Project Number:	**Contract Number: INFSO-ICT-*224287***
Project acronym:	**VITAL++**
Project Title:	**Embedding P2P Technology in Next Generation Networks: A New Communication Paradigm & Experimentation Infrastructure**
Title of Report	**Design of the overlay network architecture components**

Instrument:	STREP
Theme:	ICT-2-1.6
Report Due:	M21 (Feb.2010)
Report Delivered:	
Lead Contractor for this deliverable:	FOKUS
Contributors to this deliverable:	Jens Fiedler (FOKUS), Julius Müller (FOKUS), Stephen Garvey (WIT), Shane Dempsey (WIT), Kostas Koutsopoulos (BCT), Nikolaos Efthymiopoulos(UoP), Kostas Papanikitas (UoP), Evangelos Markakis (CTRC), Evangelos Pallis (CTRC), Argiris Sideris (CTRC), George Mastorakis (CTRC).
Estimated person Months:	25
Start date of project:	1st June 2008
Project duration	30 months
Revision:	Version 1.9
Dissemination Level:	PU - Public
Internal reviewer:	TA

This page has been left blank intentionally.

1 Table of Contents

2 List of Figures

3 Executive summary

The VITAL++ projects goal is to create a service platform, which allows the secure distribution of P2P multimedia content and community services, using the benefits of a centralistic architecture like the IP multimedia subsystem (IMS).

This deliverable describes the platform elements in terms of four sub-architectures for P2P-Authentication, Content Indexing, Content Security and Overlay Management. Each of these sub-architectures spans across the client and over a part of the IMS. The IMS component developed is an application server, holding the IMS part of each sub-architecture.

The counterpart to the application server is a user-controlled client holding the client part of each sub-architecture. Two clients are being developed. This ensures the re-usability of client components, which as a design element helps to ensure the interoperability with clients of different vendors, which may want to enable their clients with Vital++-support.

The P2P-Authentication sub-architecture uses a certificate-based message exchange in order to sign and verify P2P messages. The Content Indexing sub-architecture is used by clients in order to publish and discover content including meta-data. The Overlay management sub-architecture's purpose is to build and maintain performing P2P overlays and the Content Security sub-architecture realizes a digital rights management (DRM) in the IMS.

Also, the P2P media exchange schemes are depicted and explained. They have to address the different multimedia distribution models (Live-TV, Video-on-Demand and File sharing), which differ in their requirements for jitter, real-time and necessary bandwidth.

As further improvements and future design goals, a client-based DHT and support for the network attachment sub-system (NASS) are proposed.

This deliverable is complemented by the deliverable 3.2, which describes interaction between the elements and simple operations needed in order to build services on top.

4 Introduction

The VITAL++ projects main challenge is to combine the benefits from P2P media distribution and IMS control plane in order to build secure, effective multimedia and community services. In the deliverables 2.2 and 2.3, the essential elements of a related architecture have been outlined, which are the basis for the work in work-package 3.

The purpose of this deliverable 3.1 is to describe the functional blocks and network protocols, i.e. the elements of the platform, while the deliverable 3.2 describes the integration of these functional blocks into service building entities needed to build converged VITAL++ services, which are then described in WP4.

These envisaged aims were mainly

- Secure and authentic distribution of multimedia streams (i.e. audio and video)
- Scalable real-time multimedia delivery (from ”Live” down to ”Offline”).
- Network-topology optimized overlays to reduce network traffic.
- Enable Secure P2P Message exchange for distributed community services.

One challenge was to minimize the deployment of new network nodes in order to achieve the envisaged aims. In order to minimize deployment efforts, two main construction sites have been identified, which are the client, which is under the control of a potentially malicious user and a trusted server component in the realm of the IMS walled garden, the VITAL++ application server.

Thus, the scope of this deliverable is the description of the core elements of the VITAL++ system. This deliverable is complemented by the deliverable 3.2 to describe the full VITAL++ service platform.

The deliverable is outlined as follows. Chapter 5 analyses the requirements for the VITAL++ architecture and its components. Chapter 6 gives an overview of the overall architecture, including all components from the P2P and IMS parts, as well as newly added components. In chapter 7, all developed functionalities are described in terms of purpose, implementation, network protocols and message flows. Chapter 8 draws conclusions.

5 Requirements analysis

In this chapter, all the requirements of VITAL++ project are presented. The required features of the VITAL++ project are re-introduced. The three scenarios, content integrity, content security, topology awareness and additional functionalities for the project VITAL++ are presented.

5.1 Required Features

The required features of VITAL++ are content distribution, live streaming and video on demand.

In content distribution, a user who wants to transfer a file to a vast number of peers is using his P2P client in order to divide the file into different blocks. These blocks are transmitted to the end users. The end users will download this block and concurrently transmit them back to various end users. In order an end user to download a whole file all he need to do is to merge the various downloaded blocks. These blocks are concurrently received and transmitted in the whole infrastructure. These blocks distribution initiates the end user to utilize the network infrastructure.

Major drawbacks of content distributions are content bottlenecks. When the peers do not have different blocks to exchange and their upload bandwidth remains idle a content bottleneck is created. In this aspect VITAL++ proposes efficient algorithms that take into account various network requirements such as round trip time, bandwidth and P2P localization. More specifically, for these requirements the main algorithm implemented by VITAL++ is created by two overlays. The first overlay called base overlay contains the nodes that enter the system. The base overlay is a symmetric graph, where adjacent nodes exchange buffers and blocks in both directions. The second overlay called top overlay contains all the users that have better upload bandwidth. By achieving a better localization inside the VITAL++ test bed we are able to provide a faster content contribution In this way, the VITAL++ P2P client will be able to provide content distribution faster than common P2P clients.

In peer-assisted video on demand (VoD), the same peers that are viewing the publisher's videos distribute the content. In this way, the publishers can dramatically reduce the bandwidth cost since peer-assisted VoD can move a significant fraction of the uploading from the server to unused resources (e.g. upload bandwidth) of the peers. More specifically the bootstrap node (source of video) divides the video stream into blocks. The block size will depend from the service rate and the number of blocks into which the bootstrap node divides one second of video playback. Every block is associated with a time stamp indicating the time of its generation. All peers reproduce (play) the video with a delay called set-up time. The VITAL++ consortium main concerns are to create a mechanism that will determine which block should be sent to the

selected node aiming at minimizing duplicate block transmissions and fast diffusion of newly produced or rare blocks within an overlay "neighbourhood".

P2P live streaming is a real time application with strict delivery time constraints and very demanding in terms of the aggregate bandwidth required for the delivery of the stream to the participating peers. In general, a server generates a video stream at a given service rate which is then divided into blocks followed by their delivery to a small subset among the participating peers. As a final step, all peers exchange these blocks in order to reproduce the video stream.

An efficient P2P streaming system must be able to deliver a video stream with the smallest possible delay, called *setup time*. With the term setup time, we define the time interval between the generation of a block from the origin server and its distribution to every peer in the system. It is obvious that the main consideration of the VITAL++ consortium is to provide an algorithm for discovering of peers and for the interchange of media information. The VITAL++ project also takes into consideration various requirements such as Content integrity, Content security, Topology awareness, etc.

The content integrity is achieved be employing a distributed database that will focus on the creation of a hybrid architecture where all participating peers will enter a DHT in order to have scalable queries. This centralized architecture will manage the information in the DHT in order to increase reliability. The Content Security Sub-Architecture prohibits illegal content access. The content will be encrypted providing only to authorize peers in the IMS domain to be able to decipher and use the related content. In addition, the operator can optionally charge the publication of content as well as the consumption of content using the VITAL++ P2P operator service. Finally VITAL++ will take into consideration the need for an efficiently and scalable address content location ensuring content availability by deploying an index database providing the necessary content availability to the end users.

5.2 Functional – Architecture requirements

Transcoding

The need for transcoding arises from the fact that the bit rate requirement varies from channel to channel because of vastness in the compression standards in use. The adaptation process is distributed between different nodes and a central application that will distribute conveniently the requests to the nodes. For this process, an internal algorithm is developed internally to assure the best job charge to the different nodes.

User profiles

The IMS holds a profile for each user in the home subscriber server (HSS), a highly scalable central database. Additional information from the authentication phase (like network identifiers) can be stored there and used for P2P overlay

planning, e.g. to optimize data paths in the overlays, which is extremely important for streaming overlays (e.g. for the Live-TV scenario). In addition, the data concerning users (e.g. public/private key pairs) can be stored in that database.

DRM

Content must be secured from unauthorized acquisition and copying (assume trusted client). The use of Digital Right Management (DRM) will result in a reliable system so that the copyright laws are taken into account and the necessary trust is achieved in the whole VITAL++ test bed. The binding between the IMS profile and the digital rights management could provide the necessary answer for the clients in order to move their files from one computer to another without concerning of Digital right management

QoS

QoS is a requirement that originates from telecommunication networks and has received a lot of attention since the early 1990. VITAL++ project will exploit the adaptation and optimisation capabilities of P2P system and combine them with Virtual networks of IP Multimedia Subsystem (IMS) managing to overcome the "old" IntServ/ DIFFserv QoS paths used in various network. In this way, exploitation of the hybrid P2P-IMS client will allow VITAL++ end-users to experience Quality of Service during their communications.

Acounting

Accounting in communications systems involves the collection and analysis of service and resource usage metrics for purposes such as billing, capacity and trend analysis, cost allocation and auditing.

VITAL++ will investigate the key area of providing metering for dynamically composed ad-hoc services delivered over a P2P overlay.

5.3 Summary

In this chapter we have reflect the requirements analysis for the main functionalities, which form the base of the VITAL++ project in order to group required functionalities and re-introduce the required features of the VITAL++ project.

In the bullets below a summarized version of this chapter is presented:

- Content distribution
  - Fully utilize the network infrastructure.
- Video On Demand
  - Minimizing duplicate block transmissions and fast diffusion.
- Live Streaming
  - Provide an algorithm for discovering of peers and for the

interchange of media information.

- Content integrity.
  - Uninterrupted service functionality (QoS).
- Content security
  - Need for an authentication system.
  - Digital Rights Managements
- Topology awareness.
  - Need for an efficiently and scalable address content location ensuring content availability.

6 System Architecture

6.1 Overview

The VITAL++ overall architecture is basically distributed over the Client and the NGN/IMS area of functionalities. In this chapter we will give an overview over the whole architecture, which will then be discussed in more detail in the relevant sections in chapter 7.

In order to address the VITAL++ challenges, multiple sub-architectures have been defined, which interact among each other. These are the P2P Authentication sub-architecture (P2PA), the Content Index sub-architecture (CI), the Overlay Management sub-architecture (OM) and the Content Security sub-architecture (CS). Each sub-architecture spans over the client, the network and the IMS with its components. Sub-architectures may interact among each other in an arbitrary way, especially in the client, while on the NGN side there need to be well defined interfaces. Thus the media exchange is not entitled as sub-architecture, but it interacts with these and itself in the same as well as in remote clients.

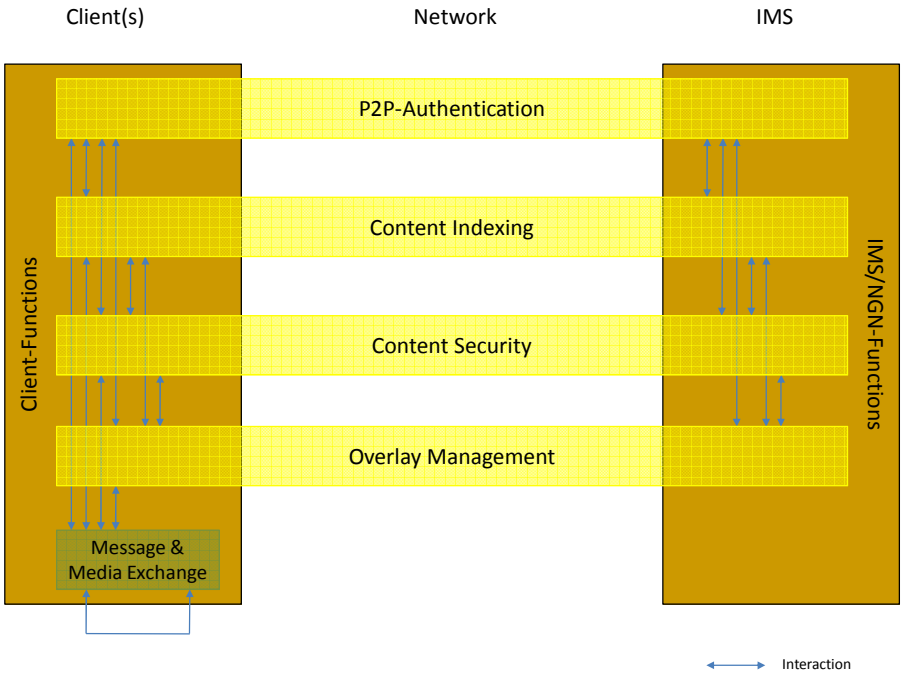


Figure 1: VITAL++ abstract view of the overall architecture

6.2 Client

The terms "Client" and "Peer" are used equivalently in this document as they refer to the same thing. The VITAL++ client is a hybrid client. This means it is an IMS client and a P2P client at the same time. The IMS functionalities are used to mainly interact with an IMS core or system for exchanging control information, while the P2P part is used to exchange content with other peers.

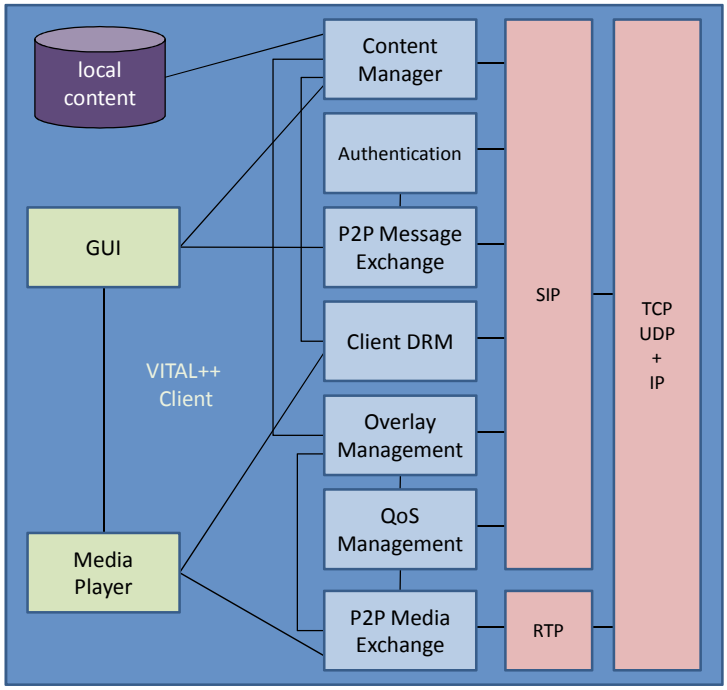


Figure 2: Client functional blocks

The components of the client are directly derived from the necessity to interact with other clients and the IMS core in order to fulfil the envisaged features. Figure 2 illustrates the functional blocks inside the client. These are the content manager, which is responsible for publishing and discovering content as well as triggering DRM operations via the client DRM module if a licence needs to be obtained. The authentication module obtains and manages certificates of VITAL++ entities (clients, application servers, root-certificate). It interacts mainly with the P2P message exchange in order to sign and verify messages. The latter has the purpose to exchange P2P messages with other peers for genric purposes (i.e. playlist exchange, etc.). The overlay management module obtains overlay changes from the application server and re-organizes its neighbourhood accordingly, also to respect to QoS requirements, issued by the QoS management module, which can also realize QoS enforcement via NGN mechanisms. Also standard IMS client functionality is realized (not depicted) for initial IMS registration and IMS session management.

6.3 Platform

The platform side of the architecture consists of four application server entities, which can be co-located in the same box (as depicted), or distributed over several machines. The communication with the client occurs mainly through the IMS core and its call/session control functions (P/I/S-CSCF). Each of the functional blocks in the application server refers to a related sub-architecture.

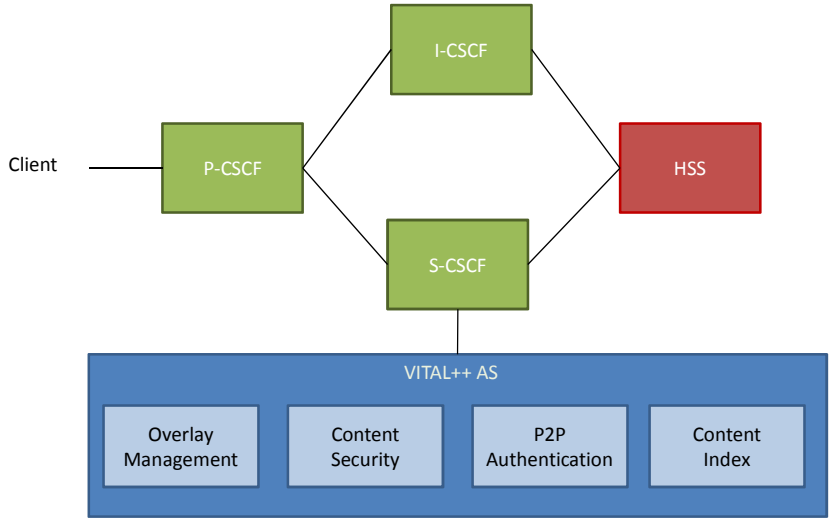


Figure 3: Platform components

Figure 3 depicts the platform components and their relation with other IMS objects. The functional blocks are the

- P2P-Authentication module, which stores client certificates for use by other modules, serves the client with initial credentials and signs the client's certificates on request.
- Content Index module, which stores content descriptions and metadata and provides search functions to the clients.
- Overlay Management module, which constructs and maintains optimised overlays according to the client's connectivity.
- Content Security module, which provides and maintains DRM licenses for published content.

