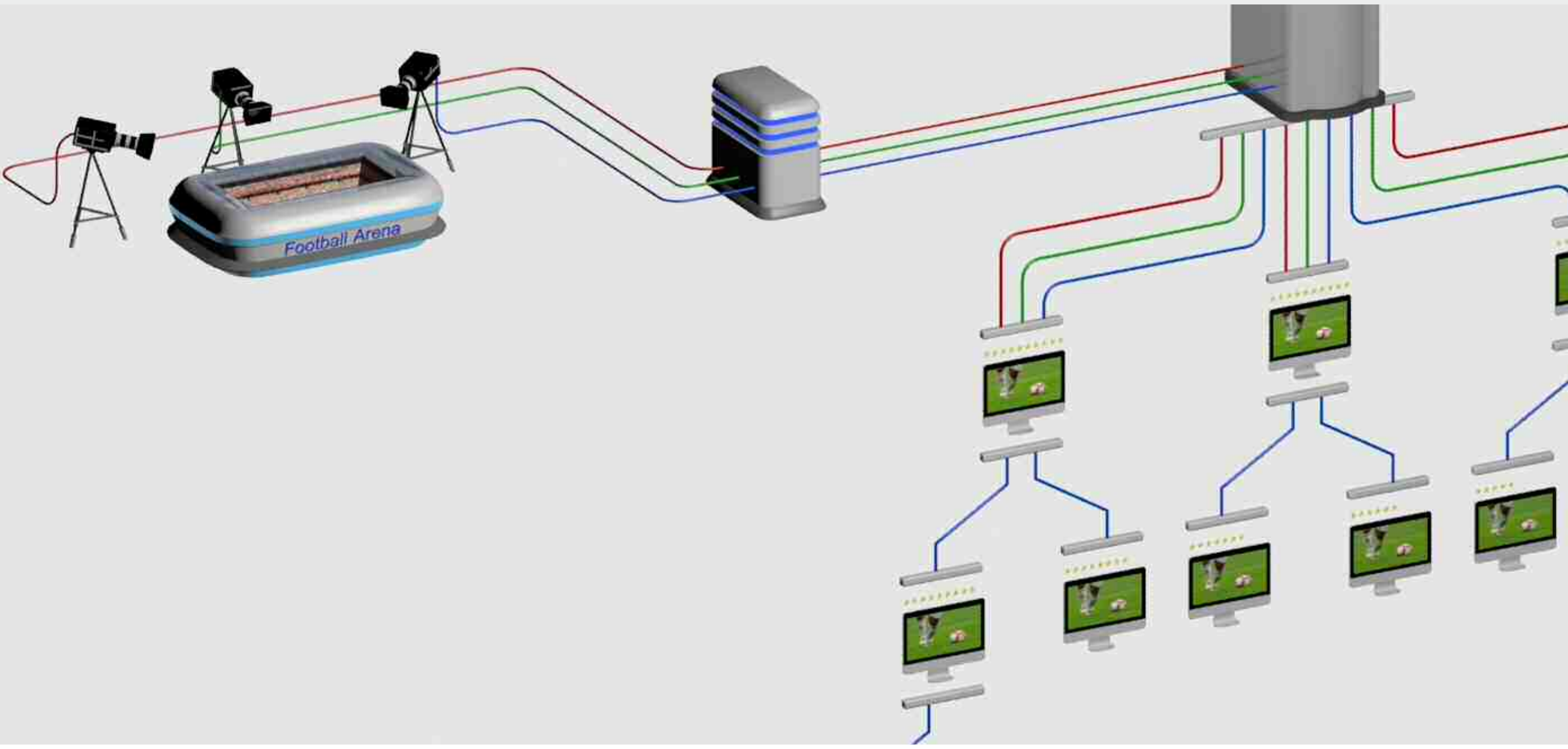
Instituto Politécnico
de Castelo Branco
Escola Superior
de Tecnologia

instituto de
telecomunicações

Multiple Multicast Trees for media distribution



Mestrado em Desenvolvimento de Software e Sistemas Interativos

Hélio Alexandre Dias da Silva

Orientadores
Professor Doutor Osvaldo Arede dos Santos
Professor Doutor Jonathan Rodriguez Gonzalez

Junho de 2014

Instituto Politécnico
de Castelo Branco
Escola Superior
de Tecnologia

instituto de
telecomunicações

Multiple Multicast Trees for media distribution

Hélio Alexandre Dias da Silva
heliosilva@av.it.pt

Orientadores
Professor Doutor Osvaldo Arede dos Santos
Professor Doutor Jonathan Rodriguez Gonzalez

Dissertação apresentada à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco e Instituto de Telecomunicações para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Desenvolvimento de Software e Sistemas Interativos, realizada sob a orientação científica do Professor adjunto Doutor Osvaldo Santos, do Instituto Politécnico de Castelo Branco.

Junho de 2014

Composição do júri

Presidente do júri

Professor Doutor Alexandre Jose Pereira Duro Fonte

Vogais

Professor Doutor Antonio Manuel de Jesus Pereira

Professor Coordenador do Departamento de Engenharia Informatica da Escola Superior de Tecnologia e Gestão de Leiria

Doutor João Manuel Leitão Pires Caldeira

Doutor da UTC de Informatica da Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

Professor Doutor Osvaldo Arede dos Santos

Professor Adjunto da UTC de Informatica da Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

Agradecimentos

Sem qualquer ordem de relevância descrevo o meu apreço e agradecimento a todos os que me rodearam na elaboração deste trabalho.

O trabalho aqui descrito não teria sido possível sem que uma equipa de pessoas do Instituto de Telecomunicações não acreditasse em mim. Um obrigado ao Professor Hugo Marques por me ensinar diversos valores como, rigor, calendarização, objetividade e finesse no detalhe do trabalho que produzimos pode fazer a diferença. Ao Doutor Jonathan Rodriguez por me ter dado a oportunidade de desenvolver algo único no mundo e ao nível do que grandes gigantes mundiais desenvolvem. Um obrigado ao grupo 4TELL - Instituto de Telecomunicações.

Gostava também de dedicar este trabalho ao Eng. Jorge Amaral (Fundador e Presidente da Mecalbi, Engineering Solutions, LDA) que me deu oportunidade de construir algo motivador e encorajador. Tem sido uma experiência de aprendizagem e desafios que dia após dia me motivam cada vez mais na perseguição de objetivos.

Agradeço também ao Professor Doutor Osvaldo Santos que me guiou e orientou incansavelmente na elaboração da tese.

À Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco agradeço a forma exemplar em como me trataram em todo este processo.

À minha companheira Patrícia que tem vindo a acompanhar-me ao longo de meses de trabalho e luta, esboço o meus sinceros agradecimentos.

Aos meus Pais, dedico-lhes este trabalho.

This work was supported by the ROMEO project (grant number: 287896), which was funded by the EC FP7 ICT collaborative research programme.

Resumo

Com a implementação massiva dos acessos de banda larga nos utilizadores, e com o aumento da capacidade dos dispositivos, o paradigma de rede P2P tem vindo a ganhar consistência e terreno comparativamente à típica tecnologia cliente-servidor. Na maioria dos países modernos as ligações à Internet têm capacidade suficiente para libertar as capacidades que o P2P pode oferecer em aplicações como, video-on-demand ou televisão em tempo real. É sabido que o uso de sistemas baseados em P2P para distribuir conteúdos sensíveis a atrasos pode levantar questões técnicas associadas à instabilidade do sistema causado pela entrada e saída de clientes. Neste relatório é proposto uma plataforma para distribuir conteúdos 3D, sensíveis a atrasos, utilizando um sistema híbrido cliente servidor e P2P hibrido. A plataforma proposta utiliza ao nível da aplicação P2P tecnologia do tipo múltiplas árvores na rede de acesso, delegando as típicas ações de servidor aos super-peers que estão distribuídos geograficamente. Esta proposta utiliza uma nova arquitetura de controlo para tirar proveito dos recursos da Internet para alimentar as técnicas de QoS rigorosas de forma escalável. Os resultados são baseados em testes efetuados em laboratório e mostram uma rápida reação nos clientes.

Palavras-Chave: Peer-to-peer, application-level multicast, end-system multicast, content distribution

Abstract

With the massive deployment of broadband access to the end-users and the improved hardware capabilities of end devices, peer-to-peer (P2P) networking paradigm is consistently gaining terrain over the typical client-server approach. In most of the modern countries, today's Internet connectivity has sufficient conditions to unleash P2P applications such as video-on-demand or real-time television. It is known that the use of P2P based systems to distribute delay sensitive applications raises technical problems mainly associated with the system's instability caused by the peer churn effect. In this report, we propose a framework to distribute delay sensitive 3D video content using a hybrid client-server and P2P approach. The proposed framework uses P2P application-level multicast trees at the access networks, delegating typical server operations at super-peers who are domain and geographically distributed. The approach uses a new control architecture to take advantage of the Internet to meet stringent QoS demands in a scalable manner. Results based on real testbed implementation show quick reaction at peer level.

Keywords: Peer-to-peer, application-level multicast, end-system multicast, content distribution

Contents

List of Figures

List of Tables

List of abbreviations

Abbreviation	Meaning
AAA	Authentication, Authorization and Accounting
ADSL	Asymmetric Digital Subscriber Line
BR	Border Router
CET	Central European Time
CPU	Central Processing Unit
DNS	Domain Name System
DVB	Digital Video Broadcasting
ER	Edge Router
GbE	Gigabit Ethernet
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISP	Internet Service Provider
JSON-RPC	Javascript Object Notation Remote Procedure Call
UDP	User Datagram Protocol
IEEE	Institute of Electrical and Electronics Engineers
LAN	Local Area Network
LTE	Long Term Evolution
MDC	Multiple Description Coding
MTM	Multicast Tree Management
P2P	Peer-to-Peer
NMS	Network Monitoring Subsystem
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
SLA	Service-Level Agreement
TB	Topology Builder
TC	Topology Controller
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator

1 Introduction

In recent years, speed for Internet access has experienced enormous upgrades (AKAMAI, 2013) both in wired and wireless links. This is especially true in the majority of the cities of modern countries, where fibre-to-the-home (Research and Markets, 2014*b*) and Long Term Evolution (LTE) (Nakamura, 2010) access is becoming a reality. In addition, computers and mobile devices have also become faster and lighter, which in fact empowered them for a new type of use the visualization of social networking media, also known as user-generated content(Mislove et al., 2007).

According to (Alexa, 2013), Facebook and Youtube are the two top contributors for this kind of content distribution. In fact, according to (Cisco, 2011-2016) Internet video from sites such as YouTube (short-form), Hulu (long-form), Netflix (video-to-TV), online video purchases and rentals, webcam viewing, and web-based video monitoring (excluding P2P video file downloads) are expected to have a compound annual growth rate (CAGR) of 34% until 2016. Video mobile data is predicted to have a CAGR of 75% in the period 2012-2017 according to (Cisco, 2012 to 2017). Netflix, by itself attracts more than 23 million subscribers in the North America, streaming high-definition quality video with an average bitrate reaching almost 4Mbps. According to (Sandvine, 2011) Netflix is the single largest source of Internet traffic in the US, consuming 29.7% of peak downstream traffic.

Current developments in 3D technology have triggered an increasing user interest in experiencing this technology(Research and Markets, 2014*a*) however the distribution of 3D media to a large amount of users raises significant technical challenges (Theodore Zahariadis, 2008) mainly due to bandwidth and delay restrictions associated with the multiple views of content.

Thus it is widely accepted that consumers will expect more features from their viewing experience. First, they want on-demand services so that they can watch the contents when they wish. Second, they want to watch content anytime, anywhere, and regardless of the device type. It could be wide screen display in a living room, a navigation screen in a car, or an handheld device such as smart phone or tablet. Third, high definition content has already gained popularity in most of European countries, and even ultra-high definition quality is expected to attract more people considering that the resolution of some tablets and laptops (e.g. iPad) is already far better than the high definition.

The trends of on-demand, mobile, and ultra-high definition quality impose formidable challenges for the delivery network of the future. As of today, the majority of video content distributed in the Internet uses the typical client-server paradigm. NetFlix is a clear example of such an approach, however the scalability of this solution, even though proved to work in the current Internet, raises server-side bandwidth and resource concerns for the distribution of high-bandwidth content. A good example of such content is related to the increasing user interest in experiencing 3D technology.

The streaming of 3D media can be considered as the transmission of multiple streams from multiple viewpoints of the same content. Typically 3D content is composed by a stereoscopic view (Tam, 2006) (base layer composed by left and right views) that can be overlaid with multiple (other) views, including layers for improving the quality of the stereoscopic view - hence 3D content is a bandwidth eager application. To make things harder, multicast technology, which allows a single stream to serve multiple users, is not supported by most Internet Service Providers (ISPs)(Diot et al., 2000), making the distribution of large amount of data to a large number of end-users still prohibitive in today's Internet. On the other hand, the typical understanding of client-server and peer-to-peer (P2P) networks is that they replace rather than complement each other. With this in mind, this work proposes a multiple multicast tree distribution mechanism that merges the advantages of both client-server and P2P approaches: centralized decision, administration and content typical in a client-server paradigm - with the split/balancing of network resources found on P2P networks

Distributing 3D video content to a large number of users in the Internet considerably raises the technical challenge, mainly due to bandwidth demands of the multiple video streams associated with the 3D video content - different views with different depths and different levels of quality may exist. This article proposes a framework, including the system architecture and its components, for the distribution of high bandwidth consuming 3D content using the Internet. The proposed system merges the concept of P2P application-level multicast trees with IP multicast trees and is designed to operate amongst different Internet Service Provider (ISP) domains.

1.1 Goals and Contribution

We propose a framework to distribute delay sensitive 3D video content using a hybrid client-server and P2P approach. The proposed framework uses P2P application-level multicast trees at the access networks, delegating typical server operations at super-peers who are domain and geographically distributed. The framework, as seen in Figure 1, has two different applications, one that acts has a server, controlling all the aspects related with the multiple multicast trees and bandwidth, and the other acts has a client, controlling all the data related with video and the correct relaying of information to other clients. Also, we pretend to evaluate how the system behaves while using the multiple multicast trees regarding network metrics like packet loss, delay and jitter.

This report presents a number of new ideas. We can summarize them as follows:

1. Efficient and intelligent multiple multicast tree management

2. Resilient parent and child relation for media interchange

3. Scalable solution

4. P2P application for media distribution

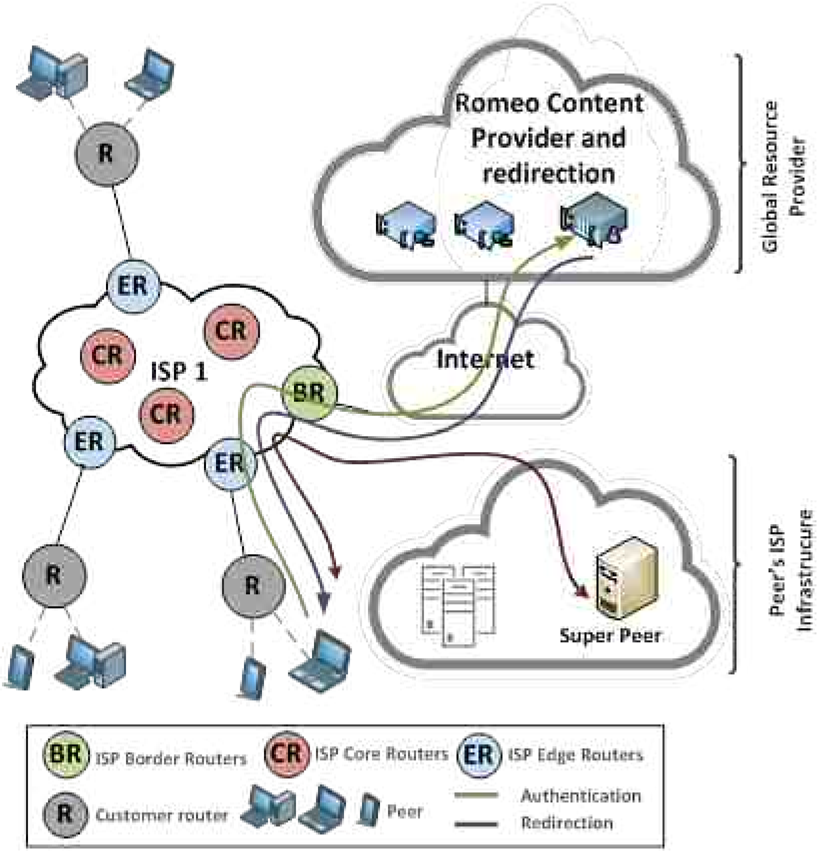


Figure 1: Simplified diagram - Super Peer at the nearest ISP and the Authentication resources

1.2 Schedule

The schedule of this report is depicted in Figure 2. The base architecture needed 6 months of development, in which several applications were developed at the server side and at the peer side. The work started by developing an multicast tree simulator that generated multiple tree based on the number of the peers present and their capabilities - the simulator is out of the scope of this report. After the simulator the real implementation started, the focus of the development was the topology builder. Topology builder is the core of this report so a special attention will be given to it's internal functionalities. After the topology builder the client's application was developed in parallel with message trade-off mechanism. Several other features were also developed to support the work presented in this report.

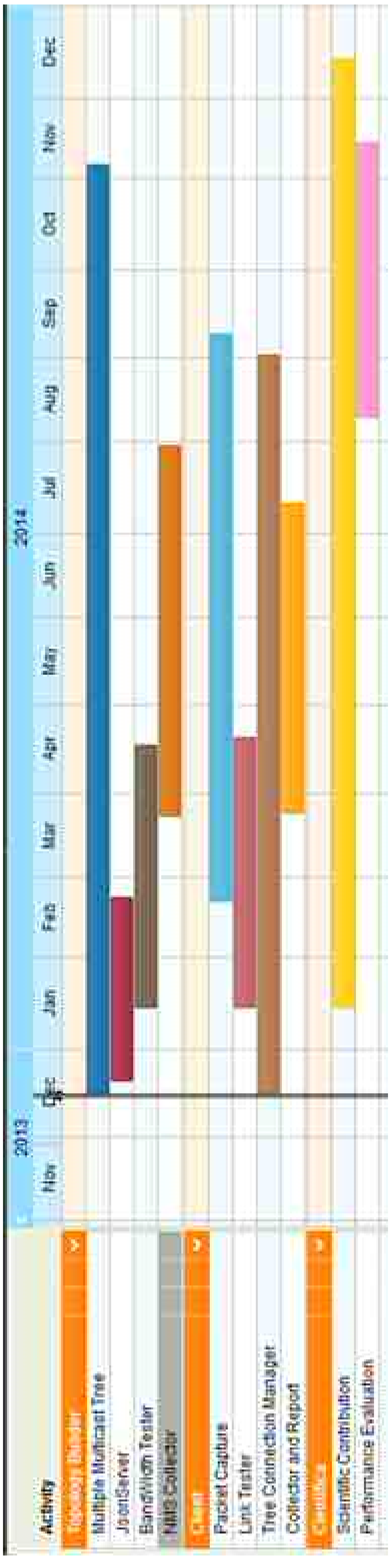


Figure 2: report schedule

1.3 Dissertation structure

This report is organized as follows: Chapter 2 presents the state of the art of similar application level multicast solutions and tree construction algorithms. Chapter 3 presents the architecture which has been developed and its internal functionalities. Chapter 4 describes how the development of the architecture was achieved, Chapter 5 presents several results of the most critical modules of this work. Finally, in Chapter 6, conclusions are drawn and it also discusses the level of achievement of the research goals. The references can be found at the end of the document.

