Figure B.2 illustrates how clients that are close to running into a buffer starvation, request segments to be delivered with higher priority leading to a continuous playout. To achieve such behavior, a previously presented SVC-based client heuristic [7] was extended with several additional features. First, downloads that are late for playout due to congestion are cancelled by closing the TCP-socket. Second, if the buffer filling level is beneath a certain threshold, the base layer segments are requested with high priority. Third, in parallel to the regular downloads, it is checked periodically if the base layer of the next segment is available for playout and if required download it in parallel with high priority. These extensions to the previous heuristic allow reliable HAS live streaming with small buffer sizes while guaranteeing continuous playout.

## B.4 Evaluation

### B.4.1 Prototype implementation

A proof-of-concept prototype of the priority-based delivery framework has been implemented. The HAS Server was implemented using Apache[4] and allows generating both VoD and Live streaming manifests. Furthermore, a script was used to periodically release and synchronize the released manifest files, allowing us to compare the proposed adaptive streaming framework with existing HAS protocols which have other settings in terms of, amongst others, buffer size or segment size.

[4]Apache HTTP server project - http://httpd.apache.org/

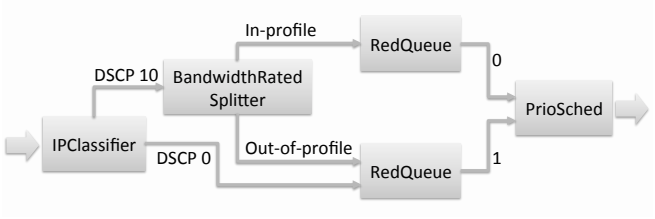



*Figure B.3: Click-implementation of AF behavior.*

The proxy is responsible for handling requests from the clients and forward them to the server. Furthermore, the proxy classifies these requests in different priority classes and marks the packets accordingly. The classification is based on regular expressions executed on the HTTP request headers for the layer-based prioritization and on the port number of the request for the deadline-based prioritization. The deadline-based prioritization could also be embedded into the HAS protocol. The proxy was implemented using Squid[5] version 3.1.6, which offers support for setting client-side TOS-fields. To prioritize the traffic, Differentiated Services (DiffServ) [24] has been used. DiffServ is a simple, scalable mechanism for classifying and managing IP network traffic. DiffServ applies the principle of traffic classification, placing each data packet into a limited number of traffic classes. For this purpose, the 6-bit DS Field of the IP header is used. DiffServ-aware routers each implement Per Hop Behaviors (PHBs), defining the packet forwarding rules for every traffic class. Next to the default PHB, which is typically best-effort traffic, the Assured Forwarding (AF) [25] and Expedited Forwarding (EF) [26] PHBs are commonly used with DiffServ. While EF traffic is often given strict priority queuing above all other traffic classes, inducing low delay, the AF PHB, allows the operator to assure the delivery of this traffic class as long as it does not exceed some subscribed rate. When traffic exceeds the subscription rate, it faces a higher drop probability when congestion occurs. The AF PHB has been implemented in a Click[6] router for this proof-of-concept as shown in Figure B.3. A combination of an IPClassifier, a BandwidthRatedSplitter, two Random Early Detection (RED) queues and a PrioSched module were used to implement this behavior. As long as the prioritized traffic is lower than the profile rate, the traffic is marked as in-profile, otherwise it is marked as out-of-profile. When congestion arises, the out-of-profile traffic is dropped more aggressively than in-profile traffic. In absence of congestion, the prioritized traffic is allowed to exceed its profile rate.

The clients were based on a previous implementation [6] and replaced the net-

[5]Squid cache - http://www.squid-cache.org/
[6]Click Modular Router - http://www.read.cs.ucla.edu/click/click

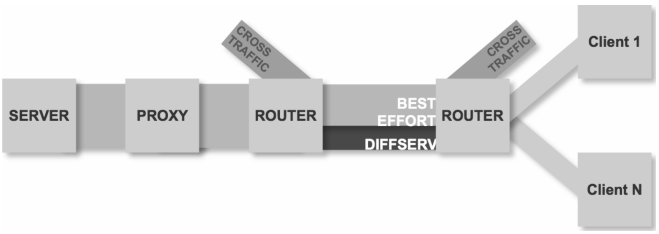



*Figure B.4: Experimental setup showing the PHB and introduction of best effort cross traffic on the bottleneck link.*

working modules by cURL-based[7] components. The clients support both SVC and AVC video streaming and allow using different client heuristics. Furthermore they allow configuring the scheduler as well: the number of simultaneous download threads or installing a pipelined download scheduler as described in previous work [27]. An additional download scheduler was added to be able to take advantage of the in-network prioritization. At the client, the segments are streamed to a GStreamer[8]-based decoding and visualization plugin. The SVC decoding is handled by an adapted implementation of Open SVC Decoder[9].

## B.4.2 Experiment setup

Figure B.4 shows the different components of the HAS delivery network and how they are interconnected. Click has also been used to inject best effort cross traffic into the network that competes with the best effort video streaming. Using this cross traffic, an available best-effort bandwidth with a realistic degree of variability was modelled, based on a bandwidth trace described by *Riiser et al.* [28]. The traces, measured on the bus path between Ljan and Oslo central station, Norway[10], have a total duration of about 220 minutes. The available bandwidth fluctuates between 202bps and 6335kbps with an average of 2192kbps and a standard deviation of 1317kbps. In the experiments, the described bandwidth trace has been scaled to match a fixed upper limit, based on the number of clients, by multiplying the values with $\frac{R}{6335kbps}$, with $R$ the bandwidth of the link. The bandwidth trace was then cut in 400 second parts which were used during the evaluations. Figure B.5 shows an example of such a bandwidth trace excerpt of 400 seconds, which was inverted to serve as cross traffic on a 20Mbps link.

To allow extended evaluation of the framework, the setup was replicated in

[7] cURL - http://curl.haxx.se/

[8] GStreamer - http://gstreamer.freedesktop.org

[9] Open SVC Decoder - http://sourceforge.net/projects/opensvcdecoder/

[10] Dataset available from: http://home.ifi.uio.no/paalh/dataset/hsdpa-tcp-logs/bus.ljansbakken-oslo/

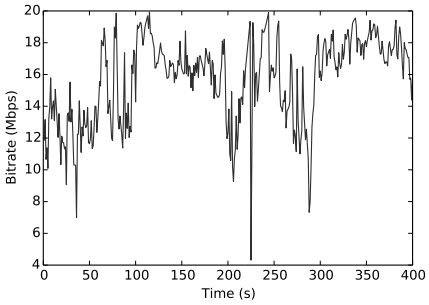



*Figure B.5: Excerpt of a cross traffic file for a 20Mbps link.*

*Table B.1: Bitrates of the different video layers.*

| | AVC | | SVC | | |
|---|---|---|---|---|---|
| | avg (bps) | max (bps) | avg (bps) | sum (bps) | max (bps) |
| layer 0 | 806377 | 4868448 | 773473 | 773473 | 4677288 |
| layer 1 | 3184253 | 16118368 | 2756137 | 3529610 | 12626760 |
| layer 2 | 5809166 | 28276704 | 4269023 | 7798633 | 18823272 |

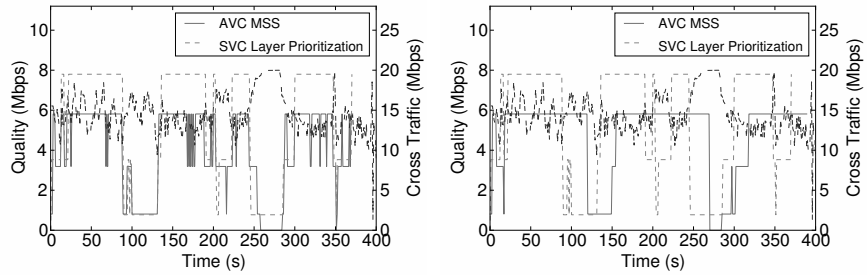
simulation. The simulation framework is based on ns-3[11] which also offers support for Click[12]. The experiments used a Variable Bitrate (VBR) video of 400 seconds encoded in three layers for both AVC and SVC. The video has both slow-moving scenes and scenes with high motion variability, which results in the bitrate characteristics shown in Table B.1. The encoding of the AVC and SVC video segments was perfomed using the reference encoders from the JM[13] and JSVM[14] test models respectively, which are provided by the Joint Video Team. Three different rate points are used, the base layer is encoded at a spatial resolution of 480p, with a fixed Quantization Parameter (QP) of 35. The first enhancement layer applies MGS quality scalability at the same resolution with a QP of 25. The highest layer is a spatial enhancement layer of 720p resolution with a QP of 25. The same resolutions, QPs and other encoding configuration parameters were applied to AVC encoding. This results in the video quality of the AVC and SVC streams being equal for equivalent rate points. However, because of the overhead introduced by the scalability in SVC, the bit rate of AVC segments will be slightly lower. The frame rate for each layer is 25 fps and to achieve the highest coding efficiency for SVC one I-frame is introduced every 25 frames. This leads to segments with a length of one second. For AVC, the same configuration was used.