6.4 Summary

In this chapter, an overview has been given to the VITAL++ overall system architecture. The sub-architectures have been introduced, as well as the functional blocks in the client and platform.

Altogether they provide functional services for building higher level transactions and thus services based on the VITAL++ architecture. Due to the complexity of the overall architecture, each sub-architecture and media exchange is explained in detail in chapter 8.

7 Sub-Architectures and Functional Blocks

Related to the previous chapters, the System's sub-architectures are explained in detail. The internal construction of each sub-architecture is explained and the intended usage is discussed. Also the used network protocols are described with a special focus on possible modification and extension.

7.1 P2P Authentication

In this section the P2P Authentication sub-architecture (P2PA) is being described. We describe the concept of p2p certificates, and how they are acquired or generated as well as the basic transactions related to P2P authentication.

7.1.1 Purpose of the P2PA-SA

The purpose of the P2PA-SA is to enable clients (peers) to verify the authenticity of messages which have been sent by other clients directly to them, without passing through any operator controlled entity. This envisages the security of services, which are based on pure P2P message exchange, like e.g. sharing of contacts or media, etc.

7.1.2 General Description

The P2P-Authentication sub-architecture works with certificates, i.e. digitally signed chunks of data, which describe an entity and its properties, e.g. identity and access rights. In the VITAL++ scope, three levels of certificates are distinguished, as shown in the following table.

Root Certificate	Self-signed. Pre-installed in every client and P2P-Authentication server module.
Server Certificate	Signed by Root-CA. Pre-installed in every P2P-Authentication server module. Describes the identity of the server domain and its public key. Acquired by each client during registration.
Client Certificate	Signed by a P2P-Authentication server module on request. Describes the identity of the client and its public key.

Table 1: VITAL++ Certificate Types

Finally, each client is equipped with these three certificates, which allow it to perform all authenticity transactions and checks as explained in then following sub-sections.

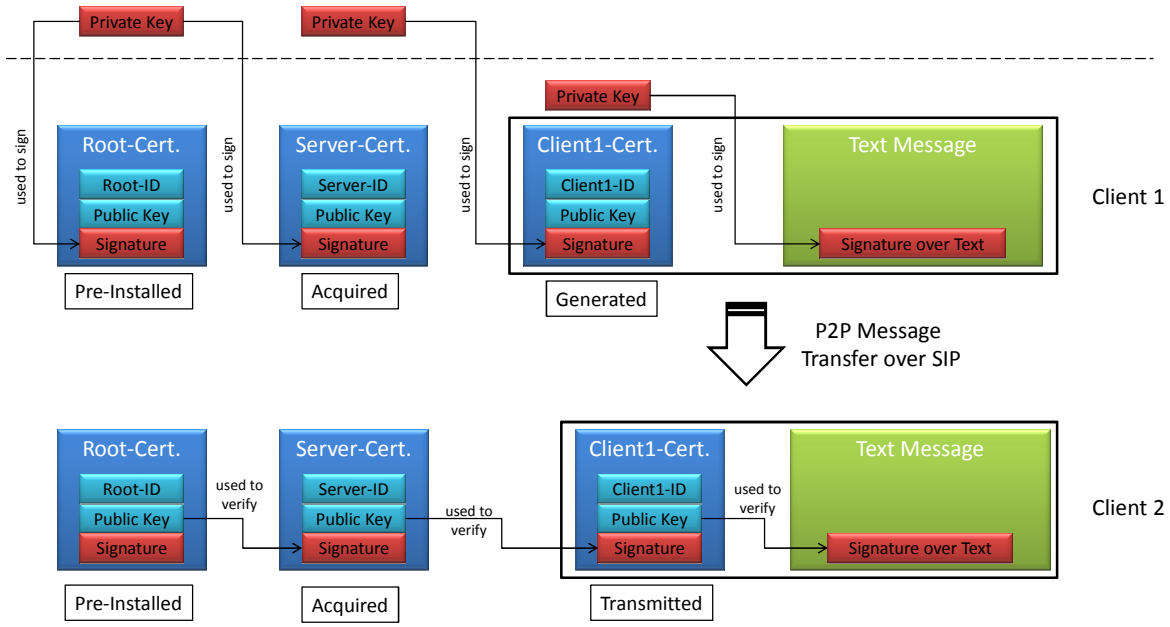


Figure 4: Relation between Certificates and Messages

The relation between the certificates and their use in order to enable authentic message exchange is depicted in Figure 4.

7.1.3 P2PA refined Sub-Architecture

During the progress of the project, two changes have been performed on the P2PA sub-architecture.

1) No support for the DHT.

The original idea was to store client certificates in a DHT, signed by the AS, to make sure that clients can access any other client's certificate to verify its messages. The purpose was to create a distributed database of client certificates. It became clear, that such a distributed storage for client certificates already exists, although it is not organized as a DHT, as every client holds its own client certificate and can send it to an interested peer when requested.

Thus, any peer which needs a foreign client certificate, can either contact that peer directly, or the sending entity sends its client certificate along with its message.

2) Kq interface has been moved directly to the KSF

The purpose of the Kq interface is to allow access to stored client certificates, especially to public keys, to other VITAL++ server components. This access must be restricted to read-only access rights. This is also achievable by moving the Kq interface from the Key management function (KMF) directly to the key storage function (KSF). The KSF is going to be realized using common database software, which provides different access levels.

Figure 5 depicts the new functional layout of the P2PA sub-architecture.

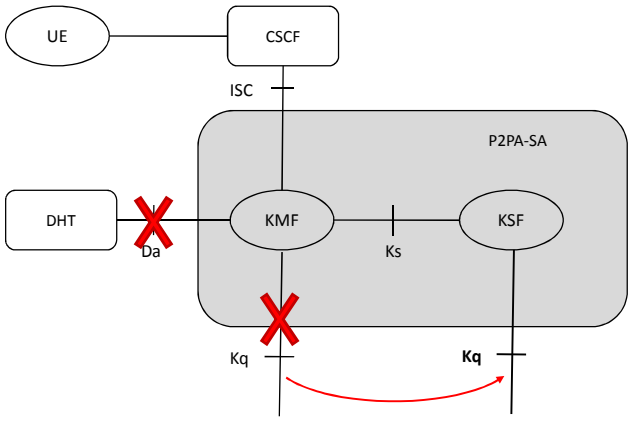


Figure 5: Revised P2PA Sub-Architecture

7.1.4 Transactions and Message Exchange

The functions of the P2P-Authentication sub-architecture are split into client-located functions and server module-located functions which interact during functional transactions. The supported transactions are explained in the following sub-sections section.

In every transaction there is either a certificate or signature being transported between the entities. Both are encoded as XML documents and attached as a MIME multipart message to the corresponding SIP message. Examples for the corresponding message types are given in the annex of this document.

In order to simplify the processing of VITAL++ related SIP messages, a new SIP header field has been introduced, namely the "`Vitalpp`" header field, this is used in P2P-Authentication related SIP transactions.

7.1.4.1 Internal public key retrieval

The public key retrieval is realised through the Kq interface, which is directly attached to the key storage function. The key storage function is realised as a

MySQL[1] database server. Thus, the exact access method in terms of username, password and table descriptions are currently subject to the local deployment.

7.1.4.2 Initial certificate provision

Whenever a client registers to the IMS-core, a 3rd party registration is performed towards the VITAL++-Application Server (VITAL++-AS), i.e. a copy of the SIP REGISTER request is being sent to the application server in order to notify about the registration. This means that the Vital++-AS is informed about every registration of a user for whom the Vital++ feature has been enabled in his IMS-profile. The P2P-Authentication module in the VITAL++-AS will process the registration hint and supply the newly registered user with a certificate, which is signed by the common Root-CA. This certificate will be referred to as "Server-Certificate", as it holds the public part of the key pair of the server, which it uses to sign client-certificates (s.a.). The following picture shows the message exchange between the client and the VITAL++-AS for this transaction.

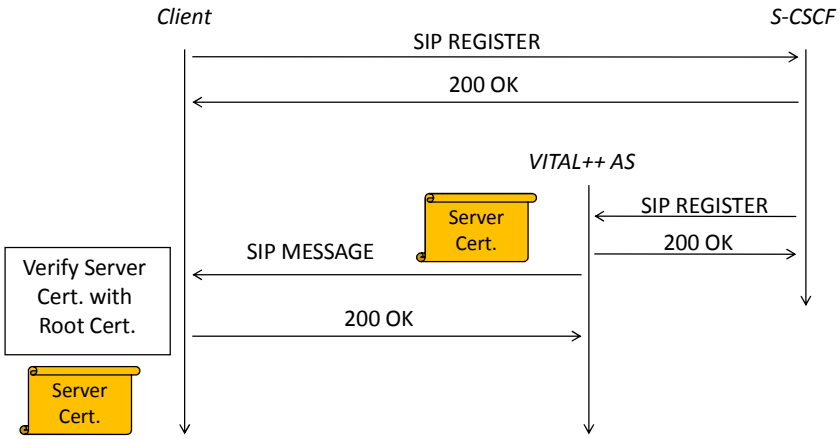


Figure 6: Initial server certificate acquisition

Figure 6 illustrates the message flow between the client, the application server and the call/session control of the IMS-core. After this transaction, which occurs only once per client and registration lifetime, the client can verify signatures of messages issued to it by other peers, which have signed their messages. It cannot sign messages itself yet; this requires the next step, which is being described in the following section. An example message can be seen in the appendix 9.3.1.

[1] http://www.mysql.com

The `Vitalpp` header field here has the value "`P2PA OfferCert`", addressing the P2P-Authentication part and triggering the function for receiving a certificate.

7.1.4.3 Client certificate authorization

The process of authorizing the client certificate serves the purpose of supplying the client with a valid personal certificate which he can use to create authentic messages, as depicted in **Figure 4**. The basic message flow is depicted in the following picture.

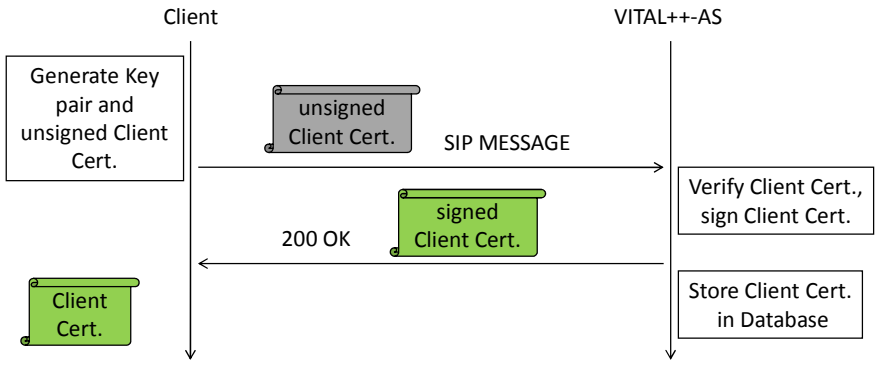


Figure 7: Client certificate exchange

The client hereby generates its personal private-public key-pair and creates an unsigned certificate with its identity and public key. This is then being sent to the VITAL++-AS, which checks the identity and other fields of the certificate before he signs it with his private server key. The signed certificate is then being sent back to the client, which stores it as its own personal certificate. After performing this transaction, the client owns a valid certificate, signed by the application server and therefore verifiable by every instance, which also knows the server certificate. Example messages can be seen in the annex 9.3.2.

The `Vitalpp` header field here has the value "`P2PA RequestSigning`", addressing the P2P-Authentication part and triggering the function for signing a certificate.

As this process is vulnerable against a man-in-the-middle attack, it is advised to encrypt this transaction. For that purpose, we suggest a Diffie-Hellman key agreement transaction before the main transaction in order to establish a common secret knowledge, which is then used to generate a symmetric key for message encryption. The corresponding message flow is depicted in the following illustration.

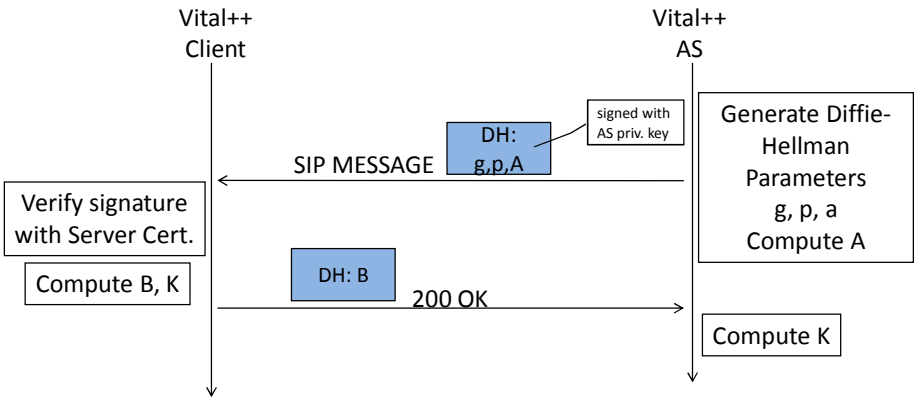


Figure 8: Diffie-Hellman Key Agreement

Here, the Vital++-AS generates the Diffie-Hellman parameters *g,p* and *A*, which he signs with his server key before he sends them to the client. This then computes the parameters *B* and *K,* and sends *B* back to the server, which completes his own computation of *K*. So both entities have the same secret knowledge *K*, which can be used for further encryption of the certificate exchange, explained before. The Diffie-Hellman algorithm is explained in more detail in the annex 9.3.4.

7.1.4.4 Client-to-client certificate retrieval

Although a client can always send its certificate along with the corresponding signed message, this will not be necessary when the receiving peer already has this certificate. As the sender does not know whether the receiver has his certificate, he can either send it along with the first message and safely assume that for subsequent messages it will not be necessary to send the certificate, or he simply does not send the certificate by its own initiative, but sends the certificate only if the receiver sends a request back which demands the certificate. This is an optimization of the message flow and the P2PA security mechanisms will also work without this transaction. The message flow for this transaction is depicted in Figure 9.

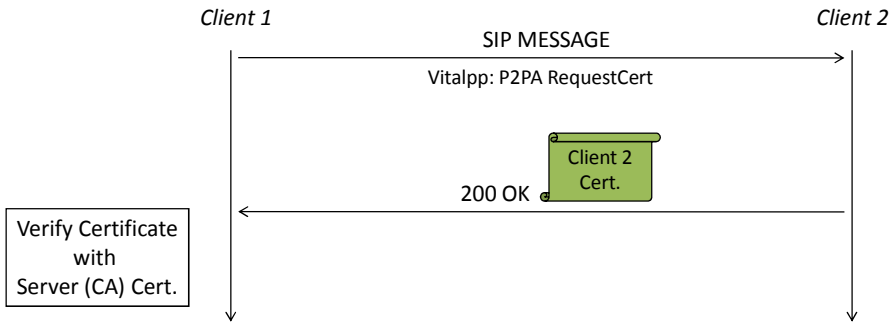


Figure 9: Request for certificate

Here, the sender creates SIP request which triggers the receiver to send its own client certificate along with the SIP response. When he receives the response, the original requestor verifies the certificate with the server (CA) certificate.

The `VITALPP` header field here has the value "`P2PA RequestCert`", addressing the P2P-Authentication part and triggering the function for sending the client certificate.

7.1.4.5 Client-to-client Message authentication

In this section, we describe, how messages need to look like if they carry a signed (authentic) message/information. In order to work, the sending client needs to have its full client certificate acquired and the receiver needs the corresponding server certificate. The message flow is depicted in Figure 10.

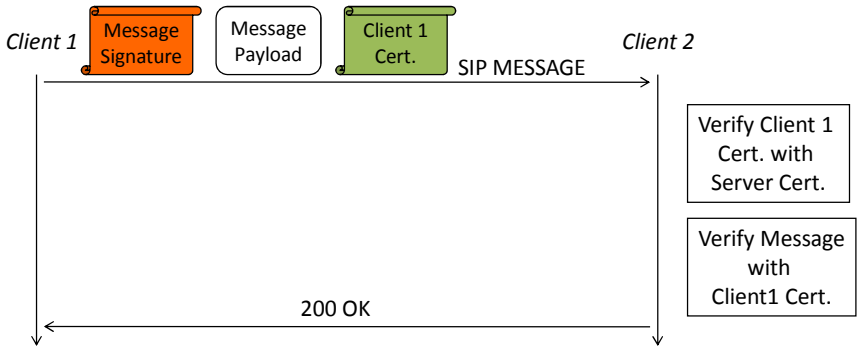


Figure 10: Authentic Message Exchange

Here, the sender creates a text message, which he signs with his private key, which corresponds to its own client certificate. He then sends the text message along with its own client certificate and the message signature to the receiver. This one can then first check the authenticity of the client certificate using its server certificate, followed by checking the message

signature with the public key from the client certificate and inform the user accordingly. An example message is depicted in the annex 9.3.3.

The `Vitalpp` header field here has the value "`P2PA message`", addressing the P2P-Authentication part and triggering the function for receiving a signed P2P message.

7.1.5 Description of the Software

The software is realized on the IMS side of the architecture using the Sailfin/Glassfish converged application server from Sun with the P2P-Authentication servlet, which has been developed in the scope of this project. The servlet is compiled from several Java classes which are in part also used by the client to perform P2P authentication tasks. The whole bundle is structured into several packages in order to build logical blocks. The following table gives an overview over the packages and their usage in the several client and server components.

vitalpp.as	In this package, classes which serve the purpose of attaching to the application server container are realized. These are used to serve the SIP Servlet API 1.1, provided by Sailfin.
vitalpp.as.p2pa	This package provides the main functionality of the server side of the P2P-Authentication sub-architecture. All certificate signing and provision functions are in this package.
vitalpp.client.P2PA	All common classes reside here, as they are also needed by the client, e.g. the VitalppCertificate class.
vitalpp.test	Functions for testing and certificate generation, like root- and server certificates and the corresponding key-pairs.

