Network Protocols and Algorithms
ISSN 1943-3581
2009, Vol. 1, No. 2

NNodeTree: A Scalable Peer-to-Peer Live Streaming Overlay Architecture for Next-Generation-Networks

Julius Müller
Fraunhofer FOKUS
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
0049-30-3463-7170 Julius.Mueller@fokus.fraunhofer.de

Thomas Magedanz
TU Berlin
Franklinstrasse 28-29, 10587 Berlin, Germany
0049-30-3463-7229 Thomas.Magedanz@tu-berlin.de

Jens Fiedler
Fraunhofer FOKUS
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
0049-30-3463-7270 Jens.Fiedler@fokus.fraunhofer.de

Abstract

The rapid evolution of the telecommunication domain increases the performance of different access networks continuously. New services, especially in the domain multimedia content distribution, require higher and higher bandwidth at the user’s and service provider’s side [1].

Multimedia services like Video on Demand, IPTV and live streaming were introduced in the past and are still improved in quality and quantity. Multimedia streams and Peer to Peer (P2P) file sharing dominates the worldwide Internet traffic nowadays and will continue further.

The user acceptance of enjoying multimedia content over the Internet will grow steadily together with the increasing quality of the available multimedia content. Network operators and service providers have to face the growths, by either increasing their service platform with higher performance and bandwidth or introducing a scalable solution.

We present the design and implementation of a scalable Peer-to-Peer live streaming overlay architecture for Next-Generation-Networks (NGN) in this paper that addresses this challenge.

Keywords: P2P, Tree Overlay, Scalability, Live Streaming, IMS, NGN

1. INTRODUCTION

The client-server (CS) architecture is currently the most prevalent and established infrastructure of the Internet. The CS architecture enables reliable security, efficient charging schemas, but is not scalable and therefore not suitable for multimedia services to millions of people at the same time. The rapid evolution of the telecommunication domain increases the performance of different access networks continuously. New services emerged out of this evolution, which is motivated by user requirements, new technologies and user adaptation. The IP Multimedia Subsystem (IMS) [2] enables and amplifies the convergence of telecommunication and Internet. Multimedia streams and P2P file sharing dominates the worldwide Internet traffic nowadays and will continue further. The user acceptance of enjoying multimedia content over the Internet will grow steadily together with the increasing quality of the available multimedia content. Network operators and service providers have to face this growth, by increasing their service platform with a higher performance and bandwidth or by introducing a scalable multimedia distribution architecture.

In this paper we present the design and implementation of a scalable Peer-to-Peer Live Streaming Multimedia Distribution Overlay Architecture for Next-Generation-Networks [3][4][5]. An IMS/P2P client is developed in the scope of this paper that enables scalable multimedia content distribution. For that purpose an existing IMS client is extended with P2P functionalities to support live streaming between peers of the same overlay. New developed services located in the IMS support the client.

An extensive testing follows the validation and proves the functionality of the new developed software prototype in different service scenarios. The conclusion highlights the outcome of the paper and the outlook of the future work characterizes IMS/P2P client extensions for Quality of Service (QoS), remote administration and Digital Rights Management (DRM).

This paper is structured as follows. We first present the related work in section 2 and discuss our architectural design of a hybrid IMS/P2P client in section 3. The implementation is described in section 4. The conclusion in section 5 summarizes the paper and is followed by future work in section 5. Section 7 finally concludes the paper with the acknowledgements in section 8.

2. RELATED WORK

In this section we give a brief overview of open source implementations of P2P solutions for live streaming. The most promising and the most user-accepted solutions are presented and grouped by their underlying architecture, which is either a tree or mesh network.
SplitStream [6] is a Microsoft research project for high-bandwidth streaming using P2P content distribution which is built on top of the overlay network Pastry [7]. The key idea is to stripe the content, and distribute the stripes using separate multicast trees with disjoint interior nodes.

