Programme ANR VERSO

Projet VIPEER

Ingénierie du trafic vidéo en intra domaine basée sur les paradigmes du Pair à Pair

Décision n° 2009 VERSO 014 01 à 06

du 22 décembre 2009

T0 administratif = 15 Novembre 2009

T0 technique = 1er Janvier 2010

Livrable 4.2

Preliminary report on the CDN/dCDN design

Auteurs :

C. Bothorel (Telecom Bretagne), Z. Li (Telecom Bretagne), G. Simon (Telecom Bretagne), F. Albanese (Eurocom), P. Michiardi (Eurocom), J. Garnier (NDS Technologies France)

Compilé par :

J. Garnier (NDS Technologies France), A. Gravey (Telecom Bretagne)

Juillet 2011

Telecom Bretagne; Eurecom; INRIA; France Telecom; NDS Technologies France; ENVIVIO

Résumé: *(15 lignes)*

The main objective of the VIPEER project is to propose novel mechanisms to integrate traffic engineering (i.e. guiding the traffic where resources are available) in the current video streaming delivery system, so as to ameliorate the client QoE. VIPEER builds upon the collaboration between a traditional CDN and a peer-assisted CDN or "distributed CDN" (dCDN), i.e. an overlay controlled by the network operator using P2P paradigms. This document presents two approaches for coordinating a classical distribution via a CDN with distribution architectures implemented within the ISP.

The first approach builds upon P2P aided content distribution, whereas the second builds upon CCN based content distribution. The efficiency of both approaches is assessed by models, simulations and experimentations. Their efficiency is shown to depend on the popularity of distributed contents.

The reported studies show that both approaches may benefit from content-oriented caching strategies, and in particular "pre-fetching" which consists in replicating content in the dCDN nodes before they are requested based on their popularity. In order to implement prefetching capabilities, we need to analyze users' behavior. The last part of the deliverable describes datasets we plan to use to analyse prefetching techniques.

Keywords: **Content Delivery Network, P2P, Content Centric Network, Peer Assisted**

Table des matières

1 Preface 7
1.1 Purpose of this document 7
1.2 Referenced VIPEER deliverables 7
1.3 List of Acronyms 7
2 The CYCLOPS approach 9
2.1 Motivations 9
2.2 Cost-Performance Tradeoff Model 11
2.2.1 Model 12
2.2.2 Considerations 14
2.3 System Design and Implementation 15
2.3.1 Overview of CYCLOPS 15
2.3.2 The CYCLOPS Swarm Monitor 16
2.3.3 The CYCLOPS Content Server 16
2.4 Experimental Method and Setup 17
2.5 Experimental Results 19
2.5.1 Flash Crowd Experiments 19
2.5.2 Waves of Arrivals Experiments 20
2.5.3 Live Internet Experiments 22
2.6 Additional Considerations 24
2.7 Related work 26
2.8 Conclusion 28
3 CCN with Cooperative Caching 29
3.1 Introduction and Background 29
3.1.1 Context: Content Centric Networking 29
3.1.2 Our Focus: ISP-friendly Time-shifted Streaming 30
3.1.3 Our Proposal: Cooperative In-Network Caching 30
3.1.4 Our Contributions: Algorithms and CCN Protocol 31
3.2 Network Model 32
3.3 Initialization Stage 32
3.4 Distributed Algorithm 33
3.5 Augmented CCR Protocol 35
3.5.1 CCN in a Nutshell 35
3.5.2 New Tables in CCN 35
3.5.3 Distribute Chunks in the Cooperative Cache 35
3.5.4 CCS Consistency 36
3.6 Analysis of Cooperative Cache 37
3.7 Experimental Results 40
3.7.1 Simulations on Time-shifted TV 40
3.7.2 Simulation Setup 40
3.7.3 Results Analysis 41

3.7.4 Simulations on VoD service 44
3.8 Conclusion 46
4 Real Datasets for prefetching and simulation 47
4.1 A real VoD dataset 47
4.2 A dataset for prefetching 51
4.3 Merging the datasets 51
4.3.1 Popularity classification 51
4.3.2 Merging method and resulting dataset 52
4.4 Conclusion 54
References 55

Table des figures

Figure 1 : Mean download time as a function of the server rate (N=100, M=2000, μ=0.5).... 14
Figure 2 : Overview of Cyclops Architecture: The content server and swarm monitor reside in the cloud in distinct virtual machines, with off-cloud bandwidth used for data feed (to the swarm) and control feed (from the swarm). 16
Figure 3 : Flash Crowd: content download times (file size: 50MB). 19
Figure 4 : Waves of Arrivals: content download times (file size: 50MB). 21
Figure 5 : Waves of arrivals: availability over time. 22
Figure 6 : Live Experiment: Evolution of swarm size over time. 23
Figure 7 : Live Experiment: content download times (file size: 357.5 MB). 24
Figure 8 : Example of cooperative cache 31
Figure 9 : Individual caches connected to a server 37
Figure 10 : Cooperative caches connected to a server 37
Figure 11 : Caches in tandem 39
Figure 12 : *Caching diversity*: the number of distinct chunks stored in the set of CRs when the number of labels *k* varies 41
Figure 13 : *ISP-friendliness*: the number of times each server located is accessed. The smaller is the bar, the more ISP-friendly is the caching strategy 42
Figure 14 : Cumulative Distribution Function. The y axis is the ration of chunks; the x axis is the time elapsed between two consecutive accesses on a CR. 43
Figure 15 : Caching diversity varies with *k* 45
Figure 16 : Chunk distribution of the 10 films 45
Figure 17 : Number of times each server is accessed 46
Figure 18 : ISP VoD downloading logs: timestamp, user ID, film ID 48
Figure 19 : Downloads during the week-end or the week. 48
Figure 20 : Downloads each different day. 49
Figure 21 : The “Top” users who download the most. 49
Figure 22 : Downloading profile of users who downloaded only once during the 6 weeks of $Data_{ISP}$. 50
Figure 23 : Comparison between the downloading profiles. 50
Figure 24 : Power distribution of ratings in $Data_{WEB}$. 52
Figure 25 : $Data_{ISP}$ types of films: popular films and films downloaded once. The “Mid-popular” films are the delta between the global red line and the blue one. 52
Figure 26 : Popularity merging between ISP downloaded films and Web annotated films. ... 53
Figure 27 : Two real datasets and our generated dataset. 53
Figure 28 : Weekday and week-end downloads in the generated dataset. 54

Liste des Tables

Table 1 : Referenced VIPEER deliverables 7
Table 2 : Flash Crowd: average server load (file size: 50MB) 20
Table 3 : Waves of Arrivals: server load & overhead (file size: 50MB) 21
Table 4 : Live Experiment: service statistics (file size: 357.5 MB) 24
Table 5 : Comparison of Response Time and Requested Time Interval 43

1 Preface

1.1 Purpose of this document

This document aims to present the approach for the design of the dCDN. Based on the state of the state of the art, we have identified two differents approachs for the design:

- The first approach, CYCLOPS, builds upon P2P aided content distribution. With CYCLOPS, the CDN server is able to adjust the CDN-to-ISP bandwidth utilization so as to achieve a specific *objective* based on a *feedback signal* related to the performance of content distribution.
- The second approach builds upon CCN based content distribution. This approach is called CCN+: cooperative caching policy based on CCN Paradigm. CCN+ relies on the storage capacity of a set nodes within the ISP to minimize the amount of queries for video streams to be treated by (CDN) servers outside the ISP network.

The reported studies show that both approaches may benefit from content-oriented caching strategies, and in particular "pre-fetching" which consists in replicating content in the dCDN nodes before they are requested, based on their popularity. For CYCLOPS, it is necessary to understand how to feed the swarm and which contents to choose. For CCN+, we think that hybrid caching policies combining passive caching policies with prefetching strategies would improve performance. In order to implement prefetching capabilities, we need to analyze users' behavior. The last part of the deliverable describes datasets we plan to use to analyse prefetching techniques.

1.2 Referenced VIPEER deliverables

Table 1 lists documents and other reference sources containing information that may be essential to understanding topics in this document.

Table 1 : Referenced VIPEER deliverables

No.	Designation	Title
1.	D4-1	State of the Art

1.3 List of Acronyms

Term	Definition
AS	Autonomous Systems
AMI	Amazon Machine Image
AWS	Amazon Web Services

Term	Definition
BT	BitTorrent
CCN	Content Centric Network
CCS	Collaborative Content Store
CDN	Content Delivery Network
CR	Content/Caching Router
CRT	Collaborative Router Table
CS	Content Store
dCDN	Distributed CDN
DoS	Denail of Service attack
Ebone	European Backbone
EC2	Elastic Computing Cloud
ECDF	Empirical Cumulative Distribution Function
FIB	Forwarding Information Base
ICT	Information and Communication Technologies
ISP	Internet Service Provider
LAP	Label Allocation Process
LFU	Least Frequently Used
LRU	Least Recently Used
OF	On-line Feedback
PIT	Pending Interest Table
POP	Point Of Presence
QoE	Quality of Experience
S3	Simple Storage Service
STB	Set Top Box
VoD	Video on Demand

2 The CYCLOPS approach

In the following sections we will describe a framework, called "CYCLOPS", that is intended to perform content distribution focusing on the resource utilization of content servers.

In the context of the VIPEER project, we assume the distributed CDN (dCDN) to play the role of a P2P swarm interested in a particular content for download. Specifically, the CDN interacts with the dCDN by managing the trade-off between CDN-to-ISP bandwidth utilization and performance of content delivery. We present here a framework in which the CDN server is able to adjust the CDN-to-ISP bandwidth utilization so as to achieve a specific *objective* based on a *feedback signal* related to the performance of content distribution. Our framework is general enough to allow for many possible combinations of objectives and feedback signals. For instance, the objective may simply be to keep dCDN alive based on a feedback signal indicating the level of redundancy for particular pieces of content in the dCDN. Alternately, the objective may be to ensure a desirable level of service based on a feedback signal gauging average delivery time to clients.

At the moment, we address the simpler case of the bulk transfer of data, not streaming. However, CYCLOPS offers space of maneuver for the adaptation of streaming strategies.

As discussed in the description of work of the project, the content servers inject content in the ISP network toward clients that consume it; the servers location can be either within the ISP network - as part of the dCDN - or can be external, e.g., in the case of servers operated by a third-party CDN.

The mechanism discussed in the following takes up the problem of gauging the resource utilization - namely bandwidth - of content servers: our approach proves useful in both cases of server location:

- In the case of a content server within the ISP, CYCLOPS reduces the resource utilization, offloading the content distribution to already available, yet unexploited, network components like the set-top-boxes (STB).
- In the case of a content server of an external CDN, our approach reduces the costs that the ISP incurs to use its transit links to fetch the content.

The remainder of the document is organized as follows: first we provide the motivations for our work, geared toward Cloud-Based content distribution. We then move to provide the intuitions, which stem from a mathematical model of content distribution, behind our scheme. Finally, we describe CYCLOPS in details and develop a methodology for an experimental evaluation of our mechanism.

2.1 Motivations

Cloud computing has emerged as a compelling paradigm for deploying Information and Communication Technology (ICT) solutions on the Internet, because it enables solution

providers to easily scale up or down, or migrate their offerings seamlessly across resources – compute servers, storage, platforms, and services – offered by one or more cloud providers, yielding significant cost savings due to economies of scale. More importantly, the elasticity of the "pay-as-you-go" paradigm enables solution providers to reign in operating costs, especially when demand is highly dynamic, or unpredictable. For many cloud-based ICT solutions, gauging demand is straightforward. For instance, a cloud-based web hosting/caching solution can easily gauge demand – and hence scale up or down its use of elastic cloud resources – by observing the number (or average response time) of its web transactions.

Increasingly, however, cloud-based solutions are evolving from simple *client-to-cloud* interactions (reminiscent of the traditional client-server model) into *swarm-to-cloud* interactions, wherein the cloud-based solution is not merely responding to individual client requests, but rather to the collective demand of a "swarm" of clients, making the determination of what constitutes demand for cloud resources for purposes of elastic resource allocation far more complicated. In the following sections, we propose a general framework and present a prototype implementation that enable elasticity for a canonical "swarm-to-cloud" application – namely peer-assisted content delivery.

Towards Elastic Cloud-Based, Peer-Assisted CDNs: Traditional Content Delivery Networks (CDNs) such as Akamai [1] were conceived as special-purpose clouds catering almost exclusively to large, highly-popular content providers such as iTunes and CNN. Today, the advent of cloud-based storage and delivery solutions such as Amazon S3 [2] and CloudFront [2] make it possible for much smaller-scale content providers to deploy and elastically provision their own cloud-based CDNs in an almost real-time fashion. The major cost contributor for such cloud-based CDNs is *off-cloud bandwidth*: the bandwidth consumed to deliver content from the CDN content servers (in the cloud) to the CDN clients (off the cloud). To reduce off-cloud bandwidth, an increasing number of CDN solutions (including those offered by major market players such as Akamai [1], Limelight [3], and Amazon [2]) rely on swarm-based, peer-assisted approaches that leverage the uplink capacity of end-users to reduce off-cloud bandwidth consumption. This approach, which is particularly effective for highly popular content, can be seen as seamlessly bridging client-to-cloud and swarm-to-cloud interactions: For less-popular content, a cloud-based, peer-assisted CDN behaves as a traditional (client-server) CDN system, whereas for high-popular content, it behaves as a peer-to-peer system.

Existing cloud-based peer-assisted CDNs rely on swarm-based protocols such as BitTorrent [4]. While such protocols are quite efficient for exchanging content among peers (in terms of download time, resource utilization, and fairness), they are not designed to provide the content source with the means to gauge the marginal utility of its contribution to the swarm. Specifically, in our cloud-based peer-assisted CDN setting, swarm-based protocols do not enable the content server (in the cloud) to gauge and hence manage the inherent tradeoffs between off-cloud bandwidth utilization and the efficiency of content delivery. This is precisely the capability that the work presented in this document aims to provide.

Work Scope and Contributions: We present a novel framework for cloud-based peer-assisted CDN solutions in which the content server (inside the cloud) is able to adjust the off-cloud bandwidth it contributes to the swarm (the set of clients outside the cloud) so as to achieve a specific *objective* based on a *feedback signal* related to the state of the swarm. Our framework is general enough to allow for many possible combinations of objectives and feedback signals. For instance, the objective may simply be to keep the swarm alive based on a feedback signal indicating the level of redundancy for particular pieces of content in the swarm. Alternately, the objective may be to ensure a desirable level of service based on a feedback signal gauging average delivery time to clients.

To establish a reference model for these as well as other combinations of objectives and feedback signals, we develop in Section 2.2 an analytical model that quantifies the cost-performance tradeoff for cloud-based, peer-assisted content delivery. Our model relates off-cloud bandwidth utilization (the cost incurred by the provider) to the average delivery time (the performance observed by clients). Along these lines, our findings suggest the existence of a quiescent (close to optimal) operating point beyond which the marginal utility from additional off-cloud bandwidth utilization is negligible.

Armed with this understanding, in Section 2.3, we present the design and prototype implementation of CYCLOPS, a peer-assisted content delivery cloud service. The content server in CYCLOPS is able to modulate its bandwidth contribution to the swarm so as to remain in the vicinity of the aforementioned quiescent operating point – thus minimizing its cost without sacrificing performance. Our design relies on the feedback signal provided through an on-line monitoring tool, which we have implemented as part of CYCLOPS.

To demonstrate the effectiveness of our approach, in Sections 2.4 and 2.5 we report on a fairly extensive series of Internet experiments, in which we compare the performance of CYCLOPS to those of "open-loop" swarm-based protocols used by cloud-based content delivery services. Our experiments are carried out both in a controlled environment (by delivering content to PlanetLab clients) and in the wild (by delivering content to a real Internet user population). These experiments show that our feedback-based approach reduces drastically the volume of data served from the cloud (and hence the cost incurred by the content provider) with negligible performance degradation. More to the point, in live experiments involving more than 10,000 users exhibiting highly dynamic arrival and departure patterns, we were able to document monetary savings of up to two orders of magnitudes for our system.

2.2 Cost-Performance Tradeoff Model

In this Section we develop a model that relates off-cloud bandwidth utilization by a content server in the cloud to the average delivery time perceived by a set of swarming users (clients) outside the cloud.

2.2.1 Model

We consider a dynamic environment, where clients join a swarm, download the content, and eventually leave the system. The number of clients in the swarm is not known *a priori*, but it can be characterized by arrival and departure rates. These rates may fluctuate drastically and such fluctuations are typical for "hot" viral Internet content, which gets published, gains significant popularity fairly quickly, but eventually dies off over time. In this work, we assume that for the content download timescale (say minutes) they remain constant, allowing the system to reach a steady state in which the arrival and departure rates equalize, and consequently the average number of clients in the swarm is constant.