Table 2: P2P-Authentication package overview

7.2 Content Index

7.2.1 Purpose of the SA

The Content Indexing Sub-Architecture (CI-SA) has been introduced to offer the following services to the Vital++ users:

- Content Publication
- Content Discovery
- Overlay Bootstrapping/Maintenance

Provision of these services involves interactions with the other sub-architectures in the context of the integrated operation of Vital++ Application

Server. In this way the realisation of the control plane services on top of IMS follows the principles that are posed by the requirements.

Implemented as a SIP Instant Messaging based service, CI-SA tries to hide all the complexity with respect to other communication protocols and formats that are required for the proper management of the information that is published and stored in the Application Server or processed, refined and sent to the Vital++ clients.

Content Publication

Registered IMS users willing to offer content that can be transferred by use of the underlying Vital++ P2P techniques can send to the CI-SA a description of the specific content item. In this way the client software declares its availability to act as the original content source in order to feed any overlay that is created to serve the transfer of the specific item. The publication information is indented to be used as the main input during searching and as such it must be complete enough so that queries can be executed against it.

Content Searching

This is a complementary service to Content Publication. Users willing to locate specific media items define certain criteria that are relevant to the publication parameters. The criteria are submitted to the CI-SA for processing. The result of the processing which is a search against the published information is sent to the requesting users as a list of descriptions of available content items. This list contains the actual descriptions along with matching factors acting as indicators to present the relevance of the result.

Overlay Bootstrapping/Maintenance

Contrary to the regular session establishment in IMS, where connection parameters are negotiated during set-up, the Vital++ client software has to join P2P overlays in order to be able to acquire the content. For this purpose, once the user has selected a specific content item to be retrieved and reproduced locally, this has to be communicated to the CI-SA. In this case the CI-SA interacts with the Overlay Management SA (OM-SA) in order to either create a new overlay or to update an existing one. In any case the outcome of the OM-SA, which is a list of peers per overlay member, is sent either to a newly added member of an overlay or to an existing member for which the list of its peers has been updated.

7.2.2 Description of functions

The services offered by the CI-SA are realised by the invocation of certain functions either at the SA side or at the client side. These functions are associated with an appropriate application context.

Function	Direction	Application Context
Publication	Client -> CI-SA	Provision of the details of an offered content item. These details are intended

		to be stored as an entry in a relevant repository so that they can be used for searching.
Publication-Modification	Client -> CI- SA	Update of the details of a previously published item.
Publication-Removal	Client -> CI- SA	Removal of a previously published item
Query	Client -> CI- SA	Conveys the parameters according which the SA should search for offered items.
Query-Result	CI-SA -> Client	Conveys the result of the query..
Content-Selection	Client -> CI- SA	Declares the willingness of a client to join an overlay through which it can receive the selected content item. The client's network address and port number that can be used in the overlay are also provided.
Peer-List	CI-SA -> Client	Conveys a list with networking addresses and port numbers that have to be configured for bootstrapping the P2P engine at the client-side so that it can join an overlay and start receiving the selected content.
Located-Content-Update	CI-SA -> Client	Conveys any modification to content items that have been already communicated to certain clients as a result of a query and for which the client has issued a selection request.
Peer-List-Update	CI-SA -> Client	Addition of newly arrived peers in existing overlays.

7.2.3 Message Exchange and Transactions

7.2.3.1 Message Content and Format

All the messages between clients and CI-SA are transferred in the body of SIP Instant Messages. The actual content stored therein depends on the actual function intended to be invoked by the specific message. Since IMs do not necessitate the existence of a SIP session, states and associations should be maintained by both the SA and the client software.

Currently, for all the transactions between clients and CI-SA for the realisation of the above mentioned functions an XML schema (Annex 1 - Content Indexing XML Schema) has been created. This XML schema caters for the composition of all the possible messages that have to be embodied in the SIP IMs to and from the CI-SA as XML documents.

A visual representation of this schema is provided in the following figure:

Figure 11: CI-SA XML Schema

The root element that is always sent in either direction is the *message* element. For this element an attribute is always provided to indicate the function to be invoked. The valid values for the attribute are the following:

- publication
- publication-successful
- publication-failure
- publication-modify
- publication-remove
- query
- query-result
- content-selection
- peer-list
- peer-list-update
- located-content-update

The root element's body contains a sequence of *content-item* elements that in turn contains a sequence of one exactly *content* element with zero or more *peer* elements. Each *content* element may contain a locator that is a sequence of fields (provider, programme, category, subcategory, series, episode) and a list of keywords that can be used for searching and additional information fileds. The *peer* element is an association of an IP address and a port number.

Depending on the *message* attribute value the following principles apply.

publication Any number of *content-item* elements can be included. Each of these **must** contain one exactly *peer* element to indicate the address-port pair of the P2P Engine through which the published item will be provided when the publisher will be involved in the overlay. The fields of the *content* element should be completed sufficiently so as to enable the CI-SA execute more accurate queries against the published items. The identifier **must** be also provided.

publication-successful It is sent from the CI-SA to acknowledge a successful publication. The identifier must be present.

publication-failure It is sent from the CI-SA to indicate a failed publication. The identifier must be present. The *description* element of the *content* can be used for providing the reason.

publication-modify The same with the publication. All modified fields **should** be resent; absence of a field does not modify any stored value of it. The identifier **must** be also included.

publication-remove	Any number of *content-item* elements can be included. Only the identifier is meaningful, the rest can be omitted.
query	No *peer* elements are required, if any of these are present, they will be ignored. Any number of *content-items* can be present. If through the enumeration of the *content* elements more than one value are detected for a specific field, all the values will be considered as alternatives (OR). More precisely, the query is executed on the basis of the enumerated locators and keywords. If more than one locator is detected, same fields are combined by OR while different fields are conditions that combined by AND operator (see Annex 2 – XPath Query Example).
query-result	After executing the query, the CI-SA will return back to the client a *message* containing any number of *content-items*. The identifiers **must** be included, and all the other fields **must** be also copied from the original publication message.
content-selection	Once the user has decided on joining an overlay for acquiring a specific content item, the client should send a *message* containing one at least *content-item* (more than one can be sent if there is such a request from the user) **including** the correct identifier (the rest fields can be omitted) and also **one** exactly address-port pair in the *peer* element to indicate the settings of the P2P Engine instance that will join the overlay.
peer-list	After interaction with the OM-SA, the CI-SA will send back to the client a *message* containing the *content-items* **with** identifiers along with a number of *peers* that will constitute the neighborhood of the specific client.
peer-list-update	The same as with peer-list. An up to date list of neighbours will be sent to indicate newly added or removed members of the overlay.
located-content-update	This is the same as with the query-result. It is intended to indicate any modifications occurred to previously located and selected content. In case only the identifier is supplied, the client should perceive this as removal of the specific content-item.

7.2.3.2 Functional Elements and Associations/Transactions

The combination of the centrally based control with the distributed content transfer scheme implies that functionality is not strictly located either at the client side or the server side. Both ends form a content distribution network the operation of which relies on the proper execution of certain functionality at certain points. Absence of one element may hinder the provision of the services.

As already stated, the CI-SA provides a single service end-point through which clients can publish and search for content as well as initialize and maintain the settings in their P2P Engine instances for joining certain overlays for content acquisition. Since the communication between CI-SA and client is based on IMs all function invocations occur is an asynchronous manner.

As a functional block, Content Indexing consists of a number of functional elements that operate at both ends in a complementary way. This distributed architecture requires that among the functional elements certain relations are established so that the overall operations are properly driven.

In the following figure (Figure 12) there is a representation of the associations that are establishment for the provision of the services of the Content Indexing.

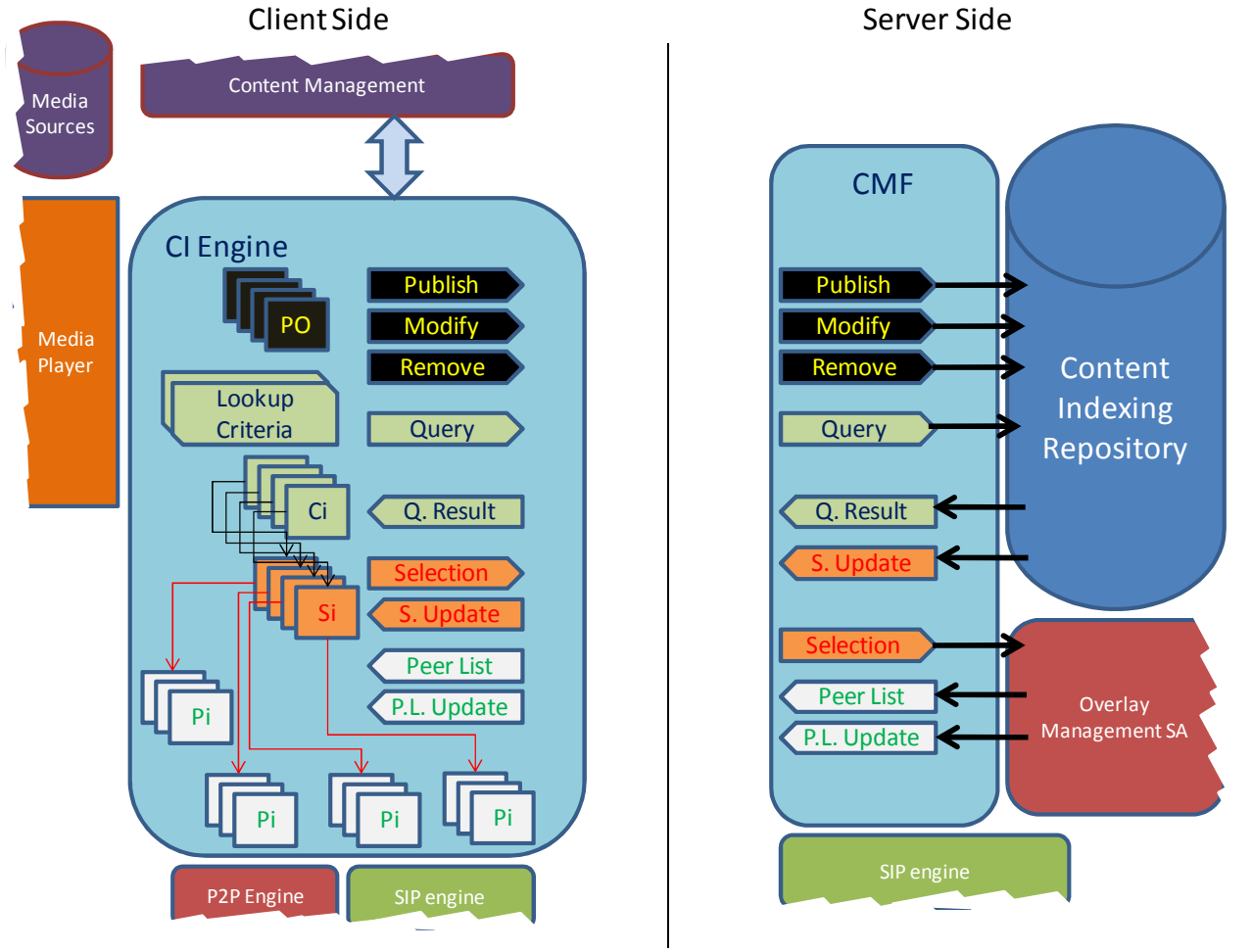


Figure 12: CI Client/Server Side Associations

The functional elements are grouped according to the content objects upon which they act during operation. The content objects (Publication Object – PO, Content indices – Ci, Selection indices – Si, Peer indices – Pi), are created at the client side and they reflect information contained at the CI Repository. For both sides every element is marked as incoming or outgoing. Around the CI functional elements other functional blocks or sub-architectures with which interactions may occur are also drawn. The sequence according which the various objects are generated at the client side to trigger the creation of outgoing messages that will trigger also server side actions is analysed in the following.

The client side generates outgoing messages (publish, update, remove) in order to reflect modifications occurring at the publications of a specific client. These are collected by the server side and their content is used to update the repository.

As already stated, lookup criteria are fed into the CI engine of the client side and generate queries that are destined to the CI-SA at the server side. Lookup criteria lead to query results that in turn generate the creation of Content index objects.

A number of the Ci objects will be used to generate Selection index objects depending on the user preferences. The existence of the Si objects will further generate selection requests to be consumed by the server side. Changes regarding existing publications may lead to the generation of selection updates that have to be processed by the client side to adapt accordingly existing Si objects. Moreover, the selections will be also sent to the Overlay Management SA so that proper peer lists can be generated, updated and propagated to the clients involved in content transmission. These peer lists are maintained per Si object and they include Peer indices. The Pi objects are used to initialise and maintain P2P Engines at the client side.

Although only one client side is displayed in the previous figure (Figure 12), there is no limitation with respect to the number of clients that are associated with server side. The associations established between client side objects and server side repository entries that realise the CI logic enable the propagation of changes to publications, overlay syntheses, etc to all involved parties. Changes occurring in the objects of one client that may express content publication or selection trigger updates towards the server side repository entries that in turn result in updates towards the rest of clients that already have established associations, as the result of a content selection, between the objects of their environment and the In this way all the clients and the server side maintain a common view of the content offering as well as an updated view of the P2P topology in which they are involved.

A typical scenario of transaction including more than one client is indicated below (Figure 13).

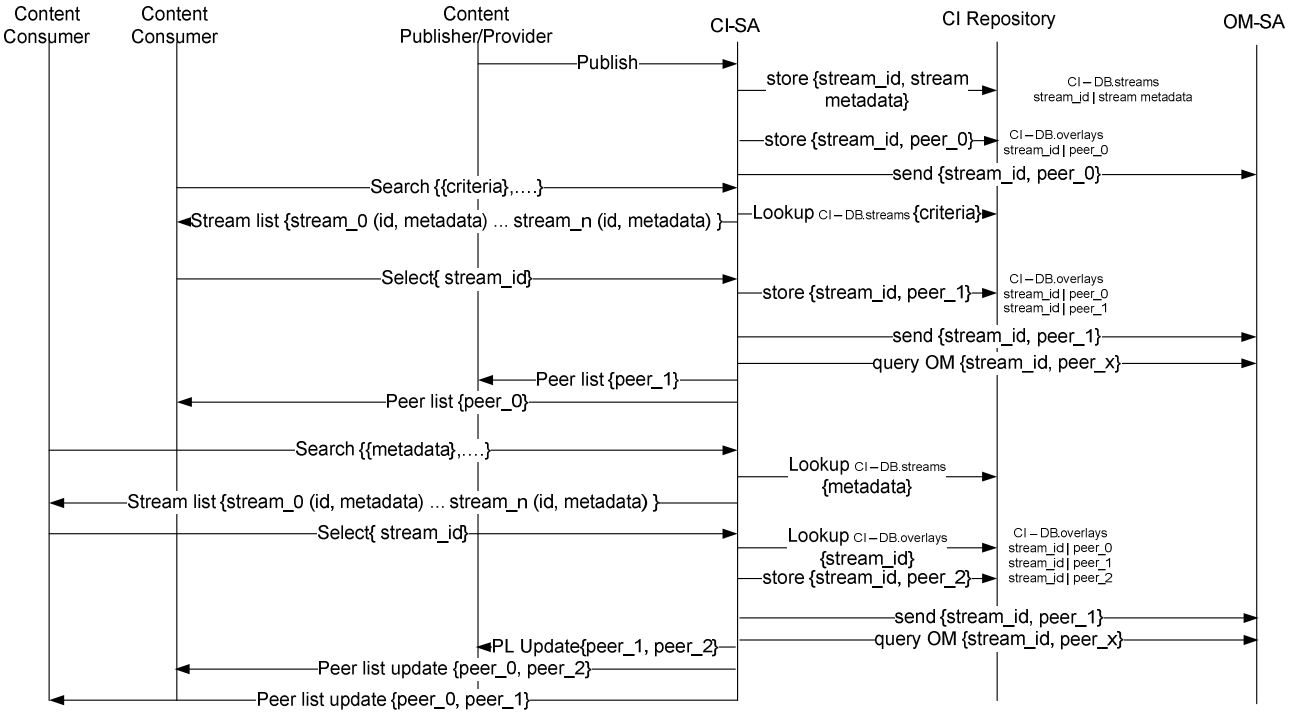


Figure 13: Typical CI Scenario

7.2.4 Description of the Software

7.2.4.1 Client Side

The Content Indexing module that exists in the client software is a Java implementation packaged as a jar library and offering a specific API to be invoked by other modules of the client. Its aim is to hide message processing and allow other components to be developed more easily on top of it. It exposes a set of functions relating to the CI functions described in the previous paragraphs. More specifically, it exposes a set of functions to cover the needs of the functional elements that generate outgoing messages. The component that utilises these functions should also supply reference to callback items that are able to receive associated incoming information. The association is based on the use of content identifiers or internal references generated in the context of the module's operation. The CI module maintains also all the content objects that hold publication, query, selection and peer information. Enumerations of these objects are returned by the use of the appropriate functions.

The Content Indexing module is not triggering any communication procedures but it can be associated with an object to which it can push outgoing messages for further adaptation inside SIP IMs and transmission. From the same object the CI module can collect any received messages. In this way the CI module is not closely related to the actual communication framework allowing for it to be re-used in other application assemblies and designs.

Processing and generation of the XML body of the messages is done by use of the appropriately generated XML Beans (Apache XMLBeans2.4.0).

