Schaalbare dienst voor flexibele toegang tot persoonlijke content

Scalable Service for Flexible Access to Personal Content

Niels Sluijs

Promotoren: prof. dr. ir. B. Dhoedt, prof. dr. ir. F. De Turck
Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Computerwetenschappen

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. D. De Zutter
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2011 - 2012

UNIVERSITEIT
GENT

ISBN 978-90-8578-519-4
NUR 986, 988
Wettelijk depot: D/2012/10.500/45

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Informatietechnologie

Promotoren: prof. dr. ir. B. Dhoedt
prof. dr. ir. F. De Turck

Juryleden: prof. dr. ir. D. De Zutter, Universiteit Gent (voorzitter)
prof. dr. ir. P. Demeester, Universiteit Gent (secretaris)
dr. ir. T. Wauters, Universiteit Gent
prof. dr. P. Lambert, Universiteit Gent
dr. K. Spaey (Universiteit Antwerpen)
prof. dr. G. Doyen (Université de technologie Troyes)

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur

Vakgroep Informatietechnologie
Gaston Crommenlaan 8 bus 201, B-9050 Gent, België

Tel.: +32-9-331.49.00
Fax.: +32-9-331.48.99

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)

Proefschrift tot het behalen van de graad van Doctor in de Ingenieurswetenschappen: Computerwetenschappen
Academiejaar 2011-2012

Dankwoord

Precies zes jaar geleden zat ik in hetzelfde schuitje, bezig met het afronden van een studie en het regelen van een emigratie. Net als toen neem ik afscheid van een leuk en bekend leventje, alle plaatsen en vooral de mensen die je minder vaak zal gaan zien. Iedere stap zet ik dan ook met dubbele gevoelens. Veel gewoonten en zeker mensen ga ik missen, maar aan de andere kant komen er weer nieuwe uitdagingen aan en ik ga maar uit van Barney Stinson's gezegde: *"new is always better"*.

Het schrijven van mijn dankwoord is de sectie waar ik de afgelopen jaren het meest naar uit heb gekeken, je schrijft het per slot van rekening wanneer alles zo goed als af is. Vele keren heb ik dan ook nagedacht over de invulling hiervan. Nu ik bezig ben realiseer ik mij dat dit het moeilijkste hoofdstuk is om neer te schrijven, je wilt namelijk niemand te kort doen. Zonder spoilers te geven, is een algemene conclusie van dit proefschrift dat betere functionaliteit gehaald kan worden als effectief samengewerkt wordt. Uiteraard geldt dit niet alleen voor computersystem en ben ik aan veel mensen dank schuldig. In dit belangrijke hoofdstuk wil ik dan ook iedereen bedanken die de afgelopen jaren mijn doctoraat hebben gesteund. Hoewel fouten je menselijk maken, zal het toch jammer zijn als ik mensen vergeet te bedanken. Hierbij alvast mijn oprechte excuses als ik je ben vergeten!

Eerlijk is eerlijk, zonder de financiele steun van de Universiteit Gent, het IBBT en het IWT was niets van dit boek tot stand gekomen. Vandaar mijn dank voor de geboden mogelijkheden van deze instanties. Ik heb mijn onderzoek ook niet kunnen doen zonder de vakgroepvoorzitter prof. Daniël De Zutter en prof. Piet Demeester, die op een inspirerende manier leiding geeft aan een grote groep onderzoekers. Martine Buysse, Davinia Stevens, Sandra, Sabrina, Ilse Van Royen, Karien Hemelsoen, Bernadette Becue, Dalila Lauwers, Nathalie Vanhijfte, Joke Staelens, Bert De Vuyst, Pascal Vandeputte, Wouter Adam, Jonathan Moreel, Bert De Knijf, Joeri Casteels, bedankt voor jullie dagelijkse ondersteunende taken. Met technische of administratieve vragen kon ik altijd bij jullie terecht en werd ik vriendelijk geholpen.

Bij onderzoek zijn promotoren erg belangrijk, daarom veel dank aan prof. Bart Dhoedt en prof. Filip De Turck. Ik heb veel van jullie geleerd en ben op de dag van vandaag nog steeds onder de indruk van jullie kennis, inzichten en vaardigheden. Hier wil ik ook dr. Tim Wauters bedanken, bij al het onderzoek ben jij een grote begeleidende factor geweest. Je adviezen en ondersteuning zijn een grote leidraad geweest in het onderzoek. Daarnaast ben je altijd een goede reviewer geweest en sta ik te kijken van je geduld dat je nooit geklaagd hebt over de vele grammatica-fouten in de teksten.

Bureau 2.21 is bij IBCN altijd mijn thuishaven geweest en ik heb dit bureau met veel mensen en plezier gedeeld. Dr. Bart De Vleeschauwer, dr. Bruno Van Den Bossche, dr. Sofie Van Hoecke, dr. Marc De Leenheer, dr. Tim Stevens, dr. Wouter Haerick, Kristof Willemyns, Jeroen De Wachter, An De Moor, Dieter Verslype,

Wim Van de Meerssche, Olivier Van Laere, Tim Verbelen, Klaas Roobroeck, dr. Steven Latré, dr. Pieter Simoens, dr. Stijn Verstichel, Philip Leroux, Kristof Steurbaut, Bram Gadeyne, Niels Bouten en Steven Van Canneyt. Bedankt voor de leuke gesprekken, het delen van grappige informatie over (thesis)studenten en collega's, de leuke poker- en bowlingavondjes, het gamen en de altijd gezellige drinks. Na de grote bureau herschikking is het wel even wennen geweest, maar de gezelligheid binnen het bureau is wel weer teruggekomen!

Vier jaar lang ben ik wekelijks op 'bijscholing' geweest waar ik altijd met veel plezier naar toe ben gegaan (wat resulteerde in een klein beetje gewichtstoename). Samen met Wim, Johan en Farida heb ik altijd lekker gekookt, erg gelachen als kookgerei vlam vatte of volgens de anderen er iets te veel cayennepeper in een gerecht zat. Na de voltooing van de kookles ben ik samen met Klaas en Yuri begonnen met een ware bierstudie. Geweldig dat dit bestaat en toch verassend veel geleerd. Natuurlijk was er ook veel lol tijdens en na het bestuderen van al het bier.

Naast de vrienden van mijn werk ben ik ook veel opgetrokken met de diergeneeskunde-bende, hoewel een totaal andere studie toch allemaal lotgenoten die naar België zijn verhuisd om kennis op te doen. Erg leuk waren alle feestjes, weekendjes, Gentse Feesten en de wintersport. Een zeer gezellig groep om mee op te trekken, bestaande uit onder andere Malou, Nathalie, Maarten, Marloes, Tessa, Henry, Marieke, Merel, Wilco, Channa, Tamar, Marte, Stefanie, Dewi, Maya, Irene, Frederike en Yuri.

Ergens waar ik vaak en graag over de vloer kom is bij de familie van Stefanie: Laurens, Jennie, Oma Mous. De verjaar- en feestdagen zijn erg gezellig en ik doe altijd graag mee met jullie passie voor eten. Rustig wakker worden met een paar lekkere Branbantse worstenbroodjes is iets waar ik nu alweer naar uitkijk. Natuurlijk wil ik ook Mark en Evelien bedanken, naast familie van Stefanie behoren jullie tot erg goede vrienden.

Veel plezier is er natuurlijk ook met mijn familie. Ik weet dat jullie het niet erg leuk vinden dat ik 'zooo' ver weg zit en daardoor misschien wat minder vaak zie. Maar gelukkig zag ik toch iedereen regelmatig bij de verjaardagen en natuurlijk steevast met oud en nieuw. De verhitte discussies zijn altijd leuk om te volgen en door de toch wel hechte band die we hebben, heb ik altijd op iedereen kunnen steunen. Helaas was het einde vorig jaar een grote domper, maar ons leven gaat verder en er staan dit jaar alvast een aantal mooie evenementen op het programma waar ik al erg naar uitkijk. Bibian, Bodine, Marco, Ilona, Harm, Daphne, Opa Sluijs, Oma Sluijs, Elmar, Rosanne, Thijs, Leonie, Jan en Marleen, iedereen erg bedankt voor al jullie steun en interesse!

Naast familie waar ik het erg goed mee kan vinden heb ik de afgelopen jaren ook veel plezier gehad met Dieter, Yuri, Olivier, Tim, Klaas en Ernst. De vele weekendjes, concerten, paintball, sport, gamen, movie-nights en ga zo maar door, staan allemaal in mijn geheugen gegrift waar ik met veel plezier aan terug denk en daardoor nu meteen veel zin krijg om weer samen iets te ondernemen.

In het rijtje van mensen die ik bedank mag en kan natuurlijk één iemand niet ontbreken, juist: Stefanie! De reden waarom het avontuur Gent in mijn leven zit en uiteraard mijn dagelijkse steun en toeverlaat. Iedere keer heb jij moeten aanhoren als iets weer eens niet lukte of juist wanneer ik dacht de uitvinding van de eeuw te hebben gedaan (iets dat bij wel meer Sluijsjes voorkomt). Lief vind ik het ook altijd als ik je aan anderen hoor uitleggen waar mijn onderzoek 'precies' overgaat. Waar ik van nature wat meer terughoudend ben, ben jij degene die mij perfect aanvult en

zo nu en dan over de streep sleurt. Hierdoor maak ik spannende avonturen en jij 'normale' dingen mee. Ik ben ontzettend blij dat jij in mijn leven bent en voel me een zeer gelukkig man die bevoorrecht is om samen met jou de toekomst aan te gaan.

Gent, Juni 2012

Table of Contents

List of Figures

List of Tables

List of Acronyms

C

CDN	Content Distribution Network

D

DHT	Distributed Hash Table
DSL	Digital Subscriber Line

E

EDG	European DataGrid

I

ILP	Integer Linear Programming
IP	Internet Protocol
ISP	Internet Service Provider

L

LFU	Least Frequently Used
LRU	Least Recently Used

M

MCL	Markov Chain Length
MDL	Most Distant Lookup

P

P2P	Peer-to-Peer
PCS	Personal Content Storage
PCSS	Personal Content Storage Service

Q

QoS	Quality of Service

R

RTD	Request Times Distance
RTDc	cooperative Request Times Distance

S

SA	Simulated Annealing
SON	Service Overlay Networks
SVC	Scalable Video Coding

T

TTL	Time-to-Live

V

VoD	Video-on-Demand
VRP	Vehicle Routing Problem

X

XML	Extensible Markup Language

Nederlandstalige samenvatting (Dutch Summary)

Een belangrijke trend van dit moment is het online delen van persoonlijke content, zoals digitale foto's en digitale films. Omdat eindgebruikers toestellen van verschillende aard bezitten die allen met elkaar verbonden zijn via een thuisnetwork of het Internet, verwachten zij dat ze hun persoonlijk archief op elke plek en op elk tijdstip kunnen raadplegen. Echter, de achterliggende technologieën die op de dag van vandaag dit soort toepassingen ondersteunen zijn nog altijd gebaseerd op de klassieke client-server modellen. Doordat de collecties van persoonlijke bestanden snel groeien zijn geavanceerdere technieken nodig om gebruikers de mogelijkheid te bieden om al hun persoonlijke content op een transparante en flexibele manier te beheren.

De 'Personal Content Storage Service' (PCSS) is een genetwerkte oplossing die gebruikers opslagruimte biedt op een kostefficiënte en schaalbare manier (i.e. zowel in het aantal gebruikers als in het aantal bestanden). Gebruikers ervaren een PCSS als een lokale (virtuele) harde schijf die op een consistente en flexibele manier toegang biedt tot hun persoonlijke content. Van de verschillende belangrijke functies die een PCSS vervult (zoals aanwezigheidsbeheer, veiligheidsvoorziening en het monitoren van de onderliggende hardware-infrastructuur), hebben we in dit proefschrift twee belangrijke concepten onderzocht: het indexeren en ophalen van content.

Doordat een PCSS een groot aantal eindgebruikers heeft, is een schaalbare architectuur vereist. Bij voorkeur dient de al bestaande netwerkinfrastructuur hierbij zo efficiënt mogelijk gebruikt te worden. Daarom zal een PCSS een (hybride) 'Peer-to-Peer' (P2P) model gebruiken voor het gedistribueerd indexeren en verzenden van de content in het netwerk. In een P2P netwerk wordt een virtuele topologie bovenop het bestaande Internet Protocol (IP) gevormd en elke 'peer' verzorgt zowel de rol van aanbieder als afnemer.

Een interessante techniek om content op een efficiënte manier te indexeren in een gedistribueerde omgeving is een 'Distributed Hash Table' (DHT). Een DHT biedt in een gestructureerd P2P netwerk zoekmogelijkheden aan vergelijkbaar met een hashtabel. Om de zoekprestaties te verbeteren van een DHT wordt er typisch een 'caching'-laag aangebracht tussen de applicatielaag en de DHT. Omdat persoonlijke bestanden locatie-afhankelijke aanvraagpatronen vertonen, zal onze voorgestelde 'caching'-oplossing zowel populariteits- als afstandsmetrieken gebruiken om het proces van het lokaliseren van de persoonlijke content te optimaliseren. We hebben een update-protocol ontworpen dat buren informeert van veranderingen in een 'cache'. Dit coöperatieve mechanisme is door middel van simulaties geanalyseerd en vergeleken met een 'state-of-the-art' pro-actief replicatieraamwerk. De resultaten

laten zien dat onze strategie beter presteert dan het bestaande replicatiemechanisme en dat de 'overhead' dat het update-protocol introduceert aanvaardbaar is, omdat prestatiewinst substantieel groter is dan de geïntroduceerde 'overhead' van het protocol.

Hoewel het verkleinen van opzoekvertragingen gunstig is, betekent dit niet dat de content zelf snel opgehaald kan worden. Daarom is de volgende uitdaging voor een PCSS het realiseren van een efficiënt transport van de persoonlijke bestanden naar de toestellen van eindgebruikers. We onderzoeken in dit proefschrift een raamwerk dat door middel van 'caching' met meerdere niveaus, veelvuldig aangevraagde persoonlijke content dichtbij gebruikers plaatst die deze vaak raadplegen. Het resultaat hiervan is dat gedeelten van het PCSS netwerk ontlast worden en dat ophaaltijden worden verkort. Door middel van een dimensioneringsstudie analyseren we de voorgestelde 'caching' met meerdere niveaus, door optimale groottes te berekenen van 'caches' in een boomnetwerktopologie, waarbij ieder 'cache'-niveau instaat voor het afhandelen van een gelijk aantal aanvragen. De uitkomsten van de simulaties liggen dichtbij de theoretische verwachtigen. Bovendien laten de simulaties zien dat de snelheid (i.e. het aantal individuele ophalingen van een persoonlijk bestand) waarmee de content wordt opgeslagen in het eerste niveau van het 'caching'-raamwerk overeenkomen met analytische berekeningen.

Omdat een groot gedeelte van de gegevens die momenteel over het Internet worden verzonden bestaan uit zogenoemde 'streaming-media', onderzoeken we geavanceerde mechanismen om efficiënt (live) videostromen te vervoeren naar eindgebruikers. In tegenstelling tot traditionele bestanden zijn videostromen nuttig gelijk vanaf het moment dat het eerste datasegment aankomt. Een goede kandidaat om een zowel kostenefficiënte als schaalbare oplossing aan te bieden, is weer het P2P netwerkmodel. Door de heterogene omstandigheden in bandbreedtecapaciteiten en uitvoermogelijkheden van de gebruikte toestellen, zal de volgende generatie P2P (live) videostreaming-diensten meerlaagse videocodering gebruiken. Het afspelen van een video kan worden begonnen zodra de basislaag wordt ontvangen en elke laag die extra aankomt verhoogt de kwaliteit van de video voor de eindgebruiker. Om het verzenden van video's te optimaliseren stellen we een dirigeercomponent voor, die kennis heeft van de netwerktopologie en het transport van elke videolaag beheert. Hiervoor hebben we een wiskundige formulering opgesteld die in staat is om de routering te bepalen van de verschillende videolagen, zowel op onder- als bovenlaagniveau. Huidige en traditionele P2P netwerken voor videostreaming hebben als doel om op een gretige manier de videokwaliteit te maximaliseren voor de gebruiker, hoewel een videoleverancier vooral geïnteresseerd is om het minimum aantal videolagen dat op elke bestemming toekomt te verhogen. Daarom gebruiken we het wiskundig model bij het analyseren en vergelijken van deze strategieën. We laten zien dat de strategie van de videoleverancier zorgt voor een aanzienlijke vermindering van het aantal eindbestemmingen die enkel de basislaag ontvangt. Als gevolg hiervan zullen meer eindgebruikers meer videolagen verkrijgen dan wanneer de traditionele gretige methode wordt gebruikt.

Het gebruik van exacte oplossingsmethoden voor het berekenen van zowel de onder- als bovenlaagroutering voor meerdere videolagen is alleen haalbaar voor relatief kleine netwerktopologieën. Vandaar dat we een heuristisch algoritme hebben ontworpen dat in staat is om het routeringsproces te bereken voor grotere netwerken. Daarnaast hebben we deze heuristische strategie uitgebreid om de ideale posities te

berekenen om bestaande knopen uit te breiden met ‘peering’-applicatie functionaliteit. De meetresultaten tonen dat, zoals intuïtief verwacht, wanneer het aantal ‘peering’-knopen vergroot wordt, het gemiddeld aantal ontvangen videolagen groeit. Doordat bestemmingen gemiddeld meer videolagen ontvangen, neemt het gemiddelde bandbreedtegebruik per link toe. De simulatieresultaten laten echter zien dat de bandbreedtecapaciteiten efficiënter worden gebruikt wanneer het aantal ‘peers’ toeneemt.

Alle voorgestelde algoritmen en protocollen in dit proefschrift hebben gemeen dat ze coöperatieve technieken gebruiken om bestaande oplossingen uit te breiden en te verbeteren. Door middel van samenwerking zijn de toegepaste methoden in staat om een betere dienstverlening aan te bieden dan dat een enkele instantie dat kan.

English Summary

An important trend today is the online sharing of personal content, such as digital photos and digital movies. Since end-users have different types of devices that all are interconnected via home-networks or the Internet, users expect that they can access their personal content archive from anywhere and at any time. However, the back-ends of currently used frameworks supporting such applications are still based on classic client-server models. Due to the fast growing personal content collections, more advanced techniques are needed to offer users the ability to manage all their personal content in a transparent and flexible manner.

The Personal Content Storage Service (PCSS) is a networked solution that offers storage space as a service to end-users in a cost-efficient and scalable way (i.e. in the number of users and (their) content). Users experience a PCSS as a local (virtual) hard disk that allows them to access their content consistently and flexibly. Although a PCSS exhibits several major functions (such as presence management, security provisioning and monitoring the underlying hardware infrastructure), this dissertation focusses on the two key concepts: content indexing and retrieval.

Since large volumes of end-users use a PCSS, the architecture is required to be highly scalable and preferably use the already installed network infrastructures as efficiently as possible. Therefore, a PCSS uses (hybrid) Peer-to-Peer (P2P) overlay models for the distributed content indexing and transferring the content items in the network. In a P2P network a virtual topology is formed on top of the actual IP (Internet Protocol) network, and all peers are performing as both suppliers and consumers.

An interesting approach to index content items efficiently in a distributed environment is a Distributed Hash Table (DHT). A DHT is a structured P2P network that offers scalable lookup similar to a hash table. To increase the lookup performance of a DHT, a caching layer is typically installed between the application layer and the DHT. Since personalized files exhibit location dependent request patterns, our proposed caching solution uses popularity and distance metrics to optimize the process of locating personal content items. We have designed an update protocol to inform neighbors of cache updates. This cooperative mechanism virtually increases the size of a node's local cache, since the node can avoid storing the same copies that can be retrieved from a neighbor (in only one overlay hop). The proposed cooperative caching solution is analyzed by using simulations and compared with a state-of-the-art proactive replication framework. The results show that our strategy significantly outperforms the existing replication mechanism and that the message overhead introduced by the update protocol are acceptable since the performance gain is substantially higher than the introduced protocol overhead.

Although reducing delays to locate items is highly favorable, no guarantees are actually made that the content itself can be accessed quickly. Therefore, a key challenge for a PCSS is efficiently transporting personal files to the devices of end-

users. We investigate in this manuscript a multi-level caching framework to store frequently accessed personal content items closers to the users that are accessing them often. As a result significant parts of the PCSS network are relieved and access times are reduced. To analyze our proposed multi-level caching solution, we have performed a dimensioning study to find optimal values for the cache sizes in a tree network topology, so that each level serves an equal share of the requests. The simulation results show that the cache hit rates are close to theoretical expectations. Additionally, the simulation measurements confirm to the analytically calculated content distribution rate, which is the number of individual downloads necessary to fill as many caches as possible at the first level.

Since a significant fraction of the data transferred in the Internet is classified as streaming media today, we investigate advanced mechanisms that efficiently transport (live) video streams to end-users. In contrast to traditional files, streaming media are useful from the moment the first data segment arrives at a destination. To offer cost-efficient and scalable solutions, P2P overlay technologies seem the right candidate. Due to the heterogeneous circumstances in both bandwidth capacities and rendering possibilities of the end-devices, next generation P2P (live) video streaming are using multi-layer video. Playback can already be started when at least the base layer is received and every additional incoming video layer increases the user's experienced viewing quality of the video. To optimally transfer video streams to end-users we present an orchestrating engine that is topology-aware and manages the transport of each video layer in the network. We have developed a mathematical formulation that models both the underlay and overlay routing of the distinct video layers. Currently, traditional P2P video streaming networks have the objective to maximize greedily the local peer's video quality. However, video service providers are mainly interested in the minimum number of video layers that can be delivered to each end-user. We use the mathematical formulation to analyze and compare both strategies, and show that by using the video service objective a significant reduction is achieved of destinations only receiving the base layer. Therefore, the number of end-users receiving higher layers increases compared to the traditional method.

Using exact solvers to calculate the underlay-overlay-routing problem of multiple video layers is only feasible for relatively small network topologies. Therefore we present a heuristic optimization method that is able to compute the routing process for larger topologies. Additionally, we extend the heuristic strategy to calculate ideal positions to upgrade existing nodes with peering application functionality. Intuitively, our results show an increase in the average number of accommodated video layers when the number of peering nodes in the network increases. As a result of the increase of the average number of received video layers, the average bandwidth usage on a link also increases. However, the simulation results show that the bandwidth capacities are used more efficiently due to an increasing number of peering nodes.

As a common denominator, all presented algorithms and protocols in the dissertation use cooperative mechanisms to extend and increase the performance of existing solutions and thereby providing a better service than one single instance is able to do.

1

Introduction

This introductory chapter sets the scene for the research carried out in the framework of this dissertation. The research is done in the context of sharing personal content. We provide a brief overview of distributed techniques for locating and transferring personalized content to end-users. Our major research contributions are highlighted subsequently. Additionally, an outline is presented of the dissertation and a list of publications that led to this manuscript.

1.1 Research context: managing personal content items

Every day our lives are getting more digitalized. In our society it is common to have access to multiple computers, most of the cellphones are equipped with high-quality (video) cameras and tablets are starting to replace printed brochures and magazines in living rooms. All devices are interconnected with each other via in-home networks or the Internet and, thereby, create a whole new set of applications and user expectations. A common desire users have is to share/store personal files 'online', such as their digital movies [1]. Currently, web sites are migrating from static pages containing pre-rendered text and images to complete frameworks offering interactive (web) applications to end-users. However, the back-ends for these systems are mainly based on classic client-server models and files are still scattered over multiple devices and storage locations. Figure 1.1 illustrates the typical situation where personal content is stored on different devices and the user is faced with the complex task to manage his/her distributed personal content collection.

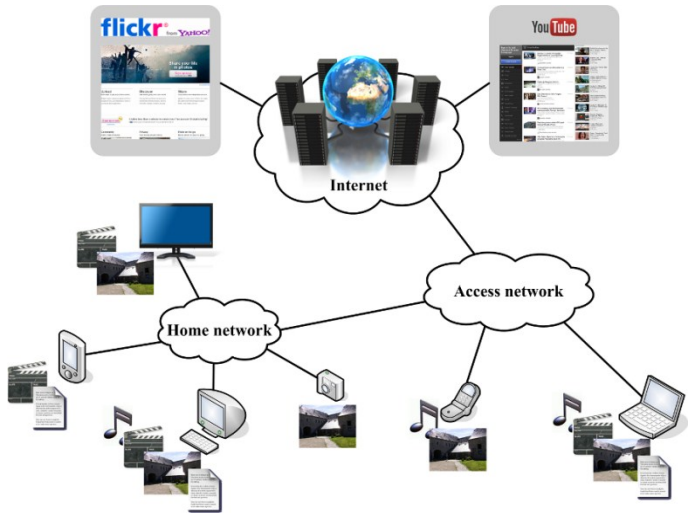

Figure 1.1: Today, personal files are stored on multiple locations across different types of devices.

In order to handle the explosive growth of content items and the user's expectation to locate, access and manage their content archive from different (types of) devices at any time and from anywhere, technologies have to be developed and optimized to offer flexible access to personal content in a cost-efficient and scalable manner (i.e. in the number of users and (their) content). The Personal Content Storage Service (PCSS) is a networked solution offering storage space as a service in a transparent manner to end-users. The main functions for a PCSS concern user

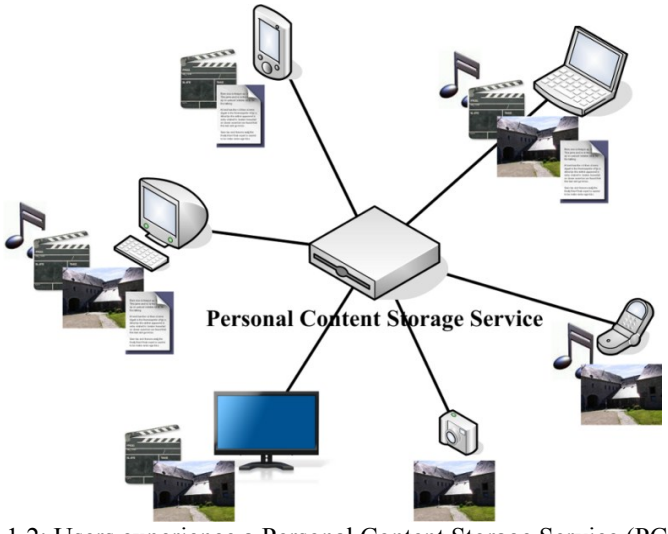

Figure 1.2: Users experience a Personal Content Storage Service (PCSS) as virtual hard disk, where all their content is accesssed in a transparent manner.

and content management (including replica management and indexing), query handling, presence management, security provisioning, monitoring of the underlying hardware infrastructure, and retrieving and uploading personal content items. Users experience a PCSS as a local (virtual) hard disk that allows them to search, access and share their content in a consistent and transparent manner (see Figure 1.2).

Different existing techniques can be incorporated into a PCSS such as cloud-services offering storage space or Content Distribution Networks (CDN) to proactively replicate files for load balancing purposes. However, these underlying frameworks have to be optimized for serving large volumes of relatively unpopular content items and deliver their services in a scalable way.

1.2 Problem statement and research objectives

Since a PCSS serves large volumes of end-users, the architecture is required to be highly scalable and preferably use the already installed network infrastructures as efficiently as possible. Therefore, a PCSS uses (hybrid) Peer-to-Peer (P2P) overlay models for key features involving distributed content indexing and efficiently transferring content items in the network. In a P2P network a virtual overlay topology is formed on top of the actual IP (Internet Protocol) network, and all peers are acting as both suppliers and consumers. In contrast, in traditional client-server networks only servers supply and clients consume, therefore, P2P services are potentially highly scalable and robust.

One of the main challenges for a PCSS is the ability to search worldwide through the data set of personal content items. Inherent to personalized files are the locality patterns in their request distributions [1]. Current solutions that provide distrusted storage for files [2-4], lack the power to efficiently handle scattered lookup request of content exhibiting location dependent request patterns. Query search in unstructured P2P networks is done via a query flooding model, where a Time-to-Live (TTL) mechanism is used to prevent overloading the network. As a result of the TTL limit no guarantees are made that personal content stored in the network can be found, making this type of search mechanisms less suitable for a PCSS. In contrast, a data structure that guarantees that (even) rare objects can be located is called a Distributed Hash Table (DHT). A DHT is a structured P2P network offering scalable lookup similar to a hash table, exhibiting per lookup request an average (overlay) hop count in the order of $O(\log N)$ with N the number of nodes in the DHT. To increase the lookup performance, a caching layer is typically used between the application layer and the DHT [5]. Usually, these caching strategies are location independent and do not exploit location dependent lookup patterns.

Although reducing lookup times to locate content items is highly favorable, no guarantees are actually made that the content itself can be accessed quickly. Therefore, another challenge for a PCSS is efficiently transporting personal files (including streaming media) to devices of end-users. Many distributed file systems exist [2-4], but none of them were designed for large-scale deployment in an access and aggregation network environment. Integrating a distributed cache in the network allows users to access (their) personal content with minimal delay.

Today, a significant fraction of the data transported in the Internet is classified as streaming media (i.e. video streams) and [6] expects that the amount of video streams on the Internet will grow in the next years. In contrast to traditional files,

Figure 1.3: Currently, video is mainly transported from source servers to the end-users client, without responding to heterogeneous circumstances (e.g. different rendering possibilities).

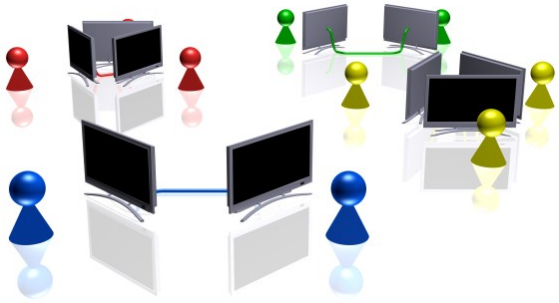

Figure 1.4: A more cost efficient, robust and scalable video streaming solution can be offered, when devices share downloaded video parts. Even when peers stream at a different number of video layers, received layers that they have in common can still be exchanged.

streaming media are useful from the moment the first part of the data stream arrives, which provides a whole new set of challenges to us. A logical transition to offer a scalable and cost efficient solution, is to switch away from traditional platforms that uses servers to provide videos (see Figure 1.3) to P2P networks where multiple peers support to each other the download of video parts (see Figure 1.4). Next generation P2P (live) video streaming networks are using multi-layered video in order to adapt to heterogeneous circumstances [7], e.g. various asymmetric link bandwidths or end-devices having different display resolutions. Another advantage is the ability for peers to exchange video layers, even if the other peer is streaming at a different video quality (i.e. expressed in the number of accumulated video layers). Current research studies mainly focus on advanced buffering strategies to increase the user's experienced playback quality (in terms of e.g. startup delay or video resolution) [8-10]. We investigate topology aware solutions to significantly increase the performance of a P2P (live) video streaming solutions (in the extent of the number of received video layers at each destination).

Our objectives can be summarized as:

- Designing and studying advanced (cooperative) caching strategies that use popularity and distance metrics to increase the process of locating personal content items.

- Investigating the effects of integrating a multi-level caching architecture to resolve the transport of content items to end-users.

- Study mechanisms that stimulate cooperation between peers, and are aware of the underlying network topology, and as a result increase the viewing experience for the network as whole.

1.3 Main research contributions

This dissertation's main focus is on designing advanced algorithms and protocols to locate and transport personal content items, for both non-streaming and streaming media. In general the developed strategies are required to be scalable in order to serve millions of end-devices that are using a PCSS. Our research can be divided in three major contributions:

1. *Cooperative caching for location dependent request patterns*: we study different strategies to increase the performance of locating personal content items in a DHT. The proposed cooperative caching strategy reacts on location dependent request patterns and efficiently uses the underlying DHT infrastructure. An advanced update protocol is designed that informs neighbors of cache updates. By utilizing neighbors' caches the size of the local cache is virtually increased, since nodes are able to avoid storing the same copies that can be retrieved from a neighbor (in only one overlay hop). The update mechanism itself introduces no extra lookup delay, in case of contacting a neighbor that very recently released the requested value at most one extra overlay hop is added to retrieve the lookup request. We compare our cooperative caching strategy to a state-of-the-art replication mechanism and show a significant increase in lookup performance when requests exhibit location dependent request patterns.

2. *Multi-level caching for personal content delivery*: using multi-level caching architectures to bring frequently accessed files closer to user(s), which are requesting those files often. In this way significant parts of the PCSS network can be relieved and decreases the delay for end-users accessing (their) personal content items. We analyze the storage dimensioning for the caches on each level and study the content distribution rate (i.e. the number of individual downloads necessary to fill as many caches as possible located at the first level).

3. *Topology aware multi-layer video streaming*: we look into efficiently transporting (live) video streams through the Internet. We assume that next generation P2P video streaming environments use multi-layer video to handle the burden of heterogeneous circumstances (such as different asymmetrical bandwidth capacities). By introducing a (centralized) orchestrating engine that controls video exchanges between peers, the efficiency of the data

transport in the network can be increased. In contrast to current strategies (such as Tribler [11]) that try to maximize greedily the local peer's video (i.e. download) quality, video service providers are mainly interested in the minimum video quality they can offer to their end-users. Therefore, the objective we propose maximizes the minimum received number of video layers at each destination. This contribution is subdivided into two subjects:

- To study the benefits of managing the download process, we present a mathematical formulation that models both the underlay and overlay routing of the distinct video layers in a network. The model is used to study the effects and consequences of both objective strategies in different scenarios. The results show that our proposed objective reduces the fraction of end-users only receiving the base layer.

