MEDIEVAL
INFSO-ICT-258053

MEDIEVAL

Deliverable D5.3

Advanced CDN mechanisms for video streaming

Editor:	Gerald Kunzmann, DOCOMO
Deliverable nature:	Public
Due date:	June 30th , 2012
Delivery date:	June 30th , 2012
Version:	2.0
Total number of pages:	83
Reviewed by:	Telemaco Melia, ALBLF; Dirk Staehle, DOCOMO
Keywords:	MEDIEVAL, system architecture, content distribution network, transport optimization

Abstract

This deliverable has been provided into two different versions. The first internal version, due on mid of January 2012 mainly focused on scientific work performed in the period June 2011 - December 2011. The final version of the same document, due on end of June 2012, extends the former and contains the final CDN architecture. Its main focus is on specification and this includes the description of the CDN components and interfaces which also take part in the final demonstrator. Therefore, the specification work was performed in strict relation with integration and experimentation activities.

List of authors

Company	Author
DOCOMO	Gerald Kunzmann
ALBLF	Sabine Randriamasy
IMDEA Networks	Joerg Widmer
CFR	Daniele Munaretto

History

Modified by	Date	Version	Comments
DOCOMO	15/01/2012	1.0	Interim version (internal)
DOCOMO	30/06/2012	2.0	Final version (public)

Executive summary

The purpose of deliverable D5.3 is to provide updates on the status of Task 5.2, presenting the ongoing research activities and key results since the delivery of deliverable D5.1. One of its main contributions is to provide detailed module and interface specifications for video transport optimisation and the integration of the CDN component in the MEDIEVAL system. The key contributions of Task 5.2 are described, along with the associated scientific results (both in publications and standardization fora). Moreover, the latest and ongoing implementation and evaluation efforts are presented.

This deliverable is introducing our concept of integrating Content Delivery Networks (CDN) techniques to mobile networks in order to optimize the transport of the video inside the mobile core network towards the Point of Access (PoA). With video streaming to mobile devices becoming more and more popular and growth rates of 100% per year, the traffic is already reaching the capacity limitations of the mobile core networks and, thus, operators are urgently looking for means of reducing the load and costs in their networks The CDN component is one crucial design choice which can significantly offload traffic from the operator's mobile core network.

Current caching solutions for mobile networks consist of one central cache in the mobile core network (e.g. Akamai [1]), which is not suitable for offloading the traffic in the core network itself. Also, Over-The-Top CDNs are not solving this problem. We are introducing a distributed CDN inside the mobile operator's network. Thereby, video streaming will be optimized with a specific focus on the selection of cache locations and peers based on metrics like network distance, availability, and content popularity. The goal is to provide video-aware transmission options to the users and the video service source. The users can choose the option with the best fit for the specific application, e.g., delay-sensitive interactive video versus Video on Demand streaming. The network then connects the user to the source, appropriately chosen with the support of a QoE aware request routing optimization.

Table of Contents

List of Figures

List of Tables

Abbreviations

ALTO	Application-Layer Traffic Optimization
API	Application Programming Interface
AM	Application Monitoring (module)
ARQ	Automatic Repeat request
BS	Base Station
CDN	Content Delivery Network (component / module)
CDNNC	CDN Node Control (module)
CM	Connection Manager (Wireless Access subsystem)
CNM	Core Network Monitoring (module)
CRC	Connection Relay and Cache
CS	Connection Specification
DM	Decision Module
DVB	Digital Video Broadcasting
E2E	End-to-End
EPSR	End Point Selection and Ranking
eMBMS	Evolved Multimedia Broadcast Multicast Services
EP	End Point (of a communication link)
FEC	Forward Error Correction
FM	Flow manager (Mobility subsystem)
HARQ	Hybrid ARQ
QoE	Quality of Experience
QoS	Quality of Service
L25	L2.5 abstraction module (Wireless Access subsystem)
LTE	Long Term Evolution
MAC	Media Access Control
MAR	Mobile Access Router
MEDIEVAL	MultimEDia transport for mobIlE Video AppLications
MN	Mobile Node
MSC	Message Sequence Chart
NEGOCODE	Network Guided Optimization of Content DElivery
OTT	Over-the-Top
P2P	Peer-to-Peer
PHY	Physical layer (of the OSI model)
PIM-SSM	Protocol Independent Multicast - Source-Specific Multicast
PoA	Point of Attachment

QoEVC	QoE & Video Control module (Video Services subsystem)
RR(O)	Request Routing (Optimizer)
SME2E	Session Management & E2E monitoring module (Video Services subsystem)
SVC	Scalable Video Codec
TE	Traffic Engineering (module)
TEEPOT	Traffic Engineered Endpoint Optimization Tool
TO	Transport Optimization (subsystem / component)
UE	User equipment
VoD	Video on Demand
XLO	Cross-layer optimization (module)

1 Introduction

The current Internet, and in particular the mobile Internet, was not designed with video requirements in mind and, as a consequence, its architecture is very inefficient for handling video traffic. Enhancements are needed to cater for improved Quality of Experience (QoE), improved reliability and efficient transport optimization in a mobile network.

The purpose of this deliverable is to introduce a component for a mobile CDN network in the context of the MEDIEVAL system, integrating CDN functionalities within the mobile network architecture. The CDN component is a crucial design choice which enables large scale content distribution at a reasonable cost and without overloading the operator's core network. Caching video data close to the users is able to partly compensate the significant increase of video traffic in mobile networks, which is reported to double every year, thus having the highest growth rate of any application category [1]. Besides, in-network caches reduce delay between server (i.e., the cache) and client for cached content.

Deploying CDN nodes inside mobile networks is in line with the plans of major network operators, content and CDN providers, and hardware manufacturers. Akamai, for example, in its Investor Summit [1], is showing its vision to extend the edge of their CDN network to the "Tower" by placing Akamai software on S-GWs and cell towers in cooperation with Ericsson.

In MEDIEVAL, video streaming is optimized with a specific focus on the selection of cache locations and peer selection based on, e.g., network distance, availability, and content popularity. The goal is to provide video-aware transmission options to the users and the video service source. The users can choose the option with the best fit for the specific application, e.g., delay-sensitive interactive video versus video-on-demand streaming. The network then connects the user to the source, appropriately chosen with the support of a QoE aware request routing optimization.

Although the amount of available content shows enormous increase, some files are more popular than others due to 'people primarily wanting to see what others have been seen before, or what others like'. Also recommendations, video ratings, and links from external sites lead to a distortion of the popularity distribution. Moreover, when splitting video files into chunks, the first minutes of the video will be watched more frequently, as some kind of preview, and some people will not watch the whole video due to several reasons (e.g. they don't like the video, wrong content, bad quality, etc.).

In this deliverable, we address the following topics:

- Why should data also be cached inside an operator's network in addition to existing Over-The-Top (OTT) content distribution networks (CDNs)?
- Which content should be cached? The most popular content as it is requested most often? However, if content is of varying sizes, one might think it is be better to store two less popular small objects than only one more popular large object (knapsack problem). This problem is related to content placement and is addressed in the decision.
- Where to store content? Several different design choices exist, from a single central CDN node towards several smaller distributed CDN nodes. This question is related to service placement. In the evaluation, we show the trade-off between the benefits and costs of in-network caching.
- How to manage and coordinate the CDN nodes? Should they act independent of each other, or can collaboration among them improve the system? Is collaboration among the CDN nodes feasible in nowadays mobile networks that tend to have a hierarchical structure? This deliverable introduces a decision manager and a CDN node controller to manage the CDN network.
- Which CDN node should be selected to serve a user request? With respect to which objectives? How can these be combined to both meet the user's QoE needs and optimize the network resource usage? Our approach is to extend the basic content/request routing with cross-layer information and QoE awareness. A multi-objective optimization algorithm is then selecting the most appropriate CDN node and transport path.

- How can the network infrastructure information needed by the advanced CDN node selection be made available to applications? Is there a way to do it which is safe for the network? Is there a standardized way to do it? The NEGOCODE module optimizing the basic request routing function is supported by the ALTO protocol and its Multi-Cost ALTO extension (see Chapter 3) that addresses this issue.
- How to best react to user mobility? We exchange information with the mobility subsystem to optimize handover decisions.
- How to integrate the above CDN functionalities into the MEDIEVAL system? We introduce module and interface specifications of the CDN component in Chapter 4, which is also the main focus of this deliverable.

The structure of the deliverable is organized as follows: Chapter 2 gives an overview on the key contributions, and Chapter 3 complements the scientific work achieved in task T5.1. The specification of the modules and interfaces of the CDN component and evaluation results of the CDN component are presented in Chapter 4 and Chapter 5, respectively.

2 Key contributions

In this chapter, we summarize the key scientific results and dissemination activities performed within Task 5.3 over the past year. The research results are presented in more detail in Chapter 3.

- The most important contribution was to combine the different pieces of CDN work and complete the design of the CDN part of the MEDIEVAL architecture. Chapter 4 describes its different components and functionalities, as well as the internal and external interfaces. The CDN components are placed around the decision module, which is the intelligence of the CDN system. It manages the content distribution among the CDN nodes and operates the NEGOCODE functions, which manage the endpoint selection when users roam across wireless networks. Thus, user requests can be routed to the most appropriate CDN node according to different parameters pre-defined by the network operator. NEGOCODE also offers relay functionality, which allows hiding the internal network topology of the operator from third parties such as end users or application clients.
- In a joint work with partners working on the transport optimisation component (Task 5.2, deliverable D5.2 [35]), we developed a cross-layer cost model for QoE optimization with network layer awareness on a hybrid wireless network. This cost model integrates physical and QoE metrics to evaluate the impact of content location and, in addition, derives costs from both the end user and the network provider. The architecture has been published in [41]. We further investigate how video applications can be served through several video delivery paths, resulting in different video quality experienced by the end user. We also consider network cost for realistic deployments. This optimization work was submitted to IEEE Globecom 2012 [46].
- Taking into account the activities covered by other work packages, we integrated the CDN technology with the mobility architecture and the session management mechanism. The most appropriate endpoint will already be selected at the session setup. The cooperation with the mobility subsystem helps switching to a better suited endpoint when users move through the network. The interface description among the different subsystems has been detailed within the prototyping work done in WP6 (see D6.3 [36]) and specified in D1.3 [29].
- As basis for the evaluation of the CDN component, we performed a survey on video usage. Therefore, we gathered usage data from various sources, e.g. ComScore in order to find out the properties of popular video portals like YouTube. This survey includes data like the number of unique viewers (i.e. users), viewing sessions, number of available videos, or the average video duration. The gathered data will be used to predict the future video usage on mobile devices and will be an important input parameter for both analytical evaluation and simulation of the CDN subsystem and its components.
- Regarding the analytical approach, we defined a cost model for the evaluation and analysis of the costs for different CDN realisations within mobile core network architecture. Using the cost model we give recommendations to mobile operators on the dimensioning of the CDN nodes in their specific network. Also, this cost model is used to estimate whether the CDN capacity should be extended in order to save costs, or not.
- Moreover, we developed a framework for multi-layer video streaming using inter-layer network coding. Layered streaming allows to cater to receivers with heterogeneous receive requirements. In this context, network coding helps to save capacity. As the overall problem is NP-complete, we propose a heuristic algorithm for inter-layer network coding problem without decoding at interior nodes. Rate allocation and code assignment decisions are based on the Edmonds-Karp maximum flow algorithm.
- An important result concerning the implementation and prototyping work was a survey of existing open source CDN solutions and selection of one candidate system. This is the basis for our own CDN implementation and prototyping work, as well as further development of the CDN components. A set of metrics has been defined and applied in the selection. The final choice was for the OpenCDN implementation from the InfoCom department of the University of Roma[1].

[1] http://labtel.ing.uniroma1.it/opencdn/

- As part of the dissemination activities we published three papers and moved forward the Multi Cost ALTO protocol extensions presented at the last IETF 82 in November 2011.

3 Scientific work

The purpose of this chapter is to provide an overview of the updates in terms of scientific work over Deliverable D5.1 [34] and to summarize the dissemination of research results. Within the MEDIEVAL project [27], many novel architectural designs, solutions and ideas have been produced, most of which have been presented in terms of contributions to standard definitions and scientific publications. In particular, MEDIEVAL's output can be found in the archives of main standardization bodies, as IETF and IEEE, in the form of standards or drafts, and also in conference proceedings and journal articles.

3.1 Work addressed in this deliverable

The list below represents the contributions produced within MEDIEVAL WP5 after June 2011 with respect to the CDN research topics addressed within this deliverable.

- **Further work on the decision module**: Definition and design of detailed inter- and intra-subsystem communication, together with final specification of its modules and interfaces with the other sub-systems. This work is based on validation, prototyping and simulation, as well as on the internal refinement work performed inside of each of the subsystems.
- **Path selection algorithm**: An optimisation algorithm for the selection of the best available video path was designed and evaluated in [46]. The details of the optimisation procedure, running in the XLO module are given in Section 3.3 of Deliverable D5.2 [35], while the simulation results are provided in Section 5.3 of the same deliverable. The selection of the best path is requested via the DM to XLO interface. In addition, we foresee the possibility (the final decision is left to the mobile operator) of placing a simplified version of such algorithm in the CDN subsystem of WP5, whenever the only Core Network metrics are required to select a new CDN video source (no need for wireless access metrics opposite to the original algorithm in D5.2, running in the XLO module). In this way, we offer to the mobile operator further degrees of freedom for selecting the video paths, according to his current needs and to the application requirements.
- **Design of CDN storage and data distribution mechanisms** between CDN nodes, extension of multicast streaming and CDN robustness algorithms.
    - The design of CDN storage mechanisms is related to the ongoing implementation based on OpenCDN and will be detailed further in the final deliverable.
    - With respect to multicast streaming, an algorithm for multicasting with inter-layer network coding has been designed (see [45]). The algorithm builds a distribution graph that intends to maximize the number of layers received at the different multicast receivers under some fairness constraint.
- Further specification of the **algorithms to select the endpoints** and adapt to the evolution of their performance. Specification and distribution of the associated functions in the TO architecture and in the network infrastructure. Discovery of the Connection Relays associated to a UE and specification of their interaction with ALTO Relays and other entities.
    - Results are IETF contributions described in Section 3.6.1 on IETF WG ALTO.
- Definition of **utility functions taking into account PHY parameters and QoE-rate relationship** to implement the cross-layer optimization algorithm. Special focus is given to ARQ/FEC implications in such an optimizer and eventually to multicast implications (possibly jointly with the Wireless Access subsystem).
    - The resulting algorithm has been designed through joint work between WP5 tasks 5.1 and 5.2. The work is summarized in this deliverable by a section entitled "QoE-based optimisation with network layer awareness on hybrid wireless network" of this deliverable and D5.2. It is detailed in D5.2.

  - A framework paper on this work is introduced in section “Publications”, see [41].

- Design of the **CDN mechanisms for optimal placement and management of CDN nodes**, the optimal content location, and optimal node selection based on the specific layout of the mobile operator’s core network.

3.2 Standardization plans/work

- IETF: Extension of the ALTO protocol to the specifics of mobile core networks and the mobility of users, promotion of the provider-confidential ALTO concept based on ALTO Relays and Connection Relaying.

3.3 QoE-based optimisation with network layer awareness

In a mobile network, a given video application can be served through different paths, resulting in different QoE, measured at the end user. These measurements can be fed-back to the network, allowing the mobile operator to manage the service i) to keep a target QoE and ii) to optimize the network resource usage. Applications are usually agnostic to the network conditions whereas the latter have a high impact in particular in wireless networks. The IETF Working Group ALTO (Application Layer Traffic Optimisation) is designing the ALTO client-server protocol that publishes an abstraction of network infrastructure information to overlay applications. The ALTO protocol is thus a standardized way to provide the applications with network infrastructure and policy related information. It is therefore gaining traction in use cases such as CDN and data centre operation.

In the CDN MEDIEVAL use case, network conditions and policy are integrated in the choice of the optimal CDN Node from which to get the requested content and also referred to as an End Point (EP). Indeed, the request routing function is optimized by a multi-objective EP selection function that uses information acquired by an ALTO Client. So the ALTO Client and EP optimization function are co-located in the network, with the request routing function.

The ALTO protocol can get information including routing cost, hop count, path and Cache load. The values for these metrics are then injected in a multi-objective optimization algorithm that outputs a reliable selection of CDN Nodes from which to deliver the content to the requested terminal. Such a selection is an optimization added to the basic request routing function that outputs a first list of eligible CDN Nodes.

This algorithm is generalized to address QoE-based optimisation with network layer awareness on hybrid wireless networks and integrate wireless network specific information. This generalization is described in D5.2 [35]. Indeed, the diversification of wireless access technologies allows addressing application requirements in different ways. This is particularly true when a mobile user can access the same content on the Internet via both cellular and wireless local area networks and from different EPs. A user might perceive a different QoE over different paths and should have the possibility to choose the best available QoE w.r.t. jointly the applicable EPs and associated paths. In this view the mobile operator needs a tool to provide the applications with the network costs associated to the different possible paths and EPs and guide the applications to better optimize network resources.

Starting from current trends in standardization, we are working on two key objectives: quasi-real time evaluation of the perceived QoE by end users and application requirements in terms of network resource usage. The online QoE computation algorithms combined with application layer optimisations give a promising solution. To this end we propose to exploit the ALTO protocol and its extensions for the mobile core and to evaluate the gathered key metrics (QoE-based video metrics and network metrics) in simulation and live test experiments.

More details about the original design of the algorithm are given in deliverable D5.2 (Sections 3.3 and 5.3) [35].

3.4 Management of CDN nodes

A range of considerations are of importance for the management of CDN nodes. First, the operation of the CDN nodes must be managed and controlled, in order to ensure a reliable operation of the system. Therefore, the Decision Module (DM) and the CDN Node Control (CDNNC) need to be closely coordinated. While the DM is responsible for content placement with respect to resource requests (content popularity, etc.), the CDNNC is responsible for CDN maintenance and may need to relocate content for this purpose.

This includes selecting an appropriate replacement strategy for the caches, in order to achieve a high hit rate. Also, the provided storage capacity has a direct influence on the hit rates. Another design choice is the dissemination strategy, i.e. whether to actively retrieve content from external sources or storing it "on the fly". The DM also decides on which storage location should be selected as the optimal source for transmitting the video to the user (request routing).

3.4.1 Replacement strategy

Given the limited size of storage at each CDN node, less frequently/recently used content should be replaced by new content in order to increase the hit rate of the cache. Thereby, the "hit rate" describes how often a searched item is actually found in the cache. Several replacement policies exist, including

- Random: replace a random content in the cache
- First-in, first-out (FIFO): the simplest replacement strategy is a FIFO queue with the most recent used content at the back and the earliest content in the front. If requested content is not in the cache, it is added to the back of the list and the content at the front of the cache (the oldest) is discarded. While this strategy is simple and intuitive, it performs poorly in practical applications.
- Least Recently Used (LRU): it discards the least recently used items first. In contrast to FIFO, when content is requested and already cached, the content will be moved to the back of the list. Thus, frequently used data will remain in the caches, whereas rare content will be dropped from the list, when a new content is requested which is not yet in the list.
- Least Frequently Used (LFU): it discards the least frequently used item in the cache. Therefore, a timestamp or counter must be added to each item in the cache, in order to monitor the last request for all items.
- Popularity-Based Caching (PBC): observe the popularity (frequency of requests) of all objects requested in a certain time interval and store the most popular items in the cache. If an item which is not in the cache is observed to be more popular than the least popular item in the cache, the item in the cache is replaced by the more popular item.

The replacement strategies mentioned above are sorted by complexity and state that needs to be maintained in order to make the replacement decision. In different areas different requirements to the replacement strategy exist. In CPU or main memory caches, small data objects are stored and speed of the caching algorithm is crucial. This, for example, is the reason why in the LFU strategy timestamps might already make up too much overhead and thus only a few bits are used as counter monitoring the content frequency. Storing the popularity of a huge number of items would not be possible. In contrast to that, in MEDIEVAL "large" video files/chunks are stored in the cache, enabling the usage of PBC.

PBC has one main benefit compared to LFU caching: If the number of existing items is much larger than the number of items that can be stored in the cache, with LFU even popular content will be flushed out of the cache by the large number of requested items, as the frequency of even the most popular content will be very small and thus will not be enough to keep the content in the comparatively small cache. In addition, the long-tail popularity distribution of video content (see also Section 5.2.1) amplifies this problem. Monitoring the popularity of the items and applying PBC provides a better usage of the cache and thus increases the hit rate and efficiency of the cache.

3.4.2 Cache size

Looking at the popularity distribution of video content (see also Section 5.2.1), we can see that 10-20% of the popular videos account for 80% of the total views. The other way round, 80% of the traffic could be saved, if 10-20% of the videos were stored in the CDN nodes. Yet, just looking at YouTube statistics, 10% of YouTube videos still correspond to 200 million videos. Assuming an average video length of 4:12 min and an average streaming rate of 0.5 Mbit/s, this would require a cache of around 3050 TB. Proving a smaller cache would not achieve the required traffic savings, as the hit rate of the cache would be much lower. As a consequence, "distributing" a central cache to multiple places in the network is negatively impacting the cache efficiency. Thus, when deploying decentralized caches instead of a central one, the size of each cache has to be dimensioned as big as a central cache.

3.4.3 Interconnection of CDNs

A mobile CDN may be connected to both a content provider CDN network and/or another mobile operator's mobile CDN. Several benefits of such a cooperation or interconnection exist, like improved quality of experience (QoE) for the end user, increasing the content availability, or dealing with flash crowds. In this area we suggest to follow the recommendations of the IETF CDNI (CDN Interconnection) WG[2], suggesting standardized control and request routing interfaces.

[2] https://datatracker.ietf.org/wg/cdni/

3.5 NEGOCODE functions and ALTO extensions

NEGOCODE adds an optimization to the Basic Request Routing (RR) function of a CDN. RR functions are usually based on proprietary methods, but published material shows that some of them involve estimation of the state and topology of network below the Autonomous system level, mostly in terms of delay which is estimated via RTT measurements. However, generating a delay-based estimation of such data is time-consuming and not always reliable as it is highly dynamic and influenced by network operators traffic management policies. Besides, other network status information such as path bandwidth is hard to measure from the overlay and the routing cost policy implemented by the different network operators is usually not available to overlay applications.

As part of the DM operations, the NEGOCODE function manages the endpoint (EP) selection when users roam across wireless (heterogeneous) networks. In fact during the lifetime of a download (especially for live content) the user can potentially connect to several networks thus being able to download content from more optimized sources.