Let N be the steady state average number of clients in the swarm, and let the content be divided into M independent pieces. If $M>>1$ then a client holds $M/2$ pieces on average. For analytical tractability, we do not model network bottlenecks or losses.

Consider a *birth-death Markov chain* whose state represents k, the number of replicas of a single (arbitrary) piece of content. Note that one can envision an identical, independently evolving Markov chain for each one of the M pieces that make up the content. For a generic state, there are two possible transitions: (1) either the piece is replicated, resulting in a *piece birth*, and thus a transition from s_k state to state s_{k+1}, or (2) a client holding a replica of the piece leaves the swarm and is replaced by a new client that does not have the piece, resulting in a *piece death*, and thus a transition from state s_k to state s_{k-1}.

Let α_k indicate the *average* rate at which the content server injects a piece in the swarm at state s_k. Let λ denote the piece replenishment rate resulting from client contributions: λ is computed by dividing the aggregate upload capacity of all N clients by the total number of pieces M. Both α_k and λ are expressed in pieces per second.

For sake of simplicity, we assume a *random* piece replication strategy: in contrast to more sophisticated replication strategies [5], random piece selection simplifies analysis and provides conservative performance bounds. Thus, the probability of choosing to replicate the particular piece (modeled by the Markov chain) out of the $M/2$ pieces available at the client is $2/M$. The probability that no client will choose to replicate that piece is $(1-2/M)^k$, since k is the number of clients holding the piece in state. This yields a probability of $1-(1-2/M)^k$ for going from state s_k to state s_{k+1}.

To compute the transition rate from state s_k to state s_{k+1} we must also account for the rate α_k at which the content server *independently* injects the piece into the swarm. This yields a transition rate of $\lambda(1-(1-2/M)^k)+\alpha_k$. Notice that state s_0 is a special state in which only the content server can inject the piece. Thus, the transition rate from state s_0 to state s_1 is equal to the server upload rate α_0.

Let μ denote the client departure rate (measured in clients per second). The probability of a death out of state s_k is the probability that any one of the k clients holding the piece leaves the swarm. The probability that a given departure is by one of these k users is k/N. Thus, the transition rate from state s_k to state s_{k-1} is given by $\mu k/N$.

In summary, the transition rates from state s_k to state $s_{k'}$, denoted by $s_{k,k'}$, can be expressed as follows:

$$s_{k,k'} = \begin{cases} \alpha_0 & if k = 0 \wedge k' = 1 \\ \lambda \cdot \left(1 - \left(1 - 2/M\right)^k\right) + \alpha_k & if k' = k+1,\ 0 < k < N \\ \mu k / N & if k' = k-1,\ 0 < k \leq N \\ 0 & otherwise \end{cases} \quad (1)$$

Note that, since the Markov chain is finite (N+1 states), the steady state solution exists.

We now compute the probability π_0 to be in state s_0. For simplicity, we consider the case in which the content server uploads a piece at an average rate $\alpha_k = \alpha$, $\forall k$, irrespectively of its state; by solving the Markov chain we get:

$$\pi_0 = \left[1 + \frac{\alpha}{\mu} N \left(1 + \Phi\right)\right]^{-1} \quad (2)$$

Where

$$\Phi = \sum_{k=2}^{N} \left(\frac{N}{\mu}\right)^{k-1} \frac{1}{k!} \prod_{i=1}^{k-1} \left[\lambda\left(1 - \left(1 - \frac{2}{M}\right)^i\right) + \alpha\right]$$

We now proceed to finding the relationship between the average server rate α and the mean download time. Each client obtains $1/N$ of the swarm's upload capacity, which is $M(\lambda+\alpha)$. Since the content is composed of M pieces, the mean download time can be computed as $T=M/(M(\lambda+\alpha)/N)=N/(\lambda+\alpha)$. This is true as long as the probability of being in state is small enough. If this probability increases, then we have an additional term for the mean time spent in state s_0: this can be computed by multiplying the probability of state s_0 (π_0) by the time spent in state s_0 ($1/\alpha$). Hence, the mean download time is bounded by:

$$T \leq \frac{N}{\lambda + \alpha} + \frac{\pi_0}{\alpha} \quad (3)$$

To illustrate the utility of this model, consider a swarm of N=100 clients downloading content consisting of M=2000 pieces, with a client departure rate of μ=0.5 clients per second, and a mean client upload rate of λ={1.5μ,3μ,6μ} pieces per second. Figure 1 shows the average download time as a function of the server upload rate, as predicated by Equation 3.

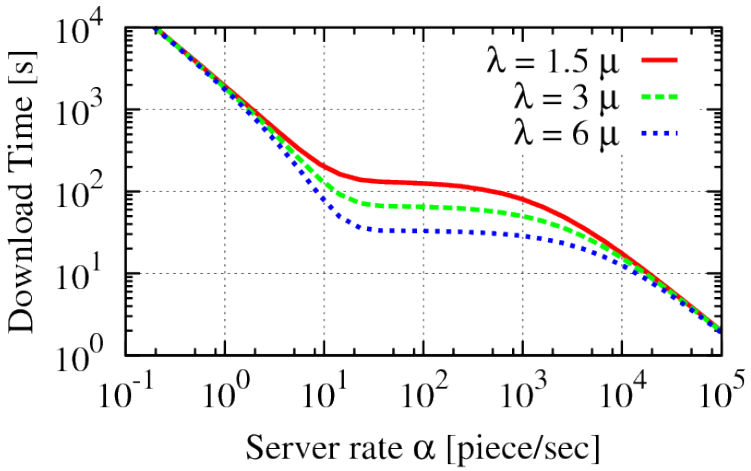


Figure 1 : Mean download time as a function of the server rate (N=100, M=2000, μ=0.5).

Figure 1 quantifies the tradeoff between the off-cloud bandwidth utilization (*i.e.*, the average upload rate α of the content server) and the average delivery rate to clients involved in a swarm with upload capacity λ. It shows three operating regions. The first operating region (left-side of the plot) is when α tends to zero, resulting in piece starvation, and a corresponding increase in download time. The second operating region (right-side of the plot) is when α tends to values that far exceed λ, resulting in a client-server-like mode of operation. The third and more interesting operating region is an intermediate one, within which an increase in α does not result in a corresponding decrease in download time. The "width" of this region depends on the health of the swarm, which is a function of the content popularity captured by the client arrival/departure rate μ, and the mean client upload bandwidth λ. For the particular settings used in Figure 1, this intermediate region is given by $\alpha \in [10,1000]$ piece/sec.

The behavior described by our model suggests the existence of a quiescent operating point (at the transition between the first and second operating regions depicted in Figure 1), beyond which the marginal utility from additional off-cloud bandwidth utilization is negligible. A content server operating around this quiescent point would be fully leveraging the uplink bandwidth of its clients, while minimizing its own cost: *operating below this quiescent point would jeopardize performance, and operating above this quiescent point would be cost inefficient.*

2.2.2 Considerations

Armed with this observation, we are now ready to describe the design and prototype implementation of a content server that uses a feedback signal to adjust its bandwidth contribution to the swarm so as to remain in the vicinity of a nominal quiescent operating

point. It is important to notice that the prototype we present in the following is *not* an explicit implementation of the model: the design is inspired by the key observations made above.

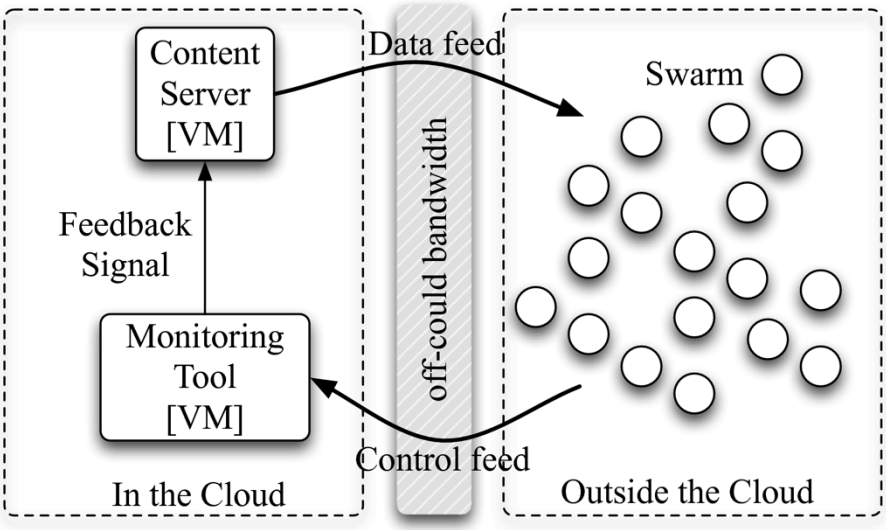
While our framework allows for many combinations of objectives and feedback signals, in the remainder of this work we focus on the objective of maximizing the performance per unit cost, using the availability of content in the swarm as the feedback signal.

2.3 System Design and Implementation

We now present the design of CYCLOPS, our cloud-based peer-assisted content delivery service.[1]

2.3.1 Overview of CYCLOPS

As depicted in Figure 2, our CYCLOPS service consists of a *content server* and a *swarm monitor*, both residing in the cloud. The swarm monitor interprets the signaling messages exchanged between swarming clients, and generates a feedback signal that enables the content server to gauge the marginal utility of its contribution to the swarm. The content server participates in the swarming protocol to satisfy client requests, but only *feeds* the swarm when its contribution is deemed necessary (based on the feedback signal). In CYCLOPS, the swarm feeding rate is set to maximize the swarm performance-per-unit-cost, using the availability of content in the swarm as the feedback signal. The model given in Section 2.2 shows that the quiescent operating point for this objective is the minimum rate that avoids swarm starvation.



1 Our CYCLOPS service can be seen as injecting bursts of content into a swarm of clients, just as in Greek mythology the primordial one-eyed giant Cyclopes were the source of Zeus'

Figure 2 : Overview of Cyclops Architecture: The content server and swarm monitor reside in the cloud in distinct virtual machines, with off-cloud bandwidth used for data feed (to the swarm) and control feed (from the swarm).

CYCLOPS is conceived to work with *any* swarm-based application/protocol that features (1) a coordinating entity that tracks all swarm participants, enabling them to establish peer-to-peer connections; (2) content that is divided into pieces to be distributed / exchanged independently; and (3) a control messaging scheme used by swarm participants to advertise piece availability.

For practical reasons, we present our system and conduct our experiments focusing on a single content server, used to deliver a single content (file) to a set of clients. Problems related to concurrent swarms are orthogonal to our approach, and the solutions proposed in the literature, *e.g.*, [6], can be integrated independently. Similarly, issues related to the efficiency of the distribution process, solved using approaches based on traffic locality, are complementary to our solution, and previous work on this topic, *e.g.*, [7, 8], can be incorporated seamlessly.

CYCLOPS was conceived and implemented as a cloud service that can be deployed on existing cloud platforms. Specifically, we focused on the Amazon Web Services (AWS) environment, and produced an Amazon Machine Image (AMI) that supports *both* the content server and the swarm monitor functionalities. We have released [9] to the research community the CYCLOPS AMI, along with set-up and configuration instructions.

2.3.2 The CYCLOPS Swarm Monitor

Swarm monitoring in CYCLOPS is achieved using a set of components residing in the cloud, called the On-line Feedback (OF) nodes. OF nodes connect to a live swarm, but neither download nor upload content: they monitor *all* clients in the swarm and collect signaling messages they exchange. Using this information, OF nodes construct snapshots in time that characterize the health / performance of the swarm. In our particular implementation, these snapshots are used to derive the instantaneous piece availability, which constitutes the feedback signal fed to the CYCLOPS content server using a complementary protocol.

To ensure scalability (and seamless elasticity), we adopted a distributed design for OF nodes, whereby new clients joining the swarm are assigned to OF node to balance load. Accordingly, a swarm S is partitioned into N_p non-overlapping sets, where N_p is the number of OF nodes in the system. Swarm partitioning is achieved using consistent hashing [10]: each OF node is responsible for a fraction of the key-space, defined by the client ID (*e.g.*, IP address).

2.3.3 The CYCLOPS Content Server

The main objective of the content server is to minimize off-cloud bandwidth consumption without running the risk of starving the swarm. Based on the feedback signal provided by the swarm monitor, the content server feeds the swarm only when necessary, *i.e.*, when piece

availability falls below a desirable threshold. To that end, in our design we adopted an ON/OFF control strategy, whereby the content server operation oscillates between two states: *serving* and *idle*.

When in the *serving state*, the content server dedicates its full uplink capacity to serve missing pieces of content. By design, the server avoids injecting duplicate pieces into the swarm. The rationale for doing so is that the swarm participants themselves can quickly replicate pieces. All clients connected to the content server are induced to request the set of missing pieces, which constitute the *serving set* maintained by the content server: this is possible since the server masquerades as a set of virtual clients holding a fraction of all available pieces. This serving set is partitioned into k non-overlapping subsets that are announced as "available." For instance, if the serving set consists of pieces {1,2,3,4} and k=2, then k messages each announcing pieces {1,2} and {3,4}, respectively, will be sent to k users that will eventually issue download requests. Once a piece has been served, it is removed from the serving set, provided that the swarm monitor has confirmed the presence of the piece in the swarm. When the server has finished injecting all missing pieces into the swarm, it transitions to the *idle state*.

When in the *idle state*, the content server simply closes all connections to remote clients, and refuses any incoming connection. The content server remains in the idle state until the feedback signal triggers a transition to the *serving state*.

2.4 Experimental Method and Setup

In this section, we summarize the specifics of the CYCLOPS instance we have experimented with, along with various details regarding deployment on a commercial cloud. We also describe the three types of experiments we have conducted: two were in a controlled environment (involving PlanetLab clients under our control), and the third was in the wild (involving thousands of real Internet users accessing content we advertised and made available).

BitTorrent-based Swarming: As we alluded to in Section 2.3, CYCLOPS can be instantiated to work with any swarm-based content distribution protocol, supporting a specific set of features. For experimental purposes, we created an instance of CYCLOPS that is compatible with the popular BitTorrent (BT) client. Note that, in all our experiments, clients execute unmodified BT code. This choice is partly motivated by the wide adoption of BT by Internet users, as well as its adoption by many cloud-based content delivery services (including Amazon S3 and many others [11]) as an underlying swarming protocol. The details of the BT protocol and algorithms are not essential to understanding CYCLOPS, thus we refer interested readers to [12] for a technical description of BT. Here we only mention that the coordinating entity that maintains the list of clients in the swarm is called the tracker, and that the two control messages used by BT to advertise pieces available at a client are the "have" and the "bit field" messages: they indicate the availability at a client of a specific (single) piece, and of a set of pieces, respectively [12].

In the remainder of this document, we use open-loop-BT to refer to an "open-loop" BitTorrent swarm-assisted content delivery system, whereas we use CYCLOPS to refer to our "feedback-controlled" BitTorrent swarm-assisted content delivery system.

Deployment Details: We used Amazon's Elastic Computing Cloud (EC2) to host, on *separate* virtual machines, the open-loop-BT content server (called the seed) and tracker, and the CYCLOPS content server and swarm monitor. To mitigate the negative impacts on networking performance due to shared resources (CPU and I/O) in a virtualized environment, we used large EC2 instances, which were all located in a *single* US-based data center. Our open-loop-BT and CYCLOPS content servers were well provisioned, with an upload capacity of 2.4 Mbps. Note that, in our experiments, a single OF node proved to be sufficient to monitor the entire swarm fed by CYCLOPS.

Flash Crowd Experiments: To emulate a flash crowd arrival process, we deployed a set of clients on PlanetLab machines, whereby all clients initiate their requests as a result of a centralized trigger: clients start downloading the content within 1 minute of that trigger signal. Once a user is done downloading the content it continues to serve other clients until the end of the experiment. We conducted our experiments using two flash crowd sizes of L=50 and L=300 clients, respectively. In order to minimize the resource utilization of PlanetLab nodes, we used a homogeneous configuration with an application level cap of 160 Kbps for the client's uplink capacity, which is the default setting for BT. The content size was set to 50 MB.