2 State of the art

2.1 Related Work

DONet

DONet (also known as CoolStreaming) (X. Zhang and Yum, 2005), proposes a P2P based data-driven overlay network for efficient live media streaming; video is divided to segments of uniform length and a buffer map is used to identify each video segment and to indicate if it is available - each node continuously exchanges its buffer map with partners where a special node, called deputy node, is responsible for providing the list of partners to new (joining) nodes. DONet also proposes a scheduling algorithm that calculates the number of potential suppliers for each video segment; basically the algorithm starts from those with only one supplier and so forth. If there are multiple suppliers, then the algorithm starts by selecting the one with highest bandwidth and enough available time. According to the authors the average distance from origin node to a destination node is bounded by O(logN), where 95% of nodes can be reached within 6 hops.

An Internet-based DONet implementation, called CoolStreaming v.0.9, was released on May 30, 2004. An unstructured P2P network (called iGridMedia) for interactive applications (such as online auction, person interview or video sharing) using a push-pull approach is proposed in (M. Zhang and Yang, 2008).

iGridMedia

iGridMedia aims to provide delay-guaranteed P2P live streaming service over the Internet - safeguarding the ISPs have dedicated servers to support the delay guaranteed interactive applications. For overlay construction, joining nodes must first contact a rendezvous point which is a server maintaining a partial list of current online nodes - then each node randomly finds other 15 nodes to establish a partnership. For the streaming delivery, in the pull mechanism, the video streaming is packetized into fixed length packets called streaming packets marked by sequence numbers. Each node periodically sends buffer map packets to notify all its neighbors what streaming packets it has in the buffer and then explicitly requests its absent packets from neighbors. Once a packet fails to be pulled, it will be requested again. In the push mechanism, iGridMedia evenly partitions the stream into 16 sub streams, and each sub stream is composed of the packets whose sequence numbers are congruent to the same value modulo 16. Once a packet in one sub stream is successfully pulled from a peer, the remaining packets in this sub stream will be relayed directly from this peer. When a neighbor quits or packet loss occurs, the pull mechanism is started again. iGridMedia is fully implemented in C++.

CoopNet

CoopNet (V. N. Padmanabhan and Sripanidkulchai, 2002) is a mechanism for distributing streaming media content using P2P cooperative networking; it uses a centralized approach where a central server is responsible to determine the path of distribution, indicating joining peers to which parent they should connect - the peer hierarchy is decided based on each peer available bandwidth (reported upon connection to the server periodically afterwards) and their proximity (based on IP/BGPP prefix). CoopNet also employs Multiple Description Coding (MDC) to address the interruptions caused by the frequent joining and leaving of individual peers.

SplitStream

SplitStream (M. Castro and Singh, 2003) is also a multicast mechanism for distributing content in P2P cooperative environments, but contrary to CoopNet, it uses a distributed approach; there is no central server and all nodes have the same responsibility - new nodes try to find a parent and join directly to the tree; trees are constructed in a distributed fashion using each peer's upload and download bandwidth. In the distribution of content SplitStream divides data in multiple stripes and distributes each stripe per each tree the source makes stripe selection and multicast each per designated tree. References (Apostolopoulos, 2001) and (D. Andersen and Morris, 2001) represent additional seminal work related with resilience to node failures in multipath distribution networks.

2.2 Summary

In single-tree systems, whole content is distributed over a single-formed tree. In these systems, leaf nodes of the tree are consumers without uploading any content to the other peers, which may be considered as an unfair job division. Besides, in case of a non-leaf peer churn, children of this peer become totally disconnected from the content delivery structure. On the other hand, use of multiple multicast trees may eliminate the unfairness issue by inserting the leaf nodes of one tree as intermediate peer in other trees. Resilience to the peer churn is also improved in multiple multicast tree systems by delivering the data redundantly over different paths. The above mentioned platforms use the concept of decentralized management, which in our case, is an unwanted feature because of the intelligent multiple multicast trees management algorithm, this algorithm knows the capabilities of the connected peers and acts accordingly they're available resources. By having a more fine grained system that evaluates how powerful a peer is in terms of capacity, multicast trees can be calculated having in mind the available resources, this gives the opportunity to exploit more resources and to calculate a backup solution in case of any failure. In this report we introduce a new metric named evaluation that is represents how powerful a resource is inside the multicast tree.

3 Architecture

3.1 3D Video Content Assumptions

For the 3D video content, we assume multiple cameras arrangement and multiple description coding (MDC) (A. Mohr, 2000), (Apostolopoulos, 2011), (J. G. Apostolopoulos, 2001) of the resulting streaming media, a concept illustrated in Figure 3. The example considers the case in which a user is consuming the content in a 3D capable monitor. Given Internet bandwidth constrains and for scalability reasons, the combined video information from the cameras is encoded in multiple IP video streams so that the video can be reconstituted from a subset of these streams, being the video quality proportional to the number of streams received. Each description is sent to the peer as a different IP stream, such that if packets of the corresponding description are lost, then the associated packet in the other description is used to reconstruct the video frame with a slightly downgraded quality.

There are different MDC and streaming techniques in the literature, all of which come at the expense of varying levels of added redundancy, computational complexity and reconstruction performance, this article however will not focus on such discussion. Figure 3. 3D video capture scenario and a subset of possible video streams. The depicted scenario illustrates the case for a user consuming the stereoscopic view with the capability of depth adjustment and fast viewpoint change.

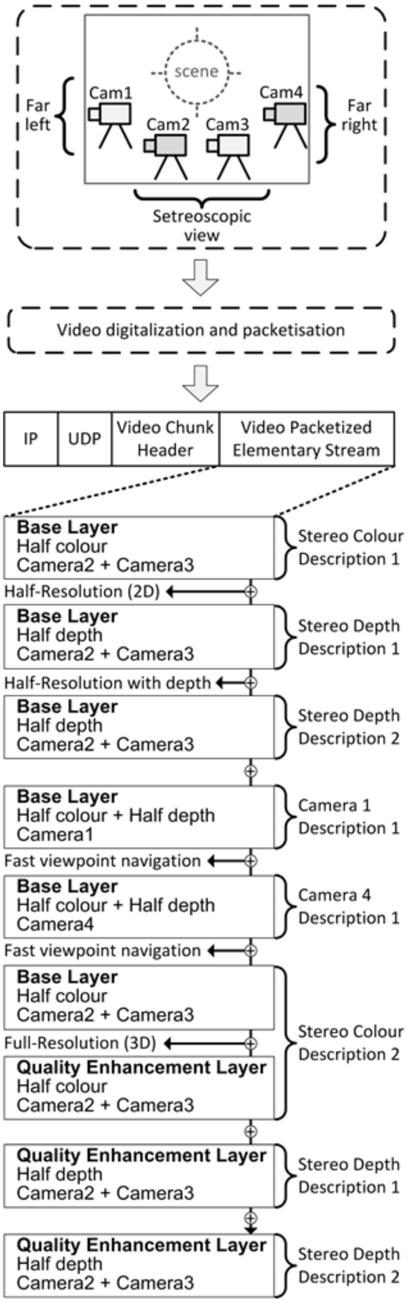


Figure 3: 3D video capture scenario and a subset of possible video streams. The depicted scenario illustrates the case for a user consuming the stereoscopic view with the capability of depth adjustment and fast viewpoint change.

From the example given in Figure 3 and considering an average of 4Mbps rate for each stream (*MUSCADE MUltimedia SCAlable 3D for Europe*, 2014),(*Remote Collaborative Real-Time Multimedia Experience over the Future Internet*, 2014), we can determine that each user would require a maximum of 36Mbps connection to the server, just to consume the stereoscopic view with depth adjustment and fast viewpoint navigation capability (either towards Camera 1 or Camera 4). If the user moves to a viewpoint between Camera 1 and Camera 2+3, then a total of 12 streams will be needed, resulting in a total consumed bandwidth of 48 Mbps. In case the user uses a multi-view display with multiple view capability (e.g. 8 views at the same time), the server would need to stream 15 views, consuming a total of 60Mbps per user, see Table 1. As it can easily be understood, when considering a client-server approach in today's Internet, these numbers can rapidly consume the server's bandwidth. A possible solution to this problem would be the use of IP multicast transmission towards the clients, but after many years of experimentation, IP multicast is not currently a ubiquitous service (Diot et al., 2000) on the public Internet, being mostly deployed on private/corporate networks. The main reasons behind this lack of support are related to interdomain routing issues, lack of standardized congestion control mechanisms for multicast traffic and inherent multicast security issues, which are essential if multicast applications are to be safely deployed. Given the problem and applicability statement, it is thus useful to research novel approaches that combine the features of both the client-server approach and IP multicast distribution, hopefully retaining the desirable properties of each. The next section will describe our proposed framework for 3D content distribution, that uses a client-server approach and multicast distribution at the core of the Internet Service Provider (ISP) and P2P networking principles to achieve multiple application-level multicast distribution trees at the customer networks.

Table 1: Complete list of 3D video streams needed for a 4 camera setup

#	Content Type	Description	Camera
1	Base Layer Half color + Half depth	1	Camera 1
2	Base Layer Half color + Half depth	2	
3	Q. Enhancement Half color + Half depth	1	
4	Q. Enhancement Half color + Half depth	2	
5	Base Layer Half color	1	Camera 2 and Camera 3
6	Base Layer Half color	2	
7	Base Layer Half depth	1	
8	Base Layer Half depth	2	
9	Q. Enhancement Layer Half color	2	
10	Q. Enhancement Layer Half depth	1	
11	Q. Enhancement Layer Half depth	2	
12	Base Layer Half color + Half depth	1	Camera 4
13	Base Layer Half colour + Half depth	2	
14	Q. Enhancement Half colour + Half depth	1	
15	Q. Enhancement Half colour + Half depth	2	

3.2 System Overview

The proposed system uses the concept of main server, super-peers, peers and ISP core network. The main server is where the content is stored, it belongs to a specific DNS domain (typically associated with the service brand) and it is property of the service owner.

The super-peers are also property of the service owner, act as proxies/replicas of the main server and are placed in the premises of every ISP that has an agreement with the service owner. The super-peers are responsible to serve peers from a specific geographical area or ISP. The modules which are responsible for overlay management functionality deal with the construction and maintenance of the multiple multicast tree structure for the P2P network and take into account the geographical optimization methods to improve the overall system performance.

Geographical optimization is the grouping of peers by geographic location, the described solution takes into consideration the following statements (Figure 4 represents the explained concept, although it's not the real representation how ISP are distributed in Portugal, it shows the concept behind the grouping mechanism):

- If the client is successfully authenticated, the Server redirects the peers to the closest Super-peer, based on the requested content and the client's (global) IP address;
- IP addresses are assigned to Regional Internet Registries (RIPE NCC in Europe) and then to local Internet Registries (ISPs). This information is publicly available and allows to determine the geographic location of any IP enabled device.

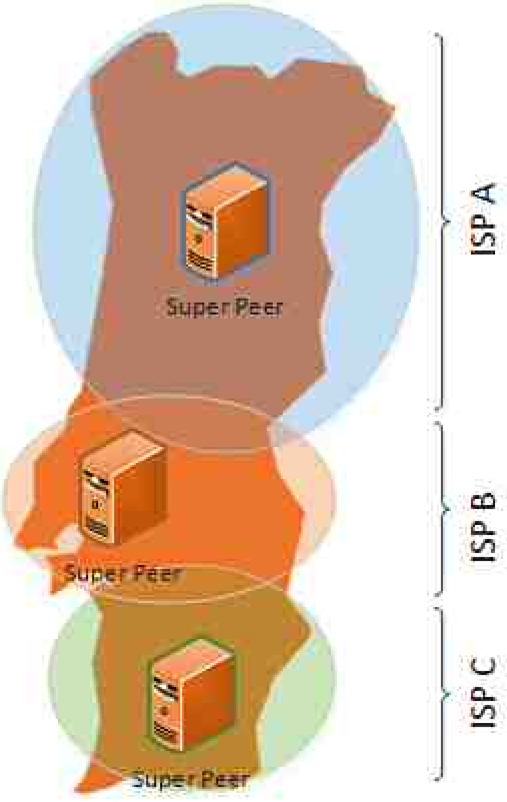


Figure 4: ISP segmentation - Peers will have their own Super Peer based on the geographic location and ISP

Peers connected to other ISPs have their own set of trees, being the root of the tree the Border Router of the system. When we have ISPs without a system like the one we are describing, in such a case, peers will be inserted on the multiple multicast trees according only to their ranking and requested content. Their location inside the ISP will not be used, thus this means a pure P2P overlay will be constructed for all peers in that ISP.

The peers are the end-users who will consume the content and the ISP core network is responsible to safeguard the Quality of Service (QoS) at the core of its network and also between the main server and the super-peers through the proper implementation of Service Level Agreements (SLAs).

3.2.1 Description

The content is distributed from the main server to the geographically distributed requesting super-peers through QoS enabled channels. Peers are located at the access network and will receive content directly from their nearest super-peer. At ISP level, for proof-of-concept purposes, we consider a modular point-of-presence topology, where the ISP services are located in a dedicated Services Network this is the place where the super-peers should be housed. In such topology, border routers (BR) connect the ISP to other ISPs, core routers (CR) provide internal ISP high speed trunk connections and the edge routers (ER) - also known as distribution or access routers - are high port density routers connecting customers (peers) with the ISP core network. Concept illustrated in Figure 5.

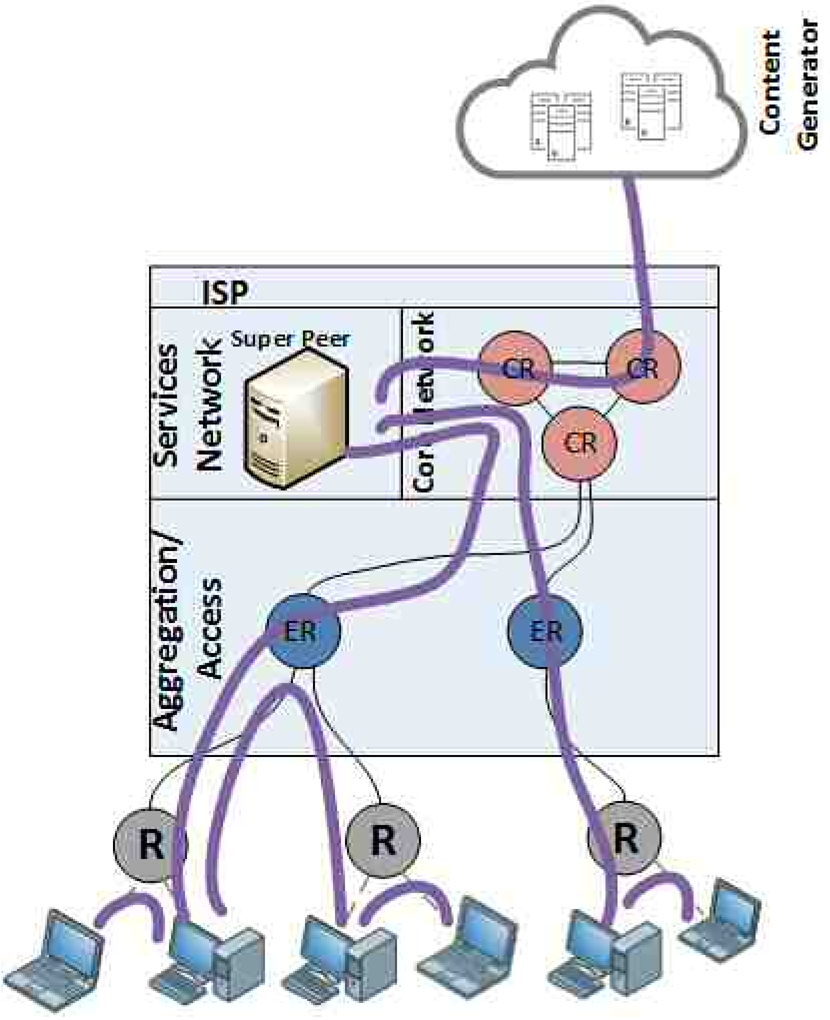


Figure 5: Simple ISP network

As in any ISP topology, customers also have their own routers, which are not managed by the ISP. This concept is illustrated in Figure 5. Subscribers of the service, called peers, will connect to the main server for authentication, authorization and accounting purposes. Upon success, the server will redirect the peers to their nearest super-peer - this decision takes in consideration the peer's geographic location and ISP (based on IP address information). Peers will then use the provided super-peer address to request specific content. For each new peer requesting content the super-peer will compute the peer's position in an application-level multicast tree, effectively distributing the content via a P2P network. Peers can either assume the role of a parent, a child/parent or a child. Parents receive the content directly from the super-peer and occupy the highest level on the P2P multicast tree. Child/parents are the peers that receive the content from another peer and also feed other peers they occupy intermediate levels on the multicast tree.

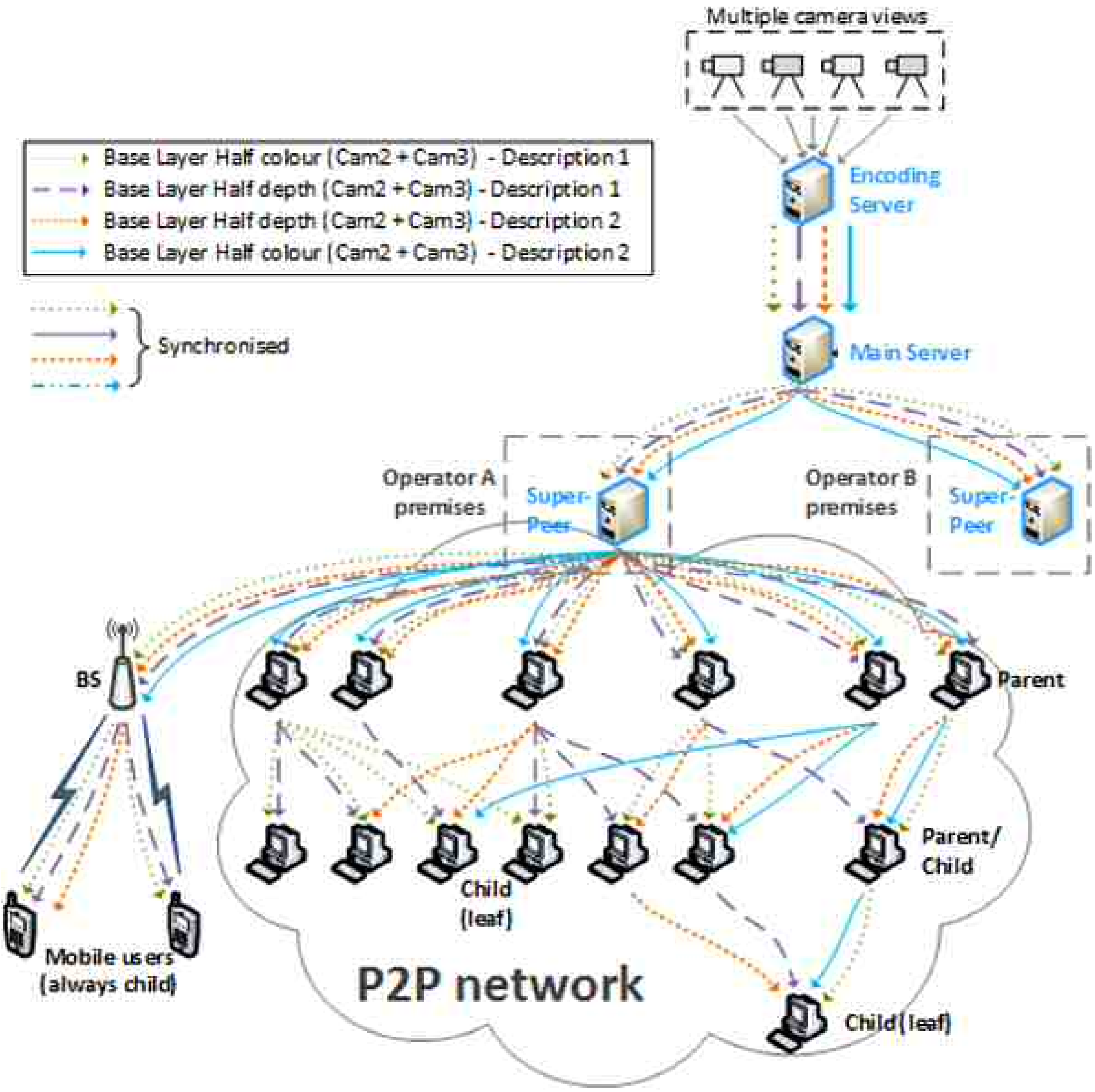

Multiple camera views
Base Layer Half colour (Cam2 + Cam3) - Description 1
Base Layer Half depth (Cam2 + Cam3) - Description 1
Base Layer Half depth (Cam2 + Cam3) - Description 2
Base Layer Half colour (Cam2 + Cam3) - Description 2
Synchronised
Encoding Server
Main Server
Operator A premises
Super-Peer
Operator B premises
Super-Peer
BS
Parent
Parent/ Child
Child (leaf)
Mobile users (always child)
P2P network
Child (leaf)

Figure 6: Content distribution using P2P concept and application-level multicast trees. Peers with better resources will occupy a higher position on the tree.

