Project Number:	**FP7-257123**
Project Title:	**CONVERGENCE**
Dissemination Level:	**Public**
Deliverable Number:	**D5.3**
Contractual Date of Delivery to the CEC:	**31.07.2012**
Actual Date of Delivery to the CEC:	**30.10.2012**
Title of Deliverable:	**Final protocol architecture**
Workpackage contributing to the Deliverable:	**WP5**
Nature of the Deliverable:	**Prototype**
Editors:	**Angelos-Christos Anadiotis, Charalampos Patrikakis, Iakovos Venieris**
Author(s):	**Angelos-Christos Anadiotis (ICCS), Aziz Mousas (ICCS), Dimitra Kaklamani (ICCS), Georgios Lioudakis (ICCS), Konstantinos Papadopoulos (ICCS), Nikolaos Dellas (ICCS), Vasso Giotopoulou (ICCS), Thomas Huebner (MORPHO), Carsten Rust (MORPHO), Mihai Tanase (UTI), Andrea Detti (CNIT), Matteo Pomposini (CNIT), Nicola Blefari Melazzi (CNIT), Fernando Almeida (INESC)**
Abstract:	**D5.3 is classified in the DoW as a prototype. The D5.3 prototype comprises: i) the software implementing the CONVERGENCE middleware protocols and technology engines APIs and ii) the software implementing the CONVERGENCE network protocols and algorithms and APIs.** **Following the completion of the D5.3 prototype, the consortium decided to issue this report, considered complementary to the prototype.** **The report, bearing the same name as the prototype, describes the conceptual final CONVERGENCE protocol architecture including the middleware protocols and**

	technology engines designed and implemented by the project. Some of the engines are included in MPEG-M, while others are CONVERGENCE-specific. The deliverable explains the CONVERGENCE user registration/authentication and licensing/authorization schemes and provides a complete description of the protocols for the CONVERGENCE protocol and technology engines. Finally, it describes the conceptual CONET routing architecture and protocols.
Keyword List:	**Protocols, Engines, Security, Video Streaming**

Executive Summary

According to the DoW, D5.3 is a prototype. This document, which describes the final CONVERGENCE middleware and network protocol architecture, should thus be considered as an accompanying report, destined to assist engineers/developers in understanding the design of the CONVERGENCE protocol, and middleware architecture.

The approach followed in the deliverable is the same used in deliverable D5.2, where the intermediate middleware and protocol architecture was described. We begin by describing the general protocol architecture for the middleware, going on to outline the CONVERGENCE user registration/authentication and licensing/user authorization schemes, going on to specify the APIs for the CONVERGENCE Protocol and Technology Engines. CONVERGENCE-specific engines and protocols are described in full. For engines and protocols included in the MPEG standard we offer references to the appropriate documentation. The report concludes with a description of CONET support for cooperative streaming, one of the most important results obtained by the project.

Readers will note that most of the information presented in this deliverable was already present in D5.2 – the description of the intermediate architecture. D5.3 replaces this document describing the final version of the architecture.

INDEX

1 Introduction

D5.3 is classified in the DoW as a prototype. The D5.3 prototype comprises: i) the software implementing the CONVERGENCE middleware protocols and technology engines APIs and ii) the software implementing the CONVERGENCE network protocols and algorithms and APIs.

Following the completion of the D5.3 prototype, the consortium decided to issue this report, considered complementary to the prototype.

The report, bearing the same name as the prototype, describes the final version of the CONVERGENCE middleware and network protocol architecture. We begin by describing the general protocol architecture for the middleware, going on to outline the user registration/authentication and licensing/user authorization schemes, which are shared by all the current CONVERGENCE applications. We then describe the protocols for the Protocol and Technology Engines. CONVERGENCE specific engines and protocols are described in full. For engines and protocols included in the MPEG standard we offer references to the appropriate documentation. The report concludes with a description of CONET support for cooperative streaming, one of the most important results obtained by the project.

2 Glossary

Term	Definition
Access Rights	Criteria defining who can access a VDI or its components under what conditions.
Advertise	Procedure used by a CoNet user to make a resource accessible to other CoNet users.
Application	Software, designed for a specific purpose that exploits the capabilities of the CONVERGENCE System.
Business Scenario	A scenario describing a way in which the CONVERGENCE System may be used by specific users in a specific context or, more narrowly, a scenario describing the products and services bought and sold, the actors concerned and, possibly, the associated flows of revenue in such a context.
CA	Central Authority
CCN	Content Centric Network
Cl_Auth_SC	Client Authentication with Smart Card (Challenge Response)
Cl_Auth_User_Pw	Client Authentication with Username and Password
Clean-slate architecture	The CONVERGENCE implementation of the Network Level, totally replacing existing IP functionality. See “Integration Architecture” and ”“Overlay Architecture” and “Parallel Architecture”.
CoApp	The CONVERGENCE Application Level.
CoApp Provider	A user providing Applications running on the CONVERGENCE Middleware Level (CoMid).
CoMid	The CONVERGENCE Middleware Level.
CoMid Provider	A user providing access to a single or an aggregation of CoMid services.
CoMid Resource	A virtual or physical object or service referenced by a VDI, e.g. media, Real World Objects, persons, internet services. It has the same meaning of “Resource” and it is used only to better specify the term “Resource” when there is a risk of a misunderstanding with the term “CoNet Resource”.
Community Dictionary	A CoMid Technology Engine that provides all the matching

Service (CDS)	concepts in a user's subscription, search request and publication.
CoNet Provider	A user providing access to CoNet services, i.e. the equivalent of an Internet Service Provider.
CoNet Resource	A resource of the CoNet that can be identified by means of a name; resources may be either Named-data or a Named service access point.
Content-based resource discovery	A user request for resources, either through a subscription or a search request to the CONVERGENCE system (from literature). See "subscription" and "search".
Content-based Subscription	A subscription based on a specification of user's preferences or interests, (rather than a specific event or topic). The subscription is based on the actual content, which is not classified according to some predefined external criterion (e.g., topic name), but according to the properties of the content itself. See "Subscription" and "Publish-subscribe model".
Content-centric	A network paradigm in which the network directly provides users with content, and is aware of the content it transports, (unlike networks that limit themselves to providing communication channels between hosts).
CONVERGENCE Applications level (CoApp)	The level of the CONVERGENCE architecture that establishes the interaction with CONVERGENCE users. The Applications Level interacts with the other CONVERGENCE levels on behalf of the user.
CONVERGENCE Computing Platform level (CoComp)	The Computing Platform level provides content-centric networking (CoNet), secure handling (CoSec) of resources within CONVERGENCE and computing resources of peers and nodes.
CONVERGENCE Core Ontology (CCO)	A semantic representation of the CoReST taxonomy. See "CONVERGENCE Resource Semantic Type (CoReST)"
CONVERGENCE Device	A combination of hardware and software or a software instance that allows a user to access Convergence functionalities
CONVERGENCE Engine	A collection of technologies assembled to deliver specific functionality and made available to Applications and to other Engines via an API
CONVERGENCE Middleware level (CoMid)	The level of the CONVERGENCE architecture that provides the means to handle VDIs and their components.
CONVERGENCE	The Content Centric component of the CONVERGENCE

Network (CoNet)	Computing Platform level. The CoNet provides access to named-resources on a public or private network infrastructure.
CONVERGENCE node	A CONVERGENCE device that implements CoNet functionality and/or CoSec functionality.
CONVERGENCE peer	A CONVERGENCE device that implements CoApp, CoMid, and CoComp (CoNet and CoSec) functionality.
CONVERGENCE Resource Semantic Type (CoReST)	A list of concepts or terms that makes it possible to categorize a resource, establishing a connection with the resource's semantic metadata.
CONVERGENCE Security element (CoSec)	A component of the CONVERGENCE Computing Platform level implementing basic security functionality such as storage of private keys, basic cryptography, etc.
CONVERGENCE System	A system consisting of a set of interconnected devices - peers and nodes - connected to each other built by using the technologies specified or adopted by the CONVERGENCE specification. See "Node" and "Peer".
Dec_Key_Unwrap	Key Unwrapping and Content Decryption
DIDL	Digital Item Description Language
Digital forgetting	A CONVERGENCE system functionality ensuring that VDIs do not remain accessible for indefinite periods of time, when this is not the intention of the user.
Digital Item (DI)	A structured digital object with a standard representation, identification and metadata. A DI consists of resource, resource and context related metadata, and structure. The structure is given by a Digital Item Declaration (DID) that links resource and metadata.
Domain ontology	An ontology, dedicated to a specific domain of knowledge or application, e.g. the W3C Time Ontology and the GeoNames ontology.
Elementary Service (ES)	The most basic service functionality offered by the CoMid.
Enc_Key_Wrap	Encryption and Key Wrapping
Entity	An object, e.g. VDIs, resources, devices, events, group, licenses/contracts, services and users, that an Elementary Service can act upon or with which it can interact.
Expiry date	The last date on which a VDI is accessible by a user of the CONVERGENCE System.

Fractal	A semantically defined virtual cluster of CONVERGENCE peers.
Group_Sig	Group Signature
ICN	Information Centric Network
Identifier	A unique signifier assigned to a VDI or components of a VDI.
Integration Architecture	An implementation of CoNet designed to integrate CoNet functionality in the IP protocol by means of a novel IPv4 option or by means of an IPv6 extension header, making IP content-aware. See “Clean-state Architecture”, “Overlay Architecture”, “Parallel Architecture”
IP	Identity Provider
License	A machine-readable expression of Operations that may be executed by a Principal.
Local named resource	A named-resource made available to CONVERGENCE users through a local device, permanently connected to the network. Users have two options to make named-resources available to other users: 1) store the resource in a device, with a permanent connection to the network; 2) use a hosting service. In the event she chooses the former option, the resource is referred to as a local named-resource.
Metadata	Data describing a resource, including but not limited to provenance, classification, expiry date etc.
MPEG eXtensible Middleware (MXM)	A standard Middleware specifying a set of Application Programming Interfaces (APIs) so that MXM Applications executing on an MXM Device can access the standard multimedia technologies contained in the Middleware as MXM Engines.
MPEG-M	An emerging ISO/IEC standard that includes the previous MXM standard.
Multi-homing	In the context of IP networks, the configuration of multiple network interfaces or IP addresses on a single computer.
Named resource	A CoNet resource that can be identified by means of a name. Named-resources may be either data (in the following referred to as “named-data”) or service-access-points (“named-service-access-points”).
Named service access point	A kind of named-resource, consisting of a service access point identified by a name. A named-service-access-point is a network endpoint identified by its name rather than by the Internet port numbering mechanism.

