Pós-Graduação em Ciência da Computação

**“AN ADAPTABLE STORAGE SLICING
ALGORITHM FOR CONTENT DELIVERY
NETWORKS”**

Por

ANDRÉ LUIS CAVALCANTI MOREIRA

Tese de Doutorado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE/2015

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANDRÉ LUIS CAVALCANTI MOREIRA

“AN ADAPTABLE STORAGE SLICING ALGORITHM FOR CONTENT DELIVERY NETWORKS”

*ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA
UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO
PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA
COMPUTAÇÃO.*

ORIENTADOR: Djamel Fawzi Hadj. Sadok
COORIENTADOR: Judith Kelner

RECIFE, AGOSTO/2015

Catálogo na fonte
Bibliotecária Jane Souto Maior, CRB4-571

M838a Moreira, André Luis Cavalcanti
An adaptable storage slicing algorithm for content delivery
networks / André Luis Cavalcanti Moreira – Recife: O Autor, 2015.
183 p.: il., fig., tab.

Orientador: Djamel Fawzi Hadj Sadok.
Tese (Doutorado) – Universidade Federal de Pernambuco.
CIn, Ciência da Computação, 2015.
Inclui referências e apêndices.

1. Redes de computadores. 2. Sistemas distribuídos. 3.
Arquitetura de redes de computadores. 4. Composição de
serviços. I. Sadok, Djamel Fawzi Hadj (orientador). II. Título.

004.6

CDD (23. ed.)

UFPE- MEI 2016-005

Tese de Doutorado apresentada por **André Luis Cavalcanti Moreira** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título “**AN ADAPTABLE STORAGE SLICING ALGORITHM FOR CONTENT DELIVERY NETWORKS**” orientada pelo **Prof. Djamel Fawzi Hadj Sadok** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Paulo Romero Martins Maciel
Centro de Informática

Prof. Ricardo Massa Ferreira Lima
Centro de Informática / UFPE

Profa. Rossana Maria de Castro Andrade
Departamento de Computação / UFC

Prof. Eduardo James Pereira Souto
Departamento de Ciência da Computação / UFAM

Prof. Glaucio Estácio Gonçalves
Departamento de Estatística e Informática / UFRPE

Visto e permitida a impressão.
Recife, 28 de agosto de 2015.

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

To my wife and
parents.

Acknowledgments

First and foremost, I would like to thank to God for his guidance, faithfulness and wisdom, granting me the chance and ability to complete this work.

I also would like to thank my supervisor Prof Djamel Sadok who gave me this opportunity to work in this fascinating area of computer networks. I would like to express sincere gratitude to Professor Judith Kelner for the support, without which, I would not be finished.

I also would like to thank all the staff at the Networking and Telecommunications Research Group for the discussions, help and companion, making this research much more enjoyable: Moisés Rodrigues, Arthur Callado, Ernani Azevedo; all GPRT people and support team (in special Manuela Melo, Ana Carolina and Rodrigo Germano).

I also would like to thank Ericsson Research, the sponsor of the project that motivated this work.

Finally, I would like to thank my parents, Luis Alves and Maria das Neves, for their continual encouragement to my studies. And to my wife, Érica, for her understanding and support.

Abstract

Several works study the performance of Content Delivery Networks (CDNs) under various network infrastructure and demand conditions. Many strategies have been proposed to deal with aspects inherent to the CDN distribution model. Though mostly very effective, a traditional CDN approach of statically positioned elements often fails to meet quality of experience (QoE) requirements when network conditions suddenly change. CDN adaptation is a key feature in this process and some studies go even further and try to also deal with demand elasticity by providing an elastic infrastructure (cloud computing) to such CDNs. Each Content Provider (CP) gets served only the amount of storage space and network throughput that it needs and pays only for what has been used. Some IaaS providers offer simple CDN services on top of their infrastructure. However, in general, there is a lack of PaaS tools to create rapidly a CDN. There is no standard or open source software able to deliver CDN as a service for each tenant through well-known managers. A PaaS CDN should be able to implement content delivery service in a cloud environment, provision and orchestrate each tenant, monitor usage and make decisions on planning and dimensioning of resources. This work introduces a framework for the allocation of resources of a CDN in a multi-tenant environment. The framework is able to provision and orchestrate multi-tenant virtual CDNs and can be seen as a step towards a PaaS CDN. A simple dot product based module for network change detection is presented and a more elaborate multi-tenant resource manager model is defined. We solve the resulting ILP problem using both branch and bound as well as an efficient cache slicing algorithm that employs a three phase heuristic for orchestration of multi-tenant virtual CDNs. We finally show that a distributed algorithm with limited local information may be also offer reasonable resource allocation while using limited coordination among the different nodes. A self-organization behavior emerges when some of the nodes reach consensus.

Keywords: dynamic CDN, storage slicing, self-adaptation, PaaS CDN.

Resumo

Vários trabalhos estudam o desempenho de Redes de Distribuição de Conteúdo (CDN) em diferentes condições e demanda e de infraestrutura. Muitas estratégias têm sido propostas para lidar com aspectos inerentes ao modelo de distribuição de CDN. Embora essas técnicas sejam bastante eficazes, uma abordagem tradicional de elementos estaticamente posicionados numa CDN muitas vezes não consegue atender os requisitos de qualidade de experiência (QoE) quando as condições da rede mudam repentinamente. Adaptação CDN é uma característica fundamental neste processo e alguns estudos vão ainda mais longe e tentam lidar com a elasticidade da demanda, proporcionando uma infraestrutura elástica (computação em nuvem) para a CDN. Cada provedor de conteúdo obtém apenas a quantidade de armazenamento e de rede necessários, pagando apenas pelo efetivo uso. Alguns provedores IaaS oferecem serviços de CDN sobre suas estruturas. No entanto, em geral, não existe padrão ou *softwares* de código aberto capazes de entregar serviços de CDN por meio de gerenciadores. Uma CDN PaaS deve ser capaz de fornecer um serviço de entrega de conteúdo em um ambiente de nuvem, provisionar e orquestrar cada *tenant*, monitorar uso e tomar decisões de planejamento e dimensionamento de recursos. Este trabalho apresenta um framework para alocação de recursos de uma CDN em ambiente *multi-tenant*. O framework é capaz de provisionar e orquestrar CDNs virtuais e pode ser visto como um passo em direção a uma PaaS CDN. Um módulo baseado em simples produto escalar para detecção de mudanças na rede é apresentado, bem como um modelo mais elaborado de gerenciamento de recursos. Resolvemos o problema ILP resultante dessa abordagem por meio de um algoritmo de divisão de cache que emprega uma heurística em três fases para a orquestração de CDN virtuais. Por fim, mostramos uma outra abordagem com algoritmo distribuído que usa informação local e que também oferece uma alocação razoável usando coordenação limitada entre os diferentes nós. Um comportamento de auto-organização surge quando alguns desses nós chegam a um consenso.

Palavras-chave: CDN dinâmica, divisão do armazenamento, autoadaptação, PaaS CDN.

List of Figures

Figure 2.1 - A flash crowd example [94].....	27
Figure 2.2 - QoS/QoE layered model [199]	32
Figure 3.1 - An abstract view of a CDN.....	35
Figure 3.2 - General CDN architecture [128]	38
Figure 3.3 - Workflow of a CDN	38
Figure 4.1- Framework overview.....	61
Figure 4.2 – Change Manager Overview	64
Figure 4.3 – Dot product of baseline and current vectors.....	67
Figure 4.4 – Basic concept of the virtual manager.....	68
Figure 4.5 – Topology used in simulations	76
Figure 4.6 – Average number of requests and video startup delay.....	78
Figure 4.7 – Comparison among virtualization strategies (6 caches)	79
Figure 4.8 – Comparison among virtualization strategies (10 caches)	79
Figure 4.9 – Startup delay (6 caches).....	80
Figure 4.10 – Startup delay (10 caches)	80
Figure 4.11 – Overview of scenario	82
Figure 4.12 – Amount of request triggered per timestamp	83
Figure 4.13- Example of the vector extracted from the router x router RTDs.....	85
Figure 4.14 – Weigh functions: a) proposed improvement b) conventional approach.....	90
Figure 4.15- Collection begins when a complete set of time slices has elapsed	90
Figure 4.16- Data collection of single and sliding window approaches	91
Figure 4.17- Replacement occurrence in the two approaches.....	92
Figure 4.18- Compared total traffic.....	92
Figure 4.19- Compared cross network traffic.....	93
Figure 4.20- Compared inner network traffic.....	93
Figure 4.21- Comparison of startup delay of both approaches	94
Figure 4.22- Surrogate placement for all configurations along time	99
Figure 4.23- Total traffic comparison for the real metrics experiments	100
Figure 4.24- Cross traffic comparison for the real metrics experiments	101
Figure 4.25- Inner traffic comparison for the real metrics experiments.....	101
Figure 4.26- Mean and standard deviation for playback continuity	102
Figure 4.27- Request distribution for stretched scenario	103
Figure 4.28- Total network bandwidth usage along simulation time	104
Figure 4.29- Cross traffic bandwidth usage along simulation time	104
Figure 4.30- Inner traffic bandwidth usage along simulation time.....	104
Figure 4.31- Dot product along simulation time.....	105
Figure 4.32- Request distribution for shrunk scenario	106
Figure 4.33- Total network traffic along simulation time	107
Figure 4.34- Total cross traffic along simulation time.....	107
Figure 4.35- Mean startup delay for shrunk scenario	108
Figure 5.1- Principle of different vCDN cache nodes deployment [65].....	112
Figure 5.2 – Cache slicing process flowchart.....	123
Figure 5.3 – Cache slicing proportional strategy	126
Figure 5.4 – Cache slicing greedy strategy.....	127
Figure 5.5 – Cache slicing decision in greedy strategy.....	127
Figure 5.6 – Example of cache slicing configuration after phase 2.....	128
Figure 5.7 – Requests evolution during time	131
Figure 5.8 – Total network traffic progress	132

Figure 5.9 – Cross traffic progress	132
Figure 5.10 – Cache hit progress	133
Figure 5.11 – Startup delay progress	133
Figure 5.12 – Request evolution along time for scenario 1	134
Figure 5.13 – Cache hit progress for scenario 1	135
Figure 5.14 – Startup delay progress for scenario 1	135
Figure 5.15 – Total network traffic progress for scenario 1	136
Figure 5.16 – Cross traffic progress for scenario 1	136
Figure 5.17 – Evolution of slice distribution	137
Figure 5.18 – Competition of slices	138
Figure 5.19 – Request evolution along time for scenario 2	139
Figure 5.20 – Cache hit progress for scenario 2	140
Figure 5.21 – Startup delay evolution along time for scenario 2	141
Figure 5.22 – Total network traffic progress for scenario 2	141
Figure 5.23 – Cross traffic progress for scenario 2	142
Figure 5.24 – Cache hit progress for scenario 3	143
Figure 5.25 – Total network traffic progress for scenario 3	143
Figure 5.26 – Cross traffic progress for scenario 3	144
Figure 5.27 – Startup delay progress for scenario 3	144
Figure 5.28 – Playback continuity progress for scenario 3	145
Figure 5.29 – Decentralized approach for slicing framework	146
Figure 5.30 – Cache hit progress for autonomic evaluation	150
Figure 5.31 – Startup delay progress for autonomic evaluation	150
Figure 5.32 – Total traffic progress for autonomic evaluation	150
Figure 5.33 – Cross traffic progress for autonomic evaluation	151
Figure I.1 – Simulator GUI tool	170
Figure I.2 – Simulator architecture	171
Figure I.3 – Explicit definition of each AS	172
Figure I.4 – Single-AS Network topology	173
Figure I.5 – Memory usage comparison between ns-3 and P2PCDNSim	177
Figure I.6 – Congestion window behavior comparing ns-3 and P2PCDNSim	178
Figure I.7 – Memory usage consumption for 20 nodes topology scenario	179
Figure II.1 – Evaluation of surrogate placement strategies	183

List of Tables

Table 4.1 – CDN Virtualization scenario configuration parameters.....	77
Table 4.2 – Comparison among virtualization strategies	78
Table 4.3 – Common configuration parameters for case study.....	83
Table 4.4 – Configuration parameters for stretched scenario	103
Table 4.5 – Configuration parameters for shrunk scenario.....	106
Table 5.1 – Configuration parameters	131
Table 5.2 – Configuration parameters for scenario 1	134
Table 5.3 – Configuration parameters for scenario 2.....	140
Table 5.4 – Autonomic results for proof of concept	149
Table I.1 – Simulation time, download time and rate for ns-3 and P2PCDNSim basic 3-node scenario.....	178
Table I.2 – Simulation time, download time and rate for ns-3 and P2PCDNSim basic 20-node scenario.....	179
Table I.3 – Scenarios configuration.....	180
Table I.4 – Simulation time, download time and rate for ns-3 and P2PCDNSim basic 20-node scenario.....	181
Table II.1 Trace configuration.....	182

Contents

Abbreviations and Acronyms	13
1 Introduction	14
1.1 Motivation	15
1.2 Objectives and contributions.....	16
1.3 Organization of thesis.....	18
2 Content Distribution	20
2.1 Content Distribution Systems	21
2.1.1 Client/Server model	21
2.1.2 Multicast	22
2.1.3 Content Delivery Networks	22
2.1.4 Peer-to-Peer Distribution Networks	22
2.2 Challenges of content delivery	24
2.2.1 Content popularity.....	24
2.2.2 Flash crowds.....	26
2.2.3 Video distribution and QoE	29
3 Content delivery networks	34
3.1 Overview	34
3.2 CDN architecture.....	37
3.2.1 The Origin Server	39
3.2.2 The surrogate server.....	41
3.2.3 The Redirector.....	45
3.2.4 The Monitor.....	50
3.2.5 The Content Manager.....	51
3.2.6 Accounting/billing mechanism	53
3.3 Categories of CDN	54
3.3.1 Conventional CDN	54
3.3.2 Academic CDN.....	56
3.3.3 Peer-Assisted CDN	57
3.3.4 Telco-CDN.....	57
3.3.5 Federated CDN.....	58
3.3.6 Cloud-based CDN	59
3.3.7 Autonomic CDN	59
3.4 Concluding remarks	60
4 CDN change manager	61
4.1 Framework overview	61
4.2 Dynamic CDN.....	62
4.3 Related work	63
4.4 Detecting changes	63
4.4.1 Change manager description.....	64
4.4.2 Detection algorithm.....	65

4.5	The case for CDN Virtualization.....	68
4.5.1	Related work.....	70
4.5.2	Virtualized CDN Change Detection.....	72
4.5.3	Simulator	73
4.5.4	Evaluation	74
4.5.5	Discussion.....	80
4.6	Improvements and case study	81
4.6.1	Scenario description	82
4.7	Round-trip delay.....	83
4.7.1	Implementation.....	87
4.7.2	Evaluation and discussion	87
4.8	Weighed Moving Average vector.....	88
4.8.1	Implementation.....	90
4.8.2	Evaluation and discussion	91
4.9	Usage Approximation based on restricted local information	94
4.9.1	Implementation.....	97
4.9.2	Evaluation and discussion	98
4.10	Concluding remarks	108
5	The slicing framework	110
5.1	Multi-tenancy	110
5.2	CDN architecture considerations	113
5.3	Framework description.....	114
5.3.1	The cache slicing solution.....	116
5.3.2	Problem formulation.....	117
5.3.3	The algorithm	122
5.4	Evaluation and discussion.....	129
5.4.1	Proof of concept.....	129
5.4.2	Scenario 1	133
5.4.3	Scenario 2.....	138
5.4.4	Scenario 3 - Playback continuity.....	142
5.5	An Autonomic CDN Slicing approach.....	145
5.5.1	Slicing algorithm.....	147
5.5.2	Evaluation	149
5.6	Concluding remarks	151
6	Contributions and Future Works	153
6.1	Contributions of this work	153
6.2	Future works	155
	References	156
	Appendix I	169
	Appendix II	182

Abbreviations and Acronyms

AS	Autonomous System
CDN	Content Delivery Network
CP	Content Provider
IM	Instant Messaging
ISP	Internet Service Provider
NFV	Network Function Virtualization
P2P	Peer to Peer Network
SDN	Software Defined Network
SLA	Service Level Agreement
VoIP	Voice over IP
VPS	Virtual Private Server
VTN	Virtual Tenant Network

1 Introduction

Over the years, the Internet has experienced a massive growth in network traffic driven by rapid adoption of broadband access and high adoption of connected devices. According to an estimate from CISCO [37], the number mobile connected devices exceed the number of people on earth, and by 2019 there will be nearly 1.5 mobile devices per capita, 11.5 billion. There are a number of estimates, but they also fall generally in the eight to ten billion range. Internet connected devices include traditional computer, mobile devices, as well as the new industrial and consumer devices which form the so-called Internet of Things (IoT). CISCO also predicts there will be 25 billion devices connected to the Internet by the end of 2015 and as much as 50 billion by 2020. Morgan Stanley estimates that this number can be as high as 75 billion by 2020 [174].

Furthermore, the Internet has also witnessed an increase in system complexity and content richness. This complexity can be primarily attributed to the number of existing applications on the Internet today and the ways they interact with each other. The Internet is nowadays used for content acquisition, creation, management and distribution. These have been pushing the infrastructure to the limit especially when it comes to video delivery, which has a significant impact on network traffic. The popularity of Internet video services is growing fast, for a variety of user platforms, making them much more ubiquitous. Examples include user generated content through services like Youtube and streaming services like Netflix, Hulu, CNN and BBC.

Demand of multimedia content is hard to predict and provisioning of such services imposes specific requirements that often are difficult to be fulfilled in the Internet environment. Also, a number of wide-reaching events may cause a high demand for such services, a phenomenon usually called flash crowds. It is characterized by a sudden spike in content requests that causes a heavy workload on Internet servers. Coping with such unexpected huge traffic causes significant strain on servers that may become totally overwhelmed, resulting in service denial and temporary unavailability.

To overcome these inherent limitations of the Internet, Content Delivery Networks have emerged as a technology to provide infrastructure and techniques to deliver content in a scalable manner [26]. A CDN is an overlay distribution network that performs a transparent and efficient delivery of content to a large number of end-users, with the goal of enhancing the perceived quality of experience (QoE). According to a

recent CISCO forecast for Internet traffic [36], due to popularity of video streaming services, CDN will prevail as dominant technology for content delivery. In the same study, CISCO says that CDNs will carry 51% of all Internet traffic and 65% of all Internet video traffic in 2017.

1.1 Motivation

CDNs have been very successful in distributing content in the Internet. Large enterprises are usually customers of traditional and high-performance CDNs who operate extensive networks across the globe, such as Akamai [64], LimeLight [77] Networks and Mirror Image [124]. In this scenario, a small number of competitors responds for almost all CDN market. Akamai itself was responsible for near half of market share by the end of 2013 [50], if considered the top 10 websites. Thus, it is extremely difficult for new competitors to build and maintain a competing CDN infrastructure. On the other hand, the prices practiced by these CDNs are far from democratizing content delivery, especially for small to medium-sized enterprises (SMEs), government agencies, universities, and charities [92].

This type of CDN is also still typically deployed in a static manner and uses centralized redirection mechanisms. While powerful, its nature makes it prone to over- or under-provisioning of resources. A paradigm typically used nowadays to address this latter issue is adopting Cloud Computing principles, being able to provision just the right amount of resources based on current demand [61].

As a result, it is enticing to utilize cloud resources to build a so-called cloud-oriented CDN [141]. A cloud-oriented CDN can reduce the effort and cost of producing a solution for content delivery. It merely uses the cloud to map the CDN infrastructure elements into virtual architectural components in the cloud. For instance, a surrogate (or cache) server can be mapped to an IaaS storage and delivery service. The cloud computing paradigm has become a de facto standard for utility computing, a model where users can consume computational resources as a service and just pay for what they use. A cloud can be seen as a conceptual layer on the Internet, which turns transparent all available software and hardware resources and services making them accessible through a well-defined interface [27]. The heterogeneity and reusability of cloud components can be integrated to support new and powerful applications leveraging business value at a lower cost [122].

Cloud storage and infrastructure is a novel mode of service where providers supply storage capacities and computation services over multiple virtualized servers through the Internet. Thus, network companies can operate their Internet service on third party public data centers, and require computational resources and storage capacity on demand. In recent years, we also witnessed an emergence of storage cloud providers. Examples are Amazon S3, Nirvanix and Rackspace. Storage cloud providers operate data centers that can offer Internet-based content storage and delivery capabilities with the assurance of service uptime and end user perceived service quality [68].

These capabilities can bring two opportunities: The first one is to allow a small business to become a customer of multiple providers in different geo-locations and countries, hence building a dynamic overlay to opportunistically take advantage of low prices at specific locations and times. The second benefit is the underlying support for new CDN providers, offering content delivery service for third parties without the cost of owning or operating geographically distributed data centers. Examples of this approach are MetaCDN [92] and the studies on cloud-assisted CDNs [48].

The advantage of such model is that, unlike traditional CDNs, in a cloud environment, one has the potential to build an adaptable overlay with a topology that may be different from the underlying network. Nowadays, Internet content can be very diversified (web pages, images, video), it can also be static or dynamic, enjoy different popularities, geographic affinities, which may lead to different SLAs requirement and algorithms to cope with the delivery process.

In this context, a cloud-oriented CDN offers a model that considers diversity where each type of content or tenant can be handled by “virtual CDNs”, build on top of a cloud environment with just the necessary amount of resources, thus avoiding over-provisioning. More specifically, Virtual CDNs may leverage advances on the slicing of Cloud resources using the emerging SDN based Virtual Tenant Network (VTN) [65].

1.2 Objectives and contributions

A cloud-based CDN has the advantage of reducing the effort, time and cost of producing a complete or dedicated solution for content delivery. It facilitates the fast expansion of infrastructure by using on-demand computational and storage capabilities of a cloud environment. This on-demand resource acquisition can leverage a resizable virtual CDN

infrastructure, allowing a CDN to be adaptable. This is the main advantage of a cloud-based CDN: a dynamic CDN that can adapt itself to changes in demand and reconfigure itself. On the other hand, mapping virtual resources of a cloud environment to CDN components and deciding the amount and location of these resources while adapting to the user demand is not a trivial task.

Some IaaS providers also offer simple CDN services on top of their infrastructure, like Amazon EC2's Cloudfront and Windows Azure [145]. By integrating with other tools, it facilitates the deployment of a CDN. For instance, Azure announced the integration of Azure Media Services with Azure CDN, increasing the capacity of the former. It can be seen as a PaaS CDN, with a pay-a-you-go model.

However, in general, there is a lack of PaaS tools to create rapidly a CDN. There is no standard or open source software able to deliver CDN as a service for each tenant through well-known managers such as OpenStack [61]. Most players, as the examples above, use proprietary software and do not expose the whole management software. A PaaS CDN should be able to implement content delivery service in a cloud environment, provision and orchestrate each tenant, monitor usage and make decisions on planning and dimensioning of resources.

Inspired by this paradigm and requirements, we propose a framework for a “virtual CDN”, in which each type of content or content provider can be handled by a resizable and dynamic overlay build on top of a traditional infrastructure or a cloud environment. Like cloud-based CDNs, we study the potential to build an adaptable overlay with a topology that may be different from the underlying network. The framework is able to provision and orchestrate multi-tenant virtual CDNs and can be seen as a step towards a PaaS CDN.

The framework is supported by two main modules, which are also contributions of this work. The first one is an optimization algorithm to orchestrate the allocation of capacity for each tenant in this multi-tenant framework, facilitating the deployment of a virtual CDN, but without considering the underlying resources to be unlimited. This assumption allows this model to be used by small CDN operators, like telco-CDNs or mobile CDNs [196][147].

Also, this approach gives CDN providers the ability to better cope with granularity constraints of resizing the infrastructure, as in traditional cloud-based CDNs, where it is impossible to book resources in a very fine-grained way. For instance, Amazon EC2

provides a resource charging policy according to different kinds of CPU, memory and storage. Those types are fixed and provided as a list of available options without customization.

The second module provides an algorithm to monitor and detect relevant changes in CDN metrics in order to allow adaptation of the virtual CDN infrastructure. A dynamic CDN provider must be able to detect the need of allocation and freeing of resources to avoid the common problem of static deployment, namely, over or under provisioning. The latter may lead to an unbalanced setup where there could be too many nodes in an area with few requests or few nodes in an area of high demand. In other words, a dynamic CDN must detect the necessity of adaptation in infrastructure by dynamically adjusting the infrastructure for quick responsiveness to user demand changes.

Considering the challenges associated with the above issues, we delineate the following objectives of the present thesis:

- Compare CDN with other content delivery approaches and delineate the common problems in content distribution;
- Propose a mechanism for monitoring CDN resource utilization and detecting the need of adaptation in the context of a CDN virtualization;
- Propose a framework that enhances the virtual CDN concept, considering a plurality of content service providers, each one with its own content diversity, incorporated into an overall infrastructure;
- Propose an optimization algorithm to perform adaptation of allocation in a multi-tenant CDN provider maintaining user perceived Quality-of-Experience;
- Implement and test the proposed algorithms in a discreet-event CDN and P2P simulator.

1.3 Organization of thesis

This thesis presents a framework for multi-tenant virtual CDN aimed to facilitate its deployment through resource allocation of underlying infrastructure. It is structured in chapters that iteratively provide the concepts necessary to understand each topic. Thus, it has been organized as follows:

The second chapter focusses on presenting content distribution systems, including content delivery networks, and the main problems related to content distribution, which

will serve as basis the next chapters. Chapter 3 discusses the content delivery networks and offers a literature review on issues and challenges and also a categorization on existing CDNs. Chapter 4 introduces the proposed monitoring technique and detection algorithm, one of the contributions of this thesis that supports the proposed framework, and the model of CDN virtualization, used as basis for testing the changes detection technique. Chapter 5 presents the framework and the adaptable slicing technique, the main contributions of this thesis; an overview of cloud-based CDNs, the slicing solution, including the algorithm and results achieved. Finally, Chapter 6 concludes this work and presents some future plans.

2 Content Distribution

The Internet plays a key role in our modern society. It is a network of networks used globally by different people with different purposes. All the way from the small child playing video games with his friends to important financial transactions from international banks, a great variety of information travels through these networks. Its usage is increasing every day resulting in new and renovated challenges. To cope with these issues, several overlay networks were proposed and deployed each of them with a specific purpose, like CDN and P2P to distribute content. An example is Microsoft's Skype for IM and VoIP. Each overlay has its purpose but they share some basic characteristics.

Authors in [7] define overlay networks as a set of nodes connected by a network, maintaining a neighbor set of nodes with which they have connections. In [144] authors enumerate three basic emerging overlay networks: **Content Delivery Networks** (CDNs), routing overlays, and security or privacy overlays (e.g. VPN). A routing overlay exists to modify and supervise the path of data through a network. In other words, endpoints remain the same but the path between points A and B may change. This is done by overriding ISP routes, based on cost and operational efficiency, with other routes selected based on metrics directly related to an application. Last but not least, security or privacy networks, such as VPNs, are overlays focused on providing different forms of communication protection. In a way, they mirror CDN and routing overlays because they change the routing and can offer different ways of content delivery. A further good example of this kind of overlay is the Tor [175] network, a very popular network anonymizing tool and represents one of many paths to a new concept of web access, the so called deep web [21].

Although presenting considerable functional differences these overlays share singularities that can be pointed by looking at the definition of overlay networks. All of them are a set of nodes interconnected and cooperating for a common purpose. In addition, for them, analyzing those nodes is a key feature to increase network's efficiency. For instance, by analyzing a CDN's network node an operator could reallocate replica servers or content due to an unexpected flash crowd or due to network malfunction, a problem focused upon in this work. Similarly, a P2P network could reassign its super nodes to respond to flash events.

This chapter discusses some content distribution models and introduces the main overlay network technologies involved in such process.

2.1 Content Distribution Systems

2.1.1 *Client/Server model*

The simplest form of content distribution is based on the client/server model. It adopts a single site model where one server or multiple load balanced servers (but located at the same site) are responsible for serving the requested content. This approach has many drawbacks. First, there is the distance between end users and the infrastructure which impacts on network resource usage, latency and availability. There is potentially a significant bottleneck at the network holding the server(s) that can lead to a denial of service. Second, this centralized infrastructure may also turn highly costly and inefficient. The reasons include [13]:

- **Underutilization of infrastructure:** resource usage is highly variable. For instance, e-commerce traffic may show peak traffic levels 5 to 10 times higher than during low periods of the day [14]. A traditional solution requires a provisioning that can handle those peaks. This overprovisioning is inefficient because the infrastructure additional capacity is unused most of the time. Virtualization can address some of this inefficiency by allowing hardware sharing among different applications and services.
- **Capacity planning challenges:** are closely tied to the issue of infrastructure underutilization. The decision on infrastructure size investment remains complicated and often requires accurate prediction of resources utilization.
- **Cost of ownership:** beyond the traditional hardware costs, the total cost of ownership includes other components such as software, load balancers, power management, monitoring, support and administration, among others. According to IDC [87], the cost of maintenance and administration can reach up to 75% of the total cost.
- **Complexity of scaling:** scaling is not simply adding resources. It may require architectural changes to services or network infrastructure that can be costly. It also adds more administrative complexity, leading to a direct impact on cost. In addition, it is not an instantaneous process, therefore it may fail to respond in a suitable time frame to business evolving needs.

- **Availability:** a single site approach also imposes challenges on availability and reliability since it presents a single point of failure. For instance, when an outage takes place as a result from a cable cut.

Because of these issues, this model is used only on small sites that address little traffic, thus, it not suitable for current demands of the majority of Internet services nowadays.

2.1.2 Multicast

Authors in [162] propose the use of multicast as a content distribution mechanism to deliver live content. They consider two forms of multicast: the perfect multicast, where there is only one source placed in the network and all clients join a multicast group at the same time; and realistic multicast, where clients join the group according to an arrival process, which means that both group's membership and distribution tree can dynamically change.

Multicast has the advantage of overcoming the problem of network usage and server load suffered by the client/server model. According to [137], it is a natural enabler for carrying high bandwidth multimedia on IP networks. However, the deployment of such approach has not been widespread. The lack of appropriate network management tools for IP multicast has proven to be a major barrier in its deployment. In addition, it is not suited for contents other than live streaming, like VoD, for instance.

2.1.3 Content Delivery Networks

The content delivery network (CDN) is an overlay network built on top of the Internet, that spreads cache servers over various site locations to replicate content. A CDN solves most of the problems described in previous solutions and its architecture will be discussed in more depth in chapter 3.

2.1.4 Peer-to-Peer Distribution Networks

A peer-to-peer network may be seen as having a distributed architecture where participants share part of their own hardware resources to provide a set of services or network content. The peer shared resources are directly accessible to others without having to go through the control of intermediary entities [163]. For example, BitTorrent represents a popular form of a peer-to-peer content distribution network.

Peer-to-peer networks have a proven track as being very efficient in content distribution. However, some stringent requirements of specific applications, such as live VoD for instance, turn peer-to-peer distribution more challenging [109]. For example, BitTorrent enables peers to exchange any segment of the content being distributed; the order in which the segments arrive is not important. Such technique is not feasible for streaming applications. Further, given the timeliness requirements, streaming video applications typically must include techniques for graceful degradation of video quality rather than involving excessive delays.

In addition, Internet Service Providers (ISPs) are not very keen on using peer-to-peer technology [22]. The main problem has been the traffic it generates on their costly inter-ISP links. The peer selection mechanism usually does not consider the locality and the transit traffic caused by data sharing among internal and external nodes incurs in high costs for ISPs. Typically, only 8% of peering traffic remains within the ISP [161]. In order to reduce this kind of traffic, ISPs made attempts to adopt traffic shaping or even blocking P2P applications altogether.

Despite of these drawbacks, a recent project called Popcorn [146] has showed some success in delivering VoD content using the BitTorrent network. It uses BitTorrent file sharing network to download video content, but instead of waiting for download the entire file, it starts playing the content as soon as enough data is downloaded. Also, the downloading of segments was modified, since the order is important. However, Popcorn algorithm cannot be seen as a “solution” for P2P streaming. The quality of experience is highly dependent on the number of clients sharing the content and the time to start playing the video usually takes a high amount of time, in the range of three to five minutes. Some files cannot even be played, since it does not have enough peers sharing it. There is no commitment with quality of experience.

Furthermore, its apparent success is also due to the fact it takes advantage of the BitTorrent file sharing network that has a reasonable number of peers sharing content in traditional way, increasing the availability of segments. For instance, if the P2P Popcorn was a separate network, not connected to the regular torrent network, the segment availability would not be evenly distributed, degrading the playback video quality as the time gets away from the beginning. In other words, Popcorn depends on the existence of peers sharing the content without watching it, using another BitTorrent client.

2.2 Challenges of content delivery

The Internet is formed by thousands of autonomous systems, independently managed by different entities with different goals that compete with each other for resources. No single entity dominates Internet traffic, the largest one controls less than 5% of the total [149]. Such heterogeneous environment poses significant challenges to stringent requirements of Internet based services, like content distribution, especially when it comes to maximizing performance and the quality perceived by the end user.

For instance, Internet directly supported applications may face problems like outages, congestion, lack of scalability and security [149]. Outages are a potential problem on the Internet that can be caused by a wide ranging number of factors such as misconfigured core routers, DDoS attacks, cable cuts, power disruption, among others [159]. Congestion takes place when the capacity of routers or links is insufficient to meet traffic demand, leading to packet loss. Nowadays, most of Internet applications, like video streaming, are very susceptible to packet loss, resulting in freezing video playback at the client. Provisioning services is also very tricky, especially when it comes to meeting the demand of users during unexpected peaks. Also, protection against attacks, like DDoS is necessary. A report from Arbor Networks found that the average size of a DDoS was 1.77 Gbps in the first quarter of 2013, an increase of 20% in comparison with a year ago [160].

Slow adoption of new Internet technology is also a major problem. For instance, the IPv6, first proposed in 1998, only supports around 1% of Internet traffic 15 years later [56]. Similar a new TCP variant takes around 10 years to be adopted. High investments and complexity of business relations are barriers for rapid adoption of new technologies. This can be seen as a natural consequence of the decentralized architecture of the Internet. An overlay network, as a content delivery network or a peer-to-peer network, may overcome most of these limitations, but faces some problems of its own. Next sections will discuss the main issues of content distribution.