A child is a peer that receives content from other peers and does not feed other peers - they occupy the lowest level on the P2P multicast tree and can be considered leafs (mobile users are a clear example of leafs, since they have considerable restrictions in their download/upload bitrates and limited battery capacity). In a specific tree, a peer can be fertile or sterile. A fertile peer can forward chunks to children peers whereas a sterile peer only receives chunks. In order for the overlay to be feasible, every peer has to contribute using its uploading bandwidth. Therefore, a peer is fertile in some trees and sterile in the others.

In single-tree systems, whole content is distributed over a single-formed tree. In these systems, leaf nodes of the tree are consumers without uploading any content to the other peers, which may be considered as an unfair job division. Besides, in case of a non-leaf peer churn, children of this peer become totally disconnected from the content delivery structure. On the other hand, use of multiple multicast trees may eliminate the unfairness issue by inserting the leaf nodes of one tree as intermediate peer in other trees. Resilience to the peer churn is also improved in multiple multicast tree systems by delivering the data redundantly over different paths. In each tree, a peer has a single source peer called parent and a set of destination peers called children. E.G.: The peer receives video chunks from its parent peer and forwards the received chunks to its children peers. In this way, video chunks are disseminated to all the peers in the tree.

Figure 6 provides an high-level overview of this concept. In order to improve resiliency and redundancy, the video streams mentioned in Table 1, are transmitted in specific multicast trees. The streams that complement each other (to improve visualization quality) are transmitted in the same multicast tree. On the other hand, the streams that provide redundancy to other streams are transmitted in a different multicast tree. Table 2 shows the mapping between ROMEO (*Remote Collaborative Real-Time Multimedia Experience over the Future Internet*, 2014) [1] video streams and the correspondent multicast tree.

Table 2: Relation between the 3D video streams and their associated multicast distribution tree

Tree ID	Content Type	Description
0	Base Layer Half color (Cam2+Cam3)	1
1	Base Layer Half color (Cam2+Cam3) Q. Enhancement Layer Half color (Cam2+Cam3)	2
2	Base Layer Half depth (Cam2+Cam3) Q. Enhancement Layer Half depth (Cam2+Cam3)	1
3	Base Layer Half depth (Cam2+Cam3) Q. Enhancement Layer Half depth (Cam2+Cam3)	2
4	Base Layer Half color + Half depth (Cam1) Q. Enhancement Layer Half +color Half depth (Cam1)	1
5	Base Layer Half color + Half depth (Cam1) Q. Enhancement Layer Half color+Half depth (Cam1)	2
6	Base Layer Half colour + Half depth (Cam4) Q. Enhancement Layer Half color+Half depth (Cam4)	1
7	Base Layer Half color+Half depth (Cam4) Q. Enhancement Layer Half color+Half depth (Cam4)	2

[1] ROMEO: Remote Collaborative Real-Time Multimedia Experience over the Future Internet - ROMEO will bring new 3D media (3D multi-view video and spatial audio) to European citizens at home as well as on the move. In order to deliver the high bandwidth high quality 3D media to mobile and fixed users with guaranteed minimum Quality of Experience for all users, ROMEO will combine the DVB technology with the peer to peer (P2P) Internet technology. In ROMEO, the broadcaster can deliver high quality stereoscopic 3D content, to the users and at the same time stream a set of supplementary 3D multi-view content (e.g. additional viewpoints with their respective spatial audio), through a master peer to the other peers on the tree. The peers can also acquire relevant information about the broadcast 3D media through other peers or other points on the Internet.

The end-point peers will also serve mobile devices with wireless access in a dedicated network, which can access the adapted and post-processed content. Peers located at the network edges will also perform various adaptations, by deploying users' equipment virtualization layer to eliminate the need for receiver set-top boxes and complicated equipment set-up, speeding up the user take up of cost effective high quality 3D media consumption. You can find more information on the following URL: http://www.ict-romeo.eu/

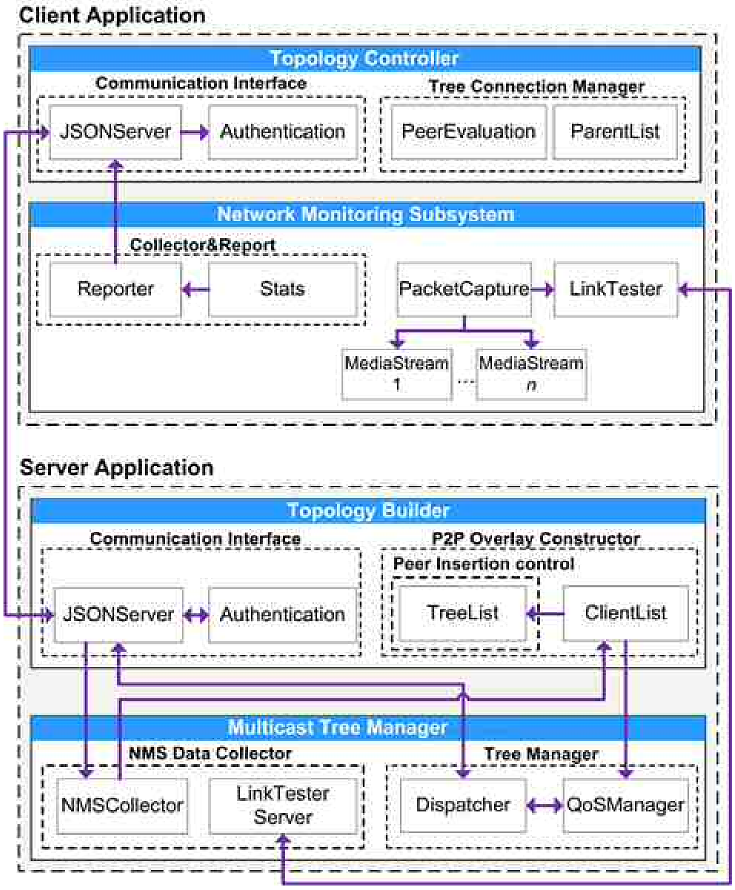


Figure 7: Client and Server software modules, submodules and their relations

3.2.2 Super-Peer

The main server is responsible for the user authentication, authorization and accounting (AAA) services and, the peerID computation - a value that uniquely identifies each peer joining the system. The main server stores (or has access to) all possible contents to be distributed. In small production environments the role of the main server can be fused with the super-peer.

The super-peer is responsible to serve peers at a specific geographical area or ISP domain. At the roof of all trees there is a server that has the role of the root peer (Super Peer). The server is responsible for the structure of the trees and the data flow towards the peers. The Super Peer maintains the structure of the trees (H. Silva, 2012), monitors the overlay, and distributes chunks to the trees. In each tree, a number of peers are directly connected to the Super Peer.

The super-peer can operate either in a reactive or proactive manner. If a reactive behavior is used, the super-peer does not store new content unless specifically requested by a serving peer. If, on the other hand, a proactive behavior is used, the super-peer will use non-peak hours to store new content (e.g. based on content request statistics provided by the main server). Proactive behavior has the advantage of providing peers with lower response time and lighter operation on network peak times. The disadvantage would be the need for higher storage capacity. To perform its functions the main server/super-peer has two main modules, a Topology Builder and a Multicast Tree Manager.

The Topology Builder (TB) is a software module that performs the following functions at the main server/super-peer:

- listens for new peer connection requests;
- acts as an authentication proxy (authenticator) for user authentication with the main server;
- computes the peerID, a unique identification that identifies each peer;
- creates multiple P2P application-level multicast trees for content distribution;
- computes peer insertion on the P2P trees; When a peer is redirected to a super-peer, it is the responsibility of the TB to compute the peer position at the P2P multicast tree at access network level. The steps in the computation are:
  - (i) to group peers according to the requested content, see Table 1;
  - (ii) group peers according to their common ER - geographical aggregation;
  - (iii) sort peers by evaluation, a metric explained in section 3.2.4, as depicted in Figure 8.

After grouping and sorting operations the multiple P2P multicast trees are computed, one per each requested content and edge router (Depicted at Figure 8). It will be the ER's responsibility to map each requested content multicast address to specific parent(s) IP addresses(s) the ER will effectively act as a replicator.

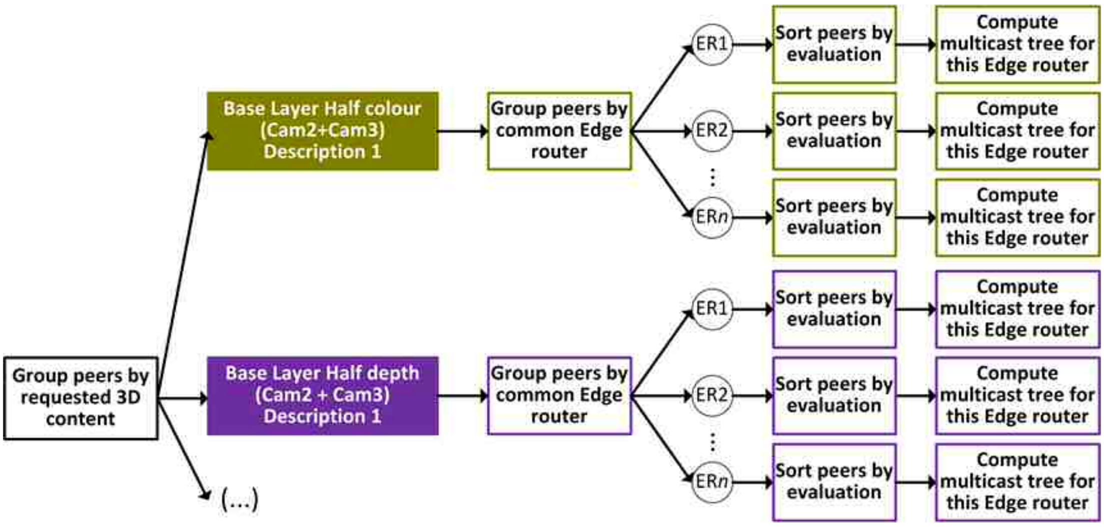

Group peers by requested 3D content
Base Layer Half colour (Cam2+Cam3) Description 1
Base Layer Half depth (Cam2 + Cam3) Description 1
(...)
Group peers by common Edge router
Group peers by common Edge router
ER1
ER2
ERn
ER1
ER2
ERn
Sort peers by evaluation
Compute multicast tree for this Edge router

Figure 8: Content and location aware construction of P2P application-level multicast distribution trees

To optimize the Edge Router resources, the super-peer predetermines how many top-level peers (parents) can be directly fed by one Edge Router. This means that, when constructing each P2P tree, the super-peer positions a predetermined number of highest ranking peers at the top of the tree and delegates in these the distribution of content to other peers in the same access network. Every time a peer is selected to forward content, its resources are diminished, and if its evaluation becomes lower than other peers, the new highest ranking peer will take the role of parent for additional content streaming. If a peer has insufficient network resources, it will not receive some of the streams. To minimize issues associated with peers joining and leaving the system, also known as churn, the TB uses the following mechanisms:

- Grounding: new joining peers are always inserted at the bottom of the P2P tree. The algorithm then suffers periodically updates (every t seconds) to maximize the efficiency of the P2P tree promoting and demoting peers.
- Graceful leaving: whenever possible, peers always inform the TB about their imminent disconnection, and only stop forwarding to their children when instructed by the TB or upon a timeout.
- Redundancy: when inserted on a tree, all peers will be informed of their active parent and a backup parent. If the active parent is not reachable within a timeout the peer switches to the backup parent and informs the TB. This behavior is further explained in section 3.2.4.

This entire approach brings the following advantages:

- Resiliency: by using tree separation, a major fault in a specific zone of the network will not affect other zones;
- Scalability: by grouping peers by common ER, tree depth is significantly reduced, since peers sharing the same access network have improved downstream and upstream bandwidth, which allows more children per parent;
- Performance/Quality: the total number of hops between top level parents and their children is significantly lower, which contributes to reduce the packet/chunk delay, jitter and loss. Recovery from minor faults (such as peer churn) can also be achieved in a faster way, since the backup parent is on the same access network.

These operations are achieved by specific TB's submodules, as depicted bellow (also depicted in Figure 7):

- **Authentication**: is responsible to exchange authentication related messages with the AAA services running at the main server.
- **ClientList**: is responsible for maintaining an updated list of all connected peers. If the peer disconnects (graceful or ungraceful), this list is updated in order to remove the peer from the tree.
- **TreeList**: uses the P2P tree construction and maintenance algorithm to compute the multiple P2P application-level multicast trees. As explained in chapter 3.2.4, peers are responsible to populate a monitoring database at the Multicast Tree Manager, the algorithm uses these updated peer data to compute each peer position on the tree.

- **JSONServer**: is responsible for sending and receiving messages to/from the server application submodules.
- The Multicast Tree Manager (MTM) is the second software module running at the main server/super-peer. The MTM is intrinsically related with the TB operations and has the following functions:
  - It collects/aggregates network monitoring data (percentage of packet loss, average delay, jitter and available bandwidth), from all connected peers, providing the TB with updated peer's network conditions;
  - It allows peers to perform bandwidth tests with the super-peer (or a replica).
  - It informs the ISP QoS mechanisms, on the endpoints of IP multicast trees at the ISP core network (between the super-peer and the ERs serving the peers). The MTM performs its functions using the following submodules, also depicted in Figure 7
- **NMSCollector**: collects network monitoring data periodically sent by the Network Monitoring Subsystem running at every connected peer, as explained in chapter 3.2.4. This information is also shared with the TB for quick P2P tree maintenance operations.
- **LinkTesterServer**: allows authorized peers to perform bandwidth tests. For the download test it sends a predefined fixed size binary file, for the upload test it expects to receive the exact same file. The results based on this transfer are then used on a composite metric, equation (1), to compute their evaluation. The bandwidth test should be performed at the first time a peer connects (new PeerID) and upon super-peer request (for troubleshooting reasons).
- **Dispatcher**: is used to interface with the TB's JSONServer whenever messages need to be sent or received by the MTM submodules

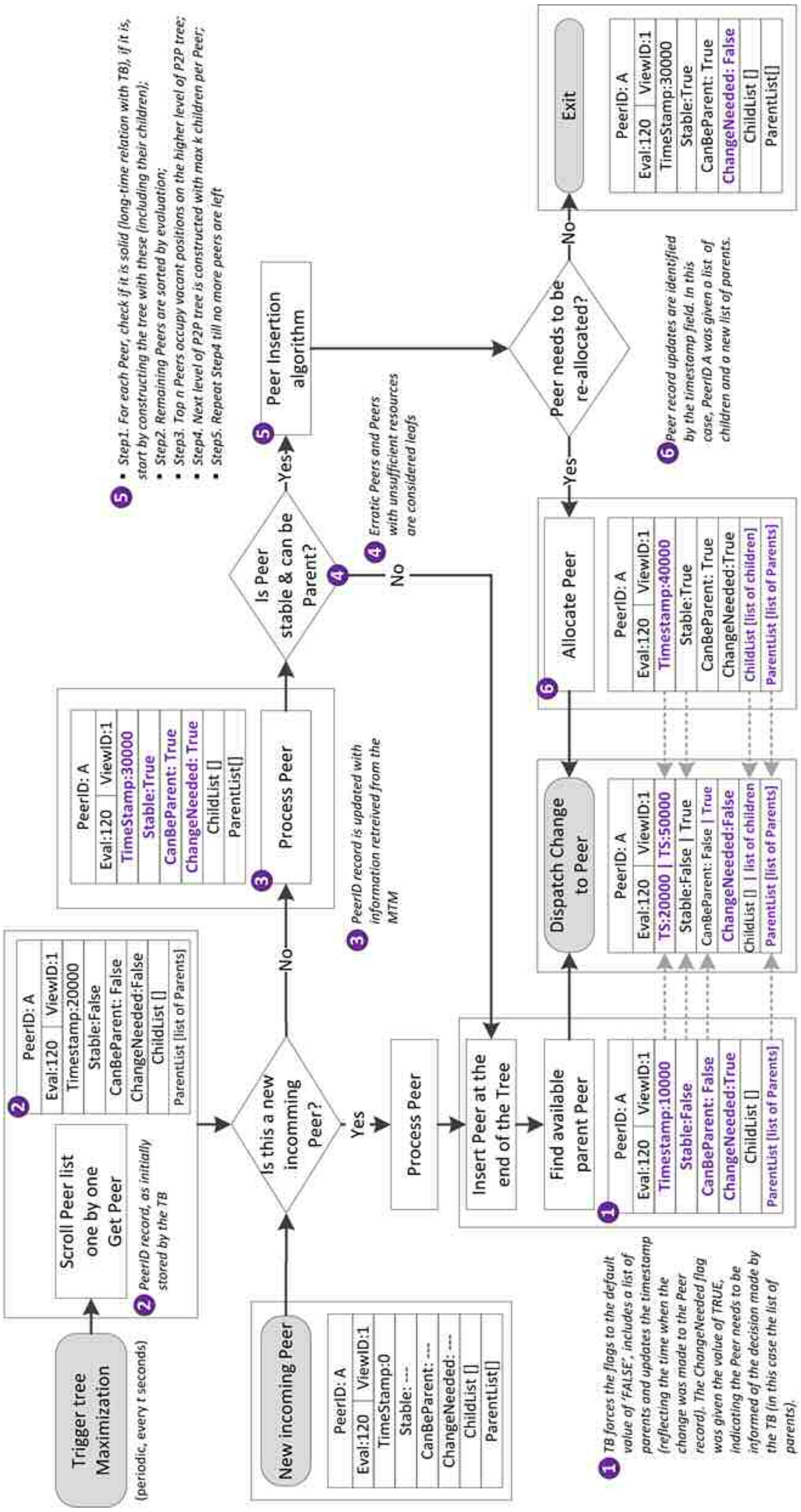
3.2.3 Multiple Multicast Trees

Regarding the tree construction and maintenance it is decided to use a tree based topology structure instead of a mesh based topology structure. The reason for this choice is that data paths are deterministic in tree based systems, which is not the case for mesh based-systems. Deterministic paths provide more predictable behavior compared to mesh based systems in terms of both jitter and latency, which is a very important point to meet the strict synchronization requirements of the project. Besides, for the purpose of load balancing, and fault tolerance, multiple multicast trees will be used instead of a single tree structure. The centralized topology management approach will be used, which provides system wide delay variation and synchronization control mechanisms. As an improvement to this centralized approach, it is decided to have geographically distributed, high performance super peers in each geographical area, these super peers will be the root of the multiple multicast trees for this area.