[11] NS-3 - http://www.nsnam.org

[12] Ns-3-click and nsclick - http://www.read.cs.ucla.edu/click/nsclick

[13] H.264/AVC JM Reference Software - http://iphome.hhi.de/suehring/tml/

[14] JSVM Reference Software - http://www.hhi.fraunhofer.de/de/kompetenzfelder/image-processing/research-groups/image-video-coding/svc-extension-of-h264avc/jsvm-reference-software.html




(a) (b)

*Figure B.6: Illustration of the continuous playout under cross traffic when SVC layer-based prioritization is enabled. With segment length of 1s and buffer size 2s (a) and 20s (b) for AVC MSS and buffer size 2s for SVC layer prioritization for a single streaming client.*

### B.4.3 Impact of layer-based prioritization

During these experiments, the base layer was prioritized over the enhancement layers. The evaluations used one of the traces mentioned earlier to introduce best effort cross traffic into the network. Figure B.6 illustrates the benefits of using a combination of SVC-based HAS and DiffServ for a single client and segment length of 1s. The SVC layer prioritization allows a continuous playout with a buffer of only 2s, while the AVC Microsoft Smooth Streaming (MSS) algorithm freezes for about 27s and 15s respectively for a buffer size of 2s and 20s. Even a buffer of 20s is not able to protect the client against prolonged throughput changes. These results show that allowing prioritization of base layer segments in the network can increase the stability of a HAS streaming client, allowing us to shrink the buffer to only 2 seconds, significantly decreasing the latency on the live signal.

During the following evaluations, the number of clients was increased up to 20, while the bandwidth for the link between both routers was increased to 400Mbps. The bandwidth of the prioritized channel was varied during these experiments, as well as various parameter settings for the algorithms. Each of the experiments was repeated 30 times, using different parts of the aforementioned bandwidth trace and averaged the results over these iterations.

Increasing the number of parallel downloads can increase the efficiency of the SVC Cursor heuristic, especially when network delay increases, as was shown in previous work [27]. Figure B.7 shows the impact of parallelizing the base layer downloads on the average buffer size, the average freezing time, the average played quality rate and the number of switches. The buffer starvations can be reduced to zero if the number of parallel threads is increased to 3. Since the video bitrate is highly variable at times, parallellizing downloads smoothens out the download process over time, preventing sudden video bitrate variations to cause late down-

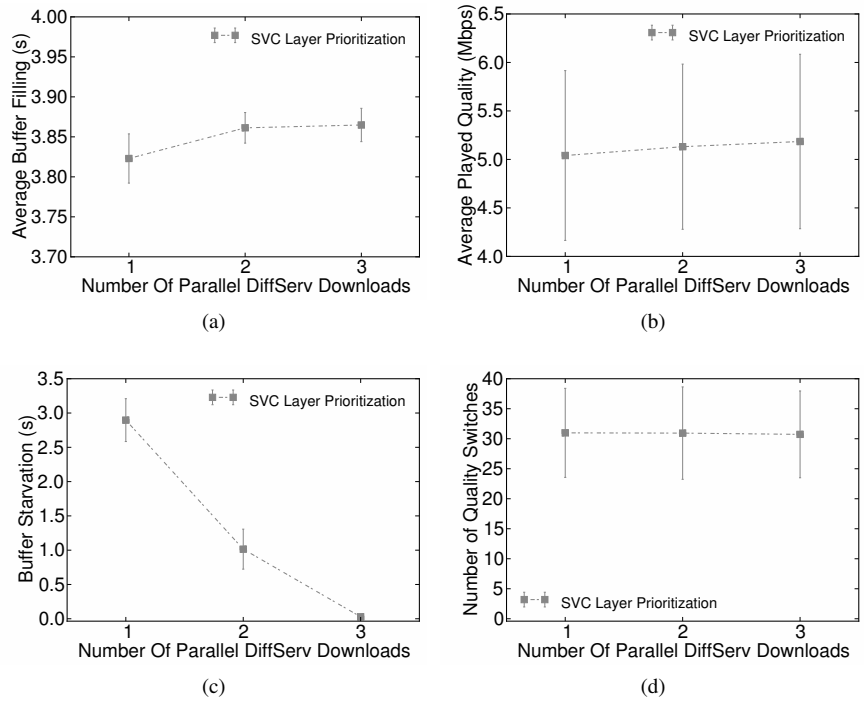



*Figure B.7: Impact of the number of parallel DiffServ downloads on the SVC Layer Prioritization for a buffer of 4 seconds and a prioritized channel of 100Mbps.*

loads. The slightly improved playout quality can be accounted to the higher performance of parallel TCP downloads, which also allows a higher average buffer filling.

### B.4.4 Impact of deadline-based prioritization

Although the quality of both the AVC and SVC representations are equal, their bitrate differs due to the SVC overhead. Therefore, the estimated Mean Opinion Score (MOS) is used to evaluate the impact on quality as presented by *De Vriendt et al.* where the QoE is calculated as a weighted combination of the average delivered quality ($\mu$) and the standard deviation of quality ($\sigma$) [29], extended with a correction factor to incorporate the impact of frame freezes ($\phi$) which is a continuous interpolation of the discrete levels proposed by *Mok et al.* [21].

$$eMOS = \alpha * \mu - \beta * \sigma - \gamma * \phi + \delta \tag{B.1}$$

The parameters ($\alpha = 5.67$, $\beta = 6.72$, $\gamma = 4.95$ and $\delta = 0.17$) were tuned by minimizing the Root Mean Squared Error (RMSE) between what the model predicts and the measured subjective MOS values obtained by a subjective screening test with 10 experts in the field of video streaming. The formula's for calculating the values $\mu$, $\sigma$ and $\phi$ are calculated as follows, with $K$ the number of played segments, $N$ the number of quality levels for video, $Q_k$ the quality played for segment $k \in [1, K]$ and $F$ the set of frame freezes:

$$\mu = \frac{\sum_{k=1}^{K} \frac{Q_k}{N}}{K} \tag{B.2}$$

$$\sigma = \sqrt{\frac{\sum_{k=1}^{K} (\frac{Q_k}{N} - \mu)^2}{K}} \tag{B.3}$$

$$F_{freq} = \frac{|F|}{K} \tag{B.4}$$

$$F_{avg} = \frac{\sum_{f \in F} duration(f)}{|F|} \tag{B.5}$$

$$\phi = \frac{7}{8} \max\left(\frac{\ln(F_{freq})}{6} + 1, 0\right) + \frac{1}{8} \min\left(\frac{F_{avg}}{15}, 1\right) \tag{B.6}$$

Figure B.8 shows the impact of the buffer size on the average buffer starvations, estimated QoE as defined by Equation B.1 and number of switches for both the AVC MSS and SVC prioritization-based heuristics for a prioritized channel of 150Mbps and 200Mbps respectively. These results show how applying prioritization in the network can significantly reduce the number of buffer starvations

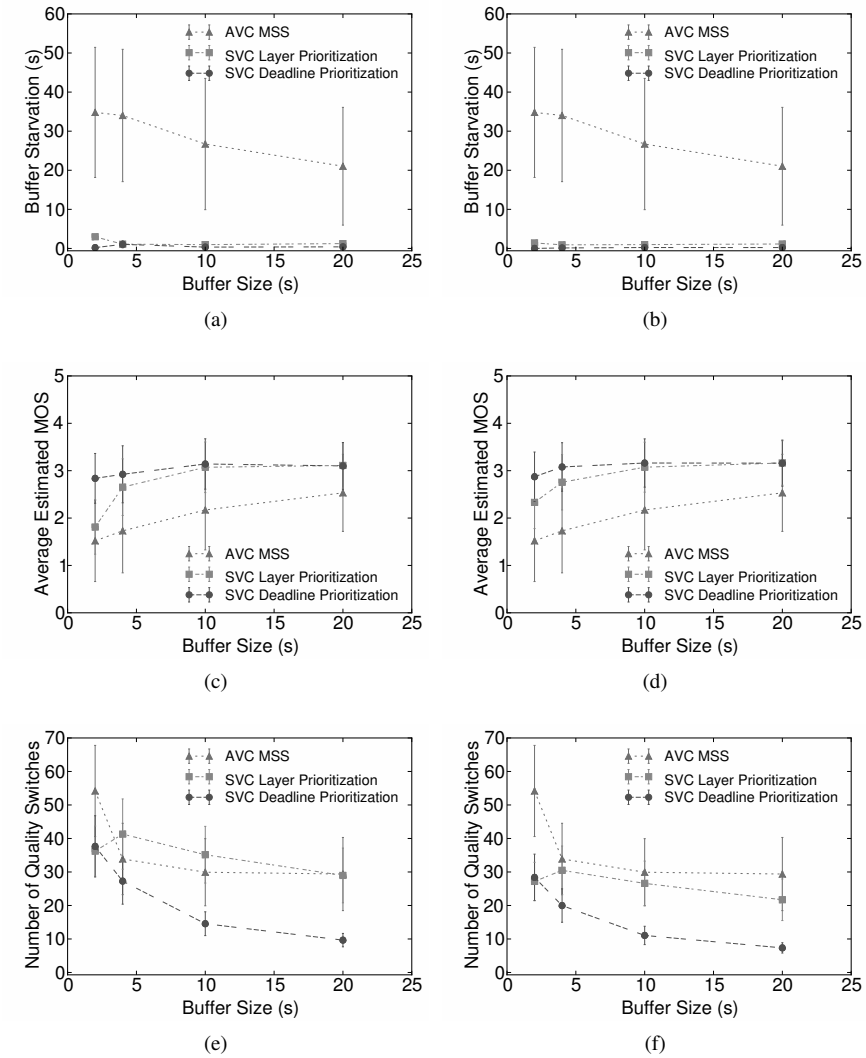



*Figure B.8: Impact of the buffer size on average freezing time, estimated MOS and number of switches for a prioritized channel of (a) 150Mbps and (b) 200Mbps respectively.*

