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	**Detailed Client Adaptations Specification**

Instrument:	STREP
Theme:	ICT-2-1.6
Report Due:	M9
Report Delivered:	
Lead Contractor for this deliverable:	TID
Contributors to this deliverable:	TID, UoP, RBB, BCT, CTRC, WIT, FOKUS
Deliverable reviewer:	RBB
Estimated person Months:	11
Start date of project:	1st June 2008
Project duration	30 months
Revision:	Version 1.5
Dissemination Level:	PU - Public

This page intentionally blank

1 Table of Contents

2 List of Figures

3 Document History

The aim of this deliverable is to provide the necessary adaptations within the Peer-To-Peer Client Software chosen by VITAL++.

Revision Month	Filename version	Summary of Changes
M7	V0.1	Initial Report - ToC
M8	V0.2	ToC with partners assignments
M8	V0.3	Fokus contributions “requirements and architecture”
M8	V0.4	Updated ToC according to Fokus contributions
M8	V0.5	TID contributions
M8	V0.6	“Introduction” by TID
M8	V0.7	Monster contributions by Fokus. BCT “IMS related” texts
M8	V0.8	Added section 7.2 Data Model section to document.
M9	V0.9	Added an updated version of paragraph 8.2.1
M9	V1.0	Added description 7.3. contributions needed by TID.
M9	V1.1	Added figure 7.1 (BCT), revision from BCT and Fokus clients, Content related functions (BCT).
M9	V1.2	UoP contributions (sections 9 and 10)
M9	V1.3	CTRC contributions (section 7.3)
M9	V1.4	Updated version with all contributions inside (TID)
	V1.4.1	First revision up until 8.2.1 (included), (RBB)
M9	V1.5	Updated contributions from TID according to revision + added section 4 from RBB

4 Executive Summary

Following the VITAL++ Use Case Scenarios and the respective requirements the clients used in the project will have to bring together features and functionalities from the IMS and the P2P worlds.

The main IMS-related functions required by the envisaged use cases are authentication (AAA), IMS session management, session negotiation and setup and management of network limitations and Quality of Service by the provision of bandwidth from centralized servers. IMS will be also responsible to manage the digital rights of each object and billing functionalities.

The P2P Overlays will have to be able to cater for different requirements (bandwidth, in particular) by using the network, storage and CPU resources on the user terminal side. To this end, peers will infuse scalability to the system by the utilization of their bandwidth and QoS by the development of a content diffusion overlay according to the traffic conditions in the underlying network and their upload bandwidth capabilities. Additionally block schedulers by using this overlay as a service they will be able to perform real time video and/or audio distribution fast and with high resource utilization. Finally we will examine the scenario of distributed content management in order to offload the resources of IMS application servers and create a highly scalable system in terms of the content volume that is able to manage.

The overlay construction will require features of both, a centralised and decentralised approach. A plug-in architecture for overlays enabling new and innovative overlay implementations to be added to Vital++ nodes seems a crucial requirement with regard to P2P Overlay modularity.

The P2P-IMS client will need to inherit basic P2P functionalities for content search, content retrieval/distribution, content mixing and IMS functionalities such as telephony, media broadcasting and full SIP support. Essential features for P2P content delivery have to cater for all three basic options: file sharing, VOD and live streaming and should be compatible with the Torrent file format.

Both, VITAL++ clients, the one by FOKUS as well as the one by BCT, were originally constructed as IMS clients so that the project's IMS requirements are already met. With regard to P2P, especially transport, playback and play out functionalities will have to be implemented. As both are relying on the Java Media Framework (JMF), high measures of synergy in the development processes should be possible.

5 Introduction

5.1 Scope

The *Detailed client adaptations specification,* developed in this deliverable, aims to define the implied changes that the pre-selected P2P clients need, in order to obtain all the functionality requirements that this project covers.

Taking into account the results of the peer-to-peer client evaluation and the trial network specification, the necessary adaptations within the peer-to-peer client software is being specified within this document for the two different clients: Monster client and BCT client.

The specification of the changes is undertaken in close correlation to the client structure and nurturing existing open interface specifications, preferably based on the IMS protocol language SIP. A close description of the underlying data models and software structures is given in this document as well.

According to the content of the deliverable, an overview of the document is detailed below:

- Overall description of the requirements that must be implemented in the Vital++ client, providing the functionality that the project needs to cover the envisaged scenarios.
- Explanation of the architecture that a generic Vital++ client must perform, without being based in a pre-selected P2P client.
- Individual adaptation specifications: a list of the necessary adaptations to be applied to both P2P clients (Monster client and BCT client). These changes are divided into three different aspects: IMS, P2P and content.
- A possible implementation plan for each client, ordered according to a timeline.

5.2 Deliverable Structure

This deliverable is structured into four parts, not corresponding with the numbers of the sections:

- Part 1 (#1-5) is the introduction to the document, in which the scope and the structure of the report are presented.
- Part 2 (#6-8) presents an overall description of the needs of the future Vital++ client and the architecture to implement these requirements.
- Part 3 (#9) elaborates on the individual implementation plans of the two clients and identifies the changes that must be executed during the P2P client modification.
- Part 4 (#10) offers the conclusions for the adaptation process and the deliverable.

6 Functionality Requirements

In order to enable the envisaged features (VoD, live streaming and file sharing), a hybrid IMS/P2P client would have to be able to perform the following tasks.

6.1 IMS related Functions

The IMS capabilities of the hybrid IMS/P2P client focus on the support of control plane operations. These operations are expected to enable the client to operate in and take advantage of the IMS infrastructure so that it can benefit from IMS communication features.

6.1.1 Authentication

Contrary to the P2P concept, IMS operates in a fully controllable and centralised context. This context caters for the support of a multitude of Authentication, Authorization, Accounting and Billing concepts so that users can be granted access to copyrighted material according to relevant billing concepts and schemes.

For this purpose the client should be able to authenticate with the IMS core. This implies support for authentication algorithms such as MD5, AKAv1-MD5, or AKAv2-MD5. Additionally, Quality-Of-Protection ("*qop*") should include integrity protection ("*auth-int*") as well.

Depending on the execution environment of the client and its relevant capabilities, charging for services can be more easily integrated through IMS procedures with already established mobile phone-related charging procedures and fee collection mechanisms. Such an aspect is expected to achieve better user penetration since no new contracts and procedures are required.

6.1.2 IMS Session Management

The client should be able to identify all SIP requests that are routed to it and translate properly any kind of SIP responses relating to its own requests. Incoming requests should be treated according to their meaning for the creation of new SIP dialogs or the modifications of existing ones. Similarly, user wishes reflected on actions on the UI should be captured and translated into relevant SIP signalling affecting any existing SIP dialogs or creating new ones.

The client being a SIP User Agent should support standard SIP methods such as SIP Instant Messages, Voice and Video calls and also handling of Presence Notifications corresponding to Presence subscriptions initiated by the client.

Messaging can also be enriched with metadata enhancements so that the client can provide added-value application content.

Presence capabilities can be also exploited to support especially SIP based content discovery.

6.1.3 Session Negotiation and Setup

Once a client is registered with the IMS it will be able to establish IMS/SIP sessions in order to initiate certain P2P functions. The session negotiation mechanisms (SDP) that are already used for the establishment of standard SIP sessions can be enriched and utilised so that apart from access control, certain aspects relating to access network conditions and client environment can be catered for prior to initiating P2P interactions. Such negotiation can better communicate any restrictions arising from the nature of the client's environment so that optimal management of network resources can take place. Additionally, relevant information can be injected into the underlying P2P operation (scheduling, etc) so that clients can receive the best of the P2P network avoiding any resources to be reserved for them in case these cannot be actually taken advantage of. Similarly, session management can better identify the P2P capabilities of the client so that its contribution to the P2P operation can be safely evaluated.

Moreover, bandwidth requirements can be communicated so that specific network resource management can be put in effect. Furthermore, any modification in these requirements should be imminently renegotiated and alterations should quickly be applied for the proper continuation of the service.

6.1.4 Management of Network Limitations and Restrictions

SIP techniques with respect to Firewall and NAT traversal should be supported by the client so that its operation remains unimpeded even in network configurations that involve private addressing and connection screening. The client should consider the 3GPP IPSec secured environment without NAT and the ETSI TISPAN NAT'ed environment using UDP encapsulation over IPSec. Media proxying should remain an option in link setup when there is no other alternative in data session setup.

6.2 P2P related Functions

An overlay network is generally considered to be an application-level network, which operates in an autonomous and decentralised way. The Vital++ concept envisages a hybrid IMS-P2P client that combines the trust, security and accounting mechanism of IMS with the scalability and fault-tolerance associated with P2P applications. Key to achieving this is the creation of an architecture permitting the construction and manipulation of P2P overlay networks that are suitable for a range of Vital++ platform features including

- Distributed information database and queries;
- Real time delivery of audio and video;
- Multimedia content broadcast;
- File sharing.

The development of a distributed database will focus on the creation of a hybrid architecture where all participating peers will enter a DHT in order to have scalable queries. We will use a centralized architecture that will manage the information in the DHT in order to increase reliability. Additionally, we suggest the creation of multiple overlay and scheduler architectures as the demands placed on it by file sharing, involving bursts of lookups and packet transfers, can be very different from isochronous stream delivery.

However, the set of functionalities that P2P overlays will address has to be defined analytically:

- Registration and de-registration of nodes within the network known in P2P terminology and joining/leaving
- Addition/removal of content to/from the network
- Addition/removal of metadata to/from the network
- Location of content within the network
- Routing of messages through the network
- Exchange of neighbours in the overlay for the adaptation of it to the network.
- Retrieval of content through block exchanges.

Answering basic requirements, other features have been added for non-functional reasons such as security, scalability, high resource utilization, stability in terms of dynamic network conditions, stability in terms of user behaviour and fault tolerance:

- Peer node and content identification mechanisms
- Encryption of overlay messages and content
- Protection and optimisation of network performance to handle particular operational or application requirements. (e.g. high user churn, network resources)
- Efficient broadcast of messages throughout the network (e.g. the availability of a new “super” peer)

An abstract architecture of the interdependent layers within a P2P Overlay has been identified by Lua [et al.] in 2005.[1]

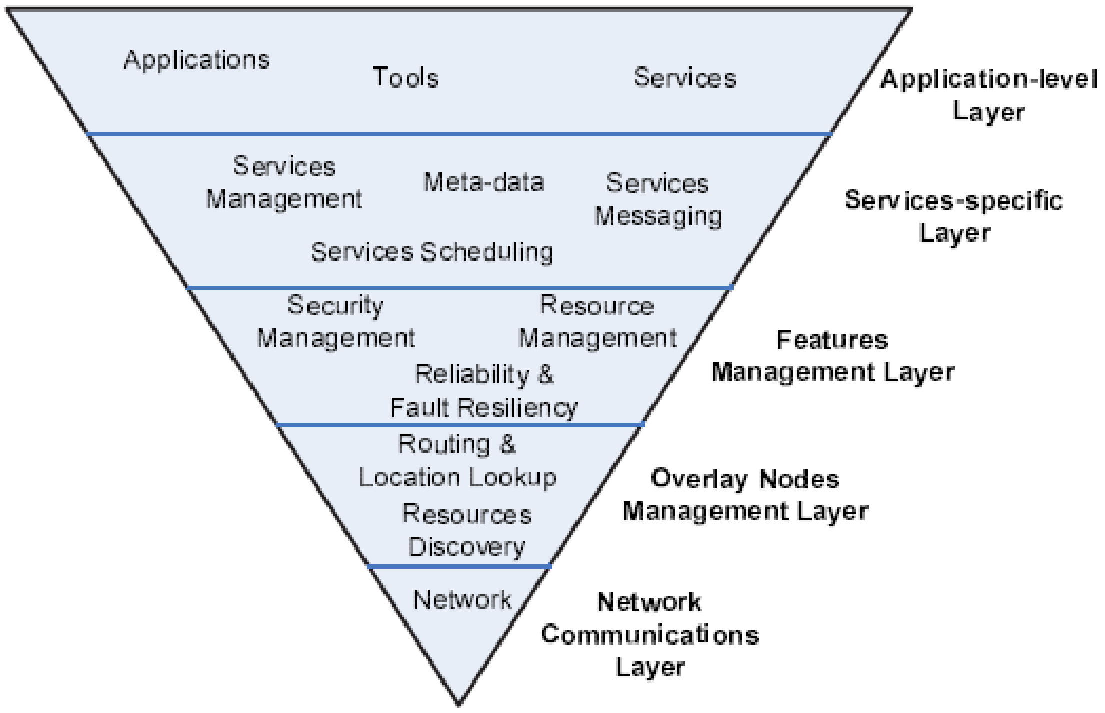

Figure 1: Interdependent Layers within a P2P Overlay

1 Eng Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, “A survey and comparison of peer-to-peer overlay network schemes,” Communications Surveys & Tutorials, IEEE, vol. 7, 2005, pp. 72-93.

The Network Communications layer describes the network characteristics of desktop machines connected over the Internet or small wireless or sensor-based devices that are connected in an ad-hoc manner. The dynamic nature of peers poses challenges in the communication paradigm.

The Overlay Nodes Management layer covers the management of peers which includes discovery of peers and routing algorithms for optimization.

The Features Management layer deals with the security, reliability, fault resiliency, and aggregated resource availability aspects of maintaining the robustness of P2P systems.

The Services-specific layer supports the underlying P2P infrastructure and the application-specific components through scheduling of parallel and computation-intensive tasks, content and file management. Meta-data describes the content stored across the P2P peers and the location information.

The Application-level layer is concerned with tools, applications, and services that are implemented with specific functionalities on top of the underlying P2P overlay infrastructure. Thus, there are two classes of P2P overlay networks towards the development of distributed and scalable query mechanisms : *Structured* and *Unstructured*.

In *Unstructured* P2P, the overlay networks organize peers in a random graph in a flat or hierarchical manner (e.g., Super-Peers layer) and use flooding or random walks or expanding-ring Time-To-Live (TTL) search, etc. on the graph to query content stored by overlay peers. Each peer visited will evaluate the query locally against its own content, and will support complex queries. This is inefficient because queries for content that are not widely replicated must be sent to a large fraction of peers, and there is no coupling between topology and data items' location. Examples of *Unstructured* P2P overlay networks include Freenet, Gnutella, FastTrack/KaZaA, BitTorrent, and Overnet/eDonkey2000.

Structured means that the P2P overlay network topology is tightly controlled and content is placed not at random peers but at specified locations that will make subsequent queries more efficient. Such structured P2P systems use the Distributed Hash Table (DHT) as a substrate, in which data object (or value) location information is placed deterministically, at the peers with identifiers corresponding to the data object's unique key. Implementations of structured P2P overlays include: Content Addressable Network (CAN), Tapestry, Chord, Pastry, Kademlia, and Viceroy.

DHT-based systems are based on the consistent assignment of uniform random NodeIDs to the set of peers into a large space of identifiers. Data objects are assigned unique identifiers called keys, chosen from the same identifier space. Keys are mapped by the overlay network protocol to a unique live peer in the overlay network. The P2P overlay networks support the scalable storage and retrieval of {*key,value*} pairs on the overlay network, Given a key, a store operation (*put(key,value)*) lookup retrieval operation (*value=get(key)*) can be invoked to store and retrieve the data object corresponding to the key, which involves routing requests to the corresponding peer.