When a new peer wants to join the P2P network, it will first connect to the main server, the server will then direct it to the nearest super peer and the super peer will insert this new peer to the available multicast trees. For small deployments, the main server and the super peer can be considered to be the same machine; however for large deployments the main server will be unique with several distributed super peers connected to it.

Figure 9 provides a flowchart description of the P2P tree computation procedures. When a new peer joins the system (peer A in this case), the TB parses the peer request to identify the peer evaluation and the requested content, amongst other data. It then creates a peer record and inserts the peer at the bottom of the tree by allocating a parent and a backup parent. It also updates the timestamp associated with this operation and flags the dispatcher that further actions are needed for this peer. The dispatcher then sends a message to the peer informing on its parent and backup parent addresses and further analysis the peer record to identify if this peer can be a parent and if it is stable (a concept explained in section 3.2.4). After all operations are completed the ChangeNeeded flag is updated.

To maximize the P2P tree efficiency, the algorithm periodically re-constructs each tree using each peer stored record. If changes are needed (e.g., peers needs to be re-allocated) this information is once more signaled to the dispatcher and the process is repeated.



1 TB forces the flags to the default value of 'FALSE', includes a list of parents and updates the timestamp (reflecting the time when the change was made to the Peer record). The ChangeNeeded flag was given the value of TRUE, indicating the Peer needs to be informed of the decision made by the TB (in this case the list of parents).

2 PeerID record, as initially stored by the TB

3 PeerID record is updated with information retrieved from the MTM

4 Erratic Peers and Peers with unsufficient resources are considered leafs

5
- Step1. For each Peer, check if it is solid (long-time relation with TB), if it is, start by constructing the tree with these (including their children);
- Step2. Remaining Peers are sorted by evaluation;
- Step3. Top n Peers occupy vacant positions on the higher level of P2P tree;
- Step4. Next level of P2P tree is constructed with max k children per Peer;
- Step5. Repeat Step4 till no more peers are left

6 Peer record updates are identified by the timestamp field. In this case, PeerID A was given a list of children and a new list of parents.

Figure 9: Peer insertion procedure at the Topology Builder running at the super-peer

3.2.4 Peer

The peer system component represents the equipment used by the end-users to consume the content. The peer can be a fixed computer, a laptop or a smaller mobile device. In order to comply with this framework, two modules need to be installed at the peer: a Topology Controller and a Network Monitoring Subsystem.

The Topology Controller (TC) is a software module that runs at the peer with the following purposes:

- Initial contact with the main server for user authentication and redirection to the nearest super-peer;
- Compute the peer evaluation using peer's hardware characteristics and network statistics, as provided by the Network Monitoring Subsystem;
- Inform the TB of its intention to consume specific 3D content;
- Perform P2P tree operations as commanded by the TB (parent, parent/child or child);
- Establish connections with parents for content request and accept connections from children peers for content forwarding.

The peer evaluation is a value that takes in consideration the peer's hardware (memory and CPU), the peer's network capabilities (upload and download throughput) and the peer's stability.

Peer evaluation is calculated according to (1), and indicates how valuable a peer is in the P2P distribution system.

$$Evaluation = K_1(U) + K_2(D) + K_3(M) + K_4(C) + K_5(S) \tag{1}$$

Where, **K1** to **K5** are weights that allow fine-tuning the metric;
U and **D** correspond to the upload and download bitrates (in kbps) respectively;
M represents the peer random access memory (in gigabyte);
C indicates the number of CPU cores and;
S represents the peer's stability as described in section 3.2.4.

In order to implement these features, TC uses the following submodules, as depicted in Figure 7:

- **Authentication**: this submodule is responsible for the user authentication, authorization and accounting interactions with the main server;
- **JSONServer**: is responsible for sending and receiving messages to/from the client application modules.
- **PeerEvaluation**: computes the peer evaluation as described in **??**;
- **ParentList**: contains this peer's list of parents and backup parents (for each content type) as indicated by the TB.

The Network Monitoring Subsystem (NMS) is the second module running at the peer side. It has the following functions:

- Collects peer hardware and software characteristics;
- Collects network traffic statistics (packet loss, average delay, jitter, available bandwidth) for each received stream;
- Periodically reports the collected data to its parent (it chooses a different parent in each iteration using a round-robin approach) or to the MTM (in case this is a top level peer). Reports can also be triggered by a request from the MTM or when changes in the peer's network conditions cross a specific threshold.
- Computes the peer stability, a metric that reflects the stability potential of this peer based on previous sessions;
- Informs the MTM about detected changes in the network, such as parent disconnection;

Table 3: Structure for the NMS data report

Field	Description	Size
ID	Message ID (identifies this is a report)	4
PeerID	The unique peer ID as given by the TB	32
LocalIP	The local IP address of the Peer (IPv4/v6)	16
NetMask	The local IP subnet mask	4
DL	The download capability in (Kbps)	4
UL	The upload capability (Kbps)	4
nChildren	The total number of children of this peer	4
nCPUcores	The number of CPU cores and its type	4
TotalMem	The size of RAM memory in the peer (MB)	4
FreeMem	The size of available memory(MB)	4
ConsMem	The size of consumed memory(MB)	4
OS	The operating system identification	var
Delay	The average packet delay (ţs)	4
Jitter	The delay jitter (ţs)	4
PacketLoss	The packet loss (in %, content specific)	4

Table3 shows the structure for the periodic NMS report. To save resources and simplify socket management at the receivers, reports are sent to one of the parents, which then collects all the received reports during a time-window and sends all collected reports to its own parent (one by one in a sticky TCP connection). This process goes on until the data gets to the highest peers in the P2P tree hierarchy, which then send the bundle of all collected reports to the MTM (NMSCollector submodule). This procedure is illustrated in Figure 10. For the stability computation, the NMS performs the calculation as depicted in the following equation:

$$stability(S) = \sum_{(i=1)}^{c} t_i - \left(\frac{\sqrt{\sum_{(i=1)}^{c} (t_i - t)^2}}{c-1} \right) \tag{2}$$

Where, **c** refers to a predefined number of connections for which the duration has to be memorized, **t_i** represents the connected time for each connection and t is the mean value for the duration of the c connections. In case of insufficient information (e.g. new peer joining the system), the NMS uses a predefined default value for stability.

The NMS is also responsible to detect, report, and possibly solve peer connectivity problems. If a major failure occurs in stream reception, the NMS first contacts the NMS of the parent responsible to stream that specific content and if it is reachable, it is up to that parent to solve the problem. If the parent wouldn't solve the problem in a pre-specified time-window, or cannot be contacted, the TC is notified in order to immediately switch to the backup parent and inform the MTM.

This set of procedures is illustrated in Figure 11. NMS features are implemented through the following submodules, as depicted in Figure 7:

- **PacketCapture**: this submodule passively collects network data such as, connection history (start time, duration) or traffic statistics (packet loss, delay, jitter), by using the libpcap library.

- **LinkTester**: provides link testing functions with the MTM to determine link characteristics such as the download and upload link capacity.

- **Stats**: is responsible for collecting hardware and software data as depicted in 3 and for computing the statistics associated with the network data collected by the PacketCapture submodule;

- **Reporter**: formats the information collected by the Stats submodule according to a report template. It is also responsible to send to the TC's JSONServer the report to the selected peer parent. If this peer is a parent, this submodule is also responsible to collect NMS reports form all of its children and to send the report bundle to the selected parent in the hierarchy;

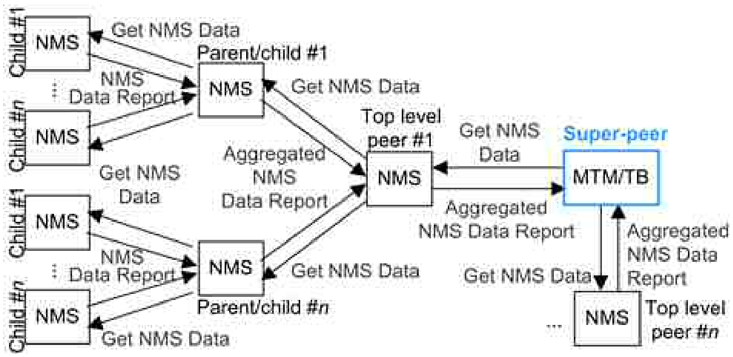


Figure 10: Procedure for collecting network monitoring statistics

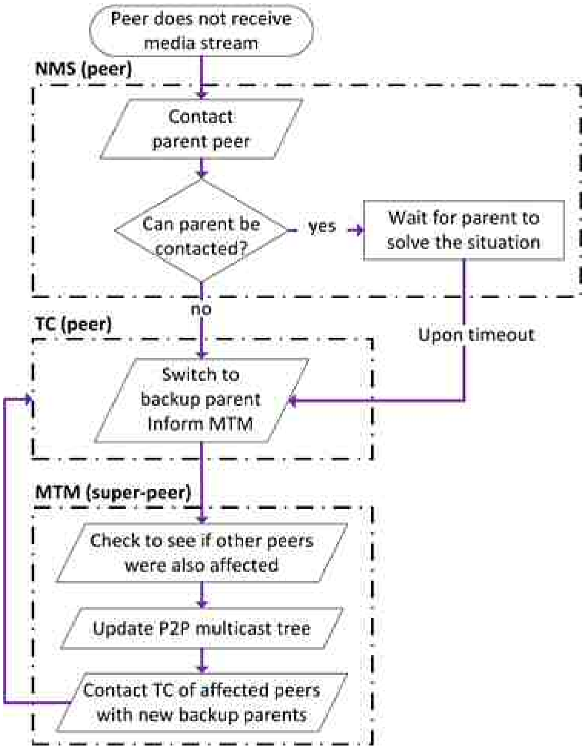

Peer does not receive media stream
NMS (peer)
Contact parent peer
Can parent be contacted?
yes
Wait for parent to solve the situation
no
Upon timeout
TC (peer)
Switch to backup parent Inform MTM
MTM (super-peer)
Check to see if other peers were also affected
Update P2P multicast tree
Contact TC of affected peers with new backup parents

Figure 11: Set of procedures performed by a peer upon detecting major failure in the media stream delivery

3.3 Architectural advantages over regular P2P networks

For scalability reasons overlay solution creates trees per each access network. Peers connected to other ISPs have their own set of trees, being the root of the tree the Border Router of the service enabled ISP connecting to the non-enabled service ISP. In such a case, and since the service enabled ISP has no information on the non-enabled service ISP network topology, the peers will be inserted on the multiple multicast tress according only to their ranking and requested content. Their location inside the ISP will not be used. This means a pure P2P overlay will be constructed for all peers on the non-enabled service ISP. This concept is depicted in Figure 12, you can clearly see that the non enabled service ISP will have a complicated network overlay, that will not take advantage of concepts explained in this report, one of the streams (RED) needs 5 hops to get into the final destination (Core Network and SLAs are not part of this report, they are represented only for better understanding and visualization).

This approach provides the following advantages:

Resiliency: by using tree separation, a major fault in a specific zone of the network will not affect other zones;

Scalability: by grouping peers by common ER, tree depth is significantly reduced, since peers sharing the same access network have improved downstream and upstream bandwidth, which allows more children per parent;

Performance/Quality: the total number of layer 3 hops between top level parents and their children is significantly lower, which contributes to reduction in the packet/chunk delay, jitter and loss. Recovery from minor faults (such as peer churn) can also be achieved in a faster way, since the backup parent is on the same access network as the peer.

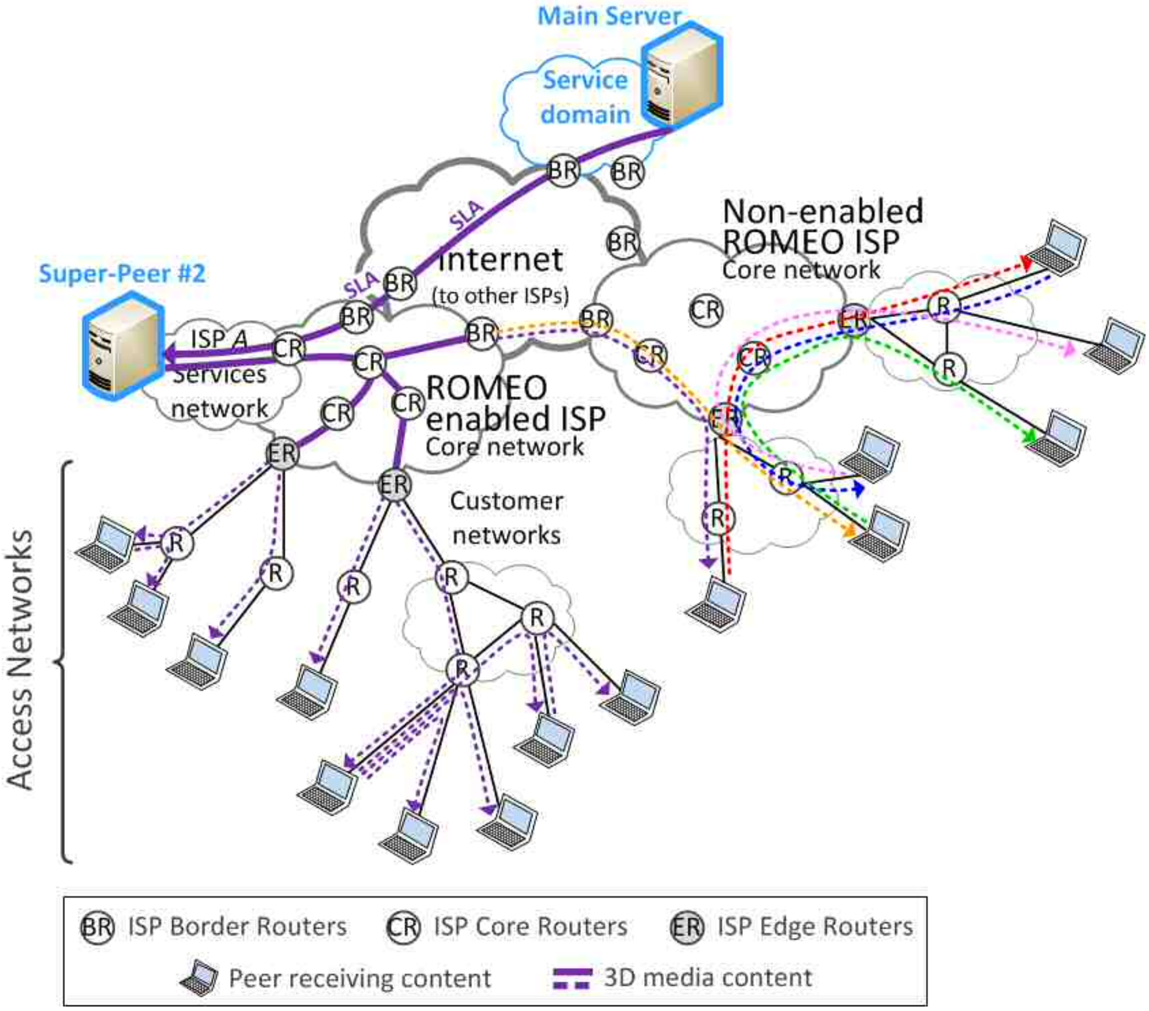


Figure 12: Inter-ISP scenario

4 Implementation and Development

4.1 Developed solution and requirements

Requirements

- Operating System Requirement: Ubuntu 12.04
- Packages/Library: lipcaputils, openssl and libcurl. In addition the scons application is also needed to install and compile the following C++ packages: JsonCpp and JsonRpc-Cpp. These modules use the C++ build essential packages.

Topology Builder and MTM software modules:

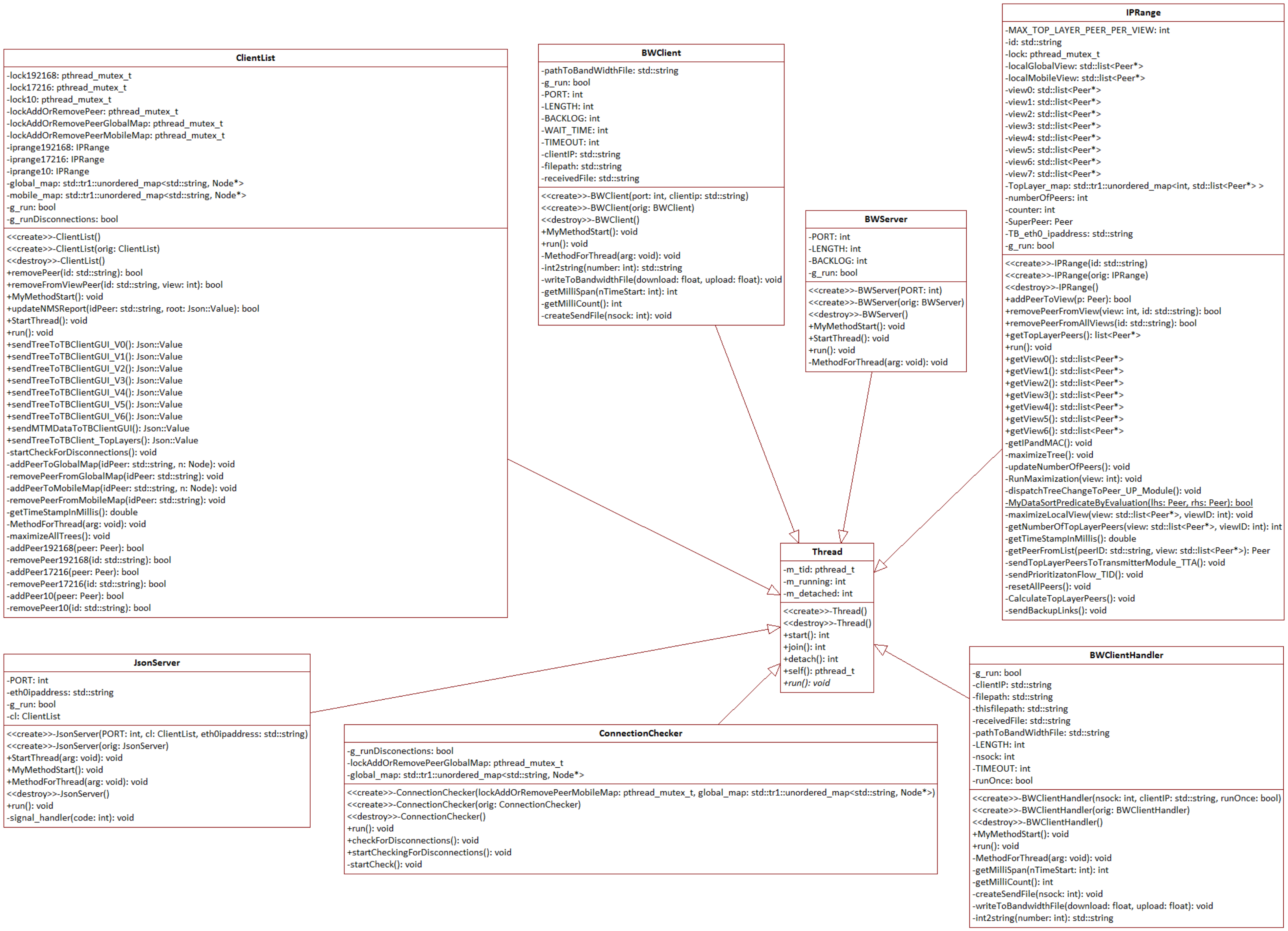
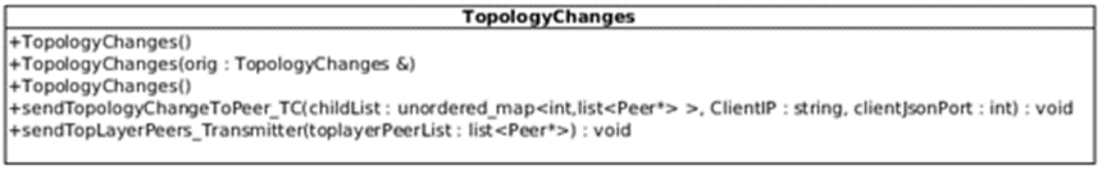