The NEGOCODE addresses the issue of selecting the best EP from where to retrieve content based on user requirements as well as mobile operator requirements. From the user perspective the key point is the perceived QoE; from the mobile operator perspective the key point is the minimization of the costs per bit routed to the final end user. To this end the NEGOCODE provides the following functionalities (see Figure 1):

- A number of Traffic Engineered EndPoint Optimization Tools (TEEPOT) performing in particular the following operations:
    - TEEPOT1 for input to QoS/QoE ranking: getting the QoE specifications from the Application client, the type of QoE and QoS information needed to select the Endpoints, and getting the actual information via the ALTO client and the core network monitoring module (CNM),
    - TEEPOT2 for EP evaluation, sorting and/or ranking: running the optimization algorithm for EP selection, sorting, ranking with the information and possibly constraints of criteria weighting sent by the TEEPOT1 and sends the EP selection to the application client.
- ALTO client: requesting from the ALTO server abstracted information on the CDN nodes and their transport network level properties, that will be used as the QoE metrics handed to the TEEPOT1.
- ALTO server: hosting and delivering information on the transport network infrastructure underlying the CDN. Updates on this information include updates of the CDN Node status provided by the CDN node control (CDNNC) functions to the DM.

Figure 1 illustrates how the different components of NEGOCODE interact among themselves and with the CDN application client and core network monitoring module. The sequence of operations explained in the next paragraph, shows that NEGOCODE is an optimization that is added to the Request Routing function. It can be deactivated and used as a simple ALTO information provider to the TEEPOT2-like multi-criteria optimization algorithm using physical metrics such as SINR that is running in the XLO.

3.5.1 Proposed Multi-Cost ALTO protocol extension

The ALTO client and the ALTO server operate according to standard ALTO specifications. A protocol extension called Multi-Cost ALTO, see [40] is being proposed to IETF ALTO WG. Its purpose is twofold:

- Gain time and resources by transporting information on N Cost Types in 1 ALTO request/response transaction rather than needing N transactions as it is the case in the current protocol,
- Enrich the set of cost types, currently restricted to « static » costs such as «routingcost» and «hopcount», in particular costs reflecting the resources usage of path and application endpoints such as CDN caches. Example additional costs are: (i) “EP memory”: the memory size of an Endpoint, (ii) “endpointoccupationcost” and “pathoccupationcost” defined below.

3.5.2 Proposed additional ALTO Cost Types

The last Multi-Cost draft, see [40], proposes to add Path Occupation Cost ('pathoccupationcost') and Endpoint occupation cost ('endpointoccupationcost'). They must be available in the «numerical» mode if requested so by the ALTO Client. This is applicable to ALTO values provided at a short time scale (e.g. some minutes), or in the opposite, provided as timeless cost.

3.5.3 Typical use cases for Multi-Cost ALTO transaction

The Multi-Cost ALTO extension is an optimization, so it provides advantages in any case when the frequency of cost values updates are compatible. This is the case for "routingcost" and "hopcount", provided that "routingcost" represents monetary costs that are unusually considered as static. Otherwise appropriate protocol features will be proposed to handle appropriate updates.

3.5.4 ALTO Costs used in MEDIEVAL

The following ALTO and Multi-Cost ALTO costs report static or dynamic information on current status of Endpoints (EP), i.e., the CDN nodes in MEDIEVAL:

- 'routingcost': the mandatory ALTO cost, a Network Service Provider defined routing cost to an EP,
- 'endpointmemory' (EP-mem): to report on CDN Node memory capacity
- 'endpointoccupationcost', (EPOC): to report on CDN Node memory occupancy or load,
- 'hopcount' (HC): ALTO cost measured in IP hops between the user and the CDN Node, to report on distance or delay if no latency indication is available,
- 'pathoccupationcost' (POC): to report Path occupancy or load.

3.5.5 Traffic Engineered EndPoint Optimization Tool (TEEPOT)

The optimization algorithm (TEEPOT) is currently being developed in conjunction with D5.2. D5.2 proposes a hybrid approach to take into account both wireless parameters related to the established connection between a mobile device and the access network and EP parameters. The vectors composed by merging the set of parameters are sorted to maximize the utility function (QoE) or minimize the costs related to the selected EP. While this approach has been firstly designed for online and quasi real time QoE evaluation at the mobile node side we argue it could be adapted to the dynamic nature of the DM and be applied also to the network side. At a first glance the algorithm will need to be tuned for network operations including the gathering of the necessary parameters. In case gathering of the parameters is not possible different vectors will be evaluated.

Figure 1 illustrates how the different components of NEGOCODE interact, in an example where the application needs three connections: one with best possible delay in good radio conditions and two with fair radio conditions, moderate cost and fair EP memory capacity. It shows the sequence of operations after step (0) of optional NEGOCODE triggering by the RR function.

Generally the DM will need to provide a general purpose API to enable NEGOCODE to gather all the information necessary to optimize the selection of the CDN nodes from which to get content. An API between the DM and the CDN Node Control enables to get status information on CDN nodes and is detailed in Section 4.2

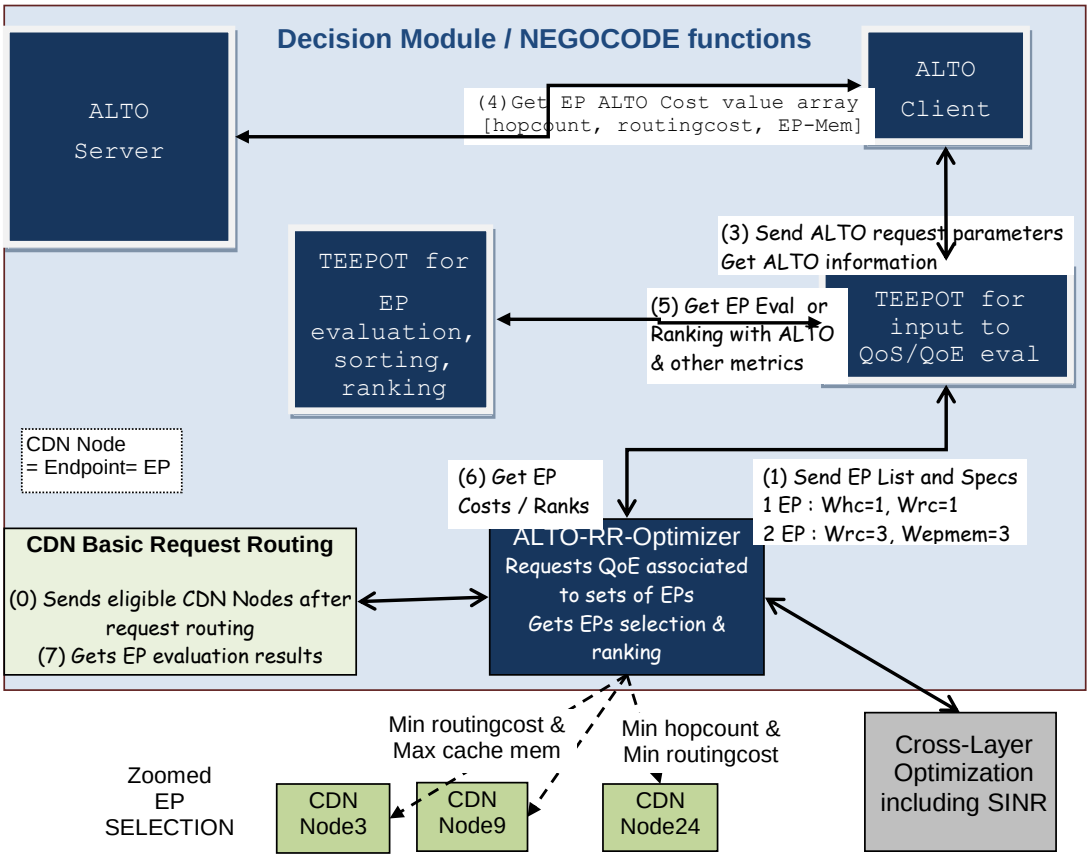


Figure 1 - Transactions between components (ALTO Client + Server, TEEPOTs) of the NEGOCODE block

It shows how multiple costs for multiple sets on connections are requested and processed. NEGOCODE components are boxes in blue.

3.6 Dissemination plan and results

This section gives an overview of the dissemination results and future plans for the work done over the past year since June 2011. For each contribution we highlight its relation with MEDIEVAL and also refer to key achievements discussed in Chapter 2.

3.6.1 IETF WG ALTO

ALBLF is editing and pushing a personal draft entitled "Multi-Cost ALTO", see [5], proposing an optimization to the ALTO core protocol by gathering several cost types in one single transaction. The first part of this draft presents the necessary extensions to the ALTO protocol and the second part proposes new Cost Types, in particular "dynamic" cost types that can be represented with several values in order to reflect the variations of a cost value in time or in space.

"Multi-Cost ALTO" and its future associate draft on Dynamic Costs are the ALTO feature that support and underlie the WP5 work on "Cross-layer cost model for QoE optimization", that is detailed in the MEDIEVAL deliverable "D 5.2" and whose scientific framework has been published in [41]. It supports the provision of QoE information related to an eligible content location.

All versions of "Multi-Cost ALTO" have been presented and discussed in the ALTO WG sessions and have drawn growing interest. Its last version see [40], has been presented at the IETF 83 ALTO WG session in March 2012. The move of this draft from "personal" to "ietf" will be considered and will probably happen once the base ALTO protocol will have passed the IESG review. People promoting the use of ALTO in the CDNI (CDN Interconnection) protocol have expressed their support to this draft.

The items proposed in "Multi-Cost ALTO" have gained traction and are largely reflected in a draft initiating discussions on extensions of both the ALTO protocol and the ALTO use case. This discussion took place at IETF 83 in the form of a non WG forming BoF. Last, "Multi-Cost ALTO" gained traction and was cited in the CDNI (CDN Interconnection) WG as it proposed a richer set of QoE cost types and is finer grained as BGP.

Given the intensity of IETF work related to "Multi-Cost ALTO", no progress has been made in the meantime on "Provider Confidential ALTO", see [4]. As a reminder, this draft proposes protocol extensions of the ALTO protocol to redirect ALTO responses to ALTO Relays, in order to keep network operator information confidential.

As for the IETF ALTO plans in the next 6 months, the effort will focus on the adoption of "Multi-Cost ALTO" as a protocol extension. The decision will be taken after the completion of the ALTO base protocol review process by the Internet Engineering Steering Group (IESG), the IETF authority responsible for technical management of IETF activities and the Internet standards process.

3.6.2 Publications

The list of published articles related to WP5 over the past year for the second work-year of MEDIEVAL is the following:

1. Title: "**QoE-based Transport Optimization for Video Delivery over Next Generation Cellular Networks**"

   Authors: N. Amram, B. Fu, G. Kunzmann, T. Melia, D. Munaretto, S. Randriamasy, B. Sayadi, J. Widmer, M. Zorzi

   Conference: IEEE MediaWiN workshop 2011

   The paper presents the MEDIEVAL's proposed architecture and lists the envisioned challenges that should be tackled. See reference [18].

2. Title: "**QoE optimisation with network layer awareness on hybrid wireless network**"

   Authors: T. Melia, S. Randriamasy, D. Munaretto, M. Zorzi.

   Conference: Next Generation Service Delivery Platforms (NG SDP), GI/ITG Workshop, October 2011, Munich, Germany.

   The paper presents the theoretical and practical framework to designing the QoE-based online optimisation algorithm that gathers application layer QoE and network resources awareness to produce a quasi-real-time performance evaluation to assist the choice of content locations. It also stresses the role of ALTO protocol to support the algorithm. See reference [41].

3. Title: "**Rate Allocation for Layered Multicast Streaming with Inter-Layer Network Coding**"

   Authors: Joerg Widmer, Andrea Capalbo, Antonio Fernández Anta, Albert Banchs.

   Conference: Proc. IEEE Infocom (mini conference track), Orlando, FL, July 2012

   Multi-layer video streaming allows to provide different video qualities to a group of multicast receivers with heterogeneous receive rates. The paper proposes a heuristic algorithm for inter-layer network coding without decoding at interior nodes. Rate allocation and code assignment are based on the Edmonds-Karp maximum flow algorithm. See reference [45].

4 Advanced CDN mechanisms for video streaming

The CDN component is used to:

- Provide a mobile CDN solution for video delivery.
- Dynamically maintain an optimal configuration of a set of servers for content distribution with respect to the current conditions of the entire system.
- Appropriately select content locations to save network resources, inter-domain traffic and delivery delay, i.e. where to store/cache content depending on the observed request patterns.
- Choose the most appropriate location that can serve the client request, e.g. based on proximity and CDN load, and route the client request to this CDN node.
- Coordinate with the Mobility subsystem to achieve handover optimization and with the Transport Optimisation module for QoE optimization.

The different building blocks of the CDN component and its interfaces are shown in Figure 2 and described in detail in the following subsections. Its central entity is the Decision Module (DM) coordinating content placement and request routing. Therefore, content popularity is monitored in the Application Monitoring module (AM). The DM also interfaces with the Video Services subsystem (mainly for session setup), the Mobility subsystem (for session setup and handover support), and the Cross-Layer Optimization module (XLO) to provide ALTO information to the XLO and optimize request routing. The actual operation of the CDN, i.e., node maintenance, load balancing, storing, moving, and deleting content, etc., is managed by the CDN Node Controller (CDNNC) via its interface with the CDN nodes.

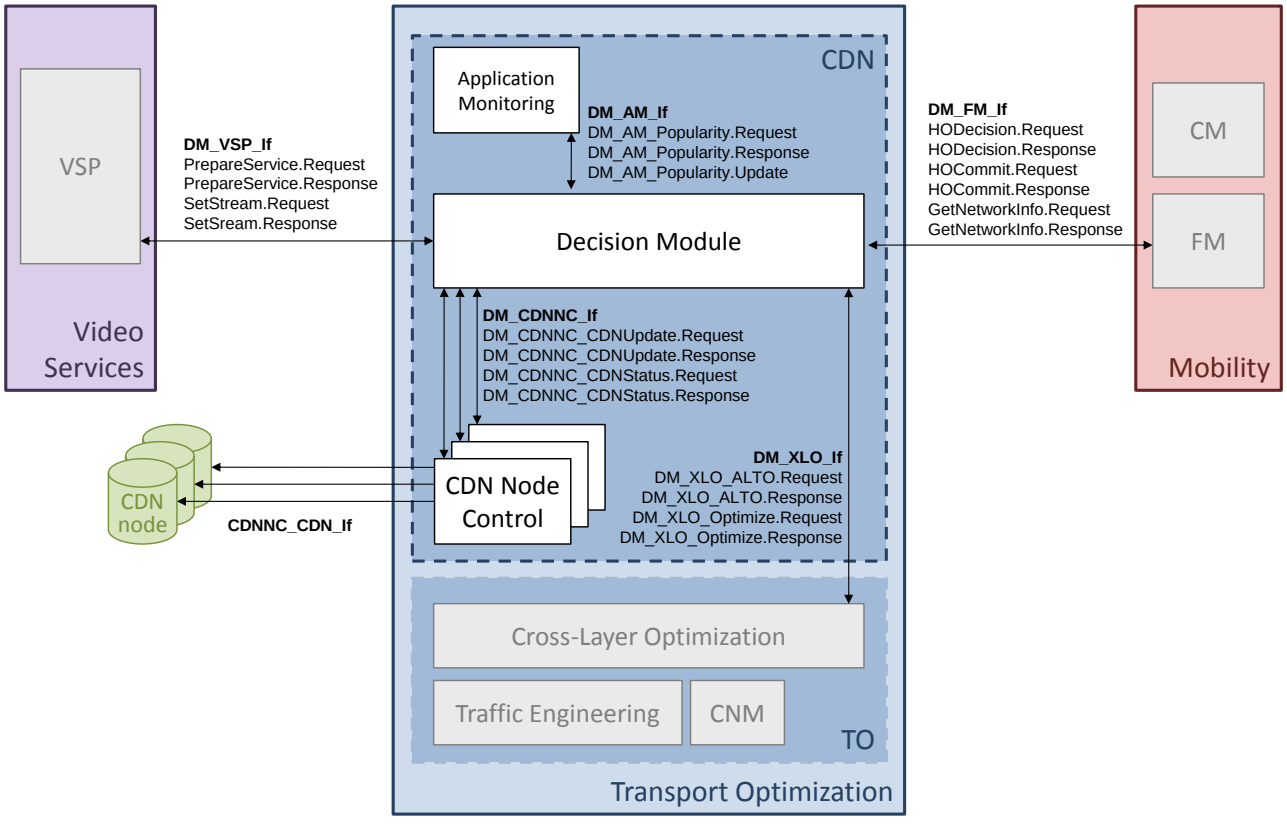


Figure 2 - Building blocks of the CDN component

The CDN infrastructure consists of the following components and functionalities [IETF RFC 3466, 3570]:

- A set of CDN nodes acting as relay servers to deliver stored content to the users.
- A content distribution mechanism that manages the CDN nodes and their content (e.g. load conditions, availability, popularity of content) and is also responsible for coordinating the distribution/replication of content from the origin server to the CDN nodes. Our design choice is a centralized approach located in the Decision Module.
- An advanced request routing solution, that redirects client requests to the most appropriate CDN node, based on different criteria (and policies) specified by the network operator. This request routing is supported by "Multi-Cost ALTO".
- Note that our CDN infrastructure is not concerned about accounting, as the main incentive for using CDNs inside the mobile core network is to save costs and network resources. An operator will likely not charge for the usage of the CDN itself, but just for the bandwidth used to stream the video. However, given the available usage monitoring, there are no major issues adding it to the operator's existing accounting system.

The following sections contain a detailed specification of all components and interfaces of the CDN component.

4.1 Modules

In this section we provide an overview of additions to and improvements over CDN related modules, namely the decision module (DM), the CDN Node Control module (CDNNC), and the application monitoring module (AM).

4.1.1 Decision module

The decision module (DM) is the main module of the CDN component. It decides when and where to store content in the CDN nodes, based on the popularity of the video files. It is part of the session initiation and handover preparations. Therefore, the decision module informs the mobile client on which source should be used for streaming/downloading the content (request routing), e.g. from either the (external) content provider or a cached copy from one of the CDN nodes.

Below is a list of functionalities provided by the DM:

- **Content routing** / request routing (RR): Appropriately select content locations and redirect user requests to the optimal candidate location. The objective of RR is to save network resources, inter-domain traffic and delivery delay, increase reliability, availability, and performance, and provide load balancing among a group of CDN nodes. This decision where to redirect a request is based on different metrics like content availability, CDN node load and health, and topology information (e.g. close nodes are preferred). Annex B.1 contains an extended list of identified metrics. The metrics related to the CDN Node are captured via the DM_CDNNC_CDNStatus interface, described in Annex A.3. The RR function outputs an initial set of one or more candidate caches. The RR algorithm can be very basic, such as returning the first or the whole set of caches containing the requested content. The DM contains a basic RR function that is optimized using NEGOCODE functionalities.

- **Operate NEGOCODE:**
  - RR optimization: NEGOCODE is an add-on of RR that takes in the initial set of caches output by the RR and performs optimizations on its elements such as evaluation and/or ranking. NEGOCODE returns the optimized set and associated information if requested to the calling basic RR. The returned information can be: the EP rank, the EP utility or cost, and the utility or cost for each component of the metric vector representing the EP; so, whether NEGOCODE is used or not with RR is transparent.

    The DM performs a basic RR consisting at least of identifying the caches hosting the required content. NEGOCODE is a standalone module that offers the possibility to output one or more candidate caches, depending on the use cases. In the planned implementation case, only 1 optimal cache is needed. The output that is provided by NEGOCODE is the result of the following functions that are applicable the CDN Node selection as well as P2P node selection:
    - **EP ranking**: sorts the input EPs by increasing rank value, where the smallest rank corresponds to the highest utility or the lowest cost,
    - **EP utility or cost**: computes the utility or cost of each input EP w.r.t. the metrics provided as application needs,
    - **EP metric evaluation**: provides the value acquired via the ALTO Server for each metric, as requested for example by the X-Layer optimization module.

Besides, NEGOCODE also acts as a relay functionality to hide internal network structure.

The NEGOCODE functions are explained in detail in the next section.

- **Content placement:** What content should be cached in which region?
  This decision is mainly based on
  - Content popularity, i.e. the request rate, in the different regions (both measured rates and predicted rates can be considered),
  - System configuration, i.e. the location of available CDN nodes,
- **Service placement:** Where to optimally place the CDN nodes in the given physical and logical network structure? Nowadays, operators still tend to manually design a fixed network structure with fixed service locations, but future (virtual) networks will be more flexible and operators will dynamically maintain an optimal configuration of a set of servers (for content distribution) with respect to the current conditions of the entire system. Thereby, also additional CDN resources can be provided if needed.
- **Resource management**: Can the request be served with the requested rate? Otherwise deny request or serve the mobile node with lower quality.
- Interaction with the Mobility subsystem to achieve **handover optimization** (see Section 4.3.2).

The Decision Module, as central module of the CDN component, has several interfaces with other internal and external modules. It is exchanging information with the Application Monitoring (AM) to monitor the top ranked content for the different regions of the network. Through an interface with the CDN Node Controller (CDNNC), the DM learns about the status of the CDN nodes and can instruct the CDNNC to store selected content. The interface to the Cross-Layer optimization module (XLO) is used to provide a list of available endpoints for a given session to the XLO. It is also used to trigger the XLO to perform an optimisation run and select the best video path based also on wireless metrics not available in the DM. An external interface with the Video Service Portal (VSP) is used for information exchange during session initiation. Thereby, another external interface with the Flow Manager (FM) is also involved to forward prepare service requests from the VSP to the Wireless Access via the Mobility subsystem. Finally, the DM to FM interface is required in the handover process to improve handovers due to node mobility.

The DM is a component of the MEDIEVAL core network. In the MEDIEVAL demonstrations it is realized as a single central component. Yet, as most of the decisions made in the DM are specific to a certain region of the network, in the core network of a large operator it is also feasible to implement the DM as a decentralized component, where its functionality is distributed among several nodes in the core network, e.g., as nodes attached to the MARs.

4.1.1.1 The NEGOCODE functions of the DM

The NEGOCODE block resides in the DM and is linked to the "basic routing request" (RR) function. It is an extension of the basic RR that optimizes it and can be activated and deactivated. The DM/NEGOCODE (NC for short) functions serve 2 purposes:

- Identify the best suited caches to get the content from. That means these functions are triggered once the basic RR functions has identified the list of eligible caches hosting the content requested for the user.
- Fetch the “cache network level” information on the CDN Nodes acquired from the CDN Node Control, such as cache load and cache capacity and put it in the ALTO server.

The only WP5 internal interfacing of DM/NC is with the CDNNC. It is indirectly related to ALTO as the CDNNC provides state information for the ALTO Server about the CDN Nodes such as "Endpoint memory"

(i.e. cache capacity) or "endpointoccupationcost" (cache load). To do this, the DM requests information from the CDNNC in order to put it in the ALTO server.

This interface called “DM_CDNNC_CDNStatus” is defined as a request by the DM to the CDNNC of CDN Node information that will then be used to feed the ALTO Server. The requests for such information are scheduled by the DM who also decides how to manage the ALTO information.

Last, this interface serves purposes for both the NC and other DM functions such as the RR and is therefore designed to allow requesting CDN node Content-List only, or ALTO information only, or both.

The term Endpoint (EP for short) will be used to designate CDN Nodes, as Endpoint is generic and is also the ALTO terminology for caches, peers, data centres, and other applications parties.