6.2.1 P2P Distributed Queries and DHT Overlay

Searching on an unstructured P2P overlay is most commonly done using techniques such as Flooding, Random Walks or expanding-ring Time-To-Live (TTL). The most common query method is flooding where a peer when performing a search queries its neighbours within a certain radius for the object required. While Flooding and Random Walks and their variants are effective in locating popular material in unstructured networks they are not overly efficient. In fact, studies have shown[2] that in the Gnutella Network almost 18% of searches return no responses even though results are available.

Several approaches have been tried to improve search recall. One approach was to increase the Time-To-Live (TTL) for search algorithms. However, a higher TTL value caused a significant increase in the consumption of overall bandwidth and provided diminishing returns, particularly in the case of search techniques such as flooding. An alternative technique employed was to replicate objects to increase the likelihood of a successful search. This method requires that entire objects are replicated, thereby significantly increasing the bandwidth and storage costs and as a result limiting their applicability. A further refinement to the idea of replication is Index Replication. In one-hop index replication, each node stores its meta-data, data pertaining to the nodes resources, on all of its one hop neighbours. This has been found to be an extremely valuable technique in scaling unstructured networks[3].

	Structured	**Unstructured**
Query Type	Key Lookup	Arbitrary
Query Cost	*O(log(N))*	*O(N)*
Hit Guarantee	High	Low
Connectivity Graph	Structured	Random

Table 1: Structured vs. Unstructured P2P Networks

There is also a hybrid approach employed by some unstructured networks to improve the search for rare objects. The idea is to identify rare objects and place them in a structured network to improve the chances of lookup and keep

2 Loo, B. T. et al, Enhancing p2p file sharing with an internet-scale query processor

3 Chawathe, Y. et al, Making gnutella-like p2p systems scalable

the unstructured network for the more popular objects. This method is effective but does have the associated costs of deploying and maintaining a new overlay network.

In structured P2P networks the network topology is controlled and content is placed in specified locations so that subsequent queries are more efficient. A distributed hash table (DHT) is used to deterministically place values, which equate to location information, at peers whose identifiers correspond to the data object's unique key. The use of a DHT allows a group of distributed hosts to collectively manage a mapping of (key, value) pairs without the need for a fixed hierarchy and with little or no direct human interaction. The characteristics of the DHT are decentralization, scalability and fault tolerance. Any participating node should be able to efficiently retrieve the value associated with a key.

Chord, for example, assumes a circular identifier space of size N. A Chord node with identifier *u* has a pointer to the first node following it clockwise on the *identifier space* as well as the first node preceding it forming a doubly linked list. The node also keeps $\log_2(N)$ pointers called fingers to other nodes.

Tapestry on the other-hand uses a Plaxton mesh as its basis. Tapestry uses multiple roots for each data object thereby avoiding a single point of failure. In a Plaxton mesh, peers can be servers, routers and clients. This data structure allows messages to locate objects and route to them across an arbitrarily-sized network, while using a small constant-sized routing map at each hop and therefore provides a scalable mechanism for accessing nearby copies of objects. In an n-node Plaxton mesh, both objects and nodes have randomly chosen labels of size log n bits independent of their location and semantic properties.

When comparing the Chord and Tapestry protocols, the major advantage that Tapestry has over Chord is that the network topology is known through the use of network locality. Therefore, queries never travel more than the network distance required to reach them. However, scalability can be an issue with Tapestry as it does not handle churn as well as Chord as it is more complicated. Tapestry routing tables need to updated, which is an expensive operation whenever a node joins or leaves the network.

If the appropriate security policies are not in place problems can occur in within DHTs. A malicious node could potentially examine, drop or manipulate messages as they pass through the node. The ability of untrustworthy nodes to inject unsolicited messages or manipulate legitimate messages means that the routing tables of other nodes can be corrupted. To avoid such problems it is necessary that some security mechanism is put in place to exclude or manage these untrustworthy nodes.

To protect the content delivered over the P2P network from unauthorized access a Digital Rights Management (DRM) system is required. OMA DRM defines a means by which to deliver copyrighted content and ensure that the content can only be used with an appropriate license. A content issuer shall deliver DRM content and a rights issuer generates a Rights Object. This Rights Object governs how the DRM content may be used. Specifically, it is an XML document which outlines the permissions and constraints associated with a piece of DRM content.

The DRM content cannot be used without an associated Rights Object. OMA DRM makes a logical separation of the content and the Rights Object. By doing this a user can download the content and the rights object possibly within the same transaction and begin to use the restricted content. Later if the Rights Object expires the user can request a new Rights Object without having to download the content again. The enforcement of the rights of the restricted content has to occur at the point of consumption using what is known as a DRM Agent. A Rights Object is cryptographically bound to a specific DRM Agent, so only that DRM Agent can access it. Since DRM content can only be accessed by a valid Rights Object this allows content to be super-distributed, i.e. content can be freely passed amongst users as content is not accessible without a valid Rights Object. It is also possible to bind the Rights Object to a group of DRM Agents known as a domain. This would allow a user to download content which would be accessible on their phone, netbook, etc.

The OMA DRM specifications define the format and the protection mechanism for DRM Content, the format and the protection mechanism for the Rights Object, and the security model for management of encryption keys. The OMA DRM specifications also define how DRM Content and Rights Objects may be transported to Devices using a range of transport mechanisms, including pull (HTTP Pull, OMA Download), push (WAP Push, MMS) and streaming.

6.2.1.1 Overlay Construction Requirements for Vital++ P2P distributed queries

- Popular content should be retrievable quickly and rare content should be available within predicted time (by the use of a structured DHT based overlay with a replication technique for popular content)
- Vital++ management information may be stored in the overlay. Management information access times should be predictable (i.e. use DHT).
- Content should be replicated for fault-tolerance (consider TAPESTRY-like architecture and the use of the VITAL++ centralized database)
- Compatibility with centralised content authorisation nodes as specified in IMS-IPTV and DRM systems

6.2.2 P2P Content Diffusion Overlay

Peer-to-Peer (P2P) live streaming is a heavily researched topic that faces a series of challenges originating both from the diversity of behaviours and capabilities of the participating peers combined with the strict delivery requirements of streaming applications. In general terms, an application server generates a video/audio stream at the appropriate service rate divided into blocks, which are delivered to the overlay. Peers in the overlay then exchange the blocks with their neighbours until all peers eventually receive all blocks. In this context a requirement to be met is that the average uploading bandwidth capability of the whole system must always be kept above the service rate in order to successfully deliver the stream while each one of the generated blocks must be timely delivered to every peer.

In addition, peers involved in these systems have heterogeneous and dynamic uploading bandwidth capabilities. When combined with the characteristics of the topology of the underlying network and the dynamic traffic conditions, e.g. latency, they create a volatile and complex environment for P2P live streaming delivery. These factors heavily influence the efficiency of a P2P system measured by a number of key performance indicators.

The first factor is *the uploading bandwidth utilization* that corresponds to the ability of the system to exploit as much as possible the sum of the uploading bandwidth of the participating peers that is noted as aggregate uploading bandwidth of the system. When a peer is not able to fully utilize its uploading bandwidth capabilities this creates a bandwidth bottleneck in the system.

Equally important is the *setup time* defined as the time interval between the generation of a block from the origin server and its delivery to the “last” peer in the system.

Furthermore, a P2P live streaming system has to remain *stable* in such an environment especially in the presence of frequent peer arrivals and departures. This results in varying numbers of peers which influences the stability of the system with respect to the uninterrupted delivery of the streaming service.

Finally, *fairness* among nodes indicates the ability of the system to continuously distribute uniformly the aggregate uploading bandwidth to the participating peers. This ability ensures that even in conditions where aggregate uploading bandwidth is insufficient for the delivery of the whole video stream, every peer will acquire a percentage of blocks above a critical threshold for an “affordable” video playback. .

A pioneer work of the first kind is SplitStream. SplitStream proposes as overlay a formation of trees whereby each node is leaf in every tree but one. These trees are maintained by mechanisms of a DHT called Pastry. Blocks are uniformly assigned to a number of stripes equal to the number of trees. Each tree distributes one stripe by sequentially propagating each one of its blocks from the parent to its children. In SplitStream the overlay is organized according to IDs assigned to each node and not according to network or node features. In this way it is not possible to achieve high uploading bandwidth utilization because the diffusion of each stripe is done according to the IDs of each node in the overlay. In contrast, this leads to fast block propagation through each tree with low control overhead. Nevertheless, the system suffers from a lack of stability and a resultant degradation of stream quality under dynamic conditions like frequent peer departures.

The limitations of tree-based overlays were overcome with the introduction of random mesh overlays where blocks are again assigned to stripes. However, there is no predefined graph topology and each stripe could be exchanged between any two nodes according to the sender's uploading bandwidth capabilities. In this system a mesh overlay is constructed wherein nodes have equal numbers of neighbours. Stripe reassignment, due to node arrivals, departures or bandwidth changes, takes place with the help of a decision function. The authors demonstrate in their original work that under static conditions, limited heterogeneity and sufficient uploading bandwidth capabilities among peers, their system achieves a relatively high degree of bandwidth utilization and low setup time. On the other hand, the randomness of the formed mesh leads to a bandwidth bottleneck in highly heterogeneous environments. Finally, the cases where aggregate uploading bandwidth is insufficient or the total number of participating peers fluctuates are not evaluated.

We will focus on an overlay with a graph topology where peers with highly heterogeneous upload capabilities are able to fully utilize their upload bandwidth. Additionally, according to measurements from the underlying network (ex. Network latency), peers will be able to dynamically and cooperatively adapt the flows for the distribution of each object according to the traffic conditions in the underlying network. A functionality that will be used for the reorganization of the overlay (the list of neighbours that each peer has each time instant) will ensure a stable rate in block reception for each peer during dynamic peer arrivals and departures.

Fair aggregate upload bandwidth distribution is very important for complete video stream distribution in case of sufficient available upload bandwidth and even much more important in case of insufficient available upload bandwidth where every node has to receive a percentage of blocks above a threshold in order to playback the video stream even with depredated quality. Towards this goal we again highlight the significance of a well connected and optimized overlay in which every node participates in the distribution of the stream in a degree analogous to its upload capabilities. Furthermore, all the participating peers should be able to obtain the whole stream, or at least the same percentage of the stream with every other node. For this reason, when a node is ready to transmit a new block, it should prefer as its next node for transmission its most deprived neighbour, meaning the neighbour who has the smallest number of blocks.

6.2.3 P2P Scheduling

P2P block scheduling in content distribution is a technical problem that has been more or less addressed. In Chapter 7.4.2.3 we describe in more detail the main principles how in p2p content distribution systems are able to address this technical challenge.

Content distribution is not a real time application and so it invokes flexible decisions on the distribution of blocks in terms of the variance in the rate at which each peer takes blocks. Additionally, in content distribution we don't have to deliver the blocks with a specific order but according to the needs of the scheduler in order to avoid a content bottleneck. Eventually, a node with vast upload bandwidth can upload blocks at a very high rate to any peer.

In contrast to that, in live streaming each time instant only a small number of blocks are available to be exchanged in the overlay. This is due to the real time production and distribution of video and/or audio. This constraints the scheduler and requires a sophisticated neighbour selection and block selection in order to have high upload bandwidth utilization by avoiding bandwidth bottleneck and content bottleneck. Bandwidth bottleneck is a condition where a node has many blocks to upload to its neighbours but it does not have the required bandwidth available. Content bottleneck is the situation where a node has available upload bandwidth but has no blocks to exchange with its neighbours.

Fair aggregate upload bandwidth distribution is very important for complete video stream distribution in case of sufficient available upload bandwidth and also much more important in case of insufficient available upload bandwidth where every node has to receive a percentage of blocks above a threshold in order to playback the video stream even with depredated quality. Towards this goal we again highlight the significance of a well connected and optimized overlay in which every node participates in the distribution of the stream in a degree analogous to its upload capabilities. Furthermore, all the participating peers should be able to obtain the whole stream, or at least the same percentage of the stream with every other node. For this reason, when a node is ready to transmit a new block, it should prefer as its next node for transmission its most deprived neighbour, meaning the neighbour who has the least blocks compared to the others. This parameter combined with the preference for nodes with high upload capacities dictates the form of the decision function running in each node for selecting the next node for block transmission.

Finally, a successful streaming system should be able to dynamically adapt to the various underlying network changes and to nodes arrivals and departures. For this reason an algorithm is needed that will run in every node and that would be responsible for recognizing those changes and dynamically reconfigure the overlay graph accordingly. Additionally, the real time decisions of the scheduler and the locality aware overlay lead to a system that adapts very fast to dynamic conditions.

6.2.4 Overlay Optimisation & QoS Management

Much research has been carried out into optimising the performance of structured and unstructured P2P overlays to meet particular application requirements, generally non-functional. Examples include the TAPESTRY modifications of the basic PASTRY architecture and the super-peer modifications to Gnutella to reduce query traffic exponentiation. Some of this work is passive and involves improvement of the underlying content location algorithm for performance, scalability or fault-tolerance purposes. Additionally, active measures to monitor and react to network conditions have also been proposed and developed. An example is the Vivaldi[4] virtual positioning system developed for the Azureus BitTorrent client. The intention of overlay positioning systems is to determine the relative logical positions between clients in order to optimise DHT searches.

The intention of the Vital++ project is to focus on optimising a P2P approach to live streaming. To achieve this, we will also consider the work of the P2P-Next project[5] which has developed the SwarmPlayer client. We will optimize the overlay performance which involves development of an efficient and reliable new overlay mechanism for distribution of live streaming content.

As the Vital++ client is an IMS client, it is also possible to reserve bandwidth using IMS mechanisms. The IMS Access Network is aware of the QoS requirements of streams setup using IMS signalling by analysing the SIP signalling used to setup and manipulate these streams. This signalling supports “hints” regarding application expectations in the form of IMS Communication Service Identifiers (CSI’s). A Vital++ client that finds the P2P overlay is unable to support its QoS requirements may use the IMS to explicitly reserve QoS.

6.2.4.1 Vital++ P2P Overlay Optimisation & QoS Requirements

- Support for instrumentation of overlay communications to enable debugging or analysis of network performance. (Probably use Overlay Weaver);
- Support for virtual position of overlay nodes;
- Overlay awareness and optimisation using IMS QoS reservation mechanisms.
- P2P Streaming. Discover peers and interchange media information.
- P2P FileSharing. Discover peers and interchange media information.
- P2P Secure DHT for VITAL++ Management information (e.g. Content index, Content overlay index, PKI information)
- P2P Overlay Management (Process/Perform Join/Leave)

[4] VivaldiView, Azuereus Wiki, http://azureuswiki.com/index.php/Vivaldi_View

[5] P2P Next, Shaping the Next Generation of Internet TV, http://www.p2p-next.org/

- P2P Overlay construction for all envisaged Overlay types (Live streaming, VoD-streaming, file sharing, secure DHT)
- QoS Management. As it is not yet defined how content security is going to be achieved yet, two approaches are presented.
    - Achieve required bandwidth by choosing appropriate peers for media delivery (P2P approach)
    - Reserve required bandwidth by setting up an NGN session with other peers (NGN approach)

The QoS Management also has to measure the actual used bandwidth and existing latency in order to decide whether to change paths or use NGN QoS.

6.2.5 Modularity of Design & Implementation

The Vital++ P2P Overlay should fulfil some requirements with regard to modularity:

- Support for common data model shared between overlay instantiations with common identifiers (or mappings) for overlay subscribers, node identifiers and content identifiers.
- Modular interfaces for Common Overlay functionality similar to, or using, Overlay Weaver.[6]
- A plug-in architecture for overlays enabling new and innovative overlays implementations to be added to Vital++ nodes. These new overlays may be implemented to provide support for specific Vital++ scenarios.