IPRange
-MAX_TOP_LAYER_PEER_PER_VIEW: int
-id: std::string
-lock: pthread_mutex_t
-localGlobalView: std::list<Peer*>
-localMobileView: std::list<Peer*>
-view0: std::list<Peer*>
-view1: std::list<Peer*>
-view2: std::list<Peer*>
-view3: std::list<Peer*>
-view4: std::list<Peer*>
-view5: std::list<Peer*>
-view6: std::list<Peer*>
-view7: std::list<Peer*>
-TopLayer_map: std::tr1::unordered_map<int, std::list<Peer*> >
-numberOfPeers: int
-counter: int
-SuperPeer: Peer
-TB_eth0_ipaddress: std::string
-g_run: bool
<<create>>-IPRange(id: std::string)
<<create>>-IPRange(orig: IPRange)
<<destroy>>-IPRange()
+addPeerToView(p: Peer): bool
+removePeerFromView(view: int, id: std::string): bool
+removePeerFromAllViews(id: std::string): bool
+getTopLayerPeers(): list<Peer*>
+run(): void
+getView0(): std::list<Peer*>
+getView1(): std::list<Peer*>
+getView2(): std::list<Peer*>
+getView3(): std::list<Peer*>
+getView4(): std::list<Peer*>
+getView5(): std::list<Peer*>
+getView6(): std::list<Peer*>
-getIPandMAC(): void
-maximizeTree(): void
-updateNumberOfPeers(): void
-RunMaximization(view: int): void
-dispatchTreeChangeToPeer_UP_Module(): void
-MyDataSortPredicateByEvaluation(lhs: Peer, rhs: Peer): bool
-maximizeLocalView(view: std::list<Peer*>, viewID: int): void
-getNumberOfTopLayerPeers(view: std::list<Peer*>, viewID: int): int
-getTimeStampInMillis(): double
-getPeerFromList(peerID: std::string, view: std::list<Peer*>): Peer
-sendTopLayerPeersToTransmitterModule_TTA(): void
-sendPrioritizatonFlow_TID(): void
-resetAllPeers(): void
-CalculateTopLayerPeers(): void
-sendBackupLinks(): void
BWClient
-pathToBandWidthFile: std::string
-g_run: bool
-PORT: int
-LENGTH: int
-BACKLOG: int
-WAIT_TIME: int
-TIMEOUT: int
-clientIP: std::string
-filepath: std::string
-receivedFile: std::string
<<create>>-BWClient(port: int, clientip: std::string)
<<create>>-BWClient(orig: BWClient)
<<destroy>>-BWClient()
+MyMethodStart(): void
+run(): void
-MethodForThread(arg: void): void
-int2string(number: int): std::string
-writeToBandwidthFile(download: float, upload: float): void
-getMilliSpan(nTimeStart: int): int
-getMilliCount(): int
-createSendFile(nsock: int): void
BWServer
-PORT: int
-LENGTH: int
-BACKLOG: int
-g_run: bool
<<create>>-BWServer(PORT: int)
<<create>>-BWServer(orig: BWServer)
<<destroy>>-BWServer()
+MyMethodStart(): void
+StartThread(): void
+run(): void
-MethodForThread(arg: void): void
ClientList
-lock192168: pthread_mutex_t
-lock17216: pthread_mutex_t
-lock10: pthread_mutex_t
-lockAddOrRemovePeer: pthread_mutex_t
-lockAddOrRemovePeerGlobalMap: pthread_mutex_t
-lockAddOrRemovePeerMobileMap: pthread_mutex_t
-iprange192168: IPRange
-iprange17216: IPRange
-iprange10: IPRange
-global_map: std::tr1::unordered_map<std::string, Node*>
-mobile_map: std::tr1::unordered_map<std::string, Node*>
-g_run: bool
-g_runDisconnections: bool
<<create>>-ClientList()
<<create>>-ClientList(orig: ClientList)
<<destroy>>-ClientList()
+removePeer(id: std::string): bool
+removeFromViewPeer(id: std::string, view: int): bool
+MyMethodStart(): void
+updateNMSReport(idPeer: std::string, root: Json::Value): bool
+StartThread(): void
+run(): void
+sendTreeToTBClientGUI_V0(): Json::Value
+sendTreeToTBClientGUI_V1(): Json::Value
+sendTreeToTBClientGUI_V2(): Json::Value
+sendTreeToTBClientGUI_V3(): Json::Value
+sendTreeToTBClientGUI_V4(): Json::Value
+sendTreeToTBClientGUI_V5(): Json::Value
+sendTreeToTBClientGUI_V6(): Json::Value
+sendMTMDataToTBClientGUI(): Json::Value
+sendTreeToTBClient_TopLayers(): Json::Value
-startCheckForDisconnections(): void
-addPeerToGlobalMap(idPeer: std::string, n: Node): void
-removePeerFromGlobalMap(idPeer: std::string): void
-addPeerToMobileMap(idPeer: std::string, n: Node): void
-removePeerFromMobileMap(idPeer: std::string): void
-getTimeStampInMillis(): double
-MethodForThread(arg: void): void
-maximizeAllTrees(): void
-addPeer192168(peer: Peer): bool
-removePeer192168(id: std::string): bool
-addPeer17216(peer: Peer): bool
-removePeer17216(id: std::string): bool
-addPeer10(peer: Peer): bool
-removePeer10(id: std::string): bool
Thread
-m_tid: pthread_t
-m_running: int
-m_detached: int
<<create>>-Thread()
<<destroy>>-Thread()
+start(): int
+join(): int
+detach(): int
+self(): pthread_t
+run(): void
BWClientHandler
-g_run: bool
-clientIP: std::string
-filepath: std::string
-thisfilepath: std::string
-receivedFile: std::string
-pathToBandWidthFile: std::string
-LENGTH: int
-nsock: int
-TIMEOUT: int
-runOnce: bool
<<create>>-BWClientHandler(nsock: int, clientIP: std::string, runOnce: bool)
<<create>>-BWClientHandler(orig: BWClientHandler)
<<destroy>>-BWClientHandler()
+MyMethodStart(): void
+run(): void
-MethodForThread(arg: void): void
-getMilliSpan(nTimeStart: int): int
-getMilliCount(): int
-createSendFile(nsock: int): void
-writeToBandwidthFile(download: float, upload: float): void
-int2string(number: int): std::string
JsonServer
-PORT: int
-eth0ipaddress: std::string
-g_run: bool
-cl: ClientList
<<create>>-JsonServer(PORT: int, cl: ClientList, eth0ipaddress: std::string)
<<create>>-JsonServer(orig: JsonServer)
+StartThread(arg: void): void
+MyMethodStart(): void
+MethodForThread(arg: void): void
<<destroy>>-JsonServer()
+run(): void
-signal_handler(code: int): void
ConnectionChecker
-g_runDisconections: bool
-lockAddOrRemovePeerGlobalMap: pthread_mutex_t
-global_map: std::tr1::unordered_map<std::string, Node*>
<<create>>-ConnectionChecker(lockAddOrRemovePeerMobileMap: pthread_mutex_t, global_map: std::tr1::unordered_map<std::string, Node*>)
<<create>>-ConnectionChecker(orig: ConnectionChecker)
<<destroy>>-ConnectionChecker()
+run(): void
+checkForDisconnections(): void
+startCheckingForDisconnections(): void
-startCheck(): void

Figure 13: Topology Builder simplified class diagram

TopologyChanges class

The TopologyChanges class is responsible to dispatch tree updates to all connected peers. This class is also responsible to send the top-layer peers to the P2P Transmitter module. These class functions are listed in Figure 14

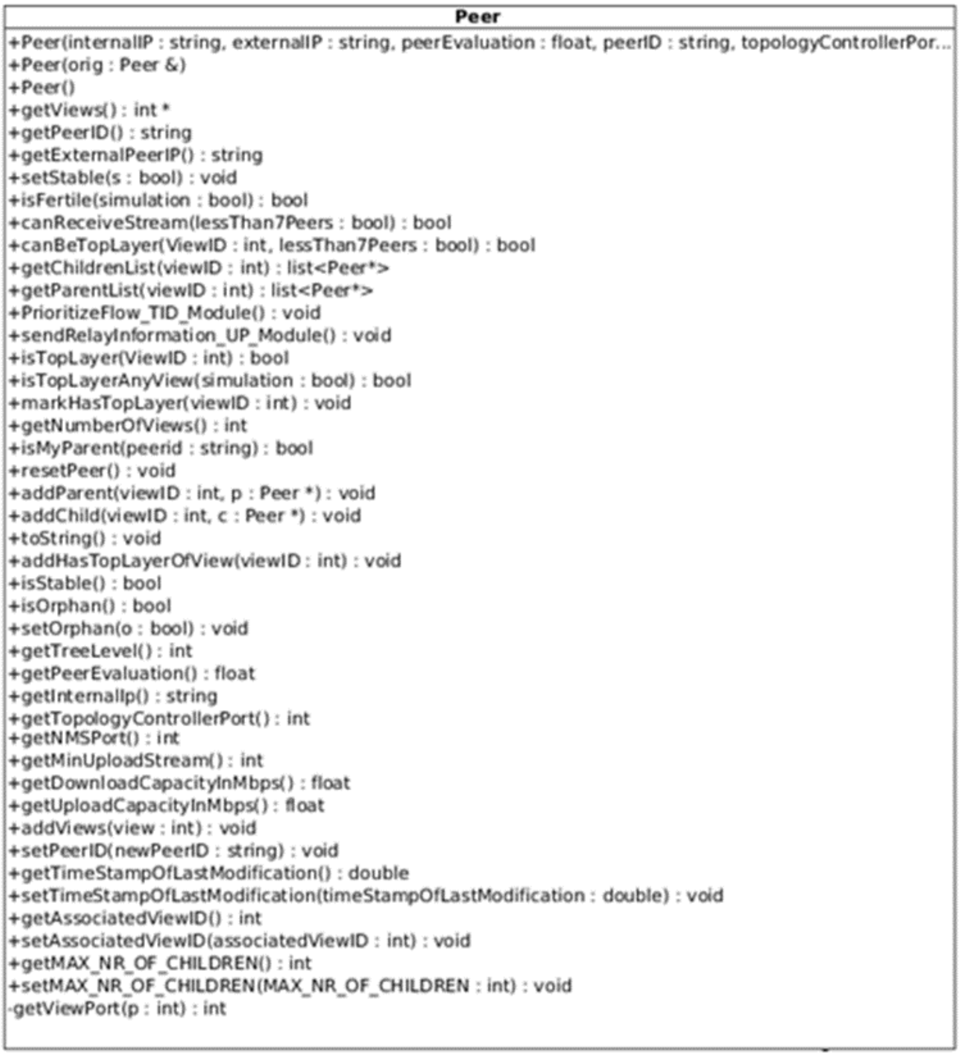


```
TopologyChanges
+TopologyChanges()
+TopologyChanges(orig : TopologyChanges &)
+TopologyChanges()
+sendTopologyChangeToPeer_TC(childList : unordered_map<int,list<Peer*> >, ClientIP : string, clientJsonPort : int) : void
+sendTopLayerPeers_Transmitter(toplayerPeerList : list<Peer*>) : void
```

Figure 14: The TopologyChanges class and its functions

Peer class

The Peer class is responsible to identify a peer. This class collects peer relevant information (such as its evaluation, parent list, children list, status - stable, fertile, orphan), dispatch the prioritization flow to the Virtualization component, for QoS optimization at the access network, and is also responsible to send the children list to the P2P Chunk Selection module. These class functions are listed in Figure 15



```
Peer
+Peer(internalIP : string, externalIP : string, peerEvaluation : float, peerID : string, topologyControllerPor...
+Peer(orig : Peer &)
+Peer()
+getViews() : int *
+getPeerID() : string
+getExternalPeerIP() : string
+setStable(s : bool) : void
+isFertile(simulation : bool) : bool
+canReceiveStream(lessThan7Peers : bool) : bool
+canBeTopLayer(ViewID : int, lessThan7Peers : bool) : bool
+getChildrenList(viewID : int) : list<Peer*>
+getParentList(viewID : int) : list<Peer*>
+PrioritizeFlow_TID_Module() : void
+sendRelayInformation_UP_Module() : void
+isTopLayer(ViewID : int) : bool
+isTopLayerAnyView(simulation : bool) : bool
+markHasTopLayer(viewID : int) : void
+getNumberOfViews() : int
+isMyParent(peerid : string) : bool
+resetPeer() : void
+addParent(viewID : int, p : Peer *) : void
+addChild(viewID : int, c : Peer *) : void
+toString() : void
+addHasTopLayerOfView(viewID : int) : void
+isStable() : bool
+isOrphan() : bool
+setOrphan(o : bool) : void
+getTreeLevel() : int
+getPeerEvaluation() : float
+getInternalIp() : string
+getTopologyControllerPort() : int
+getNMSPort() : int
+getMinUploadStream() : int
+getDownloadCapacityInMbps() : float
+getUploadCapacityInMbps() : float
+addViews(view : int) : void
+setPeerID(newPeerID : string) : void
+getTimeStampOfLastModification() : double
+setTimeStampOfLastModification(timeStampOfLastModification : double) : void
+getAssociatedViewID() : int
+setAssociatedViewID(associatedViewID : int) : void
+getMAX_NR_OF_CHILDREN() : int
+setMAX_NR_OF_CHILDREN(MAX_NR_OF_CHILDREN : int) : void
-getViewPort(p : int) : int
```

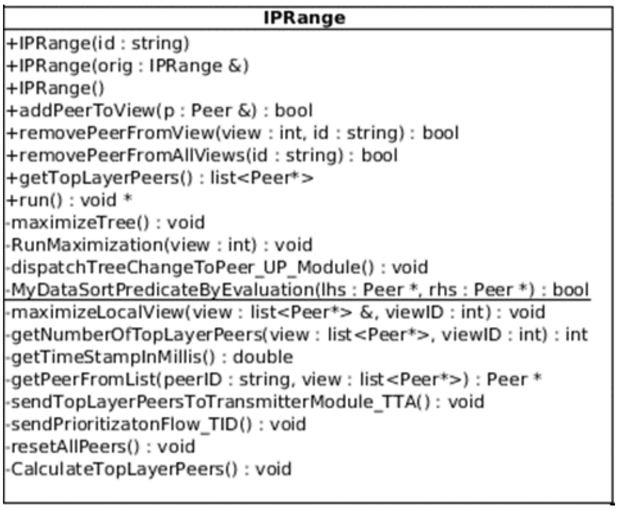
Figure 15: The Peer class and its functions

IPRange class

The IPRange class is responsible to:

- aggregate peers by geographical location, based on the peers IP addresses
- maximize and trim the multicast tree for each content
- add and remove peers from specific views, as a result of the user preferences

These class functions are listed in Figure 16.



```
IPRange
+IPRange(id : string)
+IPRange(orig : IPRange &)
+IPRange()
+addPeerToView(p : Peer &) : bool
+removePeerFromView(view : int, id : string) : bool
+removePeerFromAllViews(id : string) : bool
+getTopLayerPeers() : list<Peer*>
+run() : void *
-maximizeTree() : void
-RunMaximization(view : int) : void
-dispatchTreeChangeToPeer_UP_Module() : void
-MyDataSortPredicateByEvaluation(lhs : Peer *, rhs : Peer *) : bool
-maximizeLocalView(view : list<Peer*> &, viewID : int) : void
-getNumberOfTopLayerPeers(view : list<Peer*>, viewID : int) : int
-getTimeStampInMillis() : double
-getPeerFromList(peerID : string, view : list<Peer*>) : Peer *
-sendTopLayerPeersToTransmitterModule_TTA() : void
-sendPrioritizatonFlow_TID() : void
-resetAllPeers() : void
-CalculateTopLayerPeers() : void
```

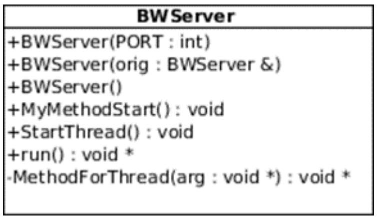
Figure 16: The IPRange class and its functions

BWServer class

The BWServer class is responsible to instantiate individual handlers for each peer that has requested a link test. These class functions are listed in Figure17 .