- Since exact solvers, computing optimal solutions, are only feasible for relatively small network topologies, we provide a heuristic method to

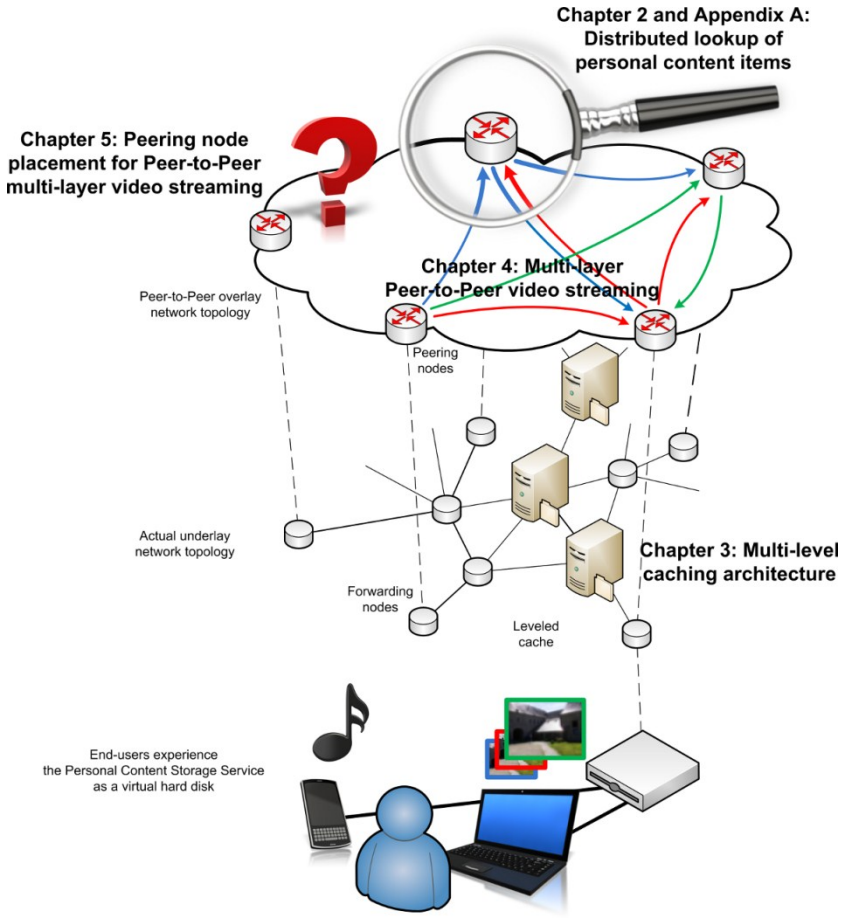

Figure 1.5: Positioning of the different chapters in this dissertation.

compute the routing process of the video layers in the topology. Additionally, the stochastic optimization strategy is extended to calculate ideal positions to upgrade nodes with peering application functionality. The results show that increasing the number of peering nodes, results in an increase in the average number of received video layers at each destination. Moreover, having more peering nodes in a network allows to increases the efficiency of link bandwidth utilization.

All proposed algorithms and protocols are systematically analyzed through simulations and compared to analytical models.

1.4 Outline of the dissertation

The dissertation is composed out of a selection of publications that were realized during the PhD. The publications provide an integral and consistent overview of the performed work. Figure 1.5 positions the different contributions that are presented in the following chapters.

Locating personal content in a distributed environment is presented in Chapter 2. Locality in user request patterns is exploited to design advanced (cooperative) caching techniques that increase the lookup performance of (general) DHTs. Although reducing lookup times to find content is highly favorable, no guarantees are made to access the content itself quickly. Therefore, Chapter 3 provides a distributed multi-level caching architecture to efficiently retrieve personal files. Chapter 4 continues with presenting a mathematical formulation to model next generation Peer-to-Peer (P2P) multi-layer video streaming frameworks. Since exact solvers are only feasible for relatively small network topologies, Chapter 5 provides a heuristic strategy to compute the routing of video layers in larger network topologies and, additionally, calculates ideal locations to upgrade nodes with peering functionality. Finally, Chapter 6 summarizes the conclusions of the dissertation, together with our perspective for future work.

1.5 List of publications

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

1.5.1 A1: publications indexed by the ISI Web of Science “Science Citation Index Expanded”

N. Sluijs, F. Iterbeke, T. Wauters, F. De Turck, B. Dhoedt, and P. Demeester, Cooperative caching versus proactive replication for location dependent request patterns, Journal of Network and Computer Applications, Vol: 34, no: 2, 2011, pp. 562-574.

N. Sluijs, T. Wauters, C. Develder, F. De Turck, P. Demeester, and B. Dhoedt, Using topology information for quality-aware Peer-to-Peer video streaming networks, Computer Networks, submitted and under review, 2012.

N. Sluijs, T. Wauters, C. Develder, F. De Turck, P. Demeester, and B. Dhoedt, Combining video layer routing with optimal peering node placement in Peer-to-Peer video streaming networks, Transactions on Emerging Telecommunications Technologies, submitted and under review, 2012.

1.5.2 C1: publications in international and national conferences

N. Sluijs, K. Vlaeminck, T. Wauters, B. Dhoedt, F. De Turck, and P. Demeester, Caching strategies for personal content storage grids, *Proceedings of the 2007 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA2007),* 2007, pp. 396-404.

N. Sluijs, T. Wauters, B. Dhoedt, and F. De Turck, Optimized search in sistributed personal content systems, *9e UGent-FirW Doctoraatssymposium,* Universiteit Gent. Faculteit Ingenieurswetenschappen, 2008, pp. 186-187.

J. Famaey, J. Donders, T. Wauters, F. Iterbeke, **N. Sluijs**, B. De Vleeschauwer, F. De Turck, P. Demeester, and R. Stoop, Comparative study of peer-to-peer architectures for scalable resource discovery, *Proceedings of the 2009 First International Conference on Advances in P2P Systems (AP2PS2009),* 2009, pp. 27-33.

N. Sluijs, T. Wauters, B. De Vleeschauwer, F. De Turck, B. Dhoedt, and P. Demeester, Caching strategy for scalable lookup of personal content, *Proceedings of the 2009 First International Conference on Advances in P2P Systems (AP2PS2009),* 2009, pp. 19-26.

1.5.3 C3: publications in national conferences

N. Sluijs, T. Wauters, B. Dhoedt, and F. De Turck, Live video streaming using peer-to-peer technologies, *11e UGent-FirW Doctoraatssymposium,* Universiteit Gent. Faculteit Ingenieurswetenschappen, 2010, p. 201.

References

[1] F. Duarte, F. Benevenuto, V. Almeida, and J. Almeida, Geographical characterization of YouTube: a Latin American view, *Latin American Web Conference (LA-WEB 2007),* 2007, pp. 13-21.

[2] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West, Scale and performance in a distributed file system, ACM Transactions on Computer Systems, Vol: 6, no: 1, 1988, pp. 51-81.

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung, The Google File System, *Proceedings of the 19th ACM Symposium on Operating Systems Principles,* 2003, pp. 29-43.

[4] W.J. Bolosky, J.R. Douceur, and J. Howell, The Farsite project: a retrospective, ACM SIGOPS Operating Systems Review, Vol: 41, no: 2, 2007, pp. 17-26.

[5] A.I.T. Rowstron and P. Druschel, Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility, *Proceedings of the 18th ACM Symposium on Operating Systems Principles,* 2001, pp. 188-201.

[6] E. Palkopoulou, C. Merkle, D.A. Schupke, C.G. Gruber, and A. Kirstädter, Traffic models for future backbone networks - a service-oriented approach, European Transactions on Telecommunications, Vol: 22, no: 4, 2011, pp. 137-150.

[7] U. Abbasi, M. Mushtaq, and T. Ahmed, Delivering scalable video coding using P2P Small-World based push-pull mechanism, *Global Information Infrastructure Symposium (GIIS2009),* 2009, pp. 1-7.

[8] Y. Ding, J. Liu, D. Wang, and H. Jiang, Peer-to-peer video-on-demand with scalable video coding, Computer Communications, Vol: 33, no: 14, 2010, pp. 1589-1597.

[9] L. Zhengye, S. Yanming, K.W. Ross, S.S. Panwar, and W. Yao, LayerP2P: using layered video chunks in P2P live streaming, IEEE Transactions on Multimedia, Vol: 11, no: 7, 2009, pp. 1340-1352.

[10] N. Ramzan, E. Quacchio, T. Zgaljic, S. Asioli, L. Celetto, E. Izquierdo, and F. Rovati, Peer-to-peer streaming of scalable video in future Internet applications, IEEE Communications Magazine, Vol: 49, no: 3, 2011, pp. 128-135.

[11] J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M.R. van Steen, and H.J. Sips, TRIBLER: a social-based peer-to-peer system, Concurrency and Computation: Practice and Experience, Vol: 20, no: 2, 2008, pp. 127-138.

2

Cooperative caching versus proactive replication for location dependent request patterns

N. Sluijs, F. Iterbeke, T. Wauters, F. De Turck, B. Dhoedt and P. Demeester.
Published in the Journal of Network and Computer Applications (2011).

One of the main challenges for a PCSS is the ability to search worldwide through the dataset of personal files. Therefore this chapter presents our research on a distributed approach to index large sets of personal content items. We provide a detailed description of our cooperative caching framework that reduces average lookup delays. The cooperative caching mechanism is compared in this chapter with a state-of-the-art proactive replication strategy (i.e. Beehive). When requests are uniformly distributed over the network, the analytical model of Beehive shows better performance increases than our caching solution. However, since lookup requests of personal content are location dependent, the measurement results obtained using location dependent request patterns indicate that our proposed solution outperforms Beehive quickly. Additionally, the simulation results show that the message overhead introduced by our cooperative framework is acceptable, since the performance increase is higher than the overhead that is introduced.

2.1 Introduction

The interaction with digital information plays an important role in our daily life. Different websites, such as YouTube and Flickr, offer platforms to store and share personal content (e.g. text documents, music files, digital photos and personal

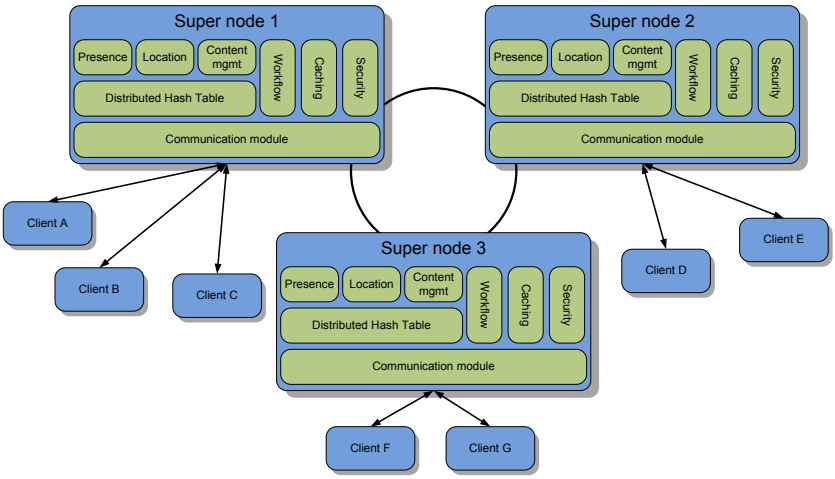

Figure 2.1: Overview of the Personal Content Storage Service architecture.

movies). Due to the explosive growth of the user's personal content collections, managing those archives becomes a complex and time consuming task. Nevertheless, end-users expect that they can access and share their personal content from any device, anywhere and at any time. Current systems that offer storage space for personal content fail to achieve this in a scalable and quality-aware way, constraints (e.g. on files sizes and formats) need to be set on the content in order to cope with the workload. To be able to deal with the future workload, a centralized approach is no longer feasible. A Personal Content Storage Service (PCSS) is a networked solution that offers storage space to end-users in a transparent manner, which can be accessed from different types of devices independent of place and time. Figure 2.1 presents an architectural view on such a distributed content management platform, where users (i.e. clients) are connected to super nodes in the PCSS overlay network.

The PCSS uses a (hybrid) Peer-to-Peer (P2P) architecture to support all necessary operations and the architecture is split-up in two high-level components: super nodes and clients. The key functions of the super node component (as schematically shown in Figure 2.1) concern user and content management (including replica management and indexing), query handling, presence management, security provisioning, monitoring of the underlying P2P network. The client component is responsible for advertising shared content as well as retrieving and uploading personal content items. For end-users the PCSS acts as a virtual hard disk, as if personal content were accessed using their local file system. Additionally, end-users are relieved from cumbersome back-up issues, since the PCSS provides data integrity through replication.

To efficiently lookup personal content references (i.e. through optimal content indexing), this chapter presents a novel cooperative caching strategy that is able to react on location dependent request patterns and making use of an underlying Distributed Hash Table (DHT) infrastructure. A DHT is a (structured) P2P network offering scalable lookup with performance and functionally similar to a traditional hash table data structure. The caching strategy introduced in this chapter is a more

adequate and detailed explanation of the base algorithm presented in [1]. In this chapter we extend our previous work by making a more thorough analysis of all parameters that are of concern for the caching framework, including results on the message overhead in relation to the network and cache size, the temporal behavior of the caching solution and a trace which elements are stored/replaced in a cache. Additionally, we provide in this chapter a detailed explanation and evaluation of our caching framework and its cache replacement strategy. The update protocol, which enables cooperative caching, is described in more detail and is extended with pseudo-code. In [1] we compared the caching framework with traditional caching algorithms (like Least Frequently Used and Least Recently Used), while, in this chapter our cooperative caching algorithm is compared to a state-of-the-art replication strategy referred to as Beehive [2]. Beehive is based on an analytical model that finds the minimal replication level for each object such that the average lookup performance is a predefined constant number of (overlay) hops. Like our caching framework, Beehive is one of the few mechanisms that increases a DHT's lookup performance without introducing any additional requirements to applications using the improved version of the DHT and is therefore an ideal candidate to compare our framework to. Our solution clearly outperforms Beehive in case of (highly) localized request patterns due to the cooperation between caches.

This chapter continues in Section 2.2 with an overview of related work, while Section 2.3 introduces the caching architecture for the PCSS. Section 2.4 provides the replication and caching algorithms, and both frameworks are validated and evaluated by simulations in Section 2.5. Conclusions and future work are presented in Section 2.6.

2.2 Related work

Different solutions exist for providing distributed storage of files, ranging from client-server systems (e.g. NFS [3], AFS [4] and Coda [5]) over cluster file systems (e.g. Lustre [6], GPFS [7] and the Google File System [8]) to global scale Peer-to-Peer (P2P) file systems (e.g. OceanStore [9], FARSITE [10] and Pangaea [11]). However, none of the distributed file system are designed for efficiently handling scattered lookup of personal content items exhibiting locality in their request distributions, which is indeed a feature inherent to personal content.

Query search in *unstructured* P2P networks is done using a query flooding mode, using a TTL (Time-To-Live) mechanism to prevent overloading the network. In order to improve the efficiency of the query flooding model, Wang *et al* describe a distributed caching mechanism for search results in [12]. However, using the TTL limit implies that personal content stored in such a network has no guarantees to be found, which makes this type of search mechanism less suitable for a PCSS. A data structure that guarantees that (even rare) objects that are stored in a network always can be located is called a *Distributed Hash Table* (DHT). A DHT is a *structured* P2P network that offers scalable lookup, similar to a hash table, where the average number of (overlay) hops per lookup request has a complexity of $O(\log N)$ with N the number of nodes in the DHT network. Different implementations of a DHT already exists, such as Chord [13], Pastry [14], and Tapestry [15]. Various research studies have been performed to improve the lookup performance of DHTs. The Beehive [2] framework enables DHTs to provide an average (i.e. for all stored

objects in the DHT) lookup performance of $O(1)$ through proactive replication. According to the evaluation made in [2], Beehive outperforms the passive caching technique used by Pastry [16] in terms of average latency, storage requirements, work load distribution and bandwidth consumption. In passive caching, objects are cached along all nodes on a query path [2], while Beehive's replication strategy consists of finding the minimal replication level for each object such that the average lookup performance for the system is a constant C number of hops [2]. Beehive assumes that there is a global power law (or Zipf-like) popularity distribution and requests are uniformly distributed over the network. However, in the scenario of the PCSS it is conceivable that locality exists in request patterns [17], which has a major influence on the performance of a caching algorithm and requires a less expensive solution than Beehive.

In [18] results of queries are cached and are re-used to answer more detailed queries. In this way unnecessary duplication of work and data movement is avoided. The results of (conjunctive attribute) queries are cached in a view tree and are used later on to resolve queries that contain (parts of) the cached query results. Although the view tree tries to avoid duplication of work and data movement, each search query is issued to the root (node) of the view tree. This aspect prevents successful deployment of a view tree in a PCSS system, since it introduces a single point of failure.

Previous studies on caching techniques [19] or distributed replica placement strategies for *Content Distribution Networks* (CDN) [20,21] show that by taking distance metrics and content popularity into account, a performance increase is obtained compared to more straightforward heuristics such as *Least Recently Used* (LRU) or *Least Frequently Used* (LFU). An even larger performance increase can be obtained by using *cooperative caching* [22], compared to independent caching. The caching strategy of [22] is not directly applicable onto a DHT, since their algorithms are designed for efficiently delivering multimedia streams and do not take the basic architecture of a DHT into concern. However, the general concepts [22] introduce still apply to our work; in cooperative caching it is important to keep track of (neighbor) cache states and as a result of using neighbor caches the load is more evenly balanced among the nodes, leading to improved system scalability. The proposed caching strategy uses the distance metrics and content popularity, as well as cooperative caching to increase the PCSS lookup performance, where references of content are stored that exhibit locality in the distribution of requests over the network.

In [23] a cooperative caching strategy is proposed for increasing the performance of queries on Extensible Markup Language (XML) documents that are stored distributed in a network. A DHT is used to let the caches cooperate with each other by using a loosely coupled or tightly coupled strategy. The loosely coupled approach caches the results of each query locally at the peer that posed it and a DHT is used to provide an index of the query results so that the results can be located by other peers. The tightly coupled strategy assigns to each peer a specific part of the query space and results of queries are cached at the peer that is responsible for the corresponding part of the query space. The advantage of the tightly coupled strategy is that there is control over the placement of cached content and thus there is no redundancy. However, compared to the loosely coupled approach it induces an additional overhead of moving query results from the posing peer to the caching peer and no advantage is taken from location dependent request patterns. Although the

cooperative caching strategy of [23] uses a DHT, it does not improve the basic working of a DHT. Therefore, like the strategy of [22], the general concepts about sharing information between caches still apply, but they cannot be directly compared to our cooperative caching solution.

2.3 Caching architecture for DHT performance optimization

Since the dataset of personal content is extremely large and in order to deal with the future workload, a distributed approach to index the personal content collection is a prerequisite for the PCSS. As explained above, a *Distributed Hash Table* (DHT) allows for highly scalable lookup in extremely large distributed datasets. A <*key*, *value*>-pair is stored into the DHT and every node participating in the DHT is able to efficiently locate *values* that correspond to a certain *key*. For the PCSS, the *key* can be a file name, or could alternatively represent tags/keywords describing the personal content item. Often, the *value* represents a link to the location where the content is actually stored. To further optimize the content lookup process, typically a caching layer is introduced on top of the DHT (e.g. Pastry [16] and Beehive [2]). The caching layer is located between the application and DHT layer, and typically stores search results of important requests, as shown in Figure 2.2.

In the example of Figure 2.2, eight nodes span the Chord-based DHT network for storing references to locations of personal content. In this chapter, Chord is used as DHT implementation, in view of its wide spread use and its inclusion in multiple

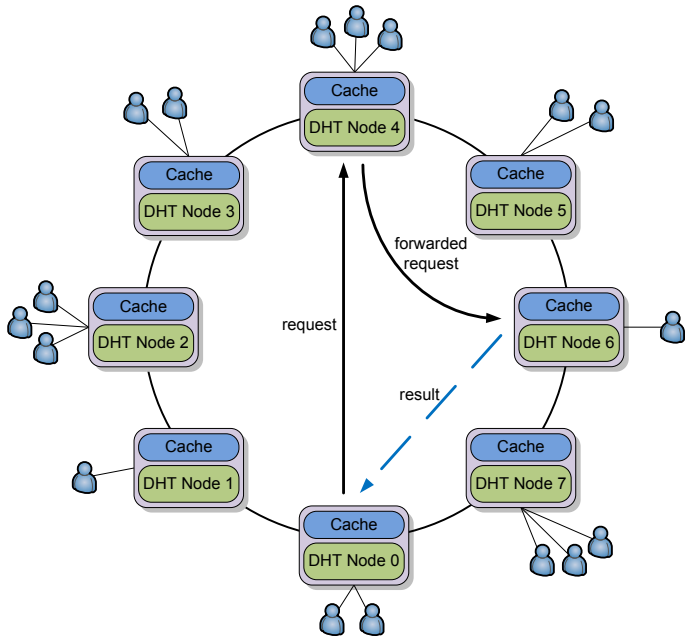

Figure 2.2: The Personal Content Storage Service enhances the distributed hash table with a caching architecture to increase the lookup performance.

P2P network simulators. The approach taken, however, can be applied to any underlying DHT implementation. By using a *hash* function both content references and nodes can be mapped to a *numeric identifier space*. In Figure 2.2 we assume that nodes depicted with a higher number have a higher numeric identifier. Each node is responsible for storing *values* belonging to *keys*, which numeric identifier is between the numeric identifier of the preceding DHT-node (excluding) and the numeric identifier of the current node (including). In order to efficiently route messages in a DHT, every node keeps a *finger table*. This finger table maps numeric identifiers to nodes, where the distance between the numeric identifier of the current node and the numeric identifiers in the finger table increases exponentially. In this way, messages are sent to a node at least half the distance of the key space closer to the destination node. When using the same numeric identifier space as the node numbers in Figure 2.2, the finger table of, e.g., node 0 contains mappings to node 1, 2 and node 4. In this way the average (and worst case) number of hops for a lookup has a complexity of $O(\log N)$, where N is the number of nodes in the DHT network [13].

When a user requests a personal content object in the PCSS, the DHT is used to lookup the link to the location the object is stored. Figure 2.2 also presents an example of a traditional lookup request, initiated by a user connected to node 0. Node 0 forwards the request to the node in its finger table with the numeric identifier closest to and smaller than the hash value (i.e. node 4), this process is repeated until the target node is reached (i.e. node 6). Finally, the target node replies directly to the requesting node (i.e. node 0). Storing references to object locations into a DHT is performed in a similar way, except no reply message is returned. Since the *value*-part of <*key*, *value*>-pairs are typically locations where (the latest version of) personal content items are stored, no synchronizations need to take place.

To improve the lookup performance, the PCSS provides each node with a relatively small amount of storage space (the cache) to temporarily duplicate <*key*, *value*>-pairs, obtained from lookup results on the DHT. The cache contains a *monitoring service* component for measuring object popularity and for keeping track of neighbor cache information. By storing <*key*, *value*>-pairs on average closer to end-users, the average time (measured in number of hops) needed for a lookup decreases. Another benefit of the caching architecture is that multiple nodes are able to handle lookup request of popular content, which alleviates the hotspot problem (i.e. sudden huge popularity of a limited set of content items) enormously.

2.4 Cooperative caching and proactive replication mechanisms

In order to utilize the available cache space on each node efficiently, a caching or replication algorithm is mandatory to decide which entry to remove for a more valuable lookup result. The popularity of personal content is typically described by a *power law* (*Zipf-like*) distribution. This distribution states that some personal content is highly popular and the remainder of the content is more or less equally popular. In (1) the Zipf-like probability mass function [24] is provided, where M denotes the number of personal content items and α is the exponent characterizing the distribution.

$$P_{Zipf-like}(x) = \frac{x^{-\alpha}}{\sum_{a=1}^{M} a^{-\alpha}} \quad (1)$$

$P_{Zipf\text{-}like}(x)$ determines the probability that a personal content object having rank x is requested, where $x \in \{1, \ldots, M\}$. This implies that a personal content object having a lower rank (i.e. a larger value for x) is less popular, $\alpha > 0$. Typically for P2P file sharing applications the value of α is between 0.6 and 0.8 [25].

In Section 2.4.1 the analytical model of Beehive's proactive replication strategy is explained in detail and in Section 2.4.2 our cooperative caching strategy is introduced.

2.4.1 Beehive: proactive replication

The replication framework of Beehive [2] enables a DHT to achieve (on average) constant lookup performance for power law (Zipf-like) popularity of stored content. Through proactive replication Beehive reduces the average number of (overlay) hops, where copies of stored content are actively propagated among multiple nodes in the network. The goal of Beehive's replication strategy is to find the maximum replication level L for each object such that the average lookup performance for the system is a predefined constant number of hops [2]. In order to reach this goal, popular items are replicated to more nodes than less popular objects, aiming to minimizing both storage and bandwidth overhead. According to [2], Beehive is a general replication framework that can operate on top of any DHT implementation using prefix-routing, such as Chord [13]. In Chord-based DHT implementations the search space halves in each step of the lookup process (i.e. Chord is a DHT with base 2) and therefore provides $O(\log N)$ lookup performance, where N is the number of nodes in the DHT network. The main idea of Beehive is that the *maximum* number of hops for a lookup is reduced by h hops if objects are proactively replicated to all nodes on all query paths that logically precede the home nodes for the last h hops. A home node is the responsible DHT node for storing an object, according to the numeric identifier produced by the hash function of the *key*. Beehive controls the number of replicas by assigning each stored object a replication level L. The *maximum* number of (overlay) hops, for every node in the DHT, to locate an object on level L equals L. When Chord (i.e. $b = 2$) is considered, each object replicated at level L is stored on $N/b^L = N/2^L$ nodes, where N is the number of nodes in the DHT. Figure 2.3 illustrates the replication level mechanism of Beehive.

In Figure 2.3 a Chord-based DHT network is considered with 8 nodes, which means that the maximum replication level $k = log_2(8) = 3$. All personal content references on level 3 are only stored on the home nodes of the objects. On level 2 personal content item references are replicated to $8/2^2 = 2$ nodes, including the home node. The number of replicas made on level 1 is $8/2^1 = 4$ and level 0 lets all nodes store a replica of the personal content reference. The lookup query inserted at node 0 to lookup a personal content reference that is located on node 7, requires 3 (overlay) hops when the object is only stored on its home node (i.e. the replication level is 3). When the replication level for this object is set to 2, Figure 2.3 depicts that the number of (overlay) hops is reduced to 2 hops. The (maximum) number of

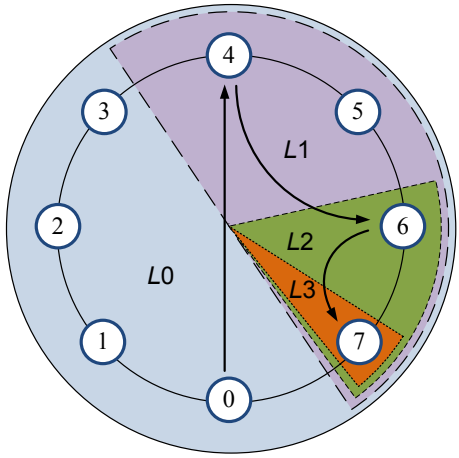

Figure 2.3: Beehive's level of replication mechanism, the maximum level *L* for this situation is three. The lower the level for a stored item in the DHT, the more it is replicated to other nodes. The goal is to find the minimal replication level for each item such that the average number of (overlay) hops per lookup is constant.

hops can be reduced to one hop, when the object has a replication level of 1. When the number of hops for the objects has to be 0, all nodes store the object (i.e. replication level 0). In this way each stored item in the DHT receives a replication level, based on the popularity of the item, so that the *weighted average of the maximum* number of hops for a lookup request for a stored item in the DHT matches a predetermined target number *C*.

Let f_i denote the fraction of items replicated at level *i* or lower (i.e. $f_k = 1$, where *k* is the maximum replication level). The fraction of items that are replicated at all nodes is expressed by f_0 and when *M* denotes the total number of items stored in the DHT, $M \times f_0$ equals the total number of objects that are stored on each node in the DHT (i.e. these are the most popular objects in the network). The number of objects that have a *maximum* number of *i* (overlay) hops per lookup request is $Mf_i - Mf_{i-1}$. The average storage (i.e. average number of objects stored) on a node for a DHT implemenation with base *b* is expressed whith the following equation [2]:

$$Mf_0 + \frac{M(f_1 - f_0)}{b} + \cdots + \frac{M(f_k - f_{k-1})}{b^k} \tag{2}$$

When $Q(m)$ represents the total number of lookup requests to the most popular *m* items, the number of queries that travel a *maximum* of *i* (overlay) hops is $Q(Mf_i) - Q(Mf_{i-1})$. The target number of hops *C* is reached when the following expression is fullfilled on the *weighted average of the maximum number of (overlay) hops*:

$$\sum_{i=1}^{k} i \cdot \frac{Q(Mf_i) - Q(Mf_{i-1})}{Q(M)} \leq C \tag{3}$$

Note that the target number of hops *C* is the weighted average of the *maximum* number of (overlay) hops for a lookup request for a stored item in the DHT and not the average number of (overlay) hops as considered in [2]. Finally, assume that in the optimal solution the problem $f_0 \leq f_1 \leq \cdots \leq f_{k'-1} < 1$, for some $k' \leq k$. In [2] this leads, using equation (3), to the following closed-form solution that minimizes the (average) storage requirement but satisfying the target number of hops *C*:

$$f_i^* = \left[\frac{d^i \cdot (k' - C')}{1 + d + \cdots + d^{k'-1}}\right]^{\frac{1}{1-\alpha}}, \forall 0 \leq i < k' \tag{4}$$

Where $d = b^{\frac{1-\alpha}{\alpha}}$, $C' = C \cdot \left(1 - \frac{1}{M^{1-\alpha}}\right)$ and α is the parameter describing the (Zipf-like) popularity distribution. The value of *k'* can be derived by satisfying the condition that $f_{k'-1} < 1$, that is, $\frac{d^{k'-1} \cdot (k' - C')}{1 + d + \cdots + d^{k'-1}} < 1$. All $f_i^* = 1, \forall k' \leq i \leq k$. With the closed-form solution of (4) the fraction f_i is approximated by f_i^*, to achieve the desired constant lookup performance and *k'* represents the upper bound for the worst case number of (overlay) hops for a lookup request.

Since the analytical model of Beehive provides the optimal solution to increase the lookup performance, we use the analytical model to compare it with our cooperative caching strategy. In the experiments we have assumed that the popularity of items is known, such that the replication level can be set for all items. This approach allows investigating the performance after warm-up of the system. Note that, unlike our cooperative caching framework, Beehive has an advanced protocol to estimate the overall popularity distribution in the network. Our cooperative caching algorithm only needs to know the popularity of the items on each individual node, which is measured by the number of local requests to an object *a* from a node *n*.

2.4.2 RTDc: cooperative caching

An important concept for a caching algorithm is that locality exists in the request patterns of nodes inserting lookup requests. This idea is supported by the research performed by Duarte *et al* in [17], where geographical characterizations of requests patterns are studied for YouTube content. However, until now no concrete and generalized probability mass function has been proposed (either based on theoretical or experimental grounds) that describes the locality based request distribution. Here, we model locality using a *Normal* function[1], where the mean is μ and σ the standard deviation.

[1] Since the request pattern is the sum over multiple aspects, the Normal function is presumed to be a valid distribution to model locality.

Let $P_{Normal}(y)$ be the probability that a personal content item is requested from node y. Parameter μ represents the uploading (super) node, since it is conceivable that the (super) node that inserts the personal content object has the highest probability to request it. The value σ is used to model the locality of requests over the network. A higher value of σ makes the distribution more uniform, since more neighboring nodes will request the personal content item.

Basic DHTs use hash functions to map nodes onto the numeric identifier space, which means that nodes are more likely to have different neighbors in the DHT than in the actual network topology. Different research studies are already performed that address the issue of including physical neighboring nodes as logical neighbors in DHTs [26,27], in order to reduce latencies in overlay hops. In [26] the network topology is embedded into the DHT by assigning a locality-aware identifier to each node. In our use case, we assume that the DHT is locality-aware, neighbors in the PCSS overlay network map to neighboring nodes in the physical network.

Since we want to reduce the average number of hops needed for a lookup, the caching algorithm we propose reacts on both *popularity* and *distance* of lookups. The popularity $p_{n,a}$ represents the total number of requests to an object a, initiated by node n. The distance $d_{n,a}$ of a personal object a is measured by the number of (overlay) hops needed to obtain the lookup result from the requesting node n and the responsible node storing the object. Since objects can be cached (multiple times) in the network, the distance for an object is the minimal number of (overlay) hops of a (previous) successful lookup retrieval. The importance $I_{n,a}$ for node n to store object a, which is used as a metric in the *Request Times Distance* (RTD) caching algorithm, is calculated as:

$$I_{n,a} = p_{n,a} \times d_{n,a} \tag{5}$$

Consequently, the references to personal content objects with the highest importance values for $I_{n,a}$ in (5), will be stored in the local cache of node n. In [1] the RTD caching algorithm is extended with a sliding window in order to react on changes in content popularity. By adding a sliding window, only the last T requests that arrived in a node are used to determine the popularity of the requested content. However, to compare the caching algorithm with the analytical model of Beehive the sliding window size is set to infinite, since the popularity distribution of the stored content is constant during each simulation run.

Since in a Chord-based DHT each node knows its predecessor and successor node on the DHT ring (to be able to update finger tables when nodes suddenly join or leave the DHT network) the performance of the caching algorithm can be increased by keeping a local copy of both neighbors' cached *keys*. In order to keep the storage overhead to a minimum, only keys of both direct neighbors are kept locally. This *cooperative caching* strategy utilizes the neighbors' caches to virtually increase the size of the local cache. In this way, nodes can avoid storing the same copies of <*key*, *value*>-pairs that can be retrieved from their neighbor, in only one hop. Figure 2.4 visualizes the *update protocol* for the three possible scenarios of performing a lookup using cooperative caching. In all scenarios the destination node for the lookup is node 6 (i.e. the node responsible for storing the *values* belonging to the search *key*), the request is initiated from node 0 and the node numbers are used

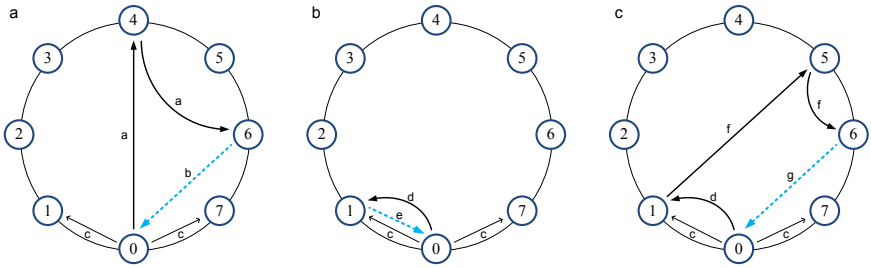

Figure 2.4: Three scenarios for a lookup using cooperative caching. Scenario (a) describes the case where local copies of neighbor cache entries do not contain the search key. In scenario (b), one of the local copies of the neighbor's cache contains the search key and in scenario (c) the situation that the requesting node wrongly assumes that its neighbor's caches the search key is depicted.

as the numeric identifier space. Figure 2.4a considers the case where the local copies of the cache entries of the neighbor nodes do not contain the search key of object *a*. The scenario in Figure 2.4b describes the case that the local copy of the cache entry of the neighbor node, in this case node 1, contains the search key. And Figure 2.4c represents the scenario where node 0 wrongly assumes that node 1 caches the search key of object *a* (i.e. in between cache update messages).