Dabek et al. proposed layered abstractions of structured overlays.[7] They separated common services such as *DHT*, *multicast*, and *anycast* from an underlying routing layer. The model suggested by Dabek enables separated design and implementation of a routing layer named Key-Based Routing (KBR) from the common higher-level services. Here, the routing layer is monolithic even with layered abstractions. However, the routing algorithm is not the only element in the routing layer. Shudo et al.[8] identified common process elements that could be abstracted for a variety of P2P overlays. They have decoupled these process elements into a commonly used P2P Overlay development toolkit known as Overlay Weaver.

Overlay Weaver is implemented in Java and comes with a selection of structured DHT-based overlay implementations to demonstrate the system's modularity and flexibility.

6 Overlay Weaver: an overlay construction toolkit, Available from: http://overlayweaver.sourceforge.net .

7 Dabek F., Zhao B., Druschel P., [et al.], Towards a common API for structured peer-to-peer overlays, in: Proc. IPTPS'03, 2003.

8 Shudo K. [et al.], Overlay Weaver – An Overlay Construction Toolkit, Communication Communications 31 (2008) pp 402-412.

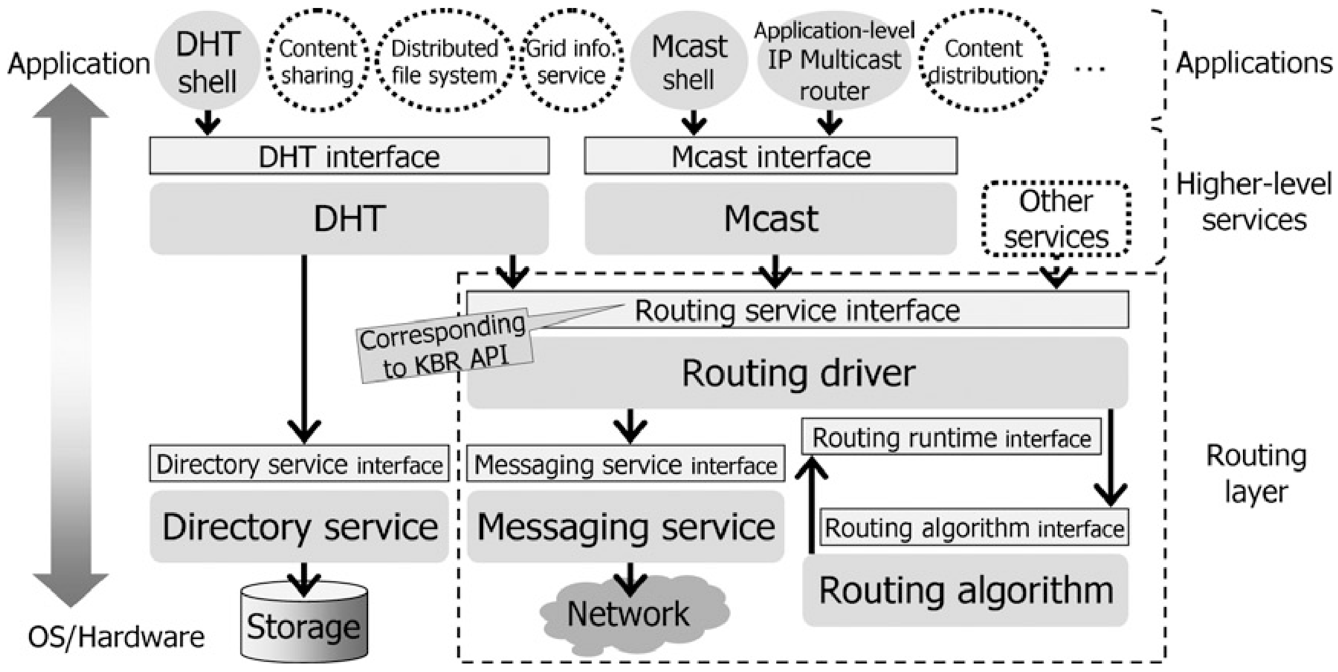

Figure 2: Overlay Weaver architecture

The design permits a P2P overlay developer to mix different overlays using a common communications process layer. This is a very powerful approach as the designer may incorporate different overlays into their P2P network design depending on application requirements. The common application programming interface means that abstract objects like *RoutingContext*, *ID & route* hide much of the complexity of the underlying overlay from the service layer developer. This is a requirement we have already identified for Vital++.

6.3 Content related Functions

Content retrieval is probably the most crucial feature for the VITAL++ client. All other required content related functionalities are related with distributing and managing the respective content.

In the case of multimedia content in the VoD and Live-TV cases, the problem of real time and quality emerges as an extra requirement that the file sharing case does not contain.

The following functionalities are required to enable the envisaged use case scenarios:

6.3.1 Audio/Video Capturing and Play-out

The audio and video contents that need to be distributed to the users may exist as files stored on a multimedia server or as an audio/video signal created by a live content producer. If the content is already stored, there is no relevant inconvenience, but in the Live-TV case in which the content is fed in real time, the platform must capture it and convert it to the formats that different users will be able to consume on their various devices.

After capturing and transcoding tasks, the content must be introduced to the media delivery platform in order to distribute it to the audience. The P2P overlay network will be in charge of this circulation of the content, assuring that all the pieces arrive at the client so that it will be able to rebuild the multimedia flow and consume it.

On a P2P distribution network, the segments of the content might arrive in a wrong order or could be lost, so the Vital++ client must be capable of receiving an appropriately sized buffer of data to present it to the user.

6.3.2 Graphical User Interface (GUI)

Like any kind of service, the user needs an infrastructure to interact with the system that offers it. This graphical user interface (GUI) helps the consumers to order what they want or what they expect from the service.

This interface should be able to run on every category of equipment that will be used in the envisaged scenarios and should be easy to use. Some functionalities that the user will be able to perform through this graphical interface will be

- Searching interface where a user will be able to locate files and/or objects that will be categorized.
- Query system where users will be able to locate objects that match with a keyword that they have entered
- Volume control
- Search functionalities during video and audio streaming.
- Selection of video quality.

6.3.3 Content Security Measures

In a P2P distribution model in which anyone could take data from other users, security is an important factor. To address the content security issue, VITAL++ might use a rights table, which allows/denies access to a specific content overlay.
In this solution, each peer of the content overlay needs to contact this table to know if the other peers that request its pieces of content have the right to consume it or not. This system needs a new module of security that interacts with this table very frequently.

6.3.4 Content Management

This content related function will have to manage all known content in terms of locations, access methods and media type. A client must know which content is available in the P2P overlay network and approximately on how many users and which its characteristics are.

In the P2P philosophy, the content in circulation is divided into blocks and is distributed all around the network. These blocks are present in all the peers so it must identify whether a block should be requested from a centralized server to acquire it or it is present in the overlay.

6.3.5 Content Discovery and Advertisement

The client must be able to browse a content index for resources, which could then be made available to the content management facility to let the user consume it. The content index should support both browsing and searching for content based on keywords. By "content index" we mean meta-data about the content such as a textual description, encoding scheme, popularity and other information pertinent to Vital++ applications. Additionally, the index would store an identifier, which may be used to retrieve the content from the overlay. Several P2P clients have suggested using or adopted the magnet URN scheme.

There are several ways a content index could be created. One is to store content descriptors in a database, be it object, relational or XML. Another possibility is to use an Open Source content indexing engine such as Apache Lucene which searches for text strings in much the same manner as Google search does. As Lucene can be adapted to index particular formats (e.g. an XML descriptor for multimedia content), both database and search engine approaches can be used satisfactorily with structured content.

To reduce dependence on a bespoke indexing or content searching solution, content descriptions may be made available using an accepted syndication format such as RSS 2.0.

There should be a way to provide a list of available contents to the user who is interested in consuming multimedia resources. This list either could be given by the application server of the service or a DHT overlay could be built to give the users the option to find which contents are being distributed over the P2P network. In this second possibility, a module with content discovery capabilities must be developed.

7 VITAL++ Generic Client Architecture

7.1 Module Architecture Overview

The architecture of a generic P2P and IMS client could look like depicted in the following figure. The architecture of the client identifies three layers:

User Interaction Layer:	It is the part of the client that interacts with the user. It provides all media playback and capturing capabilities as well as means for discovering and publishing content.
Protocols Layer:	It contains in self contained modules SIP and P2P engines as well as a local storage engine. The capabilities in this layer enable the client to cope with the specific requirements for joining the various networks as well as to manage locally stored content.
Network Layer:	It is the raw connection part not necessarily addressed by client developments. It can be provided by the underlying resources of the hosting Operating System.

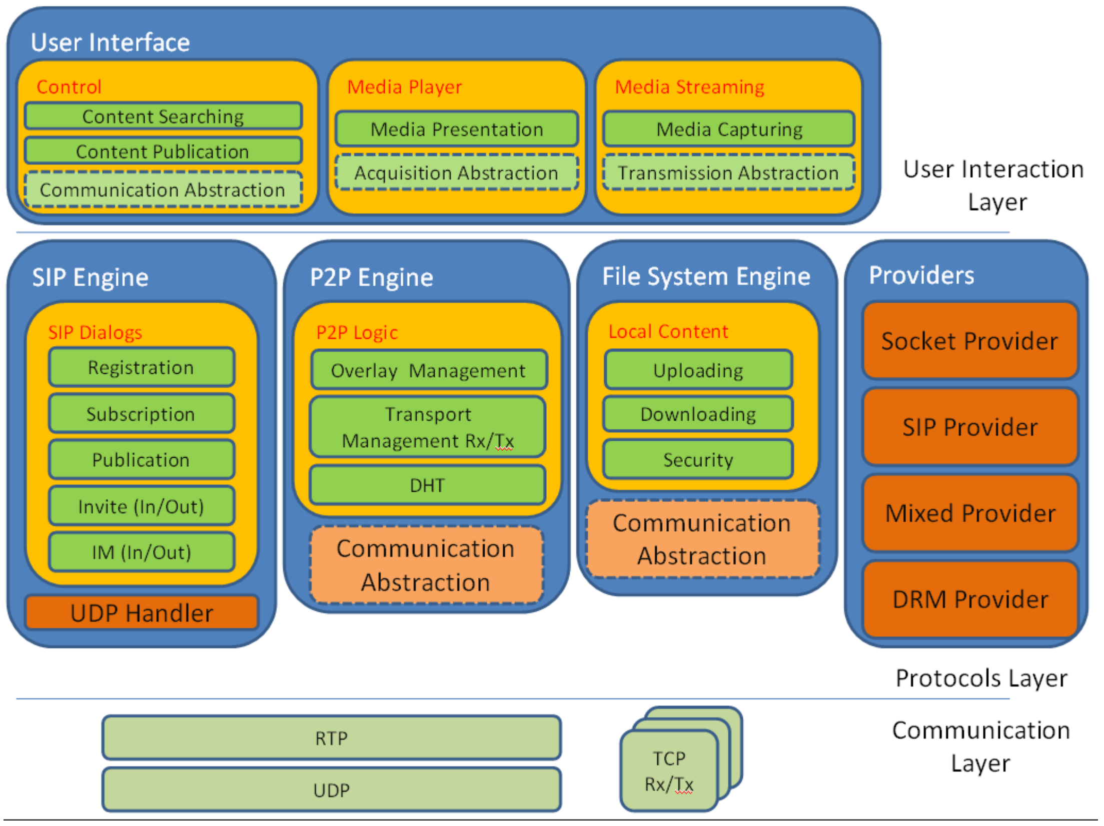

Figure 3: Possible Generic Client Architecture

7.1.1 User Interaction Layer

The User Interaction Layer holds all functional blocks which communicate directly with the user. These are the

- Graphical User Interface for interacting in terms of functional behaviour of the client.
- Browser for presentation of contents.
- Module for audio capturing and play-out, e.g. to realize voice calls or music playback.
- Video capturing, which basically reads either from a camera (interactive case) or from a file (preinstalled media case).
- Video play-out for video calls or simple video stream playback.

7.1.2 Protocols Layer

The Protocols Layer holds the protocol engines, which are used to communicate with other resources such as peers, the IMS core and its components. In this case the protocol engines are

- A protocol stack for content distribution based on P2P which is used for realizing the DHT and can be used to set media overlays. This contains the P2P logic with an abstract view of the communication procedures so that these can be provided either by standard TCP communications or by SIP/IMS procedures.
- The IMS and SIP layers can be used without P2P capabilities.
- A basic SIP stack for the basic SIP operations.
- An IMS-SIP extension, which reflects the 3GPP changes and extensions to the basic SIP.
- A File System Engine that copes with the management of the locally stored content. Again this engine is agnostic of the underlying communication module and can interact with the P2P modules to cater for content sharing.
- A list of providers that bind together the previous modules to instantiate the Vital++ logic for the provision of the respective scenarios.

The Protocols Layer holds the intelligence between the two other layers. Its functionalities can be categorised according to the following scheme:

- IMS applications
  - Session Management

    This includes creation, maintenance and termination of IMS sessions, including bandwidth reservation during session setup
  - Messaging, Presence, Telephony

    These are basic IMS applications. For VITAL++ they are optional.

- P2P applications
  - Streaming

    Realize media streaming between peers, including media splitting /recombining and working with multiple sources/destinations.
  - File sharing

    Same function as streaming but for static content.
  - Secure DHT

    Functions for realizing a secure DHT, includes signature checking etc.
  - Overlay management and construction

    Algorithms for topology aware overlay creation for all the overlay types.
  - General functions

    Anything that belongs to the P2P applications and does not fit anywhere else or is shared by other P2P application blocks.
- Content applications
  - Content security

    The structure of this needs to be defined depending on the content security model to be negotiated.
  - Content management

    Manage all known content in terms of locations, access methods, storage and media type.
  - Content discovery

    Browse a content index for contents, which can then be made available to the content management.

7.2 Underlying Data Models

Trying to identify the commonalities between P2P and IMS centric systems the focus should be on the core architectures and feature sets of both systems. Problems occur because unlike IMS, P2P architectures are not standardised. They can follow a centralised architecture, a brokered architecture or a decentralised architecture. The data models used differ between P2P architectures, however there are core concepts such as subscription, identification, anonymity, encryption etc. which are common between both IMS and P2P. Within IMS there are several places where user profile data is stored: the Home Subscriber Server (HSS), the SIP Application Servers, the XCAP Data Management Servers, Presence Servers, Generic User Profile Servers and Common Profile Stores (LDAP Servers).

In 3GPP documentation, specifically 3GPP TS 23.002 Network Architecture, the HSS is described as follows:

"The HSS is the master database for a given user. It is the entity containing the subscription-related information to support the network entities actually handling calls/sessions. A Home Network may contain one or several HSSs: it depends on the number of mobile subscribers, on the capacity of the equipment and on the organisation of the network. As an example, the HSS provides support to the call control servers in order to complete the routing/roaming procedures by solving authentication, authorisation, naming/addressing resolution, location dependencies, etc."