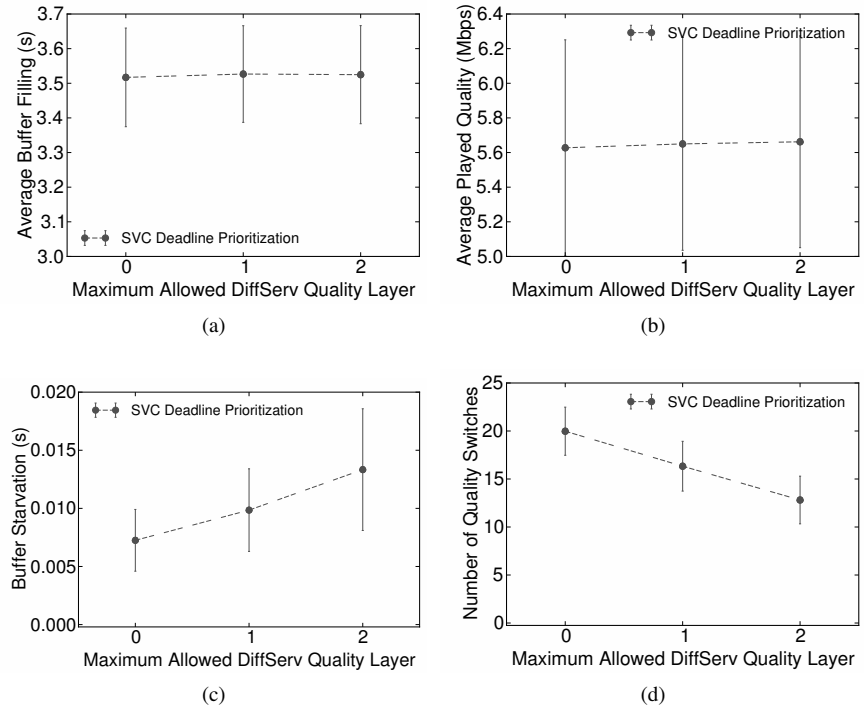



*Figure B.9: Impact of the number of allowed qualities for Deadline Based Prioritization on average buffer filling, freezing time, quality rate and number of switches for a prioritized channel of 200Mbps and buffer size of 4 seconds.*

both for small and larger buffers. It is also shown that the proposed approach also increases the average estimated MOS, while at the same time reducing the number of switches. Furthermore, the deadline-based prioritization outperforms the layer-based prioritization in terms of average freezes and playout quality. This can be accounted to the fact that the deadline-based prioritization only requests certain segments over the prioritized channel when this is really required, while the layer-based prioritization continuously occupies the prioritized channel. For a prioritized channel of 150Mbps, this can lead to freezes, since the limited bandwidth is sometimes not sufficient to transfer the high variable video segments for all 20 clients at the same time.

The deadline-based prioritization can be configured to download only the base layer or various enhancement layers with higher priority when the buffer filling drops below a certain threshold. Figure B.9 shows the impact of allowing to download enhancement layers with higher priority, leading to a slight improvement in playout quality and reducing the number of switches from 20 to 12. The higher

quality stability however comes at a cost. Due to the higher occupation of the prioritized channel, prioritized base layer segment downloads now also compete with enhancement layer segment downloads, leading to a slight increase in average freezing time.

These results show that deadline-based prioritization allows SVC-based clients to achieve higher quality with less switches, while assuring a reliable HAS service. Extending the allowed quality range from only the base layer to enhancement layers has a limited positive impact on the quality and stability of the streaming, but increases the risk of running into buffer freezes.

## B.5 Conclusion

To guarantee continuous playback, current-generation HTTP Adaptive Streaming (HAS) protocols require a large play-out buffer. This makes them ill-suited for live television, as it significantly increases the live signal delay. This appendix tackles the reliability issues that arise when shrinking the buffer by deploying a combination of Scalable Video Coding (SVC)-based video and prioritization in the network. A layer-based prioritization scheme is proposed, where only certain layers of the video are delivered with high priority. This approach allows us to increase the reliability of HAS services when dimensioning the prioritized channel appropriately. A more dynamic deadline-based approach allows the client itself to decide which segments should be prioritized based on the risk of running into a buffer starvation. This allows more efficient use of the prioritized channel, leading to less freezes and increased quality and stability.

# References

[1] C. V. Forecast. *Cisco Visual Networking Index: Global Mobile data Traffic Forecast Update 2012-2017*. Technical report, Cisco Public Information, May 2013.

[2] T. Stockhammer. *Dynamic adaptive streaming over HTTP: standards and design principles*. In Proceedings of the second annual ACM conference on Multimedia systems, MMSys '11, pages 133–144, 2011.

[3] H. Schwarz, D. Marpe, and T. Wiegand. *Overview of the scalable video coding extension of the H.264/AVC standard*. In IEEE Transactions on Circuits and Systems for Video Technology In Circuits and Systems for Video Technology, pages 1103–1120, 2007.

[4] H. Choi, J. Nam, D. Sim, and I. Bajic. *Scalable video coding based on high efficiency video coding (HEVC)*. In Proceedings of Communications, Computers and Signal Processing (PacRim), 2011, pages 346 –351, aug. 2011.

[5] R. Huysegems, B. De Vleeschauwer, T. Wu, and W. Van Leekwijck. *SVC-Based HTTP Adaptive Streaming*. Bell Labs Technical Journal, 16(4):25–41, 2012.

[6] 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 Proceedings of the seventh IFIP/IEEE International Symposium on Integrated Network Management, pages 419–426, may 2013.

[7] N. Bouten, S. Latré, J. Famaey, W. Van Leekwijck, and F. De Turck. *Minimizing the Impact of Delay on Live SVC-based HTTP Adaptive Streaming Services*. In Proceedings of the Sixth IFIP/IEEE Workshop on Distributed Autonomous Network Management Systems (DANMS), 2013, pages 1399–1404. IEEE, 2013.

[8] C. Liu, I. Bouazizi, and M. Gabbouj. *Parallel Adaptive HTTP Media Streaming*. In Proceedings of 20th International Conference on Computer Communications and Networks, pages 1–6, August 2011.

[9] N. Bouten, S. Latré, W. Meerssche, B. Vleeschauwer, K. Schepper, W. Leekwijck, and F. Turck. *A Multicast-Enabled Delivery Framework for QoE Assurance of Over-The-Top Services in Multimedia Access Networks*. Journal of Network and Systems Management, 21:677–706, 2013.

[10] N. Bouten, S. Latré, W. Van De Meerssche, K. De Schepper, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. *An autonomic*

*delivery framework for HTTP Adaptive Streaming in multicast-enabled multimedia access networks*. In Fifth IFIP/IEEE Workshop on Distributed Autonomous Network Management Systems (DANMS), 2012, pages 1248–1253. IEEE, 2012.

[11] T. Schierl, C. Hellge, S. Mirta, K. Gruneberg, 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, 2007. ISCAS 2007, pages 3455–3458. IEEE, 2007.

[12] S. Latré and F. De Turck. *Joint In-network Video Rate Adaptation and Measurement-Based Admission Control: Algorithm Design and Evaluation*. Journal of Network and Systems Management, 21:588–622, 2012.

[13] Y. Hsiao, S. Yeh, J. Chen, and Y. Chu. *A design of bandwidth adaptive multimedia gateway for scalable video coding*. In Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), pages 160–163. IEEE, 2010.

[14] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen. *Server-based traffic shaping for stabilizing oscillating adaptive streaming players*. In Proceedings of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, pages 19–24. ACM, 2013.

[15] S. Akhshabi, A. Begen, and C. Dovrolis. *An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP*. Proceedings of the second annual ACM conference on Multimedia systems, pages 157–168, 2011.

[16] C. Liu, I. Bouazizi, M. M. Hannuksela, and M. Gabbouj. *Rate adaptation for dynamic adaptive streaming over HTTP in content distribution networks*. Signal Processing: Image Communication, 27(4):288 – 311, 2012.

[17] V. Adzic, H. Kalva, and B. Furht. *Optimized adaptive HTTP streaming for mobile devices*. In SPIE Optical Engineering+ Applications. International Society for Optics and Photonics, 2011.

[18] J. Jiang, V. Sekar, and H. Zhang. *Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE*. Technical report, Carnegie Mellon University, 2012.

[19] T. Schierl, Y. Sanchez de la Fuente, R. Globisch, C. Hellge, and T. Wiegand. *Priority-based Media Delivery using SVC with RTP and HTTP streaming*. Multimedia Tools and Applications, 55:227–246, 2011.

[20] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala. *Quality Selection for Dynamic Adaptive Streaming over HTTP with Scalable Video Coding*. In Proceedings of the 3rd Multimedia Systems Conference, pages 149–154. ACM, 2012.

[21] R. K. Mok, E. W. Chan, and R. K. Chang. *Measuring the quality of experience of HTTP video streaming*. In Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 485–492. IEEE, 2011.

[22] H. Knoche and M. A. Sasse. *Getting the big picture on small screens: Quality of Experience in mobile TV*. Multimedia Transcoding in Mobile and Wireless Networks, pages 31–46, 2008.

[23] R. Stankiewicz and A. Jajszczyk. *A survey of QoE assurance in converged networks*. Computer Networks, 55(7):1459–1473, 2011.

[24] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An Architecture for Differentiated Services*. IETF RFC 2475, December 1998.

[25] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. *Assured Forwarding PHB Group*. IETF RFC 2597, June 1999.

[26] B. Davie, A. Charny, J. Bennett, K. Benson, J. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. *An Expedited Forwarding PHB (Per-Hop Behavior)*. IETF RFC 3246, March 2002.