When the local copy of the neighbor's cache does not contain the search key (Figure 2.4a), the lookup is performed as usual. A request message REQ (a) is routed via node 4 to node 6. Node 6 responds by sending the requested value using a reply message REP (b). In the case that node 0 decides to cache the lookup value, it updates the local cache table of both its neighbor nodes with the cache update message CACHE (c). These nodes then re-compute their values of the importance $I_{1,a}$ and $I_{7,a}$ of object *a*, as the distances $d_{1,a}$ and $d_{7,a}$ are now equal to one hop. No extra lookup delay is introduced by this update mechanism.

In the case that one (or both) of the local copies of the neighbor's caches contain the search key, the lookup request is routed to that neighbor node. In Figure 2.4b, the situation is presented where the local copy at node 0 of the cached entries in node 1 contains the search key. As a consequence, the request message REQ (d), initiated by node 0, is forwarded to node 1. When node 1 still has the value of the search key in its cache, it updates the popularity $p_{1,a}$ and responds the value using the reply message REP (e). Node 0 again decides whether or not to cache locally the lookup value, in the case node 0 keeps the lookup results in its local cache it uses the cache update message CACHE (c) to inform the neighbors.

The situation that node 1 no longer caches the value of the search key and has not sent the corresponding cache update message CACHE to its neighbors yet (i.e. it very recently released the value), is illustrated in Figure 2.4c. The lookup message REQ (d) is forwarded by node 1 as usual using the request message REQ (f) via node 5 to node 6. Node 6 responds with the value of the search key, using the reply message REP (g). Similar to the other two scenarios, node 0 decides whether or not to store the result in its cache by computing the importance $I_{0,a}$ of object *a* (with distance $d_{0,a} = 1$ in case the entry is stored in its other neighbor's cache), and informs the neighbors in case of a cache change with the cache update message CACHE (c). Only in the case when a neighbor is contacted erroneously because it very recently

```
A.01  initiateLookupRequest(key) {
A.02        if(storedOrCachedOnThisNode(key)) {
A.03              return lookup_result;
A.04        } else if(neighborCachesKey(key)) {
A.05              sendLookupRequest(neighbor, key);
A.06        } else {
A.07              targetNode = hash(key);
A.08              sendLookupRequest(targetNode, key);
A.09        }
A.10  }
```

Figure 2.5: Pseudo-code for initiating lookup requests.

released the requested value, one extra hop is added to the lookup delay. In all other cases, no extra delay is introduced.

To illustrate the inner working of the RTD caching algorithm, Figure 2.5 presents the pseudo-code for the method that describes the initiation of a lookup request, Figure 2.6 illustrates the process of receiving a lookup request, the method handling a lookup reply is shown in Figure 2.7 and receiving a cache update message is presented in Figure 2.8. Other routines used by the DHT are not changed by the RTD algorithm.

When a user initiates a lookup request, the method on line A.01 of Figure 2.5 is invoked. When the node is already storing (or caching) a local copy of the lookup result itself (A.02), the result is returned directly to the user (A.03). Otherwise, the node checks whether a neighbor caches the lookup result (A.04) and, if so, the request is then sent to that neighbor (A.05). In case the result is not stored or cached locally, and not available through a neighbor's cache, the request is sent as a traditional DHT lookup (A.08) by using the hash function (A.07) to determine the target node.

Upon receiving a lookup request (B.01), the node replies (B.06) the result to the requesting node when the node is storing or caching the lookup result (B.02). When the node is caching the key (B.03) the popularity counter is updated (B.04). In the

```
B.01  receiveLookupRequest(key) {
B.02        if(storedOrCachedOnThisNode(key)) {
B.03              if(cachedOnThisNode(key)) {
B.04                    updateCounters(key);
B.05              }
B.06              sendLookupResult(key);
B.07        } else {
B.08              targetNode = hash(key);
B.09              sendLookupRequest(targetNode, key);
B.10        }
B.11  }
```

Figure 2.6: Pseudo-code showing the processing when receiving a lookup request.

```
C.01  receiveLookupReply(key, value) {
C.02        updateCounters(key);
C.03        if(storedOrCachedOnThisNode(key)) {
C.04              return;
C.05        }
C.06        lowest_importance_value =
C.07        getLowestImportanceValueofCachedKeys();
C.08        lookup_importance_value =
C.09        calculateImportanceValue(key);
C.10        if(lookup_importance_value >
C.11                    lowest_importance_value) {
C.12              removed_key =
C.13        removeLowestImportanceValueKeyFromCache();
C.14              insertNewKeyIntoTheCache(key,
C.15                                       value);
C.16        updateCacheChangeToNeighbors(removed_key,
C.17                                     key);
C.18        }
C.19  }
```

Figure 2.7: Pseudo-code executed when receiving a lookup reply.

case the node is not storing or caching the lookup result (in the situation of Figure 2.4c), the request is forwarded (B.09) as usual to the target node (determined by using the hash function (B.08)).

The node initiating the lookup request receives the lookup reply through method C.01 in Figure 2.7. First, the counters that measure the popularity of objects and the distance (i.e. overlay hop count) needed to obtain the lookup request are updated (C.02), ensuring that the importance values are calculated correctly. In the case where the node already stores or caches the lookup result, no further actions need to take place (C.03). Otherwise, the entry in the local cache having the lowest importance value is retrieved (C.06) and the importance value of the lookup result is calculated (C.08) using equation (5). When the importance value of the lookup result is larger than the lowest importance value (C.10), the entry having the lowest importance value is evicted from the cache (C.12) and replaced by the lookup result (C.14). Finally, the neighbors are updated of the local cache change (C.16) using the cache update message CACHE.

```
D.01  receiveCacheChangeUpdateOfNeighbor(removed_key,
D.02                                     key) {
D.03        neighbor_cache.remove(removed_key);
D.04        neighbor_cache.add(key);
D.05        updateImportanceValueofCachedKeys();
D.06  }
```

Figure 2.8: Pseudo-code describing the process of receiving a cache update message.

When a node receives the cache update message CACHE (D.01), the node removes the old entry (D.03) from the local copy of the specific neighbor's cache entries and adds the new neighbor's cache entry into the local copy (D.04). The node than updates the importance values of its cache entries, since the distance values might be changed due to the update (D.05).

2.5 Evaluating cooperative caching with proactive replication

In order to compare the (cooperative) RTD caching algorithm to the analytical model of Beehive, the discrete-event simulator PlanetSim [28] is used. PlanetSim offers a framework to simulate large scale overlay services and networks, such as DHTs, and can be extended at the network, overlay or application layer. For the validation and evaluation of caching algorithms, we have extended PlanetSim at the application layer with a lookup service that can use the RTDc caching algorithm or Beehive's replication model. An advantage of PlanetSim is that it already has an implementation of the DHT lookup protocol Chord [13]. For each simulation the DHT network is created by PlanetSim and randomly selected <*key*, *value*>-pairs are inserted into the network, so that every personal content reference is initially stored on only one node. All stored items in the DHT are ranked according to the popularity distribution and when locality is required in the request pattern, the *Normal* distribution is used to compute the probability that an item is requested from a node. The mean value of the locality distribution is the uploading (super) node and the standard deviation (parameter σ) is used to control the uniformity of this distribution. The probability $P_{requested}(x, y)$ that an item having rank x is requested from a node y is calculated as[2]:

$$P_{requested}(x, y) = P_{Zipf-like}(x) \times P_{Normal}(y) \tag{6}$$

When the replication strategy of Beehive is used, all objects are replicated into the caches according to the analytical model of Beehive. When the RTDc (cooperative RTD) caching algorithm is used, the sizes of the caches are calculated according to the analytical model of Beehive and are left empty. To initialize (i.e. warm-up) the network properly for RTDc, a startup phase is used where search queries enter the network using the cooperative caching algorithm to decide which lookup result to cache. After the whole network is initialized properly, search queries are made by the peers according to the popularity and locality distribution. The simulation stops when the network reaches a non-equilibrium steady state, i.e. when the average number of hops and the cache hit ratio have stabilized. In order to cancel out noise due to random fluctuations, the average values over ten independent simulation runs are used.

In Section 2.5.1 both algorithms are compared using the traditional uniform distribution of requests over the DHT network. The distribution of lookup requests for personal content retrieval is expected to be more localized, therefore Beehive is

[2] $P_{requested}$ is used in section 2.5.3.4 to calculate the local theoretical importance rank of the personal content items for each node.

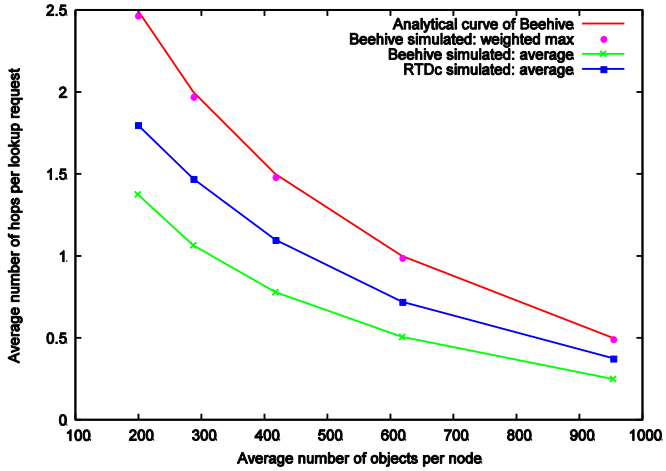

Figure 2.9: Number of hops per lookup request in relation to the average storage per node, for Beehive analytically calculated with dots representing simulated results of the weighted average of the maximum number of hops, simulated average number of hops for Beehive and simulated average number of hops for the cooperative RTD caching algorithm.

compared with RTDc using the locality in lookup requests in Section 2.5.2. Finally, section 2.5.3 makes a more detailed study of the RTDc caching algorithm, addressing the message overhead of the update protocol and the temporal behavior of the RTDc caching framework.

2.5.1 Comparing Beehive with RTDc for traditional distribution of lookup requests

The analytical model of Beehive is used to calculate the replication factor for each personal content reference. The solution that Beehive proposes aims to minimize the storage space (i.e. number of personal content references stored), while offering a predetermined average number of (overlay) hops per lookup request, where the distribution of lookup requests over the network is expected to be uniform. As explained in Section 2.4.1, the target hops C of Beehive is the weighted average of the maximum number of (overlay) hops per item stored in the DHT. Figure 2.9[3] illustrates the relation between the average number of (overlay) hops in relation to the average storage space on a node. Three different curves are presented in Figure 2.9: Beehive calculated analytically (the dots representing the simulated results of the weighted average of the *maximum* number of hops), the simulated average number of hops for Beehive and simulated average number of hops for the cooperative RTD caching algorithm. In order to compare our caching framework

[3] The maximum standard deviation over the measured averages was less than 1.3 percent, indicating that ten independent simulation runs is sufficient to cancel out noise due to random fluctuations.

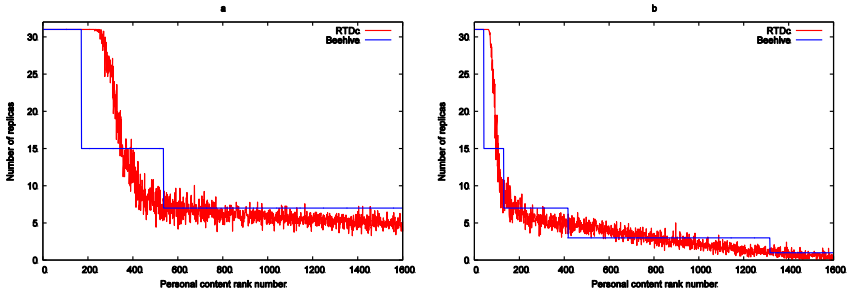

Figure 2.10: Relation between the number of caches that store a personal content reference and the popularity rank of the personal content reference for the both the model of Beehive as the cooperative RTD caching algorithm. A lower rank indicates a higher popularity and in (a) the target number of hops C is set to 1.0 and in (b) to 2.0.

with Beehive, all input parameters are set beforehand and used for each personal content item to determine in which replication level it belongs. In the simulation setup the network size N is set to 32 nodes, the number of personal content items M is $50 \times N$, the power law (Zipf-like) popularity distribution parameter α is 0.6.[4] In order to get representative results for the weighted average of the maximum number of hops for each personal content object, a random personal content item is selected and is requested by all nodes in the DHT network.

As shown in Figure 2.9, for relatively small caches, Beehive outperforms the RTDc considerably. This due to the fact that each node in the DHT makes independent estimations of the popularity distribution. When the cache space is relatively small, small mistakes in the estimations of the most important content have a high impact on the performance of the caching algorithm. As explained in Section 2.4.1, the output of Beehive's analytical model is the weighted average of the maximum number of overlay hops per stored item. Therefore, the simulated weighted average of the maximum number of hops is plotted onto the analytical curve, which shows the correct working of the simulation framework. In order to compare the average number of (overlay) hops for RTDc and Beehive, the simulation results of the average number of hops per lookup request for Beehive are used in the remainder of this chapter.

Figure 2.10 depicts the number of nodes storing a replica for each personal content item for both the Beehive and RTDc strategies, where the personal content items are sorted by their popularity rank (i.e. the smaller the rank, the more popular the personal content object). The same simulation results are used as for Figure 2.9 (i.e. $N = 32$, $M = 50 \times N$ and $\alpha = 0.6$), the target average number of hops C for Figure 2.10a is set to 1.0 and for Figure 2.10b to 2.0.

Figure 2.10 shows that Beehive replicates a larger fraction of popular content items to more nodes, in order to decrease the average number of hops. When the same amount of storage space is provided to the RTDc caching framework, the

[4] These values provide enough depth to study the improvements of our proposed solution, though the number of users and items is relatively small compared to a realistic PCSS.

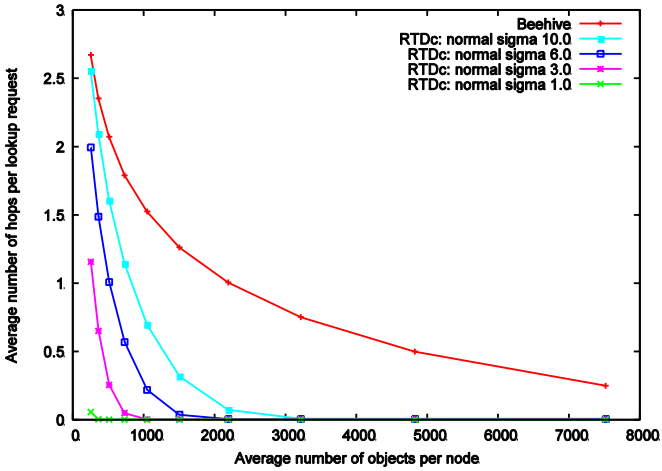

Figure 2.11: Influence on the performance of introducing locality in lookup request for RTDc with different values for the locality variance parameter σ. Since the performance of Beehive is not affected by the locality distribution of lookups, the curve representing the average number of hops for Beehive (of Figure 2.9) is plotted as a reference.

RTDc caching algorithm also tries to cache popular content more often. However, the performance increase of RTDc compared to Beehive is lower (as depicted in Figure 2.9), since RTDc also caches a lot of relatively unpopular content. Note that the analytical model of Beehive has a perfect centralized view on content popularity beforehand.

2.5.2 Comparison between RTDc and Beehive for distributed lookup of personal content

In Figure 2.11 the influence on the performance when introducing locality of lookup requests on RTDc is shown, when the simulation has reached the non-equilibrium steady state situation. Since the performance of Beehive is not affected by the locality distribution of lookups, the average number of hops curve of Beehive (see Figure 2.9) is plotted as a reference. In order to get results for a larger P2P network the network size N is scaled to 256 nodes, the number of personal content items M is $50 \times N$, the power law (Zipf-like) popularity distribution α is 0.6 and the locality parameter σ ranges from 1.0 to 10.0.
Figure 2.11 illustrates that a higher locality in the request pattern (i.e. a lower value of the locality variance parameter σ) increases the lookup performance when RTDc is used, using the same amount of storage space as Beehive requires. When the cache space is relatively low, a more localized requests distribution induces a relatively higher performance gain.

2.5.3 Detailed evaluation of the RTDc caching algorithm

In this section the RTDc caching algorithm is evaluated in more detail in terms of message overhead, and more specifically overhead generated by the update protocol described in Section 2.5.3.1. This is done by inspecting the fraction of lookups that uses cooperative information versus standard lookup requests in Section 2.5.3.2. In addition, Section 2.5.3.3 examines the dynamic behavior of RTDc. Section 2.5.3.4 investigates the content of caches in terms of popularity.

2.5.3.1 Message overhead of the update protocol for cooperative caching

Although the main goal of the caching framework is to reduce the average number of hops required to obtain a lookup result, the message overhead created by keeping cache states of neighbors up-to-date should be as a low as possible. Therefore, the average number of messages sent (and forwarded) for a lookup request is shown in Figure 2.12, in relation to the network size (Figure 2.12a and Figure 2.12b) and the cache size (Figure 2.12c and Figure 2.12d). For these simulations the network size N is set to 256 nodes, the cache size is 10 items, the number of personal content items M is $50 \times N$, the power law (Zipf-like) popularity distribution α is 0.6 and the locality parameter σ is set to 1.0 and 3.0.

In Figure 2.12a ($\sigma = 1.0$) and Figure 2.12b ($\sigma = 3.0$) the message overhead is

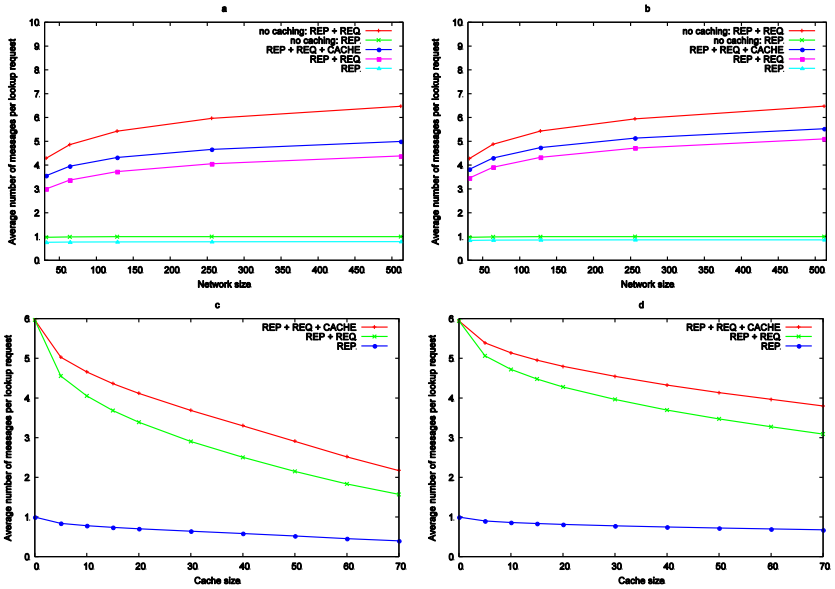

Figure 2.12: Average message overhead for a lookup request in relation to the network size (a: $\sigma = 1.0$ and b: $\sigma = 3.0$) and the cache size (c: $\sigma = 1.0$ and d: $\sigma = 3.0$). The message overhead is measured by the number of reply messages REP, request messages REQ en cache update messages CACHE. The cache size for (a and b) is set to 10 entries per node and is compared to the situation no caching is used. The network size of (c and d) is set to 256 peers.

depicted in relation to the network size, for the situation where no caching is used and the case that every node has a cache size of 10 items. The average amount of reply messages for a lookup request is (close to) one (i.e. only requests for items located on the requesting node need no lookup reply message REP) and independent of the network size, for the situation where no caching is used. When caching is enabled, the average number of reply messages REP per lookup request further reduces, since cache hits on the requesting node need no reply message as well and is still independent of the network size (i.e. the total number of items and cache size increases linearly with the network size). When the caching framework is used, both Figure 2.12a and Figure 2.12b show that the average number of sent and forwarded messages to obtain a lookup result (REP plus REQ messages) decreases significantly compared to the situation caching is disabled. The total cost (measured in terms of average number of messages for each lookup request, including cache update messages CACHE) of the cooperative RTD caching algorithm in Figure 2.12a and Figure 2.12b is considerably less than the total average cost for a lookup request when no caching is used, for all network sizes. This implies that the cooperative caching framework is able to efficiently update cache states to neighbors, without introducing extra network overhead.

When the cache size increases, Figure 2.12c ($\sigma = 1.0$) and Figure 2.12d ($\sigma = 3.0$) illustrate that the average number of reply and request messages decreases, since more objects are cached at (multiple) nodes. The message overhead created by the update protocol slightly increases when the cache size increases (i.e. the required number of cache update messages CACHE), because multiple items of similar popularity are stored in the same cache, which results in more cache changes taking place. However, the increase in the average number of cache update messages CACHE in Figure 2.12c and Figure 2.12d is much smaller than the decrease in average number of reply and request (REP plus REQ) messages and therefore has no negative impact on the performance of the caching algorithm. The benefits of using the cooperative caching via the cache update protocol are higher than the cost that is introduced to keep neighbor cache states up-to-date.

2.5.3.2 Fraction of lookup request using cooperative versus standard lookup

In [1] we show that using the update protocol to inform neighbors of cache state changes results in a performance surplus for the RTD algorithm, since the average number of cache duplicates between neighbors is reduced significantly. To understand this performance increase better, Figure 2.13 depicts the fraction of lookup requests that use cooperative information and the fraction performing standard DHT lookups. The same simulation setup is used as for Figure 2.12 (i.e. $N = 256$, $M = 50 \times N$, $\alpha = 0.6$, and σ is 1.0 and 3.0).

Figure 2.13a ($\sigma = 1.0$) and Figure 2.13b ($\sigma = 3.0$) indicate that when the network size increases, the fraction of lookup requests using cooperative information (i.e. a neighbor caches the result of the lookup request) is stable. However, when the cache size increases, Figure 2.13c ($\sigma = 1.0$) and Figure 2.13d ($\sigma = 3.0$) illustrate that the fraction of lookup requests using cooperative information initially increases and then slightly decreases. The increase can be explained by the fact that nodes get more space available to cache lookup results that are also requested often by their neighbors. When the cache space increases even further, all nodes can store those lookup results themselves and therefore the fraction of lookup requests using

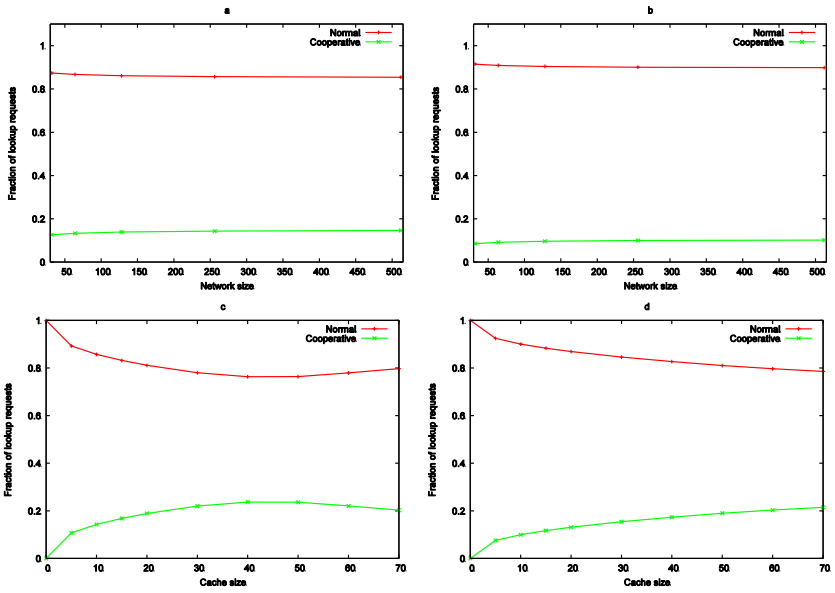

Figure 2.13: The fraction of lookup requests that is performed using standard lookup and the fraction using cooperative information is plotted as a function of the network size (a: $\sigma = 1.0$ and b: $\sigma = 3.0$) and the cache size (c: $\sigma = 1.0$ and d: $\sigma = 3.0$). The cache size for (a and b) is set to 10 items per node and the network size of (b and c) is set to 256 peers.

cooperative information decreases. In all simulations, the lookup requests that use cooperative information successfully find the result at their neighbor (i.e. scenario (c) of Figure 2.4 did not occur during the simulation experiments).

2.5.3.3 Temporal behavior of the RTDc caching framework

Another important aspect of the caching framework is the behavior of cache changes over time. In Figure 2.14 the number of local cache changes over the last 10 lookup requests are shown as a function of the number of lookup requests initiated. For each simulation run 10 nodes are randomly chosen that record the moments their cache changes, finally, the averages are taken over all nodes together (over ten independent simulation runs). The same simulation setup is used as for Figure 2.12 (i.e. $N = 256$, $M = 50 \times N$, $\alpha = 0.6$), with locality variance σ 1.0 and 3.0, and a cache size of 10 items for each node.

Figure 2.14 shows that the average number of cache changes for the first 10 lookup requests is higher when σ is higher. During initialization, the chance that the first 10 request are distinct objects is larger when the locality in requests is more uniformly distributed (i.e. a higher value for σ). Therefore when caches start empty, the average number of cache changes during the startup phase is on average larger and more requests are required to get a (more or less) stabilized cache change rate.

When reaching steady state, cache changes regularly take place, with a higher average number of cache changes for a smaller value of σ and a larger cache size.

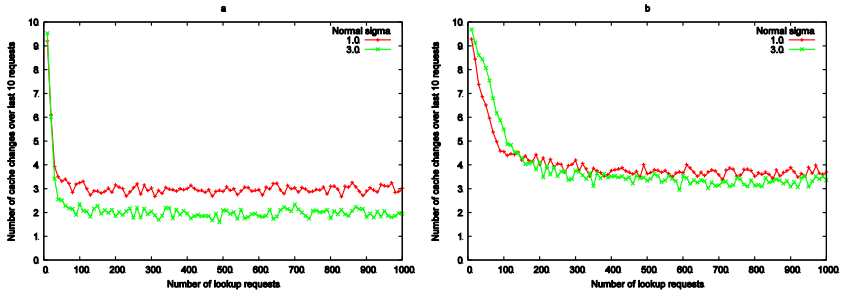

Figure 2.14: Average number of cache changes (over the last 10 requests) in relation to the number of lookup requests initiated locally, the cache size is set in (a) at 10 entries and in (b) at 50 entries.

When σ is smaller the probability that neighbors request items that are popular for a node is smaller (i.e. the distribution of requests gets more localized), and therefore neighbor's caches can be used less often which requires nodes themselves to decide whether to cache an item. A larger cache size (Figure 2.14b) indicates that more lookup results are stored locally that have similar importance values. Therefore, the average number of cache changes over the last 10 lookup requests is higher for a larger cache size.

To examine the relative importance of replaced cache entries, Figure 2.15 depicts the normalized fraction of cache removals for each personal content item stored in the PCSS. On the x-axis, the personal content items are ranked according to their afterwards calculated rank number (i.e. rank 0 is the locally most important object). The same simulation results are used as for Figure 2.14 (i.e. $N = 256$, $M = 50 \times N$ and $\alpha = 0.6$).

The optimal solution is to cache the most locally important items at all times (i.e. with highest values for $I_{n,a}$). By measuring the frequency that a specific item is located in the cache, the importance of the item for that node can be established (automatically taking cached items of neighbors into account). Figure 2.15 depicts, as expected, that the locally most important personal content items are removed less

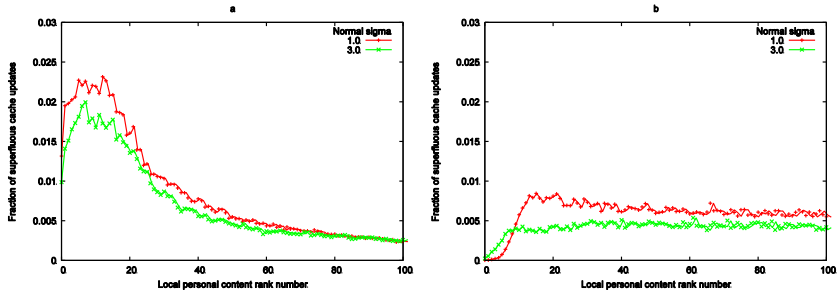

Figure 2.15: Normalized fraction of cache removals in relation to the local personal content rank number. In (a) the fraction of cache removals is shown for a cache size of 10 items and in (b) the cache size is set to 50 entries.

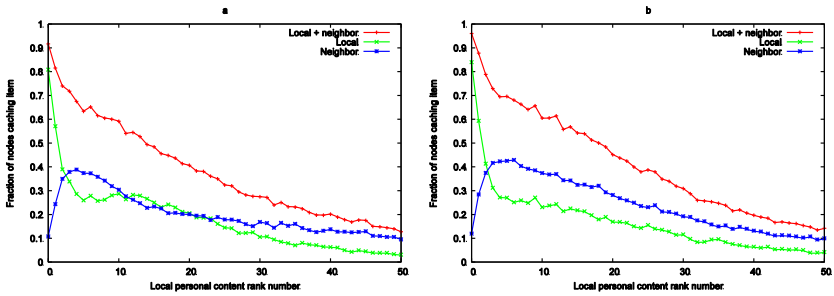

Figure 2.16: Relation between the average fraction of nodes (and their neighbors) caching an object and the objects, ordered by their local rank (i.e. the smaller the rank number, the more popular the object is on a node locally). Part (a) of the figure shows the simulation results for locality variance σ set to 1.0 and (b) the results for locality variance σ set to 3.0.

often from the cache than the objects that are just slightly less popular. The reason that the fraction of cache removals decreases when the local popularity rank decreases further, is that those less important items are sometimes accidentally cached and are removed almost instantly. The chance that a less important object is requested decreases according to the personal content rank number and therefore the number of cache removals decreases. When the lookup pattern is more location dependent, the fraction of cache removals is more concentrated on the locally important personal content items, since the probability that a less important item is requested is lower (i.e. a lower value of locality variance σ indicates a higher fraction of lookup requests of locally more important items). Figure 2.15b shows that when the cache size increases, the most important items are not removed at all once they are stored locally in a cache. Additionally, cache removals are also more evenly distributed over all items, since there is more cache space to store (more or less) equally important lookup results. Most of the cache changes involve two lookup results having a roughly equal importance value.

2.5.3.4 Fraction of nodes caching locally popular personal content items

The fraction of nodes caching a specific item in its final cache state is depicted in Figure 2.16, the location variance σ is set to 1.0 (a) and σ is set to 3.0 (b). For each node the local theoretical importance rank is calculated by multiplying $P_{Zipf\text{-}like}(x)$ with $P_{Normal}(x)$, where x is an item stored in the DHT (i.e. the popularity rank does not take neighbor values into account). For each item the availability of that item at one (or both) of the neighbors is also measured, when the specific item is not stored in the local cache. The same simulation parameters are used as for Figure 2.14 (i.e. $N = 256$, $M = 50 \times N$ and $\alpha = 0.6$) with a cache size of 10 entries at each node.

Figure 2.16 shows that on average more than 80% of the nodes are storing the locally most popular object into their cache. More than 20% of the nodes store on average their top 10 locally most popular items. In the case that a node does not store a top 10 item in its cache, chances are relatively high that one (or both) of the neighbors is caching that particular object. When the locality of lookup requests decreases (i.e. the value of locality variance σ increases), the chance increases that a

neighbor caches a specific item that is not available from the local cache. For the situations depicted in Figure 2.16, more than 60% of all nodes is able to obtain their top 10 local popular items within one (overlay) hop.[5]

2.6 Conclusion and future work

In order to successfully deploy a *Personal Content Storage Service* (PCSS), it has to provide storage space to end-users *transparently*, with *small access times*, and available at *any place* and at *any time*. One of the main features of a PCSS is the ability to search through the dataset of personal files. To optimize searching times in a PCSS, we introduced a caching solution on a *Distributed Hash Table* (DHT). The scalability of a DHT is increased by using the *cooperative Requests Times Distance* (RTDc) caching algorithm.

The RTDc caching framework is compared to the state-of-the-art proactive replication framework Beehive. When the lookup request distribution over the nodes (participating in the DHT) is uniform, the analytical model of Beehive provides a better performance increase compared to our RTDc caching solution. However, the analytical model of Beehive has a perfect centralized view on the content popularity beforehand and therefore no performance is lost by making small mistakes when estimating the popularity parameters. Furthermore, it is highly conceivable that lookup requests are localized (i.e. popularity of objects is different for each node). Unlike the RTDc caching framework, Beehive has no mechanism to take advantage of the locality pattern. When locality exists in the request distribution of lookup request, the RTDc caching algorithm outperforms Beehive quickly. Besides the comparison between the RTDc caching algorithm and Beehive, this chapter also presents a more detailed evaluation of RTDc's inner working. We show that the message overhead caused by the update protocol to enable cooperative caching is acceptable, since the performance increase (i.e. reduction of the average number of hops needed for a lookup request) is higher than the cost the update protocol introduces. Besides that, the simulation results show that more than 20% of the nodes store on average their top 10 locally most popular items. When a node does not store a top 10 item in its cache, the chances are relatively high that one (or both) of the neighbors is caching that particular item.

Although the proposed solution optimizes the scalable lookup in a DHT, it can only be used for lookup when the exact name of the *key* is known (this is the case for e.g. Domain Name System queries). This deterministic search property introduces limitations on the suitability of using a DHT for a PCSS. However, the performance of any existing DHT-based framework offering multiple keyword and range queries can already be increased by the proposed framework, since those frameworks still (have to) use the basic key-based routing mechanism of the DHT. The advantage of our caching strategy is that it extends the basic features of a DHT and increases the performance significantly. Nevertheless, we plan for further research to focus on optimizing DHTs by enabling multiple keywords and range query searches, since currently no solution exists that fulfills all needs for a PCSS. An issue not addressed

[5] Figure A.7 in Appendix A shows the fraction of nodes sharing duplicates between neighbors. A significant reduction is observed for fraction of nodes sharing duplicates between neighbors when using our cooperative caching scheme.

in this chapter is that by reducing the time it takes to obtain content locations does not imply that the actual content itself can be accessed quickly. Therefore, we plan to investigate caching/replication algorithms for personal content itself, in order to allow fast access of personal content by using a PCSS.

References

[1] N. Sluijs, T. Wauters, B. De Vleeschauwer, F. De Turck, B. Dhoedt, and P. Demeester, Caching Strategy for Scalable Lookup of Personal Content, *Proceedings of the 2009 First International Conference on Advances in P2P Systems (AP2PS2009),* 2009, pp. 19-26.

[2] V. Ramasubramanian and E.G. Sirer, Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays, *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation (NSDI2004),* 2004, pp. 1-14.