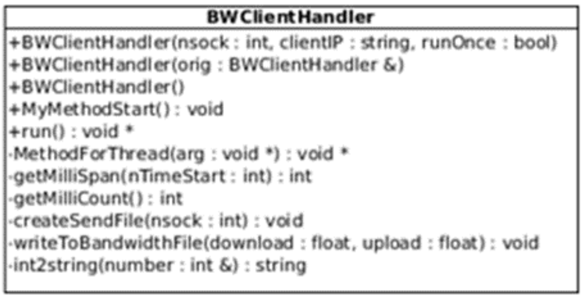
BWClientHandler class

The BWClientHandler class is responsible, at the server/super-peer, to perform the link test towards one peer. These class functions are listed in Figure 18 .



```
BWServer
+BWServer(PORT : int)
+BWServer(orig : BWServer &)
+BWServer()
+MyMethodStart() : void
+StartThread() : void
+run() : void *
-MethodForThread(arg : void *) : void *
```

Figure 17: The BWServer class and its functions

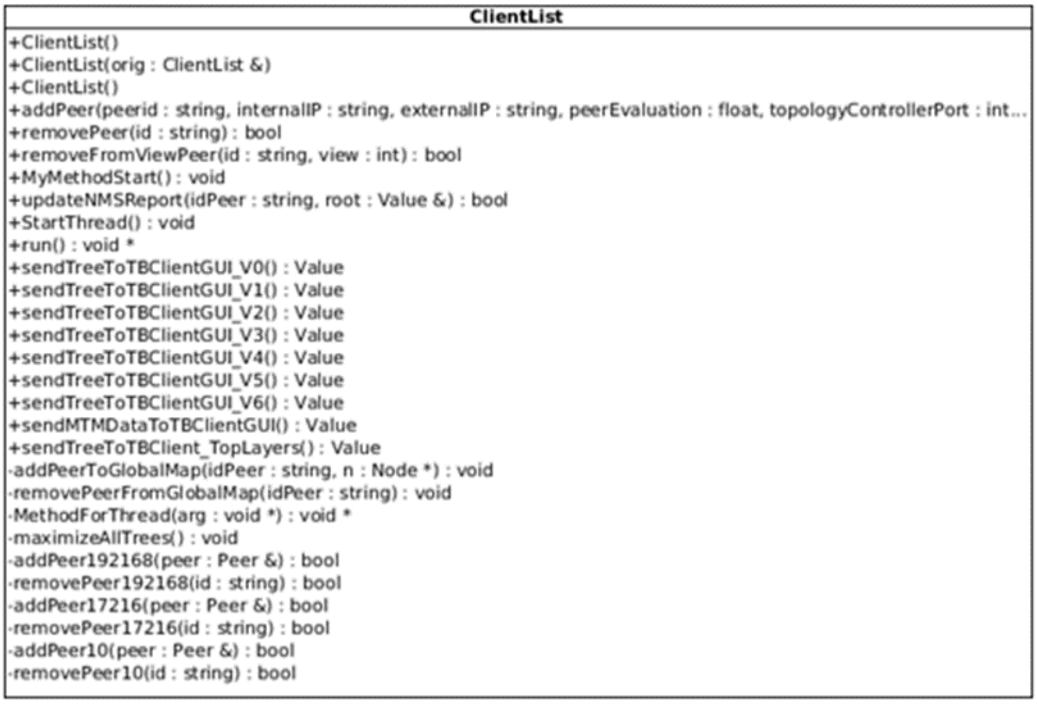


```
BWClientHandler
+BWClientHandler(nsock : int, clientIP : string, runOnce : bool)
+BWClientHandler(orig : BWClientHandler &)
+BWClientHandler()
+MyMethodStart() : void
+run() : void *
-MethodForThread(arg : void *) : void *
-getMilliSpan(nTimeStart : int) : int
-getMilliCount() : int
-createSendFile(nsock : int) : void
-writeToBandwidthFile(download : float, upload : float) : void
-int2string(number : int &) : string
```

Figure 18: The BWClientHandler class and its functions

ClientList class

The ClientList class is responsible to: (i) insert peers in the IPRange class. These class functions are listed in Figure 19



```
ClientList
+ClientList()
+ClientList(orig : ClientList &)
+ClientList()
+addPeer(peerid : string, internalIP : string, externalIP : string, peerEvaluation : float, topologyControllerPort : int...
+removePeer(id : string) : bool
+removeFromViewPeer(id : string, view : int) : bool
+MyMethodStart() : void
+updateNMSReport(idPeer : string, root : Value &) : bool
+StartThread() : void
+run() : void *
+sendTreeToTBClientGUI_V0() : Value
+sendTreeToTBClientGUI_V1() : Value
+sendTreeToTBClientGUI_V2() : Value
+sendTreeToTBClientGUI_V3() : Value
+sendTreeToTBClientGUI_V4() : Value
+sendTreeToTBClientGUI_V5() : Value
+sendTreeToTBClientGUI_V6() : Value
+sendMTMDataToTBClientGUI() : Value
+sendTreeToTBClient_TopLayers() : Value
-addPeerToGlobalMap(idPeer : string, n : Node *) : void
-removePeerFromGlobalMap(idPeer : string) : void
-MethodForThread(arg : void *) : void *
-maximizeAllTrees() : void
-addPeer192168(peer : Peer &) : bool
-removePeer192168(id : string) : bool
-addPeer17216(peer : Peer &) : bool
-removePeer17216(id : string) : bool
-addPeer10(peer : Peer &) : bool
-removePeer10(id : string) : bool
```

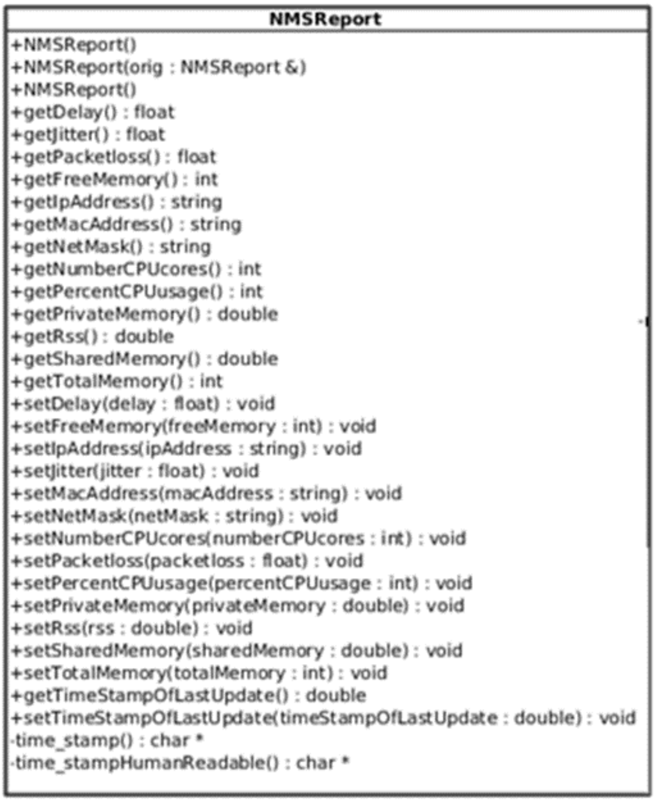
Figure 19: The ClientList class and its functions

NMSReport class

The NMSReport class defines:

- the Network Monitoring Subsystem (NMS) report structure
- the functions to parse a received JSON NMS report and
- the functions to update the node report for a specific peer

These class functions are listed in Figure 20



```
NMSReport
+NMSReport()
+NMSReport(orig : NMSReport &)
+NMSReport()
+getDelay() : float
+getJitter() : float
+getPacketloss() : float
+getFreeMemory() : int
+getIpAddress() : string
+getMacAddress() : string
+getNetMask() : string
+getNumberCPUcores() : int
+getPercentCPUusage() : int
+getPrivateMemory() : double
+getRss() : double
+getSharedMemory() : double
+getTotalMemory() : int
+setDelay(delay : float) : void
+setFreeMemory(freeMemory : int) : void
+setIpAddress(ipAddress : string) : void
+setJitter(jitter : float) : void
+setMacAddress(macAddress : string) : void
+setNetMask(netMask : string) : void
+setNumberCPUcores(numberCPUcores : int) : void
+setPacketloss(packetloss : float) : void
+setPercentCPUusage(percentCPUusage : int) : void
+setPrivateMemory(privateMemory : double) : void
+setRss(rss : double) : void
+setSharedMemory(sharedMemory : double) : void
+setTotalMemory(totalMemory : int) : void
+getTimeStampOfLastUpdate() : double
+setTimeStampOfLastUpdate(timeStampOfLastUpdate : double) : void
-time_stamp() : char *
-time_stampHumanReadable() : char *
```

Figure 20: The NMSReport class and its functions

The Topology Controller and NMS software modules

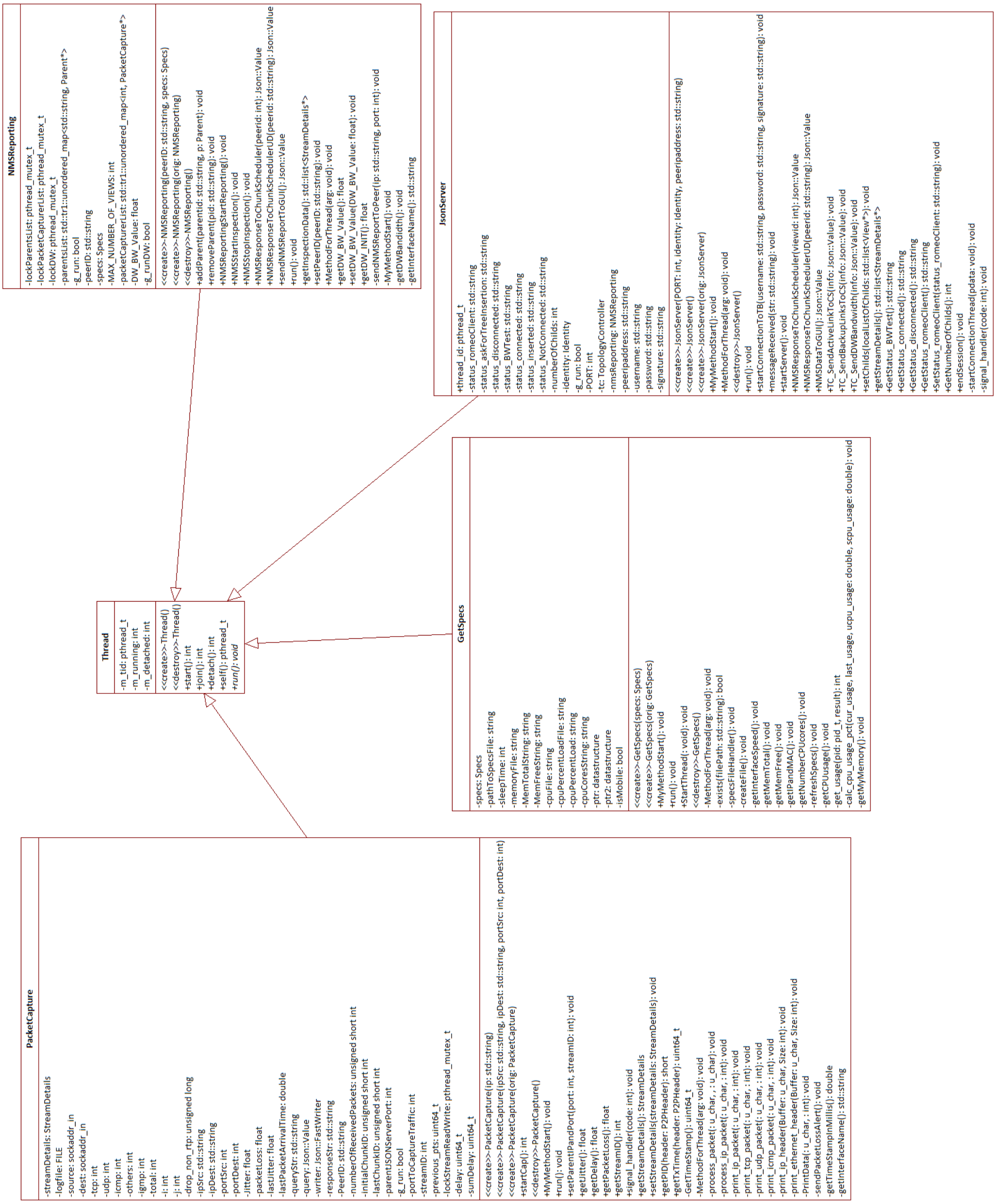

NMSReporting
-lockParentsList: pthread_mutex_t
-lockPacketCapturerList: pthread_mutex_t
-lockDW: pthread_mutex_t
-parentsList: std::tr1::unordered_map<std::string, Parent*>
-g_run: bool
-peerID: std::string
-specs: Specs
-MAX_NUMBER_OF_VIEWS: int
-packetCapturerList: std::tr1::unordered_map<int, PacketCapture*>
-DW_BW_Value: float
-g_runDW: bool
<<create>>-NMSReporting(peerID: std::string, specs: Specs)
<<create>>-NMSReporting(orig: NMSReporting)
<<destroy>>-NMSReporting()
+addParent(parentid: std::string, p: Parent): void
+removeParent(pid: std::string): void
+NMSReportingStartReporting(): void
+NMSStartInspection(): void
+NMSStopInspection(): void
+NMSResponseToChunkScheduler(peerid: int): Json::Value
+NMSResponseToChunkSchedulerUD(peerid: std::string): Json::Value
+sendNMSReportToGUI(): Json::Value
+run(): void
+getInspectionData(): std::list<StreamDetails*>
+setPeerID(peerID: std::string): void
+MethodForThread(arg: void): void
+getDW_BW_Value(): float
+setDW_BW_Value(DW_BW_Value: float): void
+getDW_INIT(): float
-sendNMSReportToPeer(ip: std::string, port: int): void
-MyMethodStart(): void
-getDWBandidth(): void
-getInterfaceName(): std::string
Thread
-m_tid: pthread_t
-m_running: int
-m_detached: int
<<create>>-Thread()
<<destroy>>-Thread()
+start(): int
+join(): int
+detach(): int
+self(): pthread_t
+run(): void
JsonServer
+thread_id: pthread_t
-status_romeoClient: std::string
-status_askForTreeInsertion: std::string
-status_disconnected: std::string
-status_BWTest: std::string
-status_connected: std::string
-status_inserted: std::string
-status_NotConnected: std::string
-numberOfChilds: int
-identity: Identity
-g_run: bool
-PORT: int
-tc: TopologyController
-nmsReporting: NMSReporting
-peeripaddress: std::string
-username: std::string
-password: std::string
-signature: std::string
<<create>>-JsonServer(PORT: int, identity: Identity, peeripaddress: std::string)
<<create>>-JsonServer()
<<create>>-JsonServer(orig: JsonServer)
+MyMethodStart(): void
+MethodForThread(arg: void): void
<<destroy>>-JsonServer()
+run(): void
+startConnectionToTB(username: std::string, password: std::string, signature: std::string): void
+messageReceived(str: std::string): void
+startServer(): void
+NMSResponseToChunkScheduler(viewid: int): Json::Value
+NMSResponseToChunkSchedulerUD(peerid: std::string): Json::Value
+NMSDataToGUI(): Json::Value
+TC_SendActiveLinkToCS(info: Json::Value): void
+TC_SendBackupLinksToCS(info: Json::Value): void
+TC_SendDWBandwidth(info: Json::Value): void
+setChilds(localListOfChilds: std::list<View*>): void
+getStreamDetails(): std::list<StreamDetails*>
+GetStatus_BWTest(): std::string
+GetStatus_connected(): std::string
+GetStatus_disconnected(): std::string
+GetStatus_romeoClient(): std::string
+SetStatus_romeoClient(status_romeoClient: std::string): void
+GetNumberOfChilds(): int
+endSession(): void
-startConnectionThread(pdata: void): void
-signal_handler(code: int): void
GetSpecs
-specs: Specs
-pathToSpecsFile: string
-sleepTime: int
-memoryFile: string
-MemTotalString: string
-MemFreeString: string
-cpuFile: string
-cpuPercentLoadFile: string
-cpuPercentLoad: string
-cpuCoresString: string
-ptr: datastructure
-ptr2: datastructure
-isMobile: bool
<<create>>-GetSpecs(specs: Specs)
<<create>>-GetSpecs(orig: GetSpecs)
+MyMethodStart(): void
+run(): void
+StartThread(: void): void
<<destroy>>-GetSpecs()
-MethodForThread(arg: void): void
-exists(filePath: std::string): bool
-specsFileHandler(): void
-createFile(): void
-getInterfaceSpeed(): void
-getMemTotal(): void
-getMemFree(): void
-getIPandMAC(): void
-getNumberCPUcores(): void
-refreshSpecs(): void
-getCPUusage(): void
-get_usage(pid: pid_t, result): int
-calc_cpu_usage_pct(cur_usage, last_usage, ucpu_usage: double, scpu_usage: double): void
-getMyMemory(): void
PacketCapture
-streamDetails: StreamDetails
-logfile: FILE
-source: sockaddr_in
-dest: sockaddr_in
-tcp: int
-udp: int
-icmp: int
-others: int
-igmp: int
-total: int
-i: int
-j: int
-drop_non_rtp: unsigned long
-ipSrc: std::string
-ipDest: std::string
-portSrc: int
-portDest: int
-Jitter: float
-packetLoss: float
-lastJitter: float
-lastPacketArrivalTime: double
-queryStr: std::string
-query: Json::Value
-writer: Json::FastWriter
-responseStr: std::string
-PeerID: std::string
-numberOfReceivedPackets: unsigned short int
-initialChunkID: unsigned short int
-lastChunkID: unsigned short int
-parentJSONServerPort: int
-g_run: bool
-portToCaptureTraffic: int
-streamID: int
-previous_pts: uint64_t
-lockStreamReadWrite: pthread_mutex_t
-delay: uint64_t
-sumDelay: uint64_t
<<create>>-PacketCapture(ip: std::string)
<<create>>-PacketCapture(ipSrc: std::string, ipDest: std::string, portSrc: int, portDest: int)
<<create>>-PacketCapture(orig: PacketCapture)
+startCap(): int
<<destroy>>-PacketCapture()
+MyMethodStart(): void
+run(): void
+setParentIPandPort(port: int, streamID: int): void
+getJitter(): float
+getDelay(): float
+getPacketLoss(): float
+getStreamID(): int
+signal_handler(code: int): void
+getStreamDetails(): StreamDetails
+setStreamDetails(streamDetails: StreamDetails): void
+getPID(header: P2PHeader): short
+getTxTime(header: P2PHeader): uint64_t
-GetTimeStamp(): uint64_t
-MethodForThread(arg: void): void
-process_packet(: u_char, , : u_char): void
-process_ip_packet(: u_char, : int): void
-print_ip_packet(: u_char, : int): void
-print_tcp_packet(: u_char, : int): void
-print_udp_packet(: u_char, : int): void
-print_icmp_packet(: u_char, : int): void
-print_ip_header(Buffer: u_char, Size: int): void
-print_ethernet_header(Buffer: u_char, Size: int): void
-PrintData(: u_char, : int): void
-sendPacketLossAlert(): void
-getTimeStampInMillis(): double
-getInterfaceName(): std::string

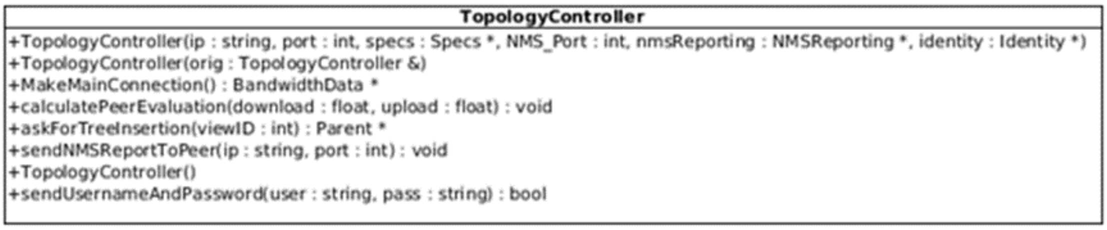
Figure 21: TC and NMS simplified class diagram

The TopologyController class is responsible for:

- triggering the peer authentication
- requesting NMS link test

- compute the peer evaluation
- request for tree insertion
- triggering the NMS reporting service.

These class functions are listed in Figure 22.