[27] N. Bouten, S. Latré, J. Famaey, F. De Turck, and W. Van Leekwijck. *Minimizing the impact of delay on live SVC-based HTTP adaptive streaming services*. In Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 1399–1404, 2013.

[28] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. *Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications*. In Proceedings of the 4th ACM Multimedia Systems Conference, pages 114–118, February 2013.

[29] J. De Vriendt, D. De Vleeschauwer, and D. Robinson. *Model for estimating qoe of video delivered using HTTP adaptive streaming*. In Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 1288–1293. IEEE, 2013.

# C

# Semantic Validation of Affinity Constrained Service Function Chain Requests

**N. Bouten, M. Claeys, R. Mijumbi, J. Famaey, S. Latré, J. Serrat.**

**Published in Proceedings of 2016 IEEE Conference on Network Softwarization (NetSoft 2016), June. 2016.**

*Chapter 6 proposes to take advantage of Network Function Virtualization (NFV) technologies to increase the agility and shorten the time to market of new HTTP Adaptive Streaming (HAS) services. For various reasons (e.g., legislative, privacy, economic, efficiency and resilience reasons), the service provider may want to put constraints on the locality of the Virtual Network Functions (VNFs) and their interconnecting virtual edges. The chapter proposes a set of affinity and anti-affinity constraints, a semantic validation framework and embedding algorithms to map affinity-constrained Service Function Chains (SFCs) to the physical substrate network. In this appendix, more details are provided on the semantic validation framework which is able to detect conflicts that may arise between the various affinity and anti-affinity constraints of an SFC. To achieve this, the SFC request and relevant information on the physical topology are modeled as an ontology of which the consistency can be checked using a semantic reasoner. The ontology and a set of inference rules are presented in detail in this appendix.*

## C.1 Introduction

In the traditional telecommunications networking approach, functionality of a network node is strongly tied with the physical network device it runs on. Typically, the network operator needs to deploy a dedicated network appliance for each Network Function (NF) (e.g., Deep Packet Inspection (DPI), Firewall). In addition, NFs have a strict chaining that must be adhered to when deploying a specific service. Thus, service deployments are tightly coupled to the underlaying network topology. These reasons, together with the ever increasing requirements for high quality and stability have led to long product cycles, limited service agility and considerable dependence on specialized hardware. To be able to compete with Over-The-Top (OTT) service providers, which typically have much shorter product development cycles, and to limit the Capital Expenditures (CAPEX) and Operational Expenditures (OPEX) involved with physical network expansions, the network operators need to devise novel and less expensive ways to meet the increased capacity requirements and at the same time reduce the time to market of new services.

The Network Function Virtualization (NFV)-paradigm [1, 2] has been introduced to alleviate the aforementioned issues by leveraging IT virtualization technology to decouple the network functionality from the physical infrastructure. This allows NFs to run on standard high volume servers, storage devices and switches. The advantages are manyfold. First there potentially is a significant reduction in total costs through more efficient maintenance which can be performed remotely. In addition, thanks to the increased flexibility offered by virtualization, resources can be shared and used more efficiently. Finally, NFV has the potential to allow network operators to deploy novel services cheaper and faster with higher service agility.

The concepts of NFV open up new business opportunities in the form of Virtual Network Function Infrastructure Providers (VNFInPs), acting as brokers between Infrastructure Providers (InPs) and Service Providers (SPs). These VNFInPs lease the infrastructure offered by different InPs and deploy, orchestrate and interconnect Virtual Network Functions (VNFs) to create Service Function Chains (SFCs) [3], that are run by SPs to offer value-added services to their customers. InPs can profit by maximizing resource utilization and optimizing energy usage by offering their virtualized infrastructure to remote parties. SPs benefit from the proposed model since it allows rapid deployment and testing in a real network environment, thus leveraging faster time to market of new services. The offered services benefit from the dynamic nature of the network, computing and storage resources offered by the Virtual Network (VN), which allows them to scale dynamically based on service requirements and user mobility.

Together with these new opportunities and stakeholders, a set of new interac-

tions arises as well. For example, the SPs need a way to express their SFC requests and requirements to the VNFInP. In traditional network embedding approaches, only node and link restrictions can be specified. However, many scenarios can be envisioned where a SP might want to attach more detailed constraints concerning the placement and routing between NFs as well as constraints on their affinity. For example, to increase efficiency, the SP might want to require the embedding of VNFs within the same datacenter or even on the same host. Other reasons for more detailed affinity and anti-affinity constraints could be resilience, economic, legislative and privacy issues. In this appendix, a set of affinity and anti-affinity constraints is proposed that increases the control of SPs on the embedding of their SFC requests.

With this newfound ability to add custom constraints, the possibility arises that conflicting constraints are introduced by SPs in their SFC requests. Therefore, the VNFInP needs to be provided with a means to check the validity of SFC requests and inform the SP on potential conflicts. Since SFC requests can contain many VNFs, virtual edges and constraints, detecting conflicts within these requests is not a straightforward task, neither for human operators, nor for computer systems. Since conflicts can arise between sets of constraints, pairwise detection will not suffice. Therefore, this appendix proposes to take advantage of semantic modelling to define an ontology and rule set, which can be enriched with individuals based on the specific SFC request. Using a semantic reasoner, the consistency of this entire ontology can be determined and subsequently the validity of the SFC request can be assessed.

The contributions of this appendix are threefold. First, the sets set of affinity and anti-affinity constraints are defined that can be attached to a SFC request by the SP. Second, an existing virtualization description language is extended to support these constraints. Finally, this appendix proposes and evaluates a semantic conflict detection mechanism that can be employed by the VNFInP to check the validity of SFC requests.

## C.2 Related Work

NFV has been proposed as a paradigm that allows more flexible service deployment by leveraging IT virtualization technology in combination with programmable networks [4, 5]. To attain the gains promised by NFV, the VNFs and interconnecting virtual links should be efficiently mapped onto the physical substrate. To achieve this, several placement algorithms have been proposed in the related fields of virtual network embedding [6] and virtual datacenter embedding [7], as well as for NFV [8]. A placement algorithm can be formulated as an optimization problem with a particular objective such as load balancing, resource utilization, acceptance ratio, etc. *Basta et al.* propose a model for placing virtualized Evolved

Packet Core (EPC) functions in a way that minimizes the network overhead introduced by Software Defined Networking (SDN) control plane interactions [9]. *Mehraghdam et al.* apply Mixed Integer Quadratically Constrained Programming (MIQCP) to solve the placement problem and conclude that to obtain efficient use of resources, the placment of functions should be different according to the desired objective [10]. *Moens et al.* propose an Integer Linear Programming (ILP)-based solution in which hybrid scenarios are considered where part of the functions are provided by dedicated physical hardware and part of them by virtualized instances [11]. Others propose a heuristic approach to deal with the intractability of the aforementioned optimization approaches. *Xia et al.* propose a greedy heuristic which sorts VNFs according to the resource demands and embeds the resource-demanding VNFs with highest priority [12]. *Yoshida et al.* propose a multi-objective resource scheduling algorithm which optimizes simultaneously possibly conflicting objectives with multifaceted costraints [13]. None of the aforementioned approaches offers support for attaching affinity or anti-affinity constraints to the SFCs nor do they take into account such constraints when evaluating the embeddings.

Affinity and anti-affinity restrictions have previously been studied in the context of grid and cloud computing. Many argued that the lack of influence on the placement of workflow or service components is a hindrance for the adoption of the technology [14, 15]. Even though performance and economical benefits of cloud computing are clear, potential users hesitate to use the technology because legal, privacy, efficiency and resilience aspects are completely out of their control. Also recent media coverage shows an increased concern by end-users about their data privacy, raising the need for SPs to take into account privacy and legal issues when offering their services. These concerns also arise for NFV when deploying VNFs at certain locations and transferring data between them over virtual paths. Therefore, it is argued that also in NFV, mechanisms should be designed to allow SPs to add constraints concerning locality and affinity, both to VNFs as well as the interconnecting paths.

The solutions proposed in affinity and anti-affinity context in cloud computing mostly relate to two aspects: developing models to describe affinity rules and developing service placement algorithms that can work under the constraints of these rules. *Konstanteli et al.* present a set of affinity rules for cloud computing applications which are added to a Mixed-Integer Non-Linear Programming (MINLP) [16]. The authors define constraints that require allocating components/services in the same subnet or physical node or prevent services to be federated. *Espling et al.* propose a model for defining Virtual Machine (VM) placement in cloud computing supporting a set of affinity and anti-affinity constraints [17, 18]. This approach is extended by defining affinity and anti-affinity restrictions for SFCs. To this end support is added for specification of constraints

on the path between network functions and furthermore a more expressive syntax is defined that allows constraints to apply to specific VNFs, VNF types, locations and location types. Furthermore, a semantic framework is proposed which allows to check the validity of these constraints.

One of the benefits of NFV is that it supports automated orchestration of services. To achieve this, a number of descriptions are needed for everything that was configured manually in the past, including VNFs and network requirements. Also Service Level Agreement (SLA)-related parameters such as affinity and anti-affinity rules should be transformed into machine-readable description formats [19]. Huawei mentions the generation of affinity and anti-affinity policies as a mechanism for fault prevention [20]. In the definition of *Service Quality Metrics* by ETSI, special attention is brought to the enforcement of NFV customer anti-affinity rules which can improve the availability mechanisms [21]. The automatically generated affinity rules for VNFs in combination with user-specific affinity requirements could lead to conflicting constraints. In this appendix, a machine-readable format for affinity and anti-affinity constraints is proposed. Furthermore, an automated way to detect conflicting constraints is established based on ontologies. The proposed conflict detection is applicable for both user-generated as well as automatically generated affinity constraint sets.