7.2.4.2 Server Side

The CI-SA is the server side of the Content Indexing is Java application. It is using a custom implementation of a SIP stack for interfacing with the IMS. There is a Instant Message Processor class that creates separate threads for incoming messages processing. The processor is using appropriately generated XML Beans (Apache XMLBeans2.4.0) for processing of the XML documents included in the body of the IMs. The CI-SA maintains an XML data base (Apache Xindice 1.1) with which it communicates by use of XPath and XUpdate queries via the XML:DB XML Database API that is supplied together with the database software. In order to communicate with the OM-SA the CI-SA is using SOAP Web Services. This is done by the integration of an Apache Axis 1.4 Engine along with the properly generated Java Objects.

An additional mySQL database is used for storing the associations of content identifiers with the publishers and also associations of user SIP-URIs with network addresses.

7.3 Overlay Management

7.3.1 Purpose of the SA

An overlay graph architecture that forms the substrate for an efficient P2P live streaming system should meet the following requirements.

Firstly, the overlay graph should be constructed in such a way that every peer has a sufficient number of neighbours proportional to its uploading bandwidth. This guarantees optimal utilization of each one's uploading capability which, in turn, has a positive impact on block scheduling. Likewise, each node should have a sufficient number of incoming connections for the undisruptive reception of the video stream regardless of the dynamic network conditions and/or peer arrivals and departures. In addition, the overlay should be dynamically reconfigurable in order to timely react to the various changes of the underlying network as well as the dynamic peer behaviour. Last but not least, it should exploit the underling network latencies, i.e. round trip times, between peers, meaning that each peer should have as its neighbours those peers that are close to him in the network. In other words, the overlay must reflect as much as possible locality information in the way that peers are kept organized. Our proposed overlay architecture derived from the aforementioned requirements (Figure 14).

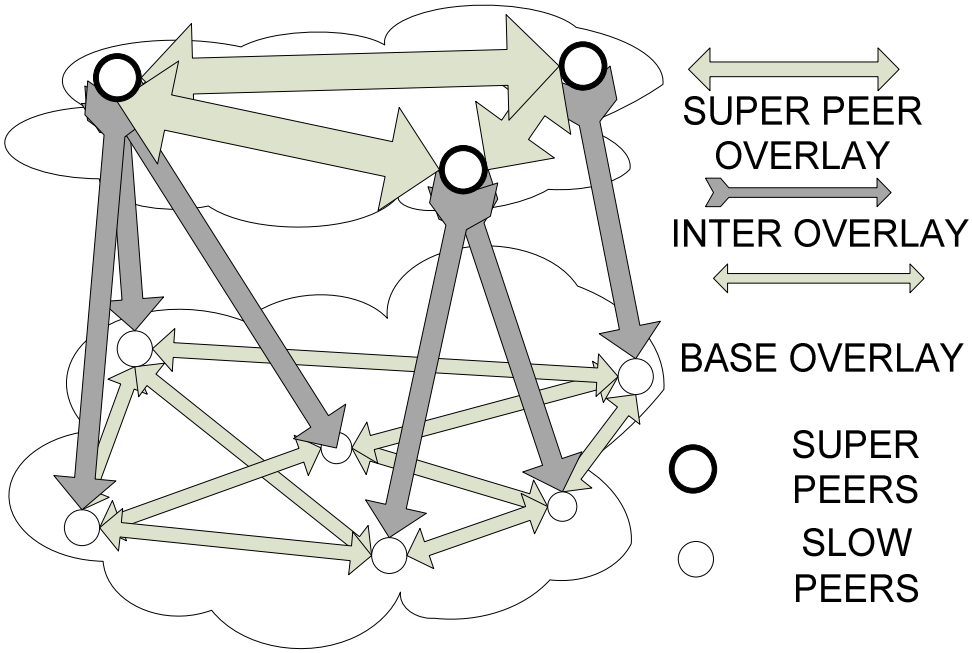


Figure 14: The system overlay and its interconnectivity structure

7.3.2 Description of functions

We distinguish between two types of peers: the super peers and the slow peers. The former are those peers with uploading bandwidth higher than the service rate of the video server whereas the latter less than the service rate.

Every slow peer that joins the system becomes part of a *base overlay* and is assigned a fixed number of neighbours, say M_B. This is a bidirectional mesh overlay, balanced with respect to the number of neighbours. If this peer also happens to be a super peer then it is also admitted to an additional overlay, called *super-peer overlay*, of similar characteristics as the base overlay. In this overlay, the peer is also assigned a fixed number of neighbours, say M_S.

Our *inter overlay* connects the base and super-peer overlays by assigning a number of super peers to each slow peer. More specifically, each slow peer in the base overlay selects a fixed number of super peers, M_I, which wishes to connect with. These interconnections are unidirectional originating from the super peers (outgoing reconnections) and terminating at the slow peers (incoming connections). They are also distributed among super peers in a manner proportional to the excess of their uploading bandwidth (uploading bandwidth minus the stream service rate). The quantities M_B, M_S and M_I are parameters of the system overlay. Figure 14F depicts such a system overlay with $M_B=3$, $M_S=2$, and $M_I=1$.

The introduction of the super-peer overlay has been proposed with a number of objectives in mind. Clustering together peers that have an uploading bandwidth higher than the video stream rate guarantees fast distribution among the super peers. Furthermore, organizing the peers such that everyone has equal number of neighbours with the smallest possible distance (locality) ensures uniform graph connectivity and short graph diameter. This results in a graph structure that can be exploited by a scheduler to achieve fast and optimal diffusion of the stream to the super peers.

The same graph organization mechanism is also followed in the base overlay in order to have the same benefits with the super-peer overlay. Also the purpose of the *base overlay* is the utilization of the uploading bandwidth of slow peers as it is the only overlay in which they transmit blocks. The need for *inter-connections* between super peers and slow peers, and the presence of super peers in the base overlay has been in order to utilize the excess bandwidth of super peers that it is not used in the super peer overlay. This process provides the sufficient incoming bandwidth for the slow peers in order to acquire the whole stream in the given time constrains while optimally utilizing their uploading bandwidth during the operation of the system.

Whenever a slow peer enters the base overlay it randomly selects $M_B/2$ peers from the base overlay as its neighbours. By doing this, a total of M_B new connections are created leaving the ratio of nodes in the overlay with the sum of connections constant. Similarly, when a super peer enters the super-peer

overlay it takes as neighbours $M_S/2$ peers. Finally, a slow peer takes as incoming neighbours M_I peers (interconnections) from the super-peer overlay.

On the other hand, when a peer leaves, its neighbours replace this peer with another one with a probability of 50 percent, except when a slow peer loses its interconnection with a super peer in which case it replaces the departed node with another one. The reasoning behind this mechanism is to keep the number of overlay connections constant in order to keep our streaming system unaffected from dynamic peer behaviour.

The proposed system overlay (base, super-peer and inter overlays) is continuously organized according to two distributed algorithms: a) a locality aware intra-overlay distributed optimization algorithm (Intra-DOA) applied to the base and super-peer overlay, responsible for organizing peers with M_B or M_S neighbours, respectively, and b) an inter-overlay distributed optimization algorithm (Inter-DOA) algorithm responsible for the interconnection of the base and super-peer overlay with M_I connections per every slow peer while taking into account the network latencies (locality) between the nodes in the two overlays.

7.3.2.1 Intra-overlay distributed optimization algorithm

Intuitively, the goal of this algorithm is to maintain a balanced overlay where nodes have equal number of neighbours while each node has neighbours that are physically close to it in the underlying network. This algorithm is responsible for handling: a) the arrival or departure of a node, b) changes in network conditions, and c) changes in the neighbourhood set of a peer.

Intra-DOA makes use of a function, called *energy function* and denoted as E(i,N(i)). It expresses the energy of node i in relation to the set of its neighbours, N(i). The energy function is defined as the sum of network latencies, Stt(i,j), of node i with all of its neighbours j$\in$ N(i), that is,

$$E(i,N(i)) = \sum_{j \in N(i)}^{|N(i)|} Stt(i,j) \qquad (1)$$

Furthermore we define E_{all} as the total energy of the nodes that participate in the overlay. Hence,

$$E_{all} = \sum_{i \in S}^{|S|} E(i,N(i)) \qquad (2)$$

The set S includes all the nodes that participate in the overlay every time instant. Our algorithm is an iterative distributed algorithm that each node executes periodically and uses integer linear optimization to minimize the sum of the energies of a small fraction of nodes that participate in an iteration while simultaneously balances the neighbours of the participating peers. It can be proved that global convergence is guaranteed but proof is omitted due to space constrains.

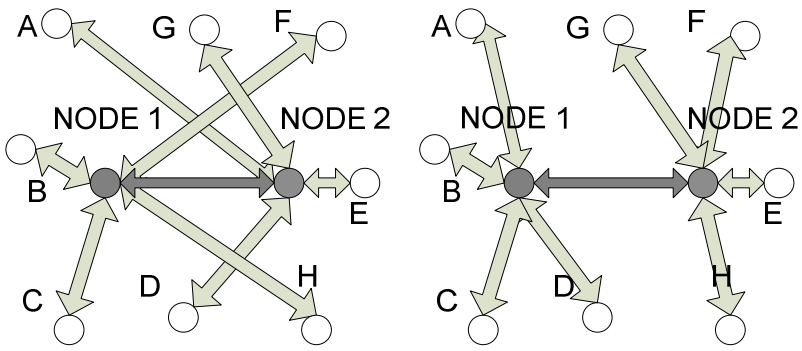


Figure 15: Overlay organization before and after an execution of Intra-DOA

Every time the algorithm is executed two adjacent peers in the overlay, which we call initiators, mutually exchange the sets with the network addresses of their neighbours. Then each initiator node measures the network latencies between itself and the neighbours of the other initiator node, which for clarity we call them satellite nodes. Left part of Figure 15shows the initiators, Node 1 and Node 2, and the set of their neighbours $N_{before}(1)=\{b,c,f,e\}$ and $N_{before}(2)=\{g,d,a,h\}$ respectively. In the figure the length of the edges between two nodes is proportional to the network latency between them.

We note as *Swap* the set of the initiators and as $N_{before}(i)$ the initial sets of satellite peers of each initiator i that could be reassigned during the execution of the Intra-DOA. We note as *a(i,j)* the parameter that its value is equal to 1 if the initiator i becomes neighbour with satellite j and 0 otherwise. The function that we want to minimize during the iteration of Intra-DOA is:

$$\min \sum_{i=1}^{|Swap|} E(i, N(i)) = \min \sum_{i \in Swap}^{|Swap|} \sum_{j \in N(i)}^{|N_{before}(i)|} a(i,j) * Stt(i,j) \qquad (3)$$

In order to ensure the balanced properties of the overlay we introduce two types of constraints. The first constraint ensures that satellite nodes will be assigned uniformly to each initiator. For both initiators we model this constraint with the following equation:

$$\sum_{j \in \{N(1) \cup N(2)\}} a(i,j) = \frac{1}{2} * \sum_{k \in Swap} |N_{before}(k)| \quad \forall i \in Swap \qquad (4)$$

The second constraint ensures that after every iteration of Intra-DOA the number of neighbours of a satellite node j that belongs to the various *N(i) | i∈Swap* remains constant and is expressed by the following equation.

$$\sum_{i=1}^{|Swap|} a(i,j) = |C(j)| \quad \forall j \in \{N(1) \cup N(2)\} \qquad (5)$$

Where *C(j)* denotes the set of nodes that belong to *Swap* and are neighbours with *j* before the execution of the algorithm. In most cases each satellite node is a neighbour of one of them.

Right part of Figure 15 illustrates the new sets of initiator's neighbours $N_{after}(1)=\{a,b,c,h\}$ and $N_{after}(2)=\{g,d,e,f\}$ respectively, after a single iteration of Intra-DOA.

Obviously the total energy (Eq. 2) of the system is a positive number as it is the sum of network latencies between nodes. If we prove that after each execution of Intra-DOA the total energy of the overlay decreases, then we have proven that our algorithm globally converges and there are no fluctuations in the overlay. By taking into consideration an execution of Intra-DOA we can rewrite the total energy of the overlay as:

$$E_{all} = \sum_{i \in S-Swap-Satellite}^{|S-Swap-Satellite|} E(i,N(i)) + \sum_{i \in Satellite}^{|Satellite|} E(i,N(i)) + \sum_{i \in Swap}^{|Swap|} E(i,N(i)) \quad (6)$$

The first term of (Eq. 6) expresses the energy of the nodes that do not participate in all iterations. The second term is the energy of the satellite nodes and the third term is the energy of the swap initiators. During each execution of Intra-DOA the first term of the equation remains unchanged while the third term decreases as we have discussed in the previous paragraph. We prove that the second term also decreases at each iteration of Intra-DOA equally to the reduction of the third term. This is due to the symmetry of the overlay since every edge participates in the energy of an initiator and a satellite node. More formally we can rewrite the energy of each satellite node as:

$$E(I,N(i))= E(I,\{N(i)-m \mid m \in Swap\}) + Stt(i,m) = Esat(i, Nsat(i)) + Stt(i,m) \quad (7)$$

Where $Esat(i, Nsat(i))= E(I,\{N(i)-m \mid m \in Swap\})$ is the energy of a node that remains constant during the algorithm execution.

The sum of all the energies of the satellite nodes is:

$$\sum_{i \in Satellite}^{|Satellite|} E(i,N(i)) = \sum_{i \in Satellite}^{|Satellite|} Esat(i,Nsat(i)) + \sum_{i \in Satellite} \sum_{m \in Swap} a(i,m) * Stt(i,m)$$

$$= \sum_{i \in satellite}^{|Satellite|} Esat(i,Nsat(i)) + \sum_{i \in Swap}^{|Swap|} E(i,N(i)) \quad (8)$$

If we replace this term in Eq. 6 we end up with the following expression:

$$E_{all} = \sum_{i \in S-Swap-Satellite}^{|S-Swap-Satellite|} E(i.N(i)) + \sum_{i \in Satellite}^{|Satellite|} Esat(i.Nsat(i)) + 2 * \sum_{i \in Swap}^{|Swap|} E(i.N(i)) \quad (9)$$

The first two terms of this equation do not change and the third one is the minimization function that reduces the total energy in every execution of Intra-DOA. This proves that our algorithm converges to a minimum energy overlay. Our system evaluation confirms the convergence of our algorithm and the vast reduction of the energy of each node.

7.3.2.2 Inter overlay distributed optimization algorithm

This distributed algorithm aims at optimizing the inter-connections that super nodes use in order to assist slow nodes to distribute blocks at the service rate of the live stream. It ensures that every super node has as its neighbours nodes that are physically close to it in the underlying network, while the number of these neighbours is kept proportional to the excess uploading bandwidth of each super node.

The excess bandwidth of each super peer *i* is defined as BE(i)=c(i) – μ, where μ is the streaming service rate and c(i) is the uploading bandwidth of i.

At the beginning of Inter-DOA two adjacent initiator nodes in the super peer overlay (indicated again as Node 1 and Node 2 in Figure X3) exchange the sets $M_I(i)$ of the network addresses of nodes in the base overlay they are connected with. For instance, assuming that Node 1 has twice as much excess bandwidth than Node 2 in Figure X3, their corresponding sets are $M_{I\_before}(1)=\{a,c,f\}$ and $M_{I\_before}(2)=\{b,d,e\}$ (left part of figure).

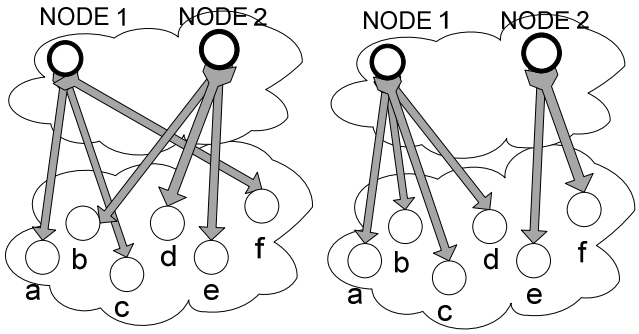


Figure 16: Inter overlay connections before and after Inter-DOA execution.

Using the same definitions for *Swap* and *a(i,j)* the function we want to minimize this time is:

$$\min \sum_{i \in Swap}^{|Swap|} E(i, N(i)) = \min \sum_{i \in Swap}^{|Swap|} \sum_{j \in N(i)}^{|M_{I\_before}(i)|} a(i,j) * Stt(i,j) \quad (10)$$

Inter-DOA is executed in exactly the same way as Intra-DOA and we introduce two types of constraints. The first ensures that after every iteration of Inter-DOA the number of neighbours of a satellite node j that belongs to the various $M_I(i) \mid i \in Swap$ remains constant and is expressed as:

$$\sum_{i \in Swap}^{|Swap|} a(i,j) = |C(j)| \quad \forall j \in \{Ms(Node\ 1) \cup Ms(Node\ 2)\} \qquad (11)$$

Where *C(j)* denotes the set of nodes that belong to *Swap* and are neighbours with *j* before the execution of the algorithm.

The second constraint (one for each initiator i) ensures that the participating super nodes will distribute their interconnection edges proportionally to their excess bandwidth BE(i), and is expressed by,

$$\sum_{j \in [N(1) \cup N(2)]} a(i,j) = [|N(1)| + |N(2)|] \frac{BE(i)}{\sum_{k \in Swap} BE(k)} \qquad (12)$$

Right part of Figure 16 illustrates the new sets of interconnections after an iteration of Inter-DOA $M_{I\_after}(1)=\{a,b,c,d\}$ and $M_{I\_after}(2)=\{e,f\}$.