NEGOCODE functions and their interactions

The NEGOCODE functions are depicted as blue boxes in Figure 1. The NEGOCODE block includes 4 functions: ALTO-based Request Routing Optimizer, TEEPOT1, TEEPOT2, ALTO Client and ALTO Server. TEEPOT stands for Traffic Engineering Endpoint Optimization Tool.

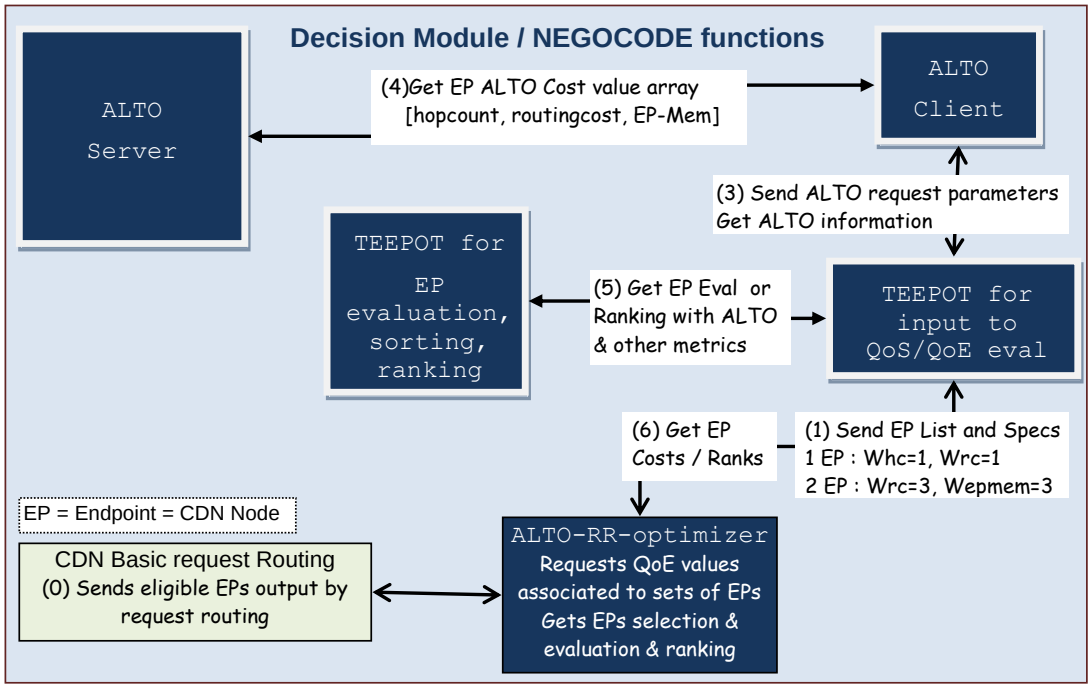


Figure 3 - The NEGOCODE functional block of the DM (components in blue) and their interaction

ALTO-RR Optimiser

is the short name for ALTO-based Request Routing Optimizer. Its role is to further optimize the selection of caches, called candidate endpoints in NC and containing a content Content_ID requested for the terminal MN_ID. When NEGOCODE is activated, ALTO-RRO may serve several requests: ranking the endpoints, evaluating the endpoints, providing values on endpoints for each metric of a given set. This set is defined by the DM, according the QoE needs of the user and possibly the network provider policy.

To this end, ALTO-RRO receives from the basic RR a request containing the input data which include:

- Request_ID abstracting Content_ID and MN_ID,
- the list of ID and IP address of the candidate EPs,
- the list of metrics against which to evaluate the EPs,

- the type of operations to be done that include: *EP_ranking, EP_eval,* and *EP_metric_eval*.

The result of *EP_metric_eval* is the value taken for each metric. This result is sent to the basic RR requesting it and it may also be sent to the XLO upon its request.

The basic RR holds thus information on which Endpoints are preferable to get content from, at which cost and can take "better that best effort" decisions.

TEEPOT1

- gets from the ALTO-RRO the input to EP list, requirements for QoE metrics and number of EPs to rank from the ALTO-enhanced-RR, via the DM_CDNNC_If
- sends back the list of EPs, with respective ranks, and cost value EP level or per component if requested. Cost metrics include QoE costs and other. The latter being specified by entities other than ALTO.
- prepares the encoding for ALTO requests and decoding of the ALTO responses
- caches the information conveyed in ALTO responses.
- sends input to TEEPOT2 the EP ranking function and gets the result

TEEPOT2

- performs the evaluation and ranking algorithm given the input: EP list, metric weights, size of ranked list(s).

ALTO Client

The ALTO Client generates a request for information to the ALTO Server when information on endpoint cost values is needed by TEEPOT1. The latter sends it the list of EPs, ALTO cost-types and properties for which values are needed.

The endpoints are the CDN Nodes hosting a requested content and an ALTO cost types is a metric w.r.t. which endpoint is evaluated. The request is sent to the ALTO server by the ALTO Client. This information is used to evaluate the performance of the candidate endpoints in order to decide which one is best suited to deliver the requested content to the requesting terminal. The ALTO values are provided for pairs of source and destination endpoints.

When the ALTO Client receives the ALTO response from the ALTO Server, it sends it to the TEEPOT1 function that will cache it and use it to either request and endpoint evaluation from TEEPOT2 or send it directly back to the ALTO-RRO.

ALTO Server

Upon receipt, the ALTO Server will retrieve the value for the cost between all pairs of source and destination endpoints encode the information in JSON format and send it to the ALTO Client via the HTTP protocol. It replies to the ALTO Client with a message that contains the ALTO value for the requested cost-types for pairs of source and destination endpoints. The ALTO information in the response is encoded in the JSON format and includes:

- Cost_type_List: List of cost metrics for which value is requested
- Endpoint_src_List : List of IP addresses and type of candidate CDN nodes
- Endpoint_destination_List : IP address and type of the MN(s) receiving the content
- Cost_type_value_List: for each source and destination pair: List of cost value for each cost type.

4.1.2 CDN Node Control (CDNNC)

The CDN node control module (CDNNC) manages all of the CDN node functions that directly affect the operation of the CDN (node maintenance, load balancing, storing, moving, and deleting content, etc.). To this end, it works closely with the decision module (DM) that decides which content to store where in the network. It is also responsible for maintaining CDN related status information such as the current load, (free) capacities, and information about stored content, and providing such information to the decision module.

The CDNNC receives commands from the decision module requesting it to store, move, replicate, or delete content, based on the changing popularity of content, the mobility of users or user groups, or congestion in certain parts of the core or access network that may require shifting flows and content to less congested parts of the network. A CDNNC may be responsible for a single CDN node or a group of CDN nodes, e.g. there is one CDNNC per region monitored in the AM. The CDNNC will also perform CDN internal decisions to move and replicate content between caches:

- Power off: move content to nearby CDN nodes to be able to power off a node for maintenance (when required) or to save energy, e.g. during night time
- Performance: replicate/delete CDN content within a group of CDN nodes to provide the required level of performance (trade off CDN performance and required storage capacity)
- Availability: replicate/delete CDN content to achieve the required level of resilience against failure (if needed) and to allow for seamless service continuation after a failover
- Recovery: rebuild the content of a failed CDN node(s) after node failure (on the same node, or on a new CDN node that is brought up for that purpose) to maintain the required level of availability and performance

The CDNNC acts as mediator between the DM and the actual CDN nodes. For this reason, it only has internal interfaces, one to the DM called DM_CDNNC_If for the exchange of control and status information, and one to the actual CDN Nodes, CDNNC_CDNnode_If, for the management and control of the CDN nodes.

Due to the close interaction between the DM and the CDNNC, for most practical purposes both modules will be co-located. As the CDNNC can manage and control a substantial number of CDN nodes, such a setup is suitable even for large deployments. It is however entirely possible to co-locate the CDNNC functionality directly with the CDN nodes, or as part of a hierarchy, where the DM interacts with a number of CDNNC modules (say of a certain region), each of which in turn controlling a number of CDN nodes.

4.1.3 Application monitoring (AM)

The application monitoring module is a database storing information about the popularity of videos. It is responsible to extract the X top rated content per CDN node according to the used popularity law. The key point behind the exploitation of the locality information is that the content consumption on a university campus is partially different from the set of contents consumed in a city mall.

4.2 Internal interfaces

The purpose of this section is to provide a high-level overview and description of all internal interfaces of the Transport Optimization subsystem, as depicted in Figure 2. A complete list of primitives is also reported for each interface. Details on the interfaces, their parameters and format, useful for the implementation phase, are documented in Annex A.

4.2.1 Interface DM_XLO_If

This interface, an API, is used by the XLO to request the list of End-Points (EPs) available in the core network from the DM. The DM provides the list of available EPs for a given session of a user to the XLO, which then runs the optimisation algorithm for selecting a video path in the network with better performance metrics, if available and necessary, using also the wireless metrics coming from the interface `L25_XLO_If` (Section 4.2.4 in Deliverable D5.2 [35]).

Moreover, this interface is used by the DM to request the XLO to perform an optimisation run (thus, using also wireless metrics) and send back the best video path available computed.

List of primitives:

DM_XLO_ ALTO

`XLO_DM_ALTO.Request (from XLO to DM)`

`XLO_DM_ALTO.Response(from DM to XLO)`

DM_XLO_ALTO

`DM_XLO_Optimize.Request (from DM to XLO)`

`DM_XLO_Optimize.Response(from XLO to DM)`

For details, please refer to Annex in D5.2 [34].

4.2.2 Interface DM_CDNNC_If

This interface, an API, is used for the DM to request and manage CDN Node related information from the CDN Node control. `DM_CDNNC_If` initiates CDN Node management operations requested by the DM such as updating the content stored in CDN Nodes. It also enables the DM to get information on a set of CDN Nodes, such as their content or operational state.

List of primitives

DM_CDNNC_CDNUpdate

`DM_CDNNC_CDNUpdate.Request (from DM to CDNNC)`

`DM_CDNNC_CDNUpdate.Response(from CDNNC to DM)`

DM_CDNNC_CDNStatus

`DM_CDNNC_CDNStatus.Request (from DM to CDNNC)`

`DM_CDNNC_CDNStatus.Response(from CDNNC to DM)`

For details, please refer to Annex A.2 of this document.

4.2.3 Interface CDNNC_CDNnode_If

The interface between CDNNC and CDN Node, called `CDNNC_CDNnode_If`, is used for all the low level CDN functionality related to the control and management of the CDN Nodes. Similar to the DM to CDNNC interface, there is a primitive to update content, i.e., to install and remove content from the CDN nodes. Status information is requested and provided via the `DM_CDNNC_CDNStatus` primitive. It is used to pass through status requests from the DM module, but also is used to request more detailed low level status information directly related to the management of the CDN. Furthermore, for maintenance purposes there is a `DM_CDNNC_PowerState` primitive that allows the CDNNC to power up and power down CDN nodes for maintenance reasons, or to balance energy consumption and performance.

List of primitives

CDNNC_CDNnode_CDNUpdate

```
CDNNC_CDNnode_CDNUpdate.Request (from DM to CDNNC)
CDNNC_CDNnode_CDNUpdate.Response(from CDNNC to DM)
```

CDNNC_CDNnode_CDNStatus

```
CDNNC_CDNnode_CDNStatus.Request (from DM to CDNNC)
CDNNC_CDNnode_CDNStatus.Response(from CDNNC to DM)
```

CDNNC_CDNnode_PowerState

```
CDNNC_CDNnode_PowerState.Request (from DM to CDNNC)
CDNNC_CDNnode_PowerState.Response(from CDNNC to DM)
```

For details, please refer to Annex A.3 of this document.

4.2.4 Interface DM_AM_If

This interface is used by the Decision Module (DM) to request content popularity information monitored by the Application Monitoring module (AM). The response contains a list of the Top X most popular content in a certain region to the DM. A Region_ID can be specified in order to get the popularity measurement for a specific region. The DM is periodically (in the order of tens of minutes) requesting an update of the content popularity to update the content placement in the CDN nodes. Upon the receipt of the response message, the DM triggers the CDN algorithm to determine whether the cached content in one of the CDN nodes should be updated. If yes, a `DM_CDNNC_CDNUpdate.Request` message is sent to these CDN nodes.

This interface is also used to update the popularity database at the AM. Therefore, a `DM_AM_Popularity.Update` message is sent from the DM to the AM. This message is generated each time a new video session is started. Upon receipt of this message, the AM updates its popularity database.

List of primitives

DM_AM_Popularity

```
DM_AM_Popularity.Request (from DM to AM)
DM_AM_Popularity.Response(from AM to DM)
DM_AM_Popularity.Update  (from DM to AM)
```

For details, please refer to Annex A.4 of this document.

4.3 External interfaces

4.3.1 Interface DM_VSP_If

This interface, an API between the video service portal (VSP) and the decision module (DM), is used during session setup to check for cached versions of the requested video file and to forward network information from the mobility subsystem to the video services subsystem.

When the user is looking for specific content through, e.g., the video portal in the video services subsystem, the VSP is establishing a handshake with the DM to identify the list of available content for the selected service. Then, the DM (through the NEGOCODE functions) is looking for the best matching cache (in case of VoD) to serve the selected content, and provides the VSP with the IP address and other information needed to retrieve the content.

In addition, network information from the mobility subsystem is requested from and forwarded to the video services subsystem via this interface.

A detailed description of this external interface is available in the final version of Deliverable D1.3 [29].

4.3.2 Interface DM_FM_If

This interface, an API, is mainly required in the handover process in order to improve handovers due to MN movement. In the HO process the FM provides the DM with a list of PoAs. The list is ordered by the path selection algorithm (running in the DM) taking into account the resources availability and IP flow requirements. The Transport Optimization subsystem weights each of the candidate PoAs (e.g. based on availability of "close" CDN nodes) and sends the updated list back to the FM. Based on this response, the FM decision algorithm selects the handover target network.

In addition, this interface is also use to consult the lower layers on the current status of the access network conditions and/or to setup the lower layers for new service (flows) activations.

`DM_FM_HODecision` primitives refer to the exchange of information during the HO process after the FM has collected the results of querying candidate MARs for available resources. After receiving this message the DM should apply its knowledge on CDNs and filter the PoAs to return the best candidate for handover. On reception of the response message the FM should indicate to the CM that the Candidate Discovery process is over.

`DM_FM_HOCommit` primitives are exchanged when the mobility protocol has finished its operation. The FM, running in the new serving MAR, informs the DM about current status of sessions and flows (if anchored to previous MAR or not). The DM should acknowledge the reception of this information.

`DM_FM_GetNetworkInfo` primitives are exchanged to consult the lower layers on the current status of the access network conditions. The message is generated by the DM, to ask the FM for the network conditions, after the DM received a request to prepare the network to provide a service. After receiving the response, the DM has knowledge of the access network and can select the best endpoint (original source or CDN node) for the session. It should also forward the access network information to the Video Service Portal.

`DM_FM_SetStream` primitives are used to setup the network's lower layers with the necessary conditions to receive a new flow, such as bandwidth allocation, QoS setup, etc…. After receiving this message the FM starts the necessary procedures to prepare the network to provide this new flow to the Mobile Terminal.

A detailed description of this external interface is available in the final version of Deliverable D1.3 [29].

4.4 Multicast support

The MEDIEVAL architecture supports IP multicast in the core and access networks. Most multicast related functionality is located at the Mobile Access Router (MAR) which acts as the rendezvous point and root of the multicast tree.

In the following we describe the layered multicast streaming architecture that uses network coding to save capacity. It is suitable whenever the multicast tree composition does not change drastically over time, for example for life broadcasts of large scale events, where most base stations subscribe to the multicast and hence user mobility does not lead to changes in the multicast subscription. It can also be used for multicast content delivery to the CDN nodes.

4.4.1 Layered Multicast Streaming

When multicasting to receivers with heterogeneous receive rates, layered coding allows to distribute the stream over several multicast flows such that the higher the number of flows (and thus the overall rate) a receiver obtains, the better the video quality at that receiver. Multicast with network coding provides capacity gains over plain multicast routing. In general, inter-layer coding outperforms intra-layer coding, but few practical heuristics for the inter-layer network coding problem exist. The two most prominent heuristics were presented in [42]. We propose an algorithm for inter-layer network coding problem that, in contrast to the algorithms proposed in [42] does not require decoding at interior nodes and does not propagate traffic along all the links of the network. Each receiver runs modified version of Edmonds-Karp max-flow algorithm to find up to a number of paths that correspond to the max-flow from to the source to the receiver. Paths have layer constraints so that information carried on those paths is not coded over more layers than can be decoded at downstream receivers. While paths are node disjoint on a per-receiver basis, paths of different receivers can overlap, as long as the layer constraints of the involved receivers are compatible. Network coding (but no decoding) is required at interior nodes where paths merge.

4.4.1.1 Network Coded Layered Multicast Algorithm

For the sake of exposition, consider the network as graph G(V,E) with nodes V and edges E. We denote the source node by $s \in V$, and the set of receivers by $T = \{t_1, t_2, \ldots, t_r\} \subset V$. Further, let In(v) be the set of incoming edges of node v and Out(v) be set of outgoing edges of node v. The source multicasts a stream of up to k layers, $L_1, \ldots, L_k$. Due to the properties of the layered coding, layer L_i is only useful to a receiver, if the receiver also receives layers $L_1, \ldots, L_{i-1}$. Let $y(u,v) = \sum_{j=1}^{i} g_j L_j$ denote the coded layer combination that is sent on edge (u,v) with coding coefficients g_j. We have that

$$y(u,v) \in \langle \cup_{(w,u)\in In(u)} y(w,u) \rangle \quad \forall u \in V \setminus T \setminus \{s\}$$

where <Y> denotes the linear subspace spanned by the set of vectors Y . For simplicity of exposition, we assume that edges have unit capacity and layers have unit rate, as in [40]. Since our algorithm is directly based on the Edmonds-Karp max-flow algorithm, it is straightforward to extend it to non-unitary edge capacities and non-unitary layer rates. The problem under consideration is to maximize the sum of the receive rates of all receivers, under a max-min fairness constraint. A given allocation is max-min fair if it is not possible to increase the number of layers received by any receiver t_i, without reducing the number of layers of another receiver t_j to less than that of receiver t_i. In a connected graph with unit capacity edges as in our model, this ensures that each receiver is at least able to receive the base layer L_1.

In our model, data of layer L_i is always combined with data from all lower layers, i.e., whenever the source needs to inject some random linear combination of layer L_i on an outgoing edge, it does so by sending a random linear combination of all layers L_j, $j \in [1, i]$.

Our heuristic first determines the max-flow from the source to each receiver using the Edmonds-Karp algorithm. It then processes the receivers in ascending order of their max-flow values. This ensures that capacity is first allocated to receivers with low max-flows before allocating capacity to higher max-flow receivers. The assignment is closely related to the notion of max-min fairness. For the assignment of layers to flows, we design a modified multi-layer version of the Edmonds-Karp maximum flow algorithm called MultiLayer-MaxFlow. The algorithm in turn uses a layer constrained version of breadth first search LayeredBFS. LayeredBFS tries to find one additional edge disjoint path with specific layer constraints for the current receiver. For simplicity we describe the algorithm as a centralized algorithm, but it's distributed implementation is straightforward.

LayerAssignment (Figure 4): The algorithm executes MultiLayer-MaxFlow for each receiver in ascending order of maxflows. MultiLayer-MaxFlow returns the set of the receiver's flow allocation F_i on edge disjoints paths, as well as the set of maximum layer constraints M_i that ensure decodability of the information received through these paths. With these, the global flow allocation F and maximum layer constraints M are updated (lines 11-12).

```
1  algorithm LayerAssignment(G, s, T)
2  ∀(u,v) ∈ E : F((u,v)) ← 0, M((u,v)) ← ∞
3  for each t_i ∈ T
4  |   maxFlow(t_i) ← EdmondsKarp(s, t_i)
5  end for
6  for m ← 1 to max_{t_i∈T} maxFlow(t_i)
7  |   for each t_i such that maxFlow(t_i) = m
8  |   |   (F_i, M_i) ← MultiLayer-MaxFlow(s, t_i)
9  |   end for
10 |   for each (u,v) ∈ E such that M_i((u,v)) < ∞
11 |   |   F((u,v)) ← max{F((u,v)), F_i((u,v))}
12 |   |   M((u,v)) ← min{M((u,v)), M_i((u,v))}
13 |   end for
14 end for
```

Figure 4 - Layer assignment algorithm

MultiLayer-MaxFlow (Figure 5): The algorithm tries to find paths from source s to receiver t that allow the receiver to decode the maximum number of layers ℓ_{max} , given the existing flow assignments (starting with ℓ_{max} = maxFlow(t) layers). This requires finding ℓ_{max} edge disjoint paths that deliver linear combinations of the first ℓ_{max} layers. To ensure linear independence, a receiver may receive at most one combination coded over the first layer, at most two combinations coded over the first two layers, etc. This implies the following minimum layer constraint for the linear combinations $y_1, \ldots, y_{\ell_{max}}$:

$$y_{\ell_{min}} = \sum_{j=1}^{\ell} g_j L_j, \ \ell_{min} = 1, \ldots, \ell_{max}, \ \ell \in [\ell_{min}, \ell_{max}]$$

with $\ell_{min} \leq l \leq \ell_{max}$. This equation holds, we say that the receiver satisfies ℓ_{max}-decodability. When assigning flows, our algorithm first searches for a flow with $\ell_{min} = \ell_{max}$, then for $\ell_{min} = \ell_{max} - 1$, and so on. If LayeredBFS fails to return a layer-l path P with $\ell_{min} \leq l \leq \ell_{max}$ (and thus found = FALSE), the receiver will not be be able to decode ℓ_{max} layers. In that case, MultiLayer-MaxFlow reduces ℓ_{max} by one and all existing temporary flows F_i and maximum layer constraints M_i are removed (lines 2-3).

```
1  procedure MultiLayer-MaxFlow(s,t)
2  for ℓ_max ← maxFlow(t) down to 1
3  |  ∀(u,v) ∈ E : F_i((u,v)) ← 0, M_i((u,v)) ← ∞
4  |  ℓ_min ← ℓ_max
5  |  do
6  |  |  (P, found, update) ← LayeredBFS(s,t,ℓ_min,ℓ_max)
7  |  |  if found then
8  |  |  |  u ← s
9  |  |  |  while u ≠ t
10 |  |  |  |  (v,ℓ) ← P(u)
11 |  |  |  |  if update(u) then
12 |  |  |  |  |  ∀(u',v') upstream of u that carry a flow
13 |  |  |  |  |  that contributes to (u,v) :
14 |  |  |  |  |  |  M_i((u',v')) ← min{M_i((u',v')), ℓ}
15 |  |  |  |  end if
16 |  |  |  |  if (u,v) ∈ Out(u) then
17 |  |  |  |  |  M_i((u,v)) ← ℓ
18 |  |  |  |  |  F_i((u,v)) ← 1
19 |  |  |  |  else // reverse edge
20 |  |  |  |  |  M_i((v,u)) ← ∞
21 |  |  |  |  |  F_i((v,u)) ← 0
22 |  |  |  |  end if
23 |  |  |  |  u ← v
24 |  |  |  end while
25 |  |  |  ℓ_min ← ℓ_min − 1
26 |  |  end if
27 |  while (ℓ_min ≥ 1) ∧ found
28 |  if found then
29 |  |  return (F_i, M_i)
30 |  end if
31 end for
```

Figure 5 - MultiLayer-MaxFlow algorithm

Whenever LayeredBFS returns with a valid path, ℓ_{min} is decremented and the path is backtracked from s to t. F' and M' are updated on all the edges of the path (lines 16-22). In addition, nodes maintain a in-out table that maps incoming to outgoing edges (for simplicity, this is not shown in the pseudo-code). A two-hop path segment (w,u),(u,v) creates an (w,u)→(u,v) at the local table of node u. (w,u) may map to multiple outgoing edges, and multiple incoming edges may map to (u,v). The algorithm also checks whether a special flag update(u) is set for a node u on the path. This indicates that the maximum layer constraint on edge (u,v) had to be reduced. As a consequence, the maximum layer constraints on all upstream edges that contribute to (u,v) have to be updated as well (lines 11-15). The edges can easily be identified through the in-out tables maintained at the nodes.