## C.3 Affinity and Anti-Affinity Constraint Model

In an NFV context, SPs have no control over the mapping of VNFs to physical hosts or SFC edges to physical paths. Nevertheless, many situations can be envisioned where an SP might want to attach constraints to the placement of certain functions or on the routing of traffic, such as:

- **Efficiency:** VNFs that exchange a lot of data may want to be positioned close to one another (e.g., within the same datacenter, or even on the same physical host).
- **Resilience:** The SP might want to spread instances of the same VNF across multiple datacenters in order to improve resilience in case a failure occurs in one of the datacenters.
- **Legislation:** The SP might want to avoid hosting VNFs in certain countries due to legislative restrictions.
- **Privacy:** SPs or their customers might not want the traffic to pass through certain domains due to privacy concerns.
- **Economic:** SPs might have economic reasons (e.g., peering agreements) to place their functions in or route their traffic through certain domains.

However, currently there is no way to specify or model such requirements in an SFC template. In this section, a set of affinity and anti-affinity constraints for VNFs and their interconnecting paths are proposed. The affinity constraints apply to a set of physical locations $P$, a set of VNF instances $V$ and a set of edges interconnecting them $E$. There are different location granularities $g \in G$ that can be considered (e.g., network domains, datacenters, hosts), leading to a hierarchical structure of locations. Two hosts in a single datacenter represent different locations at the granularity of hosts, but have the same location at the datacenter level. $P^g \subset P$ is the set of locations at a certain granularity $g$. Furthermore, each VNF instance has an associated VNF type $t \in T$ (e.g., firewall, DPI), forming subsets $V^t \subseteq V$ of VNFs with type $t$. Finally, each virtual edge $e = (a, b) \in E$ connects two VNFs $a \in V$ and $b \in V$ and maps to a single or path of physical network links. The following constraints are proposed:

- Affinity($p \in P^g, v \in V$ or $t \in T$): A specific instance $v$ or all instances $v \in V^t$ of type $t \in T$ must be located at a specific location $p$ with granularity $g$.
- Anti-Affinity($p \in P^g, v \in V$ or $t \in T$): A specific instance $v$ or all instances $v \in V^t$ of type $t \in T$ may not be located at a specific location $p$ with granularity $g$.
- Affinity($p \in P^g$ or $g \in G, v \in V$ or $s \in T, w \in V$ or $t \in T$): A specific instance $v$ or all instances $v \in V^s$ must be placed together with a specific instance $w$ or all instances $w \in V^t$ at a specific location $p \in P^g$ or at the same location at a specific granularity $g \in G$.
- Anti-Affinity($p \in P^g$ or $g \in G, v \in V$ or $s \in T, w \in V$ or $t \in T$): A specific instance $v$ or all instances $v \in V^s$ may not be placed together with a specific instance $w$ or all instances $w \in V^t$ at a specific location $p \in P^g$ or at the same location at a specific granularity $g \in G$.
- Affinity($p \in P^g, e \in E$): A virtual edge $e \in E$ must be fully embedded at a specific location $p \in P^g$ with a granularity $g \in G$.
- Anti-Affinity($p \in P^g, e \in E$): The physical links comprising the virtual edge $e \in E$ may not pass through a specific location $p \in P^g$ with a granularity $g \in G$.
- Affinity($e \in E, f \in E$): Two virtual edges $e \in E$ and $f \in E$ must overlap (i.e. all physical links comprising the virtual edges must be part of both $e$ and $f$).
- Anti-Affinity($e \in E, f \in E$): Two virtual edges $e \in E$ and $f \in E$ may not overlap (i.e. none of the physical links comprising the virtual edges may be part of both $e$ and $f$).

```
<xs:element name="Affinity" type="Constraint">
   <xs:complexType>
      <xs:choice>
         <xs:sequence>
            <xs:choice>
               <xs:element name="locType" type="LocationType" />
               <xs:element name="loc" type="Location" />
            </xs:choice>
            <xs:choice>
               <xs:element name="funcTypeA" type="FunctionType" />
               <xs:element name="funcA" type="NetworkFunction" />
            </xs:choice>
            <xs:choice minOccurs="0">
               <xs:element name="funcTypeB" type="FunctionType" />
               <xs:element name="funcB" type="NetworkFunction" />
            </xs:choice>
         </xs:sequence>
         <xs:sequence>
            <xs:element name="connC" type="Connection" />
            <xs:choice>
               <xs:element name="connD" type="Connection" />
               <xs:choice>
                  <xs:element name="locType" type="LocationType" />
                  <xs:element name="loc" type="Location" />
               </xs:choice>
            </xs:choice>
         </xs:sequence>
      </xs:choice>
   </xs:complexType>
</xs:element>
```

*Figure C.1: OVF specification extension for modelling Affinity node and link constraints.*

A wide range of languages could be used to define the constraints outlined above, depending on the language used to define the SFC template. As an SFC is generally a directed acyclic graph structure, the specification language should be capable of modelling this. To model the constraints, an extension is used of the DTMF Open Virtualization Format (OVF) version 2.1.1[1]. The OVF descriptor is an XML-based language for annotating software to be run in virtual machines, such as product details, virtual hardware requirements and licensing. *Espling et al.* [18] expanded the OVF descriptor with additional constructs for defining constraints in structured cloud services. This appendix expands upon this work by additionally modelling the affinity and anti-affinity constraints for both node and edge mapping for SFCs. Figure C.1 shows how the affinity constraint subtypes could be defined.

To further clarify the presented constraint formulations and syntax, an example of an SFC request with both affinity and anti-affinity constraints will be presented next. Given a set of location types {*Autonomous System (AS), Datacenter (DC), Host*} and a set of network function types {*Firewall, DPI, Cache,*

[1]DMTF - OVF Specification - https://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.1.1.pdf

*StreamingServer*}. An example SFC is depicted in Figure C.2, where a streaming server is connected to two DPI functions (for tagging data packets), which in turn are connected to a firewall (for filtering) and a content cache. The DPI functions may either directly forward content to the cache or may send it to the firewall for filtering. Suppose a SP wants to offer a Video on Demand (VoD) service in Belgium where two major telecom providers are active: Telenet (*AS6848*) and Proximus (*AS6774*). Let us consider the following set of affinity and anti-affinity constraints:

- *Affinity(AS6848, c1)*
- *Affinity(AS6774, c2)*
- *Affinity(DC, e3)*
- *Affinity(DC, e6)*
- *AntiAffinity(e1,e2)*
- *AntiAffinity(DC, DPI, DPI)*

Specifically, the first two constraints state that the caches need to be located in the Telenet and Proximus AS respectively (e.g., because they should be close to the end user and limit uplink traffic through other networks). The third and fourth constraint define that the edges *e3* and *e6* between firewall and DPI functions should be completely embedded within a single DC, automatically forcing the DPI and connected firewall to be deployed within that DC. Finally, to improve fault tolerance, it is stated that edges *e1* and *e2* cannot have any links in common and that functions with type DPI should not be deployed within the same DC. Using the XSD schema defined above, the constraints can be represented as shown in Figure C.3.

## C.4 Semantic SFC Request Checker

Since SPs are now free to specify their custom constraints during the SFC request, it is possible that conflicting constraints are introduced. For example, extending the previous example and adding the constraints *Affinity(DC, c1, f1)* (i.e. specifying that *c1* and *f1* should be colocated in the same DC) and *AntiAffinity(DC, Cache, Firewall)* (i.e. specifying that a VNF of type Cache can not be colocated with a VNF of type Firewall in the scope of a DC) leads to a conflicting constraint set. Also more complex conflicts can appear when multiple constraints are involved in the conflicting set that can only be detected as a conflict when considering the full set. For example, returning to the base example from the previous section and adding the constraints *Affinity(AS, c1, f1)* and *Anti-Affinity(AS6848, d1)* would lead

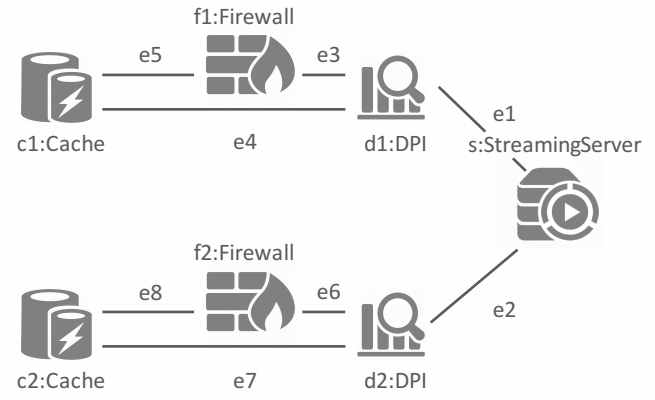



*Figure C.2: An example SFC.*

```
<Affinity>
    <loc>AS6848</loc>
    <funcA>c1</funcA>
</Affinity>
<Affinity>
    <loc>AS6774</loc>
    <funcA>c2</funcA>
</Affinity>
<Affinity>
    <locType>DataCenter</locType>
    <connC>e3</connC>
</Affinity>
<Affinity>
    <locType>DataCenter</locType>
    <connC>e6</connC>
</Affinity>
<AntiAffinity>
    <connC>e1</connC>
    <connD>e2</connD>
</AntiAffinity>
<AntiAffinity>
    <locType>DataCenter</locType>
    <funcTypeA>DPI</funcTypeA>
    <funcTypeB>DPI</funcTypeB>
</AntiAffinity>
```