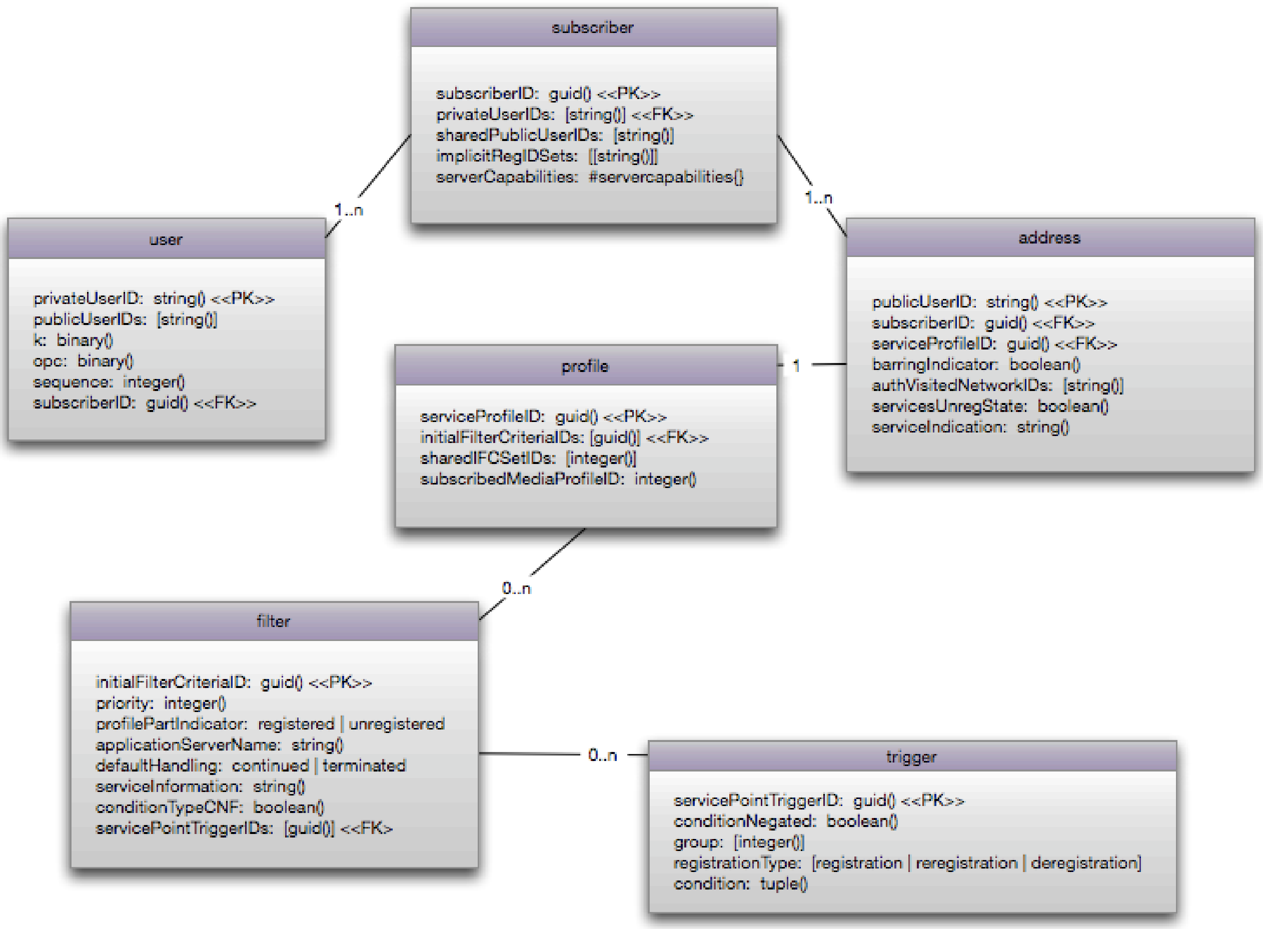
subscriber
subscriberID: guid() <<PK>>
privateUserIDs: [string()] <<FK>>
sharedPublicUserIDs: [string()]
implicitRegIDSets: [[string()]]
serverCapabilities: #servercapabilities()
1..n
1..n
user
privateUserID: string() <<PK>>
publicUserIDs: [string()]
k: binary()
opc: binary()
sequence: integer()
subscriberID: guid() <<FK>>
address
publicUserID: string() <<PK>>
subscriberID: guid() <<FK>>
serviceProfileID: guid() <<FK>>
barringIndicator: boolean()
authVisitedNetworkIDs: [string()]
servicesUnregState: boolean()
serviceIndication: string()
1
profile
serviceProfileID: guid() <<PK>>
initialFilterCriterialDs: [guid()] <<FK>>
sharedIFCSetIDs: [integer()]
subscribedMediaProfileID: integer()
0..n
filter
initialFilterCriterialD: guid() <<PK>>
priority: integer()
profilePartIndicator: registered | unregistered
applicationServerName: string()
defaultHandling: continued | terminated
serviceInformation: string()
conditionTypeCNF: boolean()
servicePointTriggerIDs: [guid()] <<FK>
0..n
trigger
servicePointTriggerID: guid() <<PK>>
conditionNegated: boolean()
group: [integer()]
registrationType: [registration | reregistration | deregistration]
condition: tuple()

Figure 4: Permanent Subscriber Data from 3Gdb Home Subscriber Server (HSS)

Therefore the HSS is a mission critical database which stores all the user centric data which is required by the IMS core network. Therefore, the HSS stores the following types of information:

- subscription
- identification
- numbering
- registration
- authentication
- ciphering
- service profile

However, the storage method employed and the specifics of the HSS schema are determined by the vendor who supplied the HSS. There is also the *sh* interface between the HSS and the SIP application server. This interface is based on Diameter and is used to access user data stored in the HSS or even storing application data in the HSS and is agnostic with regards to this data.

The IMS architecture allows for a SIP Application Server (SIP AS) to store its own user data. This data supports the services supported by the SIP AS. This is the *Ut* Interface and is based on XCAP (XML Configuration Access Protocol) which allows a client to read, write and modify application configuration data, stored in XML format on a server. XCAP is not a new protocol. XCAP maps XML document sub-trees and element attributes to HTTP URIs, so that these components can be directly accessed by HTTP. This means that user service data is located with the services that use it.

Another XCAP based data repository within IMS is the XCAP Data Management Servers (XDMS) which is an architecture defined by the Open Mobile Alliance (OMA). The purpose of the XDMS is to store common data between different services and enablers and also to store specific data for each of those data and services;

Presence can also be considered as a user data store as the presence server stores information about the user which the user can manage on a service to service basis.

The Generic User Profile (GUP) is a 3GPP standard which defines a user centric data repository and provisioning architecture. It does this through the GUP Server (GUPS) which is the key element in the architecture. The GUPS provides metadata which contains the knowledge of the location of the different data components and the different data repositories. In essence, it is a virtual centralised database which enables homogeneous access to the user data stored in the various network databases and application servers. The GUPS also acts as a gatekeeper by authorising or denying access to profile data. The GUP interfaces are based on SOAP/XML and are generic, hence independent of the GUP application data model deployed. A change to the GUP application data model, due to the introduction of new services for example, has no impact on the northbound interface. On its southbound interface (Rp), it connects to the main data repositories, e.g. the SLF, the IM-HSS, the AS, the HLR/AuC, the PDBF, etc., also via the standard SOAP/XML interface.

The Common Profile Store (CPS) is also known as the Next Generation Profile Register (NGPR). The purpose of the CPS is to change from the monolithic database structures of the existing network databases (e.g. the HLR, HSS, etc.) to a multi-tiered architecture. Using CPS will allow for the optimisation and rationalisation of network databases thereby removing the lock-in from vendor specific implementations of the various network databases. Within the context of IMS, a CPS architecture would remove the need for *Sh* interface between the HSS and the HLR by permitting user data sharing through access to the common back end. The CPS could also be integrated with other platforms such as JEE or JAIN SLEE allowing the user data to be used by services deployed on these platforms.

As mentioned in the first paragraph of this section there is no single architecture employed within peer-to-peer systems. For example take the use case of search within P2P. Some architectures use a centralised searching mechanism such as Napster, others like Gnutella use flooding, while other again use a strategy like distributed hash tables. Others again, such as Skype, use a Distributed Hash Table (DHT).

Napster contains a central indexing server which when a user searches for a particular resource the Napster central server returns a list of all nodes on the network which have that resource. The user then selects the node they wish to download from.

Gnutella, however, does not have a central repository. Instead, Gnutella floods the network with the search for the required resource. The user's Gnutella client knows of at least one other node somewhere on the network. The user's machine sends the request to the known nodes. These known nodes then search for the required resource on their local storage. As well as searching for the resource locally, the nodes forward the request onto the nodes known to them. And the process repeats until the Time To Live (TTL) for the request expires. The disadvantage with this system is that there is no guarantee that the file you want is on any of the nodes which have been queried before the TTL expires. Searches can also take quite a while to complete until all responses come in. As the user's machine is also a node on this network their bandwidth and system resources are being consumed in the background to fulfil requests from other nodes.

Issue	Problem	Possible Solution
Identity	Authentication, malicious nodes	Keys, digital signatures, tokens, certificates. Possible to integrate with IMS AAA
Presence	Not always valid for users; indicates online clients	Use presence within IMS
Agency	Proper delegation of authority	Authentication standards better in IMS
Browsing	Exposure of local data, a risk of intrusion	Authentication and filters are required
Architecture	Unintended exposure of local resources	Firewalls, clear access policy
Protocol	A lack of interoperability	Open protocols

Table 2: Security Issues in P2P

When searching in a DHT the user needs to know the exact file name or identifier as these identifiers represent keys which are mapped to node-IDs. Skype is an example of a DHT employing a super node architecture and supports call-establishment and NAT traversal. Skype is a P2P VoIP telephony system which encrypts TCP/UDP payloads, thus analysis of Skype is limited. When logging in with Skype the client first contacts the bootstrap servers trying several different protocols in a certain order: UDP, TCP, HTTP Proxy, HTTPS Proxy. Once this is successful a super-peer cache is downloaded and stored in the <HostCache> node structure in 'shared.xml' which in Linux based systems can be located under ~/.Skype/shared.xml. Once this has been completed the login is performed against a centralised login server using TCP. When the login is successful the client connects to the super-peer and checks if it can be promoted to the level of super-peer. This requires that the client has an open IP Address and available bandwidth. This is achieved using Simple Traversal of UDP through NATS (STUN) protocol between the peer and either a super-peer or one of the bootstrap servers. This is run every time a client starts up a Skype session. Skype peers discover users over the super-peer overlay using a decentralised User Directory which Skype call the Global Index (GI).

"The Global Index (GI) technology is a multi-tiered network where super nodes communicate in such a way that every node in the network has full knowledge of all available users and resources with minimal latency."[9]

7.3 General Interface Definition

The merging of Peer-to-Peer technology with the IMS infrastructure will result in a client that should be scalable and fault tolerant. This VITAL++ client would benefit from the decentralised P2P architecture while it is assisted by IMS features such as strong authentication, encryption, auditing and accounting.

However, as already mentioned in D2.1: *The integration of two technologies with very different concerns and motivations is not without its problems. IMS is realised through a collection of well specified logical nodes with defined interfaces and interactions required to achieve basic and advanced network functionality such as subscription, registration, session control, roaming and access network mediation. These nodes are centralised in a client-server architecture as, historically, they have been owned and operated in this fashion by network operators.* A P2P-IMS involves the distribution of some of these nodes towards the edge of the network. In a P2P-IMS content storage, media processing and transcoding are all likely to be delegated to peer applications running on various mobile devices or set-top boxes.

In details, the P2P-IMS client will need to inherit basic P2P functionalities for content search, content retrieval/distribution, content mixing and IMS functionalities such as telephony, media broadcasting and full SIP support.

[9] Skype P2P telephony explained: http://www.skype.com/help/guides/p2pexplained/

7.3.1 UNI Signalling Interface

7.3.1.1 Functionality

The interconnection between the two technologies must take into account functionalities such as session setup, management, update etc. More specifically, in the session setup the use of SIP within the P2P architecture must take into account the centralized nature of SIP while the use of P2P must benefit the whole system with a distributed management, provide a constant update system between the peers, while maintaining the QoS and authentication/authorization capabilities that IMS technology provides.

The main functionalities that this User Network Interface will provide is:

- Authentication/authorization Security
- Session setup & Management
- QoS Awareness
- Context advertisement

7.3.1.2 Protocols

The requirements for the creation of the VITAL++ client and most specifically the protocols needed for the UNI Signalling Interface are presented below: the XCAP for the user data management and the necessary extension on the IMS/SIP protocol for supporting parallel different media streams, P2P authentication, content security, overlay maintenance and content indexing as described in D2.2: VITAL++ Trial network description, integration description.

IMS uses the Session Initiation Protocol (SIP), originally standardized by the IETF, as its base signalling protocol. SIP is an internet protocol accommodating convergence between the Telco and the internet World. SIP enables signalling between different network entities, including endpoints and servers.

Recently, a number of emerging SIP-based control mark-up languages are extending SIP's capabilities in media server control. The use of this mark-up language can provide the necessary background for the interconnection between IMS and P2P technology.

Extensions such as Media Server Control Mark-up language (MSCML), Media Sessions Mark-up Language (MSML), Call Control XML (CCXML), Voice XML (VXML), Media Server Control (MEDIACTRL), etc. can be the base for the interconnection of SIP/IMS based P2P clients, such as the envisaged VITAL++ client.

7.3.2 UNI Transport Interface

7.3.2.1 Functionality

Three main usage features are supported for the UNI transport interface a) content distribution, b) P2P assisted Video on Demand, c) P2P live streaming. These three scenarios are analyzed below.

- **Content distribution**: A server contains large amounts of data and they must be transferred to a vast number of peers. BitTorrent is the most widespread implementation of a content distribution mechanism.
- **P2P assisted Video on Demand**: The peers that are viewing the publisher's videos also assist in redistributing the videos. Azureus is one of the most popular examples.
- **P2P live streaming**: 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. One of the main P2P live streaming clients is SopCast.

Because of their different nature and objective, the characteristics demanded for each of these scenarios may differ from each other. Moreover, the evaluation is needed in order to determine the characteristics and their compatibility with the ones specified in the IMS platforms. One of the objectives of this document is to shed a light on these aspects.

7.3.2.2 Protocols

The Protocols available for the transport part are presented together with the audio/video codecs that can be used for the UNI Transport Interface:

Transport:

- UDP
- TCP

Protocol:

- RTP
- MSRP
- HTTP

Codec:

- Audio: G711, AMR, GSM, MP3, AAC
- Video: H263, H264, MPEG2, MPEG4

7.4 VITAL++ Client Architectural Components

7.4.1 IMS related Components

7.4.1.1 P2P Authentication

Each peer needs to have a solid way of being authenticated to the overall system and to integrate with the system. This integration might involve communication to other peers and communication to server entities. In all forms of those communications, the identity of each peer should be authorized so that all involved entities will be able to trust each other. While password authentication might seem an adequate solution, the authentication that this solution provides is considered very weak. Strong authentication is achieved through the use of public key cryptography (digital signatures) and hash functions.

During authentication each peer must be able to reliably verify the identity of another peer. This can be realized by asking the other peer to provide some credentials proving the authenticity of its identity. Only then can the two peers trust each other and exchange data. However, how can a peer be certain that it communicates with peers that are what they claim to be? This authenticity can be provided by a third party trusted authority known as Certificate Authority (CA, see 6.2.1). The role of the certificate authority is to issue specific certificates for the identity and characteristics of each peer. *In the rest of this section we provide an initial solution towards a distributed and fast authentication between two peers. It is an important requirement in our system as we need the continuous reorganization of the overlay due to peer arrivals and departures and also due to the reaction of the content diffusion overlay to changes in the underlying network.*

In a strong authentication scheme, each peer has a specific, unchanged identification number and a private, public key pair. The public key is not kept secret and can be transmitted through the communication channel while the private key is kept secret and remains unknown to all other involved entities apart from the peer with which it is associated. However, for the peer key pair to be valid, a certificate of this key pair must also be addressed to the peer. This certificate can be transmitted along with the peer's public key during authentication with other peers so as to verify that the transmitted public key is not forged or altered by an eavesdropper performing spoofing or playback attack.