LayeredBFS (Figure 6): The algorithm performs breadth first search to find a path from t to s with matching layer constraints. It maintains a priority queue Q to store nodes to be visited next, as well as a mapping P:u → (v, l) to store next hop nodes for backtracking and the corresponding maximum layer constraint for edge (u,v).

The algorithms removes the first node from the queue and explores all forward edges that are not yet used by t (lines 13-28) and virtual reverse edges that are used by t (lines 29-37). Forward edges that are entirely unused are enqueued with a cost of δ(v) + 1, since capacity would have to be allocated on a new edge. Shared edges can only be traversed if they support a sufficiently high layer (line 18) and can be used at cost 0 if the current maximum layer constraint M((u,v)) can be kept (line 19). If, however, the constraint has to be reduced since ℓ < M((u,v)), this means that other receivers downstream of (u,v) may no longer be able to decode up to M((u,v)) but only up to ℓ layers (line 22). This is only allowed in case no other options exist. Only when $\ell = \ell_{max}$ < M((u,v)), i.e., the node is trying to obtain less layers than M((u,v)), is the corresponding node enqueued, and it is enqueued last with a high cost of δ (v)+|E|. The marker update is set to u to indicate that the maximum layer constraints on edges upstream of u may have to be updated by MultiLayer-MaxFlow.

As in the original Edmonds-Karp algorithm, whenever capacity is assigned to a flow on edge (u,v) ∈ E, a corresponding negative flow is assigned to a virtual reverse edge (v,u). In case the LayeredBFS algorithm finds a suitable path through such a reverse edge, the flow originally assigned on the forward edge (u,v) is removed, if (u,v) is only used by the current receiver. If however, edge (u,v) is a shared edge, the flow on that edge is required by another receiver and cannot be removed. The LayeredBFS may nevertheless traverse the reverse edge to find a suitable flow, which is then combined with the existing flows in the subtree below the shared edge. When traversing reverse edges that are only used by t (lines 31-32), the cost is set to δ(v) - 1 since capacity on an edge is freed, and the layer constraint is reset to ℓ_{max}. Traversing shared reverse edges leaves the cost δ(v) unchanged, and the maximum layer constraint is reset to that of the shared edge M((u,v)), since network coding occurs after the reverse edge.

```
1  procedure LayeredBFS(s, t, ℓ_min, ℓ_max)
2  ∀v ∈ V : P(v) ← (⊥, ∞), δ(v) ← ∞, update(v) ← FALSE
3  δ(t) ← 0;
4  Q ← ∅ // priority queue
5  enqueue(Q, (t, δ(t))) // enqueue t with highest priority 0
6  while Q ≠ ∅ do
7  |   v ← dequeue(Q)
8  |   if v = s then
9  |   |   return (P, TRUE, update)
10 |   end if
11 |   (·, ℓ) ← P(v)
12 |   ℓ ← min{ℓ, ℓ_max}
13 |   for each (u, v) ∈ In(v) such that
14 |   |           (P(u) = (⊥, ·)) ∧ (F_i((u, v)) = 0)
15 |   |   if F((u, v)) = 0 then
16 |   |   |   P(u) ← (v, ℓ); δ(u) ← δ(v) + 1
17 |   |   |   enqueue(Q, (u, δ(u)))
18 |   |   else if ℓ_min ≤ M((u, v))
19 |   |   |   if ℓ ≥ M((u, v)) then
20 |   |   |   |   P(u) ← (v, M((u, v))); δ(u) ← δ(v)
21 |   |   |   |   enqueue(Q, (u, δ(u)))
22 |   |   |   else if ℓ = ℓ_max then
23 |   |   |   |   P(u) ← (v, ℓ_max); δ(u) ← δ(v) + |E|
24 |   |   |   |   update(u) ← TRUE
25 |   |   |   |   enqueue(Q, (u, δ(u)))
26 |   |   |   end if
27 |   |   end if
28 |   end for
29 |   for each (v, u) ∈ Out(v) such that
30 |   |           (P(u) = (⊥, ·)) ∧ (F_i((v, u)) > 0)
31 |   |   if F((v, u)) = 0 then
32 |   |   |   P(u) ← (v, ℓ_max); δ(u) ← δ(v) − 1
33 |   |   else
34 |   |   |   P(u) ← (v, M((u, v))); δ(u) ← δ(v)
35 |   |   end if
36 |   |   enqueue(Q, (u, δ(u)))
37 |   end for
38 end while
39 return (∅, FALSE, update)
```

Figure 6 - Layered BFS algorithm

Code assignment: After the LayerAssignment algorithm completes, the source sends out linear combinations over as many layers as is allowed by the layer constraints on the outgoing edges.

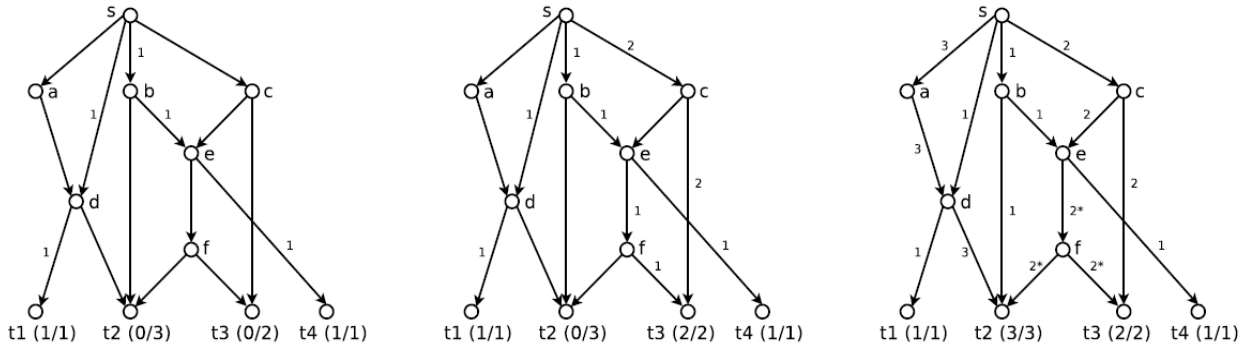
$$y(s,v) = \sum_{l=1}^{L((s,v))} g_l^{(s,v)} L_l, \quad \forall (s,v) \in Out(s)$$

The source further needs to ensure that the combinations it sends out over each edge are linearly independent. An interior node u of the network codes according to its in-out table.

$$y(u,v) = \sum_{w:(w,u)\mapsto(u,v)} g^{(w,u)} y(w,u)$$

4.4.1.2 Algorithm Example

As in the original Edmonds-Karp algorithm, we repeatedly perform a breadth-first search to find augmenting paths to the source. However, the LayeredBFS starts the search at the receivers and proceeds upstream to the source. This allows finding existing flows that can be reused, as explained later. In the example in Fig. 7(a), first receivers t_1 and t_4 with a max-flow of 1 find shortest paths to obtain L_1 directly from the source. The max-flow 2 receiver t_3 first needs to find a path to a flow coded over L_1 and L_2. If such a path is not found, the receiver would attempt to at least find a path to L_1. In the example in Fig. 7(b), a suitable L_2-path (s; c; t_3) is found. The receiver then uses LayeredBFS to find an edge-disjoint path carrying L_1 or a combination of L_1 and L_2. The algorithm may traverse edges that are already in use by other receivers at no cost, as long as the layer constraints on that path are not reduced. After traversing the two edges to reach node e, the algorithm traverses the remaining path to s before exploring any new edges that are unused as of yet. Backtracking from the source establishes path (s; b; e; f; t_3), where edges (s; b) and (b; e) are shared with receiver t_4 and new capacity only needs to be assigned to edges (e; f) and (f; t_3).



(a) Flow assignment: max-flow 1 receivers (b) Flow assignment: max-flow 2 receivers (c) Flow assignment: max-flow 3 receiver

Figure 7 – Different stages of the MultiLayer-MaxFlow algorithm: layer assignment per link and rate achieved vs. maximum flow per receiver

A shared edge can also be traversed to augment the existing flow on that edge, in case the flow is already received by the receiver via a different path, or the flow contains layers that are too low to be useful at this stage of the search. Such a shared edge traversal implies network coding across flows, since the existing flow on that edge was assigned by another receiver and thus cannot be replaced. In the example, network coding on a shared edge occurs when assigning paths to the max-flow 3 receiver t_2. The receiver first searches for a combination of L_1, L_2, and L_3 which it obtains directly from the source via a new path (s; a; d; t_2), as shown in Fig. 7(c). It then needs a combination of the first two or all three layers. It first explores the edges (b; t_2) and (f; t_2). Edge (s; b) cannot be traversed to find layer L_2 or above, since t_4 enforced that only L_1 can be transported. While L_1 is also transported on edge (e; f), the maximum layer constraint on that edge is 2, established by receiver t_3 which is able to decode L_1 and L_2 given its flow assignment. Therefore, the LayeredBFS can continue via the shared edge and subsequently finds a suitable combination of L_1 and L_2 at node c via edge (c; e). Node e thus forms a linear combination of L_1 received from b and the combination of L_1 and L_2 received from c and forwards it along edge (e; f). Note that while this alters the linear combination received on the subtree below edge (e; f), i.e., also receiver t_3 will now receive a linear combination over the first two layers rather than just layer 1, this does not change decodability at t_3.

In this example, the MultiLayer-MaxFlow algorithm achieves the max-flows for all receivers, whereas neither the Min-Req and Min-Cut algorithms nor the intra-layer approach manage to deliver all three layers to receiver t_2.

5 Status of the evaluation work

This section describes how the enhancements described in this deliverable are being evaluated. The three enhancement items are evaluated as follows:

- NEGOCODE: prototyping of the NEGOCODE block as running within the DM will show how network level information can be accessed by applications and enable a better than random CDN Node selection. Simulations as described in Section 5.3 of D5.2, will quantify the performance improvement in the XLO NEGOCODE use case, that is when the TEEPOT2 vector based algorithm is implemented in XLO, uses physical measurements in addition to the ALTO information provided via TEEPOT1.
- Mobile Operator CDN management: After analysing video characteristics, we introduce an evaluation model for content placement. Based on a typical mobile network design, we discuss different design variants for in-network caching: central, autonomous distributed, and collaborative distributed, compared to a traditional solution without CDN caches inside the mobile network. We show that providing in-network caches, the delay and the consumption of resource usage inside can significantly be reduced at the costs for operating additional CDN nodes.
- Network Coded Layered Multicast Algorithm: The performance of the network coded layered multicast algorithm is analysed for random wired network topologies. We compare against state-of-the-art approaches and show that our algorithm performs similar to algorithms that require higher complexity routers that allow decoding of network codes and significantly outperforms algorithms of the same complexity, while at the same time consuming much fewer network resources.

5.1 NEGOCODE evaluation

NEGOCODE adds an optimization to the Basic Request Routing function of a CDN. RR functions are usually based on proprietary methods, but published material shows that some of them involve estimation of network infrastructure state, mostly in terms of delay typically estimated via RTT measurements. However, fetching such data and generating an estimation of the delay is time and computation greedy and not always reliable. Besides other network status information such as path bandwidth and routing cost policy is impossible to get from the overlay.

NEGOCODE aims at improving the QoE of the Mobile users, by enhancing the selection of the EP from which the requested content is delivered. This enhancement takes into account QoE request associated to the user's needs, gets the related information from the CDN and transport network and involves it in a robust EP evaluation algorithm.

NEGOCODE provides two kinds of improvements to the RR function:

- Saving the processing time of estimating network state by providing direct access to such information, in an abstracted form, via the ALTO Service,
- Enriching the set of EP evaluation metrics with information restricted to transport network infrastructure such as routing cost policy and abstracted form of network state information, via the ALTO Service.
- Providing a robust EP evaluation and ranking through vector based multi-objective optimization.

The NEGOCODE prototype and API with request routing is currently under development. The evaluation will be qualitative, in the form of a prototype that will show:

- The difference between RR results with and without NEGOCODE, in terms of utility of the selected CDN Node.

A finer grained evaluation that measures the utility of the CDN Node selected when NEGOCODE uses a particular vector-based EP evaluation algorithm in TEEPOT2, and compared with a classical non vector based algorithm.

5.2 Mobile operator CDN management

5.2.1 Related work

5.2.1.1 Popularity distribution

YouTube is the world's largest user generated content (UGC) VoD system. "I Tube, You Tube" [38] presents trace-driven analysis of UGC video popularity distributions. The main results from this study are:

- YouTube seems to be highly skewed towards popular files, i.e. little content is extremely popular.
- The popularity distribution exhibits a power-law waist with a truncated tail.

The Pareto principle (also called 80-20 rule) states (translated into the MEDIEVAL context), that if k users have already watched a video, then the rate of other users watching the same video is proportional to k. This correlation results from several reasons and is also called the rich-get-richer principle. The more views a video has, the more people talk about it, and the more other people tend to watch the video. Automated recommendations, links in social networks and on websites, as well as searching for or sorting by the "most viewed" or "top-rated" videos.

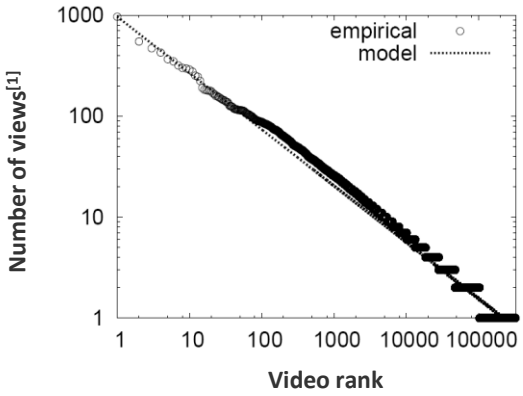


Figure 8 - You Tube Traffic Characterization

The HP Laboratories in Palo Alto performed a study on the traffic characterization of YouTube traffic [43]. Figure 8 shows the number of views of YouTube videos over the rank of the video (i.e. the most popular video is rank 1 and so on). The log-log plot of views versus frequency is a straight line, which is a distinguished feature of a Pareto distribution.

In this context, a fetch-at-most-once behaviour of users is observed [39], i.e. unlike the WWW traffic where a single user fetches a popular page (e.g. CNN) many times, video users fetch most objects once: since video content does not change (i.e. immutable) viewers are not likely to watch the same video multiple times.

Another observation is, that "young" videos can change many rank positions very fast (yet only a few; less than 1%), while "old" videos have a much smaller rank fluctuation, indicating a more stable ranking classification for old videos. This characteristic must also be considered when predicting video popularities and updating the content stored in the CDN nodes.

5.2.1.2 YouTube analysis

For the following analysis no data for a specific video service from one of the project partners is available, yet the popularity distribution will be similar to video services like YouTube. Thus, we use YouTube's popularity distribution in our analysis.

Therefore, we did a rough analysis of popular video distribution portals, mainly YouTube as the largest portal. Figure 6 shows the number of views and viewers for the US and YouTube, which had a share of 40-50% since 2005. According to ComScore[3], in August 2011, 180.4 million unique viewers accounted for 6.9 billion viewing sessions. Google Sites, driven primarily by video viewing on YouTube, ranked top with 162.1 million unique viewers and 3.5 billion views.

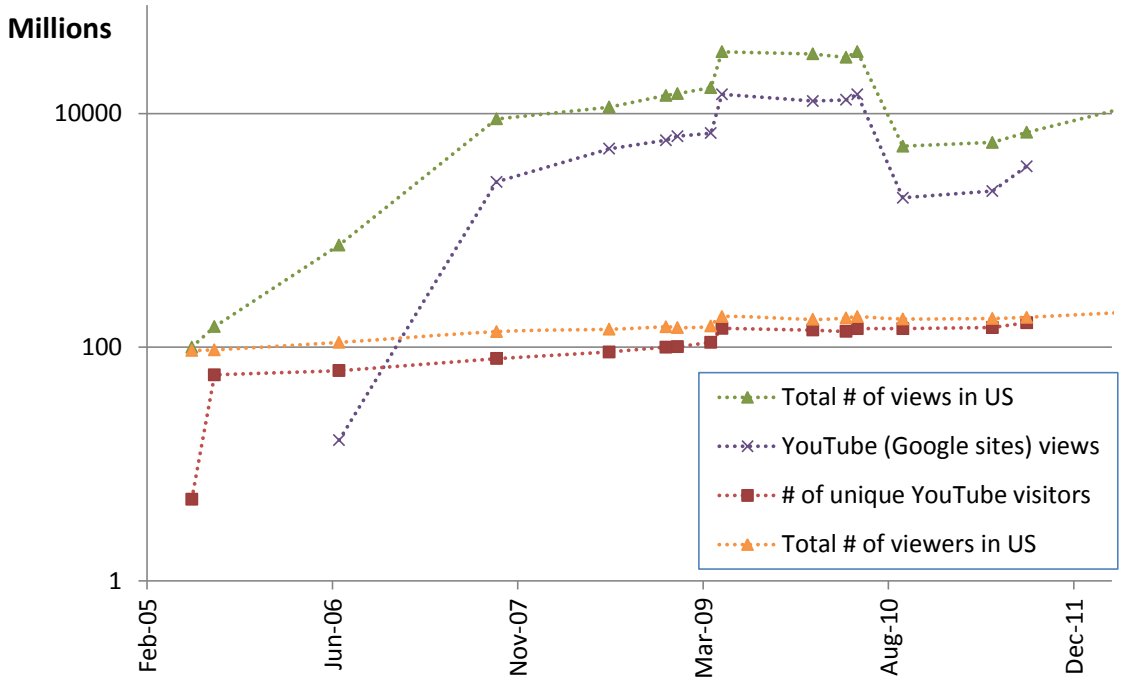


Figure 9 - Video viewers and viewing sessions in US

[3] Most data is based on ComScore press releases http://www.comscore.com/ger/Press_Events/Press_Releases/

The observed average video duration doubled from 2.7min in July 2008 to 5.3min in August 2011 (see Figure 10).

YouTube's bandwidth costs per day are estimated with $1,000,000 (as of 2009)[4]. The amount of video data uploaded to YouTube every minute is said to be 13 hours of video.

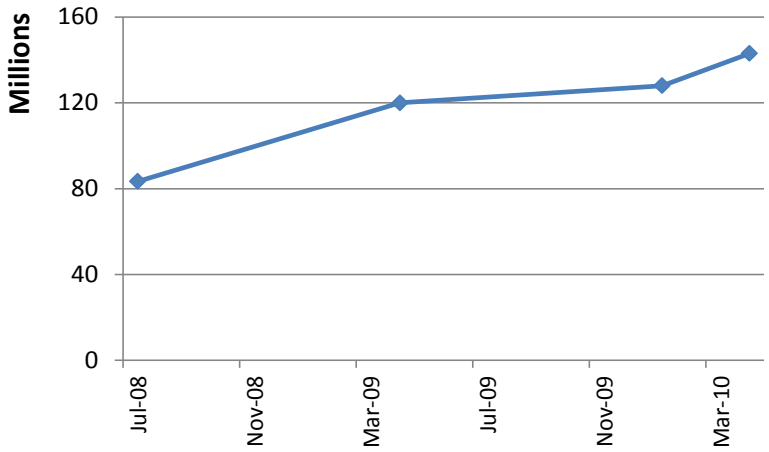


Figure 10 - Average video duration

5.2.1.3 Caching

By caching only 10% of the long-term popular videos, a cache can serve 80% of requests.

Thereby, the caching efficiency is affected by metrics like the cache size, the number of users and videos, the correlation of requests, the shifts in popularity, and so on.

About 40% of the videos that are requested daily are different from the long-term popular videos.

5.2.1.4 Network topology

Typically, a mobile operator's network topology is organized in a hierarchical structure, with a small core network (CR) that contains servers and databases for mobile services, like user registration, accounting, mail servers, and the home location register (HLR). Also, usually, there is one single ingress point for IP traffic from/to external service providers, where a firewall and other means of security control in- and outgoing traffic. This point is a bottleneck and thus must be able to handle high bandwidth.

Although there are some cross links between adjacent base stations, usually these links just provide a small data rate and are currently used for signalling purposes only.

[4] http://thefuturebuzz.com/2009/01/12/social-media-web-20-internet-numbers-stats/

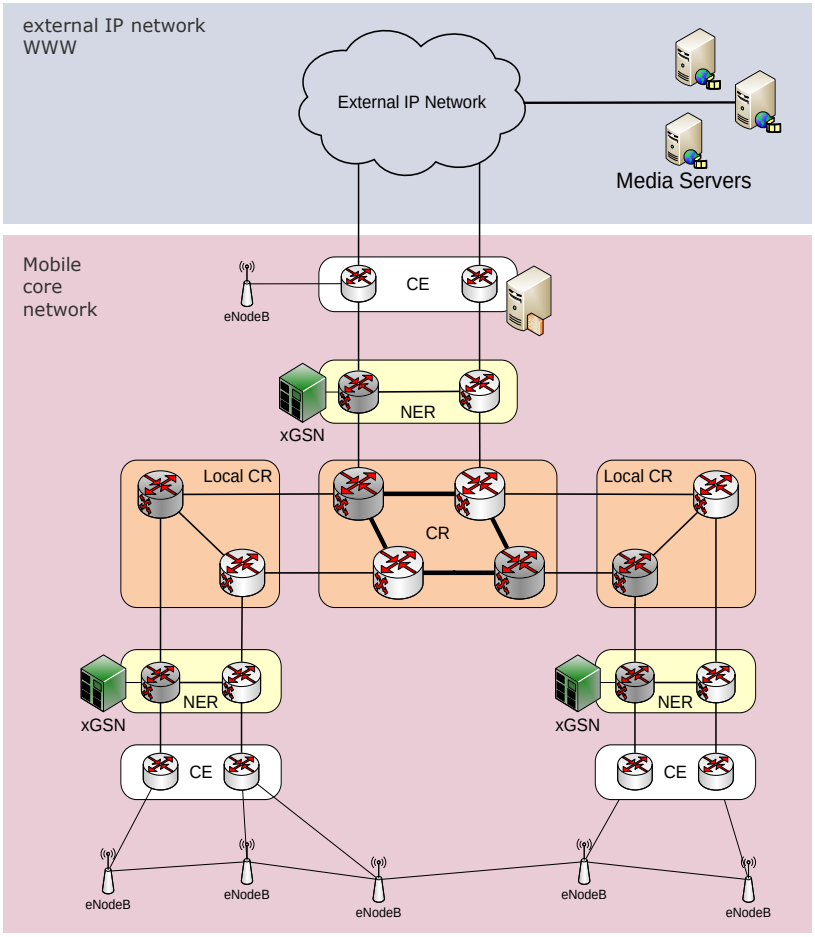


Figure 11 - Mobile operator network topology

5.2.2 Cost model

In MEDIEVAL we consider a typical Telco network organized in a hierarchical manner. The network is organized in *I* regions attached to a central core network (see Figure 11). We define a cost model following [37]. Thereby, each region *i*, *i* in {1, 2, … , *I*}, has caching capabilities of S_i bytes and clients requesting objects at an aggregated rate λ_i. The central cache in the core network has a storage capacity of S_0 bytes. There are *J* objects viewed by customers of the Telco company. Object *j*, *j* in {1, 2, …, J}, has a size of b_j bytes and clients in region *i* request the object with a request probability of $p_{i,j}$. We assume that the client request patterns are homogenous within one region. We define a matrix **x** = (x_{ij}), with

$$x_{ij} = \begin{cases} 1 & \text{object } j \text{ is cached in region } i \\ 0 & \text{otherwise.} \end{cases}$$

We denote the negation of x_{ij} by $x_{ij}^{-1} = \neg x_{ij}$.