Waves of Arrivals Experiments: We synthesized extreme swarm dynamics on PlanetLab, with the goal of studying CYCLOPS under stress: in practice, we created a scenario in which availability problems would hinder the content distribution process, requiring CYCLOPS to intervene more often than in a real swarm. The dynamics consisted of three successive bursts of client arrivals: a first burst of 100 clients arrive in a 10-minute span and leave after completing their download (within 50 minutes of arrival); a second burst of 100 clients join the swarm just before the mass exodus of the first wave of users. This process is then repeated for a third burst of arrivals. The interval between the mass exodus from one wave and the burst of arrivals from the next wave is set up in such a way that there would not be sufficient time for content pieces to propagate fully from the clients of one wave to the next (which should cause the swarm monitor's feedback signal to trigger the CYCLOPS content server to rev up its contribution to the swarm). As before, the client's uplink capacity was capped at 160 Kbps, and the content size was set to 50 MB.

Live Internet Experiments: We conducted experiments to evaluate our system under realistic CDN operating conditions, including web-driven arrival and departure processes for users drawn from a diverse set of ISPs and with diverse software settings. To do so, we distributed a non-copyrighted movie packed in a 350MB file. We created two distinct *torrent* meta-files (one for distribution using CYCLOPS and the other for distribution using open-loop-BT), and we publicized both simultaneously on popular content search websites, including *isohunt*, *mininova* and *btjunkie*. We took particular care in publicizing the two torrents exactly

on the day of their TV broadcast. In these experiments, both the CYCLOPS and the open-loop-BT content servers had no cap on their uplink capacity (beyond what is possible through a large EC2 instance), and needless to say, we had no control on the settings (or even the BT variants) of the clients.

Performance Metrics: In all of our experiments, we considered two main performance metrics. From the content server perspective, we measured the aggregate volume of data uploaded during an experiment, *i.e.*, the off-cloud bandwidth utilization. Since content servers are under our control, we can measure their bandwidth utilization using local log files. From the client side, we measured the content delivery times. For PlanetLab experiments, we did that by collecting application-level logs from the clients. For live experiments, where we do not have access to client logs, we measured the content delivery times using our swarm monitor, which aggregates information provided by OF nodes. The accuracy of this approach was validated using the PlanetLab experiments: we compared the download times computed using individual log files (of PlanetLab clients) to those obtained from OF nodes, and verified the match between the empirical cumulative distribution functions of download times for the two methodologies. Furthermore, to assert the statistical significance of our results, our PlanetLab experiments were performed five times for each configuration.

2.5 Experimental Results

2.5.1 Flash Crowd Experiments

End-users' performance in downloading content is expressed in terms of individual download times. Figure 3 reports the most important percentiles (25th, 50th and 75th) of the empirical cumulative distribution function (ECDF) of download times.

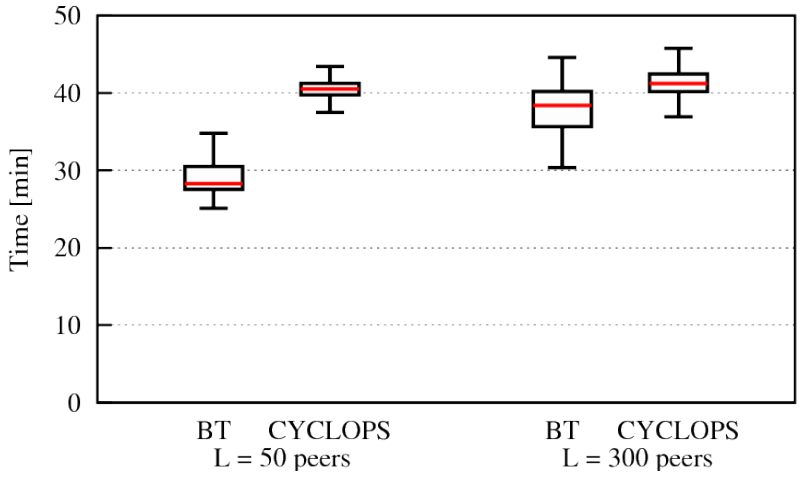


Figure 3 : Flash Crowd: content download times (file size: 50MB).

As a general trend, we observe that the median download time of open-loop-BT swarms is lower than that of CYCLOPS swarms, with the gap reduced in larger swarms. The reason lies in the fact that an open-loop-BT seed keeps feeding the swarm during the whole experiment, resulting in a larger fraction of users receiving data from the content server itself (which is faster than the user), and hence the shorter content delivery time. Furthermore, we note that aside from visible but relatively small variations, the download time for CYCLOPS clients was less sensitive to the swarm size.

Table 2 : Flash Crowd: average server load (file size: 50MB)

	BT	CYCLOPS
L=50	12.2	1
L=300	15.36	1

The above explanation is further confirmed by the results in Table 2, which reports the average off-cloud resource utilization expressed in volume of data served by both the CYCLOPS and the open-loop-BT content servers, normalized by content size. An open-loop-BT seed injects the swarm with 10–15 times the size of the original content, whereas CYCLOPS feeds the swarm only when necessary, which given the static nature of this experiment is once. These results corroborate the intuition discussed in Section 2.2. A content server that can gauge the marginal utility of its contribution to a swarm can settle in the vicinity of an operating point in which an additional expense of off-cloud resources has a marginal effect on the swarm performance.

2.5.2 Waves of Arrivals Experiments

Figure 4 shows the key percentiles of the ECDF for the delivery times experienced by clients in the successive waves of arrivals. In this case, the difference between the delivery times achieved by CYCLOPS and the open-loop-BT content servers is small: the median value of the distribution indicates an advantage of roughly 15% in favor of the latter.

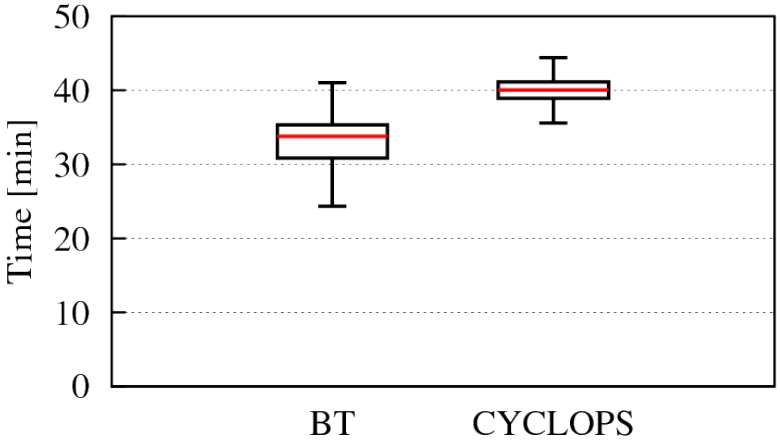


Figure 4 : Waves of Arrivals: content download times (file size: 50MB).

Table 3 shows the average volume of data served by both schemes, as well as information on traffic overhead (namely, volume of control messages involving off-cloud bandwidth resources). For CYCLOPS, we show the aggregate overhead incurred by the content server and the swarm monitor. For completeness, we report the feedback traffic exchanged between the content server and OF node, noting that these messages are exchanged within the confines of the cloud and hence do not entail additional costs.

The data in Table 3 corroborates our conclusion that CYCLOPS achieves low off-cloud resource utilization, even when the system is artificially stressed by complex client dynamics.

Table 3 : Waves of Arrivals: server load & overhead (file size: 50MB)

	BT	CYCLOPS
Normalized server load	39.86	1.5
Outgoing overhead	55 KB	52 KB
Incoming overhead	2560 KB	716 KB
Feedback overhead	–	145 KB

Next we examine the evolution in time of the feedback signal (namely, system-wide piece availability) generated by the CYCLOPS swarm monitor and the content server state transitions it triggers. Let M be the number of pieces into which a file is divided, and let $I(i,t)$, $i=1,\ldots,M$ be the indicator function for piece i at time t, *i.e.*, $I(i,t)=1$ if there is at least one copy of piece i at time t, otherwise $I(i,t)=0$. The availability feedback signal $A(t)$ at time t is computed as:

$$A(t) = \frac{\sum I(i,t)}{M} \qquad (4)$$

Figure 5 shows the time-series for the swarm size, the availability feedback signal, and the content server state transitions induced by this signal. It shows that as soon as the feedback signal indicates piece starvation (*i.e.*, availability is less than 1), the content server switches to the serving state and feeds the swarm. Piece availability is zero when the swarm bootstraps, and drops whenever clients holding the unique copy of a particular piece depart from the system. The content server switches from the idle state to the serving state only when necessary to restore piece availability to 1. Note that in this experiment we have purposefully created an extreme case of swarm dynamics: in a real swarm, user behavior is not as synchronous.

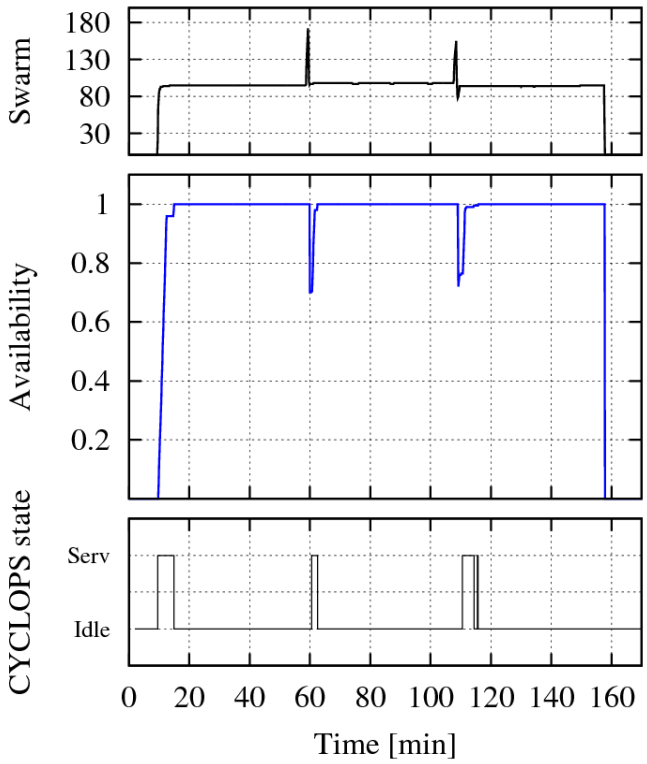


Figure 5 : Waves of arrivals: availability over time.

2.5.3 Live Internet Experiments

In the set of experiments we present in this Section, we do not control the client arrival and departure processes, but rather we let these processes reflect the popularity of the content we forged and advertised. Furthermore, clients participating in our swarms exhibit realistic uplink and downlink capacities, unlike our PlanetLab experiments in which all clients have the same uplink capacity.

For CYCLOPS, out of a total of 7633 users we tracked, 3509 obtained the full content. All other users departed before finishing the download process. For the open-loop-BT content server, 2486 out of a total of 5044 users completed the content download. Figure 6 depicts the instantaneous number of users for both swarms. In our experiments, after the transients of the first few hours have subsided, the user arrival and departure rates within each swarm equalized, with approximately 35-40 users joining each swarm per minute.

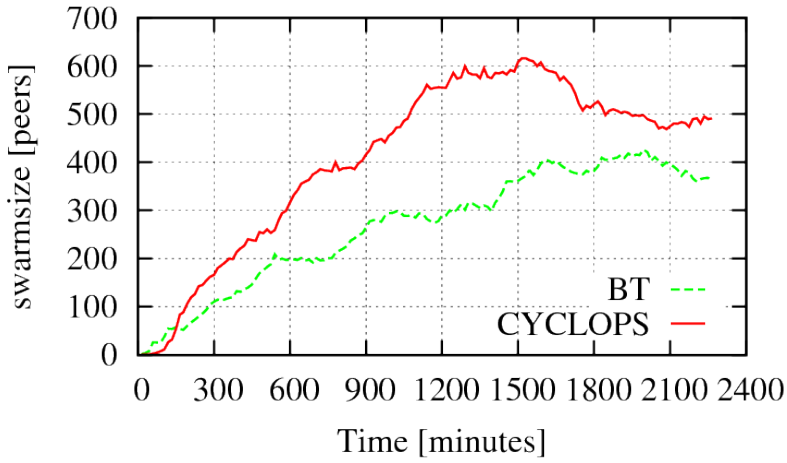


Figure 6 : Live Experiment: Evolution of swarm size over time.

Figure 7 shows the box-plot of the content delivery times achieved by all users that were able to complete the download. These results indicate that the median delivery time achieved by both content servers is very similar. For the CYCLOPS content server, the ECDF indicates longer tails: this is mainly due to a larger swarm size, which included clients with poor Internet connectivity. From the end-users' perspective, the difference in the download performance when they are served by CYCLOPS or by open-loop-BT is negligible.

The off-cloud bandwidth utilization, the associated volume of data and related costs supported by content servers underscore the superiority of CYCLOPS. Table 4 indicates that the CYCLOPS content server served a total of 731.6 MB of content data, while the open-loop-BT seed injected a whopping 133.03 GB of content data! Table 4 also reports the overhead traffic, as defined in the previous section.

These results support our conclusion that the framework discussed in Section 2.2 and the particular instance we presented in this work are viable candidates for real Internet content distribution systems. Note that both experiments lasted 38 hours, and that the swarm sizes allowed us to assume equivalent uplink capacity distributions for users in each torrent.

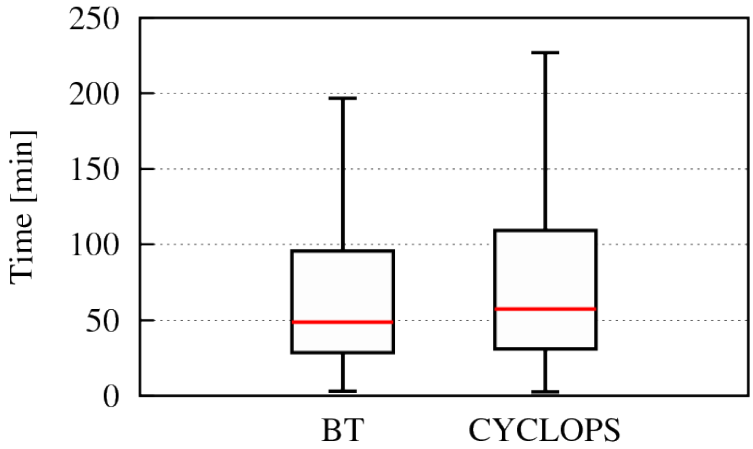


Figure 7 : Live Experiment: content download times (file size: 357.5 MB).

Since we deployed our content servers on Amazon EC2 instances, we were able to quantify the economic value of our proposed scheme: For the experiment we carried out, the total cost (including overheads) for distributing the same content when using a legacy BT seed is roughly 180 times higher that of a CYCLOPS content server.

Table 4 : Live Experiment: service statistics (file size: 357.5 MB)

	BT	CYCLOPS
Total number of users observed in the swarm	2*5044	2*7633
Normalized server load	381.04	2.05
Outgoing overhead	6.5 MB	0.2 MB
Incoming overhead	160.8 MB	24.6 MB
2***Cost of delivery**	2*$ 23.73	2*$ 0.13

2.6 Additional Considerations

We now discuss several points that complement the work presented. We start by suggesting practical ideas to implement a content server with alternative objectives and feedback signals; then we address the case for multiple content servers and conclude with a discussion of the robustness of our framework against attackers aiming at thwarting the content distribution process.

Dealing with alternative objectives and feedback signals: The framework proposed in Section 2.2 is general enough to allow many possible combinations of objectives and feedback signals. For example, an alternative objective may be to ensure some minimal level of service based on a feedback signal regarding the *average* delivery time of content to clients. The swarm monitor described in Section 2.3 can readily measure the average content delivery times, using the same swarm signaling traffic we discussed earlier. Indeed, clients

advertise whenever they receive a new content piece, information that can be simply used to compute the average download rate of the swarm. Based on this information, the content server can choose the appropriate level of off-cloud bandwidth (*i.e.*, the cost it incurs) to complement the serving capacity λ of the swarm, with the constraint of remaining in the vicinity of the quiescent operating point discussed in Section 2.2. With reference to Figure 1, this approach corresponds to a content server selecting to contribute bandwidth resources that move across the various operating regions obtained for different values of λ.

Dealing with alternative ways to collect feedback signals: The swarm monitor described in Section 2.3 is achieved using a set of OF nodes that connect to all users. We show in Section 2.5 that the cost of this solution, in terms of overheads, is not significant. Nevertheless, maintaining many connections may pose some challenges. An alternative solution is to use periodic sampling of the swarm state: The OF nodes, instead of connecting to all the users in the swarm, periodically obtain a subset of users from the tracker and connect temporarily to this subset to collect the information about pieces owned by the users. Using sampling statistics, it is possible to *infer* system-wide piece availability, subject to preset levels of confidence. Clearly, the larger the sampling set, the more precise the availability information: in practice, approximating data availability may yield higher server load, since pieces may not be detected even if they are in the swarm.