Named-data	A named-resource consisting of data.
Network Identifier (NID)	An identifier identifying a named resource in the CONVERGENCE Network. If the named resource is a VDI or an identified VDI component, its NID may be derived from the Identifier (see "Identifier").
Overlay architecture	An implementation of CoNet as an overlay over IP. See "Clean-state Architecture" and "Integration Architecture" and "Parallel Architecture"
Parallel architecture	An implementation of CoNet as a new networking layer that can be used in parallel to IP. See "Clean-state Architecture" and "Integration Architecture" and ""Overlay Architecture"
PKI	Public Key Infrastructure
Policy routing	In the context of IP networks, a collection of tools for forwarding and routing data packets based on policies defined by network administrators.
Principal (CoNet)	The user who is granted the right to use a *CoNet Principal Identifier* for naming its named resources. For example, the principal could be the provider of a service, the publisher or the author of a book, the controller of a traffic lights infrastructure, or, in general, the publisher of a VDI. A Principal may have several Principal Identifiers in the CoNet.
Principal (Rights Expression Language)	The User to whom Permissions are Granted in a License.
Principal Identifier (CoNet)	The Principal identifier is a string that is used in the Network Identifiers (NID) of a CoNet resource, when the NID has the form: NID = <namespace ID, hash (Principal Identifier), hash (Label)> In this approach, hash (Principal Identifier) must be unique in the namespace ID, and Label is a string chosen by the principal in such a way that hash(Label) is unique for in the context of the Principal Identifier.
Publish	The act of informing an identified subset of users of the CONVERGENCE System that a VDI is available.
Publisher	A user of CONVERGENCE who performs the act of publishing.
Publish-subscribe model	CONVERGENCE uses a content-based approach for the publish-subscribe model, in which notifications about VDIs are delivered to a subscriber only if the metadata / content of those VDIs match

	constraints defined by the subscriber in his Subscription VDI.
Real World Object	A physical object that may be referenced by a VDI.
REL	Rights Expression Language
Resource	A virtual or physical object or service referenced by a VDI, e.g. media, Real World Objects, persons, internet services.
Scope (in the context of routing)	In the context of advertising and routing, the geographical or administrative domain on which a network function operates (e.g. a well-defined section of the network - a campus, a shopping mall, an airport -, or to a subset of nodes that receives advertisements from a service provider).
Search	The act through which a user requests a list of VDIs meeting a set of search criteria (e.g. specific key value pairs in the metadata, key words, free text etc.).
Serv_Auth	Server Authentication without Smart Card
Service Level Agreement (SLA)	An agreement between a service provider and another user or another service provider of CONVERGENCE to provide the latter with a service whose quality matches parameters defined in the agreement.
Sig	Signature
Smart_Card Role_Auth_SC	Role Authentication towards Smart Card
SP	Service Provider
Subscribe	The act whereby a user requests notification every time another user publishes or updates a VDI that satisfies the subscription criteria defined by the former user (key value pairs in the metadata, free text, key words etc.).
Subscriber	A user of CONVERGENCE who performs the act of subscribing.
Timestamp	A machine-readable representation of a date and time.
Tool	Software providing a specific functionality that can be re-used in several applications.
Trials	Organized tests of the CONVERGENCE System in specific business scenarios.
Un-named-data	A data resource with no NID.
Us_Reg_IP	User Registration to Identity Provider
Us_Reg_SP	User Registration to Service Provider

User	Any person or legal entity in a Value-Chain connecting (and including) Creator and End-User possibly via other Users.
User (in OSI sense)	In a layered architecture, the term is used to identify an entity exploiting the service provided by a layer (e.g. CoNet user).
User ontology	An ontology created by CONVERGENCE users when publishing or subscribing to a VDI.
User Profile	A description of the attributes and credentials of a user of the CONVERGENCE System.
Versatile Digital Item (VDI)	A structured, hierarchically organized, digital object containing one or more resources and metadata, including a declaration of the parts that make up the VDI and the links between them.

3 Middleware Protocol Architecture

3.1 General Middleware Architecture

The CONVERGENCE middleware consists of Protocol and Technology Engines, as shown in Figure 1. The former are used to call Elementary Services (one Elementary Service for each Protocol Engine), the latter provide the necessary technology support. Apart from these two fundamental types of engine, there also exists a set of Orchestrator Engines, whose role is to orchestrate complex operations using either a single Technology Engine or multiple Technology Engines, working towards a single goal. Elementary Services often have to perform tasks that go beyond the functionality made possible by the Technology Engine APIs. As a result, each Elementary Service has its own Orchestrator Engine.

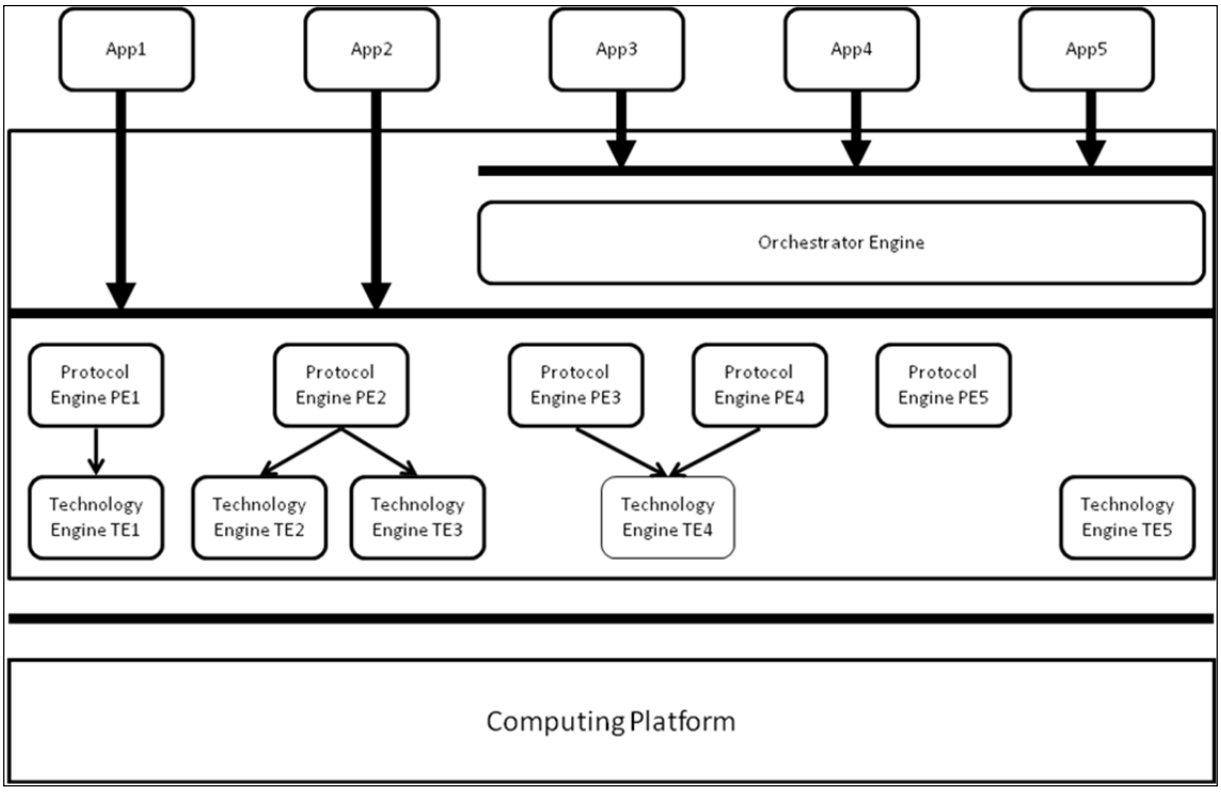

Figure 1: CONVERGENCE Middleware Architecture

3.2 Caching Content-Based Publish-Subscribe Scheme

3.2.1 Rationale

In content-based publish-subscribe systems, the subscriptions are expressed using properties that can be even subsequently matched, in whole or in part, against content (publications) [27]. Typically subscriptions are matched against future publications. In CONVERGENCE,

this functionality is extended to cover content that has already been published at subscribe time.

The CONVERGENCE middleware operates as a peer-to-peer system. The most common issue found in such systems is *scalability*. In content-based publish-subscribe over dynamic, unstructured overlays, standard approaches, followed in other publish-subscribe models (e.g. the clustering approach used in topic-based publish-subscribe), do not provide an adequate solution for this problem. To overcome this difficulty CONVERGENCE uses Caching Content-based Publish-Subscribe (CCPS). CCPS exploits search query repetition, introducing a distributed cache to store information on other peers that have subscribed with the same query. From an abstract point of view, CCPS considers that the set of publications matching a query constitute a single, shareable item of knowledge object, defined by the content of the query.

3.2.2 Protocol Specification

The proposed protocol is based on the assumption that publishers are organized in an overlay, partitioned by topic of publication, as previously explained in D3.2 and D3.3. Given that key-based approaches fail to meet requirements for partial query matching, it was not possible to use DHT or similar approaches. The idea for CCPS came from the observation that many queries are repeated [28], creating an opportunity for caching to speed up and simplify content discovery.

As shown in Figure 2, CCPS is organized in two “levels” each based on the same peers but using different protocols. The first uses a gossip communication protocol to seek matches for new queries. The second acts as a cache and is based on a DHT. At that second level, the subscriber stores in the node that corresponds to the hash of the query, the named service access points (SAP) of the other subscribers who have performed that query. The result is that the cache contains a mapping between the query and its subscribers.

When a subscriber makes a query, the system checks whether the query is already present in the cache. If it is, the system retrieves the subscription list, and adds the name of the subscriber to the cache. Otherwise, it gossips the subscription and, after receiving the results, caches the query. Since queries are repeated, the only ones that need to reach every publishers are those that are novel; the basic properties of DHTs guarantee that other queries will require at most O (logN) steps. In this way CCPS reduces the number of content discovery messages transmitted across the network. The reduction is almost equal to the caching probability and can be equal to the query repetition probability, which rises as subscription topics become more specific.

The results of cached queries are shared between peers using the BitTorrent content sharing peer to peer protocol. The content is now exchanged between the subscribers, who each carry approximately the same load (the exact load depends on the implementation of the BitTorrent protocol). The end result is better load balancing.

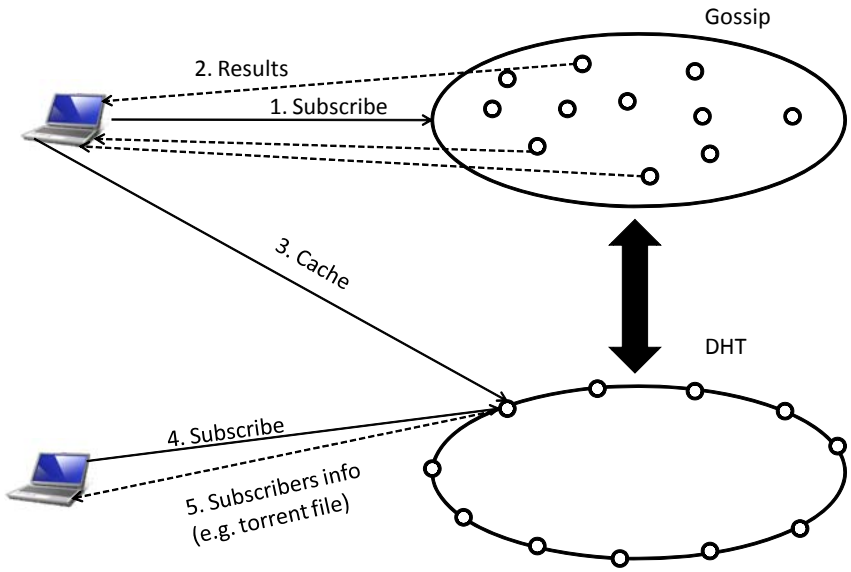

Figure 2 - Subscription to Publishers Two-Level Organization

3.3 User Registration/Authentication Scheme

Like the MPEG-M standard specification, user authentication in CONVERGENCE relies on the Security Assertion Markup Language (SAML) specification [30]. SAML does not specify specific mechanisms for verifying users' identities; rather it assumes that the security context (user identity verification) is established by external means/ protocols/technologies, and that SAML protocols are used to pass information about this security context, such as users' attributes, validity period, etc. SAML also provides Single-Sign-On (SSO) capabilities, where the Identity Providers (IdP), responsible for verifying users' credentials, may or may not be part of the CONVERGENCE ecosystem.

In CONVERGENCE, the role of the IdP is usually played by an IdentifyUser Service Provider where users can register by presenting their public key and obtaining an identifier. Every time a registered user requests to be authenticated, CONVERGENCE verifies the user's identity by verifying the public key used to obtain the user identifier and issues the appropriate SAML assertion.

However, interoperability requires that CONVERGENCE should also support non-CONVERGENCE identity providers, delegating the verification of users' credentials verification to trusted third parties, such as Google or Facebook. To achieve this, CONVERGENCE supports the OAuth 2.0 protocol[1] allowing users to access applications

[1]http://oauth.net/2/

using their Google or Facebook account, without forcing them to use CONVERGENCE identity and authentication services.

3.3.1 User Registration

User registration to CONVERGENCE uses the IdentifyUser Elementary Service[2]. To obtain an identifier, the user uses a digital certificate or a public key. The identifier is stored in a secure repository in the client device (e.g. smart card). In this way, the identifier can be used to hide the user's real identity to applications. At the same time, the identifier can still be linked back to the identity of the real user via the IdentifyUser Service Provider (always assuming the provider is trusted).

3.3.2 User Authentication

Username/password authentication schemes require the user to provide appropriate credentials. Only when these have been verified does the AuthenticateUser Elementary Service provide a SAML assertion to be stored in the client device. Key-based schemes also require interactions between the client device and the IdentifyUser service. In this case, the user identifier can be stored on a smart card, together with the private key whose public key counterpart was used to issue the identifier. When the user accesses the authentication application, the application sends a random challenge encrypted with the user's public key. This challenge is decrypted on the client side and sent back to the authentication application where it is encrypted with the applications' public key and the identifier for the user. The authentication application checks whether the identifier matches the user's public key. If so it triggers the AuthenticateUser Elementary Service, which provides a signed SAML assertion verifying the user's identity. This assertion is stored in the client device, as in the previous case.