The open source project Tribler [8] of the P2PNext consortium extends the protocol of BitTorrent by adding live-streaming over real-time generated content. Tribler uses Give-To-Get (G2G) to handle the data exchange between peers and is based on mesh architecture.

PPLive is a proprietary protocol that belongs to the mesh based family. The use of two buffers enhances the playback smoothness, but extends the playback delay. Even though it is a closed system, it has been analyzed and evaluated in a recent paper [9].

The Data-driven Overlay Network DONet [10] is implemented as CoolStreaming and founded on mesh architecture. Information about the status of the distributed data is exchanged periodically.

Analyzes of the SopCast implementation in [11] determine a mesh architecture in the SopCast overlay. SopCast is a closed system and is available in the current version 3.0.3.
GridCast is a P2P live-video streaming solution and has been definitely used on the China Education and Research NETwork (CERNET) since May of 2006. A GridCast system comprises a tracker server, one or more source servers and a content index server. This is optimized by the exchange of data and an introduction of a low level self-organization of the peers.

The Fraunhofer MONSTER IMS client is an extendible plug-and-play framework that provides IMS application developers a unified communication interface on which they can develop and bring together a suite of integrated applications such as voice, video, instant messaging, chats, news, photo sharing, contacts and much more from the telecommunication and internet domains [12].

3. DESIGN OF NNodeTree

This section first motivates design decision of the presented overlay architecture, then points out the client side and concludes with its counterpart server side.

3.1. Overlay Architecture

The most common P2P streaming architectures are tree and mesh/swarm. Advantages of the tree architecture are its simple design, efficient push data forwarding with low latencies and low state message overhead. The main disadvantage is the vulnerability against dynamic peer behavior (churning) what damages the efficient overlay structure. The task of overlay planning can either be distributed (and handled by peers) or it can be realized centralized. A centralized overlay planning creates a bottle neck, but requires less management message overhead and enables a more efficient overlay restructuring. The advantages of the swarm are its robustness against churning, the ability of self-organizing and the ability to respect fairness between peers and their up- and download ratio. Main disadvantages are the delay to forward data and the higher control message overhead in contrast to the tree architecture.

The live streaming architecture presented in this paper should be able to distinguish between the individual peer performances. The tree architecture is selected, which is supported through an IMS Supper Application Server (AS). The AS optimizes the peer placement. Peers with higher performance (CPU, storage, bandwidth, energy, mobility, etc) are located closer to the source (inner overlay nodes) and peers with fewer capabilities are located as leafs of the overlay tree. Therefore locality information out of the IMS is used to create the tree overlay.

3.2. Client Side Multimedia Distribution

Each peer is able to play either the role of content consumer, content provider and optionally the content creator. Where the content provider inserts content into a P2P overlay the content consumer distributes and enjoys downloaded content.

CAPTURE or LOAD:

The objective of the capture or load stage is the transformation of a media object into an RTP stream. Media objects are either loaded from the file system or captured from a live stream by a webcam.

DEMULTIPLEX:

The segmentation of a continuous RTP stream into several sub streams is performed by the instance Demultiplexer.

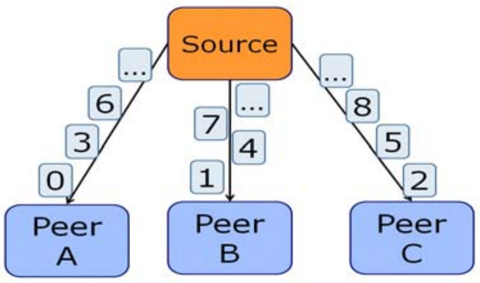


Figure 1: Distribution of Chunks

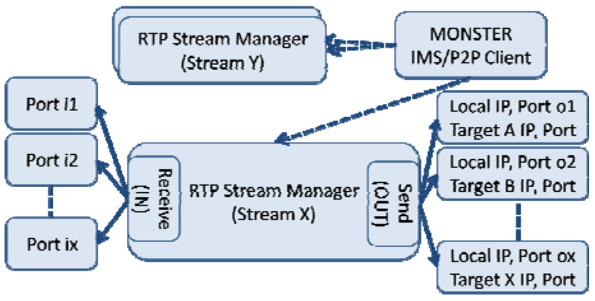

RTP Stream Manager (Stream Y)
MONSTER IMS/P2P Client
Port i1
Port i2
Port ix
Receive (IN)
RTP Stream Manager (Stream X)
Send (OUT)
Local IP, Port o1 Target A IP, Port
Local IP, Port o2 Target B IP, Port
Local IP, Port ox Target X IP, Port