The peer key pair authentication and certification general idea is as follows, assuming that peer A has to authenticate its public key with peer B using a Certificate authority CA. Note that CA has a public – private key pair itself.

1. Peer A generates a public-private key pair.

2. Peer A transmits its public key and identification number to the CA and asks for verification.
3. The CA signs peer AA's identification number and public key with its private key and sends the result (certificate) back to peer A
4. Now, when a request for communication with peer B is required, peer A sends along with its public key the certificate provided by the CA.
5. Peer B, upon receiving peer AA's certificate, uses the public key of the CA to decipher the certificate and to verify that the public key provided by peer A is indeed the one that exists inside the certificate and therefore is legitimate.

The authenticity of the CA public key can be certified by another CA or can be considered trusted by default.

The certificate authority issues X.509 certificates and is part of an overall Public Key Infrastructure (PKI) system that is set on an Authentication Server (AS). The PKI system can provide a complete management of public, private key pairs for its associated peers. More specifically it is responsible for:

1. Management of each key pair life cycle
2. Backup and recovery of keys
3. Update of key pairs and certificates
4. Issuing certificates

Note that the PKI can be enhanced to offer an authorization functionality to the system by embedding the authorization rights of a peer into the certificate provided to this peer.

The authentication procedure between peers follows the ITU – T X.509 Authentication Framework.

More specifically, assuming that peer A needs to authenticate itself to peer B:

1. A generates r_A, which is a non-repeating number that is used to detect replay attacks
2. A sends B the following: *{r_A, t_A, ID_B, A-certificate, A-pub.key, sign Data}* where t_A is a timestamp indicating the expiration time of the transmitted message, ID_B is the identification number of B, *A-pub.key* is the public key of A and *A-certificate* is the certificate of A. Finally, *sign Data* is a digital signature of the whole transmitted message needed for data integrity.
3. B verifies the certificate of A by the Certificate authority
4. B verifies the *sign Data* and thus the integrity of the sent information

5. B checks that ID_B belongs to himself and therefore is the intended information receiver
6. B checks if the timestamp is up-to-date
7. B checks if r_A is replayed

If all goes as planned B repeats the same procedure to authenticate himself to peer A and the authentication procedure is concluded.

7.4.1.2 Digital Rights Management

The way that two peers authenticate doesn't provide the necessary reliability for the exchange of content. 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 system. In order to understand how DRM works and why the authentication of P2P systems does not provide the results for a trustworthy environment an example is given:

The scenario is that BOB encrypts a book and sends it to Alice via an e-mail telling her the key to open the file. Alice now has the file and the key in order to open the book providing her the ability to redistribute the book to anyone. The whole trade presumes that there is trust between Bob an Alice but with the use of systems like Gnutella, BitTorrent and the like the word trust is in a way eliminated as a preference. The question that arises from the example is how can I send the book in a way that Alice will read the content of the book but not have the necessary rights to redistribute it?

Here comes the DRM scenario: Alice will take the book but at the same time she will return a piece of identifying information about her computer. This may be an identification number of her CPU, a serial number from her hard drive or her BIOS. Now when Alice will try to open the file on her computer, the file will include the trace of Alice's unique hardware identification so that the program that opens the file will not work if the hardware of the current machine doesn't match the hardware ID in the file sent to Alice.

Nowadays the binding of digital files to a particular piece of hardware is common but this binding has obvious drawbacks in a world where every two to three years the whole hardware is renewed. In order to create a better solution the binding between the IMS profile and the digital file could provide the necessary answer for the clients in order to move their files from one computer to another in the way that you pack up your books and move them from one house to another.

7.4.1.3 IMS Session Management

IMS is going to be used in Vital++ as a means to provide an enhanced control plane for P2P networks and transactions. In this way the IMS network will provide controllable and secured access to content indexing functions and an additional effort will focus on the exploitation of information available to IMS entities for aiding the P2P overlay initialisation and maintenance. For this purpose standard IMS procedures such as SIP calls, Presence Publication/Subscription, and Instant Messaging will be used for the provision of added value services targeting at the exploitation of acquired information in the P2P operations. The aim is not to modify these standardised SIP procedures but to map them on P2P application-specific needs.

Since the main contribution of IMS to P2P in this combination of technologies is based on control and access aspects, the expected overhead in the setting up of P2P channels by use of IMS procedures needs to be compensated by the fact that the joined P2P overlays will have significant performance and stability in content transfer. Session setup is a key element in SIP operation. It is intended therefore to utilise it not only for the obvious reason of setting up sessions for content transfer among peers, but also for exploiting the IMS network mechanisms for QoS in order to ensure the proper provision of the added value services.

The Session Description Protocol (SDP) of SIP messages might be enhanced to communicate the setting up of peer to peer channels. Although this procedure might require additional steps until the actual channel is ready to transfer content, the imposed overhead can be outweighed by the fact that the relevant channels will be automatically added to QoS mechanisms and be served as normal SIP sessions. The IMS user control and access policies can guarantee that received session setup requests have been previously authorised and authenticated by the system.

Additional sessions can be added on the fly and normal session tear-down procedures can be also put in effect.

In summary, content publication and discovery can be supported by SIP Application Servers that process the SIP exchanged information in a way to aid P2P procedures and also all peer to peer content exchange channels can be initiated and maintained in a SIP/IMS fashion.

7.4.2 P2P related components

7.4.2.1 DHT and Distributed Queries

When placing information into the DHT there are several considerations necessary for our scenarios. The likelihood is that the devices using the Vital++ system will be devices with limited resources, i.e. smart phones, set top boxes, net books, etc. and therefore will have limited storage capacity and processing power. These system resources will place constraints on the overall architecture of the P2P mechanisms employed. For instance, with Bit Torrent, a file is broken up into small parts and transferred amongst peers. The chunk size of the file needs to be small enough so that they are of a size optimal for a peer to download and don't lead to a massive torrent file. If a chunk is too large and during download it is corrupted then the down-loader will be forced to discard and download that chunk again. So a larger chunk size can mean a slower overall conversion of "*leechers*" to "*seeders*", slowing down the maturity of the swarm as a whole. The average size of a chunk for desktop based P2P clients is 256KB so some experimentation will probably be required to find an optimal chunk size for an IMS based UE.

Another point to consider will be if Vital++ will provide a Tracker server or will the use of public based Trackers suffice? Or will Vital++ use the officially supported extension, Distributed Trackers? These questions do not need to be addressed at the moment but a decision will need to be made in the design and implementation phase of the project.

The centralized Tracker is a major barrier to a fully decentralized system. The Distributed Tracker is BitTorrent which utilizes a DHT to remove the reliance on a centralized Tracker. Instead of the Tracker being a web server it can be the original seeder or a set of nodes in the DHT can be randomly picked to act as the trackers. It would possibly make sense if the Trackers are distributed amongst the super-peers in the overlay as these super-peers should have the necessary resources to sufficiently process any requests made to it for content. As IP addresses are assigned during registration, the only constant which can be used to identify the peer with the content in the tracker would be the SIP public address.

Maintaining a correct list of current valid sessions, currently connected peers, is also a problem in P2P systems. The requirement is to design a system which is robust enough to handle such an environment. Most existing DHT systems, e.g. Pastry or Chord, with excess churn return inconsistent data or suffer from high latency regardless of the lookup or routing mechanism employed. To handle this churn, there are several approaches which can be considered: Reactive vs. Periodic Recovery, Timeout calculations, Proximity Neighbour Selection.

With Reactive Recovery the system reacts to loss of a leaf node instantly, by sending out the updated leaf set to every node in the set ($O(k^2)$). Periodic Recovery uses a periodic update where each node shares its leaf set with every neighbour, which in turn responds with their own leaf set. In terms of scalability reactive recovery uses less bandwidth when there is little churn in the system but when churn increases, bandwidth usage jumps dramatically. Therefore, with networks which are prone to high rates of churn the Periodic Recovery solution is the most effective.

Timeout calculations are important when nodes disappear frequently, e.g. due to a temporary loss of network signal. The nodes have to find a reasonable timeout value. If it is too short the node can experience congestion and increased processor load, too long and the node can experience unnecessary waiting and increasing queue lengths. The peers can use several strategies to combat this. The simplest is a fixed/static timeout. This solution is easiest to implement but not very useful in the long run. The next method is to use a TCP-style timeout where each node maintains a list with each neighbour's response time in the past. Another way to approach this is to use timeouts from virtual coordinates. Each node in the DHT is assigned a synthetic coordinate. Every node then computes the distance to other nodes by sampling network latency. This information is then used to predict a coordinate and this is then shared with other nodes. An example of this type of approach is the distributed algorithm Vivaldi. In systems which experience significant amounts of churn, TCP-style timeout has been found to perform the best.

Proximity Neighbour Selection is used to build the routing table. Latency is generally lower between the nearest neighbours. The best approach is to update proximity periodically as neighbours change often. Global sampling uses prefixes to lookup new neighbours. This method is effective and gives an almost optimal proximity in most cases. Recursive sampling is also proven to be effective in some network settings. The local node retrieves all of the nodes at the top level of the routing table. Then the latency is measured for each and the best is kept. The local node then checks the next level neighbours for the nodes it kept. Overall global sampling has been found to give the least latency. See section 7.2.4.3 for a more detailed explanation.

DHTs already use proactive replication of meta-information e.g. keys and pointers for search efficiency. Proactive replication of copies of each data-item can help search efficiency, load-balancing and availability. With the Tapestry DHT implementation data itself can also be replicated, in which case pointers to the data contain pointers to all copies of the data. Chord can support data replication by storing a list of nearest successors on each node. Due to the routing scheme, this method of replication does not achieve load balancing, only data redundancy. There is no need to maintain availability of cached data as the DHT is used to maintain pointers to copies on the peers. File replication methods in structured P2P networks determine replica nodes based on node IDs or the query path. ID-based methods determine replica nodes using on the relationship between the node ID and the file's ID, while path-based methods choose replica nodes in the file query path from the file requester to the file provider. Both methods assume that replica nodes have available capacity for replicas. However, this cannot be assumed due the nature of the peers within Vital++ as the devices running Vital++ clients will most likely be running with limited resources, e.g. smart-phones, set-top boxes and net-books.

7.4.2.2 Content Diffusion Overlay

In this chapter, according to the technical objectives that we derive from the requirements that we described in section 6, we describe in an abstract way the overlay architecture where each node enters in order to acquire content or live video streams. For each object (file, video, audio) we form such an overlay as described later in order to distribute it through a P2P architecture.

The purpose of the content diffusion overlay architecture is to achieve fast and stable diffusion of each block of an object with high upload bandwidth utilization. Towards this goal we need an overlay where each node selects dynamically close neighbours in the underlying network, a graph structure where nodes with heterogeneous bandwidth capabilities are able to utilize all their upload bandwidth, and finally we focus on an overlay adaptable to the dynamic node arrivals and departures.

Every node that enters the system participates in an overlay that we call base overlay. Every node of the base overlay has an equal number of neighbours. We note the set of neighbours of a node i of the base overlay as $M_b(i)$. The base overlay is a symmetric graph, where adjacent nodes exchange blocks and control information in both directions.

As in real P2P live streaming systems nodes have heterogeneous upload bandwidths we construct another overlay, the top overlay, which includes only the nodes that have upload bandwidth greater than the rate of the video stream. In the rest of this section we will call these nodes *super nodes* and while nodes with upload bandwidth less than the video stream rate will be called *slow nodes*. Exactly like the base overlay, the top overlay is a symmetric graph and all of its nodes have an equal number of neighbours. We note as $M_t(i)$ the set of the neighbours of a super node i. In order to fully exploit the upload bandwidth of the super nodes, as we describe later in detail, we interconnect the two overlays by connecting super nodes with a number of slow nodes proportionally to the difference between their upload bandwidth and the rate of the video stream. We note the set of connections of a super node i as $M_S(i)$. In order to ensure that each slow node will acquire the video stream, we spread these connections to the slow nodes uniformly. For slow node i of base overlay, we note the set of super node neighbours as $M_l(i)$.

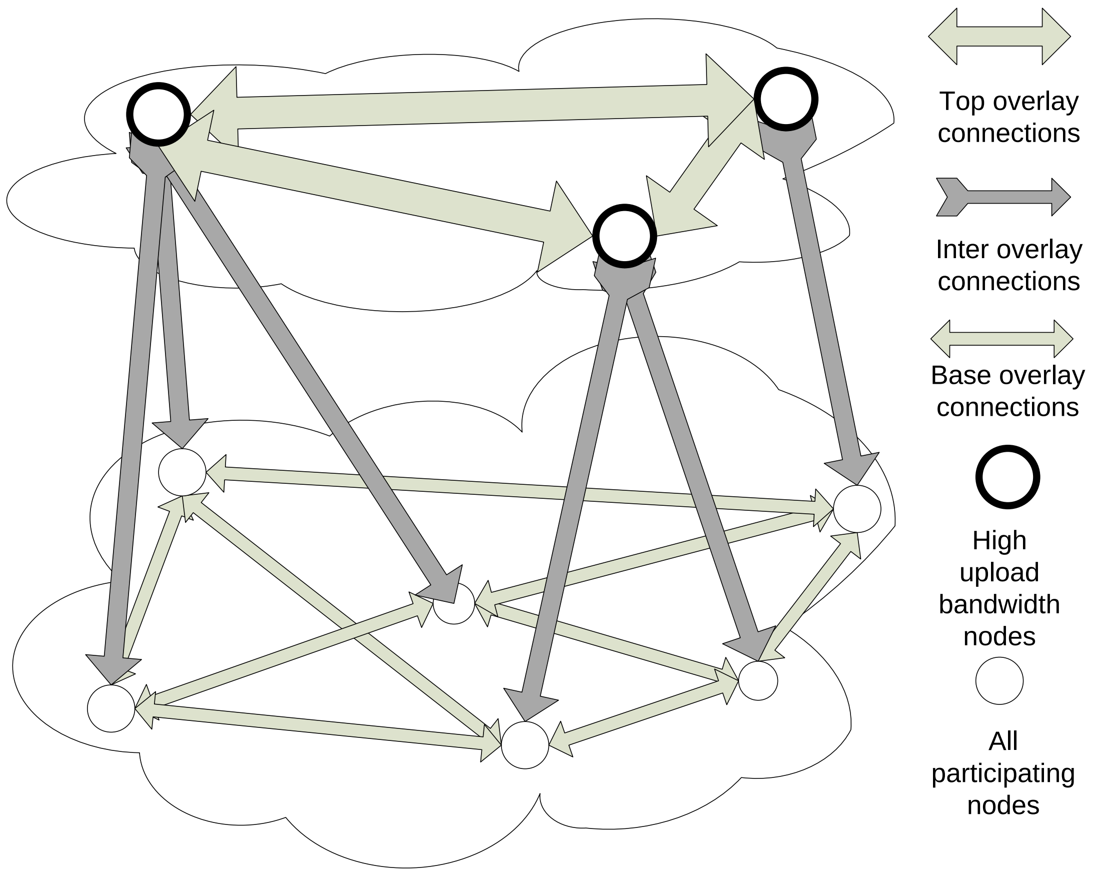

Figure 5: Graph architecture of the content diffusion overlay

In this figure we present an example overlay architecture where nodes of the base overlay have $|M_b(i)|=3$, nodes of the top overlay have $|M_t(i)|=2$ and slow nodes of the base have $|M_l(i)|=1$. The inter-overlay connections of super nodes $M_S(i)$ are proportional to the difference between their upload bandwidth and the video streaming rate.