Dealing with multiple content servers: In the previous section, we conducted experiments in which a single content server is deployed. There are many obvious reasons to consider a more general scenario involving multiple content servers. For example, a CDN operator may wish to use CYCLOPS on edge servers positioned in several locations so as to serve clients efficiently: in this scenario, end-users might be directed to their geographically closest CYCLOPS content server. Traffic locality to mitigate the impact on ISPs economics, calls for a technique to create distinct swarms. This can be achieved with techniques proposed in the literature without requiring any modification to the design of CYCLOPS. Alternatively, multiple CYCLOPS servers could be combined to contribute to the same swarm. In this case, such content servers would have to coordinate what content pieces they serve and when to avoid inefficiencies. Our current implementation does not have provisions for avoiding the overlap between the *serving sets* compiled by different content servers. That said; standard distributed algorithms could be easily used to manage such situations for production-scale systems.

Dealing with Cloud services cost models: In this work we have demonstrated the benefits, in terms of bandwidth consumption, of our approach for Cloud-based content delivery. Since we focused on a simple objective, *i.e.,* to keep a swarm alive, CYCLOPS exhibits an *intermittent behavior*: the system leaves the idle state only when the swarm risks starvation. Under this operational mode, it is natural to question whether this behavior matches current cost models that apply to resources rented in the Cloud. For example, the granularity for paying an Amazon's Elastic Computing Cloud (EC2) instance is one hour. As such, although bandwidth resources are neither used nor payed for when CYCLOPS is in the idle state (probing traffic aside), the virtual machine is payed independently of the bandwidth consumption.

With respect to the above discussion, we stress that the experimental setting we used in this work is not intended to be deployed in production. Ideally, our system is suitable for a deployment with the Amazon S3 and its CDN extension (called CloudFront): in this case, our approach would benefit the service operator (e.g. Amazon). Alternatively, our approach is suitable for more elaborate scenarios in which multiple contents are distributed via multiple instances of CYCLOPS that coexist in the same virtual machine. In such case, the unit cost of the virtual machine can be amortized by having CYCLOPS content servers coordinate: when one CYCLOPS instance is idle, another instance can be in the serving state, if necessary. Note that such coordination is not trivial: the intermitted behavior of CYCLOPS is a result of swarm dynamics, which cannot be controlled. It is outside the scope of this work to extend the CYCLOPS architecture to cope with such additional complexity.

Dealing with adversarial workloads: Denial of Service attacks as well as other improper behavior of end-users aiming to exploit swarm resources is a concern that has to be considered when embracing a peer-assisted CDN solution such as ours. Although this is an important problem to address, here we focus on deliberate attacks by a client (or a set of colluding clients) targeting the specifics of our CYCLOPS framework. Other types of attacks typical of P2P systems, such as Sybil or Eclipse attacks, can be solved using the techniques already presented in the literature [13]. We recognize two possible adversarial exploits, where the aim is to pollute the feedback signal computed by the CYCLOPS swarm monitor.

In the first, an adversary may seek to consume as much off-cloud bandwidth as possible. This can be done by inducing the content server to detect piece starvation (when none truly exists), thus causing the server to wastefully inject content. Since CYCLOPS swarm monitor tracks *all* clients in a swarm, such an attack would require a colluding set of malicious users of a size approximately equal to the whole swarm size, which can be safely assumed impractical.

In the second, a set of colluding users may engage in a DoS-like attack to hinder content distribution, by inducing the content server to conclude that the swarm is healthy (when the contrary is true). This causes starvation of legitimate clients. This can be solved by letting the swarm monitor to compute the average download rate of the swarm (as explained before in this Section). Based on this information, in case of content starvation, the swarm monitor may trigger an alarm, indicating, for instance, the less replicated pieces.

2.7 Related work

Peer-Assistance: Peer assisted content distribution have been the subject of many recent studies. Of these, the work of Huang, Wang, and Ross [14] could be seen as similar in nature to the work presented here. In that work, the authors advocate the use of peer-assisted content distribution by evaluating the potential gain from peer-assisted video distribution using real-world traces of two large CDN companies, Akamai and Limelight (the underlying architecture of both of which they characterized). Their approach uses the model in [15] to obtain bounds on the server load and download times, should swarming among end-users be allowed. They also quantify the potential reduction in ISP peering traffic, resulting from traffic localization.

In the same vein, our work is based on an analytical model that gives key insights as to the benefits of peer-assisted content distribution (although, our focus is on bulk as opposed to video transfers). Beyond a "proof of concept" using a tractable mathematical formulation, we go one step further by presenting practical feedback-control content injection policies that aim to satisfy performance objectives while minimizing provider's costs. Our implementation is evaluated in realistic contexts, and our results go beyond a purely theoretic estimation of the benefits of peer-assisted content distribution.

Frugal Seeding: To the best of our knowledge, the only work that has a similar objective to ours – in terms of reducing the load/cost on a content source, albeit in a very different setting – is Sanderson and Zappala's work [16]. In that work, once the seed has determined a subset of pieces that should be injected in a swarm, it will satisfy any number of requests for those pieces. As a consequence, their technique does not offer the same level of control on the seed workload as the policies we study in this work. Indeed, we observe that for experiments carried out in similar settings, our content servers inject orders of magnitude less traffic than what was documented in [16]. Additionally, our system does not require any parameter to be empirically set.

Chen *et al.* [17] study the "Super-seeding" mode introduced by an alternative BT client to help peers with slow Internet connections perform initial content seeding. The objectives of "Super-seeding" are different from ours. Moreover, a number of problems due to multiple peers using "Super-seeding" have been reported. The work in [18] proposes a "Smart seed" policy, which advocates serving just one copy of each piece. Besides the fact that Smart seed does not take into account dynamic scenarios, it requires the modification of clients, while our system involves changes only to the server with no modification to the client.

Models and Bounds: The literature is rich with analytical models that dissect many aspects of P2P content distribution. In [19] and [20], the authors derive lower bounds for the minimum content distribution time of a swarm-based P2P application: we build upon those works, but focus instead on the relation between the content server upload rate and the download rate achieved by peers. The work in [21] belongs to the family of fluid models of BitTorrent-like applications: however, in this model it is the number of peers (as opposed to traffic) in the system that is taken as fluid. The authors in [21] develop a differential equation for the fluid model, from which they determine the performance of the dynamic system. We also model content replication in a dynamic setting, but instead consider the number of piece replicas as the dynamic variable modeled using a Markov process.

Bandwidth Allocation in P2P Systems: While the study of alternative mechanisms that improve the bandwidth allocation in P2P systems is orthogonal to our work, results from such studies could clearly have positive implications on content server utilization. In [6], the authors design a content distribution system with the objective of maximizing the download rate of all participants in a managed swarm. This work stems from the observation that, in steady state, a swarm can be in three different states: if the upload bandwidth allocated by content servers is insufficient, peers will not be able to fill their uplink capacity and the

aggregate download rate will suffer; by increasing the amount of bandwidth awarded to a single swarm, the content server can guide the system to operate in a regime where the uplink capacity of peers is gradually filled, up to a point in which also the downlink capacity of all peers is filled; at this point, server capacity can be diverted to other swarms. The system design in [6] is based on a wire protocol that induces peer participation (using virtual currency) to achieve a global system optimization. In our work, we focus on a different objective: we try and address the question of whether it is possible to optimize the bandwidth utilization by content servers, without negatively impacting the performance perceived by clients. We note that the model we use in this work can also explain, though in more general terms, the key intuition behind the Antfarm work [6].

The problem of devising efficient uplink allocation algorithms for swarm-based P2P bulk data transfers is addressed in [22]. Instead of using empirically set parameters, as done in BT, to determine the amount of uplink capacity dedicated to each remote connection, they cast uplink allocation as a fractional knapsack problem, and design a simple heuristic utility function to decide the amount of bandwidth a peer should dedicate to each remote connection. The focus of their work is on a cooperative P2P setting, in which peers are assumed to fully abide to the prescribed algorithms.

2.8 Conclusion

In this section of the deliverable, we have demonstrated that peer-assisted content distribution could be leveraged to *supplant* as opposed to *supplement* the content provider's resources for purposes of efficient and scalable content distribution, *without* negatively impacting the performance perceived by clients. Our approach is based on a feedback-controlled swarm feeding mechanism, which we have modeled analytically and evaluated empirically using CYCLOPS – a full-fledged service that we have implemented and deployed on the Amazon EC2 cloud.

Our extensive experimental results – including the *live* distribution of content to thousands of real Internet users – show that CYCLOPS achieves enormous cost savings for the provider (as high as two orders of magnitude when compared to non-feedback-controlled BitTorrent-based services) without noticeably impacting the performance perceived by end-users. By deploying our servers on Amazon EC2 servers we were able to show that the mechanisms we developed as part of this work have a clear impact on content distribution economics, including significant reduction of costs for content providers, and much more efficient resource utilization for content hosts and distributors.

Our on-going work is focused on exploring alternative objectives and alternative feedback signaling processes in CYCLOPS, as well as extensions that take into account multiple (possibly competing) content servers involved in the distribution of content from multiple sources.

3 CCN with Cooperative Caching

The main objective of the VIPEER project is to propose novel mechanisms to integrate traffic engineering (i.e. guiding the traffic where resources are available) in the current video streaming delivery system, so as to ameliorate the client QoE.

In this deliverable, we describe a cooperative caching policy based on CCN (Content Centric Networking [26]) Paradigm ; the cooperative caching is performed within the ISP's routers, which can be considered as dCDN nodes.

CCN is potentially an efficient solution for streaming delivery thanks to its anycast nature and proxy-like caching ability. However, the limited cache capacity on routers may degrade the performance of the whole system. Therefore, intelligent storage management is necessary to guarantee the system efficiency. Instead of using basic least recently used (LRU) or least frequently used (LFU) caching policy, we propose a new cooperative caching policy between content routers (CR). The aim of our cooperative caching is to reduce the amount of requests served by the entities outside the ISP (e.g. CDN servers). As a result, the cooperative caching limits CDN-to-ISP bandwidth utilization.

In the following, we first give the motivation of our work. Then we introduce the main idea of the cooperative caching. Thereafter, we detail the augmented CCN protocol to realize our novel caching policy. We also provide theoretical analysis to show the advantage our proposition. Finally, the cooperative caching strategy is tested in both Catch-up TV and Video on demand (VoD) system to highlight the benefit for ISP.

3.1 Introduction and Background

3.1.1 Context: Content Centric Networking

The deployment of Internet routers having caching capabilities (CR for Content or Caching Router [23]) represents an opportunity to revisit the techniques that are currently used to deliver content in the Internet. So far, the flaws of the Internet, in particular the poor performances of communication links traversing several Autonomous Systems (AS) [24], have been overcome by the deployment of large-scale CDN such as the Akamai network [25]. The recent works toward CCN [26] introduce new techniques, which allow routing queries and data based on content name. These protocols enables the exploitation of the storage resources of any machine in the network, in particular the CR. However, authors of CCN suggest using a basic LRU policy for the cache management of every CR. The current paper deals with a new caching policy for CR in order to build a cooperative in-network cache. This objective requires taking into account:

- The distributed nature of this cooperative cache. Contrarily to the centralized management of CDN, the envisioned network of CR is by nature distributed: every CR must decide by itself whether a content that it routes should be cached. Moreover, a claimed objective of CCN is to retain the simplicity and scalability of current

Internet protocols. Therefore, CRs can only use local information in order to take their decision.

- The peering relationships between ASs. The equilibrium of the whole Internet depends on the selfish actions of every AS. In the CCN perspective, an operator of AS becomes a content provider through the CRs it manages. A rationale behavior is to cache in priority the most expensive content, “textit-i.e.} when the path to the server storing this content contains expensive transit inter-AS links.
- The small caching capacity of CRs. Studies show that video content will represent more than 90% of the whole Internet traffic in a few years [27]. High-definition video streams with bitrate in the order of megabits per seconds require storage capacity in the order of gigabits. In comparison, the storage capacity of CR is expected to be small (for example, only 36 gigabits in [23]).

3.1.2 Our Focus: ISP-friendly Time-shifted Streaming

We consider an ISP, which wants to minimize the cross-domain traffic related with streaming delivery including Time-shifted TV and VoD service. A series of recent works has explored CDN-based and peer-to-peer approaches for streaming delivery [27-35]. However, none of these solutions focuses on the cost of content delivery for the ISP. In CDN-based systems, the quality of the distribution is a function of the location of CDN servers, and of the efficiency of the query redirection mechanism forwards the appropriate server. An ISP that does not interact with a CDN provider is not able to manage the traffic for the end users located in its AS. This lack of interaction is expensive for the ISP because every request from end user is treated as one unique stream, resulting in larger incoming cross-domain traffic if the CDN is located outside of the AS.

Peer-to-peer and peer-assisted architectures present also some weaknesses. Despite recent efforts toward a better interaction between ISP and peer-to-peer applications [36], the proposals for video streaming delivery ignore the network location of peers. Hence, it may happen that the video is downloaded from one or several distant peers. In our previous works [33], we have addressed the problem of guarantying that all past chunks are correctly kept in a peer-to-peer system.

3.1.3 Our Proposal: Cooperative In-Network Caching

Our goal is to leverage on a set of deployed CRs to minimize the amount of queries for video streams that are treated by servers outside the ISP network.

In this report, we propose to replace the LRU policy of CCN by a new cooperative policy, with respect to the simplicity of CCN protocols. Our proposal is illustrated in Figure 8 using an example offering time-shifted TV service. In the example, we assume that a given stream is produced by a TV broadcaster. At a given time *t*, we consider that 21 chunks have been produced (from 0 to 20). Each CR has a cache capacity of 10 chunks. According to the LRU policy, the caches of the three CRs are filled by chunks -11 … 20}. At time *t*, two clients

request a time-shifted part of the stream, respectively from chunk 5 and 15. With the CCN protocol, the latter request for chunk 15 is satisfied by the CR r_1, but the request for chunk 5 has to be forwarded to the server. The lack of coordination among CR results in an inefficient caching strategy with redundant data stored on adjacent CRs.

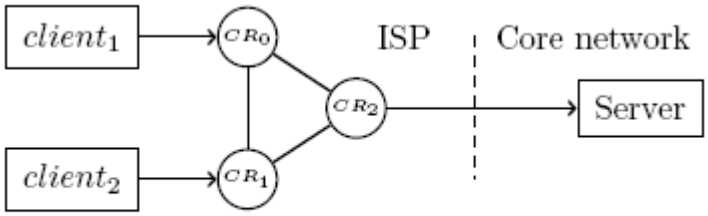


Figure 8 : Example of cooperative cache

Our proposal is that a CR does not cache all the chunks that it routes, but only part of them. Every CR is associated with a label, which is a positive integer smaller than a fixed integer *k*. Every CR uses the LRU policy only for chunks whose number modulo *k* is equal to its label. In our example, we assume that *k* is equal to 3, and every CR r_i is associated with label *i*. Them, the CR r_0 stores the chunks -0,3,6, … ,18, which correspond to the 10 last chunks routed by r_0 such that their chunk numbers modulo 3 are equal to 0. With this strategy, the request for chunk 5 is not forwarded to the server, but directly satisfied by r_2. In parallel, the request for chunk 15 is no longer treated by r_1, but r_1 forwards the request to r_0, which stores this requested chunk. With this cooperative in-network caching strategy, machines in the AS of the end users treat both requests.

3.1.4 Our Contributions: Algorithms and CCN Protocol

In this report, we do not describe all aspects of this proposal. In particular, we do not detail how an ISP notifies all CRs that are under its control about the set of streams that have to be stored for the purpose of a time-shifted service. This notification contains (i) the name of these streams, (ii) the amount of storage space devoted for these streams, and (iii) the number of different labels *k*. We focus on following contributions.

First, we give a theoretical focus on the initialization stage, the phase during which each CR determines its label. A trivial implementation consists in a random choice. In previous works, we have shown that significant gains can be obtained from a label assignment that takes into account the network linkage among CRs [37]. However, the optimal assignment has been shown to be NP-complete. We present in the current paper a distributed algorithm that allows each CR to determine its label, this assignment of labels being not worse than ((3/2)k-5/2) of the optimal assignment.

Second, we describe an augmented version of the CCN protocol that implements our cooperative caching strategy. We show in particular that the protocol keeps the simplicity of

the original CCN protocol. We present the refinements that are necessary to implement the cooperative caching.

Then, we use simplified network models to explain the advantages of our cooperative caching policy on a non-cooperative one. In particular, we show that cooperative caching improves the hit rate for middle popular videos.