The traffic in the network is monitored/measured at two specific links (indicated by the red triangles in Figure 11):

- I-C The ingress router situated at the border between the core network (CN) and external IP networks
- C-E The links between the core network (CN) and the regional network edge routers (NER)

Also, the following parameters are used in the cost model:

$c_{\text{I-C}}$	Traffic on link Internet $\rightarrow$ Core Router
$c_{\text{C-E},i}$	Traffic on link Core Router $\rightarrow$ NER of region i
c_s	Costs for providing the storage
c_t	Costs for transit traffic
l_1	Link costs for traffic between ingress point and CR
$l_{2,i}$	link costs between CR and CE of region i

The number of objects stored in the cache of region *i* is constrained by the storage capacity of the cache:

Constraint:

$$\sum_{j=1}^{J} b_j x_{ij} \leq S_i \qquad i = 1, \ldots, I$$

Then, we can define a target function minimizing the overall costs *C*(*S*):

$$\text{minimize } C(S) = l_1 \cdot c_{\text{I-C}} + \sum_{i=1}^{I} l_2 \cdot c_{\text{C-E},i} + \sum_{i=1}^{I} c_s \cdot S_i$$

5.2.2.1 Scenario 1: without caching

This scenario shows the current situation: a core network without any caches (see Figure 12). All videos requested from the users are directly streamed from the media servers in the external network. Thus, all video and signalling traffic passes both the link I-C and one of the links C-E.

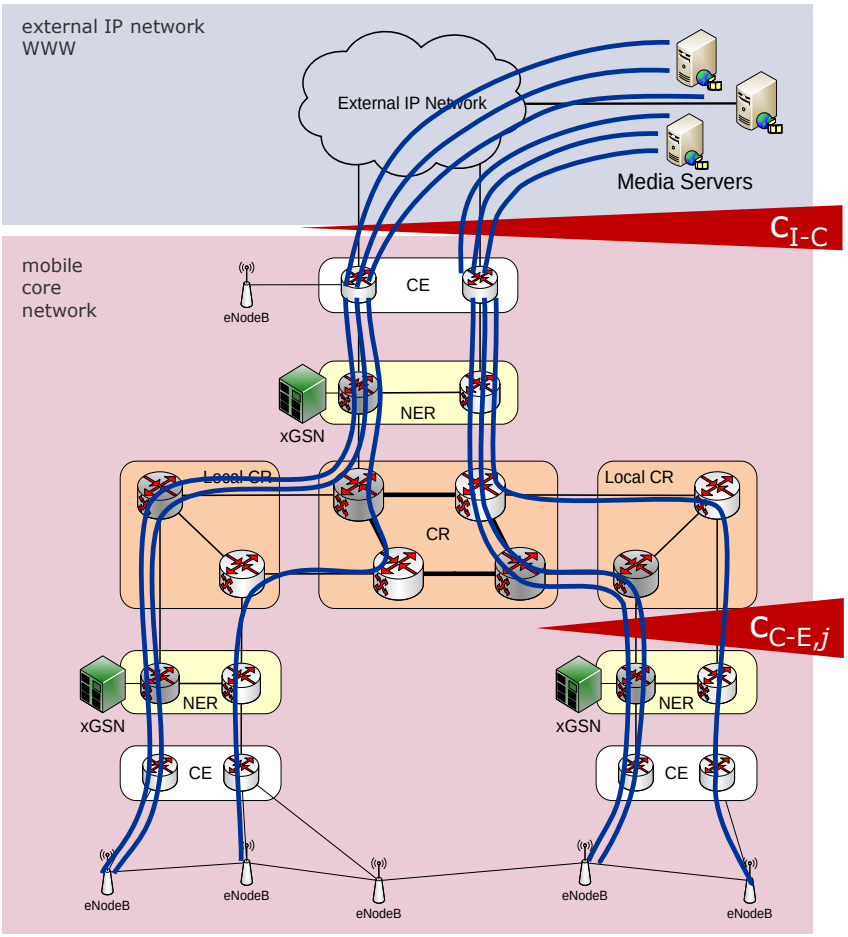


Figure 12 - Scenario 1: No caching

The traffic on the link I-C and the links C-E can be expressed as:

$$c_{\text{I-C}}(\mathbf{X}) = \sum_{i=1}^{I} \lambda_i \left(\sum_{j=1}^{J} b_j p_{ij} \right)$$

$$c_{\text{C-E},i}(\mathbf{X}) = \lambda_i \left(\sum_{j=1}^{J} b_j p_{ij} \right)$$

5.2.2.2 Scenario 2: One central cache in core network

Centralized approach: one central in inside the core networks caches the Top X most popular video files. Requests for these videos are then streamed from this cache. Videos which are requested infrequently are still streamed from the media servers in the external network. See Figure 13.

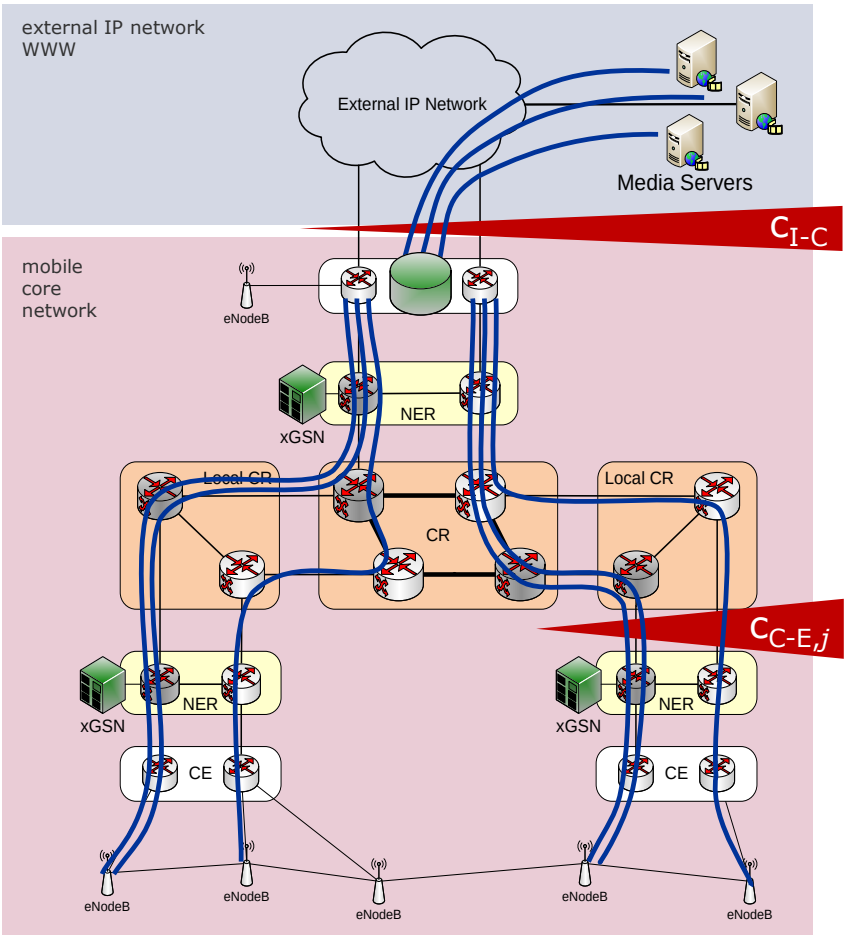


Figure 13 - Scenario 2: One centralized cache

The traffic on the link I-C and the links C-E can be expressed as:

$$c_{\text{I-C}}(\mathbf{X}) = \underbrace{\sum_{j=1}^{J} b_j x_{0,j}}_{\text{in cache}} + \underbrace{\sum_{i=1}^{I} \lambda_i \left(\sum_{j=1}^{J} b_j p_{ij} x_{0,j}^{-1} \right)}_{\text{not in cache}}$$

$$c_{\text{C-E},i}(\mathbf{X}) = \lambda_i \left(\sum_{j=1}^{J} b_j p_{ij} \right)$$

With a centralized cache, according to the 80-20 rule, by caching the Top 20% of videos, around 80% of the traffic on the link I-C can be saved (see Figure 14). This also means a reduction of peering costs at the ingress point and traffic costs on the link I-C (from ingress router to cache) by 80%.

Yet, traffic after the cache will not be reduced compared to scenario 1 (no caching).

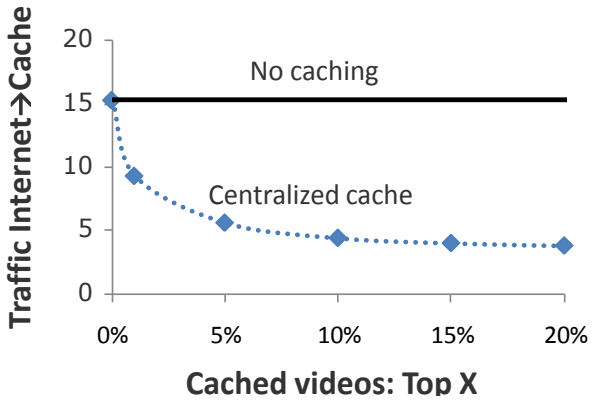


Figure 14 - Traffic saving on link I-C when caching the Top X videos in scenario 2

Note: Short term traffic projections/predictions show that the required capacity levels will be increasing in the order of one magnitude. In the long term, Akamai assumes serious scalability questions.

5.2.2.3 Scenario 3: Autonomous regional caches

The central cache is distributed among all regions. In this scenario, all caches operate independent of each other, i.e. the Top X most popular video files are cached in each region. Less popular traffic is still streamed from the external media servers. See Figure 15.

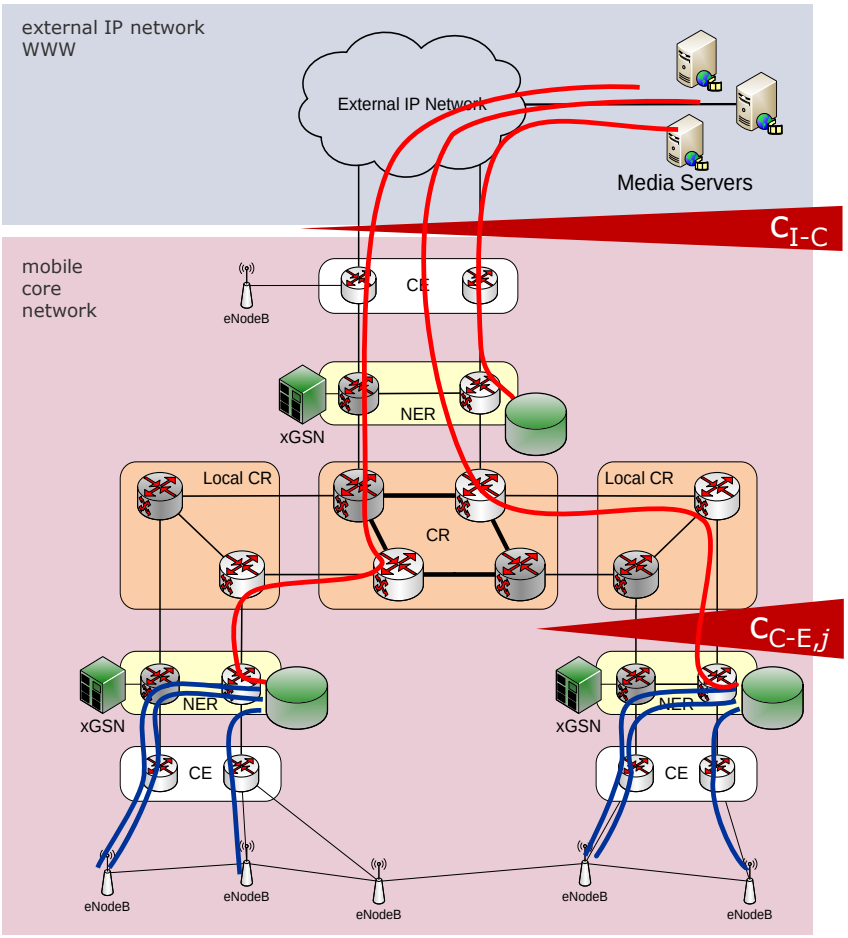


Figure 15 - Scenario 3: Autonomous regional caches

The traffic on the link I-C and the links C-E can then be expressed as:

$$c_{\text{I-C}}(\mathbf{X}) = \underbrace{\sum_{i=1}^{I}\sum_{j=1}^{J} b_j x_{ij}}_{\text{update } \textit{all} \text{ caches}} + \underbrace{\sum_{i=1}^{I} \lambda_i \left(\sum_{j=1}^{J} b_j p_{ij} x_{ij}^{-1} \right)}_{\text{not in cache of related region}}$$

$$c_{\text{C-E},i}(\mathbf{X}) = \underbrace{\sum_{j=1}^{J} b_j x_{ij}}_{\text{in cache}} + \underbrace{\lambda_i \left(\sum_{j=1}^{J} b_j p_{ij} x_{ij}^{-1} \right)}_{\text{not in cache}}$$

Applying Scenario 3, the distributed caches can reduce the load in the core network (i.e. traffic load and processing at the network components) by more than 50%. Moreover, users will benefit from a better QoE due to shorter delay and startup time, as the cached content can be served from a location closer to the user compared to streaming the content from an external service provider.

Note: when considering UCG, the locality of data also needs to be considered!

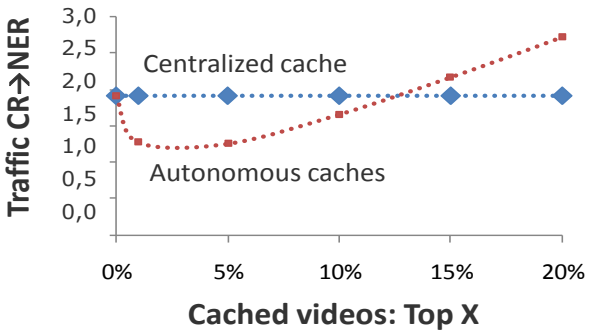


Figure 16 - Traffic saving on link CR-NER when caching the Top X videos in scenario 3

5.2.2.4 Scenario 4: Collaborative regional caches

In this scenario, regional caches in all of the *I* regions exists. The caches cooperate with each other (see Figure 17), i.e. each of the Top X most popular videos is downloaded from the external source by one cache only and is then distributed among the regional caches. Thereby, the traffic at the ingress router (link I-C) is further reduced, but traffic at the links C-E is slightly increased.

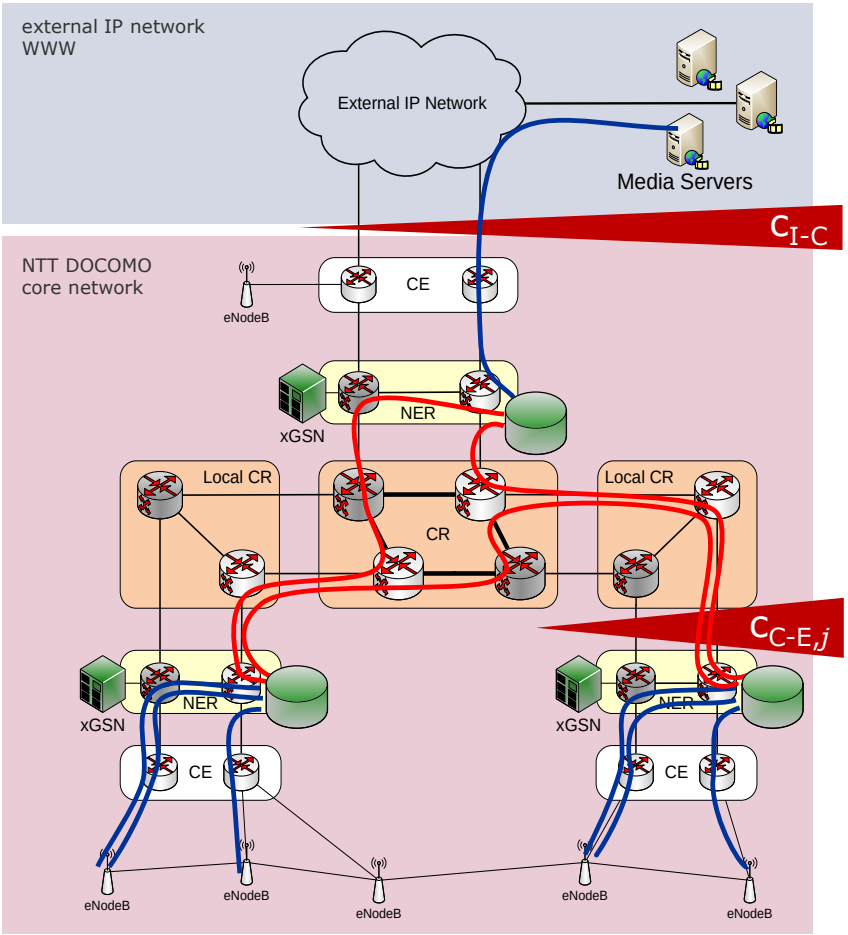


Figure 17 - Scenario 4: Collaborative regional caches

The costs cC-E,i are composed of four components:

I. Download files to local cache i (from external IP network or any other cache i')

II. Download requests for objects that are not cached locally (from external or any other cache i')

III. Uploads to caches in other regions i'

IV. Uploads to users in regions where the objects are not cached

In that scenario we have to make two assumptions.

Traffic from external sources costs tt + l1, whereas downloading traffic from another cache i' to cache i costs t2,i' + t2,i.

First, we assume that objects are only downloaded from the external source if the content is not available in any other cache, either due to company policies or as traffic from internal caches is cheaper than downloading from the external IP network (t2,i' + t2,i < tt + l1).

Second, we assume that requests from region i that could not be served from the local cache of region i, but from caches in ΣIi"=1 (xi"j) regions will be evenly served from these regions (load balancing), either due to company policies or as traffic costs among all caches are equal l2,i = l2.

$$c_{\text{I-C}}(\mathbf{X}) = \underbrace{\sum_{j=1}^{J} b_j \bigvee_{i=1}^{I} x_{ij}}_{\substack{\text{objects stored} \\ \text{in } any \text{ cache}}} + \underbrace{\sum_{i=1}^{I} \lambda_i \sum_{j=1}^{J} b_j p_{ij} \bigwedge_{i=1}^{I} x_{ij}^{-1}}_{\substack{\text{objects } not \text{ stored} \\ \text{in any cache}}}$$

$$c_{\text{C-E},i}(\mathbf{X}) = \underbrace{\sum_{j=1}^{J} b_j x_{ij}}_{\text{I}} + \underbrace{\lambda_i \sum_{j=1}^{J} b_j p_{ij} x_{ij}^{-1}}_{\text{II}} + \underbrace{\underbrace{\sum_{\substack{i'=1,\\ i' \neq i}}^{I}}_{\substack{\text{Requests from} \\ \text{regions } i'}} \underbrace{\sum_{j=1}^{J}}_{\substack{\text{for all} \\ \text{objects } j}} \underbrace{x_{ij} x_{i'j}}_{\substack{\text{with object } j \\ \text{being cached in} \\ \text{both } i \text{ AND } i'}} b_j \underbrace{\frac{1}{\sum_{i''=1}^{I} x_{i''j}}}_{\text{Load balancing}}}_{\text{III}} + \underbrace{\underbrace{\sum_{\substack{i'=1,\\ i' \neq i}}^{I}}_{\substack{\text{Requests from} \\ \text{regions } i'}} \lambda_{i'} \sum_{j=1}^{J} \underbrace{x_{ij} x_{i'j}^{-1}}_{\substack{\text{object } j \text{ is} \\ \text{cached in } i \\ \text{but NOT in } i'}} b_j p_{i'j} \underbrace{\frac{1}{\sum_{i''=1}^{I} x_{i''j}}}_{\text{Load balancing}}}_{\text{IV}}$$

c_s	Costs for providing the storage
c_t	Costs for transit traffic
l_1	Link costs for traffic between ingress point and CR
$l_{2,i}$	link costs between CR and CE of region i
$C(S)$	Total costs
S	Storage capacity of caches
J	Number of objects (videos)
λ_i	Request rate in region i
b_j	Size of object j
p_{ij}	Request probability of object j in region i
x_{ij}	Boolean variable: 1 if object j is cached in region i
x_{ij}^{-1}	Negation of x_{ij}

Using cooperation among caches can further reduce the overall costs. The above cost model considers storage, traffic, and peering costs. Using this information the goal is to find the optimal value X (Top X videos) for each cache (see Figure 18). Also, we plan to work on a distributed algorithm for managing the collaboration among the caches.

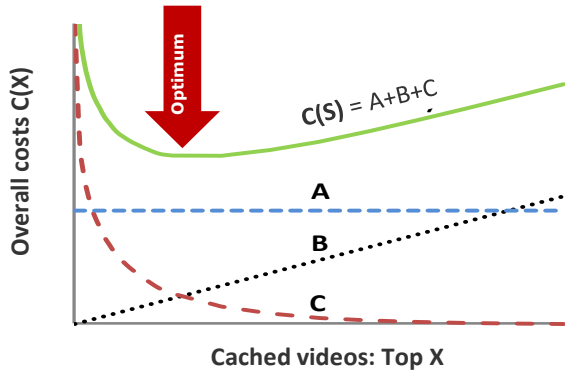


Figure 18 - Overall costs C(X) depending on caching the Top X videos

5.2.3 Evaluation

In order to evaluate the above cost model, we make several approximations, starting from a homogenous network and service usage towards a more realistic setting.