*Figure C.3: A simplified constraint specification for the example SFC*

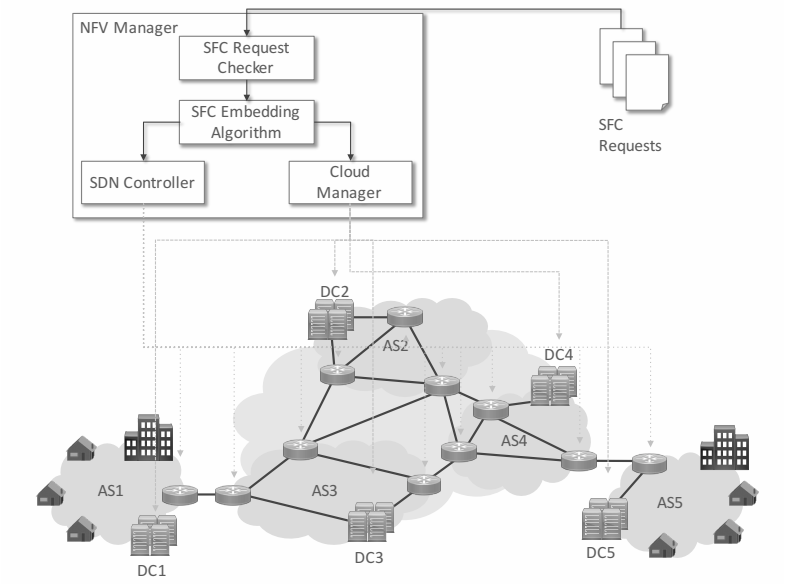



*Figure C.4: An overview of the NFV architecture with support for semantic SFC request checking.*

to a conflict set {*Affinity(DC, c1, f1), Anti-Affinity(AS6848, d1), Affinity(AS6848, c1), Affinity(DC, e3)*}. Since *d1* and *f1* should be colocated in the same DC due to the link affinity constraint and *f1* and *c1* are colocated at the DC level, *d1* and *c1* should be colocated at the AS level as well. Furthermore, since *c1* should be fully located in *AS6848*, *d1* should be located in the same AS. However, this inferred constraint conflicts with the defined constraint *Anti-Affinity(AS6848, d1)*.

When the VNFInP tries to deploy the requested SFC, none of the resulting embedding configurations will lead to a feasible realisation of the SFC request. The VNFInP should however be able to differentiate between a non-acceptance of the SFC request caused by a shortage of appropriate resources and conflicting request constraints in order to inform the SP on the reason why the SFC deployment failed. The previous example shows the need for the VNFInP to check the validity of an SFC request upon reception in order to exclude any conflicting constraints when trying to provision the requested SFC.

### C.4.1 NFV architecture for SFC request checking

Figure C.4 depicts how the SFC request checking system could be integrated into the *NFV Manager*. In this architecture, the *SFC Embedding Algorithm* is responsible for assigning physical hardware and resources to the SFC requests. Concretely,

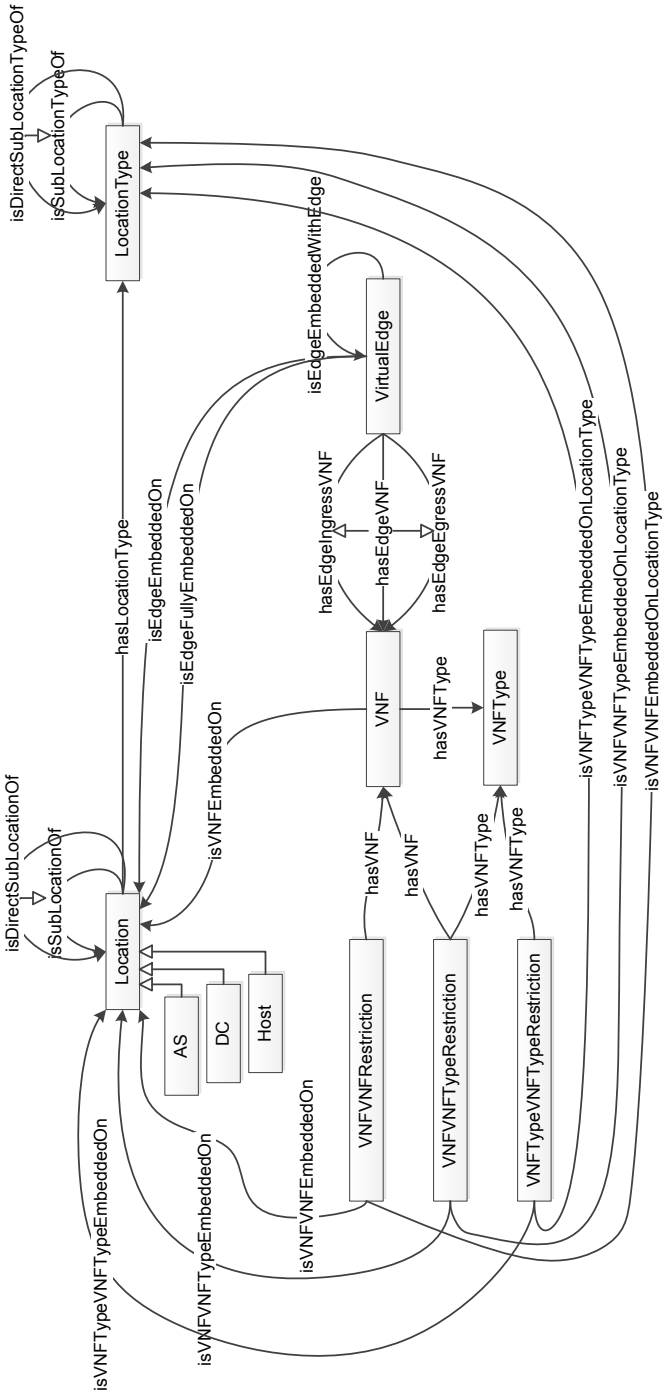



*Figure C.5: Graphical representation of ontology.*

it decides on which VNFs should be deployed on which physical hosts and how many resources should be assigned to them. The *Cloud Manager* performs the management of deployed VNFs and server resources. Moreover, the algorithm selects the forwarding paths interconnecting the VNFs and assigns network resources to them through the *SDN Controller*. Before the SFC request is forwarded to the *SFC Embedding Algorithm* it needs to be checked by the *SFC Request Checker* to confirm the validity.

### C.4.2 Ontology for SFC request modelling

This appendix proposes to exploit ontology representations for the purpose of modelling the physical substrate, the SFC request and defining a set of rules that can be used to infer additional information. Figure C.5 represents the proposed semantic model. The SFC request is modelled as a set of *VirtualNodes* with a certain *VNFType* and *VirtualLinks* containing an ingress and egress *VirtualNode*. The physical resources are modeled at the granularity level of *Hosts*, *DCs* and *ASs*. Each of these *Locations* has a certain *LocationType* (i.e., AS, DC or Host). The hierarchical relations between these *Locations* and *LocationTypes* are modeled by *isSubLocationOf* and *isSubLocationTypeOf* respectively. To model affinity (respectively anti-affinity) constraints for single virtual nodes and edges, positive (respectively negative) object property assertions of the type *isNodeEmbeddedOn*, *isLinkEmbeddedOn* and *isLinkFullyEmbeddedOn* are attached to *VirtualNodes* and *VirtualLinks*.

To be able to model more complex affinity and anti-affinity relationships between two *VirtualNodes*, two *VNFTypes* or between a *VirtualNode* and *VNFType*, the additional concepts *NodeNodeRestriction*, *NodeNodeTypeRestriction* and *NodeTypeNodeTypeRestriction* were added to the ontology. By adding the respective positive (respectively negative) property *isNodeNodeEmbeddedOn* or *isNodeNodeEmbeddedOnType*, one is able to model affinity (respectively anti-affinity) restrictions for more complex constraints on the *Location* or *LocationType*. By using the *isLinkEmbeddedWith* relationship, affinity and anti-affinity constraints between links can be modeled.

The Protégé editor[2] was used to develop the SFC request modelling ontology using the Web Ontology Language (OWL)[3].

### C.4.3 Rules

To be able to infer new information out of existing knowledge, a set of rules is defined. For example, a new relationship is defined in Rule (C.1) and (C.2), which is called *isSubLocOrEqualOf* that checks if a certain *Location a* is either equivalent

[2]Protégé - http://protege.stanford.edu/
[3]OWL2 - http://www.w3.org/TR/owl-features/

to *b* or a *isSubLocationOf b* is valid. This relationship will be used later on to infer knowledge on affinity and anti-affinity characteristics at a certain *LocationType*.

$$SameAs(a,b) \rightarrow SubLocOrEqualOf(a,b) \tag{C.1}$$
$$SubLocOf(a,b) \rightarrow SubLocOrEqualOf(a,b) \tag{C.2}$$