Figure 2: RTP Streaming Manager

The source dispatches the first RTP frame in **Figure 1** to Peer A, the second to Peer B and the third to Peer C period of 3. A new period starts with the fourth piece which is send to Peer A again.

REASSEMBLE and DISTRIBUTE:

The objective of the current stage is to store the received RTP frames in a data structure that allows a non ordered insertion and retrieval. Thereby incoming streams of non continuous RTP packages can be stored and are interleaved to a complete continuous stream in the data structure. A replacement policy for frames is used to handle frames, which have been played or have been forwarded to other peers. The continuous RTP stream is addressed additionally to the local playback engine for play out.

The RTP Stream Manager design has two faces: The server side and the client side. The client side receives the media stream and the server side forwards the media stream to other peers. The sum of all bit rates from the incoming connections of the *RTP Stream Manager* is equal to the bit rate of the demanded media object. The bit rate of the media object in contrast to the outgoing bit rate is:

- Less, if no other peers needed to be served or the performance of the local peer doesn't allow upload
- Equal, if the same data rate is uploaded and downloaded
- More than the bit rate of the receiving media object, if the peer has a higher out-degree then its in-degree and the performance or network conditions allow a higher upload factor.

The RTP Stream Manager is depicted in **Figure 2**. One RTP Stream Manager is responsible for exactly one stream. A client can run multiple instances to participate in multiple overlays and retrieve different streams in parallel, what is shown with the dashed line. Each RTP Stream Manager has a given number of connections. Each connection to download (receive) data from another peer terminates at a specific local port. Outgoing connections to upload data to other peers, start at a local port and terminate remote at a given IP and Port combination.

The RTP Manager in **Figure 2** receives incoming data on the ports i1, i2 to ix and sends outgoing data from the local port o1 to the IP and port of target A, to mention only one of the depicted connections.

These incoming and outgoing data connections are dynamically controlled by a centralized instance, which may run on an application server (AS) placed in the IMS. The control information is send via SIP from the AS to an IMS client instance (UE). Therefore the UE needs to be registered in the IMS to be addressable with a SIP message. This message contains an identifying header line plus an XML body type containing the control information.

PLAYBACK

The playback engine plays this continuously RTP stream in a player window.

A new codec only needs a suitable adapter for capturing and playback in this design to be supported. The data transport is codec independent and don't need to be updated with the introduction of a new codec. This modular design achieves a maximal flexibility together with a minimal dependency of the workflow modules. Content security could be applied between the capturing and demultiplexing process plus between reassembling and playback.

3.3. Server Side Live Streaming Overlay Planning

The most important task of the P2P Support Application Server (AS) is the creation and maintenance of overlay(s). The overlay construction will be done with the knowledge about the user endpoints characteristics (performance, location, etc.), which are located in the Home Subscriber Server (HSS) and/or which are stored in the local user profile and are announced to the AS before participating.

Overlay Architecture

An IMS aimed P2P overlay creation is able to take peer characteristics into account to determine a set of neighbor peers that are grouped in parents and children. Connections to the parents where used to download data and the connections to the children are used to upload data.

The IMS based optimization allows the creation of an overlay in a tree structure, in which high performance peers are closer at the source (root) than low performance peers with less forwarding capabilities.

An IMS AS realizes this centralized task. Clients indicate their wish of participation into a specific overlay to enjoy the distributed content by sending a demand in form of a SIP message to a dedicated AS.