Finally, we show some simulation results. We enhanced the open-source CCNx prototype by integrating our cooperative caching policy. User behavior is simulated by synthetic traces provided in [33] for time-shifted TV and statistics reported in [38]. We used an ISP topology measured by Rocketfuel [39].

The performance improvement due to cooperative caching are impressive. For example, assuming that the ISP reserves 1 gigabytes of cache for streaming delivery service, the cooperative caching strategy is shown to perform 60% better than the LRU policy.

3.2 Network Model

We consider a network *N* consisting of a set of routers, and a set of bidirectional links between these routers. We note by *V* the subset of routers that are CR (*i.e.* having caching capacity); those routers are the dCDN nodes. We assume that the ISP is able to compute a static "distance" d_{ij} between two CRs r_i and r_j. This "distance" represents the connectivity between two CRs. Examples of such "distance" metrics are :

- the length of the shortest path joining r_i to r_j in *N*,
- the inverse of the capacity of routers on this path, or
- the average latency measured between these two routers.

The (k-1) CRs in *V* that are the **nearest** from the CR r_i are expected to cooperate with r_i. Here, nearest means having the smallest distance. Our goal is to avoid that these CRs store the same chunks. We note by *N(i)* this subset of CRs in *V*, and, by extension, *N[i]* is the set $N(i) \cup r_i$.

In the following, we assume that non-CR routers are able to transmit the messages from one CR to another without troubles. The CRs do not experience failures.

3.3 Initialization Stage

Each CR should initially determine its label. Our goal is to ensure that every CR is as close as possible from all the labels that are different that its own label. We note by *L(i)* the k-1 CRs having the k-1 other labels and that are collectively the closest from r_i. The sum of distances from a given CR r_i to the CRs in *L(i)* is called the *rainbow distance* of r_i, and it is noted d_i. Formally $d_i = \sum_{r_j \in L(i)} d_{ij}$. Determining the optimal assignment of labels, *i.e.* the assignment such that the sum of all rainbow distances is minimal, is NP-hard [37].

To prove the performance ratio of our algorithm, we begin with the definition of lower bound. Given an instance of the problem, it is possible to determine a lower bound solution by setting that every CR with its k-1 closest CRs store collectively the *k* labels, formally *L(i) = N(i)* for every CR r_i. This obvious optimal assignment is impossible in many cases, but it gives a lower bound. We call *fractional distance*, denoted by $\overline{d_i}$, the sum of the distances between a CR r_i and its k-1 nearest neighbors, so $\overline{d_i} = \sum_{r_j \in N[i]} d_{ij}$.

3.4 Distributed Algorithm

There are two rounds. First, each CR exchanges information with its 2-hop neighbors. Then, each CR allocates labels on its neighbors and itself. For each CR r_i the first round goes as follows: 1) it collects from its k-1 nearest neighbors their (k-1) nearest neighbors, thus, every CR knows all CRs that are at 2 hops in the (k-1)-nearest neighbor graph. 2) It sends to this 2-hops neighborhood the fractional distance $\bar{d}_i$. 3), then it enters waiting mode. 4) It waits until all two-hop neighbors having a fractional distance that is lower than $\bar{d}_i$ emit a release message. 5) It executes a *Label Allocation Process* (LAP), and then broadcasts a release message. 6) When all two-hop neighbors have sent a release message, if r_i is both marked as *saved* and not assigned label, then it chooses the farthest label for itself.

The second round, namely LAP, is label allocation. The algorithm tests the condition that *no two CR r_j and $r_{j'}$ in N[i] can hold the same label*. If N[i] satisfies the condition, *i* allocates labels on every CR both in N[i] and holding no label, such that no *j* and *j'* hold the same label. Then *i* marks itself as *optimized*. If N[i] does not satisfy the condition, i marks itself as saved. Note that some of the saved CRs are labeled but others not.

Correctness and Analysis

Provided that the algorithm runs in a correct environment, i.e., there is neither faulty links nor faulty nodes, it returns a solution satisfying the following conditions. First, it runs infinite time. Second, each CR eventually holds a label. Third, there is no missing label in the system.

Theorem 1 The algorithm gives a valid solution in a correct environment.

Proof. The last condition is easily satisfied when the first CR (the CR possess the local minimum rainbow cost) executes LAP. To show that the first and second conditions are also tenable, we just need to prove that *i* will receive all release messages from its two-hop neighbors in a finite time. If the algorithm does not terminate, it must be some nodes *i* and *j* such that *i* never receives a release message from *j*, so *i* never executes LAP, and broadcasts the release message. Yet, the fractional distance being a unique real number, there is always a CR with a smallest distance, which can enter LAP and broadcast the release message. This also leads to the fact that each CR will execute LAP. Together with the fact that the distance of each CR is broadcasted only once, we conclude that no CR will be in waiting mode for infinite time. Since the number of nodes is finite, the algorithm terminates in finite time; thereafter each CR holds a label.

Each CR executes LAP, and, as the distance function *d* gives a total order on nodes, no two nodes within two hops are local minimal at the same time, so no two nodes within two hops execute LAP at the same time.

Theorem 2 For any k ≥ 3, the distributed algorithm gives a solution no more than (3k/2-5/2) times the lower bound.

Proof. For an optimized CR $r_{i'}$, we know that $d_{i'} = \bar{d}_{i'}$. For a saved CR r_i, there are two cases: 1) the label on r_i has been assigned by another CR, and this label coincides with the label held by one of its k - 1 nearest neighbors, 2) two nodes in N(i) hold the same label.

In the first case, the label on r_i has been assigned by an optimized CR $r_{i'}$. It means that $r_i \in N(i')$, and that $\bar{d}_{i'} < \bar{d}_i$ because $r_{i'}$ executed LAP before r_i. Assume that the label of r_i is 1, and the neighbor of $r_{i'}$ hosting label *l* is noted r_j^l. Then the rainbow cost of r_i can be calculated as follows. Since $r_j \in L(i)$ is the nearest neighbor of r_i, we have $\sum_{r_j \in L(i)} d_{ij} \leq \sum_{r_{j'} \in N[i']} d_{ij'}$.

$$\sum_{r_j \in L(i)} d_{ij} \leq \sum_{r_{j'} \in N[i']} d_{ij'} = \sum_{l=2}^{k} d_{ij'^l} \leq \sum_{l=2}^{k} (d_{ii'} + d_{i'j'^l}) =$$

$$(k-1)d_{ii'} + \sum_{l=2}^{k} d_{i'j'^l} = (k-2)d_{ii'} + \sum_{l=1}^{k} d_{i'j'^l} \leq (k-1)\bar{d}_i$$

In the second case, there must be an optimized CR $r_{i'}$ within two hops from r_i, such that $\bar{d}_{i'} < \bar{d}_i$. Assume that r_{j1} and r_{j2} are the two nodes that prevent r_i from entering the optimized state, and $d_{ij_1} < d_{ij_2}$. Without loss of generality, we can assume label 1 at j_1. If r_i chooses label *h* in the second phase, then $h \neq 1$, as r_{j1} is among the nearest neighbor of r_i. According to the algorithm, we have $r_{j_1} \in N(i) \cap N(i')$. After labels allocation is finished, r_i and r_{j1} hold different labels. Thus $rj_1 \in L(i)$. Then the rainbow distance of r_i can be calculated as follows:

$$\sum_{r_j \in L(i)} d_{ij} \leq \sum_{r_{j'} \in N[i']} d_{ij'} = \sum_{l=1}^{k-1} d_{ij'^l} \leq d_{ij_1} + \sum_{l=2}^{k-1} (d_{ij_1} + d_{j_1 i'} + d_{i'j'^l}) =$$

$$\sum_{l=1}^{k-1} d_{i'j'^l} + (k-3)d_{i'j_1} + (k-1)d_{ij_1} \leq \sum_{l=1}^{k-1} d_{i'j'^l} + (k-3)d_{i'j_1} + \frac{k-1}{2}(d_{ij_1} + d_{ij_2}) \leq$$

$$(k-2)\sum_{l=1}^{k-1} d_{i'j'^l} + \frac{k-1}{2}\sum_{l=1}^{k-1} d_{ij'^l} \leq (\frac{3}{2}k - \frac{5}{2})\bar{d}_i$$

As $k-1 \leq \frac{3}{2}k - \frac{5}{2}$ for any k greater than 3, our algorithm gives a solution no more than $\frac{3}{2}k - \frac{5}{2}$ times the lower bound.

3.5 Augmented CCR Protocol

We start by a quick summary of the main principles of CCN. Please refer to [26] for more details. Then, we present the changes that we propose in order to implement our cooperative caching strategy.

3.5.1 CCN in a Nutshell

In CCN, every content is identified by a hierarchical name like URL and divided into multiple chunks. The content name plus a sequence number identify each chunk. When a provider publishes the content, the CR connected with that provider floods an advertisement of the content to adjacent CRs. A Forwarding Information Base (FIB) is established to redirect any incoming interest (*a.k.a.* request) toward content provider. When an interest is forwarded according to the FIB, an entry into the Pending Interest Table (PIT) is created to trace the requesting interface, so that the content can be sent back along the reverse path of interest. The content is then cached by the CRs on its forwarding path. If the content is requested again, the replica in the Content Store (CS), or cache, is directly delivered by the CR.

3.5.2 New Tables in CCN

In order to implement our cooperative caching strategy, we require two new tables. First, every CR r_i maintains the information of its k-1 closest CRs in L(i) in a new table, namely Collaborative Router Table (CRT). There are three fields in CRT of a CR: the label, the identifier of the collaborative router and the interface. Thus, every CR knows where to redirect an interest or forward a chunk. The second table added on the basic CR is the Collaborative Content Store (CCS). In CCS, a CR keeps the names and the sequence numbers of all the chunks that may be found in its collaborative cache. When an interest arrives, the preference of the four prefix matches is CS match to CCS match to PIT match to FIB match.

3.5.3 Distribute Chunks in the Cooperative Cache

When a chunk *c* is sent back to consume an interest, a CR r_i with label l_i, which receives *c*, should take a decision (whether to cache it or not) based on l_i, on the identifier *c* of this chunk, and the match result. We describe the action as follows:

- This chunk is handled by r_i, that is $c \bmod k = l_i$. The CR r_i adds *c* into its cache, and removes the least recently used chunk. Then r_i calculates a PIT match. If a PIT match is found, it forwards the data to the interfaces indicated by the PIT; otherwise, the process is finished.

- This chunk is not handled by r_i, which is $c \bmod k \neq l_i$. The CR r_i first finds in its CRT the router r_i having the label l_j that matches with the chunk *c*. Then r_i sends the chunk *c*

to r_j. Moreover, if r_i finds a match in its PIT for this chunk, it also forwards *c* to the requesters. Finally, r_i adds *c* in the CCS Table, so that later interests requiring the same chunk will be forwarded to r_j, but no longer according to the FIB.

In this scheme, each data packet should carry a random nonce to prevent broadcast storm. When a duplicated packet with the same nonce is received, it should be immediately discarded.

3.5.4 CCS Consistency

At every time, the CS of a given CR r_i should be consistent with all the CCS tables of all CRs that consider r_i among its closest CR. In particular, when an entry of the CS r_i is discarded by the caching policy, the corresponding entry in the CCS of a CR r_j with $r_i \in L(j)$ should also be deleted; otherwise interests for the eliminated content may be lost in the forwarding process. For example, if r_j receives an interest requiring chunk *c*, it finds the CCS match point to r_i. Assume that chunk *c* in r_i has been discarded. The CR r_i forwards the interest following the FIB entry. If r_j is an intermediate CR between r_i and the corresponding server, the interest will be regarded as a duplicated one, and discarded by r_j. Therefore, the interest for chunk *c* is lost. We should remind that the lost interest can be recognized as a duplicated one because every interest is given a random nonce when it is generated.

To both maintain consistency and avoid increasing control messages, we use piggyback interest (p-interest) to carry the control information. A CR r_i with label l_i acts as follows when an interest for chunk *c* is received:

- The requested chunk *c* is handled by r_i, that is $c \bmod k = l_i$. The CR r_i first calculates the CS match. If a CS match is found, it sends back the data directly. Otherwise, if a PIT entry is found, it adds the requiring face into the pending list. If neither CS match nor PIT match is found, r_i changes the interest into a p-interest; it generates a new nonce for the p-interest, and forwards this p-interest according to FIB entry.
- The requested chunk *c* is not handled by r_i, that is $c \bmod k \neq l_i$, and the interest is a p-interest. The CR r_i needs to determine whether the CR r_i indicated in the p-interest is in the CRT of r_i. When r_j is not the relative collaborative router, r_i executes normal process. Otherwise, r_i should eliminate the CCS for the chunk required in the interest, and then adds the requiring face in its PIT. Finally, r_i forwards the interest according to the FIB, even if PIT already existed. The final step ensures that the interest arrives at a provider.
- The requested chunk *c* is not handled by r_i, that is $c \bmod k \neq l_i$, and the interest is not a p-interest. The CR r_i just executes the normal CCN process (collaborative CS match is preferred than PIT match, and PIT match is preferred than FIB match).

3.6 Analysis of Cooperative Cache

We now use some network models in order to highlight the advantages provided by this cooperative caching policy. Although the analyzed structures are simple, the real network can be seen as a combination of them. The performances of LRU policy have been previously studied in [18,19]. We extend our analysis based on the results presented in [40]. The first model contains several caches directly connected with a server. The two approaches are shown in Figure 9 & Figure 10:

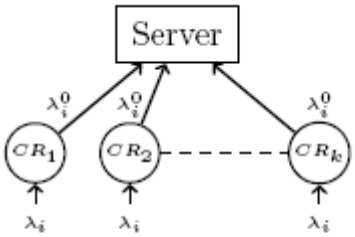


Figure 9 : Individual caches connected to a server

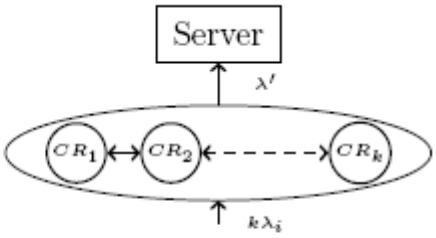


Figure 10 : Cooperative caches connected to a server

In Figure 9, *k* homogeneous caches with cache size *C* are connected with the server. Each cache receives requests for segment *i* with the same request rate λ_i. If *i* is not stored in the cache, a cache miss happens. The cache miss rate for segment *i* is denoted as λ_i^0. Following the results in [40], the total cache miss rate of the *k* caches is $\lambda = k \cdot \lambda_i^0 = k \cdot \lambda_i e^{-\lambda_i \tau_i}$, where τ_i is the maximum inter-arrival time between two adjacent cache hits for segment *i*.

In Figure 10, the *k* homogeneous caches work cooperatively. Each cache stores distinct chunks and form a cooperative group with cache size $k \cdot C$. Although the cache replacement policy of the cooperative group is not exactly LRU, we approximate the group as a single cache using LRU policy. The cache miss rate generated by the cooperative group is denoted as $\lambda' = k \cdot \lambda_i e^{-k \cdot \lambda_i \tau_i'}$.

Theorem The cooperative group caching achieves at least the same performances as the individual caches.

