enhanced COntent distribUtion with Social INformation
www.ict-ecousin.eu

Deliverable D4.2

Final Report and Initial Software Release of the Design Extensions and Preliminary Implementation

Public deliverable, Version 1.0, 30 April 2014

Authors

FT	Ali Gouta, Yannick Le Louédec
A-LBELL	Danny De Vleeschauwer **(editor)**, Chris Hawinkel
IMDEA	Foivos Michelinakis, Nicola Bui, Joerg Widmer
TSP	Angel Cuevas**,** Reza Farahbakhsh, Noel Crespi
ALUD	
TUD	Julius Rückert, Christian Koch, Timo Thräm, David Hausheer
TI	
UCAM	
UC3M	Ruben Cuevas, Juan Miguel Carrascosa
Reviewers	Angel Cuevas, Chris Hawinkel

Abstract

The present deliverable D4.2 is the second deliverable of Work Package 4. This document first briefly recaps the status of the work reported in deliverable D4.1 and then concentrates on new results obtained in the first half of the second project year. The new contributions fall in three categories. First, some statistical characteristics of dataset acquired in WP3 are described, only those results that are required to understand the performance results later described are mentioned. Second, software components are detailed. Some software components are simulators used to assess the performance of the proposed algorithms. Others are (parts of) applications that will be ultimately transferred to WP6 to be incorporated in a larger demonstrator. Third, thorough performance studies of the algorithms studied in WP4 were conducted and their results are reported. These pertain to pre-fetching, i.e., algorithms for content consumption prediction and algorithms for predicting the available bandwidth (to determine the optimal time for downloading), to caching strategies for various types of caches (e.g., small Telco caches and large caches in data centres) and to peer-to-peer content distribution. All these algorithms are characterised by the fact that implicit or explicit social or geographical information is used in the decision processes of the algorithms. The performance results will give guidelines on how to tune the software components when they are transferred to be incorporated in the demonstrators of WP6.

Project funded by the European Union under the
Information and Communication Technologies FP7 Cooperation Programme
Grant Agreement number 318398

EXECUTIVE SUMMARY

This deliverable establishes an interim step towards the completion of the work committed in WP4. Starting from the work described in deliverable D4.1 [D4.1] we present extensions and new elements that show the progress of the work within WP4. The contributions of this deliverable D4.2 can be divided into four different areas.

First, we have completed the design of the main components related to content delivery. We present in this deliverable a comprehensive description of the main elements related to: pre-fetching, caching and resource allocation. For pre-fetching, we discuss a pre-fetching architecture for mobile networks and pre-fetching strategies for content consumption. In terms of caching, we present different caching algorithms to be used in CDNs (Content Distribution Networks) in different contexts (i.e., OSN (Online Social Network) content consumption, content distribution via small TELCO (TElecommunication COmpany) CDN). Finally, we also discuss a resource allocation algorithm that takes into account different parameters.

Second, in order to evaluate the performance of the algorithms integrated within the different components we have developed accurate simulation tools. We have developed a simulator referred to as “PrefSim” to assess the performance of the pre-fetching solution. In addition, we have used extensive simulations to understand the impact of different caching strategies on the distribution of social content through Twitter.

Third, based on the previous simulators we have performed a thorough evaluation of all the algorithms described in this deliverable. We have evaluated the performance of pre-fetching algorithms in terms of content consumption, cost reduction, buffer size and robustness. In addition, as stated in the previous paragraph, we have evaluated the performance of different caching strategies to disseminate content through CDNs in one of the most popular OSNs, i.e., Twitter. This performance evaluation is extended with real data measurements to test the performance of a commercial CDN such as Akamai to serve Facebook content. In addition, we also use real dataset to evaluate caching algorithms in small TELCO CDN.

Fourth, we present an initial version of software tools for two particular aspects. We first provide a mobile tracing application for enhancing end-users experience through pre-fetching content of interest to them. This application relies on Facebook to exploit user social information to predict the interest of a given end user. In addition, we present preliminary software for the “Peer-to-Peer-Social-Assisted Content Delivery” use case introduced in D4.1 [D4.1]. This software also relies on Facebook and it is implemented through a Facebook application.

These results show that the studied algorithms (which make use of social and location information) are useful in content distribution. The software components and simulation results together with guidelines how to optimally tune the components will be transferred to WP6.

TABLE OF CONTENTS

INTRODUCTION

Following deliverable D4.1 [D4.1], which reported the initial design of technical solutions in content placement and delivery, the present deliverable D4.2 continues the work carried out during the first year of the project towards providing the final design of content placement and delivery building blocks as well as measuring their performance, either through simulation or via experiments on test beds. In addition, this deliverable D4.2 includes some initial pieces of software associated with WP4, which will be used in the demonstrators of WP6.

To keep this document self-contained we first start by briefly summarizing the main contributions presented in deliverable D4.1 [D4.1] to set up the starting point of this deliverable D4.2. In addition, we introduce the data loads that will be used in different elements described in this deliverable D4.2. These data loads are actually extracted from some of the data sets and traces described in deliverable D3.1 [D3.1] that serve as input (after a processing phase) for testing some of the algorithms introduced in this deliverable D4.2.

We then describe in detail the components of the "Content Dissemination Layer" (first introduced in the context of use cases in deliverable D2.1 [D2.1] and further detailed in the architecture of deliverable D2.2 [D2.2]) that will be further used in the demonstrators of WP6. In particular the components described in this deliverable D4.2 are: a pre-fetching architecture for mobile networks and an algorithm for network resource allocation. For the former we also present a mobile application that implements a social predictor to infer the content that should be pre-fetched to the end user device. Related to these components, we describe a "pre-fetching simulator", referred to as "Prefsim", developed in eCOUSIN.

Next we assess the performance of the caching components of the "Content Dissemination Layer" with a clear focus on CDNs (Content Delivery Networks) for OSN (Online Social Network) content. We first analyze the performance offered by a commercial CDN. Subsequently, we present the comparison for a set of caching strategies in order to populate caches for disseminating content shared through OSNs. We also thoroughly assess the performance of an extension of a known caching algorithm that exploits the idea of using lead users (previously studied in deliverable D3.1 [D3.1]) for caching in the context of Small TELCO (TElecommuncation COmpany) CDN caches.

In addition to caching, we also present and evaluate a pre-fetching algorithm through a couple of real datasets using four different performance metrics. This solution is evaluated in terms of its capacity to predict content consumption under different circumstances.

Finally, the last part of the deliverable is dedicated to peer-to-peer solutions. First, we describe various algorithms for community detection and compare their performance. In addition, we present an initial software implementation of the P2P (Peer-to-Peer) social-assisted solution introduced in deliverable D4.1 [D4.1].

We can summarize the work included in this deliverable in four main points:

- Final design of content delivery components;
- Development of simulation software tools to carry out a suitable performance evaluation of different components;
- Thorough performance evaluation of caching, pre-fetching and resource allocation algorithms;
- Initial version of software related to core tasks of WP4 including pre-fetching mobile application and P2P social-assisted content delivery.

The work contained in this deliverable D4.2 establishes an interim progress towards the final implementation of technical solutions for “Content Dissemination Layer” of the eCOUSIN architecture described in deliverable D2.2 [D2.2].

1. SUMMARY OF DELIVERABLE D4.1

1.1 Content dissemination layer in the WP2 architecture

In deliverable D2.2 [D2.2] the eCOUSIN architecture is detailed, one of the layers is the “Content Dissemination Layer”. Broadly speaking this layer takes past content consumption and explicit and implicit social relations (e.g., in an OSN) and network status extracted from the social and network layer respectively, as input and gives advice on where to place, store or cache content items. Some initial building blocks in this layer were introduced in deliverable D4.1 [D4.1] where we also provided an initial performance analysis of them. The current deliverable D4.2 provides more details for these building blocks and presents a thorough performance analysis of them.

1.2 Network topology

Realistic network topologies were discussed in deliverable D4.1 [D4.1]. We refer the interested reader to that deliverable for more information on the wireline and wireless network (WLANs (Wireless Local Area Networks) and cellular networks) that are taking into consideration in the eCOUSIN project.

1.3 Building blocks discussed in D4.1

1.3.1 Popularity prediction using social information

In deliverable D3.1 [D3.1] we described a method to identify leading users, while in deliverable D4.1 [D4.1] we introduced a method to exploit social relations between users and user groups. This idea is further explored in Section 2.3.2.1 where a modified version of the LRU (Least Recently Used) caching strategy is adapted so that it can take the fact that there are lead users and followers into account.

1.3.2 Pre-fetching

The pre-fetching concept was introduced in deliverable D4.1 [D4.1]. It described how historical content consumption and social links between users can be considered to predict upfront which user is going to consume which content item in the near future. This knowledge together with the knowledge of the status of the network (see Section 1.3.2.2) is used to transport relevant content items closer (even on the user device) before the user requests it.

Additionally in deliverable D4.1 [D4.1] we discussed related work and identified potential social properties of content items shared over OSNs that could be used for content prediction and, thus base pre-fetching decisions on them. Furthermore, the basic architecture of the mobile pre-fetching application was presented. It consisted of two main building blocks, the “Pre-fetching App” and the “Crawler”. These building blocks were further studied in the meantime and were the blueprint of the “Content Pre-fetcher” and the “Social Data Collector and Manager” components that are described in detail in Section 2.2.1.1.

1.3.2.1 Prediction of content consumption

To study how well content can be predicted based on past content we have developed a simulator, referred to as “Prefsim”, which is described in detail in Section 2.2.3. With this simulator we assess the performance of various prediction algorithms in Section 2.3.3.1.

1.3.2.2 Network resource allocation optimization

1.3.2.2.1 Optimization under perfect information

The scope of our work at this stage is to investigate the challenges and their possible solutions of applying a theoretical model of optimized stored video delivery to an LTE (Long Term Evolution) network. Our work combines predictions of future content requests and of wireless capacity, in order to optimize the delivery of video content, in a transparent way to the user and requires as few radio resources as possible. In that manner the number of users that can be served in a cell and the quality of service that they enjoy increases. Currently, we have implemented a first working version of our practical model which builds upon a purely theoretical approach presented in the state of the art. We have tested our approach through NS-3 simulations and the results show that our solution achieves a good balance between robustness and low channel utilization, while it significantly reduces the cost compared to the benchmark application.

The core concept of our model is to deliver stored video by transmitting data only when the capacity of the user is above a certain threshold and rely on the buffer otherwise. This objective can be mathematically expressed as the minimization of $\frac{1}{T}\int_0^\infty \frac{r(t)}{c(t)}dt$, where c(t) is the capacity of the user, r(t) is the allocated rate to that user by the algorithm and T is the duration of the video. It is further assumed that $r(t) = 0 \ \forall \ t > T$, thus the integral is finite. Usually, a user enjoys high capacity and has good signal reception. The amount of radio resources required to send the same amount of data under good signal conditions is significantly lower, compared to the bad ones, because the UE (User Equipment) can make use of an MCS (Modulation and Coding Scheme) that is more efficient (i.e., more useful bits can be sent, because less (parity) bits need to be spent to correct bit errors). The abundant data sent during the good channel conditions are stored in the buffer of the video application, so the main constraint of this approach is the finite size of the buffer. The theoretical foundations of our work, regarding optimization under perfect information can be found in [VECI13].

1.3.2.2.2 Optimization under uncertainties

An optimal resource allocation depends on the capability of estimating the throughput that a user is going to experience in the future. However, as it can be seen in deliverable D5.2 [D5.2], where the state of the art of bandwidth and throughput predictors are analyzed, a perfect knowledge of the future is not possible and degrades the further in future the forecast is made.

Thus, in order to provide a solution able to leverage on future forecast and, at the same time, is robust against prediction uncertainties (see Section 2.2.2), we will be studying the trade off between buffer under-run risk and cost optimality in Section 2.3.3.2.

For the time being a simple iterative solution is given: the bandwidth allocation is performed on the prediction without accounting for uncertainties and the allocation is then updated after a few steps when better future estimations are available.

However, this solution requires to reiterating the optimization process very often and might be computationally complex for constrained devices, so that there is still room for improving the algorithm in order to reduce the computational complexity and improve its robustness.

1.3.3 Exploiting Social Relationship and Personal Background for Content Discovery in P2P Networks

In deliverable D4.1 [D4.1] we presented an exhaustive work presenting the benefits that the use of social information had for content look-up in unstructured P2P networks. Our work proposes a network model in which neighbours in the P2P network happen to be actual friends in an OSN. The reason behind it is that using friends for look-up operations increases the efficiency of the content search procedure. Furthermore, we proposed a neighbour selection algorithm that only chooses the K friends with higher probabilities of successfully resolving the look-up process to receive the source node look-up queries. Those K neighbours were selected based on knowledge and similarity (with respect to the source node) features. The obtained results demonstrated that involving social information in content look-up provides an important improvement in the context of P2P networks.

1.3.4 P2P-Social-Assisted Content Delivery

Deliverable D4.1 [D4.1] presented an incipient idea that proposes a P2P-social-assisted content delivery to offload the cost associated to the delivery of low-popular content (e.g., UGC (User Generated Content)) through standard solutions like CDNs. The idea was based on the following reasoning. A given user U sharing a low-popular video (i.e., a UGC videos) through an OSN will have as effect that only some of her friends will consume that video. For standard delivery solutions it is expensive to cache a content item that is going to attract very few (or even none) views. Our proposal is that the referred user U locally stores the shared video in her local premises (e.g., home set-up box with 24/7 functionality), so that when an OSN friend clicks on the video within the OSN page of user U, the P2P-social-assisted content delivery system decides whether it informs that OSN friend to retrieve the video from the local premises of U in a P2P fashion, or the video is retrieved using standard mechanisms instead. This decision will be based on the location of both users. For instance, if both users are localized in the same country the P2P-social-assisted content delivery is efficient. However, if they are localized in Europe and America respectively, the standard existing facilities will provide a more efficient transport.

Following this work in this deliverable we present the first version of a Facebook application that is implementing this functionality for Facebook in Section 2.3.4.2.

1.3.5 Enhanced Content Placement Using Social and Coarse-Grain Location Information

In deliverable D4.1 [D4.1] an analysis was performed on a Twitter data set: using a modified version of K-means it was determined where the optimal location of data centres was located to assist the distribution of tweets. In Section 2.3.1.2 of this deliverable D4.2 various caching strategies are investigated given data centres located at these optimal locations.

2. DESIGN AND PERFORMANCE OF BUILDING BLOCKS

2.1 Realistic loads

In this section we describe statistical properties of the data set that we use in this deliverable. These statistical properties make it easier to interpret the results obtained in later sections.

2.1.1 CUTV data set

This data set was described in Section 1.2.4 and 2.3.1.3 of deliverable D3.1 [D3.1]. This trace of (664396) records spans about 32 days of CUTV (Catch-Up TeleVision) content consisting of 13585 unique content items consumed by 218104 unique users over a mobile network. Each record consists of a time stamp (with the resolution of a second), a user identifier and a content item identifier.

Figure 1 shows the popularity distribution of the content items on a log-log plot, where the content items were ordered in decreasing order of popularity. In theoretical models, often a Zipf distribution is used for such a distribution (which would show as a straight line on this plot). It can be seen that the empirical distribution has a flatter top (which could be modelled as a Zipf-Mandelbrot distribution).

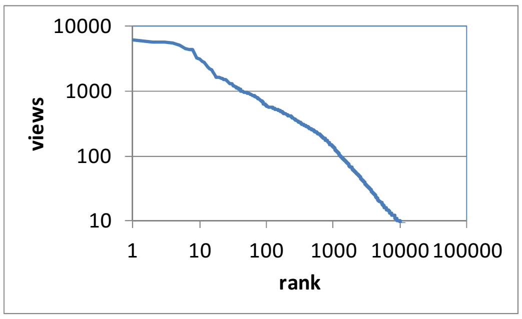


Figure 1. Item popularity in the CUTV data set.

Figure 2 shows the activity of the users on a log-log plot, where the users were ordered in decreasing order of activity. There are some very active users and users that only request a few content items over the period of about a month.

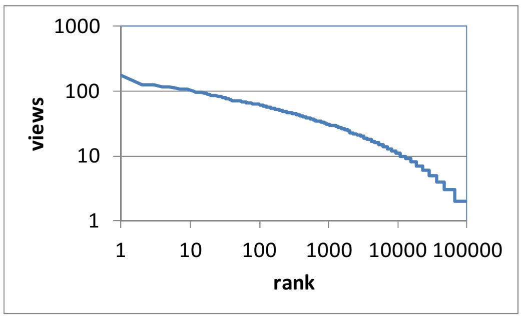


Figure 2. user activity in the CUTV data set.

Figure 3 depicts the time evolution of the requests over the complete period. A clear diurnal pattern can be seen with busy period during the day time and quiet periods at night.

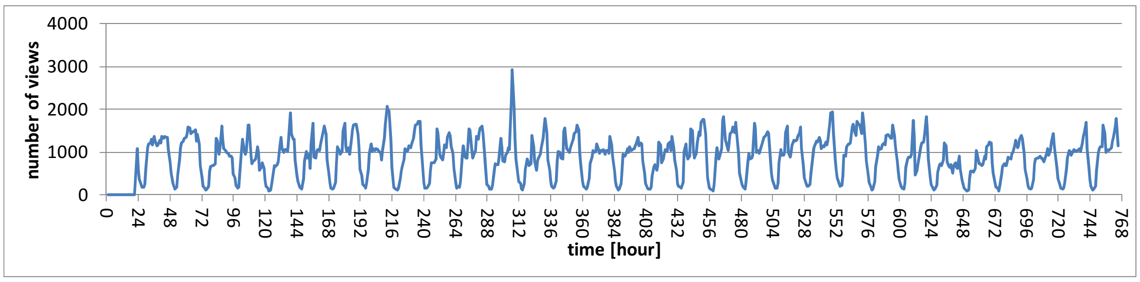


Figure 3. diurnal patterns (averaged over 1h) in the CUTV data set.

Figure 4 illustrates that the content items are only popular over a limited period. It shows the number of requests per hour for the 15 most popular content items. For most of the content items all user requests are issued over a period of a few days at the most and often all requests are issued within a day.

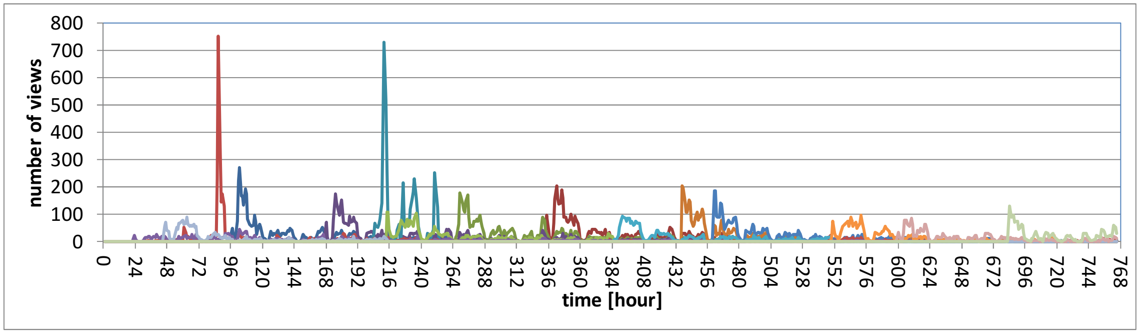


Figure 4. Life time of the 15 most popular content items of the CUTV data set.

2.1.2 Flixter data set

This data set is publicly available [FLI]. It contains a timed list of (8196077) ratings that 147612 unique users gave to 48794 unique content items over a period of roughly 4 years. Each record contains a timestamp, a user identifier, a content item identifier and a rating that that user gave to that content item. The timestamps are coarse-grained (with resolution of a day) and in principle denote the time when the user rated a content item, rather than the time when the user consumed that content item. We assume that these two time instants are more or less the same (an assumption that is not always true). Also the relation of users in a social network is given in this data set.

Figure 5 shows the number of ratings that each content item received (ordered from most "popular" item to least) on a log-log plot. It can be seen that this does not follow a Zipf law.

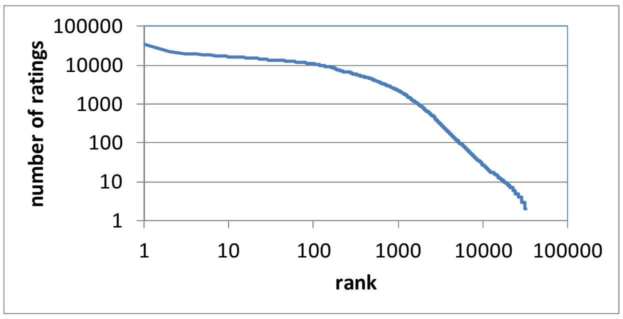


Figure 5. Item rating popularity in the Flixter data set.

Figure 6 shows the number of ratings that each user gave on a log-log plot, ordered from most active user to the least active. Notice that there are a few users that gave an extreme number of ratings (these users are probably bots).

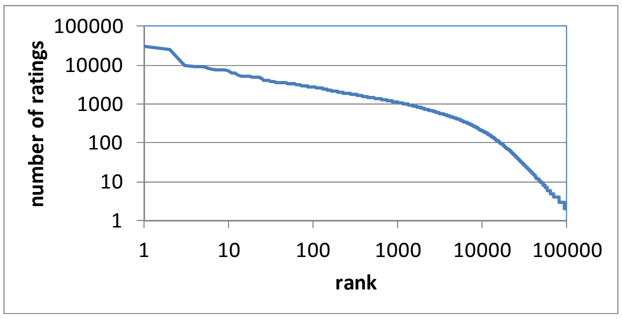


Figure 6. User activity in the Flixter data set.

Figure 7 depicts the histogram of the ratings given. As people tend to watch only content that is of interest to them they tend to give high ratings more frequently than low ratings.

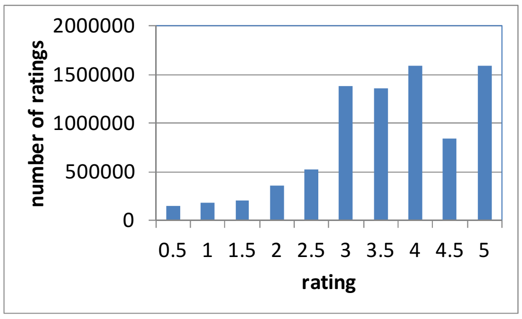


Figure 7. Rating occurrence in the Flixter data set.

Figure 8 shows how the average rating of a movie depends on the movie (rating) popularity. There is a slight tendency to rate “popular” movies higher, but the correlation between those two is only 0.04.

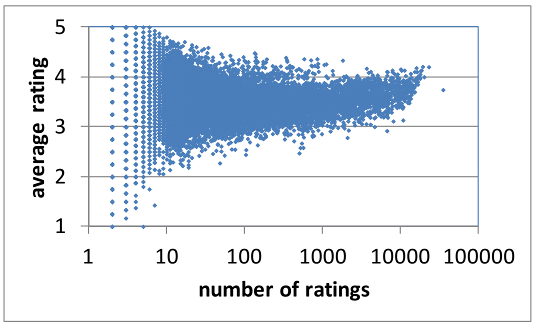


Figure 8. Correlation of average rating and popularity in the Flixter data set.

Figure 9 shows the time evolution of the number of ratings given over the period of 4 years.

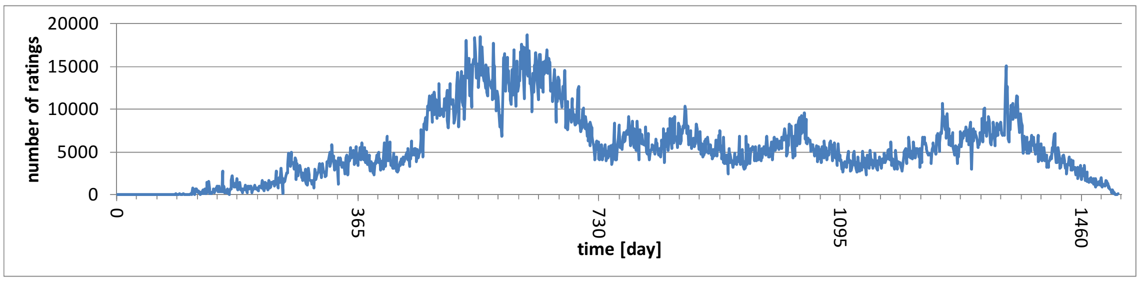


Figure 9. Activity over time in the Flixter data set.

2.1.3 Facebook data set

The Facebook data set is described in Section 1.2.1 of deliverable D3.1 [D3.1]. Here we describe the subset that was used. Data was collected by crawling the publicly available information in Facebook of some users and their friends. The data set consists of M=50 anchor users of which the list of all friends is known. In total there are N=10769 users (i.e. 10719 other users than the anchor users) occur in the data set. Of each of the users (anchor or other) the data set contains the movies they are interested in. In total K=7365 movies occur in the data set.

Figure 10 shows the number of friends the anchor users have. On average an anchor user has 253.2 friends. Figure 11 shows the number of friends amongst all users, i.e., each (anchor and other) user, have. Notice that there is one user that has all anchor users as friend. On average a users has 1.18 friends amongst the anchor users. A user other than an anchor user may have many more friends besides anchor users, but we have no knowledge about them in the data set.

Figure 12 shows the number of times a movie interest was declared by all users ordered in decreasing popularity on a linear-log plot. The most popular movie was declared 133 times, 5295 movies were declared only once and on average there were about 2.36 interests expressed per movie.

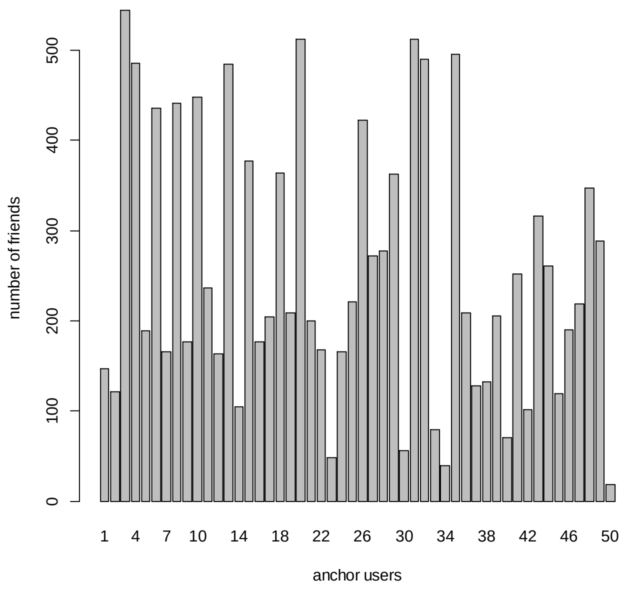


Figure 10. Number of friends of anchor users.

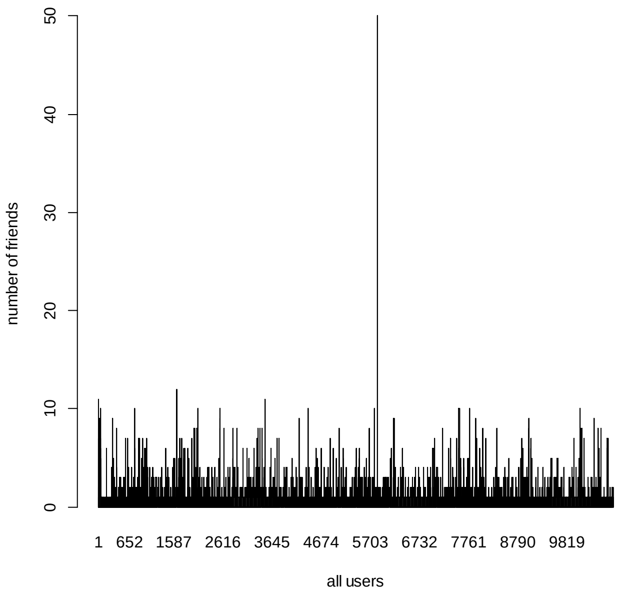


Figure 11. Number of friends amongst the anchor users of all users.

Figure 13 shows the activity of each (anchor and other) users, i.e., the number of movies each user declared. The most active user declared 579 movies, 8047 users declared no interests and on average there were 1.62 interests declared per user.

Finally we also consider the co-interests between users. The (n,n')-entry of the (NxN) co-interest matrix V indicates how many movies both user n and n' expressed a joint interest in. The diagonal elements of this matrix, which originally just contain the activity of each user, we put to 0. The first M rows (corresponding to the anchor users) of the resulting matrix we refer to as the matrix AV.

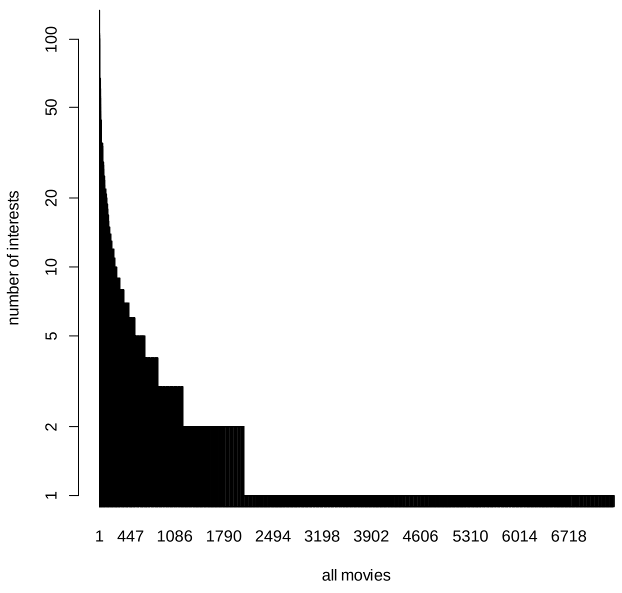


Figure 12. Movie popularity in the Facebook data set.

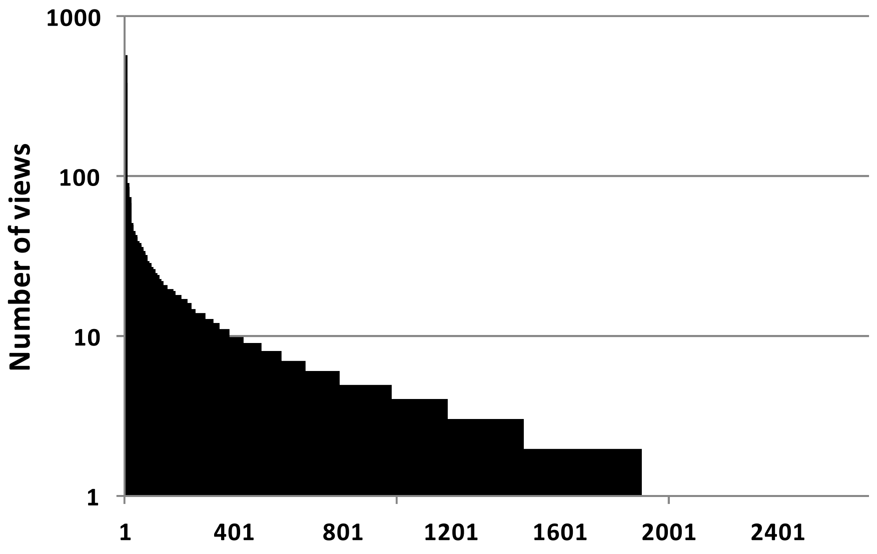


Figure 13. Activity of all users expressing interest.

The total number of co-interests in AV is 3129 (i.e., the sum of all elements of AV), while the total co-interest between friends is 425 (i.e., summing only the entries of AV for which anchor user m and user n are friends), indicating that even outside the list of friends there is a large potential to find a user that shares co-interest. This is illustrated in Figure 14 (only the anchor users who did express interest are shown here).

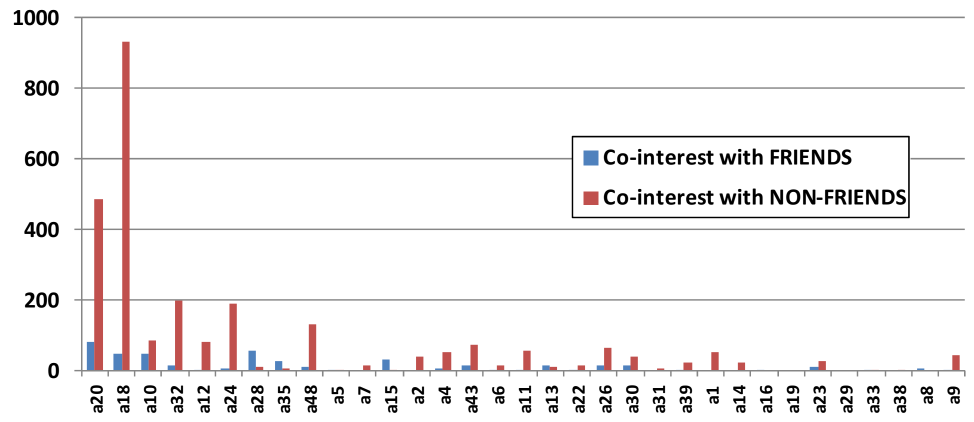


Figure 14. Co-interest of anchor users with friends/non-friends.

Note, however, that the list of non-friends of an anchor user is much larger than the list of friends. If we divide the above numbers by the total number of user pairs and the total number of friend pairs respectively, we obtain 0.6% and 3.4% respectively (see Figure 15), showing that the co-interest per couple is larger within the list of friends.

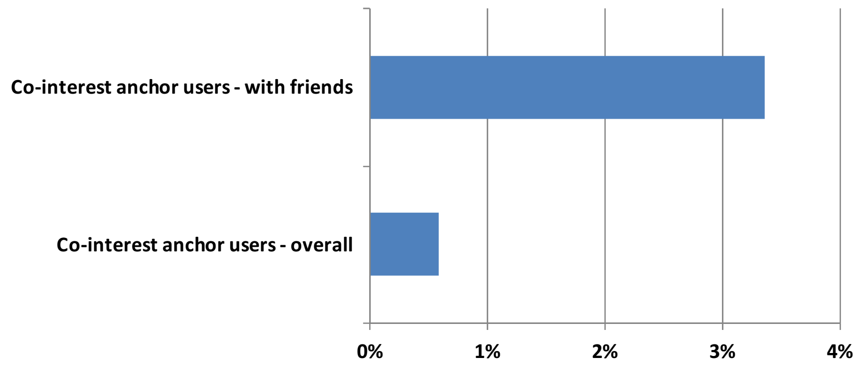


Figure 15. Co-interest % of anchor users with friends/non-friends.

Next we sum the rows of AV giving an (1xN)-matrix, shown in Figure 16, that indicates how much co-interest each (anchor or other) user has with the total set of anchor users. The correlation of this (1xN)-matrix with the activity of users is 56%, indicating that the activity of a user and his or her co-interests with the set of anchor users is somewhat, but not strongly correlated.

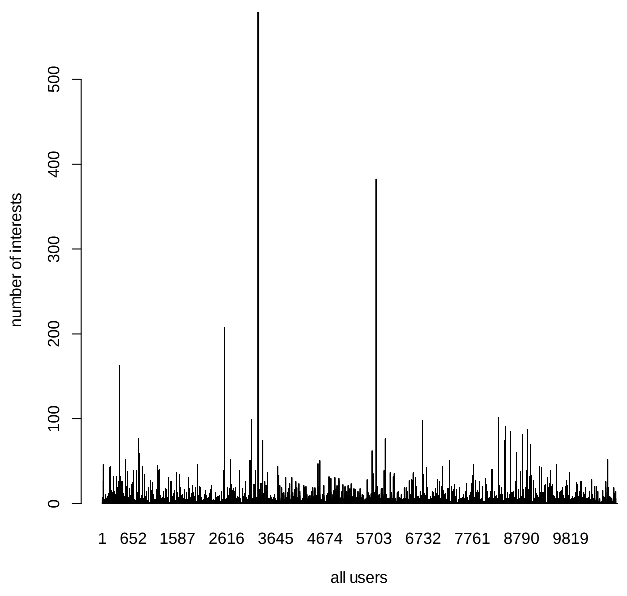


Figure 16. Co-interest of all users with the set of anchor users.

2.1.4 Mobile traces

This dataset has been collected in the frame of the project over the national mobile network of a telecom operator. This dataset is used in particular in the development and assessment phases of the pre-fetching app integrated in eCOUSIN's use case "Social-Assisted Time-Unconstrained Content Delivery". The dataset spans over 26 days, from 8 January 2014 to 2 February 2014. It contains anonymized information about video sessions from users connected to the mobile network. Users are anonymously identified via a unique integer; content items are also identified in this dataset via a unique integer.

This dataset includes only Youtube and Facebook video sessions. YouTube and Facebook videos (or video links) are largely shared via OSNs. They constitute a major source of content consumption dictated by users' behaviours in today's OSNs, and hence, are a relevant dataset for the present study. In the considered Facebook video sessions the videos are requested by users on Facebook pages and delivered from Akamai's CDN, while in the YouTube video sessions the videos are delivered from Google's CDN.