A header of this message indicates the intention to join or to leave the specific overlay. The AS determines whether the node is already present in the overlay and needs to be added or removed out of the overlay.

To better understand the structure of the overlay, we use the term node equal to the term peer at this point. A node in the following theoretical graph model represents a peer in the overlay as well as an UE in the physical network.

Thereby three groups of nodes are recognizable:

- Source node
- Inner nodes
- Leaf nodes

The source and all inner nodes in an overlay have the same degree (two, three, four, ..., eight, ...). The term degree is used to describe the physiology of the graph whereas the later more in detail discussed terms *out degree* and *in degree* describe the receiving/incoming and forwarding/outgoing degree what can be distinct.

The source has to distribute multimedia content to peers of the first level in the graph. Thereby multimedia content can be pure audio or video which can either be a static file or a live stream captured with a proper device.

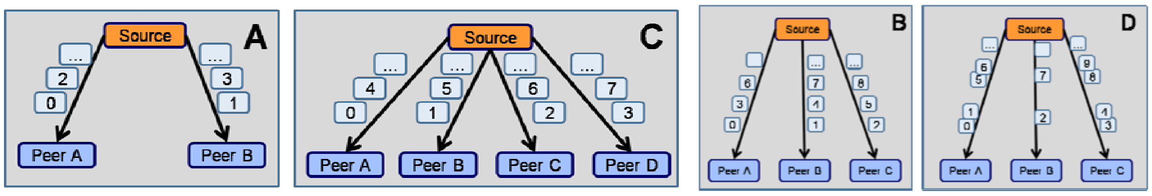


Figure 3: Media Source Distributes Content with different Degrees

Figure 3 depicts the source node with a degree of two (A), three (B) and four (C). Image (D) illustrates an unbalanced distribution by the source. The first three examples distribute the stream equally to all peers of the first level. Therefore a possible scenario could be a homogenous network. In a heterogonous network the peers would be served individually according to their performances. (D) depicts such a scenario where Peer A and Peer C receive each 40% whereas Peer B receives only 20%.

Overlay Creation and Data Forwarding

The algorithm that creates the media distribution overlay is now explained in detail. The first implementation of this algorithm covers a homogenous network and does not take peer characteristics into account.

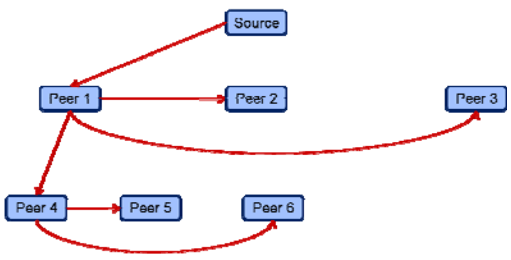


Figure 4: First Distribution Path

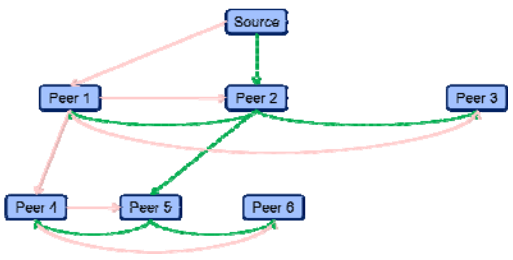


Figure 5: Second Distribution Path

The first distribution path (red/solid) in Figure **4** starts at the source and streams data to Peer 1, which stores a copy of the data local and forwards the incoming data on to Peer 2 and

Peer 3 of the second level, as well as to Peer 4 of the subset of Peer 1. Peer 4 handles the incoming data analogue and recursive to Peer 1, but with different Peers and on level below Peer 1.

The second path (green/dotted) in Figure 5 starts at the source, too and is first received by Peer 2, which forwards the data to Peer 1 and Peer 3 on the same level and down in the overlay graph to Peer 5, which distributes the data analogue and recursive to Peer 2.

The third path (blue/dashed) in Figure 6 starts at the source, too and is first received by Peer 3, which forwards the data to Peer 1 and Peer 3 on the same level and down in the overlay graph to Peer 6, which distributes the data analogue and recursively to Peer 3.