5.2.3.1 Approximation A: all regions are similar (from both network layout and service access)

$$
\begin{aligned}
I &= 9 \\
S_i &= S, \text{ same for all regions} \\
\lambda_i &= \lambda, \text{ same for all regions} \\
\\
J & \text{ fixed, e.g. } J = 100, \text{ sorted by popularity/rank} \\
b_j &= b, \text{ same for all objects, e.g. } b_j = 16 \text{ MB} \\
p_{ij} &= p_j, \text{ request probability of object } j \text{ (same for all regions)} \\
\sum_{j=1}^{J} p_j &= 1 \\
\\
x_{ij} &= x_j, \text{ determined by } p_j \text{ as the most popular objects are cached} \\
\sum_{j=1}^{J} x_j &= \lfloor S/b \rfloor = x \\
\vec{x} &= (\underbrace{1 \,..\, 1}_{\lfloor S/b \rfloor} \quad \underbrace{0 \,..\, 0}_{J-\lfloor S/b \rfloor})^T \\
p_j\vec{x} &= (\, p_1 \,..\, p_x \; 0 \,..\, 0 \,)^T \\
p_j\vec{x^{-1}} &= (\, 0 \,..\, 0 \; p_{x+1} \,..\, p_J \,)^T \\
\\
l_{2,i} &= l_2, \text{ same for all regions}
\end{aligned}
$$

Then, the total costs C can be expressed as:

$$C = c_1 + c_2 + c_3$$

$$C = \underbrace{(l_1 + t_t) \cdot c_{\text{I-C}}}_{\text{link costs } \textit{prior} \text{ to CN}} + \underbrace{\sum_{i=1}^{I} l_{2,i} \cdot c_{\text{C-E},i}}_{\text{link costs } \textit{after} \text{ to CN}} + \underbrace{\sum_{i=1}^{I} w \cdot S_i}_{\text{storage costs}}$$

Regarding the above scenarios, the costs C are as follows:

A.1 Scenario 1: Without caching

$$c_{\text{I-C}} = I\lambda b \sum_{j=1}^{J} p_j = I\lambda b$$

$$c_{\text{C-E},i} = \lambda b \sum_{j=1}^{J} p_j = \lambda b$$

$$C = (c_t + l_1 + l_2) \cdot Ib\lambda$$

A.2 Scenario 2: Centralized cache

$$c_{\text{I-C}} = S + I\lambda b \sum_{j=1}^{J} p_j x_j^{-1}$$

$$c_{\text{C-E},i} = \lambda b \sum_{j=1}^{J} p_j = \lambda b$$

$$C = (c_t + l_1 + c_s) \cdot S + (c_t + l_1) \cdot Ib\lambda \sum_{j=1}^{J} p_j x_s^{-1} + l_2 \cdot Ib\lambda =$$

$$= (c_t + l_1 + c_s) \cdot S + (c_t + l_1) \cdot Ib\lambda \sum_{j=x+1}^{J} p_j + l_2 \cdot Ib\lambda \qquad \text{with } x = \lfloor \frac{S}{b} \rfloor$$

A.3 Scenario 3: Autonomous regional caches

$$c_{\text{I-C}} = IS + I\lambda b \sum_{j=1}^{J} p_j x_j^{-1}$$

$$c_{\text{C-E},i} = S + b\lambda \sum_{j=1}^{J} p_j x_j^{-1}$$

$$C = (c_t + l_1) \cdot (IS + Ib\lambda \sum_{j=1}^{J} p_j x_s^{-1}) + l_2 \cdot Ib(b \sum_{j=1}^{J} x_j + b\lambda \sum_{j=1}^{J} p_j x_j^{-1}) + Ic_s S =$$

$$= (Ic_t + Il_1 + Il_2 + Ic_s) \cdot S + (c_t + l_1 + l_2) \cdot Ib\lambda \sum_{j=x+1}^{J} p_j \qquad \text{with } x = \lfloor \frac{S}{b} \rfloor$$

A.4 Scenario 4: Collaborative regional caches

$$c_{\text{I-C}} = S + I\lambda b \sum_{j=1}^{J} p_j x_j^{-1}$$

$$c_{\text{C-E},i} = \underbrace{S}_{\text{I}} + \underbrace{b\lambda \sum_{j=1}^{J} p_j x_j^{-1}}_{\text{II}} + \underbrace{0}_{\text{III}} + \underbrace{S}_{\text{IV}}$$

$$C = (c_t + l_1 + 2Il_2 + Ic_s) \cdot S + (c_t + l_1 + l_2) \cdot Ib\lambda \sum_{j=x+1}^{J} p_j \qquad \text{with } x = \lfloor \frac{S}{b} \rfloor$$

Summarizing, the equations of all approximations can be described as:

$$C(S) = \underbrace{d \cdot S}_{\text{A}} + \underbrace{e \cdot \sum_{j=x+1}^{J} p_j}_{\text{B}} + \underbrace{f}_{\text{C}}$$

Plotting the three components A, B, C, and the overall costs C(S), results in the diagram given in Figure 19.

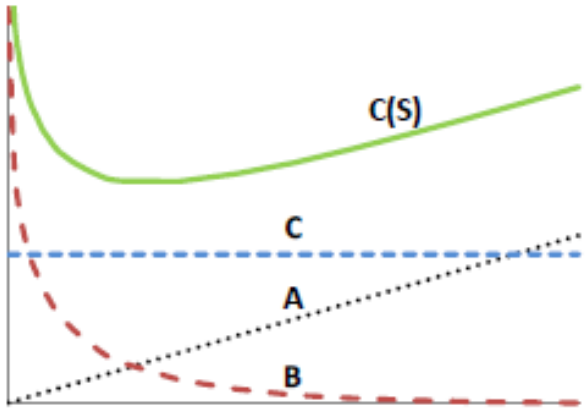


Figure 19 - Overall costs C(X) consist of three components A, B, C

5.2.3.2 Approximation B: Regions are different, but objects have same filesize bj = b = const.

The optimal selection of files in the cache is now a 0-1-knapsack problem, with b_j being the "weight" of object *j* and the product $b_j \cdot p_{ij}$ its "value". Thus, a knapsack problem must be solved for finding the optimum solution of the centralized cache, and *J* independent knapsack problems must be solved for regional caches without cooperation. Regarding the scenario with cooperation between the caches, the selection of files for the caches is even more complex, as the knapsack problems are no longer independent of each other.

Assuming S » b_j and b_j = b we can reduce the complexity of the problem:

If all objects have similar "weights" b the knapsack is simply filled with the most popular items. Sorting all objects by decreasing popularity, the vectors x_i are composed of x = $\lfloor S_i / b \rfloor$ "ones" and J-x "zeros":

$$\vec{x_i} = (\underbrace{1 \ldots 1}_{\lfloor S_i/b \rfloor} \quad \underbrace{0 \ldots 0}_{J-\lfloor S_i/b \rfloor})^T$$

For sake of simplicity, we define $l_t = t_1 + t_t$ and w.l.o.g S = n / b with n = 0,1,2,… .

The following results show that for scenarios 1-3 (no caching, centralized and autonomous caches) the resulting costs have a similar shape than the curves shown on page 10. The reason behind this is that the placement of the objects can still be derived from the popularity distribution in the different regions.

Only in the case of collaborative regional caches, the specific placement of objects in one cache will also influence the decision which objects to cache in another region. Thus, in order to find out the best overall solution, an integer program has to be solved. Basically this is a sort of knapsack problem. Yet, in a basic knapsack problem each object has some fixed value and weight, whereas in our problem the objects can be put into multiple knapsacks („regions“) and the value of caching a specific file changes depending on where/if the objects are additionally cached in other regions.

The problem can be shown to be NP complete.

Assuming approximation B, the four scenarios result in the following equations.

B.1 Scenario 1: Without caching

$$\begin{aligned} C(S=0) &= l_t \sum_{i=1}^{I} \lambda_i \sum_{j=1}^{J} b_j p_{ij} + \sum_{i=1}^{I} l_{2,i} \lambda_i \sum_{j=1}^{J} b_j p_{ij} = \\ &= \sum_{i=1}^{I} (l_t + l_{2,i}) \lambda_i b \end{aligned}$$

B.2 Scenario 2: Centralized cache

$$\begin{aligned} C(S_0) &= l_t \left(\sum_{j=1}^{J} b_j x_{0,j} + \sum_{i=1}^{I} \lambda_i \sum_{j=1}^{J} b_j p_{ij} x_{0,j}^{-1} \right) + \sum_{i=1}^{I} l_{2,i} \lambda_i \sum_{j=1}^{J} b_j p_{ij} + \sum_{i=1}^{I} c_s S_0 = \\ &= l_t \sum_{j=1}^{J} b_j x_{0,j} + l_t \sum_{i=1}^{I} \lambda_i \sum_{j=1}^{J} b_j p_{ij} x_{0,j}^{-1} + \sum_{i=1}^{I} l_{2,i} \lambda_i \sum_{j=1}^{J} b_j p_{ij} + \sum_{i=1}^{I} c_s S_0 = \\ &= l_t S_0 + l_t b \sum_{i=1}^{I} \lambda_i \sum_{j=S/b+1}^{J} p_{ij} + b \sum_{i=1}^{I} l_{2,i} \lambda_i + \sum_{i=1}^{I} c_s S_0 \\ &= (l_t + \sum_{i=1}^{I} c_s) S_0 + l_t b \sum_{i=1}^{I} \lambda_i \sum_{j=S/b+1}^{J} p_{ij} + b \sum_{i=1}^{I} l_{2,i} \lambda_i \end{aligned}$$

B.3 Scenario 3: Autonomous regional caches

$$\begin{aligned} C(\vec{S}) &= l_t \left(\sum_{i=1}^{I} \sum_{j=1}^{J} b_j p_{ij} + \sum_{i=1}^{I} \lambda_i \sum_{j=1}^{J} b_j p_{ij} x_{ij}^{-1} \right) + \sum_{i=1}^{I} l_{2,i} \left(\sum_{j=1}^{J} b_j x_{ij} + \lambda_i \sum_{j=1}^{J} b_j p_{ij} x_{ij}^{-1} \right) \\ &= l_t \sum_{i=1}^{I} \sum_{j=1}^{J} b_j p_{ij} + l_t \sum_{i=1}^{I} \lambda_i \sum_{j=1}^{J} b_j p_{ij} x_{ij}^{-1} + \sum_{i=1}^{I} l_{2,i} \sum_{j=1}^{J} b_j x_{ij} + \sum_{i=1}^{I} l_{2,i} \lambda_i \sum_{j=1}^{J} b_j p_{ij} x_{ij}^{-1} + \sum_{i=1}^{I} c_s S_i = \\ &= l_t I b + l_t b \sum_{i=1}^{I} \lambda_i \sum_{j=S/b+1}^{J} p_{ij} + \sum_{i=1}^{I} l_{2,i} S_i + b \sum_{i=1}^{I} l_{2,i} \lambda_i \sum_{j=S/b+1}^{J} p_{ij} + \sum_{i=1}^{I} c_s S_i = \\ &= \sum_{i=1}^{I} (l_{2,i} + c_s) S_i + b \sum_{i=1}^{I} (l_t + l_{2,i}) \lambda_i \cdot \sum_{j=S/b+1}^{J} p_{ij} + l_t I b \end{aligned}$$

B.4 Scenario 4: Collaborative regional caches

$$
\begin{aligned}
c_{\text{I-C}}(\mathbf{X}) &= b\sum_{j=1}^{J}\bigvee_{i=1}^{I} x_{ij} \quad + \quad \sum_{i=1}^{I}\lambda_i\sum_{j=1}^{J} b_j p_{ij}\bigwedge_{i=1}^{I} x_{ij}^{-1} \\
c_{\text{C-E},i}(\mathbf{X}) &= \underbrace{b\sum_{j=1}^{J} x_{ij}}_{\text{I}} + \underbrace{b\lambda_i\sum_{j=1}^{J} p_{ij}x_{ij}^{-1}}_{\text{II}} + \underbrace{b\sum_{j=1}^{J} x_{ij}\frac{(\sum_{i'=1}^{I} x_{i'j})-1}{\sum_{i''=1}^{I} x_{i''j}}}_{\text{III}} + \underbrace{b\sum_{j=1}^{J} x_{ij}\frac{\sum_{\substack{i'=1,\\ i'\neq i}}^{I} x_{i'j}^{-1}\lambda_{i'}p_{i'j}}{\sum_{i''=1}^{I} x_{i''j}}}_{\text{IV}} = \\
&= S_i + b\lambda_i\sum_{j=S/b+1}^{J} p_{ij} + b\sum_{j=1}^{S/b}\frac{\sum_{i'=1}^{I} x_{i'j} + \sum_{\substack{i'=1,\\ i'\neq i}}^{I} x_{i'j}^{-1}\lambda_{i'}p_{i'j} - 1}{\sum_{i'=1}^{I} x_{i'j}} = \\
&= S_i + b\lambda_i\sum_{j=S/b+1}^{J} p_{ij} + b\sum_{j=1}^{S/b}\left(1 + \frac{\sum_{\substack{i'=1,\\ i'\neq i}}^{I} x_{i'j}^{-1}\lambda_{i'}p_{i'j} - 1}{\sum_{i'=1}^{I} x_{i'j}}\right)
\end{aligned}
$$

A. Putting values to the equations

No data for a specific video service from any of the Telcos in the MEDIEVAL project is available. Thus, our analysis is based on YouTube data. Yet, the popularity distribution will be similar.

We take into account the popularity distribution measured in [43], and combine it with current YouTube statistics (2 billion views per day; a total of 143 million stored video files; average length of videos of 4.3 minutes with an average bitrate of 500 kbps and an average file size of 16MB) (the data based on different analysis from 2010). Considering the fact that 5% of the Internet users live in Japan and assuming that 10% of YouTube videos are watched on mobile devices, there are 97 million YouTube views per month in Japan. In addition, Japanese people are not watching all available YouTube videos (e.g. regional content, foreign languages, …) we assume that the maximal number of different YouTube videos accessed in Japan is only 50% of the available video content in YouTube. Moreover, we assume that the mobile core network is divided in 9 regions.

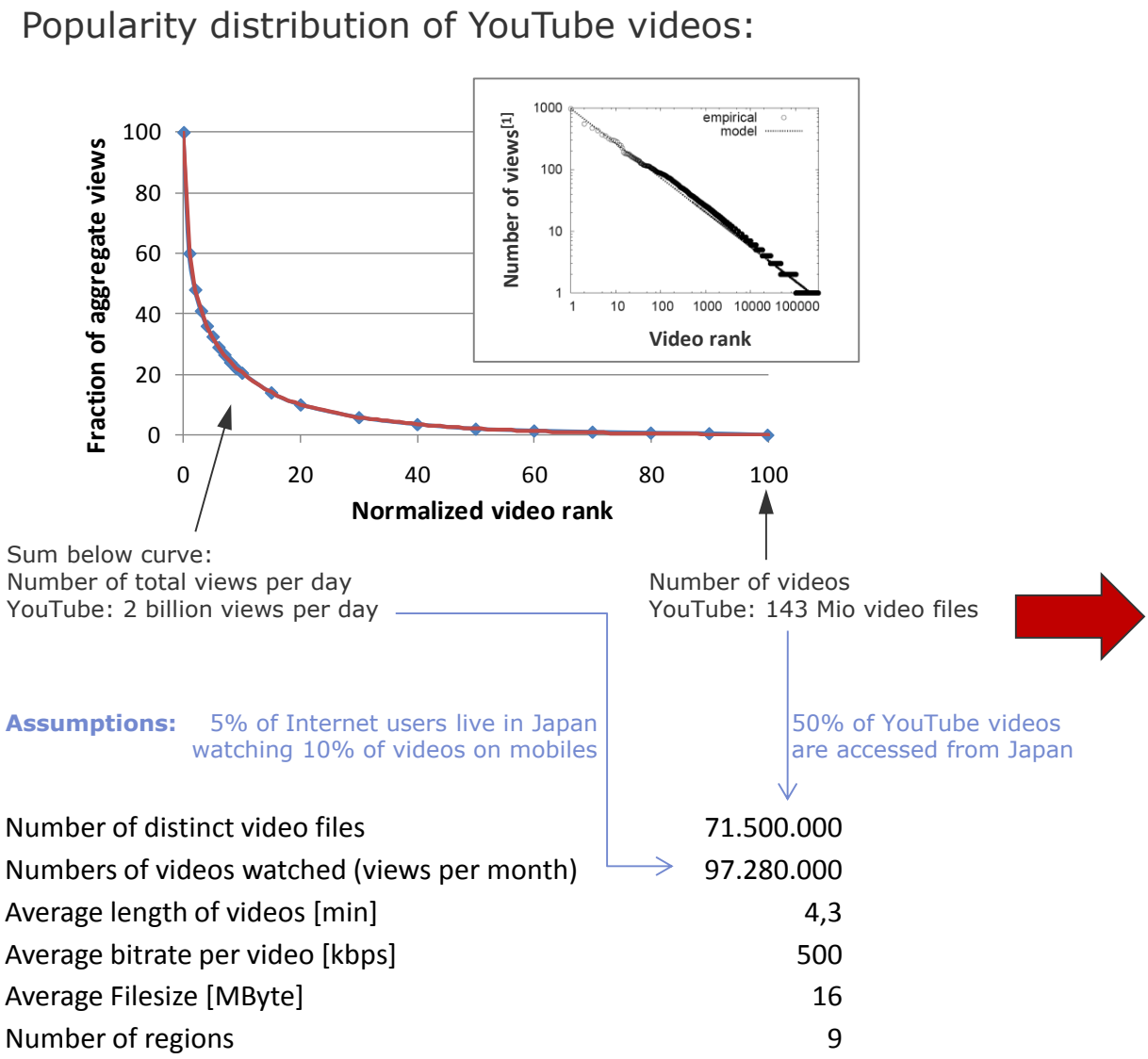


Figure 20 - Parameters and values used in the evaluation

5.2.3.3 Resulting traffic in the core network:

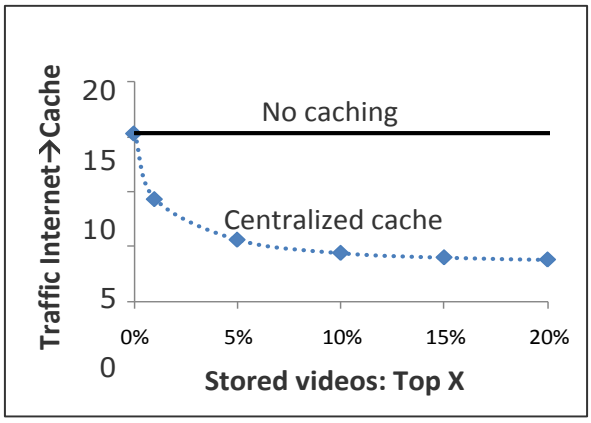


Figure 21 - Traffic at the ingress router

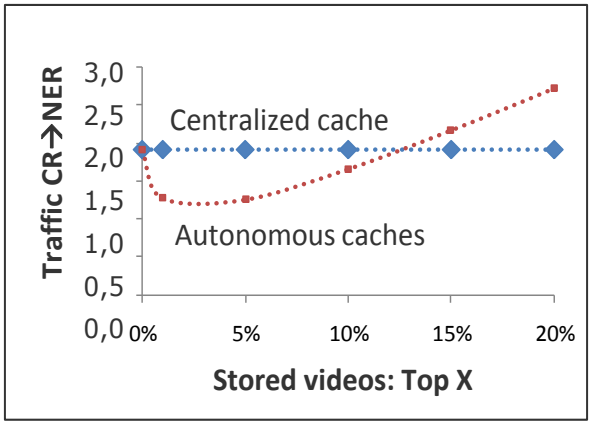


Figure 22 - Traffic on the regional links

Our traffic analysis clearly shows the advantage of caching in contrast to no caching. Based on our assumptions (content popularity distribution), the traffic at the ingress router and on the connected links is getting the smaller the more videos are stored. However, you can also observe saturation if more than 15% of the top-ranked content is cached, as content of rank 15% to 100% is not accessed frequently. Storing around 10% of the top ranked videos will reduce the traffic by around 60%. The traffic in scenarios 3 and 4 is slightly higher, as multiple caches download the most popular content once; however, it is still similar to the depicted curve.

Looking at the traffic on the routers and links in the regions (traffic CR—NER, see Figure 22), neither scenario 1 (no caching) nor scenario 2 (centralized cache) result in a traffic reduction. Distributed caches that are downloading the Top X most popular videos in the whole system are able to reduce the traffic in the regions, when storing the Top 2-5% of videos. However, downloading more videos is going to reduce the effect and is even increasing the traffic on these links, as videos are downloaded which are actually never accessed by any of the nodes in this region. Thus, when the storage capacity of the CDN nodes is more than the required capacity for the Top 5% of videos, the CDN node should not download the videos proactively (which would already improve the startup delay for the first user), but just get a copy of the files on the fly. We can also learn from the figure that in a distributed CDN solution, the individual storage capacities may be designed significantly smaller than compared to a centralized cache, as less unique video files are accessed in smaller regions.

5.2.3.4 Cost data used in the analysis:

Peering costs: 60 $ per Mbps per year

Storage costs, e.g. Amazon S3:

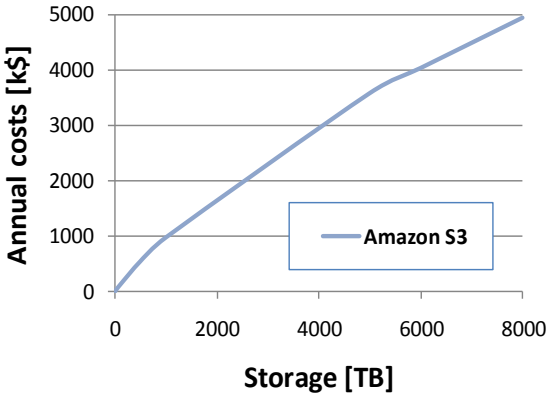


Figure 23 - Costs for storage

Maintenance/operational costs: we still try to get some values on these costs.

Costs for new equipment: we also need to estimate average costs for new network equipment.