3.4 Licensing/Authorization Scheme

CONVERGENCE middleware is aligned with the MPEG-21 and the corresponding MPEG-M Rights Expression Languages (REL) [31][32], which it uses to specify and enforce access control rules for resources referenced by VDIs. The licenses created by the middleware can grant a set of rights, as specified in the MPEG-21 Rights Data Dictionary (RDD) [33], either to a specific user or to a group. Groups are also represented by licenses. Thus, each user who is a member of this group must have a corresponding license.

The fundamental protocols supporting the licensing and authorization schemes are specified by the MPEG-M CreateLicense and AuthorizeUser Elementary Services. The CreateLicense protocol is used to create a license stating the rights of a particular user or group over a given

[2]in the OAuth case, CONVERGENCE registration may not be required. All that is necessary is that the Authenticate User Elementary Service trusts the third party Identity Provider.

resource under specified conditions. The AuthorizeUser protocol is responsible for verifying that a principal is allowed to access the resource protected by a license and, if that resource is encrypted, for providing the appropriate decryption key. Given that responsibility for access control is delegated to the AuthorizeUser Service Provider, it is essential that the Service Provider should be trusted.

The license uses a standard language (the REL) to describe the conditions that have to be to meet to access a resource. However, the license does not provide the actual protection. This comes from encryption. The license creation process includes a step that generates an encryption/decryption (i.e. a symmetric) key for the resource. This key is then used to encrypt the resource. The decryption key is included in the license. Since the license is often included in the VDI whose resource it protects, the portion containing the decryption key must be encrypted as well. This encryption is performed using the AuthorizeUser Service Provider's public key. The encryption of the resource with the key included in the license, and the encryption of the specific part of the license is performed on the application/user side.

Figure 3 illustrates this process. In the first step, the application creates a license with the access rules governing the resource, either by directly accessing the middleware or by using a licensing tool that accesses the middleware on the application's behalf. The license that is returned in the second step contains a symmetric key. In step 3, the application uses this key to encrypt the resource. In step 4, the encrypted resource is stored in a repository where it can be accessed via a URI included in the VDI. The application then encrypts the license using the trusted AuthorizeUser SP public key. Finally, in step 6 the application creates the content VDI containing the encrypted license and a reference (URI) to the encrypted resource.

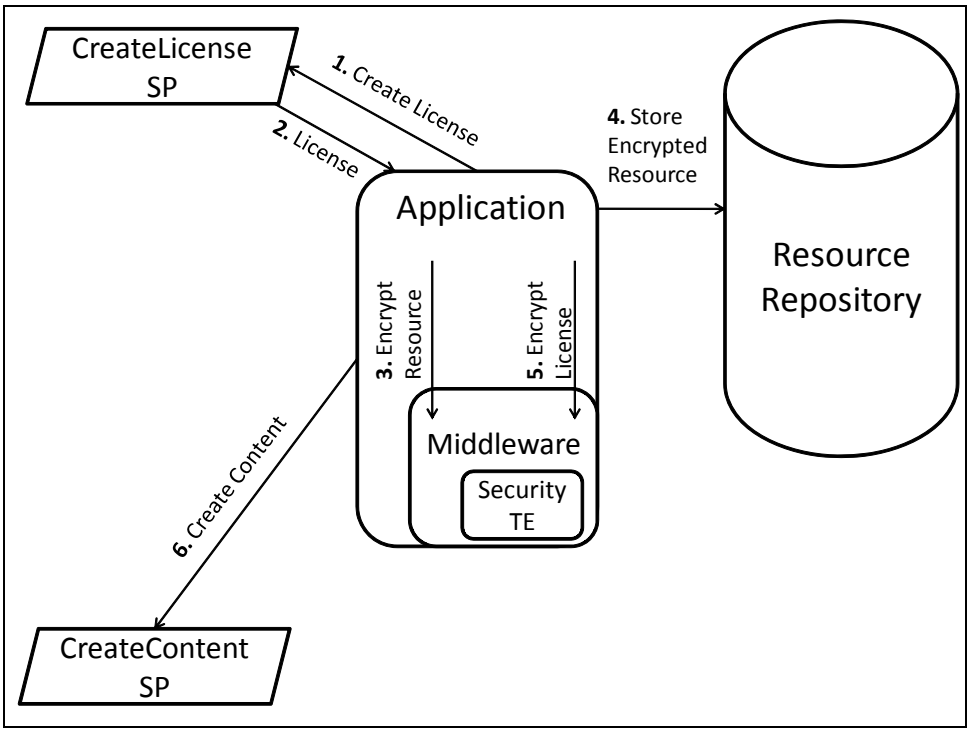

Figure 3: Resource Protection with License

Enforcement of the access control rules is performed by the AuthorizeUser SP. The protocol specifies that the entity requesting the authorization[3] passes to the SP the license protecting the content, the identifier or public key of the principal to be authorized, the right that the principal is attempting to exercise and the conditions under which the principal is attempting to exercise the right. As mentioned earlier, the license is encrypted with the public key of the AuthorizeUser SP. This SP now decrypts it and checks whether it contains a rule allowing the principal to exercise the requested right on the given resource. If this is the case, the decryption key is extracted from the license and passed to the requester in the protocol response.

Figure 4 depicts an example of an application accessing a protected resource. As a first step, the application fetches the VDI containing the resource reference from the network. It then opens the VDI, locates the license protecting the resource and requests authorization for the user who asked to access the resource. The user has previously been authenticated by the application, which therefore knows who is trying to access the resource. If the license allows the user to exercise the requested right, the AuthorizeUser SP returns the symmetric key contained in the license. Finally, the application fetches the resource from the repository, using the URI included in the VDI, decrypts it with the symmetric key and delivers it to the requesting user.

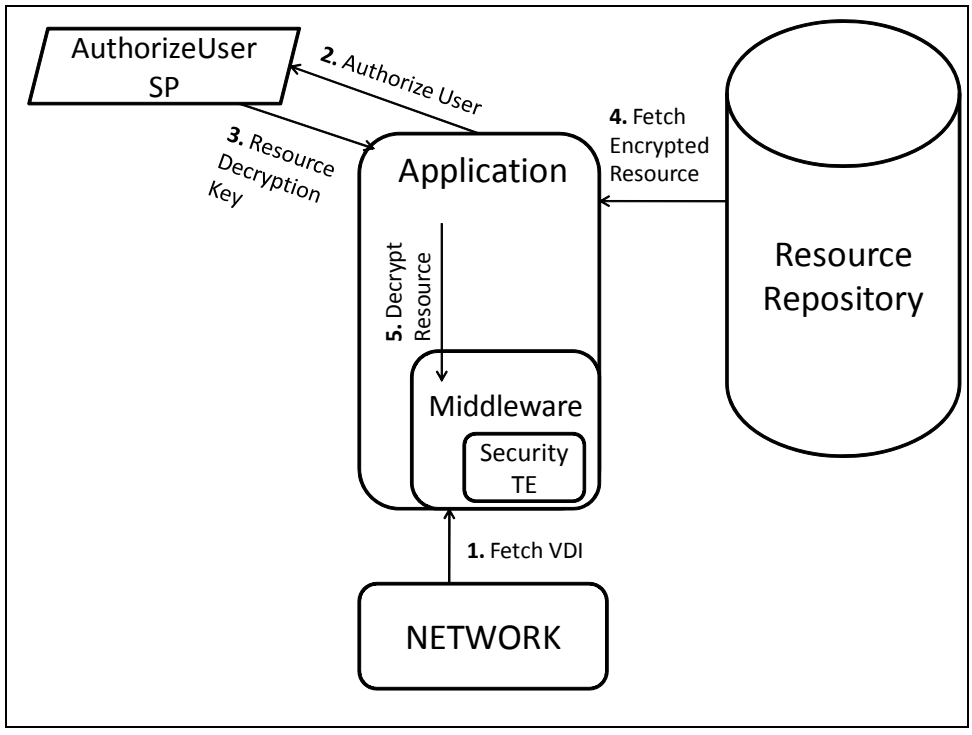

Figure 4: Single User Authorization for Resource Access

[3]This is not necessarily the user –it may also be an application that needs to check whether a particular user is allowed to access a resource.

The previous example covers the case where the license grants a set of rights to a single user. However, a license may also grant rights to a user group, in which case the requesting user must prove that she is part of the group before obtaining authorization. In this case, the authorization is granted not to the user but to the group principal. Group membership is proved by means of licenses held by each member and signed by a trusted authority.

Group member authorization involves a two-step workflow. The first step uses the CheckWithLicense Elementary Service to test whether the user is part of the group; the second uses the AuthorizeUser Elementary Service to check that the group has the requested right on the resource. The combination of the two services creates an aggregated service whose input is the request to the CheckWithLicense service and the license protecting the resource. The response is the response of the AuthorizeUser protocol. Given that a malicious user could try to get authorization by pretending to be a so-called *trusted root* (an authority which is trusted to verify that a particular user is a member of a group), the AuthorizeUser SP has to be sure that the request to the protocol request is not malicious. To achieve this, it requires that the request be signed by a trusted aggregated Service Provider. At this point, the SP follows the same procedure it would follow to authorize an individual user.

The workflow is depicted in Figure 5. In the case shown here, the application requests an aggregated Service Provider to check that a requesting user is part of a group and, then, obtains authorization for the group. In step 2, the application provides the license proving that the user is part of the group together with the license protecting the resource. The license providing group membership is used by the CheckWithLicense SP; the license protecting the resource is used by the AuthorizeUser SP. If the authorization is successful, the decryption key is returned to the application, which fetches the resource and decrypts it.

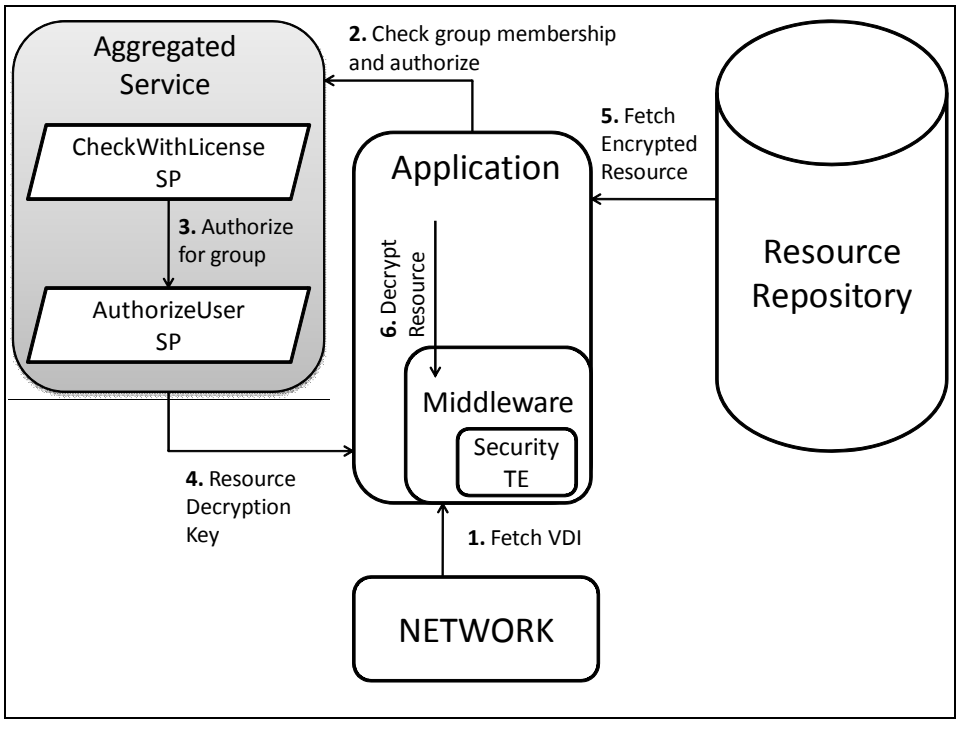

Figure 5: Group Member Authorization

3.5 Protocol and Technology Engines APIs

CONVERGENCE relies on MPEG-M for its middleware protocols. Where necessary, CONVERGENCE has proposed new protocols (Identify User, Post Content, and Revoke Content) for inclusion in MPEG-M Part 4 [32], which is currently in the final stage of standardization. Specifications of the APIs for the corresponding Protocol Engines have been included in MPEG-M Part 2 [34].