[3] B. Callaghan, B. Pawlowski, and P. Staubach, NFS Version 3 Protocol Specification: http://tools.ietf.org/pdf/rfc1813.pdf, 1995.

[4] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West, Scale and performance in a distributed file system, ACM Transactions on Computer Systems, Vol: 6, no: 1, 1988, pp. 51-81.

[5] P.J. Braam, The Coda Distributed File System, Linux Journal, Vol: 1998, no: 50, 1998, pp. 46-51.

[6] P. Schwan, Lustre: Building a File System for 1,000-node Clusters, *Proceedings of the 2003 Linux Symposium,* 2003, pp. 380-386.

[7] F. Schmuck and R. Haskin, GPFS: A shared-disk file system for large computing clusters, *Proceedings of the Conference on File and Storage Technologies (FAST2002),* 2002, pp. 231-244.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung, The Google File System, *Proceedings of the 19th ACM Symposium on Operating Systems Principles,* 2003, pp. 29-43.

[9] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, OceanStore: an architecture for global-scale persistent storage, ACM SIGPLAN Notices, Vol: 35, no: 11, 2000, pp. 190-201.

[10] W.J. Bolosky, J.R. Douceur, and J. Howell, The Farsite project: a retrospective, ACM SIGOPS Operating Systems Review, Vol: 41, no: 2, 2007, pp. 17-26.

[11] Y. Saito and C. Karamanolis, Pangaea: a symbiotic wide-area file system, *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop,* 2002, pp. 231-234.

[12] C. Wang, L. Xiao, Y.H. Liu, and P. Zheng, DiCAS: an efficient distributed caching mechanism for P2P systems, IEEE Transactions on Parallel and Distributed Systems, Vol: 17, no: 10, 2006, pp. 1097-1109.

[13] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for Internet applications, IEEE-ACM Transactions on Networking, Vol: 11, no: 1, 2003, pp. 17-32.

[14] A.I.T. Rowstron and P. Druschel, Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems, Lecture Notes in Computer Science: Middleware 2001, Vol: 2218/2001, 2001, pp. 329-350.

[15] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz, Tapestry: a resilient global-scale overlay for service deployment, IEEE Journal on Selected Areas in Communications, Vol: 22, no: 1, 2004, pp. 41-53.

[16] A.I.T. Rowstron and P. Druschel, Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility, *Proceedings of the 18th ACM Symposium on Operating Systems Principles,* 2001, pp. 188-201.

[17] F. Duarte, F. Benevenuto, V. Almeida, and J. Almeida, Geographical Characterization of YouTube: a Latin American View, *Latin American Web Conference (LA-WEB 2007),* 2007, pp. 13-21.

[18] B. Bhattacharjee, S. Chawathe, V. Gopalakrishnan, P. Keleher, and B. Silaghi, Efficient peer-to-peer searches using result-caching, Lecture Notes in Computer Science: Peer-to-Peer Systems II, Vol: 2735/2003, 2003, pp. 225-236.

[19] J. Liu and J. Xu, Proxy caching for media streaming over the Internet, IEEE Communications Magazine, Vol: 42, no: 8, 2004, pp. 88-94.

[20] J. Kangasharju, J. Roberts, and K.W. Ross, Object replication strategies in content distribution networks, Computer Communications, Vol: 25, no: 4, 2002, pp. 376-383.

[21] T. Wauters, J. Coppens, B. Dhoedt, and P. Demeester, Load balancing through efficient distributed content placement, *Next Generation Internet Networks,* 2005, pp. 99-105.

[22] Y. Chae, K. Guo, M.M. Buddhikot, S. Suri, and E.W. Zegura, Silo, rainbow, and caching token: schemes for scalable, fault tolerant stream caching, IEEE Journal on Selected Areas in Communications, Vol: 20, no: 7, 2002, pp. 1328-1344.

[23] K. Lillis and E. Pitoura, Cooperative XPath caching, *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD2008),* 2008, pp. 327-338.

[24] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, Web caching and Zipf-like distributions: evidence and implications, *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM1999),* 1999, pp. 126-134.

[25] P. Backx, T. Wauters, B. Dhoedt, and P. Demeester, A comparison of peer-to-peer architectures, *Conference Proceedings of Eurescom 2002 Powerful Networks for Profitable Services,* 2002, pp. 1-8.

[26] W. Wu, Y. Chen, X. Zhang, X. Shi, L. Cong, B. Deng, and X. Li, LDHT: locality-aware Distributed Hash Tables, *International Conference on Information Networking (ICOIN2008),* 2008, pp. 1-5.

[27] M. Castro, P. Druschel, Y. Hu, and A. Rowstron, Topology-aware routing in structured peer-to-peer overlay networks, Lecture Notes in Computer Science: Future Directions in Distributed Computing*,* Vol: 2584/2003, 2003, pp. 103-107.

[28] J. Pujol Ahulló, P. García López, M. Sànchez Artigas, and M. Arrufat-Arias, An extensible simulation tool for overlay networks and services, *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC2009),* 2009, pp. 2072-2076.

3

Caching strategies for personal content storage grids

N. Sluijs, K. Vlaeminck, T. Wauters, B. Dhoedt, F. De Turck and P. Demeester.
Published in the proceedings of Parallel and Distributed Processing Techniques and Applications (PDPTA2007).

Although the solutions addressed in the previous chapter increases the lookup performance of personal content items, no guarantees are made that the content can be accessed quickly. To reduce access times for retrieving personal content files, this chapter provides our work on multi-level caching. The concepts Personal Content Storage Grids and Personal Content Storage Service are similar and are used in the manuscript interchangeably. The proposed framework allows storing frequently accessed files closer to (their) users, which relieves significant parts of the network and decreases delays when accessing the personal content items. The simulation results in this chapter confirm to analytical calculations for both the dimensioning of the cache sizes and the rate the personal content items are distributed in the caching framework.

3.1 Introduction

Using the Internet, one is able to communicate and share information with others. One of the latest trends is to share your *personal content* with other people. Personal content typically consists of text documents, digital photos, music files, personal movies, etcetera. For instance you can share your personal movies or pictures on central-server architectures like YouTube and Flickr.

However, systems that provide the possibility to store personal content for users still have limitations, namely: *scalability* in the number of users and files, and the

delay caused by central-server architectures. Ultimately users want to have space available to store their personal content, where *access properties* of the online storage is the same as using your own hard disk. The main benefit of such a system is that one can access content fast at *any time* and from *anywhere*. Another advantage of such a system is that users can be relieved from the burden of making backups of their precious files.

In order to overcome the limitations caused by the infrastructure one can use the technology of *Grid computing*. Grid computing offers *computational* and *storage resource*s in a transparent way to users. Transparency means that the exact geographical locations of the physical resources are made abstract for users [1]. In this way one tries to increase the utilization of underused resources, in order to enhance the efficiency of a system as a whole.

A Grid that provides the possibility to store personal files is called a *Personal Content Storage* (PCS) *Grid*. Although a lot of research has already been done into Grid technology, there has not been done a lot in dimensioning cache sizes for a Grid that stores personal content. This is due to the fact that Grids, at the time of writing, are mostly used to solve large and computationally complex problems. Most of the research that tries to improve Grid technology tries to increase the efficiency of the utilization of the computational resources, thereby realizing huge savings on execution times of computationally intensive jobs. However, savings can also be obtained by increasing the efficiency of data transfers. Grids that are optimized to transfer data efficiently for computational intensive jobs are called *data Grids*.

A PCS Grid differs from a data Grid, in the sense that the latter is designed to store a set of relatively large data files, which will typically be accessed by a few hundred to a few thousand researchers. In contrast, a PCS Grid will store a large set of relatively small data files and will typically be accessed by thousands to millions of users.

When designing such a PCS Grid, an important question that needs answering is where files are cached in the Grid, in order to meet the user requirements. With data caching in the Grid, frequently accessed files can be brought closer to the user(s) that are requesting that file often. In this way a big part of the Grid can be relieved and the quality of the service of the Grid will remain, even when the number of users and files grows.

At first glance, such a caching strategy seems very similar to caching strategies in *Content Distribution Networks* (CDN). In a CDN streaming content, which is very sensitive to jitter and packet loss, is replicated to so-called surrogate servers at the edge of the network in order to tackle the performance issues of the classical client-server-approach [2]. However, CDNs are designed to distribute a limited amount of very popular content, while a PCS Grid will store a huge amount of relatively unpopular content. For such a PCS Grid, where each user adds his/her data, storage requirements are more important. Furthermore, guaranteeing low latency and high bandwidth in an environment where end users each access different files simultaneously, requires data to be cached even closer to the end user.

Nowadays there exists many distributed file systems, ranging from client-server systems (e.g. NFS [3], AFS [4] and Coda [5]) over cluster file systems (e.g. Lustre [6], GPFS [7] and the Google File System [8]) to global scale peer-to-peer file systems (e.g. OceanStore [9], FARSITE [10] and Pangaea [11]). None of the distributed file systems enumerated above, were designed for large-scale deployment in an access and aggregation network environment. However,

OceanStore, for which a prototype (Pond [12]) is being developed, seems a good candidate for this purpose. The OceanStore's core system is composed of a multitude of highly connected pools, among which data is allowed to flow freely [9]. A pool could for instance be associated with an access and aggregation network. Most of the time, data will be accessed from within the pool, but when a user is traveling, his data is still accessible. Pangaea [11], with its pervasive replication mechanism that replicates data based on user activity, also seems a good candidate. Data that is only accessed from within the access and aggregation network will be kept locally. Users on the move will trigger replication of their data in other access and aggregation networks.

We describe a caching strategy and an evaluation of the results in this chapter. A description of the caching strategy and the test scenario that we use is provided in section 1.2. The measurements that we did with the discrete event simulator are described in section 1.3. Finally, we provide a discussion and future work in section 1.4.

3.2 Personal content management

As stated in the introduction, this section presents a test scenario for personal content storage. Figure 3.1 represents a typical (Digital Subscriber Line – DSL) access network with a tree topology, having split s and depth d. Users at the leaf nodes are connected to the level one caches, the server is located at level d. We assume that sufficient capacity is available on the links.

In our simulations, users make their personal files available in the network by uploading them to the central server. The uploaded file of a user is cached at the caches on the path to the central server. In total, N files with equal size are uploaded, on average once every A seconds. The number of uploads is a lot smaller than the number of downloads. The popularity of each file is equal at the time of upload, but decreases *exponentially* afterwards. The popularity distribution of file i at time t, where λ_0 represents the initial request rate, τ is a time constant and $T_{i,0}$ determines

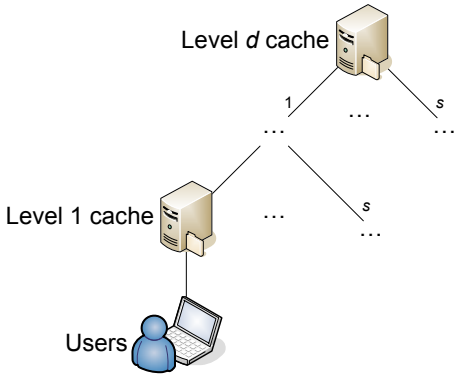

Figure 3.1: Access network with a tree topology, having split s and depth d. Users are situated at the leaf nodes and connected to a level one cache. The server cache is located at level d. On the links sufficient capacity is available.

the upload time stamp of file *i*, is given by (1):

$$\lambda_i(t) = \lambda_0 \cdot e^{\frac{-(t-T_{i,0})}{\tau}}, \text{ where } t > T_{i,0} \quad (1)$$

The function in equation (1) describes an exponentially decreasing popularity for files, which implies that the longer a file is in the system, the less attractive it will be for a user to request it. When a user downloads a file for the first time, each intermediate cache stores that file locally and serves consecutive requests for that file. When a cache is full, older files are deleted according to a *Least Recently Used* (LRU) policy.

Although previous studies on proxy caching techniques [13] or distributed replica placement strategies for CDNs [14-16] show that greedy algorithms that take distance metrics and content popularity into account perform better than more straightforward heuristics, such as LRU or LFU (Least Frequently Used). We use the LRU algorithm to be able to compare our analytical solution with the simulation results.

3.2.1 Test scenario

Before we present our analytical and simulation results, we have to define the parameters that we use in our test scenario. For the parameters depth *d* and split *s*, we take the value four. This implies that we have 85 cache servers in the topology and in total 64 users, where each user represents the aggregation of an access network. Users are connected to a single level one cache in the network.

Figure 3.2 shows an example of the popularity distribution for each file, with λ_0 = 0.01/s and τ = 100,000 s. The area below the function shown in Figure 3.2 represents the total number of file requests that are made for each file. This means

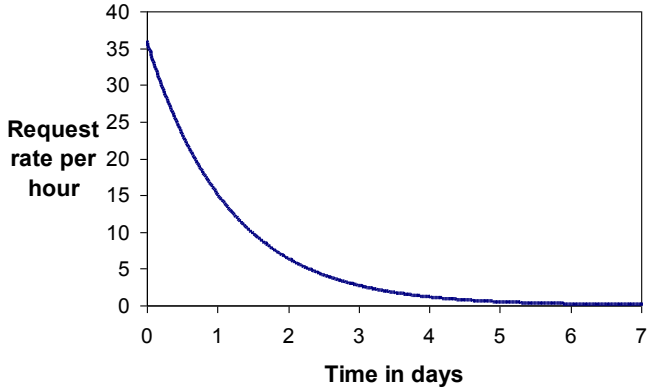

Figure 3.2: Popularity distribution $\lambda_i(t)$ for a file *i*, with λ_0 = 0.01/s and τ = 100,000 s, assumed that the file is uploaded at time 0. The popularity distribution is represented as the request rate per hour against the time in days.

that a total number of $\int_{T_{i,0}}^{\infty} \lambda_i(t)\,dt = \lambda_0 \cdot \tau = 1000$ requests (downloads) are made per file. The number of files N is set to 1,024 and each user has an equal probability to download a file, implying that users have no preference for a certain file. Since we have stated that we assume there is sufficient capacity available on the links, the file size is neglected. Furthermore, we assume that the cache at level d has enough capacity to store all files that will be uploaded. The last parameter that we need to define is the inter-arrival time between uploads A, we assume that every hour a new file is uploaded by a random user.[6]

3.2.2 Storage dimensioning

First, we present an analytical solution for the content placement that determines the storage capacity on each level of the network, so that the total *cache serve ratio* on each level is equal. *The cache serve ratio is the ratio between the number of files served by a cache layer and the total number of requested files.* An equal cache serve ratio implies that load is balanced for each cache level. Afterwards, these results are compared to those of the discrete event simulator, using the LRU caching algorithm.

3.2.2.1 Analytical model

When file i becomes available in the network at time $T_{i,0}$, it should be located at each of the caches on level one, closest to the end users, so that the delay and transport cost are minimized.

As its popularity decreases, a file will be relocated to all caches on level two after $T_{i,0} + t_1$ seconds, and so on, until the file is stored in the server at the top after $T_{i,0} + t_{d\text{-}1}$ seconds. To achieve an equal total cache serve ratio for this file on each level in the tree, all t_l (l = 1, 2, …, d-1) have to be calculated so that the total number of requests made for that file in the intervals $[T_{i,0} + t_0, T_{i,0} + t_1]$, …, $[T_{i,0} + t_{d\text{-}1}, \infty]$ is equal, or in other words:

$$\int_{T_{i,0}}^{t_l} \lambda_i(t)\,dt = \frac{l \cdot \lambda_0 \cdot \tau}{d}, \text{ or } t_l = -\tau \cdot \ln(1 - \frac{l}{d}) + T_{i,0} \qquad (2)$$

When the same procedure is used for all files, each level in the tree serves an equal total number of requests. As we assume that in non-equilibrium steady state new files enter the system at a (nearly) constant rate, the cache serve rate per level is always equally distributed.

[6] We neglect in this model different file sizes, however, to perform a more detailed study an advanced cache replacement policy should be designed that reacts both on popularity and file sizes.

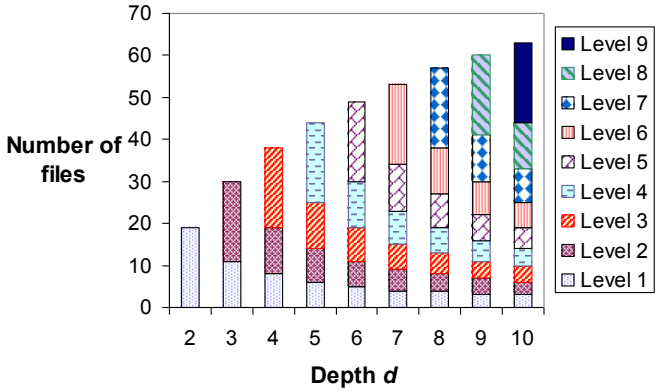

Figure 3.3: Cache size on each level expressed in number of files, for different tree depths d, to get an equally distributed cache serve rate per cache level. We assume that the inter-arrival time of new files in the system is 3,600 seconds.

3.2.2.2 Analytical example

When 1,024 files are available, each with the request rate shown in Figure 3.2, on a tree network with depth $d = 4$ and split $s = 4$, we find that $t_1 = 8.0$ hours, $t_2 = 19.3$ hours and $t_3 = 38.5$ hours. If a constant entry rate of one file per hour is assumed, this means that each cache on level one has to store the eight most recent (i.e. most popular) files, each cache on level two the next eleven most popular files, each cache on level three the next nineteen files and the central server the least popular already available files.

Doubling the entry rate doubles the number of files on each level, the split has no influence. The solution for different values of the depth d is shown in Figure 3.3.

Since the cache server at level d is assumed to have enough capacity available to store all files, only the cache sizes at level one till level d-1 have to be calculated. Figure 3.3 shows that increasing the number of cache levels results in a decrease of the needed cache size at a cache level, in order to have an equally distributed cache serve ratio. However, the sum of the files to be stored over all cache levels increases when parameter depth d increases

3.2.3 Content distribution rate

We know, however, that in a more realistic situation, where an LRU caching algorithm is used instead of an optimal dynamic replacement over all caches of the appropriate level, the location of the files is very suboptimal. In this section, we study the time it takes to store one file on as many level one caches as possible, through individual downloads. In section *'1.3 Simulation and evaluation'* we compare these results to those of the discrete event simulator, using the LRU caching algorithm.

3.2.3.1 Analytical model

At random, each of the users at the leaf nodes sends one of the M ($= \lambda_0 \cdot \tau$) requests for a file i to one of the J ($= s^{d-1}$) caches located at the lowest level in the tree. We look for the probability $P[k]$ that k of the J caches store the requested file, after request m ($m = 1, ..., M$). In the beginning, $P[0] = 1$, $P[k \neq 0] = 0$. After one request ($m = 1$), $P[1] = 1$, $P[k \neq 1] = 0$. The probability that the first k caches store file i, and the other $J - k$ cache do not store file i is given by $P[k]$.

Identify S_j ($j = 1, ..., k$) as the set of possible ways to distribute all m requests over k caches so that cache j remains empty. All sets S_j can be combined into intersections of p subsets, each with cardinality $(k - p)^m$ to distribute m requests over $k - p$ caches, in C_k^p different ways. Following the principle of inclusion and exclusion, the number of possible distributions with at least one cache where a file i is not stored is represented by equation (3):

$$\begin{aligned} \#(S_1 \cup \ldots \cup S_k) &= \sum(\#S_j) - \sum(\#(S_j \cap S_h)) + \ldots \\ &= C_k^1 (k-1)^m - C_k^2 (k-2)^m + \ldots \end{aligned} \tag{3}$$

The number of possible distributions where none of the first k caches is empty is then $k^m - C_k^1(k-1)^m + C_k^2(k-2)^m - \ldots$

In total, J^m distributions (all with an equal probability) are possible, so that the general probability distribution of the number of caches at level one storing a file i after m downloads becomes:

$$P[k] = \frac{C_J^k \cdot (k^m - C_k^1 (k-1)^m + C_k^2 (k-2)^m - \ldots)}{J^m} \tag{4}$$

In the next section we use equation (4) in an analytical example.

3.2.3.2 Numerical example

For the same parameter values as described in '*1.2.1 Test scenario*', 64 level one caches are present and file i is requested a thousand times. A plot (see Figure 3.6) of the probability distribution of the number of level one caches storing file i after one hundred downloads for this example is given in *'1.3.2 Content distribution rate'*. The analytical result is that after a hundred downloads of a file i on average 51 caches store file i. For the development of the number of filled caches with file i against the number of downloads we refer to Figure 3.7 in *'1.3.2 Content distribution rate'*. We notice that the optimal situation (i.e. all caches store the particular file i) in Figure 3.7 is only (almost) reached after two hundred requests and not immediately, as we presumed in the analytical model described in section *'1.2.2 Storage dimensioning'*.

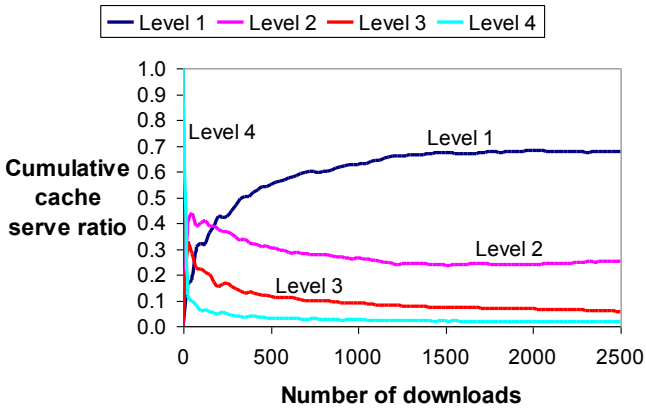

Figure 3.4: Convergence of the cumulative cache serve ratios for each cache level against the total of number of downloads during the simulation. The number of downloads depicted in this figure, is limited to 2500.

3.3 Simulation and evaluation

Besides solving the problem analytically, we use a *discrete event simulator* to approximate the statistics. In [17] a number of data Grid simulators are described, like: Bricks, SimGrid, GridSim, GangSim and OptorSim. We use the simulator OptorSim [18], since it is an event driven simulator and was originally designed to explore effects of dynamic data replication in the European DataGrid (EDG) project [19].

We use the same parameter values as described in *'1.2.1 Test scenario'*; this means that there are 64 level one caches, 1,024 different files and each file is upload once and downloaded a thousand times. For the simulation we use the calculated optimal values for cache sizes at each level; the level one caches have a capacity to store the eight most popular files, the caches located at level two can store the next eleven most popular files and the level three caches are able to store the next nineteen most popular files. Since the cache at level four should have sufficient capacity to store all files, this cache can store 1,024 files. In the next two subsections we show that the analytically obtained results and the results obtained with the simulator are similar.

3.3.1 Storage dimensioning

Since we used the analytical calculated cache size in our simulations, we should get an approximately constant cache serve ratio for each of the cache layers in *non-equilibrium steady state*.[7] In Figure 3.4 and Figure 3.5 the convergence of the cache

[7] Since no time variance exists in the relative popularity between files, the cache serve ratio converges since files are cached according to popularity.

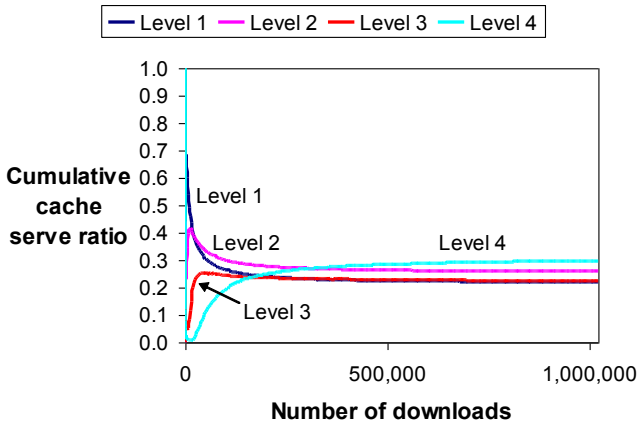

Figure 3.5: Convergence of the cumulative cache serve ratios for each cache level against the total of number of downloads during the simulation.

serve ratios for each cache level in relation to the number of downloads of all files, is presented. Figure 3.4 depicts the first 2500 downloads and Figure 3.5 shows the convergence over all downloads.

In the first 2500 downloads, you see that the cache serve ratio of cache level four starts at 1.0 and the other cache levels begin at 0.0, after the first download. In the simulation a user does an upload of the file to the cache at level four (i.e. the central server), and on every cache on the path from the user to the cache at level four the file is cached. When a user downloads the file for the first time in this simulation, the closest copy of the file was located at the cache in level four. This explains why the cache serve ratio for level four is 1.0 after one download.

The further developments of the cumulative cache serve ratios in Figure 3.4, is that the caches at level one mainly serve the users; this agrees with the observation in Figure 3.7, Figure 3.7 shows that after approximately two hundred downloads all caches at level one store file *i* and thus serve the requests to file *i*.

The reason why the cache serve ratio for cache layer two is higher than the cache serve ratio for cache level three (and four), is that when a user downloads a file for the first time it uses the closest replica of a file in the system. If one of the direct neighbors (i.e. users that share the same cache at level two) of the user already downloaded the file, the file is available at cache level two and cache level two gets a cache hit. The same explanation can be given for the difference between the cache serve ratios of cache level three and four.

As mentioned above, Figure 3.5 presents the convergence of the cache serve ratios for all downloads.

When the total number of downloads advances, the cache serve ratio of cache level one decreases, since more new files enter the system. The relative number of older files (files that are served by cache level two, three or four) increases, but users will still produce some requests to these files. The same is valid for the caches at level two and three. For these lower level caches, the drop in cache serve ratio happens after more downloads, since these caches store the next most popular files.

Cache level	*Cache serve ratio*
Level 1	0.2196
Level 2	0.2584
Level 3	0.2241
Level 4	0.2978

Table 3.1: Cache serve ratios for each cache level at the end of the simulation.

Eventually the caches at level four becomes important when the caches at level three have no space left to store the old files. The requests that users make to these old files will all be served by cache level four, so the cache serve ratio of level four will increase.

The cache serve ratio numbers of the caches at the end of the simulation are summarized in Table 3.1.

According to the analytical example in *'1.2.2 Storage dimensioning'* cache serve ratio should be equal for each cache level. Since we have four cache levels, the cache serve ratio should be 0.25.

The ratios of Table 3.1 more or less correspond to the calculated values. The cache serve ratio of cache level one has the lowest cache serve ratio. This is due to the empty caches, when the simulation starts. It will take some time, after a user uploads the ninth file, before the first file is deleted from cache level one and further requests to the first file are served by cache level two. Since all requests of a user that downloads a file for the first time is handled by a cache level other than cache level one, cache level one misses requests that were assigned to cache level one in the analytical calculations. Cache level four profits from this, which explains why the ratio of this cache level is higher. The cache serve ratios of cache level two and three more closely approximate to the calculated value, despite all caches start empty.

3.3.2 Content distribution rate

In this section we study the time (or number of downloads from file *i*) it takes to store one file on as many level one caches as possible, through individual downloads. According to the analytical calculations after one hundred downloads of file *i* on average 51 caches should store file *i*. This is visualized in Figure 3.6.

Besides the analytical solution, the measured values of the simulation are also depicted in Figure 3.6. From Figure 3.6 we can conclude that the probability distribution obtained with the simulation confirms the analytical probability distribution. The small difference is due to the random number generator in the simulations.

Besides the probability distribution of the filled level one cache with file *i* after one hundred downloads of file *i*, we are also interested in the evolution (i.e. in number of downloads) of the average number of filled level one caches in time. Figure 3.7 provides this information.

Both the analytical solution and the measured average number of level one caches that store a file *i* in relation to the total number of downloads of a file *i* are

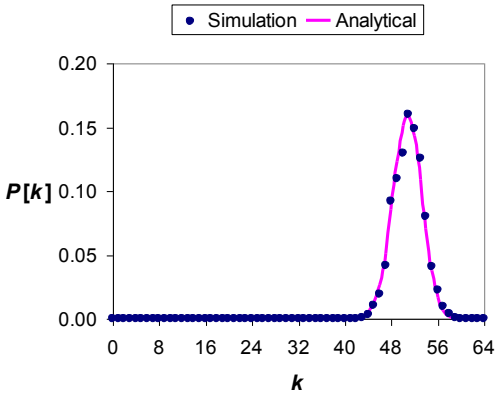

Figure 3.6: Probability distribution of the number of level one caches storing file *i* after one hundred downloads. The line depicts the analytical solution of equation (4); $m = 100$ and $J = 64$. The dots represent the measured values, obtained from the simulation.

shown. The upper limit in this example is 64, since there are only 64 level one caches present. We can conclude that the measured approximation fits the analytical solution. The small differences can again be explained by using a random number generator in the simulation.

3.4 Conclusion

To realize a PCS Grid, the Grid should be scalable in the number of users and files, and the delay should be limited. In order to meet these requirements an optimized caching strategy for personal content should be used to increase the efficiency of a

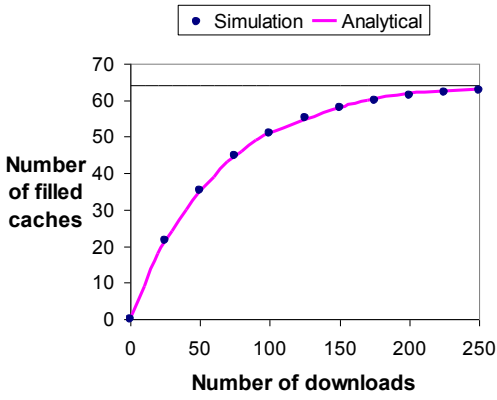

Figure 3.7: Average number of level one caches that store a file *i* in relation to the total number of downloads of a file *i*.

Grid. We show with a basic test scenario on an access network with a tree topology, that the results that we obtain with our simulator confirm analytical calculations.

We have determined the required cache capacities at each level of the tree network based on an analytical model, in order to obtain an equal *cache serve ratio* for each cache level. The simulation shows that the cache serve ratios converge closely to the calculated values.

We are aware that in a more realistic situation, where a *Least Recently Used* (LRU) caching algorithm is used instead of an optimal dynamic replacement over all caches of the appropriate level, the location of the files is suboptimal. This is why we also studied the time it takes to store one file on as many level one caches as possible, through individual downloads. The measurements of the simulation of the probability distribution of the number of caches at level one that store a file that is downloaded one hundred times, approximates the analytical calculation closely.

Now that we have a simulator that is able to closely match analytical calculations, we will use it in future work to investigate properties and topologies for which analytical calculations are too complex. Future work will include studying different caching strategies, different topologies where links have a different and limited bandwidth, a more realistic file size distribution, and users having preferences for some common and their own files. Our study will lead to caching strategies where the user experience of the online storage system will be similar to using a local hard disk.

References

[1] I. Foster, The Grid: a new infrastructure for 21st century science, Grid Computing: Making the Global Infrastructure a Reality, 2003, pp. 51-63.

[2] J. Coppens, T. Wauters, F. De Turck, B. Dhoedt, and P. Demeester, Evaluation of replica placement an retrieval algorithms in self-organizing CDNs, *Proceedings of the IFIP/IEEE International Workshop on Self-Managed Systems & Services (SELFMAN2005),* 2005.

[3] B. Callaghan, B. Pawlowski, and P. Staubach, NFS Version 3 Protocol Specification: http://tools.ietf.org/pdf/rfc1813.pdf, 1995.

[4] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West, Scale and performance in a distributed file system, ACM Transactions on Computer Systems, Vol: 6, no: 1, 1988, pp. 51-81.

[5] L.B. Mummert, M.R. Ebling, and M. Satyanarayanan, Exploiting weak connectivity for mobile file access, ACM SIGOPS Operating Systems Review, Vol: 29, no: 5, 1995, pp. 143-155.

[6] P. Schwan, Lustre: Building a File System for 1,000-node Clusters, *Proceedings of the 2003 Linux Symposium,* 2003, pp. 380-386.

[7] T. Jones, A. Koniges, and R.K. Yates, Performance of the IBM general parallel file system, *Proceedings of the 14th Parallel and Distributed Processing Symposium (IPDPS2000),* 2000, pp. 673-681.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung, The Google File System, *Proceedings of the 19th ACM Symposium on Operating Systems Principles,* 2003, pp. 29-43.

[9] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, OceanStore: an architecture for global-scale persistent storage, ACM SIGPLAN Notices, Vol: 35, no: 11, 2000, pp. 190-201.

[10] W.J. Bolosky, J.R. Douceur, and J. Howell, The Farsite project: a retrospective, SIGOPS Operating Systems Review, Vol: 41, no: 2, 2007, pp. 17-26.

[11] Y. Saito and C. Karamanolis, Pangaea: a symbiotic wide-area file system, *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop,* 2002, pp. 231-234.

[12] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, Pond: the OceanStore Prototype, *Proceedings of the 2nd Usenix conferences on File And Storage Technologies (FAST),* 2003, pp. 1-14.

[13] J. Liu and J. Xu, Proxy caching for media streaming over the Internet, IEEE Communications Magazine, Vol: 42, no: 8, 2004, pp. 88-94.

[14] J. Kangasharju, J. Roberts, and K.W. Ross, Object replication strategies in content distribution networks, Computer Communications, Vol: 25, no: 4, 2002, pp. 376-383.

[15] K. Magnus, K. Christos, and M. Mallik, A framework for evaluating replica placement algorithms, *Technical Report HPL-2002,* 2002, pp. 1-12.

[16] Q. Lili, V.N. Padmanabhan, and G.M. Voelker, On the placement of Web server replicas, *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM2001),* 2001, pp. 1587-1596.

[17] B. Quetier and F. Cappello, A survey of Grid research tools: simulators, emulators and real life platforms, *17th IMACS World Congress (IMACS2005),* 2005, pp. 1-8.

[18] OptorSim Release 2.1: http://sourceforge.net/projects/optorsim last accessed in Oct 2006.

[19] A.T. Doyle and C. Nicholson, Grid data management: simulations of the LCG 2008, *Computing in High Energy and Nuclear Physics (CHEP06),* 2006, pp. 1-8.

4

Using topology information for quality-aware Peer-to-Peer video streaming networks

N. Sluijs, T. Wauters, C. Develder, F. De Turck, P. Demeester and B. Dhoedt.
Submitted to Computer Networks.

Watching videos via the Internet is becoming a popular activity of users. Therefore, we explore in this chapter next generation P2P video streaming frameworks. The routing of video layers in both the underlay and overlay network is modeled in an Integer Linear Programming (ILP) formulation, and by using an ILP solver we investigate the advantages of an orchestrating engine that manages video exchanges between peers. The objective of current P2P video streaming networks maximizes greedily the local peer's video quality. However, video service providers are generally interested in the minimum number of video layers that can be transported to end-users. We have studied both objective strategies and our simulation results indicate that by orchestrating the video streams a significant increase is obtained of the fraction of destinations that are able to receive more than only the base layer.