Proof. We need to prove that $\lambda > \lambda'$ for any $k \in \mathbb{N}^+$ and $C \in \mathbb{N}^+$. We begin our proof from the fact that the function $f(\tau_i) = e^{-\lambda_i \tau_i}$ is continuous and monotone decreasing. The function $u(\tau_i) = \sum_{j=1,j\neq i}^{N} f(\tau_i) = \sum_{j=1,j\neq i}^{N} e^{-\lambda_j \tau_i}$ is also continuous and monotone decreasing since $u(\tau_i)$ is the sum of $f(\tau_i)$. According to [40], τ_i and τ_i' can be calculated by the following two equations.

$$\sum_{j=1,j\neq i}^{N} (1 - e^{-\lambda_j \tau_i}) = C \tag{1}$$

$$\sum_{j=1,j\neq i}^{N} (1 - e^{-k\cdot\lambda_j \tau_i'}) = kC \tag{2}$$

Minus equation (2) by equation (1), then we have:

$$\sum_{j=1,j\neq i}^{N} e^{-\lambda_j \tau_i} - \sum_{j=1,j\neq i}^{N} e^{-k\cdot\lambda_j \tau_i'} = (k-1)C \tag{3}$$

Since $k \in \mathbb{N}^+$ and $C \in \mathbb{N}^+$, we can obtain:

$$\sum_{j=1,j\neq i}^{N} e^{-\lambda_j \tau_i} \geq \sum_{j=1,j\neq i}^{N} e^{-k\cdot\lambda_j \tau_i'} \tag{4}$$

As $u(\tau_i)$ is monotone decreasing, we conclude that $\tau_i \leq k\tau_i'$. Because $f(\tau_i)$ is also monotone decreasing, we know that:

$$e^{-\lambda_i \tau_i} \geq e^{-k\lambda_i \tau_i'} \tag{5}$$

Multiply both sides of equation (5) by $k\lambda_i$, the proof follows

Another structure is the tandem of *k* caches shown in Figure 11. The request rate from client for *i* at each cache is still identical and denoted as λ_i. Instead of forwarding the missed stream directly to the server, the missed stream is passed to the next hop cache in the direction of the server. Since only the *k*th cache is connected with the server, the missed stream λ^0_{ki} of the cache *k* is the missed stream of the multi-cache system. Therefore, the breakthrough point is to find the expression of λ^0_{ki}. Although the structure is simple, it is not trivial to deduce λ^0_{ki}, since the exact distribution function of the missed stream $f^0_{ki}(t)$ contains infinitely many terms [40]. Consequently, we cannot deduce the exact miss rate for these objects because of the computational complexity. The only knowledge we have is that the incoming request rate at the *k*th cache is $\lambda_{ki} = l \cdot \lambda_i$, where *l* is a constant and $1 \leq l \leq k$. Let us regard the miss rate

as a function f^0_{ki} of λ_i, then we have $f^0_{ki} = \lambda^0_{ki} = l\lambda_i e^{-l\lambda_i \tau_{ki}}$. Recall that the miss rate of cooperative caching is $f'_i = \lambda' = k\lambda_i e^{-k\lambda_i \tau'_i}$.

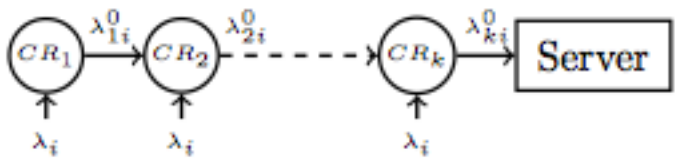


Figure 11 : Caches in tandem

Theorem The maximum miss rate of cooperative caching is less than the tandem of caches working individually.

Proof. We need to prove that, for any $k \in \mathbb{N}^+$ and $C \in \mathbb{N}^+$, we have max(f^0_{ki}) ≤ max(f_i'). The value of τ_{ki} can be calculated as follows:

$$\sum_{j=1, j\neq i}^{N} (1 - e^{-l \cdot \lambda_j \tau_{ki}}) = C \qquad (6)$$

Instead of directly comparing τ_{ki} with τ'_i, we use another variable τ''_i and setup the equation below:

$$\sum_{j=1, j\neq i}^{N} (1 - e^{-k \cdot \lambda_j \tau''_i}) = C \qquad (7)$$

Since $k \geq l$, combining equation (6) and (7) we have $\tau_{ki} \leq \tau''_i$. Applying the same method in the proof of theorem 1 on equation (7) and (2), we can obtain that $\tau''_i \leq \tau'_i$. Thus, we have $\tau_{ki} \leq \tau'_i$.
Let the first deviation of $f^0_{ki} = 0$, that is:

$$le^{-l\lambda_i \tau_{ki}} + l\lambda_i \cdot e^{-l\lambda_i \tau_{ki}} \cdot (-l\tau_{ki}) = 0 \qquad (8)$$

then we have $\lambda_i = (l\tau_{ki})^{-1}$. Since the second derivative of f^0_{ki} is less than zero, we know that max $(f^0_{ki}) = (e \cdot \tau_{ki})^{-1}$. As the same reason, we have max $(f'_i) = (e \cdot \tau'_i)^{-1}$. Since $\tau_{ki} \leq \tau'_i$, we conclude that max(f^0_{ki}) ≥ max(f'_i).

Please note that, the exponential part of f'_i decreases more rapidly than the same part of f^0_{ki}, which means that an approach like ours is expected to have at least the same performances for highly popular videos, and better performances for middle popular videos. As seen in the next Section, experimental results confirm this theoretical analysis.

In conclusion, both tandem and individual models are less efficient than a cooperative policy. Our approach, which combines cooperative and tandem approaches, is hence expected to outperform the classic CCN policy.

3.7 Experimental Results

3.7.1 Simulations on Time-shifted TV

The goal of these simulations is to evaluate the benefits one can expect from the cooperative in-network caching strategy for the time-shifted TV service. We develop our simulator over OMNET++, a simulation framework for communication networks.

3.7.2 Simulation Setup

To build a typical ISP network, we use the real backbone topology measured by *Rocketfuel* [39]. We choose 87 routers, 5 point of presences (POPs) and 161 bidirectional links with latencies from the AS of European Backbone (Ebone). Every POP is connected with one server, which stores all the produced chunks. Chunks are pushed into servers from 6 TV providers with different popularities. We deploy 200 clients uniformly on the access routers locating at the edge of the topology. We reserve 1 gigabytes in each CR to cache time-shifted TV streaming. The basic data unit of the TV streaming is a chunk, which contains the streaming for 1 minute playback. One new chunk is produced every simulation minute by each TV provider. We assume the streaming playback rate is 1 megabits per second, so that the size of one chunk is 7.5 mega-Bytes. Therefore the cache of a CR can store 130 chunks, approximately two hours of video.

We use the same synthetic model as [33] for modeling the behavior of users of time-shifted services. This model is based on two measurement studies conducted in 2008 and 2009 [42,43]. This model includes that a TV stream is divided into programs, associated with a genre. The popularity of programs decreases with time. Moreover, the number of clients varies following a given distribution. In our case, according to different hours in a day, the number of activated clients ranges from 20 to 180. Every client get assigned a role: half of the clients are surfers (watch a same program during 1 or 2 chunks before to switch to another program), 40% of them are viewers (switch after a duration uniformly chosen between 2 and 60 minutes), and only 10% are leavers (stay on a program during a time comprised between 60 and 1000 minutes).

We run our simulation for 9,000 minutes, *i.e.* about one week. Since six TV streams are in the system, 54,000 chunks are produced during the simulation. We measure in particular:

- The caching diversity of the policy by counting the number of distinct chunks that is

stored in the network. The more distinct chunks are stored in the system; the better is the cooperative caching system. With 87 CRs having each a maximum caching capacity of 130 chunks, the maximal caching diversity is 11,310 chunks.

- The ISP-friendliness of the policy by measuring the number of requests that are treated by servers outside the network. The lesser is the number of requests, the friendlier is the caching policy.

3.7.3 Results Analysis

We first investigate the impact of k on the performance of the system. We change k from 1 to 6, where k = 1 is exactly the basic LRU policy. In Figure 12, we show the caching diversity at the end of the simulation. For any k > 2, the system using collaborative cache can keep at least 700 distinct chunks more than the system using basic LRU. The number of distinct chunks keeps increasing although it grows slower after k = 4. When k = 6, the caching diversity reaches 4,500 chunks, that is, the collaborative cache with k = 6 outperforms the basic LRU by almost 60%. As can be expected, the cooperative caching policy increases the caching diversity by avoiding redundant chunk caching.

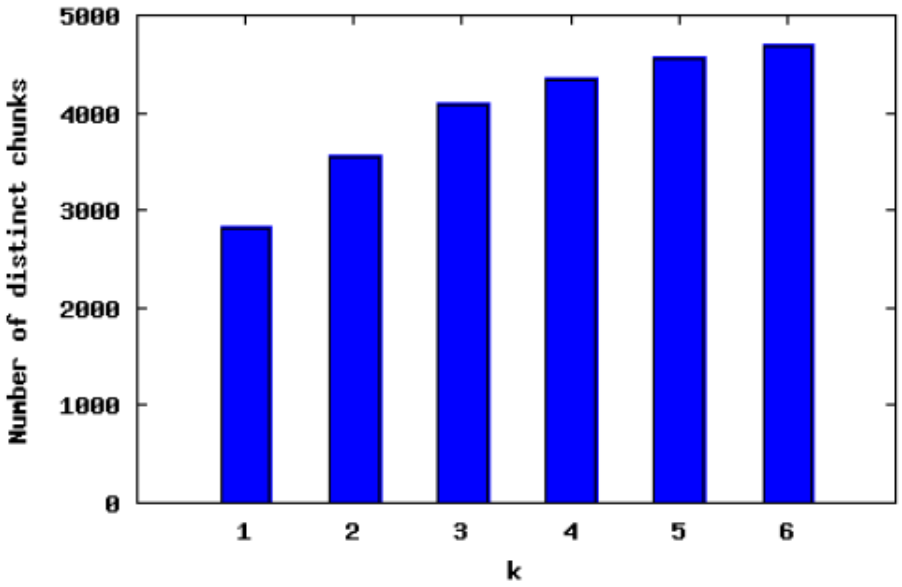


Figure 12 : *Caching diversity*: the number of distinct chunks stored in the set of CRs when the number of labels *k* varies

We demonstrate the efficiency of our proposal in Figure 13, where we compare the ISP-friendliness of the basic LRU policy implemented in CCN to our cooperative caching strategy

with k = 6. In average, every server should upload 20.56 chunks by minute with the basic LRU system, and only 8.92 in our proposal. In other words, the ISP can expect a reduction of around 60% of the cross-domain traffic.

Moreover, we observe that the workload in basic LRU system is not well balanced, with servers 3 and 5 exhibiting two times more traffic than server 4. The workload depends on the network topology: less CRs locate around the POP which is connected with server 4, so fewer requests for the old chunks, which no longer exist in the cache, arrive at server 4. The reverse situation, which happens on server 3 and 5 causes the unbalance of the workload between servers. However, in collaborative cache system, every server sustains approximately the same number of requests. Since most of the chunks for shifted streaming are kept in the collaborative cache, a majority of the requests redirected to servers are the requests for live streaming.

To further study the popularity of chunks stored in the system, we investigate the time interval between last two requests for each cached chunk. This indicates the volatility of content in the cache: the smaller are the time intervals, the more frequent are the read-write operations on the cache. In average, the basic LRU policy has a more intensive usage of the cache. We show the Cumulative Distribution Function of the number of chunks with regard to their time interval in Figure 14. A point at (40, 0.85) means that 85% of the chunks have been accessed at most 40 minutes ago. As can be expected, our cooperative caching policy produces a less intensive caching strategy. On one hand, it means that operations on the disks are less frequent. On the other hand, the content would have higher probability to be removed if ISP were unable to reserve a certain storage space in the cache because unpopular chunks should be replaced by other data.

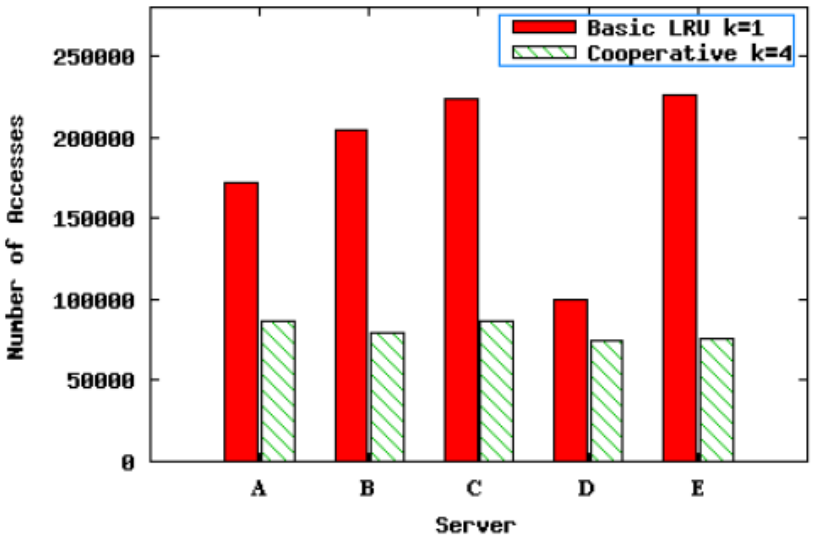


Figure 13 : *ISP-friendliness*: the number of times each server located is accessed. The smaller is the bar, the more ISP-friendly is the caching strategy

To further study the popularity of chunks stored in the system, we investigate the time interval between last two requests for each cached chunk. This indicates the volatility of content in the cache: the smaller are the time intervals, the more frequent are the read-write operations on the cache. In average, the basic LRU policy has a more intensive usage of the cache. We show the Cumulative Distribution Function of the number of chunks with regard to their time interval in Figure 14. A point at (40, 0.85) means that 85% of the chunks have been accessed at most 40 minutes ago. As can be expected, our cooperative caching policy produces a less intensive caching strategy. On one hand, it means that operations on the disks are less frequent. On the other hand, the content would have higher probability to be removed if ISP were unable to reserve a certain storage space in the cache because unpopular chunks should be replaced by other data.

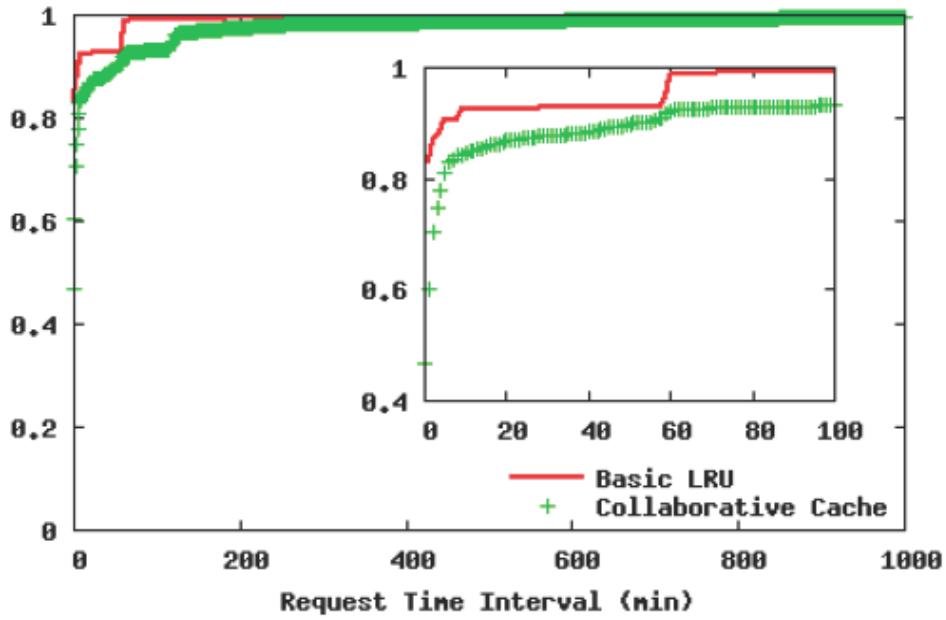


Figure 14 : Cumulative Distribution Function. The y axis is the ration of chunks; the x axis is the time elapsed between two consecutive accesses on a CR.

Finally, in Table 5, we compare the average response time of each request, that is, the round trip time between the sending of a request and the receiving of the corresponding chunk. The response time in collaborative cache is just 40ms more than that in the basic LRU. Thus, our collaborative cache does not cause any significant degradation of the Quality of Experience.

Table 5 : Comparison of Response Time and Requested Time Interval

Cache scheme	*Average response (ms)*
Basic LRU	243.38
Cooperative cache	288.25

3.7.4 Simulations on VoD service

Different from the simulation on time-shifted TV, we implement our cooperative caching policy into the open-source CCNx code and test the augmented daemon on VoD service. For the portability of the program, we have not changed the original cache organization. Instead, we mark the undesirable chunks as stale, so that the chunks with $c \bmod k \neq l$ are eliminated from the cache right after it is forwarded. The label of each router is assigned and published at the beginning of the simulation.

Simulation Setup

Our modified CCNx prototype is deployed on 40 machines with dual 2.70 GHz Pentium processor and 4 GB RAM. Each machine uses Ubuntu 10.04 system and is connected to a switch via 100 Mb/s Ethernet card. The ISP network is still emulated according to Ebone. Every machine works as a router in the network. The routers are interconnected by 80 bidirectional links with negligible delays. Among the 40 routers, 20 of them act as edge routers, with the responsibility to emit the requests from 1000 end users, and 3 routers run as point of presences (POPs). Servers are assumed to be just near these POP routers.

Initially, servers publish all the chunks for the 500 available videos. The size of each video varies uniformly from 60 to 120 chunks. We limit the cache capacity of every router to 100 chunks.