7.3.2.3 Inter-ISP traffic minimization

We can extend our DOA in order to take also into account the ISP to which each peer belongs. In order to exploit this information and minimize the traffic between ISPs we follow exactly the same methodology that we have described in the previous sections but this time we calculate the energy of each peer as:

$$E(i, N(i)) = \sum_{j \in N(i)}^{|N(i)|} [Stt(i,j) + p(i,j)] \qquad (13)$$

Where p is zero where i and j belong to the same ISP and a large number if they belong to different ISPs. An extension of this work could be the factorization of the costs that different inter ISP links have through p we leave this now as future work.

7.3.3 Description of the Software

The core component of the software is the P2PEngine. This engine is integrated in both BCT and Monster clients so as to perform the distribution of the stream among the peers of the overlay. The P2PEngine consists of the scheduler, the transport mechanism, the control mechanism, the overlay management component and the statistics component.

The software is implemented in Python 2.6. For the construction of transport and control layer the "twisted" library was used. The communication protocol is

UDP with separate listening ports for data and control messages, but the modular architecture of the software can support TCP, mixed UDP and encoded messages.

Considering that BCT and Monster clients are implemented in Java, there is a web service interface in Java (P2PEngine.jar) which is used to perform the interconnection of Python and Java. More precisely, in the python side a light web server is launched with a set of registered web service functions. The Java part communicates with web service calls with the engine and the data exchange is fulfilled with dedicated sockets.

The media part of the software is implemented in BCT and Monster clients. They both use the JMF (Java Media Framework) to visualize the stream. This library produces RTP packets (streamer) which in turn are inserted in p2p blocks of the P2Pengine and are distributed in the p2p network. The clients that act as consumers of the stream reassemble the RTP packets and feed the media player.

7.3.4 Overlay management evaluation

For the evaluation of our P2P streaming system we have used the OPNET Modeler v.14[2] in order to test our proposed system under various underlying network topologies and conditions. Here we present its performance based on a topology from[3] that is also used as a reference topology in[4]. However, similar performance has been observed with all topologies we have worked with. In order to model heterogeneous uploading bandwidth capabilities we set the uploading bandwidths equal to 4000 kbps (class 4), 1000 kbps (class 3), 384 kbps (class 2), and 128 kbps (class 1) corresponding to a distribution of 15%, 25%, 40%, and 20 % of peers. The average uploading bandwidth of this distribution is around 1030 kbps; this value will be used hereafter as the average uploading capacity of the system.

In order to demonstrate the performance of Intra-DOA and Inter-DOA algorithms we form a randomly created overlay with 2000 nodes where $M_B=M_S=M_I=8$. Later in the evaluation we demonstrate the performance of our system under different values of these parameters. Each node executes the DOA algorithms every second and the service rate μ of the stream equals 95% (around 975 kbps) of the average uploading capacity.

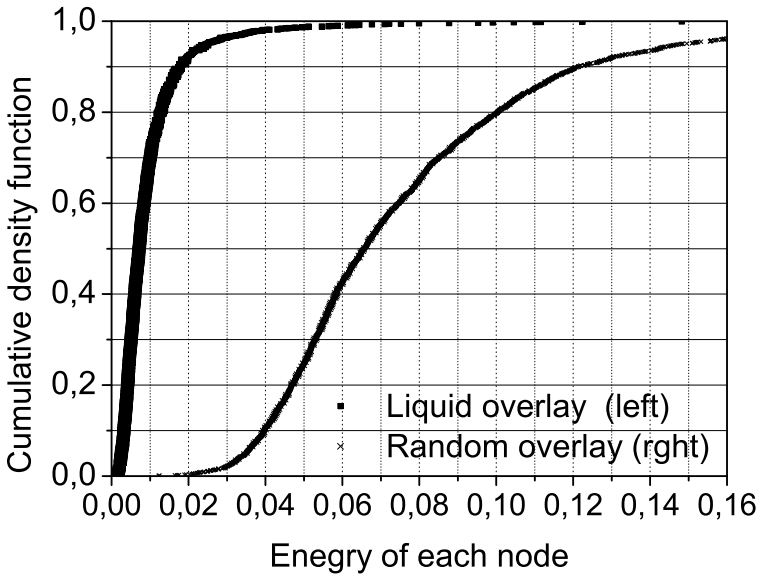
In Graph 1 we present the cumulative density function of the energy of each node divided by the number of its neighbors in the randomly formed overlay before the appliance and after converge of our algorithms. As we observe the mean energy (50^{th} percentile of the CDF) has been reduced from 0.07 to around 0.006 (more than 90%) which corroborates the locality properties of

[2] www.opnet.com

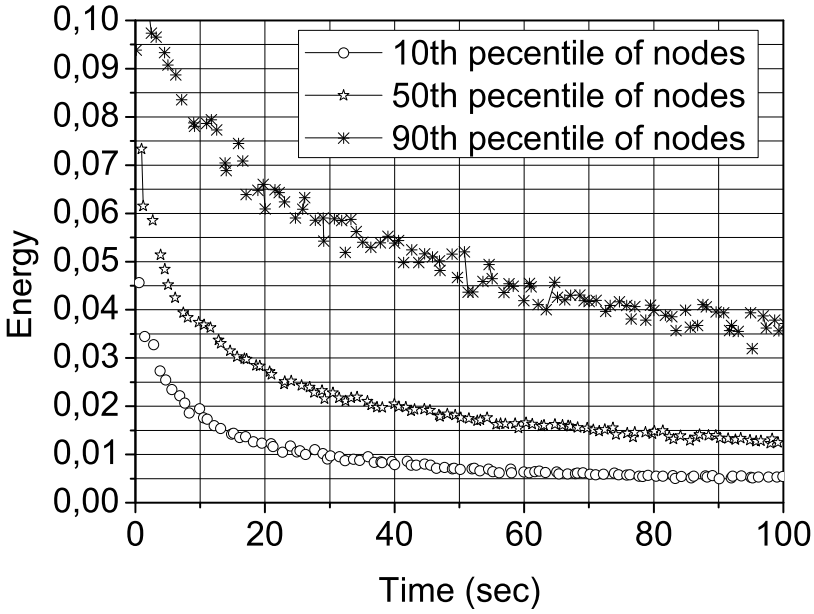
[3] http://www.cs.cornell.edu/People/egs/meridian/data.php

[4] Meng ZHANG, Qian ZHANG, Lifeng SUN, Shiqiang YANG, Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better?, IEEE JSAC 2007

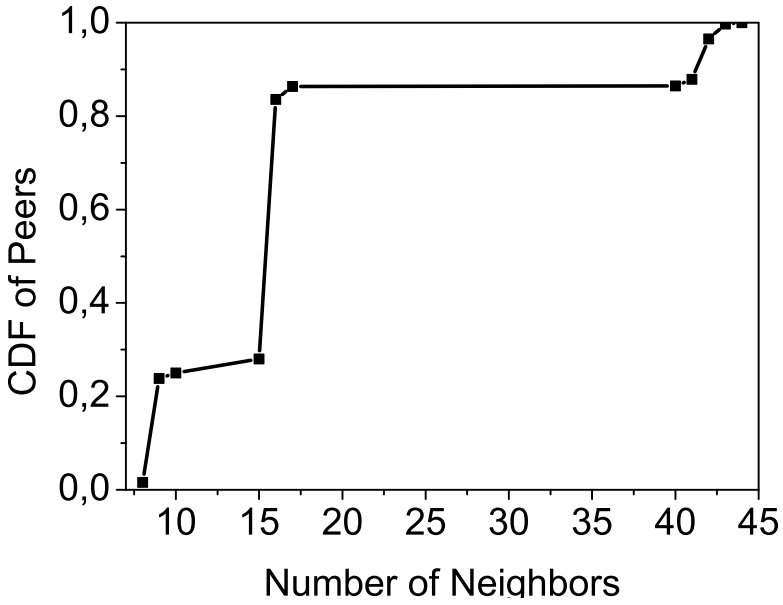
our overlay. In Graph 2 we demonstrate the speed of the reduction of the energies by executing again the same extreme scenario according to which we randomly insert initially 2000 peers and then we apply our algorithms. We observe that the mean energy (50^{th} percentile of the CDF) is reduced by a factor of 65% (from 0.07 to 0.023) in only 20 seconds, testifying the fast convergence properties of our optimization algorithms. Using the previous scenario, Graph 3 presents the CDF of the number of neighbours that each peer has after convergence of algorithms. Every slow peer (class 3,4) has exactly 16 neighbours, 8 neighbours in the base overlay and 8 interconnections to the super overlay. These interconnections are distributed to the class 1 and 2 peers in the super overlay according to their capacity. As the excess bandwidth of class 2 peers is minimal compared to class 1 peers the interconnections are distributed to these classes with a ration 31/1. So every class 2 peer has around 9 neighbours (8 in the super overlay plus one interconnection to the base overlay on average) while every class 1 peer has between 39 and 43 neighbours accordingly. Graph 3 shows that our algorithms eventually balance the number of neighbours that peers of the same class have.



Graph 1Energy of peers.



Graph 2 Speed of energy minimization.



Graph 3 Number of neighbours.

7.4 Content Security

The Vital++ Content Security (CS) architecture has been designed to enable content providers to control the distribution of their content using a Digital Rights Management technology. Vital's DRM system took its requirements from RBB, the consortium's content provider, and relate to real-world business requirements. The requirements include:

- Conditional Access to streaming Content
- Encryption of file-based and streamed content where required.
- Allow flexible rights expression
- Integrate with Accounting
- Respect for privacy and consumer rights
- Assertion of fair-use for purposes such as backup/education etc.
- Identity-based conditional access (providing a better alternative to Geo-IP blocking)

7.4.1 Purpose of the SA

This section elaborates on the requirements described previously and relates them to particular usage scenarios.

7.4.1.1 Identity-based Conditional Access

By focussing on techniques for fair use and DRM the architecture actually enables content distribution scenarios, which would otherwise be difficult. One example is identity-based conditional access. RBB, like many broadcasters, are state funded to provide content free of charge to citizens of the state. However, issues arise where citizens are travelling but still want to avail of their right to access the content, which is funded through their taxes and/or license fees. Currently, many Internet users are frustrated in attempting to access such streamed content when they travel because of the process known as Geo-IP blocking. Here, a black-list of public IP addresses is maintained based on the network class assignments of the Internet Assigned Number Authority. (IANA)[5]. This is a crude mechanism that is obviously problematic when content is accessed by travelling users, potentially using mobile devices.

The Content Protection Subsystem contains a conditional access system that evaluates licensing requests against business rules set by the content provider. These rules can be set across multiple parameters including:

- User Identity (user-based access)
- Group Membership of the User (group-based access)
- Time (i.e. content may only be licensed for a certain timeframe)
- Device capabilities and media encoding
- Accounting rules such as pre-paid versus post-paid billing.

[5] Internet Assigned Numbers Authority – IANA, http://www.iana.org

Identity based conditional access enables Vital++ to leverage the strong authentication, and potentially message encryption, of IMS in identifying whether a network subscriber is entitled to access content. For example, A Vodafone Germany subscriber could be allowed to access RBB content anywhere in the world providing they authenticate using their IMS.

7.4.1.2 Fair Use

The resulting architecture permits "fair use". Fair use is a legal doctrine, which applies in both United States and under European law, which permits user to reproduce portions of copyrighted material for various limited scope purposes. Examples of fair use include reproducing sections in a long work for educational use or using thumbnail images in a website. The idea of "Fair Use" within a DRM system appears to be an oxymoron but it has obvious benefits in that it promotes the distribution of content while not diminishing the legal rights of the owners of the intellectual property.

No attempt is made to enforce fair use since what constitutes fair use is not easily defined. Therefore the courts ultimately decide what constitutes fair use. This makes it impossible to automate a system to determine if an activity is indeed fair use. It is possible, however, to keep track of content consumer's requests to use content under a "fair use" license. The Vital++ Content Protection subsystem does so while anonymising the identity of the content consumer from the content provider. This information is stored in an anonymisation agent and may be divulged under legal criteria according to the jurisdiction. This decoupling gives assurance to the content consumer of that their fair usage is being protected while assuring the content provider that licensing is being audited for non-repudiation purposes.

7.4.1.3 Integration with Accounting

The CP subsystem integrates with a standardised IMS accounting system using the Rf and Ro diameter interfaces required for post-paid and pre-paid billing, respectively. Additionally, a Vital++ accounting and billing system is provided. This permits a content provider to register a charging scheme for a particular item or collection of content. A charging scheme may use charging data such as a cost/byte, a direct item cost and incentivisation schemes based on being a good "netizen" of the Vital++ overlay by contributing bandwidth and storage to the overlay. The accounting system is described in more detail in D4.1.

7.4.1.4 Flexible rights expression

By flexible rights expression we mean a mechanism that is rich enough to permit a content provider to describe rights using a range of parameters including:

- Group and user content access rights;
- Content expiry scheduling; (e.g. RBB mandate that content is only available for 7 days)
- Different rules for roaming users (some content may not be available when roaming)
- Rules determining the quality (codec and bit-rate) of licensed content

To achieve these goals the CP subsystem uses the rules engine from the Open Source Java Drools[6] project. This is capable of processing rules and data-types of effectively arbitrary complexity. A side-effect is that the Content Protection rules are decoupled from the implementation of the CP subsystem and may be created and tested using graphical tools. Drools logic is processed using a workflow system, which we also use within the CPS.

7.4.2 Description of functions

The CPS is integrated within the IMS network as shown below.

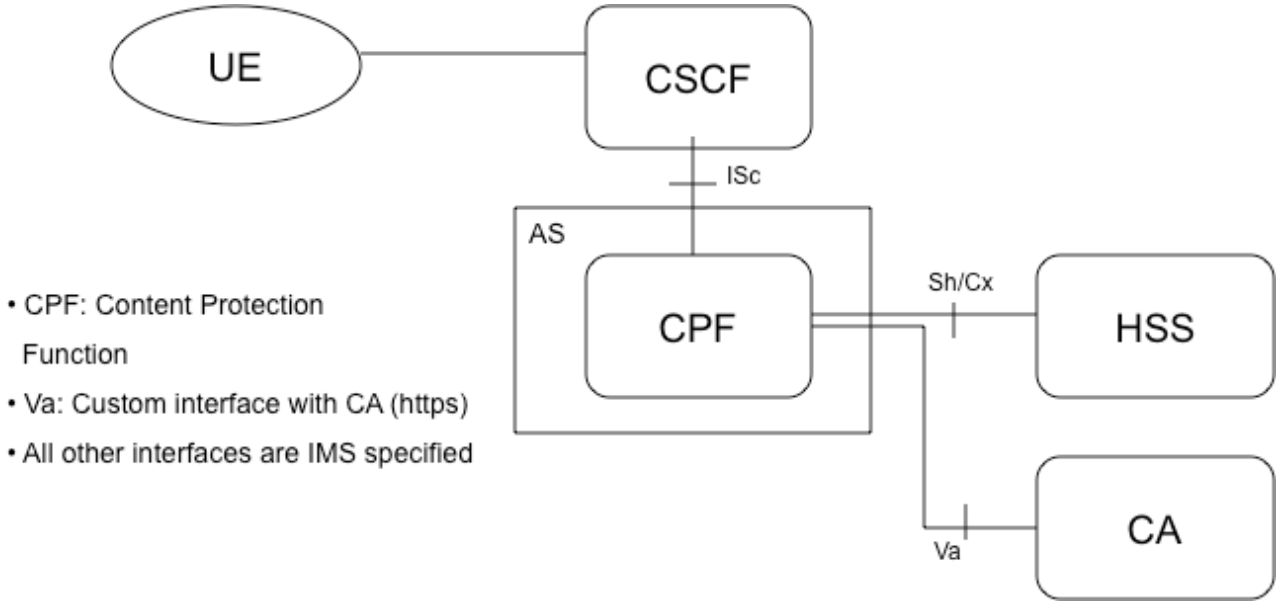


Figure 17 : CPS integration within the IMS

The User Equipment (UE) here represents a Vital++ node. The node accesses the functions of the Content Protection Function, the logic of the Content Protection Subsystem) using the IMS ISc interface. The ISc is a SIP protocol connection that is used when the S-CSCF loads a trigger point corresponding to the message that has been presented to it. In our case the message will be matched base on a known "service identifier" e.g. content-protection@<vital-domain> and the Vital++ SIP header that is added to all Vital++ messages.

[6] Drools Business Logic Integration Platform, http://jboss.org/drools/

The process of licensing a piece of content follows a Request/Response model and uses the SIP Instant Messaging conversation mechanism defined by the 3GPP. By re-using an existing mechanism we rely on the existing IMS authentication and message security model. This is explained in further detail in D4.1.

The Content Protection Function (CPF) is deployed within a standard IMS application server corresponding to the Java Community Process's JSR 289[7] specification. This is the latest specification for Sip Servlets. The CP subsystem runs successfully in the open source "SailFin"[8] servlet container.

The CPF may additionally use the HSS to verify a subscriber's credentials using the Sh (profile information) and Cx interface. However, this is a non-standard use of the Cx interface and has not been implemented within Vital++.

The content protection system uses Public Key Infrastructure (PKI) to mutually authenticate content provider and content consumer. The CPS acts as an trusted intermediary meaning that the content consumer and provider do not have to interact directly in the licensing process. The mutual authentication means that the content consumer can be confident the licensed content is being licensed by the correct provider and hasn't been maliciously tampered with. The content provider benefits from IMS and PKI-based identity management being used to verify the identity of the consumer making it difficult for the licensing party to impersonate another user. A Certificate Authority (CA) is used to associate public-private key pairs with IMS identities.

7.4.2.1 VITAL++ DRM Interface Component

The design and implementation follows the format of the "Mother May I" (MMI) protocol specified by the Open Media Commons (OMC) initiative[9]. Our modified protocol specification is described in greater detail in D4.3. It is based on sending text-based attribute value pairs and is therefore inherent extensible. For Vital++ we have re-implemented core components of the OMC DReaM entirely using Java and using standardised components for workflow management and rules processing.