In defining the protocols for the protocol and technology engines, CONVERGENCE has followed a unified approach. Each engine is split into a schema handler (mandatory) and a technology handler (optional). The schema handler is required for performing essential operations on the schema. For example, the CreateContent Protocol Engine relies on the schema specified in MPEG-M Part 4 for the CreateContent Elementary Service. Similarly, the REL technology engine relies on the REL schema, as specified in MPEG-21 Part 5. The technology handler provides the API for the operations that a given engine can perform. Thus, a Protocol Engine technology handler provides the API for making a call (local or remote) to the corresponding Elementary Service. For example, the technology handler for the Security technology engine provides functionality for signing, encrypting, decrypting etc.

3.5.1 CONVERGENCE Engines

CONVERGENCE makes use of the following engines.

3.5.1.1 Protocol Engines

3.5.1.1.1 BasePE

BasePE		**API : MPEG-M Part 2 [34]**
Engine dependencies	DigitalItemTE	Create and parse digital items
	MetadataTE	Create and parse MPEG-7 schema
	RELTE	Create and parse REL license expressions
	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content.

3.5.1.1.2 AuthenticateContentPE

AuthenticateContentPE		**API : MPEG-M Part 2 [34]**
Relevant orchestrations	AuthenticateContentOrchestrator	
Engine dependencies	BasePE	AuthenticateContentPE extends BasePE
	OrchestratorTE	Provides the authenticate content orchestrator

3.5.1.1.3 AuthenticateUserPE

AuthenticateUserPE		API : MPEG-M Part 2 [34]
Relevant orchestrations	AuthenticateUserOrchestrator	
Engine dependencies	BasePE	AuthenticateUserPE extends BasePE
	OrchestratorTE	Provides the authenticate user orchestrator

3.5.1.1.4 AuthorizeUserPE

AuthorizeUserPE		API : MPEG-M Part 2 [34]
Relevant orchestrations	AuthorizeUserOrchestrator	
Engine dependencies	BasePE	AuthorizeUserPE extends BasePE
	OrchestratorTE	Provides the authorize user orchestrator

3.5.1.1.5 CheckWithLicensePE

CheckWithLicensePE		API : MPEG-M Part 2 [34]
Relevant orchestrations	CheckWithLicenseOrchestrator	
Engine dependencies	BasePE	CheckWithLicensePE extends BasePE
	OrchestratorTE	Provides the check with license orchestrator

3.5.1.1.6 CreateContentPE

CreateContentPE		API : MPEG-M Part 2 [34]
Relevant orchestrations	CreateContentOrchestrator	
Engine dependencies	BasePE	CreateContentPE extends BasePE
	OrchestratorTE	Provides the create content orchestrator
	IPMPTE	Create and parse Intellectual Property Management and Protection schema
	EventReportTE	Create and parse event report requests

3.5.1.1.7 CreateLicensePE

CreateLicensePE	API : MPEG-M Part 2 [34]

Relevant orchestrations	CreateLicenseOrchestrator	
Engine dependencies	BasePE	CreateLicensePE extends BasePE
	OrchestratorTE	Provides the create license orchestrator

3.5.1.1.8 DescribeContentPE

DescribeContentPE		**API : MPEG-M Part 2 [34]**
Relevant orchestrations	GetDescribeContentOrchestrator	
	SetDescribeContentOrchestrator	
Engine dependencies	BasePE	DescribeContentPE extends BasePE
	OrchestratorTE	Provides the get and set describe content orchestrators

3.5.1.1.9 IdentifyContentPE

IdentifyContentPE		**API : MPEG-M Part 2 [34]**
Relevant orchestrations	IdentifyContentOrchestrator	
Engine dependencies	BasePE	IdentfyContentPE extends BasePE
	OrchestratorTE	Provides the identify content orchestrator

3.5.1.1.10 IdentifyDevicePE

IdentifyDevicePE		**API : MPEG-M Part 2 [34]**
Relevant orchestrations	IdentifyDeviceOrchestrator	
Engine dependencies	BasePE	IdentifyDevicePE extends BasePE
	OrchestratorTE	Provides the identify device orchestrator

3.5.1.1.11 IdentifyUserPE

IdentifyUserPE		**API : MPEG-M Part 2 [34]**
Relevant orchestrations	IdentifyUserOrchestrator	
Engine dependencies	BasePE	IdentifyUserPE extends BasePE
	OrchestratorTE	Provides the identify user orchestrator

3.5.1.1.12 PackageContentPE

PackageContentPE		API : MPEG-M Part 2 [34]
Relevant orchestrations	PackageContentOrchestrator	
Engine dependencies	BasePE	PackageContentPE extends BasePE
	OrchestratorTE	Provides the package content orchestrator
	MP21FFTE	Create and parse MPEG-21 formatted files

3.5.1.1.13 PostContentPE

PostContentPE		API : MPEG-M Part 2 [34]
Relevant orchestrations	PostContentOrchestrator	
Engine dependencies	BasePE	PostContentPE extends BasePE
	OrchestratorTE	Provides the post content orchestrator

3.5.1.1.14 RequestContentPE

RequestContentPE		API : MPEG-M Part 2 [34]
Relevant orchestrations	RequestContentOrchestrator	
Engine dependencies	BasePE	RequestContentPE extends BasePE
	OrchestratorTE	Provides the request content orchestrator

3.5.1.1.15 RequestEventPE

RequestEventPE		API : MPEG-M Part 2 [34]
Relevant orchestrations	RequestEventOrchestrator	
Engine dependencies	BasePE	RequestEventPE extends BasePE
	OrchestratorTE	Provides the request event orchestrator
	EventReportTE	Create and parse event reports and event repost requests

3.5.1.1.16 RevokeContentPE

RevokeContentPE	API : MPEG-M Part 2 [34]

Relevant orchestrations	RevokeContentOrchestrator	
Engine dependencies	BasePE	RevokeContentPE extends BasePE
	OrchestratorTE	Provides the revoke content orchestrator
	ConetTE	Deletes named-resource

3.5.1.1.17 SearchContentPE

SearchContentPE		**API : MPEG-M Part 2 [34]**
Relevant orchestrations	SearchContentOrchestrator	
Engine dependencies	BasePE	SearchContentPE extends BasePE
	OrchestratorTE	Provides the search content orchestrator
	SearchTE	Create and parse MPEG Formatted Queries

3.5.1.1.18 StroreContentPE

StoreContentPE		**API : MPEG-M Part 2 [34]**
Relevant orchestrations	StoreContentOrchestrator	
Engine dependencies	BasePE	StoreContentPE extends BasePE
	OrchestratorTE	Provides the store content orchestrator

3.5.1.1.19 StoreEventPE

StoreEventPE		**API : MPEG-M Part 2 [34]**
Relevant orchestrations	StoreEventOrchestrator	
Engine dependencies	BasePE	StoreEventPE extends BasePE
	OrchestratorTE	OrchestratorTE provides the Storeevent orchestrator
	EventReportTE	Create and parse event reports and event repost requests

3.5.1.2 Technology Engines

3.5.1.2.1 CDSTE

CDSTE		API :3.5.2.1
Engine dependencies	-	

3.5.1.2.2 ConetTE

ConetTE		API :3.5.2.2
Engine dependencies	-	

3.5.1.2.3 ConvergenceMetadataTE

ConvergenceMetadataTE		API :3.5.2.3
Engine dependencies	DigitalItemTE	Create and parse structured data
	SearchTE	Create and parse MPEG formatted input queries

3.5.1.2.4 DigitalItemTE

DigitalItemTE		API : MPEG-M Part 2 [34]
Engine dependencies	EventReportTE	Create and parse event report schemas
	MetadataTE	Create and parse MPEG-7 metadata
	RELTE	Create and parse REL license expressions
	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content
	IPMP TE	Create and parse Intellectual Property Management and Protection schema

3.5.1.2.5 EventReportTE

EventReportTE		API : MPEG-M Part 2 [34]
Engine dependencies	RELTE	Create and parse REL license expressions
	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content

3.5.1.2.6 IPMPTE

IPMPTE		API : MPEG-M Part 2 [34]
Engine dependencies	RELTE	Create and parse REL license expressions
	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content

3.5.1.2.7 MatchTE

MatchTE		API :3.5.2.5
Engine dependencies	-	

3.5.1.2.8 MediaFrameworkTE

MediaFrameworkTE		API : MPEG-M Part 2 [34]
Engine dependencies	-	

3.5.1.2.9 MetadataTE

MetadataTE		API : MPEG-M Part 2 [34]
Engine dependencies	-	

3.5.1.2.10 MP21FFTE

MP21FFTE		API : MPEG-M Part 2 [34]
Engine dependencies	DigitalItemTE	Create and parse digital items
	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content

3.5.1.2.11 MpqfTE

MpqfTE		API :3.5.2.6
Engine dependencies	SearchTE	MpqfTE extends SearchTE

3.5.1.2.12 OverlayTE

OverlayTE		API : MPEG-M Part 2 [34]
Engine dependencies	DigitalItemTE	Create and parse digital items

	ConetTE	Send and receive network messages

3.5.1.2.13 RELTE

RELTE		**API : MPEG-M Part 2 [34]**
Engine dependencies	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content

3.5.1.2.14 SearchTE

SearchTE	**API : MPEG-M Part 2 [34]**
Engine dependencies	-

3.5.1.2.15 SecurityTE

SecurityTE	**API : MPEG-M Part 2 [34]**
Engine dependencies	-

3.5.1.2.16 VDITE

VDITE		**API :3.5.2.7**
Engine dependencies	DigitalItemTE	VDITE extends DigitalItemTE

3.5.1.3 Orchestrator Engines

3.5.1.3.1 OrchestratorTE

OrchestratorTE		**API : MPEG-M Part 2 [34]**
Orchestrations	Engine dependencies	
AuthenticateContentOrchestrator	DigitalItemTE	Create and parse digital items
	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content.
AuthenticateUserOrchestrator	MetadataTE	Create and parse MPEG-7 schema
	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content.

AuthorizeUserOrchestrator	RELTE	Create and parse REL license expressions
	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content.
CheckWithLicenseOrchestrator	RELTE	Create and parse REL license expressions
	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content.
CreateContentOrchestrator	MetadataTE	Create and parse MPEG-7 schema
	DigitalItemTE	Create and parse digital items
	RELTE	Create and parse REL license expressions
	IPMP TE	Create and parse Intellectual Property Management and Protection schema
	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content.
CreateLicenseOrchestrator	RELTE	Create and parse REL license expressions
GetDescribeContentOrchestrator	MetadataTE	Create and parse MPEG-7 schema
SetDescribeContentOrchestrator	MetadataTE	Create and parse MPEG-7 schema
IdentifyContentOrchestrator	DigitalItemTE	Create and parse digital items
IdentifyDeviceOrchestrator	IPMP TE	Create and parse Intellectual Property Management and Protection schema
	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content.
IdentifyUserOrchestrator	MetadataTE	Create and parse MPEG-7 schema
	SecurityTE	Create and parse XML signatures, certificates, SAML assertions and encrypted content.
PackageContentOrchestrator	DigitalItemTE	Create and parse digital items
	MP21FFTE	Create and parse MPEG-21 formatted

		files
PostContentOrchestrator	DigitalItemTE	Create and parse digital items
	OverlayTE	Post to overlay network
RequestContentOrchestrator	DigitalItemTE	Create and parse digital items
RequestEventOrchestrator	EventReportTE	Create and parse event report schemas
RevokeContentOrchestrator	DigitalItemTE	Create and parse digital items
SearchContentOrchestrator	MetadataTE	Create and parse MPEG-7 schema
	SearchTE	Create and parse MPEG formatted queries
StoreContentOrchestrator	DigitalItemTE	Create and parse digital items
StoreEventOrchestrator	EventReportTE	Create and parse event report schemas

3.5.1.3.2 ConvergenceOrchestratorTE

ConvergenceOrchestratorTE		**API :3.5.2.4**
Engine dependencies	OrchestratorTE	ConvergenceOrchestratorTE extends OrchestratorTE
Orchestrations	Engine dependencies	
ConetOrchestrator	ConetTE	Advertise and fetch resources in CoNet
EventReportOrchestrator	EventReportTE	Generate event reports
MatchOrchestrator	MatchTE	Store and match incoming subscriptions and publications
	VDITE	Parse P/S-VDIs
	ConvergenceMetadataTE	Parse Convergence Metadata
	CDSTE	Expand P/S-VDI metadata
	EventReportTE	Store P/S-VDI event report requests
	MpqfTE	Parse MPEG formatted subscription queries
MetadataOrchestrator	ConvergenceMetadataTE	Create Convergence Metadata
	MpqfTE	Create MPEG formatted subscription queries
OverlayOrchestrator	VDITE	Parse P/S-VDIs
	OverlayTE	Propagate P/S-VDIs in the semantic

		overlay
TimerOrchestrator	-	
VdiOrchestrator	VDITE	Parse P/S-VDIs
	IdentifyContentPE	Identify VDIs
	CreateContentPE	Create VDIs

3.5.2 APIs for CONVERGENCE Engine

3.5.2.1 CDSTE

The Community Dictionary Service Technology Engine (CDS TE) provides semantic services to the CONVERGENCE middleware. There are basically two types of services: management services for ontologies and exploration services for the CDS's knowledge base.