We model the user behavior following the statistic conducted in [38]. More precisely, the number of users to activate and the daily access pattern are based on the formulas given in [38]. Once a client is activated, it chooses a video based on a Zipf's law with the skew factor equal to 1. The duration for each watching session is as follows: 50% of sessions end in 10 minutes, 75% of them stop in 25 minute, 90% of them terminated in 50 minutes, and the rest sessions last until the end of the video. We run our simulation for 10,000 minutes, *i.e.* about one week. Besides the overall chunk diversity and ISP-friendliness, we examine the result for the per-video caching diversity, which is the percentage of chunks (including replicas) belonging to each video that are cached. The increment of chunks of middle popular videos alleviates the server load since more than 40% of requests ask for the 10 most popular videos.

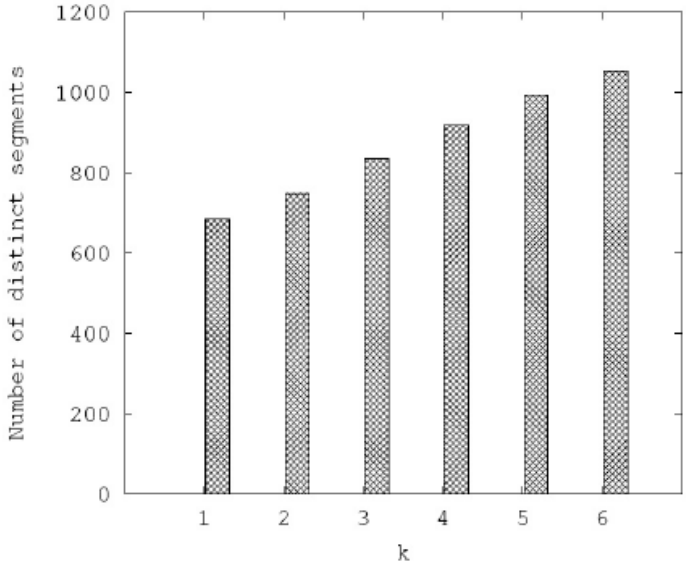


Figure 15 : Caching diversity varies with *k*

In Figure 15, we show the caching diversity at the end of the simulation. Different from the result obtained in time shifted TV system, the diversity augments regularly with the increment of k. When k = 6, the caching diversity reaches 1,050 chunks, that is, the cooperative cache with k = 6 is almost 1.5 times the diversity of the basic LRU policy. As can be expected, the cooperative caching policy increases the caching diversity by avoiding redundant chunk caching.

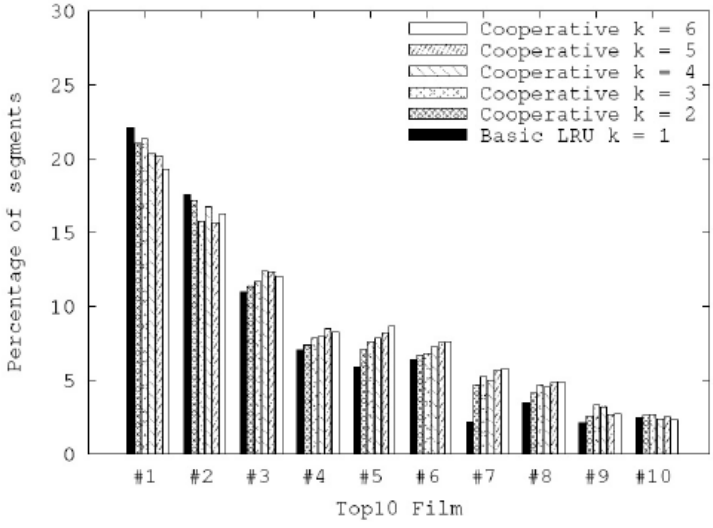


Figure 16 : Chunk distribution of the 10 films

Then we study the per-video caching diversity. We focus on the influence of cooperative caching on the 10 most popular videos that attract a lot of requests. As seen in Figure 16, while the number of stored chunks from the two most popular videos decreases, while it increases from the third to the eighth most popular videos. That is, the aforementioned higher diversity focuses on the chunks from these middle-popular videos, frequently accessed, but not necessarily considered as blockbusters. For the long tail, as shown for the ninth and tenth most popular video, the caching policies have approximately the same behavior. These experimental results are consistent with our theoretical analysis in the previous section.

Finally, we highlight the efficiency of our proposal in Figure 17, where we compare the ISP-friendliness of the basic LRU policy implemented in CCN to our cooperative caching strategy. In average, every server should upload about 150 chunks by minute with the basic LRU system, and only 52 chunks in our proposal with k=6. In other words, the ISP can expect a reduction of more than 60% of the cross-domain traffic.

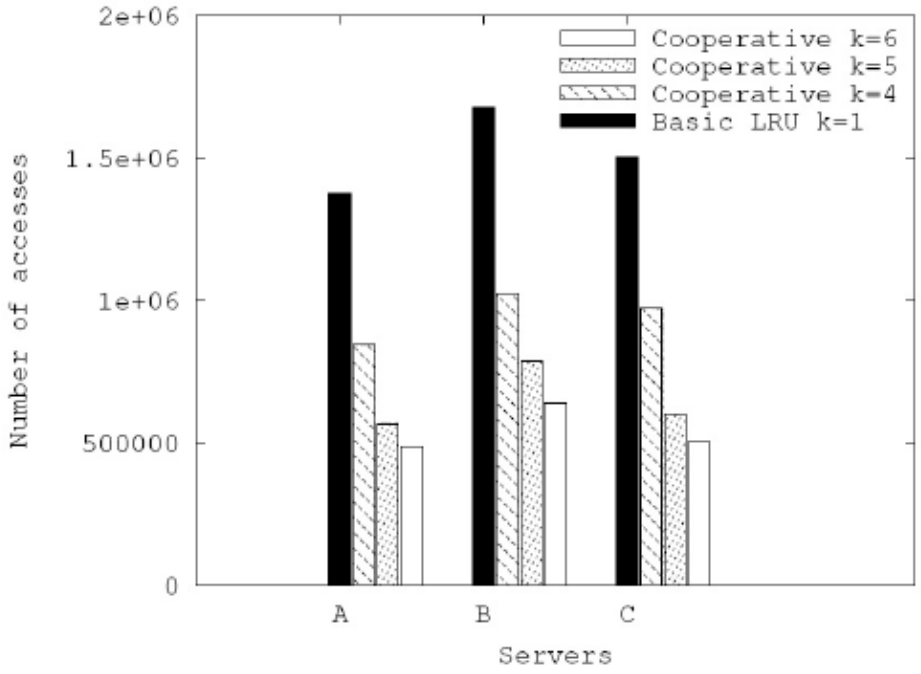


Figure 17 : Number of times each server is accessed

3.8 Conclusion

This section of the deliverable focuses on the impact of a novel cooperative caching policy on the benefit of an ISP. The ISP is supposed to use CCN infrastructure to deliver video streams. In the basic CCN design, the simple LRU or LFU policy is propose to manage the storage space. We show by our experimental results that our cooperative caching strategy can reduce 60% the requests that go out of the ISP comparing with the original CCN. So that the

proposed cooperative caching significantly save the cost for the ISP offering both time shifted TV and VoD service.

Our future work is to further improve the augmented CCNx daemon, try to integrate different caching policy (*i.e*. LFU, k-LRU) into our cooperative caching and compare the performance. On the other hand, deeper theoretical study on the multi-caching system is also interesting.

4 Real Datasets for prefetching and simulation

The peer-assisted network has to optimize the offer of peer-cached contents to face the demand. In the previous sections, we discussed the CYCLOPS approach, where a server feeds peers with contents, and where a monitoring mechanism controls its bandwidth contribution to the peers so as to minimize a cost without sacrificing performance.

We also propose a CCN+ infrastructure to deliver video streams and a cooperative caching approach. The preliminary experimental results are encouraging. We plan to integrate different caching replacement policies such as LRU, k-LRU.

For both approaches, we may introduce content-oriented caching strategies. For the first one, a question could be how to feed the swarm, which contents choose? For the second one, we may envision hybrid caching policies combining passive caching policies with prefetching strategies. LRU-like passive replacements are efficient to manage contents [44], especially popular ones. But for less popular contents, external access to the server is more frequently needed. A solution to address this issue is the prefetching of contents in extra caches [45-47].

We are indeed working on techniques to pre-load in the peer-assisted network the contents that are to be downloaded, and then reduce the requests that go out of the ISP domain.

In order to provide prefetching capabilities, we need to analyze users' behavior. The richer the learning dataset is, the more accurate the predictions of future behavior will be. Such rich dataset will allow us to compute similarities between VoDs, predict future downloads and then feed extra-caches.

We describe in this section the datasets we plan to use to compute prefetching on the one hand, and to test our peer-assisted networks on the other.

4.1 A real VoD dataset

For CCN simulations described in the previous section, we model the user behavior following the statistic conducted in [38]. We have now the opportunity to test on a real dataset. A French ISP provides us an extraction of the VoD downloading history from a regional zone. The commercial VoD service is legal and comes with the ISP offering.

Let $Data_{ISP}$ denote this dataset. For each download, the logs give the timestamp, the user ID and the film ID (Figure 18). 8,935 customers downloaded 108,108 VoD during 6 weeks (February-March 2010). 5,777 different movies were requested.

1265385351 c3794 f5729
1265385362 c4058 f3344
1265385384 c1128 f1971
1265385404 c4687 f121
1265385432 c1483 f2390

Figure 18 : ISP VoD downloading logs: timestamp, user ID, film ID

We detail below some aspects of the data.

Downloaders' profiles :

Figure 19 and Figure 20 depict the downloading behavior during the weeks and the week-ends. With no surprise, we see low activity during the night (1h-7h), with a slow increase during the day and a peak around 21h. This peak is higher for a mean weekday evening than for a mean week-end evening. But we observe the highest peaks on Saturday and Friday evenings. During the week-end afternoons, especially on Sundays, the graph shows more activity than during the week.

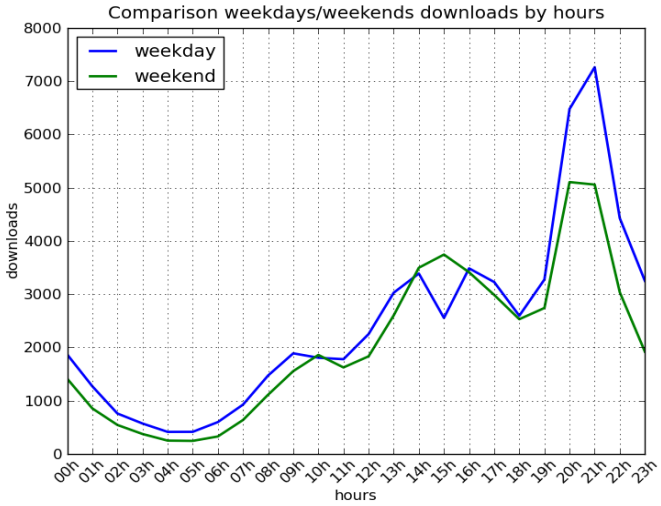


Figure 19 : Downloads during the week-end or the week.

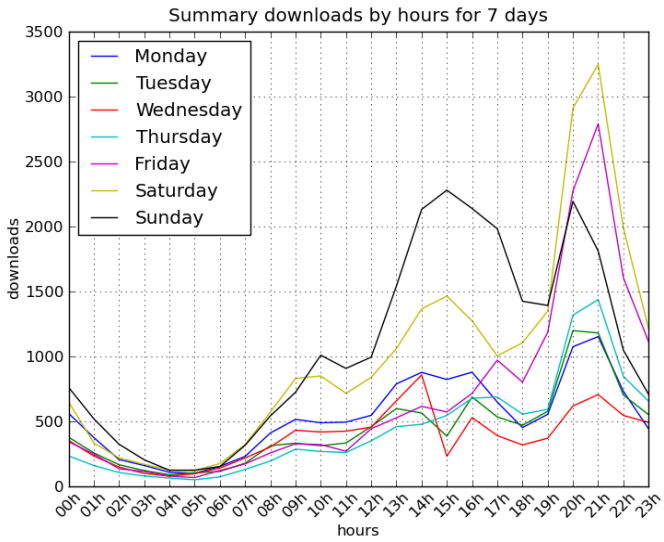


Figure 20 : Downloads each different day.

We found 3 classes of downloaders: the "Top" ones who request 40 to 230 movies per month. 5.6% of all the users are "Top" users. The mean number of downloads is 66 movies per month, or something like 2 movies per day.

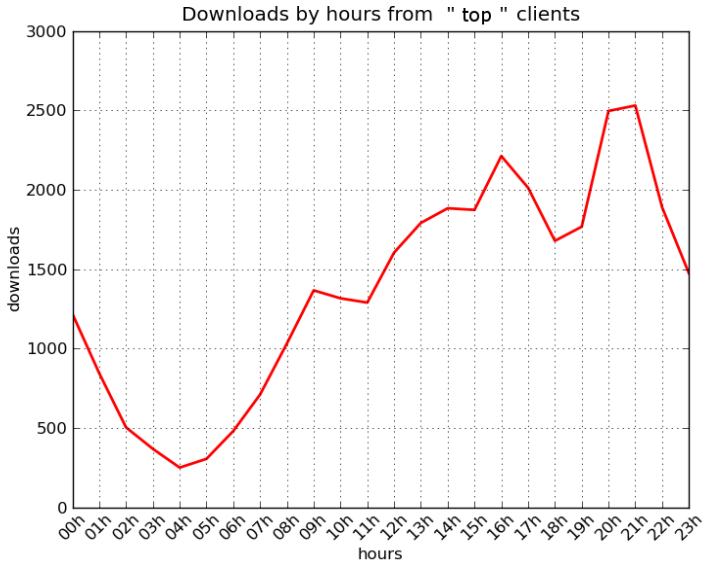


Figure 21 : The "Top" users who download the most.

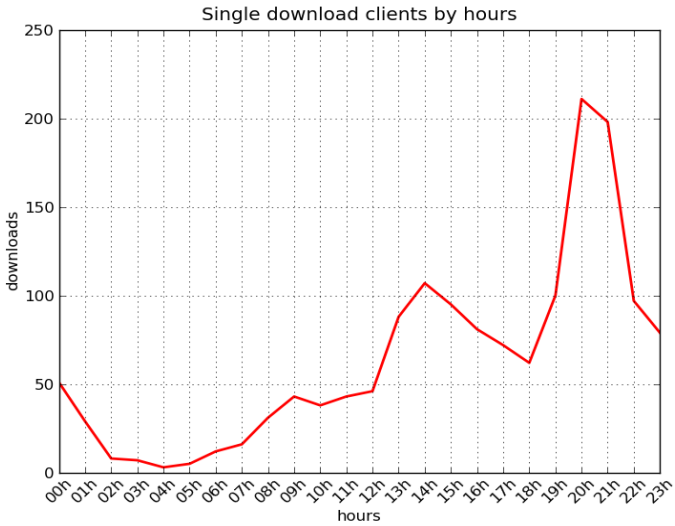


Figure 22 : Downloading profile of users who downloaded only once during the 6 weeks of $Data_{ISP}$.

Figure 21 and Figure 22 compare the high consumers of VoD and the ones who downloaded only once in our dataset. The bahavior is the same as the global one described above. Only a remarquable difference, in addition to the volume, is that the "Top" clients download significatively more in the afternoon.

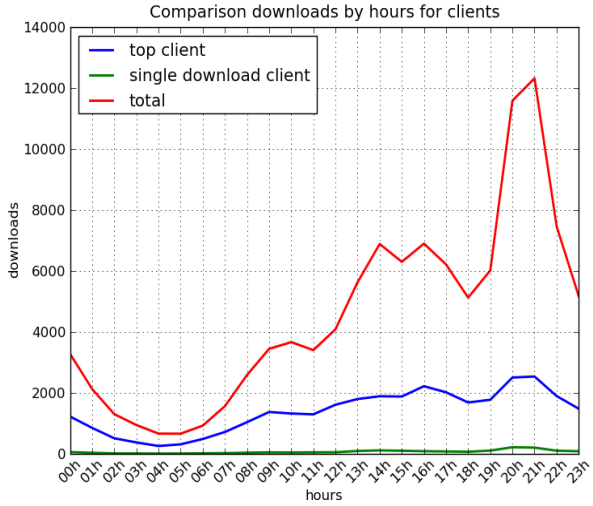


Figure 23 : Comparison between the downloading profiles.

On the same scale of the Figure 23, we observed that "single download" users are very insignificant in volume of data requested. Note that 64% of the films they requested are very popular films, only 2% of the films they resquested are films downloaded once.

4.2 A dataset for prefetching

Let $Data_{WEB}$ denote a dataset that comes from Flixster[2], a social web site where people share their opinion about movies. We collected in july 2009 data where 201,107 users annotated 38,656 different movies with 6,028,491 rates: 0 if they dislike; 5 if they like it.