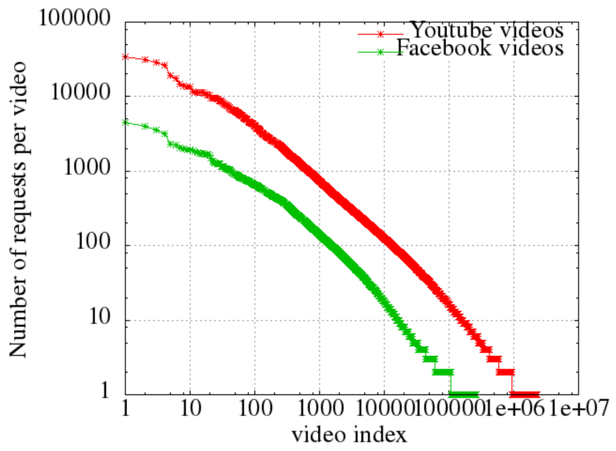


Figure 17. Youtube and Facebook videos popularity distribution in the dataset

Figure 17 depicts the popularity of YouTube and Facebook videos in the collected dataset. Both follow a Zipf distribution. The long tail in the popularity distribution of Youtube videos is more pronounced than the one of Facebook videos. About 60% of YouTube videos are requested only once, and 15% twice. Pre-fetching long tail videos might be risky and costly as there is a high probability that users do not request a given content again, even at the scale of the whole network. Therefore appropriate mechanisms need to be defined for optimizing the delivery of these videos. The objective in the use case under consideration is to exploit social information (i.e., social relationships) to improve performances in prediction and pre-fetching.

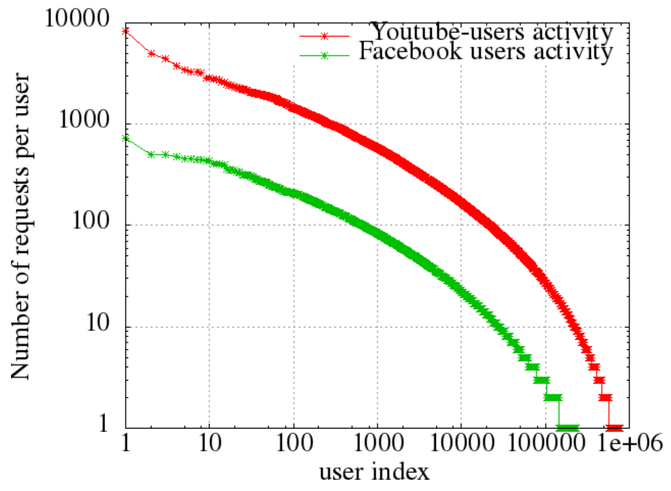


Figure 18. Activity of all users in the dataset

Figure 18 presents the number of requests over the whole duration of the dataset (26 days) for each user, sorted based on the total number of generated requests. This figure shows that there is a high heterogeneity among users regarding their activity.

Based on a finer analysis of the users' activity, we can distinguish two kinds of users: *heavy users* and *light users*. The former ones often request videos and they tend to request various videos with very different popularity ranks. Thus the long tail in the video popularity distributions (see Figure 17) is largely driven by these heavy users. In contrast the latter ones rarely request video contents. So much less information can be collected at the network level about the preferences of these users. Yet we observe that they usually request popular contents. This means that one needs to give preference in the pre-fetching logic to globally popular videos, especially in the case of light users.

2.2 Detailed description of components

This section describes various components of the "Content Dissemination Layer" that will be either further used in the demonstrators of WP6 or will be used to evaluate their performance further in this deliverable.

2.2.1 Mobile Pre-fetching Components

As introduced in deliverable D4.1 [D4.1], to study pre-fetching mechanisms that are developed in the context of the eCOUSIN project, a prototype of a mobile pre-fetching application for Android smartphones has been implemented. The following Section 2.2.1.1 presents an updated view on the overall architecture of this application and details the previously high-level versions introduced in deliverable D4.1 [D4.1] and the refined version that was presented as part of the demonstrator work in deliverable D6.1 [D6.1]. In Section 2.2.1.2, the Mobile Tracing App is described that was used to collect traces of user interactions with the OSN as well as content profiles to study the potential of different social content properties to decide on content items to be pre-fetched. The results of a first user study using this app are presented in Section 2.3.3.1.6.

2.2.1.1 Mobile Pre-fetching Architecture

It is in line with the eCOUSIN architecture as presented in deliverable D2.2 [D2.2]. An early version of the mobile pre-fetching architecture was presented in deliverable D4.1 [D4.1] and first parts of its detailed implementation as mobile pre-fetching application for Android devices were presented in deliverable D6.1 [D6.1]. The architecture builds the general framework (see Figure 19) that the content prediction and pre-fetching mechanisms are supposed to work in. Therefore it is important for the understanding of the possibilities as well as the limitations for the design of these mechanisms.

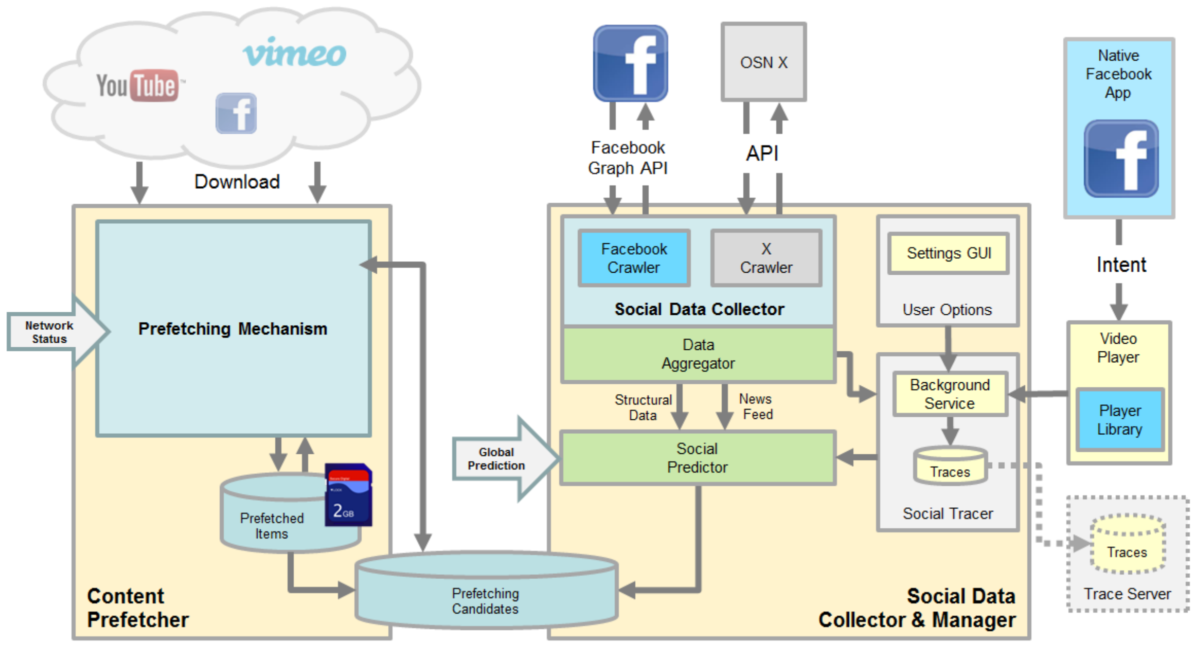


Figure 19: The overall pre-fetching architecture.

The two main building blocks of the architecture are the "Social Data Collector and Manager" as well as the "Content Pre-fetcher". While the first is mainly responsible for the access to social information (e.g. from Facebook) and the content prediction based on this information, the latter is responsible for the actual pre-fetching of content. In addition to these two building blocks, we decided to skip our initial plans to provide an own user frontend for the access to the OSNs (which was primarily Facebook in our case). Instead, the native Facebook app is used for a better user acceptance and to avoid undesired influence of the presentation or the ordering of content items on the pre-fetching performance, which showed to be a major concern for this work. The pre-fetching app, thus, would mainly run as a background service. The downside of this approach is that a detailed study of the user interaction with the mobile OSN client and browsing behaviour is not possible anymore.

The "Social Data Collector and Manager" module still can access the OSN using available APIs (Application Programming Interface) and gain insights into the interaction once a user, e.g., likes or comments content items. Yet, it is only able to trace pure consumption actions (e.g., displaying a picture or watching a video) of content if it triggers an external event outside the Facebook app that can be traced by the pre-fetching app. Such an external event is triggered for accessing videos from content providers other than Facebook itself (e.g. YouTube). Starting with an update of the Facebook app in February 2014, Facebook-hosted videos are displayed using an internal player within the

Facebook app. Although this means that we cannot trace the access to these videos, it must be noted that it does not make sense to pre-fetch Facebook videos as the playback of these videos cannot easily be redirected to the playback of a pre-fetched version of the video. For all other videos, the Android intent concept is used to use an own video player component for the playback of videos, allowing us to redirect the access to a local pre-fetched copy once the user chooses to play a video from inside the Facebook app.

Within the “Social Data Collector and Manager”, the “Social Data Collector” and the “Data Aggregator” modules regularly retrieve and aggregate new entries from the user’s newsfeed, their attached social information (i.e., likes, shares, comments, etc.), as well as information on the structure of the user’s friend graph and memberships in groups. Based on this information, the “Social Predictor” module identifies, filters, and ranks content items according to their relevance for pre-fetching. This module builds the fundament of all further pre-fetching steps and fundamentally influences its achievable performance. Content items that were not identified as being relevant for pre-fetching are not considered by the “Content Pre-fetcher” module.

The “Content Pre-fetcher” module uses the pre-fetching candidates as input and, first of all, retrieves further content-related metadata directly from the content provider (e.g., video length or file size of the video). Second, it plans and schedules the download of the content item and, third, transfers it to the mobile device. Planning and scheduling can be rather simple and work reactively once the mobile device connects to a WiFi (Wireless Fidelity) network and immediately downloads the pre-fetching candidates according their ranking, or be more complex and include predictions on the network quality and also involve the mobile network connection.

This mobile pre-fetching app is implemented as a prototype for Android devices and is reported as a demonstrator in WP6 and was described in a first version in deliverable D6.1 [D6.1]. It includes all major components of the pre-fetching approach and is planned to be used as basis for further user studies once its implementation is completed. Several extensions to “Content Pre-fetcher” component are planned by other partners to be integrated with the demonstrator to show the potential of this approach.

In case content has to be transmitted over mobile networks, a planned download scheduler like the one described in section 1.3.2.2 can be used. This scheduler will take as input the predictions of network quality described in deliverable D5.2 [D5.2], e.g., the CQIs (Channel Quality Indicators), and will try to plan the downloads of the various content segments according to their QoE (Quality of Experience) constraints, while utilizing the channel only when the predictions provide favourable values. A detailed description of a first version of such a scheduler is presented in section 2.2.2 of this deliverable D4.2.

2.2.1.2 Mobile Tracing App

In course of the investigations and the development of the previously described architecture and its prototypical implementation, a user study was planned as basis for the investigation of the potential to use available social information of content items to define the “Social Predictor” component. To execute the study, a first simplified version of some parts of the mobile pre-fetching application was developed as shown in Figure 20. It includes the “Social Data Collector”, the “Data Aggregator”, as well as a mobile frontend to present the newsfeed data of Facebook to the users. Besides, it also includes a tracing component that collects data items from the newsfeed, the structural information of the user’s friendship graph, as well as the user interaction with the frontend when consuming content.

While the approach to implement an own frontend has some drawbacks as described in the previous section, it allows collecting detailed information on the user actions, including access to all types of content, also covering Facebook videos and pictures that would otherwise be displayed within the OSN application. We decided to follow this approach early on and used the gained insights and experiences while building the mobile tracing app to define the pre-fetching architecture described in the previous section.

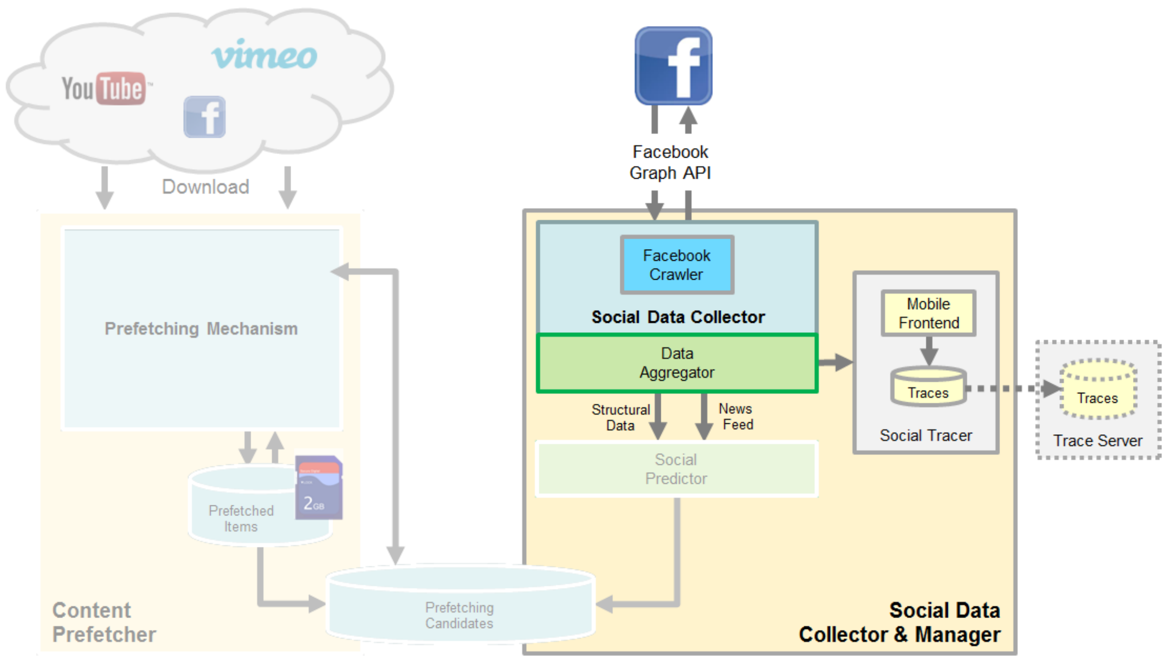


Figure 20: The mobile tracing app and its context in the pre-fetching architecture.

In addition to the mobile app, also a “Trace Server” was developed that collects the traces uploaded by individual clients. To comply with local privacy regulations, all collected data was anonymized before even storing it locally on the user device itself. This way, no personal information or any plaintext information was collected. For the investigations, only aggregated as well as statistical information on the social information were required.

2.2.2 Network resource allocation algorithm

Throughout this section, we assume that the content that is requested has been pre-fetched to an entity that we control. So, we have access to the whole video. We will further assume perfect knowledge of the channel capacity that can be allocated to a user in the near future. Essentially, the channel capacity that a user can enjoy is related to the following parameters:

- **User position in the cell and speed.** The CQI, such as the RSRP (Reference Signal Received Power), RSRQ (Reference Signal Received Quality) and SINR (Signal to Interference and Noise Ratio), depend heavily on the location of the user. Based on the CQI, LTE subsequently determines which MCS achieves the best trade-off between efficiency and robustness[1].
- **Presence and activity of other users in the cell.** The amount of radio resources of the eNodeB is finite. Hence, if a base station does not have enough capacity to serve all the users (i.e., when it is congested) at the rate that they initially request, then all of them are allocated a fraction of the available resources based on a fairness scheme. The scheduler of the base station determines the fairness scheme and for the rest of the paper we will use the

[1] The mapping of CQI to MCS in LTE can be found at table 7.1.7.1-1 of 3GPP TS 36.213 V9.2.0 (2010-06) found at http://www.3gpp.org/ftp/Specs/html-info/36213.htm

proportional fairness scheme[2]. This scheme tries to maintain a good balance between the overall throughput of the cell and a minimum service guarantee for all the users.

Hence, if one can predict the future values of the above parameters, it is trivial to predict the maximum allocated capacity for each user. As presented in deliverable D5.1 [D5.1], there are various techniques that can predict the mobility and activity of the users, so it is possible to have a good estimation for the near future capacity within an acceptable error margin. For the remainder of this work, this scenario will be referred to as the "online case". In the present work, we will investigate though the simplified "offline case", where we assume the future capacity of each user is perfectly known. The future is divided into slots of fixed time duration and we assume to have the knowledge of the average capacity allocation of every user per slot. Usually, capacity is measured in bits per second, but in the case of the LTE radio access network, a more representative unit would be the amount of radio resources required to carry the data (measured in bits) transmitted to the users. In LTE, radio frequencies are organized in groups of 14, called resource blocks and resource blocks in turn are organized in groups of varying size (2, 3 or 4), forming a unit called RBG (Resource Block Group). RBGs are the quantum of resources that can be allocated to the users. The amount of bytes that each RBG can "carry" depends on the modulation and coding scheme used for it. Thus, if we allocate resources to a user when he has good channel conditions and avoid allocating him resources when he has bad channel conditions we will be able to transfer the same amount of data (bits), using significantly fewer RBGs. Minimizing the number of RBGs per user can boost the number of users being served in a congested cell. Our goal is to transfer as much data as possible to the UEs when they are enjoying good conditions and rely on the buffer to guarantee uninterrupted video playback when they are facing bad channel conditions. When a proportional fairness scheme is used in a congested LTE cell (all users are generating saturation traffic, so they are allocated less rate than they request), we can infer the channel quality of each user by comparing the size of the TBs (Transport Blocks)[3]. A big TB leads to big capacity and reflects good channel conditions. A small TB leads to low capacity and reflects bad channel conditions.

Based on that observation we will propose a first working practical implementation of the algorithm (see [VECI13]), aiming at delivering stored video, using as few radio resources as possible. As stated above, our objective is to minimize $\frac{1}{T}\int_0^\infty \frac{r(t)}{c(t)}dt$, which mathematically expresses that we seek to consume bandwidth, only when it is high.

Modern video codecs express frames in relation to their neighboring frames. This is reflected in the manner in which video applications request data from the buffer. Instead of requesting the data on a per frame basis, they request data that refers to a group of frames at a time. Thus, the video application requests data from its buffer a limited number of times every second. In the rest of the section we will assume that the period between two consecutive requests for data is 100ms. In order to ensure an uninterrupted playback experience all the related data of a group of frames, should be present to the buffer before the video application requests them. If this requirement is not fulfilled, then the playback has to stop until new data arrives in the buffer, causing "rebuffering delay". At this point we only investigate scenarios where uninterrupted video delivery is possible (the rate allocation algorithm can provide a feasible solution that does not require rebuffering). Hence, if the algorithm cannot provide such solution, we assume that the algorithm has failed. In later versions of the algorithm where rebuffering will be allowed, the algorithm will always be able to provide a feasible solution regardless of the network conditions. In order to force this limitation as well as to

[2] http://en.wikipedia.org/wiki/Proportionally_fair

[3] LTE terminology to refer to the packets transmitted at the MAC layer of LTE

ensure that the UE does not receive more data than it can store at the buffer the following constraints are defined:

1. **Low Constraint**: At any time t, the cumulative amount of data delivered to the buffer of the video application (symbolized β) should be at least equal to the amount of data that the video application has requested up to this point. In addition, the video delivery process should have finished before the last data request.

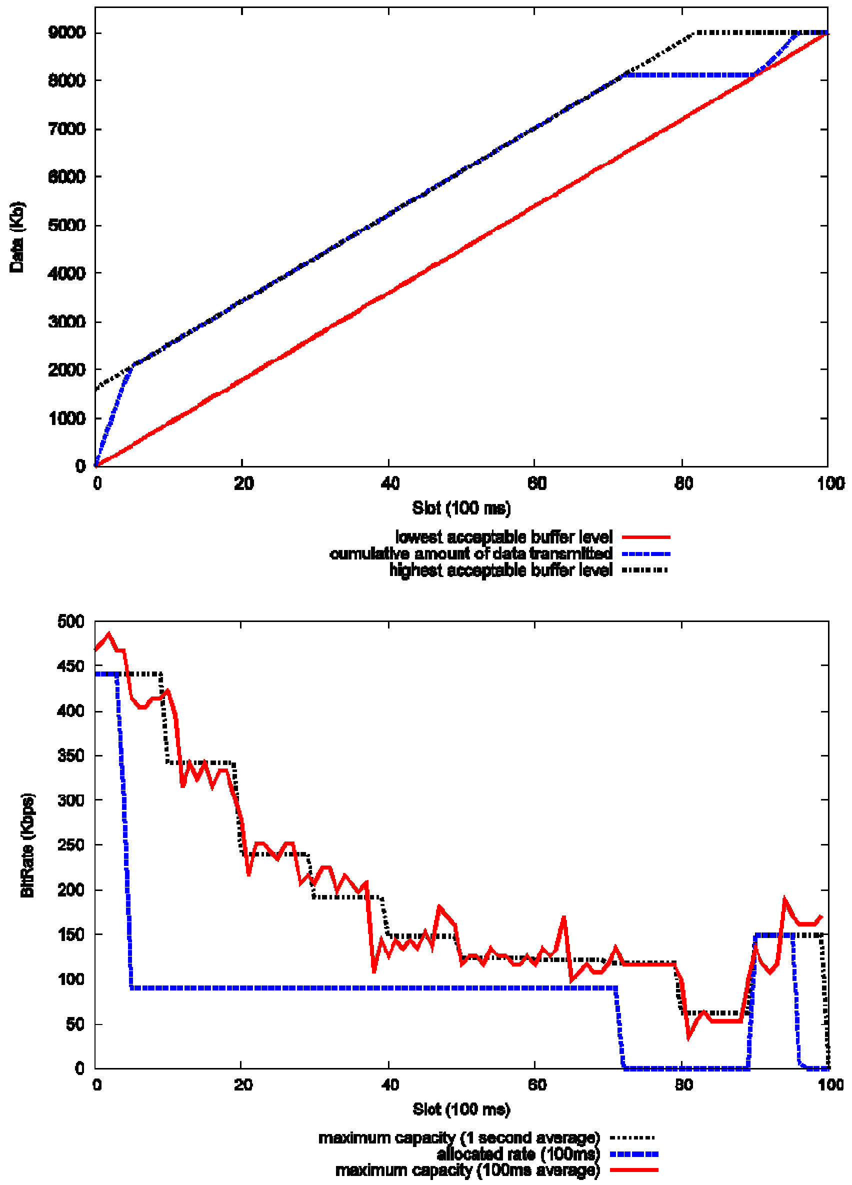


Figure 21: Example run of the Vanilla optimization algorithm: (top) β alongside the constraints; (bottom) real capacity (red), calculated capacity (blue) and allocated capacity (black).

2. **High Constraint:** At any time t, the cumulative amount of data delivered to the buffer of the video application (β) should be at most equal to the amount of data that the video application has requested up to this point plus the size of the buffer at time t.

These constraints are checked at time τ, when either the buffer changes size and/or the video application requests data from the buffer. A rather high level overview of how the algorithm functions follows:

1) Define a time interval that starts at the start time of the video and ends at the end time of the video.
2) Calculate the average future capacity of a user per 1 second slot.
3) Sort the slots according to capacity in a descending order. If two slots have the same value sort first the one that appears earliest.
4) Start allocating rates from the sorted slot list equal to the capacities, until the sum of the transmitted data is at least equal to a target. The assigned rate of all the slots is equal to the capacity, with the exception of the last slot allocated where its rate is equal to the size of the video minus the sum of the capacities of all the previously allocated slots. In the first iteration of the algorithm the target is equal to the size of the video.
5) Check if there are any violations of low and high constraints. If there is no violation, the rates of the previous step represent the optimal feasible solution, for the time interval under examination. If this time interval ends at the end of the video stop and in the opposite case, run again the algorithm starting from the time point t, where the present time interval ended.
6) Given that there are violations of low and high constraints, identify the type of the first violation that occurs and find the first time point τ, where both constraints are again satisfied.
7) Run again the algorithm from step 3 starting from the beginning of the current time interval until τ and setting a new target of either an empty buffer at τ, in case the low constraint was violated (buffer underflow), or a full buffer at τ, in case the high constraint was violated (buffer overflow).

If the algorithm cannot provide a feasible solution for the whole duration of the video for the given constraints we claim that it has failed. When rebuffering will be taken into account, in future versions of this model, the algorithm will always be able to provide a feasible solution, regardless of the constraints. Thus, it will never fail.

In Figure 21, we can see a realisation of the optimization algorithm over a 10 second video with a bit rate of 900kbps. In this example we have also assumed that the size of the buffer of the client video application at the UE is constant. At any time instant τ, the value of the low constraint is equal to the amount of data that should have been received, in order to have an uninterrupted playback up to τ and an empty buffer at τ. On the other hand, the value of the high constraint is equal to the amount of data that should have been received, in order to have an uninterrupted playback up to τ and a full buffer at τ. Thus, both constraints any time instant τ, differ by the size of the buffer and as a consequence change at exactly the same time (when a group of packets is removed from the buffer).

So whenever a group of frames is consumed, causing the low constraint to jump, the high constraint jumps by the same amount. The upper part of Figure 21 shows how the cumulative amount of transmitted data increases over time as well as the values of the constraints for the different time instances. In order to have a feasible solution the curve of the cumulative amount of transmitted data (β) should always be within the area defined by the curves of the restrictions.

The lower part of Figure 21 compares the computed average capacity averaged over 1 sec intervals, the actual rate, and the allocated rate for all the 100ms slots. The main goals of the algorithm are illustrated in the figures. When the current capacity is high and the future capacity is expected to be low, the allocation of capacity is well above what is required for a smooth video playback, so the buffer gets full. When the UE is experiencing low capacity, no bandwidth is allocated. Instead the playback relies on the data stored in the buffer. Also if the buffer is already full and the future capacity is expected to be low, the algorithm allocates just enough capacity to maintain the full buffer state, in order to be at the best possible position when the transition to low capacity takes place.

The application of the algorithm in practice poses a number of problems, according to our simulations results. An exhaustive discussion of these problems and their possible solutions can be found in [MICH13]. A very brief summary is presented in the following table. If a possible solution is marked in red letters, it is implemented in the final solution.

Table 1: overview of the features of the allocation algorithm.

Problem	Possible solutions
Buffer overflows between two consecutive states that have as a target a full buffer. This is caused because the algorithm might schedule IP (Internet Protocol) packets when the buffer is already full, when it tried to maintain a full buffer state.	set the constraint of the maximum data received at any given time, equal to β + [size of the buffer] - [size of a group of frames]
Averaging the capacity over 1 second time intervals. This causes wrong estimation of the bandwidth, which can cause the sender to overflow the buffer of the eNodeB, since it is not able to transmit the IP packets at a rate close to the rate it receives them.	Increase the buffer at the eNodeB. More specifically add a new buffer dedicated to such scenarios. A simple method for calculating the size of the buffer is presented in [MICH13]
	Change the granularity of the average. As discussed above this could be a possibility in the future where CQI measurement updates will be more frequent.
	Spread the rate over the 1 second slot if possible. This reduces the possibility of transmission spikes that may cause buffer overflows at the eNodeB. This solution though relies on future slots (increasing the risk of buffer underflow at the UE) and is applicable usually when the channel is underutilized.
The assumption that capacity is equal to goodput. The algorithm does not take into account retransmissions that can delay the delivery of the proper data.	set the constraint of the maximum data received at any given time, equal to f + [size of a group of frames]
Segmentation of IP packets to LTE TBs. The algorithm measures the capacity in bit rate at the LTE level. This can be misleading since an IP packet is delivered to the upper layers of the UE only when all the LTE chunks of it are received. So, it is possible to have enough capacity to ensure playback, but the actual IP packets not being delivered (see also Figure 22).	Increase the low constraint by a group of frames.
	Measure the capacity based on packets delivered to the video application.
	Make IP packets tiny, so that they can be transferred by one TB, regardless of the capacity of the TB. This cannot be applied in practice for obvious reasons.
IP packet delivery delays (between layers – constant value). There is a non negligible delay at various stages of the IP packet delivery	Transmit at a slightly higher rate at the beginning of a 100ms and at a slightly reduced rate at its end.

process.	
	Measure the capacity based on packets delivered to the video application.
	Increase the low constraint by a group of frames
Properly syncing video server and video application can be challenging (variable amount of delay). This problem arises only when we measure capacity based on the number of received IP packets by the application buffer	Delay remains constant for as long as the competition in the cell remains constant.
	Knowing the kind of competition in the cell might lead to an accurate prediction of the expected delay, for a given scheduler

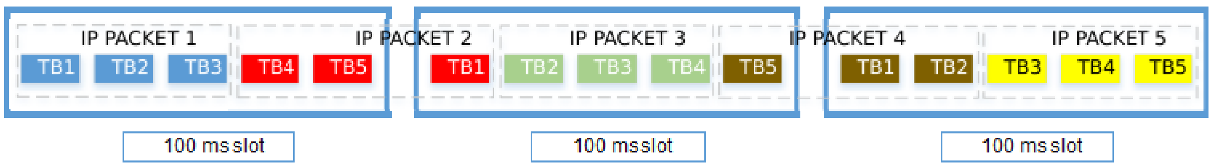


Figure 22. Segmentation of IP packets to TBs: On some occasions, this can cause the late delivery of data to the buffer application

Based on the observations above we propose the following enhancements to the original algorithm.

- Calculation of the capacities at the application layer.
- Lower the constraint for the maximum amount of data that can be delivered at the UE buffer at any time instance, by a group of frames (10 IP packets in our simulations).
- Increase the constraint for the minimum amount of data that can be delivered at the UE buffer at any time instance, by a group of frames (10 IP packets in our simulations).
- Calculation of the average capacity over smaller time intervals.
- The distribution of the calculated capacities within the time interval over which we compute the average is not even.
- Increase the buffers of the transmission queues of the eNodeBs (or add a specific buffer on top).

At this point, we do not set a fixed value on the above variables. In the sequel, where we present our evaluation of this model, we explore the various tradeoffs imposed by the different values.

2.2.3 Description of the pre-fetching simulator ("Prefsim")

The purpose of the pre-fetching simulator "Prefsim" is to assess the performance of content prediction algorithms developed in WP4.

Prefsim considers the following inputs:

- Traffic file. This file is mandatory to run any simulation. This file consists of a list of requests for content from a given set of users. Each row corresponds to a request and has the following format:

  Timestamp UserID VideoID Duration

- Link file. This file describes the social graph of the considered set of users, i.e., the set of OSN relationships between the users.
  It must be noted that this file is not mandatory. If the link file is provided to the simulator; the social graph considered in the simulation is the explicit social graph given by this link file. Otherwise the simulator uses the data from the traffic file to generate an implicit social graph.

When executed, Prefsim runs three main threads:

- The first one consists of generating the social graph and communities. The simulator generates the social graph of the considered users. Then for each user it lists its set of neighbours; a given user and its neighbours constitute a community.
- The second one is the prediction thread. For each user the simulator determines the set of items to pre-fetch and when to pre-fetch them. The items are selected by considering their popularity within the user's community.
- The third one is the assessment thread. The simulator assesses the performance (i.e., level of correctness) of the content consumption prediction.

Prefsim requires a set of parameters to initialize the simulation. These parameters are defined in the file input.xml.

An example of input.xml is given below:

```
<?xml version="1.0" encoding="UTF-8"?>
<input>
<parameters>
 <SimulationStep>10</SimulationStep>
        <Graph>
        <TypeGraph>social</TypeGraph>
        <relationship>undirect</relationship>
                <UpdateGraph>false</UpdateGraph>
        <EndTrainingData>12</EndTrainingData>
        <NeighborsOffset>50</NeighborsOffset>
        </Graph>
        <User>
                <CacheSize>200</CacheSize>
                <SizeHistoryActivity>10</SizeHistoryActivity>
                <CacheManager>LRU</CacheManager>
                <NotifManager>EdgeRank</NotifManager>
        </User>
        <Prefetch>
                <SelectMode>load</SelectMode>
                <Mode id="load">
        <MaximumSimultaneousConnections>18000</MaximumSimultaneousConnections>
                </Mode>
                <Mode id="scheduled">
                        <tc>10800</tc>
                        <delta>10800</delta>
                </Mode>
                <Mode id="Wifi">
                </Mode>
                <expA>35</expA>
```

```
				<expB>0.00011</expB>
				<ScheduleNextPrefetch>864000</ScheduleNextPrefetch>
				<AverageDownloadSession>20</AverageDownloadSession>
				<ThresholdNotification>1</ThresholdNotification>
				<DiscardTime>3</DiscardTime>
				<PrefetchSince>2</PrefetchSince>
		</Prefetch>
</parameters>
<absolutePath>
				<TrafficFile>/home/user/Prefsim/traces/file_test</TrafficFile>
		<SocialGraphFile>/home/user/Prefsim/traces/link_test</SocialGraphFile>
		<InterestGraphDirectory>/home/user/Prefsim/traces/data/</InterestGraphDirector>
		<outputDirectory>/home/user/Prefsim/results/</outputDirectory>
</absolutePath>
</input>
```

We detail below the most important fields of the file input.xml:

- Field <Graph>

  The simulator can run in two different running modes: social or interest.

  In the social mode, the link file must be provided as input to the simulator; the social graph considered in the simulation is the explicit social graph given by this link file.

  In the interest mode, the simulator uses the data from the traffic file to generate an implicit social graph; the graph is created based on the common interest of the users: the simulator considers the items consumed by each user and creates communities based on the interest. The size of the communities, i.e., of the node degrees for the implicit social graph, is defined in the field <NeighborsOffset>.

  Two methods have been implemented in the simulator to create the interest graph:

  - Method 1: The interest graph is generated (first thread of the simulator) by using all the requests from the Traffic file issued during a first period of time defined in the Field <EndTrainingData>. Then the simulator uses this interest graph to run the second and third threads.
  - Method 2: The simulator rebuilds the social graph every day D from scratch. The interest graph is generated by using all the requests from the Traffic file issued until Day D.
- Field <User>

  The purpose of this field is to set the characteristics of the user terminal.

  The size of the cache on the user terminal must be specified in the field <CacheSize>.

  The cache replacement algorithm implemented on the cache of the user terminal must be specified in the field <CacheManager>.

  The field <NotifManager> is for selecting the pre-fetching algorithm to be used by the pre-fetcher agent running on the user terminal. When the pre-fetching task is executed, this agent applies the pre-fetching algorithm to select and rank the content items to pre-fetch.

  The following pre-fetching algorithms are implemented in the present version of the simulator: LRU, EdgeRank, First In First Out (FIFO).

- Field <Prefetch>

  This field is for defining when the selected content must be pre-fetched. There are 3 different modes at this stage, to be selected with the field <selectMode>.

  The first mode is "load". In this case pre-fetching is allowed only if the number of simultaneous connections is below <MaximumSimultaneousConnections>.

  The second mode is "scheduled". In this case pre-fetching is only allowed in a specific period <tc>+-<delta>, where the field <tc> corresponds to a given hour and the field <delta> to a given interval around <tc>. For example, if <tc>=10800 seconds the users are allowed to pre-fetch content at 3 am; in addition if <delta> = 10800 seconds the users are allowed to pre-fetch content anytime within the interval [0..6] am.

  The third mode, "Wifi", is not implemented yet; it is to be specified in relationship with WP5.

  We assume that the number of items pre-fetched on the mobile device of a given user is noted "Nitems" and that Nitems depends on the user's activity so as to take account of the distinction between heavy users and non-heavy users, as reported in Section 2.1.4. We propose that Nitems be defined the following way: Nitems = expA * exp(-expB*t), expA and expB being two constant values and t being in seconds. expB represents the exponential decay. The field <expA> and <expB> in the file input.xml are used to set these figures.

  The field <ThresholdNotification> is used to set the "social threshold" for triggering a pre-fetching. When it is set to 1, this means that the terminal of a given user may pre-fetch all contents that have been viewed by at least one of her direct neighbours in the social graph. With it is set to 2, this means that the terminal of a given user may pre-fetch all contents that have been viewed by at least 2 of her direct neighbours in the social graph.

The simulator generates three output files, providing the different statistics of the simulation when it is over.

1. The first file is outputUser-xxxxx

In this output file, each row reports the outputs relative to each user, with the following format:

UserID

Number of neighbours

Number of requested items

Number of pre-fetched Items to userID

Hit ratio

Miss_type_I_ratio

Miss_type_II_ratio

Miss_type_III_ratio

Correct Prediction Ratio.

There are 3 different types of miss ratio, each corresponding to a specific nature. This helps to analyse precisely the reasons of a miss:

- MISS type I: the requested content item was correctly predicted and pre-fetched but it was then removed from the cache before the user requested it because of the limited cache size, leading to a cache miss.

- MISS type II: the user terminal was aware about a given content item (i.e., at least one of its neighbours had already requested it), but this content was not pre-fetched on time (before the user requested it) because the pre-fetching algorithm did not estimate it was relevant.
- MISS type III: The user was not aware about the content item and apparently it was impossible to the pre-fetcher agent to pre-fetch the requested content.