The CDSTE provides the following interfaces:

- *Entity Explorer*: an interface exposing methods to search for ontology entities stored in the CDS repository
- *Semantic Expander*: an interface exposing methods for expanding semantic metadata exploiting the CDS knowledge base
- *Knowledge Manager:* an interface exposing methods to maintain the CDS knowledge base
- *Repository Manager*: an interface exposing methods for storing and accessing meta-models
- *Inference Manager:* an interface exposing methods for reasoning over the CDS knowledge base and on demand inference over metadata
- *CommunicationManager:* an interface exposing methods to access the CDS Engine remotely.

3.5.2.2 CoNetTE

The CoNet Technology Engine (CoNet TE) provides the CONVERGENCE Middleware with access to the features of the CoNet, based on Information Centric Networking. Specifically, it provides support for:

- Storing a resource in the CoNet with a given Network Identifier (NID)
- Revoking a content with a given NID from the CoNet
- Retrieving a resource corresponding to the input NID from the CoNet
- Advertising a Service Access Point (SAP) for receiving unNamed-data, providing the callBack to be invoked when unNamed-data are received
- Sending unNamed-data towards a NID (the unNamed-data are expressed as a byte array)
- Sending unNamed-data towards a Location ID (LID)

3.5.2.3 ConvergenceMetadataTE

The ConvergenceMetadataEngine defines methods for creating and parsing the CONVERGENCE metadata schema and provides the following interfaces:

- *ConvergenceMetadata Parser*: an interface exposing methods for parsing convergence metadata
- *RdfMetadata Parser*: an interface exposing methods for parsing RDF metadata
- *ConvergenceMetadata*: an interface exposing methods for creating convergence metadata structures
- ResourceMetadata: an interface defining methods for creating and parsing R-VDI metadata
- *PublicationMetadata*: an interface defining methods for creating and parsing P-VDI metadata
- *SubscriptionMetadata*: an interface defining methods for creating and parsing S-VDI metadata

3.5.2.4 ConvergenceOrchestratorTE

The ConvergenceOrchestratorTE extends the OrchestratorTE and provides orchestrations for specific CONVERGENCE operations. The ConvergenceOrchestratorTE provides the following orchestrations:

- *ConetOrchestrator*: methods for advertising and fetching resources from CoNet
- *EventReportOrchestrator*: methods for generating event reports
- *MatchOrchestrator*: methods for parsing and matching S-VDIs with P-VDIs
- *MetadataOrchestrator*: methods for creating R/P/S-VI metadata
- *OverlayOrchestrator*: methods for handling and propagating P/S-VDIs in the semantic overlay
- *TimerOrchestrator*: methods for scheduling and running periodic middleware tasks
- *VdiOrchestrator*: methods for creating, identifying and storing VDIs

3.5.2.5 MatchTE

The Match Technology Engine (TE) is responsible for performing matches between subscriptions and publications. The Match TE offers the following interfaces:

- *Publication Manager*: an interface exposing methods for indexing publications
- *Subscription Manager*: an interface exposing methods for indexing subscriptions
- *Rdf Repository Manager*: an interface exposing methods for storing and matching Rdf metadata of publications with sparql subscription queries
- *Field Value Repository Manager*: an interface exposing methods for storing and matching field/value publication metadata with field/value subscription queries

3.5.2.6 MpqfTE

The Mpqf Engine interface defines methods for creating and parsing queries expressed in the MPEG Query Fromat. The MpqfEngine provides the following interfaces:

- *FieldValueQueryEditor*: an interface exposing methods for creating and parsing field/value queries
- *SparqlQueryEditor:* an interface defining methods for creating and parsing SPARQL queries

3.5.2.7 VDITE

The VDIEngine extends the DigitalItem engine and provides methods for creating and parsing inter-vdi relationship descriptors. The VDIEngine provides the following interfaces:

Relationship: an interface exposing methods for creating and parsing inter-vdi relationships

4 Network Protocols Architecture

4.1 Supporting P2P Content Sharing over CONET

Information Centric Networks can be seen, from an abstract point of view, as a special case of peer-to-peer networks, characterized by the introduction of caching functionality. Since content is cached to network routers, hosts can access that content on intermediate network routers, without attempting to reach the original serving node.

ICNs based on the CCNx prototype (such as CONET) have an intrinsic problem in supporting peer-to-peer protocols, like BitTorrent, owing to the way they perform routing. Consider the following example. Suppose that a set of peers wants to share a file.

The first option would be to use the network directly; that is, to advertise the file to CONET under the name ccnx:convergence/file and, then, let the interested peers make their requests using that name. Some of these requests would be served from the network caches; others would be served from the serving node directly. In this case, all the traffic is forwarded towards the serving node, except from some requests that are served from the cache.

The second option would be to use a peer-to-peer protocol. With this option, each peer would use the CONET option and advertise a service access point (SAP) to the network, for instance ccnx:peerX/p2p. Based on the protocol specification, the peers would exchange pieces and blocks as un-named data. The problem with this solution is that the data is unnamed, making it impossible to exploit caching – one of the main advantages of ICN.

We therefore propose to merge the two approaches, introducing a new option that exploits both the advantages of peer-to-peer protocols and those of ICN. Specifically, instead of exchanging un-named data, the peers name each block using a standard scheme (e.g. ccnx:convergence/file/<block_hash>). Thus, each packet passed to the network layer has two names: (a) the name of the remote peer service access point (this maintains the p2p nature of the protocol), and (b) the name of the block. The network layer uses the first (SAP) name to search its Forward Interest Base (FIB) entries, so that the block is forwarded towards the right peer and the second (block) name to store to or retrieve the block from the Content Store (CS). Going back to the previous example, each peer joining the network, advertises its service access point allowing the ICN to update the FIB. When a peer requests a block, the request will specify the SAP of the target peer (in BitTorrent, the peer is retrieved from the list sent by the tracker) and the block name. Each router receiving this request will check the CS for the block with the given name; if it is not found it will check the PIT to check whether it has already forwarded another request for the block; if so, it adds the incoming network interface of the new peer. If not, it checks the FIB for the SAP of the destination peer and forwards the request accordingly. When a peer decides to send a block to the requester, the reverse operation is performed; each router receiving the message will (a) store the block in

the CS based on the block name, and (b) check the PIT for requests for the block and forward the message to the corresponding network interfaces.

Figure 6 depicts the change in the Interest CIU format and the way the new fields are used in the context of a CONET router. In detail, the packet contains one new field, namely the named service access point of the peer that the p2p protocol has selected as the receiver of the request. The chunk name corresponds to the name of the requested block, as would be the normal case in an information centric network.

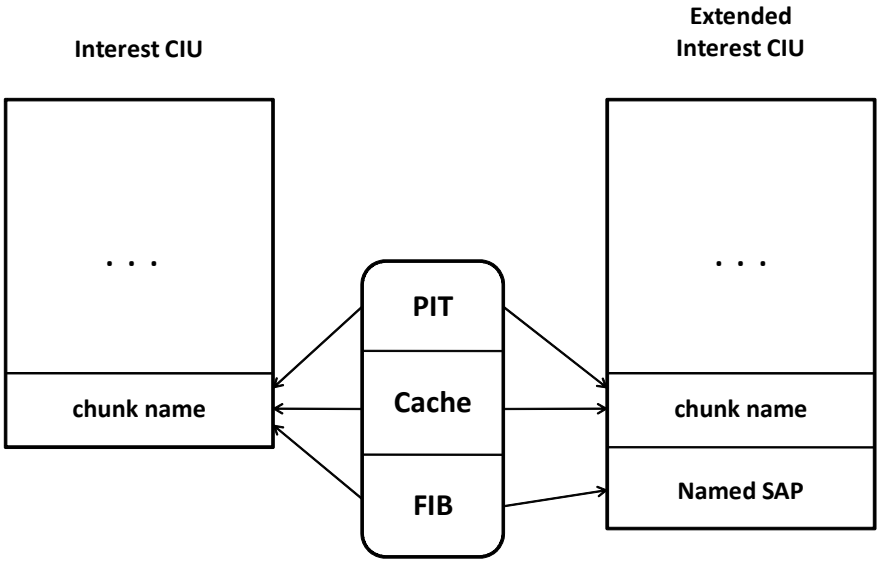

Figure 6 - Use of CONET Data Structures

The proposed approach requires some modifications in the network layer, so that it can support the two different names in the messages. Given that BitTorrent traffic contributes about 20% of total European Internet traffic[4], these modifications could yield significant advantages to ICN providers.

4.2 The CONET Lookup-and-Cache Routing Architecture

In this section, we describe the CONET Lookup-and-Cache routing architecture. The architecture was firstly introduced in D3.2. The results of the first phase of test (namely, track 2 – phase 1) are described in D8.3.

The goal of the architecture is to make it possible to deploy a worldwide CONET network using existing hardware technology. In what follows, therefore, we describe a scenario in which current Internet users use CONET to replace TCP/IP as a means of fetching web

[4] http://www.myce.com/news/us-bittorrent-traffic-well-down-but-booming-everywhere-else-61776/

content. Given the huge volume of Web content and the limited aggregability of their names within routing tables, this means we need to handle tens of billions of name-based routes. This would create two major problems if we want to use current IP routers to support ICN. First, current Forwarding Information Base (FIB) technology is unable to handle this number of name-based routes. Second, implementing a Routing Information Base (RIB) of this size would require very costly hardware. Therefore, we propose a routing-by-name architecture, named Lookup-and-Cache, where the FIB is used as a cache of routes, while the RIB is stored in a remote centralized Routing Engine.

In the following subsections we briefly present the CONET network model, estimate the number of ICN routes that CONET would have to handle in our scenario and discuss scalability issues for generic Information Centric Network. We then go on to analyse the possibility of deploying CONET with current hardware. Finally, motivated by this analysis, we present the Lookup-and-Cache Architecture.

4.2.1 CONET model

To facilitate the task of the reader, we briefly recap the CONET network model (see Figure 7) already presented in D3.2. In the model CONET nodes are interconnected by "sub-systems" [11] that can be implemented in several different ways. For instance, a sub-system could be a public or private IP network, an overlay UDP/IP link, a layer-2 network, a PPP link, etc. This is the same concept used in current IP networks, in which IP hosts and routers can be connected via different layer 2 technologies. A CONET sub-system may include CONET end-nodes (or clients) that download content, CONET serving-nodes (or servers) that provide content and CONET nodes that relay CONET data-units between sub-systems and optionally cache data.