2.2.1 Content popularity

A content delivery provider needs to intelligently manage its content ensuring, for example, that for any content object, there is always at least one copy kept by its distributed servers. However, not all content is equal; some are more popular and sought after than other at a given point in time. Experience has shown that content popularity is

an important criterion in the selection of what content to cache (store) within a CDN. Video popularity, for example, has been shown to follow a Power Law distribution such as the Zipfian distribution [78].

To satisfy those requests for unusual content, a different provider may decide to adopt a different business model whereby it specializes in offering content that is usually little requested and caches objects at the tail of the above distribution, i.e. low popularity ones. There are many decisions regarding the caching of objects to be made, in addition to deciding cache size and locations. These include the choice of a suitable cache replacement policy that decides when should objects enter or leave a cache space. The right choice has a direct impact on two main quality metrics: the **cache hit ratio** and the **startup time**. The cache hit ratio is important as it contributes to content locality (often at the Edge Network) hence saving on backbone and inter ISP traffic. When a CDN cannot locate a requested content, it must then obtain this from the originator of such content. The use of surrogates attempts to minimize such operation. The startup time quality metric is important in the case of real-time services such as streaming video distribution. In these cases, CDNs usually divide multimedia objects into units known as segments and cache, or pre-fetch [6] the initial ones preferably to shorten startup delay.

Segmentation of content and partial caching is necessary because the full object approach is not feasible due to large amount of storage needed by some content types, specially video. This approach is useful as parts of the content may be accessed more often, then, it is natural to prioritize such parts and store only those, improving cache efficiency. Typical approaches of partial caching include prefix caching [156], uniform segmentation [147] and exponential segmentation [97]. Prefix caching always caches the prefix of objects to reduce client perceived startup latency because content can immediately be served by the proxy. In uniform segmentation, the objects are divided in uniform segments and each one is treated like a unique object. The exponential segmentation is similar, but the segment sizes increase exponentially.

Authors in [165] have proposed and adaptive and lazy segmentation, to adapt user access behaviors and lazily segment objects as late as possible. The scheme also consists of an aggressive admission policy. When the object is first accessed, it is fully cached and is kept in the proxy until an eviction policy is executed. Then, during the eviction process, the object is segmented, and only the segments, later defined, are evicted. The

segmentation strategy is similar to the exponential segmentation, with the specifics of being defined a posteriori, during the cache eviction.

Another approach is the hierarchical caching [136][181][186]. It was first proposed in the context of the Harvest [1] project, where a series of caches are arranged hierarchically in a tree-like structure. With hierarchical caching, caches are placed at different network levels. At the bottom level of the hierarchy there are the client caches. When a request is not satisfied by the client cache, the request is redirected to the institutional cache. If the document is not found at the institutional level, the request is then forwarded to the regional cache which in turn forwards unsatisfied requests to the national cache. Hierarchical caching is often combined with partial caching, in which different portions of content are stored at different levels. The first cache, for instance, can store the first or base segment of the content [89].

In recent years, with social-networking sites that carry user-generated content such as photos, and videos, the amount of unpopular cold content on the Internet has exploded. For such types of content, providing origin offload can be challenging since there is a large footprint of objects that are each accessed only a few times. Maintaining copies of such content in caches is prohibitive because of the big amount of storage needed for marginal increase in hit ratio. On the other hand, if all requests for such content are redirected to the origin, users may face outages due to congestion and denial of service, especially when the popularity changes, an inherent problem of traditional Internet architecture.

2.2.2 *Flash crowds*

A flash crowd is a sudden increase in the request rate that can deplete a content distribution system due to congestion or denial of service. It is becoming more frequent since the rapid spread of Internet capable devices, like smartphones, which makes information access more ubiquitous and demanding. The number of Internet-connected devices reached 8.7 billion in 2012 [71]. Because of a usually highly volatile content and short lifespan, an over-provisioning of the CDN is inefficient and uneconomical. The traditional CDN performs well with a relative constant workload, but the unanticipated surge in traffic provoked by a flash crowd renders traditional techniques ineffective at best. Figure 2.1 depicts an example of a flash crowd event.

According to [26], the term “flash crowd” was coined in 1973 by a science fiction writer Larry Niven in his short novel *Flash Crowd*, "where cheap and easy teleportation

enabled tens of thousands of people worldwide to flock to the scene of anything interesting almost instantly, incurring disorder and confusion". The term was then applied to the phenomena observed on the Internet. When a site catches attention of a large number of people, it gets an overwhelming and unexpected surge in traffic, usually causing malfunction and network saturation, which makes it temporarily unreachable. This phenomenon is also referred to as slash dot effect [5] and web hotspot [198].

Figure 2.1 - A flash crowd example [94]

There is no model for a flash crowd request pattern. Although it is possible to sometimes predict its occurrence (for instance, a planned launch of new content, a sports event), it is extremely difficult to predict the demand, as several well planned events could attest [23]. In addition, even when it is possible to have prior knowledge that a surge in demand will be experienced, there may not be enough time or resources to react and provision for it. Both predictable and unpredictable flash events can thus pose a serious risk.

A flash crowd event is in some aspects similar to DDoS attacks, but it has an important distinction. A DDoS attack "is characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service" [34]. On the other hand, a flash crowd is a result of a surge of legitimate accesses. However, they both result in similar consequences to the CDN provider, and there is therefore a need to distinguish these in order to avoid a denial for requests from legitimate users. Authors in [94] differentiate both phenomena in the following dimensions:

1. Traffic pattern: per-client request rate during an attack is stable and deviates significantly from normal. Besides, the distribution of files (which may not exist in the server) are unlikely to be zipfian.
2. Client characteristics: understanding client behavioral characteristics can help identify malicious attackers who intentionally stress a server pretending to be legitimate clients. In an attack, client distribution across ISPs and networks does not follow a population distribution. Cluster overlap before and during an attack is very small. Client clustering allows one to aggregate individual clients into groups belonging to the same administrative domain, the various client IP addresses are grouped into clusters based on longest prefix matching.
3. File reference characteristics: could potentially provide additional differentiators to rule out suspicious activities. For instance, the World Cup Web site study [119] presented a peak workload analysis pointing out that file referencing is more concentrated on a few extremely popular pages and that many clients repeatedly visited the site with shorter inter-session time.

Using the above dimensions, some research efforts traced further characteristics of a flash crowd [26]: 1. the increase in request rate is dramatic, but relatively short in duration; 2. the increase is not instantaneous (for instance, in September, 11th event, a massive load on the CNN Web site doubled every 7 minutes, until it reached 20 times the normal load) allowing a certain time to the CDN to adapt itself; 3. network bandwidth is the primary bottleneck. CPU may be a bottleneck if the server is dynamically generating content. Thus, the focus should be on alleviating the network consumption around servers; 4. Less than 10% of the contents is responsible for a large percentage, more than 90% of requests. This small world like behavior allows the CDN to maximize the cache-hit ratio since a small number of cached content may suffice to serve most user requests. 5. the popularity still follows the zipf-like distribution; 6. more than 60% of contents are only accessed during the flash crowd. Thus, after a flash crowd, most of these objects can be evicted from the caches; and 7. the number of clients in a flash crowd is commensurate with the request rate. The spikes in requested volumes correspond with the spikes in the number of clients accessing the site. This behavior is useful to determine whether the increase in request rate is a result of DDoS attacks.

The ability to handle flash crowds is an important aspect of CDN design. Chapter 4 presents a technique that detects and deals effectively with changes in traffic behavior, providing a dynamic adaptation of CDN capable of handling flash crowds events.

2.2.3 Video distribution and QoE

Videos-on-demand (VoD), also called video streaming services are already prevalent nowadays. VoD offers an easier, cheaper, faster, and more comfortable way to watch movies. Also, the video quality has been increasing to satisfy customers by adopting new high definition resolutions. Such popularity can be demonstrated, for instance, with the YouTube site reaching more than 1 billion visitors each month where over 6 billion hours of video (almost one hour for every person on Earth) are watched. In other words, each minute 100 hours of video are uploaded [197]. Other successful cases include services from CNN [38], BBC [20], Google and Netflix [132].

Video streaming belongs to the class of IPTV services that also include Live Streaming and Voice over IP (VoIP). IPTV is defined by ITU-T as "multimedia services such as television/video/audio/text/graphics/data delivered over IP based networks managed to provide the required level of QoS/QoE, security, interactivity and reliability" [105]. It is expected to be "the killer application for the next-generation Internet" [187]. Interactive multimedia services and media-rich applications are forecasted to be widely used in the near future [109].

As seen earlier, VoD is already a success case, but, despite the high popularity of such services, video distribution on the Internet still poses new specific challenges. Particularly, the provisioning of VoD suffers from serious limitations, mostly because of typical issues that must be fulfilled in the Internet environment, such as a low packet loss, end-to-end delay and network congestion. To overcome these issues, extensive studies have been made to improve user experience while considering several aspects of a VoD system, such as start-up delay, playback continuity and responsiveness.

2.2.3.1 Requirements

The authors in [49] describe various requirements of streaming applications and propose some techniques to tackle them. These are summarized as follow:

1. Bandwidth: video on demand and IPTV applications have a minimum required bandwidth in order to achieve an acceptable quality. But the Internet does not provide support for bandwidth reservation. Also, VoD services must employ

congestion control to avoid congestion when network is heavily loaded. A technique to deal with this problem is rate control [49]. It is used to determine the sending rate of video traffic based on the estimated available bandwidth in the network.

2. Delay: VoD services are extremely sensitive to end-to-end delay, they require that packets arrive at a certain time or deadline in order to be properly decoded and displayed. If a video packet does not arrive in time, the playout process will pause, which is annoying to human eyes. A video packet that arrives beyond its delay bound (e.g. its playout time) is useless and is discarded consequently. Since the Internet introduces time-varying delay, to provide continuous playback, a buffer at the receiver is usually introduced before decoding. On the other hand, especially for live content, a delay due to the use of a large buffer (also known as buffer-bloat problem), may not be acceptable, depending of course on the application.
3. Packet loss: a VoD system must expect some packet loss to occur, since it is an almost inevitable Internet phenomenon. A loss of video information potentially degrades video quality by damaging pictures, which is in turn displeasing to human eyes. To turn a system more robust against packet loss, a compression technique with multiple description coding layers could for example be used.
4. VCR operations: video-cassette-recorder operations are functions usually seen in video playback devices like play, pause, fast forward, rewind and random access. The Internet Real Time Streaming Protocol (RTSP) implements such operations [51]. Lin et al. [28] proposed a dual-bit-stream least-cost scheme to efficiently provide VCR-like functionality for MPEG video streaming. The video is coded with I-B-P frames in forward and reverse order, providing a dual bit-stream structure.
5. Decoding complexity: a video distribution system must assume operation over heterogeneous network environments where clients may request content using different devices, each one with specific requirements and limitations. For instance, a mobile device requires low power consumption, thus, streaming video to these devices must employ a simpler decoding algorithm. A client located in a network with low bandwidth is better served with a low bitrate video and/or a better compression rate algorithm.

These requirements draw an intrinsic relationship with the quality of service (QoS) of a VoD system. QoS relates to the agreements between a service provider and its customers. These agreements are usually defined in SLAs, a contract between the end user and the service provider. However, the perception of quality varies from different perspectives and views. Pathan and Buyya [129] proposes three views:

- **Quality as functionality**– According to this view, quality is considered in terms of the amount of functionality that a service provider can offer its customers. Quality as functionality characterizes the design of a service and can only be measured by comparing the service against others offering similar functionalities.
- **Quality as conformance**– Under this view, quality is seen as being synonymous with meeting provider's commitments and specification such as SLAs. Quality as conformance can be monitored for each service individually. It usually requires the user's experience feedback information related to a service in order to measure the 'promise' against the actual 'delivery'. For example, a provider who commits to its customer that 95% of their requests will be served within less than 2 seconds, and manages to maintain it at all times in its operation, is then considered as offering good QoS.
- **Quality as reputation**– In this view, quality is linked to user's perception of a service in general. This perception is developed gradually over the time of a service provider's existence. Quality as reputation can be regarded as a reference to a service provider's consistency over time in offering both functionality and conformance qualities, and can therefore be measured through the other two types of qualities over time. It can also be interpreted as being the 'goodwill' of a particular service provider.

2.2.3.2 *Quality of experience*

From the user point of view, fulfilling those requirements of a VoD service described earlier is very important to the perception of quality, but not sufficient. Measurements of network impairments do not represent the user experience. User experience, perception and expectation from a VoD system are not limited to network metrics and can be far more complex. For instance, a user may specifically care about video quality, delay when zapping through channels, audio-to-video synchronization, among others [199].

A users' viewing experience is crucial for a successful service deployment. If users experience frequent freezes in video playback, significant delays at startup or when switching channels, or large time lags, they may abandon the service [188]. Despite being an important requirement, there is yet no direct definition for quality of experience (QoE). Different flavors of QoE definitions can be found in the literature [115][172][91][101]. In [3], QoE is defined as "the characteristics of the sensations, perceptions, and opinions of people as they interact with their environments. These characteristics can be pleasing and enjoyable, or displeasing and frustrating". Goodchild [120] makes a distinction, stating that QoS is related to the network perspective, while QoE stands on the user perspective.

Zapater and Bressan [199] observe that QoE stands at a high abstraction level when compared to QoS. They propose to consider a perceptual pseudo-layer which is concerned with the end user experience, as an extension to the application layer in OSI model. Their proposal combines the QoE concepts with QoS in a one layered QoE/QoS model, as shown in Figure 2.2.

Figure 2.2 - QoS/QoE layered model [199]

In this model, each layer has its own set of key metrics. For instance, for the Transport/Network layer, the important metrics are those of QoS domain: packet loss ratio, bit error rate, packet latency, packet jitter, out-of-order packets, etc. For the application layer, they propose the following metrics: middleware delay, encoder and decoder delay, and set-top-box (STB) boot time.

Finally, in the QoE domain, the proposed metrics are [199]:

- **Startup time (or startup delay):** it is the latency or the delay a user experiences when he requests for a content and it effectively starts to play and is displayed. It is different from the "time to first byte" TTFB metric [63] because the startup delay must consider the buffering time.
- **Overall video quality:** the video quality can be measured by subjective and objective methods. Subjective methods are unpractical for most situations because

they try to establish the performance from a score attributed by a group of users asked to watch the video. The results are statistically treated and ratings are known as the Mean Opinion Score (MOS) [199]. Objective methods try to capture the MOS by using algorithmically calculated measures of quality that can be obtained through automated testing and analysis routines.

- **Channel zapping time:** is defined as the delay involved when changing channels in live streaming. In VoD, this can be considered as the delay when the user finishes a session and requests another video.
- **Playback continuity:** represents the percentage of received video frames in a preferred time period to be displayed in relation to the total frames. A delayed frame, that can be a result of network jitter (QoS metric), degrades the video playback.
- **VoD control time responsiveness:** is closely related to the VCR operations. VoD systems differ from live streaming as they allow users to arbitrarily seek parts of a video.

In this work, we are interested in measuring the proposed algorithms against a simplified model of content delivery network for VoD. The model is simplified because, as will be shown later, it does not simulate all aspects of video delivery due to limitations of the simulation environment. But the experiments analyze startup delay and in some cases playback continuity, among the main network metrics.

3 Content delivery networks

3.1 Overview

CDN is an overlay network that distributes content to cache servers strategically located close to end users. It is a collaborative set of network elements in which content is replicated in order to perform its transparent and effective delivery [26]. The purpose of a CDN is to improve the perceived user's QoS, by providing a fast and efficient manner of distributing content [139]. This is accomplished by maintaining multiple Points of Presence (PoP) with cache servers, also called surrogates or mirror servers that store copies of original content and serve users' requests on behalf of the original server. Figure 3.1 depicts an abstract view of a CDN.

Usually, a CDN topology consists of a set of **surrogates** scattered over the edge of the network, to which requests are redirected; **router** and other networks elements, that compound the CDN topology; an **origin server**, that holds the original content to be distributed through the CDN; a **request redirector**, that chooses an appropriate surrogate to satisfy user requests; and a **monitoring** system and an **accounting** mechanism, that provide logs and information for origin servers.

Content is an object requested by the user and is created or primarily provided by the origin server. It may be replicated either on demand or beforehand, by pushing content to distributed caches located at the edges of networks. It consists of two parts: encoded media, e.g. video, audio, documents, images, web pages; and metadata [144]. It can also be pre-encoded or live data, persistent or system transient. It may include User Generated Videos (UGV) and varying contents (e.g. directory service, file transfer service). The objective of a CDN can be either to facilitate delivery, for instance by dealing with different file formats or encoding types of a video, and/or increase availability, to cope with different user clients (e.g. web browser, PDAs, smartphones). A CDN is a virtual overlay considering the OSI reference model [88], whose functionality is supported by other application level protocols such as HTTP [151] and RTSP [51].

A CDN can be characterized according to this structure, i.e., the number of surrogates and redirectors, the protocols involved in its operation, the SLA requirements of content objects, among others. Their configuration represents an important engineering decision to the CDN provider and poses several implications in term of service efficiency.

Figure 3.1 - An abstract view of a CDN

At least three main actors can be distinguished in a CDN:

- Content provider:** represents the content owner. It can be either the actual producer of content or an entity that holds it to serve end users with. It is, in turn, a customer to the CDN provider as it seeks to lower its infrastructure costs and consequently hires a CDN service to have content placed on caches. Thus, the content provider delegates a Uniform Resource Identifier of the objects to be distributed. The content provider has the relationship with the end users who do not usually engage in deals with ISPs. Content providers want to keep that relationship exclusive, since it is the primary way to monetize service and need to keep track of each customer's usage. Therefore, security becomes an important issue [24], and traffic regarding the content providers and their users must not be disclosed.
- CDN provider:** is the owner of content delivery infrastructure. It focuses on building its network infrastructure (servers, network backbones) for content delivery. It must provide the following services and functionalities [51]: storage and management of content, content provider isolation, distribution among edges, content delivery, backup and disaster recovery solutions, monitoring, reporting, and performance measurement. It charges its customers (i.e. the content provider) according to content delivery to end users. Thus, it must not only monitor and collect usage information to support correct billing, but also perform maintenance

operations and service expansion. Typical costumers of CDN providers are ISPs, online music retailers, media producers, mobile operators, etc. The relationship between CDN provider and content provider is governed by specific Service Level Agreements (SLAs). Such SLAs may state some monetary penalties in case of failure to meet SLA commitments.

- **End users:** are the actual customers of a content provider. They ultimately access the services provided by the CDN provider. These services can be either free or paid, and are determined by the content provided. The CDN provider is transparent to end users. Their relationship with the content provider starts by subscribing or simply accessing its service. When a content is requested, the user is redirected to one of the caches of CDN. End users can be very heterogeneous in terms of their access hardware, software and communication capabilities using devices such as: desktops and laptop web browsers, smartphones, set-top-boxes, media player applications, etc.

Collaboration among these actors and CDN components can occur over nodes in both homogeneous and heterogeneous environments. Hence, CDN administrators must ensure that surrogate servers are strategically placed across the network. For the CDN provider, the decision of where to place surrogates directly affects network operation costs. Location of surrogates is a key CDN research challenge and is often modeled as a lower cost problem, widely known as the minimum K-median problem and it will be discussed in details in the next section. While dependent on the number of surrogates, two typical approaches arise to determine the optimal number [178][66]:

1. **The single ISP approach.** An ISP with a global network can achieve adequate geographical coverage without relying on other ISPs. It usually has a relative small number of surrogates and the most commonly adopted strategy is to deploy them in a couple of locations in key countries. Each surrogate must support a high number of objects (large caches). This simplifies the location decision problem, but, on the other hand, the main disadvantage is to increase the distance from the end users.
2. **The multi ISP approach.** Places numerous surrogates in as many global ISPs as possible. The main goal is to get surrogates close to end users and thus deliver content quickly. Large CDN providers such as Akamai have more than 90,000 servers [12] [57]. The disadvantage of this approach is that it is more complex to

optimally solve the location problem causing sometimes surrogates to receive fewer hits, resulting in poor performance.

A provider may also deploy different levels of local, regional, international caches at geographically distributed locations. Such arrangement is referred to as hierarchical caching. This may provide additional performance improvements and bandwidth savings [19].

Beyond the traditional actors of a CDN, the ISP is gaining further visibility either because of its relationship with the CDN provider as well as the formation of the so-called Telco-CDNs, as described later. This leads to new scenarios of better collaboration between those actors as well as between CDN providers themselves. There are initiatives like the Content Distribution Internetworking (CDI), a model that allows CDNs to have a means of affiliating their delivery and distribution infrastructure with other CDNs [111]. In this model, each CDN is treated like a black box and uses a common internetworking protocol, while using a proprietary one internally.

3.2 CDN architecture

The main components of the CDN architecture are depicted on Figure 3.2 [112]: 1. the origin server, that can either be provided by the content provider itself or be a mirror server within the CDN provider infrastructure; 2. the client or end user; 3. a set of surrogate servers, that store copies of content from the origin server and are in charge of delivering these to end users; 4; a redirector, or request routing component, which is responsible for directing client requests to appropriate surrogates; 5. a content manager, also called distributor, which moves content from the origin server to surrogates and ensures consistency of content in caches, keeping them up to date; 6. a set of monitors, which gather data and statistical information from different key elements of the CDN and conducts measurements in the network to support the decision processes in the CDN, like redirection and content distribution; and 7. an accounting mechanism, which maintains logs of clients access and records the usage of CDN servers. Not all CDNs have all of these components. For instance, the surrogates themselves may be responsible for distributing content, avoiding the need of a content manager. Also, some functionalities may be converged in the same component, for instance, the content manager may also be responsible for request routing and accounting.

The interactions between the CDN elements are shown in Figure 3.3, and work as follows [73]:

1. The origin server delegates its URI name space for objects to be distributed and delivered by the CDN;
2. The origin server publishes content in the content manager;
3. The content manager moves the content to the surrogate servers and updates the information at the request routing system in order to allow it to choose the best suitable surrogate;
4. The client requests an object from what it perceives to be the origin, but, in fact, the request is actually directed to the request routing system;
5. The request routing chooses a suitable surrogate to serve the user's request;
6. The selected surrogate delivers the requested content to the client and send accounting information to the accounting system;

7. The accounting system aggregates and distills the information into content details records for use by the origin server and billing organization. This information may also be used by the content manager and request routing system to provide, for instance, load balancing.
8. The organization uses the generated usage records to charge the CDN user (content provider) for the delivery service.

3.2.1 The Origin Server

The Origin Server embodies the role of the owner (or represents the owner) of the actual content of the CDN. It publishes the content in the CDN by copying the objects to the surrogate servers. A technical issue that arises here is the selection of content to be delivered to end user by the CDN. This has a major impact in terms of CDN pricing and performance [95].

The content of the origin server can be replicated to surrogate servers in full or partial mode [2]. Full site selection and delivery is a simplistic approach where surrogate servers perform entire replication of origin contents. It seems feasible when objects are small, like, for instance, Web pages. However, the continuing increase in size of web objects, especially video, and the presence of dynamic content make the problem of storing and updating content unmanageable. Furthermore, this approach also can result in an overuse of the network resources, because one may unnecessarily replicate a part of content that is seldom requested. Under partial replication, the CDN must decide which objects will be outsourced.

The goal of the replica placement problem is to decide the location of object replicas in the system, in order to either maximize the client perceived performance given an existing infrastructure, or minimize the infrastructure's cost given a specified system performance. Such system goals are abstracted into a problem definition that is used as an optimization goal [114].

Models for replica placement date back to early 70s under the context of the file allocation problem [17] and received attention from diverse research areas. Authors in [114] provide a thorough categorization of replica placement papers and the assumptions they use. In the context of CDNs, the target functions considered include client-replica distance, read access cost, read and update cost, and replica availability.

Some policies have been proposed in order to govern objects replication [58] and can be categorized as follow:

- **Empirical-based outsourcing:** the CDN engineer decides which content will be outsourced. Authors in [90] proposed customizing caching objects, treating them differently according to policies decided by the engineer. Heuristics are then applied to make such an empirical decision. The problem with this approach lies in the uncertainty of choosing the right heuristics;
- **Popularity-based outsourcing:** only the most popular objects are outsourced to surrogates [16]. This approach is largely studied. Its main drawback is that defining popular object is time consuming and reliable statistics are not guaranteed to last due to popularity variation of each object [2]. Moreover, such statistics are often not available on new objects.
- **Object-based outsourcing:** the content is replicated to surrogate servers in units of objects. It is a greedy approach to choose an object to be replicated according to the highest performance gain [191]. This approach achieves the best performance, but suffers from high complexity to implement.
- **Cluster-based outsourcing:** the content is replicated in units of clusters. The cluster is defined as a group of objects that have some common characteristics. Authors in [130] propose replication schemes to address the problems of both entire replication and object replication by aggregating objects in the origin server into multiple content groups and performing replication on per-group granularity. A group can be created either statically or dynamically based on changing parameters such as server load.

The authors in [148] propose a dynamic replication algorithm to replicate objects in response to changing demand. It takes server load and proximity into account to decide where to put content. In [130], authors propose a family of pre-fetching algorithms. They are based on popularity, Good Fetch (explained next), and objects lifetime.

The popularity approach is expected to offer the highest hit rate. The top-ten [62] technique proposes keeping in cache the 10 most popular objects. The object popularity is monitored and objects are pushed into caches whenever they are updated. A slight variance is to pre-fetch a defined number of the most popular objects from the entire system.

In Good Fetch [179], the criterion is to balance the access frequency and update frequency (lifetime) of objects. The objects that have the highest probability of being accessed during their average lifetime are selected for pre-fetching. The strategy adopts a

long term decision to increase bit rates using global access patterns, rather than short term pre-fetch based on recent history.

In pre-fetching, extra bandwidth consumption is required since to keep objects up to date, a download from origin to caches is performed, especially when changes to objects are frequent. To avoid this overhead, in pre-fetch by lifetime [192], objects with the longest lifetime are selected for download first.

A new approach is also proposed in [130], called Objective-Greedy pre-fetching, aimed to improve the performance in terms of various objectives: hit-rate, bandwidth or H/B metrics. The H/B-Greedy pre-fetching seeks to improve the H/B metric, which combines the effect of increasing hit rate (H) and limiting the extra bandwidth (B) consumed by pre-fetching.

3.2.2 *The surrogate server*

The surrogate or replica servers are close servers that store replicas of the origin server content. A CDN is usually classified according to its structure, number of surrogates, location and algorithm used for cache management. Each surrogate server has a database that contains a list of all available streaming sessions, the objects stored in the surrogate and information from the monitor and for the management of the CDN. Usually, two types of problem can arise for surrogates: the **location problem** and **cache management**.

3.2.2.1 *Location problem*

The location problem or surrogate servers' placement is closely related to the content delivery process. The placement puts extra emphasis on the issue of choosing the best location for each surrogate and the number of surrogates needed within the network infrastructure. Some theoretical approaches model the problem as the "center placement problem": for the placement of a given number of centers, they minimize the maximum distance between a node and the nearest center. Some variants of this problem are the facility location problem, k-hierarchically well-separated trees [93] and the minimum K-center problem [189]. The minimum k-center problem is NP-complete, highly complex and computationally intensive to be used in practice [121]. Also, there are capacitated variants that constrain the service capacity at centers, requiring that each facility serves no more requests than the capacity defined at that location.

Due to the computational complexity of these algorithms, some heuristics such as Greedy replica placement and Topology-informed placement were developed, which take into account the existing information from CDN, such as workload patterns and the network topology, providing efficient solutions with lower computational cost. Below, a brief description of some surrogate placement strategies:

- Greedy algorithm [135]: This approach works as follows. It chooses M replicas among N potential sites by choosing one replica at a time. In its first iteration, each of the N potential sites is evaluated in order to determine its suitability to host a replica. The cost is computed with the assumption that all clients will access that site. Then, the site with the lowest cost is chosen. In the second iteration, the greedy algorithm searches for a second site in conjunction with the site already selected. The iteration continues until M servers have been chosen. The greedy algorithm works well even with imperfect input data. Nonetheless, it requires the knowledge of the clients' network locations and all pair wise inter-node distances. The complexity of the greedy algorithm is $O(N^2M)$.
- Topology-informed placement strategy [155]: This model considers a fixed number of candidate machines where mirrors can be placed (Constrained Mirror Placement problem). An ISP may have a large number of machines scattered around the Internet capable of hosting mirrors. The question is on which subset of the candidate machines a content provider shall put its mirrors. In this approach, the replicas are placed (using the greedy algorithm) on nodes with the highest outdegrees (outgoing links) first. The assumption is that nodes with more outdegrees can reach more nodes with smaller latency. Nonetheless, it is shown that, in a constrained minimum K -center problem, simply increasing the number of mirrors yields very little performance improvement beyond a relative small number of mirrors.
- Hot Spot [103]: The hot spot algorithm attempts to place replicas near the clients generating the greatest load. It sorts the N potential sites according to the amount of traffic generated within their vicinity. It places the replicas at the top M sites that generate the largest amount of traffic. The vicinity is defined as the circle centered on a node with some radius (number of nodes distant). This approach performs worse than the greedy and tree-based strategies.

- Tree-based [15]: It is a placement algorithm based on the assumption that the underlying topologies are trees, and it is modeled as a dynamic programming problem. At a very high level, it divides a tree T into several small trees T_i , and shows that the best way of placing $t > 1$ proxies in the tree T is to place t_i proxies the best way in each small tree T_i , where $\sum t_i = t$. The algorithm is shown to find an optimal placement when the underlying topologies are trees, and clients request from the proxy on the path toward the Web server, that is, clients cannot request from a sibling proxy. The model can be extended to handle the case of hierarchical caching, where the down-stream proxies only hold a subnet of the up-stream proxies, and to handle hierarchies with different links bandwidth. However, the assumption of tree topology does not reflect the Internet topology and can prune possibly better placement choices.
- Scalable replica placement [190]: *It is a dissemination tree*, a dynamic content distribution system built on top of a peer-to-peer location service. It uses a self-organized replica placement protocol that builds the tree while meeting QoS and server capacity constraints. This approach minimizes the number of replicas while meeting clients' latency constraints and server' capacity constraints.

These algorithms have been proposed in the beginning of the last decade to deal with the placement problem of web server proxies. There seems to be little improvement in the later years taking place although the problem of content delivery can potentially bring some changes in these strategies. To understand the main reason for this lack of interest one needs to recall studies such as [103][155] carried out to efficiently place Web server replicas on the network concluded that a greedy placement strategy, which selects replica locations in a greedy fashion iteratively, can yield close to optimal performance (within a factor of 1.1–1.5) at a low computational cost.

3.2.2.2 Cache management

Management of content is an essential task of a CDN. There is a debate on how a CDN should decide the placement of content: whether using a centralized or distributed approach by letting the surrogate servers decide which content should be replicated [139]. The former is content replication approach and was discussed previously, the latter is known as a Web caching approach. Cache organization may involve integration with

other entities of a CDN architecture, especially the content manager. It directly affects important CDN performance metrics, like hit ratio, byte hit ratio and perceived latency [166].

In the Web caching approach, the Last Recently Used (LRU) is a widely known and adopted caching management policy mostly because of its efficiency despite its simplicity. Some caching techniques integrate Web caching with content replication by adopting a hybrid approach [17]. In [167], a hybrid approach is proposed to handle dynamic and static cache content.

Most caching algorithms segment video files and store only parts of the video instead of the entire one [9]. The caching algorithm must deal with the popularity of the content and somehow predict the changes on the popularity to prepare the CDN for possible flash crowd events.

Another issue is cache updating [76]. The most common method is the periodic update which has obviously some drawbacks. Other methods are: update propagation, which delivers an updated version to the cache servers whenever a change is made in the origin server; on-demand update, which propagates the last copy of a content to the surrogate based on prior requests (the content is not updated until getting requested); and the invalidation approach, which sends an invalidation message to all surrogate servers when there is a change at the origin server, then, the caches need to fetch an updated version later.

Cache organization is in turn composed of the caching techniques used and the frequency of cache update to ensure freshness, availability and reliability of content. Content caching in CDNs can be made on an intra-cluster and inter-cluster basis [32]:

- *Intra-cluster caching*
 - Query-based: On a cache miss, a CDN server broadcasts a query to other cooperating CDN servers (leading to a significant query traffic) [153];
 - Digest-based: Each CDN server has a digest of content held by the other cooperating surrogates (there is consequently an update traffic overhead) [185];
 - Directory-based: A centralized server keeps information of all the cooperating surrogates in a cluster (potential bottleneck and presence of a single point of failure) [75];
 - Hashing-based / Semi-hashing based for streaming: Cooperating CDN servers

maintain the same hash function (More efficient, highest content sharing efficiency) [99][177].

- *Inter-cluster caching*: Necessary when an intra-cluster content routing fails. Uses query-based scheme for inter-cluster content routing.

3.2.3 The Redirector

The Redirector or request-routing system performs the task of estimating the most adequate surrogate server for each different request and client. The most adequate surrogate may not always be the “closest” one. Hence, a request-routing system uses a set of metrics such as network proximity, client perceived latency, distance, available bandwidth and replica server load in an attempt to direct users to the closest surrogate that can best serve the request [55]. The redirector contacts the Content Manager to learn the location of replicas and has the support of monitoring information collected through the entire CDN.

According to the authors in [158], the request-routing system has two parts: request-routing algorithm and request-routing mechanism. The former is responsible for selecting an edge server in response to the given client request, and the latter informs the client about the selection.

3.2.3.1 Request routing algorithms

The algorithms invoked by the request-routing mechanisms can be adaptive or non-adaptive. Non-adaptive algorithms select a surrogate through heuristics, and are easier to implement than the adaptive algorithms. Adaptive algorithms chose a surrogate considering the current system conditions for content delivery, providing higher robustness during events like flash crowds. An example of an adaptive algorithm is the one used by Akamai [57]. It takes into consideration a number of metrics such as replica server load, the reliability of loads between the client and each of the replica servers, and the bandwidth that is currently available to a replica server.

One of the most common and simple non-adaptive request-routing algorithms is the *Round-robin* [116]. This algorithm consists of distributing all requests to the surrogate servers, balancing load among them. Round robin replies to requests with a list of addresses of surrogates' addresses that have the replicated content. In each request response, the address sequence in the list is changed. These algorithms are efficient for clusters because the replica servers are located at the same place. One of the drawbacks

of this algorithm is that it does not perform well when the surrogates are situated at distant places because it does not take into consideration the inherent distance cost of the surrogates [138]. Client requests may be directed to distant surrogates, resulting in poor performance and higher network resource occupation.