[7] JSR SIP Servlet v1.1, http://jcp.org/en/jsr/detail?id=289

[8] Glasfish SailFin, http://wiki.glassfish.java.net/Wiki.jsp?page=SailFin

[9] Open Media Commons initiative, http://openmediacommons.org/

7.4.2.2 CP Subsystem Constituent Components

Required CPS components are shown in purple. Optional nodes that may be aggregated and/or provided by third parties are shown in green.

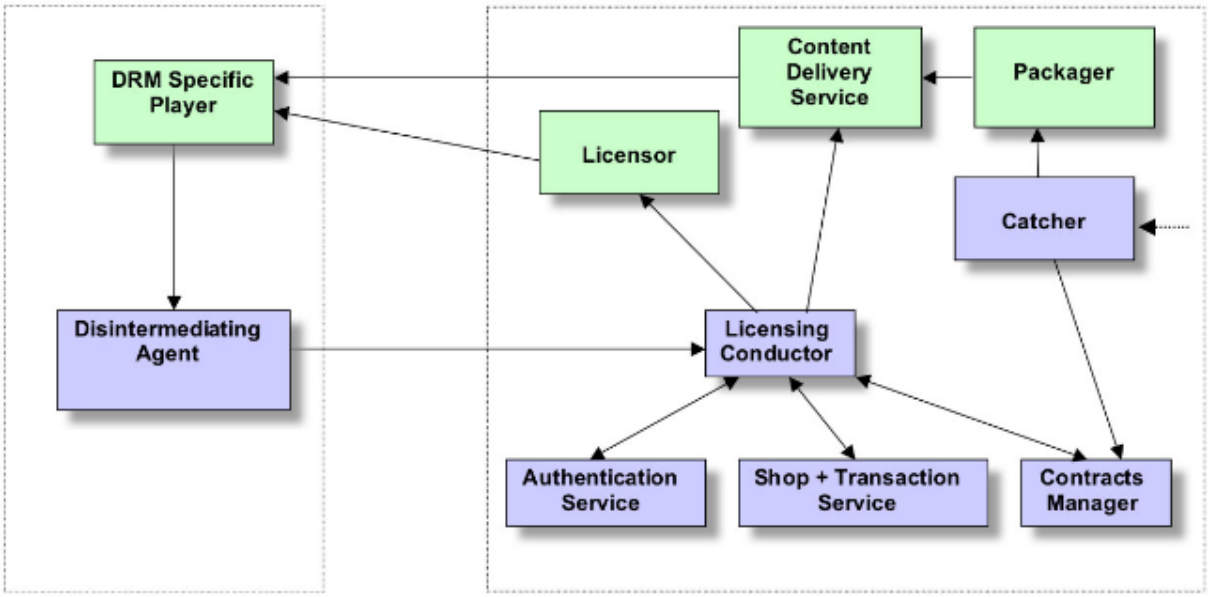


Figure 18 - CPS model is based on Open Media Commons

7.4.2.3 Client - DRM Specific Player

The DRM Specific player is a client-side player application that has DRM specific support for handling protected content and licenses. This is the Vital++ client, enhanced with the CPS client library.

7.4.2.4 Client - Disintermediating Agent

This refers to the client side library implementing the Request-side of the licensing protocol supported by the CPS. It dis-intermediates in that it supports a flexible licensing model (MMI) that can be customised to fit content protection and rights management solutions from different vendors. The server-side counterpart of the client disintermediating agent is the licensing conductor.

7.4.2.5 Licensor

The licensor is tightly bound to the DRM specific content protection technology. In Vital++ we have implemented a content protection system suitable for both live streaming and file sharing.

7.4.2.6 Licensing Conductor

The Licensing Conductor plays the role of managing the licensing processes involved in the CP Subsystem solution. It has interfaces to the CP Subsystem Client, Shopping and Transaction Service, Authentication Service, Contracts Manager and the Licensor. It performs the necessary e-commerce transactions and authentication of the user. It instructs the Licensor to generate the license for a given user for specific content. The licensing conductor is implemented using a java workflow engine and may therefore be customised based on future requirements.

7.4.2.7 Contracts Manager

The Contracts Manager stores business rules associated with content, as well as user rights. This component has interfaces to the Licensing Conductor and the Licensor. The Licensor generates a license for a given piece of content based on the business rules and user rights that are available in the Contracts Manager.

7.4.2.8 Authentication Service

The authentication service is where subscribers, users and devices are cleared for access to services and content. Authentication functions in Vital++ are provided by the IMS Core and augmented by the CA for the purposes of mutually authenticating content provider and consumer.

7.4.2.9 Shop and Transaction Service (Accounting Services)

The work flow functions of shopping and transacting purchases includes everything from collecting payments from buyers to paying sellers and making sure that everyone is appropriately compensated in a secure manner. This component will be used to ensure that Ro and Rf charging is applied correctly for consumed media.

7.4.2.10 Content Delivery Server

The OMC content delivery server is replaced by a P2P overlay in Vital++. A usage example is further described in D4.2. The overlay distributes protected content while keys are distributed using IMS signalling. A “super distribution” mechanism is used to ensure that all Vital++ distribution paradigms (live streaming, file sharing, media-on-demand) can be supported.

7.4.2.11 Packager

The packaging process involves combining content data/files with associated metadata and creating logical packages that include the defined business rules.

7.4.2.12 Catcher

The Catcher performs content ingestion. It receives content and associated business rules from the content supplier. The content, which is unprotected at

this stage, is passed to the Packager. The business rules associated with the content are passed to the Contracts Manager.

7.4.2.13 Disintermediation

One of the key features of the CP Subsystem architecture is the ability to accommodate the inclusion of specific rights management and conditional access components from third parties while avoiding the need to incorporate all their back-end components. This is an important feature in any commercial Vital++ system. A mobile operator may, for example, have deployed an Open Mobile Alliance DRM system that is currently supported by many handsets from vendors such as Sony Ericsson and Nokia. The CPS will permit the incorporation of such a system so long as the required components are implemented.

The disintermediation system enables multiple instances of these components to exist in a Content Protection or Conditional Access system.

The content protection specific components of CP Subsystem include: player, licensor and packager.

Components that are not content protection specific include: a disintermediating agent, conductor, catcher, licensing conductor, contracts manager, authentication service, shop and transaction system, and content delivery service.

The process of disintermediation happens as follows:

1. Client requests a license
2. Front-end service redirects client to a client disintermediation agent
3. Disintermediating agent contacts Conductor (back-end service)
4. Conductor contacts back-end services for authentication and rights verification
5. Conductor signals front-end service with instructions to deliver license to client

Front-end service delivers license

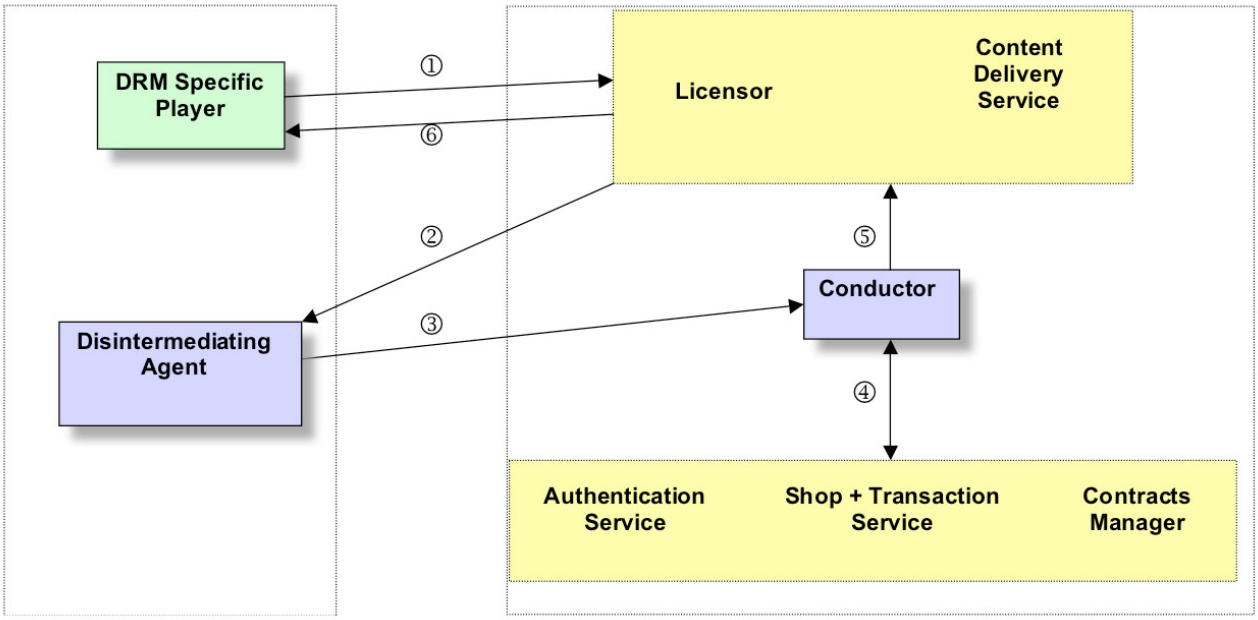


Figure 19 - Disintermediation process

7.4.3 Message Exchange and Transactions

In this section we give an overview of the MMI profile, which is used for Vital++. As described earlier, MMI defines a request-response protocol for licensing content based on the work of SUN's DReaM project. The Vital++ MMI profile is described in more detail in D4.3. Additionally we include message sequence diagrams for content publishing and content licensing using the CP subsystem.

7.4.3.1 Licensing Workflow

As described previously the licensing conductor uses the Java Business Process Management (JBPM) open source workflow management engine to describe the licensing process. This is an inherently flexible approach as we have sought to decompose the licensing functions into modular blocks corresponding to the different states of the licensing process. The diagram below shows how request handling, rules processing and accounting are integrated within a single workflow.

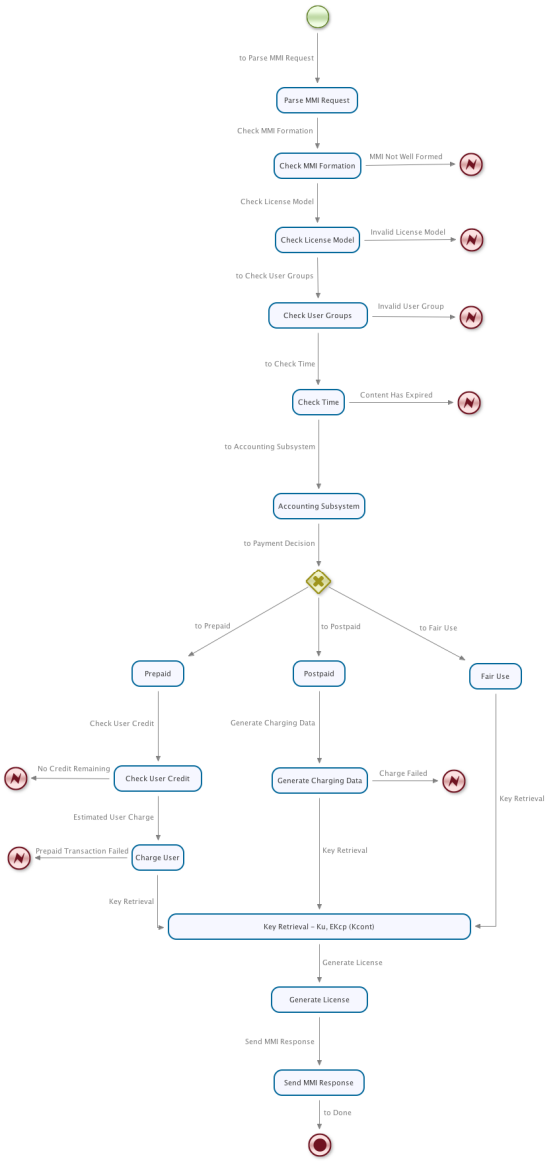


Figure 20. Licensing Workflow Diagram

The JBPM graphical workflow design tool may be used to modify the workflow without the need to re-implement or recompile any java code.

7.4.3.2 CI/CS interactions

Elements of the workflow are also re-used by the Content Indexing subsystem. The CI and CP subsystems interact during the content discovery and licensing phases. The CI requests information from the CP subsystem about:

The licensing model available for a particular content item (e.g. fair use, pre-paid, etc.)

Whether the content is available to the user or user's group?

Whether the content has been successfully licensed by the user? Once content has been licensed the CI can make overlay details available to the licensed user. Also, a user already in possession of a valid unexpired license for a content item does not have to re-license it.

This interaction is described in more detail in D4.3.

Content Publishing

The following sequence diagram shows how content publishing is achieved, demonstrating the interactions between a platform user's Vital++ client, CI and CP subsystems.

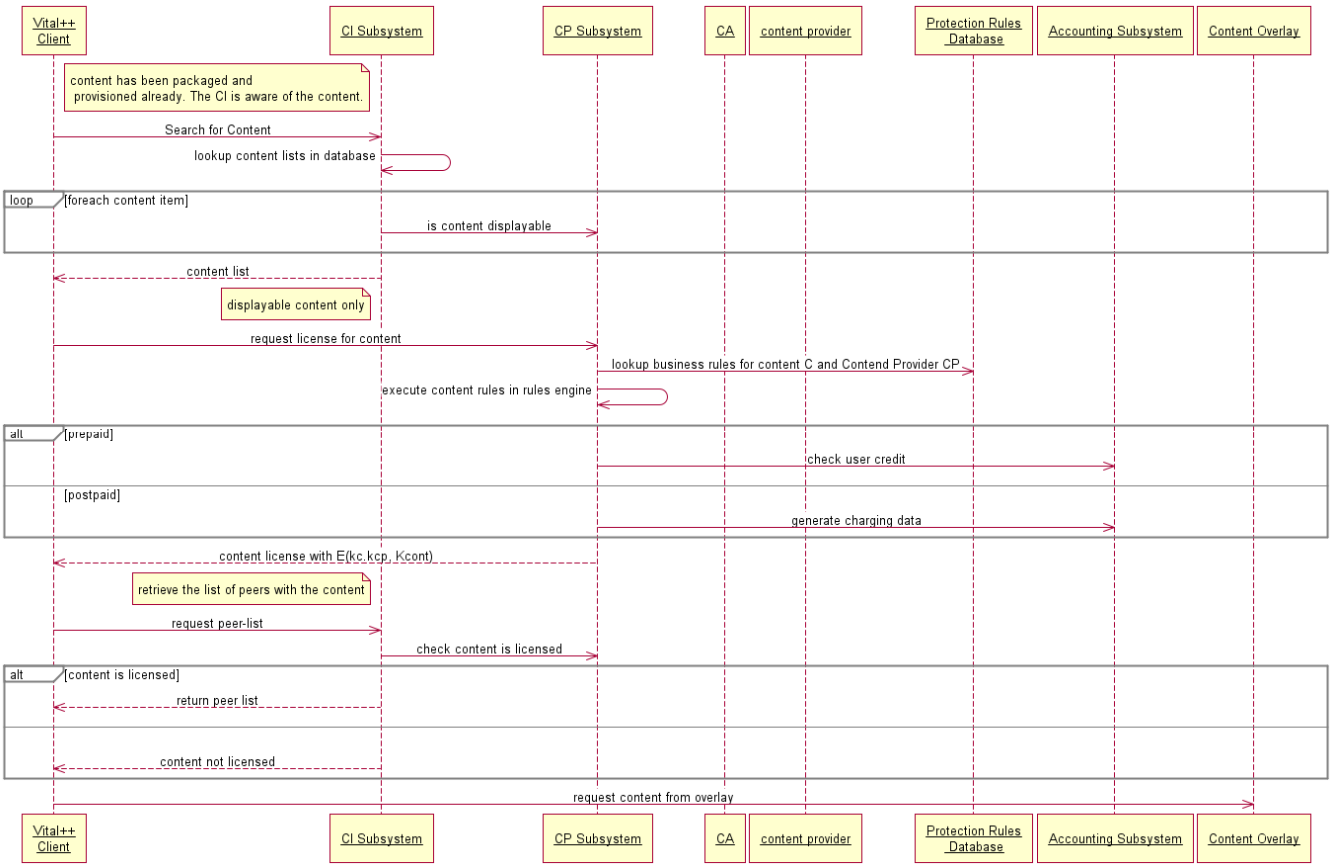


Figure 21. DRM Content Publishing

Content Licensing

The following sequence diagram shows how published content is licensed, detailing interactions between a user's Vital++ client, the CP and CI subsystems.