The purpose of the *top overlay* is the relatively fast and complete diffusion of the video stream between the super nodes. The equal number of neighbours for each node of this overlay ensures the good graph connectivity and the low graph diameter. Furthermore, the average upload bandwidth in every neighbourhood of this overlay is higher than the video stream rate. Combining these two facts with the neighbour selection function, which prefers nodes with high upload bandwidth, we can guarantee that all super nodes will successfully acquire the video stream.

The purpose of the *base overlay* is the utilization of the upload bandwidth of slow nodes. Neighbourhoods of this overlay have average upload bandwidth similar to the video streaming rate. Due to this property of the base overlay, we maximize the upload bandwidth utilization of each slow node.

The purpose of the *inter-connections* between super nodes and slow nodes is to utilize the bandwidth of super nodes that it is not used in the top overlay.

The remaining bandwidth of each super node varies and thus it is distributed to a number of connections proportional to the difference between its upload bandwidth and the rate of the video stream. These connections are spread uniformly to slow nodes of the base overlay. The neighbour selection function, using the inter-connections, is capable to ensure that the video stream is fairly distributed to every slow node.

Our overlay architecture combined with the neighbour selection function adapts the bit rates of the flows between the nodes of the overlay. This provides us a tolerant and stable P2P live streaming system, which is minimally affected by the dynamic network conditions and node behaviour.

In order to dynamically manage our overlay we will develop two distributed algorithms that run periodically in our system. The first algorithm is called Intra-overlay distributed optimization algorithm (Intra-DOA) and it is responsible for the dynamic reconfiguration of the top and the base overlay separately. It determines the neighbours of each node in the overlay according to the network latencies between the nodes. The second algorithm is called inter-overlay distributed optimization algorithm (Inter-DOA) and it is dedicated to determining dynamically the number of the inter-connections of each super node and to redistributing them uniformly to the slow nodes according to the network latencies.

Intra-overlay distributed optimization algorithm (Intra-DOA)

This algorithm aims at keeping both overlays dynamically reconfigured and optimized with respect to node latencies in the underlying network. Intuitively, the goal of this algorithm is to maintain a balanced overlay where nodes have equal numbers of neighbours and where each node has neighbours physically close to it in the underlying network. Three events may trigger two neighbouring nodes to execute this algorithm: a) the arrival or departure of a node, b) a change in network conditions and c) the reconfiguration of their neighbours. All these events result in new neighbour sets and/or Stt values, which, in turn, invoke this algorithm to rearrange a graph neighbourhood in the base or the top overlay.

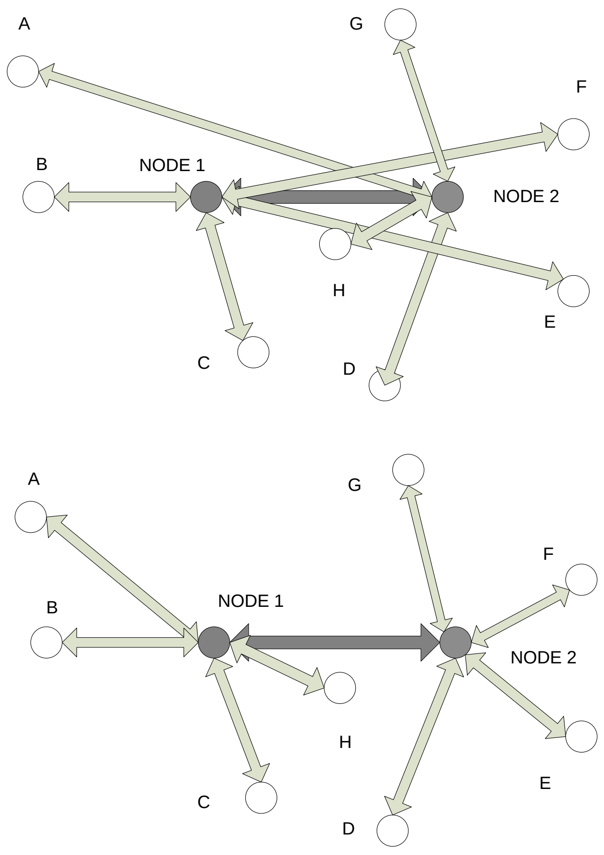

Figure 6: Iteration of the Intra-DOA

The upper figure shows 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. The figure below illustrates the new sets of the initiators' neighbours $N_{after}(1)=\{a,b,c,h\}$ and $N_{after}(2)=\{g,d,e,f\}$ respectively, after a single iteration of Intra-DOA. In this figure the length of the edges between two nodes is proportional to the network latency between them.

The objective is to develop an algorithm that will dynamically rearrange the overlay according to the "distance" between nodes. This distance could be the network latency between participating nodes or any other metric that expresses the suitability of neighbouring between two nodes. There are three requirements from this algorithm.

The first is its global convergence to an optimal overlay according to the aforementioned distance. The second is its distributed implementation and the third is its fast distributed execution in order to have an overlay that reacts fast under dynamic conditions.

Inter-overlay distributed optimization algorithm (Inter-DOA)

This algorithm aims to optimize in a distributed manner the inter-connections that super nodes use in order to diffuse the video stream to the slow nodes. It ensures that every super node has neighbours, which have small distance between them, and the number of these neighbours is proportional to the exceed bandwidth of each super node. We define exceed bandwidth of a super node i as BE(i)=c(i) – μ, where μ is the video streaming bit rate.

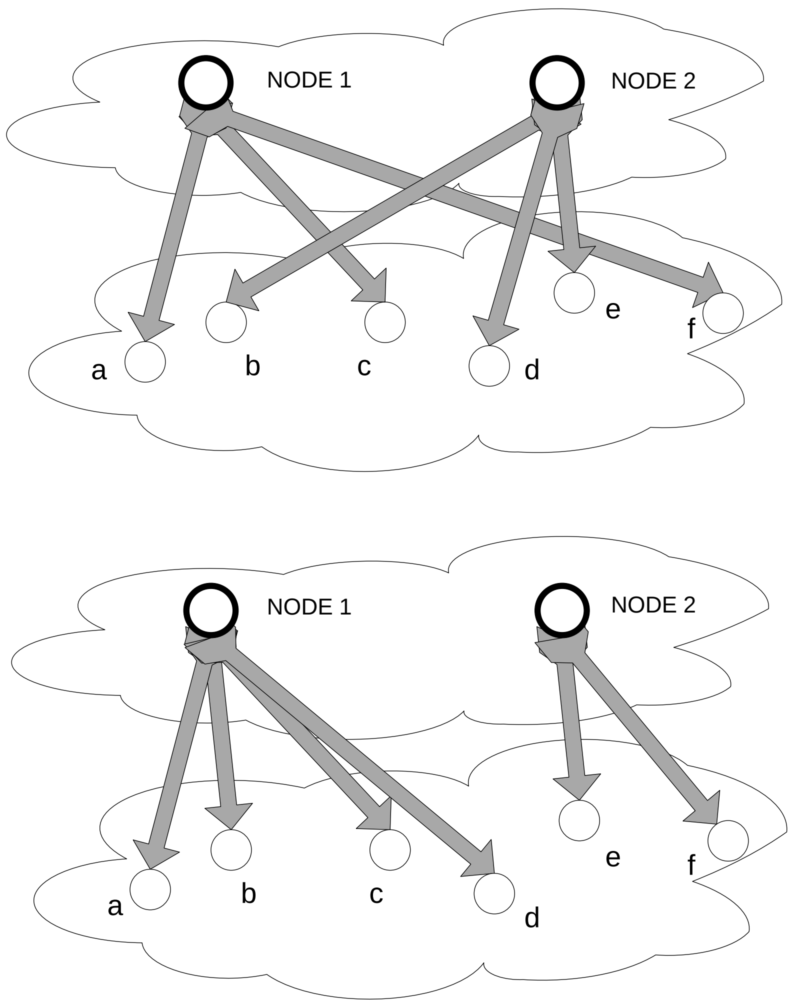

Figure 7: Iteration of the Inter-DOA

We present an execution of an iteration of the Inter-DOA where Node1 and Node2 are the initiators and participate in the top overlay. We consider that Node 1 has double exceeded upload bandwidth compared with Node2. The set of their neighbours are $Ms_{before}(1)=\{a,c,f\}$ and $Ms_{before}(2)=\{b,d,e\}$. After an iteration of Inter-DOA $Ms_{after}(1)=\{a,b,c,d\}$ and $Ms_{after}(2)=\{e,f\}$.

7.4.2.3 File sharing block exchange scheduler

After the insertion of a peer to the overlay it has a list of peers that have the file. The downloader then initially establishes a connection to these peers and finds out what pieces reside in each of the other peers. The overlay maintenance is then responsibility of Inter-DOA and Intra-DOA as was described above.

Neighbour selection function

A downloader then requests pieces which it does not have from all the peers to which it is connected. But each peer is allowed to upload only to a fixed number of peers (a typical value is four) at a given time. Uploading is called *unchoke*. Which peers to unchoke is determined by the current downloading rate from these peers, i.e., each peer uploads to the four peers that provide it with the best downloading rate even though it may have received requests from more than four downloaders. This mechanism is intended to deter free-riding. Since a peer is only uploading to four other peers at any time, it is possible that a peer, say Peer A, may not be uploading to a peer, say Peer B, which could provide a higher downloading rate than any of the peers to which Peer A is currently uploading.

Therefore, to allow each peer to explore the downloading rates of other peers, we will use a process called *optimistic unchoking*. Under optimistic unchoking, each peer randomly selects a fifth peer from which it has received a downloading request and uploads to this peer. Thus, including optimistic unchoking, a peer may be uploading to five other peers at any time. Optimistic unchoking is attempted once every 30 seconds and to allow optimistic unchoking while keeping the maximum number of uploads equal to five, an upload to the peer with the lowest downloading rate is dropped.

There are two types of peers, namely *downloaders* and *seeds*. Downloaders are peers who only have a part (or none) of the file while seeds are peers who have all the pieces of the file but stay in the system to allow other peers to download from them. Thus, seeds only perform uploading while downloaders download pieces that they do not have and upload pieces that they have.

Block selection algorithm

In practice, a swarming network is a very complicated system. There may be hundreds of peers in the system. Each peer may have different parts of the file. Each peer may also have different uploading/downloading bandwidth. Further, each peer only has partial information of the whole network and can only make decisions based on local information. In addition, BitTorrent has a protocol (called the *rarest-first policy*) to ensure a uniform distribution of pieces among the peers and protocols (the so-called *endgame mode*) to prevent users who have all but a few of the pieces from waiting too long to finish their download. Like with any good modelling exercise, we trade-off between the simplicity of the model and its ability to capture all facets of the protocol.

7.4.2.4 Live multimedia streaming block exchange scheduler

Without loss of generality, we assume that in a P2P streaming system there is a bootstrap node which acts as a source for providing the video stream. Furthermore, the video stream is divided into blocks. The block size depends on the service rate, say *μ* (measured in bps that the video playback requires), and the number of blocks into which the bootstrap node divides one second of video playback. We define this number as N_b *blocks/sec* representing also the frequency of new blocks generated by the source. So each block is generated every $1/N_b$ *seconds* at the bootstrap node, with a size equal to $L_b=\mu/N_b$ *bits*.

Every block is also associated with a time stamp indicating the time of its generation. All peers reproduce (play) the video with a delay called *set-up* time which we denote as t_s. As mentioned above, setup time is the time that elapses from the generation of a block at the source until its distribution (propagation) to every node in the P2P system. Accordingly, at every time instant every peer plays the block that was generated t_s se before in the origin server, provided of course that this block has eventually reached its destination.

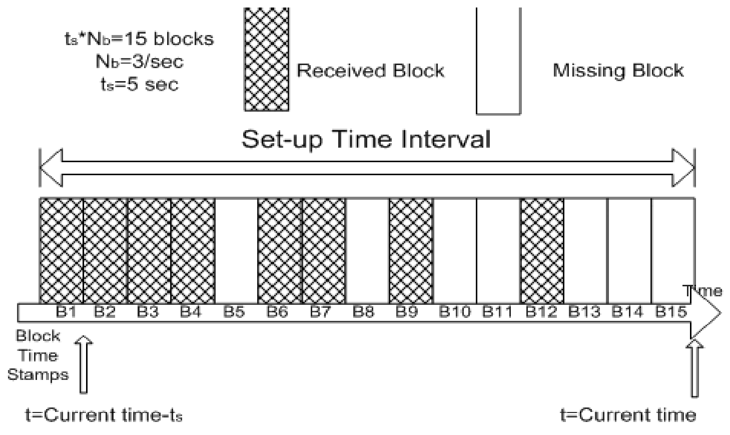

Figure 8: Snapshot of a buffer in a node with the states of the blocks

During this setup time a number of blocks have been generated, equal to N_b*t_s, the first of which will be played by every node after t_s seconds. Therefore, at every instant every node is required to keep track of all N_b*t_s blocks generated within a sliding window of t_s seconds. For this reason every node maintains a buffer of size N_b*t_s that holds the state of these blocks. Two states are of interest: received blocks and missing blocks (not delivered yet). Figure 8 provides a snapshot of the states of blocks of a buffer in a node.

Whenever the origin server produces a new block it forwards it first to a small subset of the peers that participate in the system. Each peer maintains connections and exchanges blocks with a relatively small number of nodes, which we call its neighbours, in order to retrieve the whole video stream. To know exactly which blocks should be exchanged, each peer exchanges the contents of its buffer with every one of its neighbours. Then a scheduler that runs in the nodes decides which block should be transmitted next to which neighbouring node.

The Neighbour Selection Function

Our scheduler architecture consists of a function that runs in every node and exploits the upload bandwidth of the node's neighbours and the situation of their buffers. Neighbours in the overlay periodically exchange between them their buffers and their temporal upload capacity.

We first define a decision function, d(i,j), that provides a metric used for the selection of a neighbouring node j for block transmission by node i. The decision function could be given by a formula similar with the following:

$$d(i,j) = \frac{diference(i,j)}{per * buf\_size} - \frac{rank(i,j)}{|neigbors(i)|} \qquad (14)$$

The node selected for block transmission is the one with the maximum *d(i,j)* $\forall j$. In this equation |*neighbours(i)*| denotes the total number of neighbours of i. Additionally *rank(i,j)* is a function that returns the position of *node j* in a list with neighbours ordered in incremental upload bandwidth value. We have chosen to factorize the network bandwidths in this way in order to make our scheduler independent of the upload bandwidth values and as such suitable for every upload bandwidth distribution. *buf_size* that is equal to $N_b * t_s$ denotes the number of blocks that nodes exchange at each time instant. Finally, the parameter *per* is a constant representing the percentage of the buffer size. We have successfully experimented with values of parameter *per* close to very small percentages of buffer size (between 5%-10%).

Content diffusion optimization algorithm

In the previous section we have analyzed the factors that affect the selection of a node for block transmission. In this section we will focus on the mechanism that determines which block should be sent to the selected node aiming at minimizing duplicate block transmissions and at fast diffusion of newly produced or rare blocks within an overlay "neighbourhood". The decision is receiver driven, in other words the receiver sends periodically in each of its neighbours a block id that is a suggestion in case that they select it for block transmission. A new reception of a suggestion overwrites an older.