Rule (C.3) stipulates that if a certain *VirtualNode x* is embedded on a *Location y* and if *y* is a sublocation of *z*, this node is also embedded on *Location z*. When a *VNFType y* is embedded on a *Location z*, each *VirtualNode x* of that *VNFType y* needs to be embedded at the *Locationz* (Rule (C.4)). Rule (C.5) determines that if two *VirtualNodes x* and *y* are both embedded on a *Location a*, then the *NodeNodeRestriction z* containing *x* and *y* is embedded on the same *Location a*. Similar rules can be determined for *NodeNodeTypeRestriction* and *NodeTypeNodeTypeRestriction*, which are omitted due to space restrictions. Rule (C.6) stipulates the opposite: if a *NodeNodeRestriction z* containing *VirtualNodes x* and *y* is embedded on *Location a*, then both *VirtualNode x* and *y* are embedded on *Location a*. Similar rules can be determined for *NodeNodeTypeRestriction* and *NodeTypeNodeTypeRestriction*. Rule (C.7) determines that if *VirtualNodes x* and *y* are embedded on *Location a* and *Location b* respectively and if both *a* and *b* are either sublocations of or equal to *Location c* with *LocationType l* and *x* is not equal to *y*, then the *NodeNodeRestriction z* containing both *x* and *y* is embedded on *LocationType l*. Similar rules can be determined for *NodeNodeTypeRestriction* and *NodeTypeNodeTypeRestriction*. If a *VirtualLink z* contains a *VirtualNode x* embedded at *Location a*, this *VirtualLink z* is embedded at *Location a* as well (Rule (C.8)). Rule (C.9) states that if two *VirtualLinks x* and *y* share a *VirtualNode v* as *Ingressnode* and *Egressnode* respectively and *VirtualLink x* should overlap with *VirtualLink y*, this should lead to an inconsistency. Several other rules are added to the ontology, but are left out here due to space restrictions.

$$\begin{aligned} &isNodeEmbeddedOn(x,y) \wedge isSubLocOf(y,z) \\ &\rightarrow isNodeEmbeddedOn(x,z) \end{aligned} \tag{C.3}$$

$$\begin{aligned} &hasNodeType(x,y) \wedge isNodeTypeEmbeddedOn(y,z) \\ &\rightarrow isNodeEmbeddedOn(x,z) \end{aligned} \tag{C.4}$$

$$\begin{aligned} &hasNode(z,x) \wedge isNodeEmbeddedOn(x,a) \wedge hasNode(z,y) \\ &\wedge isNodeEmbeddedOn(y,a) \wedge Different(x,y) \\ &\rightarrow isNodeNodeEmbeddedOn(z,a) \end{aligned} \tag{C.5}$$

$$\begin{aligned} &isNodeNodeEmbeddedOn(z,a) \wedge hasNode(z,x) \wedge hasNode(z,y) \\ &\rightarrow isNodeEmbeddedOn(x,a) \wedge isNodeEmbeddedOn(y,a) \end{aligned} \tag{C.6}$$

$$hasNode(z,x) \wedge isNodeEmbeddedOn(x,a) \wedge hasNode(z,y) \wedge isNodeEmbeddedOn(y,b) \wedge isSubLocOrEqualOf(a,c) \wedge isSubLocOrEqualOf(b,c) \wedge hasLevel(c,l) \wedge Different(x,y) \rightarrow isNodeNodeEmbeddedOnType(z,l) \quad \text{(C.7)}$$

$$hasLinkNode(z,x) \wedge isNodeEmbeddedOn(x,a) \rightarrow isLinkEmbeddedOn(z,a) \quad \text{(C.8)}$$

$$hasLinkIngressNode(x,v) \wedge hasLinkEgressNode(y,v) \wedge isLinkEmbeddedWithLink(x,y) \rightarrow SameAs(x,y) \quad \text{(C.9)}$$

Semantic Web Rule Language (SWRL)[4] was used to express the aforementioned rules using concepts from the ontology defined in Section C.4.2. The Protégé editor was also used to define the rules using the Manchester syntax[5].

### C.4.4 Conflict detection

When a new SFC request arrives at the VNFInP, this request is parsed and the set of virtual nodes and links are added as individuals to the OWL ontology using the OWL API[6]. Next, the set of affinity and anti-affinity constraints are also added by either creating new individuals (i.e. *NodeNodeRestriction*), adding property assertions (i.e. *isNodeEmbeddedOn*) or both.

The HermiT OWL Reasoner[7] was used to check the consistency and the classification of the ontology. HermiT is a semantic reasoner for ontologies written in OWL. It is able to determine whether or not the ontology is consistent, identify subsumption relationships between classes, etc. The reasoner is based on a hypertableau calculus which provides efficient reasoning. The output of the reasoning process allows us to determine whether the SFC request at hand is valid or not. In the case of an invalid request, this is communicated to the requesting SP, otherwise the request is passed on to the embedding engine.

## C.5 Evaluation

A Java-based simulation framework was created in which the physical substrate and the SFC requests can be modeled. The components contained in the physical infrastructure are added as individuals to the ontology using the OWL API. Upon reception of an SFC request, the required individuals for the VNFs and their interconnecting links are added to a copy of this ontology. The affinity constraints

[4]SWRL - http://www.w3.org/Submission/SWRL/
[5]Manchester Syntax - http://www.w3.org/2007/OWL/wiki/ManchesterSyntax
[6]OWL API - http://owlapi.sourceforge.net
[7]HermiT OWL Reasoner - http://hermit-reasoner.com

*Table C.1: Overview of the evaluation parameters.*

| | | Network Size | | | # Affinity and Anti-Affinity Constraints | # VNFs |
|---|---|---|---|---|---|---|
| | | #AS | #DC | #Hosts | | |
| All | Individuals | 1 | 4 | 100 | 5 | 5 |
| | | 2 | 8 | 200 | 10 | 10 |
| | | 4 | 16 | 400 | 20 | 20 |
| | | 8 | 32 | 800 | 40 | 40 |
| | | 16 | 64 | 1600 | 80 | 80 |
| | | 32 | 128 | 3200 | 160 | 160 |
| Relevant | Individuals | 5 | 50 | 25000 | 5 | 5 |
| | | 10 | 100 | 50000 | 10 | 10 |
| | | 20 | 200 | 100000 | 20 | 20 |
| | | 40 | 400 | 200000 | 40 | 40 |
| | | 80 | 800 | 400000 | 80 | 80 |
| | | 160 | 1600 | 800000 | 160 | 160 |
| | | 320 | 3200 | 1600000 | 320 | 320 |

defined in the SFC request are parsed and the necessary individuals and rules are instantiated. Using the HermiT OWL Reasoner, the consistency of the resulting ontology is checked. The execution time and resulting consistency are logged. Two implementations of the validation component are evaluated. In the first version, all locations of the physical topology are instantiated in the ontology. In the second version, rather than including all physical locations into the ontology, only the physical locations that occur in the SFC request are added to the ontology as individuals, as well as the hierarchy of their parent locations. This is done so to reduce the size of the ontology as it is known that semantic approaches suffer from scalability issues when ontology sizes increase. The evaluations were performed using the Flemish Supercomputer Center (VSC) which contains nodes with 2x8-core Intel E5-2670 (Sandy Bridge @ 2.6 GHz) processors and 32 GB RAM.

Two implementations of the validation component are evaluated. In the first version, all locations of the physical topology are instantiated in the ontology. In the second version, for scalability reasons, only individuals are added for locations and their parent locations that are involved in a location-based affinity constraint. This reduces the number of individuals in the ontology and improves the performance of the SFC validation significantly, as will be demonstrated in the evaluation section.

To evaluate the performance of the proposed semantic validation framework, a number of random network topologies and SFC requests are generated. The topologies that are used during the simulations are generated randomly. First a number of ASs are generated to which a random number of hosts are added. Next, inter-AS and intra-AS links are added with a certain degree (between 2 and 10). Afterwards, a random number of DCs are added to each AS. Each of the hosts has a specific physical location that can be seen as a hierarchical combination of AS, DC and host identification. The total number of hosts, DCs and ASs are listed in Table C.1 for both the experiments where all individuals are included and where only relevant individuals are considered. The SFCs and their respective constraints

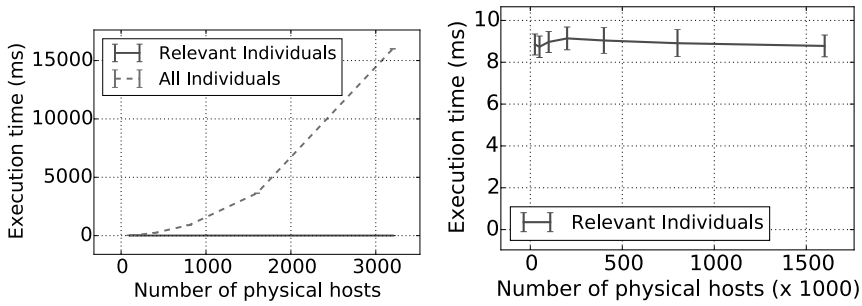



*Figure C.6: Impact of physical network size.*

are generated randomly: first a set of virtual nodes and links is generated, second a random affinity/anti-affinity constraints is added and the required virtual nodes, virtual links and VNF types are randomly selected from the ones that are present in the SFC. Finally, the locations are randomly selected from all available physical locations and added to the constraints. The type of the constraints is uniformly distributed among the constraints defined in Section C.3. The parameter values for both the number of VNFs and the number of constraints per SFC request are shown in Table C.1. For each parameter configuration, $1000$ random SFC requests are generated and fed to the semantic request checker. The execution times are averaged and the $95\%$ confidence intervals are shown in the graphs.

### C.5.1 Impact of physical network size

To evaluate the impact of the physical network size, a fixed number of $5$ virtual nodes and $5$ (affinity or anti-affinity) constraints are used in the requested SFC, and the number of physical hosts is varied as shown in Table C.1. For the case where an individual is included in the ontology for each physical location in the topology, it can be seen from Figure C.6(a) that the execution time grows exponentially with increasing physical network size as was expected. Even for a network of only $3200$ physical hosts, the semantic request validation already takes about $15s$. When an individual is only included if a physical location is used in a constraint, there is no significant impact on the evaluation time, since the number of constraints is fixed in this case as demonstrated in Figure C.6(b). These results demonstrate that it is indeed beneficial to only consider the relevant physical locations when semantically validating an SFC request.