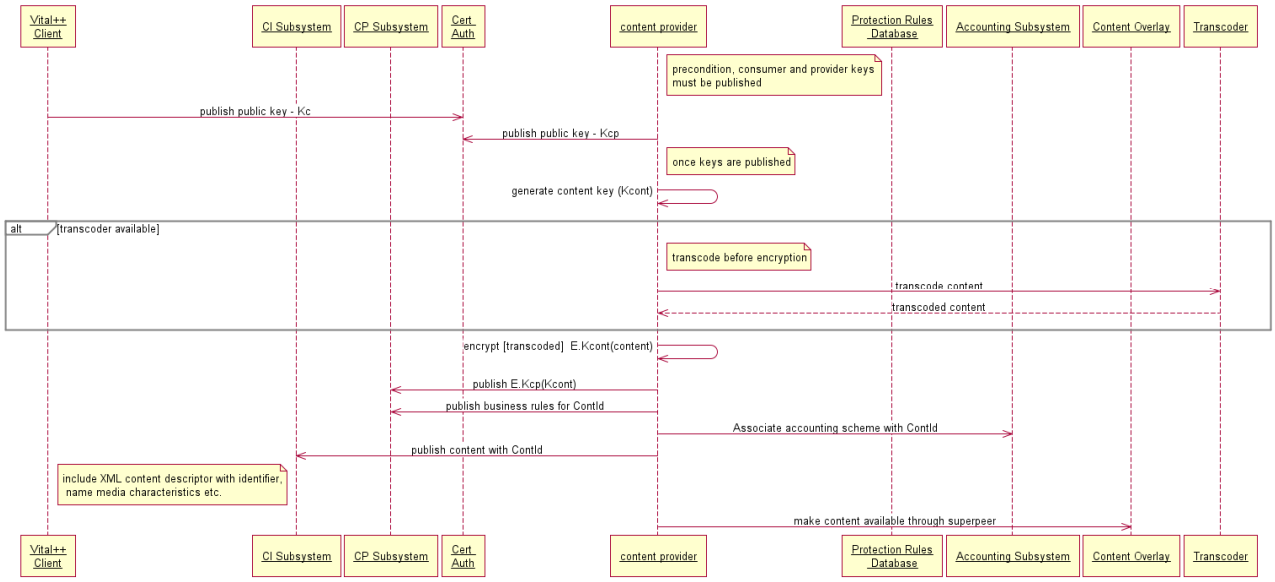


Figure 22. DRM Content Licensing

7.4.4 Description of the Software

The Vital++ Content Protection subsystem is implemented entirely in Java. This has been done to promote portability of the code to multiple server and client platforms. Java was also an excellent choice as there are many open source software libraries and components that could be incorporated within the Content Protection Subsystem's codebase in order to accelerate the development process.

The CPS uses components from multiple open source software projects including.

- Glassfish *SailFin* SIP Application Server
- Java Business Process Management (JBPM)
- JBOSS "Drools" Rules Engine
- The GNU SRTP Implementation – modified and extracted from the GNU ZRTP4J[10]
- The Java Cryptographic Architecture[11]
- The Open Media Commons DReaM Rights Management project[12]

The CPS server components described in ? are deployed in a VMWARE virtual machine which is run on an ESX hypervisor in the TSSG server laboratory. The

[10] GNU Telephony ZRTP4J, http://www.gnutelephony.org/index.php/GNU_ZRTP4J_How_To

[11] Oracle, Java SE Security, http://java.sun.com/javase/technologies/security/

[12] Open Media Commons, http://www.openmediacommons.org

server is accessible via a VPN connection and may also be given a public IP address for test purposes.

The architecture is easily scaled as the License Conductor's workflow engine may be run in a cluster configuration as can the GlassFish servlet container. The use of the scalable and decoupled software components means the CPS can theoretically be of use in trials of varying scales. However, we have not analysed performance figures of the current solution.

7.5 P2P Media Exchange

7.5.1 Introduction

Distribution of media items in the context of live streaming or video on demand requires the use of a media streaming protocol such as RTP or RTSP. Definition of proprietary media streaming formats or adoption of other not open source compatible solutions such as Microsoft's ASF was out of the scope of the project and it was also posing certain limitations with respect to the development of the client architecture.

Both clients (Monster and BCT Client) are Java Applications and the only media support that can be integrated is based on the Java Media Framework implementation. JMF supports both RTP and RTSP reception but it cannot create outgoing RTSP streams. So the remaining solution was the use of RTP that is supported by JMF with respect of both sending and receiving.

Moreover, integrating RTP with P2P transport capabilities stems also from the fact that IMS media transport utilises RTP in most cases. So it was more straightforward to proceed with the integration by using existing items and not replacing them. This procedure allows for more transparent deployment of the P2P layer in the client as far as the media management components are concerned.

The way the P2PEngine is integrated with the JMF player component is based on the provision of a custom RTPConnector implementation. While the default implementation of JMF uses UDP/IP for transport in the integrated JMF/P2P solution, the RTP Managers are initialised with instances of a custom implementation of an RTPConnector. All of the instances are configured to use the same P2P Engine through a wrapper object. This object receives all the outgoing data and concatenates these into P2P Engine compliant blocks to be pushed to the engine for transmission. For blocks retrieved from the P2P Engine a segregation process follows. This process leads to the acquisition of separate RTP packets that are distributed according to their audio/video type to the proper RTP Managers and Players.

The following picture presents the architecture of the above approach. In typical JMF/RTP receive/transmit setup there is need for feeding the JMF Manager(s) Objects with the networking settings (IP Addresses/UDP Ports). In the case of the P2P based transport, RTP Manager(s) are configured to use

specific RTPConnector implementations that in turn initialise and maintain the appropriate PushSourceStreams for incoming RTP packets and OutputDataStreams for outgoing RTP packets. The Block Receiver and P2P Engine wrapper have been initialised in advance using the overlay bootstrapping information acquired through the CI procedures.

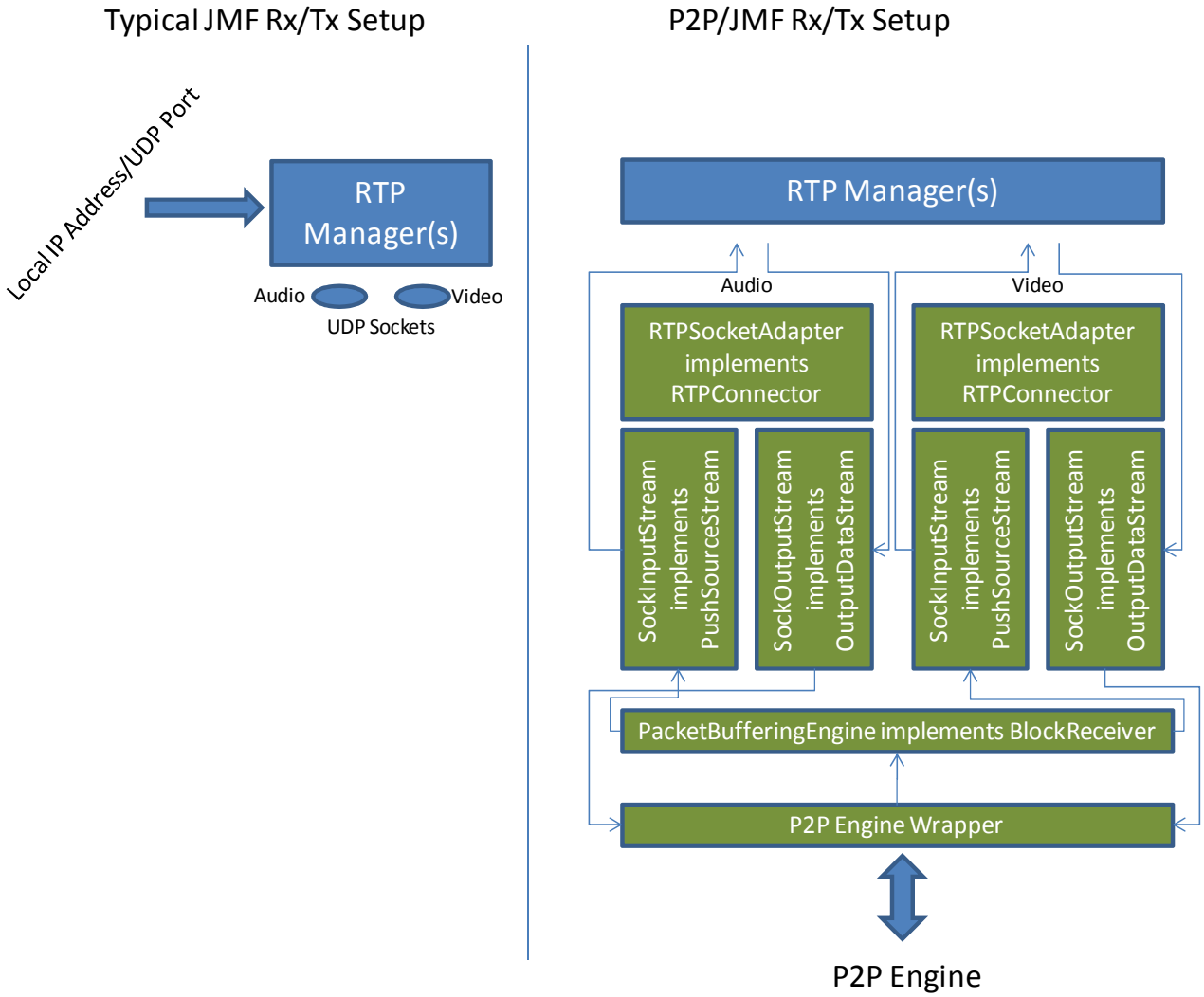


Figure 23: JMF Adaptation

7.5.2 P2P Client Engine

The p2p engine is integrated in the vital++ client and it is responsible for the distribution of data between the peers of the overlays. According to the type of the media (Video-on-demand, Live-Streaming and file sharing) a different scheduler is triggered. Therefore the transport mechanism inside the engine and the interface with the media part of the client is independent to the running scheduler.

More specifically, the functions of the interface that are responsible for pushing and retrieving data from the p2p engine are transparent to the type of the exchanged data. File chunks are encapsulated for file sharing (bit torrent approach) and rtp packets for vod and live streaming. Data and message transfer use completely independent chain in order to allow usage of different transfer protocol both on transport and application level (i.e. to allow the usage of TCP and UDP for messages and data). The implemented protocol transmits UDP packets, it is developed under Python and involves the use of Twisted framework[13] and cPickle (part of the standard Python library). The main target of the protocol is to reduce network fragmentation by reducing the size of transferred packets while keeping data in bigger blocks, to avoid network congestion and put the foundations for flow control by strictly controlling the sending rate. In order to fulfill these requirements, the protocol is provided with a receive buffer, a sending queue, a splitting/reconstructing routine that takes care also of byte encoding/decoding and a timed recursive call to the send procedure. The exchanged data are primitive types inside Python dictionaries. This is to improve encoding flexibility by giving the ability to choose among many encoding methods (for example bencode[14], the BitTorrent way for this, netstring[15] or others).

On the sender side, the dictionary is first encoded by cPickle. Then, if the size of data block is bigger than the given MTU parameter, it is split according to the MTU and the resulting chunks are put in other dictionaries tracking the sequence number, the number of chunks and the actual chunk number. Then they are put inside the sending queue. On the receiver side an entry is inserted in the incoming buffer in order to collect all the chunks and reorder them. When they are all arrived, the full block is decoded and sent upstream in order to perform the desired actions.

In order to give a minimum protection from network congestion and subsequent packet loss, the sending queue is exploited to send packets at a given rate. Also, all the parameters given to the protocol are adjustable at runtime, thus allowing the presence of flow control algorithms in the future. The performance of the protocol is pretty good: despite the use of a quite high level language, the Twisted framework gives the ability to keep a very low CPU consumption (around or less than 10% on a modern machine) and to achieve a very good timer resolution.

7.6 Future Enhancements

In this section, we will describe possible elements of the VITAL++ architecture which can improve the performance and robustness of the Vital++ system.

[13] http://twistedmatrix.com

[14] http://pypi.python.org/pypi/BitTorrent-bencode/5.0.8

15 http://cr.yp.to/proto/netstrings.txt

Those elements are not depicted in the overall architecture but provide additional or alternative ways to realize functionalities.

7.6.1 Client DHT

In order to store client related data, a distributed hash table (DHT) is advisable. Relevant client data can be SIP contacts to enable direct P2P message exchange. But also other client related data, like its client certificate, playlists or profiles can be stored in a DHT to overcome device changes or offline times. We propose to realize such a DHT with the P2PSIP[16].

Bootstrapping

In order to join a DHT, a client queries the overlay management sub-architecture for its position in the DHT overlay to retrieve a list of its neighbours. This can be signed by the corresponding application server in order to justify the initial communication with the new neighbourhood, i.e. to legitimize the new node.

Store and get

For the *store* operation, the storing node does not know if the information it is going to store is really related to the writer. Also, for the *get* operation, the reading node does not know whether the received information is unchanged/genuine. Figure 24 shows these relations.

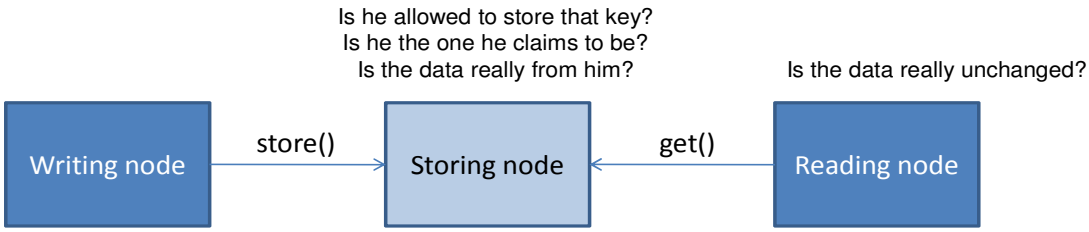


Figure 24: DHT Security issues

Whenever a node performs a *store* operation, it will store not only the data chunk, but also a signature over the data chunk and a sequence number. The sequence number can be used by the storing node to detect a replay attack. The storing node must also store its own certificate in the DHT with the same mechanism to allow an offline-verification of its signature. When a reading node now retrieves the data chunk, it can verify the integrity of the data by checking the signature. This way, data stored in a DHT can be secured against falsification by either writing or storing node.

[16] http://tools.ietf.org/wg/p2psip/

7.6.2 NASS Attachment

The network attachment sub-system (NASS) is an NGN element, standardised by ETSI/TISPAN. The NASS holds information about the physical connectivity of the user, i.e. bandwidth properties etc. Thus, an entity like the overlay management can retrieve this information and use them in order to position a peer in an overlay. Overlays can thus be created more related to the real network topology in terms of neighbourhood relations, but also in terms of intended data flows, regarding to the upload and download bandwidth available at the users connection.

This attachment is optional, because the deployments of a NASS in real operator networks are still rare. This is because the NASS itself is not a mandatory element in the ETSI/TISPAN NGN architecture.

Without NASS support, overlays can only be built based on reported bandwidth measurements between the peers. As this is quite imprecise due to additional network traffic, additional information about a user's internet connectivity will help to improve the process of overlay building with the aim to create overlay paths, which are related to the real network.

7.7 Summary of Network Components

During the process of integrating two totally different technologies such as IMS and P2P, several issues arose since there are similar aspects of functionality implemented differently by each technology. However, the Vital++ orientation dictated that IMS maintains the control part of all the transactions mainly regarding signalling and meta-information circulation while P2P has the control over content transfer.

Therefore a number of Network Components were designed to provide all the required information for creating and accessing overlays so that authorised and fully controlled access to multimedia items can be achieved. The functionalities offered by those network components were grouped and organised in a way that provides solutions for:

- Peer Authentication (P2PA-SA)
- User Authorisation, Accounting and Content Protection (CP-SA)
- Content Publication, Searching (CI-SA)
- Overlay Maintenance (OM-SA)

Client software has to interact in a specific way with the sub-architectures realising the above functionalities in order to join overlays either for providing or consuming content. By adapting SIP Instant Messages as the main communication protocol of the client side with the network elements it is ensured that only IMS authorised users can have access to the Vital++ services.

The certificate management that is offered by the P2P Authentication sub-architecture allows the users to generate certificates and also validate signed

messages against the claimed identity of the signer so that trust can be established among peers.

Published content is secured against unauthorised access by the use of encryption keys generated during publication. Circulation of those keys to users interested in acquiring the offered content is restricted by relevant business rules communicated to the systems by publishers again this is done through the CP-SA. Having granted listing access a user can identify a content item and proceed further by attempting to join an overlay. Accepting to receive the encryption key for content retrieval the user agrees on using the received item in a fair way. This item is stored in the accounting system not only for billing purposes but also for creating a relevant record that binds the user identity with its acceptance regarding the fair use.

Publication of content is parameterised to allow for provision of sufficient metadata that can allow for more complex services to be created through the interaction with the CI-SA. Close relation is established between Content Protection and Content Indexing subsystems so that possible limitations expressed during content publication can be enforced.

Finally, while operating behind the CI-SA, the OM-SA bootstraps the overlays by taking into account the location of the involved peers so that optimal syntheses are guaranteed. However, not actually being a network side element but operating as a distributed algorithm across the network the P2P Engine ensures the continuous evolution of the overlays towards optimal operation.

8 Conclusions

In this deliverable, we have described the fundamental building blocks for all VITAL++ services, scenarios and transactions. The functionalities and their intended purpose have been explained, as well as the network interfaces and protocols. The draft architecture, as defined in the deliverables 2.1 and 2.2 has been refined and extended according to the real-world environment as well as to the requirements, which have been reviewed and analysed.

Based on the introduced functionalities, higher functions and services can be realised, e.g. community services, which require authentic P2P messaging, or commercial P2P media distribution, which requires a content security mechanism and content indexing. As well, e.g. content distribution networks can be planned more efficiently, using the central intelligent overlay management.

This deliverable is complemented by deliverable 3.2, where application programmer's interfaces and functional block interworking are in the focus.