Due to the symmetric property of the overlay, a receiver node is already informed about the buffer contents of its neighbours because it is also a potential sender to them. Therefore, by applying a matching process a receiver node can proactively request different blocks from its neighbours thus resulting in the elimination of duplicate block transmissions. The matching process is accomplished by performing a weighted matching algorithm between the missing blocks (as presented in the node's buffer) and those neighbours that have them (as presented in their buffers), while favouring requests to those nodes that have higher upload bandwidth capabilities. Accordingly, whenever a node's neighbour selection function chooses the next node for transmission, it also takes into account whether the selected node has already requested a specific block and if there is no request it selects one node randomly. Further details for content diffusion optimization algorithm will be analyzed in WP3.

7.4.2.5 P2P QoS Functionality

A function acts as manager in order to ensure that for each file or video that we distribute there are enough resources with respect to the user or application requirements. As we described earlier, through our self-organized P2P overlay and our schedulers we are capable to fully and optimally exploit the upload bandwidth of the participating peers. On the other hand, these bandwidth resources may not be able to meet the application requirements or the user agreement. For example, an overlay where peers have an average upload bandwidth of 900 kbps is not able to deliver to its peers a video stream with 1000 kbps.

In this case we can develop two mechanisms that will further enforce the capabilities of our system towards QoS optimization.

The first is further bandwidth provision from the centralized servers. This technique will be applied in content distribution and in live video streaming.

The second is the distribution of encoded videos with different playback bitrates according to the available peer and server resources.

When a peer enters an overlay in order to acquire a file or a video its upload bandwidth is kept by a function that we call Resource Management Function (RMF). In order to keep our system scalable we do not report to this function through a time driven protocol but only when we monitor in an upload bandwidth a change above a predefined threshold. In this way, each time instant the RMF is able to calculate the aggregate upload bandwidth of all peers, who participate in each distribution of a specific file or video (in other words the sum of their upload bandwidths or the average).

Additionally, it is able to calculate the additional upload bandwidth that is required. So it can calculate the excess bandwidth that is required.

Then a resource provisioning entity (RPE) (QoS- Coordinator) contributes the additional resources that are required in order to have an object distribution that meets our requirements. The total bandwidth that RPE has to provide, each time instant, is equal to the difference between the requirements of the video stream or file distribution multiplied by the number of users and the aggregate upload bandwidth that the RMF reports. There is no need for a server to provide to a specific peer more bandwidth than its application requires. On the other hand, with the provision of its content from the server, other peers are not consuming upload bandwidth for it and through our distributed scheduler they can feed the rest of their neighbours with content.

In the remaining WPs we will develop an algorithm that takes care of this functionality.

When we consider that the aggregate upload bandwidth that peers have and the additional bandwidth that servers can provide are insufficient for the distribution, the only thing that we can do is to lower the video playback rate. On the other hand, when we have much more resources and peers with high upload bandwidth we can raise the video playback rate. Towards this goal there is a need for a decision function (RSF – Rate Selection Function) that will select the appropriate play back rate according to the dynamic behaviour of peers and available bandwidth. Additionally, in order to apply such a technique we need the video to be encoded in different rates and stored to a specific server so that this function will be able to select the appropriate one.

7.4.3 Content related components

7.4.3.1 Content Indexing

One significant client feature is that it must allow for content searching and publication to be done without the use of separate protocols and software (Internet browser). This is achieved by the use of existing SIP procedures in IMS. The innovation is based on the use of Application Servers that collect the communicated information and process it in a way that allows for queries to be executed against the accumulated information so that clients can be fed with results that are meaningful in the process of locating media items that can be shared in a P2P way.

Application Servers, apart from storing the details regarding the clients that publish or retrieve through P2P any kind of content and thus being candidate members of further overlays, they store information relating to the nature of the published content. The information sets may include codecs, bitrates, file sizes and other media related details the circulation of which to the involved peers may ease and enhance content processing especially in the case of the video or audio items the reception of which is of live or nearly live (on demand) nature.

This information, unless created and maintained by the Application Servers (QoS and topology related), is provided by the originating content sources. The mechanism for transferring it is based on the use of SIP message fields such as the Message Body of an Instant Message, generic tags.

7.4.3.2 Content insertion and content removal

Second generation P2P networks like Tapestry, Chord, Pastry and CAN implement basic key-based routing interfaces (KBR) which support deterministic routing to a live node which has responsibility for that destination key. Tapestry and Chord construct locally optimal routing tables from initialization and maintains them in order to reduce routing search. Tapestry allows applications to publish according to their needs by publishing location pointers throughout the network to facilitate efficient routing to these objects.

Tapestry dynamically maps each identifier *G* to a unique live node known as the identifier's root *Gr*. To route messages, each node maintains a routing table consisting of node ids and IP addresses of its neighbour nodes. When routing towards the root node the messages are routed across neighbour links to nodes whose node ids are progressively closer to G in the ID space.

Each root node inherits a unique spanning tree for routing its messages from the leaf nodes traversing the tree until reaching the root. This property can be utilized to locate objects by storing meta-data across nodes including the root. For example, a Vital++ client has a file or object *O* with a unique *GUID* O_G and a root node O_{R5}. The Vital++ client should periodically advertise or publish this object by routing a publish message towards O_R. Each node along the publication path stores a pointer mapping, not a copy of the object itself, although object replication and caching are means by which further guarantees can be given to ensure that objects are available to other nodes when the root node is unavailable. When separate clients have their own copies or replicas of a file or object then each client publishes its own copy.

8 Specific Clients Adaptations

8.1 Monster Client

This chapter of *Specific Clients Adaptations* compares the results of the previous chapter *Generic Client Architecture* with the FOKUS MONSTER framework and depicts its components.

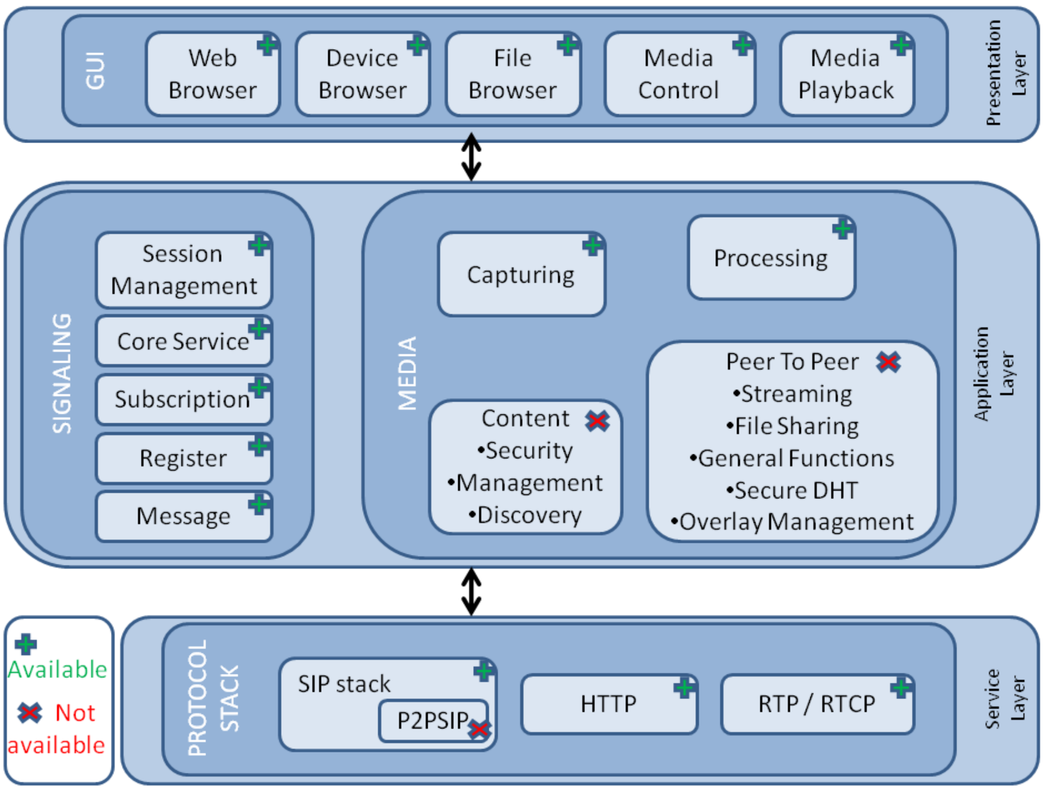

Figure 9: Layered architecture of MONSTER depicts adaptations

Figure 9 above shows the layered architecture of MONSTER grouped by those components which are most important in the context of this chapter. The *available* and the *not available* symbols depict whether the MONSTER framework contains the specific component or not.

8.1.1 IMS related functions

Since MONSTER is an IMS Client, it is absolutely applicable for all the required IMS related functionalities. The components used for IMS are divided into the three layers: Service Layer, Application Layer and Presentation Layer.

The **Service layer** contains the protocol stack. The complete IMS signalling and messaging is done through the SIP stack, which is once instantiated by the client on start up.

The signalling plane of the **Application Layer** contains the main IMS signalling functionalities, which are offered by MONSTER.

These are:

- *Register* (and *unregister* as counterpart) an IMS Client at the IMS Core. The registration is done for a specific duration time (*expires*), which has to be determined at the registration process. This is an optional parameter, which is overwritten by its default value, in case of misuse or when it is missing.
- A *session* represents a multimedia connection between two or more devices or to a third party resource. The Session Management is used to dynamically optimize the session's parameters in a specific way. A session is needed to exchange information between two or more devices.
- One of multiple methods for IMS client communication is *messaging*, which is done with the SIP request type *message* at the service layer. The application layer contains an interface for the instant message type *page message* and its implementation provides full message functionality to send an instant message over SIP.
- A *subscription* monitors changes of event states of remote devices or remote resources. A change of a state of a subscribed event triggers the referring signalling from the IMS to the subscriber.

The **Presentation Layer** contains the Graphical User Interface (GUI) which offers the functionality of the underlying layers to the layer on top and thereby to the user. Received and sent messages are displayed and tasks like e.g. *create* new message, *delete* or *reply* to a message as well as the management of contacts are provided at this stage. The presentation layer is independent from the display technology. It supports JAVA SWT for mobiles as well as SWING for rich clients.

8.1.2 P2P related functions

The requirements concerning the domain of P2P are not supported by MONSTER and will need to be developed in the course of the project.

The protocol stack of the Service Layer has to be extended with a protocol to provide communication in a P2P overlay. The existing JAIN[10] SIP stack in MONSTER is extensible, e.g. with the protocol P2PSIP[11]. The main advantage of this extension would be to reuse most of the suitable functionality of the SIP stack and to get a dual communication channel in turn.

The Peer To Peer plane of the Application Layer will contain a Secure DHT unit which enables security issues, something that is missing in the normal P2P

[10] https://jain-sip.dev.java.net/

[11] http://www.p2psip.org/drafts/draft-ietf-p2psip-concepts-02.txt

environment. Especially a Public Key Infrastructure (PKI) allows authorization and authentication between clients which act as peer or lets a client prove/check the integrity of content.

As a supplement, it provides the general functions to participate and act in an overlay using a DHT. These functions are *join*, *leave*, *put* and *get* as well as time driven management tasks to maintain the overlay e.g. fixing the finger table.

8.1.2.1 Streaming

The streaming module provides an interface to demand a specific stream and to retrieve an overlay position in a distribution graph as a set of neighbour peers to exchange data.

This set can be divided into two subsets, one with peers to receive data from and a second subset of peers to send data to.

This overlay information could be generated by an *Overlay Construction and Management AS*. The neighbouring peers exchange tables over P2PSIP in which available parts are listed. An automatic or an implicit request let the initiator receive a chunk map of a neighbouring peer. The module receives the chunk map request and answers them by sending its own chunk map.

The same module demands and receives specific parts (part=chunk=set of continuous RTP packets) of the requested stream from neighbouring peers. Thereby peers can be queried automatically. It also handles receiving chunk-requests and serves them with the demanded chunk or a suitable error message.

A received chunk is stored in a buffer in which the available parts are ordered referring their playback position. Additionally, a scheduler assigns priorities to missing chunks to ensure interruption-free playback by enforcing the requesting of higher prioritized chunks. Eventually, the ordered parts are combined and reassembled to one media object, which can be played in the Media Playback.

8.1.2.2 File sharing

A module for file sharing arranges peers with demands for identical static content objects in a way to let them exchange data. Therefore, one file object is divided into several chunks, which are managed in local maps to keep track of their availability status. This module supports the exchange of chunk maps where maps can be sent and received bidirectionally. Additionally, requests for chunks are addressed or received in return with a proper protocol. A positive response for a chunk request is answered with the referring chunk that initiates a communication for explicit data transfer.

8.1.3 Content related functions

The control of media with MONSTER is realized with the use of the Java Media Framework (JMF[12]). All received streaming media is processed at the Application Layer as well as all sent media such as live streams which had to be captured previously as depicted in Figure 10 below at the index *2*.

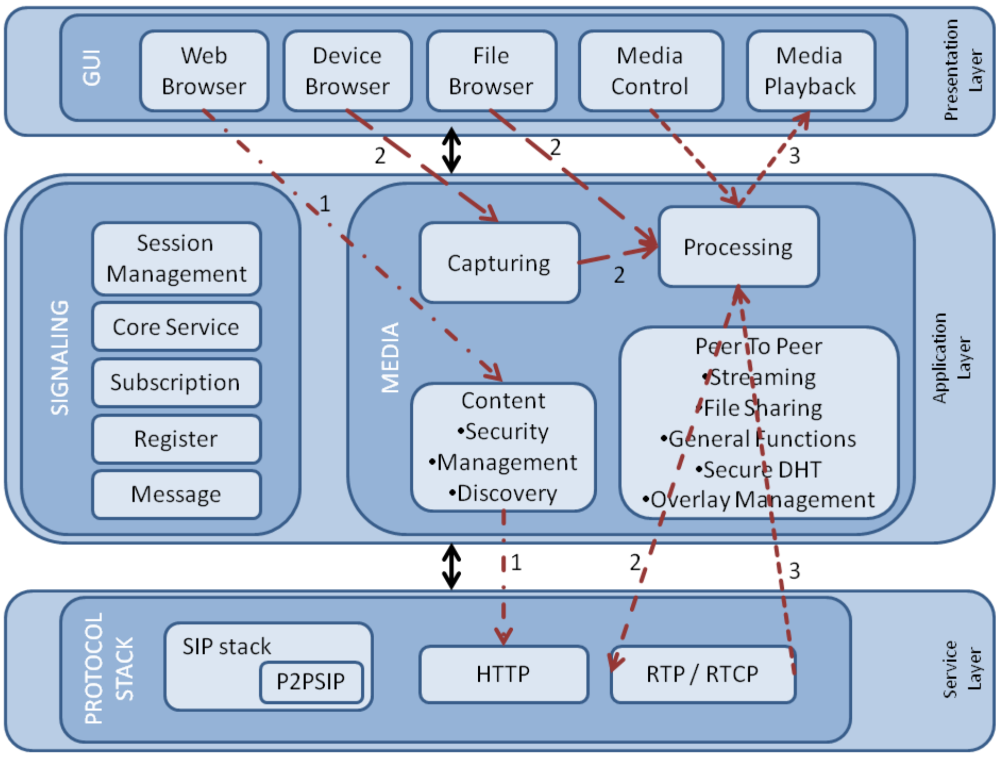

Figure 10: Signal- and workflow for media exchange

A demanded media object and its referring streaming type are clearly identified, e.g. over a Content Index on a website with a web browser as depicted in Figure 10 above at the index *1*, which is supported by MONSTER.