HR (Hit Ratio): is the ratio of videos that have been pre-fetched and requested out of all videos requested by the user.

CPR (Correct Prediction Ratio): is the ratio of videos that have been pre-fetched and requested out of all pre-fetched videos by the pre-fetcher agent.

2. The second file is output-xxxxx

In this output file, each row reports statistics relative to day N, where N ranges from 1 to the last day of simulation, with the following format:

Day

Cumulative Number of requested items during the simulation

Cumulative number of pre-fetched items

Cumulative Average hit-ratio

Cumulative average miss_I

Cumulative average miss_II

Cumulative average miss_III

Cumulative average-accuracy-ratio

3. The third file is HitInfoLog

In case of a hit, this file reports the userID of the user who made the request, the userID of its neighbours and the time of the request/hit. This gives insights on how the friend list of the user changes across the time.

2.3 Performance of caching components

2.3.1 Global CDN

2.3.1.1 Performance to access Facebook services in native servers and Akamai caches

2.3.1.1.1 Introduction

Facebook is an OSN with more than one billion subscribers all over the world. In addition, according to Alexa Ranking, Facebook is the most popular website in the world. A system of that dimension needs to be sustained by a robust and reliable architecture. Toward this end, Facebook owns and manages a number of centralized data centres located in US and Ireland [WON12]. However, those data centres are far away from a large number of Facebook subscribers, which suffer from very high delays to access them. Therefore, in order to provide an efficient service, a worldwide popular system like Facebook needs to rely on a distributed infrastructure that allows subscribers accessing the offered service with a good quality of service (e.g., low delay). To achieve this goal Facebook uses Akamai [AKA13], a CDN with 60K servers deployed in 1400 data centres located in 900 different networks worldwide. In this context, an intriguing question is how this complex infrastructure offers Facebook services to Facebook subscribers, and whether all countries around the world are experiencing the same quality of service in terms of delay to access those services. The goal of this research is to present a rigorous measurement study that allows us to draw the actual Facebook infrastructure (including Akamai servers) and how it is being used to attend to subscribers demand.

To answer the previous question it is essential to determine how Akamai servers that offer Facebook services are distributed around the world, and to what Akamai location Facebook subscribers are redirected when they access a particular service. Toward this end, we have followed a systematic methodology that allows us to identify what Akamai servers are offering what Facebook services and geolocate them. This methodology is composed of four basic steps: (i) identify URLs (Uniform Resource Locators) associated to Facebook services, (ii) execute ping and traceroute commands from edge machines distributed worldwide to extract IP addresses associated to servers attending queries related to the discovered Facebook services, (iii) geolocate those IPs and determine which ones are associated to native Facebook servers and which ones belong to Akamai servers, (iv) determine which source nodes (in which location) are attended by which Akamai servers. To apply this methodology we have used 463 PlanetLab [CHU03] nodes distributed across 41 countries all over the world, that have sent ping and traceroute probes to 47 different Facebook URLs (grouped into 16 different service categories) 6 times a day during two weeks, from May 7th to May 21st, 2013. Overall we collected almost 2M delay samples from PlanetLab nodes to Facebook native servers and Akamai nodes.

Based on the results obtained from our measurements, we present a discussion that mainly covers two aspects: (i) the quality of service (in terms of delay) experienced by subscribers depending on their location, (ii) the picture of where Akamai nodes offering access to Facebook services are located and which geographical area they cover (i.e., whether an Akamai node located in a country A only receives queries from nodes located in that country or it actually also serves nodes placed in other countries, and in such case whether these are neighbour countries or not).

The main outcomes obtained from our analysis are: (i) the location of subscribers defines the quality of service in terms of delay for both native Facebook servers and Akamai nodes providing access to Facebook services, and (ii) Akamai nodes providing access to Facebook, usually serve local subscribers as well as subscribers located in nearby countries. However, we can find a number of situations where Facebook users are redirected to long distance Akamai nodes, which increase their access delay, and thus degrades their quality of service.

2.3.1.1.2 Methodology

The goal of this research is twofold: (i) analyze the user experience to access Facebook services from different countries in terms of latency, and (ii) describe a geographical picture for the location of those servers (with an especial focus on Akamai nodes) offering Facebook services, and, linked to that location, whether they just cover a local region or they also serve users located in different countries. Toward this end, we followed the next steps to gather the necessary data to perform the study. First, we replicate a normal Facebook activity and detect by means of a network protocol analyzer tool (i.e., Wireshark [ORE06]) the URLs associated to common Facebook operations such as authentication, profiling, photo view, chat, etc. We found 47 URLs that correspond to 16 different service categories. Table 2 shows the 16 Facebook service categories included in this study as well as the information of which servers, Akamai and/or Facebook, are in charge of replying the queries for these services. Once we had determined the URLs, we prepared a simple script that executes a ping and a traceroute operation from a source node to all the 47 URLs, and we distributed this script to 463 PlanetLab [CHU03] nodes located in 41 different countries distributed as shown in Table 3. We run the script in each node six times a day during two weeks from May 7th to May 21st, 2013, in order to have a large enough dataset to avoid eventual network effects that could corrupt the average delay results. Finally, to geolocate both PlanetLab nodes and Facebook and Akamai servers we used Maxmind database [MAX13] for binding the IP address to its respective country. Globally, our dataset contains almost 2M sample measurements.

Table 2. Facebook Service Categories, number of URLs for each Service, and Service Provider (Facebook and/or Akamai).

Service Category	#URLs	Service Provider
Access website	2	Facebook, Akamai
Authentication	4	Facebook, Akamai
Blog site	1	Facebook
Chat	2	Facebook
Developer site	1	Facebook
Error	1	Facebook
Friend finder	1	Akamai
Friend site	1	Facebook
Game applications	3	Facebook
Group site	1	Facebook
Multi-services	4	Facebook, Akamai
New Feed	4	Facebook
Photo upload	1	Facebook
Photo view	19	Facebook, Akamai
Post site	1	Facebook
Video view	1	Akamai

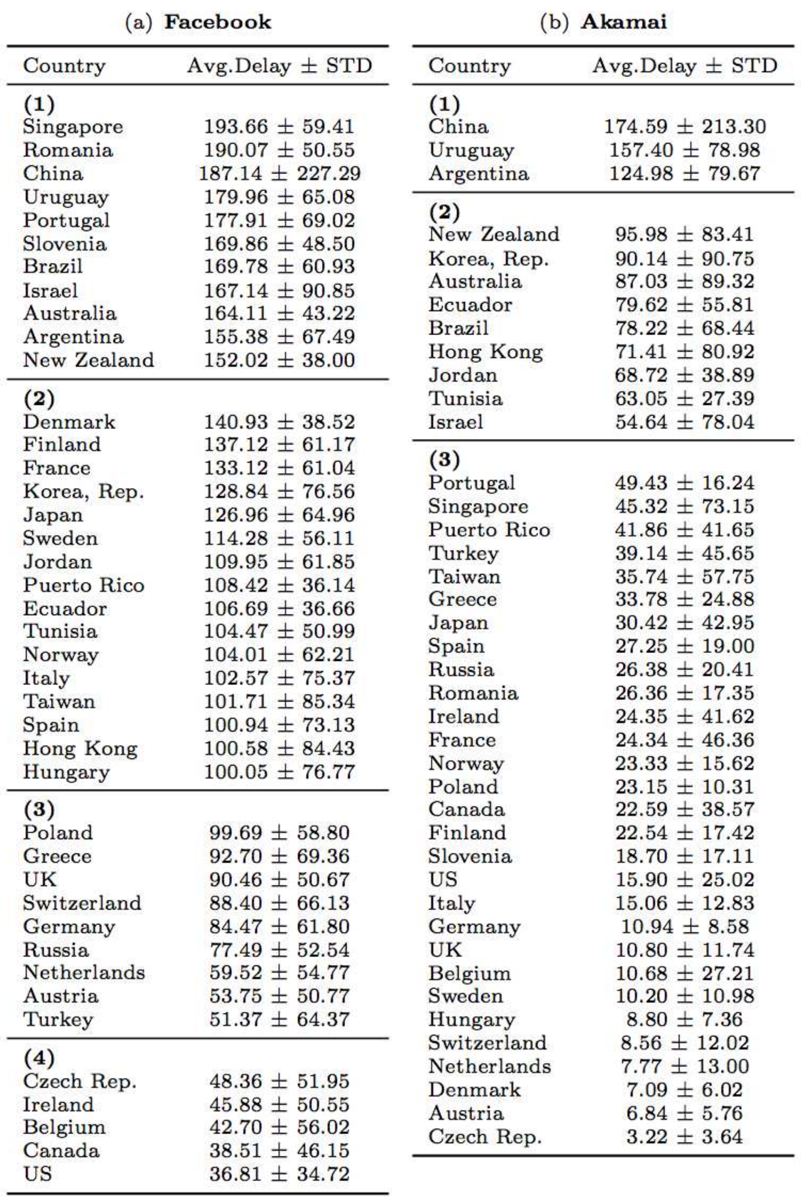
Table 3. Distribution of the 463 PlanetLab nodes per country.

Country	Acr.	#PL_nodes	Country	Acr.	#PL_nodes
United States	US	169	Argentina	AR	4
Germany	DE	40	Hungary	HU	4
China	CN	19	Korea, Rep.	KR	4
France	FR	18	Netherlands	NL	4
Italy	IT	16	Australia	AT	3
Poland	PL	16	New Zealand	NZ	3
Spain	ES	16	Norway	NO	3
Greece	GR	12	Singapore	SG	3
Japan	JP	12	Slovenia	SI	3
Switzerland	SZ	12	Turkey	TR	3
Canada	CA	11	Austria	AT	2
United Kingdom	UK	11	Czech Rep.	CZ	2
Belgium	BE	9	Jordan	JO	2
Brazil	BR	8	Puerto Rico	PR	2
Finland	FI	8	Russia	RU	2
Portugal	PT	8	Taiwan	TW	2
Israel	IL	6	Tunisia	TN	2
Sweden	SE	6	Denmark	DK	1
Hong Kong	HK	5	Ecuador	EC	1
Ireland	IE	5	Romania	RO	1
Uruguay	UY	5			

2.3.1.1.3 Access delay of end users to Facebook services

In this section we aim at understanding what is the performance experienced by end users in terms of latency to access Facebook services either located in native Facebook or Akamai servers. Table 4(a) shows the average access delay (and its standard deviation) per country required to access Facebook services in servers located in Facebook facilities, and Table 4(b) shows the same parameters referred to Akamai servers. Overall, Facebook users need 113ms in average to access native Facebook servers, but only 43ms to reach Akamai nodes providing Facebook access. This means that accessing Facebook services in Akamai nodes reduces 2.5× the delay. Next, we provide a detailed analysis of the delay access performance per country.

Table 4. Average delay (± Standard Deviation) to access Facebook services from different countries



(a) **Facebook**

Country	Avg.Delay ± STD
(1)	
Singapore	193.66 ± 59.41
Romania	190.07 ± 50.55
China	187.14 ± 227.29
Uruguay	179.96 ± 65.08
Portugal	177.91 ± 69.02
Slovenia	169.86 ± 48.50
Brazil	169.78 ± 60.93
Israel	167.14 ± 90.85
Australia	164.11 ± 43.22
Argentina	155.38 ± 67.49
New Zealand	152.02 ± 38.00
(2)	
Denmark	140.93 ± 38.52
Finland	137.12 ± 61.17
France	133.12 ± 61.04
Korea, Rep.	128.84 ± 76.56
Japan	126.96 ± 64.96
Sweden	114.28 ± 56.11
Jordan	109.95 ± 61.85
Puerto Rico	108.42 ± 36.14
Ecuador	106.69 ± 36.66
Tunisia	104.47 ± 50.99
Norway	104.01 ± 62.21
Italy	102.57 ± 75.37
Taiwan	101.71 ± 85.34
Spain	100.94 ± 73.13
Hong Kong	100.58 ± 84.43
Hungary	100.05 ± 76.77
(3)	
Poland	99.69 ± 58.80
Greece	92.70 ± 69.36
UK	90.46 ± 50.67
Switzerland	88.40 ± 66.13
Germany	84.47 ± 61.80
Russia	77.49 ± 52.54
Netherlands	59.52 ± 54.77
Austria	53.75 ± 50.77
Turkey	51.37 ± 64.37
(4)	
Czech Rep.	48.36 ± 51.95
Ireland	45.88 ± 50.55
Belgium	42.70 ± 56.02
Canada	38.51 ± 46.15
US	36.81 ± 34.72

(b) **Akamai**

Country	Avg.Delay ± STD
(1)	
China	174.59 ± 213.30
Uruguay	157.40 ± 78.98
Argentina	124.98 ± 79.67
(2)	
New Zealand	95.98 ± 83.41
Korea, Rep.	90.14 ± 90.75
Australia	87.03 ± 89.32
Ecuador	79.62 ± 55.81
Brazil	78.22 ± 68.44
Hong Kong	71.41 ± 80.92
Jordan	68.72 ± 38.89
Tunisia	63.05 ± 27.39
Israel	54.64 ± 78.04
(3)	
Portugal	49.43 ± 16.24
Singapore	45.32 ± 73.15
Puerto Rico	41.86 ± 41.65
Turkey	39.14 ± 45.65
Taiwan	35.74 ± 57.75
Greece	33.78 ± 24.88
Japan	30.42 ± 42.95
Spain	27.25 ± 19.00
Russia	26.38 ± 20.41
Romania	26.36 ± 17.35
Ireland	24.35 ± 41.62
France	24.34 ± 46.36
Norway	23.33 ± 15.62
Poland	23.15 ± 10.31
Canada	22.59 ± 38.57
Finland	22.54 ± 17.42
Slovenia	18.70 ± 17.11
US	15.90 ± 25.02
Italy	15.06 ± 12.83
Germany	10.94 ± 8.58
UK	10.80 ± 11.74
Belgium	10.68 ± 27.21
Sweden	10.20 ± 10.98
Hungary	8.80 ± 7.36
Switzerland	8.56 ± 12.02
Netherlands	7.77 ± 13.00
Denmark	7.09 ± 6.02
Austria	6.84 ± 5.76
Czech Rep.	3.22 ± 3.64

2.3.1.1.3.1 Access delay to native Facebook services

We have defined four groups regarding their access delay to Facebook services. (1) The first group refers to all those countries having an access delay longer than 150ms. This group is formed by very distant countries from US (e.g., Australia, New Zealand), South-American countries, and three countries we did not expect to find in this group such as Portugal, Slovenia and Israel since their surrounding neighbours show a considerably lower delay. (2) The second group is formed by those countries whose delay is ranging between 100ms and 150ms. This group includes Northern European countries, Asian countries with a deep penetration of high speed access connections (e.g., Japan, South Korea, Hong Kong), countries from central America and Mediterranean countries including some important European countries like France, Italy and Spain. (3) The third group includes those countries with a delay higher than 50ms but lower than 100ms. It is mainly formed by countries located in central Europe plus Greece, Turkey and UK. (4) Finally the last group contains those countries with access delay below 50ms. This includes those countries hosting native Facebook servers, US and Ireland [WIT01], and Canada due to its proximity and good connectivity to US.

Surprisingly, this group also includes Belgium and Czech Republic that intuitively had fitted better on the third group.

2.3.1.1.3.2 Access delay to Akamai caches

In the case of Akamai nodes we just define three groups for our discussion. (1) The first group is formed by only three countries that experience an average delay higher than 100ms. These countries are China, Argentina and Uruguay. This happens because an important portion of the Facebook queries from these countries are redirected to remote Akamai nodes located for instance in the US. (2) We have formed a second group with countries showing an average access delay ranging between 50ms and 100ms. We find Far East countries like Australia, New Zealand, South Korea and Hong Kong, two countries in South-America such as Brazil and Ecuador, and three countries from North-Africa and middle east, i.e., Jordan, Tunisia and Israel. The first six countries count with their own Akamai nodes but a relevant portion of their demand is attended from foreign Akamai servers. In addition, Jordan and Tunisia do not host any Akamai node, but are served by Akamai nodes located in Europe, which is relatively close. (3) Finally, we define a group including the countries with an access delay below 50ms. This group mainly includes developed countries from Europe, Asia (i.e., Japan and Singapore) and North America (US and Canada). This is a good estimation of a short-list of important countries for Facebook, where Facebook is interested on offering a better quality of service through Akamai nodes.

Furthermore, it is interesting to note that Akamai offers the best delay performance (i.e., below 10ms) to small countries roughly located in Central Europe (Hungary, Switzerland, Netherlands, Denmark, Austria and Czech Republic). This happens because these are very small countries (in size) that experience a very reduced delay (including the standard deviation) due to the short distance from any point of the country to a large number of Akamai nodes located in Central Europe. In contrast, countries such as Germany, France or US, which include a large number of Akamai nodes, has a large extension that makes some users (e.g., in remote towns) to incur in relatively high delays to access Akamai nodes, leading to a higher average delay (and standard deviation).

2.3.1.1.4 Akamai caches distribution to provide access to Facebook services

This section provides a global picture of the deployment of Akamai nodes to serve Facebook worldwide. Toward this end, we present two tables that summarize the measurement results and are the basis of our discussion in this section. Table 5 shows for each country which portion of the queries (i.e., ICMP echo) is managed by Akamai servers hosted in the same country than the source node(s) and which portion is served by Akamai nodes placed in foreign countries. In addition, Table 6 shows for each country hosting Akamai nodes, the overall number of IPs linked to Akamai nodes located in that country and the list of countries hosting nodes that access those IPs. For each source-querying country we represent the overall number of IPs (between brackets) accessed in the destination country hosting Akamai nodes. Therefore, we obtain an overview of how Akamai is covering Facebook subscribers demand.

Table 5. Portion of Facebook queries from each country served by local and foreign Akamai nodes.

Country	%Inside	%Outside	Country	%Inside	%Outside
Belgium	0	100	Czech Rep.	19.23	80.77
China	0	100	Russia	19.23	80.77
Denmark	0	100	Canada	24.36	75.64
Hungary	0	100	Austria	25	75
Jordan	0	100	Finland	25.99	74.01
Portugal	0	100	Brazil	26.59	73.41
Slovenia	0	100	Norway	26.92	73.08
Tunisia	0	100	Puerto Rico	26.92	73.08
Turkey	0	100	Israel	28.51	71.49
Uruguay	0	100	New Zealand	29.2	70.8
Poland	0.52	99.48	Spain	31.83	68.17
Ireland	7.75	92.25	Germany	34.6	65.4
Netherlands	9.64	90.36	Romania	34.62	65.38
Korea, Rep.	10.62	89.38	France	34.84	65.16
Japan	10.93	89.07	Switzerland	38.44	61.56
Hong Kong	11.53	88.47	Australia	38.46	61.54
Ecuador	11.54	88.46	Sweden	40.85	59.15
Italy	15.17	84.83	Taiwan	50	50
Greece	17.98	82.02	Singapore	62.3	37.7
UK	18.13	81.87	US	89.63	10.37
Argentina	19.23	80.77			

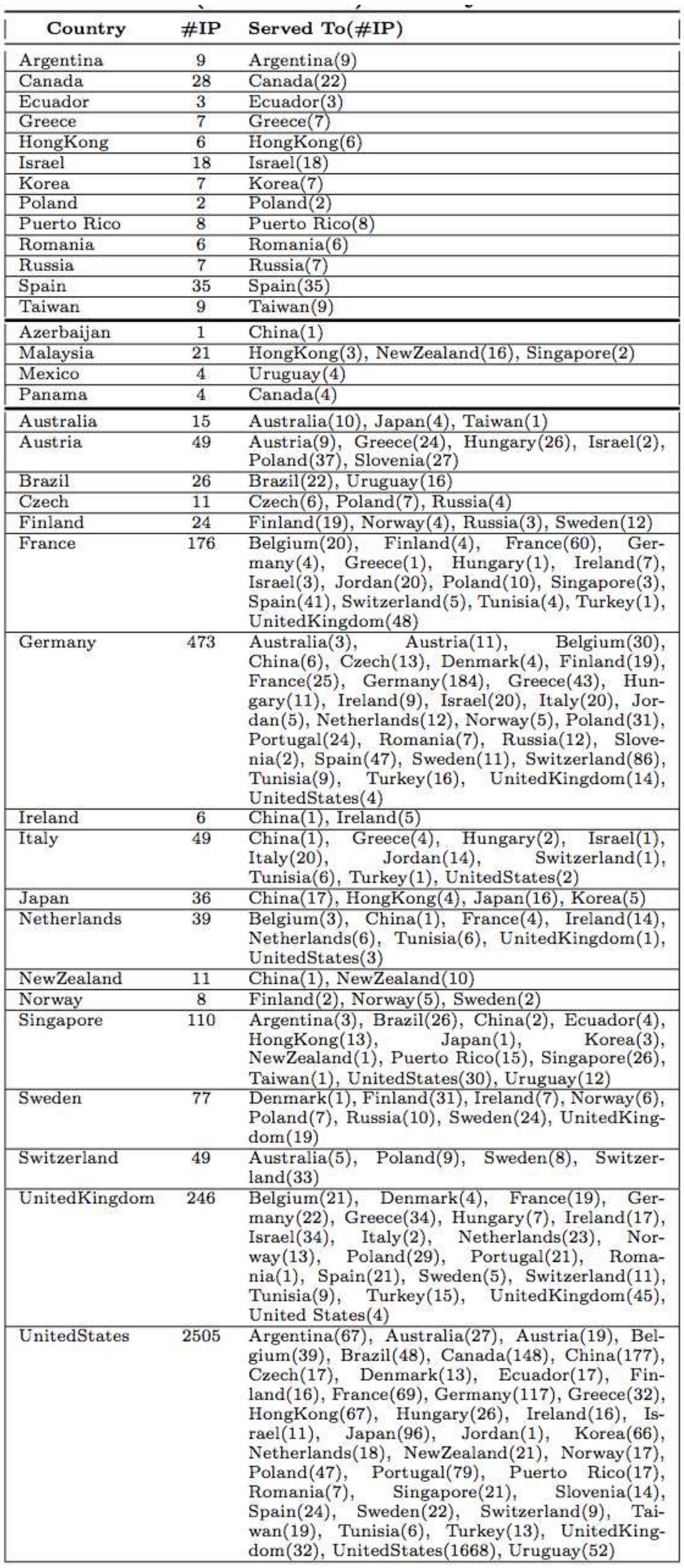
2.3.1.1.4.1 Local vs. External Access

There are only two countries showing a higher portion of local access to Akamai servers as compared to external access. These countries are US and Singapore with 90% and 62% of the queries going to local Akamai servers. The case of Singapore could be unexpected, but as we will show later, Singapore counts with a large number of Akamai nodes (i.e., IPs). This also explains that although Singapore is the country showing the largest access delay to native Facebook servers, it presents a similar delay to some European countries to reach Akamai nodes since a large portion of the queries are served locally. Close to Singapore performance, we find the case of Taiwan in which half of the queries are attended by local servers and half by foreign servers.

We find a reduced number of countries that keep between 30%-40% of their queries served by local Akamai nodes. These are: (i) the largest European countries by size (i.e., Germany, France and Spain) all of them counting with a large number of Akamai servers. (ii) Australia, which is also a large and remote (from native Facebook servers) country that probably counts with a large number of Facebook subscribers, and hence, Facebook is interested on using Akamai CDN to offer a good performance to Australian subscribers (iii) Three European countries such as Switzerland, Sweden and Romania, that are geographically distributed in Europe (center, north and east, respectively). Switzerland has a privileged position in the center of Europe and seems to be a good location for Akamai to place some servers since they could provide an efficient service to a large number of European Internet users (to reduce delay access to many different websites in addition to Facebook). Sweden shows a large presence of Akamai servers. In this case, the reason could be that it is a cold country, and hence, a proper location to place a data center and reduce cooling costs, and in addition it still provides a very good connectivity to northern, central and Eastern Europe. The Akamai infrastructures in Switzerland and Sweden bring them to have access delays to Akamai nodes in the order of 10ms. Finally, Romania presents just six Akamai servers that attend 35% of the queries generated in Romanian nodes. This helps Romania to keep an average access delay as small as the major part of the most representative European countries.

Next, we have a large number of countries keeping between 7% and 30% of its queries replied locally, while the major part is attended by foreign Akamai servers. All these countries have more or less Akamai nodes that allow keeping part of the queries locally, but their delay is mostly affected by how far are those Akamai nodes attending the major part of their queries.

Table 6. First column shows the list of countries hosting Akamai nodes offering access to Facebook services. Second column shows the number of Akamai-related IPs in each country. The third column shows the list of countries including nodes querying Akamai IPs in the country referred in the 1st column. The number between brackets reflects the number of IPs accessed in the reference (1st column) country.



Country	#IP	Served To(#IP)
Argentina	9	Argentina(9)
Canada	28	Canada(22)
Ecuador	3	Ecuador(3)
Greece	7	Greece(7)
HongKong	6	HongKong(6)
Israel	18	Israel(18)
Korea	7	Korea(7)
Poland	2	Poland(2)
Puerto Rico	8	Puerto Rico(8)
Romania	6	Romania(6)
Russia	7	Russia(7)
Spain	35	Spain(35)
Taiwan	9	Taiwan(9)
Azerbaijan	1	China(1)
Malaysia	21	HongKong(3), NewZealand(16), Singapore(2)
Mexico	4	Uruguay(4)
Panama	4	Canada(4)
Australia	15	Australia(10), Japan(4), Taiwan(1)
Austria	49	Austria(9), Greece(24), Hungary(26), Israel(2), Poland(37), Slovenia(27)
Brazil	26	Brazil(22), Uruguay(16)
Czech	11	Czech(6), Poland(7), Russia(4)
Finland	24	Finland(19), Norway(4), Russia(3), Sweden(12)
France	176	Belgium(20), Finland(4), France(60), Germany(4), Greece(1), Hungary(1), Ireland(7), Israel(3), Jordan(20), Poland(10), Singapore(3), Spain(41), Switzerland(5), Tunisia(4), Turkey(1), UnitedKingdom(48)
Germany	473	Australia(3), Austria(11), Belgium(30), China(6), Czech(13), Denmark(4), Finland(19), France(25), Germany(184), Greece(43), Hungary(11), Ireland(9), Israel(20), Italy(20), Jordan(5), Netherlands(12), Norway(5), Poland(31), Portugal(24), Romania(7), Russia(12), Slovenia(2), Spain(47), Sweden(11), Switzerland(86), Tunisia(9), Turkey(16), UnitedKingdom(14), UnitedStates(4)
Ireland	6	China(1), Ireland(5)
Italy	49	China(1), Greece(4), Hungary(2), Israel(1), Italy(20), Jordan(14), Switzerland(1), Tunisia(6), Turkey(1), UnitedStates(2)
Japan	36	China(17), HongKong(4), Japan(16), Korea(5)
Netherlands	39	Belgium(3), China(1), France(4), Ireland(14), Netherlands(6), Tunisia(6), UnitedKingdom(1), UnitedStates(3)
NewZealand	11	China(1), NewZealand(10)
Norway	8	Finland(2), Norway(5), Sweden(2)
Singapore	110	Argentina(3), Brazil(26), China(2), Ecuador(4), HongKong(13), Japan(1), Korea(3), NewZealand(1), Puerto Rico(15), Singapore(26), Taiwan(1), UnitedStates(30), Uruguay(12)
Sweden	77	Denmark(1), Finland(31), Ireland(7), Norway(6), Poland(7), Russia(10), Sweden(24), UnitedKingdom(19)
Switzerland	49	Australia(5), Poland(9), Sweden(8), Switzerland(33)
UnitedKingdom	246	Belgium(21), Denmark(4), France(19), Germany(22), Greece(34), Hungary(7), Ireland(17), Israel(34), Italy(2), Netherlands(23), Norway(13), Poland(29), Portugal(21), Romania(1), Spain(21), Sweden(5), Switzerland(11), Tunisia(9), Turkey(15), UnitedKingdom(45), United States(4)
UnitedStates	2505	Argentina(67), Australia(27), Austria(19), Belgium(39), Brazil(48), Canada(148), China(177), Czech(17), Denmark(13), Ecuador(17), Finland(16), France(69), Germany(117), Greece(32), HongKong(67), Hungary(26), Ireland(16), Israel(11), Japan(96), Jordan(1), Korea(66), Netherlands(18), NewZealand(21), Norway(17), Poland(47), Portugal(79), Puerto Rico(17), Romania(7), Singapore(21), Slovenia(14), Spain(24), Sweden(22), Switzerland(9), Taiwan(19), Tunisia(6), Turkey(13), UnitedKingdom(32), UnitedStates(1668), Uruguay(52)

Finally, we have 10 countries for which we could not identify any local Akamai server. That is Facebook or/and Akamai do not consider them critical countries for their interests, and Akamai can dedicate nodes in close by countries to attend their queries. We have five European countries (Belgium, Denmark, Hungary, Portugal and Slovenia) with a common characteristic, all of them have a small population below 10M. In addition, all of them are close to countries counting with an important deployment of Akamai nodes running Facebook services. Therefore, all of them present a rather short delay ac- cess to Facebook services through Akamai servers below 25ms, except Portugal that needs in average 49ms (see Table 4(b)). The reason behind this result is that Portugal is accessing Akamai severs in UK, Germany and US, which is surprising since Spain seems the best option to obtain a shorter access delay. In addition, we find Turkey, a big country with two clearly defined regions. The west region is close to Europe and thus to a large number of relatively nearby Akamai nodes, and the central-east region that is placed in the core of Asia surrounded by countries with a poor Internet access infrastructure like Armenia, Syria, Georgia, Iran and Iraq. Thus, we would expect to see a higher average delay and standard deviation than the ones reported in Table 4(b) for Turkey, which in fact are similar to some European countries like Greece or Portugal. The reason is that the three PlanetLab nodes used for our experiments are located in the west part of Turkey (i.e., Istanbul and Izmir).

Next, we discuss the case of Uruguay, a small South- American country surrounded by Argentina and Brazil that already counts with some Akamai servers. Interestingly, results in Table 6 show that the five PlanetLab nodes placed in Uruguay access Akamai servers located in Brazil, but also in Mexico and US that are far away, and never go to Argentina. Following, we find two small countries located in north of Africa, Tunisia, and the Middle East, Jordan. Both of them are served by Akamai nodes located (mainly) in Europe. Finally, we find China that is actually blocking Facebook, thus it does not make sense to deploy Akamai nodes to serve Facebook subscribers. In our experiments, ping commands performed from Chinese PlanetLab nodes go to Akamai nodes all over the world.

2.3.1.1.4.2 Country coverage by Akamai nodes

We have found 35 countries hosting Akamai nodes to provide Facebook access to the 41 countries represented by planetlab nodes. Among them, at the top of Table 6, we find 13 countries where Akamai nodes only serve local users. In the middle of the table we find four countries: Azerbaijan, Malaysia, Mexico and Panama, whose Akamai nodes only serve foreign countries. In fact, this behaviour responds to the fact that we do not have any PlanetLab node located in those countries. Otherwise, we would very likely observe that these Akamai nodes also serve local users. Finally, at the bottom of the table, we find a major part of the countries (18 in total) whose Akamai nodes attend queries from both local and foreign PlanetLab nodes. Next, we discuss the most interesting points for this group.

First, we observe that large countries with an important weight in the geopolitical environment such as US, UK, France, Germany and Italy has a large number of Akamai nodes (i.e., associated IPs) that serve a large number of countries. The four European countries mainly serve nodes from all over Europe, in minor level other nearby non-European countries like Israel, Jordan, Tunisia and Turkey, and in a very small scale US and China. We also found a similar pattern in the Netherlands, though it has a lower Akamai presence. Furthermore, we have discovered more Akamai nodes in US than in the rest of the countries together. These servers attend queries from users located all over the world. This clearly will impact the delay for those countries that despite being far from US, still access Akamai nodes in US for a large portion of their queries. This is the case of Uruguay, Argentina, China or Korea that present a large volume of Facebook queries resolved by Akamai nodes in US. Second, we observe that Akamai nodes in Northern European countries (Norway, Finland and Sweden) mainly attend the demand of users located within these northern countries. Third, Ireland and New Zealand should actually be located in the top of the table since they mostly attend local Facebook demand with some eventual queries coming from China. Fourth, Akamai nodes located in small Central European countries such as Austria, Czech Republic and Switzerland, attend Facebook demand mainly from local and nearby countries users. We can find a similar pattern in Japan and Brazil as well, and additionally in Australia, that mostly attends internal demand for Facebook services but also receives some queries from nodes located in Japan and Taiwan. Finally, Singapore (the 4th country in number of Akamai IPs) presents the more rare results. On the one hand, Akamai nodes in Singapore present an expected behaviour by serving users located in Asia. On the other hand, we discovered a very strange pattern in which Akamai nodes in Singapore attend quite a few nodes located all over America (including North and South-America).

In summary, we can conclude that Facebook subscriber queries are usually attended by Akamai nodes located either locally or in some nearby country. This provides a bounded access delay leading to the result presented in Section 2.3.1.1.3 that shows a delay 2.5x lower when a Facebook query is resolved by an Akamai node instead of a native Facebook server. However, we can still find some strange cases where source nodes are accessing Akamai nodes located far away which has a harmful impact in their access delay to Facebook services.

2.3.1.1.5 Related work

Authors in [WIT10] analyze the connections established when Facebook users login in the system. They identify different sections in the Facebook wall page of a user, and they analyze how the information filling those sections is retrieved. They conclude that different sections of the page are downloaded in parallel from different severs placed in different locations. However, this study is carried out only from a single PC in Thailand, thus the obtained conclusions are not valid to obtain global conclusion about Facebook infrastructure. An earlier work [WIT10] identified some performance degradation (e.g., delay, packet losses, etc) for some users accessing Facebook from outside US, and proposed a regional OSN cache system to improve the experience of those users. They evaluated the proposed system via simulation and conclude that it would reduce 70% the access latency, and use 91% less bandwidth. However, this research does not provide a global view of the current Facebook infrastructure deployment to serve users worldwide, since it only refers to five regions: Russia, Egypt, Sweden, New York and Los Angeles. Finally, we have found another interesting study [BEA10] that states that photo view is the most critical service for Facebook, and presents a detailed description on how Facebook photos are distributed to CDN Akamai servers. However, it does not study which regions are served by those nodes as we do in this work.

2.3.1.1.6 Conclusion

The research presented in this section describes a comprehensive measurement-based analysis of the current Facebook network infrastructure with a special emphasis on depicting how Akamai nodes replying Facebook queries from subscribers are distributed over the world. In this context, we have analyzed which is the average access delay that Facebook subscribers experience to access Facebook services delivered from native Facebook nodes as well as Akamai nodes depending on the country they are located. Moreover, we have thoroughly discussed the coverage offered by those Akamai nodes serving Facebook. We plan to extend this work by exploiting the current captured traces to understand what are the reasons driving a node to suffer different delay to access the same Facebook service over the time. Furthermore, we aim at studying whether Akamai performs some kind of load balancing by redirecting users to different nodes at different time periods.

2.3.1.2 Caching algorithms to populate the caches for dissemination of Twitter content

In order to evaluate the performance of the caching algorithms, a simulation tool has been developed based on discrete-event simulation model.

The simulation tool simulates the behaviours of main CDN components based on events generated by OSN users. The simulator generates a log file of the events that have occurred during the simulation process, this log file can later be analysed to evaluate the performance of the CDN. The inputs to the simulator are OSN user profiles, social graph (i.e. user → follower relationship file), a cluster file consisting of centroids of each cluster together with its members and configuration settings that include predefined thresholds and simulation run time. Figure 23 depicts the simulation organization.

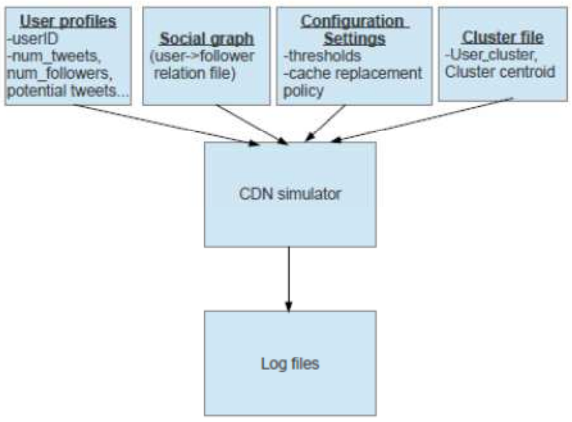


Figure 23. Simulation organization.

For our simulations we will employ two commonly used variants of cache replacement policies: LRF (Less Reused Filter) and LRU. In case of LRF the less recently arrived content, an object which has been on the cache for the longest time, is selected for eviction, whereas in case of LRU the content selected for eviction is the one that has been least recently used in the current cache.