4.1 Introduction

Watching videos on websites, e.g. YouTube or Eurovision Sports, is a popular activity of users today. Using the Internet to watch videos is expected to become more prominent, since the new HTML5 standard natively supports videos on websites. Although the video quality and length of the videos that are transported via the Internet have significantly increased over the last couple of years, offering a full

video-on-demand or live broadcasting service is still not viable. Different television broadcasters are already setting up streaming services to watch (live) television programs via their websites (e.g. BBC iPlayer, iWatch and RTL XL), but often the quality of the stream and the total number of viewers is limited. An interesting technology that offers a cost efficient mechanism for distributing live video is the Peer-to-Peer (P2P) overlay network model. In a P2P network the peers form a virtual overlay network, on top of the actual IP (Internet Protocol) network, and all peers act as both suppliers and consumers. In contrast, in traditional client-server networks only servers supply and clients consume, therefore, P2P services are potentially highly scalable and robust.

The open source research project Tribler [1] enables users to find and share (live) video content and is a representative example of a typical P2P (live) streaming framework. Inspired by BitTorrent's [2] file sharing protocol, Tribler has incorporated mechanisms to enable (live) video in a P2P fashion. The main idea is to let peers download a video in small parts and immediately share the downloaded parts with other peers in the network. To support playback while downloading a video, Tribler prioritizes in-order downloading for parts that are close to the current playback position. The traditional peer selection policy (i.e. BitTorrent's tit-for-tat mechanism [2]), makes it difficult to find good sharing partners for pieces at the current playback position. Therefore Tribler introduces a peer selection strategy that

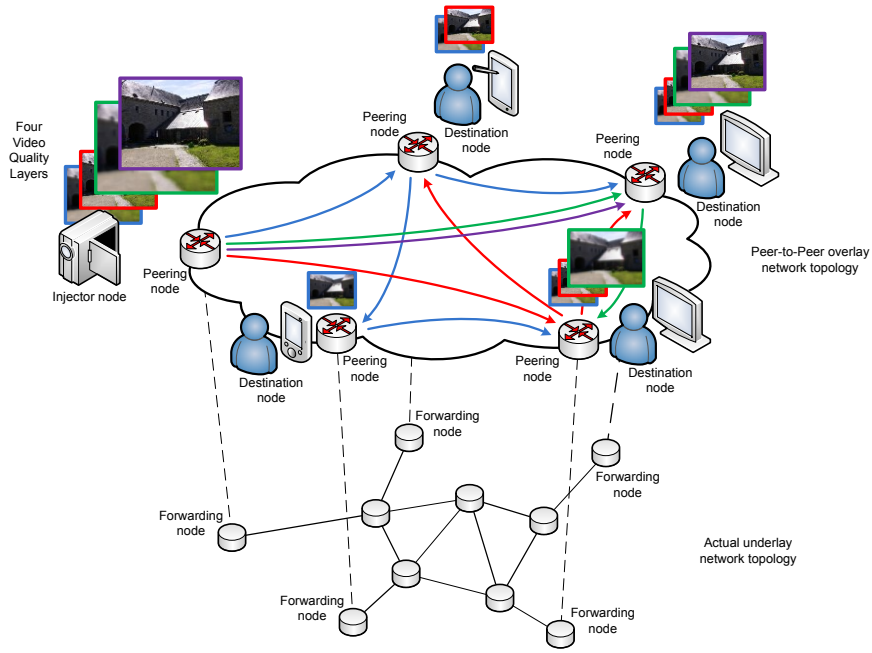

Figure 4.1: Next generation Peer-to-Peer (live) video streaming network uses multi-layer video coding, which allows to start watching a video when only the base layer is downloaded. Additional received layers increase the video quality, and this strategy allows peers to adopt to their output abilities and the network to offer a higher average (or minimal) received video quality to end-users.

chooses peers based on their forwarding capabilities [3]. In this way in-order downloading is not a problem, because peers get a positive incentive to share downloaded parts to peers even if those are not able to give anything in return. Since peers can only receive parts when they are sharing to others, free-riding is still prevented.

However, to adapt to typical heterogeneous circumstances in a P2P network (i.e. various asymmetric link bandwidths, end-devices having different display resolutions, etc.), the next generation P2P systems offering over-the-top (live) video streaming use multi-layer video (such as Scalable Video Coding (SVC) [4]). The resulting P2P network, illustrated in Figure 4.1, consists of the following entities:

- *Injector node*: offers the video stream, separated into distinct video layers (e.g. four).
- *Tracker node*: a (often centralized) bootstrap server, providing all necessary information for peering nodes to start the download process.
- *Peering node*: nodes located at the destinations in the overlay topology and containing peer functionality, allowing them to download the video layers from the injector node or other peering nodes.
- *Forwarding node*: nodes that forward data through the network and are usually located in the core network (i.e. underlay network topology).

A video is encoded into multiple video layers, which allows playing the video when only the base layer is received. Every additional layer that a peering node receives increases the video quality. This allows peering nodes to only download those video layers that they are able to output (e.g. based on screen resolution or stereoscopic rendering abilities). Moreover, the bandwidth requirements of the injector node can be reduced significantly, since at least one stream (containing each distinct video layer) has to be provided in order to enable each device to select the right number of video layers to stream. In this chapter we use the term peering node and peer interchangeably.

An important aspect when using multi-layer video coding in a P2P network is the piece picking and peer selection mechanism. Current strategies try to maximize the local peer's video (i.e. download) quality. However, for a video streaming network to become successful, it is important to optimize the overall received video quality in the network. When peering nodes would collaborate, the average video quality can be increased by the peers that are currently streaming at a higher than average quality. When these peers decide to give up a bit of their high video quality (i.e. drop a few of the top video layers), more bandwidth and (possible) lower layer quality pieces will become available to the rest of the network. Peers streaming at a much lower video quality benefit from this strategy and are provided a chance to increase their streaming quality to acceptable levels. The strategy we propose in this chapter maximizes the minimum received video quality at each destination, by using topology information of the actual network to orchestrate the peer selection and thereby forming a future prove and robust framework for distributing (live) video streams over the Internet.

To optimize the received video quality, we setup an Integer Linear Problem (ILP) formulation that describes the underlay-overlay-routing problem of the video layers in the network. The core network topology in our case study is based on the GÉANT research network [5]. Our use case focuses on efficiently distributing a live video feed from the France site of the European Parliament to a select group of end-users (e.g. journalists, translators, …) located in different countries in Europe. The deployment perspective of our work is provider centric, considering a limited number of end-users and a complete knowledge on the network infrastructure.

This chapter continues in Section 4.2 with an overview of related work. Section 4.3 provides the generic problem formulation and shows correctness of our model in a tree-like network topology. In section 4.4 the use case is used to evaluate our proposed strategies. Conclusions and future work are presented in section 4.5.

4.2 Related work

Several commercial platforms (e.g. Octoshape, RawFlow and RayV) offer streaming video solutions using the Internet. However, these platforms mainly use traditional client-server based network models, inducing large bandwidth and server costs for broadcasters. On the other hand, freeware/open-source applications (such as Alluvium, End System Multicast, PeerCast, PPTV (formerly known as PPLive) and Tribler [1]) provide video streaming solutions using P2P mechanisms. In order to achieve stable streams with decent quality, these systems require a large number of users watching the same video stream and, to our knowledge, do not use location-aware protocols to optimize the load in the P2P network.

Since IP Multicast [6] is only sparsely deployed on the Internet, this technology cannot be used to offer a scalable video streaming service. Therefore, our research aims at advanced P2P technologies since these solutions can be deployed on the Internet without requiring specific hardware changes or deployments.

BitTorrent's tit-for-tat mechanism [2] and Tribler's give-to-get algorithm [3] try to motivate/force peers to contribute to each other in the download process. According to [7-9], unbalanced data exchanges decrease a P2P system's potential performance in terms of bandwidth utilization. Not only unwillingness to contribute by users forms a burden, also inefficiencies in the algorithms used by P2P streaming applications are a challenge. Therefore, incentives that stimulate contributing content are necessary in order to offer a real robust and scalable (live) video streaming framework. Our solution agrees to this and extends them by using a tracker node to orchestrate data transfers. In this way we can provide the optimal strategy for the complete network, reaching the highest possible performance gain.

Multiple recent research studies, like [10-12], use P2P services combined with layered video coding and mainly focus on altering piece picking algorithms and/or neighbor selection mechanisms for advanced buffering strategies. Our aim is to improve these systems by using topology information to orchestrate the peer selection so that the minimum video streaming quality is maximized for each destination.

In [13] a P2P video-on-demand (VoD) strategy to optimally (pre-)fetch video segments is presented, integrating localization and congestion-aware peer selection schemes. Simulation results show that utilizing location information (and preventing congestion) increases the average supported playback rate of a video. However, in

[13] multi-layer video coding is not considered and peers are classified to a fixed set of domains. In our research study we consider multi-layer video coding, offering a whole new set of optimization possibilities. Moreover, we do not restrict a peer to be part of a fixed set of domains, which allows us to calculate the optimal solution.

Modeling video streaming in a P2P network by using an ILP formulation is performed by [14-16]. However, the mathematical formulations in these studies only model overlay network routing and assume the upload and download capacities per peer to be the main bottlenecks. Besides overlay routing our proposed model also takes the underlay network into account, increasing the complexity of the problem significantly and, therefore, allowing more realistic research studies.

Contributions of this chapter can be summarized as:

- Providing a mathematical formulation that is capable to model the underlay-overlay-routing problem of multi-layer video in a P2P network.

- Proposing a piece picking and peer selection strategy for next generation P2P video streaming networks that maximizes the minimum video quality for each destination, which is accomplished by orchestrating the download using a tracker node that has a precise view on both the underlay and overlay network topology.

4.3 Problem formulation

In order to find an optimal solution and to present a precise view on the problem, an ILP (Integer Linear Programming) formulation is given here. First, section 4.3.1 describes the network model, introducing all parameters and variables. Then, section 4.3.2 provides the formulation in terms of the objective function and a set of constraints that specify the relation between the parameters and the variables. Finally, in section 4.3.3 the correctness of our model is shown by comparing the results of solving our formulation on a basic network structure, with an analytical solution.

4.3.1 Model description

The problem can be characterized by the network topology, an injector node, forwarding nodes, peer nodes, destination nodes and a list of video layers.

4.3.1.1 The network

The underlay network is represented by a directed graph G, characterized by a set of nodes V of size $|V|$ and a set of directed edges E of size $|E|$. The graph is presumed to be bi-directional but the edge properties can be asymmetric. Each (direction of) edge e from E is characterized by a constant maximum bandwidth capacity $u_e \geq 0$. I_v and O_v are the sets of respectively ingoing and outgoing links of a node v. To enforce shortest-path routing, Dijkstra's algorithm is used to compute the k-shortest-paths (measured in network hops) between node x and y. Each link e is provided with a constant binary parameter $m_{e,d}$ that denotes whether or not link e can be used to transport data to node d, using one of the k-shortest-paths. The value $m_{e,d}$ is 1 if and

only if link e is one of the k-shortest-paths to destination node d from a peer or injector node.

4.3.1.2 Forwarding and peer nodes

All network nodes that contain no application intelligence act as forwarding nodes. F is the set of all forwarding nodes, $F \subset V$ and has size $|F|$.

Peer nodes are the nodes running the P2P streaming application (i.e. an incoming stream can be sent to multiple destination peers) and are typically located at a user's home, connected via an asymmetric bandwidth connection to the Internet (i.e. rest of the network). P is the set of all peer nodes, $P \subset V$, $P \cap F = \emptyset$ and has size $|P|$.

4.3.1.3 Injector and destination nodes

The original source of the live video stream is provided by the injector node z. Destination nodes request the video stream and are usually connected to a peer node. D is the set of all destination nodes, $D \subset V$ and has a size of $|D|$.

4.3.1.4 Video layers

There is an ordered list of video layers L of size $|L|$, containing each different video layer sorted by increasing layer rank (i.e. layer l_0 is the base layer).[8] Each layer l from L has a constant bandwidth cost per time unit of $c_l \geq 0$. Note that receiving a layer l_{i+1} is useless, without also receiving l_i, where $i \geq 0$.

In order to prioritize situations for fairness where multiple destinations all receive a layer l_i over situations that only a few receive layer $l_{>i}$ and the rest receive $l_{<i}$ a constant value b_i per layer l_i is set to express the benefit of receiving the layer. The values $b_i = |D|^{(|L|-i-1)}$ guarantee the following principle: $b_i > (|D| - 1) \times \sum_{j=i+1}^{|L|-1} b_j$, for $0 \leq i < |L|$. This means that the benefit contribution b_i when a node receives layer l_i is greater than combining the benefits b_j for every other destination for all video layers l_j where $j > i$.

4.3.1.5 Variables

In this subsection e is an edge from set E, d and d' are destination nodes from set D and l is a video layer from set L. The binary variable is:

- $h_{e,d,d',l}$ is 1 iff edge e is used to carry traffic destined for node d, which is (in)directly sent to node d' and has layer l (with $|E| \cdot |D| \cdot |D| \cdot |L|$ h-variables).

Index d represents the destination node of the direct video traffic (i.e. underlay routing) and node d' indicates that another node can benefit from this traffic via P2P routing (i.e. indirect or overlay routing). The h-variable is used to ensure flow

[8] In our model we consider one scalability axis by representing the scalable video stream as one single (totally) ordered set of elements. However, the model can be adjusted to relieve this requirement or embed several scalability axes, the objective function and constraint (14) have to be altered.

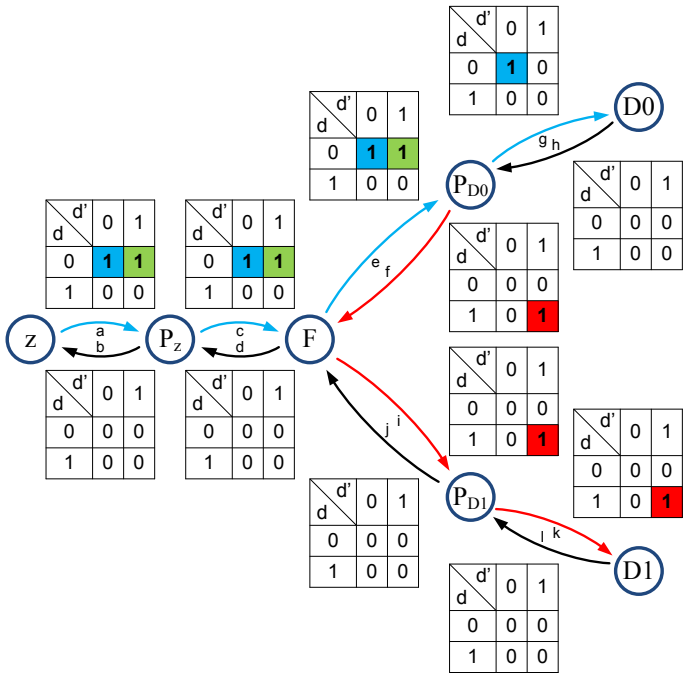

Figure 4.2: Usage scenario of the $h_{e,d,d',l}$ variables in the ILP formulation. Node z inserts a video, consisting of one quality layer, into the network. Since all the network links have a capacity to transport one layer, the direct traffic is send from the injector node z to destination node $D0$. The peering node directly connected to $D0$, duplicates the video stream and also sends it to destination $D1$.

conservation through the network and, where $d = d'$, allows the calculation of the bandwidth carried by the edges.

To illustrate the behavior of variable h, Figure 4.2 shows a scenario of a tree network with injector node z at its root. The video stream in this example consists of one layer and is transported over links that all have a bandwidth capacity of one layer. Two destination nodes receive the video layer and in between there are three peer nodes connected by a forwarding node. To ensure a correct solution, the injector node z sends the layer directly to node $D0$ and indirectly to node $D1$ over link a (i.e. another correct solution is sending the stream directly to node $D1$ and indirectly to node $D0$). Node P_z and F forward this traffic over respectively link c and e. Peer node P_{D0} duplicates the video layer and sends this traffic to destination node $D0$ and $D1$ via, respectively, link g and f. Forwarding node F and peer node P_{D1} forwards the direct traffic to $D1$ via link, respectively, link i and k.

This approach allows to perform both underlay and overlay routing through the network and guaranteeing that the injector node z is the origin of every video layer that a destination node receives. However, the downside of this strategy is the increased complexity of solving the underlay-overlay-routing problem of the video layers and as a consequence limits the size of our network topologies.

4.3.2 Formulation

Now that all symbols and variables are presented, the objective function to optimize can be formulated as follows:

$$Q = maximum\left(\sum_{l_i \in L}\sum_{d \in D}\sum_{e \in I_d} b_i \times h_{e,d,d,l_i}\right) \quad (1)$$

By optimizing objective function Q each node receives a maximum video quality, without requiring any other node in the network to lower its receiving quality. Hereby, the minimum quality that is received by each node is maximized. In order to solve the problem, a set of constraints have to be considered to make sure the relation between all parameters and variables comply with the general network model.

4.3.2.1 Capacity and routing constraints

$$\sum_{l \in L}\sum_{d \in D} h_{e,d,d,l} \times c_l \leq u_e \quad \forall e \in E; \quad (2)$$

$$h_{e,d,d',l} \leq h_{e,d,d,l} \quad \forall e \in E; \forall d, \forall d' \in D; \forall l \in L; \quad (3)$$

$$h_{e,d,d,l} \leq m_{e,d} \quad \forall e \in E; \forall d \in D; \forall l \in L; \quad (4)$$

Constraint (2) restricts the total flow through the edges. This flow may not exceed the capacity of the edge. Constraint (3) imposes that link *e* can only be used in a virtual path to destination *d'* for video layer *l*, when *e* is directly transporting *l* to a destination node *d*. Constraint (4) ensures that edge *e* can only be used to transport (any) layer *l* directly to destination node *d* when *e* is on one of the shortest paths to node *d*.

4.3.2.2 Ingoing and outgoing constraints

$$\sum_{e \in I_v} h_{e,d,d,l} \leq 1 \quad \forall v \in V; \forall d \in D; \forall l \in L; \quad (5)$$

$$\sum_{e \in O_v} h_{e,d,d,l} \leq 1 \quad \forall v \in V; \forall d \in D; \forall l \in L; \quad (6)$$

$$\sum_{e \in I_p} \sum_{d \in D} h_{e,d,d,l} \leq 1 \quad \forall p \in P; \forall l \in L; \tag{7}$$

Constraint (5) guarantees that for every node in the network, there is maximum one incoming edge transporting a specific video layer to a specific destination node. This constraint is necessary to make sure each destination node receives a video layer only once. Constraint (6) imposes that for every node in the network, there is maximum one outgoing edge containing traffic for a specific video layer to a specific destination. Constraint (7) prevents that multiple streams of the same video layer *l* are incoming on a peer node *p*. These constraints reduce the solution space of the ILP formulation.

4.3.2.3 Flow conservation and peer node constraints

$$\sum_{e \in I_f} h_{e,d,d',l} = \sum_{e \in O_f} h_{e,d,d',l} \quad \forall f \in F; \forall d, \forall d' \in D; \forall l \in L; \tag{8}$$

$$\sum_{e \in I_p} \sum_{d \in D} h_{e,d,d',l} = \sum_{e \in O_p} \sum_{d \in D} h_{e,d,d',l} \quad \forall p \in P; \forall d' \in D; \forall l \in L; \tag{9}$$

Constraint (8) guarantees that (direct and indirect) traffic flows through the forwarding nodes in the network (i.e. all nodes except for injector, destination and peer nodes). Constraint (9) ensures that all incoming direct and indirect flows leave the peer node, where indirect flows can be converted into direct traffic by the peer node.

4.3.2.4 Injector and destination node constraints

$$h_{e,d,d',l} = 0 \quad \forall e \in I_z; \forall d, \forall d' \in D; \forall l \in L; \tag{10}$$

$$h_{e,d,d',l} = 0 \quad \forall e \in O_x; \forall x, \forall d, \forall d' \in D; \forall l \in L; \tag{11}$$

$$h_{e,d,d',l} = 0 \quad e \in I_x; \forall x, d' \in D; \forall d \in D \backslash x; \forall l \in L; \tag{12}$$

$$h_{e,d,d',l} = 0 \quad e \in I_d; \forall d \in D; \forall d' \in D \backslash d; \forall l \in L; \tag{13}$$

$$(i+1) \times h_{e,d,d,l_i} \leq \sum_{g \in I_d} \sum_{k \in L} h_{g,d,d,k} \qquad \forall d \in D; \forall e \in I_d; \forall l_i \in L; \tag{14}$$

$$\sum_{e \in I_d} h_{e,d,d,l} = \sum_{e \in O_z} \sum_{x \in D} h_{e,x,d,l} \qquad \forall d \in D; \forall l \in L; \tag{15}$$

Constraint (10) ensures that the injector node z only sends data. Since the injector has all layers available, no other constraints are necessary. Constraints (11), (12) and (13) guarantee that all destination nodes only receive data meant for them, and prevent destinations from creating data. Constraint (14) ensures that when a destination node d receives video layer l_{i+1}, also video layer l_i is received on one of d's direct incoming links. Constraint (15) imposes that when a destination node d receives layer l, there must be (at least) a virtual path starting from the injector node

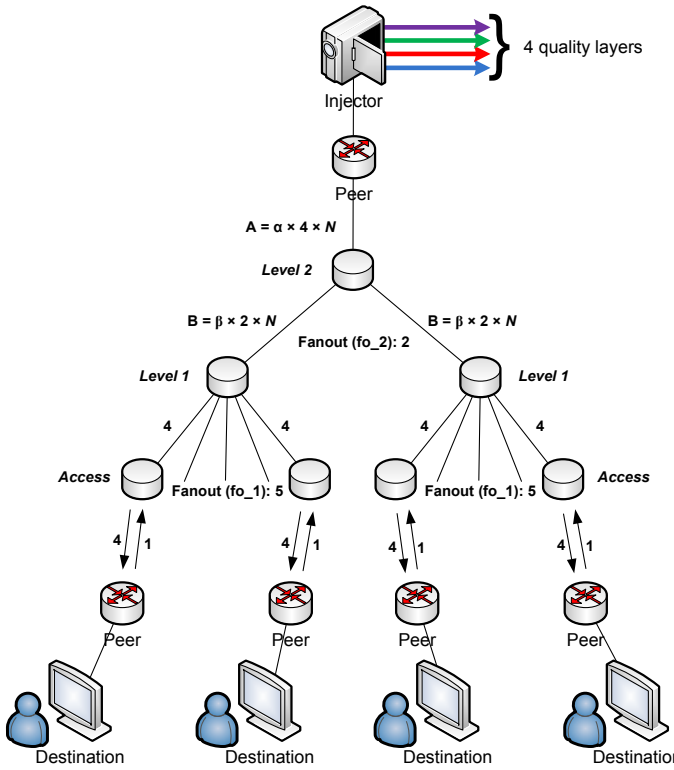

Figure 4.3: A tree network with the injector node at the root providing a video stream consisting of four quality layers. The injector node is directly connected to a peer node, which on its turn is connected to a level 2 forwarding node. The level 2 forwarding node is connected to two level 1 forwarding nodes. Each level 1 forwarding node is connected to five forwarding nodes, that each provides access to one destination's peer node.

z.

4.3.3 Solving our problem on a tree video distribution network

To find the optimal solution, the formulation described above was implemented in an ILP solver [17]. For validation purposes, a video distribution tree is investigated, for which the optimal solution can be derived analytically. Figure 4.3 illustrates the tree network topology with the injector node at the root, which inserts the video stream consisting of four video layers into the network. We assume that all video layers have the same constant transport (i.e. bandwidth) cost $c_l = 1$. The injector node is directly connected to a peer node, which on its turn is connected to a level 2 forwarding node. The bandwidth on this level 2 link equals $\alpha \times 4 \times N$ (the influence of the parameter α is investigated), with N the number of destination nodes. The level 2 forwarding node is connected to two level 1 forwarding nodes, with each connection having a bandwidth equal to $\beta \times 2 \times N$ (the influence of parameter β is investigated). In the scenario that both α and β are set to 1, each destination is able to stream the video at full quality directly from the injector's peer node.

The destination nodes (N=10) are directly connected to a peer node (e.g. a residential gateway), which in turn is connected via an access forwarding node to one of the level 1 forwarding nodes. Each level 1 forwarding node is connected to five access forwarding nodes. The downstream bandwidth on these connections is enough to transport all video layers to each of the destinations. Since the destination's peer node is able to (re)distribute (a part of the) received video layers and is typically located in the access network, the upload bandwidth of these peers is limited (i.e. asymmetrical connection).

The basic network structure of Figure 4.3 allows us to form an analytical formulation, to calculate the average received video layers on a destination node E:

$$A = \alpha \times |L| \times N \tag{16}$$

$$B = \frac{\beta \times |L| \times N}{fo\_2} \tag{17}$$

$$C = \frac{fo\_2 - peer\_l1}{fo\_2} \tag{18}$$

$$C' = \frac{peer\_l1}{fo\_2} \tag{19}$$

$$H = min\left(\frac{A + fo\_1 \times peer\_l1 \times \left(B - min(|L|, A)\right)}{N - fo\_1 \times peer\_l1}, \frac{B}{fo\_1}\right) \quad (20)$$

$$I = \min(|L|, A, B, C \times (H + up \times peer\_a) + C' \times \min(|L|, B)) \quad (21)$$

In (16) - (21) N is the number of destination nodes in the tree network (i.e. $fo\_1 \times fo\_2$), fo_1 and fo_2 are respectively the fan-out on level 1 (i.e. 5) and level 2 (i.e 2), up is the upload bandwidth, $peer\_a$ is the fraction of peer nodes at the destination that are able to distribute received video layers and $peer\_l1$ is the number of level 1 nodes with peer functionality. The average number of received video layers I (equation (21)) is the minimum over four arguments. The first argument of (21) states that the average received number of layers is never larger than the number of distinct video layers. The second and third element (i.e. A defined by (16) and B defined by (17)) state that the average received number of layers is less than the amount of traffic that the link from the injector's peer to the level 2 node or the link from the level 2 to the level 1 node are able to transport. The fourth argument in (21) describes that the highest average quality that can be received is the weighted sum between the part of the tree that has no level 1 peer functionality (i.e. denoted by expression C of (18)) and the fraction that has level 1 peer functionality (i.e. defined by expression C' of (19)). When the level 1 peer exhibits no peer functionality, the average received quality for that sub-tree is not larger than dividing the bandwidth contributed by the injector peer and the level 1 peers, over all destinations that are part of the sub-tree as expressed by equation H in (20). Since the destination's peer node can contribute to this sub-tree, the fraction of peer nodes times the uplink has

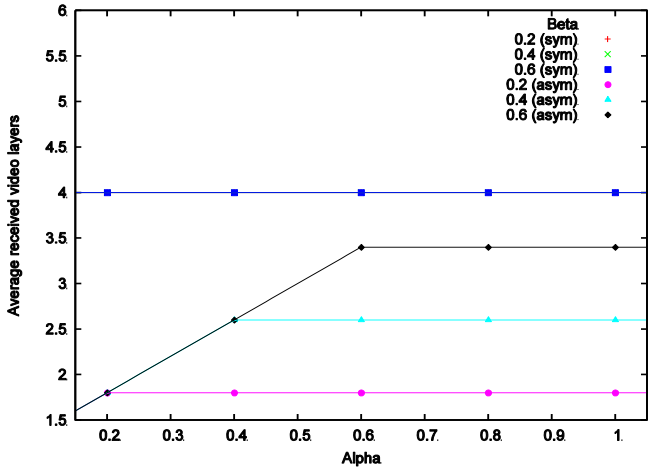

Figure 4.4: Results of using our model on the tree based topology (represented as dots) compared to the analytical solution (depicted by solid lines), when using symmetrical versus asymmetrical (access) link bandwidths. In case of symmetrical access links all results coincide for presented situation.

to be added. The destinations that are located under the level 1 nodes having peer functionality, the weighted result is simply limited by the available bandwidth it receives according to expression B (and no more than the number of video layers). Note that in the case that $peer\_l1 = fo\_2$ (i.e. this generates division by zero), we simply neglect that part since the value of C is 0.

To show correctness of our proposed model and to study the effect of parameter α and β, a parameter sweep is performed by setting α to 0.2, 0.4, 0.6, 0.8 and 1.0, and β to 0.2, 0.4 and 0.6. The value for $peer\_a = 1.0$, $peer\_l1 = 0.0$ and k is one, since only one shortest-path is possible from a source to a destination. Figure 4.4 compares the results (represented as dots) of using our model on the tree topology with the analytical solution (depicted by solid lines), when the access link bandwidth are symmetrical (i.e. up- and download capacity of four video layers) and asymmetrical (i.e. upload capacity of one and download capacity of four video layers). When symmetrical (access) links are used, each destination receives all video layers if α and β sufficiently high; however, asymmetrical (access) links require more bandwidth in the core network in order to allow each destination to receive the video in full quality. Note that our ILP solution indeed produces the results predicted by the analytical approach.

Figure 4.5 depicts the results of the parameter sweep in the situation where asymmetrical access links are used when all, six out of ten or no access peer have uploading capabilities of one layer and none, one or two out of two level 1 node(s) exhibit peer functionality. Bringing peer functionality into the core network increases the average received quality significantly, even when a large part of the

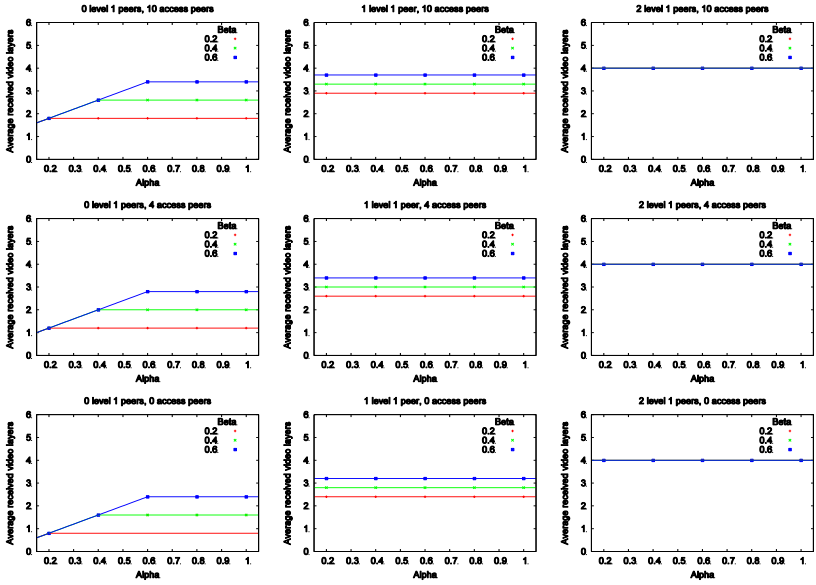

Figure 4.5: Results of our model compared with the analytical solution, when peer functionality is brought into the core network, when all, 60% or no access peer have uploading capabilities. None, one or two of the level 1 nodes exhibit peer functionality and asymmetrical bandwidths are used in the access network.

access peer cannot contribute downloaded information to the rest of the network. Like the results of Figure 4.4, Figure 4.5 shows that both the analytical solution as our model produce the same results.

4.4 Use case: European Parliament streaming

In traditional P2P video distributing networks, selection of nodes to download video chunks from (i.e. choking) is performed in a selfish manner. Typically, those nodes are selected that have the highest bandwidths connections [1-3]. In order to model this, the ILP formulation is extended with a constant parameter $r_{x,y}$ that represents the minimum bandwidth on the shortest-path (i.e. k=1) between node x and y (i.e. the link that has the smallest maximum bandwidth u_e is used for $r_{x,y}$).

The objective function to optimize that represents the traditional form of node selection can be formulated as follows:

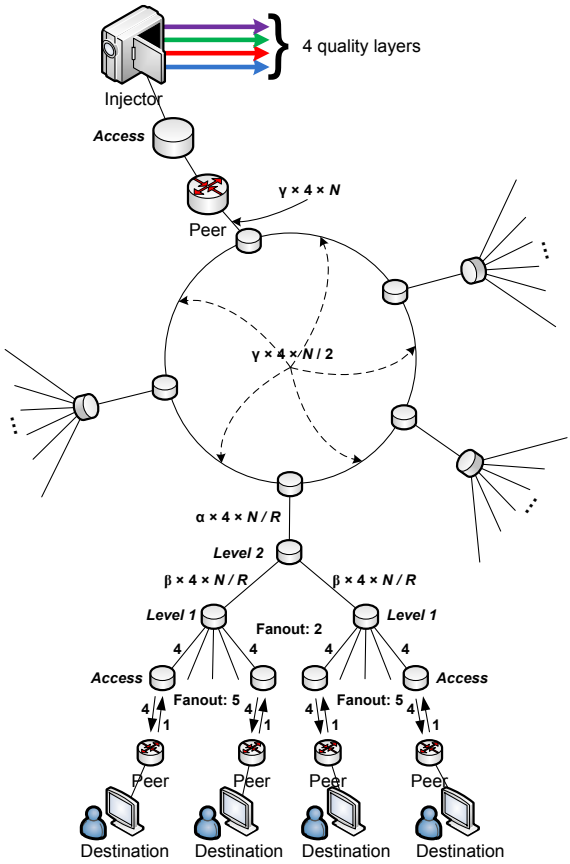

Figure 4.6: A ring network connecting R forwarding nodes. Each ring node acts as a root for a tree topology, build up in the same way as in Figure 4.3.

$$R = maximum\left(\sum_{s\in\{z\cup P\}}\sum_{e\in O_s}\sum_{d\in D}\sum_{l\in L} h_{e,d,d,l}\times r_{s,d}\right) \quad (22)$$

By optimizing objective function R, peer nodes have the incentive to transport video layers to destinations that seem to have the highest bandwidth connection, which, in principle, is exactly the same as selecting (the injector or peer) nodes to download from, that have the highest bandwidth connection. Since the injector peer nodes are encouraged by objective function R to send as much video layers as possible, it is important to prevent unnecessary transport of data between nodes, which is taken care of by constraint (7).

In Figure 4.6 a ring-of-trees topology is depicted, which describes the network model that is representative for a core network in Belgium. The injector node is placed on the main ring and γ is the parameter controlling the available bandwidth on the main ring links connecting the four other root nodes. The number of video qualities is (again) set to four layers, the fan out for the level 2 node is set to two and for the level 1 node to 5. The peers on the access level are assumed to be asymmetrical with a download capacity of four layers and an upload capacity of one layer. Since two shortest-paths are possible between a source and a destination, k is set to two. Parameters α and β are both set to 0.6 (in order to see effects when changing parameter γ) and γ is varied between 0.1 and 1.0.