$Data_{WEB}$ is very interesting to study recommendation mechanisms. As the users rate all the movies, we can conceive and test algorithms to predict rates on movies. Traditionaly, such ratings are used by collaborative filtering methods [48]. Similarities of users are computed based on their rating behavior (e.g. Pearson correlation). Then rates are predicted on unrated movies, given the rates provided by similar users.

Recommendation domain is very well studied, and the recent NetFlix challenge[3] has brought very efficient methods. But there are still scientific issues, especially regarding the non-popular contents for which the precision/recall highly decreases. Such movies indeed are not rated enough for an algorithm to build relevant similarities between users. We are interested in this long tail issue (French RNTI revue chapter to be published soon).

We will propose later in the project a method to fill the extra caches dedicated to prefetching.

4.3 Merging the datasets

4.3.1 Popularity classification

In both $Data_{ISP}$ and $Data_{WEB}$, 65% of the requests concern the most 500 popular movies. The VoD downloaded only once during the 6 weeks of logs represent 25% of all the requests in $Data_{ISP}$. Similarly, 30% of the movies have been rated once in $Data_{WEB}$. Note that the volume of downloads of films requested once is negligible comparing with the popular ones (cf. Figure 24). We also observed similar power law distributions in movies popularity in both $Data_{ISP}$ and $Data_{WEB}$ (see Figure 24).

2 http://www.flixster.com

3 Netflix, Inc. organized a challenge and offered $1,000,000 to the winners who achieved a gain of 10% accuracy (RMSE measure) in 2009.

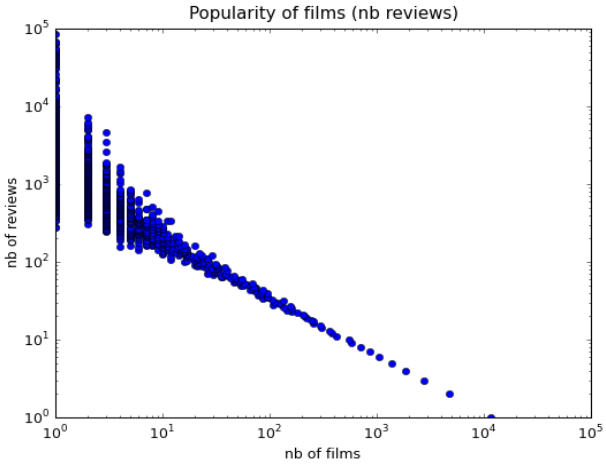


Figure 24 : Power distribution of ratings in $Data_{WEB}$.

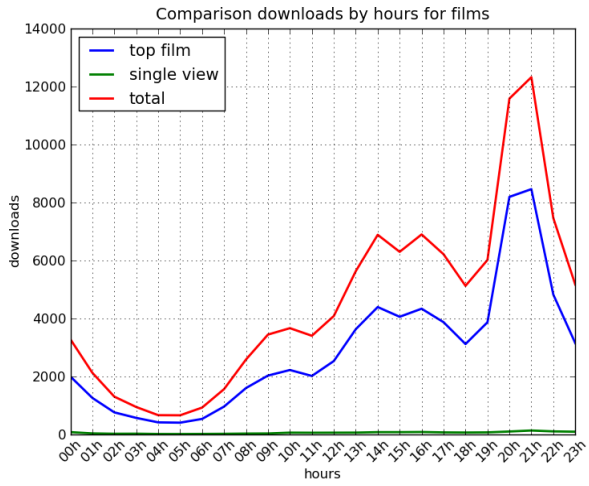


Figure 25 : $Data_{ISP}$ types of films: popular films and films downloaded once. The "Mid-popular" films are the delta between the global red line and the blue one.

Let us consider the Popularity classification in 3 classes: "Top 500" films, "Mid-popular" and "Single view" films. This classification fits both real datasets.

4.3.2 Merging method and resulting dataset

As prefetching is facilitated by work on $Data_{WEB}$, and both datasets are very similar regarding the popularity distribution and classes of movies, we propose to merge the 2 sets. The idea is to generate downloading logs from the rated movies from $Data_{WEB}$ with the downloading behavior observed in $Data_{ISP}$. The matching of movies is based on their popularity. In the first

case, the popularity is the relative number of ratings, whereas in the second case, the popularity is the relative number of downloads (cf. Figure 26).

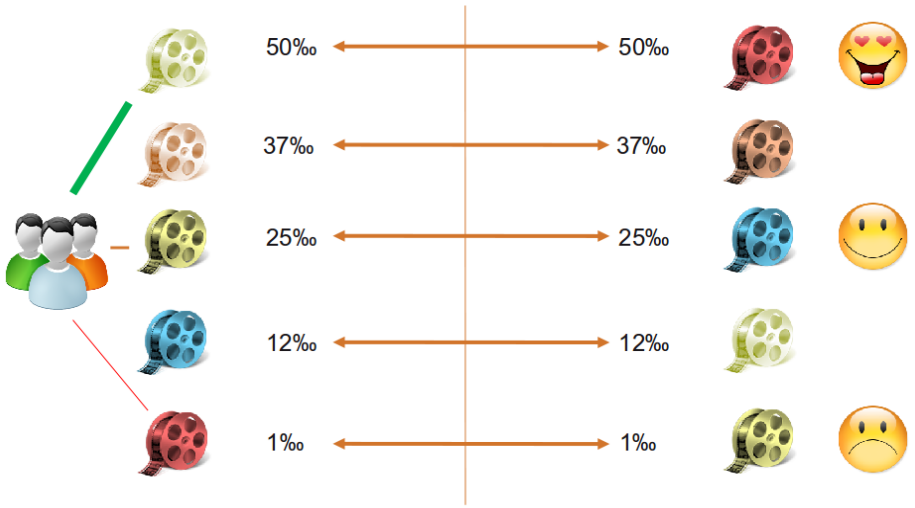


Figure 26 : Popularity merging between ISP downloaded films and Web annotated films.

For each movie from $Data_{WEB}$, we find the film from $Data_{ISP}$ with the closest popularity. In case of conflict the correspondance is chosen randomly. In the next step, downloading history is assigned. As several ISP films may be attached to one Web film, we introduce temporal disorder to delay concomitant requests (random delay in [-20s, 20s]).

As a result, we get a new dataset $Data_{ISP+WEB}$ where, during 6 weeks, users download movies for which we know whether they were appreciated or not. As a matter of fact, this new dataset "only" contains 144,296 requests (to be compared with the 6,000,000 entries of $Data_{WEB}$, Figure 27). The 38,500 web films now match the 5,800, but 30% of the 40,000 are films with only one rate, generating only one line in the final log file. The first 500 web top films generate the same amount of logs, leaving 12,300 web films to be matched with 3,800 mid-popular films with quite low download volume. As we see in the Figure 28, the generated data shows similar downloading behavior than the original one shown in Figure 19.

	Nb users	Nb films	nb downloads	nb ratings
$Data_{ISP}$	8,935	5,777	108,108	
$Data_{WEB}$	201,107	38,656		6,028,491
$Data_{ISP+WEB}$	201,107	38,656	144,296	6,028,491

Figure 27 : Two real datasets and our generated dataset.

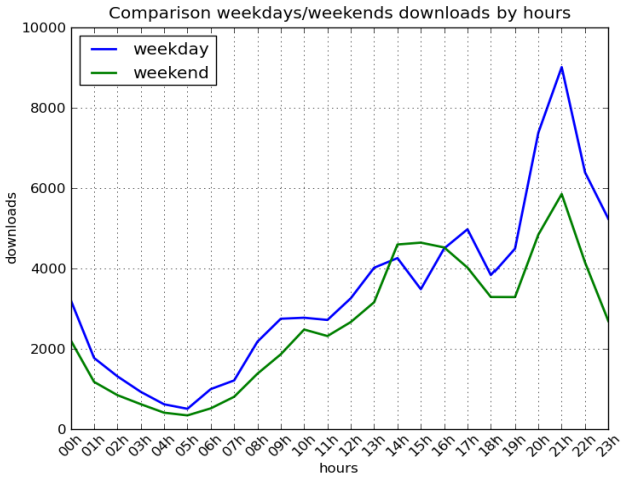


Figure 28 : Weekday and week-end downloads in the generated dataset.

4.4 Conclusion

In the previous section about CCN, the simulations were run on simulated data, reproducing about one week with 1,000 end users and 500 available videos ($Data_{SIM}$). It would be interesting to test on $Data_{ISP}$ and challenge a larger volume of data, with more popular movies (500 instead of 10).

$Data_{SIM}$ simulates more than the downloading. It introduces the viewing duration of content. Unfortunately, the real datasets described in this document do not contain this information. But we may for example consider that a bad rate in $Data_{ISP+WEB}$ means a short viewing duration.

We are aware that the $Data_{ISP+WEB}$ we using are not easily obtained. Hopefully, future VoD services will be enhanced with Web2.0 functionalities, which automatically yield this type of logs.

In the next steps of the project, we will propose prefetchning mechanisms, and simulate the datasets described in this document.

References

[1] "http://www.akamai.com."
[2] "http://aws.amazon.com."
[3] "http://www.limelightnetworks.com."
[4] "http://www.bittorrent.com."
[5] F. Bin, D.-M. Chiu, and J. C. Lui, "Stochastic analysis and file availability enhancement for bt-like file sharing systems," in *Proc. of IEEE IWQoS*, 2006.
[6] R. S. Peterson and E. G. Sirer, "Antfarm: Efficient content distribution with managed swarms," in *Proc. of USENIX NSDI*, 2009.
[7] D. R. Choffnes and F. E. Bustamante, "Taming the torrent: A practical approach to reducing cross-isp traffic in p2p systems," in *Proc. of ACM SIGCOMM*, 2008.
[8] R. Cuevas, N. Laoutaris, X. Yang, G. Siganos, and P. Rodriguez, "Deep diving into bittorrent locality," arxiv.org/abs/0907.3874, Telefonica Research, Tech. Rep., 2009.
[9] "http://www.eurecom.fr/ albanese/pcdn.html."
[10] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *Proc. of ACM STOC*, 1997.
[11] "http://en.wikipedia.org/wiki/BitTorrent\s\do5(()protocol)."
[12] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke are enough," in *Proc. of ACM IMC*, 2006.
[13] M. Steiner, E. W. Biersack, and T. En-Najjary, "Exploiting kad: Possible uses and misuses," *Computer Communication Review*, vol. 37, no. 5, 2007.
[14] C. Huang, J. Li, A. Wang, and K. Ross, "Understanding hybrid cdn-p2p: Why limelight needs its own red swoosh," in *Proc. of ACM NOSSDAV*, 2008.
[15] C. Huang, J. Li, and K. Ross, "Can internet vod be profitable? " in *Proc. of ACM SIGCOMM*, 2007.
[16] B. Sanderson and D. Zappala, "Reducing source load in bittorrent," in *Proc. of IEEE ICCCN*, 2009.
[17] Z. Chen, Y. Chen, C. Lin, V. Nivargi, and P. Cao, "Experimental analysis of super-seeding in bittorrent," in *Proc. of IEEE ICC*, 2008.
[18] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a bittorrent networks performance mechanisms," in *Proc. of IEEE INFOCOM*, 2006.
[19] R. Kumar and K. Ross, "Optimal Peer-Assisted File Distribution: Single and Multi-Class Problems," in *Proc. of IEEE HOTWEB*, 2006.
[20] R. Sweha, A. Bestavros, and J. Byers, "Angels – in-network support for minimum distribution time in p2p overlays," Boston University, Tech. Rep. BUCS-TR-2009-003, 2009.
[21] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *Proc. of ACM SIGCOMM*, 2004.
[22] N. Laoutaris, D. Carra, and P. Michardi, "Uplink allocation beyond choke/unchoke or how to divide and conquer best," in *Proc. of ACM CONEXT*, 2008.

[23] U. Lee, I. Rimac, and V. Hilt, “Greening the internet with content-centric networking," in International Conference on Energy-Effcient Computing and Networking, 2010.
[24] T. Leighton, “Improving performance on the internet," Communications of the ACM, vol. 52, no. 2, pp. 44-51, 2009.
[25] E. Nygren, R. K. Sitaraman, and J. Sun, “The akamai network: a platform for high-performance internet applications," ACM SIGOPS Operating Systems Review, vol. 44, no. 3, pp. 2-19, 2010.
[26] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, “Networking named content," in Proc. of the Int'l Conf on emerging networking expe. and tech. (CoNEXT), 2009.
[27] The Cisco Visual Networking (VNI) Forecast 2009-2014, Cisco, June 2010. 18
[28] J. Zhuo, J. Li, G. Wu, and S. Xu, “E_cient cache placement scheme for clustered time-shifted TV servers," IEEE Transactions on Consumer Electronics, vol. 54, no. 4, pp. 1947-1955, November 2008.
[29] T.Wauters, W. de Meerssche, F. Turch, B. Dhoedt, P.Demeester, T. Caenegem, and E.Six, “Co-operative proxy caching algorithms for time-shifted iptv services," in IEEE Computer Society Washington, 2006.
[30] J.-C. Zhuo, J. Li, G. Wu, and L.-Y. Zhu, “A novel data replication and placement scheme for time-shifted tv cluster," in International Conference on Computer Science and Software Engineering, 2008.
[31] F. V. Hecht, T. Bocek, C. Morariu, D. Hausheer, and B. Stiller, “LiveShift: Peer-to-Peer Live Streaming with Distributed Time-Shifting," in Proc. Of 8th Int. P2P Conf., 2008, pp. 187-188.
[32] D. Gallo, C. Miers, V. Coroama, T. Carvalho, V. Souza, and P. Karlsson, “A Multimedia Delivery Architecture for IPTV with P2P-Based Time-Shift Support," in Proc. of 6th IEEE CCNC, 2009, pp. 1-2.
[33] Y. Liu and G. Simon, “Distributed Delivery System for Time-Shifted Streaming System," in 35th IEEE Conf. on Local Computer Networks (LCN), 2010.
[34] E. Nygren, R. K. Sitaraman, and J. Sun, “The akamai network: a platform for high-performance internet applications," SIGOPS Oper. Syst. Rev., vol. 44, pp. 2-19, August 2010.
[35] Y. Liu, Y. Guo, and C. Liang, “A survey on peer-to-peer video streaming systems," Peer-to-Peer Networking and Applications, vol. 1, no. 1, 2008.
[36] H. Xie, A. Krishnamurthy, A. Silberschatz, and Y. Yang, “P4P: Explicit Communications for Cooperative Control Between P2P and Network Providers," P4PWG Whitepaper, May, 2008.
[37] Y. Chen, J. Leblet, and G. Simon, “On reducing the cross-domain traffic of box-powered cdn," in Proc. of IEEE ICCCN, 2009.
[38] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng, “Understanding user behavior in large-scale video-on-demand systems," SIGOPS Oper. Syst. Rev., vol. 40, no. 4, 2006.
[39] N. Spring, R. Mahajan, and D. Wetherall, “Measuring isp topologies with rocketfuel," in SIGCOMM, 2002.

[40] H. Che, Y. Tung, and Z. Wang, “Hierarchical web caching systems: Modeling design and experimental results," IEEE Journal on Selected Areas in Communications, vol. 20, no. 7, 2002.
[41] E. J. Rosensweig, J. Kurose, and D. Towsley, “Approximate models for general cache networks," in IEEE INFOCOM, 2010.
[42] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatrianin, “Watching television over an ip network," in Proc. of Usenix/ACM SIGCOMM Internet Measurement Conference (IMC), 2008.
[43] Nielsen, “How DVRs Are Changing the Television Landscape," Nielsen Company, Tech. Rep., April 2009.
[44] J. Wu and B. Li, “Keep Cache Replacement Simple in Peer-Assisted VoD Systems,” in IEEE INFOCOM 2009 - The 28th Conference on Computer Communications, pp. 2591–2595, IEEE, Apr. 2009.
[45] C. Huang, J. Li, and K. W. Ross, “Peer-assisted vod: Making internet video distribution cheap,” in Processding of IPTPS, 2007.
[46] B. Wu and A. D.Kshemkalyani, “Objective-optimal algorithms for long-term web prefetching,” IEEE Transactions on Computers, vol. 55, no. 1, 2006.
[48] Y. He, G. Shen, Y. Xiong, and L. Guang, “Optimal prefetching scheme in p2p-vod applications with guided seeks,” IEEE Transactions on Multimedia, vol. 11, no. 1, 2009.
[49] P. Resnick and H. R. Varian, “Recommender systems - introduction to the special section,” Commun. ACM, vol. 40, no. 3, pp. 56–58, 1997.