Another non-adaptive algorithm is based on the prediction of the *Load* [158]. It takes the distance into account and clients are redirected in order to balance the load among surrogate servers. One variation is called *Response Time*, which assigns a selection probability in inverse proportion to the response time previously experienced by the users [120]. A further non-adaptive algorithm ranks all replica servers according to the predicted load on them. The prediction is based on the number of requests that each surrogate has served so far. This scheme takes server-load and client-server distance to provide load balance to the system [26].

The *Worst Surrogate Exclusion* (WSE) is an algorithm that uses a set of previously collected metrics to redirect the client request to the surrogates. The metrics are latency among surrogate and clusters, load and speed (in Mbit/s) of links. In this approach, the surrogates with a value of latency higher than the average system response time are discarded [120]. It is based on three main concepts: the exclusion of surrogates that are too far according to a given criteria, the equalization of the average system response time, and the overload prevention.

Several non-adaptive request-routing algorithms are implemented in the Cisco Distributed Director [55]. One of these considers the percentage of all client requests that each surrogate receives. The idea is to send more requests to more powerful servers, to achieve a better resource utilization. Another algorithm defines preferences of one surrogate over others, and the third one proposes random request redirection. Other algorithms consider clients' geographic location to redirect requests to nearby replicas.

An adaptive approach is aimed to adapt redirection to hotspots [43]. It calculates a hashing function h from a large space of identifiers and uses this function for routing client requests to a logical ring. This logical ring consists of cache servers with IDs from the same space; the cache server with smallest ID larger than h is chosen for holding copies of the referenced data.

Globule [74] employs an adaptive algorithm that chooses the closest surrogates to the clients by considering network proximity [116]. Latencies in Globule are estimated by positioning nodes in an M-dimensional geometric space. The latency between any pair

of nodes is then estimated as the Euclidean distance between their corresponding M-dimensional coordinates. Such measure is passive and does not add any traffic to the network. This metric determines the distance through periodic refresh of the AS-based map of the internet and it is periodically rebuilt to adapt to changes on network. Although it is a very simple strategy, the proximity estimation is not very accurate.

The authors in [24][110] proposed an adaptive request-routing algorithm based on client-server latency. The algorithms select to which surrogate client requests are sent based on passive server-side latency measurements. Client requests are redirected to a replica according to minimal latency recently reported to the client. These algorithms are efficient, but require the maintenance of a central database of measurements, limiting the scalability of systems [158].

The authors in [182] evaluate the performance of some algorithms for request-routing and define a class of new algorithms that carefully balance load, locality, and proximity. The goal was to examine how these strategies respond under different loads, and especially how robust they are in the face of flash crowds and other abnormal workloads that might be used for a DDoS attack. Two schemes were proposed: Fine Dynamic Replication (FDR) and Coarse Dynamic Replication (CDR). The goal is to dynamic adjust the number of servers selected for a given object, even when redirectors do not have enough information about the server load.

Akamai [57] uses a complex request-routing algorithm adaptive to flash crowds. The system determines the *nearest server* as a function of network topology and dynamic network characteristics; *availability* as a function of load and network bandwidth and *likely* as a function of which servers have the content. The details of such technology are proprietary to Akamai and not revealed.

3.2.3.2 Redirection Mechanisms

Redirection mechanisms inform the client about the suitable surrogate through information generated by the redirection policies [10]. According to [73], they can fall into the following categories: client multiplexing, anycasting and peer-to-peer.

Client Multiplexing

In client multiplexing approach, a client obtains the addresses of candidate surrogates and chooses one to send requests to. Mechanisms of this category include global server load balancing, DNS-based, HTTP redirection and URL rewriting.

The global server load balancing extends Layer 4–7 switches and allows them to navigate world-wide networks. Two new capabilities extend these service nodes to allow global server load balancing. The first is global awareness, and the second is smart authoritative DNS. In server load balancing, the Web switch in a service node is aware of the health and performance of the real Web servers attached to it. The result is that the Web switches making up each service node are globally aware, each knowing the addresses of all the other service nodes. They also regularly exchange performance information among the Web switches. This allows each switch to estimate the best server for any request choosing not only from its pool of locally connected real servers, but from the remote service nodes as well. To make use of this global awareness, the switches act as intelligent authoritative DNS servers for certain managed domains.

DNS-based request-routing provides a proximity solution based on agents running in reverse proxy Web caches and cooperates with an intelligent authoritative DNS. It is a mechanism widely used in several CDNs due to the ubiquity of the DNS system. The disadvantage is that it increases network latency because of the increase in DNS lookup times. In DNS based Request-Routing techniques, a specialized DNS server is inserted in the DNS resolution process. It is divided in single or multiple reply and it can also have a multi-level resolution [18].

Work in [142] describes a cooperative DNS (CoDNS), a lightweight lookup service that can be independently and incrementally be deployed to augment existing name-servers. It uses a locality and proximity-aware design to distribute DNS requests, and achieves low-latency, low-overhead name resolution, even in the presence of local DNS name-server delay/failure.

Although DNS-based server selection is transparent and general, it has two inherent limitations [118]. First, it is based on the implicit assumption that clients are close to their local DNS servers. The second limitation of DNS-based server selection is that a single request from a local DNS server can represent differing numbers of Web clients — this is called the hidden load factor (and makes load balancing difficult).

In [8], authors present an architecture and an evaluation of a request routing DNS server that decouples server selection from the rest of DNS functionality. The DNS server, referred to as MyXDNS, exposes a well-defined API for uploading an externally computed server selection policy and for interacting with an external network proximity

service. In [116], another redirection system that separates the redirection mechanism from policies aimed for Apache is presented.

Several CDNs use this scheme for request-routing. Akamai [57] employs a two-tier DNS redirection jointly with URL rewriting. This mechanism considers server load and network-related metrics. Globule [74] applies a single-tier DNS redirection based on proximity.

HTTP redirection is a mechanism that propagates information about surrogate servers in HTTP headers [73]. HTTP protocols allow a Web server to respond to a client request with a special message that tells the client to re-submit its request to another server. This mechanism can be used to build a special Web server which accepts client requests, chooses replica servers for them and redirects clients to these servers. One drawback of this mechanism is the lack of transparency. On the other hand, it is flexible and can be managed at a fine granularity level, down to individual objects.

URL rewriting is mainly used for partial-site content selection and delivery. In this approach, dynamic generated links are rewritten in order to redirect clients to specific surrogates. CDNs usually provide scripts that parse Web page content and replace embedded URLs [102]. One advantage is that like HTML redirection, fine granularity can be achieved with this mechanism. Some of the disadvantages are the delay for URL-parsing and bottleneck introduced by an in-path element. Akamai is a major provider of content delivery services based on URL rewriting.

Anycasting

An anycasting redirection mechanism, as defined by [42], provides "a stateless best effort delivery of an anycast packet to at least one host, and preferably only one host, which serves an anycast address". This mechanism can be classified into two classes: IP anycasting and application-level anycasting.

Under IP anycasting, an IP address is assigned to a host and each IP router holds a path in its routing table to the host that is closest to this router. Thus, different IP routers have paths to different hosts with the same IP address [73]. The main disadvantage is that some parts of the IP address space are allocated and do not scale well. In [59], authors propose a framework for scalable global IP anycast.

With application-level anycasting, the network layer is better suited for choosing the shortest path and therefore selects the closest surrogate. It can rely on metrics such as

hop count [73]. On the other hand, the application layer can handle a variety of other metrics, such as server throughput, latency, etc. Because of this, in [70] a framework is proposed to identify a server using an application-layer anycasting service. It consists of a set of resolvers, which perform anycast domain names to IP address mapping. The goal is to allocate servers to clients in a way that minimizes response time. An anycast domain name uniquely identifies a potentially dynamic collection of IP addresses, which constitutes an anycast group (an additional DNS lookup is required). Each resolver has a metric database that contains performance data about servers forming a group. This mechanism works in layers 4-7 of the OSI reference model.

Peer-to-peer

Since peer-to-peer systems build the information retrieval using network members themselves instead of relying on additional dedicated infrastructure, they are consequently more fault-tolerant than common CDNs [73]. In a peer-to-peer mechanism, requests are forwarded hop-by-hop, a way similar to IP routing, with a limited hops-to-live as well. It is an adaptive system that enables storage and retrieval of data while maintaining the anonymity of reader and authors [86]. It is not only a request routing mechanism but a complete distribution network, like the CDN itself. A noticeable example is the BitTorrent network [22].

3.2.4 The Monitor

Its role is to probe the network to collect data that will support the tasks performed by the redirector and the content manager. It includes probing the set of servers, to know the latency inherent to communicating with surrogate servers and origin server, and to acquire information about congestion paths. The monitors can be SNMP-based or agent-based. The SNMP-based one is seen as a straight-forward implementation. The disadvantage of this technique is lack of flexibility. In the agent-based approach, the logic is implemented in monitor agents and their interactions can be programmed and also changed at run-time. Also, agents can be arranged in a hierarchical manner and some agents can be used to aggregate information gathered by others. Furthermore, prior knowledge about part of the network topology could be provided by the operator and combined with active measurements, to give a more complete view of the network topology and current conditions.

From the content provider perspective, the most important CDN criteria are cost, performance and availability. In terms of costs, using a CDN can be initially more expensive than not using one [66]. As traffic increases, however, the overall economic declines.

For CDN performance, typically the metrics used to evaluate it are [178][96][194]:

- Cache hit ratio: the ratio of cached objects versus the total objects requested. A high hit rate reflects an effective cache policy [154].
- Saved bandwidth: the decrease in bytes retrieved from origin servers.
- Latency: the set of surrogate servers that least delay the requested objects. Reducing latency generally decreases the saved bandwidth.
- Surrogate server utilization: the fraction of time that the surrogate servers are busy. Administrators use this metric to calculate CPU load, the number of connections being served, and each surrogate server's storage I/O.
- Reliability: the packet-loss ratio. Managers use packet-loss measurements to calculate a CDN's reliability. High reliability indicates that the content is always available.

According to [178], it is recommended that CDN customers either perform their own tests or use an independent performance-measuring company such as Keynote Systems [100] or Web Lens [183]. The performance testing of a CDN provider can be misleading because the service might be good for a specific site but performs poorly for another. The performance depends on both content and location. Also, some CDNs may support only specific file types or stream formats, or even certain countries or geographical regions. Independent performance-measuring companies can efficiently measure CDN performance because they support benchmarking networks of strategically located measurement computers connected through major Internet backbones in several cities, providing an end user perspective of performance in critical areas [96].

3.2.5 The Content Manager

The content manager controls the way the replica objects are stored in each surrogate server. It provides this information to the Redirector, to get each client served by the most suitable surrogate. Furthermore, the information managed by the Content Locator is stored in a database (DB content). The Content Locator determines:

- Number of replicas of a media object;
- In which surrogate server a new object has to be stored;
- Elimination policy of non-popular objects from the surrogates;
- Interaction of the CDN within the origin servers;
- Update media objects in the surrogates;
- When and which media objects to move among surrogates;

Some of these tasks were explored in previous sections. As mentioned before, they can be performed in cooperation or exclusively by other entities. A common task performed by the content manager is moving media among surrogates. There are four policies defined in the literature: cooperative pull-based, non-cooperative pull-based, cooperative push-based and non-cooperative push-based.

Cooperative pull-based: This policy consists of pulling the content from the nearby surrogates in case of cache misses. The surrogates find nearby copies of the requested objects using a distributed index, storing them in their caches. This scheme is reactive, since a data object is cached only under client requests. Such policy implies a high communication overhead when the number of clients is high. An academic CDN, Coral [40], uses this approach.

Non-cooperative pull-based: In this policy, the surrogates pull content from the origin server when there is a cache miss. Many successful CDNs such as Akamai and Limelight [106] use this approach, since cooperative push-based is considered optimal but costly. The drawback for this approach is that CDNs do not always choose the optimal surrogate to serve the content and the object is replicated several times in different surrogates, suffering from excessive replication redundancy.

Cooperative push-based: This policy consists of proactively pushing the content from the origin server to the surrogate servers. The CDN maintains a mapping between content and surrogate servers. If the requested object has not been outsourced, then the origin server is responsible for serving the content to the client and, in case of a cache miss, the request is forwarded to a nearby surrogate. In this policy, the surrogate servers share efficiently the bandwidth among them, reducing the cost to implement the cooperation, as well as reducing the maintenance costs. The main algorithm for content

replication used by this approach is the greedy-global heuristic algorithm [98].

Non-cooperative push-based: This policy consists of proactively pushing the content from the origin server to the surrogate servers. There is no cooperation between surrogates to provide information about redirection in this scheme, which does not provide flexibility in adjusting replication. This is a limited scheme that is not used in practical deployments and can be considered as a baseline for comparison for example. The authors in Chen et al. [31] state that the cooperative or non-cooperative-push based policy has a better result when compared to the other approaches.

3.2.6 *Accounting/billing mechanism*

There are technical and business challenges in pricing CDN services. CDN providers charge their customers according to the content delivered (i.e. traffic) to end-users by their replica servers. The average price of CDN services is relatively high [44], often out of reach for many Small-to-Medium Enterprises (SME) or not-for-profit organizations. Prices for CDN services are mainly affected by [139]:

- Bandwidth usage, which is measured by the CDN provider to charge (per Mbps) customers typically on a monthly basis;
- Variation of traffic distribution, which characterizes pricing under different situations of congestion and bursty traffic;
- Size of the content replicated over edge servers, which is a critical criterion for posing charges (e.g. price per GB) on customer audiences;
- Number of edge servers, which captures the ability of a CDN to offer content at charges that will not exceed the charges in typical caching scenarios; and
- System reliability, stability and security issues for outsourcing content delivery. It also includes a cost of sharing confidential data, which varies over different content providers on the basis of the protected content type.

CDNs support an accounting mechanism that collects and tracks information related to the Redirector. This mechanism gathers information (via monitors) for each CDN component. This information can be used in CDNs for accounting, billing and maintenance purposes [25].

3.3 Categories of CDN

3.3.1 *Conventional CDN*

The traditional CDNs were the first commercially available. Because of the inherent proprietary nature of this type of CDN, it is difficult to reveal detailed information about their adopted technical strategies [26]. It is possible, though, to outline some of the main characteristics of some successful CDNs for comparison purposes. Many agree that among the most successful ones are Akamai, Limelight Networks and Mirror Image.

Akamai technologies was founded in 1998 at Massachusetts. It evolved out of an MIT research project aimed at solving the flash crowd problem [57]. When it first launched its service, this was designed to deliver only web objects (documents and images). Since then, it evolved to deliver dynamic content (animations, scripts, DHTML) and streaming of audio and video [26].

By the end of 2011, the Akamai network grew up to 61,000 servers distributed over 1,000 networks in 70 countries [82]. It is, by far, the biggest CDN in the world, carrying between 15 and 30 percent of all Web traffic, with major customers like Facebook, Twitter, Apple, and the US military [64].

Akamai is organized to be conceptually a delivery network, “a virtual network built as a software layer over the actual Internet, deployed on widely distributed hardware, and tailored to meet the specific systems requirements of distributed applications and services” [64]. For performance concerns, Akamai divides the distribution system according to the type of content being handled (streaming, web content, video, etc.), forming **multiple delivery networks**, each tailored to their type. These distributions systems share a similar architecture, but their main components differ. The basic operation works as follow.

The user types a URL into his/her browser, the domain name of the URL is then translated by the mapping system into the IP address of an edge server to serve the content. The redirection system is based on large amount of historical and current data to choose an edge server close to the end user. Akamai uses a DNS-based redirection system, it combines topology information from BGP and live networks statistics to map user requests to edge servers. Akamai has also a transport system to move content from the origin to the edge servers; a *communications and control system*, used for disseminating status information; and a *data collection and analysis system*, responsible for collecting

and processing reported data from various sources such as server logs, client logs, and network and server information. Finally, a *management portal* provides two functions: a configuration management platform that allows an enterprise customer to retain fine-grained control over how its content and applications are served to the end user, and a visualization of the way users are interacting with content and applications [64].

Limelight Networks was founded in 2001 at Tempe, Arizona [77]. Limelight CDN supports combination of content delivery directly from its servers or from costumer's servers, if connected to the Limelight network. The content can be sent to surrogates through ISPs or the public Internet, if appropriate. Upon user requests, the redirection is also based on DNS to choose a suitable surrogate, which can be either from Limelight or the content provider itself. Limelight offers a consumer's content library within the Limelight Architecture and a Custom CDN solution (to allow the combination of surrogate servers) with Web based management. In terms of content, LimeLight supports the distribution of on-demand streaming, web content, and dynamic objects.

Throughout the last decade, Limelight and Akamai engaged in a series of patent disputes with partial victories for both sides [81][133] and a final decision favored Limelight [79], with 40 million dollars grant in damages. Both companies are big players in the CDN market and performed various acquisitions of smaller CDN companies [46].

Also founded in Massachusetts, in 1999, Mirror Image is a provider of online content, application, streaming media and Web. Like Akamai and Limelight, the redirection is DNS-based, using a load balancing mechanism to determine a suitable surrogate with best response time. Mirror Image takes advantage of a Content Access Point Network [124].

Mirror Image CAP is a proprietary architecture to deal with the problem of "conventional" models, as follow: a) caching architecture: that uses pre-positioning in combination with cache to place medium size content servers. This model has some scalability problems; b) "convenience store": a model that distributes small content servers at POPs across local ISP networks. This model suffers from Internet Exchange (IX) point congestion and bigger latency; c) Dedicated architecture: small servers across regional ISPs (like "b"), but connected by a dedicated network. It solves the problem of Internet congestion but requires significant amount of processing power and coordination of data transmission; d) Superstore architecture: massive servers in central locations close to large regions of users (IX points). While a significant improvement over the caching,

dispersed and dedicated architecture models, a concentrated architecture does not always solve the performance and cost efficiency issues associated with Internet content delivery, such as in transaction based processing.

The Mirror Image CAP network is a combination of afore mentioned architectures, based on modular design of server **appliances**. CAP network is collocated with strategic IX peering points with high concentration of end users, after an evaluation of demographics. When a user requests content from a Mirror Image provisioned Web site, the distributed URL automatically routes the request to a global load balancer on the CAP network, where DNS racing determines the IP address of the CAP location that can provide the fastest response time [124]. Because of this modular design, Mirror Image claims to have a “configurable, framework-based approach to enabling edge computing that does not require coding or deployment of customer's proprietary code to its network” [53].

3.3.2 *Academic CDN*

Among the academic CDN proposals, there is CoDeeN, Globule and Coral [41]. CoDeeN [104] is an academic CDN running over the PlanetLab infrastructure. This CDN provides high-performance proxy servers, where each proxy server behaves both as a request redirector and surrogate server. Several projects are related to CoDeeN such as CoBlitz, CoDeploy, CoDNS, CoTop, CoMon, CoTest, and CoVisualize [51]. CoDeeN nodes are deployed as open proxies in order to allow access from outside the hosting organization.

Globule [57] is an open-source collaborative CDN developed at the Vrije Universiteit. Globule allows the operation of hosting platform of Web content providers. Moreover, it provides replication, monitoring of servers and redirecting of client requests to available surrogates. In Globule, a site is defined as a collection of documents that belong to one specific user (the site's owner) and a server is a process running on a machine connected to a network, which executes an instance of the Globule software. Each site is composed of the origin, backup, replica, and redirector servers [26]. The proximity measure is based on inter-node latency.

Coral CDN [143] is a P2P content distribution network designed to distribute web content. This CDN uses DNS redirection to redirect requests to nearby web caches. These web caches decrease both the load on the origin server and the user perceived latency,

applying a Distributed Sloppy Hash Table (DSHT). Using Coral, under-provisioned web sites can achieve higher capacity, thus Coral can handle large amounts of traffic with reasonable performance. It is comprised of three main parts: a network of cooperative HTTP proxies for handling client requests; a network of DNS name-servers that map clients to nearby CoralCDN HTTP proxy; and an underlying indexing infrastructure and clustering machinery on which the first two applications rely [26].

3.3.3 *Peer-Assisted CDN*

A peer-assisted CDN is a hybrid architecture that takes advantage of both traditional CDN and peer-to-peer systems. A CDN can be seen as a centrally managed content delivery architecture that benefit of professional administration and amply provisioned resources. CDNs can achieve high performance and reliability, but often are also very expensive to scale [123]. On the other hand, peer-to-peer systems must rely on resources contributed by their peers, and properties are the opposite, require little investment, in fact, they require no infrastructure at all, exhibit a decentralized nature, but lack a predictable QoS.

Thus, it is enticing the idea of combining the best of both worlds. A hybrid approach of a peer-to-peer system and CDN. For instance, in [60], authors propose a hybrid architecture that integrates CDN and P2P based streaming media distribution. The two streaming technologies complement each other: when a media file needs to be distributed to a community of clients, it will first be distributed by a CDN server. While fulfilling streaming requests from clients, the CDN server will also create the seed, supplying peers in its service area. Combined, the CDN server and peers serve the streaming media requests with higher capacity than that of the server alone. Besides, when the P2P capacity grows above a certain level, the CDN can even stop serving and let the peers take over the task: allowing a “CDN-to-P2P” handoff.

There is a significant amount of work on providing video streaming using peer-assisted networks [176].

3.3.4 *Telco-CDN*

ISP and Content providers are traditionally independent entities [184]: ISPs provide connectivity and are only responsible for transporting the content. But, as in most transportation systems, connectivity and bandwidth are becoming commodities and ISPs find their profit shrinking. They want to participate in the market of content delivery and

distribute content to their customers. The Internet, from the business perspective is becoming more content-centric. In this context, many ISPs are considering the deployment of their own CDN [4]. Besides, deploying its own CDN allows the ISP to have more control of the traffic inside its network, decreasing bandwidth usage and keeping traffic locally when possible [200].

In this context, it is possible to envision a new kind of CDN, deployed by and inside an ISP, able to provide storage space with guaranteed links to customers [45], forming the so-called Telco-CDNs. Examples include Verizon [35] and Comcast[164]. Comcast also released a commercial content delivery service primarily targeting medium and large size content owners [171]. The service is live and the company has a few paying customers already delivering their content on Comcast's CDN. This new offering allows content owners to go directly to the ISP and have their content stored and delivered via the last mile, thereby displacing some traffic currently delivered by third-party CDNs like Akamai and Limelight Networks [47].

3.3.5 *Federated CDN*

The dissemination of CDN providers, mainly when considering the opportunities to the Telco-CDNs, also brings the possibility of collaboration among them. Lazar and Terrill [84] explain some concepts behind the CDNs and how they can interconnect with each other in a profitable way. The idea of CDN peering was presented and an IETF workgroup called Content Distribution Internetworking (CDI) [39] was created in March 2002, but finished the activities in March 2003.

CDN peering [25] is an approach to reduce expenses and avoid adverse business impacts. It can avoid the creation of "Islands of CDNs" that requires an enormous capital and labor. By cooperating with each other, CDN providers can increase their geographic presence, reduce costs, increase the availability and also ensure SLAs. A CDN federation is formed by a set of autonomous CDNs who cooperate through a mechanism to share resources while enjoying larger scale and reach. This set of autonomous CDNs can be seen as a **Federation of CDNs** or a **Federated CDN**. There is an ongoing IETF workgroup on CDN peering. The Content Delivery Networks Interconnection [40] workgroup progressed with discussions about protocol and standards to provide CDN peering requirements. Some concerns rose such as the services providers' lack of track

record with peering and the complex mechanisms that have to be deployed (billing and management systems, service-level agreements) to make peering successful.

Last but not least, one cannot review CDNs without mentioning Netflix's own CDN, the Open Connect. It is a CDN aimed to reduce the traffic within ISPs [54]. Netflix is informing ISPs that they can choose to have Open Connect Appliances within their datacenters, or to peer with the Open Connect network at common Internet Exchanges; Netflix provides either form of access at no cost to the ISP.

3.3.6 Cloud-based CDN

As discussed earlier in the first chapter, a cloud-based CDN can reduce the effort and cost of producing a solution for content delivery. With the introduction of cloud-computing, a de facto standard for utility computing, the CDN landscape has experienced the growth of cloud-based CDNs. It uses the cloud to map the CDN infrastructure elements into virtual components. For instance, a surrogate (or cache) server can be mapped to an IaaS storage and delivery service. By doing so, Clouds endeavor to improve cost efficiency, accelerate innovations, attain faster time-to-market, and achieve application scalability [171]. An example and first effort of this type of CDN is the MetaCDN [92].

In the cloud-based CDN, another actor is added to the model: the infrastructure provider or the cloud provider. The CDN provider is still the owner of the infrastructure, controlling all aspects of content deliver, but not the physical one. It is now responsible for not only assuring content delivery, but to allocate the necessary resources from the cloud. A cloud-based CDN is, by nature, a dynamic one.

3.3.7 Autonomic CDN

A CDN can also take advantage of an autonomic strategy for surrogate management and self-organization based in roles, forming, thus, an Autonomic CDN. The work in [80] proposes a distributed replica placement algorithm for self-organizing replica placement. Each surrogate runs the algorithm independently of each other and finds a convergence for a replica placement.

3.4 Concluding remarks

In this chapter we discussed the main issues and problems related to content delivery networks. Most of the studied techniques were implemented in a simulator we use to experiment our proposals. We also provided a categorization of CDNs. As we will detail later, the first CDN model proposed in our framework is inspired in cloud-based CDNs but it also includes concepts of Telco and federated CDNs. We finalize our proposed framework with an autonomic CDN approach.

In next chapter we start introducing the first component of our framework, which is responsible to detect flash crowds or other relevant changes in monitored metrics to allow a CDN reorganization. This is the first contribution of our work. In chapter 5, we present the complete framework architecture and the slicing algorithm and finally present an extension for an autonomic approach.

4 CDN change manager

This chapter presents one of the modules of the proposed framework. First, we focus on an important aspect of any distribution system: its ability to handle flash crowds. After all, it was this problem that led to the birth of Akamai first as a project then as a successful service.

4.1 Framework overview

The framework is aimed to solve the allocation problem discussed in the introduction, allowing the CDN provider to decide how much resources must be allocated to each tenant. The components are depicted in Figure 4.1. The basic CDN entities, like request redirector, origin server, accounting system are not shown. The framework is supported by two main modules. The first one, called Resource Manager, is an optimization algorithm to orchestrate the allocation of capacity for each tenant in this multi-tenant environment, facilitating the deployment of a “virtual CDN”. The resource manager is supported by the slicing algorithm and will be studied in next chapter.

The second module, called Change Manager, provides an algorithm to monitor and detect relevant changes in CDN metrics in order to allow adaptation of CDN infrastructure. This is done in context of dynamic CDN which will be later used as a basis to the idea of adaptation and orchestration of resources within the framework.

Figure 4.1- Framework overview

4.2 Dynamic CDN

Many strategies have been proposed to deal with aspects inherent to the distribution model, however, they often fail to meet the high throughput demands of unexpected and unplanned events, like flash crowds, although very effective most of the rest of time. Furthermore, due to the distributed nature of CDNs and their varying demands, it remains difficult to predict an optimal configuration, for example, a suitable placement of caches given the network topology. **There is no single CDN configuration that fits all solutions.** This leads to isolated spikes of poor performance, despite an overall good one.

Some solutions have already been proposed to address these problems. Promising solutions should incorporate a certain principle: changing the static CDN into an adaptive dynamic CDN [131]. The idea is not new, several network services and management systems allow the optimization of network and/or services operating by rearranging network configuration or resource allocation according to traffic profile or service usage. The novelty of this work is how the adaptation is performed and the algorithm to detect the necessity of adaptation.

While this adaptation is desirable and often required for the adequate functioning of some networks and services, its usage is costly in terms of network resources and should be avoided when not absolutely needed. Typically, such adaptations are performed on a periodical basis, and the success of adaptation algorithms often depends on the length of period between re-adaptations. Sometimes, the adaptation is performed on long periods and allows the network/service to operate for a long time on sub-optimal conditions. On other occasions, the adaptation is performed very often and its cost may be comparable or could sometimes even outweigh the gains derived from the optimization itself.

In summary, the main challenges of a dynamic CDN is how to organize or reorganize a temporary overlay of resources quickly and without involving much overhead. In this context, we think it is important to establish a manner to detect, as early as possible, the beginning and the end of flash crowds or other relevant changes in traffic behavior, in order to perform adaptation only when necessary. Hence the question is not only to know how to adapt but equally to identify when to do this. This problem is part of the proposed framework, and, for the sake of testing, we consider the model of

virtualized CDNs to evaluate and study some strategies, before incorporating in the solution.

4.3 Related work

To detect the beginning of flash crowds, most solutions rely on monitoring the load of surrogate servers and comparing this with their overall load [48][60][94][198]. The work in [131] classified the solutions for dynamic adaptation of CDN into three categories: **server layer, intermediate layer and client layer**. Intermediate-layer solutions utilize network resources to perform offloading. Examples include CoralCDN [72] and multi-level caching [11] (see Chapter 3). Client-side solutions make clients help each other in sharing objects to distribute the load burden from a centralized server. An origin Web server can mediate client cooperation by redirecting it to another client that has recently downloaded the objects [140]. Peer-Assisted CDNs fall into this category.

Most solutions, however, are server layer. For instance, DotSlash [198] uses a mutual-aid rescue model. A web server joins a mutual-aid community by registering itself with a DotSlash service registry, and contributing with its spare capacity to the community. A self-configuring service discovery is used to allow servers of different web sites to learn about each other dynamically. Rescue actions are triggered automatically based on load conditions, and a rescue server can serve the content of its origin servers on the fly without the need of any advance configuration. Other examples of server layer adaptation are the cloud-based CDNs [157].

In this work, we focus on the server layer by proposing the virtualization of a VoD CDN (section 4.5), allowing programmatic modifications in a CDN infrastructure designed for video distribution, in order to quickly adapt it to new operating conditions [125]. Before, in section 4.4, we introduce the module that will be incorporated in proposed framework, aimed to detect flash crowds or any other significant changes in monitored metrics by using information from the CDN Monitor.

4.4 Detecting changes

4.4.1 *Change manager description*

Due to the distributed nature of overlay networks and their varying demands, it remains difficult to predict major network changes in Content Delivery Networks. To accomplish this and assist in other management level decisions, a new component is added to the CDN infrastructure: the Change Manager, shown in Figure 4.2.

Figure 4.2 – Change Manager Overview

The Change Manager is responsible for detecting significant changes in network conditions and triggering the necessary adaptation, by informing the resources manager, which interfaces with the host physical infrastructure in order to allow the dynamic allocation and release of resources. The main feature is the detection of network and resource state changes, which allows adaptation of the infrastructure to new user demands and resource usage states. The Change Manager collects information from the resource monitor including important network traffic metrics and, based on its policies, may therefore decide whether and how to perform adaptation.

For instance, in a CDN, if it detects a change in traffic behavior, like the presence of a hotspot in the topology that is suffering from a sudden increase in content requests, it may inform the resources manager to move some replica servers around in order to avoid an increase in average startup delay or network congestion. The resources manager is the entity that effectively modifies the CDN infrastructure and enforces requested configuration updates. The availability of changes and configuration is defined in an associated configuration parameters DB. The reactive measures taken are understandably dependent on the desired system goals and policies.

Policies may include information like the granularity of change to watch for, rate of monitoring or the desired CDN overlay business goals such as: increased availability, decreased response time, high network usage or inter-AS traffic. Furthermore, there may be restrictions on aspects like the maximum number of allowed configuration changes that can be made within a given timeframe, or that some surrogates are fixed and cannot be migrated or removed. For instance, the system may not allow more than three reallocations per hour, leave a certain AS without surrogate, for a QoS for some content, etc. In other words, the present problem may be formulated according to constraints with regard to the usage, configuration and location of the underlying related resources and services for example.

4.4.2 *Detection algorithm*

The change manager is based on the monitoring and processing of information provided by the monitor (probes) to constantly check changes in the network condition. This is accomplished by a **utility function** U that receives the probes as input to build the current network state (pattern). This latter could be represented as a set of vectors of dimension n , where n is the number of elements, services or resources being monitored, *e.g.*, the number of nodes in the network. This utility function takes into consideration the supported policies and goals of the overlay and weighs accordingly the data provided by the monitor. These are often considered as system constraints. Thus, the utility function takes all probes in $\mathbb{R}^{m \times n}$ space, where m is the number of metrics being monitored. Let P_{mn} be the probe of metric m in dimension n , and φ_{mn} the weight for that metric defined in accordance to the desired policies of the system. Note that these weights may change in time, but for simplicity we do not include such notation. The resulting utility function can be defined as follow:

$$U: \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^n \quad (1)$$

$$U(\text{all probes}) = \hat{V}_u,$$

$$\hat{V}_u = \frac{V_u}{\|V_u\|} \text{ and } V_u = \{u_1, u_2, \dots, u_n\}, \text{ where } u_i \text{ has the form}$$

$$u_i = \sum_{j=1}^m P_{ij} \cdot \varphi_{ij}$$

Thus:

$$\begin{aligned} V_u = \{ & P_{11} \cdot \varphi_{11} + P_{21} \cdot \varphi_{21} + \dots + P_{m1} \cdot \varphi_{m1}, P_{12} \cdot \varphi_{12} + P_{22} \\ & \cdot \varphi_{22} + \dots + P_{m2} \cdot \varphi_{m2}, \dots, P_{1n} \cdot \varphi_{1n} + P_{2n} \cdot \varphi_{2n} + \dots + P_{mn} \\ & \cdot \varphi_{mn} \} \end{aligned}$$

The system periodically collects probes and calculates the result of the utility function, setting then the **current vector** $\hat{C}$. Note that a filter such as a moving average may be used to avoid instable reactions that could lead to highly frequent system state changes. The filter parameters may also be subject to user policies for example to reflect CDN provider willingness to reconfigure its resources. The current vector is passed as input to a **trigger function** T , which is, in turn, responsible for detecting whether the changes over time are significant and, if so, the system must perform a pre-established adaptation, like, for instance, when considering the CDN overlay, a new placement of replica servers (performed by resources manager). The detection of changes over time is made by comparing the recent system probes to a **baseline vector** $\hat{B}$ that reflects the system state after the last adaptation. Both $\hat{B}$ and $\hat{C}$ are calculated using the same metrics and weights and differ only by the time data were collected. The utility function could be any form of data analysis made with both vectors, for instance a dot product between them or even a fully-fledged Artificial Intelligence strategy like Adaptive Resonance Theory (ART). Considering as an example that the utility function is the dot product between the vectors generated from baseline and current probes, we have:

$$T: \mathbb{R}^n \rightarrow \mathbb{R} \quad (2)$$

$$T(\hat{B}, \hat{C}) = \hat{B} \cdot \hat{C}$$

The result of the trigger function is then compared to an established configurable **threshold** t or the result of a second reference function if determined by the system policies (see Algorithm 4.1). The reference function has as output system reference values and states that are seen as the target for maintaining some of the important system parameters. In the present example, a configurable threshold value is considered for simplicity. Hence, when $T(\hat{B}, \hat{C}) < t$, the system would take a decision to adapt. In this example, the dot product can be understood as similarity between baseline and current vectors. The closer it is to one, the less change there was in traffic behavior as shown in Figure 4.3. A diagram illustrating the work flow of the strategy, considering the CDN scenario and the replacement of surrogates, is shown in Figure 4.4. After fixing the baseline and comparing with the current vector, the system trigger the process of adaptation when the threshold is reached. If not, the system continues monitoring and testing vectors. When an adaptation is triggered, the next vector is fixed as baseline.