Figure 4.7 shows the results of using our strategy, which purpose is to deliver each destination a maximum minimal quality, and the traditional solution, where each destination maximizes its received quality. In order to obtain results from the ILP solver [17] within reasonable times, we select ten random destination nodes that

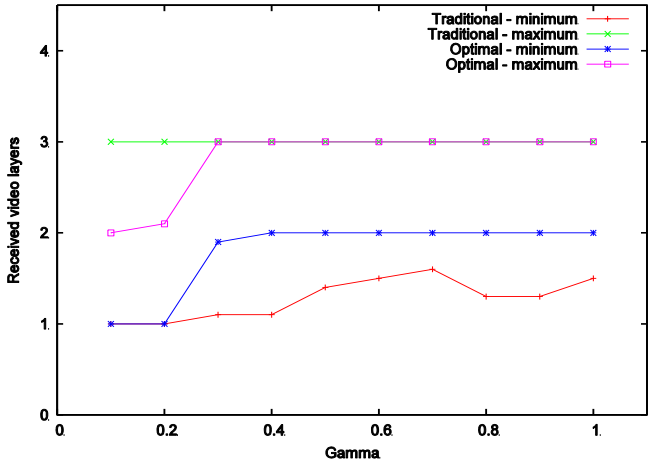

Figure 4.7: Comparing our strategy with the traditional method on the ring-of-trees network topology. Since both methods produces similar results in terms of average received quality, this figure shows the minimum and maximum number of received video layers (averaged over ten independent simulation runs).

are allowed to receive the video and the results show the average over ten independent simulation runs. Because of the symmetrical nature of the topology, both methods produce similar results in terms of average received quality by the destinations. However, as Figure 4.7 illustrates, the gap between the minimum and maximum received quality for our strategy is smaller, indicating that our strategy indeed maximizes the minimum quality that is received by all nodes.

An interesting use case is distributing a live video from the European Parliament, where we use the GÉANT backbone topology as it is in June 2011 [5]. The GÉANT topology [5] depicts the network connections (and their bandwidth capacities) between research and education networks in the European area. Figure 4.8 illustrates the topology that we use in our experiments, where the network link bandwidths are expressed in the maximum number of transportable video layers. The (live) video consists of four layers and the injector peer is located in France. We set $k = 1$ for the traditional strategy and for our optimal method $k \in \{1,2,3\}$ shortest-paths. Again, in order to obtain results from the ILP solver [17], ten random destination nodes are selected and connected to one of the country's forwarding nodes. All peer nodes that are connected to a destination node have an uplink capacity of one video layer and a download bandwidth of four layers in the case of homogeneous end-devices. To model heterogeneous end-devices 60% of the peer nodes connected to a destination node have a capped download capacity of one layer, 30% are able to receive two video layers and 10% are allowed to get the video in its highest quality (i.e. four layers). The results are based on ten independent simulation runs.

Figure 4.9a illustrates the fraction of the destination nodes receiving a specific

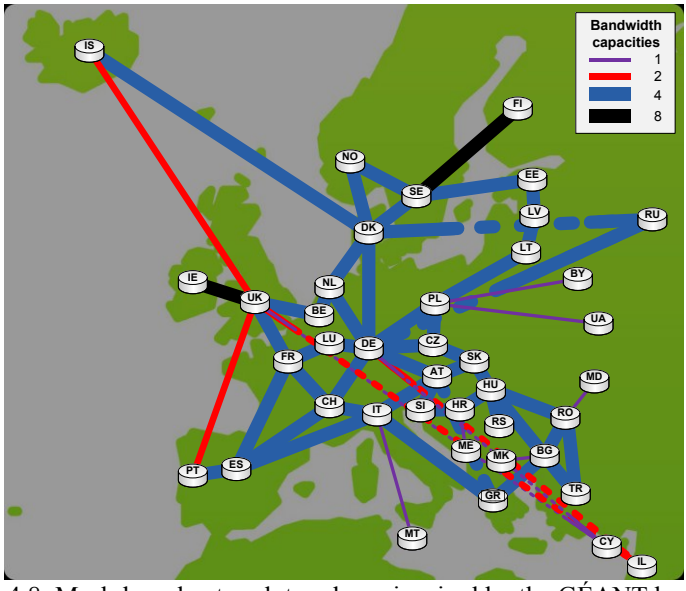

Figure 4.8: Mesh-based network topology, inspired by the GÉANT backbone topology. The injector node inserts a live video stream from the European Parliament, located in France, into the network. A random group of destination nodes try to receive the video feed. The link bandwidth capacities are expressed in number of quality layers.

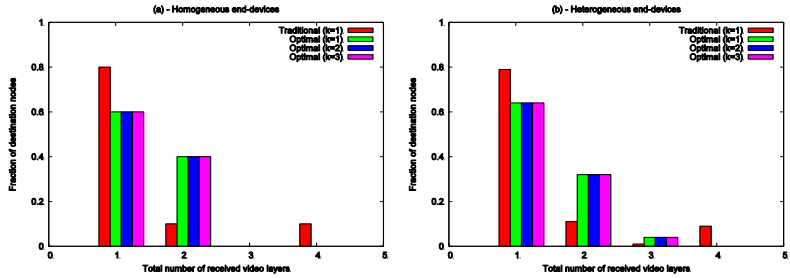

Figure 4.9: Comparing our optimization strategy with a typical traditional (i.e. k is one) Peer-to-Peer video streaming methodology on mesh network topology (i.e. k is one, two or three). Figure 4.9a depicts the situation for homogeneous end-devices (i.e. each destination is allowed to receive the video feed at its highest quality). Figure 4.9b shows the results when heterogeneous end-devices are modeled, where 60% of the users are able to stream at maximum one layer, 30% at maximum two layers and 10% at four video layers.

number of video layers, when using homogeneous end-devices. When applying the traditional (selfish) method for neighbor/piece selection, most destinations watch the video in its base video layer. Only a few destinations are able to receive the video in two or four video layers. When our optimization strategy is used, compared to the traditional method, a smaller fraction of the destinations watch the video stream consisting out of only one video layer and a much larger part of the destination nodes receives the video in two video layers. Although no destination node is able to watch the video at full quality using our strategy (i.e. receiving all four video layers), Figure 4.9a shows that our method offers a more robust solution since a larger fraction of nodes receive layer l_1.

In the situation that heterogeneous end-devices are modeled, Figure 4.9b depicts that less nodes are receiving two video layers. However, since a quality cap is enforced into the network, unused bandwidth becomes available and our strategy allows some nodes to receive three video layers.

4.5 Conclusion and future work

The P2P network model in combination with multi-layer video coding allows to offer a cost efficient mechanism that uses over-the-top video stream routing to optimally adopt to the specifications of heterogeneous end-devices. We believe that, from a video service provider's perspective, the objective should be to maximize the minimum received video quality at each destination, rather than the selfish approach in traditional P2P networks, where each node maximizes its own video quality. To accomplish this, a (centralized) orchestrating unit (e.g. performed by the tracker node) is necessary that manages the video data exchanges by the peers and therefore needs a precise view on both the underlay and overlay network. To study the advances that this orchestrating unit offers, this chapter presents an ILP formulation that is capable to model the underlay-overlay-routing problem of the video layers in the network. By implementing both our solution and the strategy applied in

traditional P2P video streaming networks, we are able to make a thorough evaluation of both and show that our strategy increases the number of nodes that stream the video consisting out of more layers than only the base layer.

To find the optimal solution, the formulation is implemented in an ILP solver [17]. For validation purposes we have investigated a video distribution structure, for which the optimal solution can be derived analytically. Both the analytical solution and our model produce exactly the same results. When applying both our and the traditional strategy on a ring-of-trees network topology, we show that the difference between the minimum and maximum received video layer is smaller for our solution, providing a more robust solution, without reducing the average received quality. A more realistic use case focuses on distributing a live video feed from the France site of the European Parliament to a select group of end-users (e.g. journalists, translators, …) located in different countries in Europe, where the network topology is based on the GÉANT research network [5]. We model homogeneous end-devices by allowing each destination node to download the video feed in its full quality. Heterogeneous end-devices have a limited video quality that they can download and we assume that 60% of the end-devices are capable to receive the base layer, 30% can receive two video layers and only 10% is allowed to get the video in its highest quality. The results show that our optimal strategy significantly decreases the fraction of destinations receiving only the base layer and increases the fraction of destinations downloading two video layers, for both the situation of homogenous and heterogeneous end-devices. However, compared to the traditional method, no destination receives the video in maximum quality.

Our work in this chapter focusses on a steady-state situation, where each video layer is obtained from one source. For future work we plan to study the effects when multiple sources deliver a video layer and look into the robustness of our solution when nodes (suddenly) join or leave the network. In order to analyze larger network topologies we plan to design heuristics that deliver acceptable results for our proposed model within reasonable times. Another approach to compute the routing of video layers for more realistic scenarios (i.e. larger network topologies) is to use a hybrid solution where our solver is used in a multi-step strategy. End-users are divided in groups and in a first step the video layer routing is determined between these groups. In the next step our solver can again be used to compute the optimal solution within each (sub)group separately. Of course, solving the video distribution problem in a multi-step way, will yield sub-optimal solutions, and the quality of these solutions will depend on the topology at hand.

References

[1] J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M.R. van Steen, and H.J. Sips, TRIBLER: a social-based peer-to-peer system, Concurrency and Computation: Practice and Experience, Vol: 20, no: 2, 2008, pp. 127-138.

[2] B. Cohen, Incentives build robustness in BitTorrent, *1st Workshop on Economics of Peer-to-Peer Systems,* 2003, pp. 1-5.

[3] J.J.D. Mol, J.A. Pouwelse, M. Meulpolder, D.H.J. Epema, and H.J. Sips, Give-to-Get: free-riding resilient video-on-demand in P2P systems, *Proceedings of SPIE, Multimedia Computing and Networking Conference (MMCN2008),* 2008, pp. 1-8 (article: 681804).

[4] H. Schwarz, D. Marpe, and T. Wiegand, Overview of the Scalable Video Coding Extension of the H.264/AVC Standard, IEEE Transactions on Circuits and Systems for Video Technology, Vol: 17, no: 9, 2007, pp. 1103-1120.

[5] GÉANT the pan-European research and eduction network: http://www.geant.net/Network/NetworkTopology/pages/home.aspx last accessed in Jun 2011.

[6] S.E. Deering and D.R. Cheriton, Multicast routing in datagram internetworks and extended LANs, ACM Transactions on Computer Systems, Vol: 8, no: 2, 1990, pp. 85-110.

[7] F. Azzedin, Trust-based taxonomy for free riders in distributed multimedia systems, *International Conference on High Performance Computing and Simulation (HPCS2010),* 2010, pp. 362-369.

[8] I. Chatzidrossos, D. György, and V. Fodor, Server guaranteed cap: an incentive mechanism for maximizing streaming quality in heterogeneous overlays, Lecture Notes in Computer Science: Networking 2010, Vol: 6091/2010, 2010, pp. 315-326.

[9] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, D. Zhang, and A. Jaffe, Contracts: practical contribution incentives for P2P live streaming, *Proceedings of the 7th USENIX conference on Networked systems design and implementation,* 2010, pp. 1-14.

[10] Y. Ding, J. Liu, D. Wang, and H. Jiang, Peer-to-peer video-on-demand with scalable video coding, Computer Communications, Vol: 33, no: 14, 2010, pp. 1589-1597.

[11] L. Zhengye, S. Yanming, K.W. Ross, S.S. Panwar, and W. Yao, LayerP2P: using layered video chunks in P2P live streaming, IEEE Transactions on Multimedia, Vol: 11, no: 7, 2009, pp. 1340-1352.

[12] N. Ramzan, E. Quacchio, T. Zgaljic, S. Asioli, L. Celetto, E. Izquierdo, and F. Rovati, Peer-to-peer streaming of scalable video in future Internet applications, IEEE Communications Magazine, Vol: 49, no: 3, 2011, pp. 128-135.

[13] M. Fouda, Z. Fadlullah, M. Guizani, and N. Kato, A novel P2P VoD streaming technique integrating localization and congestion awareness strategies, Mobile Networks and Applications, 2011, pp. 1-10.

[14] R. Iqbal and S. Shirmohammadi, An analytical approach to model adaptive video streaming and delivery, *Proceedings of the 2010 ACM workshop on Advanced video streaming techniques for peer-to-peer networks and social networking (AVSTP2P2010),* 2010, pp. 55-58.

[15] H. Hu, Y. Guo, and Y. Liu, Peer-to-peer streaming of layered video: efficiency, fairness and incentive, IEEE Transactions on Circuits and Systems for Video Technology, Vol: 21, no: 8, 2011, pp. 1013-1026.

[16] M. Kucharzak and K. Walkowiak, Optimization of flows in level-constrained multiple trees for P2P multicast system, *The Second International Conference on Advances in P2P Systems (AP2PS 2010),* 2010, pp. 106-111.

[17] IBM ILOG CPLEX Optimizer: http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/ last accessed in Nov 2011.

5

Combining video layer routing with optimal peering node placement in Peer-to-Peer video streaming networks

N. Sluijs, T. Wauters, C. Develder, F. De Turck, P. Demeester and B. Dhoedt.
Submitted to Transactions on Emerging Telecommunications Technologies.

The previous chapter addresses our research on modeling multi-layer (live) video streaming in a P2P network. However, using exact solvers to calculate the routing of video layers for both the underlay and overlay is only feasible for relatively small network topologies. Therefore, this chapter provides a heuristic method that is able to compute the routing process of multi-layer video for larger networks sizes. Moreover, we extend the optimization strategy to calculate the locations to upgrade nodes with peering application functionality. The simulation results show that increasing the number of peering nodes in a network, increases the average number of received video layers at a destination, with same network capacities. Additionally, bandwidth capacities are used more efficiently by adding extra peering nodes.

5.1 Introduction

Today, broadcasting of video is mainly performed in a traditional manner; where servers send video feeds directly via dedicated networks to end-users. Due to growing bandwidth capacities (especially in the access networks) the Internet gets a more prominent role as being the main medium for transporting (live) video streams to millions of heterogeneous end-devices. Websites of e.g. YouTube and Eurovision Sports are already exploring the possibilities of broadcasting popular (live) events all

over the world, such as World Championships and the Olympic Games. However, the number of users that can watch the (live) video feed at the same time is still limited since bandwidth capacities on the broadcasting servers form the main bottleneck. An interesting network model that offers a scalable mechanism for distributing (live) video is the Peer-to-Peer (P2P) overlay technology. In a P2P network, the peers form a virtual overlay network on top of an actual network, and all peers act as both supplier and consumers (contrary to traditional client-server networks). Since peers can supply downloaded data to each other (i.e. parts of the video stream), not only a scalable and robust solution is offered, but also the server loads on the original source nodes (i.e. injector node) are reduced.

In order to optimally adapt to typical heterogeneous circumstances (such as various asymmetric link bandwidths, end-devices having different display resolutions or even stereoscopic rendering possibilities), next generation P2P video streaming networks use multi-layered video (e.g. Scalable Video Coding [1]) [2]. The video is encoded into multiple layers (i.e. usually divided into a temporal, spatial or quality resolution or a combination of the three) and allows playback of the video when only the base layer is received. Every additional received video layer increases the user's experienced viewing quality of the video feed (such as an increased frame rate or the resolution scaled from e.g. 720p to 1080p). Using multi-layered video coding, end-devices can choose to download only the video layers that they are able to output. Even when two nodes are choosing to receive a different video quality, using multi-layer video has the advantage that both nodes have the ability to exchange video layers. The bandwidth requirements for the node inserting the (live) video stream into the network can be mitigated significantly, since one stream (containing each distinct video layer) might already be sufficient to allow each device to select the right number of video layers to stream.

Our P2P framework that is used to transport multi-layer video consists out of the following entities:

- *Injector node*: offers the video stream to the rest of the network, separated into distinct video layers.

- *Tracker node*: a (often centralized) bootstrap server, providing all necessary information for peering nodes to start the download process. When the streaming of video is started, the tracker node has a coordinating role.

- *Peering node*: nodes containing the peering software functionality and (usually) acting as the entry points for end-devices (i.e. *destination nodes*) to get the video stream from. In this chapter we use the term peering node and peer interchangeably.

As a use case we study the possibility of using a P2P multi-layer video framework to transport a (live) video feed from the European Parliament to end-users located all over Europe. We assume that Internet Service Providers (ISPs) are willing to make relatively small hardware investments in order to turn their networks into a good alternative for distributing (live) video streams (i.e. by upgrading nodes in their network to exhibit peering application intelligence). Since ISPs are mainly interested in the minimum video quality they can offer, our objective focusses on maximizing each destination's minimum video quality while using network

resources as smartly as possible. Therefore, we propose in this chapter a heuristic strategy to compute both the underlay and (consequently) overlay routing for a video stream. The deployment perspective of the proposed method is provider centric, indicating that a limited number of end-users is considered and we have a precise view on the (underlying) network topology. Additionally, our optimization method also allows calculating the locations to install/upgrade nodes to exhibit peering application functionality. We assume that connections between peers are orchestrated by a (central) unit that has the power to control the number of video layers received by a destination in function to increase the number of video layers for multiple other destinations. Although the orchestrating unit (i.e. tracker node) forms a single-point-of-failure in our architecture, the benefit of using a (centralized) coordination instance is to alleviate the burden of decreased network performance formed by free-riders or inefficient data exchanges caused by the distributed algorithms and protocols used by traditional P2P streaming applications. Since the number of nodes ISPs are willing to upgrade with peering functionality mostly depends on the extra benefit they deliver, our proposed heuristic is able to calculate the video layer routing information (e.g. to be used by the orchestrating unit) and the network locations for the peering nodes.

The core network topology used in this chapter is illustrated in Figure 5.1, which is a derived version of the GÉANT backbone topology as it is in June 2011 [3]. The GÉANT topology represents a schematic overview of the connections and their

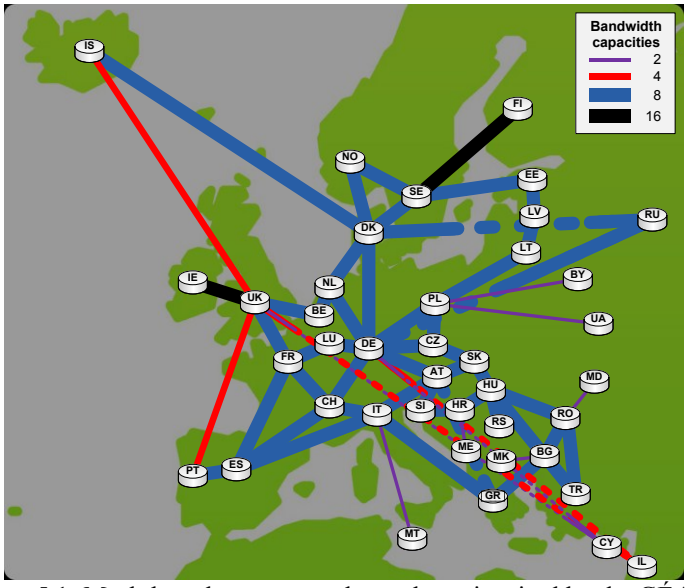

Figure 5.1: Mesh-based core network topology, inspired by the GÉANT backbone topology. We consider the injector node, inserting the multi-layer (live) video feed from the European Parliament, to be located in France. The bandwidth capacities represent the maximum number of video layers the link is allowed to carry, in each direction separately. We assume that each video layer has the same constant bandwidth cost and dashed lines are used to indicated non-intersecting links.

bandwidth limitations between research and education networks in the European area. Although complex models exist to generate (background) traffic representing different kinds of applications (e.g. general P2P software or Content Distribution Networks (CDN)) [4], we use a simplified method by directly transforming the bandwidth capacities to represent the maximum number of video layers the network link is allowed to carry in each direction.[9] We assume that each layer has the same constant bandwidth cost. In our use case the (live) video stream is inserted from the European Parliament located in Strasbourg, France (abbreviated as FR). End-users requesting the (live) video feed are modeled by connecting a <peering, destination> node couple to a country's forwarding node. To study the effects of asymmetrical bandwidth capacities, which is one of the currently limiting factors for the overall performance of P2P networks, the peering node's uplink (i.e. the link from the peering node to the country's forwarding node) is limited to the average of the incoming link capacities. For instance, the uplink of the peering node at Iceland (abbreviated as IS in Figure 5.1) has a maximum bandwidth capacity of three video layers. We assume that at most one <peering, destination > node pair is connected to a country's forwarding node.

This chapter continues in section 5.2 with an overview of related work. Section 5.3 provides and validates our heuristic for solving the routing of video layers in a network. An extension of our optimization strategy is given in section 5.4 that calculates the locations in the network to place peering node functionality. Conclusions and future work are presented in section 5.5.

5.2 Related work

A number of commercial platforms exist that offer streaming video using the Internet as a transport medium (e.g. Octoshape, RawFlow and RayV). However, often traditional client-server based network models are used to transport the video data, resulting in large server and bandwidth costs for broadcasters. On the other hand, several freeware/open-source frameworks employ P2P techniques to offer (live) video streaming solutions, e.g Alluvium, End System Multicast, PeerCast, PPTV (formerly known as PPLive) and Tribler [5]. In order to get decent and stable streams, large numbers of end-users are required that watch the same video feed. To the best of our knowledge, none of the P2P (multi-layer) video streaming networks use topology information to optimize the load in the network.

Another option would be to use IP multicast [6]. However, IP multicast is not a feasible solution for a scalable video streaming service due to the sparse deployment on the Internet. Therefore, our solutions focus on advanced P2P techniques, without requiring large hardware changes to deploy the designed frameworks.

Inherent to P2P networks are mechanisms that motivate/force peers to share with each other when downloading, such as BitTorrent's tit-for-tat mechanism [7] and Tribler's give-to-get algorithm [8]. An unbalanced data exchange is a problem that decreases a P2P system's potential performance in terms of bandwidth utilization [9-11]. Both the unwillingness to share by users and inferior data exchanges as a result of the distributed algorithms form a huge challenge when designing a P2P streaming

[9] The bandwidth values are multiplied with two in Figure 5.1 compared to Figure 4.8, to challenge our optimization strategy when calculating peering node positions.

applications. Therefore, incentives mechanisms are necessary to allow successful deployment of robust and scalable (live) video streaming framework. Our approach to solve these issues is based on extending tracker node privileges with an orchestrating function, managing all data transfers in the network. Based on the orchestration our heuristic method is used to compute the optimal routing strategy for the complete network and providing the highest possible performance gain.

Studies combining P2P technologies with layered video are performed in [12-14] and mainly focus on advanced and selfish buffering strategies by altering the piece picking algorithms and/or neighbor selection strategies. Our solution complements that work by using topology information to manage the video streaming process and increase the network's performance as whole.

In [15-17] mathematical formulations are introduced to model video streaming using P2P mechanisms. In these studies only overlay routing is considered and the authors assume the upload and download capacities for each peer to be the main bottlenecks. In addition to overlay routing, our proposed optimization strategy takes the underlay network into account, allowing to take constraints into account imposed by shared network links in data exchanges. As a consequence the complexity of the problem significantly increases and therefore we use a heuristic algorithm to solve a multi-layered video routing problem in a P2P network.

Capone et al. [18] study underlay and overlay routing optimization in combination with overlay node positioning to create virtual topologies on Internet-like networks, so called Service Overlay Networks (SON). Since the Internet was designed to provide best effort delivery, SONs are used to provide end-to-end Quality of Service (QoS) without requiring any modification to the underlying network infrastructure. Compared to [18], our proposed model considers the routing of multiple video-layers to each destination, possibly via distinct overlay routes (including asymmetric bandwidth properties on the access links). Additionally, we require the video layers to be delivered in-order and originated from the source (i.e. injector) node, resulting in a significant increase in the complexity to solve our problem.

Several generic heuristic strategies exist that are used to find (almost) optimal solutions for various kinds of (combinatorial) problems, such as Ant Colony Optimization, Tabu Search, Genetic Algorithms and Simulated Annealing (SA). As a common denominator, all above mentioned techniques use probabilistic approaches to prevent getting trapped in local optima. The problem we solve in this chapter is in essence similar to the classic and heavily studied vehicle routing problem (VRP), where SA has proven to be a good candidate to solve computationally more complex versions of the standard VRP [19,20]. Therefore, SA forms the basis for our global optimization strategy.

Contributions of this chapter can be summarized as:

- Providing a stochastic heuristic optimization method to calculate the routing of multi-layered video in a P2P overlay network, taking into account the underlay topology.

- Using our heuristic strategy to find ideal positions to upgrade forwarding nodes to contain peering application functionality.

5.3 Heuristic method for routing of multi-layer video in a P2P network

Our heuristic method is based on Simulated Annealing (SA), a stochastic optimization strategy used to find a (close to) optimal solution for a problem. The solution space is randomly sampled and occasionally inferior solution states are accepted in order to jump out of local minima. Before we provide our proposed algorithm based on SA, we introduce in section 5.3.1 all parameters and variables that describe the underlay-overlay-routing problem of multi-layer video streaming in a P2P network. Section 5.3.2 continues with a detailed description of our heuristic strategy, while a validation and evaluation of the proposed strategy is provided in section 5.3.3.

5.3.1 Problem formulation for distributing multi-layer video over a P2P network topology

The problem of transporting a multi-layer video stream over a P2P network is characterized by a network topology, containing an injector node, multiple forwarding nodes, peering and destination nodes and a list of the distinct video layers. Table 5.1 provides an overview of all symbols declared in sections 5.3.1.1 to 5.3.1.5.

Symbol	*Description*
E	Set of all links in the network topology
u_e	Total bandwidth capacity of link $e \in E$
V	Set of all nodes in the network topology
I_v	All incoming links on node $v \in V$
O_v	All outgoing links from node $v \in V$
z	The node that inserts the (live) video stream into the network, $z \in V$
F	Set of forwarding nodes, $F \subset V$
P	Set of nodes containing peering application intelligence, $P \subset V$
D	Set of nodes where an end-user is connected to, $D \subset V$
L	Set of distinct video layers
l_i	Video layer $l_i \in L$
b_i	Benefit value of receiving layer l_i
c_l	Bandwidth cost of transporting layer l
$h_{e,x,l}$	Binary variable indicating whether link e is used to transport layer l to destination $x \in \{P \cup D\}$

Table 5.1: Symbols that define the problem solved by our optimization strategy.

5.3.1.1 The network topology

A directed graph G represents the underlay network topology and contains a set of nodes V and a set of directed edges E. We presume the graph is bi-directional, but asymmetric edge properties can be specified. Each (direction of) edge e from E is characterized by a given maximum bandwidth capacity $u_e \geq 0$. The sets of ingoing and outgoing links of a node v are respectively represented by I_v and O_v. We assume that video layers are routed through the network using a standard shortest-path method (i.e. measured in network hops) [21]. Therefore, Dijkstra's algorithm is used to calculate the shortest-path between two nodes.

5.3.1.2 Forwarding and peering nodes

All nodes in the network that exhibit no peering application intelligence simply forward received data on the shortest-path to the destination, with F the set of all forwarding nodes. P is the set of all peering nodes, which are the nodes running the P2P streaming software and are able to send an incoming stream to multiple destination nodes.

5.3.1.3 Injector and destination nodes

The injector node z is the original source of the (live) video stream and we presume that the injector node is connected to the rest of the network via a peering node. All video streaming requests are performed by the destination nodes, which are located in set D.[10]

5.3.1.4 Video layers

The ordered set L contains each different video layer, sorted by increasing layer rank with l_0 the base layer. We assume that each layer l from L has a constant bandwidth cost per time unit of $c_l \geq 0$. Receiving a layer l_{i+1} is only useful if layer l_i is also received, where $i \geq 0$.

In order to prioritize situations for fairness where multiple destinations all receive a layer l_i over situations that only a few receive layer $l_{>i}$ and the rest receives $l_{<i}$ a constant value b_i per layer l_i is set to express the benefit of receiving the layer. The values $b_i = |D|^{(|L|-i-1)}$ guarantee the following principle: $b_i > (|D| - 1) \times \sum_{j=i+1}^{|L|-1} b_j$, for $0 \leq i < |L|$. This means that the benefit for contribution b_i when a node receives layer l_i is greater than combining the benefits b_j for every other destination for all video layers l_j where $j > i$. Therefore, b_i allows the strategy of maximizing the minimum number of received video layers per destination.

5.3.1.5 Variables

In this subsection e is an edge from E, x a node from set $\{P \cup D\}$, and l is a video layer from list L. The binary variable is:

[10] Heterogeneous end-devices can be modeled by limiting the incoming bandwidth property on the incoming destination node.

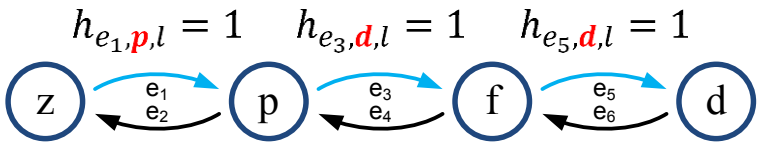

Figure 5.2: Example indicating the usage of variables h, where the injector node z sends a video layer l destined for node d. The video layer is transported over link e_1 to peering node p. Peer p sends the layer over links e_3 and e_5 to destination node d.

- $h_{e,x,l}$ is 1 iff link e is used to transport layer l to node x (i.e. a peering or destination node).

The h-variables in combination with c_l allow the calculation of the bandwidth carried by the links and will be used to assure flow conservation through the network. Figure 5.2 illustrates an example of transporting a layer l to destination node d. The injector node z sends the stream over link e_1 to peering node p. Peer p sends the video layer via link e_3 to forwarding node f, which on its turn forwards l to destination d using link e_5. All other h-variables (not depicted in Figure 5.2) are set to 0.

The combination of the fixed parameters and variables h allows us to describe an underlay-overlay-routing model and is solved using our stochastic heuristic strategy.

5.3.2 Stochastic heuristic optimization strategy

Our optimization strategy is inspired by the generic optimization strategy Simulated Annealing (SA), where random states are generated and accepted based on the quality of a new state compared to the current solution state. In order to compare two generated states, equation (1) depicts the sum that expresses a solution into a numeric value, which adheres to our proposed objective strategy; the numeric value of the solution state is the sum of the benefit values b_i according to the received video layers l_i over all destination nodes d in the network topology. When the solution state value is optimized, the minimum received number of video layers at each destination node is maximized.

$$solution\ value = \sum_{l_i \in L} \sum_{d \in D} \sum_{e \in I_d} b_i \times h_{e,d,l_i} \tag{1}$$

In order to jump out of local minima (or maxima), SA uses the Metropolis criterion [22] to calculate the probability of temporarily accepting an inferior proposal.

$$M(delta, avg\_delta, T) = e^{\frac{-delta}{\frac{-avg\_delta}{\ln(\beta)} \times T}} \tag{2}$$

Parameter *delta* is the numeric difference between two proposals, e.g. calculated by the sum in equation (1). New proposals that are slightly worse than the current

```
B.01  accept(delta, avg_delta, T) {
B.02        if(random() < M(delta, avg_delta, T)){
B.03              return true;
B.04        } else {
B.05              return false;
B.06        }
B.07  }
```

Figure 5.3: Metropolis criterion is used to decide whether or not to accept a proposal (i.e. a newly created route or solution state).

state, have a higher chance to be temporarily accepted. The generated (pseudo) random numbers are uniformly distributed between 0 and 1. Since the magnitude of *delta* is problem dependent, we generalize the probabilities by using an average value *avg_delta* and parameter β to control the starting probability of the average *delta* value. During the course of the simulation, the chance of accepting a less attractive proposal decreases according to the so-called cooling schedule. The temperature T of the cooling schedule starts with a relatively large value, causing the simulation to start with relatively high chances of accepting a less attractive state (i.e. when $T_0 = 1$, β is the initial probability of a accepting the average inferior proposal). By lowering the temperature, the acceptance probability of inferior states decreases over time. In this way, the optimization strategy is able to search a large part of the solution space and finally narrows down to an (almost) optimal solution.

The decision to accept a new proposal (i.e. a newly created route or a solution state) is given in Figure 5.3. Line B.02 indicates that the Metropolis criterion of equation (2) is used to decide whether to accept or reject the proposal. Table 5.2

Parameter	*Description*	*Default value*
T_0	Starting temperature, used to define the acceptance probability for the Metropolis criterion	1
T_{stop}	Stopping temperature, when the temperature drops below this value the optimization process finishes	0.001
α	Used by the exponential cooling schedule to define the rate the temperature T decreases in each temperature update	0.99
β	Initial probability of accepting an inferior proposal, where the value of *delta* = *avg_delta*	0.9
N	Number of dummy inferior proposals needed to calculate *avg_delta*	100
MCL_1	Markov chain length: the number of iteration per temperature step	500

Table 5.2: List of parameters used by our optimization method.

summarizes all input parameters for the optimization method, appended with a brief description of the parameters and the standard value used in our experiments.

5.3.2.1 Initialization of configuration parameters

The main process of finding the optimal solution is given by the algorithm in Figure 5.4. First, all configuration parameters are initialized (C.02):

- Average delta values when constructing new routes and solution states, computed by generating N inferior proposals using a dummy local optimization strategy.
- The start value for the temperature parameter T is set (i.e. T_0).

5.3.2.2 Starting the temperature steps and the Markov chain

The optimization strategy then continues as long as the temperature parameter T is larger than the predetermined stopping temperature T_{stop}. During each temperature step a Markov chain is executed which results in the generation of a number of solution states (i.e. MCL_1, the Markov chain length) that are accepted (or rejected) based on the Metropolis criterion (see Figure 5.3). We propose the use of a

```
C.01  findOptimalSolution() {
C.02        initializeParameters();
C.03        while(T > T_stop) {
C.04              for(i = 0 : MCL_1) {
C.05                    initiateTransaction();
C.06                    if(congestedLinks()) {
C.07        dropVideoLayerFromCongestedLink();
C.08                    }
C.09                    sources = cleanUp();
C.10                    createNewRoute(sources, T);
C.11                    if(acceptNewSolution(T)) {
C.12                          if(bestSolution() &&
C.13                          !congestedLinks()) {
C.14                          markNewBestSolution();
C.15                          }
C.16                    } else {
C.17                          performRollBack();
C.18                    }
C.19              }
C.20              T *= α;
C.21        }
C.22        useBestSolution();
C.23  }
```

Figure 5.4: Main method highlighting the Simulated Annealing inspired optimization approach.

transaction mechanism, for an easy switch back to the current network state in case of a rejection (C.05).