5.2.3.5 Resulting annual costs [T$]:

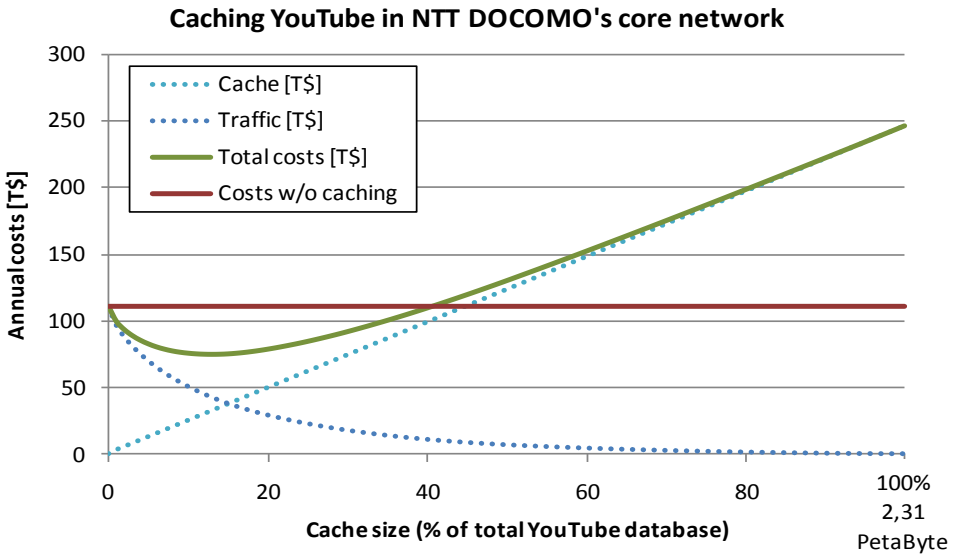


Figure 24 - Resulting annual costs

Based on our preliminary evaluation results (and without explicitly considering maintenance costs and costs for new equipment), a cache size of the Top 12% of videos results in lowest overall costs. Although we don't have estimates of maintenance, operational, and equipment costs, we believe that our results are realistic, as these costs are also already considered in the prices offered by third parties like Amazon. However, gross costs for storage and might be calculated different within a specific operator.

In the upcoming period, we are planning to refine our cost model and evaluations; in particular we want to consider autonomous caches that are downloading content on-the-fly, which should further reduce the traffic in the system, in particular in the regions. We are also planning to simulate the CDN system in order to verify our analysis and implement the probability based caching (PBC) in the CDN nodes. Moreover, we started looking at cooperative CDN nodes, their information exchange and the benefits of the cooperation. As the benefits of cooperation among CDN nodes are limited in hierarchical networks, we aim at defining a list of requirements for a future mobile network architecture, where adjacent nodes are better connected to each other (meshed network topology).

5.3 Performance Evaluation of the Network Coded Layered Multicast Algorithm

For the performance analysis, we implement our MultiLayer-MaxFlow heuristic (called ML-MaxFlow in the legend of the graphs). For comparison, we also implement the Min-Req and Min-Cut algorithms from [42] as well as a simple intra-layer coding mechanisms that simply packs multicast trees without performing coding across different layers in Matlab. For ease of comparison, we use the same metrics of [42] for "Happy Nodes", the fraction of nodes that achieve their max-flow value, and "Rate Achieved", the average of actual rate achieved normalized by the max-flow value. In addition, we measure network load in terms of number of links used vs. total number of links. For the coding we use random linear network coding over a finite field GF(210). Simulation results are averaged over 1000 runs. Topologies are generated by randomly establishing links between the nodes of the network, ensuring that the resulting topology is connected and free of cycles. For the number of links we set $|E|=\gamma|V|$ with $\gamma=3.7$ for the graphs shown here.

5.3.1 Impact of network size

Figure 21 shows the performance of the different algorithms for networks of 20-320 nodes (including 95% confidence intervals). All algorithms perform relatively well for small networks since the low max-flow values are easier to achieve. The disadvantage of the Min-Req algorithm becomes apparent as soon as the network size increases. The fraction of paths that intersect with low max-flow receivers increases as well, whereas the fraction of happy nodes that achieve their max-flow, as well as the normalized rate achieved, both decrease. The Min-Cut algorithm performs significantly better. While the normalized rate achieved is relatively stable as the number of nodes increases, the fraction of happy nodes in fact increases. As the network becomes more homogeneous with increasing size, lower max-flow receivers manage to achieve the max-flow but this is compensated by the loss of normalized rate at some high max-flow receivers that no longer manage to obtain the higher layers they require. For ML-MaxFlow, both the fraction of happy nodes as well as the normalized rate increase with a network size beyond 80 nodes. Similarly to the Min-Cut algorithm, lower max-flow receivers achieve their max-flow. Since only the necessary number of paths is allocated to low max-flow receivers, high max-flow receivers can exploit the remaining paths to obtain sufficiently high layers.

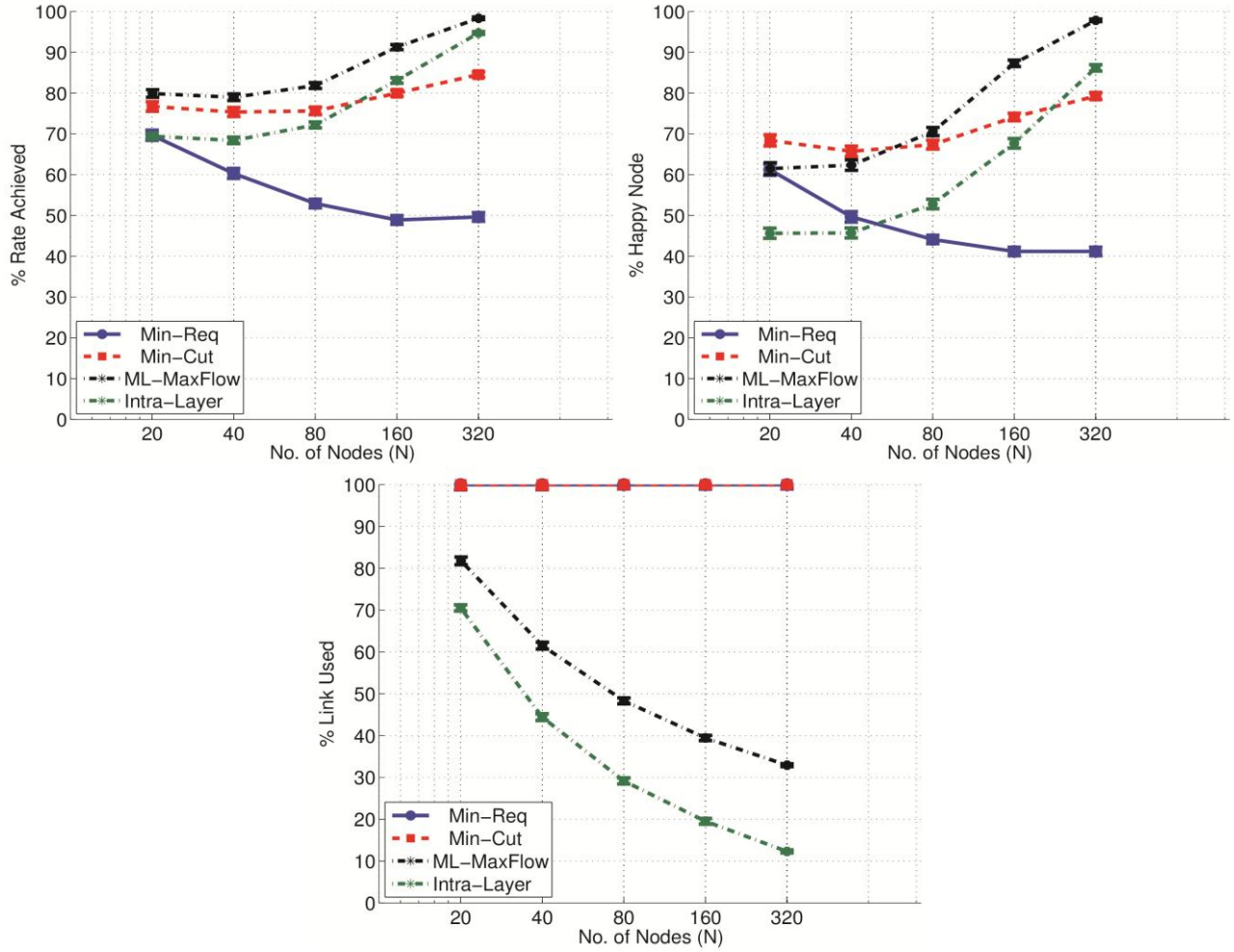


Figure 25 - Different network sizes of 20 - 320 nodes, for 10 receivers

ML-MaxFlow also uses substantially fewer links. For a small topology of 20 nodes, around 80% of the links are used, and as the network size increases (while maintaining a constant number of receivers of 10) this fraction drops down to around 30%.

5.3.2 Impact of number of receivers

We vary the number of receivers from 1 to 80 in a network with |V | = 160 nodes. For only a single receiver, all algorithms achieve the full rate of 100%. When the number of receivers becomes a significant fraction of the total number of nodes (20% or more) the Min-Cut algorithm performs better, but even then the performance of the ML-MaxFlow algorithm is very similar. In this case, paths of high max-flow and low max-flow receivers frequently overlap. Since without decoding, low max-flow receivers enforce low layer constraints on the whole path from the receivers to the sender, few available paths for high max-flow receivers remain, despite ML-MaxFlow assigning only the necessary number of paths. In contrast, in a sufficiently dense network the Min-Cut algorithm maintains high maximum layer constraints throughout the core of the network and only needs to decode close to the receivers to deliver exactly the right number of layers to each. Here, the additional flexibility with respect to layer assignments provided by the decoding at interior nodes pays off. However, for the more interesting scenario of a low to medium number of receivers, the ML-MaxFlow algorithm outperforms the Min-Cut algorithm both in terms of rate achieved and in terms of happy nodes.

As expected, with an increasing number of receivers ML-MaxFlow uses more and more links, until 75% of the links are active for the case where half of the nodes of the network are receivers.

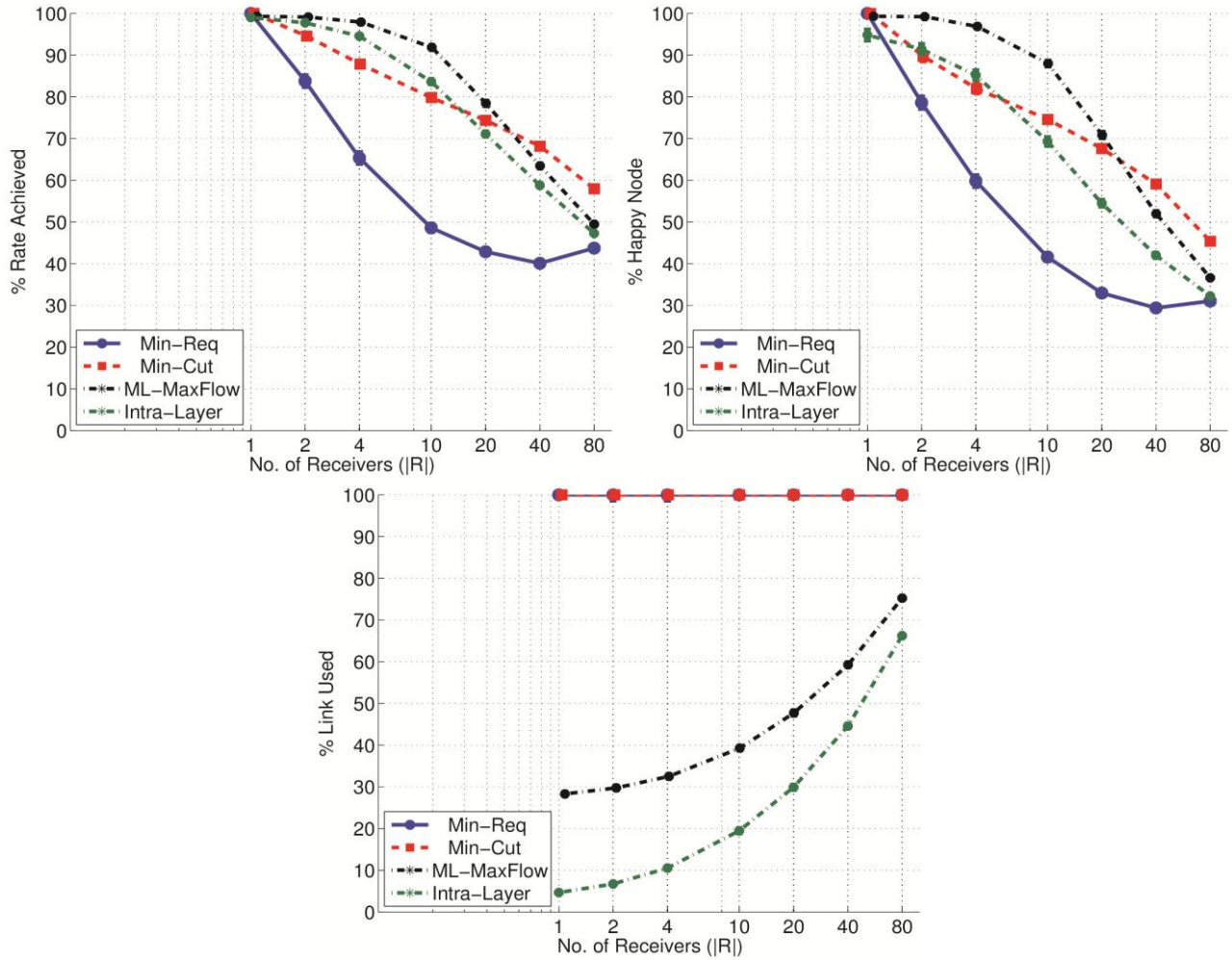


Figure 26 - Different number of receivers of 1 - 80 nodes, for a network with 160 nodes

5.4 CDN Implementation

We plan to implement (the main part of) the CDN components and algorithms for proof-of-concept and further evaluation.

We started with a survey of existing open source CDN solutions. Therefore, we defined the following metrics for selecting the most appropriate CDN solution:

- maturity: stable + advanced solution
- modularity / cleanness of code
- adaptability
- possibility to set up our own CDN
- multicast integration
- licencing issues: BSD / GNU / Apache / …

Several free CDN networks[5] were considered and we agreed to base our implementation on OpenCDN [44], as it met all of our above metrics. In particular, the design choice of Relay Nodes and a central Request Routing and Distribution Management (RRDM) fit very well to our Decision Module and CDN (Control) Nodes architecture. OpenCDN is vendor-agnostic by using an Adaptation Layer and Control Layer. It comes with several existing adaptations for common streaming solutions and can easily be ported to new streaming technologies, which might be required by the MEDIEVAL testbed. Moreover, multicast is supported. OpenCDN is based on Perl scripting and the code is available under the Perl Artistic License. We were already able to successfully set up an OpenCDN network consisting of:

- Streaming server (Apple's Darwin Streaming Server)
- RRDM (resource routing and distribution management)
- Published Contents Portal
- Cache nodes (for Darwin Streaming)
- Streaming Client (Totem Player)

As next step, we need to specify more details on the requirements of the different work packages and the joint testbed, in order to adapt the OpenCDN implementation to our needs. In addition, we will soon start on figuring out, which parts of the code need to be changed in order to implement both the NEGOCODE functionality and the new dissemination strategy.

Our CDN implementation will be based on and extend the OpenCDN framework (see Section 5.4). Hence, the CDNNC needs to interface to OpenCDNs API for content management. In OpenCDN, the Request Routing and Distribution Management entity (RRDM) is responsible to manage the CDN nodes (so-called relays). The RRDM already provides much of the required functionality of the CDNNC [44]. The RRDM combines request routing and distribution management. For the CDNNC, only the distribution management is of relevance, whereas request routing is related to DM functionality. Communication between the RRDM's distribution management module and the CDN nodes is realized through the exchange of simple XML messages using the XML-RPC framework. It provides the following primitives:

- Registration (re-registration, de-registration) of CDN nodes and their capabilities with the RRDM when they become (un-)available
- Distribution of content copies to CDN nodes through relay setup requests
- Removal of content copies to CDN nodes through relay tear-down requests
- Higher-level intelligence such as load balancing is not directly supported and needs to be implemented in the CDNNC.

[5] http://www.fromdev.com/2011/06/create-cdn-content-delivery-network.html

6 Summary and Conclusion

In MEDIEVAL, video streaming is optimized with a specific focus on the selection of cache locations based on e.g. "network distance", availability, and content popularity. The goal on one hand is to provide video-aware transmission features that best fit the user needs for the specific application, e.g., delay-sensitive interactive video versus video-on-demand streaming. On the other hand, the use of the scarce resources of the network need to be optimized.

This deliverable introduces an architecture for a mobile CDN network in the context of the MEDIEVAL system, integrating CDN functionalities within the cellular network architecture. The CDN component is a crucial design choice which enables large scale content distribution at a reasonable cost and without overloading the operator's core network. Caching video data close to the users is able to partly compensate the significant increase of video traffic in mobile networks, which is reported to double every year, thus having the highest growth rate of any application category [1].

This deliverable focuses on three innovations: NEGOCODE-based Request Routing Optimization, Mobile Operator CDN management and a Network Coded Layered Multicast Algorithm.

For each of these 3 items, it presents:

- the functional architecture and description,
- the APIs within WP5 and with other MEDIEVAL WPs,
- the evaluation method and progress.

The NEGOCODE module is currently under development. The resulting prototype will be evaluated wrt the performance value associated to the CDN Nodes selected with the assistance of this module.

The next and final step of the MEDIAVAL WP5 work will be the finalization of this performance assessment and consequent system improvements. We also plan to elaborate on potential requirements to a next generation mobile network architecture from the perspective of the CDN architecture.

A. Annex – Internal interface specification

A.1 Interface DM_XLO_If

This interface is used to request the list of End-Points (EPs) from the XLO to the DM. Also, DM informs XLO of the handover to overcome congestions. This interface sets a flag to distinguish multicast/unicast sessions for the handover messages.

A.1.1 Interface DM_XLO_ALTO

A.1.1.1 DM_XLO_ALTO.Request

Function

This service primitive is used by the XLO to request the list of EPs available in the network from the DM module, for a specific session of a mobile user.

Semantics of the service primitive

```
DM_XLO_ALTO.Request(
    Session_ID,
    MN_ID
)
```

Parameter	Type	Description
Session_ID	UID	Identifier of the session
MN_ID	UID	Identifier of the mobile user

Table: DM_XLO_ALTO.Request parameter list

When generated

The XLO is periodically requesting an update of the EPs available with their descriptions.

Effect on receipt

The DM provides the list of available EPs for a given session of a user to the XLO, which then runs the optimisation algorithm for selecting a video path in the network with better performance metrics, if available and necessary, using also the wireless metrics coming from the interface L25_XLO_if (**Error! Reference source not found.**).

A.1.1.2 DM_XLO_ALTO.Response

Function

This service primitive is used by the DM to provide the list of EPs available in the network to the XLO module.

Semantics of the service primitive


```
DM_XLO_ALTO.Response(
    Session_ID,
    ReturnCode,
    End_Points_List
)
```

Parameter	Type	Description
Session_ID	UID	The session identifier.
ReturnCode	UINT	ReturnCode of the required operation: 200 OK 500 ERROR
End_Points_List	List{UID, REAL, UINT}	Vectors of: EP ID, cache load and distance cost (hopcount)

Table: DM_XLO_ALTO.Response parameter list

When generated

The DM sends the EPs available with their descriptions upon request.

Effect on receipt

The XLO runs the optimisation algorithm for selecting a video path in the network with better performance metrics, if available and necessary, using also the wireless metrics coming from the interface L25_XLO_If (see D5.2).

A.1.2 DM_XLO_Optimize

A.1.2.1 DM_XLO_Optimize.Request

Function

This service primitive is used by the DM to request the XLO to perform an optimisation run (thus, using also wireless metrics) and send back the best video path available computed.

Semantics of the service primitive

```
DM_XLO_Optimize.Request(
    End_Points_List{
        EP_ID,
        cache_load,
        distance
    }
)
```

Parameter	Type	Description

End_Points_List{
EP_ID,
cache_load,
distance
}
	List{
	UID,
	REAL,
	UINT
	}
		Vectors of: EP ID, cache load and distance cost (hopcount)

Table: DM_XLO_Optimize.Request parameter list

When generated

The DM is periodically requesting an update of the best video path.

Effect on receipt

The XLO run the optimisation algorithm to select the best video path and provide the selected EP

A.1.2.2 DM_XLO_Optimize.Response

Function

This service primitive is used by the XLO to provide the best EP available to the DM module, based also on the wireless metrics.

Semantics of the service primitive

```
DM_XLO_Optimize.Response(
     ReturnCode,
     End_Point{
          EP_ID,
          cache_load,
          distance
     }
)
```

Parameter	Type	Description
ReturnCode	UINT	ReturnCode of the required operation: 200 OK 500 ERROR

End_Point{
EP_ID,
cache_load,
distance
}
	{UID, REAL, UINT}	EP ID, cache load and distance cost (hopcount)

Table: DM_XLO_Optimize.Response parameter list

When generated

It is generated after performing the optimisation algorithm in the XLO module.

Effect on receipt

The DM gets the best EP available among a list of EPs, for specific purposes within the DM.

A.2 Interface DM_CDNNC_If

This interface is used for the DM to request and manage CDN Node related information from the CDN Node control. DM_CDNNC initiates CDN Node management operations requested by the DM such as updating the content stored in CDN Nodes. It also enables the DM to get information on a set of CDN Nodes, such as their content or operational state.

A.2.1 DM_CDNNC_CDNUpdate

A.2.1.1 DM_CDNNC_CDNUpdate.Request

Function

This message is sent from the DM to the CDN node controller to trigger an update of the content stored in one of the CDN nodes. The message may contain a store, delete, or update request.

Semantics of the service primitive

```
DM_CDNNC_CDNUpdate.Request(
    CDN_ID,
    Type,
    Content_ID
)
```

Parameter	Type	Description
CDN_ID	UID	Identifier of the CDN node
Type	Byte	Type: 0: store 1: delete 2: update (reload the content)
Content_ID	UID	Identifier of the content

Table 1: DM_CDNNC_CDNUpdate.Request parameter list

When generated

The message is generated after the DM algorithm decided to update the content cache in the CDN node(s).

Effect on receipt

The CDNNC will communicate with the CDN node to perform the requested action. For store and update requests this also includes the optimal retrieval of the content from the original source or one of the other CDN nodes. For delete requests the CDNNC will check if there is still an active session for the requested content and in that case returns a REDIRECT code.

A.2.1.2 DM_CDNNC_CDNUpdate.Response

Function

This message is a simple ACK message to a `DM_CDNNC_CDNUpdate.request`. The CDNNC will report to the DM whether the request could be performed, resulted in an error, or has to be delayed as there are still active sessions for the content to be deleted.

Semantics of the service primitive

```
DM_CDNNC_CDNUpdate.Response(
    ReturnCode
)
```

Parameter	Type	Description
ReturnCode	UINT8	ReturnCode of the required operation: 200 OK 300 REDIRECT (ongoing session; users must first be redirected) 50x ERROR

Table 2: DM_CDNNC_CDNUpdate.Response parameter list

When generated

The message is generated after the CDNNC tried to perform a content update triggered by a `DM_CDNNC_CDNUpdate.Request`. The returned message contains a return code stating whether the action could be performed successfully or not.

Effect on receipt

Upon receipt of a 300 REDIRECT message, the DM will trigger the Path Selection algorithm to redirect all users connected to the respective content on the respective CDN node to another content source. After

successful redirection, a new `DM_CDNNC_CDNUpdate.Request` for the content and CDN node will be sent to the CDNNC. In case of a 50x ERROR message, an action depending on the reported error is performed and the `DM_CDNNC_CDNUpdate.Request` will be resent afterwards.