Therefore, we evaluate the performance of our proposed algorithms with the simulation tool briefly discussed above using real user profiles and their social graph from our dataset. Table 7 presents the simulation setup we will be using to do a preliminary evaluation of the performance of proposed SACDN (Social-Aware CDN). For our simulation we have considered five distinct overlay topologies composed of 2, 8, 20, 50 and 100 surrogate servers using number of tweets as a surrogate placement parameter. In this report results from 8 and 50 surrogate serves are shown in the next section. UGC with uniform file size is assumed. In each topology the designed SACDN is compared with pull based, push based and dummy scheduling algorithms defined as follows:

- **Pull:** upon user request, if the content is not available in user's home server (i.e., in case of a cache miss) the content is cached in the local surrogate anticipating subsequent requests for the content.
- **Push:** We simulate a push algorithm in which content is cached in all surrogate servers as soon as it is uploaded.
- **Dummy:** All content is retrieved from the original server where the content was uploaded to. In this approach no caching mechanism is used.

Table 7. Simulation Settings

INPUT	VALUE
FOLLOWER	Threshold 2
Total number of users	10,000
Total number of edges	302,919
Cache replacement policy	LRF
Surrogate placement criteria	Number of tweets

To compare the performance of various alternatives we will investigate two important performance metrics: HR and cost associated with each simulation. To measure the cost associated with each simulation we will consider cost of content distribution (cost associated with replicating content between surrogates) and cost of content access by users.

- HR is the fraction of local accesses (cache hits) to the total number of requests, using this metric we measure which fraction of user requests are served from user's home server. A high HR indicates an effective content distribution algorithm. By definition if the cache size is sufficient to host all files, the push strategy is expected to have a HR = 1, since it pushes every content to all surrogates. However, it is expected that it will imply a higher cost. HR for a content *j* from surrogate *i* is given by the following formula:

$$hit\_ratio\left(content_j^i\right) = \frac{cache\_hits(content_j^i)}{cache\_hits(content_j^i) + cache\_missess(content_j^i)}$$

- The cost of distribution as a function of geographic distance between surrogates to measure how expensive it is to fetch an object. The cost function is approximated with the logarithm of geographic distance between surrogates. The distance, d, between two surrogates *a* and *b* with given coordinates (lat, lon) is given by the following haversine formula[4].

$$d = 2R * \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta lat}{2}\right) + \cos(a_{lat}) * \cos(b_{lat}) * \sin^2\left(\frac{\Delta lon}{2}\right)}\right)$$

(R= radius of earth ≈ 6371KM)

2.3.1.2.1 Case study of 2 surrogates

The content distribution infrastructure to be investigated here is a case where the OSN is partitioned into two regions (clusters) where each cluster is primarily being served by a surrogate near its centroid. The "followers threshold" is set to two. This threshold is only used by social-aware algorithms, as a result during content distribution phase the social aware algorithm copies a content item to a given surrogate only if the target surrogate is home server for more than two followers of a user generating the content. During a cache miss event the social aware algorithm computes the follower threshold of both uploader and the user requesting the content and a content item is fetched to the local surrogate only if both users have more than two followers in the surrogate.

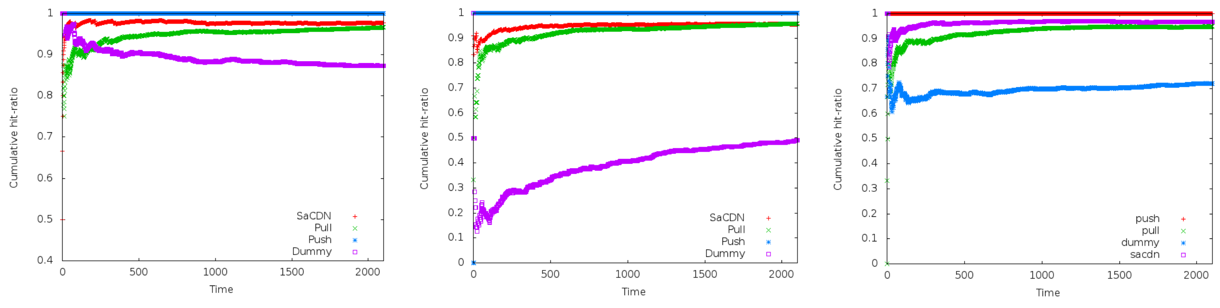
[4] http://en.wikipedia.org/wiki/Haversine_formula

Hit ratio

The first metric investigated is HR, which is measured from the simulation logs as follows:

```
set cumulative-hit=0
set cumulative-miss=0
set event-counter=0
for a given cluster on each event:
if Event type is of type 'Hit' or 'miss'
      event-counter+=1
      if this event type is 'hit'
            cumulative-hit+=1
      else if this event-type is 'miss'
            cumulative-miss+=1
hit-ratio=cumulative-hit/(cumulative-hit + cumulative-miss)
```

From the simulation results shown on Figure 24 we can observe the intuitive fact that the dummy algorithm always has the lowest HR, because dummy based algorithms never fetch contents in any event. Focusing on the evolution of SACDN, we can observe that it always maintains a steady HR of around 0.9 throughout the simulation run. The pull-based algorithm, on the other hand, increased its HR from a lower value of between 0.6 and 0.7 progressively to an equivalent ratio as SACDN and the push based algorithms. This progressive increase is as a result of pull events that occur due to initial misses experienced by users that requested the missing contents. We can also observe that, as it is expected, the push based algorithm steadily maintains highest HR as long as there is enough cache size to accommodate all contents. From our experiment, we found that the cumulative HR accumulated by surrogate 1 exceeds that of surrogate 2, this is due to the fact that surrogate 1, which is the American continent, is populated by a larger population of active users that have a relatively lower friends that belong to the other surrogate, on the other hand, users on surrogate 2 follow more users in Americas requesting contents from the first surrogate, which results in a lower HR on surrogate 2.



(a) Surrogate 1 (b) Surrogate 2 (c) Both Surrogates

Figure 24. HR per surrogate: case of 2 surrogates.

Another interesting observation we found is that, when the amount of content on a given surrogate exceeds the maximum size allocated to the surrogate, the HR exhibits an interesting behaviour different from the one demonstrated above on Figure 24(a), which is found to be the result of cache replacement. Even though one may expect a higher HR from the push mechanism, pushing content to all surrogates while the cache size gets exhausted, results in surrogates being preoccupied with contents that may not be required by users assigned to that surrogate. As a result in each event of user upload the distribution component of push based CDN blindly replicates the content to every surrogate evicting an existing content from list of contents already available in the cache of the surrogate. The measured cumulative HR of the four algorithms is presented in Table 8.

Table 8. Comparison of hit-ratio as cache becomes full.

Algorithm	Hit-ratio on surrogate1	Hit-ratio on surrogate2
push	0.611	0.460
pull	0.990	0.976
dummy	0.708	0.448
SACDN	0.963	0.926

From the table we can observe that the pull based CDN has recorded a higher HR, followed by SACDN, push, and dummy. For this simulation scenario we have set a pull threshold to two followers per cluster. As a result in the event of cache miss, pull CDN will automatically copy requested content from the remote surrogate, while SACDN, computes the number of followers of the content generator, and also the user that requested content, SACDN copies the content only if that threshold is greater than 2. To investigate this scenario further lets observe the temporal evolution of cumulative HR by all surrogates when the surrogate cache is substantially occupied. The simulation result is depicted in Figure 25. From the figure we can observe that initially push based algorithm had the highest HR followed by SACDN, pull and dummy; however, as time advances the HR of the pull based algorithm increases to a higher value and maintains a steady HR then SACDN, while push based algorithm dramatically decreased its HR, eventually maintaining the same HR as that of dummy algorithm. This is due to the fact that push based algorithms are continuously distributing each upload to both surrogates by removing contents already pre-fetched to the surrogate. On the other hand pull and SACDN based algorithms only pre-fetch interesting contents.

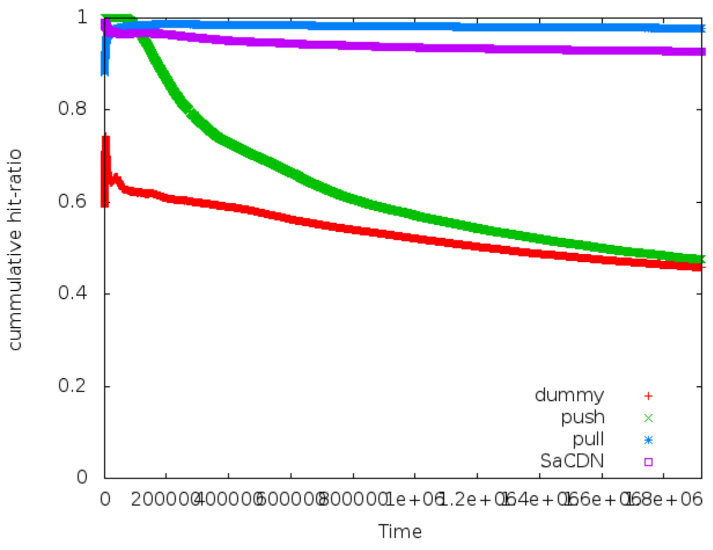


Figure 25. Cumulative HR as cache exceeds capacity.

Cost of content distribution

In this section the cost of distributing content between surrogates is investigated. The algorithms evaluated are pull based, push based, and SACDNs. Dummy CDN is not considered for this investigation as dummy does not involve content replication between surrogates; instead, each user request is served only by the home server of the user that generated the content - irrespective of the location of the user.

The cost of distribution is calculated from the events found in the generated log file of each simulation trace as follows: from the simulation run, on each event if the event involves a content transfer between the two surrogates a unit cost is added to the cumulative cost accumulated by the algorithm.

Table 9 and Table 10 present the total cumulative cost of distribution associated with each algorithm and the total cost. From the table we can see that the algorithm that used push based CDN involved the highest cost of 11772, followed by push based algorithm, with SACDN incurring the lowest cost. We can also see that SACDN reduces the cost of distribution by 15.86% and by 70.6% as compared to pull based algorithms and push based algorithms respectively.

Table 9. Cost comparison: case of two surrogates.

Algorithm	Distribution Cost	Access Cost
push	11772	0
pull	4104	14872
SaCDN	3453	10936
Dummy	-	188023

Table 10. Cost comparison (cost scaled to 0-100): case of two surrogates.

Algorithm	Distribution Cost	Access Cost	Total Cost
push	53,66	0	53,66
pull	18,71	6,91	25,62
SaCDN1	15,74	5,08	20,82
SaCDN2	11,9	0,63	12,53
dummy	0	87,37	87,37

The evolution of the content distribution cost is shown Figure 26. In addition to maintaining a very low distribution cost on a level similar with pull based algorithm, the SACDN has maintained a higher and steady HR which dictates small latency.

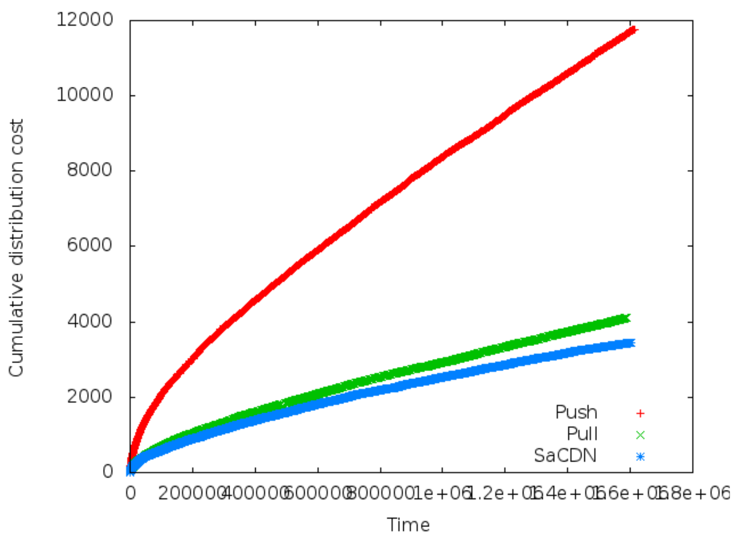


Figure 26. Comparison of content distribution cost: case of two surrogates.

Cost of content access

The cost of content access is calculated from the events found in the generated log file of each simulation trace as follows: from the simulation run, on each event if a content request is served by the user's home server (i.e., when there is a cache hit) we assume cost=0, but if the content request is not available in the local surrogate (i.e., when there is a cache miss) and the request is served from the other surrogate a unit cost is recorded. The algorithms evaluated are dummy, pull based, push based, and SACDNs. The last column on

Table 9 presents the total cumulative cost of access associated with each algorithm.

From the table we can see the evident fact that push based algorithms incurred the lowest access cost of 0, while dummy algorithm incurred the highest cost of 188023, followed by pull based algorithm, then the SACDN. We can also see that the SACDN reduces access cost by 26.46% and by 94.18% as compared to pull based algorithms and dummy based algorithms respectively.

2.3.1.2.2 Case study of 8 surrogates

The content distribution infrastructure to be investigated here is a case when there are 8 zones of the OSN, where a user in each zone is primarily served by a surrogate near its centroid. The geographic locations of surrogates are shown in Figure 27. As in the previous case, we will investigate the performance of our designed algorithms against the three content distribution mechanisms listed earlier. The two performance metrics: HR and cost associated with content transfer will be evaluated. To measure the cost associated with each simulation we will consider cost of content distribution (cost associated with replicating content between surrogates) and cost of content access by users.

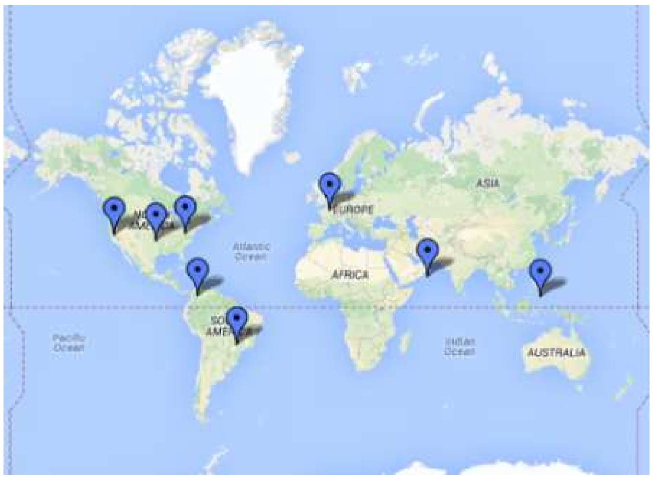


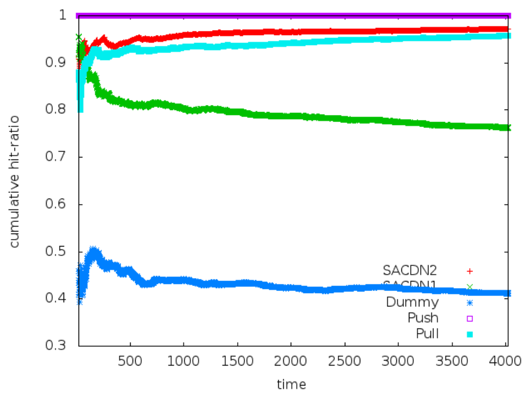
Figure 27. Geographic Location of 8 Surrogates

To investigate the effect of follower thresholds, we will consider two simulations of SACDN, SACDN1 and SACDN2, varying the follower threshold in computing pull threshold. For SACDN1, the simulation settings for case of two surrogates will be kept unchanged, for SACDN2 in computing pull threshold the follower threshold for both content generator and requester is set to zero. As a result SACDN2 pulls content to a local surrogate any time a cache miss event occurs, irrespective of number of followers of the user that requested the content. This scenario can be considered as a social-enhanced pull algorithm, unlike pull algorithm that copies content only on the event of cache miss. Pulling a requested missing content item has an advantage to the users as it reduces latency if users re-request a previously consumed content. When a user uploads content, like SACDN1, SACDN2 also pushes the uploaded content to a surrogate if there are at least two potential users that might be interested in it. This approach reduces the latency experienced by first users that will request a content item not available in their home server; however, it is expected to have a higher cost as compared to SACDN1 that selectively pulls contents.

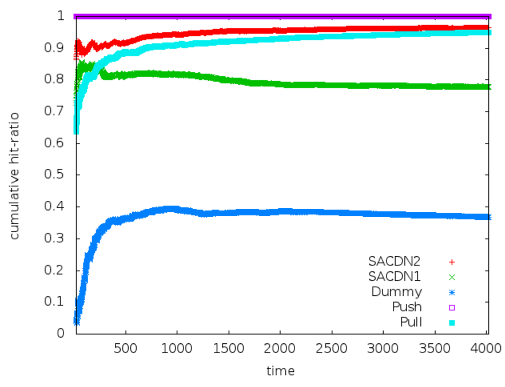
Hit ratio

We follow the same procedure of calculating the HR as in the case of 2 surrogates. While evaluating the simulation logs for the case of 8 surrogates, a cache hit is recorded if a user request is served from its home server, if instead a content is not found in user's home-server and the request is served from a remote surrogate a cache miss is recorded. As shown in Figure 28(a) to Figure 28(h), SACDNs scored the highest HR in the range [0.85, 1), closer to the push based algorithm (that always scores 1) as simulation time progresses.

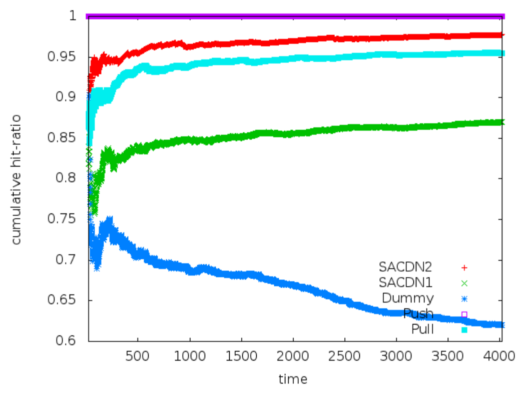
The difference in evolution of HR between surrogates offers a significant insight, for example, dummy algorithm shows better performance in surrogates 2 and 4 placed in California and Oman respectively, as compared to other surrogates. The relatively higher HR in Oman can be partially attributed to the language and cultural similarity between users in the area. Among surrogates placed in the US, the surrogate in California has scored a higher HR, which can be explained in part by the area being prominent for its places as the home of the entertainment and hi-tech industries such as Silicon Valley, Hollywood and Las Vegas and the concentration of celebrities in the area. Though placed in the US, Surrogates 5 and 6 experience a lower HR, again this can be due to the reason that users in this area might as well be interested in content items generated by "friends" from the Californian surrogate. We can also see that users in this area are highly benefited from the pre-fetching feature of SACDN2, as SACDN2 pushes UGC to a surrogate only if the generator has at least two followers in the surrogate. Users served from surrogate 8 (in Brazil) did not benefit much from the pre-fetching feature of SACDN2 this may indicate a conversation between users assigned to the surrogate and less popular users from other surrogates.



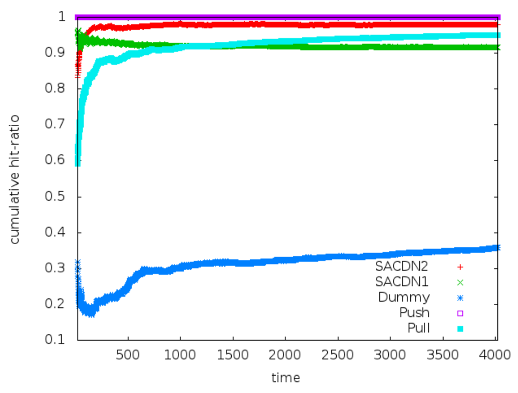
(a) Surrogate 1 (Columbia)



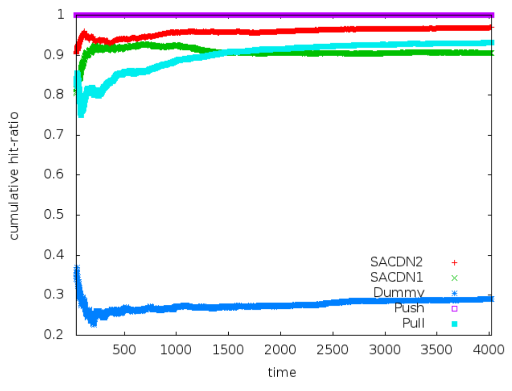
(b) Surrogate 2 (Oman)



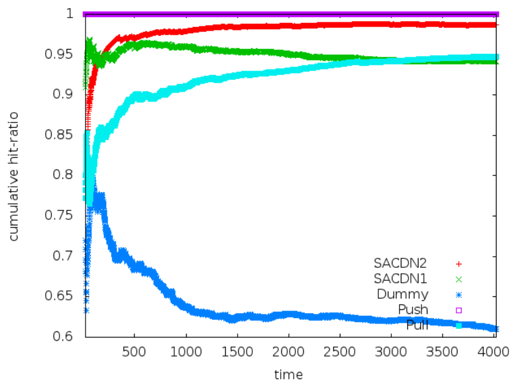
(c) Surrogate 3 (Philippines)



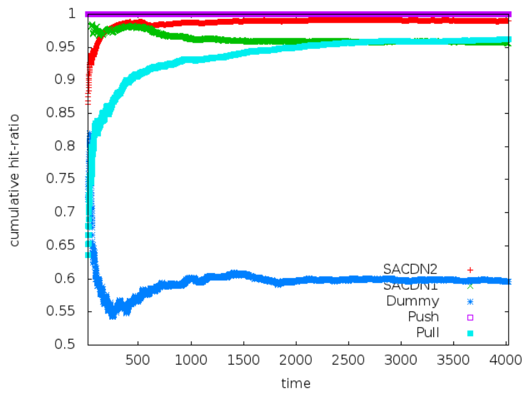
(d) Surrogate 4 (CA, USA)



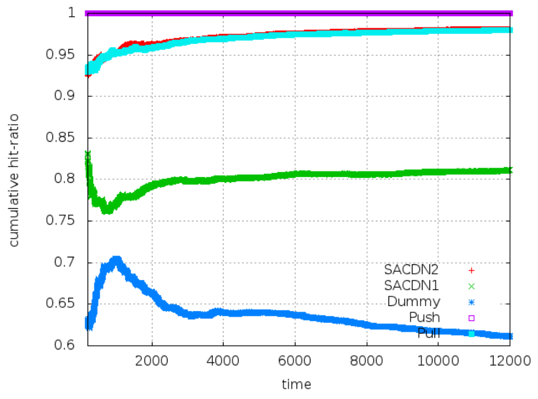
(e) Surrogate 5 (OK, USA)



(f) Surrogate 6 (WV, USA)



(g) Surrogate 7 (France)



(h) Surrogate 8 (Brazil)

Figure 28. per surrogate hit-ratio: case of 8 surrogates.

Cost of content distribution

To calculate cost of content distribution, the same approach as in the case of two surrogates will be followed. However, since there are 8 surrogates distributed geographically, the distance formula will be applied to reflect the proximity between surrogates. Hence, in the event of content replication between surrogates the logarithm of distance between surrogates will be recorded as the cost associated in doing so. The last two columns on Table 11 present the number of distribution events that occurred and the associated distribution cost respectively.

Table 12 shows the cost of content distribution associated with each surrogate.

Table 11. Cost Comparison: case of 8 surrogates

Algorithm	Miss Events	Access Cost	Distribution Events	Distribution Cost
push	0	0	81795.000	316283.059
pull	11139.000	41802.900	11139.000	41802.900
SaCDN1	65746.000	249991.077	5873.000	21848.537
SaCDN2	5332.000	20215.049	11165.000	41882.134
dummy	408663.000	1511461.150	0	0

Table 12. Content Distribution Cost Comparison: Case of 8 Surrogates

Surrogate	push	pull	SACDN1	SACDN2	Dummy
1	5937.704	1358.608	453.307	20672.172	0
2	12463.094	2961.343	1152.015	21153.597	0
3	38450.120	4722.584	1137.904	21374.180	0
4	54040.973	7558.639	5571.691	26196.555	0
5	32628.527	7080.646	2353.420	22642.492	0
6	75769.654	9399.512	6545.813	75769.654	0
7	88714.087	7542.585	4284.843	88714.087	0
8	8278.899	1178.983	349.545	20522.348	0

Cost of content access
On each event of content request if that request is served from user's home server, we assume cost=0, but if the content is served from a remote surrogate the associated cost is the logarithm of the geographical distance between surrogates, using the distance formula presented earlier. The miss event column on Table 11 presents the number of cache miss events associated with each simulation; while the “Access Cost” column is the total content access cost resulted from each algorithm. Table 13 shows the content access cost per each surrogate. In the subsequent sections scalability of the designed algorithms will be investigated by increasing the number of surrogates to 8, 20, 50 and 100s.

Table 13. Content Access Cost Comparison: Case of 8 surrogates

Surrogate	Push	Pull	SACDN1	SACDN2	Dummy
1	0	1358.608	13686.787	1050.394	29749.273
2	0	2961.343	25543.513	1997.012	77564.641
3	0	4722.584	35841.134	2847.380	128339.690
4	0	7558.639	35182.348	3178.730	266122.671
5	0	7080.646	40071.931	3531.105	220608.891
6	0	9399.512	49067.901	3697.177	405254.080
7	0	7542.585	37353.335	2841.850	359367.490
8	0	1178.983	13244.127	1071.401	24454.415

2.3.1.2.3 Case Study of 20 surrogates

In this simulation scenario the number of surrogates is scaled to 20, serving each zone of the OSN where the OSN is partitioned into 20 zones (i.e. clusters) with surrogates placed near cluster's centroid. The CDN algorithms (SACDN1, SACDN2, Push, Pull and Dummy) that were investigated for the case of 8 clusters will be investigated again for the case of 20 surrogates using the same performance metrics.

Hit ratio
Figure 29 presents the evolution of cumulative hit ratio along the simulation run.

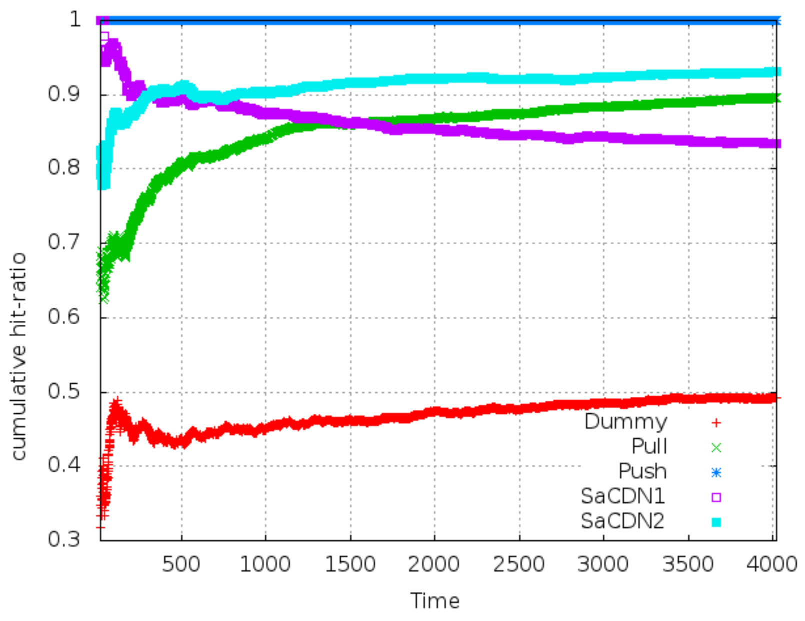


Figure 29. Comparison of cumulative hit ratio (case of 20 surrogates)

Cost of content distribution

The last two columns on Table 14 and Table 15 present the number of distribution events that occurred and the associated distribution cost respectively.

Cost of content access

The “Miss Events” column on Table 14 presents the number of cache miss events associated with each simulation, while the “Access Cost” column is the total content access cost resulted from each algorithm.

Table 14. Cost comparison (case of 20 surrogates).

Algorithm	Miss Events	Access Cost	Distribution Events	Distribution Cost
push	0	0	219697	855881.47
pull	11489	44300.09	11489	44300.09
SACDN1	247848	947356.40	11109	43275.67
SACDN2	10232	39310.83	22270	86257.11
dummy	403601	1551455.11	0	0.00

Table 15. Cost comparison (cost scaled to 0-100): case of 20 surrogates.

Algorithm	Distribution Cost	Access Cost	Total Cost
push	83.04	0.00	83.04
pull	4.30	1.72	6.02
SACDN1	4.19	36.68	40.87
SACDN2	8.37	1.52	9.89
dummy	0.00	60.08	60.08

2.3.1.2.4 Case Study of 50 surrogates

In this simulation scenario the number of surrogates is further scaled to 50, serving each zone of the OSN, where the OSN is partitioned into 50 zones (clusters) with surrogates placed near cluster's centroid, the surrogate locations are shown in Figure 30. The CDN algorithms (SACDN1, SACDN2, Push, Pull, Dummy) that were investigated for the case of 8 clusters will be investigated again for the case of 50 surrogates using the same performance metrics and methodologies.

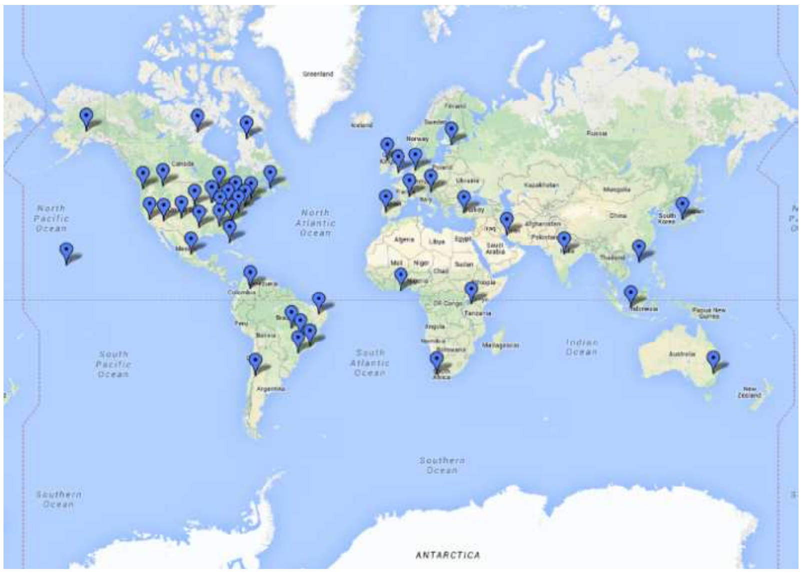


Figure 30. Geographic Location of 50 Surrogates

Hit ratio

Figure 31 presents the evolution of cumulative hit ratio along the simulation run.

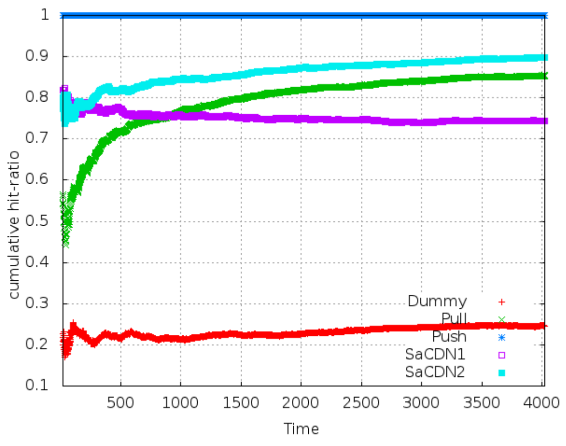


Figure 31. Comparison of Cumulative Hit Ratio: case of 50 surrogates

Cost of content distribution

The last two columns on Table 16 present the number of distribution events that occurred and the associated distribution cost respectively.

Table 16. Cost Comparison: case of 50 surrogates

Algorithm	Miss Events	Access Cost	Distribution Events	Distribution Cost
push	0	0	488888	1834712
Pull	24419	86552	24419	86552
SACDN1	175687	625744	14580	51485
SACDN2	15802	56504	25016	88578
dummy	546730	1928175	0	0

Cost of content access
The "Miss Events" column on Table 16 presents the number of cache miss events associated with each simulation; while "Access Cost" column is the total content access cost resulted from each algorithm.

2.3.1.2.5 Case Study of 100 surrogates

In this last simulation scenario the number of surrogates is further increased to 100, serving each zone of the OSN, where the OSN is partitioned into 100 zones (clusters) with surrogates placed near cluster's centroid. The CDN algorithms (SACDN1, SACDN2, Push, Pull and Dummy) that were investigated for the case of 2, 8, 20 and 50 clusters will be investigated again for the case of 100 surrogates using the same performance metrics and methodologies.

Hit ratio
Figure 32 presents the evolution of cumulative hit ratio along the simulation run.

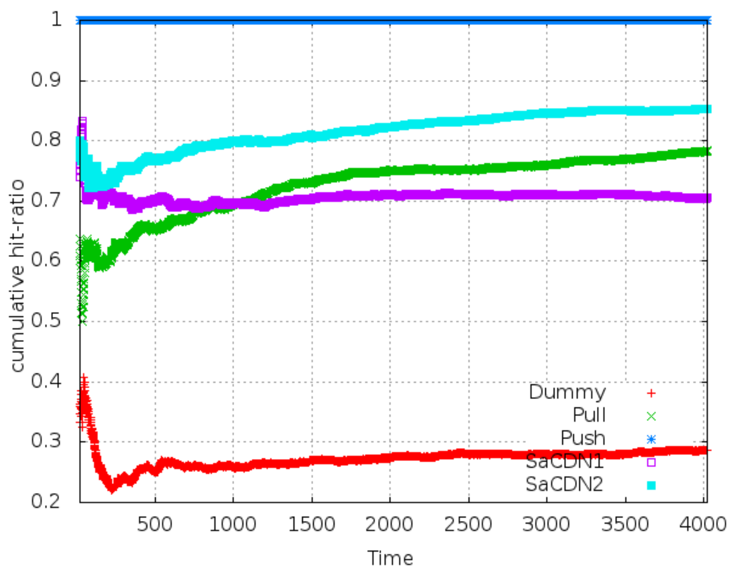


Figure 32. Comparison of cumulative hit-ratio: case of 100 surrogates.

Cost of content distribution
The last two columns on Table 17 present the number of distribution events that occurred and the associated distribution cost respectively.

Cost of content access

The “Miss Events” column on Table 17 presents the number of cache miss events associated with each simulation, while the “Access Cost” column is the total content access cost resulted from each algorithm.

Table 17. Cost comparison: case of 100 surrogates

Algorithm	Miss Events	Access Cost	Distribution Events	Distribution Cost
push	0	0.00	722674	2680815.09
pull	28219	98319.92	28219	98319.92
SACDN1	225089	792390.69	14821	51355.65
SACDN2	21891	77026.31	30668	107038.06
dummy	574586	1990349.77	0	0.00

Table 18. Cost comparmparison (cost scaled to 0-100): case of 100 surrogates.

Algorithm	Distribution Cost	Access Cost	Total Cost
push	91.26	0.00	91.26
pull	3.35	3.32	6.67
SACDN1	1.75	26.79	28.54
SACDN2	3.64	2.60	6.24
dummy	0.00	67.28	67.28

2.3.1.2.6 Summary

In this section we have performed a preliminary evaluation of the designed social aware content distribution algorithm, scalability of the algorithm is demonstrated by increasing the number of surrogates, and we also compared it against existing CDN algorithms. From our experiments we found the following important observations:

- From the users' perspective push based algorithms offer the lowest cost of content access as shown in Figure 33, this cost can be interpreted as lowest content access latency experienced by users as they consume UGC. However, push based algorithm incurs a very high content distribution cost that proportionally increases as the number of surrogates is increased; this behaviour is shown on Figure 34.

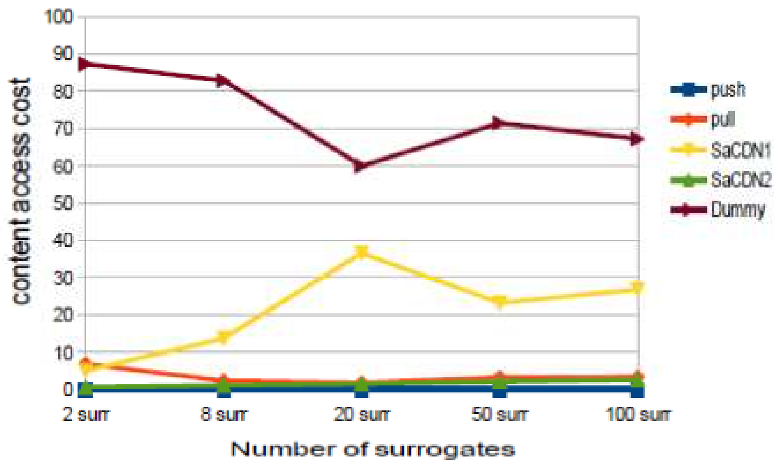


Figure 33. Comparison of content access costs.