9 Annexes

9.1 Annex 1 - Content Indexing XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://xml.netbeans.org/schema/ci"
    xmlns:tns="http://xml.netbeans.org/schema/ci"
    elementFormDefault="qualified">
    <xsd:complexType name="keywords">
        <xsd:sequence>
            <xsd:element        name="keyword"        type="xsd:string"                minOccurs="0"
maxOccurs="unbounded"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="message">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="tns:content-item" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="function">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:pattern          value="publication|publication-successful|publication-
failure|publication-modify|publication-remove|query|query-result|content-selection|peer-list|peer-
list-update|located-content-update"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="peer">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="net-address" type="xsd:string"></xsd:element>
                <xsd:element name="port" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="content-item">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="tns:content"/>
                <xsd:element ref="tns:peer" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="content">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="id" type="xsd:string" ></xsd:element>
                <xsd:element name="locator" minOccurs="0">
                    <xsd:complexType>
                            <xsd:sequence>
```

```
                                <xsd:element            name="provider"            type="xsd:string"
minOccurs="0"></xsd:element>
                                <xsd:element            name="programme"           type="xsd:string"
minOccurs="0"></xsd:element>
                                <xsd:element            name="category"            type="xsd:string"
minOccurs="0"></xsd:element>
                                <xsd:element          name="subcategory"           type="xsd:string"
minOccurs="0"></xsd:element>
                                <xsd:element             name="series"             type="xsd:string"
minOccurs="0"></xsd:element>
                                <xsd:element            name="episode"             type="xsd:string"
minOccurs="0"></xsd:element>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="title" type="xsd:string" minOccurs="0"></xsd:element>
                    <xsd:element name="author" type="xsd:string" minOccurs="0"></xsd:element>
                    <xsd:element name="publisher" type="xsd:string" minOccurs="0"></xsd:element>
                    <xsd:element name="shorttitle" type="xsd:string" minOccurs="0"></xsd:element>
                    <xsd:element name="description" type="xsd:string" minOccurs="0"></xsd:element>
                    <xsd:element name="thumbnail" type="xsd:anyURI" minOccurs="0"></xsd:element>
                    <xsd:element name="duration" type="xsd:duration" minOccurs="0"></xsd:element>
                    <xsd:element name="keywords" type="tns:keywords" minOccurs="0"></xsd:element>
                    <xsd:element name="publishdate" type="xsd:dateTime" minOccurs="0"></xsd:element>
                    <xsd:element name="datefrom" type="xsd:dateTime" minOccurs="0"></xsd:element>
                    <xsd:element name="dateto" type="xsd:dateTime" minOccurs="0"></xsd:element>
                    <xsd:element name="lang" type="xsd:language" minOccurs="0"></xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:schema>
```

9.2 Annex 2 – XPath Query Example

Query Message:

```
<message function="query" xmlns="http://xml.netbeans.org/schema/ci">
  <content-item>
    <content>
      <locator>
        <provider>provider-1</provider>
        <programme>programme-1</programme>
        <subcategory>subcategory-1</subcategory>
        <episode>episode-1</episode>
      </locator>
      <keywords>
        <keyword>key-11</keyword>
        <keyword>commonkey</keyword>
      </keywords>
      <dateto>2010-03-11T10:25:30.539+02:00</dateto>
      <lang>EL</lang>
    </content>
  </content-item>
  <content-item>
    <content>
      <locator>
        <provider>provider-2</provider>
```

```
      <category>category-2</category>
      <series>series-2</series>
    </locator>
    <keywords>
      <keyword>key-20</keyword>
      <keyword>key-22</keyword>
    </keywords>
    <dateto>2010-03-11T10:25:30.539+02:00</dateto>
    <lang>EL</lang>
  </content>
</content-item>
<content-item>
  <content>
    <locator>
      <provider>provider-3</provider>
      <programme>programme-3</programme>
      <subcategory>subcategory-3</subcategory>
      <episode>episode-3</episode>
    </locator>
    <keywords>
      <keyword>key-31</keyword>
      <keyword>commonkey</keyword>
    </keywords>
    <dateto>2010-03-11T10:25:30.540+02:00</dateto>
    <lang>EL</lang>
  </content>
</content-item>
<content-item>
  <content>
    <locator>
      <provider>provider-4</provider>
      <category>category-4</category>
      <series>series-4</series>
    </locator>
    <keywords>
      <keyword>key-40</keyword>
      <keyword>key-42</keyword>
    </keywords>
    <dateto>2010-03-11T10:25:30.541+02:00</dateto>
    <lang>EL</lang>
  </content>
</content-item>
<content-item>
  <content>
    <locator>
      <provider>provider-5</provider>
      <programme>programme-5</programme>
      <subcategory>subcategory-5</subcategory>
      <episode>episode-5</episode>
    </locator>
    <keywords>
      <keyword>key-51</keyword>
      <keyword>commonkey</keyword>
    </keywords>
    <dateto>2010-03-11T10:25:30.541+02:00</dateto>
```

```
      <lang>EL</lang>
    </content>
  </content-item>
</message>
```

XPath Query:

```
/ns0:content-item[( ns0:content/ns0:locator/ns0:provider='provider-1' or
ns0:content/ns0:locator/ns0:provider='provider-2' or ns0:content/ns0:locator/ns0:provider='provider-
3' or ns0:content/ns0:locator/ns0:provider='provider-4' or
ns0:content/ns0:locator/ns0:provider='provider-5' ) and (
ns0:content/ns0:locator/ns0:programme='programme-1' or
ns0:content/ns0:locator/ns0:programme='programme-3' or
ns0:content/ns0:locator/ns0:programme='programme-5' ) and (
ns0:content/ns0:locator/ns0:category='category-2' or ns0:content/ns0:locator/ns0:category='category-
4' ) and ( ns0:content/ns0:locator/ns0:subcategory='subcategory-1' or
ns0:content/ns0:locator/ns0:subcategory='subcategory-3' or
ns0:content/ns0:locator/ns0:subcategory='subcategory-5' ) and (
ns0:content/ns0:locator/ns0:series='series-2' or ns0:content/ns0:locator/ns0:series='series-4' ) and
( ns0:content/ns0:locator/ns0:episode='episode-1' or ns0:content/ns0:locator/ns0:episode='episode-3'
or ns0:content/ns0:locator/ns0:episode='episode-5' ) and (ns0:content/ns0:keywords/ns0:keyword =
'key-11' or ns0:content/ns0:keywords/ns0:keyword = 'commonkey' or
ns0:content/ns0:keywords/ns0:keyword = 'key-20' or ns0:content/ns0:keywords/ns0:keyword = 'key-22'
or ns0:content/ns0:keywords/ns0:keyword = 'key-31' or ns0:content/ns0:keywords/ns0:keyword =
'commonkey' or ns0:content/ns0:keywords/ns0:keyword = 'key-40' or
ns0:content/ns0:keywords/ns0:keyword = 'key-42' or ns0:content/ns0:keywords/ns0:keyword = 'key-51'
or ns0:content/ns0:keywords/ns0:keyword = 'commonkey')]/ns0:content
```

9.3 P2P Authentication Messages

9.3.1 Server Certificate Acquisition

SIP Message from the application server to the client after its registration.

```
MESSAGE sip:joe@10.147.65.111:5062 SIP/2.0
Max-Forwards: 15
Vitalpp: P2PA OfferCert
P-Asserted-Identity: <sip:joe@open-ims.test>
To: <sip:joe@open-ims.test>
CSeq: 1 MESSAGE
Via:
   SIP/2.0/UDP 10.147.160.42:4060;
      branch=z9hG4bK66ff.9490aaa1.0,
   SIP/2.0/UDP 10.147.160.42:6060;received=10.147.160.42;rport=6060;
      branch=z9hG4bK66ff.06d6bc46.0;i=1,
   SIP/2.0/TCP 10.147.65.111:5060;
      branch=z9hG4bKdaac28ceafc1fc0d4f82bc3d3dbe01ac1e08
Content-Type: text/xml
Call-ID: 10.147.65.111_2_6309805174015877988
From: <sip:Vitalpp@open-ims.test>;tag=g6f8sc3x-4
P-Called-Party-ID: <sip:joe@open-ims.test>
Content-Length: 1252

<?xml version="1.0"?>
    <vitalpp-cert>
<publickey>308201b73082012c06072a8648ce3804013082011f02818100fd7f53811d75122952d
f4a9c2eece4e7f611b7523cef4400c31e3f80b6512669455d402251fb593d8d58fabfc5f5ba30f6c
```

```
b9b556cd7813b801d346ff26660b76b9950a5a49f9fe8047b1022c24fbba9d7feb7c61bf83b57e7c
6a8a6150f04fb83f6d3c51ec3023554135a169132f675f3ae2b61d72aeff22203199dd14801c7021
5009760508f15230bccb292b982a2eb840bf0581cf502818100f7e1a085d69b3ddecbbcab5c36b85
7b97994afbbfa3aea82f9574c0b3d0782675159578ebad4594fe67107108180b449167123e84c281
613b7cf09328cc8a6e13c167a8b547c8d28e0a3ae1e2bb3a675916ea37f0bfa213562f1fb627a012
43bcca4f1bea8519089a883dfe15ae59f06928b665e807b552564014c3bfecf492a0381840002818
02dd45eb5afb3c8dc1a92ac255a771827e80f26455d6a413a8bfd22e9919ee66056ae4a481f7c162
7f24ac09804c31c60e2fcb79c4966ac2445319b07d98b59a4c6e7ef43504944054095ea1adb9e711
1b04968ce5cb375713066253e59a8a0d889ec3cfca69f00eea131e5d04e1b65d509e453ed020dfba
6eb9d28b136c0fb27</publickey>
        <attributes>
            <avp><name>identity</name><value>sip:Vitalpp@open-
ims.org</value></avp>
            <avp><name>signer</name><value>Vital++-Root</value></avp>
        </attributes>
<signature>302c02140a1aa1c289d39efcc86c6a0d20bd78414193b26e021404c21d12e8e16be62
87d3a843d2fde2f2b52093a</signature>
</vitalpp-cert>
```

9.3.2 Client Certificate Authorization

SIP Message holding an unsigned certificate from client, as it is being sent to the server for signing.

```
MESSAGE sip:10.147.65.111:5060 SIP/2.0
Call-ID: bf9b33c8c820c8ce533ff02a4820d581@10.147.65.90
CSeq: 4 MESSAGE
From: <sip:alice@open-ims.test>;tag=1006
To: <sip:10.147.65.111:5060>
Via:
   SIP/2.0/UDP 10.147.65.90:5060;branch=z9hG4bKf9929ecd08ec35a1c3a157e14691c3d3
Max-Forwards: 20
Route: <sip:orig@scscf.open-ims.test:6060;lr>
VitalPP: P2PA RequestSigning
Content-Type: text/xml
User-Agent: Fokus MONSTER Version: ${project.version}
Content-Length: 1049

<vitalpp-cert>
<publickey>308201b73082012c06072a8648ce3804013082011f02818100fd7f53811d75122952d
f4a9c2eece4e7f611b7523cef4400c31e3f80b6512669455d402251fb593d8d58fabfc5f5ba30f6c
b9b556cd7813b801d346ff26660b76b9950a5a49f9fe8047b1022c24fbba9d7feb7c61bf83b57e7c
6a8a6150f04fb83f6d3c51ec3023554135a169132f675f3ae2b61d72aeff22203199dd14801c7021
5009760508f15230bccb292b982a2eb840bf0581cf502818100f7e1a085d69b3ddecbbcab5c36b85
7b97994afbbfa3aea82f9574c0b3d0782675159578ebad4594fe67107108180b449167123e84c281
613b7cf09328cc8a6e13c167a8b547c8d28e0a3ae1e2bb3a675916ea37f0bfa213562f1fb627a012
43bcca4f1bea8519089a883dfe15ae59f06928b665e807b552564014c3bfecf492a0381840002818
0706163767146329efba6295c39490bfe9bb74ae91592a09095d0b6386d7c124ef9581a9ad7a39f9
34d79ef1d5a8152d0cc299391f745ff39093f97219457b57b9f42f79911f0f782f75f0cdc4c4dcf5
6185602113f417b872794303a7895ed4fb68de911c7d25861991f2778ab911980aa62b6cf97b5d5d
d70f8c2db1efd155d</publickey>
  <attributes>
    <avp><name>identity</name><value>sip:alice@open-ims.test</value></avp>
  </attributes>
</vitalpp-cert>
```

This message is then answered with the following response, which holds also a signature by the application server.

```
SIP/2.0 200 OK
Content-Length: 1242
To: <sip:10.147.65.111:5060>;tag=g6kux0p9-4c
Cseq: 4 MESSAGE
Via: SIP/2.0/UDP
   10.147.65.90:5060;rport=5060;branch=z9hG4bKf9929ecd08ec35a1c3a157e14691c3d3
Content-Type: text/xml
From: <sip:alice@open-ims.test>;tag=1006
Call-Id: bf9b33c8c820c8ce533ff02a4820d581@10.147.65.90
Server: Glassfish_SIP_1.0.0

<vitalpp-cert>
<publickey>308201b73082012c06072a8648ce3804013082011f02818100fd7f53811d75122952d
f4a9c2eece4e7f611b7523cef4400c31e3f80b6512669455d402251fb593d8d58fabfc5f5ba30f6c
b9b556cd7813b801d346ff26660b76b9950a5a49f9fe8047b1022c24fbba9d7feb7c61bf83b57e7c
6a8a6150f04fb83f6d3c51ec3023554135a169132f675f3ae2b61d72aeff22203199dd14801c7021
5009760508f15230bccb292b982a2eb840bf0581cf502818100f7e1a085d69b3ddecbbcab5c36b85
7b97994afbbfa3aea82f9574c0b3d0782675159578ebad4594fe67107108180b449167123e84c281
613b7cf09328cc8a6e13c167a8b547c8d28e0a3ae1e2bb3a675916ea37f0bfa213562f1fb627a012
43bcca4f1bea8519089a883dfe15ae59f06928b665e807b552564014c3bfecf492a0381840002818
0706163767146329efba6295c39490bfe9bb74ae91592a09095d0b6386d7c124ef9581a9ad7a39f9
34d79ef1d5a8152d0cc299391f745ff39093f97219457b57b9f42f79911f0f782f75f0cdc4c4dcf5
6185602113f417b872794303a7895ed4fb68de911c7d25861991f2778ab911980aa62b6cf97b5d5d
d70f8c2db1efd155d</publickey>
  <attributes>
    <avp><name>identity</name><value>sip:alice@open-ims.test</value></avp>
    <avp><name>signer</name><value>sip:Vitalpp@open-ims.test</value></avp>
  </attributes>
<signature>302c02140b16486c7b250935a19e54682b25c04f3cf8614f02147798ae9245e146505
41cdcca29181cce076d2e7f</signature>
</vitalpp-cert>
```

9.3.3 Authentic P2P Message Exchange

The following SIP message is used to transport a signed message, along with the sender's client certificate.

```
MESSAGE sip:bob@open-ims.org SIP/2.0
Call-ID: 004b38a41051b4661deae58204165eb1@10.147.65.90
CSeq: 5 MESSAGE
From: <sip:alice@open-ims.test>;tag=1009
To: <sip:bob@open-ims.org>
Via: SIP/2.0/UDP
   10.147.65.90:5062;branch=z9hG4bK0402ca2be335a30b98933b4e43a27f3f
Max-Forwards: 20
Route: <sip:orig@scscf.open-ims.test:6060;lr>
VitalPP: P2PA message
Content-Type: multipart/mixed; boundary="vitalpp-separator-XFDLJFIX"
User-Agent: Fokus MONSTER Version: ${project.version}
Content-Length: 1606
```

```
--vitalpp-separator-XFDLJFIX
Content-Type: text/plain

Hello, you received a signed and very important message!
--vitalpp-separator-XFDLJFIX
Content-Type: text/xml

<message-signature>
302c02142f21d8cf62450188be10edd0dd0ae1cbcc47e93f0214123753a0a896a92efd4b298fad47
bf5708f4aa6d
</message-signature>

--vitalpp-separator-XFDLJFIX
Content-Type: text/xml

<vitalpp-cert>
<publickey>308201b73082012c06072a8648ce3804013082011f02818100fd7f53811d75122952d
f4a9c2eece4e7f611b7523cef4400c31e3f80b6512669455d402251fb593d8d58fabfc5f5ba30f6c
b9b556cd7813b801d346ff26660b76b9950a5a49f9fe8047b1022c24fbba9d7feb7c61bf83b57e7c
6a8a6150f04fb83f6d3c51ec3023554135a169132f675f3ae2b61d72aeff22203199dd14801c7021
5009760508f15230bccb292b982a2eb840bf0581cf502818100f7e1a085d69b3ddecbbcab5c36b85
7b97994afbbfa3aea82f9574c0b3d0782675159578ebad4594fe67107108180b449167123e84c281
613b7cf09328cc8a6e13c167a8b547c8d28e0a3ae1e2bb3a675916ea37f0bfa213562f1fb627a012
43bcca4f1bea8519089a883dfe15ae59f06928b665e807b552564014c3bfecf492a0381840002818
06269b38adc18f3c57312d1efd58dba483ff90cad17769d9c3a59984829e2cd627fcd6cd3d6ec797
14392da6bf435b017f2f486de152a68470fcc4c96e8a65a569bd18f7467244ecea5bd6b81198dd55
dd8721c7ed4f484f4e71316123e17c85700a48ee6b6aeba272df64a0876b2d0574764a5c22cbf8ae
31ad07a0acaede272</publickey>
  <attributes>
    <avp><name>identity</name><value>sip:alice@open-ims.test</value></avp>
    <avp><name>signer</name><value>sip:Vitalpp@open-ims.test</value></avp>
  </attributes>
<signature>302c0214661113c1276c9518eecfc064fc9dd4aaaffa477b021468d30a3d61699b447
12a8fddcc4a2be724161922</signature>
</vitalpp-cert>
```

9.3.4 Diffie-Hellman Key Agreement

For reference, the Diffie-Hellman key agreement process is depicted in the following table as sequence of operations.

Step	Server	Client
1	Chooses a prime p and an integer g.	
2	computes $A=g^a \bmod p$	
3	sends A, g and p to the client	
4	computes $K_s=A^b \bmod p$	computes $B=g^b \bmod p$
5		computes $K_c=B^a \bmod p$

$K_c=K_s$, because

$$K_s = A^b \bmod p = (g^a \bmod p)^b \bmod p$$
$$= g^{ab} \bmod p$$
$$= (g^b \bmod p)^a \bmod p = B^a \bmod p = K_c$$

Thus, both entities have the same key material and can use it to create a common symmetric key, which can then be used e.g. for AES encryption.