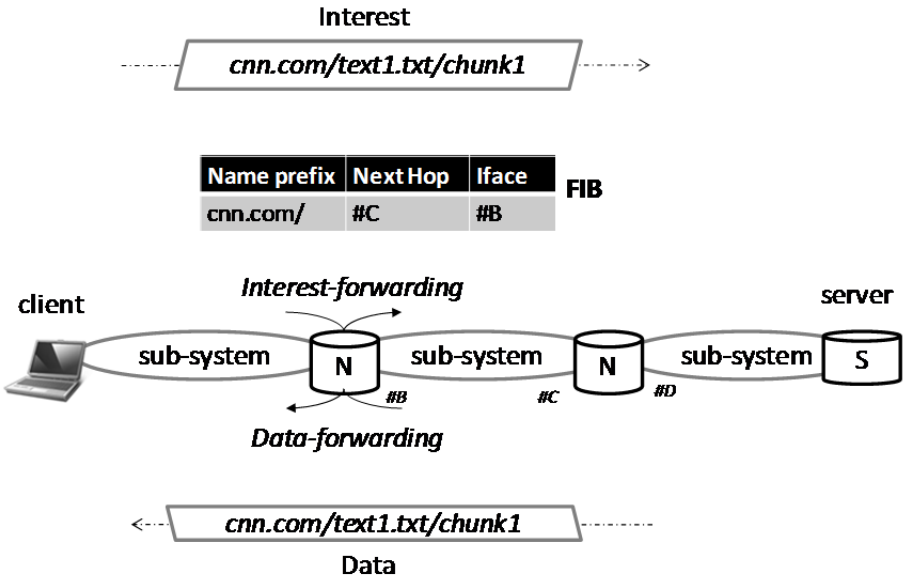

Figure 7 - CONET network model

To provide content, a server splits it into blocks of data, named *chunks*, assigning a unique network identifier to each chunk. A network identifier is a string (e.g. “cnn.com/text1.txt/chunk1”), also referred to as the “name” of the chunk.

The role of the CONET protocols is to discover and deliver named chunks [10]. To fetch a chunk, a user issues a message, in the form of an *Interest* Content Information Unit (CIU), which contains the name of the chunk. CONET nodes *route-by-name* the Interest message, using a longest prefix matching forwarding strategy and a name-based routing table. The entries in the table are *Information Centric Network (ICN) routes.* An ICN route is written in the format <name-prefix, output port identifier, next hop address>. A name-prefix should be either the full name of a chunk, e.g. “cnn.com/text1.txt/chunk1”, or a continuous part of it, starting from the first left character e.g. “cnn.com/”.

The first en-route device that has the chunk sends it back as part of a message, named *Data* CIU, which includes the chunk name. Network nodes forward the Data message towards the requesting client, through the same sequence of CONET nodes previously traversed by the Interest message. The Data forwarding process exploits reverse-path information either temporarily stored in the traversed nodes during the Interest forwarding process (see Pending Interest Table of [3]), or contained in the header of Data message, and previously collected in the Interest message during the forwarding process (see reverse-path source-routing in [11] or D3.2). In this way, the route-by-name process does not have to involve Data messages, but only Interest messages

Downloading a whole item of content is achieved by sending a *flow* of Interest messages to retrieve all the chunks of the content. The sending rate for Interest messages is regulated by a receiver-centric congestion control mechanism [5], which could be based on the same logic used by TCP. Thus, our ICN model has endpoints that exchange Interest-Data sequences. The message exchange rate is regulated by the receiver. This scheme contrasts with the scheme used in TCP/IP, where endpoints exchange Segment-Ack sequences and the exchange is regulated by the sender.

To achieve high transport performance (see D8.3), the CONET architecture further segments the Data CIUs into smaller data units, called *carrier-packets*. However, this segmentation is not relevant to the routing issue discussed here. Therefore, in what follows we will ignore carrier-packets.

4.2.2 The number of ICN routes

In this section, we estimate how many routes a CONET node would have to handle, to properly forward Interest messages. The analysis that follows is valid not only for CONET but for any Information Centric Network that uses the route-by-name paradigm. Therefore, in what follows, we will refer not to a CONET but, more generically, to an ICN.

We assume that:

i) the ICN serves Web contents

ii) current Web servers become ICN servers

iii) the ICN adopts the hierarchical component-based naming scheme described in D4.3 [5]

iv) the node lies within the "default-free" zone of the network, i.e. it does not use a default route (this is a worst case).

Figure 8 (right) shows the logical process used to derive routes for an Information Centric Network (ICN) from content-names. Figure 8 (left) shows the same process for IP. Figure 8 also summarizes the expected number of content-names, name-prefixes and ICN routes, which we will derive later on in this section. The figures for IP are derived from the BGP analysis in [19][21].

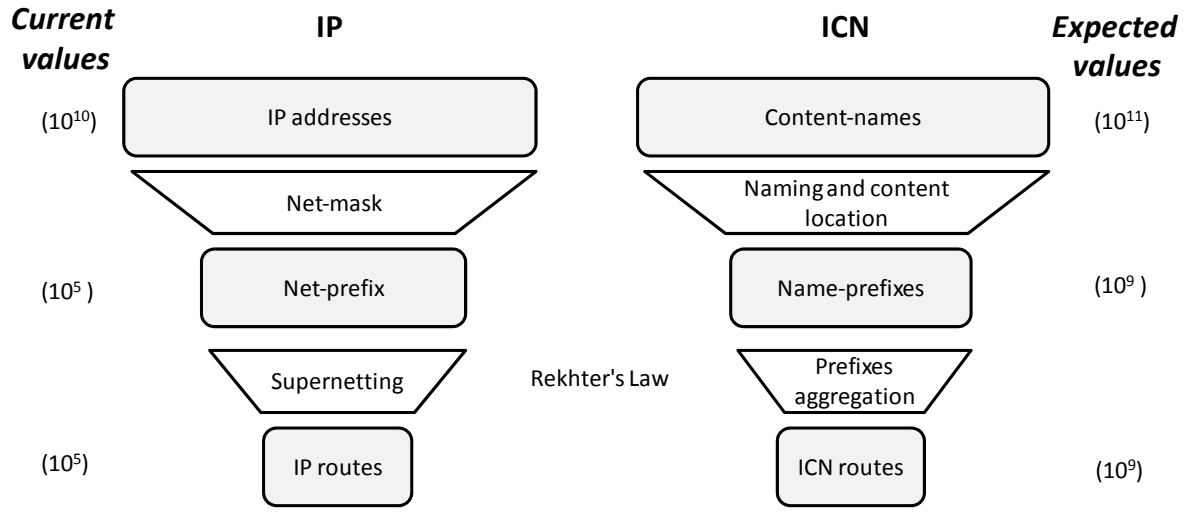

Figure 8 - From content-names to ICN routes

The left (IP) part of the figure is well understood. Therefore, in what follows, we discuss the right part.

At the top we have contents-names. Since we are assuming that the ICN will serve Web content, we set an initial target of 10^{11} content-names [2]. In reality, the current Web is believed to contain approximately 10^{10} [20]. Therefore our assumption leaves a reasonable margin of error.

The sources of content are the ICN servers, which advertise content reachability information in the routing plane. For each item of content, a server can advertise either the full sequence of Components of the content-name, such as "cnn.com/text1.txt" or a part of the sequence, such as "cnn.com". In the second case, we have an aggregation of routes and we can exploit a longest prefix forwarding strategy. We name the advertised string *name-prefix*, and we say

[5] Briefly, a name has the form Component_1/Component_2/....Component_N

that for the routing plane of an ICN (e.g. CONET), a name-prefix is equivalent to a net-prefix in the IP routing plane.

The expected number of name-prefixes depends on the number of content-names (10^{11}), on the naming scheme and on the spatial distribution of contents in the network. We use a hierarchical naming scheme, where a content-name is formed by a sequence of Components. On the routing plane, a hierarchical naming scheme enables the aggregation of many content-names in a single name-prefix. For instance, "cnn.com" is a name-prefix that aggregates all the content-names whose first Component is "cnn.com". This kind of aggregation resembles the aggregation obtained by using an IP net-mask.

The effectiveness of the aggregation of content-names in a name-prefix depends on the proximity of contents sharing a common prefix. Generally, a Web server provides all contents whose URLs have the same domain name, e.g. "cnn.com". Therefore, we assume that a CONET server provides all contents whose names contain the same domain name, i.e. the same Component_1[6]. In this case, CONET servers need to advertise only the first Component of their content names and the number of advertised name-prefixes is equal to the number of Internet domain names. In 2010, the number of second level domains was in the order of 10^8 [18]. We can therefore set an initial target of 10^9 name-prefixes[7] to be supported by CONET. This takes account of the fact that for redundancy and load sharing purposes, some domain names need to be resolved to more than one server address, (e.g., for DNS round robins or HTTP redirects).

The name-prefixes advertised by servers are inserted in the routing tables of CONET nodes as ICN routes. During this last step (Figure 8 right), it is possible to achieve a further aggregation resembling the aggregation obtained in IP, when using the super-netting technique.

To achieve efficient aggregation of routes, and ensure that a routing system is scalable with respect the number of address items, Rekhter's Law [7] states that: "Addressing can follow topology or topology can follow addressing. Choose one."

In an ICN addressing means naming. Hence, the addressing-follows-topology option would imply that content-names contain information about "where" the content is. This clearly contrasts with one of the founding principle of ICN, therefore the first option of Rekhter's Law is out of the question.

[6] We note that this assumption does not prevent to support server replication.

[7] We remark that the value 10^9 is derived by our assumptions on hierarchical naming and on proximity of contents that share a common prefix. Conversely, in case of flat-names or in case of very low proximity, then the number of name-prefixes would be higher and likely close to the number of content-names, i.e. 10^{11}.

The second option, "topology-follows-addressing" (used for example by Peer-to-peer DHT-based [23] systems), intrinsically provides a stretch of network paths. The path stretch is critical for CONET, as well for any other ICN, where signalling (i.e. content requests) and traffic have to be forwarded on the same path, so as to exploit the caches provided by en-route nodes. To avoid path-stretch and obtain shortest-path routing, the ICN topology has to follow the topology of the underlying infrastructure, rather than the addressing scheme. Therefore, the second option of Rekhter's Law is also unusable. We conclude that

In a ICN where i) content names do not include reference to location and ii) content requests are routed on the shortest-path using name-based routing tables, routing functionality cannot scale versus the number of name-prefixes advertised by servers. Consequently, a node in the DFZ should be able to handle a number of ICN routes of the same order of magnitude as the number of name-prefixes[8].

Specifically, in the Web scenario considered here, the expected number of ICN routes that a node would need handle to forward Interest messages is close to 10^9, that is the expected number of name-prefixes, i.e. domain names.

Obviously this conclusion depends on the initial assumptions defined at the beginning of this section. Changing the assumptions would change the results. For example, with "flat" non-hierarchical naming, the number of ICN routes would be higher and probably close to the number of content-names, i.e. 10^{11}. If we allow more than one route per name-prefix, e.g. for routing redundancy or multi-homing purposes, the number of ICN routes would again be higher than 10^9. In nodes with a default route, e.g. nodes corresponding to a tier-2 or a tier-3 node in the current Internet, the number of ICN routes would much be lower.

4.2.3 Deploying CONET nodes using existing technology

Routing-by-name of Interest messages is very similar to the routing of IP packets except that, routing-by-name uses name-prefixes instead of IP-prefixes. It is useful, therefore, to examine the feasibility of reusing an IP router for a CONET node. Figure 9 shows a typical architecture for a carrier-grade IP router [15][16][17]. The router is composed of three major components: one or two routing engines, line cards that host a forwarding engine and a switch fabric. The routing engine handles the routing protocols and stores the routes in a routing table, called the Routing Information Base (RIB). In general, the RIB contains several routes to the same destination and is implemented using cheap, slow memory such as DRAM.

[8] We observe that this scalability problem also occur for the IP routing of the current Internet [7] but, obviously, the problem has a lower impact since the number of IP net-prefixes advertised by border router is very lower than the expected number of name-prefixes advertised by servers of ICN.

The forwarding engine of a line card receives incoming packets and selects the output line card by looking up an on-board routing table, called the Forwarding Information Base (FIB) [13]. The FIB contains one route per destination, and thus has a smaller number of routes than the RIB. To support packet forwarding at line rate, the forwarding process is carried out by dedicated ASIC chips and the FIB is implemented with fast memories, such as SRAM or TCAM. These memories are costly, consume a lot of power, and do not follow Moore's Law [7]. After the selection of the output interface, the forwarding engine injects the packet in the switching fabric. The switching fabric is an *N*x*N* non-blocking crossbar where *N* is the number of line cards (including the routing engine).