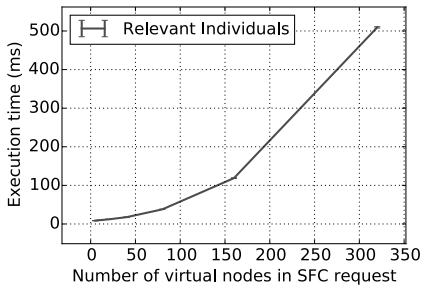



*Figure C.7: Impact of virtual network size.*

*Table C.2: Impact of number of constraints on number of consistent SFC requests.*

| #Constraints | #Consistent |
|---|---|
| 5 | 910 |
| 10 | 714 |
| 20 | 296 |
| 40 | 14 |
| 80 | 0 |
| 160 | 0 |
| 320 | 0 |

### C.5.2 Impact of virtual network size

To evaluate the impact of the size of the requested virtual topology in the SFC, a fixed number of $5$ constraints and a network size of $25000$ hosts is used, while the number of virtual nodes in the SFC request is varied as shown in Table C.1. Figure C.7 shows an exponential increase of the execution time when the number of requested VNFs in the SFC request increases. For $320$ VNFs in a single request, which is quite high, the execution time is $0.5s$, which could be considered as an acceptable calculation delay for SFC requests of that size.

### C.5.3 Impact of number of constraints

To evaluate the impact of the number of constraints in the SFC, a fixed virtual network size of $5$ VNFs and a physical network size of $25000$ hosts is used, and the number of constraints in the SFC request is varied as shown in Table C.1. Figure C.8 shows how the total execution time is affected by increasing the number of constraints in the request. Two additional lines are plotted differentiating between the execution times for consistent and inconsistent SFCs. As can be seen from Table C.2, when the number of constraints increases, the number of consistent SFCs is reduced significantly since the probability of conflicting constraints is increased. When the number of constraints exceeds $40$, all generated SFC requests are inconsistent. Figure C.8 shows a linear increase of the execution time when the number of constraints increases.

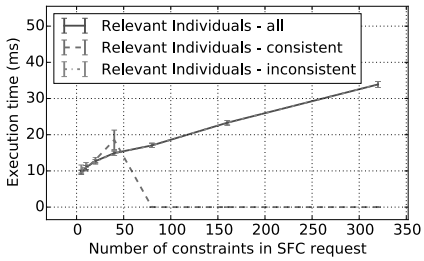
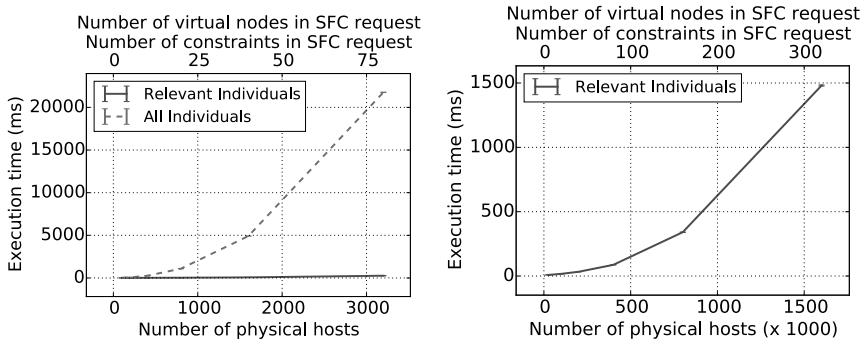



*Figure C.8: Impact of number of constraints on number of consistent SFC execution time.*




(a) (b)

*Figure C.9: Combined impact of increasing physical network size, requested virtual network size and number of constraints.*

### C.5.4 Combined impact of relevant parameters

Finally the combined impact of all parameters was evaluated as stated in Table C.1, both for the case when all physical locations are considered and for the case when only relevant physical locations are taken into account. For both approaches, an exponential increase can be observed in Figure C.9 when the physical network size, requested virtual network size and the number of constraints increase. When considering all physical locations as individuals, Figure C.9(a) shows that for a physical network of 3200 hosts, a virtual network of 160 VNFs and 160 constraints, the execution time is as high as $21.76s$. When reducing the ontology size by only including relevant locations, the execution time can be reduced to $0.34s$. Figure C.9(b) shows the results for larger network topologies and when only relevant locations are added to the ontology. Also here, an exponential increase in execution times can be observed. However, when considering the size of the SFC request in both number of virtual nodes (320) and affinity constraints (320), a total execution time of $1.5s$ can be considered acceptable. Bearing in mind that in-

creasing the size of the physical topology has a very limited impact on the total execution time since only the relevant individuals are added.

## C.6 Conclusion

In this appendix, a means for Service Providers (SPs) to attach location constraints to the mapping of Service Function Chains (SFCs) onto the physical substrate, is proposed. The SP's main interests to do this are for efficiency, resilience, legislative, privacy and economic reasons. This appendix proposes and defines a set of affinity and anti-affinity constraints which allow the SP to define whether certain Virtual Network Functions (VNFs) or instances of certain VNF types are allowed to reside in the same host or the same part of the network. Furthermore, specific locations or sets of locations can be defined where certain VNFs or VNF types may or may not be placed. To allow the Virtual Network Function Infrastructure Provider (VNFInP) to check the validity of these requests, a semantic request checking framework is proposed. This allows the VNFInP to model both its physical substrate and the SFC request as an ontology of which the consistency is checked using a semantic reasoner. This appendix proposes an optimization where only the relevant physical locations are modeled as individuals of the ontology. This ensures that the execution times are not subject to increasing topology sizes and that for reasonable SFC request and constraint sizes, the validations of the SFC request can take place in less than a second.

# References

[1] ETSI. *Network Functions Virtualization: An Introduction, Benefits, Enablers, Challenges and Call for Action*. ETSI Document, October 2012. Available from: http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[2] ETSI. *Network Functions Virtualization: Network Operator Perspectives on Industry Progress*. ETSI Document, October 2013. Available from: http://portal.etsi.org/NFV/NFV_White_Paper2.pdf.

[3] S. Boucadair, D. Lopez, I. Telefonica, D. Guichard, and C. Pignataro. *Service Function Chaining: Framework & Architecture draft-boucadair-sfc-framework-00*. 2014.

[4] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Denf, et al. *Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action*. In SDN and OpenFlow World Congress, pages 22–24, 2012.

[5] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. *Network function virtualization: Challenges and opportunities for innovations*. Communications Magazine, IEEE, 53(2):90–97, 2015.

[6] A. Fischer, J. Botero, M. Till Beck, H. de Meer, and X. Hesselbach. *Virtual Network Embedding: A Survey*. Communications Surveys Tutorials, IEEE, 15(4):1888–1906, Fourth 2013.

[7] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang, and M. Zhani. *Data Center Network Virtualization: A Survey*. Communications Surveys Tutorials, IEEE, 15(2):909–928, Second 2013.

[8] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. *Network Function Virtualization: State-of-the-art and Research Challenges*. Communications Surveys Tutorials, IEEE, PP(99):1–1, 2015.

[9] A. Basta, W. Kellerer, M. Hoffmann, H. J. Morper, and K. Hoffmann. *Applying NFV and SDN to LTE Mobile Core Gateways, the Functions Placement Problem*. In Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges, AllThingsCellular '14, pages 33–38. ACM, 2014.

[10] S. Mehraghdam, M. Keller, and H. Karl. *Specifying and Placing Chains of Virtual Network Functions*. CoRR, abs/1406.1058, 2014.

[11] H. Moens and F. De Turck. *VNF-P: A model for efficient placement of virtualized network functions*. In Network and Service Management (CNSM), 2014 10th International Conference on, pages 418–423, Nov 2014.

[12] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs. *Network Function Placement for NFV Chaining in Packet/Optical Datacenters*. Lightwave Technology, Journal of, 33(8):1565–1570, April 2015.

[13] M. Yoshida, W. Shen, T. Kawabata, K. Minato, and W. Imajuku. *MORSA: A multi-objective resource scheduling algorithm for NFV infrastructure*. In Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific, pages 1–6, Sept 2014.

[14] S. Benkner and C. GEMSS. *Report on COTS Security Technologies and Authorisation Services*. Project report, February 2004. Available from: http://eprints.cs.univie.ac.at/3311/.

[15] P. Massonet, S. Naqvi, C. Ponsard, J. Latanicki, B. Rochwerger, and M. Villari. *A Monitoring and Audit Logging Architecture for Data Location Compliance in Federated Cloud Infrastructures*. In Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, pages 1510–1517, May 2011.

[16] K. Konstanteli, T. Cucinotta, K. Psychas, and T. Varvarigou. *Admission Control for Elastic Cloud Services*. In Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, pages 41–48, June 2012.

[17] L. Larsson, D. Henriksson, and E. Elmroth. *Scheduling and monitoring of internally structured services in Cloud federations*. In Computers and Communications (ISCC), 2011 IEEE Symposium on, pages 173–178, June 2011.

[18] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth. *Modeling and Placement of Cloud Services with Internal Structure*. Cloud Computing, IEEE Transactions on, PP(99):1–1, 2014.

[19] H. Basilier, M. Darula, and J. Wilke. *Virtualizing network services–the telecom cloud*. Ericsson Review, 2014. Available from: http://www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2014/er-telecom-cloud.pdf.

[20] H. Technologies. *White Paper - Huawei Observation to NFV*. 2014. Available from: www.huawei.com/ilink/en/download/HW_399662.

[21] E. I. S. G. I. NFV. *Network Functions Virtualisation (NFV); Service Quality Metrics*. 2014. Available from: http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/010/01.01.01_60/gs_nfv-inf010v010101p.pdf.