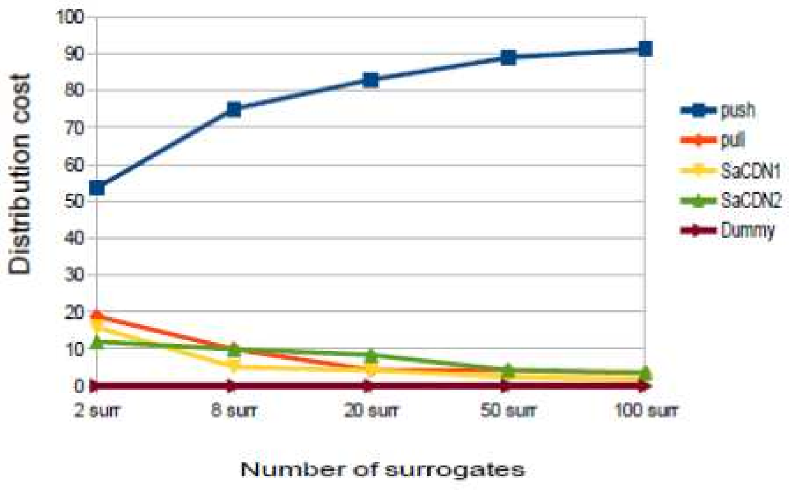


Figure 34. Comparison of content distribution costs.

- On the other hand, the dummy algorithm, which always serves contents from their origin without replicating contents between different surrogates, is associated with zero content distribution cost, but this algorithm introduces a very high cost of access to users since requested content must traverse long distances (see Figure 33).
- Pull based algorithm, even though it exposes the first users that request contents not available in their home servers to a higher content access cost, has shown to offer a high hit ratio and reduced network cost. Pull based algorithm is found to be more suitable for distributing UGC as compared to push based and dummy algorithms.
- We have shown that by tuning threshold parameters of SACDN, one can achieve a steady and very high HR between 0.8 and 1, with a very low access cost as comparable as that of push based algorithm but with a very low distribution cost at the scale of pull based algorithms.
- Push based algorithm introduces unnecessary distribution cost by replicating UGC to all surrogates even if there are no followers of the content generator in some surrogates. This extra cost of content distribution by push based algorithms can be avoided with SACDN by assigning a value 1 to the follower threshold in computing the push threshold, this threshold avoids replicating UGC in areas where the content will not be requested.

In computing Pull threshold, setting a Follower threshold of 0 helps SACDN to make the UGC unconditionally to a follower in the event of content re-request.

2.3.2 Small TELCO CDN caches

2.3.2.1 Exploiting Lead Users for caching

2.3.2.1.1 Introduction

In deliverable D3.1 [D3.1] it was explained and illustrated, by means of the analysis of two real datasets (a VOD (Video On Demand) dataset and a CUTV dataset), that video content consumption shows strongly different patterns amongst users and that so-called "lead users" can be identified. As explained in that deliverable, lead users are characterized by a typical 'early' consumption of new content, and by having many 'followers' that consume the same content at a later stage.

It was further suggested in deliverable D3.1 [D3.1] that this lead user concept might have a potential to improve caching systems, provided suitable metrics for the detection of such lead users can be identified.

In the current Deliverable D4.2 we report on the results obtained after checking this hypothesis by means of a variant of MLRU (Modified Least Recently Used), applied to the VOD dataset described and used in deliverable D3.1 [D3.1]. MLRU has been studied in an earlier project; in [OCE11] it is reported that MLRU improves the HR compared to LRU, especially for smaller caches.

Characteristic for MLRU – in comparison with LRU – is that it does not put a new content item at the head of the cache, but at an intermediate position, determined by the so-called jump parameter J. In case of a hit, the content item is promoted again by a number of positions compared to its' current position, and depending on the value of J.

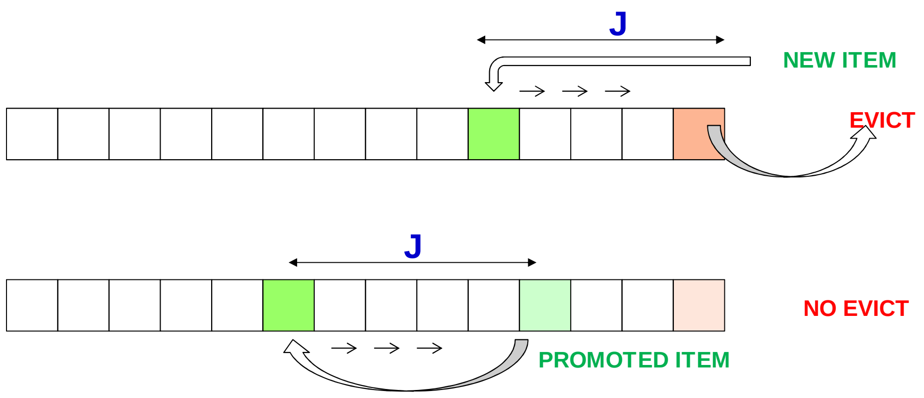


Figure 35. Operation of MLRU.

Further on we express J as a fraction of the total cache size. The illustration in Figure 35 shows an example with J around 35%. Note that J = 100% corresponds with standard LRU.

2.3.2.1.2 Differentiated MLRU – dMLRU

In eCOUSIN we intend to exploit the degree of freedom offered by this jump parameter J to assign differentiated values of J to lead users and non-lead users. Essentially, the idea is to put a new content item requested by a lead user in a more frontal position of the cache, since there is a fair chance that other users will consume the same content item soon thereafter. For non-lead users, a lower value for J is used. The mechanism is illustrated in Figure 36 for the case that a requested content item is not in the cache: in case the requesting user is a lead user, the system uses J1; otherwise it uses J2. In case the requested item is already in the cache, illustrated in Figure 37, J1 is used to promote the content item in case it is requested by a lead user; otherwise J2 is used.

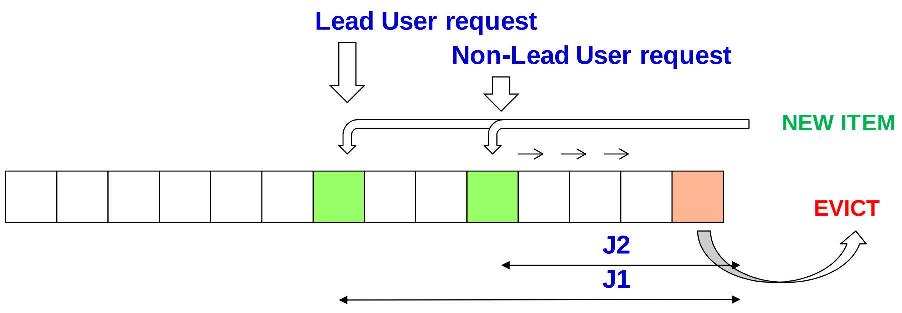


Figure 36. dMLRU – Placement of new item

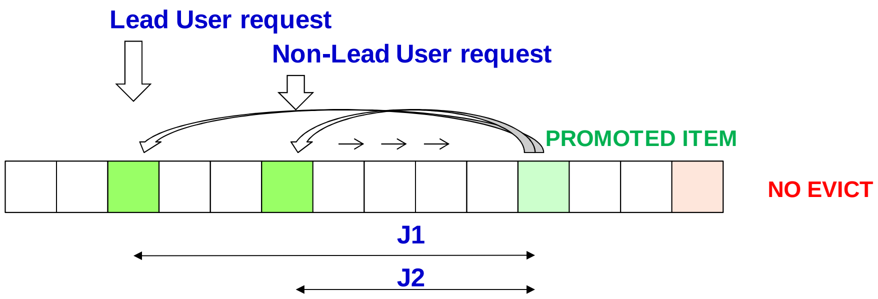


Figure 37. dMLRU – Promotion of cached item

2.3.2.1.3 Application to the VOD dataset - Simulations

In order to evaluate the effectiveness of dMLRU, it has been applied to the VOD dataset described in deliverable D3.1 [D3.1]. The main characteristic for this dataset is the diurnal consumption pattern with an outspoken peak during the evening hours between 20h00 and 22h00. Since the network supporting the VOD service is expected to be dimensioned in function of this peak traffic, our analysis focuses on the comparison of the hit ratio of LRU, MLRU and dMLRU during the peak hour. This is because the cache HR is a good measure for the traffic gain in the network.

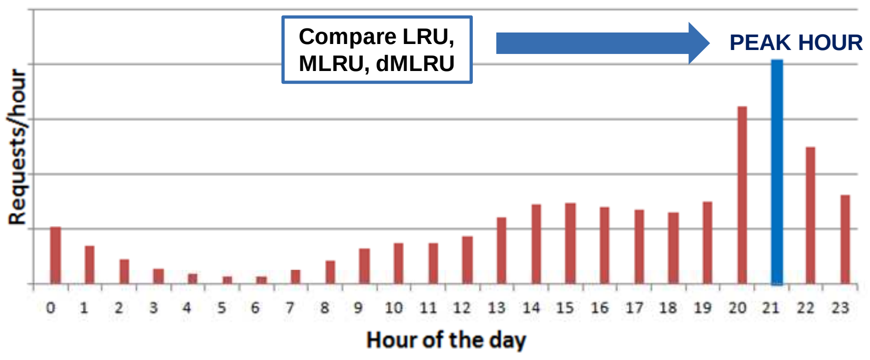


Figure 38. Hit ratio at PEAK HOUR as basis for traffic gain

The used criterion for determining lead users is based on the comparison of the number of *firsts* generated by a user with the number of *follow-ups* attributed to the same user. For a full description of *firsts* and *follow-ups* please refer to deliverable D3.1 [D3.1]. We briefly repeat here that a *first* refers to the first view of a certain content item, whereas a *follow-up* refers to all subsequent views of that same content item and is attributed to the same user that generated the *first*.

The users are ranked based on the ratio *follow-ups/firsts.* The first N users are defined to be lead users. Each content item requested for the first time by such a lead user is placed in the cache according to jump parameter J1, and this is extended to the 2nd, 3rd, ... M-th request for that content item (in case M=1, only the strict *firsts* are considered). All other content requests are placed in the cache according to jump parameter J2 (normally J2 < J1). Optionally the last T users in the ranking can be treated in a special way, by limiting the caching of the content items they request; this because there may be a fair chance that no other user will request the same content item in the near future.

J, J1, J2 are expressed as a fraction of the cache size. Note that for MLRU J1 = J2, and for LRU J1 = J2 = 100%.

N, M, J1, J2 and T are parameters that are tuned by running a number of Monte Carlo type of simulations. These simulations are done for various cache sizes ranging from 50 to 500 content items.

2.3.2.1.4 Results and discussion

The graph in Figure 39 shows the result of the simulations. It compares the HR for LRU, MLRU and dMLRU for the mentioned cache sizes.

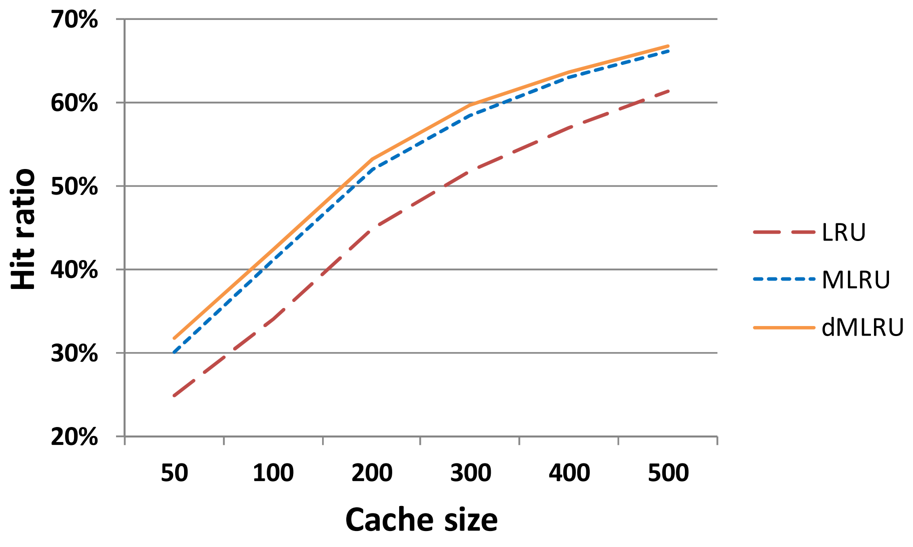


Figure 39. Hit ratio for LRU, MLRU and dMLRU.

Table 19 shows the main parameter settings corresponding to the presented results.

Table 19. Results and main parameter settings.

Cache size	50	100	200	300	400	500
LRU - Hit ratio	24,8%	34,1%	44,9%	51,9%	57,1%	61,4%
MLRU - Hit ratio	30,1%	41,1%	52,0%	58,5%	62,9%	66,2%
MLRU - Jump Size	20%	23%	27%	30%	27%	28%
dMLRU - Hit ratio	31,8%	42,5%	53,2%	59,6%	63,7%	66,9%
dMLRU - Jump Size normal	14%	20%	20%	20%	22%	22%
dMLRU - Jump Size Lead	24%	30%	36%	39%	39%	42%
dMLRU - Nbr of Lead users	515	810	485	735	700	800

The difference between LRU and MLRU is in line with the results presented in [OCE11].

The graph indicates that dMLRU performs only slightly better than MLRU; the difference is too small to be relevant in practical situations. This may be due to the nature of the VOD service: the amount and the popularity distribution of the content items, and the amount of users and their variation in activity level. It may also be due to limitations of the applied metric to determine lead users.

Further investigation is required to assess if this also applies to different types of content (CUTV, OTT (Over-The-TOP) videos, ...)

What can be concluded at this point is that differentiation based on knowledge of user behaviour and characteristics may have some potential to improve caching performance, but that this has to be assessed carefully for each particular case.

In this set of simulations, based on the VOD dataset, the used metric was not based on data obtained from any OSN, but on information derived from the VOD service itself. In the context of eCOUSIN, it is the intention to use alternative metrics based on recommendations and hints from the OSN, which should allow to make better predictions and hence to realize larger improvements.

2.3.3 Pre-fetching

As described earlier a pre-fetching algorithm consists of two parts: 1) an algorithm to predict which user is going to be interested in which content at which time and 2) an algorithm to predict when the network will be in a favourable state to pre-fetch the content items that were predicted by the former algorithm to the user device. Algorithms of the former type are studied in this deliverable. Algorithms of the latter type are studied in deliverable D5.2 [D5.2].

2.3.3.1 Predicting content consumption

2.3.3.1.1 Introduction

The aim of the algorithms studied in this section is to predict at a certain time t that a specific content item k will be requested by a particular user n in the (near) future (>t). It is assumed that users are anonymously identified via a unique integer n (between 1 and N), while content items are identified via a unique integer k (between 1 and K).

Three kinds of pieces of information may be available at time t:

- the consumption of content items by the users in the past (<t),
- the appreciation that users have given for content items in the past (<t) (which may take various forms, e.g., a rating, a number of “stars” or “like”s a user gave to a content item, the number of comments the user provided on a content item, or the fraction of the total duration the user viewed of the content item, ...), and
- the social relations of the users in an online social network.

We organise this available information in matrices, which are all sparse and are time-dependent.

The (NxK)-matrix T contains the time instants in the past (<t) at which the users consumed content items. In particular the element T_{nk} contains the time instant that user n consumed content item k. If user n did not consume content item k (yet) this entry is undefined.

The (NxK)-matrix S contains the rating that user n gave to content item k. We assume that users rate content items that they actually consumed (more or less) coincident with the time of consumption. In particular the element S_{nk} (i.e., the rating that user n gave to content item k) is only defined if T_{nk} is defined.

The (NxN)-matrix C contains the social relations between the users. In particular C_{nm}=1 if user n is a friend (or a follower) of user m and 0 otherwise. By convention C_{nn}=0. This matrix is symmetric only if the relationship expresses a mutual friendship, but is not necessarily so, in case the relationship expresses user n following user m.

Only a subset of these pieces of information may be available to the pre-fetching algorithm. Either complete matrices may be unavailable or a pre-fetching algorithm, e.g., running on a user device, may only have the rows of some of these matrices that pertain to that user.

2.3.3.1.2 Predicting unknown entries of a sparse matrix

As described previously the problem at hand is to predict that a user n will consume a content item k in the near future based on information stored in sparse matrices (known at time t). This boils down to predicting missing entries in these sparse matrices. Below two different methods are described to predict missing entries in matrices and how these methods can be used to predict content consumption.

2.3.3.1.2.1 Predicting consumption based on past consumption within communities

This method starts from the information in matrix T. Since we want to weigh behaviour in the distant past less than in the recent past, we construct the matrix R at time t as follows:

$$R_{nk} = \begin{cases} \exp\left(-\alpha(t - T_{nk})\right) & \text{if } T_{nk} \le t \\ 0 & \text{otherwise} \end{cases}$$

where α is a parameter that determines the history the prediction system will take into account. The parameters α should be chosen inversely proportional to the typical life time of a content item. We define the (NxK)-matrix A as the matrix that indicates which users consumed which content items, i.e., $A_{nk}=1$ if user n consumed content item k prior to t and 0 otherwise, i.e., $A=1_{\{T<t\}}$, where $1_{\{\}}$ is the (element-wise) indicator function. Notice that if α tends to 0, R tends to A.

The (1xK)-matrix

$$e_N{}^t \cdot R$$

where e_N is the (Nx1)-matrix with all entries equal to 1 (and ^{t} denotes the matrix transposition), contains the popularities measured at time t over an exponentially decaying window in the past. This momentary global popularity information would be used by a traditional global caching system to decide which content items to cache. When α tends to infinity, this caching system is equivalent with LRU, while when α tends to 0 this caching system is equivalent to LFU (Least Frequently Used) with a very large window. With this is in mind the element R_{nk} of the matrix R can be interpreted as the contribution of user n to the momentary rate with which content item k is consumed. The problem then boils down to making a prediction for the elements of R for which $A_{nk}=0$, based on the matrices T, S and C. If such a prediction for which $A_{nk}=0$ exceeds a threshold θ, which is a parameter in the system, the algorithm predicts that the user n will consume content item k in the near future after t. There are various ways to predict the elements of R for which $A_{nk}=0$.

The first prediction is based on global popularity defined above. In particular the prediction is

$$\left(\frac{e_N \cdot e_N{}^t}{N} \cdot R\right)_{nk} \text{ if } A_{nk} = 0$$

This prediction does not take into account that a user may be correlated more to one than to another user. In order to take this into account we define a (NxN)-matrix W where W_{nm} is the weight with which user m influences user n, where by definition $W_{nn}=0$ and $W \cdot e_N = e_N$, the latter of which just says that the rows of W should sum to 1. So, the second prediction is

$$(W \cdot R)_{nk} \text{ if } A_{nk} = 0$$

There are many possible choices for W (see also Section 2.3.4.1), but two obvious choices are C (if available) and R·R^t, for which, in both cases, the rows need to be rescaled so as the entries of each row sums to 1.

2.3.3.1.2.2 Predicting consumption via predicting user appreciation

This approach starts from the matrix S with user ratings. It is inspired by algorithms in recommendation engines. It is based on the idea of approximating the missing elements of the matrix S via the product P·Q^t, where P is a (NxL)-matrix, Q is a (KxL)-matrix and L is referred to as the number of features. The underlying idea is as follows. Each content item k is characterized by L features, which are stored in the k-th row of the matrix Q. Element Q_{kl} expresses how much of feature l content item k exposes. Likewise each user n is characterized by a set of L weights, which are stored in the n-th row of the matrix P. Element P_{nl} expresses how strong user n responds to the presence of feature l. The response of user n to content item k is the dot product of these two vectors, which is in fact the (n,k)-th element of the matrix P·Q^t.

The values of the matrix elements of P and Q are determined by demanding that elements of the product P·Q^t for which the corresponding element of S is known (i.e., the elements for which A_{nk}=1) approximate the value of that element of S as close as possible. Typically it is also demanded that the elements of P and Q should be as small as possible. Therefore, to determine the elements of the matrix P and Q the following function is minimised:

$$\frac{\sum_{n,k}\left[\left(P\cdot Q^t - S\right)_{nk}\right]^2 A_{nk}}{2\phi NK} + \frac{\lambda}{2N}\sum_{n,l}\left[P_{nl}\right]^2 + \frac{\lambda}{2K}\sum_{k,l}\left[Q_{kl}\right]^2$$

where λ is referred to as the regularisation parameter and ϕ is the sparsity of matrix A

$$\phi = \frac{\sum_{n,k} A_{nk}}{NK}$$

It turns out that only matrices P and Q of the following form need to be considered

$$P = U\cdot\Lambda_P$$

$$Q = V\cdot\Lambda_Q$$

where U and V are obtained via the SVD (Singular Value Decomposition) of S

$$S = U\cdot\Lambda_R\cdot V^t$$

and all matrices Λ are diagonal matrices. It is easy to prove that for this choice of P and Q the function to minimise becomes

$$\frac{\sum_{l,l'}\alpha_{l,l'}\left(\lambda_{P,l}\lambda_{Q,l} - \lambda_{R,l}\right)\left(\lambda_{P,l'}\lambda_{Q,l'} - \lambda_{R,l'}\right)}{2NK} + \frac{\lambda}{2N}\sum_{l}\left(\lambda_{P,l}\right)^2 + \frac{\lambda}{2K}\sum_{k,l}\left(\lambda_{Q,l}\right)^2$$

with

$$\alpha_{ll'} = \frac{1}{\phi}\sum_{n,k} U_{nl}U_{nl'}A_{nk}V_{kl}V_{kl'}$$

Note that because U^t·U=V^t·V=I (where I is the identity matrix), α=I if A is fully populated.

We first determine the stationary points (i.e., the points where the gradient with respect to $\lambda_{P,l}$ and $\lambda_{Q,l}$ is 0) of the function to minimise and then eliminate the local maxima from this set of points. After some tedious calculations, it is found that, either

$$\frac{\lambda_{P,l}}{\lambda_{Q,l}} = \sqrt{\frac{N}{K}}$$

$$\lambda_{P,l}\lambda_{Q,l} = \lambda_{R,l} - \sqrt{NK}\,\lambda(\alpha^{-1}e_L)_l$$

where $^{-1}$ denotes the matrix inverse, if the right hand side of the latter equation is positive, or

$$\lambda_{P,l} = \lambda_{Q,l} = 0$$

otherwise.

A content item k is predicted to be consumed by user n if $(P \cdot Q^t)_{nk}$ exceeds a threshold θ, which is a parameter of the system.

2.3.3.1.3 Performance metrics

The consumption predictions of the algorithm running at time t (i.e., whether or not user n will consume content item k at a time >t) is successful if that user actually consumes that content item in the future (at time >t) and is wasteful if that user never consumes that content item. To assess the performance of the algorithm we introduce the following metrics:

- The number of TP (Timely Predictions) = the number of content items that are predicted to be consumed *and* that are actually requested by the user
- The number of UP (Useless Predictions): the number of content items that were predicted to be consumed *and* that are *not* requested by the user
- The CPR (introduced earlier) = the number of content items that were predicted to be consumed *and* are actually requested by the user divided by the total number of predicted content items = TP/(TP+UP)
- The HR (introduced earlier) = the number of content items that were predicted to be consumed *and* are actually requested by the user divided by the number of requests of the users = TP/#requests

The algorithm should try to maximize TP, or equivalently HR, while keeping UP small, or equivalently CPR large.

As an intermediate step we also would like to assess the performance of a recommendation engine. Such a recommendation engine aims at predicting the rating that a user will give to content items that they did not consume yet. Typically their performance is evaluated by considering the RMSE (root mean square error) between the actual value (when the user consumes the content item) and the prediction.

2.3.3.1.4 Preliminary performance study

To assess the performance of these two content consumption prediction methods we use subsets of

- the CUTV data set captured by FT for which only the matrix T is available, and
- the Flixter data set [FLI] for which the matrix T, S and C are available.

The statistical properties of these data sets are described in Section 2.1.

First we consider the CUTV data set. We only consider the (N=3212) most active users and the (K=5069) most popular content items. Each user viewed at least 20 content items and each content item was viewed at least 20 times. We run the algorithm at a time t half-way (at day 16) in the trace, at which time instant there is enough information to train the algorithm and still has enough information in future to evaluate the algorithm. From Figure 4 it can be seen that the average life time of content items is not long. Therefore, to calculate the matrix R we set the value of α^{-1}=10 [days].

Figure 40 shows the trade-off between timely and useless predictions for three prediction algorithms discussed in Section 2.3.3.1.2. For the community-based prediction (labelled with “CB on R”) the weights are determined as $R \cdot R^t$ where the rows are scaled so that the sum of each row is 1. For the method (labelled with “SVD on R”) based on decomposition of the matrix R in a product $P \cdot Q^t$, the parameters were set to L=3 and λ=0.1 (with which the algorithm performed best). The third method (labelled “LFU”) just used the global popularity (up to time t). This figure shows that the prediction based on calculating the popularity of content items within communities based on past content consumption has the best performance.

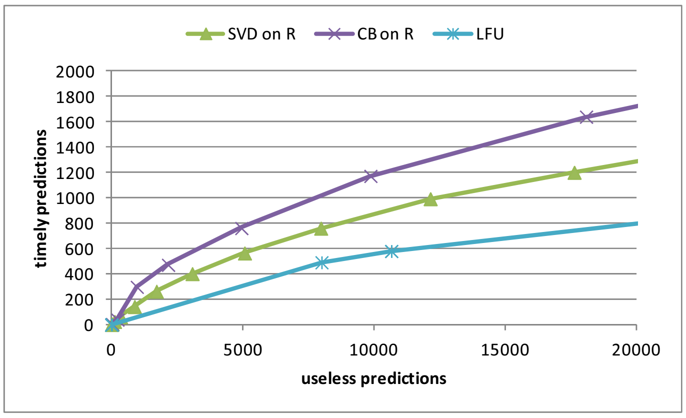


Figure 40. Timely predictions as function of useless prediction for the CUTV data set, LFU (black), SVD (red) and communities (green).

Next we consider the Flixter data set. We first use the algorithm to decompose the matrix S obtained at the end of the trace (no matter what timestamp they have) with ratings as a product $P \cdot Q^t$. We only consider the ratings of the (N=3987) most active users for the most (K=2545) popular movies. A random fraction of 10% of the ratings are used as test set, while the 90% others are used as training set with which the matrices P and Q are determined. Figure 41 shows the RMSE of the predictions for the ratings in the test set for various values of the parameters L and λ. It can be seen that L=32 and λ=0 are the best choices. The RMSE when the ratings in the test set are predicted as a weighted sum of ratings where the weights are determined via a suitably scaled version of $S \cdot S^t$ is 1.038, where we only average over ratings that users actually gave. The conclusion is that in order to calculate an accurate prediction of a rating the method based on matrix decomposition performs better than the method relying on averaging with communities.

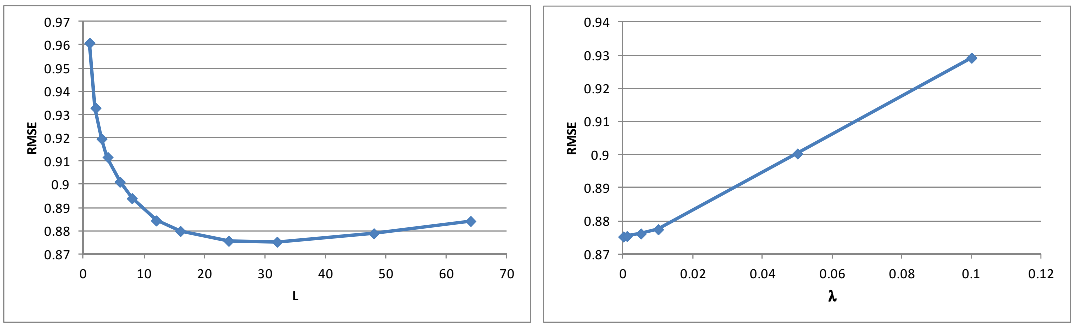


Figure 41. Error on the prediction of the ratings in the test set as a function of L (left) and λ (right).

Next we consider all ratings given until a fraction day 1365 (which is 90% of the duration of the trace) as training set and all ratings given after that day as test set. Figure 42 shows the RMSE of the predictions for the rating in this test set for various values of the parameters L and λ. It can be seen that the performance is much worse than Figure 41 and that L=32 and λ=0.01 are the best choices. The fact that the performance is much worse stems from the choice of the test set. In the choice of the test set based on 90% of the total duration there are 28 users in the training set that did not rate the movie and 7 movies in the training set did not receive any rating, while in the random choice of the training set there were no user that gave no rating and no content items that received no rating. For those users and movies that had no ratings in the training set the algorithm has no basis to determine the ratings in the test set (and just chooses the average rating). This decreases the performance considerably. The RMSE when the ratings in the test set are predicted as a weighted sum of ratings where the weights are determined via a suitably scaled version of $S \cdot S^t$ is 1.099 and still worse than for the algorithm based on matrix decomposition.

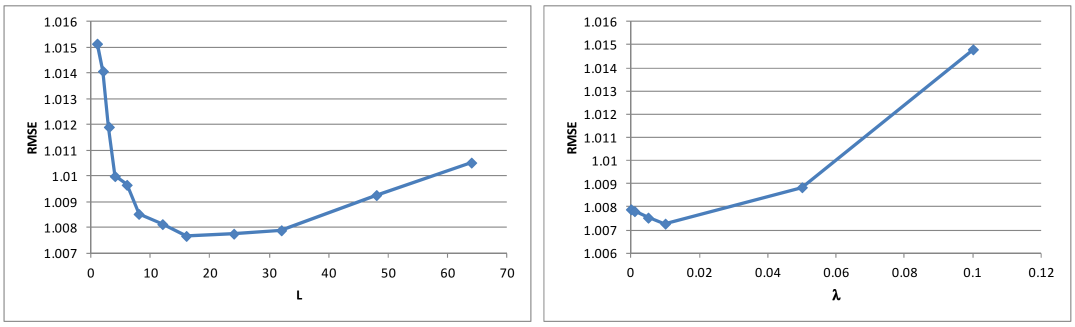


Figure 42. Error on the prediction of the ratings in the test set as a function of L (left) and λ (right).

Finally we use the prediction of unknown elements of S_{nk} as an indication that a user n is going to consume content item k. Figure 7 suggests that if a user gives or is predicted to give a higher rating, he or she is more likely to consume that content item. In particular we indicate that user n will consume content item k if that predicted value is larger than some threshold θ for various values of that parameter. For this case we evaluate the performance half way through the trace (for similar reasons as in the CUTV case). At that time there are 806 users that did not rate any content yet and 201 content items were not rated yet.

Figure 43 shows the trade-off between timely predictions and useless predictions as this threshold changes for various methods.

The reason that LFU works so well is that for the 806 users for which no information is available at the time of evaluation, LFU uses the global popularity. This is better than all other methods that make no prediction at all. This effect will only play when new users arrive in the system. When the number of users is more or less stationary (like in the CUTV case) this effect does not play. Apart from that, it can be seen that making predictions based on past content consumption (labelled with "... on R") is better than on ratings the users gave (labelled with "... on S"). In both sub-cases the method in which an average within a community is taken (labelled with "CB on ...") outperforms the method based on a matrix decomposition (labelled with "SVD on ...").

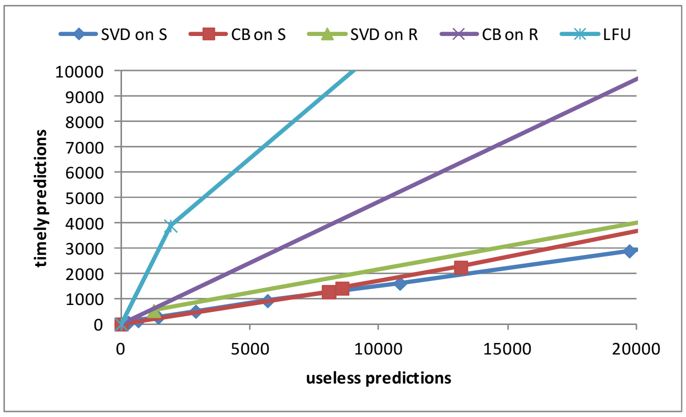


Figure 43. Timely predictions as function of useless prediction for the Flixter data set.

The conclusions from these preliminary investigations are as follows.

- For predicting how users will appreciate content (that they did not rate yet) the method based on matrix decomposition performs better than an averaging within a community. However, the rating users (will) give to content is not an ideal indication of whether they are going to consume it or not.
- For predicting which content items users will consume in the future, it is best to start from the matrix that collects past consumption (suitable weighted over an exponentially decaying window over the past). The method that averages over a community yields a better performance than the method based on matrix decomposition.
- There is one exception: in the case new users arrive in the system, it is beneficial to use global popularity for these users, but as soon as the user community of these users is stable enough, it is best to use an average within this community.

2.3.3.1.5 Maximum performance based on Mobile traces - Preliminary results

The objective of the assessment works reported in this section is to get insights on the optimal performances the pre-fetching app integrated in eCOUSIN's use case "Social-Assisted Time-Unconstrained Content Delivery" could achieve with the real-life mobile dataset presented in Section 2.1.4.

The performances are evaluated using the CPR and HR.

The performances are said to be optimal as we assume that the social graph is an optimal connected graph, i.e., we trained the social graph all along the trace before running the prediction.

We ran the simulations using the pre-fetching simulator introduced in Section 2.2.3 and configured with the following parameters. The running mode is the interest mode, with parameter expA equals to 30 and expB equals to 0.00011. This value for expB assumes that non-heavy users will pre-fetch a smaller number of videos than heavy users.

For the first set of simulations, which are reported in Figure 44, the parameter "*ThresholdNotification*" is set to 1. This means that every user requests all contents that have been viewed by at least one of her direct neighbors in the social graph. Here we compare the performances of LRU and EdgeRank as *NotifManager* algorithm. As described in Section 2.2.3 the NotifManager algorithm is used by the user's device to sort items that will be pre-fetched in a particular order. This comparison is achieved for different sizes of cache on the user terminal: 10, 50 and 100 videos. Figure 44 shows that EdgeRank clearly outperforms LRU both in terms of CPR and HR.

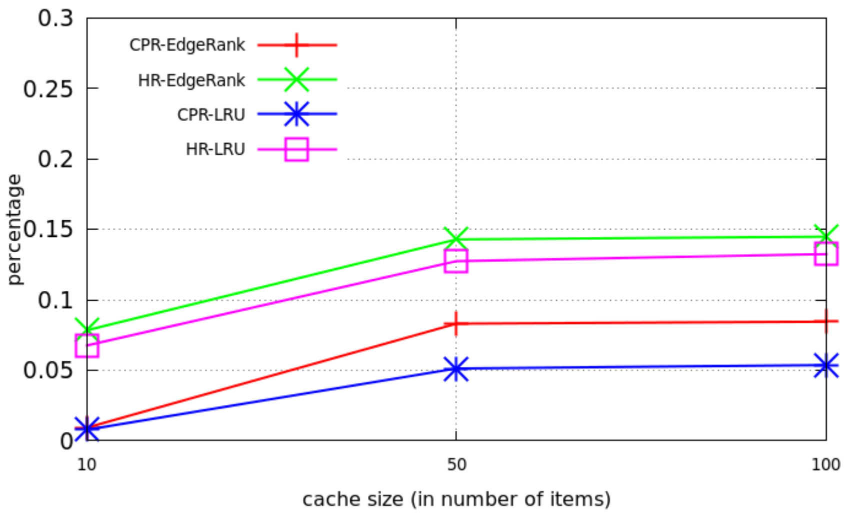


Figure 44. HR and CPR as a function of the cache size for LRU and EdgeRank, assuming Thresholdnotif=1.

Since EdgeRank performs better than LRU, we set it as reference NotifManager algorithm, then we fine-tune other parameters to evaluate their impacts on CPR and HR.

In Figure 45, we set the parameter ThresholdNotif to 2, and we evaluate CPR and HR as a function of the size of the cache. We observe a significant improvement of CPR and HR when changing the parameter ThresholdNotif from 1 to 2. This is mainly due to the long tail of the video popularity distribution: most of the videos are watched only once by a given single user.

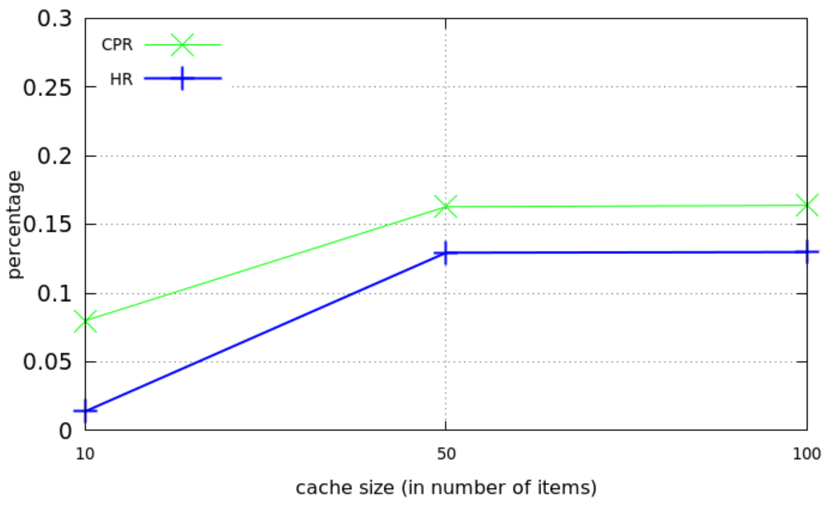


Figure 45. HR and CPR as a function of the cache size, assuming NotifManager=EdgeRank and Thresholdnotif=2.