Figure 4.3 – Dot product of baseline and current vectors

```

BEGIN
  currentNetworkState <- INTEGER []
  baselineNetworkState <- INTEGER []
  threshold <- INTEGER

  IF dotProduct (currentNetworkState, baselineNetworkState) < threshold
    THEN
      triggerAdaptation()
      baselineNetworkState <- currentNetworkState
  ENDIF;

END.

```

Algorithm 4.1 – Trigger Procedure

Figure 4.4 – Basic concept of the virtual manager

We chose to use such a simple but yet efficient function. This is an important tradeoff in our design.

4.5 The case for CDN Virtualization

Virtualization and particularly OS virtualization has been around since the early days of computing as witnessed by those who developed software for or used mainframe computers in the 70's. More recently, VMWare emerged as a popular technology to offer different OSs over personal computers.

When using network virtualization, the introduction, expansion, collapsing or removal of a network is an operation that affects others as these are merely logically separated. Each virtual network would then have the liberty to install its applications, services, routing protocols, naming, billing, etc. In the following, we expose some of the benefits of virtualization in general:

1. There is clear limitation in terms of choices providers are faced with when dealing with the introduction and testing of new services. They would benefit from the use of virtualization to build specifically tailored content networks, giving them more flexibility and a chance to distinguish themselves from the competition;
2. Changes to virtual networks are software based and hence are more accommodating to frequent demands.
3. The emergence of new business models and opportunities inherent to such separation between the physical and logical resources gives costumers new options among the available solutions.
4. The co-location of different virtual networks should enable savings, facilitate exchange points among these networks when needed and ultimately reduce maintenance and human resource costs.
5. Concurrent architectures are better than a single one. The management of several small networks running independently within the same physical space is easier than managing a big and complex multi-service network.
6. A given provider may decide to introduce a new value added service and does not need to coordinate this with the other networks or run the risk of disturbing existing services. The use of virtual platforms speeds the time to market of such new services. Note that the physical provider may itself run a virtual network for the management of the virtual networks of the content providers.

Furthermore, there are additional specific benefits for providing virtualization specific to the application of Content Distribution Networks:

1. The outsourcing of the physical infrastructure should work in the benefit of the CDN providers as they do not have to deal with network failures and management. They nonetheless, need to manage their services in order to ensure customer satisfaction, know when they need to ask for further resources or release idle ones.
2. There should be some befits also if one considers that various service providers may interact with each other and negotiate new services or resources dynamically. One may think of a content provider asking for a feed from another one to cover a new event in a remote place where it currently has no presence.

3. A CDN provider may negotiate with more than a physical provider and combine their networks to build a single virtual network. This gives the CDN provider independence in choosing its underlying transport provider(s) and the possibility to reach new customers previously not covered.

To summarize, virtualization is a mechanism or a design concept that may be used at different system levels including: server design, allocation of communication resources, resource management, operating system design, dynamic coverage expansion and negotiations among providers of both physical and content objects. It is this same flexibility that makes virtualization a powerful and hard functionality to fully manage, develop adequately and explore its potentials.

4.5.1 Related work

The Internet has grown rapidly, however it has occurred in a disorganized way. This lack of organization has negative consequences, such as the increase of network complexity (including the increase of its management cost), and the wide difficulty of applying changes in the network core infrastructure. Coordination is usually a requirement for the deployment of some new technologies and protocols (for example: differentiated services, IP multicast, secure routing and IPv6). The cost of implementing such a technology is usually not backed up by revenue increase since the service will only work when coordinated among networks. Based on this problem, Feamster et al [69] propose a connectivity model where infrastructure providers are separated from service providers. In this view, the coordination of infrastructure upgrade is made easier since it will not require several entities to agree simultaneously with the change. This decoupling allows each provider to implement its own network architecture and even different architectures for different services, resulting in a virtualization of the network infrastructure.

Using the same approach of virtualization through the separation between service and infrastructure providers, Zhu et al [201] propose a three-layer network model called Cabernet. The infrastructure layer is connected to a service layer through the intermediation of a connectivity layer. This connectivity layer is responsible for providing a single view of the network (multi-provider infrastructure) for the service layer and also for guaranteeing the quality of service requested. Zhu et al provide a discussion on the use of this model for IPTV delivery using the traditional but not widely deployed

multicast in the network layer as a means of video distribution, which is much more effective than performing multicast in the application layer.

Another approach for virtualization, proposed by Carapinha and Jiménez [29], is the formalization of the basic components of network virtualization as links (physical and virtual ones), nodes (substrate and virtual) and networks. This allows for the creation of fully virtualized network on top of a substrate. The virtual network provider will find the appropriate means for establishing the service requested by the service provider and allocate resources from the infrastructure provider. Carapinha and Jiménez provide a tentative list of which parameters are required for the creation of a link (endpoints and traffic characteristics), for nodes (ID, physical location and CPU/Memory/Storage capacities) and for networks (ID, start time, duration and levels of reliability and preemption).

Server level virtualization is an expanding market. Companies such as HostingAce offer Virtual Private Servers (VPS) as a solution for webmasters, designers, developers, and business owners. They charge a flat rate per month, for CPU, memory, disk space, and bandwidth allotments, which may be utilized for any number of websites in any way that a client desires. They also allow for the creation of multiple isolated Virtual Private Servers (VPS) on a single physical server to share hardware, licenses and management effort with maximum efficiency. Each VPS performs and executes exactly like a stand-alone server for its users and applications as it can be rebooted independently and has its own root access, users, IP addresses, memory, processes, files, applications, system libraries and configuration files.

In the telecom arena, virtual circuits have been seen as logical connections with possibly varying capacities while sharing the same physical circuits. Similar logically isolated tunnels have been used to offer the separation of traffic between concurrent users and increase security. Such concept has been at the heart of communication technologies such as Virtual Private Networks (VPNs), Virtual LANs (VLANs) and MPLS tunneling. A VPN user is then under the false impression that its network is isolated from others and that it has all the network resources for itself. In a way, virtualization is all about concurrent but safe isolated sharing.

One approach for the use of network virtualization without altering the business model is the use of Netlets [180], which are programmable containers for network protocol stacks. The container is composed of a data plane and a control plane. This

allows the rapid reorganization of overlay networks, thus allowing virtualization and fast deployment of novel networks.

In this context of network virtualization, there is a paradigm called Software defined networking (SDN). In SDN, the main idea is to remove the control plane from each switch and put it in an external and centralized entity that enables a new programmability approach taking decisions based on global network status. With this, SDN makes possible the creation of a network infrastructure able to adapt according to future network demands in a simple way [117].

There is a work in the SDN domain called virtual tenant network (VTN) [65]. It allows operators to design and deploy virtual networks for their customers without needing to know the physical network topology or underlying operating characteristics. VTN is an open source product from the OpenDaylight project (Extreme Networks SDN Platform). A CDN provider can take advantage of VTN as an enforcement technology that helps manage the virtual CDNs. With VTN, once the network is designed, it will automatically be mapped into underlying physical network, and then configured on the individual switch leveraging SDN control protocol.

An architecture for CDN sharing, called peering, is presented in [25]. In this paper, authors cite known proprietary CDN networks and deliver an open, service-oriented architecture that allows inter-CDN communication and Service Level Agreement (SLA) negotiation. However, the paper does not present protocol solutions or implementation results.

Finally, Plagemann et al [144] advocate that the basic operations of a CDN, such as content creation, retrieval, modification and (re)location should be part of the underlying infrastructure, then called Content Networks. Authors assert that the field lacks a thorough common terminology and try to provide one.

4.5.2 Virtualized CDN Change Detection

We test the proposed module (change manager) of relevant network changes detection in a virtualized CDN model. This provides the **dynamic nature** of CDN and the decision on when to adapt is provided by the change manager, as described earlier. Chapter 5 presents the framework description and functioning, providing a different approach for a dynamic CDN, but still using the change manager. In this CDN virtualized model, the

surrogate servers may now be virtual machines, forming the Virtual CDN. Thus, new surrogates can be created and added to the CDN infrastructure as operating conditions change. But, most importantly, the surrogates can be repositioned, adapting the infrastructure to new user demands and resource usage states. Thus, during a flash crowd event, for example, it should be possible to replace surrogates to keep them closer to the source of the event, decreasing the startup delay experienced by clients.

The change manager is responsible for detecting significant changes in network conditions and triggering the necessary CDN resources adaptation. Thus, it interfaces with the host physical infrastructure through resources manager, in order to allow the dynamic allocation and release of resources (i.e. surrogates). The Manager receives information from the CDN resource Monitor including important network traffic metrics, and based on its policies, may therefore decide whether and how to perform adaptation. In the special case of Virtual CDN, we took the adaptation as replacement of surrogates, i.e., running the placement algorithm with current network and request data to place surrogates in the available nodes. This special case is shown in Algorithm 4.2.

- a. Collect CDN information from DB Monitor (probes)*
- b. Detect the pattern (utility function from a set of information) and fix it as baseline*
- c. Continuously monitor same information and compare it to baseline*
 - Information in a set is weighted according to a window function (weight) and a window size*
 - Windows size and function may differ in baseline and probing*
- d. If probe value abruptly changes (trigger function) in relation to baseline*
 - Adapt topology (replacement)*
 - Fix new baseline*

Algorithm 4.2 – The adaptation algorithm

4.5.3 Simulator

All the evaluations presented in this work were performed in P2PCDNSim simulation tool [152] (see appendix I for a detailed description of the simulation tool). This tool was developed during this work and it is designed to simulate and test complex CDN and other overlay scenarios. It is a Java discrete-event based simulation tool used for engineering and debugging CDN projects, multiple CDNs and P2P network overlays. It also provides clear separation between layers, facilitating the process of implementing and testing new strategies; an easy GUI to create new scenarios; a collector for a set of built-in metrics with real time display of graphs during the simulations while new metrics

can be seamlessly added; a realistic representation of Autonomous Systems, AS aware routing and AS-cross traffic monitoring; and a GUI tool for observing a running simulation by monitoring relevant metrics and their color coded evolution on the graphs¹.

The simulator adopts a number of abstraction levels. There is a clear separation between these different overlays implementations and the common underlying support strategies and functionalities. From a structural perspective, the simulator has been divided into three main parts: core, network and overlay (see Figure I.2). At its core, we find the clock, an event scheduler and data collectors. Above this lies the network level that faithfully mimics a communication network, topology information, protocols and links properties such as their symmetry, capacity, delay, packet loss, congestion, jitter and throughput information. All TCP functionalities including congestion control, packet retransmissions and timeouts are supported according to the RFC5681.

The third abstraction level is that of the overlay, i.e., the actual simulation target. CDN entities like origin server, surrogates and monitor are implemented as process by extending special classes provided by the simulator, according to object oriented paradigm. These processes are then mapped onto their discrete events by the simulator. At the simulator's base layer, a number of XML based configuration files are used to select among the supported strategies, scenario definition, expected output metrics, reports and logs with detailed event tracing, etc. For instance, the XML request file contains a list of tuples formed by client, object and time. During the simulation, the simulator will spawn a client at that specific time, which will, in turn, request that object.

The simulator was validated by comparing it with two reference models, according to [159]. First, we compared our networking protocols, such as the TCP implementation, to that from the known network simulator (ns-3). The CDN overlay was compared with CDNsim [168]. See [152] and appendix I for details.

4.5.4 Evaluation

To evaluate this CDN virtualization concept, a Virtual CDN designed for video distribution was implemented in the CDN simulator. The virtualization model implemented is very simple. The target of the system is to decrease network usage, so the main metric watched by the monitor is the traffic at the links. The adaptation is performed

¹ See <https://youtu.be/2ZovxyHyJrg>

by replacing the surrogates over the available nodes when a significant network state (pattern) change is detected.

The replacement is achieved in the same way as for the placement: using one of the placement algorithms implemented in our simulator. In some previous experiments with the simulator (see appendix II), when we compared those, we verified that the HotSpot with radius 1 (number of hops) has a similar performance to that of the Greedy algorithm, but at a lower computational cost. Thus, we used the Hotspot strategy in the first scenarios. The placement algorithm uses the information regarding the last requests seen to decide where to setup a surrogate. However, old requests information, i.e. those before a stipulated amount of time, are discarded to avoid interference from an old behavior that no longer represents the system state.

All the information regarding traffic in the system provided by the monitor is mapped into a vector. This represents the load on each link being monitored. The vector is then normalized and the utility function is defined as the dot product between the vectors from baseline and current probes. If the dot product is below an established configurable threshold, the replacement is triggered. The threshold is set by the CDN operator.

4.5.4.1 Scenario and Trace Data Description

For all the following scenarios, we adopted a network topology with 500 nodes and links of 1000 Mbit/s. We placed the origin server at a center node, according to min-K center placement strategy. The topology has 10 autonomous systems and was generated with the IGen Tool and is depicted in Figure 4.5. We used a non-cooperative pull-based outsourcing strategy. The video objects have a constant bitrate of 300 Kbits/s and follow a Zipf-like size (or popularity) distribution. The monitor collects information every 1000 seconds. To avoid interference from old requests on new replacement decisions, as described in section 4.5.2, we only use the requests from the last 1000 or 2000 seconds (two scenarios), hence shortening the system's memory. In addition, the experiments were performed with 6 and 10 caches whereas the comparison threshold was set to 0.9, after some empirical tests. To determine the threshold, some previous experiments were performed in order to observe the behavior of the dot product. Then, by comparing the progress of values the dot product obtained during the simulation with the start and end of flash

crowds, where they reached values typically above 0.9. These values is highly dependent of the scenario being test and the number of nodes, topology, number of requests, size of objects, etc, have all influence in the overall values obtained. The operator must observe the behavior of the system to have an idea of the “best” threshold.

Figure 4.5 – Topology used in simulations

For comparison purposes, we also implemented a virtualized model where adaptations are always performed at fixed time intervals. This amount of time is a parameter called virtualization time. We call this scenario **static virtualization** as opposed to dynamic virtualization described above. Also, we have a scenario, called none, with no replacements taking place. In this scenario, the input for the placement algorithm (initial placement) is the actual entire request file (i.e., it considers the requests that will take place during the simulations, just as if it had *Omniscience*), which means that the cache will always have the file requested. Thus, it works as a reasonable prediction of future (since all videos requested will already be available in the surrogate) demand and is used as a baseline for comparing static and dynamic virtualization, both of which perform placements based on recent history. It is not, however, a perfect prediction since the location of the surrogates will not be perfect throughout the simulation (it may not be considered a *perfect* scenario since the surrogates do not change location and therefore

may not “move” towards the vicinity of the clients). Consider this scenario a case where replacements are performed only once (in the beginning) and the window is the whole simulation time.

Each scenario has one million requests distributed in 24 hours. Most requests arrive at a rate of 5 req/s from all nodes. However, in two time intervals, between 5 and 8 hours and between 12 and 15 hours approximately, the rate suddenly increases to near 50 req/s and to almost 70 req/s respectively. Those are the flash crowd events, modeled the same way as in [33]. Differently from the rest of the simulation, the requests of flash crowd moments come from a specific AS in the topology. Thus, they are concentrated in two of the 10 ASes, one at each time interval. This should force the system to adapt itself by moving surrogates closer to that AS at the flash crowd, and then, when it finishes, distributing them in the whole topology subsequently. All experiments conducted in this work used the Java-based discrete-event simulator described in Appendix I.

Table 4.1 – CDN Virtualization scenario configuration parameters

	Scenario 1	Scenario 2
Simulation time	24h	
Link capacity	1000Mbit/s	
Baseline request rate	5 requests/s	
1 st Flash crowd request rate	50 requests/s	
1 st Flash crowd start time	5h	
1 st Flash crowd end time	8h	
2 nd Flash crowd request rate	70 requests/s	
2 nd Flash crowd start time	8h	
2 nd Flash crowd end time	12h	
# of surrogates	6	10
Video bitrate	300Kbit/s	
Threshold	0.9	

4.5.4.2 Results

Figure 4.6 (b) shows the behavior of the mean startup delay during the simulation in a typical dynamic virtualization scenario. As it can be seen, during flash crowd events the startup delay understandably decreases. This happens because the majority of clients, which are located on the same AS (source of the flash crowd event) can reach the surrogate more rapidly as they are closer (due to surrogate relocation), at the expense of the ones outside that AS. Thus, the other clients, the ones located outside of the source, will have a higher startup delay. This is expected because during the flash crowd, most of

the surrogates will be closer to the AS where it is originated, then, decreasing the mean startup delay. As the number of clients with less startup delay is much bigger, the mean is smaller for this metric. Note that the number of surrogates is not changed throughout the simulation time.

The results are very similar for network usage (the target). As, in these scenarios, there is no influence of external traffic in the network, the network usage is closely related to the observed startup delay. This only happens due to the low level of congestion in the network, and having more requests means greater availability of content through peer-to-peer connections.

Figure 4.6 – Average number of requests and video startup delay

Table 4.2 – Comparison among virtualization strategies

	None	Static Virtualization	Dynamic Virtualization
Monitor	N/A	1000 s	1000 s
Replacement	N/A	Each 2000 s	According to trigger function Considering the last 2000 s in requests for placement algorithm
# of replacements	1	44	15
Transferred bytes	8869304 MB	7201875 MB	6997881 MB

The replacement of surrogates in a CDN is a costly task. It involves reallocation of resources. Thus, the virtualization time parameter must be a balance between a faster adaptation, which is related to more replacements, and the cost of the replacement. Concerning this issue, dynamic virtualization could achieve very good results. Table 4.2 shows the total amount of traffic in the simulation and the number of replacements. Dynamic virtualization performed 15 replacements while static virtualization performed as many as 44 on the same scenario. So, it has a better result with fewer replacements. In other words, it performs the adaptation at the right time, saving costs resulting from resource reallocation. Figure 4.7 shows all-network bytes transferred results for the scenarios with six caches and all types of virtualization. Figure 4.8 shows results with ten caches.

Figure 4.7 – Comparison among virtualization strategies (6 caches)
All-network Bytes Transferred

Figure 4.8 – Comparison among virtualization strategies (10 caches)

According to Figure 4.7 and Figure 4.8, both dynamic and static virtualizations achieve very similar results (20% decrease) in terms of network utilization. Since static

virtualization induces considerably more reallocations, dynamic virtualization is by far the most recommended method. The migration process imposes a penalty to the system, thus, the decision of reallocation could also take this cost into consideration. This is something that can be included in policies DB.

For startup delay, results are shown in Figure 4.9 and Figure 4.10. As it can be seen, the results are closely related to network bytes transferred since there is no influence of external traffic. Furthermore, it achieved a decrease of about 20% (best case).

Figure 4.9 – Startup delay (6 caches)

Figure 4.10 – Startup delay (10 caches)

4.5.5 Discussion

The proposed virtualized CDN model was designed to adapt the CDN infrastructure to the conditions of the network, mainly during some flash crowd events. The goal was to test the change detection strategy, since the overall idea of dynamic CDN is not new. The adaptation (the dynamic nature of this particular CDN) is performed by the reallocation

of resources. It can be either static, i.e. when adaptations occur at fixed time intervals, or dynamic, when performed according to a monitored trigger function.

The utility function and triggering algorithm based on the dot product of the current and baseline usage data of the network have shown to be effective at adapting the topology of the virtualized CDN. A CDN provider can use this technique to detect significant changes in network conditions and perform reconfiguration of resources, like we did by replacing surrogates. The main contribution so far is this simple detection algorithm.

However, since the data used to feed the monitor in this study is not realistic, because it was collected in all links of the topology, further studies included testing our technique with restricted data, as described below.

4.6 Improvements and case study

Adaptation is a costly change in CDN infrastructure, so it should not be performed very often. A sudden change in traffic pattern with small duration in time may trigger undesired replacement with the above replacement algorithm. As mentioned earlier, a policy may be defined to avoid a high frequency of adaptation (granularity) or a penalty to server migration can be inserted. It is also possible to increase the period between checks of trigger function. However, these approaches suffer from a major drawback: they may decrease the responsiveness of the system to changes in network patterns, leading it to poor performance. The prime goal in detecting such changes accurately is to trigger the replacement algorithm (a CDN adaptation) only when truly necessary, ignoring insignificant changes (“false positives”).

Another important issue is the metric used to calculate the vectors. The network traffic used in the simulated environment is not feasible in real world and it could also be easily polluted by irrelevant traffic, not generated by the CDN. Typically, a CDN operator will only be able to access the network usage for each desired link when it is also the network operator and its CDN has no surrogates outside of its own infrastructure. Thus, the next sections will also present some possible solutions to this problem. Another important metric is the CPU usage on cache servers (for dynamic generated content), although difficult to be measured in a simulated environment.

4.6.1 Scenario description

A scenario was designed to observe the behavior of next approaches in situations that contemplate the necessity of the new developments, as described below. In the scenario, two flash crowds take place, and in case the traffic boom is detected earlier, the content placement can be improved sooner and the CDN traffic, directly associated with maintenance cost, is diminished. Additionally, we intend to observe the best manners to use real CDN data to evaluate its state and the contribution of a new metric for the same evaluation.

In a global topology, 13089 clients of a given CDN are spread among geographically remote areas (Autonomous Entities - AS), what causes the use of many hops and, consequently, yields a big traffic (cost) over the CDNSP network. In such a scenario, massive requests are more representative to the network state, and contribute to decaying the users' quality of experience. To prevent this decay, the surrogate replacement occurs, and as important as the replacement is its detection. This is the reason that leads us to use the flash crowd scenario to evaluate the sliding window proposal. Figure 4.11 represents how the elements of the simulation are distributed. In the figure, the bright large circles represent the router nodes, the dark large circles stand for the router in which the origin server is hanging, the triangles are the regions among which the users are distributed and the small circles represent the users, whose amounts are shown in the caption.

Figure 4.11 – Overview of scenario

The 13089 requests that rule the simulation are split as follows: 8077 randomly divided requests spread among all nodes; a 100 second flash crowd of 1007 requests from

AS3 users (from time 900s to time 1000s) and another flash crowd from time 7000s to time 9000s, holding 4005 requests from AS7 users (see Table 4.3). Figure 4.12 below illustrates this distribution.

Table 4.3 – Common configuration parameters for case study

Simulation time	9000s
Link capacity	1000Mbit/s
Baseline requests	8077 requests
1 st Flash crowd AS	AS3
1 st Flash crowd requests	1007
1 st Flash crowd start time	900s
1 st Flash crowd end time	1000s
2 nd Flash crowd AS	AS7
2 nd Flash crowd requests	4005
2 nd Flash crowd start time	7000s
2 nd Flash crowd end time	9000s

Figure 4.12 – Amount of request triggered per timestamp

All the experiments described next were run apart, that is, the improvements' evaluations were not cumulative. In other words, the stable version of the simulator was the base for three different implementations, not interacting with each other.

4.7 Round-trip delay

A metric which resembles more to the end user's Quality of Experience was chosen to increase the amount of analyzed data. Initially, one-way delay² was considered, because it is the most straightforward indicator of how much time end users wait before actually accessing the expected content. However, since there is a concern in using real-world feasible information, this metric was disregarded – while extracting delay from a

² RFC 2679 - <http://www.ietf.org/rfc/rfc2679.txt>

simulated environment is simple, this measurement, even in a short sized real-world environment, would demand expensive synchronization equipment. Another improvement promoted by the use of a new metric is the correction of a flaw in the previous evaluation.

A metric that approximates the delay the user will experience and is more feasible to collect from actual network environments is the Round-trip Delay (RTD)³ between the nodes. The RTD between pairs of nodes is obtained using the Internet Control Message Protocol (ICMP), a lightweight protocol which permits the time taking of the communication between two network nodes. A setback to consider in using this strategy regards the network nodes configured to discard ICMP packets. In real world scenario, we expect to circumvent this issue by monitoring packets in the delivery process.

The option to add the Round-trip delay should provide increased representativeness of the network state, extracted from time to time to assist management aspects of the model. In other words, the new metric should be preferably merged with other metrics in order to get a more realistic quotient of CDN usage and user QoE. Then, a formula would gather and combine all the collected metrics in an appropriately weighted manner. Our approach for these steps is shown later in this section.

Once a set with all N routers relevant is defined, the round trip delays from all involved routers to all other routers are gathered in a $M_{(i,j)}$ matrix representation where, $0 \leq i < N$ and $0 \leq j < N$, such as the example observed in Figure 4.13. The rows of the matrix are then separately added to become a vector R with as many elements as there are columns (denoted as j above):

$$R_j = \sum_{t=1}^N M_{(t,j)} \quad (3)$$

This sum is represented in the figure by the grey arrows, while the resulting vector is showed by the numbers between the brackets.

³ RFC 2681 - <http://tools.ietf.org/html/rfc2681>

	R1	R2	R3	R4	R5
R1	-	8	2	9	2
R2	2	-	3	4	2
R3	4	5	-	11	3
R4	5	2	3	-	2
R5	3	1	4	3	-

$M_{(i,j)} =$

$R = \{ \begin{matrix} \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 14 & 16 & 12 & 27 & 9 \end{matrix} \}$

Figure 4.13- Example of the vector extracted from the router x router RTDs

The extracted vector R becomes a snapshot of the network status in the measured time, and can be gathered with other vectors with different metrics, such as the total router traffic vector T , also with N elements. T was used alone at first, in order to be applied in equation (1) presented in the introduction of this section. Both vectors R and T have the respective baseline representations B_R and B_T . The baselines have the same structure of their representing vectors, but hold the values of the last representative state, most cases the last change detected on the network, of the evaluated metric:

$$B_R = \{r_1', r_2', \dots, r_N'\} \therefore R = \{r_1, r_2, \dots, r_N\} \quad (4)$$

and

$$B_T = \{t_1', t_2', \dots, t_N'\} \therefore T = \{t_1, t_2, \dots, t_N\} \quad (5)$$

where the elements marked with a single quote (') refer to the values collected in a past representative state. The baseline vectors are used to obtain Δ_R and Δ_T , the dot product between the baselines and the current state vectors, respectively from the round-trip delays and the total router traffic:

$$\Delta_R = R \cdot B_R \quad (6)$$

$$\Delta_T = T \cdot B_T \quad (7)$$

It is important to remember that both vectors as well as Δ_R and Δ_T are normalized to have possible assigned values between the decimal range from 0 to 1. The aim of this operation is to simplify the comparison of metrics which can have possibly very distinct

values (e.g. transfer time usually has a value of a few seconds, while transferred data may have values of millions of bytes).

Hereinafter, a function to join the observed change Δ for both metrics is used to weigh the most relevant attributes and come out with a single decimal value, ranging from 0 to 1:

$$f(\Delta_R, \Delta_T) = \Delta_R \cdot w_R + \Delta_T \cdot w_T \quad (8)$$

Where $w_R + w_T = 1$ and values of w_R and w_T are obtained empirically.

The resulting value of the function f is compared to a pre-defined threshold which will states whether an update of the baseline vectors will occur. In the positive case, the current vectors are now the most recent representative state and will replace their respective baselines.

The algorithm for the monitoring module is abstractly shown in the box Algorithm 4.3 below. An optimization for this algorithm was envisioned to yield less traffic between the nodes, but firstly, an implementation with the simplest complexity will be assessed, and from the initial results, possible improvements will be developed.

```

roundTripDelays <- INTEGER [numberOfRouters] [numberOfRouters];
RTDVector <- INTEGER [numberOfRouters];

FOR EACH Router R
  FOR EACH Router knownRouter
    IF R <> knownRouter THEN
      roundTripDelays[R][knownRouter] <- calculateRoundTrip(R, knownRouter)
    ENDIF;
  ENDFOR;
ENDFOR;

FOR EACH column IN roundTripDelays
  rowSum <- 0
  FOR EACH row in roundTripDelays [column]
    rowSum <- rowSum + row
  ENDFOR;
  RTDVector [column] <- rowSum
ENDFOR;

END.

```

Algorithm 4.3 – Collecting RTD to define replacement

4.7.1 *Implementation*

During the implementation of the round-trip delay metric, it was observed that sending ICMP packets to other routers (active measurement) consumed considerable network and valuable processing resources (in the simulated environment). The use of this metric imposed an unreasonable burden in the simulation scenarios and it would be best if that did not occur. Therefore, during the implementation another version of this metric was also considered, where the CDN traffic passing on a router was measured and the RTD metric was estimated based on the one-way delay from the source machine to this router. This allows us to choose between two similar metrics for RTD: one using a one-way delay in a simulator with a lower processing overhead (multiplied by two), the other using network pings (ICMP Echo and Reply messages). The first one is feasible for use in our simulator, the second one is suitable for a real network implementation and both should provide similar results.

4.7.2 *Evaluation and discussion*

After running several experiments using the 13089-request scenario described above, we encountered some problems when analyzing the results. Initially, the main problem was to find adequate thresholds and monitor interval values to efficiently collect significant data from the monitor, but after a series of simulations another problem appeared, one that is described next.

Since our main idea was to detect network changes through RTD variation we had to configure the topology in such a way that collected RTD values could be used as an indication for network changes. In other words, we choose to observe network usage variation RTD values which also change above an observable and relevant threshold. Tuning this turned out to be a difficult task in a **simulated environment**. Although it is considerably easy to configure a scenario where RTD values change according to the network changes and raise the flag of a threshold, to do that we had to change variables in ways that the result would be nonrealistic. For instance, we had to use lower link capacities and increase link latencies to values incoherent to the size of the network we are simulating, otherwise the changes in RTD would not be significant enough for detecting network changes. And even with those values RTD variations were too small and highly correlated to the changes in traffic, thus, virtually useless.

Our opinion on the matter is that a suitable way to deal with this issue would be to use a network traffic generator along with the simulator for including background traffic in the experiments. The extra network usage in addition with CDN overlay's traffic should result in a considerable RTD variation not correlated to the traffic, thus resulting in relevant monitoring data. Considering the time we could spend incorporating the network traffic generator to our simulator, and the importance of having results from the Real Data Restriction, coupled with the doubts regarding the feasibility of this strategy we decided to focus our experiments and efforts instead on Real Data experiments, described in next sections.

4.8 Weighed Moving Average vector

A possible improvement to avoid sensitiveness to small duration pattern changes in replacement algorithm is a weighted average vector utility function. The goal is to smooth out the detection in pattern changes by combining multiple probes to represent baseline or current vectors. Thus, a small sudden event may not have sufficient significance to force the system to adapt. The new utility function U_w is then defined by a combination of various vectors. The number of samples S is a parameter for the system. For the baseline, multiple probes within a window defined by the number of probes are collected right after the last replacement. The resulting vector of utility function is a weighted average of these sampled values. The weight is defined by a weight function $W(V, s)$ with the following feature: the closer the probe vector is to last replacement, the greater is the weight. The weight function should also agree to the following definitions:

$$\sum_{s=1}^S W(V, s) = 1$$

$$\sum_{s=1}^S dW(V, s) = 0 \rightarrow \text{Pareto optimal}$$

An example of weight function is $W(V, s) = 2 \frac{(S-s+1)}{S(S+1)}$. See Figure 4.14a for another example. The new utility function is defined as follow:

$$U_w: \mathbb{R}^{S \times n} \rightarrow \mathbb{R}^n$$

(9)

$U_s: \{U_1, U_2, \dots, U_S\}$, where

U_i is the result of utility function on probe $i \in [1, \dots, S]$

$$U_w(U_s, S) = \sum_{i=1}^S W(U_i, i)$$

The previous algorithm was a special case of this proposed improvement where only one probe was collected ($S = 1$) and the weight function is the constant 1 ($W(V, s) = 1$), see Figure 4.14b. The same procedure can be used to calculate the resulting current vector, using the same or different number of probes and weight functions. Algorithm 4.4 and Algorithm 4.5 compare the traditional and weighted moving average vector approaches for baseline utility function.

BASELINE-UTILITY-FUNCTION:

baselineNetworkState <- INTEGER [0,0,...,0]

baselineNetworkState <- **ProbeVector**

baselineNetworkState <- **Normalize(baselineNetworkState)**

Algorithm 4.4 – Conventional approach

BASELINE-UTILITY-FUNCTION:

INTEGER N: number of probes

INTEGER I: i^{th} Vector

baselineNetworkState <- INTEGER [0,0,...,0]

FOR EACH **ProbeVector** I < N

baselineNetworkState <- **baselineNetworkState** + $W(I) * \text{ProbeVector}$

 I <- I + 1

ENDFOR;

baselineNetworkState <- **Normalize(baselineNetworkState)**

Algorithm 4.5 – Weighted average vector for baseline

Figure 4.14 – Weigh functions: a) proposed improvement b) conventional approach

4.8.1 Implementation

The weight for the probes was defined according to an **exponential** function. The algorithm considers any number of windows and, after applying the formula, it normalizes the values to define the weights (see Algorithm 4.5). For the first proof of concept implementation, the formula was obtained empirically, and satisfied the case study.

The approach only begins to assess the traffic when the predefined amount of time windows is reached. Figure 4.15 compares the functioning of the sliding window sized 3, for a window length defined as 100 seconds, and its equivalent to a 300s sized window on the previous approach. While the single window approach produces the first usable collection in time 300s, and the next collections every 300s forth, the weighted strategy has a usable collection in the third 100s window as well, but is able to evaluate a new collection every 100s.