5.3.2.3 Link dropping strategy

When the network contains congested links (i.e. links that currently require more bandwidth than they have available according to u_e), a random network link e is selected from the set of congested links and a random layer l is dropped transported to a random node x (i.e. by setting parameter $h_{e,x,l} = 0$ and x is a peering or destination node) (C.07). After dropping a layer from a link, a clean-up process is necessary to guarantee that all streams flow correctly in the network.

5.3.2.4 Clean-up process

A clean-up process is started (C.09), that makes sure that each destination node d from D receives video layers in-order (and removing layers that do not fulfill this property) and each peering node p from P is only receiving a video layer when it is actually (re)sending it to another peering or destination node (and vice versa). The clean-up method returns per quality layer a list of nodes that are able to act as a source to another peering or destination node (i.e. injector combined with multiple peering nodes).

5.3.2.5 Constructing a new download route

To find an optimal solution, we propose an opportunistic method for creating new routes. Even if a new proposal for transporting a video layer to a node crosses already occupied links, we still consider these routes temporarily. By accepting solution states that are infeasible, we allow the optimization strategy to jump out of local minima and perform a natural way to select links to drop video layers from. The process of starting a new video stream download is explained in Figure 5.5 and starts with selecting a random destination node (i.e. a destination node that is not yet receiving all video layers) (D.02). The next video layer that the destination currently is not receiving is requested from a random node, *can_source,* that acts as a candidate source (i.e. injector or one of the peering nodes) (D.04). Next, the shortest-path (i.e. measured in network hops) between the final destination and candidate source node is constructed using Dijkstra's algorithm. In the situation that the candidate source is not yet part of the list of available sources (i.e. the list containing the injector and peering nodes that are already receiving the video layer, gained by the clean-up method) (D.07), a random node is selected from the list of already active source nodes (D.08) and the shortest-path between the actual source node and the candidate source node is prepended to the video stream path (D.10). Next, the decision to accept or reject the proposed path is based on the number of full links it uses, which is provided to the Metropolis algorithm (D.15). When accepting the new route, line D.17 marks the download of the video layer in the network topology (i.e. by setting $h_{e,x,l} = 1$ on each link e on the constructed path for the specific video layer l, where x is the randomly chosen actual or candidate source node, or destination node x that is closest to e).

```
D.01  createNewRoute(sources, T) {
D.02        (destination, layer) =
D.03        selectRandomDestinationAndLayer();
D.04        can_source = selectRandomNode(P ∪ {z});
D.05        path = shortestPath(can_source,
D.06                                destination);
D.07        if(!sources.contains(can_source)) {
D.08              source =
D.09              selectRandomNode(sources);
D.10              path.prepend(
D.11        shortestPath(source, can_source));
D.12        }
D.13        full_links =
D.14              countNumberOfFullLinks(path);
D.15        if(accept(full_links, avg_full_links,
D.16                                              T)) {
D.17              insertStream(path, layer);
D.18        }
D.19  }
```

Figure 5.5: Process of creating a new route from a source node (i.e. injector or peering node) to a destination. The Metropolis algorithm is used to decide whether or not to accept the proposal, based on the number of already occupied links on the shortest-path.

5.3.2.6 Accepting or rejecting the new solution state

Figure 5.6 introduces the process for accepting or rejecting a new solution state. First, the objective value is calculated on line E.02 by using the algorithm specified in equation (1). When the new solution state produces a better objective value, the proposed state is accepted instantly and the current objective value is adjusted (E.06), else the inferior solution state is accepted according to the Metropolis criterion of Figure 5.3. Note that we adjust the objective values to a logarithmic scale to handle the exponential nature of parameter b_i.

5.3.2.7 Finalizing the temperature steps and the Markov chain

Finally, we determine in line C.11 whether to accept or reject the new solution state (i.e. according to the dropped video layers and the newly created route). In case the proposed state is accepted, the new solution is marked as the best solution (C.14) when the objective value of the new state is better (i.e. larger) than the objective value of the best solution found yet, given that the current (i.e. new) proposed solution state has no congested links (i.e. the proposed solution is practically feasible). When the solution state is rejected, the transaction mechanism is used to roll-back to the situation before the proposal (C.17). After performing the predetermined number of iterations of the Markov chain, the temperature parameter

```
E.01  acceptNewSolution(T) {
E.02        new = solution value, equation (1);
E.03        if(new >= cur ||
E.04           accept(log|D|(cur-new),
E.05              log|D|(avg_difference), T)) {
E.06              cur = new;
E.07              return true;
E.08        } else {
E.09              return false;
E.10        }
E.11  }
```

Figure 5.6: Accepting a solution state for the P2P underlay-overlay-routing problem of streaming videos is based on the numeric value of the new state compared to the current solution. When the new proposal is inferior to the current one, the Metropolis criterion is used in the decision to accept the new state.

T is updated for the next iteration process using an exponential cooling schedule (C.20). Logically, the optimization strategy selects the best found solution as the final result (C.22).

5.3.3 Evaluation of the heuristic approach for multi-layer P2P video streaming

To benchmark the optimization heuristic, we compare the output results with measurements of an Integer Linear Programming (ILP) model implemented with IBM ILOG Cplex solver. In earlier work we have developed an ILP model, that was used to compare our optimization strategy (i.e. maximizing the minimum number of received video layers per destination) with the methodology implemented by current P2P video streaming networks (i.e. maximizing the number of received video layers at each destination). Each topology used in this experiment is the GÉANT network containing ten <peering, destination> node pairs that all are connected to a randomly chosen country forwarding node. We assume that the video feed exists out of four distinct layers. In total we have generated ten network topologies and Figure 5.7 compares over all destination nodes, the minimum, average and maximum received number of video layers for the ILP solver and our optimization strategy. The parameters for our strategy are set according to the values of Table 5.2. The network topologies are ranked according to the average number of video layers a destination node receives. Figure 5.7 validates our heuristic for the proposed topologies by showing identical results compared to the ILP solver.

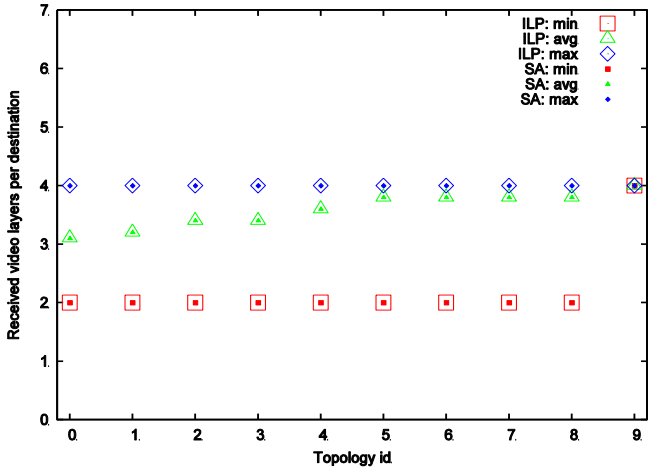

Figure 5.7: Comparing the number of received video layers of using our optimization heuristic with the exact results measured by the ILP model. Ten distinct topologies are generated, with each ten randomly chosen destination nodes on the GÉANT topology. The topologies are ordered on ascending average received number of video layers, each having a constant bandwidth unit $c_l = 1$.

5.4 Combining video layer routing and optimal peering node allocation

Now that we have a heuristic method that is capable to produce (almost) optimal solutions for small topologies, we can scale up the network size (i.e. the number of <peering, destination> node couples in the topology). An interesting question that arises when examining larger network topologies is where to install peering node functionality. Offering peering application intelligence in the network introduces extra costs for network providers, therefore, knowing ideal positions to place the peering nodes can be crucial. To accomplish finding optimal peering node locations, section 5.4.1 provides the extension of our optimization strategy. An evaluation of our proposed optimization method is given in section 5.4.2.

5.4.1 Extending the optimization strategy to locate ideal positions to place peering nodes

An extra parameter is added to our model to represent the number of allowed peering nodes, *allowed_peers*. The set P now represents forwarding nodes that can be upgraded with peering node functionality. Therefore, the initialization phase (C.02) randomly selects *allowed_peers* peering nodes and marks them as the initial peering nodes (i.e. the remainder of the nodes in P are marked as a forwarding node). An extra Markov chain level is introduced that starts with invoking a method

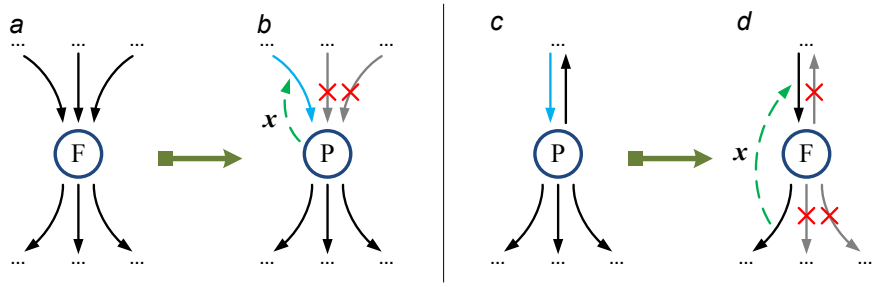

Figure 5.8: Transition strategy for upgrading a forwarding node (i.e. a peer previously marked as forwarding node) to have peering application functionality is shown from *a* to *b*. Downgrading a peering node to a forwarding node is illustrated from *c* to *d*. In both situations the binary variable *h* on an incoming link has to be altered for node *x*, to represent the new situation correctly.

that downgrades a random peering node to a forwarding node and upgrades a random node in P to have peering functionality. The parameter MCL_2 (= 200) represents the number of times a down- and upgrade swap is performed per temperature step. The original (i.e. inner) Markov chain stays untouched and delivers the best objective value that can be reached when using the selected locations for peering nodes. The result of the original Markov chain is then used in the decision to accept or reject the performed swap, similar to algorithm in Figure 5.6 and using the algorithm in Figure 5.3. An extra transaction mechanism layer is required to set back the best solution in case of a rejection.

Figure 5.8 illustrates the transition strategy per distinct video layer for swapping a forwarding node into a peering node (from *a* to *b*) and downgrading a peering node into a forwarding node (from *c* to *d*). By gracefully changing the node's behavior, we are able to keep most of the previously calculated routes. When a forwarding node is upgraded to exhibit peering functionality, per video layer only one incoming stream is kept. The binary variable $h_{e,x,l}$ is changed to one for the particular video layer l and all links e on the path of the incoming stream, where x is the id of the new peering node. Limiting a peering node to a forwarding node means for each video layer replacing variable $h_{e,x,l}$ on the path of the incoming stream, where x is changed into an id of one of the outgoing streams and still satisfying the shortest-path routing principle. All other outgoing streams are removed and to prevent occurring cycles in a route, the chosen outgoing stream is not allowed to be on the link connecting to the node of the incoming link.

5.4.2 Evaluating peering node placement in a P2P video streaming framework

The network topology that we use is depicted in Figure 5.1, with the injector node located in France (FR). Each country's forwarding node connects to a destination node via a peering node. The peering node's upload capacity is limited to the average of the maximum incoming bandwidths on the country's forwarding node. In total the network topology represents 31 countries, which means 1 injector, 30 destination and at most 31 peering nodes. In order to cancel out noise due to random

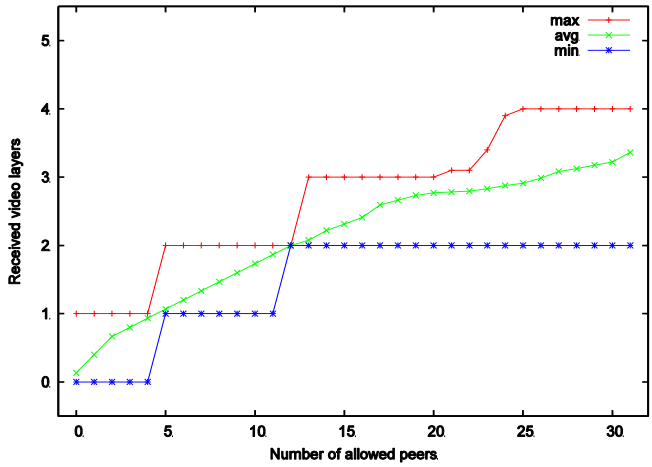

Figure 5.9: Maximum, average and minimum number of received video layers as a function of the number of allowed peering nodes in the network. Intuitively, the result of increasing the number of peering nodes is a higher average number of video layers per destination node.

fluctuations, the results presented in this section are average over ten independent simulation runs. The values for the parameters are chosen as listed in Table 5.2, where $MCL_2 = 500$.

Figure 5.9 shows the minimum, average and maximum number of received video layers as a function of the number of *allowed_peers* (i.e. the number of peering

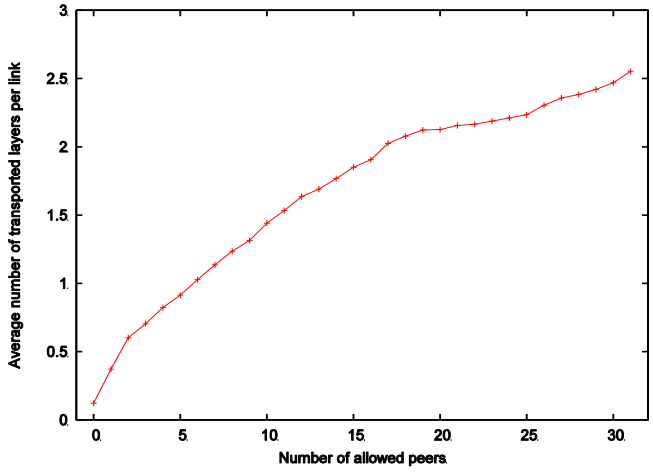

Figure 5.10: Average number of video layers transported per link in relation to the number of allowed peering nodes. Increasing the number of peering nodes means that the average number of received video layers is increased (see Figure 5.9), causing on average more layers to be transported per link.

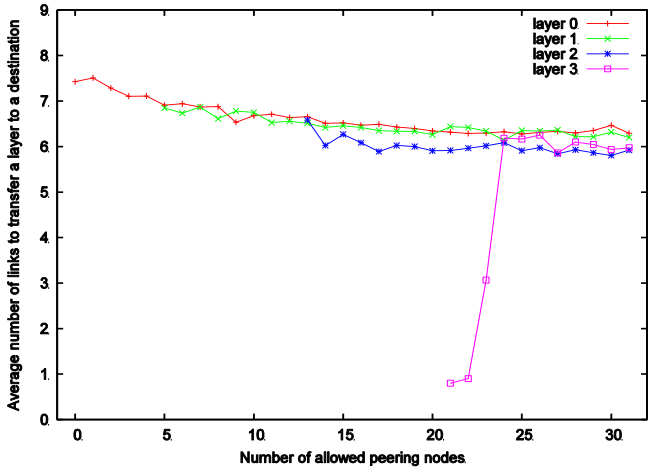

Figure 5.11: Average number of links transporting a video layer divided by the number of destination nodes receiving the layer, in relation with the number of allowed peers.

nodes exhibiting peering application functionality). As expected, increasing the number of peering nodes in the network increases the average received number of video layers at a destination. Note that the overall network minimum number of received layers is limited by the nodes that only have a download bandwidth of two layers (e.g. Malta: MT). Figure 5.9 clearly illustrates our proposed objective strategy, nodes are only allowed to receive a higher video layer when all other nodes are accommodating the lower layers (if possible). Due to the combination of a large number of possibilities to place peering nodes and the relatively small benefit the top layer contributes in the optimization process, fluctuations in the distinct simulation results are observed when the number of allowed peers ranges between 20 and 25.

The bandwidth usage per link in the network is expressed in Figure 5.10 by measuring the average number of video layers transported by a network link. As a consequence of the results in Figure 5.9, increasing the number of peering nodes results in an increase in the average bandwidth usage per link. Figure 5.11 depicts the average number of links transferring a layer, divided by the number of destinations receiving the layer. As Figure 5.11 illustrates, an increase in the number of peering nodes results in more efficient usage of the link bandwidths since the average number of links transporting a layer gradually decreases. For our proposed network topology, only 5 peering nodes are necessary to provide all endpoints the base video layer and 12 peers are required to provide all destinations layer 1. When 31 peering nodes are available, the average number of links to a destination decreases with 9% for the base layer and 5% for layer 1, although no extra endpoints are receiving these video layers. Again, the combination of large possibilities to position the peering nodes and the relatively small benefit layer 3 provides causes small fluctuations in the distinct simulation results when the number of allowed peers is between 20 and 25.

5.5 Conclusion and future work

Due to increasing bandwidth capacities in the Internet (especially in access networks), next generation P2P video streaming frameworks using multi-layer video coding solution provide a good alternative to distribute video feeds. Advantages of using P2P techniques in combination with multi-layered video are the ability to easily adapt to heterogeneous end-devices, providing a cost-efficient, robust and scalable solution that naturally exhibits load balancing and reduces server load. From a video service provider's perspective the objective could be to maximize each destination's minimum video quality (expressed in number of received video layers). The challenge is to optimally use the network infrastructure to transport the video layers to their destinations. To accomplish this, we propose to integrate an orchestrating engine that directs the routing process in the overlay network, assuming that this unit has a precise view on the (physical) underlay topology. In this chapter we propose a stochastic heuristic that is used to find a (close to) optimal solution that can be used by the orchestrating unit. We benchmark the heuristic by comparing the results with the results of an exact solver for our evaluation scenario.

Since small hardware investments, by upgrading nodes with peering functionality, can result in major increases in the system's performance (e.g. the number of received layers is increased), an interesting question that arises is which nodes to upgrade. Therefore, we have extended our proposed optimization strategy to compute along with the video layer routing the best locations to place peering application functionality. The simulation results show that increasing the number of peering nodes in a P2P video streaming network results in an increase in the average received number of video layers at a destination. Consequently, the average bandwidth usage per network link increases due to the increase of the average number of layers an endpoint receives. However, extra peering nodes results in more efficient usage of the network links since the average number of links needed to transfer one layer to one destination decreases gradually. This decrease continues even when the number of destinations receiving the layer stays the same, i.e. a 9% and 5% decrease is seen for respectively layer 0 and 1 when no extra endpoints are accommodating these layers.

Since our work focusses on steady-state situations where a video layer is fully obtained from one source peer, interesting future work is to expand our model to allow multiple peering nodes to deliver segments of a video layer to a node and study the effects of nodes (suddenly) leaving or joining the P2P network. Interesting future work involves studying different mechanisms to (further) increase the number of nodes and destinations in the network topology. For instance, we can already lower the number of Markov steps and the stopping temperature to reduce the computation time, however the effects on the solution quality is not studied yet. Another approach to solve the problem for larger network topologies is incorporating a multi-step mechanism, where destinations are divided into groups and first the routing between groups is computed. In the next step the video layer routing between nodes in each (sub)group can be computed by our heuristic method.

References

[1] H. Schwarz, D. Marpe, and T. Wiegand, Overview of the Scalable Video Coding Extension of the H.264/AVC Standard, IEEE Transactions on Circuits and Systems for Video Technology, Vol: 17, no: 9, 2007, pp. 1103-1120.

[2] U. Abbasi, M. Mushtaq, and T. Ahmed, Delivering scalable video coding using P2P Small-World based push-pull mechanism, *Global Information Infrastructure Symposium (GIIS2009),* 2009, pp. 1-7.

[3] GÉANT the pan-European research and eduction network: http://www.geant.net/Network/NetworkTopology/pages/home.aspx last accessed in Jun 2011.

[4] E. Palkopoulou, C. Merkle, D.A. Schupke, C.G. Gruber, and A. Kirstädter, Traffic models for future backbone networks - a service-oriented approach, European Transactions on Telecommunications, Vol: 22, no: 4, 2011, pp. 137-150.

[5] J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M.R. van Steen, and H.J. Sips, TRIBLER: a social-based peer-to-peer system, Concurrency and Computation: Practice and Experience, Vol: 20, no: 2, 2008, pp. 127-138.

[6] S.E. Deering and D.R. Cheriton, Multicast routing in datagram internetworks and extended LANs, ACM Transactions on Computer Systems, Vol: 8, no: 2, 1990, pp. 85-110.

[7] B. Cohen, Incentives build robustness in BitTorrent, *1st Workshop on Economics of Peer-to-Peer Systems,* 2003, pp. 1-5.

[8] J.J.D. Mol, J.A. Pouwelse, M. Meulpolder, D.H.J. Epema, and H.J. Sips, Give-to-Get: free-riding resilient video-on-demand in P2P systems, *Proceedings of SPIE, Multimedia Computing and Networking Conference (MMCN2008),* 2008, pp. 1-8 (article: 681804).

[9] F. Azzedin, Trust-based taxonomy for free riders in distributed multimedia systems, *International Conference on High Performance Computing and Simulation (HPCS2010),* 2010, pp. 362-369.

[10] I. Chatzidrossos, D. György, and V. Fodor, Server guaranteed cap: an incentive mechanism for maximizing streaming quality in heterogeneous overlays, Lecture Notes in Computer Science: Networking 2010, Vol: 6091/2010, 2010, pp. 315-326.

[11] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, D. Zhang, and A. Jaffe, Contracts: practical contribution incentives for P2P live streaming, *Proceedings of the 7th USENIX conference on Networked systems design and implementation,* 2010, pp. 1-14.

[12] Y. Ding, J. Liu, D. Wang, and H. Jiang, Peer-to-peer video-on-demand with scalable video coding, Computer Communications, Vol: 33, no: 14, 2010, pp. 1589-1597.

[13] L. Zhengye, S. Yanming, K.W. Ross, S.S. Panwar, and W. Yao, LayerP2P: using layered video chunks in P2P live streaming, IEEE Transactions on Multimedia, Vol: 11, no: 7, 2009, pp. 1340-1352.

[14] N. Ramzan, E. Quacchio, T. Zgaljic, S. Asioli, L. Celetto, E. Izquierdo, and F. Rovati, Peer-to-peer streaming of scalable video in future Internet applications, IEEE Communications Magazine, Vol: 49, no: 3, 2011, pp. 128-135.

[15] R. Iqbal and S. Shirmohammadi, An analytical approach to model adaptive video streaming and delivery, *Proceedings of the 2010 ACM workshop on Advanced video streaming techniques for peer-to-peer networks and social networking (AVSTP2P2010),* 2010, pp. 55-58.

[16] H. Hu, Y. Guo, and Y. Liu, Peer-to-peer streaming of layered video: efficiency, fairness and incentive, IEEE Transactions on Circuits and Systems for Video Technology, Vol: 21, no: 8, 2011, pp. 1013-1026.

[17] M. Kucharzak and K. Walkowiak, Optimization of flows in level-constrained multiple trees for P2P multicast system, *The Second International Conference on Advances in P2P Systems (AP2PS 2010),* 2010, pp. 106-111.

[18] A. Capone, J. Elias, and F. Martignon, Routing and resource optimization in service overlay networks, Computer Networks, Vol: 53, no: 2, 2009, pp. 180-190.

[19] S.W. Lin, V.F. Yu, and S.Y. Chou, Solving the truck and trailer routing problem based on a simulated annealing heuristic, Computers & Operations Research, Vol: 36, no: 5, 2009, pp. 1683-1692.

[20] S.W. Lin, V.F. Yu, and C.C. Lu, A simulated annealing heuristic for the truck and trailer routing problem with time windows, Expert Systems with Applications, Vol: 38, no: 12, 2011, pp. 15244-15252.

[21] M. Menth, R. Martin, M. Hartmann, and U. Spörlein, Efficiency of routing and resilience mechanisms in packet-switched communication networks, European Transactions on Telecommunications, Vol: 21, no: 2, 2010, pp. 108-120.

[22] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller, Equation of state calculations by fast computing machines, The Journal of Chemical Physics, Vol: 21, no: 6, 1953, pp. 1087-1092.

6

Conclusions and research perspectives

In this dissertation we have presented our research results carried out in the context of sharing personal content. We focused on two key concepts: locating and transferring personal content items to the end-users. Current solutions are still using traditional client-server models. However, due to increasing heterogeneous circumstances and growing personal content collections, advanced mechanisms are necessary to fulfill all expectations of end-users.

6.1 Main research results

Chapter 2 addresses our findings on distributed locating of personal content items. We have extended a structured P2P network technology (i.e. a DHT) with a cooperative caching framework to reduce lookup delays. Our proposed cooperative caching strategy reacts on the location dependent request patterns that characterize personal content items. Through an update protocol neighbors in the DHT ring are informed of local cache changes. In this way nodes can avoid storing the same copies that can be retrieved in only one (overlay) hop and thereby virtually increases a local node's cache size. The cooperative caching framework is compared with a state-of-the-art proactive replication strategy (i.e. Beehive). In the situation of uniformly distributed lookup request, the analytical model of Beehive provides a larger performance increase compared to the cooperative caching solution. However, since lookup requests for personal content items exhibit location dependent request patterns (i.e. popularity of objects are different for each node), our cooperative caching strategy outperforms Beehive significantly. Besides a direct comparison between our advanced caching mechanism and Beehive, Chapter 2 presents a detailed study of the inner working of our proposed solution. We show that the message overhead introduced by the update protocol to enable cooperation between peers is acceptable, since the performance increase (i.e. reduction of the average number of hops per lookup request) is higher than the cost the update protocol

introduces. Additionally, the results show that on average more than 20% of the peers store their top ten most popular items in the local cache. When a node is not caching a top ten item locally, the chances are relatively high that one (or both) of the neighbors has the particular personal content item. A disadvantage of using a DHT to locate personal items is the limitation of the deterministic lookup property; objects can only be found when the exact lookup *key* is known. However, since frameworks that are offering multiple keyword and range queries still (have to) use the basic key-based routing mechanism of the DHT, our proposed mechanisms are able to increase the performance of such frameworks already.

As pointed out earlier, reducing lookup time does not imply that content itself can be accessed more quickly. Therefore Chapter 3 proposes a multi-level caching architecture to enable storing personal content items closer to end-users that are frequently accessing them. By adopting a multi-level caching solution in a PCSS, significant parts of the network can be alleviated and access times are reduced. We have dimensioned the capacities of the caches for a tree network (by using analytical calculations), so that each level serves an equal share of the requests. The simulation results agree to the designed analytical model by showing that the cache hit ratios converge closely to the same (i.e. calculated) values. Additionally we analyze the time it takes to store a particular file on as many level one caches as possible (i.e. expressed in the number of individual downloads for the file). Again, the measurements obtained through simulations confirm the analytical results of the number of level one caches storing (on average) a particular file.

Due to an ongoing increase of bandwidth capacities, the Internet has become an interesting medium to transport streaming media. In contrast to traditional files, streaming media are useful from the moment the first data segment arrives. Since classic client-server models are limited in scalability, P2P overlay technologies seem the right candidate to transfer video streams to end-users. Due to the heterogeneous circumstances in both bandwidth capacities and rendering devices, next generation P2P (live) video streaming networks are incorporating multi-layered video. To optimally transfer video streams to end-users we propose an orchestrating engine that is topology-aware and manages the transport of each video layer in the network. To study the advantages of directing video streams in a P2P network, Chapter 4 presents a mathematical formulation that models both the underlay and overlay routing of the distinct video layers. Since video service providers are mainly interested in the minimum quality they can offer to their end-users, for a given topology the objective for our orchestrating unit is to maximize the minimum number of video layers received at each destination. On the other hand, current P2P video streaming networks objectives is to maximize greedily the local peer's video quality. Both objective strategies are studied using our mathematical model and the main results in Chapter 4 show a significant reduction in the fraction of destinations only receiving the base layer and an increase in the fraction of nodes downloading more layers, when comparing our optimization strategy with the traditional objective.

Since exact solvers are only feasible to calculate the underlay-overlay-routing problem on relatively small network topologies, Chapter 5 continues our research by presenting a heuristic optimization method that is able to compute the routing process for multi-layer video on larger network topologies. Moreover, we extend the heuristic strategy to calculate ideal positions to upgrade existing nodes with peering application functionality. In this way ISPs are able to investigate the benefits extra

peers deliver to the network at the expense of installation costs. Intuitively, our results show an increase in the average number of received video layers when the number of peering nodes in the network increases. As a side effect, by increasing the number of peering nodes the bandwidth capacities are more efficiently used, since the average number of links needed to transfer one layer to one destination decreases gradually (even when all nodes already receive the particular layer).

6.2 Future application and research areas

Current upcoming technologies change the main way software is provided to end-users. Instead of building complete solutions, a more modular approach is taken where each application fulfills a single service. The underlying middleware is responsible for complete interworking between the set of different software modules and, thereby, forms a whole new range of applications. Current examples are often found for mobile devices, where for instance one module provides transparent synchronized storage space (such as Dropbox), another module allows performing image manipulations (such as Instagram) and the user's social network module (such as Facebook) is used to share the results with the rest of the world. Although virtualization and modularization is a concept existing for a long time, the explosive growth of end-users, increasing (computing, storage and bandwidth) capacities and growing numbers of heterogeneous devices, make these concepts more important by the day.

In our perspective a PCSS is not a solution offered by one instance or technology, but is operated and offered by a whole set of hardware and software providers. Therefore, the research study performed in this dissertation concentrates on increasing the performance and efficiency of back-end systems and technologies that are necessary to offer such a service.

Although our use case focuses on locating and delivering personal content items to end-users, there are many other usage scenarios where our designed techniques are applicable. Advantages can be provided in e.g. the medical sector, where information of patients is shared transparently between different instances. When for instance a surgeon has immediate access to a full medical dossier of a patient's history at good quality, it is assumable that this can lead to faster and more accurate diagnosis.

Although the usage scenarios and designed optimization techniques are promising, future research is still necessary. We assume that current limiting factors will hold for the (near) future:

- Storage and computation capacities for indexing of all digital items cannot be performed cost-efficiently from one single location.

- Bandwidth capacities (even without asymmetrical properties) are not able to transfer all data (streams) in full quality without the end-user experiencing (startup) delays.

Therefore, research is needed that combines our proposed cooperative framework with advanced multi-keyword lookup and range query mechanisms, so they can be fine-tuned together. Although different solutions exists that make physical neighbors also logical neighbors in a DHT, no study is performed that

shows the consequences of these strategies on the cooperative caching performance. Additionally, interesting future work is to see what the effects are on using neighbor groups instead of only using the predecessor and successor of a node as the neighbors.

Since caching architectures benefit especially from location dependent request patterns, detailed studies are needed that provide accurate models for different kinds of media. For our multi-level caching framework, realistic file size distributions and time dependent request patterns (especially flash-crowds) also have a major impact on the performance of the proposed architecture. Cache replacement strategies that react on a combination of location, time and size dependent patterns are a challenging task to design.

The results of routing multiple video layers and positioning peering nodes are based on steady-state situations, where one source (injector or peering node) delivers a complete video layer to a destination. Therefore, interesting future work is to expand the proposed model where multiple sources deliver segments of video layers to destinations. Moreover, destination/peering nodes that (suddenly) join or leave the overlay network can have major impacts on the performance of the network. Altering our proposed solution to optimally deal with these situations is necessary to optimize the robustness of the framework. Since multiple overlay services (centrally) route their traffic in the Internet, optimized protocols to enable cooperation between orchestrating engines is essential for ISPs to maximize the throughput to their end-users.

As a common denominator, all our presented mechanisms use cooperation to provide a better service than one single instance is able to. Deploying these mechanisms is a great challenge and therefore we are interested to see what the future brings by integrating the advanced algorithms and protocols that are designed today by all researchers.

A

Caching strategy for scalable lookup of personal content

N. Sluijs, T. Wauters, B. De Vleeschauwer, F. De Turck, B. Dhoedt and P. Demeester.
Published in the Proceedings of the 2009 First International Conference on Advances in P2P Systems (AP2PS2009).

This appendix complements Chapter 2 by extending the RTDc caching algorithm with a sliding window (Section A.4.4), which allows the framework to react on (sudden) changes in content popularity. The simulation results in Section A.5.3 show that a larger sliding window size increases the efficiency of the algorithm when the popularity distribution of the requests is stable. However, when content popularity changes frequently (or new content is introduced often), a smaller size of the sliding window can provide a more robust solution, since the changes are noticed more quickly. Additionally, Section A.5.4 shows that introducing by using the caching framework the request load is spread more efficiently over the nodes in the DHT, decreasing the hotspot problem.

A.1 Introduction

An important trend today is to create and share *personal content*, such as text documents, digital photos, music files and personal movies, with others. End-users can share their personal content using websites such as YouTube[11] and Flickr[12], for personal movies and digital photos respectively. Since the number of personal files

[11] http://www.youtube.com/
[12] http://www.flickr.com/

grows enormously, managing a personal archive has become a complex and time consuming task. Nevertheless, end-users expect they can locate, control, access and share their personal content from any device, anywhere and at any time. However, current systems that provide storage for personal content, such as YouTube and Flickr, set limitations in order to cope with the workload. In many cases the file size is limited, restrictions are set on file formats or no possibility is provided to access personal content from different types of devices, such as desktop computers, laptops, PDAs (Personal Digital Agent) and mobile phones. This implies that these systems are not able to offer a real *quality-aware* and *scalable* solution for *transparent* storage of personal content.

A networked solution that offers storage space to end-users in a transparent manner, from different types of devices and is able to cope with the expected workload is a *Personal Content Storage Service* (PCSS) [1]. In order to come to a successful deployment of a PCSS, research is needed for each different aspect, such as user centric security, content presence, content replica management and content indexing. An essential feature of a PCSS is the ability to search worldwide through the dataset of personal files, therefore this chapter presents the research on *content indexing*.

A data structure that allows searching through extreme large datasets is a *Distributed Hash Table* (DHT). A DHT is a (structured) peer-to-peer network that offers scalable *lookup*, similar to a hash table. A <*key*, *value*>-pair is stored into the DHT and every node participating in the DHT is able to efficiently locate *values* that correspond to a certain *key*. In the case of a PCSS, the *key* is for instance a file name or represents tags/keywords that describe the personal file. A *value* in a PCSS is a link to the location the file is stored, for instance YouTube or Flickr. Different implementations of a DHT already exists, such as Chord [2] and Pastry [3].

A disadvantage of a DHT is that it only offers content lookup when the exact keyword is known. However, users want to be able to search through content using *multiple keywords* and *range queries*. In order to provide end-users the ability to search through the dataset of personal content, DHT architectures and algorithms have to be improved first. In this chapter we focus on improving the performance of lookup requests in DHTs. An important aspect is that some keywords are more popular than others. Nodes responsible for popular keywords need to handle more requests than others, which results in so-called *hotspots* in the DHT. To reduce the hotspot problem and optimize the lookup performance we introduce a *caching layer* on top of a DHT.

This chapter continues in Section A.2 with an overview of related work, Section A.3 provides an overview of the caching architecture and Section A.4 introduces the caching algorithm, the validation and evaluation of the caching algorithm is provided in Section A.5, and finally, we conclude this chapter in Section A.6.