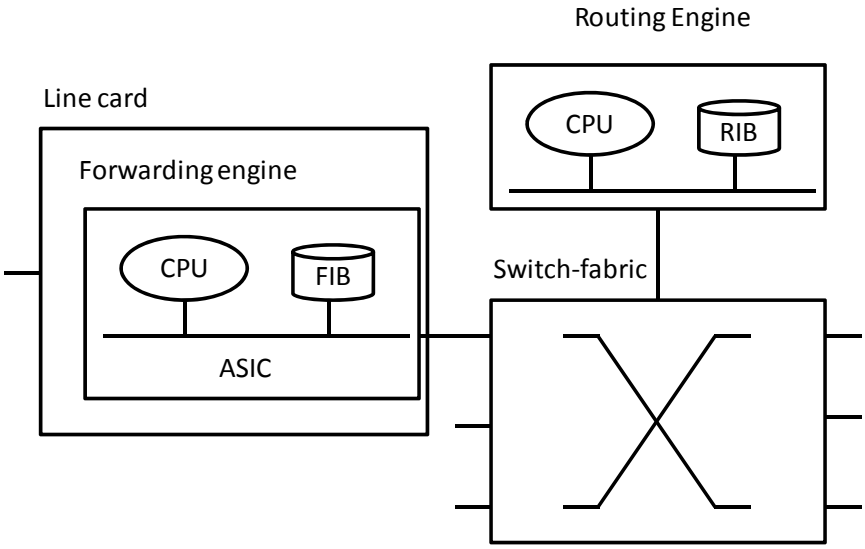

Figure 9 - Typical IP carrier-grade router architecture

If we want to reuse this architecture to route-by-name Interest messages, we should store ICN routes in the FIB and RIB, and update the routing and forwarding logic. As a first step it is useful to verify the feasibility of storing all required routes in a FIB and in a RIB.

As far as concerns the FIB, the maximum size of a modern SRAM chip is 32 MByte[6]. Assuming that an ICN routing entry occupies 45 bytes[9] [2], a FIB can store a number of routes of the order of 10^6 (i.e. 32MB/45B). In the previous section we estimated that an ICN node should handle 10^9 routes. We conclude that current FIB technology cannot store the whole set of ICN routes.

Let us now analyse the RIB issue. In IP, the RIB contains more than one route per name-prefix; supporting peering relationships among Autonomous Systems. For instance, BGP data from the AS6447 node [19] show that its RIB contains an average of 31 routes per destination. As a consequence, we assume that the RIB of an ICN node should handle approximately 10^{10} routes i.e. one order of magnitude more than the number of name-prefixes

[9] An ICN routing entry is the tuple <name-prefix, output port identifier, next-hop address>. We assume that 40 bytes are for the name-prefix, 1 byte for the output port identifier and 4 bytes for the next hop address.

[10]. In this case, the RIB would require hundreds of Gbytes (i.e., 10^{10} *45B) of DRAM memory and a motherboard with hundreds of memory slots. Current DRAM chips provide 4 GB of member and the motherboards of "expensive" carrier-grade IP routers provide up to 4 memory slots [16][17]. This means that storing routes in the RIB would require a 100-fold increase in current capacity. Supplying each network node with a motherboard with hundreds of memory slots would dramatically increase the cost of deploying CONET, with respect to an IP network.

4.2.4 The Lookup-and-Cache routing architecture

In this section we discuss a routing architecture that copes with these issues. Although the architecture does not solve all ICN scalability issue, it nonetheless makes it possible to deploy a worldwide network using current technology. To make a comparison, IP also has a scalability problem [7]. Nevertheless the memory capacity provided by current SRAM technology is sufficient to have a working Internet.

4.2.4.1 Data Plane

To cope with the FIB capacity issue and the RIB cost issue, we propose a *Lookup-and-Cache* routing architecture. In our solution, we use the FIB of a Forwarding Engine as a *route cache* and deploy a *centralized routing engine* that logically serves all the nodes of a sub-system. Figure 10 describes a typical sequence of Lookup-and-Cache operations. Node *N* receives an Interest message for "ccn.com/text1.txt/chunk1". Since the FIB lacks the related route, the node temporarily queues the Interest message, looks up the route in a remote RIB, gets the routing information and stores it in the FIB. It can then forward the Interest message.

In what follows, we discuss the rationale underlying this approach.

[10] As pointed out by a Project Reviewer, the spread factor between RIB and FIB may be even greater. Indeed, in a ICN the same name-prefix could be advertised by different servers, distributed in different location of the network. A similar phenomenon occurs in current IP in case of multi-homing but the expected impact on RIB-FIB spread is lower, as the number of net-prefixes is lower than the expected number of name-prefixes.

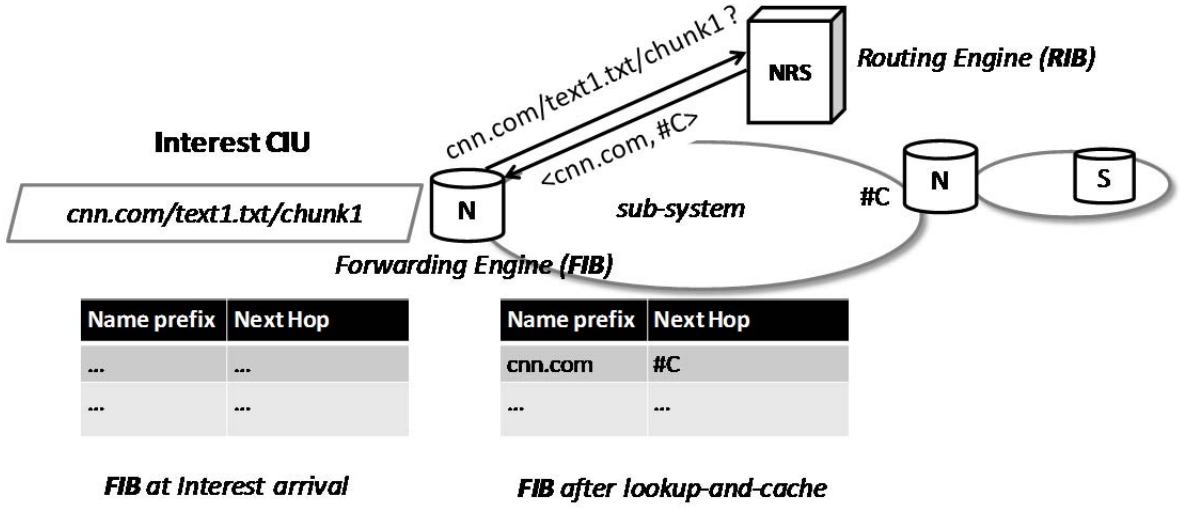

Figure 10 - Lookup and cache concept

FIB as a route cache– It is well-known that the relative frequency with which Web contents are requested follows Zipf's law [22] and that interests in specific Web content are restricted in time and space. Therefore, a large proportion of *flows* of Interest messages that a CONET node would have to *concurrently* route-by-name refer to a small set of content. What is more, given that in our model ICN routes address servers rather than single items of content (see Section 4.2.2), these flows use an even much smaller set of ICN routes. In D8.3 we show that all these *active-routes* can be comfortably stored in SRAM memory. Therefore, we propose to use the FIB of a CONET node as a *route cache*, which should contain, at a minimum, the entire set of active-routes[11]. When the FIB lacks a route, the node looks up the route in a "remote" RIB and caches the route in its FIB. When all FIB rows are filled in, new routing entries replace old ones, according to a specific *route replacement algorithm*. Routing entries can also be removed or updated through a *FIB-RIB consistency mechanism*.

Centralized Routing Engine–In our model, all ICN routes are contained in the RIB of a Routing Engine, which logically serves all nodes of a sub-system and runs on a centralized server[12], called the *Name Routing System* (NRS) node. Thus, only one network device requires an expensive Routing Engine.

Since many Interest flows use a small set of active-routes, the temporal dynamics of active-routes is *slower* than the flow dynamics. This means that routes can be used for longer than the duration of a single flow. This limits the lookup rate that a centralized Routing Engine has to support. D8.3 shows that it can easily be supported by current technologies.

[11] In addition, we could use pre-fetching algorithms to fill in unused FIB entries with most popular ICN routes. Pre-fetching aims at reducing the delay suffered by Interest messages for lookup procedures. ICN routes to be pre-fetched could be chosen based on a statistical analysis on lookup requests measured at the NRS node.

[12] We observe that the centralized routing engine could be a service offered by a *Cloud*

We conclude the section by observing that, as it occurs in the current Internet for BGP messages, the CONET nodes should give highest priority to routing signalling (e.g., lookup and routing messages), to limit the number of failed communication attempts and the delay.

4.2.4.2 Routing Plane

So far we have described the "data-plane", i.e. the procedures carried out to forward ICN messages. However, the Lookup-and-Cache architecture (as the IP one) also needs "routing-plane" procedures that run on NRS nodes and whose goal is to setup the RIBs.

On the routing plane, the dissemination of name-prefixes is based on the REGISTER and UNREGISTER functions proposed by the DONA architecture [2], adapted to our specific ICN model.

As show in Figure 11, a CONET server (SN) REGISTERs and UNREGISTERs the name-prefixes (e.g., "cnn.com") of the content it provides in the local NRS node. The local NRS node and the next ones forward the REGISTER/UNREGISTER messages toward their parents and peers until they reach a tier-1 NRS. Similarly to Interest messages, REGISTER/UNREGISTER messages are routed by name. Each NRS node has a Service Access Point (SAP) used to receive routing messages. Like content, CONET addresses SAPs by name, e.g. "ss1.org/routing.sap" (see [11] and deliverable D3.2). ICN routes toward the SAPs of the peer NRS nodes are permanently stored within the nodes' FIBs.

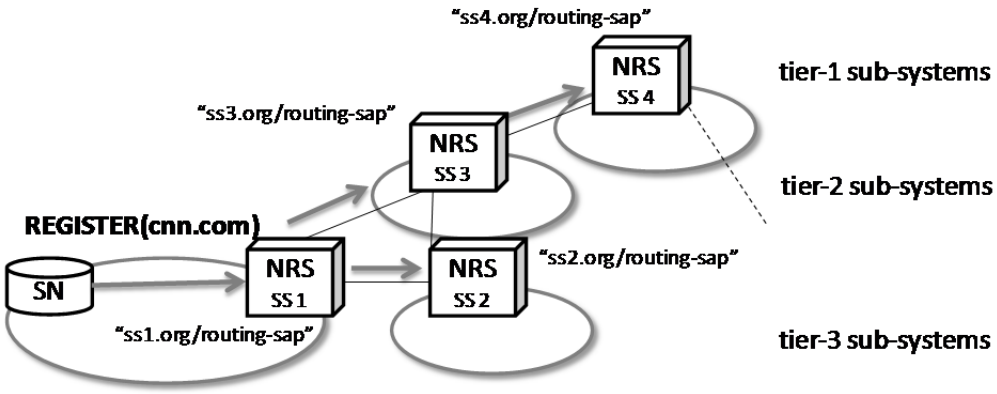

Figure 11 - Lookup-and-Cache Routing Plane

4.2.4.3 Route Replacement Algorithm

When a node receives an Interest message for a given content and it is not possible to find a matching route in the FIB, we have a *route-cache-miss* event. In this case the node can lookup the route in the remote RIB and store it in the FIB. The node can then forward the Interest message and the following ones that will use the same route. In case of a route-cache-miss event, the forwarding of Interest messages will be subject to a route-lookup delay. Obviously,

if the node does not perform a route lookup in the RIB, the Interest message is dropped and, eventually, retransmitted by transport level mechanisms.

When the FIB is not full, a node always performs a lookup and the new route is added to the FIB. When the FIB is full, the insertion of a new route implies the replacement of an old route. In this condition, the route replacement algorithm decides whether to ask the NRS node for the new route to be inserted and, if so, which old route to replace.

Since the network decouples the RIB from the FIB, an inefficient route replacement algorithm would result in an excessive rate of route lookups, poor delay performance (more Interest messages would subject to the route-lookup delay) and increased load on the NRS node. To mitigate these issues, it would be desirable to replace *inactive* routes, i.e. routes not used by the traffic that the node is forwarding at a given time. Consequently, the design of the route replacement algorithm should solve two central problems: first, how to understand that a route is inactive and, second, how to behave in case of *FIB overload*, i.e. when there are no inactive routes and a new route needs to be added in the FIB to forward an incoming Interest message.

Below, we: i) propose a novel but simple route replacement algorithm, based on the estimation of an Inactivity Time Out (ITO); ii) analyse the Least Recently Used (LRU) replacement strategy, to understand how LRU deals with these two problems; iii) study the relationship between the route replacement strategy and the fulfilment of Quality of Service requirements. For an evaluation of the performances of these two strategies the reader is referred to D8.3.