Finally, in Figure 53, we set the cache size to 100 and changed iteratively the value of thresholdNotif. We observe that the value of CPR increases while the value of HR decreases. We also note that both CPR and HR evolutions are not proportional and give more weight to CPR.

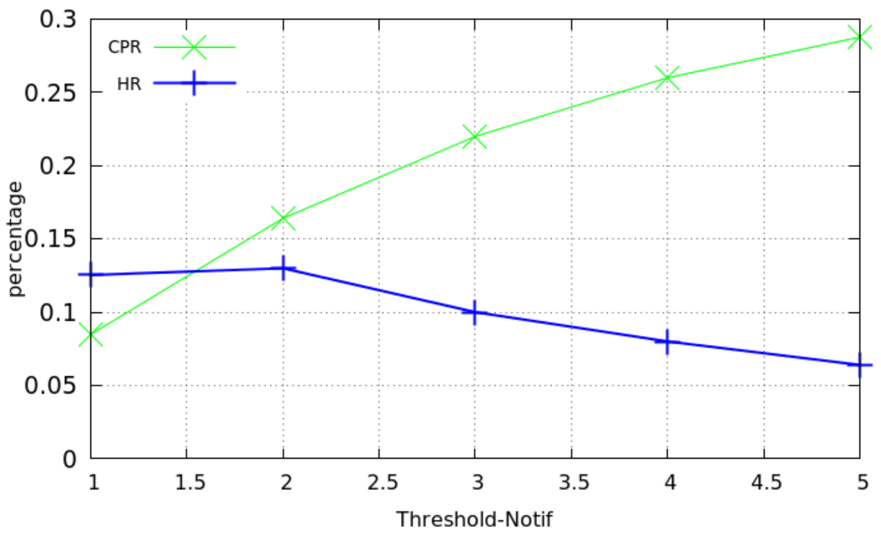


Figure 46. HR and CPR as a function of Thresholdnotif, assuming NotifManager=EdgeRank and a cache size=100.

2.3.3.1.6 Initial results with the mobile tracing app

As introduced in Section 2.2.1.2, a prototype of a mobile application has been developed to investigate the potential of user-centric pre-fetching and to design social-enhanced pre-fetching mechanisms. The following section describes the first results towards this goal. We refer to the first version of the pre-fetching app as "*Mobile Tracing App*" as the focus of this initial Android application was on the social data collection and aggregation and, thus, builds the basis for the social data prediction. To test the collection of social data and identify relevant pieces of information that could be used for the prediction of content consumption, a first user study was conducted [THR14]. This study focused on analysing which social properties are appropriate for predicting video content that a user is likely to consume in the near future. In the following, some general information about the user group is given. Subsequently, the gained insights from social properties to predict video consumption in OSNs are explained in detail.

General information on the study supported by the Mobile Tracing App

The Mobile Tracing App was offered to the users over the official Google Play Store[5]. According to the German data privacy act each user has to sign a declaration of consent, so we are allowed to collect and use the traces collected from the users. Therefore the app is protected with an activation key. This activation key has been only retrieved from those users that have signed the declaration of consent. Without this key, it is not possible to use the app. In addition, all personal data within the traces is anonymized. All participating users were informed what was the goal of the study. Furthermore we encouraged the participants to

- Not act in a study friendly way
- Use the app like they would normally use the Facebook wall to explore their friends activities
- Not change their user behaviour

During the study, 18 people used the Mobile Tracing App, but only 11 users provided useful datasets for the study. The reasons behind none useful datasets were corrupted databases caused by data transmission failures or non UTF-8 conform text elements. These issues will be tackled in the next version of the Mobile Tracing App. The 11 datasets useful for the study were retrieved from four

[5] https://play.google.com/store [Accessed March 31th, 2014]

different mobile device models. Nine participants used a single device and a single Facebook account each. Two participants used two different devices to connect to their Facebook account. One participant was female, the rest were male. The participant's age ranged from 22 to 38. The average age was 28 and the median 25. Only one participant did not specify his age and we assumed the average age for this participant.

Due to the small number of datasets, the obtained results cannot be generalized and it is expected that findings will differ in case of using a larger user base. Yet, the study is valuable to discuss first interesting insights that helped us to redefine of the design of the final mobile pre-fetching app and as guidance for the development process.

General statistics on the data collected by the study

In the following some key numbers for the collected datasets are stated. In total 1615 wall posts that were displayed to users were traced during the study, consisting of:

- 374 links
- 649 photos
- 133 videos

Besides simply accessing content items, the users of the Mobile Tracing App were able to mark video posts as interesting, in case they would have liked this item to be pre-fetched but do not want to watch right away (e.g., due to the limitations of their mobile data plan). Watched and marked items were both counted as being watched to get a better understanding on which items ideally would have been selected by the social predictor component and, as a next step, would have been a pre-fetched automatically. In sum, the participants watched 134 wall posts. This corresponds to 9.29% of all wall posts. The distribution among the different content types is the following:

- 43 videos
- 74 pictures
- 17 links

In summary, based on the results, the average study participant can be described as follows:

- Number of friends: 196, with a spread from 11 to 554, the median is 130.5
- Group memberships: 8, with minimum 0 and maximum 36
- The feed contains 91 posts, minimum 17, maximum 239

General statistics on social properties

Next we analyse comment and like distribution for wall posts. The exact average number of comments and likes could not be determined for all of the posts, because only a maximum of 1,000 of each was loaded when retrieving the meta data of the posts in the app. Setting this restriction was necessary to reduce the time and traffic caused when retrieving this data using the Facebook API. Yet, this restriction does not impact our findings on the amount of likes and comments below the threshold. Overall, 98% of all likes and 94% of the comments have a count of less than 1000 (see Figure 47), thus the restriction impacts only a small fraction of the traced posts.

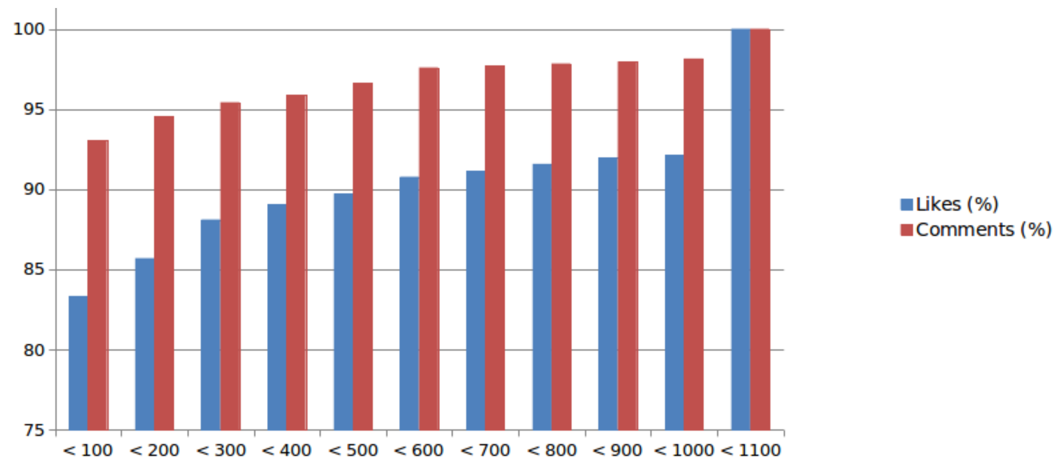


Figure 47: Cumulative distribution of posts in buckets of likes and comments ([THR14]).

For future studies, the limitation was addressed in a new version of the application so that traces will also include statistics about items with more likes and comments. With respect to this limitation, the current datasets include

- 188317 likes and
- 68522 comments.

The distribution of the amount of likes and comments for posts with less than 100 interactions is shown in Figure 48. Here, it becomes apparent that most posts have less than 3 likes or comments, which could be a result of most posts being relevant only within small social groups and belonging to the so called "long tail" of videos in the popularity distribution. This in turn supports the assumption that the prediction of content consumption could greatly benefit from a consideration of social properties of the individual users. The count of interactions and in particular the number of likes of a post could be used as a good indicator of the popularity of a post if no additional information, e.g., by the content provider is available.

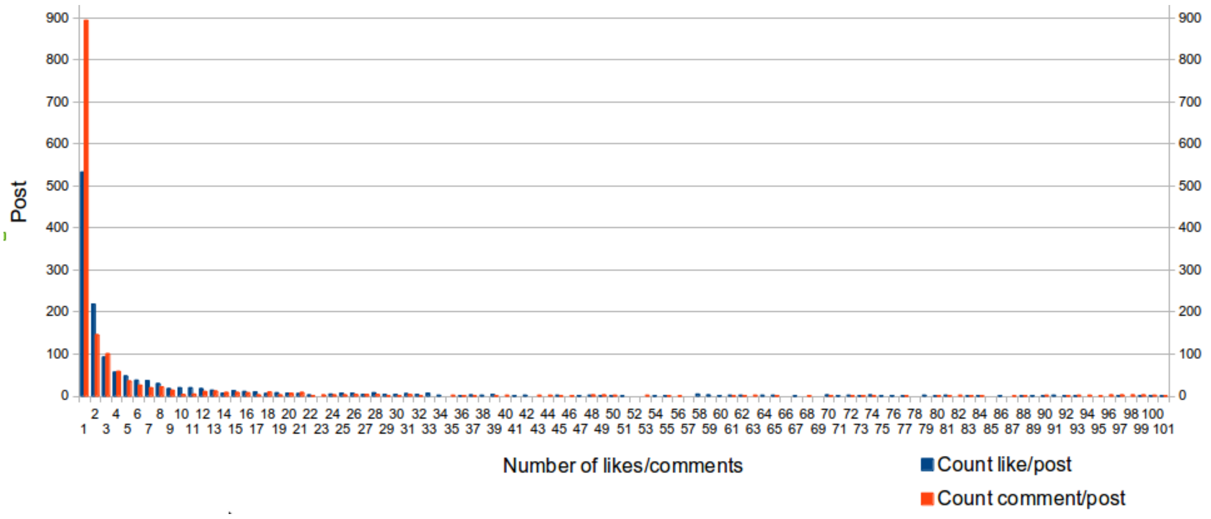


Figure 48: Distribution of posts across like and comment count ≤ 100 ([THR14]).

We also tracked shares of the posts, which spread from 0 to 132,114. Here, it is remarkable that the median for the shares is 0, which indicates that a large number of posts were not shared at all. Based on this observation, the number of shares might only be interesting for very popular posts. This will be further investigated in future studies.

Correlation between social properties

Each post has comments and likes as two important social properties. To further study their characteristics, their correlation was investigated to better understand if they influence each other. In a first analysis, we observed a correlation of 0.74 with a significance of 0.00 for our datasets by using the Pearson-Moment Correlation Coefficient. This leads us to the assumption that the comments and likes appear together frequently. Figure 49 shows a plot of the data.

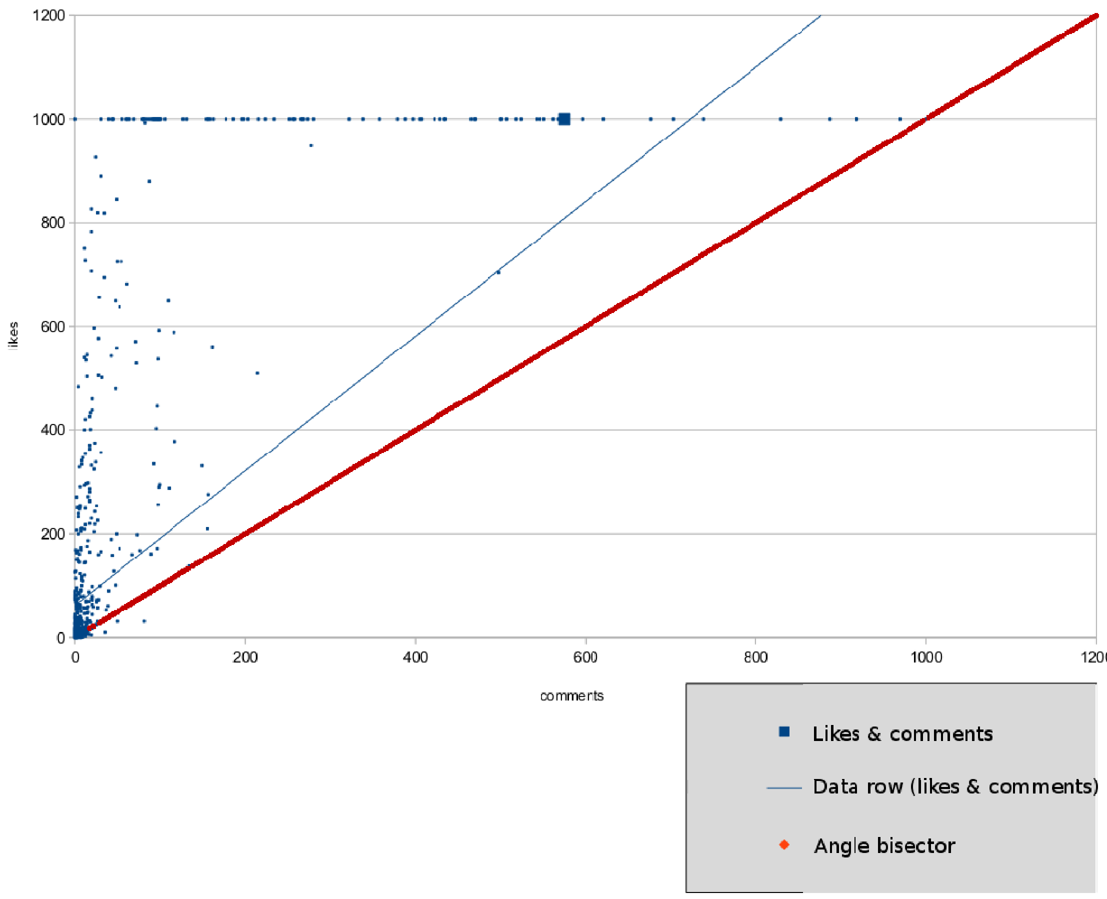


Figure 49: Correlation of comments and likes ([THR14]).

In the study we also investigated the correlation between the comments and shares of posts as well as likes and shares. To calculate these correlations, some outliers had to be filtered. For the reason that the used like and comment count has a maximum of 1000 and the share count in the dataset is not limited, the 57 posts with a share count higher than 1000 were filtered. This was done only for the correlation analysis. The results of the correlation analysis are summarized in Table 20. It is remarkable that fewer videos were consumed based on our datasets than we assumed.

Table 20: Correlation Share with likes respectively comments ([THR14]).

Values	***Pearson Correlation***	***Significance***
Comments & Shares	*0.44*	*0.00*
Likes & Shares	*0.53*	*0.00*

In contrast to the correlation of the watched posts, the correlation of the not watched posts is much smaller. The correlation results for likes, comments, and shares for not watched posts are stated in Table 21. The same correlation results for the watched posts only are stated in Table 22.

Table 21: Correlation between likes, shares, and comments of the not watched posts ([THR14]).

Pearson Correlation/ Significance	Likes	Comments	Shares
Likes	1.00	0.73/0.00	0.08/0.00
Comments	0.73/0.00	1.00	0.11/0.00
Shares	0.08/0.00	0.11/0.00	1.00

Table 22: Correlation between likes, shares, and comments of the watched posts ([THR14]).

Pearson Correlation/ Significance	Likes	Comments	Shares
Likes	1.00	0.83/0.00	0.66/0.00
Comments	0.83/0.00	1.00	0.84/0.00
Shares	0.66/0.00	0.84/0.00	1.00

Interactions by direct friends

To understand if the interaction of direct friends gives an indication on the relevance of posts to users, we investigate this property in the following. Here, we also take into consideration if posts were actually watched by the user.

In the collected datasets, 273 posts were liked by friends and 0 posts were commented by friends. Table 23 details how these posts spread over the amount of interactions (likes or comments): 225 out of the 273 commented posts, for example, were liked only by one friend each, 27 by two friends, etc. In the last row, the spread among the amounts of likes is shown for videos and photos only as these could be used as potential candidates to be pre-fetched.

Table 23: Total number of friend interactions spread across number of interactions ([THR14]).

#Interactions / Type of Interaction	1	2	3	4	6
Posts liked by "count" friends	225	27	15	4	2
Posts commented by "count" friends	0	0	0	0	0
Videos, photos, and links liked by "count" friends	100	20	9	3	2

We note videos, links, and photos as watchable. In Table 24, it is shown that 25 of the 135 watched posts were liked by friends and how these two properties compare to each other. This number corresponds to 16.9% of all posts and 11.87% of the videos, links, and photos.

In 18.65% interactions with the posts by the user's friends were observed. It is possible that more of the posts are liked or commented by friends. As explained above, the reason is that only 1000 likes or comments are collected per post. Therefore, a user's friend could have interacted with the post but not be amongst the 1000 collected comments or likes. Overall, we only observed 14 out of the 134 watched posts with more than 1000 likes or comments to which the above limitation could apply.

Table 24: Comparison of likes by friend and watched items ([THR14]).

#Likes by friends	1	2	3	4	5	6
Watched	20	3	0	1	0	1

User content consumption

Due to the small number of user traces and the apparent differences in accessing content, we decided to investigate the potential to identify different groups of users with more similar content consumption. Table 25 shows the individual access to content, taking into consideration the properties of the content.

Table 25: Summary of content properties.

Property/User	A	B	C	D	E	F	G	H	J	K	L
Posts	149	147	25	28	25	396	237	53	423	108	24
Liked posts	99	77	18	21	20	243	167	42	162	63	16
Commented posts	59	52	10	19	14	155	90	31	95	47	17
Shared posts	38	28	3	6	5	78	29	7	53	11	9
Post type=video	11	12	1	12	1	27	19	5	20	16	5
Post type= picture	55	70	12	14	11	188	74	16	113	33	7
Post type=link	29	33	6	4	8	83	51	13	45	22	7
Likes > 200	23	21	3	4	3	58	15	3	35	5	4
Likes > 500	18	15	1	2	0	49	11	1	30	2	1
Likes > 800	16	12	0	1	1	43	9	1	27	1	0
Likes >= 1000	16	12	0	1	1	39	8	1	24	1	0
Comments > 200	13	2	0	1	1	36	4	0	23	0	0
Comments > 500	8	0	0	0	1	28	1	0	16	0	0
Comments > 800	0	0	0	0	1	24	0	0	15	0	0
Comments >= 1000	0	0	0	0	0	21	0	0	13	0	0
Shares > 200	14	11	0	2	0	34	8	2	21	2	1
Shares > 500	9	5	0	2	0	26	6	1	16	1	1
Shares > 800	9	3	0	1	0	23	5	1	14	0	0
Shares >= 1000	9	2	0	1	0	19	4	1	11	0	0

Between 52.38% and 80.70% of the posts consumed by each user are liked. Between 35.37% and 70.83% of them are commented on and between 10.19% and 37.50% of the consumed posts were shared. Furthermore, there seems to be a dependency between likes, comments, and shares of posts. Intuitively this could be explained by the different level of intensity that the three interactions have. The burden to like a post could be seen as quite low, in comparison to commenting or even re-

sharing a post. Re-sharing might be the most intense interaction as it requires an explicit position to or even identification for a particular content item.

By comparing the different types of posts (video, picture, and link), it can be seen that the most common type of the three are pictures, independent from the user. Second most common are links, except for user C. In most cases, videos have a share of less than 10% except for the users C, K, L. In the following rows, the posts are ordered according to their amount of shares, likes, and comments. For each of these three properties, four steps are differentiated to show the spread of likes, comments, and shares for the posts consumed by each user.

Unfortunately, it was not possible to determinate the posts which are posted in groups due to a problem in the collection software as the collected group IDs and the poster IDs of posts posted in groups do not match. The assumption is made that a large number of posts with a high count of likes, comments and shares are likely to be posted in groups.

In Table 26, the number of watched posts per user and the properties of the watched posts are shown. The last row details how many posts are watched as a percentage of total posts. The second row shows that most users watched posts that have already been liked. The amount of liked posts each user watched ranged from 35.71% (user K) to 100% (users H, L, E) of total watched posts of that respective user. Row 3 lists how many watched posts are commented. Between 40% (user B) and 100% (user L) of watched posts are commented. Only in the case of user B the amount of shares seems to be relevant. Similar to the previous part, the likes, comments, and shares are less than 200. Only for one user (user F) more than 20% of watched posts are liked, shared, and commented on by more than 999 users. By inspecting the watched posts of user F in detail all 10 watched posts with more than 999 likes had more than 400 comments and 900 shares. 8 of them had more than 999 likes comments and shares. User F seems to be a consumer of globally relevant content.

Table 26: Summary of properties for watched content.

Property/User	A	B	C	D	E	F	G	H	J	K	L
Overall watched posts	8	10	6	2	2	36	17	5	22	12	1
Watched & liked	5	8	6	1	2	32	8	5	20	10	1
Watched & commented	4	4	5	1	1	17	10	3	11	6	1
Watched & shared	0	5	1	0	0	11	4	0	7	1	0
Watched & liked by friends	0	0	2	1	0	5	4	2	2	2	0
Watched videos	4	3	1	1	0	10	8	1	7	4	1
Watched photos	4	5	3	1	1	23	8	2	15	7	0
Watched Links	0	2	2	0	1	3	1	2	0	1	0
Overall posts watched [%]	5,4	6,8	24	7,1	8	9,1	7,2	9,4	9,1	11,1	4,2
Watchable watched [%]	8,4	8,7	31,6	10	11,1	12,1	11,8	14,7	12,2	16,9	5,3

Studying the relative number of watched posts in Table 26 (last row) and the consumed posts (first row), the users could be split in two groups: high and low consumers. User C, D, E, and L could be considered as part of the low consumers since they consumed less than 30 posts in total. Due to the fact that the mobile tracing app retrieves about 25 posts from the Facebook news feed by default, these users seem to have used the app at most two times. Even if the percentage of consumed content for these users is high, overall, they only show a very small number of consumed posts and are considered low consumers. Their low number of consumed content, thereby, could be the result of a rare use of the app or of a rare usage of Facebook in general. Based on the number of consumed content users F, G, and J are high consumers. All of them consumed more than 200 posts in total. Users A, B, and K consumed more than 100 posts in total and watched between 5 and 11 posts. They are classified as high consumers even if users A and B are at the bottom end of the high consumer bracket, because of their low watched percentage. User H, which consumed 52 posts and watched about 10% of the posts, also belongs to the high consumer bracket. An overview of the qualitative classification can be seen in Table 27.

Table 27: Low and high consumers ([THR14])

Low consumer	**C, D, E, L**
High consumer	F, G, J, A, B, K, H

High consumers

As explained, high consumers are Facebook users who consumed a huge amount of posts compared to the other users as observed in the datasets. These high consumers can be further separated in those who consumed more than 200 posts and those who consumed more. It seems that high consumers who consumed more than 200 messages tend to consume more posts which are globally relevant according to likes, comments, and shares. Further insights that we gained from the datasets with respect to the high consumers are:

- 60% of all watched posts are liked
- 16 out of 124 posts watched are liked by more than 500 people
- 14 out of 124 posts watched are liked by more than 999 people
- Only a few posters posted highly shared, liked, or commented posts
- 22 out of 124 (17.74%) watched posts are liked by the user friends
- 40 out of 124 (17.74%) consumed video posts were watched

Low consumers

Characteristic for the low consumers is that they consume only a few posts. In our datasets each low consumer consumed less than 30 posts in total. On average this group watched 10% of all posts. The low consumers watched no posts with a comment count higher than 27 or a share count higher than 85. None of the low consumers consumed posts with a like count higher than 71 with exception to user C who consumed one post liked by more than 500 users. From our datasets, two of the four low consumers watched comparable many posts liked by their friends. User C watched 2 out of 7 posts liked by his friends and user D watched 1 of 2. Three out of the low consumers watched one video, the fourth user watched none. One of these three watched more than ten videos. But only three out of these 19 videos were actually watched.

Social indicators appropriate for the Social Prediction Module

Certainly, one has to be careful when trying to generalize the results presented above to other user groups or settings. A larger number of users would be required in this case. Yet, the collected datasets give a first insight and help us to define the focus of future studies.

Taking into consideration the pre-fetching use case, where videos are downloaded to the user device when it is connected to a Wi-Fi network, more data can be downloaded than later accessed. The reason for this is that WiFi is more energy-efficient. Depending on the device and other parameters, up to 10 times ([GKSR13]) more data can be pre-fetched than accessed. Following this assumption and considering the currently available data, one could argue that it would be an appropriate strategy to simply pre-fetch all new videos that appear on a user's news feed. A key reason for this finding is that the users in our study only accessed a small number of videos. This, for sure, might change with other users and longer study periods.

Our data suggests that the interaction of friends with the posts do not have a high impact on the decision of a user to watch a video or not. Yet, this finding might be also biased due to the small dataset and the specific users. Regarding the high consumers, we saw that every third video is watched. If every video would be pre-fetched, it would still be beneficial comparing the data transfer over Wi-Fi, instead of the mobile network with respect to energy efficiency and QoE of the user.

Nevertheless, the order in which the videos are pre-fetched might be important since WiFi is not always available, resulting in only a part of the video being downloaded. Therefore, the videos with higher probability of being consumed should be downloaded first. The following insights could be gained from the study:

- The videos with the highest global relevance should be pre-fetched first, therefore likes are a first good indication.
- The very high consumers consumed mainly content out of a few content sources. A machine-learning approach could be used to learn which sources/publishers are especially interesting for individual users and give them a high pre-fetching priority.
- Content posted in groups has often a high global relevance. Therefore also content published in groups should get a high pre-fetching priority.

Also for low consumers, pre-fetching turned out to be beneficial since it helps to offload traffic from the often volume-limited mobile data plans of the users.

2.3.3.2 Impact on the network

2.3.3.2.1 Network resource allocation performance

After we implemented the rate allocation algorithm we set up simulations in order to evaluate its applicability in real scenarios. We used a modified version of ns-3.18 simulator[6] and more specifically the LTE module of NS-3 called LENA[7][8].

2.3.3.2.2 Simulation setup

We placed it at a video server in the Internet that transmits IP packets to a LTE UE Video client. The video server transmits data based on the results of the optimization algorithm. We set the capacity of all the links between the server and the eNodeBs to very high values (practically infinite) and their transmission delay to 0ms. This had the effect that the delay between the transmission of IP packets

[6] http://www.nsnam.org/

[7] http://lena.cttc.es/manual/

[8] The main difference between our modified version and the default version of the simulator is that the proportionally fair scheduler located at the eNodeBs takes into account whether the UEs that are competing for a RBG have any more data to transmit. In case they do not have any more data destined for them at the related buffers of the eNodeB, they are automatically excluded by the competition for the remaining RBGs of that scheduling period contrary to the default behavior of the simulator which would continue to allocate resources to UE that have nothing more to receive, thus wasting them.

from the server and their appearance at the eNodeB to be 1ms, which we assume is equivalent of transmitting the data directly a nearby cache. Further, in all simulations we used UDP traffic, because we wanted to test the feasibility of the delivery of the IP packets based on the different allocated data rates, free from the often unexpected behaviour of TCP (Transport Control Protocol).

Our workflow was as follows.

We set up a scenario in the NS-3 simulator. The scenario includes various static and moving UEs, some of them performing handovers to neighbouring eNodeBs. Then we run the scenario setting all the participating UEs to receive saturation traffic. That way we can infer the worst case maximum capacity for any given time for all the UEs. It is important to note that in order to have consistent results with the rest of the simulations of the work flow, we set the size of the transmitted IP packet in this saturation scenario equal to the size of the IP packet transmitted by the video application. We set the size of the IP packet to be 1125 bytes.

We collect the traces of the TBs transmitted from the eNodeBs to the UEs, as well as the IP packets delivered to the buffer of the video application of the UEs.

Based on the above traces we calculate the 1 second average capacities of all the participating UEs.

Next we feed these capacities to the optimal rate allocation algorithm and we assume that the video that will be transmitted has a 10 second duration and its bit rate is 900kbps (we set the video application to request 90kb of data every 100ms). Thus the constraints change value every 100ms, causing final time duration of the slots in which the algorithm is run to be 100ms. We leave as a variable the size of the buffer of the UE video application. The tested values for the buffer size are 1620kb, 1890kb, 2160kb, 2430kb and 2700kb. Since the simulation is only 10 seconds, the more interesting value for the buffer turned to be the smallest of the above (1620kb). By using this relatively small buffer the problems of the real life application of the algorithm become more evident. So, in the simulations that will be discussed below the value of the buffer is set to 1620kb[9].

If the algorithm fails for one UE, we assume that it is impossible to fulfil the objectives of the experiment and is therefore ignored.

If it succeeds, we first calculate the value of $\sum_{i=0}^{99} \frac{r_i(t)}{c_i(t)}$[10], where $r_i(t)$ is the allocated rate for the slot $[t_i, t_{i+1}]$ and $c_i(t)$ is the corresponding capacity, in order to theoretically calculate the cost of this allocation.

For every UE that has a successful allocation, we set up a simulation in order to test whether it can receive the video without problems. We use the same scenario as in the first step, and we assign saturation traffic to all the UEs but the target UE. The target UE receives traffic from a video server. This server changes its rate according to the allocation.

We keep track of the utilized RBGs.

Finally, for benchmarking purposes we also run the same simulation but we set the video server to send CBR (Constant Bit Rate) traffic of 900kbps (an IP packet of 1125 bytes is transmitted every 10ms) and we record the number of received RBGs.

[9] The evaluation process is similar to the one used in [VECI13]

[10] This sum is the equivalent of $\frac{1}{T}\int_0^\infty \frac{r(t)}{c(t)} dt$, in the discrete time. Also, because all the videos tested have the same time duration of T=10 seconds, we can omit the factor $\frac{1}{T}$.

We calculate the gains of the application against the CBR benchmark, both theoretically by comparing the values of $\sum_{i=0}^{99} \frac{r_i(t)}{c_i(t)}$ of the two simulation runs and practically by comparing the number of received RBGs over time.

2.3.3.2.3 Simulation Scenarios

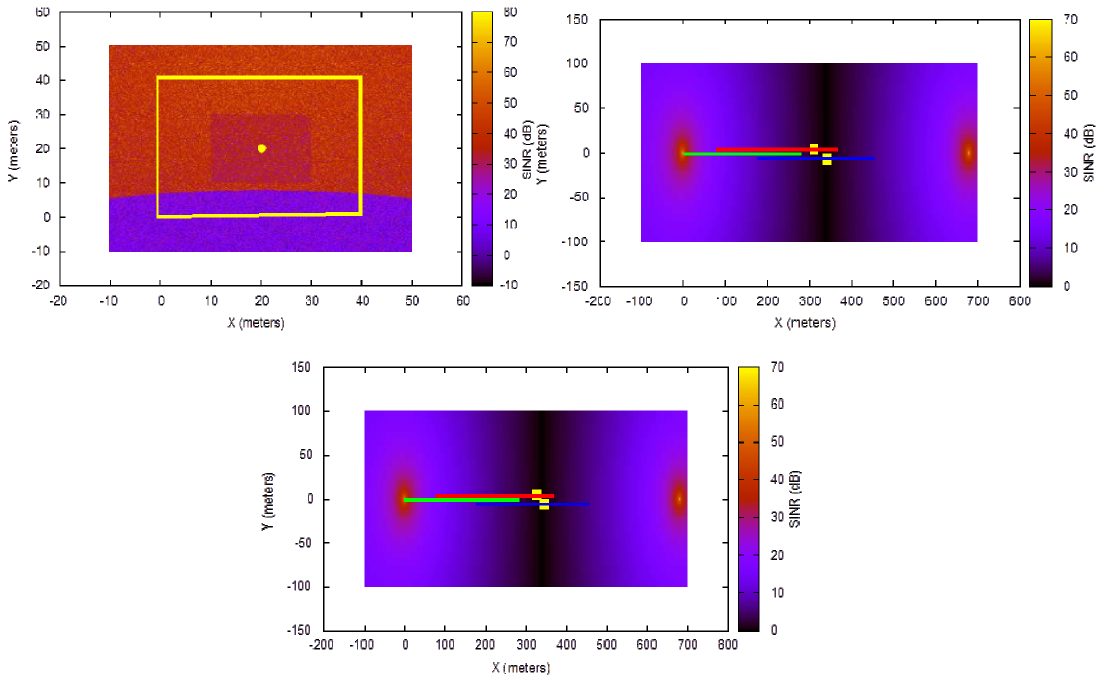


Figure 50: Radio environment maps of the 3 simulation scenarios, where we tested our model. The yellow boxes represent the locations of the handovers and the coloured lines the trajectories of the nodes.

We investigated the performance of the resource allocation algorithm in three characteristic LTE scenarios. In the vehicular scenario, depicted in the lower part of Figure 50, three moving nodes with different starting positions move from the cell on the left of the figure to the cell on the right of the figure with a speed of 108 km/h (30m/s). This very fast movement allows for quick variations of the CQI and thus of the capacity. The yellow squares in the figure symbolize the locations of the handover events. The pedestrian scenario, depicted in the upper right part of Figure 50, has exactly the same topology and node trajectories as the vehicular scenario, but the nodes move with speed of 1m/s. Also the handover of the node with the blue trajectory takes place 1 second earlier. Finally, the building scenario, depicted in the upper left part of Figure 50, has 4 moving nodes wandering around a building with a speed of 30m/s and one stationary node inside the building, shown in the center of the figure. The base station in that scenario is located in the upper part of the topology (not visible in the figure), thus, when the UEs move to the lower region of the topology they experience a significant drop in the SINR, because of the presence of the building.

In all scenarios one node is receiving video traffic with rates pre-calculated by the optimization algorithm while the rest of the nodes receive saturation traffic. In the vehicular and building scenario the video has 10 seconds duration and 900kbps bit rate and in the pedestrian scenario the video has 300 seconds duration and 900kbps bit rate. In every run of the simulation we vary the target UE, the granularity of the time interval over which the average is computed, the buffer size of the video application of the UE and the safety margin by which we raise the constraint for the minimum amount of data that can be received at any time instance. For benchmarking purposes, we also compare our solution to the constant bit rate scenario, where the video server transmits data with a constant rate of 900kbps.