A content creator selects a device out of a device list with a device browser and a content owner offers a file object via file browser to the processing unit. At this stage *filters* or *format encoding* are applied to captured media streams or existing static file objects that are going to be streamed. The processing unit instantiates a data source and creates an outgoing RTP/RTCP stream. MONSTER offers all the listed and necessary functionalities up to this point with the use of JMF. The Content Management unit applies demands (*format*, *used*

12 http://java.sun.com/javase/technologies/desktop/media/jmf/

filter) of the content creator to the stream and announces the stream, e.g. at a Content Index.

The Presentation Layer provides media control functionalities for playback control (*start*/*stop*/*pause*/*skip*) and media discovery, e.g. via web browser. The content is managed with tasks like *select*, *store* or *delete* which in turn are performed in the Content plane of the Application Layer as depicted in Figure 10, index *3*.

Additionally, the Media Control monitors the status of open connections to other peers with details like throughput, finger table entries or other transmission statistics.

8.2 BCT Client

8.2.1 IMS related functions

The BCT client is a Java application that runs on top of an internal SIP machine built entirely by BCT. Recently, it has been redesigned to allow for SIP extensions according to application needs. The client was initially used for the provision of on-demand video streams from an Application Server through pure IMS networks. It also supports applications, such as collaborative drawing and chatting, that were built as proof-of-concept by use of a proprietary protocol running over SIP Instant Messaging (IM).

Built initially as an IMS client and having been tested as such in an IMS environment, the BCT client supports already or is planned to be adapted to support the following SIP and IMS related functions:

- **Registration:** The initial module was adopted to support AKAv1-MD5 so that it is compatible with Fokus' OpenIMS platform
- **Instant Messaging**: Generation of SIP IMs and also handling of incoming ones can be a very fast and sufficient way for communicating control information. The current design allows for customisable use of messages by the application logic.
- **Call setup and teardown:** session setup by use of outgoing invitations is already supported, however, further SDP negotiations have to be implemented as well as handling of incoming invitations so that the client can support additional application logic beyond the initial User Agent ***Client*** functionality.
- **Presence:** SIP Presence capabilities have been recently started being embedded in the client. Again the design allows for customisable use of the presence logic.
    - **Publication:** The application should be able to publish the SIP/IMS identity of the user on a presence server. This capability should be modular so as to keep the published information up-to-date and

according to the changes in the underlying access network so that precise knowledge of the client's networking capabilities are available.

- **Notification Handling:** Apart from making available the details of its actual presence, the client should be also able to subscribe to events relating with other IMS entities so that it receives notifications regarding the presence state of these resources. Such an aspect can be used to update the view of the synthesis of overlays which the client might have joined prior to discovering any modification through P2P means.

- **NAT and Firewall traversal:** The BCT client should be also enhanced to adequately cope with networking restrictions so that the controlled media sessions can be successfully established and maintained. The level of these adaptations will be defined according to the network architecture design outcomes so that the project's needs are fully satisfied. However, full support of NAT and Firewall traversal is planned to implemented in a more generic framework.

The presentation module is based on Java Swing and the media playback engine is the one provided by the Java Media Framework. The media handling part will be reconsidered according to the final network architecture and may be replaced on demand to cater for any further functionality not currently available by JMF. Ongoing work is also focusing on creating a Mobile Client. The main concept is to keep most of the modules common for both platforms (desktop and mobile) and this was a major reason why a custom SIP engine and protocol stack have been developed.

8.2.2 P2P related functions

The BCT client, as already mentioned, is based on a pure IMS implementation which has to be enhanced with P2P capabilities. These are meant to be built around a P2P algorithms library. This library will be wrapped in a P2P engine that can be plugged in the client environment.

The integration of the P2P engine will allow message exchange and content circulation to be transparent to the engine's procedures. In this way, the integration of the two technologies can follow discrete steps leading from simple scenarios of IMS/P2P combinations to more complex ones by replacing certain parts of the P2P operation with functions and procedures provided by IMS. This will be realised along the following roadmap:

- Only initial overlay synthesis is provided by IMS Application Servers
- Overlay management is also done by these servers
- Experienced quality is reported by clients to dedicated AS that process the information for overlay modifications that are circulated among users
- Invocation of SIP session setup mechanisms for the content transfer channels

However, each step has to be evaluated regarding the achieved improvements or the problems that may impose on P2P operations.

The major adaptation, however, concerns the media handling infrastructure of the client since the content in VoD and live streaming will be provided through P2P content exchange schemes. The relevant adaptations are presented in the following paragraphs.

8.2.2.1 Video on Demand and Live Streaming

VoD is already supported by the BCT client. However, this feature has to be adapted so that P2P transport means are used instead of RTP. Live Streaming (LS) is a new feature to be supported but from the media handling point of view is quite similar to VoD. The main adaptation effort will be focusing on making JMF media handling libraries work over a different kind of transport medium.

The main aspects that development will particularly focus are:

- Provision of a metadata handling feature that will replace either the Session Description Protocol (SDP) in the case of RTP streams or the file processing mechanism for local files, so that the proper JMF codecs can be invoked on the fly.
- Implementation of a transport layer that will be hiding the P2P layer procedures from the media engine and that will also be creating a buffering space so that P2P content acquisition can be smoothly integrated with the media engine.

8.2.2.2 File Sharing

File sharing is considered to be simpler than the VoD and LS cases. Once the P2P engine has been integrated with the IMS part of the client it is matter of developing a proper file management console and engine so that local storage can be properly combined with the P2P engine block.

8.2.3 Content related functions

8.2.3.1 Audio/Video playback and capturing and play-out

The BCT client utilises the JMF package for playing video streams acquired from RTP connections that have been set up by SIP Invite messages. In the context of Vital++, the BCT client needs to be adapted so that it supports reception and playback of media streams beyond the session setup means offered by SIP and also from different transport protocols than RTP and mainly through P2P procedures. Additionally, the client should be able to capture video content through a webcam and microphone and stream it to the appropriate audience according to the corresponding use case. This feature will allow the client to operate as a live stream source.

There are therefore two major adaptations with respect to multimedia playback and play-out:

1. Introduction of a new transport layer that provides an interface towards the JMF classes that is compatible with the supported transports and media containers of JMF. Such a layer will be operating over the P2P content exchange engine and will be aggregating content and content metadata so that these can be fed seamlessly into the JMF environment for proper playback of the received stream whether this is VoD or a Live Stream.
2. Utilisation of the capturing capabilities of JMF and integration with the above described "transport" module for the proper injection of the outgoing content into the P2P overlay.

8.2.3.2 Present a graphical user interface (GUI)

The BCT client already has a GUI that caters for the needs of VoD, Collaborative Drawing and Chat services. The GUI is organised in tabs that provide separate functionalities such as application configuration, video playback, drawing, chatting, debugging. This set will be augmented to present to the user content searching capabilities by means of keyword based searching and depiction of the obtained list of available content. There will be also a tree-based representation of content discovery to cater for hierarchically organised publication of content in case such a content storing schema is supported by the Vital++ Content indexing functions. Items discovered and represented on the GUI which may be interesting for the user to receive will be associated with rich GUI controls that will allow for actions regarding the reception of the specific content to be triggered through those GUI controls. An additional control will be added in the tabs of the client to allow for content publication. This will allow for definition of metadata (keywords) that will accompany the submission of the item to the Vital++ platform.

8.2.3.3 Content security measures

In case content is to be encrypted prior its uploading on the P2P network the client will provide a means for defining digital certificates to be used in this process. On the other hand, if mechanisms will be enforced to allow resolution of rights and policies towards other peers, those will be provided by plug-in modules in the P2P engine. There are two possibilities depending on the actual content security scheme that will be adopted in Vital++. The introduced mechanisms will be accommodated in the client as a separate module

- either with a specific API to cater for content encryption/decryption
- or embedded into the P2P and SIP engines to formulate a mechanism for preserving access rights and policies.

8.2.3.4 Content Management

At the user interaction layer, the client will be utilising a module that will be mapping discovered content or published content on protocol layer transactions. According to the specific instantiation of the client, this module will be triggering all the required actions so that the underlying layers initiate the appropriate procedures. However, the actions will be abstractly triggered through the specific provider module that will be responsible for mapping these on specific SIP and P2P procedures.

8.2.3.5 Content Discovery

This is a complementary functionality with the previous since it will be providing the "content" which is to be managed. Such a module will be implemented in an agnostic manner with respect to the underlying discovery protocol and procedures so that these are driven according to the specific instantiation details of the client.

Both content discovery and management functionality will be subject to the appropriate mapping on specific protocol layer functions. This mapping will be instantiated according to a provider module that will be integrating all the underlying functionalities (SIP and P2P) in a specific manner. The provider will be binding P2P control actions with SIP procedures and vice versa so that the P2P engine is controlled by information obtained and propagated through SIP/IMS messages and techniques.

9 Implementation plan

9.1 Monster Client

The implementation plan for the MONSTER client framework covers the necessary changes in order to comply with the generic client architecture and requirements. The implementation plan is split into three phases:

Phase I: Basic implementations regarding media processing, network protocols and GUI extensions.

Phase II: Implementation of higher functions, which require the presence of other VITAL++ architectural components, like DRM, QoS or content indexing (CI). The exact functional blocks depend on the corresponding availability of infrastructural components. Basically, at least one of the three shall be implemented in order to demonstrate true P2P-IMS cooperation.

Phase III: Finalization. In this phase, all missing higher functionalities shall be implemented.

A demonstrator with basic P2P media streaming functionalities, based on the AS-based overlay construction, is envisaged as a milestone for the end of phase I. After phase II, a P2P-IMS cooperation demonstration will be possible (2nd Milestone). The 3rd milestone is planted after phase III, showing a full VITAL++ client. The following figure illustrates the implementation plan.

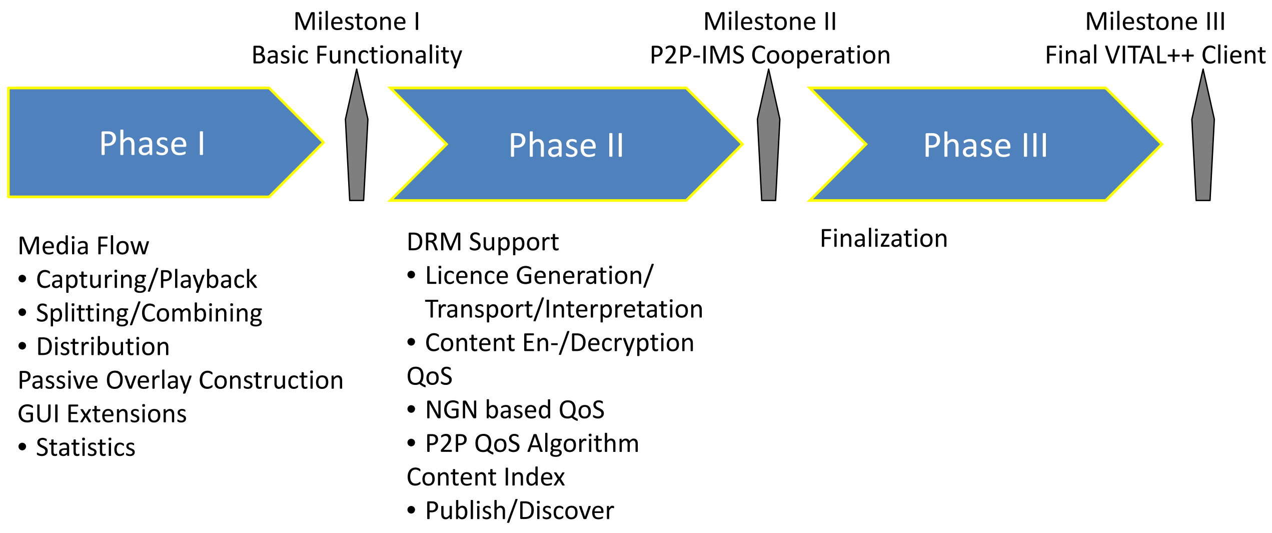

Figure 11: MONSTER P2P Implementation plan

Time plan:

- Milestone I: ready for June 17th 2009.

- Milestone II: ready for end of Sept. 2009.
- Milestone III: aligned with the deliverables 3.1 and 3.2 end of Feb. 2010.

9.2 BCT Client

The implementation plan for the BTC client is structured in four phases. In each phase we will develop a prototype that we will be able to demonstrate according to the timeline that we present at the end of this section.

- User Interface
  - Control
    - Communication abstraction (1st Prototype)
    - Content Searching
      - Logic (1st Prototype)
      - UI Element (1st Prototype)
    - Content Publication
      - Logic (2nd Prototype)
      - UI Element (2nd Prototype)
  - Media
    - Transmission abstraction (1st Prototype)
    - Acquisition abstraction (1st Prototype)
    - Media Capturing (3rd Prototype)
    - Media Playback (2nd Prototype)
    - Control Panel
      - UI Element (1st Prototype)
      - Logic (1st Prototype)
- Protocols
  - SIP engine (1st Prototype)
  - P2P engine
    - P2P Algorithms (1st Prototype)
    - DHT (4th Prototype)
    - Communication abstraction (1st Prototype)
    - Media buffering and exchange (1st Prototype)
  - File System engine
    - Uploading (4th Prototype)
    - Downloading (4th Prototype)
  - Security
    - DRM (3rd Prototype)
  - Service Providers
    - SIP Provider (for content indexing) (1st Prototype)

- Raw socket communication provider (for media exchange) (2nd Prototype)
- Mixed provider (incorporating parts of the SIP and Raw socket provider) (3rd Prototype)
- DRM or Security Provider (licenses, key exchange) (3rd Prototype)

- 1st Prototype: End of June 2009
- 2nd Prototype: Ready for the next project review (September 2009)
- 3rd Prototype: aligned with the deliverables 3.1 and 3.2 end of Feb. 2010
- 4th Prototype: aligned with the deliverables 3.1 and 3.2 end of Feb. 2010

10 Conclusions

In this deliverable we analyzed the functionalities that we have to implement in order to deliver P2P services such as: content distribution, live streaming and video on demand all of which will be managed with IMS.

The functionalities that we have described focus on two complementary objectives. The first is the extension and the optimization of the aforementioned P2P services through research into the mechanisms that they may use for scalable, distributed content and video diffusion. The second is the combination of these services with IMS mechanisms in order to exploit its centralised management, its authentication mechanisms, its charging mechanisms and the bandwidth provision from central servers. Through the exploitation of such mechanisms we aim to offer security functionalities, high and scalable performance and to minimise service interruption during dynamic network behaviour and peer arrivals and departures.

A major functionality towards this goal is the research, the design and the implementation of distributed scheduling mechanisms that are able to deliver the blocks of each object by avoiding the content bottleneck between two neighbours and meet the real time constraints of the video and audio streaming.

Additionally, it is vital to formulate and maintain a content diffusion overlay that is dynamically adapting to underlying network conditions, to peer arrivals and departures.

Also by embedding a DHT in the client we will be able to offload the responsible severs from object and key location and allow peers to contribute with their storage and their processing capabilities to object and key location.

Furthermore, by developing an authentication mechanism we will be in a position to maintain a system which only authorized peers will be able to enter and we can charge them for the participation in it. Distributed authentication will infuse scalability properties to our system and dynamic overlay reconfiguration for optimizing the system's performance.

The monitoring of the distribution of each object and the provision of bandwidth from centralized servers will make object distribution and streaming stable to the underlying network behaviour.

- End of document -