4.2.4.4 Inactivity Time Out (ITO) - route replacement algorithm

The ITO algorithm assumes that each route contained in the FIB has an inactivity time out (ITO), after which the route is considered inactive. At the current time `now`, the route is considered inactive when the time between `now` and the arrival of the last Interest message for the route is greater than the inactivity timeout. The timeout value is evaluated, using the same algorithm used to define the TCP retransmission time out [14], but replacing round-trip-time measurements with measurements of inter-arrival times for two consecutive Interest messages along the same route (see the next paragraph).

When it encounters a FIB overload, the ITO algorithm is *partly non-preemptive*. An active-route cannot be removed from the FIB, unless the FIB has been in a “pressuring” state for a period of time greater than a Maximum Pressuring Time (MPT).

The pressuring state starts when it becomes impossible to enter new route in the FIB and ends when a new route is inserted. This avoids the risk that active routes in the FIB block new traffic flows for long periods of time.

The ITO algorithm works as follows. When a new route needs to be inserted there are three possibilities: i) if there are inactive-routes (*FIB underload*), the least recently used inactive route is replaced by the new route; ii) if there are no inactive-routes (*FIB overload*) *and* the

pressuring period is lower than MPT, the new route is not inserted and the incoming Interest message is discarded; iii) if there are no inactive-routes (*FIB overload*) and the pressuring period is greater than MPT, the least recently used active-route is replaced by the new route.

The ITO algorithm may temporarily prevent the forwarding of traffic. However, the non-preemptive characteristic of the algorithm strongly limits in/out flapping of routes in the FIB during route overload conditions. As we will show in the next section, in/out flapping is harmful since it overloads the NRS node, increasing the time required by users to download content and hampering long downloads.

4.2.4.4.1 Computation of the route inactivity timeout

The computation of the route inactivity timeout (TO) requires five variables: the last arrival time (LAT), the Smoothed Inter-arrival Time (SIT), the Inter-arrival Time VARiation (ITVAR), the Message Counter (MC), the Maximum Inter-arrival Time (MIT). Using these variables, the value of the Timeout is computed using the usual TCP time out formulas [14], and applying a multiplier of 2 to avoid competition between transport level congestion control and network level timeout. In this scheme:

```
TO = 2 * (SIT + max (G, K*ITVAR))
```

where K=4 and G = 500 ms.

SIT and ITVAR are computed as follows. When the first Interest message of an inactive route or of a route not contained in the FIB is received, the variables are initialized as shown below:

```
LAT = now
SIT = ISIT
ITVAR = 0
MC = 1
MIT = 0
```

where ISIT is a default initial value for the SIT. We used ISIT = 1s in our test.

When the second Interest message of an active-route arrives the variables are updated as follows:

```
MIT = now-LAT
SIT = MIT
ITVAR = SIT/2
LAT = now
MC = 2
```

When the next Interest messages of an active-route are received, the variables are updated as shown below:

```
If PC <MCTh
      MIT = max(MIT, now-LAT)

      LAT = now
      MC = MC + 1
Else
      MIT = max(MIT, now-LAT)
      ITVAR = (1 - beta) * ITVAR + beta * |SIT - MIT|
      SIT = (1 - alpha) * SIT + alpha * MIT
      LAT = now
      MC = 0
```

where alpha=1/8, beta=1/4 and MCth = 16.

From a conceptual point of view, we are using the TCP algorithms to update SIT and ITVAR. However, in TCP the algorithm is fed with RTT samples, while here the algorithm is fed with the maximum inter-arrival time (MIT) that we measure during a window of MCth packets. The rationale underlying the use of the window is that, when RTTs are long, transport level congestion control generates bursty traffic and the throughput is RTT limited. In these cases, the inactivity timeout has to be configured so as to not elapse during the void time between consecutive bursts. On these grounds, we trust that there will always be at least two bursts, during a window of MCth. In this case, the maximum measured value of the inter-arrival time is exactly the void time between consecutive bursts, the value used to calculate SIT and ITVAR.

4.2.4.5 Least Recently Used (LRU) - route replacement algorithm

With LRU, a new route that needs to be inserted in the FIB always replaces the least recently used one. LRU does not consider whether a route is active or inactive. Thus, in case of FIB overload, LRU is *pre-emptive:* the least recently used route is replaced even if it is active.

4.2.5 Quality of Service

Quality of Service (QoS) refers to techniques used to provide specific performance levels to traffic classes with different service requirements [e.g., RFC 2990]. Examples include scheduling algorithms or access control policies devised to assure pre-defined loss and/or delay performance. Typically QoS is concerned with allocating transmission or storage

resources. In our case, we have the rather unusual problem of controlling how different traffic classes can use FIB space. The limited caching space impacts both delay and the jitter performance of traffic flows. When a new traffic flow reaches a node that does not have the corresponding ICN route in the FIB, this traffic suffers an *entry* delay composed of two components:

1) the time needed to store the route in the FIB, which depends on the replacement algorithm: in the case of LRU it is equal to zero; with ITO it depends on the state of activity of FIB entries and on the MPT timeout;
2) the time needed to carry out the lookup and cache procedure, which depends on the query processing time and on the network delay between FIB and RIB.

A traffic flow experiences the entry delay not only the first time that it reaches a node, but also each time that its route flaps in/out from the FIB, further worsening jitter performance.

This means that a QoS algorithm has to limit the value of the entry delay and the frequency of in/out flapping. As a simple example of such an algorithm, we propose a modification of the ITO algorithm, which we call priority-ITO. We discuss the performance of the priority-ITO algorithm in D8.3.

When Priority-ITO is applied to a situation of FIB overload, routes belonging to high priority traffic classes pre-empt routes belonging to low priority traffic classes. In this way, traffic of a class with priority *#K* sees the FIB space as it were used only by routes of classes with priority greater or equal to *#K*.

4.2.6 RIB-FIB consistency mechanism

Current IP routers implement a RIB-FIB consistency checker. Since RIB and FIB run in the same motherboard it is not necessary to optimize the logic. Here, we need a consistency mechanism ensuring that the information cached in the FIB is consisted with the one in the RIB; in our case, where the FIB and RIB are separate, it is essential that the mechanism makes efficient use of the path between RIB and FIB.

The problem is well-known in the area of Web caching [24] where there are several mechanisms already in use. Routing information is critical, so we need a *strong* level of consistency, i.e. the FIB must not contain stale entries. This implies that consistency mechanisms that only use expiration-time are not suitable. Given that a CONET node cannot afford to check the validity of FIB entries packet-by-packet, polling mechanisms are also unsuitable. Consequently, we propose to use an *invalidation* mechanism, where the NRS node keeps track of ICN routes cached in “its” FIBs: each time an entry of the RIB changes, the NRS notifies the related remote FIBs that the cached entry is no longer valid. The NRS keeps

track of the routes cached by a FIB thanks to lookup requests coming from the FIB. Each FIB entry has a long expiration time (ET), e.g. 10 min, after which the entry is removed. The routes cached in the FIB are the ones looked up in the last ET seconds. If their number is greater than the FIB size (*S*), the cached routes are the latest *S* looked up routes. The expiration time limits the number of unpopular/occasional routes in the FIB, reducing the frequency of invalidation notifications. Given that these typically refer to servers with unreliable (or poorly managed) connections to the rest of the Internet, it is these routes that generate most of the updates [25].

5 Bibliography

[1] D. Cheriton, M. Gritter, “TRIAD: a scalable deployable NAT-based internet architecture”, Technical Report (2000)”

[2] T. Koponen, M. Chawla, B.G. Chun, et al.: “A data-oriented (and beyond) network architecture”, ACM SIGCOMM 2007

[3] V. Jacobson, D. K. Smetters, J. D. Thornton et al., ”Networking named content”, ACM CoNEXT 2009

[4] D. Smetters, V. Jacobson: “Securing Network Content”, PARC technical report, October 2009

[5] A. Kuzmanovic, E.W. Knightly. “Receiver-Centric Congestion Control with a Misbehaving Receiver: Vulnerabilities and End-point Solutions”, Elsevier Comput. Network. 2007, 51, 2717–2737.

[6] D. Perino, M. Varvello, “A Reality Check for Content Centric Networking”, ACM SIGCOMM 2011, Workshop on Information-Centric Networking

[7] D. Meyer , L. Zhang , K. Fall , “Report from the IAB Workshop on Routing and Addressing”, IETF RFC 4984

[8] A. Ghodsi, T. Koponen, B. Raghavan, S. Shenker, A. Singla, and J. Wilcox , "Information-Centric Networking: Seeing the Forest for the Trees", in Proc. of the 10th ACM Workshop on Hot Topics in Networks (HotNets-X), Cambridge, Massachusetts

[9] S. Salsano, A. Detti, M. Cancellieri, M. Pomposini, N. Blefari-Melazzi, “Transport-layer issues in Information Centric Networks”, ACM SIGCOMM Workshop on Information-Centric Networking (ICN 2012), August 17, 2012, Helsinki, Finland

[10] B. Baccala, “Data-oriented Networking” Internet draft, IETF, August 2002.

[11] A. Detti, N. Blefari-Melazzi, S. Salsano, M. Pomposini, “CONET: A Content Centric Inter-Networking Architecture“, ACM SIGCOMM Workshop on Information-Centric Networking (ICN 2011), August 19, 2011

[12] D.C. Feldmeier, “Improving gateway performance with a routing-table cache”, in Proc. of IEEE INFOCOM 1988

[13] G. Trotter , “Terminology for Forwarding Information Base (FIB) based Router Performance”, IETF RFC 3222

[14] V. Paxson, M. Allman, “Computing TCP's Retransmission Timer”, IETF RFC 2988

[15] Xiaoliang Zhao, Member, IEEE, Dante J. Pacella, and Jason Schiller, “Routing Scalability: An Operator’s View”, IEEE Journal on Selected Areas in communications, vol. 28, no. 8, October 2010

[16] “Cisco Carrier Routing System”, available at http://www.cisco.com/en/US/prod/collateral/routers/ps5763/prod_brochure0900aecd800f8118.pdf

[17] “Juniper T Series Core Routers” available at http://www.juniper.net/elqNow/elqRedir.htm?ref=http://www.juniper.net/us/en/local/pdf/datasheets/1000051-en.pdf

[18] “Verisign 8-k Current Report", available at https://investor.verisign.com/secfiling.cfm?filingID=1193125-10-213453

[19] “BGP Routing Table Analysis Report”, available at http://bgp.potaroo.net

[20] “Daily estimated size of World Wide Web”, http://www.worldwidewebsize.com/

[21] “BGP Routing Table Analysis - DIX-IE Data”, http://thyme.apnic.net/current/

[22] L. Breslau et al., “Web Caching and zipf-like Distribution: Evidence and Implications”, in Proc. IEEE INFOCOM, 1999

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network", in Proceedings of ACM SIGCOMM, August 2001

[24] Gwertzman, J., Seltzer, M., World-Wide Web Cache Consistency Proceedings of the 1996 USENIX Technical Conference, San Diego, CA January 1996, 141-152.

[25] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, “BGP routing stability of popular destinations,” in Proc. ACM IMW, 2002.

[26] V. Jacobson, D. K. Smetters, N. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, R. Braynard, “VoCCN: voice over content-centric networks”, Proceedings of the 2009 Workshop on Re-architecting the Internet (ReArch 2009).

[27] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, “The many faces of publish/subscribe,” ACM Comput. Surv., vol. 35, pp. 114–131, June 2003.

[28] B. Smyth, E. Balfe, J. Freyne, P. Briggs, M. Coyle, and O. Boydell, “Exploiting query repetition and regularity in an adaptive community-based web search engine,” User Modeling and User-Adapted Interaction, vol. 14, pp. 383–423, Jan. 2005.

[29] R. Ahmed and R. Boutaba, “A survey of distributed search techniques in large scale distributed systems,” Communications Surveys Tutorials, IEEE, vol. 13, pp. 150 –167, quarter 2011.

[30] OASIS, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, OASIS Standard, 15 March 2005.

[31] ISO/IEC 21000-5:2004, Information technology - Multimedia framework (MPEG-21) - Part 5: Rights Expression Language.

[32] ISO/IEC FDIS 23006-4:2012(E), Information technology – Multimedia service platform technologies – Part 4: Elementary Services.

[33] ISO/IEC 21000-6:2004, Information technology - Multimedia framework (MPEG-21) - Part 6: Rights Data Dictionary.

[34] ISO/IEC CD 23006-2, Information Technology – Multimedia service platform technologies – Part 2: MPEG extensible middleware (MXM) API.