2.3.3.2.4 Rate usages and buffer states

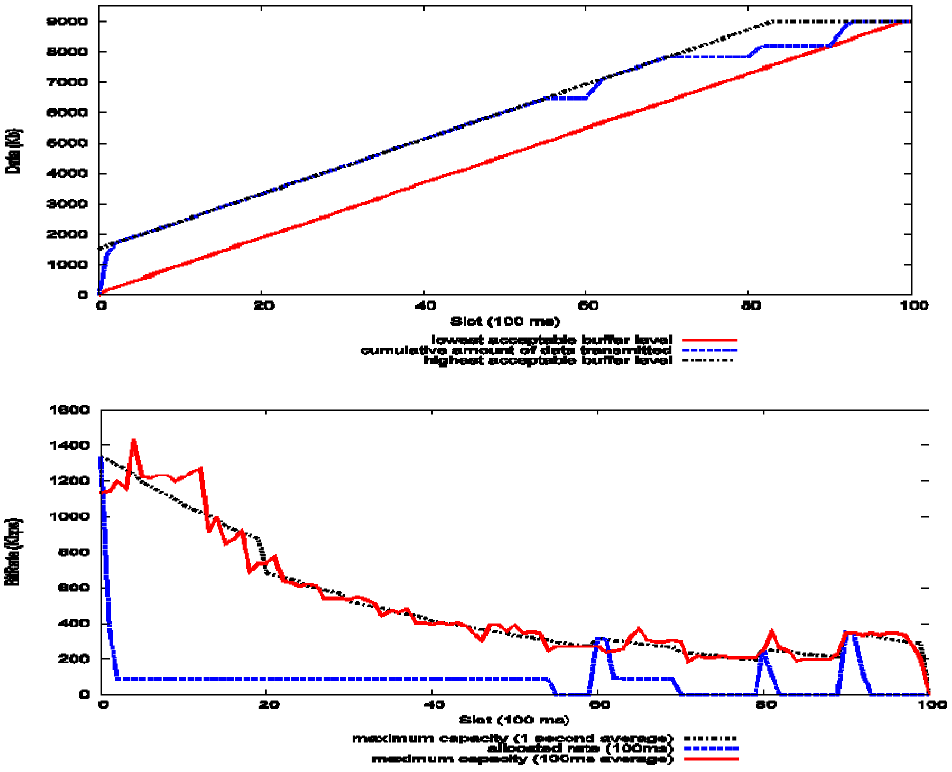


Figure 51. Example distribution of rates and buffer occupancy for node 1 of the vehicular scenario

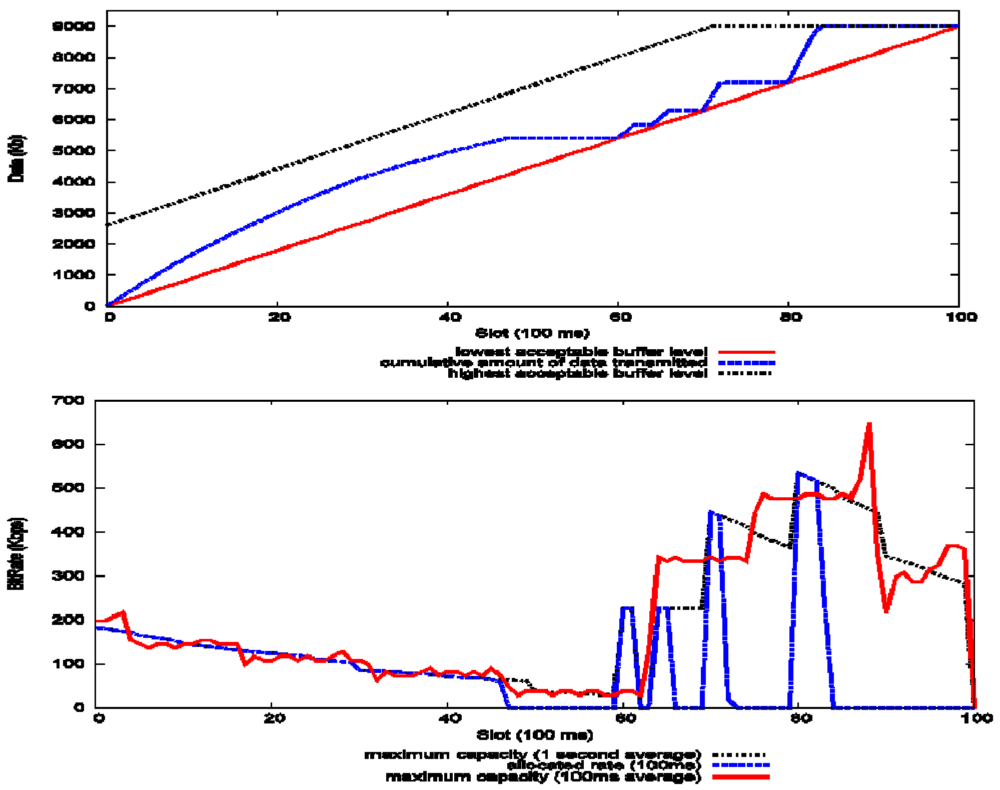
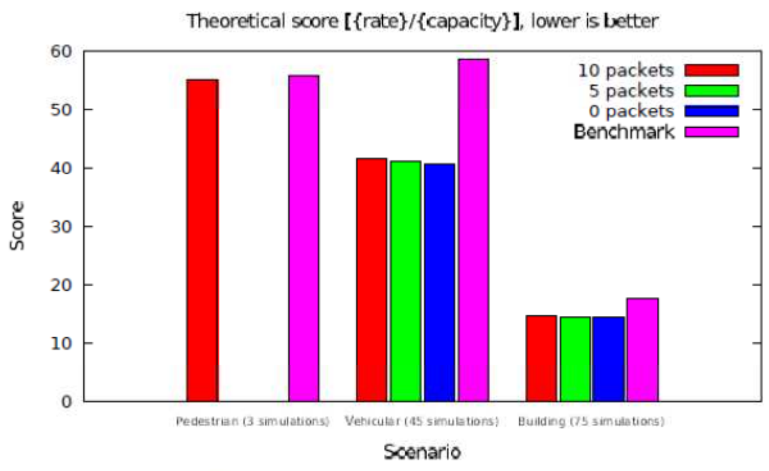


Figure 52. Example distribution of rates and buffer occupancy for node 3 of the vehicular scenario

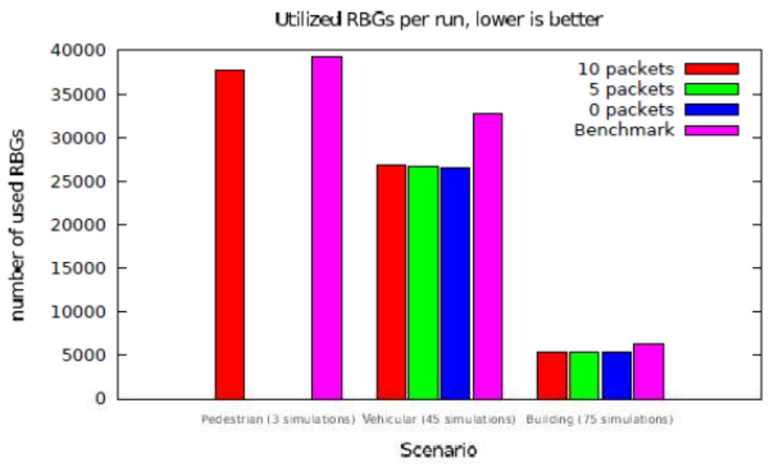
Initially, the figures above show the variations of the buffer levels and rate allocations for some of the nodes of the vehicular scenario.

These figures demonstrate how the rates are adapted to the channel conditions: the algorithm selects to use the full capacity when the conditions are good and relies on the UE buffer when capacity is low.

2.3.3.2.5 Cost reduction



(a) Theoretical score (average over 10 seconds)



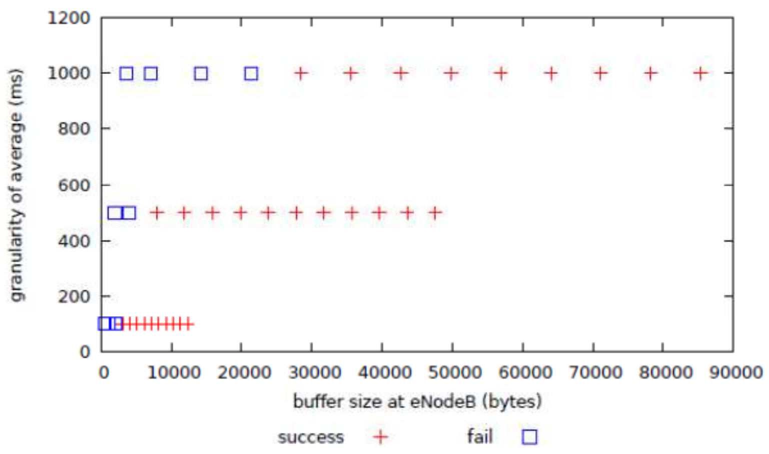
(b) Number of used RBGs (average over 10 seconds)

Figure 53. Comparison between the cost of different solutions in the three tested scenarios.

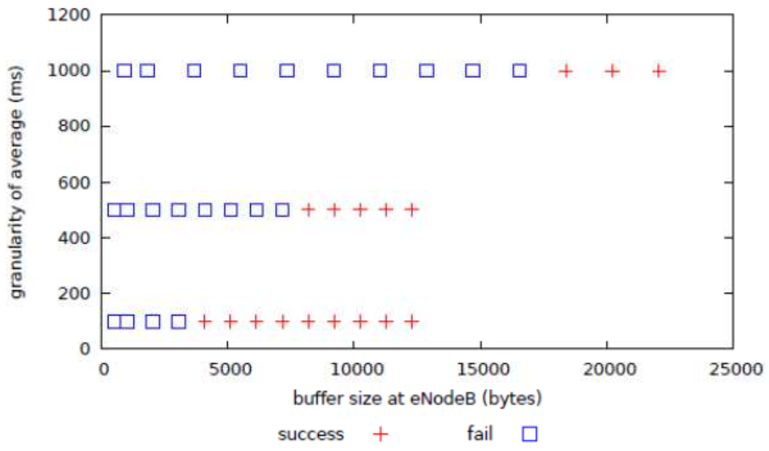
The aggregated results of both the theoretical cost and the cost in terms of used RBGs, among all the simulation runs is depicted in Figure 53. All versions of the algorithm greatly reduce both the theoretical cost and the number of RBGs needed to transmit the video data. As expected, the more robust a solution is, the more resources it requires. The difference for the various values of the safety margin is not very big though, especially in contrast to the cost of the benchmark application.

It should be noted that, in the pedestrian scenario, we only tested the approach that requires ten packets as safety margin for the low constraint. In this scenario we observe minimum gains between our optimized approach and the benchmark. This is caused by the slow variation of the channel. For long periods of time (significantly longer than time for which the buffer can support uninterrupted playback, without receiving any new packets), the CQI that the UE encounters exhibits very small variations. The optimization algorithm reduces the cost by scheduling packets sooner than when the benchmark would, in time intervals where conditions are favourable. The constraints though limit the amount of time in advance each packet can be transmitted. Thus, the cost of transmitting packets in the pedestrian scenario is almost the same regardless of how sooner the optimization algorithm schedules them, because the slots used have almost the same capacity.

2.3.3.2.6 Buffer size at eNodeB



(a) node 1



(b) node 2

Figure 54. The granularity of the time interval over which the average capacity is calculated, determines the error between the allocated rate and the real capacity and therefore the value of buffer at the eNodeB. Two of the nodes of the vehicular scenario are pictured.

Next, we show the relationship between the granularity of the time interval over which the average capacity is calculated and the buffer size at the eNodeB in Figure 54. We simulated the same scenario, gradually increasing the buffer size of the eNodeB. In all of our simulations, retransmissions are disabled, so if a packet is lost because of a buffer overflow at the eNodeB the total number of received packets is lower than the total number of transmitted packets and we suppose that we have a failure. As expected, higher granularity forces us to use significantly bigger buffers, in order to alleviate the effects of the wrong estimation of the real capacity and the drawbacks of our design decision to assume that the first slots of a time interval have higher capacity than calculated. The greater the error between the real capacity and the rate is, the greater the difference between the rate at which packets arrive at the eNodeB and depart from it, therefore requiring a bigger buffer in order to avoid overflows.

It should be noted that if the time interval is 100ms, the minimum required buffer size is smaller than the size of buffers that are currently present at the eNodeBs.

2.3.3.2.7 Robustness of the different safety margins of the low constraint

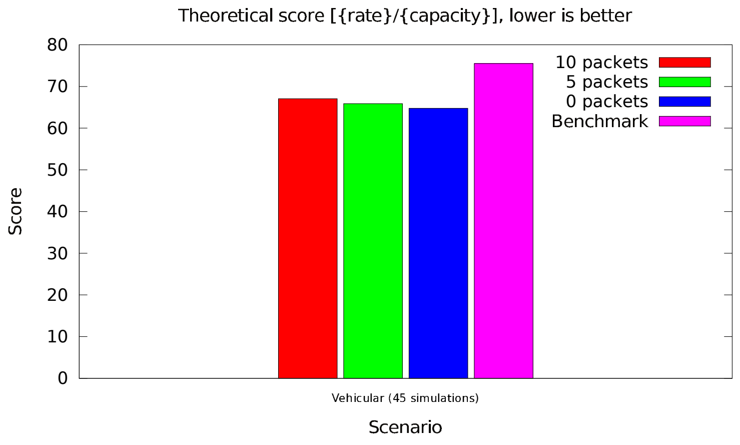
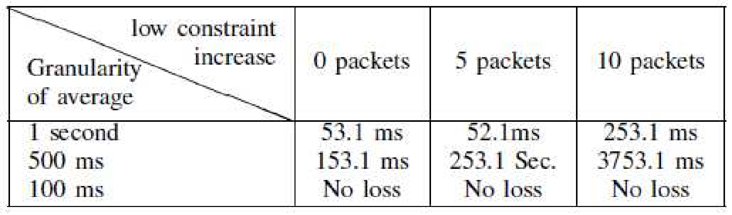


Figure 55. Average of the theoretic scores achieved for different values of the safety limit of the UE buffer, over all the simulations where packet losses occurred.

Table 28: Time instance when the first buffer underflow occurs.



Granularity of average \ low constraint increase	0 packets	5 packets	10 packets
1 second	53.1 ms	52.1ms	253.1 ms
500 ms	153.1 ms	253.1 Sec.	3753.1 ms
100 ms	No loss	No loss	No loss

In Figure 55 and Table 28 we present the details of a scenario where a node suffers from a loss of consecutive packets. This incident takes place right before the handover of the node with the red trajectory in the vehicular scenario. Before the handover, the node is at the edge of a congested cell, suffering from low CQI. After the handover the node arrives in an uncongested cell with similar CQI and as a result his capacity is significantly higher. Because the handover takes place in the middle of a time interval over which the average capacity is calculated, the last high capacity slots raise the average a lot more than the capacity of the first slots. Before the handover, packets are transmitted with the nominal capacity and as a consequence congregate on the buffer. Then, a seamless handover[11] takes place, resulting in loss of all the packets that were in the eNodeB buffer. Since, our scenarios do not support retransmissions, we are interested in detecting how much time is needed until the packet loss causes a buffer underflow. We consider that in case of retransmissions, the time difference between the loss of a packet and the buffer underflow event is equal to the amount of time that the UE would have to retransmit the packet. So, a bigger time interval is desirable.

In Table 28 we show when the buffer underflow events take place. It is clear that the 10 packets safety margin grants the system significantly more time to recover from the packet loss. In the 100ms scenario, we do not experience any packet loss, because the rate differs very little from the real capacity. In Figure 55, we depict the theoretical scores achieved in the same simulation runs.

[11] The packets of the first cell that are destined to the UE that did the handover are dropped instead of being forwarded to the new cell, as it is the case in the lossless handover.

2.3.3.2.8 Remarks

The simulations have shown that our solution achieves a good balance between robustness and low channel utilization, while significantly reducing the cost compared to the benchmark application.

2.3.3.2.9 Performance under uncertainties

In this section we show some preliminary results on the previous algorithm when a perfect knowledge of the future throughput availability is not available.

In order to study the impact of imprecise forecasts, we first prepared a set of synthetic throughput traces, where the correct values are summed to random errors of different amplitude. Then we fed the modified traces to the optimizer and we used the resulting bandwidth allocation on the real traces in order to evaluate the algorithm performance under uncertainties.

Unfortunately, a single run of the optimizer may not only be sufficient to allocate enough bandwidth for the whole content to be downloaded. Thus, we slightly modified the algorithm so that it is rerun after a number, K, of slot is used. As a result, the bandwidth knowledge on which the used allocation is based on is never older than K slots.

Although it is possible to run the algorithm at each slot (K=1), there exist pathologic condition for which it is not possible to compensate an allocation error. In the following plots (see Figure 56) we illustrated the performance degradation of the allocation algorithm for different size of the error and different values of K.

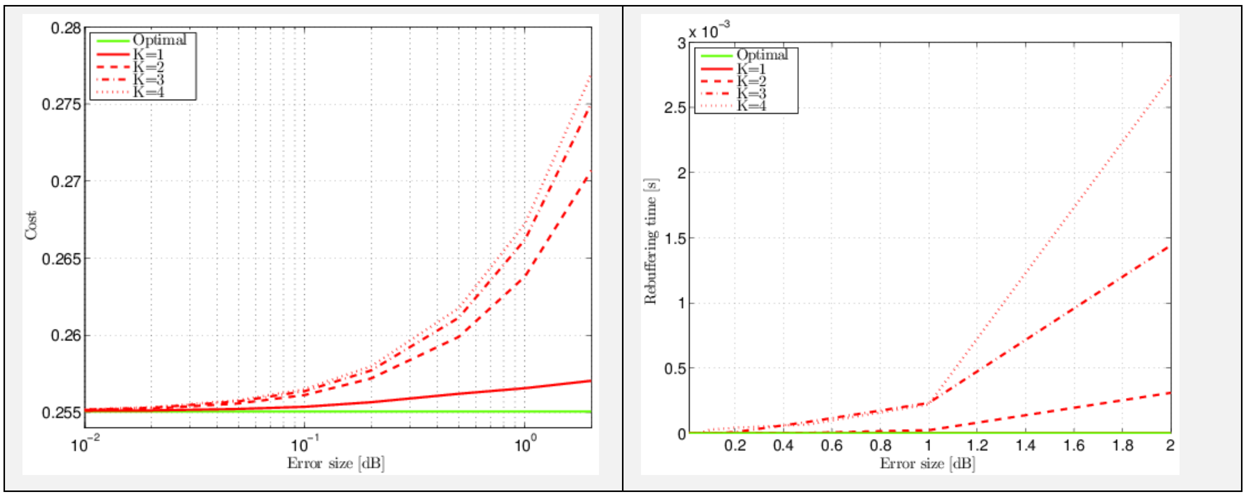


Figure 56. Resource allocation performance under uncertainties. Left: average download cost. Right: average buffer underrun time. Both plots show curves for different values of K (red plots) compared to the optimal results (green) for varying the size of the error.

To run these first tests we choose a particularly simple scenario where the bandwidth fluctuations are only due to the fading effect as the reference user is static in the cell. The reason of doing this was to verify whether the optimization algorithm suffered from uncertainties even under very favourable conditions.

As expected, the performance degradation is not very relevant, but it is increasingly large with the size of the error. Also, rerunning the algorithm after every slot is effective in reducing the performance degradation, but it is not always possible to compensate wrong predictions.

To this extent we are developing a new algorithm, which works on the predicted throughput time series so that at each step the size of the error is accounted for statistically in order to grant that the likelihood for the performance to be worse than the optimal is below a given probability.

2.3.4 Peer-to-peer

2.3.4.1 Identify communities

2.3.4.1.1 Introduction

The fact that users are interested in certain (multimedia) content items is often shared via online social networks, e.g., via the profiles that users provide or via the "like"s or comments they give on posts related to those content items. Information on who has declared interest (i.e., has seen, "liked" or commented on) in content items that were published and discussed upon in the past, can be used to predict the possible interest in a newly published content item. This, in turn, may be used to optimally distribute that new content item. In this section we tackle the question which is the best community to search for a (new) content item, given that users (friends or not) expressed their interests in pieces of content in the past.

There are two ways to determine communities (i.e., user clusters): 1) based on the content they consume [VLE13][HAN13] and 2) how they are related in a social network. How to exploit these clusters is tackled in other papers. In [HAN13] it is concluded that communities based on past content consumptions are better in predicting future content consumption. Other papers exploit relations between users (based on their content consumption) to improve content dissemination only implicitly constructing the communities. In [CH09] and [KUL10] content interdependencies on YouTube are exploited to improve networking performance. While [CHE09] designs a heuristic search technique that exploits the YouTube video graph to reduce the time for finding a video clip of interest in an overlay network, in [KUL10] the authors use various centrality measures of the related video graph to enhance the hit ratio of a network cache. Social information is also used in the peer-to-peer systems described in [LIN10] and [POU08] to enhance performance. Similarly, [CHE09] shows how analyzing tweets in twitter can improve the dissemination of the content tweeted about. In [CHA10] the user preferences are used for more efficient media delivery in online communities.

2.3.4.1.2 Defining user communities

We represent the information upon which the user communities are based in two matrices. The users are (assumed to be) labelled from 1 to N, where the first M users are the anchor users of which we know all the friends. The (MxN)-matrix U expressed the friendship relations: U_{mn}=1 if anchor user m is a friend (which is a bi-directional relation in Facebook) of (anchor or other) user n and 0 otherwise. Note that, by definition, the diagonal entries of U are 0, i.e., U_{mm}=0. Note that the data set contains no information on the friend list of the users other than the anchor users: of the other users we only know if they have friends among the anchor users; we do not know anything about their other friends. The movies are labelled by identifiers from 1 to K. The (NxK)-matrix R expresses the content interests: R_{nk}=1 if (anchor or other) user n has expressed an interest in content item k (at least if that user shared that information publicly). The characteristics of these matrices are discussed in Section 2.1.3.

We define two auxiliary matrices. The (Nx1)-matrix A with user activities as is $A=R\cdot e_K$, where e_K is the (Kx1)-matrix with all entries equal to 1. The (MxN)-matrix V with co-interests. It consists of the first M rows of the matrix $R\cdot R^t$ of which diagonal elements (which are just the entries of A) were put to 0.

We only define user communities for anchor users. We represent the information which (anchor or other) users are in the community of the anchor users in a (MxN)-matrix C: C_{mn}=1 if (anchor or other) user n is in the community of anchor user m. By definition we set the diagonal entries to 0, i.e., C_{mm}=0 (which is also why we adopted the definition U_{mm}=0 and V_{mm}=0). Notice that an obvious choice for communities is just the list of friends, in which case C=U. Usually this list of friends is large (on average 253.2 users in the Facebook dataset).

We consider three variants of reducing these communities based on the list of friends. Starting from C=U we prune the communities based on one of three rules. In the first scenario we just randomly remove users with a probability (1-P) (such that on average the user communities reduce to a fraction P of the original size). In the second scenario we prune the list based on the user activity, i.e., we only keep user n (member of the friend lists) in the community if it has an activity A_n larger than a threshold α. In the third scenario we prune the list based on the co-interests, i.e., we only keep n (member of the friend lists) in the community of anchor user m if the co-interest V_{mn} is larger than a threshold β.

Finally, we consider one method to construct communities without taking the friend lists into account. We put user n in the community of anchor user m if the co-interest V_{mn} is larger than a threshold β. Notice that this method yields larger communities than the third variant of the method based on the friend list: users having a large co-interest with the anchor user, but not in the list of friends are included.

2.3.4.1.3 Performance analysis

We use the Facebook data set described in Section 2.1.3 to assess the performance. We try to answer the following question: if an anchor user has declared an interest in a new movie, what is the probability that he/she finds a user with interest for the same movie in his/her community. Remark that this is a trade-off: the larger the user communities, the higher this probability.

Since the data set we consider here, does not have the concept of time, we split the set of movies in a training and a test set by making this binary choice for each movie with a probability of 0.5. The resulting training set (of about half of the movies) is considered to be the set of “old” movies, upon which the communities are built, while the test set (with the rest of the movies) is considered to be the set of “new” movies that is used to assess the performance.

Concretely, the matrix R is split into two sub-matrices, the columns of this matrix that are associated in to a movie in the training set are put in the (NxK_{tst})-matrix R_{tst} while the others are put in the (NxK_{trn})-matrix R_{trn}, where $K_{tst}+K_{trn}=K$. The activity matrix A and co-interest matrix V are constructed (as explained in Section 2.3.4.1.2) based on the training matrix R_{trn}. This provides enough information (together with probability P and thresholds α and β) to determine the community of each of the anchor users, which as explained we store under the form of a (MxN)-matrix C.

Once the communities are known, we construct the (MxK_{tst})-matrix $H=C\cdot R_{tst}$, where H_{mk} indicates the number of users in the community of anchor user m that have expressed an interest in movie k. The average community size can then be easily calculated as the sum of the number of 1s in the matrix C divided by M.

The first M rows of the matrix R_{tst} identify the interests of the anchor users. For every entry that is 1 in R_{tst}, say (m,k), we check if the corresponding entry of H, i.e., H_{mk} is 0 or not. If it is 0 movie k that anchor user m expressed interest in, was not found interesting by any of the users in his/her community, and we catalogue this as a miss. In the other case we have a hit. We define the hit ratio as the ratio between the total number of hits and the total number interests of anchor users in the test set (i.e., the number of 1s in the first M rows of R_{tst}).

Figure 57 shows the hit ratio as a function of the average community size for the four methods to determine communities. The error bars (corresponding to one standard deviation) stem from the fact we ran the experiments 10 times, splitting the data set over different training and test sets. The dotted line, labelled with “all users”, is for the case when the communities consist of all users (except the anchor users), in which case the hit ratio is on average 86% and the community size 10768 users. Randomly pruning the list of friends (with a probability P=0, 0.1, …, 0.9) performs the worst. For pruning the friend list based on activity we used threshold values α=0, 5, …, 45 movies. For the two methods based on co-interest, we used the threshold values β=1, 2, …, 5. All three other methods perform more or less the same, i.e., within the standard deviation, which implies that on average we do not need any information about friendship relations and that we can just use the co-viewing information to determine communities.

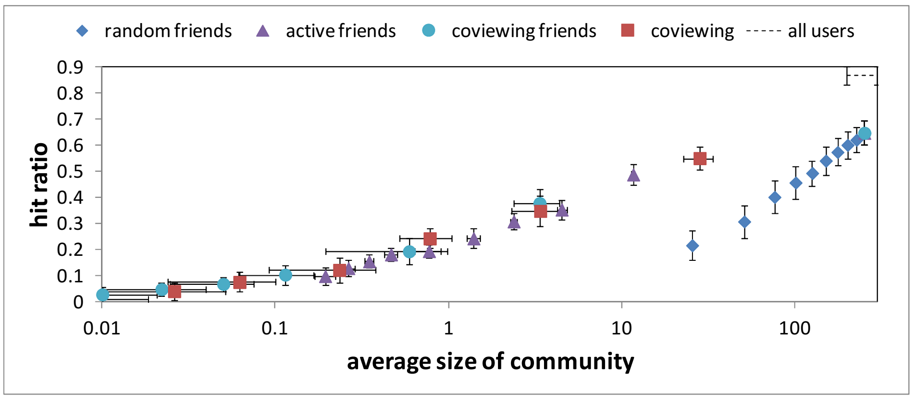


Figure 57. Hit ratio as a function of the average community size

2.3.4.2 Peer-assisted content delivery

2.3.4.2.1 Introduction

Nowadays we are witnessing a huge growth in users' engagement to OSNs. Major and popular OSNs such as Facebook with more than 1 Billion subscribers are producing a considerable portion of the Internet traffic because in addition to the social activity performed on these OSNs, a large portion of the content shared has big size such as Videos and Photos. Furthermore, major Internet video streaming providers, such as YouTube, integrate their services in the popular social networks. Therefore users can watch the 3rd party videos directly in their social network pages and the social network will redirect the traffic load for them.

Furthermore, we have to consider that a large portion of the videos shared in OSNs correspond to UGC, independently whether they are natively uploaded to OSNs or they are videos uploaded in third party services (e.g., Youtube) and shared through the OSN. The vast majority of these videos present actually a low popularity receiving at most a few tens of views. Therefore, using standard delivery mechanisms such as CDNs for this kind of content is rather expensive, since you need to cache the videos for only a few views in many cases. In order to alleviate this load, end-users could directly cache locally the user-generated videos they share through their OSNs accounts and serve them to other users clicking through their OSNs pages. It is very likely that most of the views for a video that has been shared by a user U in its OSN wall will consecutively come from U's friends in that OSN. In this situation, the OSNs can refer each of the friends' click to either the CDN node or central server hosting the video or to U's local premises. This decision will be made based on the location of U's friend with respect to U. Overall the Social-peer-assisted content delivery will help to offload a considerable portion of user-generated video traffic by means of applying a p2p approach.

In summary, our solution will locate videos shared by a user U through an OSN both in, (i) U's local premises with a 24/7 connectivity like U's set-up box in her home, and (ii) in the standard location where the video is located with current solutions (e.g., CDN node, centralized server, etc). Therefore, there will be two locations from where U's social friends can retrieve the video. Both locations will have the ability to stream the video.

2.3.4.2.2 Practical demonstrator through a Facebook application

We have implemented an application named “SocialiVideo” to demonstrate the use of the proposed solution in the context of the most popular OSN, i.e., Facebook. Our application manages Local (e.g., link to the local Facebook user set-up box) and Facebook (links to videos stored in CDN nodes, Facebook severs, or third players such as Youtube) links and stores for each uploaded video both links in its database. In case that the social connected friends try to watch a video inside the profile of the person, the application will forward one of the links to the requester based on the location of the video owners and the requester. To identify the locations, the application is able to collect the location of people in two ways: (i) *Facebook Profile location information attribute* that is the current city of the user (ii) *IP address of the users* that shows the users’ actual location. By using other 3rd party databases we are able to locate the users’ continent, country, city, postal code and ISP (Internet Service Provider).

The application will collect the mentioned location information for all users that are using the application and store them in its database. Later for each video request the application first checks the location of both parties (video location and viewer location) and if they are in the same location zone (can be considered same city or same country) the application will stream the video from the local link otherwise the stream will be from Facebook link.

This simple but efficient way of content delivery mechanism can reduce the load of Social network CDNs as well as the traffic load. Next we look to the application in detail by providing comprehensive detail information.

2.3.4.2.2.1 Software Technical detail

In this section we aim to provide technical details about the implementation of the application as well as explanation of how it works. It is worth mentioning that the current version of the application is a first version and is not finalized yet.

Apart from the technical part of the implementation, the application has three main parts.

(i) First a procedure to retrieve users’ location information based on their Facebook profile attributes and the geographical information from their IP address
(ii) Users videos both storage in a local accessible server
(iii) A decision making procedure based on the video and users location and streaming the video

Next we explain each of these parts in detail.

Implementation details and Architecture Overview

The application is developed on *herokuapp* with capability to login with Facebook accounts. Figure 58 shows the global view of the application architecture.

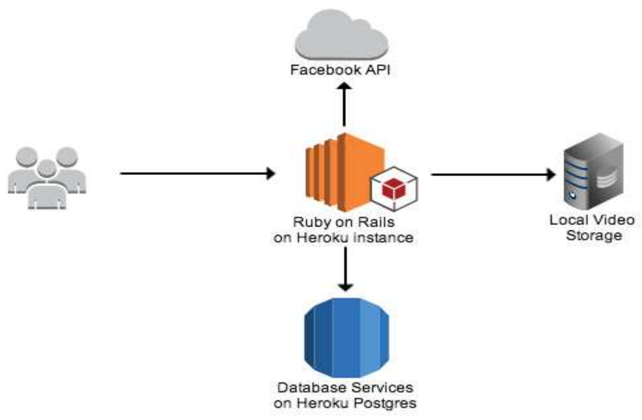


Figure 58. The architecture of the application

The core of the application is written in Ruby on Rails platform and it is hosted in Heroku instance, which is a PaaS (Platform as a Services) that enables the development to host the Ruby on Rails application. The Heroku instance also communicates with an internal database service on Heroku Postgres that provide postgresql database service to store application data. On the other hand, the application communicates with Facebook API to retrieve Users information for (i) Location and (ii) Videos' List.

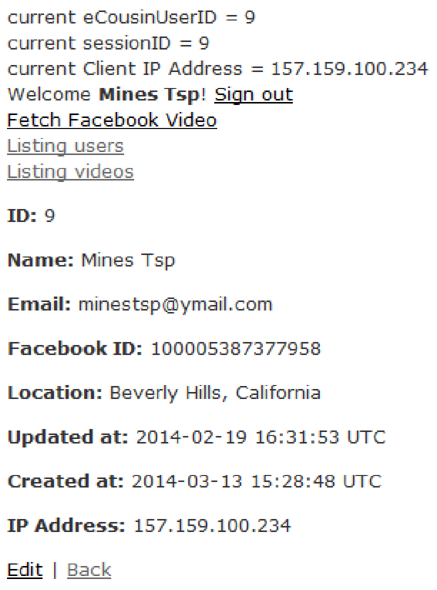
Also the application communicates with the local server to retrieve users' local video information.

On the development environment side, we used online development service from Nitrous.io that ensures the development environment will be the same to the production environment. Every committed code in nitrous.io box will be deployed in the production and Heroku instance and it also includes Unit Test for validation of codes.

We have implemented the first version of application and an open access to it is available for testing and validation. The URL address of the application is as follows: http://ecousin-tsp-fb.herokuapp.com/

Users' Location information retrieval

As it is mentioned before, users should login to the application with their Facebook account, gives permission to the application and after that the application will be able to retrieve users' information. Figure 59 shows a summary of the information that the application collects after the users' login for a sample user namely "Mines TSP". As it is shown in the figure, the main information is the location information from Facebook location attributes and the IP address of the user.



current eCousinUserID = 9
current sessionID = 9
current Client IP Address = 157.159.100.234
Welcome **Mines Tsp**! Sign out
Fetch Facebook Video
Listing users
Listing videos

ID: 9

Name: Mines Tsp

Email: minestsp@ymail.com

Facebook ID: 100005387377958

Location: Beverly Hills, California

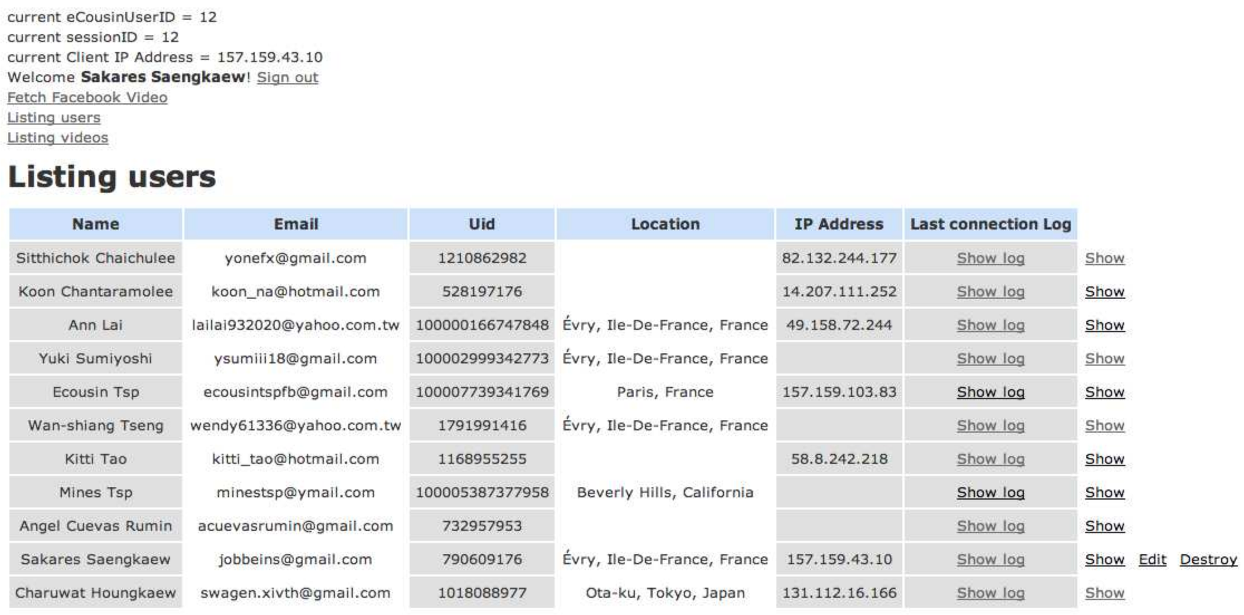
Updated at: 2014-02-19 16:31:53 UTC

Created at: 2014-03-13 15:28:48 UTC

IP Address: 157.159.100.234

Edit | Back

Figure 59. User information, collected from login to the application.



current eCousinUserID = 12
current sessionID = 12
current Client IP Address = 157.159.43.10
Welcome **Sakares Saengkaew**! Sign out
Fetch Facebook Video
Listing users
Listing videos

Listing users

Name	Email	Uid	Location	IP Address	Last connection Log			
Sitthichok Chaichulee	yonefx@gmail.com	1210862982		82.132.244.177	Show log	Show		
Koon Chantaramolee	koon_na@hotmail.com	528197176		14.207.111.252	Show log	Show		
Ann Lai	lailai932020@yahoo.com.tw	100000166747848	Évry, Ile-De-France, France	49.158.72.244	Show log	Show		
Yuki Sumiyoshi	ysumiii18@gmail.com	100002999342773	Évry, Ile-De-France, France		Show log	Show		
Ecousin Tsp	ecousintspfb@gmail.com	100007739341769	Paris, France	157.159.103.83	Show log	Show		
Wan-shiang Tseng	wendy61336@yahoo.com.tw	1791991416	Évry, Ile-De-France, France		Show log	Show		
Kitti Tao	kitti_tao@hotmail.com	1168955255		58.8.242.218	Show log	Show		
Mines Tsp	minestsp@ymail.com	100005387377958	Beverly Hills, California		Show log	Show		
Angel Cuevas Rumin	acuevasrumin@gmail.com	732957953			Show log	Show		
Sakares Saengkaew	jobbeins@gmail.com	790609176	Évry, Ile-De-France, France	157.159.43.10	Show log	Show	Edit	Destroy
Charuwat Houngkaew	swagen.xivth@gmail.com	1018088977	Ota-ku, Tokyo, Japan	131.112.16.166	Show log	Show		

Figure 60. List of users (current front end of the application in Beta version).

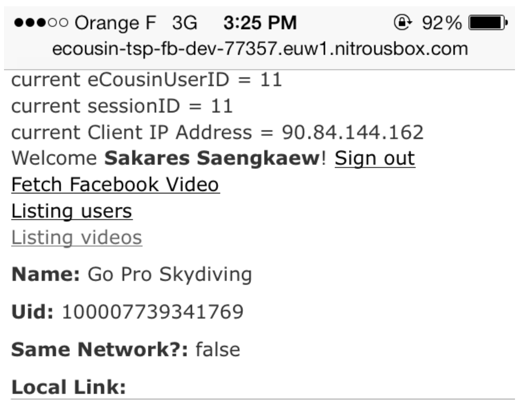
Next, the application uses an online API namely "*ipaddresslabs*"[12] to identify the geographical location of the users based on the retrieved IP address in the login process. By this API we are able to find the geographical information as follows: continent name, country name, city, postal code (if available), ISP name. Table 29 shows an example of the result obtained from this API that will be stored in our application database.