A.2.2 DM_CDNNC_CDNStatus

A.2.2.1 DM_CDNNC_CDNStatus.Request

Function

This message is used to request a list of all CDN nodes registered at the CDNNC and the content currently stored in these CDN nodes. This information is used to synchronize the state of DM and the CDNNCs.

Optionally this message can also request information types on CDN Nodes that relates to the caches such as capacity or load of the caches. This list of information types is not restrictive and subject to implementation specific extensions.

Semantics of the service primitive

```
DM_CDNNC_CDNStatus.Request(
    Request_ID,
    CDNNode_info_id_LIST,
    CDNNode_id_LIST,
    NB_requested_info_types
)
```

Parameter	Type	Description
Request_ID	UID	Unique Identifier of the request
CDNNode_info_id_LIST	List(UID) Typedef enum{Content_List, Capacity, Load, …. } CDNNode_info_id	Array of the index of desired CDN Node information type
CDNNode_id_LIST	List(UID)	List of CDN nodes for which the information is requested
NB_requested_info_types	UINT	Number of requested CDN Node information types

Table 3: DM_CDNNC_CDNStatus.Request parameter list

When generated

This message is sent to synchronize the state between DM and connected CDNNCs. Optionally the request can be done on a restricted set of CDN Nodes that must be listed in the request parameters.

Effect on receipt

Upon receipt, the CDNNC will retrieve an up-to-date list of content from the attached CDN nodes, compile all information in a response message and return it to the DM.

A.2.2.2 DM_CDNNC_CDNStatus.Response

Function

A report of the content stored at the CDN nodes registered with the CDNNC.

Semantics of the service primitive

It is assumed that, when applicable, the units of the numerical values provided in the response are known by the DM. Assuming that the value of request parameter "CDNNode_info_id_LIST" is [Content_List, Capacity, Load], the format of the response is the following:

```
DM_CDNNC_CDNStatus.Response(
      Request_ID,
      CDN_List{
         Content_List
      },
      CDN_List{
         CDNNode_ID
      },
      CDN_List{
         Capacity
      },
      CDN_List{
         Load
      }
)
```

Parameter	Type	Description
Request_ID	UID	Unique Identifier of the request
CDN_List{CDNNode_ID}	List(UID)	List of CDN nodes for which the information is requested. The following lists preserve the same order as this list to map the information to the respective node.

CDN_List{
Content_List{
Content_ID
}
}
	List{
	List{
	Content_ID
	}
	}
		List of all CDN nodes attached to the CDNNC and the content currently stored in these CDN nodes.

CDN_List{Capacity}	List(UINT)	List of capacity value for each CDN Node
CDN_List{Load }	List(REAL)	List of load value for each CDN Node

Table 4: DM_CDNNC_CDNStatus.Response parameter list

When generated

After successfully compiling the information retrieved from the attached CDN nodes, this response message is sent to the DM. When the request is done on a restricted set of CDN Nodes, this message carries the full list of Node IDs. A specific value may indicate that information is provided for all the nodes of the CDN.

Effect on receipt

Upon receipt of this message, the DM will update its internal state of the CDN nodes and content stored. Information that relates to the cache network such as capacity or load is typically used to update the ALTO server.

A.3 Interface CDNNC_CDNnode_If

This interface is used for the CDNNC to request status information from as well as control and manage CDN Nodes.

A.3.1 CDNNC_CDNnode_CDNUpdate

A.3.1.1 CDNNC_CDNnode_CDNUpdate.Request

Function

This message is sent from the CDNNC to the CDN node to trigger an update of the content stored in one of the CDN nodes. The message may contain a store, delete, or update request. For consistency reasons, it maintains the same format as the DM_CDNNC_CDNUpdate.

Semantics of the service primitive

```
CDNNC_CDNnode_CDNUpdate.Request(
     CDN_ID,
     Type,
     Content_ID
)
```

Parameter	Type	Description
CDN_ID	UID	Identifier of the CDN node
Type	Byte	Type: 0: store 1: delete 2: update (reload the content)
Content_ID	UID	Identifier of the content

Table 5: CDNNC_CDNnode_CDNUpdate.Request parameter list

When generated

The message is generated after the CDNNC decides to update the content cache in the CDN node(s).

Effect on receipt

The CDN node will perform the requested action and updates its content.

A.3.1.2 CDNNC_CDNnode_CDNUpdate.Response

Function

This message is a simple ACK message to a CDNUpdate request. The CDN node will report to the CDNNC whether the request could be performed or resulted in an error.

Semantics of the service primitive

```
CDNNC_CDNnode_CDNUpdate.Response(
     ReturnCode
)
```

Parameter	Type	Description
ReturnCode	UINT8	ReturnCode of the required operation: 200 OK 50x ERROR

Table 6: CDNNC_CDNnode_CDNUpdate.Response parameter list

When generated

The message is generated after the CDN node tried to perform a content update triggered by a CDNUpdate request. The returned message contains a return code stating whether the action could be performed successfully or not.

Effect on receipt

When the CDNUpdate was initially requested by the DM and just passed through CDNNC, the corresponding response is relayed to the DM. Otherwise, in case of a 50x ERROR message, an action depending on the reported error is performed (request status information, node maintenance, ...).

A.3.2 CDNNC_CDNnode_CDNStatus

A.3.2.1 CDNNC_CDNnode_CDNStatus.Request

Function

This message is used to request status information from a CDN nodes registered at the CDNNC. This information is used to for CDN control and maintenance, as well as to process status requests from the DM to the CDNNC.

This message can request information types on CDN Nodes that relates to the caches such as capacity and load of the caches as well as cache content. This list of information types is not restrictive and subject to implementation specific extensions.

Semantics of the service primitive

```
CDNNC_CDNnode_CDNStatus.Request(
     Request_ID,
     CDNNode_info_id_LIST,
     CDNNode_ID,
     NB_requested_info_types
)
```

Parameter	Type	Description
Request_ID	UID	Unique Identifier of the request
CDNNode_info_id_LIST	List(UID) Typedef enum{Content_List, Capacity, Load, …. } CDNNode_info_id	Array of the index of desired CDN Node information type
CDNNode_ID	UID	CDN node for which the information is requested
NB_requested_info_types	UINT	Number of requested CDN Node information types

Table 7: DM_CDNNC_CDNStatus.Request parameter list

When generated

This message is sent by the CDNNC to obtain state information from a specific CDN Node.

Effect on receipt

The CDN node will compile the local status information and will send a response message back to the CDNNC.

A.3.2.2 CDNNC_CDNnode_CDNStatus.Response

Function

A report of the content stored at the CDN node registered with the CDNNC.

Semantics of the service primitive

It is assumed that, when applicable, the units of the numerical values provided in the response are known by the CDNNC. Assuming that the value of request parameter "CDNNode_info_id_LIST" is [Content_List, Capacity, Load], the format of the response is the following:

```
DM_CDNNC_CDNStatus.Response(
      Request_ID,
      CDN_List{
         Content_List
      },
      CDNNode_ID,
      Capacity,
      Load
)
```

Parameter	Type	Description
Request_ID	UID	Unique Identifier of the request
CDNNode_ID	UID	CDN node for which the information is requested.

Content_List{
Content_ID
}
	List{
	Content_ID
	}
		List the content currently stored in the CDN node.

Capacity	UINT	Capacity value for the CDN Node
Load	REAL	Load value for the CDN Node

Table 8: DM_CDNNC_CDNStatus.Response parameter list

When generated

After successfully compiling the local status information of the CDN node, it will send this response message back to the CDNNC.

Effect on receipt

Depending on the purpose of the status request the CDNNC will either act upon the information received or compile and relay all information (usually from multiple CDN Nodes) in a response message and return it to the DM, if the original request came from there.

A.3.3 CDNNC_CDNnode_PowerState

A.3.3.1 CDNNC_CDNnode_PowerState.Request

Function

This message is sent from the CDNNC to the CDN node to power up or power down the node or switch to a sleep state.

Semantics of the service primitive

```
CDNNC_CDNnode_PowerState.Request(
     CDN_ID,
     Type,
)
```

Parameter	Type	Description
CDN_ID	UID	Identifier of the CDN node
Type	Byte	Type: 0: power off 1: sleep 2: power on

Table 3: CDNNC_CDNnode_PowerState.Request parameter list

When generated

The message is generated after the CDNNC decides to change the power state of a CDN node.

Effect on receipt

The CDN node will perform the requested action.

A.3.3.2 CDNNC_CDNnode_PowerState.Response

Function

This message is a simple ACK message to a PowerState request. The CDN node will report to the CDNNC whether the request could be performed or resulted in an error.

Semantics of the service primitive

```
CDNNC_CDNnode_PowerState.Response(
    ReturnCode
)
```

Parameter	Type	Description
ReturnCode	UINT8	ReturnCode of the required operation: 200 OK 50x ERROR

Table 4: CDNNC_CDNnode_CDNUpdate.Response parameter list

When generated

The message is generated after the CDN node tried to change its power state. The returned message contains a return code stating whether the action could be performed successfully or not.

Effect on receipt

In case of a 50x ERROR message, an action depending on the reported error is performed (request status information, node maintenance, ...).

A.4 Interface DM_AM_If

This interface is used to request the content popularity of the Top X most popular content measured by the Application Monitoring (AM) module. Also, for each new session, an update message is send from the DM to the AM to update the popularity database.

A.4.1 DM_AM_Popularity

A.4.1.1 DM_AM_Popularity.Request

Function

This service primitive is used by the DM to request the content popularity of the Top X most popular content measured by the Application Monitoring (AM) module. A region_ID can be specified in order to get the popularity measurement for this specific region.

Semantics of the service primitive

```
DM_AM_Popularity.Request(
    Request_ID,
    X,
    Region_ID
)
```

Parameter	Type	Description
Request_ID	UID	Identifier of the request
X	INT	Request for the Top X most popular content in the requested region
Region_ID	UID	Identifier of the considered region (i.e. belonging to one MAR)

Table 9: DM_AM_ Popularity.Request parameter list

When generated

The DM is periodically requesting an update to the most popular content for each region.

Effect on receipt

Upon receipt, the AM is generating a database report for the Top X most popular content in the requested region and is returning this report to the DM in a DM_AM_Popularity.Response message.

A.4.1.2 DM_AM_Popularity.Response

Function

This service primitive is used to send a list of the Top X most popular content in a certain region to the DM.

Semantics of the service primitive

```
DM_AM_Popularity.Response(
    Request_ID,
    ReturnCode,
    Popularity
)
```

Parameter	Type	Description
Request_ID	UID	Identifier of the related request
ReturnCode	UINT_8	ReturnCode of the required operation

Popularity	List{
	Rank,
	Popularity
	}
		List of the Top X most popular content in the considered region sorted by rank

Table 10: DM_AM_Popularity.Response parameter list

When generated

After the database report for the Top X most popular content in the requested region was successfully generated, this message is returning the database report to the DM.

Effect on receipt

Upon the receipt of this message, the DM triggers the CDN algorithm to determine whether the cached content in one of the CDN nodes should be updated. If yes, a DM_CDNNC_CDNUpdate.Request message is sent to these CDN nodes.

A.4.1.3 DM_AM_Popularity.Update

Function

This message is used to update the popularity database at the application monitoring module (AM). It is sent every time a new session is started.

Semantics of the service primitive

```
DM_AM_Popularity.Update(
    Content_ID,
    Region_ID
)
```

Parameter	Type	Description
Content_ID	UID	Unique identifier of the content (as defined by the Video Service Portal)
Region_ID	UID	Identifier of the considered region

Table 11: DM_AM_Popularity.Update parameter list

When generated

This message is generated each time a new video session is started.

Effect on receipt

Upon receipt of this message, the AM updates its popularity database.

B. Annex - Metrics needed for cache selection

This section classifies the set of parameters that need to be considered to select the cache from which to download content. Beside each parameter, the ALTO Cost metric that is suitable to report it is given. The ALTO metric 'routingcost' is specified in the base ALTO protocol. Other like 'hopcount', are mentioned there. "'EPmemory', 'EPoccupationcost' and 'Pathoccupationcost', are proposed in the "ALTO Multi-Cost" draft. The metrics for which no ALTO correspondent is found will be updated in a local CDN specific database and possibly considered in further implementations of NEGOCODE optimization functions.

Class: Distance, network proximity

- Number of traversed AS ➔ ALTO hopcount (AS hops in network hop units)
- Number of network hops (static information) ➔ ALTO hopcount
- RTT, delay, jitter (dynamic information) ➔ ALTO hopcount (default loose report).
    - Note that ALTO is not meant to provide any real-time performance information.

Class: Bandwidth, throughput

- Capabilities: Cache capacity ➔ ALTO EP memory
- Cache load ➔ ALTO EPoccupationcost (EPOC)
- Path to cache load ➔ ALTO Pathoccupationcost

Class: Monetary and Administrative cost

- Costs for accessing an (external) cache ➔ ALTO routingcost obtained via the ALTO endpoint cost service.
    - Note that ALTO routingcost is generic so can be a monetary cost.
- Transit costs ➔ ALTO routingcost"between PIDs, obtained via the ALTO Cost Map Service (optionally Filtered)
- Operation and maintenance costs ➔ ALTO 'routingcost'
    - A metric called e.g. 'endpointcost' should be proposed in ALTO

Class: "Green IT" aspects:

- "load concentration" in order to be able to switch off some caches during nighttime, instead of running all caches at e.g. 10% load. Night need only 1/5 storage farms. Switch off cluster nodes (machines). Caches associated to business + homes. ➔ Cached ALTO cost values or scheduled ALTO cost values.

Class: Load ➔ EPoccupationcost / "Pathoccupationcost"

For load balancing aspects.

- Load of the cache server ➔ ALTO EPoccupationcost
- Load of related hardware,
- Load of the whole data path ➔ ALTO pathoccupationcost

- Available cache processing and bandwidth ➔ ALTO EPmemory (loose report) – ALTO EPoccupationcost
- Cache service rate to be able to predict future load

Class: Availability

- In particular when dealing with edge caches in P2P networks, trying to contact a cache which is no longer available would lead to high latency due to timeouts.

Class: Content information = application specific QoE metrics requested

- Whether the user requests a small low quality video or a huge HD movie might influence the decision taking, i.e. for short videos (short sessions) the delay is more important, whether for large videos (long sessions) e.g. load balancing might be more important.

➔ the set of cache selection metrics is decided by the application client in the Decision Module, that also maps the metrics with the ALTO Cost Types available in the Serving ALTO Server.

➔ there is a need to have an implemented correspondence table between QoE requirements, QoE metrics and ALTO Cost Types.

Class: QoE/user experience – related ➔ RELIABILITY (see early multi-cost)

- Session start-up duration (waiting 10 seconds when watch 90 minute video ok but too long for a short one, or inconveniences are OK for MT but not at home with fixed equipt). Ex: when start video. Cache change during the session == due to change of weight: in beginning needs be close, in the end BW/load is more important.
- Delay + jitter ➔ ALTO hopcount

E.g. later on, the cache selection could for large videos first select a close cache (short start-up delay) and then switch to a less loaded cache while the user is watching the video.

Class: Multicast – related

- Availability of multicast: A complex selection algorithm could decide that a user should be added to an existing multicast stream instead of sending the video unicast to the user. Of course this is only applicable if the time offset between the (already ongoing) multicast stream and the new request is still small. The missed video data could be transmitted from a close cache via unicast; later video data will be received from the multicast tree.

If we consider multicast, then the decision is not only about optimal cache selection but also selecting among (multiple) multicast trees and starting a new unicast/multicast stream.

Acknowledgements and Disclaimer

This work was partially funded by the European Commission within the 7th Framework Program in the context of the ICT project MEDIEVAL (Grant Agreement No. 258053). The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the MEDIEVAL project or the European Commission.

References

[1] Akamai, Investor Summit, 2011. http://www.akamai.com/dl/investors/akamai_ir_summit_2011.pdf

[2] C. J. Bernardos, A. de la Oliva, J. C. Zuniga, and T. Melia, "Applicability Statement on Link Layer implementation/Logical Interface over Multiple Physical Interfaces". Internet Engineering Task Force, draft-bernardos-netext-ll-statement-01.txt (work-in-progress), March 2010.

[3] E. Amir, S. McCanne, and H. Zhang, an Application Level Video Gateway, In Proc. ACM Multimedia 1995.

[4] S. Randriamasy, "Provider Confidential ALTO with Relays", draft-randriamasy-alto-relay-01, April 15, 2011.

[5] S. Randriamasy, "Multi-Cost ALTO", draft-randriamasy-alto-multi-cost-02, March 2011.

[6] G. Carle and E. W. Biersack, "Survey of error recovery techniques for IP-based audio-visual multicast applications", IEEE Network, vol. 11, pages 24-36, 1997

[7] A. Shokrollahi, "Raptor codes," IEEE/ACM Transactions on Networking (TON), vol. 14, pp. 2551–2567, 2006.

[8] DVB: the global standard for digital television. [Online]. Available: http://www.dvb.org/

[9] A. Naghdinezhad, M. R. Hashemi, and O. Fatemi, "A novel adaptive unequal error protection method for scalable video over wireless networks," in Proceedings of IEEE International Symposium on Consumer Electronics, 2007.

[10] H. Mansour, P. Nasiopoulos, and V. Krishnamurthy, "Joint media-channel aware unequal error protection for wireless scalable video streaming," in Proceedings of IEEE ICASSP, 2008

[11] D. Munaretto, D. Jurca, and J. Widmer, "A resource allocation framework for scalable video broadcast in cellular networks". Springer Mobile Networks and Applications, 2011

[12] D. Kaye, "Strategies for Web Hosting and Manages Services", John Wiley & Sons, 2002

[13] M. Rabinovich and O. Spatsheck, "Web Caching and Replication", Addison Wesley, 2002

[14] I. Lazar and W. Terill, "Exploring Content Delivery Network", IEEE IT Professional, vol. 3, no. 4, 2001, pp. 47-49

[15] A. Vakali and G. Pallis, "Content Delivery Networks: Status and Trends", IEEE Computer Society, 2003

[16] S. Borst, V. Gupta, and A. Walid, "Distributed Caching Algorithms for Content Distribution Networks", IEEE Computer Society, 2010

[17] ALTO Status Pages, http://tools.ietf.org/wg/alto/

[18] N. Amram, B. Fu, G. Kunzmann, T. Melia, D. Munaretto, S. Randriamasy, B. Sayadi, J. Widmer, M. Zorzi, "QoE-based Transport Optimization for Video Delivery over Next Generation Cellular Networks", accepted for MediaWiN workshop 2011

[19] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in Proc. Int. Workshop Peer-to-Peer Syst., 2002

[20] A. G. Dimakis, K. Ramchandran, Y. Wu, C. Suh, "A Survey on Network Codes for Distributed Storage", Proceedings of the IEEE, Vol 99, No 3, March 2011.

[21] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity", IEEE Trans. Image Process., vol. 13, no. 4, pp. 600-612, Apr. 2004.

[22] M. Pinson, S. Wolf, "A new standardized method for objectively measuring video quality", IEEE Trans. Broadcast., vol. 50, no. 3, pp. 312-322, Sep. 2004.

[23] Huifang Sun, Xuemin Chen, and Tihao Chiang, "Digital Video Transcoding for Transmission and Storage", New York, CRC Press, 2005.

[24] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of H.264/AVC," IEEE Trans. Circuits Syst. Video Technology, vol. 17, no. 9, pp. 560–576, 2003.

[25] Joint Video Team of ITU-T and ISO/IEC JTC 1, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)," document JVT-G050r1, May 2003; technical corrigendum 1 documents JVT-K050r1 (non-integrated form) and JVT-K051r1 (integrated form), March 2004; and Fidelity Range Extensions documents JVT-L047 (non-integrated form) and JVT-L050 (integrated form), July 2004.

[26] Cisco, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2010–2015", http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf, 2011.

[27] European Commission FP7 Project: "MEDIEVAL: MultiMEDia transport for mobIlE Video AppLications", http://www.ict-MEDIEVAL.eu/, retrieved June 2011

[28] MEDIEVAL Project, Deliverable D1.1 – "Preliminary architecture design", June 2011

[29] MEDIEVAL Project, Deliverable D1.3 – "Final architecture design", will be published in December 2012

[30] MEDIEVAL Project, Deliverable D2.1 – "Requirements for video service control", June 2011

[31] MEDIEVAL Project, Deliverable D3.1 – "Concepts for Wireless Access in relation to cross-layer optimisation", June 2011

[32] MEDIEVAL Project, Deliverable D4.1 – "Light IP Mobility architecture for Video Services: initial architecture", June 2011

[33] MEDIEVAL Project, Deliverable D4.2 – "IP Multicast Mobility Solutions for Video Services", June 2011

[34] MEDIEVAL Project, Deliverable D5.1 – "Transport Optimisation: initial architecture", June 2011

[35] MEDIEVAL Project, Deliverable D5.2 – "Final specification for transport optimisation components & interfaces", June 2012

[36] MEDIEVAL Project, Deliverable D6.3 – "First Periodic Testing Report", June 2012

[37] J. Kangasharju, J. Roberts, and K.W. Ross, "Object Replication Strategies in Content Distribution Networks", Computer Communications, Vol. 25, No. 4, pages 367-383, 2001

[38] M. Cha, H. Kwak, P. Rodriguez, Y.Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system", Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC '07), ACM, 2007

[39] K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload", Proceedings of the 19th ACM symposium on Operating systems principles (SOSP '03), ACM, pages 314—329, 2003

[40] "Multi-Cost ALTO", S. Randriamasy (Editor), N. Schwan, October 31, 2011, IETF ALTO personal draft draft-randriamasy-alto-multi-cost-06, March 12, 2012, Presented at the 83rd IETF, Paris (March 2012), http://tools.ietf.org/id/draft-randriamasy-alto-multi-cost-06.txt

[41] T. Melia, S. Randriamasy, D. Munaretto, M. Zorzi. "QoE optimisation with network layer awareness on hybrid wireless network", Next Generation Service Delivery Platforms (NG SDP), GI/ITG Workshop, October 2011.

[42] M. Kim, D. Lucani, X. Shi, F. Zhao, and M. Medard, "Network coding for multi-resolution multicast," in IEEE Infocom, San Diego, CA, Mar. 2010.

[43] P. Gill, M. Arlitt, Z. Li, A. Mahanti. "YouTube traffic characterization: a view from the edge", Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC'07), ACM, 2007

[44] A. Falaschi, F. Fiumana, M. Krsek, E. Verharen, C. Sganga, Report on live-streaming infrastructure, February 2004 (http://labtel.ing.uniroma1.it/opencdn/report_on_live_streaming_infrastructure.pdf)

[45] J. Widmer, A. Capalbo, A.F. Anta, A. Banchs. "Rate Allocation for Layered Multicast Streaming with Inter-Layer Network Coding", in Proc. IEEE Infocom (mini conference track), Orlando, FL, July 2012

[46] D. Munaretto, T. Melia, S. Randriamasy, and M. Zorzi, "Online path selection for video delivery over cellular networks", submitted to IEEE Globecom 2012.