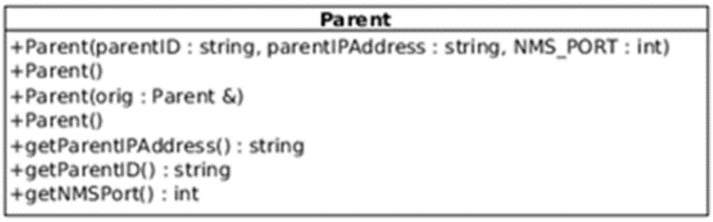


```
TopologyController
+TopologyController(ip : string, port : int, specs : Specs *, NMS_Port : int, nmsReporting : NMSReporting *, identity : Identity *)
+TopologyController(orig : TopologyController &)
+MakeMainConnection() : BandwidthData *
+calculatePeerEvaluation(download : float, upload : float) : void
+askForTreeInsertion(viewID : int) : Parent *
+sendNMSReportToPeer(ip : string, port : int) : void
+TopologyController()
+sendUsernameAndPassword(user : string, pass : string) : bool
```

Figure 22: The TopologyController class and its functions

Parent class

The Parent class provides the list of parents (active and backup) for each peer and content type. These class functions are listed in Figure 23.



```
Parent
+Parent(parentID : string, parentIPAddress : string, NMS_PORT : int)
+Parent()
+Parent(orig : Parent &)
+Parent()
+getParentIPAddress() : string
+getParentID() : string
+getNMSPort() : int
```

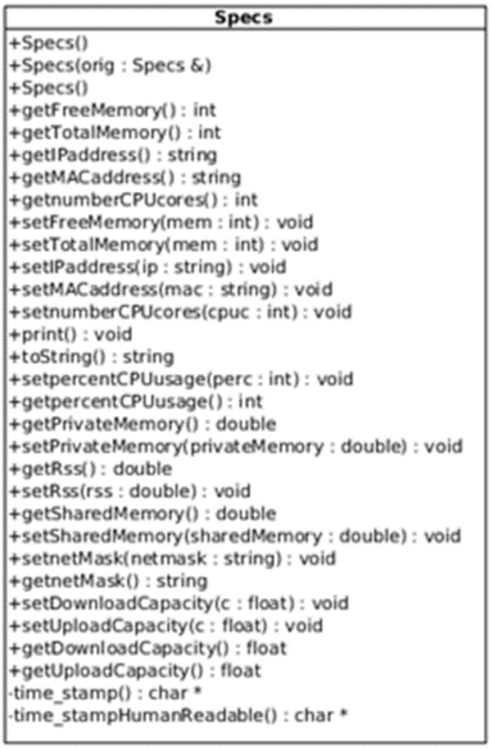
Figure 23: The Parent class and its functions

Network Monitoring Subsystem

The NMS software module is constituted by the following classes:

Specs class

The Specs class provides the functions to retrieve the peers hardware information. These class functions are listed in Figure 24

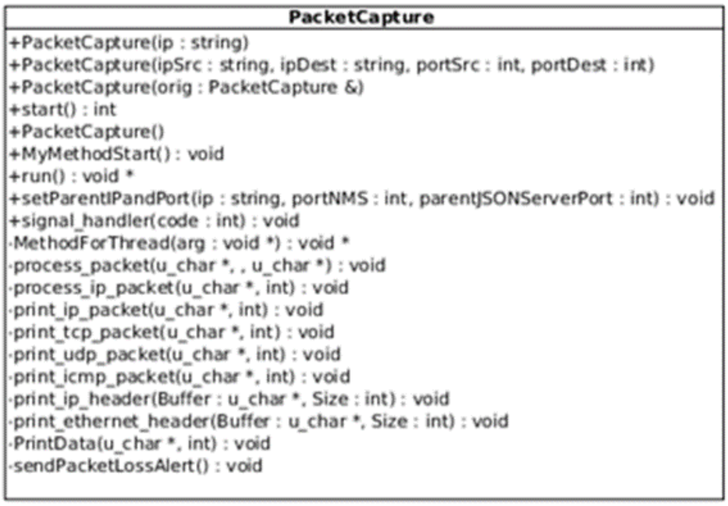


```
Specs
+Specs()
+Specs(orig : Specs &)
+Specs()
+getFreeMemory() : int
+getTotalMemory() : int
+getIPaddress() : string
+getMACaddress() : string
+getnumberCPUcores() : int
+setFreeMemory(mem : int) : void
+setTotalMemory(mem : int) : void
+setIPaddress(ip : string) : void
+setMACaddress(mac : string) : void
+setnumberCPUcores(cpuc : int) : void
+print() : void
+toString() : string
+setpercentCPUusage(perc : int) : void
+getpercentCPUusage() : int
+getPrivateMemory() : double
+setPrivateMemory(privateMemory : double) : void
+getRss() : double
+setRss(rss : double) : void
+getSharedMemory() : double
+setSharedMemory(sharedMemory : double) : void
+setnetMask(netmask : string) : void
+getnetMask() : string
+setDownloadCapacity(c : float) : void
+setUploadCapacity(c : float) : void
+getDownloadCapacity() : float
+getUploadCapacity() : float
-time_stamp() : char *
-time_stampHumanReadable() : char *
```

Figure 24: The Specs class and its functions

PacketCapture class

The PacketCapture class is responsible to collect network data such as, connection history (start time, duration) or traffic statistics (packet loss, delay, jitter), by exploiting the libpcap[2] library. These class functions are listed in Figure 25



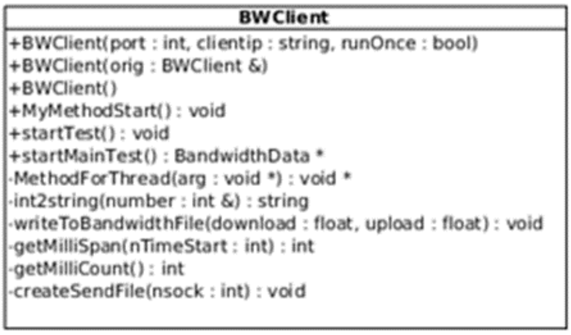
```
PacketCapture
+PacketCapture(ip : string)
+PacketCapture(ipSrc : string, ipDest : string, portSrc : int, portDest : int)
+PacketCapture(orig : PacketCapture &)
+start() : int
+PacketCapture()
+MyMethodStart() : void
+run() : void *
+setParentIPandPort(ip : string, portNMS : int, parentJSONServerPort : int) : void
+signal_handler(code : int) : void
-MethodForThread(arg : void *) : void *
-process_packet(u_char *, , u_char *) : void
-process_ip_packet(u_char *, int) : void
-print_ip_packet(u_char *, int) : void
-print_tcp_packet(u_char *, int) : void
-print_udp_packet(u_char *, int) : void
-print_icmp_packet(u_char *, int) : void
-print_ip_header(Buffer : u_char *, Size : int) : void
-print_ethernet_header(Buffer : u_char *, Size : int) : void
-PrintData(u_char *, int) : void
-sendPacketLossAlert() : void
```

Figure 25: The PacketCapture class and its functions

[2] libpcap, a portable C/C++ library for network traffic capture - http://www.tcpdump.org/

BWClient class

The BWClient class is responsible, at the peer, to perform the link test towards the MTM running at the server/super-peer. These class functions are listed in Figure 26



```
BWClient
+BWClient(port : int, clientip : string, runOnce : bool)
+BWClient(orig : BWClient &)
+BWClient()
+MyMethodStart() : void
+startTest() : void
+startMainTest() : BandwidthData *
-MethodForThread(arg : void *) : void *
-int2string(number : int &) : string
-writeToBandwidthFile(download : float, upload : float) : void
-getMilliSpan(nTimeStart : int) : int
-getMilliCount() : int
-createSendFile(nsock : int) : void
```

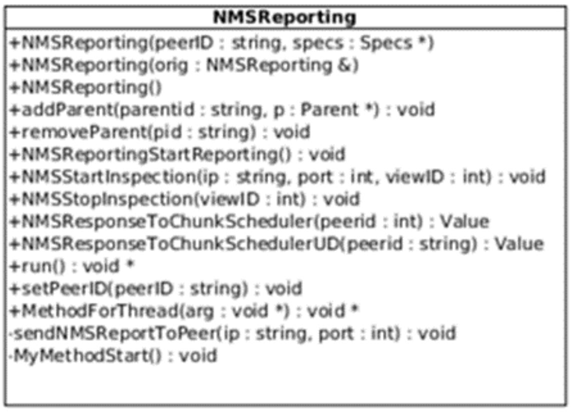
Figure 26: The BWClient class and its functions

NMSReporting class

The NMSReporting class is responsible for:

- knowing the list of parents of the peer.
- initiating the packet inspection mechanism;
- creating the NMS periodic reports;
- sending the NMS reports towards a parent
- collect NMS reports from children peers.

These class functions are listed in Figure 27



```
NMSReporting
+NMSReporting(peerID : string, specs : Specs *)
+NMSReporting(orig : NMSReporting &)
+NMSReporting()
+addParent(parentid : string, p : Parent *) : void
+removeParent(pid : string) : void
+NMSReportingStartReporting() : void
+NMSStartInspection(ip : string, port : int, viewID : int) : void
+NMSStopInspection(viewID : int) : void
+NMSResponseToChunkScheduler(peerid : int) : Value
+NMSResponseToChunkSchedulerUD(peerid : string) : Value
+run() : void *
+setPeerID(peerID : string) : void
+MethodForThread(arg : void *) : void *
-sendNMSReportToPeer(ip : string, port : int) : void
-MyMethodStart() : void
```

Figure 27: The NMSReporting class and its functions

5 System Evaluation

The system evaluation has been performed by testbed implementation. The system evaluation consisted in measuring the memory footprint and the P2P tree computation time at the super-peer, measuring the CPU consumption and NMS bandwidth consumption at the peer. The server and client modules (TB, MTM, TC and NMS) were developed in C++ and implemented in Ubuntu Server 12.10 AMD64 with a Kernel version of 3.8.0-19-generic #30-Ubuntu SMP, powered by an Intel core i7 720QM with 8GB of RAM.

5.1 Test bed experimentation for performance evaluation

The test bed simulates a single ISP's core and access networks consisting of one Super-peer and seven peers, running the all modules as described in the previous sections. The overall test bed topology is depicted in Figure 28 where the colored dotted lines are representations of the media flows used for the test bed (Some modules are represented at the Figure 28 but they are not part of this work, although they were necessary to complete the tests - P2PTx - CS - P2PRx).

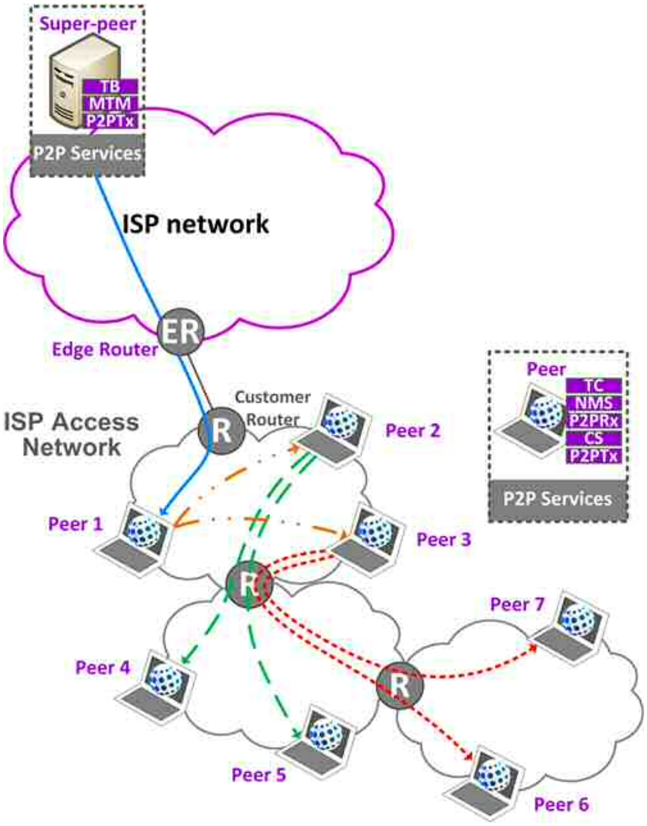


Figure 28: P2P network test bed topology

During the tests, peers join the service and the Super-peer updates the application-level multicast trees accordingly. To measure the P2P network performance, the tree depth consists of three levels, with a single top-level parent. Table 4 depicts the used streams from the prepared scalable multiple-descriptions content, take in consideration that the first row represents the content that is sent by the DVB[3], so it is out of scope of the Topology Builder operation, anyway this stream is necessary for correct P2P behavior of the system.

Table 4: Streams used for the experiment and their tree id

Tree ID	Stream Name
No Tree ID	Base Layer Half color (Camera 2 + Camera 3) Description 1
0	Base Layer Half color (Camera 2 + Camera 3) Description 2
	Q. Enhancement Layer Half color (Camera 2 + Camera 3) Description 2
1	Base Layer Half depth (Camera 2 + Camera 3) Description 1
	Q. Enhancement Layer Half depth (Camera 2 + Camera 3) Description 1
2	Base Layer Half depth (Camera 2 + Camera 3) Description 2
	Q. Enhancement Layer Half depth (Camera 2 + Camera 3) Description 2

Following this set up, the test bed allows measuring the following performance metrics:

- CPU consumption by the different modules;
- Memory footprint at the TB;
- Tree computation time at the TB;
- Consumed bandwidth by the NMS periodic reporting system;

5.1.1 P2P Tree Computation Time

The evaluation of P2P tree computation time is crucial to understand how the system would perform to the constant changes in the access network. Figure 29 shows the evolution of this performance indicator when computing a single tree versus the number of peers in the system. If we assume the worst case in which all 8 trees (see Table 2) would have 100.000 peers, the system would have taken approximately 4,5 seconds.

For the general cases, the computation of the full P2P multicast trees would be less than 1 second. It is important to note that the proposed framework can work with any application-level multicast tree construction algorithm.

[3] DVB: Digital Video Broadcasting is a suite of internationally accepted open standards for digital television

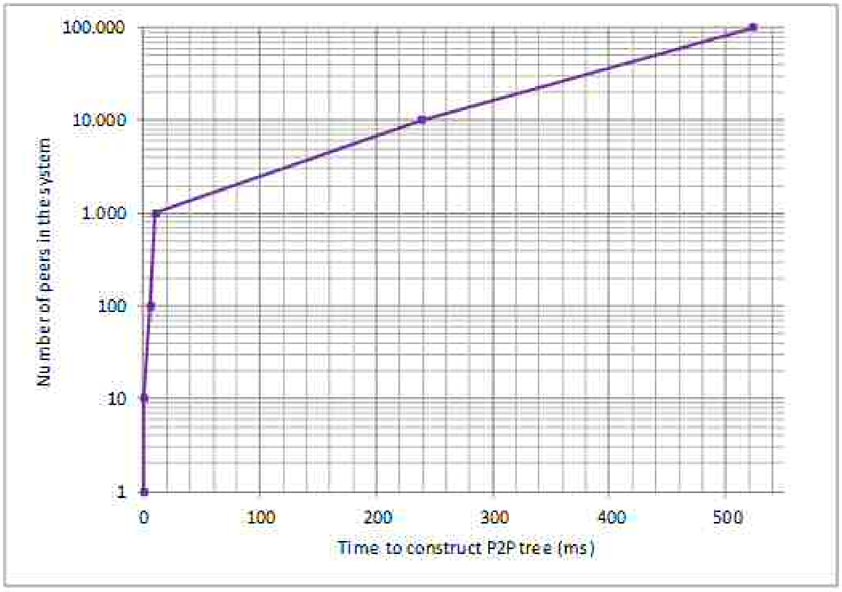


Figure 29: Time to compute the P2P multicast tree versus number of peers in the system.

5.1.2 Peer CPU consumption

At the peer side, the networking operations performed by the TC have a negligible impact (<0,1%), being the network monitoring and reporting performed by the NMS the most relevant part of the CPU consumption at the peer. As depicted in Figure 30, when the peer receives the full range of the video streams, the NMS will consume a maximum of 10% of its CPU resources.

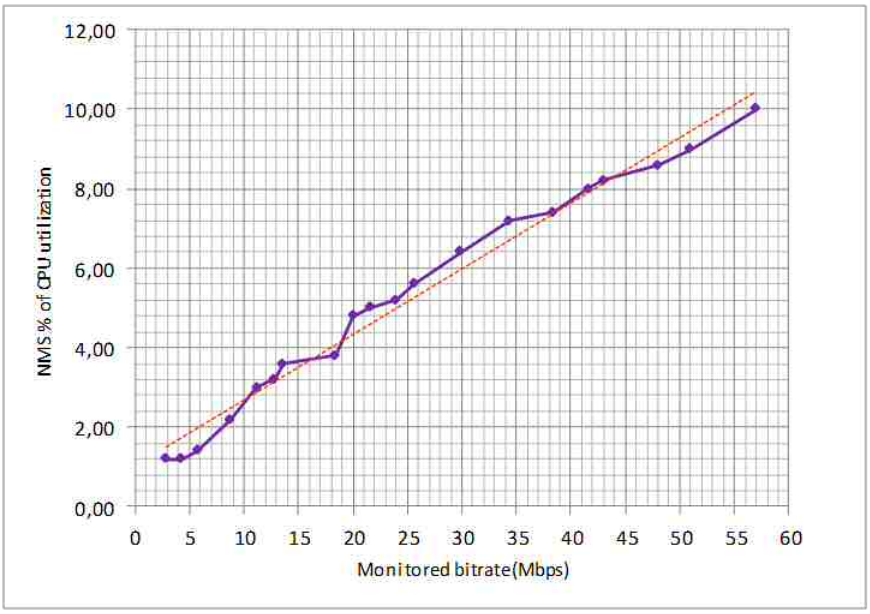


Figure 30: Percentage of CPU utilization at the peer for multiple bitrates received. At the highest point, the peer will consume 10% of its CPU when monitoring the 15 video streams.

5.1.3 Memory Footprint

Figure 31 depicts the memory footprint for the super-peer in light and heavy conditions. The results were computed for P2P tree construction and maintenance operations only. The experiment consisted on the super-peer receiving requests from virtual peers up to a maximum of 100.000 requests. At its maximum, the total memory consumed at the super-peer was approximately 48Mbytes, which indicate a highly scalable algorithm. Please note that due to the distributed nature of the system, super-peers are expected to support much less peers and only in very rare occasions a super-peer will need to support such a high number of peers. ISPs in such conditions may split clients by zone or perform load-balancing techniques amongst two or more super-peers.

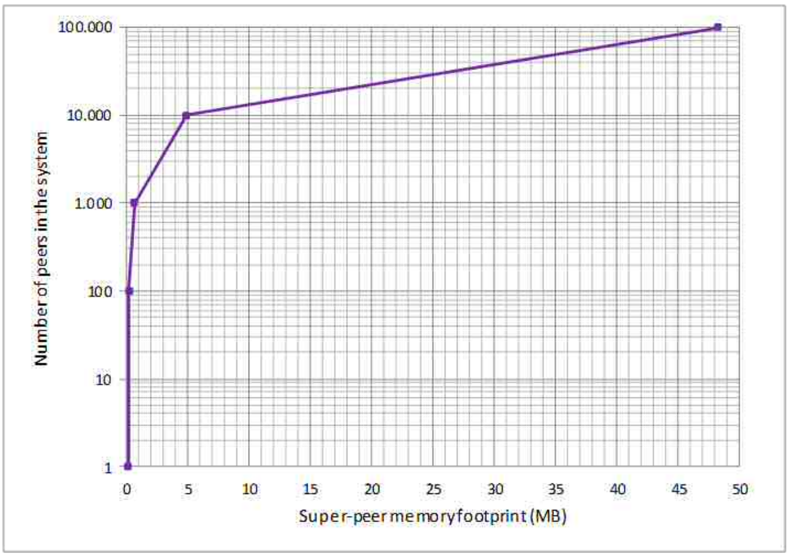


Figure 31: Memory footprint for the P2P tree construction and maintenance operations at the super-peer versus number of peers in the system.

5.1.4 NMS bandwidth consumption

As explained in chapter 3.2.3, for tree maintenance purposes, each peer periodically sends towards the MTM, a report with the structure depicted in Table 3. In our test environment a single report has a size of 387 Bytes, 58 come from Ethernet, IP and TCP related overheads and the remaining from the JavaScript Object Notation (JSON)[4] and report structure. Considering an access network of 1Gbps, the bandwidth of each stream to be approximately 4Mbps and that a minimum of 3 streams needs to be at least received at each peer, we calculate the worst case scenario in which one single parent will have to support 75 children (this leaves 10% available bandwidth for other networking operations). In such scenario, the average bandwidth consumed by the NMS is dependent from the periodicity of the report and the number of children being supported.

[4] JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999.