[12] http://api.ipaddresslabs.com/iplocation/v1.7/locateip?key=demo&ip=58.8.242.218&format=json

Table 29: The Stored Log from ipaddresslabs API information for a sample user "Mines TSP" in DB

```
{"id":9,"name":"Mines Tsp","email":"minestsp@ymail.com", "uid":"100005387377958", "location":"Beverly Hills, California","created_at":"2014-02-19T16:31:53.035Z","updated_at":"2014-03-13T15:28:48.501Z","ip_address":"157.159.100.234", "geolocation":{"continent_code":"EU","continent_name":"Europe","country_code_iso3166alpha2":"FR","country_code_iso3166alpha3":"FRA","country_code_iso3166numeric":"250","country_code_fips10-4":"FR","country_name":"France","region_code":"FRA8","region_name":"Ile-de-France","city":"Évry","postal_code":"-","metro_code":"-","area_code":"-","latitude":48.6328,"longitude":2.4405,"isp":"TMSP (ex INT Evry)","organization":"TMSP (ex INT Evry)"}}
```

Also the application has the capability to update the IP address of users when they are connecting from different location or different networks. For example Figure 61 shows when a user is connected from his mobile, the IP address changed and get updated.


●●●○○ Orange F 3G 3:25 PM 92%
ecousin-tsp-fb-dev-77357.euw1.nitrousbox.com

current eCousinUserID = 11
current sessionID = 11
current Client IP Address = 90.84.144.162
Welcome **Sakares Saengkaew**! Sign out
Fetch Facebook Video
Listing users
Listing videos

Name: Go Pro Skydiving

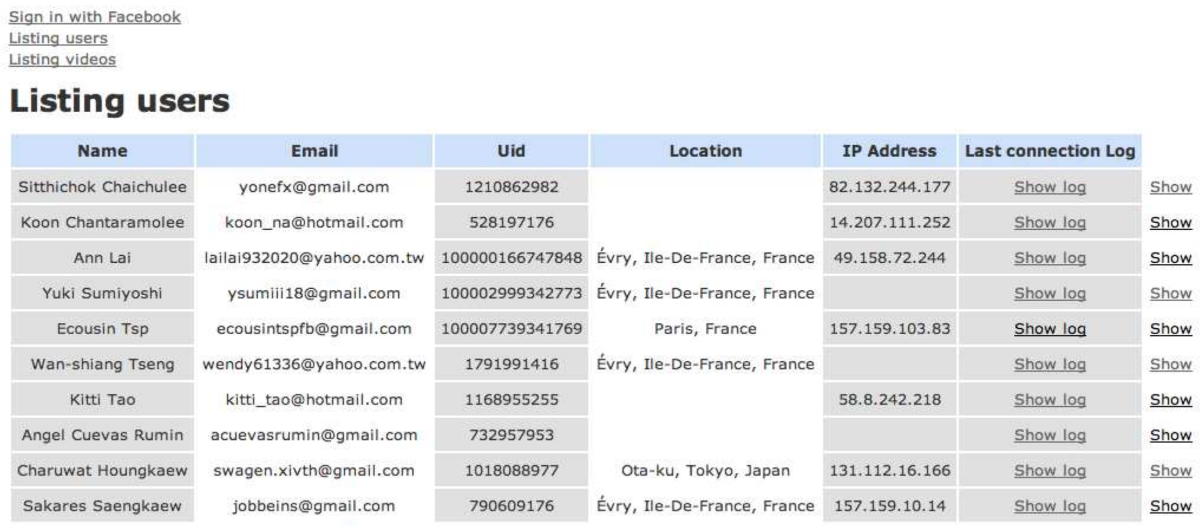
Uid: 100007739341769

Same Network?: false

Local Link:

Figure 61. Home Page of Application

Therefore, for each user who has been subscribed to the application we have the Facebook location information as well as their IP address locations. Figure 62 shows a summarized list of subscribed users to the application that in the current beta version of the application is considered as home page of the application.



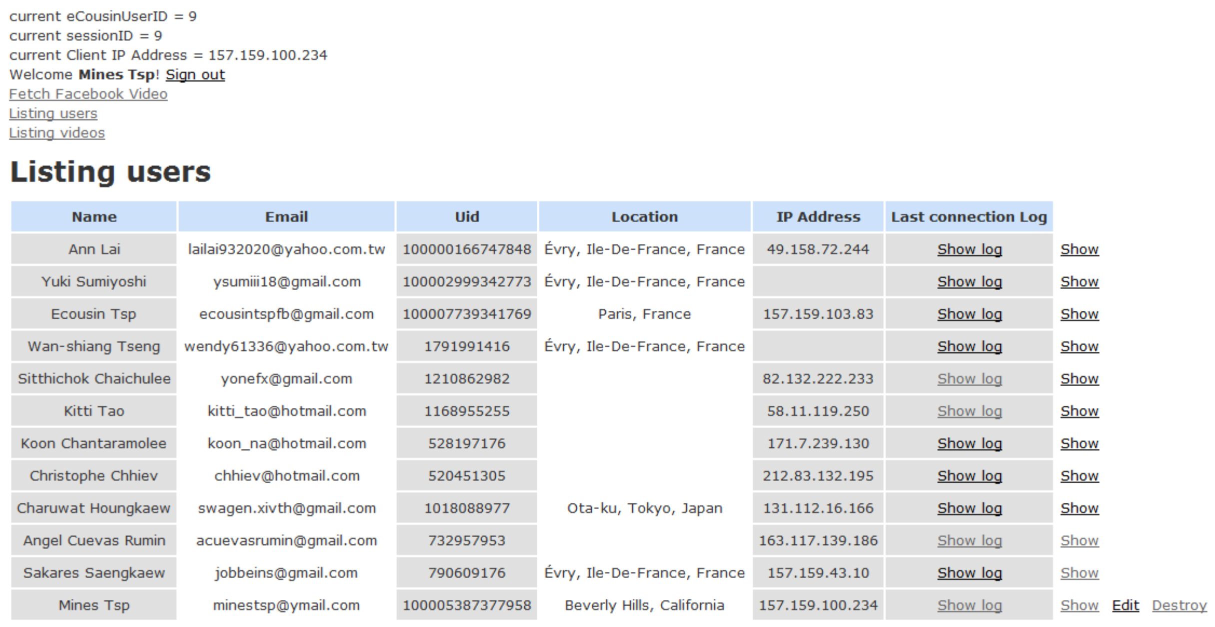
Sign in with Facebook
Listing users
Listing videos

Listing users

Name	Email	Uid	Location	IP Address	Last connection Log	
Sitthichok Chaichulee	yonefx@gmail.com	1210862982		82.132.244.177	Show log	Show
Koon Chantaramolee	koon_na@hotmail.com	528197176		14.207.111.252	Show log	Show
Ann Lai	lailai932020@yahoo.com.tw	100000166747848	Évry, Ile-De-France, France	49.158.72.244	Show log	Show
Yuki Sumiyoshi	ysumiii18@gmail.com	100002999342773	Évry, Ile-De-France, France		Show log	Show
Ecousin Tsp	ecousintspfb@gmail.com	100007739341769	Paris, France	157.159.103.83	Show log	Show
Wan-shiang Tseng	wendy61336@yahoo.com.tw	1791991416	Évry, Ile-De-France, France		Show log	Show
Kitti Tao	kitti_tao@hotmail.com	1168955255		58.8.242.218	Show log	Show
Angel Cuevas Rumin	acuevasrumin@gmail.com	732957953			Show log	Show
Charuwat Houngkaew	swagen.xivth@gmail.com	1018088977	Ota-ku, Tokyo, Japan	131.112.16.166	Show log	Show
Sakares Saengkaew	jobbeins@gmail.com	790609176	Évry, Ile-De-France, France	157.159.10.14	Show log	Show

Figure 62. Home Page of Application

Figure 63 shows the home page after the sample user logged in.



current eCousinUserID = 9
current sessionID = 9
current Client IP Address = 157.159.100.234
Welcome **Mines Tsp**! Sign out
Fetch Facebook Video
Listing users
Listing videos

Listing users

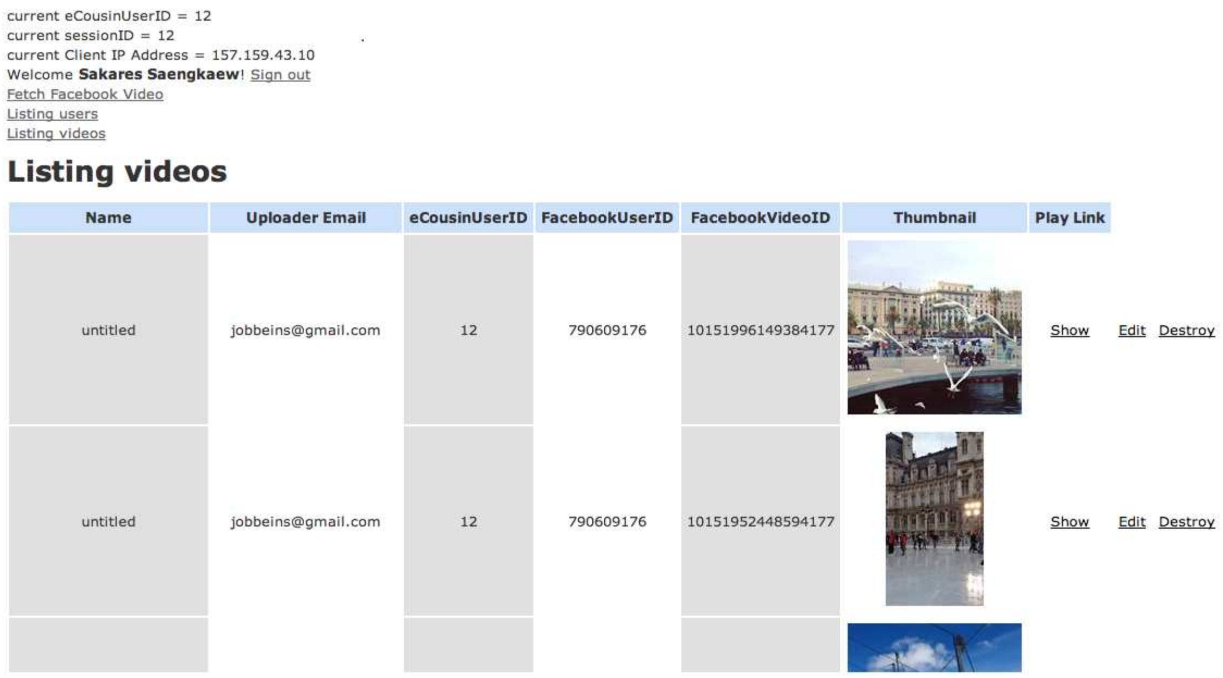
Name	Email	Uid	Location	IP Address	Last connection Log			
Ann Lai	lailai932020@yahoo.com.tw	100000166747848	Évry, Ile-De-France, France	49.158.72.244	Show log	Show		
Yuki Sumiyoshi	ysumiii18@gmail.com	100002999342773	Évry, Ile-De-France, France		Show log	Show		
Ecousin Tsp	ecousintspfb@gmail.com	100007739341769	Paris, France	157.159.103.83	Show log	Show		
Wan-shiang Tseng	wendy61336@yahoo.com.tw	1791991416	Évry, Ile-De-France, France		Show log	Show		
Sitthichok Chaichulee	yonefx@gmail.com	1210862982		82.132.222.233	Show log	Show		
Kitti Tao	kitti_tao@hotmail.com	1168955255		58.11.119.250	Show log	Show		
Koon Chantaramolee	koon_na@hotmail.com	528197176		171.7.239.130	Show log	Show		
Christophe Chhiev	chhiev@hotmail.com	520451305		212.83.132.195	Show log	Show		
Charuwat Houngkaew	swagen.xivth@gmail.com	1018088977	Ota-ku, Tokyo, Japan	131.112.16.166	Show log	Show		
Angel Cuevas Rumin	acuevasrumin@gmail.com	732957953		163.117.139.186	Show log	Show		
Sakares Saengkaew	jobbeins@gmail.com	790609176	Évry, Ile-De-France, France	157.159.43.10	Show log	Show		
Mines Tsp	minestsp@ymail.com	100005387377958	Beverly Hills, California	157.159.100.234	Show log	Show	Edit	Destroy

Figure 63. After clicked "Sign in with Facebook"

Storing Videos

The next part of the application is retrieving users' videos information and storing them in the local server of each user that is accessible for other people. The goal is for videos that users want to upload or share in their Facebook wall page, the application stores two links in its database. First the main Facebook link that would stream the video from Facebook servers and second a local link that would stream video directly from the local server of the users. For each video of each user, these two links will be generated and will be stored in the application databases.

After users login in the application there is a possibility to fetch users' videos links and afterward we will have a list of videos per user. By clicking to the "Fetch Facebook Video" link, the application will download all Facebook videos that are also stored in the local server of a given user as it is shown in Figure 64. For each video we will have its associated information such as uploader name, id, IP, location and so on.



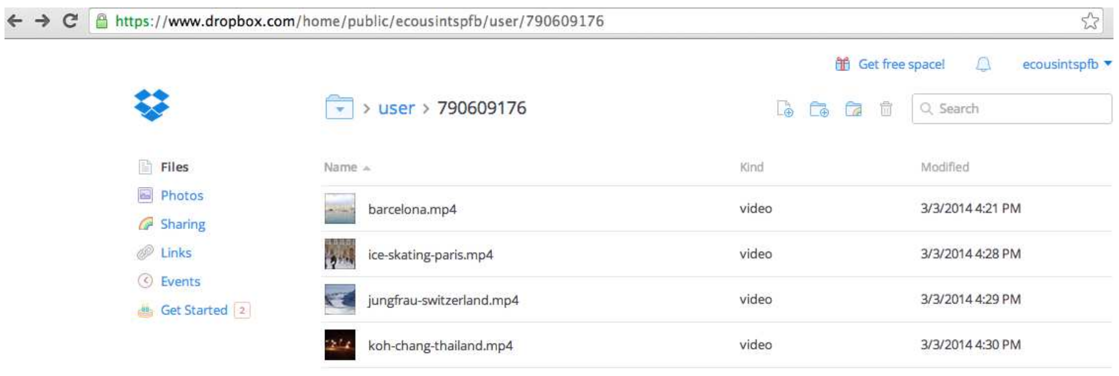
current eCousinUserID = 12
current sessionID = 12
current Client IP Address = 157.159.43.10
Welcome **Sakares Saengkaew**! Sign out
Fetch Facebook Video
Listing users
Listing videos

Listing videos

Name	Uploader Email	eCousinUserID	FacebookUserID	FacebookVideoID	Thumbnail	Play Link		
untitled	jobbeins@gmail.com	12	790609176	10151996149384177		Show	Edit	Destroy
untitled	jobbeins@gmail.com	12	790609176	10151952448594177		Show	Edit	Destroy

Figure 64. After clicked "Fetch Facebook Video". App will update the user's videos list.

For the current version of this application, we have considered a dropbox folder as our local server. Therefore the videos will be stored in that dropbox folder and will be served directly from dropbox. As it is shown in the Figure 65, for each user, we create (in first version manually) a folder named as the user facebook id and store the user's videos.

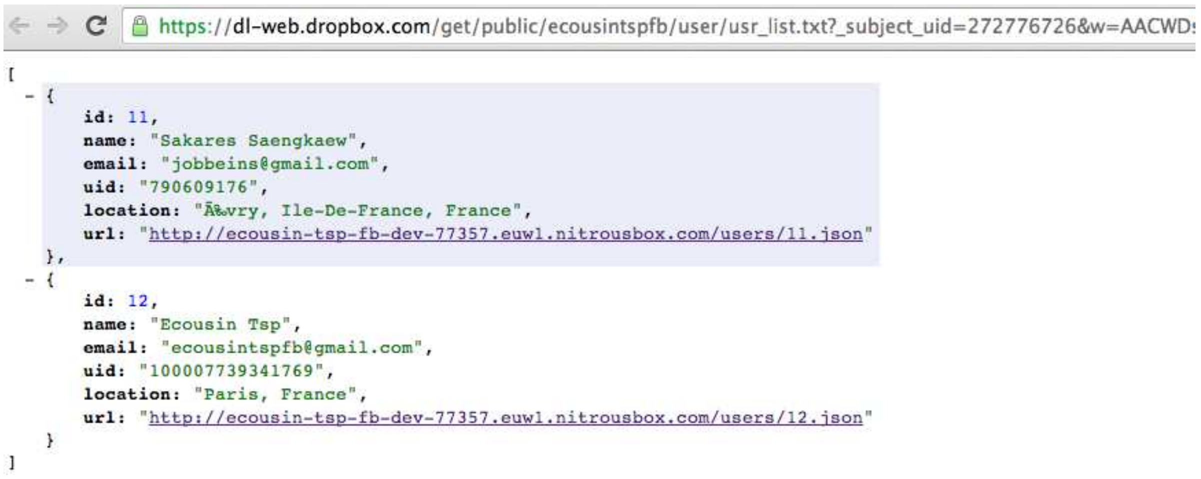


https://www.dropbox.com/home/public/ecousintspfb/user/790609176

Name	Kind	Modified
barcelona.mp4	video	3/3/2014 4:21 PM
ice-skating-paris.mp4	video	3/3/2014 4:28 PM
jungfrau-switzerland.mp4	video	3/3/2014 4:29 PM
koh-chang-thailand.mp4	video	3/3/2014 4:30 PM

Figure 65. Simple local storage with Dropbox.

Next in dropbox we will have a list of credential users who have uploaded their video in the local server as demonstrated in Figure 66.

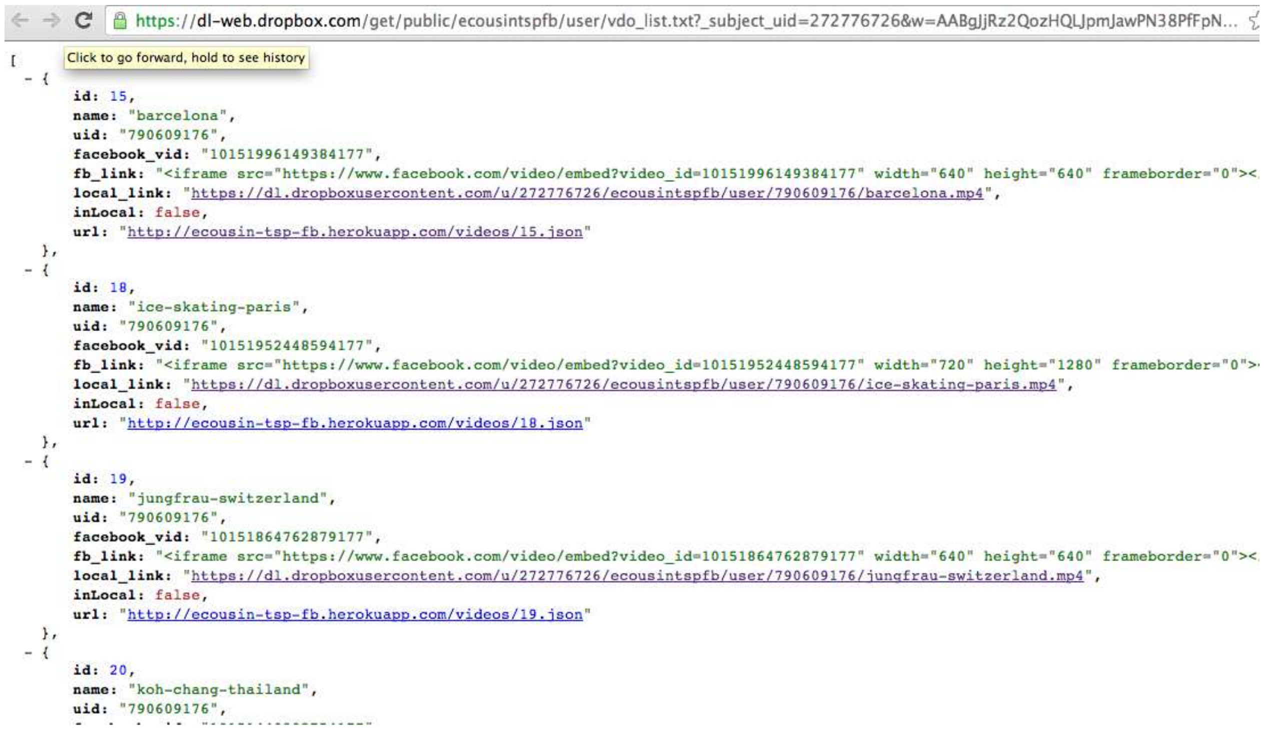


https://dl-web.dropbox.com/get/public/ecousintspfb/user/usr_list.txt?_subject_uid=272776726&w=AACWD

```
[
  - {
        id: 11,
        name: "Sakares Saengkaew",
        email: "jobbeins@gmail.com",
        uid: "790609176",
        location: "Ã‰vry, Ile-De-France, France",
        url: "http://ecousin-tsp-fb-dev-77357.euwl.nitrousbox.com/users/11.json"
    },
  - {
        id: 12,
        name: "Ecousin Tsp",
        email: "ecousintspfb@gmail.com",
        uid: "100007739341769",
        location: "Paris, France",
        url: "http://ecousin-tsp-fb-dev-77357.euwl.nitrousbox.com/users/12.json"
    }
]
```

Figure 66. List of credential user who has uploaded their video in local server

We also have a list of the uploaded video to the local server with their associated information including their local and Facebook links as shown in Figure 67.



https://dl-web.dropbox.com/get/public/ecousintspfb/user/vdo_list.txt?_subject_uid=272776726&w=AABgJjRz2QozHQLJpmJawPN38PfFpN...

```
[
  - {
        id: 15,
        name: "barcelona",
        uid: "790609176",
        facebook_vid: "10151996149384177",
        fb_link: "<iframe src="https://www.facebook.com/video/embed?video_id=10151996149384177" width="640" height="640" frameborder="0"><
        local_link: "https://dl.dropboxusercontent.com/u/272776726/ecousintspfb/user/790609176/barcelona.mp4",
        inLocal: false,
        url: "http://ecousin-tsp-fb.herokuapp.com/videos/15.json"
    },
  - {
        id: 18,
        name: "ice-skating-paris",
        uid: "790609176",
        facebook_vid: "10151952448594177",
        fb_link: "<iframe src="https://www.facebook.com/video/embed?video_id=10151952448594177" width="720" height="1280" frameborder="0">
        local_link: "https://dl.dropboxusercontent.com/u/272776726/ecousintspfb/user/790609176/ice-skating-paris.mp4",
        inLocal: false,
        url: "http://ecousin-tsp-fb.herokuapp.com/videos/18.json"
    },
  - {
        id: 19,
        name: "jungfrau-switzerland",
        uid: "790609176",
        facebook_vid: "10151864762879177",
        fb_link: "<iframe src="https://www.facebook.com/video/embed?video_id=10151864762879177" width="640" height="640" frameborder="0"><
        local_link: "https://dl.dropboxusercontent.com/u/272776726/ecousintspfb/user/790609176/jungfrau-switzerland.mp4",
        inLocal: false,
        url: "http://ecousin-tsp-fb.herokuapp.com/videos/19.json"
    },
  - {
        id: 20,
        name: "koh-chang-thailand",
        uid: "790609176",
```

Figure 67. List of credential user who has uploaded their video in local server

Decision mechanism and streaming

In the current version of the application, the decision part is as follows: for a video (video A) that a user (User A) wants to watch, the application will check the location of the user A and the location of the Video A (owner of the Video A) to see if they are in a same domain or not. Domain here can be considered as a City, ISP or even country. In this first version of the application we are considering City as domain for this decision step.

Next the application will stream the video from one of the available links (Local or Facebook original).

Figure 68 shows a video that is running from local link and Figure 69 a video that is running from the Facebook link.

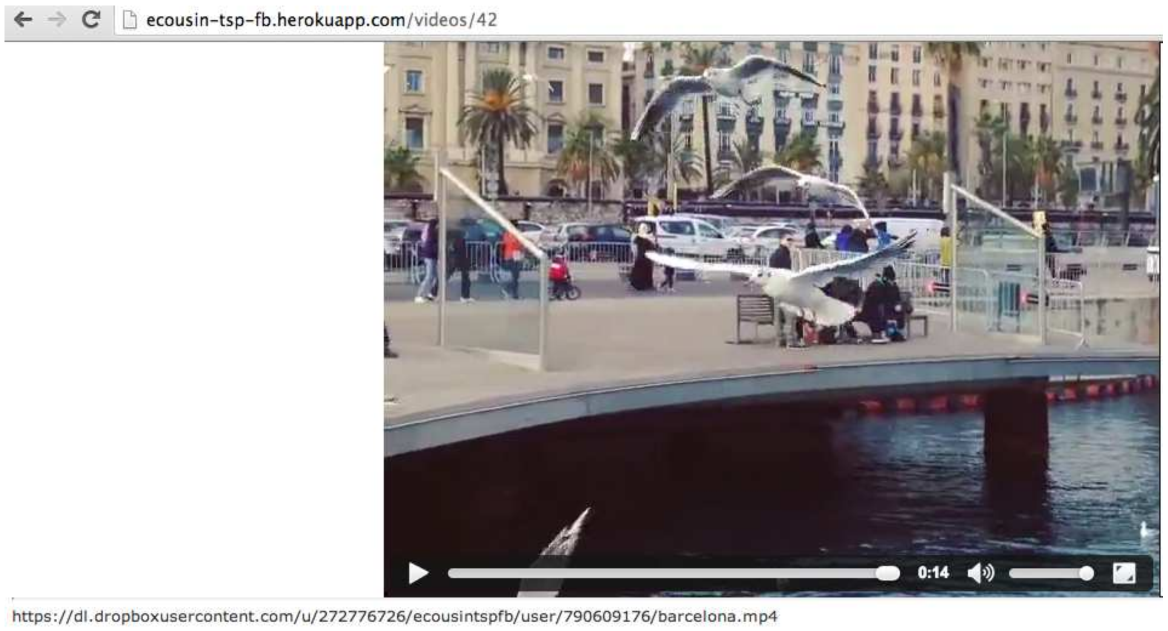

ecousin-tsp-fb.herokuapp.com/videos/42
0:14
https://dl.dropboxusercontent.com/u/272776726/ecousintspfb/user/790609176/barcelona.mp4

Figure 68. Video stream from local link

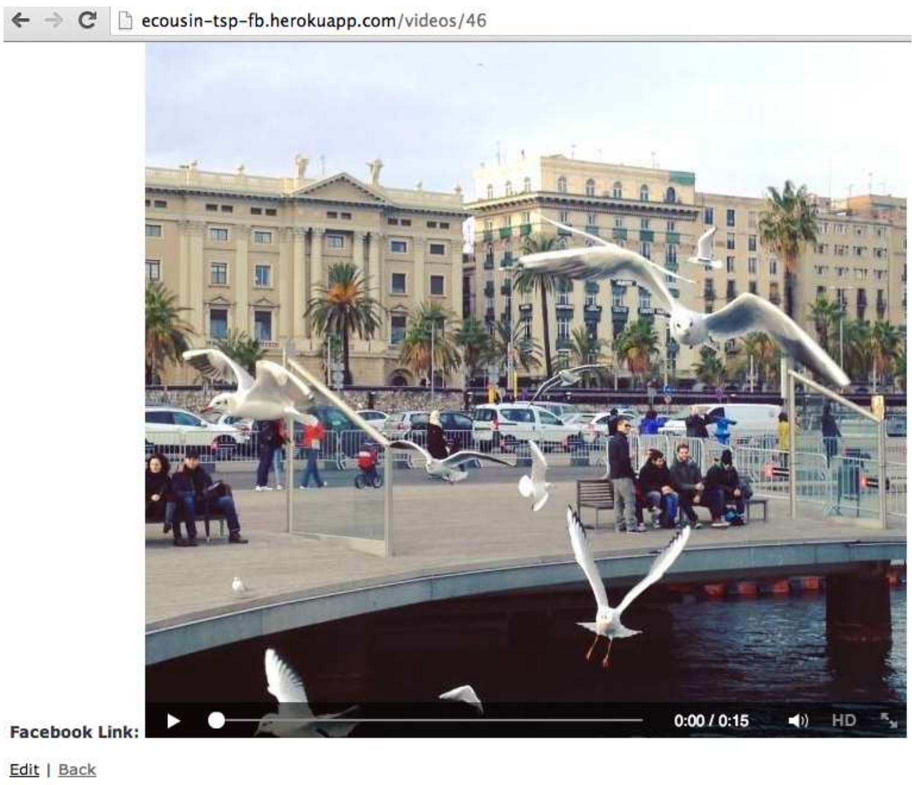


Figure 69. Video stream from Facebook link

2.3.4.2.2.2 Next steps

As it is mentioned before, the current version of the application is a beta version and several improvements have been planned for next steps as follows:

- Design a user friendly interface
- Adding the capability to download shared YouTube videos and add to the local server
- Capability to upload the user-generated videos directly and automatically to the local server
- Improve security features of the application and comply with the Facebook privacy levels for each video and user

3. CONCLUSIONS

The present deliverable D4.2 progresses towards the completion established for WP4. Towards this end, in this deliverable we have completed the design of the main components related to content dissemination, which is captured in the Content Dissemination Layer of the eCOUSIN architecture (introduced in D2.2 [D2.2]), This deliverable presents contributions to four main axis: caching, pre-fetching, resource allocation and peer-assisted content delivery.

We have provided first insights on what are the main parameters to be considered to decide which content has to be pre-fetched based on a prediction model that estimates which content is going to be downloaded by a given user and when, a real study based on a Facebook application that has collected behavior from real-end users, and resource availability estimation (for the case of mobile networks). Therefore, we have performed a thorough evaluation of pre-fetching approaches implementing an application, using real traces and running exhaustive simulations.

Furthermore, we have evaluated the performance that caching algorithms provides in the context of CDNs for two particular cases: Twitter content and Small Telco CDNs. In the former, we have run extensive simulations testing standard caching algorithms demonstrating that using social information improves the hit ratio. In the latter, we have used VoD real traces and have shown via simulation that the introduced dMLRU algorithm slightly improves the hit ratio achieved by the standard MLRU approach. In the context of CDNs network we have also presented a large measurement study that evaluates the performance (in terms of delay) that users distributed all over the world experiences to access Akamai nodes that are serving Facebook content.

Finally, we have presented a first version of a Peer-Assisted content delivery solution based on a Facebook application. This deliverable present the intial design and first implementation of this application that aims at offloading CDN and central application server to distribute low-popular videos that have been shared through OSNs.

REFERENCES

[AKA13] (2013) Akamai technologies. [Online]. Available: http://www.akamai.com/

[BEA10] D. Beaver, S. Kumar, H. C. Li, J. Sobel, P. Vajgel et al., “Finding a needle in haystack: Facebook’s photo storage,” Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI), 2010.

[CHA10] J. Chakareski and P. Frossard, “Context-adaptive information flow allocation and media delivery in online social networks,” IEEE J. Selected Topics in Signal Processing, vol. 4, no. 4, pp. 732–745, Aug. 2010.

[CHE09] X. Cheng and J. Liu, “Nettube: Exploring social networks for peer-to-peer short video sharing,” in Proc. conf. on Computer Communications (INFOCOM). Rio de Janeiro, Brazil: IEEE, Apr. 2009, pp. 1152–1160.

[CHE10] X. Cheng and J. Liu, “Tweeting videos: coordinate live streaming and storage sharing,” in Proc. Int’l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV). Amsterdam, The Netherlands: ACM, Jun. 2010, pp. 15–20.

[CHU03] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, “Planetlab: An Overlay Testbed for Broad-Coverage Services,” ACM SIGCOMM Computer Communication Review, vol. 33, no. 3, pp. 3–12, 2003.

[D2.1] EU eCOUSIN: Public Deliverable D2.1 - Initial report on uses cases and requirements. May 2013.

[D2.2] EU eCOUSIN: Public Deliverable D2.2 - Initial system architecture specification, July 2013.

[D3.1] EU eCOUSIN: Public deliverable D3.1 - Measurement, Modelling & Prediction of Social-Content Interdependencies (First Version), November 2013.

[D4.1] EU eCOUSIN: Public deliverable D4.1 – Preliminary Report on the Design of Technical Solutions on Content Placement and Delivery, November 2013.

[D5.1] EU eCOUSIN: Public deliverable D5.2 – Requirements for Social-Enhanced Content Centric and Mobile Network Infrastructures, November 2013.

[D5.2] EU eCOUSIN: Public deliverable D5.2 - Modules and Interfaces for Social-Enhanced Content Centric and Mobile Network Infrastructures (V1), April 2014.

[D6.1] EU eCOUSIN: Public Deliverable D6.1 - Preliminary Plan for System Integration and Assessment, January 2014.

[FLI] http://www.cs.sfu.ca/~sja25/personal/datasets/

[GKSR13] Gross, C., Kaup, F., Stingl, D., Richerzhagen, B., Hausheer, D., Steinmetz, R, 2013. EnerSim : An energy consumption model for large-scale overlay simulators. In: IEEE LCN.

[HAN13] X. Han, et. al., “Community Similarity Degree: Finding Similarity to Improve Recommendations in On-line Social Networks”, Social Network Analysis and Mining journal, in press.

[KUL10] V. Kulkarni and M. Devetsikiotis, “Communication timescales, structure and popularity: Using social network metrics for Youtube-like multimedia content distribution,” in Proc. Int’l Conf. Communications. Cape Town, South Africa: IEEE, May 2010.

[LIN10] K. C.-J. Lin, et. al., “Socionet: A social-based multimedia access system for unstructured P2P networks,” IEEE Trans. Parallel and Distributed Systems, vol. 21, no. 7, pp. 1027–1041, Jul. 2010.

[MAX13] (2013) Maxmind GeoIP database. [Online]. Available: http://www.maxmind.com/

[MICH13] Michelinakis, Foivos. Practical challenges of network optimized stored video delivery. Diss. Universidad Carlos III de Madrid, 2013.

[OCE11] FP7 OCEAN Deliverable D4.2 – Highly Dynamic and Distributed Caching, 2011.

[ORE06] A. Orebaugh, G. Ramirez, and J. Beale, Wireshark & Ethereal Network Protocol Analyzer Toolkit. Syngress, 2006.

[POU08] J. A. Pouwelse, et. al., “Tribler: A social-based peer-to-peer system,” Concurrency and Computation: Practice & Experience, vol. 20, no. 2, pp. 127–138, Feb. 2008.

[THR14] Timo Thräm, “Mobile Prefetching Strategies for Improved Video Access in Online Social Networks”, March 2013, Bachelor’s Thesis, Supervisor: Julius Rückert, Peer-to-Peer Systems Engineering, Technische Universität Darmstadt, Germany.

[VECI13] Z. Lu and G. de Veciana, “Optimizing stored video delivery for mobile networks: The value of knowing the future,” in Proceedings IEEE INFOCOM, 2013, pp. 2706–2714.

[VLE13] D. De Vleeschauwer, C. Hawinkel, Y. Le Louédec, “Determining Leaders and Clusters in Video Consumption”, Proceedings of the Joint SmartenIT/eCOUSIN Workshop on Social-aware Economic Traffic Management for Overlay and Cloud Applications (SETM13), Zurich (Switzerland), October 18, 2013.

[WIT10] M. P. Wittie, V. Pejovic, L. Deek, K. C. Almeroth, and B. Y. Zhao, “Exploiting Locality of Interest in Online Social Networks,” in International Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2010.

[WON12] W. Wongyai and L. Charoenwatana, “Examining the Network Traffic of Facebook Homepage Retrieval: An End User Perspective,” in International Joint Conference on Computer Science and Software Engineering (JCSSE), 2012.

ACRONYMS

API	Application Programming Interface
CBR	Constant Bit Rate
CDN	Content Delivery Network
CPR	Correct Prediction Ratio
CUTV	Catch-Up TeleVision
CQI	Channel Quality Indicators
dMLRU	Differentiated MLRU
eCOUSIN	enhanced COntent distribUtion with Social INformation
HR	Hit Ratio
IP	Internet Protocol
ISP	Internet Service Provider
LFU	Least Frequently Used
LRF	Less Reused Filter
LRU	Least Recently Used
LTE	Long Term Evolution
MLRU	Modified LRU
MCS	Modulation and Coding Scheme
OSN	Online Social Network
OTT	Over the Top
P2P	Peer-to-Peer
PaaS	Platform as a Service
QoE	Quality of Experience
RBG	Resource Block Group
RSRP	Reference Signal Received Power
RSRQ	Reference Signal Received Quality
SACDN	Social-Aware CDN
SINR	Signal to Interference and Noise Ratio
SVD	Singular Value Decomposition
TB	Transport Block
TCP	Transport Control Protocol
TELCO	Telecommunication Company
TP	Timely Prediction
UGC	User Generated Content

UDP	User Datagram Protocol
UE	User Equipment
UP	Useless Prediction
URL	Uniform Resource Locator
VOD	Video On Demand
WiFi	Wireless Fidelity
WP	Work Package