Figure 4.15- Collection begins when a complete set of time slices has elapsed

4.8.2 Evaluation and discussion

In order to have a fair comparison between the single window approach and the weighted one, the same use case – detailed below – was executed for both proposals and the same time interval for traffic assessment was selected for both, that is, if the single collection window is defined to have 300 seconds, the sum of all the windows in the weighted approach was as well set to be 300 seconds. By doing this, we state that both approaches have the same data to collect and compute over, setting equal conditions for comparison. This is shown in Figure 4.16 – both collection intervals are set for 300 seconds, while Single window collects every 300 seconds, weighted windows (whose effective collection also begins at T3, same timestamp as single approach) optimizes the collection every 100 seconds, gathering the data from the last 3 windows (hence, the same 300 seconds).

Figure 4.16- Data collection of single and sliding window approaches

The experiment with the following results was ran using a window width of 500s – 1x500s window for the single approach and 5x100s windows for the weighted – defined empirically from an observation of Figure 4.12. With this configuration, the weighted windows approach showed an improvement in placement, because of its fine-grained data collection and the ability to consider the collections with different degrees of importance (weights).

A number of charts were generated always contrasting the performance of both observed replacement schemas. For most of the charts, the x axis represents the elapsed

simulation time, and the behavior of the y axis should be checked against the requests and timestamp chart in Figure 4.12.

Once the purpose of this experiment is to compare how two approaches – the current and the weighted – behave to perform surrogate replacement. Figure 4.17 illustrates that, for the weighted windows, replacement took place more often and earlier if compared with the single window behavior. The weighted windows approach showed an improvement in placement, both because of its fine-grained data collection and the ability to consider the collections with different degrees of importance (weight). The surrogate replacement of the weighted approach was expressively better than the single window.

Figure 4.17- Replacement occurrence in the two approaches

For the weighted windows, the replacement happened more often and earlier comparing with the single window behavior. This amount of replacements took effect in making the CDN perform better even in the expressive flash crowd of time 7000s, as observed in Figure 4.18, Figure 4.19 and Figure 4.20, when the single window approach shows a performance decrease.

Figure 4.18- Compared total traffic

Figure 4.19- Compared cross network traffic

Figure 4.20- Compared inner network traffic

Likewise, because of the improved replacements, the weighted window approach also had better results concerning user experience. Figure 4.21 shows a slightly lower start up delay of weighted windows approach, compared with single window – 0.296 for the former and 0.311 for the latter.

Figure 4.21- Comparison of startup delay of both approaches

4.9 Usage Approximation based on restricted local information

The network metrics used so far, namely one-way delay and aggregated network traffic, are not feasible to be collected in a real-world experiment. A considerable problem with the network traffic metric used in simulation or even the proposed RTD is that it may not be available in real CDN scenarios. It is unlikely that the CDN operator will be able to collect network traffic information on all the nodes or RTD between them. Even if there could be a way to collect the network traffic through each and every node, another challenge would be to separate what is traffic directly related to the CDN. Therefore, these metrics cannot be used in a real CDN implementation for input to the decision of cache placement. New metrics need to be defined and evaluated, ones that can also be easily computed in a simulation as well as in a real environment. This section presents our ideas on how to tackle this problem.

An interesting approach would be to reuse the algorithm presented earlier while changing only how metrics are collected. Our next solution considers two metrics, one-way delay and aggregated network traffic. The first represents network latency and the second network traffic passing through all nodes. To reuse the algorithm we need to generate a vector with the new values collected in a new approach that will later be used as input by the change detection algorithm. The new values should be semantically equivalent as the old ones and must be easily accessible by any CDN. For network latency we believe that client startup delay would represent well. On the other hand, for network traffic we propose using all traffic sent by surrogates.

Although promising, both metrics have limitations in comparison to our previous metrics. For instance, the aggregated network traffic shows us exactly how much data passed through each and every node, from source to every client, and our new proposed metric shows only how much traffic passed through each surrogate. Remembering that, previously, we had all traffic that passed through all nodes, now we only have all traffic that passed through each node that has a surrogate attached to it. Thus, packets coming from node A to node C passing through B, where A has a surrogate and C has the client requesting content, would previously result on registering the traffic in all three nodes now will only be registered in A. We consider that it could be interesting to approximate as much as possible our new metrics with the old ones therefore we propose using network information along with traffic information to shorten the distance between what we had and what we have now. But first, an initial consideration is the possibility of traditional metrics usually available in a CDN, like traffic sourced by the surrogates, location of clients and startup delay, bring a good approximation concerning information completeness scenario. Perhaps no adjustments are necessary. If considered individually, they are not sufficient, but perhaps a combination of some of them will suffice, by bringing higher quality information.

Consider that we have P surrogates and M total clients and that we will collect data every T seconds. M_p is the number of clients that requested content from surrogate p during the last window, where $0 < p \leq P$. Since each client requests content from a surrogate, therefore the formula for the amount of traffic sent by every surrogate is:

$$T_p = \sum_{i=1}^{M_p} Tc_i^p \quad (10)$$

Where Tc_i^p represents the traffic from surrogate p to client i . Calculating this traffic is a very simple task for the surrogate. It could be calculated by adding up all data sent or even recording bitrates and time window for each bitrate used to send data to client. This way, every T seconds we have P values, that will compose the vector representing the status of the network regarding network traffic. To generate the latency vector we propose using the sum of all start up delays experienced by clients connected to the given surrogate. Therefore, the latency for each surrogate will be:

$$L_p = \sum_{i=1}^{M_p} S_i^p \quad (11)$$

Where S_i^p is the start up delay experienced by the client i when getting content from surrogate p . Again, calculating this metric should be very simple for the CDN. There could be a packet counter for each client or even a marker when sending a specific block of the content that could be pre-calculated. Another way to compute it is to change the user agent to send a simple packet signaling the start up delay. We have now two vectors and can proceed on using the strategy presented earlier.

This idea of combination leads to a second thought on how to approximate the network traffic metric. Another method of substituting the aggregated network traffic by another meaningful network utilization metric is to measure the amount of traffic network clients are receiving and multiplying this amount by the number of hops from the client to the cache. The amount of traffic sent to the client is known by surrogates, thus, it can be easily collected by the Monitor. Thus, formula (10) will change to:

$$T_p = \sum_{i=1}^{M_p} Tc_i^p H_i^p \quad (12)$$

Where H_i^p is the number of hops between the client, or client's AS, and the surrogate server. However, this is a costly metric, since periodically measuring the number of hops from each client to the cache it uses may significantly increase network usage. Therefore, to circumvent it, the surrogates, when communicating with clients during content retrieval, can count the number of hops to reach them. Since we are interested in quantifying the network traffic from the network operator point of view, by multiplying the traffic directed from the surrogate to the client by the number of hops to reach it, the result could be used to fill the matrix as described in section 4.5.1 and bring a good approximation. Sampling is an additional technique that we will consider in future extensions to reduce the monitoring overhead. If, for instance, client applications cannot be modified, or even the overhead of the previous approach is too high, another approximation can be used: all clients are considered one hop away from the edge router in their respective ASes and the number of hops from the cache to the edge router of the client's AS is measured periodically (i.e., a traceroute to the first machine in the same AS of a client). This results in very few measurements, since each cache will need to measure

network distance for each AS it has clients downloading content from, not for all clients. If the cache and the client are in the same AS, a value of 1 is considered for the distance. See Algorithm 4.6 below.

```

surrogateNetworkUsageSum <- 0

PRODECUDRE INTEGER getNetworkDistance(Client client)
  IF isDistanceCalculated(client) <> true THEN
    knownRouter <- getLastKnownRouter(client)
    roundTripDelays [knownRouter] <- calculateRoundTrip(surrogate,
knownRouter)
  ENDIF;
  Return distanceToClient(client)
END;

FOR EACH Client C Served by the Surrogate
  networkUsageSum <- networkUsageSum + clientTraffic *
  (getNetworkDistance(client)+1)
END-FOR;

```

Algorithm 4.6 – Using approximated information to collect network data

4.9.1 Implementation

Five experiments were proposed to detect the changes comparing the previously presented method, described in section 4.5.4, with variations of the proposed improvement. The experiment named NONE is the baseline for the further experiments, and it is defined to run without resource reallocation during the simulation. Surrogates are placed, based on prediction of all future requests, in the beginning of the simulation and remain there during all the experiment time. The second experiment, labeled BITSENT, also establishes an observation baseline, since it was built upon an optimal omniscient environment – the previous work described in section 4.5.4; therefore, the values extracted in this configuration are likely to be the most relevant for management purposes. The three remaining experiments are variations of the real network data approach (i.e with only limited information). They combine the use of real traffic and startup delay metrics. Configurations considering knowledge of only traffic (REALDATA_T100_SD0) and only start up delay (REALDATA_T0_SD100) were evaluated, so the contribution and impact of each metric can be individually identified and compared with the established baselines. Additionally, an empiric configuration of mixing up both metrics to obtain a weighted network behavior point of view was set up, using 80% relevance for traffic and 20% relevance for startup delay (REALDATA_T80_SD20). With this configuration we hope to observe the impact

(positive or negative) in using more than one metric to evaluate the network, and of course, continue setting different configurations based on previous experiments results in order to find out the best balance between the considered metrics.

The scenario described in section 4.6.1 was used to validate the use of restricted information to assess and manage CDNs. In order to obtain the best configuration parameters for the simulation, a number of experiments were ran. Hence it is possible to extract how the use case will behave, to set the parameters properly. Using these results, the setup of another needed configuration was assigned – the threshold for network status changes was set to 0.75 in the real data configuration, the best value for the configuration, as observed empirically.

Excited about the promising results (further detailed next), two variations of the regular scenario were designed specifically for this improvement, in order to validate the changes detection along with differently dimensioned scenarios. In the STRETCHED SCENARIO, the goal was to observe the behavior of the usage of real metrics to notify network changes for management purposes in a more lasting simulation and with more user requests. The SHRUNK SCENARIO, on the other hand, is an attempt to intensify and concentrate the amount of requests per time – a bigger amount of requests in smaller simulation time. Both new scenarios follow the guideline of having flash crowd situations amidst their approximate 30000 requests (differently of the regular scenario's 13000 requests).

4.9.2 Evaluation and discussion

All scenarios are configured on the same topology of earlier description, have a general background traffic and flash crowds positioned in a manner that the traffic boosts occur during a big background traffic and in lull moments of network general traffic. Due to clarity issues, if in any of the figures shown in this section there is more than one strategy labeled with the same color that means that both strategies had statistically equivalent results. The test used was the Kolmogorov–Smirnov [113] and strategies were considered equivalent if the test provided a p-value < 1 .

The charts below were generated to observe how the management behaves concerning surrogate replacement – the main indicative that useful actions are being taken

for the sake of CDN performance improvement. Again, for most of the charts, the x axis represents the elapsed simulation time, and the behavior of the y axis should be checked against the requests and timestamp.

The following sections present the results of the three scenario variations.

4.9.2.1 Regular scenario

The timeline below (Figure 4.22) permits an overview of which configurations lead to the expected replacements, which in this scenario, should happen about time 900s and time 7000s (the flash crowds). The results show that NONE configuration doesn't end up with replacements, as expected, since it has no data collection to do so. All the placements that happen in time 0s are due to the startup of the network. The replacements that take place at the ending of the simulation time are probably false positives, once the network presents representative changes from current traffic to zero. Also as expected, the first flash crowd (time 900s) isn't noticed because of the single 500s data collection window, which ignores the 100s duration traffic boost. BITSSENT configuration, our baseline, performs the replacements after detecting enough amount of traffic after the second flash crowd, while REALDATA_T100_SD0 and REALDATA_T80_SD20, even using limited data, are able to follow the replacements of the baseline by the same timestamp. REALDATA_T0_SD100, due to assigning too much weight to a not so relevant metric, only performs the replacement two collection frames (two 500s windows) later – which is an indicative that weighting adequately many metrics is probably better than using a chosen one individually. The same explanation is worth for the replacements unnecessarily performed in time 9500s by configurations REALDATA_T0_SD100 and REALDATA_T100_SD0: weighing the metrics properly is more accurate than giving full relevance to a single metric.

Figure 4.22- Surrogate placement for all configurations along time

The traffic metric charts show the impact of each collection strategy in network's overall performance. Scenarios with no relevant difference ($p < 1$) were represented with the same color. As expected, the behavior of Total Traffic (Figure 4.23) and its components Cross Traffic (Figure 4.24) and Inner Traffic (Figure 4.25) follow equal until the first relevant CDN action is taken (from Figure 4.22 one can see it is in time 7500s). In Cross traffic chart, the difference becomes noticeable from the point in which the first replacement occurs, by that time, all the traffic of the simulation was inside the requester AS, behavior reflected in Inner traffic chart. In time 7500s, the occurrence of the first surrogate replacement, sets a milestone in simulation. From this point forward the approaches diverge: NONE and REALDATA_T0_SD100 show a traffic increase whereas BITSENT, REALDATA_T100_SD0 and REALDATA_T80_SD20 configurations keep similar, using less bandwidth. Figure 4.22 also shows that when the demand is critical (flash crowd at time 7000s), the configuration that considers only start up delay fails to point out a replacement necessity – an issue possibly corrected with a fine tune of start up delay threshold. Therefore, REALDATA_T0_SD100 behaves like NONE (no surrogate replacement), and the other three configurations successfully detect the need for replacing, improving the afterwards bandwidth usage observed in the chart below.

Figure 4.23- Total traffic comparison for the real metrics experiments

Figure 4.24- Cross traffic comparison for the real metrics experiments

Figure 4.25- Inner traffic comparison for the real metrics experiments

A simple observation of ‘playback continuity’ metric leads to tricky conclusions. The cases without data collection for replacement and using real data only considering start up delay metric obtained the best results, while a decrease of performance was observed in the other configurations. In a more thorough observation, though, we noticed that the configurations with replacement taking place, generating less workload (fewer bandwidth usage) and improving the experience of the users sitting on the same AS whose requests triggered the replacement. A big number of other users located on different ASes suffer an instability that causes a decrease in playback continuity, as well as a large variability, observed by the standard deviation metric shown in the chart. On the other hand, even yielding more traffic and bandwidth usage, the configurations that don’t trigger replacements during the flash crowd at 7000s are still able to keep the 100%

playback continuity. The mean and standard deviation for playback continuity for all the configurations are depicted in Figure 4.26.

Figure 4.26- Mean and standard deviation for playback continuity

4.9.2.2 Stretched scenario

The STRETCHED SCENARIO is composed of 30047 requests during 25000 seconds, split as follows: 20026 request as background traffic, 1007 requests from AS3 representing the first flash crowd during the interval between times 900s to 1000s, 4005 requests from AS7 in the second flash crowd from times 7000s to 9000s and the last flash crowd comprising 5009 requests from AS7 from time 20000s to time 22000s. The first flash crowd is positioned in a high background traffic moment, the second one in an intermediate network usage state and the third, in a lull state (see Table 4.4). Figure 4.26 illustrates the new request distribution.

Table 4.4 – Configuration parameters for stretched scenario

Simulation time	25000s
Link capacity	1000Mbit/s
Baseline requests	20026 requests
1 st Flash crowd AS	AS3
1 st Flash crowd requests	1007
1 st Flash crowd start time	900s
1 st Flash crowd end time	1000s
2 nd Flash crowd AS	AS7
2 nd Flash crowd requests	4005
2 nd Flash crowd start time	7000s
2 nd Flash crowd end time	9000s
3 rd Flash crowd AS	AS7
3 rd Flash crowd requests	5009
3 rd Flash crowd start time	20000s
3 rd Flash crowd end time	22000s

Figure 4.27- Request distribution for stretched scenario

The purpose of this experiment was to observe the power of the metrics to detect changes in a larger scenario. Figure 4.28, Figure 4.29 and Figure 4.30 show respectively total traffic, cross traffic and inner traffic for the omniscient (BITSSENT) and the total traffic real metric (REALDATA_T100_SD0). The results were similar to the regular scenario, but the BITSSENT monitor performed slightly better than REALDATA_T100_SD0. On the other hand, the real metric monitor was better in keeping the traffic local, resulting in more inner traffic and less cross traffic.

Figure 4.28- Total network bandwidth usage along simulation time

Figure 4.29- Cross traffic bandwidth usage along simulation time

Figure 4.30- Inner traffic bandwidth usage along simulation time

Figure 4.30 reports the dot products variations along time and shows when the replacements occur, when the value exceeds threshold for BITSENT and for REALDATA_T100_SD0. When comparing the number of replacements between the two approaches, BITSENT performed more replacements (6) than the real metric (4) and was able to detect the first flash crowd, while the real metric didn't. The second flash crowd was not detected by any. The reason is in the value for the dot product. REALDATA_T100_SD0 reached smaller values than the BITSENT: respectively, a peak of 0.5 and 0.8.

(a): Dot product along simulation time for BITSENT

(b): Dot product along simulation time for REALDATA_T100_SD0

Figure 4.31- Dot product along simulation time

The explanation for this behavior is in the fact that the designed flash crowds were not “crowded” enough. In the requests distribution picture, one can observe that in the first two flash crowd events, the increase in the number of requests is not significant in comparison to the normal operation of the CDN. Thus, there is a smoother change in network conditions, making difficult for both strategies to detect changes and hence adapt

itself. This is somewhat more difficult for the real metric virtualization since it is less sensitive due to its inherent less precise monitoring of the environment.

4.9.2.3 Shrunk scenario

The SHRUNK SCENARIO has 30215 requests spread among 15000 seconds of simulation. As observed, this scenario is much denser than the previous ones, that is, it has a bigger amount of requests per interval. The requests are divided in 16199 background requests, 7008 requests from AS3 between the times of 5000s and 7000s, during the rush time, and the second flash crowd, 7008 requests from AS7 between times 12000s and 14000s, during the lull time. Figure 4.32 illustrates this scenario.

Table 4.5 – Configuration parameters for shrunk scenario

Simulation time	15000s
Link capacity	1000Mbit/s
Baseline requests	16199 requests
1 st Flash crowd AS	AS3
1 st Flash crowd requests	7008
1 st Flash crowd start time	5000s
1 st Flash crowd end time	7000s
2 nd Flash crowd AS	AS7
2 nd Flash crowd requests	7008
2 nd Flash crowd start time	12000s
2 nd Flash crowd end time	14000s

Figure 4.32- Request distribution for shrunk scenario

Charts in Figure 4.33 and Figure 4.34 compare the performances of all experiments altogether.

Figure 4.33- Total network traffic along simulation time

Figure 4.34- Total cross traffic along simulation time

As the lines show, the real metrics scenarios (REALDATA*) performed as good as the omniscient one (BITSENT). In this scenario, REALDATA metrics were able to identify the changes in network conditions on the expected times (flash crowds), reducing considerably the cross traffic when comparing to the NONE scenario. However, likewise the STRETCHED scenario, the first flash crowd event was not detected. We assume this was consequence of the way the base traffic (normal operation) was distributed, since it concentrated on the same ASes that generated the flash crowd.

As for the QoE point of view, in this scenario, as expected, the result was better than REGULAR scenario where QoE was considerably worst in the experiment with replica servers' replacement. Likely, this behavior occurs due to requests being concentrated on the same ASes that generated the flash crowd. Due to the considerable

increase on localized requests during a flash crowd, the placement strategy replaces all the replica servers together in the flash crowd region. Considering the REGULAR scenario, requests aside from localized flash crowds were scattered on all ASes thus, reallocating all replica servers to the flash crowd's region resulted in QoE problems to all other clients. Figure 4.35 shows mean startup delay for the SHRUNK scenario. There is no considerable difference between scenarios with and without cache replacement.

Figure 4.35- Mean startup delay for shrunk scenario

4.10 Concluding remarks

The experiments on the weighted moving average vector approach pointed out that window length clearly is an impacting factor. A first row of experiments, using a window length of 300 seconds presented lower performance than the presented results of length 500s window. A possible justification for this is that the weighted windows can take advantage on more data to analyze and consider the more recent time frame whereas the single window approach “dilutes” information among the whole interval.

Therefore, in order to fully validate the proposal as useful to the intended purposes, either an empirically found window configuration must be used, to comply with most of the scenarios, or a self-configuring approach has to be implemented, so it adapts to the dynamic of the events accordingly. The design and evaluation of such fixes are left for future work. It could use AI techniques to self-fixation of window size. After this first issue is properly addressed and validated, the solution could be tested with a bigger variety of scenarios, such as geographic absence and surrogate overloads.

The main contribution is the method to detect significant changes in an overlay network. We showed that even with this limited information, we can achieve results similar to an optimal omniscient environment where the system has full information of all possible monitored data. Our second and third attempts to validate the strategy already presented animating results. Increasing the number of requests and the observed experiment time (STRETCHED) resulted in detecting changes consistently with the regular scenario approach. Likewise, when the designed scenario was denser (SHRUNK) than the regular, we could observe a similar performance in change detection.

5 The slicing framework

In recent years, we witnessed an emergence of storage Cloud providers. Examples are Amazon S3, Nirvanix and Rackspace. Storage cloud providers operate data centers that can offer Internet-based content storage and delivery capabilities with the assurance of service uptime and end user perceived service quality [68]. These capabilities can bring two opportunities: The first is to allow a small business to become customer of multiple providers in different locals and countries, building a dynamic overlay to take advantage of prices at specific location and a certain moment. The second is to become a CDN provider, offering content delivery service for third parties without the cost of owning or operating geographically distributed data centers. An example of this approach is MetaCDN [92].

Our framework is similar to the latter business case and inspired by it. Like cloud-based CDNs, we study the potential to build an adaptable overlay with a topology that may be different from the underlying network. We propose a model for a CDN that considers diversity, in which each type of content or content provider can be handled by “virtual CDNs”, and can build on top of a traditional CDN or a cloud environment. We present a model that enhances content delivery capabilities, considering a plurality of content service providers, each one with its own content diversity, incorporated into an overall infrastructure.

5.1 Multi-tenancy

Network design, including that of CDNs, has mainly been about resource sharing. Recent years witnessed the widespread of some global storage Cloud providers. They offer content storage and delivery capabilities with the assurance of service uptime and end user perceived service quality. Nonetheless, a small business may prefer contracting multiple CDN providers, building a dynamic overlay that takes advantage of competitive prices, geolocation and location diversity. MetaCDN adopts this business model. Introduced by Broberg et al.[92], it is a Cloud-oriented CDN system commercially available that leverages several existing Storage Clouds, creating an integrated overlay network that aims to provide a low cost, high performance, easy to use content delivery network for content creators and consumers. It offers a web portal allowing users to define different options related to cost and QoS by means of redirection policies. These options

include: maximizing coverage and performance, deploying many replicas as possible to all available locations; deploy content to specific location; a cost optimized deployment, where storage and transfer budget can limit the active time of replica deployment; and a Quality of Service deployment.

Lin et al. [108] proposed an architecture for cloud-oriented CDN. Its system uses Hadoop Distributed File System as a cloud infrastructure services, and creates many data clusters around the world. Jin et al. [193] introduced a conceptual architecture for Content Delivery as a Service (CoDaaS). It enables on-demand virtual content delivery service (vCDS) overlays for User Generated Content providers to deliver their contents to a group of designated consumers. The proposed CoDaaS solution is built on a hybrid media cloud, and offers elastic private virtual content delivery service with an agreed Quality of Service (QoS). VTN, discussed in section 4.3.1, can also be seen as a slicing technique.

Moreira et al. [125] proposed the virtualization of VoD CDN, allowing programmatically modification in a CDN infrastructure designed for video distribution, in order to adapt it to new operating conditions. A similar approach is followed by Srinivasan et al. [157] that allows content providers to dynamically deploy and instantiate CDN modules on core and edge routers with programming functionality over content distribution paths from the origin server to the users. The particular cloud-based CDN solution, called ActiveCDN, allows users to be redirected to the most appropriate node based on geolocation. CDN modules are deployed on an “as needed” basis leading to a dynamic network topology.

Li et al. [107] proposed a hybrid VoD delivery service that combines the traditional CDN and a cloud-assisted deployment, where user requests are partly served by the self-owned servers and partly served by the cloud. This model can save 30% bandwidth expense compared with the Clients/Server mode and can also handle unpredicted flash crowd traffic with little cost. Another hybrid system is in Carlsson et al. [30]. They studied the trade-offs of peer assisted and cloud-based CDNs with service time guaranties.

Chen et al. [68] investigate the problem of building distribution paths and placing Web server replicas in cloud-oriented CDNs in order to minimize the cost while satisfying QoS requirements of end-users. They enhance previous works on cache location problem [135], formulate the problem as an Integer Program and provide greedy heuristics. Another cost model for cloud-oriented CDN is presented by Papagianni et al. [141]. They model the problem in a multiprovider cloud environment with intra and inter cloud

communication and adapt heuristics to provide cost efficient splitting of content distribution among cloud providers.

Dan and Carlsson [48] propose a dynamic allocation for a hybrid system that considers content workload prediction. They formulate the problem of allocating content as a finite horizon dynamic decision and provide computationally efficient approximation. Using traces of commercial CDNs, their solution can yield gain up to 50% when compared to static allocation.

Considered perhaps more relevant to our work, the ETSI Industry Specification Group (ISG) included the virtualization of CDN in use cases of interest as part of its development of Network Functions Virtualization (NFV) technology [65], see Figure 5.1. As earlier stated, NFV organizes network functions into building blocks that may be connected to create services without the cost of acquiring new physical devices. Network operators are able to reduce costs (when compared with dedicated hardware implementations), to improve the time to deploy new network applications, to facilitate the network management, and to design a new adaptable scenario for future innovations. NFV achieves this by modifying the traditional network architecture adding a virtualization layer for a set of network services. This work examines a similar vCDN scenario.

Figure 5.1- Principle of different vCDN cache nodes deployment [65]

The vCDN architecture should find support not only in NFV as explained earlier, but also in the emerging Software Defined Networks (SDNs) paradigm. A case in the point is the OpenDayLight project for an open SDN architecture. Due to the large scope of the effort, it has been divided up into independently evolving sub-projects. Among these, there is work on the Virtual Tenant Network. OpenDayLight VTN allows users to

define the network with a look and feel of conventional L2/L3 network that is automatically mapped into underlying physical network, and then configured with SDN control protocols. Hence a multi-tenant CDN provider may take advantage of VTN dynamic programmability as an enforcement technology that helps manage the virtual CDNs allocated to each tenant or provider.

We are not focusing on the actual mechanisms or control planes used to enforce virtualization such as SDN, VTN and NFV, but instead on the dimensioning of the shared CDN resources among the virtual tenants. They have been discussed in this introduction to show the vCDN use case importance in the context of future network design.

5.2 CDN architecture considerations

The content to be distributed by a CDN can be very diversified. It may include different types of objects such as web pages, images, and video; it can also be static or dynamic, have different popularities, geographic affinities, which may lead to different SLAs requirements. Lu et al. [195] showed that differentiated caching services is necessary to cope with the increasing heterogeneity in Internet clients and content. **This raises the question of how many CDN architectures are necessary for a single provider [144].**

The first answer is to have one architecture shared by all content providers or content types. The problem with this approach is that the presence of multiple content providers may result in several inefficiencies. For instance, differentiated caching services can be problematic: each set of policies and strategies aimed a specific type of content or QoS requirements must have an allocated storage size. As a result, deciding how much size will be attributed to each type can be very tricky.

Furthermore, it is common that a single CP has popular periods (several content requests) as well as unpopular ones, even throughout a day [94]. This leads to cache usage inefficiency, since the cache of a CDN serving an instantly unpopular CP will have more space than needed at that time while the cache of a CDN serving a popular CP will need more space than it has in that moment.

The uneven assignment of resources leads to setbacks that deplete end user's QoE. Because of on-demand reservation and liberation of memory addresses, disk space and processing units, it is expected that the remaining resources become unable to host content for another provider. In addition, these resources get fragmented and idle – that is, yet

having an associated maintenance cost even when not being used. The combination of such “voids” in a certain domain and an unexpected increase on resource demand in another domain may harm the overall performance of the cache, possibly causing breach of SLAs.

Thus, the second answer is to have as many architectures as the number of applications. An application can be aimed to a specific content provider or a specific content type. Each content type has its corresponding set of policies and algorithms that can handle it in order to satisfy its QoS requirements.

The cloud-oriented architecture seems to be appropriate to solve the problem of deciding the amount of space for each application. The solution is to use the virtualization provided by the cloud. Each application can have its own virtual CDN where its components are instantiated on top of the cloud environment. A surrogate can be seen as a Virtual Machine (VM) or virtual server, provided by the cloud infrastructure, with on demand resource reallocation for each application.

The resource reallocation, however, may encounter some pitfalls. The live re-assignment is usually a time consuming task and cloud providers may impose some coarse-grained restrictions. For instance, Amazon EC2 provides a resource charging policy according to fixed sizes of CPU, memory and storage. Also, the elastic nature of cloud resources allows the cloud-based proposals to assume the capacity to be infinity, however, in practice, they may be not, especially for private clouds.

For small CDN operators, like, for instance, telco-CDNs or mobile CDNs, the necessary underlying resources may not be sufficient to satisfy the demand of all of its virtual CDNs. Thus, the need for storage space demanded by each application may exceed the capacity. As a result, the CDN provider must decide how much space to allocate to each application, leading to the same problems faced by the first approach.

5.3 Framework description

The components of slicing framework were depicted in Figure 4.1. Each tenant correspond to a virtual CDN and it can be a specific content provider or set of CDN strategies designed to a specific content type. For instance, a virtual CDN for VoD distribution and another one for Web pages. The framework comprises the module described in last chapter, but the main difference is the concept of virtual CDN. The

virtual surrogates are no longer necessarily VMs instantiated on top of a cloud (but it still can be). The underlying resource (storage, for instance) in nodes are divided into **slices** and each one is eventually assigned to a tenant. A group of slices assigned for the same tenant in a node is seen as a virtual surrogate of that tenant. The virtual CDN thus comprises the virtual surrogates. The clients of a tenant are redirected to one of the physical nodes that contains a slice assigned to the corresponding tenant it belongs to.

Using this strategy, our approach simplifies the concept of virtual CDN. On the other hand, in our framework, the resources manager is more complex. When it receives a command for adaptation from the change manager, it needs to allocate slices to each tenant, according to the request demand on a specific area, the slices availability and the amount of capacity a tenant is allowed to use. We assume that the physical resource can ultimately be a private or public cloud resource (which makes our framework very similar to the cloud-based CDN in this case), a physical resource, or a CDNI contract with another CDN provider. However, we free the CDN operator from the assumption that the underlying physical resources are unlimited, like the cloud-based CDN does by using elastic resources from cloud. Also, we want to avoid the cloud management or granularity constraints of a VM allocation.

This is achieved by dealing internally with the allocation for each tenant a “shared” surrogate that runs on top of the underlying physical resource, while still forming the “virtual CDN” for each one, as in cloud-based CDN. Hence, there is no need to have as many VMs as the tenants in a physical node. One surrogate is shared by all of them, and each tenant has a slice of it.

We envision a scenario composed by several Content Providers (CPs) hiring the CDN provider to distribute content. The CDN will have a set of replica servers/caches. The agreement between the CP and CDN may differ in several aspects depending on CPs interests or the application being handled, for instance better startup delay or better geographical coverage. However, a basic concept is usually required, and shall be present in most of CPs contracts: **storage space**. We may see this as service packages that the CDN offers, for instance 10 GB of storage space spread through CDN’s replica servers. Thus, the problem the **resources manager** must solve can be summarized as the following question: how to distribute this storage space hired by the CP throughout all replica servers?

In summary, we propose a variable distribution of the capacity hired (storage space), where, for instance, a CP may have a bigger slice (bigger number of slices) of the cache in an area where it has higher content request rate. Our approach is based on the OS resource sharing nature to offer equal opportunities to all processes with minimum resource idleness. We advocate the same paradigm can be applied to sharing a physical (or virtual) server with many domains using virtualization to properly allocate resources to the corresponding demands.

5.3.1 The cache slicing solution

We start with use case of storage partition within a CDN provider. We consider a scenario where multiple tenants (CPs) hire a CDN to outsource content. In this initial scenario, we consider a contracting process between the content provider and the CDN in which the former purchases storage space for their objects, and the latter undertakes to distribute this space among its caches in order to give the best possible distribution.

The technical knowledge of the process of content delivery remains with the CDN. It is still responsible for performing typical CDN tasks like cache management, request redirection, content outsourcing, among others, and to meet the QoS requirements. The content provider cannot determine in which cache content will be effectively stored. It only hires **capacity** and the CDN defines how it will be distributed among its caches.

The only way for the content provider to interfere in the delivery process is indirectly through SLA, which is part of the contract. An SLA can define the QoS requirements to be met by the CDN, the penalties for non-compliance, rules regarding the geographical distribution and property rights, content replication, security etc.

In this model, the first question that arises, and will be discussed in more detail later, concerns the relationship between the storage space hired by the content provider and the amount of content to be delivered. It is important because, if the space purchased, for example, is just sufficient to store each piece of content only once and the entire content is stored in the CDN caches, obviously there will be no object replication. In this situation, each content is available in only one surrogate. On the other hand, some content may be kept off in order to allow the most popular be replicated in more than one surrogate. We remind the reader that this behavior cannot be directly controlled by the content provider. It is the CDN, which defines how each content will be effectively stored in their caches,

according to their caching replacement or outsourcing content algorithms. Storage slicing may also be used within the same cache to virtually separate between the different types of content it may serve.

Because of this, the contracting process can be performed basically in two ways: the CP defines the storage or the CP defines the SLA. In the first model, since content providers decide how much storage they intend to hire and it directly interferes with the plausibility the of QoS attendance, the CDN will define or negotiate the SLA with which it is possible to comply. The second model is just the opposite. The CP presents the QoS requirements and the CDN determines the amount of storage needed.

The second important issue is that the CDN can sell more storage space than it has if it is confident enough that, by doing so, the SLA and QoS requirements will still be met (or the penalties for non-compliance is less than the revenue). This is often practiced in link access multiplexing at the last loop by Telecom companies. As we will see later, this is not a decision for our algorithm but an additional parameter for it.

5.3.2 Problem formulation

We model the problem as a capacity allocation. We divide the problem in two parts. First, we describe the capacity allocation problem [133] within one surrogate and one CSP only, then, we extend the formulation to an arbitrary number of CSPs in the multi-tenant environment as described earlier, introducing, or course, the proper constraints.