A.2 Related work

In *unstructured* Peer-to-Peer (P2P) networks query search is done using a query flooding model. The TTL (Time-To-Live) limit is used to prevent overloading the network. In order to improve the efficiency of the query flooding model, Wang *et al* describe a distributed caching mechanism for search results in [4]. However, using the TTL limit implies that personal content stored in such a network has no

guarantees to be found, which makes this type of search mechanism less suitable for a PCSS.

The Beehive [5] framework enables DHTs to provide an average lookup performance of $O(1)$ through proactive replication. According to the evaluation made in [5], Beehive outperforms the passive caching technique used by Pastry [6] in average latency, storage requirement, network load and bandwidth consumption. With passive caching, objects are cached along all nodes on a query path [5], while Beehive's replication strategy is to find the minimal replication level for each object such that the average lookup performance for the system is a constant C number of hops [5]. Beehive assumes that there is a global power law (or Zipf-like) popularity distribution and requests are uniformly distributed over the network. However, in the scenario of the PCSS it is conceivable that locality exists in request patterns, which has a major influence on the performance of a caching algorithm and requires a less expensive solution than Beehive.

In [7] the concept of view trees is introduced, that uses result-caching in order to avoid unnecessary duplication of work and data movement. Results of (conjunctive) attribute queries are cached in the view tree and are later on used to resolve queries that contain (parts of) the cached query results. Although the view tree tries to avoid duplication of work and data movement, each search query is issued on the root (node) of the view tree. This aspect prevents successful deployment of a view tree in a PCSS system.

Previous studies on caching techniques [8] or distributed replica placement strategies for *Content Distribution Networks* (CDN) [9,10] show that by taking distance metrics and content popularity into account, a performance increase is obtained compared to more straightforward heuristics such as *Least Recently Used* (LRU) or *Least Frequently Used* (LFU). By using *cooperative caching* [11] a performance increase, compared to independent caching, can be achieved through load balancing and an improved system scalability. In this case it is important to continuously keep track of cache states. Our caching strategy uses the distance metrics and content popularity, as well as cooperative caching to increase the PCSS lookup performance.

A.3 Caching architecture for DHT performance optimization

To increase the performance of the lookup process for the PCSS, we introduce a caching layer between the application and DHT layer. Figure A.1 shows the caching architecture for the PCSS and illustrates the generic idea of a (Chord) DHT. In the example of Figure A.1, eight nodes span the DHT network for storing references to locations of personal content. By using a *hash* function both content references and nodes can be mapped to a *numeric identifier space*. In Figure A.1 we assume that nodes depicted with a higher number have a higher numeric identifier. Each node is responsible for storing *values* belonging to *keys*, which numeric identifier is between the numeric identifier of the preceding DHT-node (excluding) and the numeric identifier of the current node (including).

In order to efficiently route message in a DHT, every node keeps a *finger table*. This finger table maps numeric identifiers to nodes, where the distance between the numeric identifier of the current node and the numeric identifiers in the finger table

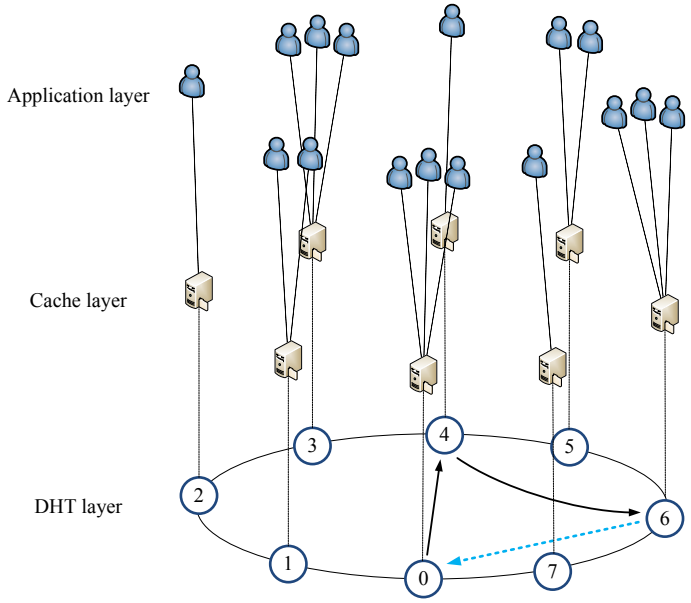

Figure A.1: To increase the lookup performance and reduce hotspots, the Personal Content Storage Service uses a caching layer between the application layer and the Distributed Hash Table.

increases exponentially. In this way, messages are sent to a node minimal half the distance of the key space closer to the destination node. When using the same numeric identifier space as the node numbers in Figure A.1, the finger table of, e.g., node 0 contains mappings to node 1, 2 and node 4. In this way the average (and worst case) number of hops for a lookup has a complexity of $O(\log N)$, where N is the number of nodes in the DHT network.

When a user requests a personal content object in the PCSS, the DHT is used to lookup the link to the location the object is stored. At the bottom of Figure A.1 an example is depicted of a traditional lookup request, initiated by a user connected to node 0. Node 0 forwards the request to the node in its finger table with the numeric identifier closest to the hash value (i.e. node 4), this process is repeated until the target node is reached (i.e. node 6). Finally, the target node replies directly to the requesting node (i.e. node 0) the result. Not depicted in Figure A.1, storing references to object locations into a DHT is performed in a similar way, except no reply message is returned.

Since the *value*-part of <*key*, *value*>-pairs are typically locations where (the latest version of) personal content items are stored, no synchronizations need to take place in the caching architecture introduced in this chapter.

To improve the lookup performance, each DHT node contains a relatively small amount of storage space (the cache) to temporarily duplicate <*key*, *value*>-pairs. This implies that <*key*, *value*>-pairs are stored on average closer to end-users and therefore the average time (measured in number of hops) needed for a lookup decreases. Another benefit of the caching architecture is that multiple nodes are able

to handle lookup request of popular content, which alleviates the hotspot problem enormously.

A.4 Cooperative caching algorithm

In order to efficiently utilize the available cache space on each node, a caching algorithm decides which cache entry is removed in order to store a more valuable lookup result. The caching algorithm designed for a PCSS is introduced in this section. The Request Times Distance (RTD) caching algorithm uses an update protocol to enable cooperation between caches and a sliding window algorithm to be able to react efficiently on sudden changes in the popularity distribution of the personal content.

The popularity of personal content is best described by a *power law* (*Zipf-like*) distribution. This distribution states that some personal content is highly popular and the rest has more or less the same low popularity. In (1) the Zipf-like probability mass function [12] is provided, where C denotes the number of personal content items and α is the exponent characterizing the distribution.

$$P_{Zipf-like}(x) = \frac{x^{-\alpha}}{\sum_{j=1}^{C} j^{-\alpha}} \quad (1)$$

$P_{Zipf\text{-}like}(x)$ determines the probability that a personal content object having rank x is requested, where $x \in \{1, \ldots, C\}$. This implies that a personal content object having a lower rank (i.e. a greater value for x) is less popular, $\alpha > 0$. In [13] Backx et al show, with a number of practical experiments using popular P2P file sharing applications, that α is usually between 0.6 and 0.8.

We assume that locality exists in the request patterns of nodes requesting personal content. This idea is supported by the research performed by Duarte *et al* in [14], where geographical characterizations of requests patterns are studied for YouTube content. However, until now no concrete and generalized probability mass function is extracted that describes the locality based request distribution. Therefore, we model the distribution of requests using the *Normal* probability mass function. In (2) the Normal probability mass function is provided, where the mean is μ and σ the standard deviation.

$$P_{Normal}(y) = \int_{y-\frac{1}{2}}^{y+\frac{1}{2}} \frac{1}{\sigma.\sqrt{2\pi}} e^{\frac{-(z-\mu)^2}{2\sigma^2}} dz \quad (2)$$

$P_{Normal}(y)$ is the probability that a personal content item is requested from node y. Assuming that the node that uploads the personal object has the highest probability to request it, the $(y\text{-}\mu)^{th}$ neighbor of the uploading node μ will request the personal content object. The value σ is used to increase or decrease the spreading of requests over the network. A higher value of σ makes the distribution more uniform, since more neighboring nodes will request the personal content item.

Basic DHTs use hash functions to map nodes onto the numeric identifier space, which means that nodes are more likely to have different neighbors in the DHT than in the actual network topology. Different research studies are already performed that address the issue of including neighboring nodes as neighbors in DHTs [15,16]. For instance, by assigning a locality-aware identifier to each node, the network topology is embedded into the DHT [15]. In our use case, we assume that the DHT is locality-aware. Neighbors in the PCSS are also neighboring nodes in the actual network.

A.4.1 Standard caching algorithms

Standard caching algorithms exist, such as *Least Recently Used* (LRU), *Least Frequently Used* (LFU) and *Most Distant Lookup* (MDL). LRU stores the *r* recent, distinct lookup results in a cache of size *r*, LFU stores the results of the most frequently requested lookups, and MDL caches results that require most hops to perform the lookup.

In Figure A.2 we illustrate the importance of the power law distribution for the popularity of personal content and the Normal distribution to model locality of lookup requests, for the performance of a caching algorithm. The caching algorithm used in Figure A.2 is LFU, with cache size S is 10, each of the N nodes uploads 50 distinct personal content objects and is in non-equilibrium steady state. For the Normal request distribution σ is equal to 2.0 and for the power law (Zipf-like) content popularity distribution α is set to 0.6 and C is equal to $50 \times N$. Figure A.2 depicts that for a relatively smaller network size the average number of hops needed for a lookup is comparable for both the uniform and Normal request distribution, when using a standard caching algorithm (i.e. LFU). Due to those relatively small network sizes, both the request distributions let all nodes perform requests to each

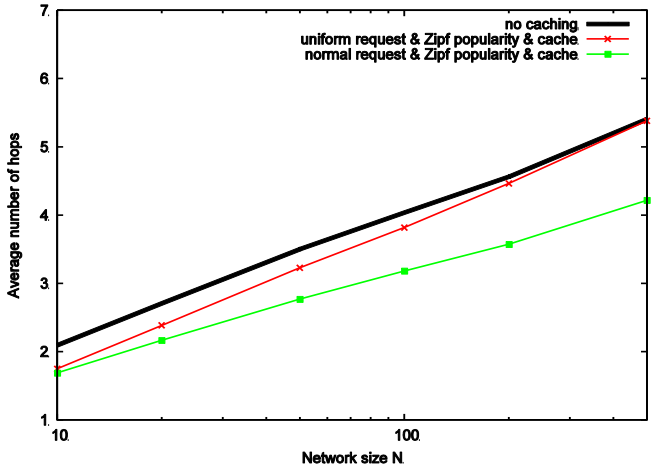

Figure A.2: Relation between the network size *N* and the average number of hops per lookup; for the situation no caching is used and when LFU is used for the uniform and Normal request distribution.

personal content item. However, for a relatively larger network the average number of hops for a lookup becomes significantly smaller when the Normal request distribution (i.e. locality exists in the distribution of requests) and the Zipf popularity distribution are used.

A.4.2 Request Times Distance caching algorithm

Although standard caching algorithms increase the efficiency of the system, for a PCSS a more optimal increase in efficiency can be obtained by a dedicated caching algorithm. Since we want to tackle the hotspot problem and reduce the average number of hops needed for a lookup, the caching algorithm we propose reacts on both *popularity* and *distance* of lookups. Intuitively, the popularity $p_{n,i}$ is measured by the total number of requests to a file i, initiated by node n. The distance $d_{n,i}$ of a personal file i is measured by the number of hops needed to obtain the lookup result from the requesting node n and the responsible node storing the file. In (3) the expression of the *Request Times Distance* (RTD) caching algorithm is provided.

$$I_{n,i} = p_{n,i} \times d_{n,i} \tag{3}$$

The references to personal content objects with the highest importance values for $I_{n,i}$ in (3), will be stored in the local cache of node n.

By using *cooperative caching*, introduced in Section A.4.3, we increase the performance of the caching algorithm. In order to make the algorithm more robust on sudden changes in personal content popularity, we extend it with a *sliding window*. The sliding window algorithm is explained in Section A.4.4.

A.4.3 Update protocol for cooperative caching

In order to be able to update finger tables when nodes suddenly join or leave the (Chord) DHT, the predecessor and successor nodes have to be known by every node. We can increase the performance of the caching algorithm by keeping a local copy

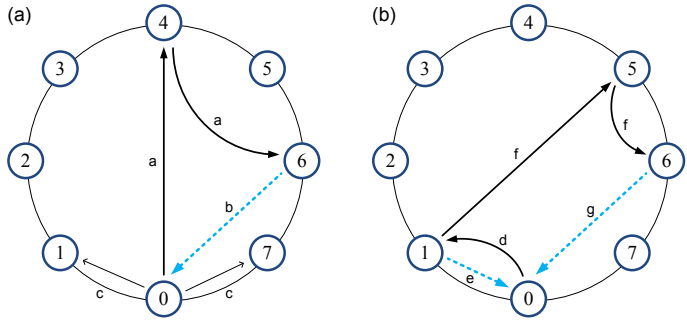

Figure A.3: Two scenarios for a lookup using cooperative caching. Scenario (a) describes the case where local copies of neighbor cache entries do not contain the search key. In scenario (b), one of the local copies of the neighbor's cache (should) contain(s) the search key.

of both neighbor's cached keys. This *cooperative caching* strategy utilizes the neighbor's caches to virtually increase the size of the local cache and allows to avoid storing the same copies of <*key*, *value*>-pairs that can be retrieved in only one hop.

Figure A.3 visualizes the *update protocol* of two possible scenarios for performing a lookup using cooperative caching. In both scenarios the destination node for the lookup is node 6 (i.e. the node responsible for storing the *values* belonging to the search *key*), the request is initiated from node 0 and the node numbers are used as the numeric identifier space. Figure A.3a considers the case that the local copies of the cache entries of the neighbor nodes do not contain the search key of file *i*. The scenario in Figure A.3b describes the case that the local copy of the cache entry of the neighbor node, in this case node 1, contains the search key.

When the local copy of the neighbor's cache does not contain the search key (Figure A.3a), the lookup is performed as usual. A request message (a) is routed via node 4 to node 6. Node 6 will respond the requested value using a reply message (b). In the case that node 0 decides to cache the lookup value, it updates the local cache table of both its neighbor nodes with the cache update message (c). These nodes then re-compute their values of the importance $I_{1,i}$ and $I_{7,i}$ of file *i*, as the distances $d_{1,i}$ and $d_{7,i}$ are now equal to one hop. No extra lookup delay is introduced by this update mechanism.

In the case that one (or both) of the local copies of the neighbor's caches contain the search key, the lookup request is routed to that neighbor node. In Figure A.3b the local copy in node 0 of the cache entries of node 1 has the search key, so the request message (d) is forwarded to node 1. When node 1 still has the value of the search key in its cache, it updates the popularity $p_{1,i}$ and responds the value using the reply message (e). In the situation that node 1 no longer caches the value of the search key, i.e. it very recently released the value and still has to send the corresponding cache update to its neighbors, the lookup is forwarded by node 1 as usual using the request message (f) via node 5 to node 6. Node 6 responds with the value of the search key, using the reply message (g). Similar to the scenario in Figure A.3a, node 0 decides whether or not to store the result in its cache by computing the importance $I_{0,i}$ of file *i* (with distance $d_{0,i} = 1$ if the entry is stored in its neighbor's cache). Only in the case when a neighbor is contacted erroneously because it very recently released the requested value, one extra hop is added to the lookup delay. In all other cases, no extra delay is introduced.

A.4.4 Sliding window

Changes in content popularity have a big influence on the performance of caching algorithms. By adding a sliding window, only the last T requests that arrived in a node are used to determine the popularity of the requested content. A larger value for T, the sliding window size, implies that more information is available to determine popularity relations, which increases the efficiency of the algorithm when the popularity distribution of the requests is stable. When content popularity changes frequently or new content is introduced often, using a relatively smaller value for T makes the caching algorithm more robust since changes are noticed more quickly.

A.5 Validation and evaluation

For the validation and evaluation we use the discrete-event simulator PlanetSim [17], which offers a framework to simulate large scale overlay services and networks, and is used in research projects all over the world in order to analyze overlay services, such as DHTs. The PlanetSim framework consists of three main extension layers, namely the network, overlay and application layer. In order to analyze the caching algorithms, we have extended PlanetSim at the application layer with a lookup service that uses a caching strategy. An advantage of PlanetSim is that it already has an implementation of the DHT lookup protocol Chord [2].

Each simulation run is initialized by inserting <*key*, *value*>-pairs into the DHT network, which implies that every object is initially stored on only one node. Then search queries enter the simulation according to the popularity distribution (1) and are distributed over the nodes using the Normal distribution (3). For all figures we take averages based on ten independent simulation runs and each simulation run stops when the network reaches a so-called non-equilibrium steady state.

We make a comparison between the basic versions of the caching algorithms (i.e. without cooperative caching and sliding windows) in Section A.5.1. In Section A.5.2 an evaluation of the caching algorithms is made when using cooperative caching. The results of the algorithms when using a sliding window are provided in Section A.5.3, and finally, we present the alleviation of the hotspot problem in Section A.5.4.

A.5.1 Comparison of RTD to standard caching algorithms

In Figure A.4 and Figure A.5, results are shown for different caching algorithms for the average number of hops for a lookup in relation with network and cache size. For the simulations of both figures we use the same parameters as for Figure A.2

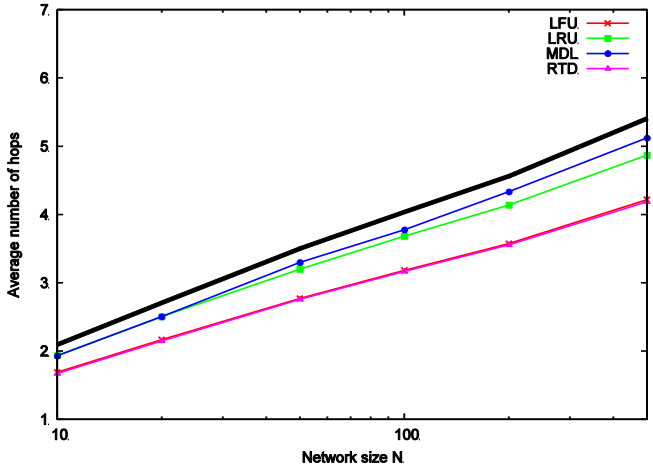

Figure A.4: Average number of hops in relation to the network size *N*, comparing RTD to the caching strategies LFU, LRU and MDL.

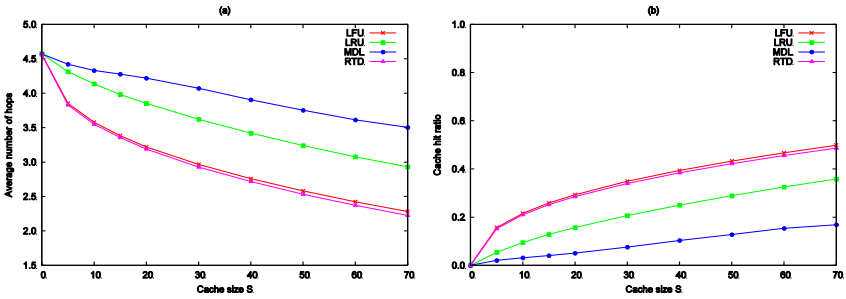

Figure A.5: Average number of hops per lookup (a) and cache hit ratio (b) in relation to the cache size for the basic algorithms.

(i.e. cache size S is 10, network size N is 200, σ is 2.0, α is 0.6 and C is $50 \times N$). The results compare the RTD algorithm to the standard caching algorithms.

For each of the caching strategies shown in Figure A.4, the $O(\log N)$ complexity with the network size N for a lookup still holds. The best optimization is provided by the RTD caching algorithm, with LFU showing comparable results. RTD only requires 78% of the average number of hops per lookup in the situation no caching is used (depicted with the solid line, not shown in the legend), when the personal content references are replicated for 20% in the network.

In Figure A.5 the average number of hops for a lookup (a) and the cache hit ratio (b) is plotted in relation to the cache size, for each of the caching algorithms. The best performance measured is provided by the RTD caching algorithm. When the cache size increases, the average number of hops needed for a lookup is lower and the cache hit ratio is higher.

For a larger cache size, the improvement of RTD compared to a standard caching algorithm becomes more noticeable in Figure A.5. The improvement of RTD compared to LFU for a cache size S of 70 entries, for the average number of hops per lookup and for the cache hit ratio, is 2%. Although this is not a huge improvement, it shows that the RTD algorithm is able to use extra cache space more efficiently than the standard caching algorithms, even without cooperation.

A.5.2 Comparison of RTD to standard caching algorithms, with cooperative caching

In order to increase the performance of the caching algorithms, we enable cooperation between nodes. By exchanging the cache entries with both neighbor nodes (i.e. successor and predecessor node), utilization of the limited cache space can be optimized. In this way the caching algorithm can avoid storing the same copies, which can be retrieved in only one hop. Figure A.6 illustrates the average number of hops needed for a lookup (a) and the cache hit ratio (b) in relation to the cache size for each algorithm using cooperative caching.

The cooperative caching algorithms in Figure A.6 show the same behavior and relationships as the basic caching algorithms (see Figure A.5). However, the performance surplus of the RTDc algorithm is clearly noticeable, especially when the cache size increases. For the average number of hops per lookup and the cache

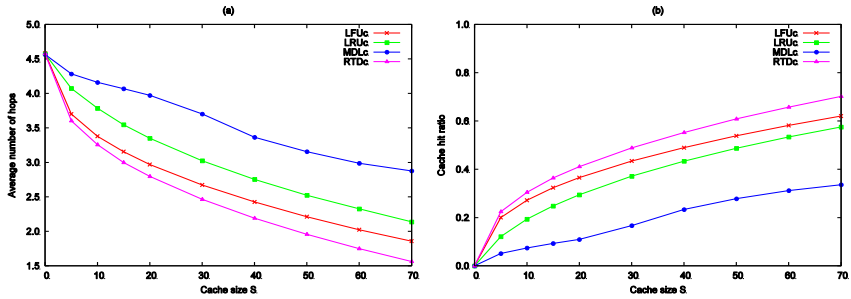

Figure A.6: Average number of hops per lookup (a) and cache hit ratio (b) in relation to the cache size for the cooperative algorithms.

hit ratio the performance increase is up to 16% and 13% respectively, compared with the results of LFUc.

To illustrate that the cooperative RTD caching algorithm uses the extra (neighbor's) cache space more efficiently, Figure A.7 depicts the number of cache duplicates *D* between neighboring nodes against the fraction of nodes having *D* duplicates. The cache size *S* used in this simulation is 10.

In Figure A.7 the scenarios of non-cooperative caching and cooperative caching for the LFU and RTD algorithm are shown. The LFU, RTD and the LFUc caching algorithm show roughly the same distribution for the number of cache duplicates. This means that enabling cooperative caching does not improve the way cache space is used for the LFU caching algorithm. However, the RTD and RTDc caching algorithm show a significant change of the distribution for the number of cache duplicates. The cooperative RTD caching algorithm uses the available cache space more efficiently, since it avoids storing the same copies that can be retrieved in one

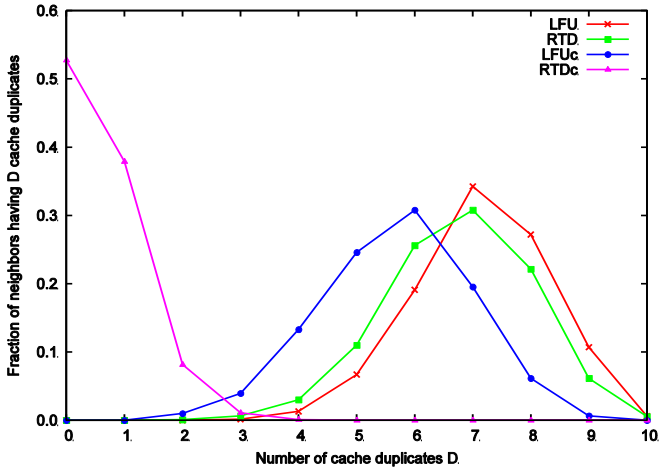

Figure A.7: Number of cache duplicates between neighbors is expressed against the fraction of nodes in the DHT network, for both the non-cooperative as the cooperative version of the LFU and RTD caching algorithm.

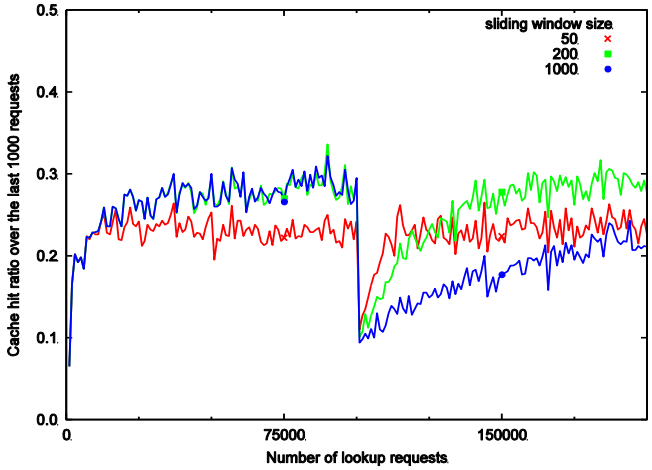

Figure A.8: Cache hit ratio over the last 1,000 lookup requests is plotted in relation with the total number of lookup requests. The sliding window is set to 50, 200 and 1,000 requests and after 100,000 requests (i.e. on average 500 per node) the popularity of the personal content objects is reshuffled.

hop.

A.5.3 Sliding window

By using a sliding window in the caching algorithm, the system is able to efficiently react on changes in content popularity. This effect is made visible in Figure A.8, for different sliding window sizes. Again, we use the same simulation setup as in Figure A.2 (i.e. $S = 10$, $N = 200$, $\sigma = 2.0$, $\alpha = 0.6$ and $C = 50 \times N$). After 100,000 lookup requests (i.e. on average 500 per node), each personal content item receives a new random popularity rank, while keeping the same value of α for the overall popularity distribution. In Figure A.8 the results of one simulation run, with the cooperative RTD caching algorithm, are presented for a sliding window size T of 50, 200 and 1,000 requests. The cache hit ratio over the last 1,000 requests is plotted against the total number of lookup requests.

Figure A.8 clearly shows that a larger sliding window size provides a higher cache hit ratio, when the popularity distribution of the personal content remains unchanged. However, the time (measured in total lookup requests) it takes to recover from a sudden change of the popularity distribution is significantly lower for a relatively small sliding window size.

A.5.4 Hotspots

Besides a reduction in average lookup latency (measured in hops), the caching algorithm also decreases the hotspot problem. In Figure A.9 the fraction of incoming lookup requests at each node is plotted ranked by decreasing load for caches with size zero (i.e. no caching is used), ten and fifty entries, where the cooperative RTD

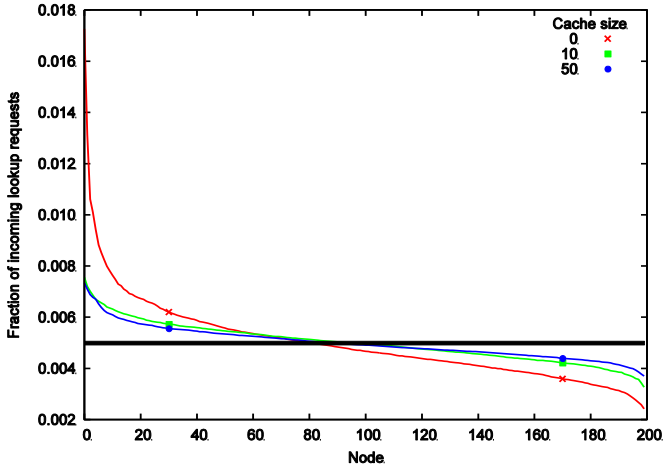

Figure A.9: Fraction of incoming lookup requests visualized for each node in the DHT. The size S of the cache per node is 0, 10, or 50 entries.

caching strategy is used. We use the same simulation setup as in Figure A.2 (i.e. $N = 200$, $\sigma = 2.0$, $\alpha = 0.6$ and $C = 50 \times N$) and since no change in popularity of personal content occurs, the sliding window size T is set to infinite.

Since the references of popular personal content are stored on multiple nodes in the network, the request load is efficiently spread over multiple nodes, even with very low cache sizes. In the optimal situation each node would receive the same fraction of incoming lookup requests $1/N$ (i.e. 0.005), depicted by the horizontal line. In the case a cache size of 10 entries per node is used, the load imbalance decreases with 4.8 times on the node having the highest number of incoming lookup requests (i.e. node 0). The load on this node is 51% higher than in the optimal situation.

A.6 Conclusion and future work

In order to successfully deploy a *Personal Content Storage Service* (PCSS), it has to provide storage space to end-users *transparently*, with *small access times*, and available at *any place* and at *any time*. One of the main features of a PCSS is the ability to search through the dataset of personal files. To optimize searching times in a PCSS, we introduced a caching solution on a *Distributed Hash Table* (DHT). The scalability of a DHT is increased by using the *Requests Times Distance* (RTD) caching algorithm.

We show that the RTD caching algorithm is more efficient than standard caching algorithms, such as Least Frequently Used (LFU), Least Recently Used and Most Distant Lookup (MDL), especially for relatively high cache sizes. To further increase the performance of the caching algorithm we use *cooperative caching*. When neighboring nodes on the DHT work together, the RTD algorithm is able to utilize neighbor's caches efficiently and show a performance increase of up to 16% and 13% compared to LFU for respectively the cache hit ratio and average number of hops per lookup. In this way the RTD caching algorithm avoids storing the same

copies on nodes, if it can be retrieved in only one hop. In order to quickly react to sudden changes in content popularity, we extend the caching algorithm with a *sliding window*, where a size of a few hundred requests is sufficient. Besides a reduction in access times (measured in number of hops) to the references of personal content, the RTD caching algorithm also reduces the hotspot problem. The fraction of incoming lookup requests per node is balanced more evenly and even for a cache size of 10 entries per node, the number of incoming lookup requests is reduced 4.8 times.

Although the proposed solution optimizes the scalable lookup in a DHT, it can only be used for lookup when the exact name of the *key* is known. This deterministic search property possesses limitations on the suitability of using a DHT for a PCSS. However, the performance of any existing DHT-based framework offering multiple keyword and range queries can already be increased by the proposed framework. Nevertheless, we plan for further research to focus on optimizing DHTs by enabling multiple keywords and range query searches, since currently no solution exists that fulfills all needs for a PCSS.

An issue not addressed in this chapter is that by reducing the time it takes to obtain content locations does not imply that the actual content itself can be accessed quickly. Therefore, we plan to investigate caching/replication algorithms for personal content itself, in order to allow fast access of personal content by using a PCSS.

In this chapter we use synthetic models to generate the workload, however real workload traces are more preferable to evaluate the caching algorithm. Therefore we plan to set up a more realistic content popularity and workload distribution in collaboration with Netlog[13], a popular European community website located in Belgium.

[13] http://www.netlog.com/

References

[1] N. Sluijs, T. Wauters, B. Dhoedt, and F. De Turck, Optimized Search in Distributed Personal Content Systems, *9e UGent-FirW Doctoraatssymposium,* 2008, pp. 186-187.

[2] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, Chord: A scalable peer-to-peer lookup protocol for Internet applications, IEEE-ACM Transactions on Networking, Vol: 11, no: 1, 2003, pp. 17-32.

[3] A.I.T. Rowstron and P. Druschel, Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, Lecture Notes in Computer Science: Middleware 2001*,* Vol: 2218/2001, 2001, pp. 329-350.

[4] C. Wang, L. Xiao, Y.H. Liu, and P. Zheng, DiCAS: an efficient distributed caching mechanism for P2P systems, IEEE Transactions on Parallel and Distributed Systems, Vol: 17, no: 10, 2006, pp. 1097-1109.

[5] V. Ramasubramanian and E.G. Sirer, Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays, *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation (NSDI2004),* 2004, pp. 1-14.

[6] A.I.T. Rowstron and P. Druschel, Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility, *Proceedings of the 18th ACM Symposium on Operating Systems Principles,* 2001, pp. 188-201.

[7] B. Bhattacharjee, S. Chawathe, V. Gopalakrishnan, P. Keleher, and B. Silaghi, Efficient peer-to-peer searches using result-caching, Lecture Notes in Computer Science: Peer-to-Peer Systems II*,* Vol: 2735/2003, 2003, pp. 225-236.

[8] J. Liu and J. Xu, Proxy caching for media streaming over the Internet, IEEE Communications Magazine, Vol: 42, no: 8, 2004, pp. 88-94.

[9] J. Kangasharju, J. Roberts, and K.W. Ross, Object replication strategies in content distribution networks, Computer Communications, Vol: 25, no: 4, 2002, pp. 376-383.

[10] T. Wauters, J. Coppens, B. Dhoedt, and P. Demeester, Load balancing through efficient distributed content placement, *Next Generation Internet Networks,* 2005, pp. 99-105.

[11] Y. Chae, K. Guo, M.M. Buddhikot, S. Suri, and E.W. Zegura, Silo, rainbow, and caching token: schemes for scalable, fault tolerant stream caching, IEEE Journal on Selected Areas in Communications, Vol: 20, no: 7, 2002, pp. 1328-1344.

[12] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, Web Caching and Zipf-like Distributions: Evidence and Implications, *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99),* 1999, pp. 126-134.

[13] P. Backx, T. Wauters, B. Dhoedt, and P. Demeester, A comparison of peer-to-peer architectures, *Conference Proceedings of Eurescom 2002 Powerful Networks for Profitable Services,* 2002, pp. 1-8.

[14] F. Duarte, F. Benevenuto, V. Almeida, and J. Almeida, Geographical Characterization of YouTube: a Latin American View, *Latin American Web Conference (LA-WEB 2007),* 2007, pp. 13-21.

[15] W. Wu, Y. Chen, X. Zhang, X. Shi, L. Cong, B. Deng, and X. Li, LDHT: Locality-aware Distributed Hash Tables, *International Conference on Information Networking (ICOIN 2008),* 2008, pp. 1-5.

[16] M. Castro, P. Druschel, Y. Hu, and A. Rowstron, Topology-Aware Routing in Structured Peer-to-Peer Overlay Networks, Future Directions in Distributed Computing: Lecture Notes in Computer Science*,* 2003, pp. 103-107.

[17] Pujol Ahulló, Jordi, García López, Pedro, Sànchez Artigas, Marc, Arrufat Arias, Marcel, París Aixalà, Gerard, and Bruchmann, Max, PlanetSim: An extensible framework for overlay network and services simulations, Architecture and Telematic Services Research Group, Universitat Rovira i Vigili, Tarragona, Spain, DEIM-RR-08-002, 2008.