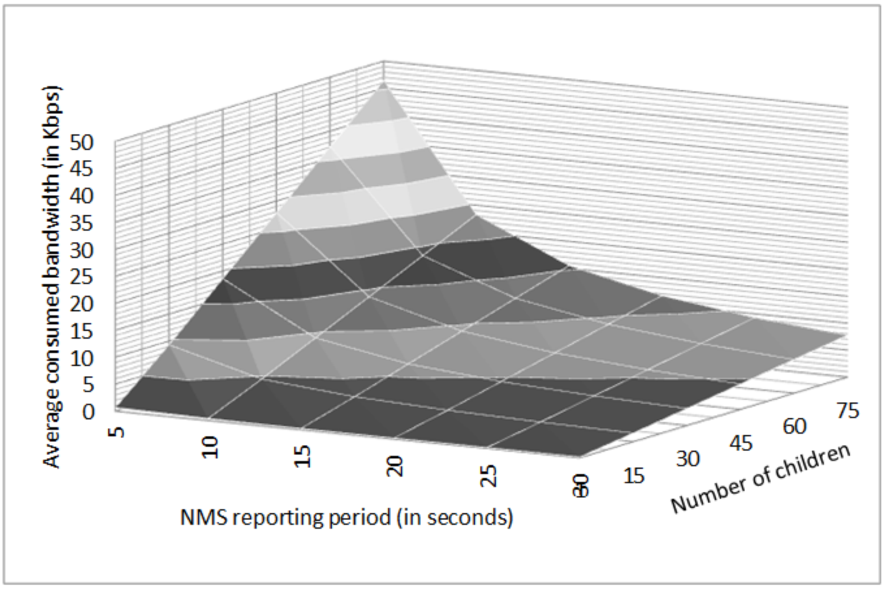


Figure 32: Average consumed bandwidth by a parent when sending the aggregated NMS reports towards the MTM at the super-peer.

As it can be seen from the graphic in Figure 32, the NMS running at a peer with 75 children consumes an average 45Kbps if the report is to be sent every 5 seconds. Even though this is the worst case scenario it still consumes a very low bitrate.

5.1.5 Topology Controller CPU consumption

The results measured for the TC are less than 0.1%, this is the minimum unit that our tool could capture. The collected values represent a negligible load on the CPU.

5.2 Proof of Concept by testbed

5.2.1 Metrics

Several network metrics were retrieved to evaluate the network impact and QoE of the received media. Bellow you can find a description of the used metrics.

Average end-to-end delay

The objective is to determine if the P2P paths chosen by the TB are adequate. Congested paths will typically increase the end-to-end delay. In a stream of n packets sent by the server/super-peer and successfully received by the peer, the average end-to-end delay is given by:

$$Delay = \frac{\sum_{j=1}^{n} EndTime_j - StartTime_j}{n} \tag{3}$$

Where:

- **StartTime**: is the time at which packet j was sent.

- **EndTime**: is the time at which packet j was received at the destination node.

Interarrival jitter

Interarrival jitter determines how stable are the P2P paths chosen by the TB. This metric also evaluates if the TB's algorithm for the construction of P2P application-level multicast trees is suitable for project. The interarrival jitter (J) is defined to be the mean deviation (smoothed absolute value) of the difference (D) in packet spacing at the receiver compared to the sender for a pair of packets. If Si is the timestamp from packet i, and Ri is the time of arrival for packet i, then for two packets i and j, D may be expressed according to:

$$D(i,j) = |(R_j - S_j) - (R_i - S_i)| \tag{4}$$

As per RFC 3550[5], the interarrival jitter should be calculated continuously as each data packet i is received from source, using this difference D for that packet and the previous packet i-1 in order of arrival (not necessarily in sequence).

$$Jitter = J(i-1) + \frac{|D(i-1,i)| - J(i-1)}{16} \tag{5}$$

The jitter calculation must conform to uppermentioned equation in order to allow independent monitors to make valid interpretations of reports coming from different applications. The gain parameter 1/16 gives a good noise reduction ratio while maintaining a reasonable rate of convergence as stated in RFC 3550.

Percentage of packet loss

In order to compute packet loss rates, the number of packets expected and actually received from the system server/super-peer must be known. Since the Transmission mechanism (Out of the scope of this report) only distributes content via the User Datagram Protocol (UDP), there is no transport layer flow control. To deal with this limitation the following approach is used: the number of received packets corresponds to the count of packets as they arrive, including late or duplicate packets, and the number of packets expected can be computed by the receiver as the difference between the highest and first sequence numbers received within a time-window. To avoid the sequence number wrap-around problem, the packets includes a 32 bit sequence number field.

5.3 Inter-ISP evaluation for service assessment

5.3.1 Description

This section describes an attempt to measure the performance of inter-ISP connections between the ISPs that are not enabled with the discussed service. (please refer to 3.3 for more

[5] RFC 3550: http://www.ietf.org/rfc/rfc3550.txt

details). In such a test, the Core and Access networks of the involved ISPs are agnostic to the our traffic, treating it with a best effort policy. The performance evaluation is performed using the iperf [6] tool to measure jitter and packet loss in the set up scenarios presented in Table 4.

Table 5: Inter-ISP scenarios

	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Super-peer	Germany	United Kingdom	Greece	Portugal	Portugal
Peer	Portugal	Portugal	Portugal	Spain	Turkey

The inter-ISP network topology used for this test is depicted in Figure 33. For each of the scenarios in Figure 33, the iperf tool was configured to send multiple IP streams with the following configuration:

- Transport protocol: UDP
- Size of the protocol data unit: 1380 Bytes (average size as measured in test-bed)
- Sending bitrate: Constant bitrate of 3.5 Mbps. (average bitrate as measured in the test-bed).

The first three scenarios present no bandwidth link limitation and the measurements intend to evaluate the performance of the inter-ISP connection for best-effort service. Scenarios 4 and 5 present bandwidth link limitation and the goal is to measure how much it could affect the visualization of described system content in today's Internet.

[6] Iperf is a tool to measure maximum TCP bandwidth, allowing the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay jitter, datagram loss - URL: http://iperf.fr/

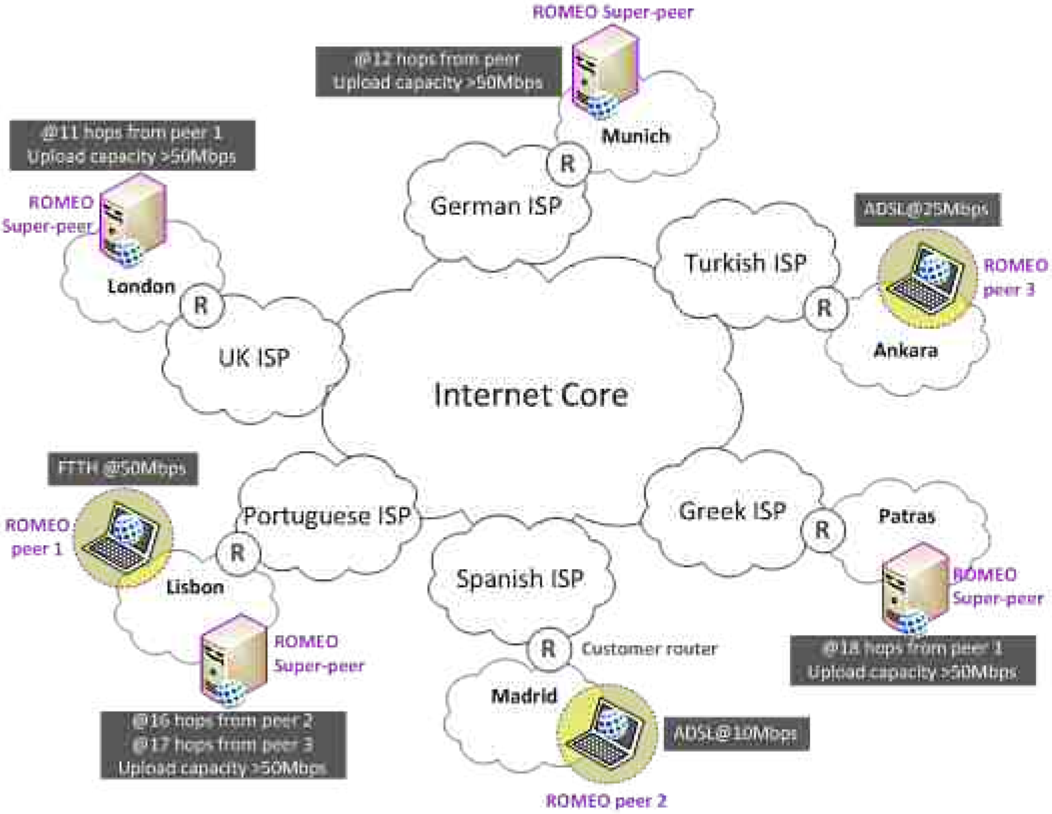


Figure 33: Inter-ISP topology

All tests have been performed on working days during peak hours, between 10:00h and 17:30h (CET), hence they reflect the worst case scenario for such transmissions. The only exception was scenario 5, performed at 21:00h CET, in which the goal is to measure the performance in a typical end-of-day residential use case.

5.4 Results and summary

Using the above mentioned scenarios and procedures, Figure 34 and Figure 35 depict the obtained results for scenarios 1 to 3, and Figure 36 and Figure 37 depict the results obtained for scenarios 4 and 5, respectively.

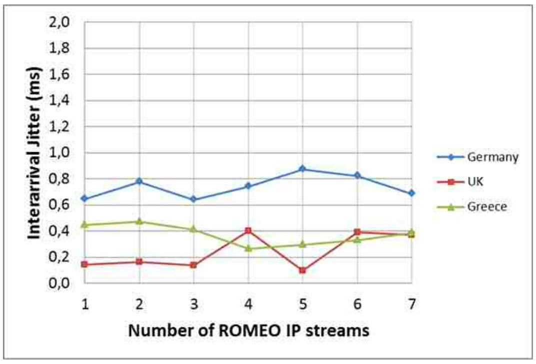


Figure 34: Inter-arrival jitter (ms) between Super-peer and Peer

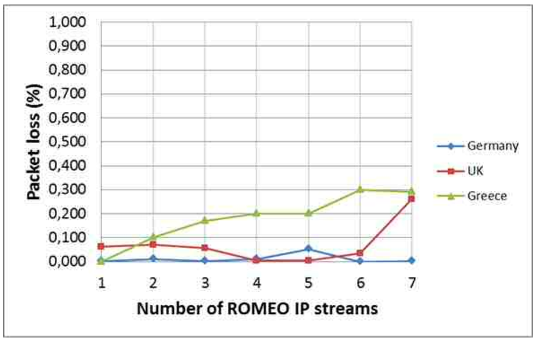


Figure 35: Percentage of packet loss between Super peer and Peer

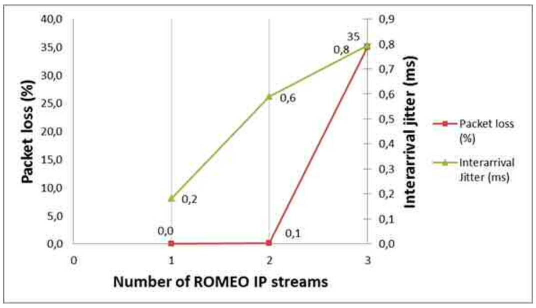


Figure 36: Inter-arrival jitter (ms) and percentage of packet loss in a residential 10Mbps ADSL connection under peak hours

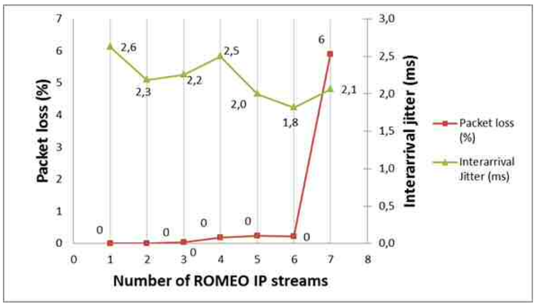


Figure 37: Inter-arrival jitter (ms) and percentage of packet loss in a residential 25Mbps ADSL+ connection during non-peak hours

Form the performance graphics shown it can be concluded that the packet inter-arrival jitter does not constitute a major issue in any of the tested scenarios. This is the case even when the traffic is traversing multiple ISPs under the best-effort treatment. From Figure 34 and Figure 35, it can be concluded that the service, if deployed in a small to medium scale, could be supported by today's Internet without any change in the ISPs equipment or topology. In larger-scale deployments, with the increase in the number of clients, ISPs would be forced to take improvement measures, and the P2P overlay approach could pose a possible solution. From Figure 36 it can be expected that a residential 10Mbps ADSL connection is not suitable for users during peak hours. The server was only able to deliver 2 IP streams, above which the packet loss rate increases to unacceptable levels. For out-of-the-peak hours, Figure 37 suggests that a 25Mbps connection can provide acceptable conditions for up to 6 IP streams.

6 Conclusion and Future Work

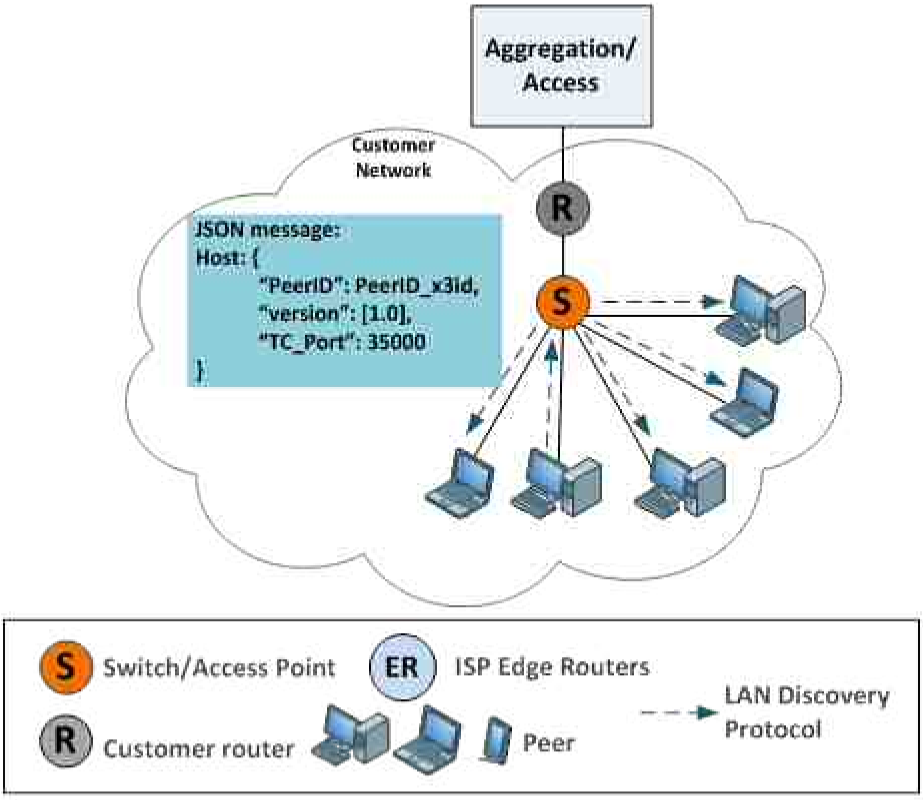
As stated on section 1, recent studies (Cisco, 2011-2016, 2012 to 2017) have forecast a major growth on Internet based video traffic with a compound annual growth rate of 34% till 2016 for general consumers and 75% for mobile video traffic (mobile category includes laptops with mobile data cards, USB modems, and other portable devices with embedded cellular connectivity).

Globally, Internet video traffic will represent 55% of all consumer Internet traffic in 2016. This trend suggests Internet video traffic will play a major role in the Future Internet. In fact, high-definition video-on-demand (VoD) surpassed standard definition by the end of 2011 and it is expected that by 2016, high-definition Internet video will comprise 79% of VoD and 3D VoD is expected to achieve a compound annual growth rate of 109%.

Major challenges for service providers lie ahead, especially concerning the efficient use of their network resources. IP multicast technology may have its opportunity to finally be deployed in large scale. Meanwhile, the top sites on the Internet continue to an use unicast client-server approach, which is known to be inefficient when distributing large volumes of data. However, it is not possible to predict with any certainty how the Internet will mature - it is unclear what would be the globally adopted solution in the next years. With this in mind this article proposes a framework that introduces the concept of a hybrid client-server/P2P approach for large content media distribution over the Internet, hopefully retaining the desirable properties of each.

The approach uses multiple multicast trees to distribute the multiple streams associated with the visualization of bandwidth consuming 3D content. When compared to pure client-server or P2P distribution scenarios, the proposed approach is able to effectively distribute the load among most participating nodes while respecting individual node bandwidth constraints and achieving a fast insertion and tree reconstruction time - only possible when using a centralized approach. The presented solution uses multiple distribution trees, which differs from many other approaches, and enables a fair share among participating peers proportional to their rank - being the only exception the leaf peers. Leaf (child) peers are the lowest rank peers and in most situations their participation on the distribution of content should be avoided mobile terminals are provided as a good example of leaf peers since they have considerable restrictions in their download/upload capacity, memory and CPU, and also traffic and battery.

As future work, this platform can benefit from having a peer discovery protocol for Local Area Networks. The purposed protocol, depicted in Figure 6 could be used to automatically discover peers that are benefiting and participating on the P2P overlay network, this will bring the advantage of maximizing data streaming at the LAN level and reducing the consumed bandwidth between the access network and the customer's router.



```
JSON message:
Host: {
        "PeerID": PeerID_x3id,
        "version": [1.0],
        "TC_Port": 35000
}
```

Figure 38: Peer Discovery protocol - Purposed protocol

References

A. Mohr, E. Riskin R L 2000 *IEEE JSAC* pp. 819–828.

AKAMAI 2013 State of the internet Technical report AKAMAI.

Alexa 2013 Alexa - the web information company, the top 500 sites on the web Technical report.
URL: *http://www.alexa.com/topsites*

Apostolopoulos J G 2001 *In Visual Communications and Image Processing* .

Apostolopoulos J G 2011 *In Visual Communications and Image Processing* .

Cisco 2011-2016 Cisco visual networking index: Forecast and methodology Technical report.

Cisco 2012 to 2017 Cisco visual networking index: Global mobile data traffic forecast update Technical report.

D. Andersen, H. Balakrishnan F K and Morris R 2001 *In SOSP01* .

Diot C, Levine B, Lyles B, Kassem H and Balensiefen D 2000 *Network, IEEE* **14**(1), 78–88.

H. Silva, H. Marques J R C V 2012 *HP-MOSys* .

J. G. Apostolopoulos S J W 2001 *IEEE International Conference on Image Processing* .

M. Castro, P. Druschel A M K A N A R and Singh A 2003 *IPTPS, Berkeley, CA* .

M. Zhang L S and Yang S 2008 *IEEE GLOBECOM* .

Mislove A, Marcon M, Gummadi K P, Druschel P and Bhattacharjee B 2007 *in* 'Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement' IMC '07 ACM New York, NY, USA pp. 29–42.
URL: *http://doi.acm.org/10.1145/1298306.1298311*

MUSCADE MUltimedia SCAlable 3D for Europe 2014.
URL: *http://www.muscade.eu*

Nakamura T 2010.

Remote Collaborative Real-Time Multimedia Experience over the Future Internet 2014.
URL: *http://www.ict-romeo.eu/*

Research and Markets 2014*a* Global 3dtv market 2014-2018 Technical report Research and Markets.

Research and Markets 2014*b* Global fibre-to-the-home (ftth) equipment trends to 2017 Technical report Research and Markets.

Sandvine 2011 Global internet phenomena report Technical report.
URL: *http: //www.sandvine.com/news/global broadband trends.asp*

Tam W J 2006 *Multimedia and Expo* .

Theodore Zahariadis, Petros Daras I L B 2008 *NEM Summit* .

V. N. Padmanabhan, H. J. Wang P A C and Sripanidkulchai K 2002 *ACM NOSSDAV* .

X. Zhang, J. Liu B L and Yum P 2005 *In Proc of IEEE INFOCOM* .