In the first part, the input of the problem consists of the following: a set content objects, first assumed to be unit-sized, $\mathcal{O}$; an available storage capacity $\mathcal{S}$ of storage units, the same measure of objects; a set of m clients, $\mathcal{J}$, each client j have a distinct request rate λ_j and a distinct object demand distribution $p_j: \mathcal{O} \rightarrow [0,1]$; where 1 represents the demand of the object. The input also contains a tree graph $\mathcal{T}$ with a node set of n nodes, $\mathcal{V}$, and a distance function $d_{j,v}: \mathcal{J} \times \mathcal{V} \rightarrow R^+$ associated with the j th leaf node and node v ; this distance captures the cost paid when the client j retrieves an object from the node v . Each client issues a request for an object that must be fulfilled by an ancestor the holds that object or by the origin server (not represented in this model). A client always receive an object from a unique node.

The storage capacity allocation problem is defined as identifying a set $\mathcal{A} \subseteq \mathbb{A}$ with no more than $\mathcal{S}$ elements (node-object pairs) $(v, k), v \in \mathcal{V}, k \in \mathcal{O}$; $\mathbb{A}$ is the set that contains all node-object pairs. $\mathcal{A}$ must be chosen so as to minimize the following expression cost:

$$\min_{\mathcal{A} \subseteq \mathbb{A}: |\mathcal{A}| \leq \mathcal{S}} \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{O}} p_j(k) \cdot d_j^{min}(k) \text{ where } d_j^{min}(k) = \min\{d_{j,os}, d_{j,v}\}: \quad (13)$$

$$v \in \text{ancestors}(j), (v, k) \in \mathcal{A}$$

Where $d_{j,os}$ is the distance between the j th client and the origin server, while $d_{j,v}$ is the distance between the j th leaf node and an ancestor node v . To define the capacity allocation problem as a integer linear program, let $X_{j,v}(k)$ denote a binary integer variable which is equal to one if client j gets object k from node v where v is ancestor of client j , and zero otherwise. Also let $\delta_v(k)$ denote an auxiliary binary integer variable which is equal to one if object k is placed at the ancestor of node v , and zero otherwise. The two types of variables are related as follows:

$$\delta_v(k) = \begin{cases} 1, & \text{if } \sum_{j \in \text{leaves}(v)} X_{j,v}(k) > 0 \\ 0, & \text{otherwise} \end{cases} \quad (14)$$

Equation (14) express the requirement that an object must be placed at a node if some clients are to access it from that node. Thus, according to [133], the ILP gives an optimal solution to the storage capacity allocation problem.

Maximize:

$$z = \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{O}} p_j(k) \cdot \sum_{v \in \text{ancestors}(j)} (d_{j,os} - d_{j,v}) \cdot X_{j,v}(k) \quad (15)$$

Subject to:

$$\sum_{v \in \text{ancestors}(j)} X_{j,v}(k) \leq 1 \quad j \in \mathcal{J}, k \in \mathcal{O} \quad (16)$$

$$\sum_{v \in \text{leaves}(j)} X_{j,v}(k) \leq U \cdot \delta_v(k) \quad v \in \mathcal{V}, k \in \mathcal{O}, U \leq |\mathcal{J}| \quad (17)$$

$$\sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{O}} \delta_v(k) \leq \mathcal{S} \quad (18)$$

where $X_{j,v}(k)$, $\delta_v(k)$ binary decision variables $v \in \mathcal{V}, k \in \mathcal{O}, j \in \mathcal{J}$.

The maximization of equation (15) is equivalent to the minimization of equation (13) [133]. Equation (16) states the a client will have only one ancestor where it gets objects from, while equation (17) express the number of times an object will accessed can be no more than the number a requests from all the leaves from the node the object is placed. Equation (18) states the obvious restriction that the total placement of objects in the topology cannot exceed the capacity $\mathcal{S}$.

To adapt this equations to cache slicing problem formulation, the first necessary change is to relax the restriction of a tree topology. The second is to restrict the nodes an object can be placed (see equation 20 bellow), since only the nodes that have a surrogate can be candidate to receive an object. The last restriction is to consider the capacity of that surrogate (see equation 21). Let Φ , $\Phi \subseteq \mathcal{V}$, be the set of nodes v that has an installed surrogate. Thus, equation (13) must be rewritten to:

$$\min_{\mathcal{A} \subseteq \mathbb{A}: |\mathcal{A}| \leq \mathcal{S}} \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{O}} p_j(k) \cdot d_j^{\min}(k) \quad \text{where } d_j^{\min}(k) = \min\{d_{j,os}, d_{j,v}\}: \quad (19)$$

$$v \in \Phi, (v, k) \in \mathcal{A}$$

And equation (14) must, then, be rewritten to:

$$\delta_v(k) = \begin{cases} 1, & \text{if } \sum_v X_{j,v}(k) > 0, \text{ where } v \in \Phi \\ 0, & \text{otherwise} \end{cases} \quad (20)$$

A surrogate installed at the node v has a capacity $\mathcal{C}_v$, $\sum \mathcal{C}_v \leq \mathcal{S}$, thus, the allocation must also be subject to the following constraint:

$$\sum_{k \in \mathcal{O}} \delta_v(k) \leq \mathcal{C}_v \quad (21)$$

So, the modified storage allocation problem can be written as follow:

Maximize:

$$z = \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{O}} p_j(k) \cdot \sum_{v \in \Phi} (d_{j,os} - d_{j,v}) \cdot X_{j,v}(k) \quad (22)$$

Subject to:

$$\sum_{v \in \Phi} X_{j,v}(k) \leq |\Phi| \quad j \in \mathcal{J}, k \in \mathcal{O} \quad (23)$$

$$\sum_{k \in \mathcal{O}} \delta_v(k) \leq \mathcal{C}_v \quad (24)$$

$$\sum \mathcal{C}_v \leq \mathcal{S} \quad (25)$$

where $X_{j,v}(k)$, $\delta_v(k)$ binary decision variables $v \in \mathcal{V}$, $k \in \mathcal{O}$, $j \in \mathcal{J}$.

Now that we have the problem statement of storage allocation, the second part is to extend the model to the multi-tenant architecture of the cache slicing problem. Each tenant, a content provider, has its own sets of objects, clients and performs an individual storage allocation. In other words, the problem, as formulated in equations 22-25 must be solved for each tenant. But the overall solution has itself its own set of constraints.

Let T be a content provider that belongs to a set $\mathbb{T}$, of all content providers of cloud-based CDN. Each content provider forms a virtual CDN installed on top of the cloud infrastructure of the cloud-based CDN and must, independently of each other solve the problem of equations 22-25 to find an optimal allocation of objects on the available caches. The set Φ is related to all points of presence of the cloud-based CDN, L , but not

necessarily $\Phi = L$, in other words, the virtual CDN of a content provider may not have presence on all available locations of the cloud-based CDN.

Each content provider T hires a certain amount of storage space from the cloud-based CDN, $\mathcal{S}_T$, and must construct the virtual CDN infrastructure subject to this hired capacity. Each node $v \in L$, has also a capacity $\mathbb{C}_v$, which represents the size of underlying physical infrastructure available to cloud-based CDN. Consider $\mathcal{C}_{v,T}$ the capacity of surrogate installed on node v of the content provider T . Then, the following constraints must be satisfied:

$$\sum_{T \in \mathbb{T}} \mathcal{C}_{v,T} \leq \mathbb{C}_v \text{ for each } v \in L \quad (26)$$

$$\sum_{v \in \Phi} \mathcal{C}_{v,T} \leq \mathcal{S}_T \text{ for each } T \in \mathbb{T} \quad (27)$$

$$\sum_{T \in \mathbb{T}} \mathcal{S}_T \leq \mathbb{S} \quad (28)$$

Where $\mathbb{S}$ is total capacity of entire cloud underlying physical infrastructure. The caching slicing problem can be formulated as follow:

Maximize:

$$\mathbb{Z} = \sum_{T \in \mathbb{T}} \sum_{j \in \mathcal{J}_T} \sum_{k \in \mathcal{O}_T} p_j(k) \cdot \sum_{v \in \Phi_T} (d_{j,os} - d_{j,v}) \cdot X_{j,v,T}(k) \quad (29)$$

Subject to:

$$\sum_{v \in \Phi_T} X_{j,v,T}(k) \leq |\Phi_T| \quad j \in \mathcal{J}_T, k \in \mathcal{O}_T, T \in \mathbb{T} \quad (30)$$

$$\sum_{k \in \mathcal{O}_T} \delta_{v,T}(k) \leq \mathcal{C}_{v,T} \quad (31)$$

$$\sum \mathcal{C}_{v,T} \leq \mathcal{S}_T \quad (32)$$

$$\sum_{T \in \mathbb{T}} \mathcal{C}_{v,T} \leq \mathbb{C}_v \text{ for each } v \in L \quad (33)$$

$$\sum_{v \in \Phi_T} \mathcal{C}_{v,T} \leq \mathcal{S}_T \text{ for each } T \in \mathbb{T} \quad (34)$$

$$\sum_{T \in \mathbb{T}} \mathcal{S}_T \leq \mathbb{S} \quad (35)$$

where $X_{j,v,T}(k)$, $\delta_{v,T}(k)$ binary decision variables of virtual CDN T , $v \in \mathcal{V}$, $k \in \mathcal{O}_T$, $j \in \mathcal{J}_T$, $T \in \mathbb{T}$.

The cache slicing is a class integer linear programming (ILP) associated with a two-way optimization problem. It is a mixed linear programming problem with complicated constraints [39]. The master problem is in equation (29), defined as $\mathbb{Z} = \sum_{T \in \mathbb{T}} \mathbb{Z}$ part, and the sub-problems are related to the equations 22-25. The complicating constraint is the equation (33).

The ILP formulation of cache slicing problem is NP-hard, thus, it cannot be used for practical purposes. The solution of each sub-problem has a complexity of at least $O(|V| \cdot |\mathcal{O}| \cdot |\mathcal{J}| \cdot |\Phi|)$. In this work, we provide an heuristic solution with an approximation by simplifying the process of computing the $\delta_{v,T}(k)$ binary decision.

5.3.3 The algorithm

Considering this model, described in the previous sections, we present next the cache slicing. The algorithm is entirely performed by the framework. The CDN has a number of caches / surrogates, which can be changed when it decides to expand or reduce its infrastructure. This process of expansion or reduction can be done, for example, with the acquisition of new equipment, space allocation in the cloud or CDNI contracts. The important thing is that the change in structure will trigger a new execution of the algorithm, as explained below.

The storage space on each surrogate (cache) is divided into fixed-size small portions, called slices. The slices have a fixed size and are unique for all CDNs and all content providers. Each content provider receives a portion of each cache, but not necessarily of all caches. When the CP receives a portion of the cache, we say that it has a slice of it. The aim of this approach is to create a virtual CDN for each content provider. This virtual CDN will have as virtual surrogates the slices of the underlying surrogate the corresponding content provider received. All slices have the same size in the entire system.

The number of slices a content provider can receive from all caches depends on the contracted storage capacity, i.e., the sum of the number of slices a CP has on each cache is equal to the total hired storage capacity. Example: in a scenario where the slice size is 1TB, a CP that contracted 10 TB can receive a maximum of 10 slices from all the caches. A possible configuration could have: 5 slices from cache #1, 3 slices from cache #2 and 2 slices from cache #3, no slices from all the other caches.

The problem that we intent to solve with our approach is how to distribute the slices in each cache among the content providers. For this, our algorithm is divided into three phases. Figure 5.2 illustrates the Cache Slicing process.

Figure 5.2 – Cache slicing process flowchart

5.3.3.1 Phase I

At this stage the virtual CDN is formed by defining where the slices allocated to each content provider will be located, i.e., exactly which slices on which caches each content provider will receive, limited to number of slices that depends on the hired storage capacity, as previously mentioned. We call this phase placement of slices because it is similar to the placement of surrogates, a common problem of a traditional CDN. In fact, our approach is inspired by it and uses the same approach to deal with slices as explained next.

As it can be easily seen, that surrogate placement problem is computationally complex, which led to development of some heuristics, such as Greedy replica placement and scalable replica placement, described in chapter 3. These heuristics take into account the existing information from the CDN, such as workload pattern and the network topology.

The Greedy algorithm brings very good results and we use it as the basis of our approach. Also, it happens to be useful to understand the problem and how we adapt it in this phase. It works as follows: It chooses M surrogates among N potential sites by choosing one surrogate at a time. In a first iteration, each of the N potential sites is evaluated in order to determine the suitability of a site to host a surrogate. The cost is computed with the assumption that all clients will access that site. Then, the site with the lowest cost is chosen. The cost considers the goal of the placement, as explained earlier. In the second iteration, the greedy algorithm searches for a second site in conjunction with the sites already selected. The iteration continues until M surrogate servers have been placed. The greedy algorithm works well even with imperfect input data, but it requires the knowledge of the clients' locations in the network and all pair wise inter-node distances.

We tailor our approach to this problem by replacing potential sites by available slices on nodes, and surrogate servers by CP slices. Thus, now we want to optimally place slices on nodes. For instance, a certain content provider has 10 slices to be placed on the physical caches that the CDN provider has available. It must choose where to “distribute” each of these CP slices on the available cache slices. In addition, we added the capacity constraints of each slices and take into consideration the number of requests the slice can handle. The algorithm is more complex but still solved by a greedy approach.

Note that since we have many slices on each cache, we can have many slices for the same CP on that physical cache. This is desirable. In fact, this is somewhat similar to the concentration of surrogates around a spot of high demand. Imagine that a CP has most of its clients concentrated on a specific geographic region, in this situation, during this phase, it will likely have many slices allocated on the physical cache located near it.

In summary, in this phase, each CP has an amount of “desired” slices on each cache. However, there may be the situation that the sum of “desired” slices of each CP on a specified cache exceeds the capacity of that cache (i.e. the number of slices the cache can physically allocate). When this is the case, phase 2 of our algorithm is executed.

5.3.3.2 Phase 2

This phase is only executed after the first one, and only if the number of slices allocated for each content provider for a specific cache exceeds the capacity of that cache. Note that this is something not assumed to be possible if you consider the elastic capacity nature of a cloud. However, as stated earlier, we free the CDN operator from this assumption. Then, those content providers will have to compete for availability, in order to define which one should get the slices available. This competition is performed according to the approach presented below, but first some considerations must be made regarding the goals of the system.

We propose two different strategies to competition of slices, Proportional and Greedy. The former uses a simple criterion to decide which CP should win the competition for an available slice. The decision is based on the CP’s popularity (i.e., the CP with more requests in that area will receive more slices). It is important to notice that CP’s popularity is considered independently, this means that each CSP will define its desired slices configuration based only on its content popularity. Figure 5.3 shows an illustration of a Proportion strategy slicing example. Based on CSP’s desired slices we calculate the Slice Proportion (SP) and then multiplying SP by the number of desired slices the real number of slices allocated to each CP is defined (5 slices for each CSP in the example shown).

Figure 5.3 – Cache slicing proportional strategy

The other approach is a heuristic based on the number of requests, the size and the location of objects. The goal of the heuristic is to maximize CDN's profit by minimizing cost with content delivery. This is done by ranking objects according to their number of requests and calculating the score for each CP. This score represents the benefit of a CP receiving a slice. To estimate the benefit we test what would represent to content delivery the allocation of a new slice for each competing CP. Thus, according to the size of the slice we can estimate that a CP would place a new set of objects in caches and then estimate the gain that the CDN could achieve by giving a new slice to each CP. This is done by summing the estimate gain obtained by all objects that could be placed in the new slice and subtracting all objects left out of the cache. The gain for each object is calculated as the multiplication of object's size and the number of times it was requested during the last time window. The process of calculating the score must incorporate the specifics of SLA of a certain tenant. For instance, suppose that CP 1 has more stringent requirements as it delivers video content. It should be prioritized in relation to a CP 2 that deliver web content. The cost of not assigning slices to CP 1 must be higher in order to incorporate the penalty of breaching the SLA.

Figure 5.4 and Figure 5.5 illustrate a slice competition between two CPs. The size of the new slice is 150Kb. In the former the score is better for CP 1. But with a similar scenario and greater popularity, as shown in Figure 5.5, for CSP 2's first object the score changes making it more interesting for the CDN to give this new slice to CP 2.

Figure 5.4 – Cache slicing greedy strategy

Figure 5.5 – Cache slicing decision in greedy strategy

Because of the competing process, after the execution of phase 2, there could be some content providers that asked for a slice on a given cache but did not get it, see Figure 5.6. All unallocated slices will be handled during the next step, phase 3.

Figure 5.6 – Example of cache slicing configuration after phase 2

5.3.3.3 Phase 3

The third is the simplest phase, it only happens when, after phase 2, there are yet slices to be placed. In this case, CPs are probed to place one of the remaining slices in a Round Robin fashion, but considering only available slices remaining after the execution of Phase 2. Each CP places one, and only one, slice per round and it does so according to its content demand.

More importantly, though, is the maintenance of the system. The **allocation of slices for each CSP is not static, it can change during time**. We argue that this is the advantage of using virtual CDNs for each CP in our approach, providing it with the dynamic nature. Thus, the CDN monitor will provide information to a database to trigger the adaptation of the system, controlled by the Changer Manager (Chapter 4). The trigger will run the algorithm (phases 1, 2 and 3) when a change in the traffic behavior happens, which is periodically monitored, according to the algorithm in [125] (described in previous chapter). Also, it can be triggered when one of the following conditions are met:

- A new content provider enters the system. By obvious reasons, when a new CP enters the system, a new arrangement may be needed, thus the algorithm must be performed.
- A content provider hires more capacity. This is similar to the entrance of a new CP: a new arrangement is needed.

- The costs of the infrastructure change. This is the monetary cost of the infrastructure from the point of view of CDN provider (equipment, network links, etc.). A change in the cost of the infrastructure will impact on the revenue and, therefore, on the decision of allocation of slices. Because of this, the provider may want to establish weights on certain locations to discourage usage of resources on that. Thus, the previous allocation may not be optimal anymore.
- CDN provider changes the infrastructure. The addition of new equipment has the potential to interfere in the costs/revenue of the system. For the same reason of the previous topic, a new arrangement may be needed.
- There is a change on the SLA of a CP. The change on the SLA of content provider has the potential to change costs/revenue of the system and, therefore, impact on the decision of allocation of slices.

The main advantage of our approach is not having a static configuration for the placement of surrogates like the traditional CDN. Also, the adaptation can only be needed by a specific CP and it may not impact the allocation of the others.

5.4 Evaluation and discussion

To evaluate the slicing strategy, we created four scenarios with increased size and complexity. The first one we called proof of concept. It is a very small scenario and was used to test the implementation and, as the name suggests, was used to test the feasibility of the proposal. The following two scenarios increase the number of requests and content providers, with the last one designed to test the competition strategy.

5.4.1 *Proof of concept*

For the proof of concept, we created a small scenario with two CPs. Each CP has its own objects with specified sizes and requests to them. The main metric we observe is the hit ratio and we compare two executions of each scenario, one with the slicing strategy active. When not active, the objects of both CPs are inserted into surrogate caches without identifying its origin, i.e., they are treated as if they belong to the same CP. In other words, insertion and removal of objects depends solely on the cache replacement technique used in the surrogate cache. When slicing is active, each CP has a portion of surrogate cache,

which means that objects are treated separately. Insertion and removal are still performed by the same cache replacement technique, but independently since each CP has its own virtual cache. The algorithm for cache replacement is the same for both execution, thus, the only difference when slicing is active is the division of caches in opposition to treating them as one big cache for all CPs as in conventional CDN. Also, in both scenarios, the total storage capacity of CDN is the same, for a fair comparison. Specific scenario configuration is described as follow.

The first scenario has two CSPs with 3,498 and 3,527 requests, thus a total of 7,025 requests for VoD objects. The workload for each CP was generated with Medysin [173]. Each CP has a baseline traffic that lasts approximately 2,100s (total simulation time) and one flash crowd that overlaps each other during execution time. Also, both flash crowds are concentrated on a specific AS of the network topology, forcing the cache slicing technique to request more slices on that AS for both CPs and, thus, compete for slices if the number of available slices is not sufficient. The goal of this is to force the slicing strategy to redistribute slices among physical surrogates when the flash crowd starts and finishes.

However, they do not overlap precisely on the same time. The flash of the first CP starts first and lasts longer. The idea is to have more slices given to the first CP on the physical surrogate located in its flash crowd AS while the second CP does not need it yet. Then, when the flash crowd of CP 2 starts, the slicing algorithm is triggered again. Figure 5.7 pictures the requests evolution for both CPs.

Figure 5.7 – Requests evolution during time

In the beginning of simulation, the slices hired for each CP (equal amount) are uniformly distributed, and then they change over time according to the slicing algorithm. The cache replacement technique is LRU. Both CPs have 10 slices to be distributed. Other configuration parameters are summarized on Table 5.1 below.

Table 5.1 – Configuration parameters

	CP1	CP2
Simulation time	2,100s	
# of baseline requests	2,121	2,438
# of flash crowd requests	1,406	1,060
Start time of flash crowd	900s	1100s
# of slices	10	
# of slices per surrogate	10	
# of surrogates	10	

The results are shown below. As it can be seen, the scenario with slicing active has better results for hit ratio and, consequently, for total and cross network traffic. There was a considerable increase in hit ratio on the slicing scenario, from 1980 to 2558. The distribution of slices occurred as expected. When the first flash crowd starts, there is redistribution of slices requested by the first CP, but phase 2 of algorithm did not run,

because the number of slices requested did not exceed the total capacity of the physical surrogate. Note that the second CP did not request more slices on that surrogate because its flash crowd have not started. However, when it starts, the phase 2 is executed because the total number slices requested by both CPs exceed the maximum physical surrogate capacity.

Figure 5.8 – Total network traffic progress

Figure 5.9 – Cross traffic progress

Figure 5.10 – Cache hit progress

Figure 5.11 – Startup delay progress

5.4.2 Scenario 1

This scenario has two CPs with a total of 81,689 requests for VoD objects. The workload of each CP was also generated with Medisyn [173]. Figure 5.12 pictures the requests evolution for both CPs, where each CP suffers two flash crowds and the second one happens at the same time for both CPs. The second flash crowd is concentrated on a single AS, forcing both CPs to need more slices on that AS and therefore competing for slices there.

Figure 5.12 – Request evolution along time for scenario 1

In the beginning of simulation, the slices hired for each CP (equal amount) are uniformly distributed, and then they change over time according to the slicing algorithm. The cache replacement technique is LRU. Both CPs have the same number of slices to be distributed. Configuration parameters are summarized on Table 5.2 below.

Table 5.2 – Configuration parameters for scenario 1

	CP1	CP2
Simulation time	21,000s	
# of baseline requests	10,081	10,808
# of requests for 1 st flash crowd	15,328	14,855
Start time of 1 st flash crowd	900s	1100s
# of requests for 2 nd flash crowd	15,312	15,330
Start time of 2 nd flash crowd	14400s	

The results are shown below.

Figure 5.13 – Cache hit progress for scenario 1

As it can be seen, the scenario with slicing active has better results for hit ratio and, consequently, for total and cross network traffic. There was a considerable increase in total hit ratio on the slicing scenario, from 16192 hits to 47637 hits, which means that with cache slicing more requests were fulfilled with content that was already present on the cache. The distribution of slices occurred as expected. When the first flash crowd starts, there is redistribution of slices requested by the first CP, but in that moment phase 2 of algorithm did not run, because the number of slices requested did not exceed the total capacity of the physical surrogate. Note that the second CP did not request more slices on that surrogate because its flash crowd had not started yet. However, when it starts, the phase 2 is executed because the total number slices requested by both CPs exceed the maximum physical surrogate capacity.

Figure 5.14 – Startup delay progress for scenario 1

The startup delay time progress graph, shown in Figure 5.14, presents small but consistent improvement in cache response times during flash crowds when using cache slicing.

Figure 5.15 – Total network traffic progress for scenario 1

Figure 5.16 – Cross traffic progress for scenario 1

A significant difference in network traffic (Figure 5.15), especially in cross traffic (Figure 5.16), can be seen when using cache slicing. The positive effect on cross traffic may represent significant savings in connectivity/traffic costs for the CDN provider, which will help increase CDN profit.

Figure 5.17 pictures the slices distribution over time. Each physical surrogate has 10 empty slices. At the beginning of the simulation, each CP has one slice allocated in each surrogate. 100 seconds (the monitor interval time) after the start of the first flash crowd, the slicing algorithm is triggered because of the changes in network conditions

detected for CP2. Thus, it can be seen on Figure 5.17 that, since the flash crowd is concentrated on AS7, the number of slices desired on surrogate 7 is bigger, but phase 2 is not executed since the total number of slices desired is 9, less than the maximum capacity of surrogate 7. At 7,300s, the CP1 triggers a new execution of slicing algorithm, but again phase 2 is not executed since its flash crowd occurs in a different AS. The following configurations reflects the adjustments after the end of each flash crowd.

Figure 5.17 – Evolution of slice distribution

As expected, competition for slices occurs only at time 14,400s, when both CPs desires slices on the same physical surrogate since their flashes crowd are concentrated on same AS. Figure 5.18 pictures the slicing configuration from that point on. It can be seen that after phase 1, CP1 desires 10 slices on surrogate 3 and CP2 desires 7 slices, exceeding the maximum capacity of the surrogate.

Figure 5.18 – Competition of slices

5.4.3 Scenario 2

The second scenario is similar to the first one. The main differences are the number of CPs, now three, and the number of requests, which is higher, as shown in

Figure 5.19, with a total of 589,399 requests. Two different algorithms were used for distributing the slices in case of concurrency, **Proportional** and **Greedy**. The CP1 has a baseline traffic of 43,008 requests, a first flash crowd of 62,792 requests concentrated on AS2 that starts at 3,600s, a second flash crowd of 90,802 requests concentrated on AS8 that starts at 14,400s. The CP2 has a baseline traffic of 43,076 requests, a first flash crowd of 62,856 requests concentrated on AS4 that starts at 5,400s and a second flash crowd of 90,543 requests concentrated on AS8 that starts at 14,400s. The CP3 has a baseline traffic of 43,706 requests, a first flash crowd of 63,165 requests concentrated on AS6 that start at 7,200s and a second flash crowd of 89,455 requests concentrated on AS8 that start at 14,400s.

Figure 5.19 – Request evolution along time for scenario 2

As it can be seen in Figure 5.19, the first flash crowds occur on different ASes. Thus, the CPs should not compete for slices on the physical surrogates located on those ASes. This will happen only on the last flash crowd. Other parameters are similar to those of scenario 1, except for simulation time, which is 22,000s. Configuration parameters are summarized in Table 5.3 below.

Table 5.3 – Configuration parameters for scenario 2

	CP1	CP2	CP3
Simulation time	22,000s		
# of baseline requests	43,008	43,076	43,706
# of requests for 1 st flash crowd	62,792	62,856	63,165
AS of first flash crowd	AS2	AS4	AS6
Start time of 1 st flash crowd	3600s	5400s	7200s
# of requests for 2 nd flash crowd	90,802	90,543	89,455
AS of last flash crowd	AS8		
Start time of 2 nd flash crowd	14400s		

The simulation results for scenario 2 are shown below.

Figure 5.20 – Cache hit progress for scenario 2

In the case of cache hits, shown in Figure 5.20, both cache slicing algorithms performed similar to each other, with the majority of requests being fulfilled by content already on the cache, and much better than the case without cache slicing. The Greedy algorithm represents a small improvement on cache hits, 339,262 to 331,009. In the case of startup delay (Figure 5.21), there is a small difference between Proportional and Greedy algorithms during the second flash crowd, even though the total average for both algorithms is similar. During the second flash crowd, however, the Greedy algorithm performed systematically better, though with a very small overall advantage.

Figure 5.21 – Startup delay evolution along time for scenario 2

When considering network traffic (Figure 5.22), especially the cross traffic (Figure 5.23), it is observable that using cache slicing represents a significant benefit for a CDN, especially a network provider-operated CDN, since it saves network bandwidth and may significantly decrease the CDNs connectivity costs. Once again, the Greedy algorithm yielded slightly better results when compared to the Proportional algorithm: respectively 365 GB and 389 GB for cross traffic; 2,269 GB and 2,308 GB for total traffic. The reduction, when compared to the traditional CDN, can reach 15% for total traffic and 37% for cross-traffic.

Figure 5.22 – Total network traffic progress for scenario 2

Figure 5.23 – Cross traffic progress for scenario 2

5.4.4 Scenario 3 - Playback continuity

All the previous experiments executed so far assumed that the content providers had the same SLA. The main consequence is that the greedy slicing will treat them as equal and this explain why the greedy and proportional strategy had similar results. In this section we test a different approach where one of the content providers have more stringent requirements forcing the greedy strategy to provide more slices to it during the competition on phase 2.

The main metric to observe in this scenario is the playback continuity. As stated previously, the playback continuity is directly related to the QoE of users. In this experiment, when the number of slices is not sufficient to fulfil all the requests from the clients, some of them will be redirected to the origin server. This impacts the playback continuity of those clients, since they will be subjected to network congestion which increase the possibility of dropped packets or reduced bandwidth. This leads to a decrease in the playback continuity metric as the video fragments may not arrive on time. There is also the possibility of capacity depletion of the origin server, but this use case is not tested (we assume an infinity capacity for origin server).

This scenario is similar to scenario 2. All the simulation parameters are the same, but we introduce a different method in the slicing competition to prioritize the content provider 1. During the cost calculation, the penalty for not fulfilling a request from CP 1 must be higher than the others, increasing the chances of providing more slices to it. This

is done by introducing a new parameter for each CP that represents a weight that multiply the cost for contents not present on cache. We test 4 values for weight: 1 (which is equivalent to the previous greedy experiments), 2, 4 and 8. Values above 8 do not show improvement as will be discussed later. The results are shown below.

Figure 5.24 – Cache hit progress for scenario 3

Figure 5.25 – Total network traffic progress for scenario 3

Figure 5.26 – Cross traffic progress for scenario 3

Figure 5.27 – Startup delay progress for scenario 3

The results show that for the main metrics analyzed so far, the different configuration for weight parameter does not yield significant differences (less than 1%) in performance for the greedy approach, as it can be seen in Figure 5.24, Figure 5.25, Figure 5.26 and Figure 5.27. The accumulated graphics for traffic metrics show that there is almost no variation between the different weights. This is expected because, although CPs 2 and 3 are suffering during the second flash crowds (for the weighted configurations), the system is still providing a slicing distribution that maximizes the desired goals. The difference is that the content provider 1 is being prioritized. Thus, the playback continuity results show significant differences, as depicted in Figure 5.28 below.

Figure 5.28 – Playback continuity progress for scenario 3

Figure 5.28 show the average playback continuity perceived by the clients during the simulation. During the first flash crowd, there is no difference in the metric for the slicing strategies because there were no competition of slices. The first flash crowd happens at isolated ASs for the CPs, so even the proportional strategy achieve the same result. The fall of the perceived playback continuity is due to the fact that the system is not able to fulfill all requests coming from the clients, even with all slices assigned to CP 1. It is a limitation of the system, i.e., the available capacity is not enough for all requests, independently of the slicing distribution. This explains why weight values above 8 does not interfere in the results, since, from this configuration, for the competing strategy, all slices is already assigned to CP1. Finally, as expected, for higher weight values, the playback continuity increases.

5.5 An Autonomic CDN Slicing approach

The growing dynamics of a large variety of distributed computing systems raises the need for more autonomous systems that may adapt to changing working conditions with minimal manual intervention. The proposed framework is focused on that need, however, the management process is centralized and, as systems grow, the monitoring information held by the monitor module to support the change and resource manager can become prohibitive and may reduce the scalability.

In this section, we explore a different approach to resources management in the framework, using the self-organizing model. Case studies of self-organizing systems show that they commonly consist of a group of relatively simple entities that interact to achieve complex adaptive group behavior. From a technical viewpoint, lower-level components that are simple and homogeneous are easy to manufacture and maintain. Moreover, the redundancy inherent to such a system implies robustness as well as scalability [80].

In self-organizing systems, each node performs local decisions and somehow coordinates its small scale actions while leading to a global objective through limited interactions. According to [67], self-organization is a process in which a pattern, at the global level of a system, emerges solely from numerous interactions among the lower-level components of a system. The result is called **emergent behavior**.

Based on this concept, we remove the central manager from our framework. Instead, each node now has its own set of monitor, change manager and resources manager (as well as possibly other components of the CDN architecture). Figure 5.29 depicts the new architecture. The information collected by the monitor is now limited to the entities on the same node. The change manager is also local under this new view and makes decisions based only on the locally monitored and collected information.

Figure 5.29 – Decentralized approach for slicing framework

When the change manager detects the need for adaptation, the resources manager will distribute the slices on the same node, based on the local demand information provided by the monitor. A different distribution algorithm is provided and detailed in next section. The new resources manager also provides another function which is a process of communicating or consulting with the other nodes in order to reach a consensus of the overall slices distribution.

This new approach of the framework can be categorized as an autonomic computing. In [85], authors define that Autonomic Computing is “a computing environment with the ability to manage itself and dynamically adapt to change in accordance with policies and objectives”.

5.5.1 Slicing algorithm

Under the new autonomous approach, there is no phase 1. After the trigger from change manager takes place, each node will independently execute the competition algorithm of phase 2 in its resource manager. The input for the algorithm is a configuration where all CPs “desire” all slices on that node and, then, they will immediately compete with each other using only information of “local” requests. After this phase, all slices in all nodes are assigned to a CP and, clearly, the overall number of slices for all CPs in the system will exceed the hired capacity. This is especially true at the beginning of the system operation. It is necessary, then, a process of changes in slice distribution until it converges to the final “expected” allocation that must consider the hired capacity of the content provider.

To solve this, each node will start communicating to each other to try to reach a consensus on how the slices will be effectively distributed. The first message exchanged by the resources managers includes the slice allocation performed on that node and the final calculated cost of each content provider according to that configuration. The cost is the output of that phase 2 described earlier, using the same algorithm. After a message is received, the resources manager rearranges the slices distribution according to the Algorithm 5.1 – Distribution algorithm executed by resources manager. The output of the algorithm is the “final” allocation for the entire system, according to the information the resources manager possess at that moment. However, the node is only able to apply the calculated slicing distribution to itself, the rest is included in the

messages exchanged on next cycles. This information is used to determine whether the system has reached the consensus.

```

INPUTS:
initialAllocations[ContentProvider,surrogate];
costs[ContentProvider,surrogate];
capacity[surrogate];
slices[ContentProvider];

PROCEDURE Alloc(initialAllocations[], costs[], slices[], capacity[])
  allocation[ContentProvider,surrogate];
  FOR EACH Content Provider CP
    IF initialAllocations > slices THEN
      totalcost <- sum(costs[ContentProvider])
      FOR EACH Surrogate
        allocation[CP,Surrogate] <- costs*slices[CP]/initialAllocations[CP,Surrogate] /totalcost;
      END-FOR;
    ENDIF;
    IF initialAllocations < slices THEN
      distributeRemaining(allocation,slices,costs,capacity);
    ENDIF;
  END-FOR;
  FOR EACH Surrogate
    IF allocation > capacity THEN
      allocation[surrogate] = initialAllocations[surrogate];
    ENDIF;
  END-FOR;
  FOR EACH Content Provider CP
    distributeRemaining(allocation,slices,costs,capacity);
  END-FOR;
  return allocation;
END-PROCEDURE;

PROCEDURE distributeRemaining(allocation,slices,costs,capacity)
  WHILE allocation < slices for each Content Provider DO
    FOR EACH Content Provider CP
      FIND surrogate with max(costs[CP,surrogate]/slices[CP]) for all Surrogate
      slices[CP] <- slices[CP] + 1;
    END-FOR;
  END-WHILE;
END-PROCEDURE;

```

Algorithm 5.1 – Distribution algorithm executed by resources manager

The final goal of the system is to allocate slices where the cost is highest. Each node communicate with each other and inform its calculated cost (from algorithm of phase 2). At the beginning of the process (trigger), each node has its own calculated costs. They transmit to others its state (the calculated costs) and also receive from them. When there is no change between two iterations (same allocation matrix for that node), it stops propagating the messages.

Although not necessary, the classical consensus protocol by Chandra and Toueg [169] can be used to accelerate the process of reaching a consensus. The Chandra and Toueg consensus protocol is based on the rotating coordinator paradigm. The protocol proceeds in asynchronous rounds and every round is coordinated by one of the nodes engaged in the system. It plays a central role in the protocol, trying to impose a consensual decision value to the others. In any given round, messages are only sent to the coordinator or received from the coordinator. As soon as a node suspects the current coordinator to have failed, it skips to the next round and tries to reach consensus with the help of the coordinator of that round. The properties of the failure detector is used to guarantee that eventually there will be a coordinator that is not suspected by the others and therefore will succeed in imposing its decision. Although not implemented in our simulator, a multi-level approach of the Chandra and Toueg protocol can be employed to avoid increased delay in large systems [150].

5.5.2 Evaluation

We conducted experiments with the proposed autonomic approach in two scenarios: a small proof of concept scenario, to test the feasibility of autonomic approach; and the scenario 2 described above. Since we are using a simulated environment, we simplified the process of reaching a consensus by allowing a coordinator to receive all the messages and propagate the final allocation, like the Chandra and Toueg protocol. The experiments show that the autonomic approach achieves almost the same performance when compared to the greedy slicing. Table 5.4 below shows the results for the proof of concept scenario, which is almost exactly the one described in section 5.4.1, but with four surrogates.

Table 5.4 – Autonomic results for proof of concept

	Slicing	Autonomic
Hit Ratio	3696	3692
Total traffic	24,377 Mb	24,381 Mb
Cross traffic	1,780 Mb	1,775 Mb
Startup Delay	0.02202	0.02207

For the scenario 2, the results are almost equal (Figure 5.30 and Figure 5.32), but the autonomic is slightly worse in some points, as it can be seen in Figure 5.31. In Figure 5.33, which show the accumulated cross traffic, it is possible to observe a decrease in performance after the second flash crowd.

Figure 5.30 – Cache hit progress for autonomic evaluation

Figure 5.31 – Startup delay progress for autonomic evaluation

Figure 5.32 – Total traffic progress for autonomic evaluation

Figure 5.33 – Cross traffic progress for autonomic evaluation

The autonomic approach is a very promising strategy given that it uses incomplete or limited information collected from the local monitor and still achieves almost the same result of the centralized greedy approach and, clearly, scalability is an important advantage. Also, since the change manager, monitor and resources manager involved only control local information, it should be easier to configure and maintain.

The main drawback, though, is the possible time the strategy may take to converge to the final configuration. This may lead to moments of undesired configuration that waste resources. As stated above, this can be avoided by employing some consensus protocol and, in future, we plan to test some of them.

5.6 Concluding remarks

Cache slicing represents an elegant and effective solution to the problem of virtualizing network resources for CDN providers with multiple tenants. The economy represented by resource allocation stability, fewer processing activities due to frequent network rearrangement, fewer network resources used, and better performance delivered to users can certainly leverage a CDN provider efficiency and profitability.

We evaluated two different cache slicing algorithms, based on the Hotspot and Greedy algorithms described in the literature, for content distribution and compared them to the use of a CDN without cache slicing. Cache preference is given to the CP that has more content requests or more stringent QoE requirements. This optimization is enforced to save resources. From the perspective of our results, using cache slicing not only represents considerable savings of network resources, but also provides better performance for a CDN serving more than one content provider, especially during flash crowds. Throughout our tests, using cache slicing allows a CDN to achieve much higher cache hit rates and results in lower response times, while at the same time reducing cross-traffic.

Thus, the use of cache slicing greatly improves the QoE results of the CDN. This happens because the use of slicing allows the CDN to behave as a set of CDNs, each one acting in the interest of its content provider, instead of a big CDN where content providers compete for resources and QoE. The management of the caches in a per-CP manner allows each set of content to be managed independently, as opposed to having content among different providers competing per popularity. The slicing, however, prevents this competition among content and changes it to a controllable competition for slices.

The autonomic approach has the advantage of using limited information collected from the local monitor and still achieves almost the same result of the centralized approach and, clearly, scalability is an important advantage. Also, since the change manager, monitor and resources manager involved only control local information, it should be easier to configure and maintain.

In future, we plan to link our framework with the emerging VTN effort proposed in the context of SDN to support CDN slicing. The goal is to map our strategies of CDN slicing as the controller box of VTN, thus they can be seen as the decision point whereas VTN is the enforcement technology that helps realize and manage the different virtual CDNs allocated to each tenant (content provider or set of CDN strategies).

6 Contributions and Future Works

In this thesis, we presented a categorization of Content Delivery Network and a framework for “virtual CDN”, in which each type of content or content provider can be handled by resizable and dynamic overlay built on top of a traditional infrastructure or a cloud environment. The framework is supported by an algorithm to detect relevant changes in CDN metrics and a module that performs CDN adaptation, by rearranging its infrastructure. This last chapter presents the main contributions of this thesis and delineates some future works.

6.1 Contributions of this work

Chapter four and five use a simulation tool developed to evaluate content distribution strategies. The simulation tool implementation was an essential step to a deeper understanding of problems related to CDN strategies. It is one of the contributions of this thesis. The detailed information and validation of the tool can be seen in Appendix I. Using this infrastructure, the overlay networks analyzed during this work were implemented, composing the necessary environment for the proposed approaches assessment. Several studies were taken to provide a better understanding about these architecture models and these analyses guided the definition of main features of the proposed approaches in this thesis work.

Chapter 2 presented some content distributions systems and the main issues related to content distribution. The main goal was to present two important metrics related to the QoE and used to analyze our proposals in evaluations, among the traditional ones: the startup delay and playback continuity.

In the chapter 3, we introduced the Content Delivery Networks, its model, architecture and main research challenges in order to have a systematic understanding of CDNs in general, as well as a categorization of CDNs, inserting our approach in model similar and inspired in the cloud-based CDNs.

One of the two main contributions, a technique to detect important changes in a monitored metric to allow adaptation in a dynamic CDN is presented in Chapter 4. We provided some improvements to the technique, the weighted moving average vector and

the ability to work with limited information from the monitor DB. We showed that even with this limited information, we can achieve results similar to an optimal omniscient environment. The results obtained from this Chapter were published in [125] and [127].

Finally, in the chapter five, we present the framework and an optimization algorithm to orchestrate the allocation of capacity in a multi-tenant CDN provider maintaining user perceived Quality-of-Experience. We extended the model to the context of self-organizing networks and achieved similar results. The results obtained from this Chapter, without the autonomic approach, were published in [126].

The main contributions of this thesis proposal can be resumed as follows:

- A literature review of content distributions techniques and the main challenges;
- A literature review of content delivery networks and architecture as well as its categorization;
- An extensible and powerful simulator tool designed for simulation of content delivery networks and other overlay networks to test and evaluate algorithm and strategies;
- An algorithm to detect significant changes in network conditions aimed to allow CDN adaptation, as well as an improvement to deal with restricted information;
- An algorithm to organize caching and storage adaptability of multi-tenant CDNs with maximum size restriction for small or cloud-based CDNs.
- We also examined the efficiency of using limited knowledge distributed slicing algorithms.

Some results presented in this Doctoral Thesis were developed in cooperation with the MCP - A Multi-CDN Peering Platform, a research project carried by the GPRT (Grupo de Pesquisa em Redes e Telecomunicações) and funded by Ericsson Telecomunicações S.A., Brazil.

6.2 Future works

A possible extension to the cache slicing solution is the possibility to the CP to return slices if they are not using them, the need for slices may change over time and the monitoring or trigger interval is not small enough to detect that change. Thus, this is a case of another condition for phase 3. The CP could also rent (at lower price) the unused space. Furthermore, the unused spaces of all CPs could form a pool that, at lower cost and with less stringent commitments by the CDN, could be used in some kind of secondary mode. The next steps include analyze the benefits of this model.

We also plan to investigate the feasibility of using the well-known principal component analysis method in our network changes detection system. The principal component analysis is a technique that can reduce multidimensional data, like our traffic information collected on nodes or links, by projecting it onto a lower dimensional subspace. It is widely used in anomaly detection studies.

The results of our self-organizing slicing proposal may be extended to the context of the emerging OpenDayLight Virtual Tenant Network, currently under development in the OpenDayLight forum. We also plan to test different consensus protocols to facilitate our autonomic approach to reach a consensus.

Finally, our research group managed to get a huge amount of real trace data and logs from a big content provider in Brazil. We are already studying these data, tracing some relevant pattern, behavior and characteristics of users. With these data, we plan to conduct some additional experiments and simulations.

References

- [1] A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, K.J. Worrell, **A hierarchical internet object cache**, in: ATEC'96: Proceedings of the Annual Technical Conference on USENIX 1996 Annual Technical Conference, USENIX Association, Berkeley, CA, USA, 1996. p. 13.
- [2] A. K. Pathan, and R. Buyya, **A Taxonomy and Survey of Content Delivery Networks**, Tech Report, Univ. of Melbourne, 2007.
- [3] A. S. Patrick, et al., **A QoE Sensitive Architecture for Advanced Collaborative Environments**, in Proceedings of the First International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks, 2004.
- [4] Abhigyan Sharma, Arun Venkataramani, and Ramesh Sitaraman. **Distributing content simplifies isp traffic engineering**. <http://arxiv.org/abs/1209.5715>, 2012.
- [5] Adler S **The Slashdot Effect, an Analysis of Three Internet Publications**. Available at <http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html>. 1999.
- [6] Alexandros Nanopoulos, Dimitrios Katsaros and Yannis Manolopoulos. **A Data Mining Algorithm for Generalized Web Prefetching**. IEEE Transactions on Knowledge and Data Engineering archive. Volume 15 Issue 5, September 2003. Page 1155-1169.
- [7] AtulSingh et al. **"Eclipse attacks on overlay networks: Threats and defenses"**, In IEEE INFOCOM, 2006.
- [8] Alzoubi, H. A., Rabinovich, M., and Spatscheck, O. **MyXDNS: a VoD dns server with decoupled server selection**. Proceedings of the 16th international conference on World Wide Web, 2007.
- [9] Anna Satsiou , Michael Paterakis. **Efficient caching of video content to an architecture of proxies according to a frequency-based cache management policy**. Proceedings of the 2nd international workshop on Advanced architectures and algorithms for internet delivery and applications, October 10-10, 2006, Pisa, Italy.
- [10] Ardaiz, O., Freitag, F, and Navarro, L., **Improving the Service Time of Web Clients Using Server Redirection**. ACM SIGMETRICS Performance Evaluation Review, Vol. 29, No. 2, ACM Press, NY, USA, pp. 39-44, 2001.
- [11] Arlitt M, Cherkasova L, Dilley J, Friedrich R, Jin T (1999) **Evaluating Content Management Techniques for Web Proxy Caches**. In: ACM SIGMETRICS Performance Evaluation Review, 27(4):3-11.
- [12] Akamai Technologies, www.akamai.com
- [13] Akamai. Improving Web Infrastructure Efficiency with Akamai. White Paper. Available at http://www.akamai.com/html/perspectives/whitepapers_content.html.
- [14] Akamai. **Visualizing Global Internet Performance with Akamai**, http://www.akamai.com/html/technology/visualizing_akamai.html
- [15] B. Li, M. J. Golin, G. F. Italiano, D. Xin, and K. Sohraby,. **On the Optimal Placement of Web Proxies in the Internet**. In Proceedings of IEEE INFOCOM, NY, USA, pp. 1282-1290, March 1999.
- [16] B. Wu and A.D. Kshemkalyani, **Objective-Optimal Algorithms for Long-Term Web Prefetching**. IEEE Trans. Computers, vol. 55, no. 1, pp. 2-17, Jan. 2006.

- [17] Bakiras, S., Loukopoulos, T.: **Increasing the performance of cdns using replication and caching: A hybrid approach**. In: IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), p. 92.2. IEEE Computer Society, Washington, DC, USA (2005).
- [18] Barbir, A., Cain, B., Nair, R., and Spatscheck, O. **Known content network request-routing mechanisms**. Internet Engineering Task Force RFC 3568, 2003. www.ietf.org/rfc/rfc3568.txt.
- [19] Bartolini, N., Casalicchio, E., and Tucci, S. **A walk through content delivery networks**. InProc. of the 11th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), LNCS Vol. 2965/2004, pp. 1–25, April 2004.
- [20] BBC. <http://www.bbc.co.uk>
- [21] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. **Accessing the deep web**. Commun. ACM, 50(5):94–101, 2007
- [22] Blond, S. L.; Legout, A. and Dabbous, W., **Pushing BitTorrent Locality to the Limit**, INRIA, Tech. Rep. 0034382, Dec. 2008.
- [23] Bob Trott. **Victoria's Secret for Webcasts is IP multicasting**. Aug. 1999. Available at <http://www.infoworld.com/articles/hn/xml/99/08/16/990816hnmentors.xml>.
- [24] Brussee, R., et al. **Content distribution network state of the art**. Telematica Instituut, 2001.
- [25] Buyya, R.; Pathan, A. K.; Broberg, J. & Tari, Z. (2006), **A Case for Peering of Content Delivery Networks**, in , IEEE Educational Activities Department, Piscataway, NJ, USA, pp. 3.
- [26] Buyya, Rajkumar, Pathan, Mukaddim, Vakali, Athena. **Content Delivery Networks**. 2008, XVI, 417 p.
- [27] Buyya, R., et al, **Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility**, Future Generation Computer Systems. Volume 25, Number 6, Pages: 599-616, June 2009.
- [28] C. W. Lin, J. Youn, J. Zhou, M.-T. Sun, and I. Sodagar, **MPEG video streaming with VCR functionality**. IEEE Trans. Circuits Syst. Video Technol., vol. 11, Mar. 2001.
- [29] Carapinha, J. and Jiménez, J., **Network virtualization: a view from the bottom**, Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures, 2009
- [30] Carlsson, N.; Dan, G.; Eager, D.; Mahanti, A, **Tradeoffs in cloud and peer-assisted content delivery systems**. Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on Sept. 2012.
- [31] Chen, Y., Qiu, L., Chen, W., Nguyen, L., Katz, R.H.: **Efficient and adaptive Web replication using content clustering**. IEEE J. Sel. Areas Commun. 21(6), 2003.
- [32] Cheng, B., Stein, L., Jin, H., and Zhang, Z. (2008). **Towards cinematic internet video-on-demand**. SIGOPS Oper. Syst. Rev. 42, 4 (Apr. 2008), 109-122.
- [33] Chenyu, P., Merdan, A., Mohammad, B.H., Toshihiko, S., Norihiko, Y., **FCAN: Flash Crowds Alleviation Network Using Adaptive P2P Overlay of Cache Proxies**, IEICE Trans. Commun., 2006.

- [34] CERT. **Denial of Service Attacks.** Available at http://www.cert.org/historical/tech_tips/denial_of_service.cfm. Access on Jul. 15th 2013.
- [35] Christina Warren. **Verizon FiOS Customers Get HBO's Online Video Service.** Available at <http://mashable.com/2010/02/17/hbo-go-verizon/>. Feb 17, 2010.
- [36] CISCO, 2014. **Cisco Visual Networking Index: Forecast and Methodology, 2012-2017.** Available at: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf. Access in Feb. 2014.
- [37] CISCO, 2015. **Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014-2019.** White paper. Available at http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf
- [38] CNN. <http://www.cnn.com>
- [39] Conejo, A.J., Castillo, E., Minguez, R., Garcia-Bertrand, R. **Decomposition Techniques in Mathematical Programming.** Engineering and Science Applications. 2006, XVI, 542 p
- [40] Content Delivery Network Interconnection (CDNI), <https://datatracker.ietf.org/wg/cdni/charter/>
- [41] CoralCDN. <http://www.coralcdn.org/>
- [42] Craig Partridge, Trevor Mendez, and Walter Milliken. **Host anycasting service.** IETF RFC 1546, November 1993.
- [43] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, “**Web Caching with Consistent Hashing,**” Computer Networks, Vol. 31, No. 11-16, pp. 1203-1213, 1999.
- [44] D. Rayburn, **CDN research data: Market sizing and pricing trends.** Streaming Media West: The Business and Technology of Online Video, <http://www.streamingmedia.com/east/presentations/CDNSummit09-CDNPricing.pdf>, 2009.
- [45] Dan Rayburn. **More isps not letting cdn place servers in-side their network, doing it themselves.** http://blog.streamingmedia.com/the_business_of_online_vi/2009/04/more-isps-not-letting-cdn-place-servers-inside-their-network-doing-it-themselves.html, April 2009. xv, 3
- [46] Dan Rayburn. **Here's What The Current CDN Landscape Looks Like, With List Of Vendors.** Available at <http://blog.streamingmedia.com/2013/12/heres-current-cdn-landscape-looks-like-list-vendors.html>. Dec. 10, 2013.
- [47] Dan Rayburn. **Comcast Launches Commercial CDN Service Allowing Content Owners To Deliver Content Via The Last Mile.** Available at <http://blog.streamingmedia.com/2014/05/comcast-launches-commercial-cdn-service-allowing-content-owners-to-deliver-content-inside-the-last-mile.html>. May 19, 2014.
- [48] Dán, G. & Carlsson, N. (2014). **Dynamic Content Allocation for Cloud-assisted Service of Periodic Workloads.** In: 33rd Annual IEEE (INFOCOM'14); Toronto, Canada, 27 April - May 2, 2014.

- [49] Dapeng Wu; Hou, Y.T.; Zhu, Wenwu; Ya-Qin Zhang; Peha, J.M., **Streaming video over the Internet: approaches and directions**. Circuits and Systems for Video Technology, IEEE Transactions on , vol.11, no.3, pp.282,300, Mar 2001.
- [50] DATANYZE. 2013. Available at <http://blog.datanyze.com/cdn-market-share-update-december-2013/> . Last access in July, 2014.
- [51] Day, M., Cain, B., Tomlinson, G., and Rzewski, P. **A model for content internetworking (CDI)**. Internet Engineering Task Force RFC 3466, 2003.
- [52] Dave Evans. **The Internet of Things. How the next evolution of Internet is changing everything**. Cisco Internet Business Solutions Group (IBSG). Available at. http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf April, 2011.
- [53] David Hamilton. **Mirror Image's New Edge Computing Framework Simplifies Edge Logic**. Web Host Industry Review. 27 September 2010.
- [54] Darren Murph. **Netflix introduces its own CDN, Open Connect Network, to give ISPs more control**. Available at <http://www.engadget.com/2012/06/04/netflix-introduces-its-own-cdn-open-connect-network/>. 2012.
- [55] Delgadillo, K., **Cisco Distributed Director**. Cisco White Paper, Cisco Systems, Inc., June 1997.
- [56] Dhamdhare, Amogh et al. **Measuring the Deployment of IPv6: Topology, Routing and Performance**. ACM Internet Measure-ment Conference (IMC), November 2012.
- [57] Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., and Weihl, B. **Globally distributed content delivery**.IEEE Internet Computing, pp. 50–58, 2002.
- [58] Dimitrios Katsaros, George Pallis, Konstantinos Stamos, Athena Vakali, Antonis Sidiropoulos, Yannis Manolopoulos, **CDNs Content Outsourcing via Generalized Communities**. IEEE Transactions on Knowledge and Data Engineering, vol. 99, no. 1, pp. 5555, 2008.
- [59] Dina Katabi and John Wrolawski. **A framework for scalable global ip anycast (gia)**. In Proceedings of ACM SIGCOMM 2000, pages 3-15, Stokholm, Sweden, August 2000.
- [60] Dongyan Xu, Sunil Suresh Kulkarni, Catherine Rosenberg, Heung-Keung Chai. **Analysis of a CDN-P2P Hybrid Architecture for Cost-Effective Streaming Media Distribution**. Multimedia Systems. April 2006, Volume 11, Issue 4, pp 383-399.
- [61] Dudouet, F. Harsh, P. ; Ruiz, S. ; Gomes, A. ; Bohnert, T.M. **A case for CDN-as-a-service in the cloud: A Mobile Cloud Networking argument**. ICACCI, 2014 International Conference on Advances in Computing, Communications and Informatics. Pages 651-657, Sep. 2014.
- [62] E.P. Markatos and C.E. Chironaki, **A Top 10 Approach for Prefetching the Web**. Proc. INET '98: Internet Global Summit, July 1998.
- [63] Elkotob, M.; Andersson, K., **Challenges and opportunities in content distribution networks: A case study**. Globecom Workshops (GC Wkshps), 2012 IEEE , vol., no., pp.1021,1026, 3-7 Dec. 2012.

- [64] Erik Nygren, Ramesh K. Sitaraman and Jennifer Sun. **The Akamai Network: A Platform for High-Performance Internet Applications**. ACM SIGOPS Operating Systems Review archive. Volume 44 Issue 3, July 2010. Pages 2-19.
- [65] ETSI GS NFV. **Network Functions Virtualization (NFV); Use cases**. Available at http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf
- [66] F. Douglis and M.F. Kaashoek, **Scalable Internet Services**. IEEE Internet Computing, vol. 5, no. 4, 2001, pp. 36–37.
- [67] F. Dressler. **Self-Organization in Sensor and Actor Networks**. Wiley, 2007.
- [68] Fangfei Chen; Guo, K.; Lin, J.; La Porta, T., **Intra-cloud lightning: Building CDNs in the cloud**. INFOCOM, 2012 Proceedings IEEE , vol., no., pp.433,441, 25-30 March 2012.
- [69] Feamster, N., Gao, L. and Rexford, J., **How to Lease the Internet in Your Spare Time**, ACM SIGCOMM Computer Communication Review, 2007
- [70] Fei, Z., Bhattacharjee, S., Zugura, E. W., and Ammar, M. H. **A novel server selection technique for improving the response time of a replicated service**. InProc. of IEEE INFOCOM, San Francisco, California, USA, pp. 783–791, 1998.
- [71] FORBES. **How Many Things Are Currently Connected To The "Internet of Things" (IoT)?** Forbes. Available at <http://www.forbes.com/sites/quora/2013/01/07/how-many-things-are-currently-connected-to-the-internet-of-things-iot/>. 1/07/2013. Access on Aug. 1st 2014.
- [72] Freedman MJ, Freudenthal E, Mazieres D (2004) **Democratizing Content Publication with Coral**. In: Proc. 1st USENIX/ACM Symp. on Networked Systems Design and Implementation..
- [73] G. Peng, **CDN: Content Distribution Network, Technical Report TR-125**, Experimental Computer Systems Lab, Department of Computer Science, State University of New York, Stony Brook, NY 2003.
- [74] G. Pierre, and M. van Steen, **Globule: A Collaborative Content Delivery Network**. IEEE Communications, Vol. 44, No. 8, August 2006.
- [75] Gadde, S., Rabinovich, M., and Chase, J. **Reduce, reuse, recycle: an approach to build-ing large Internet caches**. In Proc. of 6th Workshop on Hot Topics in Operating Systems.
- [76] Gayek, P., Nesbitt, R., Pearthree, H., Shaikh, A., and Snitzer, B. **A Web content serving utility**. IBM Systems Journal, 43(1), pp. 43–63, 2004.
- [77] Gordon, M. **The Internet streaming media boom: a powerful trend that represents fundamental change**. Limelight Networks, 2007. www.limelightnetworks.com
- [78] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng. **Understanding User Behavior in Large-Scale Video-on-demand Systems**. In Proc. of ACM Eurosys, 2006.
- [79] Hayley Ringle. **Limelight Networks wins 8-year-old patent infringement case against Akamai**. Phoenix Business Journal. Available at <http://www.bizjournals.com/phoenix/blog/techflash/2014/06/tempes-limelight-networks-wins-in-a-8-year.html>. Jun 2, 2014.

- [80] Herrmann, K., **Self-organizing Replica Placement - A Case Study on Emergence**. Self-Adaptive and Self-Organizing Systems, 2007. SASO '07. First International Conference on , vol., no., pp.13,22, 9-11 July 2007.
- [81] Hiawatha Bray. **Akamai, MIT hit Limelight with patent suit. Companies are rivals in Net content delivery**. Bostom.com. Available at http://www.boston.com/business/technology/articles/2006/07/15/akamai_mit_hit_limelight_with_patent_suit/ July 15, 2006.
- [82] High Scalability. **Paper: The Akamai Network - 61,000 Servers, 1,000 Networks, 70 Countries**. Available at <http://highscalability.com/blog/2011/8/18/paper-the-akamai-network-61000-servers-1000-networks-70-coun.html> Aug. 18, 2011
- [83] Hofmann, M. and Beaumont, L. R. **Content Networking: Architecture, Protocols, and Practice**. Morgan Kaufmann Publishers, San Francisco, CA, USA, pp. 129–134, 2005.
- [84] I. Lazar and W. Terrill, **Exploring content delivery networking**. IEEE IT Professional, Vol.3, Issue 4, Jul/Aug 2001.
- [85] IBM, **An architectural blueprint for autonomic computing**. Technical Report, 2005. Available at: http://www.ginkgo-networks.com/IMG/pdf/AC_Blueprint_White_Paper_V7.pdf, Last access in October, 2013.
- [86] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. **Freenet: A distributed anonymous information storage and retrieval system**. In Workshop on Design Issues in Anonymity and Unobservability, pages 46-66, 2000.
- [87] IDC. **Worldwide Server Power and Cooling Expense 2006–2010 Forecast**. Doc #203598. September 2006.
- [88] International Standards Organization (ISO), Open systems interconnection–basic reference model. ISO 7498, 1989.
- [89] J. Dai, Z. Hu, B. Li, J.-C. Liu and B. Li, **Collaborative Hierarchical Caching with Dynamic Request Routing for Massive Content Distribution**. IEEE Infocom, Orlando, Florida, March 25-30, 2012.
- [90] J. Fritz Barnes and R. Pandey, **CacheL: Language Support for Customizable Caching Policies**. Proc. Fourth Int’l Web Caching Workshop (WCW), 1999.
- [91] J. Goodchild, **Integrating data, voice and video – Part II**. IP Video Implementation and planning guide, United States Telecom Association, 2005.
- [92] James Broberg, Rajkumar Buyya, Zahir Tari. **MetaCDN: Harnessing ‘Storage Clouds’ for high performance content delivery**. Journal of Network and Computer Applications, Volume 32, Issue 5, September 2009, Pages 1012-1022.
- [93] Jamin, S., Jin, C., Jin, Y., Raz, D., Shavitt, Y., and Zhang, L. **On the placement of Internet instrumentation**. In Proc. of IEEE INFOCOM, Tel-Aviv, Israel, pp. 295–304, 2000.
- [94] Jung J, Krishnamurthy B, Rabinovich M (2002) **Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites**. In: Proc. 11th Int. World Wide Web Conf., 252–262.

- [95] K. Hosanagar, R. Krishnan, M. Smith, and J. Chuang, **Optimal Pricing of Content Delivery Network Services**. Proc. IEEE Hawaii Int'l Conf. System Sciences (HICSS '04), vol. 7, 2004.
- [96] K. L. Johnson, J. F. Carr, M. S. Day, and M. F. Kaashoek, **The measured performance of content distribution networks**. Computer Communications, vol. 24, no. 2, pp. 202-206, 2001.
- [97] K. Wu, P. S. Y u, and J. Wolf. **Segment-based proxy caching of multimedia streams**. In Proceedings of WWW, Hongkong, China, May 2001.
- [98] Kangasharju, J., Roberts, J., and Ross, K., **Object Replication Strategies in Content Distribution Networks**. Computer Communications, Vol. 25, No. 4, March 2002.
- [99] Karger, D., Sherman, A., Berkheimer, A., Bogstad, B., Dhanidina, R., Iwamoto, K., Kim, B., Matkins, L., and Yerushalmi, Y. **Web caching with consistent hashing**. Computer Networks, 31(11-16), pp. 1203-1213, 1999.
- [100] Keynote Systems, www.keynote.com
- [101] Kishigami, J. 2007. **The Role of QoE on IPTV Services style**. In Proceedings of the Ninth IEEE international Symposium on Multimedia (ISM 2007) - December, 2007.
- [102] Krishnamurthy, B., Willis, C., and Zhang, Y. **On the use and performance of content distribution network**. In Proc. of 1st International Internet Measurement Workshop, ACM Press, pp. 169-182, 2001.
- [103] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, **On the Placement of Web Server Replicas**. In Proceedings of IEEE INFOCOM, Anchorage, Alaska, USA, pp. 1587-1596, April 2001.
- [104] L. Wang, K. S. Park, R. Pang, V. S. Pai, and L. Peterson, **Reliability and Security in CoDeeN Content Distribution Network**. In Proceedings of the USENIX 2004 Annual Technical Conference, Boston, MA, USA, June 2004.
- [105] Lee, Chae-Sub, **IPTV over Next Generation Networks in ITU-T**. Broadband Convergence Networks, BcN '07. 2nd IEEE/IFIP International Workshop on, pp. 1-18, 21-21, May 2007.
- [106] Limelight. <http://www.limelightnetworks.com/>
- [107] Li, H., Zhong, L., Liu, J., Li, B., Xu, K.: **Cost-effective partial migration of vods services to content clouds**. In: Proc. of IEEE CLOUD (2011)
- [108] Lin, C.F., Leu, M.C., Chang, C.W., Yuan, S.M.: **The Study and Methods for Cloud based CDN**. In: Proc. of IEEE CyberC (2011)
- [109] Liu, J., Rao, S. G., Li, B., Zhang, H., **Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast**. Proceedings of the IEEE, 2006.
- [110] M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, and F. Zane, **Clustering and Server Selection Using Passive Monitoring**. In Proceedings of IEEE INFOCOM, NY, USA, 2002.
- [111] M. Day, B. Cain, G. Tomlinson, and P. Rzewski, **A model for content inter-networking (CDI)**. Internet Engineering Task Force RFC 3466, <http://www.ietf.org/rfc/rfc3466.txt>, 2003.
- [112] M. Green, B. Cain, G. Tomlinson, and S. Thomas. **CDN peering architecture overview**. Internet-Draft, November 2000.

- [113] M. Hollander, D. A. Wolfe, and E. Chicken. **Nonparametric statistical methods**. Vol. 751. John Wiley & Sons, 2013
- [114] M. Karlsson, C. Karamanolis, and M. Mahalingam. **A framework for evaluating replica placement algorithms**. Technical Report HPL-2002-219, HP Labs, 2002.
- [115] M. Siller and J. Woods, **Improving quality experience for multimedia services by QoS arbitration on a QoE framework**. in Proceedings of the 13th Packed Video Workshop 2003, Nantes, France, 2003.
- [116] M. Szymaniak, G. Pierre, and M. van Steen, **Netairt: A DNS-based Redirection System for Apache**. In Proceedings of International Conference WWW/Internet, Algrave, Portugal, 2003.
- [117] M. Taylor, **A Guide to NFV and SDN**. Metaswitch Networks (Whitepaper), 2014.
- [118] Mao, Z. M et al. **A precise and efficient evaluation of the proximity between Web clients and their Local DNS servers**. In Proc. of the USENIX 2002 Annual Technical Conference, Monterey, CA, USA, pp. 229–242, 2002.
- [119] Martin Arlitt and Tai Jin. **Workload Characterization of the 1998 World Cup Web Site**. HPL-1999-35R1.
- [120] Masa, M. Parravicini, E. **Impact of request-routing algorithms on the delivery performance of content delivery networks**. Proceedings of the IEEE International Conference on Performance, Computing, and Communications, pp. 5-12, 2003.
- [121] Mihael R. Garey and David S. Johnson. **Computers and intractability**. NY, NY: W. H. Freeman and Co., 1979.
- [122] Mikkilineni, Rao; Sarathy, Vijay, **Cloud Computing and the Lessons from the Past**. In Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE '09. 18th IEEE International Workshops, July, 2009.
- [123] Mingchen Zhao et al. **Peer-Assisted Content Distribution in Akamai NetSession**. IMC '13 Proceedings of the 2013 conference on Internet measurement conference. 2013. Pages 31-42.
- [124] Mirror Image Internet. **Content Delivery and the Mirror Image Adaptive CAP Network**, 2007, www.mirror-image.com
- [125] Moreira, A., Moreira, J., Sadok, D., Callado, A., Rodrigues., Neves, M., Souza, V., and Karlsson, P. 2011. **A Case for Virtualization of Content Delivery Networks**. In: IEEE. Winter Simulation Conference.
- [126] Moreira, A. L. C. et al. **An Adaptable Storage Slicing Algorithm for Content Delivery Networks**. In: International Symposium on Integrated Network Management, 2015, Ottawa. International Symposium on Integrated Network Management, 2015.
- [127] Moreira, A. L. C. et al. **Analyzing Strategies to Effectively Detect Changes in Content Delivery Networks**. In: International Symposium on Integrated Network Management, 2015, Ottawa. International Symposium on Integrated Network Management, 2015.
- [128] Moreno, B. M et al., **On content delivery network implementation**. Computer communications 29(12), 2006.

- [129] Mukaddim Pathan, Rajkumar Buyya. **Architecture and Performance Models for QoS-Driven Effective Peering of Content Delivery Networks**. Multiagent and Grid Systems, Volume 5, Number 2, Pages: 165-195, ISSN: 1574-1702, IOS Press, Amsterdam, The Netherlands, July 2009,
- [130] N. Fujita, Y. Ishikawa, A. Iwata, and R. Izmailov, **Coarse-Grain Replica Management Strategies for Dynamic Replication of Web Contents**. Computer Networks, vol. 45, no. 1, pp. 19-34, 2004.
- [131] Norihiko Yoshida. Dynamic CDN Against Flash Crowds. Book Chapter. **Content Delivery Networks**. Lecture Notes Electrical Engineering Volume 9, 2008, pp 275-296
- [132] NetFlix. <http://www.netflix.com>
- [133] Nikolaos Laoutaris, Vassilios Zissimopoulos and Ioannis Stavrakakis. **On the optimization of storage capacity allocation for content distribution**. Computer Networks: The International Journal of Computer and Telecommunications Networking. Volume 47 Issue 3, February, 2005. Pages 409-428.
- [134] Om Malik. **Akamai Sues Limelight Networks**. GigaOm. Available at <http://gigaom.com/2006/07/11/akamai-sues-limelight-networks/> . Jul. 11, 2006.
- [135] P. Krishnan, D. Raz, and Y. Shavitt, **The Cache Location Problem**. IEEE/ACM Transaction on Networking, Vol. 8, No. 5, 2000.
- [136] P. Rodriguez, C. Spanner, and E. W. Biersack, **Analysis of Web Caching Architectures Hierarchical and Distributed Caching**. IEEE/ACM Transactions on Networking, vol. 9, no. 4, 2001.
- [137] P. Sharma, Edward Perry, and Radhika Malpani. **IP multicast operational network management: design, challenges, and experiences**. in IEEE Network, p. 49-55, 2003.
- [138] Pai, V. S., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W., Nahum, E. **Locality-aware request distribution in cluster-based network servers**. ACM SIGPLAN Notices, 33(11), ACM Press, NY, USA, pp. 205–216, 1998.
- [139] Pallis, G. and Vakali, A. **Insight and perspectives for content delivery networks**. Communications of the ACM, 49(1), ACM Press, NY, USA, pp. 101–106, 2006.
- [140] Padmanabhan VN, Sripanidkulchai K. (2002) **The Case for Cooperative Networking**. In: Proc. 1st Int. Workshop on Peer-to-Peer Systems, 178-190.
- [141] Papagianni, C.; Leivadeas, A; Papavassiliou, S., **A Cloud-Oriented Content Delivery Network Paradigm: Modeling and Assessment**. Dependable and Secure Computing, IEEE Transactions on , vol.10, no.5, pp.287,300, Sept.-Oct. 2013.
- [142] Park, K.S. and Pai, V.S. and Peterson, L. and Wang, Z., **CoDNS: improving DNS performance and reliability via cooperative lookups**. In Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation. USENIX Association Berkeley, CA, vol. 6, 14-14, 2004.
- [143] Pierre, and M. van Steen, **Globule: A Collaborative Content Delivery Network**. IEEE Communications, Vol. 44, No. 8, August 2006.

- [144] Plagemann, T., Goebel, V., Mauthe, A., Mathy, L., Turetti, T. and Urvoy-Keller, G., **From content distribution networks to content networks - issues and challenges**, Computer Communications, vol. 29, issue 5, 2006.
- [145] Platform as a Service Magazine. **Announcing Azure Media Services integration with Azure CDN**. Available at <http://www.paasmag.com/2015/03/17/announcing-azure-media-services-integration-with-azure-cdn-content-delivery-network/>.
- [146] Popocorn Time, <http://www.time4popocorn.eu/>
- [147] Poularakis, K. ; Dept. of Electr. & Comput. Eng., Univ. of Thessaly, Volos, Greece; Iosifidis, G. ; Tassioulas, L. **Approximation Algorithms for Mobile Data Caching in Small Cell Networks**. Communications, IEEE Transactions on (Volume:62 , Issue: 10). Pages 3665 - 3677, Oct. 2014.
- [148] Rabinovich, M. et al., **A dynamic object replication and migration protocol for an Internet hosting service**. in: Proceedings of the IEEE International Conference on Distributed Computing Systems, May 1999.
- [149] Ramesh K. Sitaraman, Mangesh Kasbekar, Woody Lichtenstein and Manish Jain. **Overlay Networks: An Akamai Perspective**. Akamai Technologies Inc. University of Massachusetts, Amherst. 2014.
- [150] Rahul Agarwal, Mahender Bisht, S. N. Maheshwari, and Sanjiva Prasad. 2005. **Divide and concur: employing chandra and toueg's consensus algorithm in a multi-level setting**. In Proceedings of the Second international conference on Distributed Computing and Internet Technology (ICDCIT'05), 172-183.
- [151] RFC 1945, Available at <http://www.isi.edu/in-notes/rfc1945.txt>.
- [152] Rodrigues, M. B. E. ; MOREIRA, A. L. C. ; AZEVEDO, E. ; NEVES, MARCIO ; SADOK, DJAMEL ; Callado, A. ; Moreira, J. . **On Learning How to Plan Content Delivery Networks**. In: 46th Annual Simulation Symposium, 2013, San Diego. Proceeding of 46th Annual Simulation Symposium, 2013.
- [153] Rousskov, A. and Wessels, D. Cache digests. **Computer Networks and ISDN Systems**, 30(22), pp. 2155–2168, November 1998.
- [154] S. Gadde, J. Chase, and M. Rabinovich, **Web Caching and Content Distribution: A View From the Interior**. Computer Comm., vol. 24, nos. 1-2, 2001, pp. 222–231.
- [155] S. Jamin, C. Jin, A. R. Kure, D. Raz, and Y. Shavitt, **Constrained Mirror Placement on the Internet**. In Proceedings of IEEE INFOCOM, Anchorage, Alaska, USA, April 2001.
- [156] S. Sen, J. Rexford, and D. T. Owsley . **Proxy pre-caching for multimedia streams**. In Proceedings of IEEE INFOCOM, New York City NY, March 1999.
- [157] S. Srinivasan, J.W.J. Lee, D. Batni, and H. Schulzrinne, **Active-CDN: Cloud Computing Meets Content Delivery Networks**. technical report, Columbia Univ., 2012.
- [158] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. Van Steen, **Replication of Web Hosting Systems**. ACM Computing Surveys, Vol. 36, No. 3, ACM Press, NY, USA, 2004.
- [159] Sargent, R. G. 2005. **Verification and validation of simulation models**. In Proceedings of the 37th Conference on Winter Simulation. 130-143.

- [160] Sean Michael Kerner. **DDoS Attacks: Growing, but How Much?**. Available at <http://www.esecurityplanet.com/network-security/ddos-attacks-growing-but-how-much.html>. eSecurity Planet. April 26, 2013.
- [161] Shen, G.; Wang, Y.; Xiong, Y.; Zhao, B. Y. and Zhang, Z., **HPTP: Relieving the Tension between ISPs and P2P**, in IPTPS, 2007.
- [162] Shibsankar Das and Jussi Kangasharju. **Evaluation of Network Impact of Content Distribution Mechanisms**. InfoScale '06 Proceedings of the 1st international conference on Scalable information systems. Article No. 35, 2006.
- [163] Schollmeier, R., **A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications**. Peer-to-Peer Computing, 2001. Proceedings. First International Conference on , vol., no., pp.101,102, Aug 2001.
- [164] Scott M. Fulton. **Vomcast on xbox: Does a cdn, by nature, threaten the internet?** Available at http://www.readwriteweb.com/archives/comcast_on_xbox_does_a_cdn_by_nature_threaten_the.php, March 2012.
- [165] Songqing Chen; Haining Wang; Xiaodong Zhang; Bo Shen; Wee, S., **Segment-based proxy caching for Internet streaming media delivery**. MultiMedia, IEEE , vol.12, no.3, pp.59,67, July-Sept. 2005.
- [166] Stamos, K., Pallis, G., and Vakali, A. **Integrating caching techniques on a content distribution network**. In Proc. of 10th East-European Conference on Advances in Databases and Information Systems (ADBIS 2006), Springer-Verlag, Thessaloniki, Greece, 2006.
- [167] Stamos, K., Pallis, G., Thomos, C., and Vakali, A. **A similarity-based approach for integrated Web caching and content replication in CDNs**. In Proc. of 10th International Databased Engineering and Applications Symposium (IDEAS 2006), IEEE Press, New Delhi, India, 2006.
- [168] Stamos, K., Pallis, G., Vakali, A., Katsaros, D., Sidiropoulos, A. and Manolopoulos, Y. 2009, **CDNsim: A Simulation Tool for Content Distribution Networks**, ACM Transactions on Modeling and Computer Simulation, Volume 5, 2009.
- [169] T. Chandra and S. Toueg. **Unreliable failure detectors for reliable distributed systems**. Journal of the ACM, 43(2):225–267, Mar 1996.
- [170] T. Henderson, S. Roy, S. Floyd, and G.F. Riley, **ns-3 project goals**, in Proceedings of the 2006 workshop on ns-2: the IP network simulator, Pisa, Italy, October 2006.
- [171] T. Leighton. **Akamai and cloud computing: A perspective from the edge of the cloud**, White Paper, Akamai Technologies, Inc., <http://www.akamai.com/cloud>, 2009.
- [172] T. M. O' Neil, **Quality of Experience and Quality of Service for IP Video Conferencing**, Polycom, Whitepaper, 2002.
- [173] Tang, W., Fu, Y., Cherkasova, L., and Vahdat, A. 2003. **Medisyn: A synthetic streaming media service workload generator**. In Proceedings of the 13th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video, pp. 12-21.
- [174] Tony Danova. **Morgan Stanley: 75 Billion Devices Will Be Connected To The Internet Of Things By 2020**. Business Insider. Available at <http://www.businessinsider.com/75-billion-devices-will-be-connected-to-the-internet-by-2020-2013-10>. Oct. 2, 2013.

- [175] Tor project, <https://www.torproject.org/>
- [176] Vaishnav Janardhan and Henning Schulzrinne. **Peer assisted VoD for set-top box based IP network**. P2P-TV '07 Proceedings of the 2007 workshop on Peer-to-peer streaming and IP-TV. 2007. Pages 335-339.
- [177] Valloppillil, V. and Ross, K. W. **Cache array routing protocol v1.0**. Internet Draft, 1998.
- [178] Vakali, A. and Pallis, G. **Content delivery networks: status and trends**. IEEE Internet Comput-ing, 7(6), IEEE Computer Society, pp. 68–74, 2003.
- [179] Venkataramani, A. et al. **The Potential Costs and Benefits of Long-Term Prefetching**, Computer Comm., vol. 25, no. 4, pp. 367-375, 2002.
- [180] Völker, L., Martin, D., El Khayaty, I., Werle, C. and Zitterbart, M., **A Node Architecture for 1000 Future Networks**, Workshop of the IEEE International Conference on Communications, 2009.
- [181] W. Li, E. Chan, G. Feng, D. Chen, and S. Lu, **Analysis and Performance Study for Coordinated Hierarchical Cache Placement Strategies**. Computer Communications, vol. 33, no. 15, 2010.
- [182] Wang, L. Pai, V. Peterson, L. **The effectiveness of request redirection on CDN robustness**. Proceedings of the 5th symposium on Operating systems design and implementation (OSDI 02), vol. 36, pp.345-360, 2002.
- [183] Web Lens, www.appliant.com
- [184] Wenjie Jiang, Rui Zhang-Shen, Jennifer Rexford, and Mung Chiang. **Co-operative content distribution and traffic engineering in an isp network**. In ACM SIGMETRICS, 2009.
- [185] Wessels, D. and Claffy, K. Internet cache protocol (ICP) version 2. Internet Engineering Task Force RFC 2186, 1997. www.ietf.org/rfc/rfc2186.txt.
- [186] X. Jia, D. Li, H. Du, and J. Cao, **On Optimal Replication of Data Object at Hierarchical and Transparent Web Proxies**, IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 8, 2005.
- [187] Xiao, Y.; Du, X.; Zhang, J.; Hu, F.; Guizani, S., **Internet Protocol Television (IPTV): The Killer Application for the Next-Generation Internet** IEEE Communications Magazine, vol.45, n° 11, pp.126-134, November 2007.
- [188] Xiaojun Hei, Yong Liu, Ross, K.W. **Inferring Network-Wide Quality in P2P Live Streaming Systems**. IEEE Journal on Selected Areas in Comm., Vol 25, Dec 2007.
- [189] Y. Bartal, **Probabilistic Approximation of Metric Space and its Algorithmic Applications**. In Proceedings of 37th Annual IEEE Symposium on Foundations of Computer Science, October 1996.
- [190] Y. Chen, R. H. Katz, and J. D. Kubiawicz, **Dynamic Replica Placement for Scalable Content Delivery**. In Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS 02), LNCS 2429, Springer-Verlag, pp. 306-318, 2002.
- [191] Y. Chen, L. Qiu, W. Chen, L. Nguyen, and R.H. Katz, **Efficient and Adaptive Web Replication Using Content Clustering**. IEEE J. Selected Areas in Comm., vol. 21, no. 6, pp. 979-994, 2003.
- [192] Y. Jiang, M.Y. Wu and W. Shu, **Web Prefetching: Cost, Benefits and Performance**, Proc. 11th World Wide Web Conf. (WWW), 2002.

- [193] Y. Jin, Y. Wen, G. Shi, G. Wang, and A. Vasilakos, **CoDaaS: An Experiment Cloud-Centric Content Delivery Platform for User-Generated Contents**, Proc. Int'l Conf. Computing, Networking and Comm. (ICNC), pp. 934-938, Feb. 2012
- [194] Y. Li and M. T. Liu, **Optimization of performance gain in content distribution networks with server replicas**, Proc. International Symposium on Applications and the Internet (SAINT'03), pp. 182–189, IEEE CS Press, Los Alamitos, CA, USA, 2003.
- [195] Ying Lu, Tarek F. Abdelzaher and Avneesh Saxena. **Design, Implementation, and Evaluation of Differentiated Caching Services**. IEEE Transactions on Parallel and Distributed Systems archive. Volume 15 Issue 5, May 2004 Page 440-452.
- [196] Yousaf, F.Z. ; Liebsch, M. ; Maeder, A. ; Schmid, S. **Mobile CDN enhancements for QoE-improved content delivery in mobile operator networks**. Network, IEEE (Volume:27 , Issue: 2). Pages 14 - 21, Mar. 2013.
- [197] Youtube Statistics. Available at <http://www.youtube.com/yt/press/statistics.html>. Access on Aug. 1st 2014.
- [198] Zao W, Schulzrinne H. **DotSlash: A Self-Configuring and Scalable Rescue System for Handling Web Hotspots Effectively**. In: Proc. Int. Workshop on Web Caching and Content Distribution, 1–18. 2004.
- [199] Zapater, Marcio Nieblas; Bressan, Graca. **A Proposed Approach for Quality of Experience Assurance of IPTV**, In 27th International Conference on Distributed Computing Systems, (ICDCS '07), June 2007.
- [200] Zhe Li et al. **Network friendly video distribution**. In International Conference on the Network of the Future, 2012. xiii, 1
- [201] Zhu, Y., Zhang-Shen, R., Rangarajan S. and Rexford, J., **Cabernet: connectivity architecture for better network services**, Proceedings of the ACM CoNEXT Conference, 2008.
- [202] Zink, M., Suh, K., Gu, Y. and Kurose, J. **Watch Global Cache Local: YouTube Network Traces at a Campus Network - Measurements and Implications**, IEEE MMCN, 2008.

Appendix I

CDN and P2P Simulator

P2PCDNSim is a user friendly simulator designed to evaluate CDN strategies. The tool is highly flexible, as it allows for seamless CDN creation; instantiation and addition into a scenario; definition and experimenting with new surrogate placement strategies; determining how a given CDN configuration would attend a usage scenario; predicting requirements and constraints before deployment; and the evaluation of potential deployment scenarios.

It is a Java discrete-event based simulation tool used for engineering and debugging CDN projects, multiple CDNs and P2P network overlays. It simulates complex scenarios, including hybrid CDN and P2P overlays, and evaluates the gain from CDN cooperation and peering benefits. It also provides clear separation between layers, facilitating the process of implementing and testing new strategies; an easy GUI to create new scenarios; a collector for a set of built-in metrics with real time display of graphs during the simulations while new metrics can be seamlessly added; a realistic representation of Autonomous Systems, AS aware routing and AS-cross traffic monitoring; and a GUI tool for observing a running simulation by monitoring relevant metrics and their color coded evolution on the graphs.

The interested reader may see a demonstration scenario in GUI at <https://youtu.be/2ZovxyHyJrg>.

1 Simulator overview

The simulator adopts a number of abstraction levels. There are many ways in which a CDN project may be built. Some are based on fixed infrastructure, others use P2P, and there those that are hybrid or Cloud based. As a result, our first decision was to establish a clear separation between these different overlays (operating modes) and the common underlying support strategies and functionalities which include caching strategies, placement algorithms, the simulator's event scheduler, metric collection functions and graph plotting services. A case that demonstrates the wisdom of this design choice is the recent CDN Interconnection (CDNI) module we developed to evaluate CDN

peering. All we needed to add was a new CDNI overlay while we continued using the base P2PCDNSim (underlay) module and other CDN overlays.

A second design decision was made to support geographical location through an advanced realistic globe shaped interface. A user may examine real-time relevant metrics by monitoring their color coded evolution on the graphs (Figure I.1). Animated output conveys important knowledge and allows observing system evolution at a controlled pace. Animators have successfully used this technique in simulators such as the ns-2⁴.

Figure I.1 – Simulator GUI tool

To speed up development, the Desmo-J⁵ framework and JFreeChart⁶ were used for modeling discrete events and drawing graphs respectively. From a structural perspective, this simulator has been divided into three main parts: core, network and overlay. At its core, we find the clock, an event scheduler and data collectors. Above this lies the network level that faithfully mimics a communication network, topology information, protocols and links properties such as their symmetry, capacity, delay, packet loss, congestion, jitter and throughput information. The third abstraction level is that of the overlay CDN, i.e., the actual simulation target. Processes make up the entities at the first layer which are then mapped onto their discrete events. To create a new entity,

⁴ <http://www.isi.edu/nsnam/ns/>

⁵ Desmo-J, <http://desmoj.sourceforge.net/>

⁶ JFreeChart, <http://www.jfree.org/jfreechart/>

like a surrogate for example, an abstract class needs to be extended with new functionalities, the surrogate behavior, according to the Object Oriented paradigm.

A third important decision made in this project was to actually implement transport and network level protocols at the second abstraction architectural level (Figure I.2). Hence, all TCP functionalities including congestion control, packet retransmissions and timeouts are supported according to the RFC5681. Furthermore, to reflect user broadband access, the simulator supports symmetric and asymmetric connectivity as in ADSL access networks. Symmetric versus asymmetric node contributions are important to examine, especially in the case of P2P based CDNs. At the simulator's base layer, a number of XML based configuration files are used to select among the supported strategies, scenario definition, expected output metrics, reports and logs with detailed event tracing, etc.

Figure I.2 – Simulator architecture

The current simulator's primary drawback, memory consumption, is caused by its faithful portrayal of functionalities such as underlying communication protocols and real-time animation. For example, for each packet, a Java object is associated with all the inherent overhead. A router's queue maintains these in memory. Another possible limitation comes from our usage of processes to represent entities. Operating systems usually limit their number at configuration time, but this may be altered if needed. Furthermore, during the development phase, we sought to mitigate this problem by factoring several functions into single multirole CDN entities.

1.1 Link Layer

This layer is represented by the Link class and is responsible for the exchange of frames and updating time-related calculations according to the intrinsic link delay, the data that will be sent, link capacity and the size of buffer. When a buffer is full, the data is discarded and a packet loss event is raised. This event can be collected and logged for future analysis.

Each Node has a set of links where each one connects two nodes. There are two main Node types: Router and Client nodes. The Router nodes are responsible for taking decisions regarding packet switching. The set of links for each node will depend on the type of node and its connections. For a Router node, all its links are symmetric in terms of bandwidth capacity as well as being full duplex. Client nodes differ due to user churn and possible presence of asymmetric links. Link characteristics are set through the client's XML configuration file.

1.2 Network Layer

The layer three routing table is generated once per simulation and depends on the type of AS. The table may be set through the simulator's properties file and represents AS information in the forms: Regional, Unique and Multi. When using the Regional mode, each AS will be a subset of nodes previously set by the topology XML file (Figure I.3). In the Unique mode, all nodes will be seen as being part of a single AS (Figure I.4). Finally, the Multi AS representation, tells the planner that every node is considered a separate AS.

Figure I.3 – Explicit definition of each AS

Figure 1.4 – Single-AS Network topology

In Unique and Multi AS representations, the shortest path algorithm is used to obtain the routing tables. When using the Regional representation, other concerns are taken into account, since an AS is a set of nodes, intra AS routing is different from inter AS routing in a way that all intra AS traffic, which represents the traffic that has its source and sink nodes belonging to the same AS, should cross only nodes from that AS even if it leads to using a longer path. This default behavior enforces the common belief that intra-AS paths are preferred to inter-AS ones even when longer (for example, in Figure 3, nodes 13 and 42 are separated by 7 intra-AS1 hops or 5 inter-AS hops, but the first path is preferred). The second restriction is that in inter AS traffic, this traffic should only pass through the AS source, the destination AS and border nodes in the way to the destination AS. This is ensured when generating router tables, as all possible paths are checked against these two restrictions.

1.3 Transport Layer

Both TCP and UDP transport protocols are effectively supported by the simulator. Their implementations are completely transparent to the layers above so that the implementation of a new transport strategy or technique can focus on the design of its specifics on a high abstraction level. For example, for the inclusion of asymmetric links, only the parsing of the configuration file was changed. In another example, including cache cooperation only required changes in the cache object and the request redirector.

Our UDP simulation code is simple. The TCP protocol implementation covers RFC5681, the connection handshake negotiation and the packet timeout calculations according to RFC2988. The implemented TCP based flow control uses the sliding

window mechanism. The behavior of the congestion window includes Fast Retransmit and Fast Recovery algorithms. RTT is calculated according to RFC2988.

1.4 Application Layer

This section will describe the supported CDN and CDN Interconnection overlays designed for video distribution.

1.4.1 CDN Overlay

The Origin Server, Surrogate Server and Request Redirector have all been designed. The Origin Server is responsible for storing the complete original content. As in a CDN overlay, it holds the content serving as a content source which remains always available for access when caches do not have the requested content. Note that clients may never contact the origin server directly. The Surrogate servers, also known as cache or replica servers, are servers located near clients with considerable storage space capacity in order to cache content requested by clients. The Request Redirector is the entry point of the CDN for any client and is responsible for selecting the most suitable surrogate server for a client's request.

Four types of strategies or CDN sub-modules have been included into the overlay. They offer decisions for the placement of surrogate servers, caching optimization techniques, request routing algorithms, and content outsourcing.

Surrogate placement refers to the replica/cache placement problem. It emphasizes the issue of choosing the best, according to previously selected metrics, location for each surrogate and the number of surrogates needed for an optimal operation within a network infrastructure. The simulator offers the following built-in cache placement strategies: Greedy, HotSpot and Random.

With regard to caching strategies the P2PCDNSim offers Least Recently Used (LRU) and Least Frequently Used (LFU) Whole media. The LRU Whole media caching strategy considers the video as a single media object. It caches only the whole media according to the LRU policy. The LFU Whole media is similar.

Content outsourcing sets the relationship between surrogates and the origin server into a CDN. There are two main policies for content outsourcing available for use: cooperative pull-based and non-cooperative pull-based.

The request-routing module is responsible for directing client requests to a suitable surrogate server, based on traffic forwarding policies or metrics. This is an important part of the simulator that must be well engineered for resource optimization. A chosen surrogate server is considered closest to the client through the use of metrics such as network proximity, bandwidth availability, client perceived latency, distance, surrogate load, network load, and content availability. We use a DNS-based request routing mechanism that replies with candidate IP addresses according to one of the following strategies: Network Proximity, Number of Requests, Average Bandwidth, Shortest Distance and Round Robin. Network Proximity calculates latencies between surrogates and clients. Those surrogates that offer smaller latencies will be selected as the nearest ones. The Number of Requests metric identifies which surrogates are overloaded in terms of requests and ignores their distance to clients in an attempt to restore load balancing. The Average Bandwidth metric calculates the average bandwidth between clients and candidate surrogates. Those surrogates that score higher bandwidth than others are selected by the redirector as the nearest ones. Shortest Distance estimates the distance between a client and certain surrogates through the number of router hops. Hop count has been successfully used as a metric for Internet routing in the past. Finally, the Round Robin request routing strategy distributes requests to the surrogate servers, while balancing load among them.

1.4.2 CDN Interconnection Overlay

This section describes all relevant entities and basic concepts behind the CDNI overlay implemented in simulator. It is seen as the most recent overlay added to the simulator. Under the CDNI context, there are four basic questions. When to peer? How to peer? Whom to peer with? And how to enforce and manage peering policies? In this overlay, we introduce a new entity, the broker that acts as the mediator between two or more CDNs. It handles tasks regarding two of the above questions, “how to enforce and manage policies” and “whom to peer with”. Through the broker, CDNs negotiate cooperation and establish the peering agreement. This is done through a peering request

sent to all known CDNs. The peering request contains information about what the upstream CDN needs, the CDN in need of assistance, etc. This information will be used by each and every broker to decide if its CDN is able to provide assistance for the upstream CDN, if the answer is yes then a budget response is sent to the upstream's broker. The latter collects all budget responses and then decides among the peering options.

A second important new entity, the Clock-Cycle, answers when the CDN should trigger a peering request. The decision is based on requests and network characteristics. Two basic scenarios are covered, the geographic absence and surrogate utilization. The geographic absence covers a situation where the CDN is receiving a significant amount of requests from a location without direct coverage. Surrogate utilization represents a scenario where one or more surrogates are overloaded.

The question "How to peer?" is not directly mapped to an entity. Peering agreements are handled by redirecting clients from the up-stream to the down-stream request redirectors. CDN peering should not be something totally ad-hoc, but instead based on pre-established peering agreements.

2 Two-level Validation

According to [159], for validation purposes, a simulated model should be compared to another reference model. Therefore, we divided the validation process into two parts. First, we compared our networking protocols, such as the TCP implementation, to that from the known network simulator (ns-3). In the second stage, we built a very simple CDN scenario and compared its performance with that from the public CDN simulation tool, CDNsim [168].

2.1 Packet level validation

To validate our TCP implementation, we ran experiments with the same topology and characteristics using our simulator and the proven packet level ns-3 [170]. Two different topologies were considered. The first one is inspired by ns-3's fifth.cc release example while using 10Mbps and 1ms links. The second tested topology was more general, as it was generated using the BRITE topology generation tool. It consists of 4

AS with 5 nodes in each AS. Then, all topology variables were set to their default BRITE values.

In ns-3, a TCP-large-transfer example was used, while accounting for our topology and changing the object size. In the CDNP2PSim case, a Simple-Overlay with a simple client-server application was created.

The following four metrics are observed: simulation time, download rate, download time and memory usage. The download rate represents, in the case of ns-3, the amount of all the data passing through the links during the download time, whereas in P2PCDNSim download rate reflects all the useful data that passed through such links during download time. Our simulator uses a great deal more memory, as it implements both packet and CDN levels, see Figure I.5.

Figure I.5 – Memory usage comparison between ns-3 and P2PCDNSim

First, we used the example from the ns-3 tutorial folder (tfifth.cc). The objective of this experiment was to analyze the congestion window behavior. Figure I.6 shows how the congestion window changes during the simulations for both tools (CDNP2PSim and the ns-3). It is well known that ns-3 uses the New Reno TCP implementation, which differs from the RFC5681 standard used on our TCP implementation. The router's buffer was set to 128KB, which is the same default send buffer limit used in the ns-3. As we can see, the behavior of the congestion window was as expected.

Figure I.6 – Congestion window behavior comparing ns-3 and P2PCDNSim

Table I.1 shows that download time and rate are very similar and are both within the acceptable levels for the topology used. The main difference for this scenario is seen in terms of simulation time. P2PCDNSim needs three times more time than the ns-3.

Table I.1 – Simulation time, download time and rate for ns-3 and P2PCDNSim basic 3-node scenario

	ns-3	P2PCDNSim
Simulation Time	3 m 6,799 s	9 m 39,232 s
Download Time	188,95	185,00412866276577
Download Rate	1,240875 MB/s	1,081056640625 MB/s

We believe that the difference is explained in the simplicity of the scenario, along with the consideration that the ns-3 is programmed in C++, which is known to be faster than the interpreted language Java, used in our simulator. The same justification applies to the understanding of Figure I.5, where ns-3 uses much less memory than P2PCDNSim. This first analysis shows the total virtual memory used for each process. The next experiment should provide more insight on the memory usage of the simulators.

The second scenario carries out the same transfer while changing the size of the object to 200MB, the time that the application will be running during the simulation, and the second topology (the second topology described above). Figure I.6 shows memory usage for both simulators considering virtual and residual memory. Comparing Figure I.5 with Figure I.6, one can see that a ten-fold topology increase did not considerably affect memory consumption. From our previous experience with P2PCDNSim we believe that

the main aspect responsible for memory consumption is the number of clients in the network, the size and the quantity of objects being requested.

Figure I.7 – Memory usage consumption for 20 nodes topology scenario

Table I.2 shows the download time, simulation time and download rate for both ns-3 and our simulator. These metrics exhibit very close results. When comparing the simulation time through, the ns-3 needs considerably much more time to run the same scenario than P2PCDNSim. By conducting further experiments, it was realized that the reason for such larger simulation time difference was the amount of data written into the ns-3 traces. By disabling the pcap trace next, used along with the Wireshark Ethernet sniffing software for trace analysis, the simulation time dropped to 21 minutes and dropped even further to 6 minutes when no tracing was required. Considering that a simulation without any trace or record is useless, it is fair to state that the minimal practical ns-3 simulation time was almost twice P2PCDNSim simulation time for the same exact topology and application.

Table I.2 – Simulation time, download time and rate for ns-3 and P2PCDNSim basic 20-node scenario

	ns-3	P2PCDNSim
Simulation Time	51 m 33,423 s	11 m 12,173 s
Download Time	188,95	185,00412866276577
Download Rate	1,240875 MB/s	1,081056640625 MB/s

2.2 Overlay level validation

To validate our CDN Overlay implementation, we replicated the experiments provided from the publically available CDNsim tutorial onto our own simulator. We

followed instructions provided in CDNsim's tutorial to prepare the scenarios. The first step is to set the redirection policy. We set it to non-cooperative with closest origin, since it is the simplest policy option (with the exception of random that would not be useful to compare). Table I.3 summarizes the scenarios' configuration.

Table I.3 – Scenarios configuration

	Scenario1	Scenario2
# nodes	50	3037
# surrogates	1,2,3,10	10
Trace	Provided by CDNsim tutorial: trace_file50000 (first 5000 lines)	Provided by CDNsim tutorial: trace_file50000
Link Capacity	100 Mbit/s	100 Mbit/s

2.2.1 CDNP2PSim Overlay Setup

The next phase was the replication of all scenarios into our simulator. We started by converting the input files to the format used in our P2PCDNSim. Using the topology configuration, the P2PCDNSim can automatically create, at the start of simulation, a router table based on the shortest path, in number of hops. Alternatively, the user can provide one. We set the routing to automatic. There is the possibility of setting different link capacities to the clients' links, these may even be asymmetric. However, this was not done to maintain congruence between our program and CDNsim.

To keep the two environments as close as possible, we turned off several additional features of our simulator. For instance, in the objects definition file of our simulator, one may provide more information than simply the object's size, such as the bitrate necessary for the video's distribution. Our simulator also offers the option of selecting from different placement strategies such as Greedy or Hotspot, whereas the CDNsim tutorial states that those entities are randomly placed. CDNsim also randomly attaches clients to nodes. We used the Round Robin strategy with equal and symmetric links, also supported by CDNsim.

The common collected metric is limited to the cache hit as shown in Table I.4. One can see that the cache hit ratios are very similar for both tools. The small difference, seen in the one surrogate scenario, can be attributed to differences on the simulation of the

network (delay of links, TCP implementation etc.). It can also be attributed to the CDNsim random client and surrogate attachment to nodes.

Table I.4 – Simulation time, download time and rate for ns-3 and P2PCDNSim basic 20-node scenario

#surrogate	CDNsim Scenario1	P2PCDN Sim Scenario1	CDNsim Scenario2	P2PCDN Sim Scenario2
1	24%	20.44%	-	-
2	14.4%	12.78%	-	-
3	10.5%	10.26%	-	-
10	3.72%	3.78%	30.71%	27.52%

Appendix II

Surrogate placement strategies

In scenarios we conducted these experiments, we used an AS network topology with 3037 nodes and links of 100 Mbit/s. We placed the origin server on center node, according to min-K center placement. To generate the requests, we employed the network traces used in [202], where authors studied the behavior of YouTube traffic. The network traces were collected in Massachusetts Institute of Technology (MIT) through monitoring traffic at the interconnection between this university campus access network and the Internet. The definition of clients and video objects were parsed from the same set of Youtube trace files. We used the non-cooperative pull-based outsourcing strategy. The number of requests, clients and objects depends on the trace used, as shown in Table II.1.

Table II.1 Trace configuration

Trace	Clients	Requests	Objects
A	2031	27434	18715
B	2377	18750	13501
C	734	12838	9339
D	989	14785	10402
E	2447	32367	23240
F	16337	611968	263970

Each placement algorithm was calculated considering the surrogate servers initially containing all objects that would be requested in order to isolate the placement strategy from the influence of caching strategy. Figure II.1 shows the performance for evaluated algorithms.

Figure II.1 – Evaluation of surrogate placement strategies

Except for Random, we observe that the bandwidth consumption always decreases as the number of surrogates increases. It shows that the requests are being distributed to the closest surrogates. We show the evaluation of three variations of Hot-Spot algorithm, with a radius of 0, 1 and 2. It can be seen that the Hotspot with radius equals to 1 has the best performance for our scenario, followed by Hotspot with radius 2 and Greedy. Note that the Hotspot algorithm is based on vicinity, which is defined as the circle centered on a node with some radius. In our scenarios, the Hotspot algorithm achieves its best performance with radius equals to 1. It is expected since increasing the radius put the calculation of workload under the influence of nearby loads (which may be already supplied by a surrogate), thus interfering in the process of identifying the greatest loads.