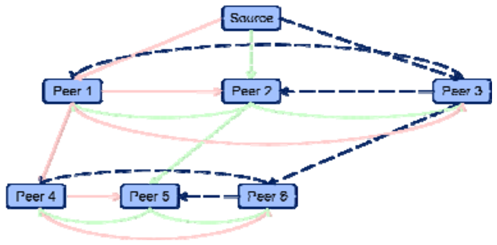


Figure 6: Third Distribution Path

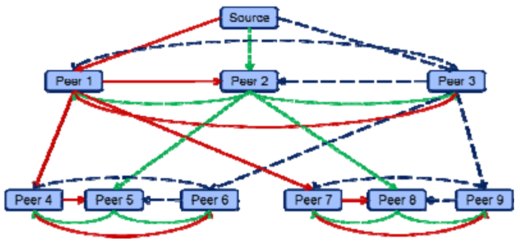


Figure 7: Two Full Subsets

The overlay of the previous examples has a full first level and a full first subset. A distribution with a second full subset is shown in Figure 7.

A distribution with a third full subset is shown in Figure 8.

New joining peers enter the tree overlay by recursively creating or joining new subsets. An example is shown in Figure 9 in which *Peer 13* (red highlighted) joins the overlay and is assigned by the P2P Support AS to use *Peer 4* as its root for a new created subset.

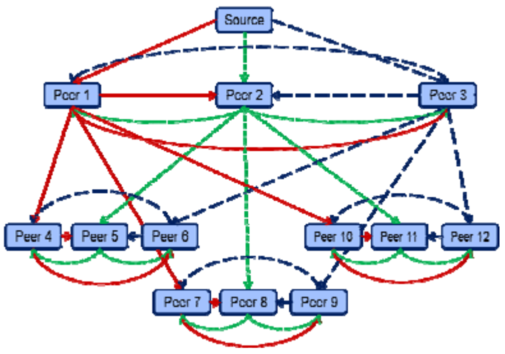


Figure 8: Three Full Subsets

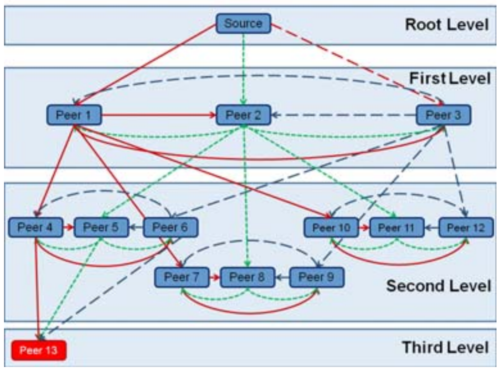


Figure 9: Leveled Tree Overlay

One can regard that only neighbored levels are connected. The first level and the third level are independent what points out the scalability of the designed architecture. The total size of the tree does only increase the connectivity of each participating peer up to a fixed limit, which depends on the degree of the node.

Overlay Maintenance

A scenario with an incomplete subset needs a changed forwarding rule. A leaving Peer 6 in Figure 10 leads to the fact that Peer 4 receives data from Peer 1 and Peer 3 direct and forwards both to Peer 5. Peer 5 in turn only forwards data of path to Peer 4. A joining Peer 6 would redirect the stream from Peer 3 to Peer 4 to a stream from Peer 3 to Peer 6. Peer 4 loses its forwarding tasks of this part of the stream and would receive this part now by Peer 6.

The churning of peers forces a quick rearrangement of the streaming overlay to maintain a continuous playback of all peers at all levels in the overlay graph. An incomplete subset contains less than the maximum of peers and requires a special strategy to ensure the reception of all sub streams. All Peers located in the same subset have to be informed about the churning of peers of their subset to adapt the outgoing forwarding connections. Peer 4 in Figure 10 still receives data at the same port independent of a sending Peer 3 or a sending Peer 6.

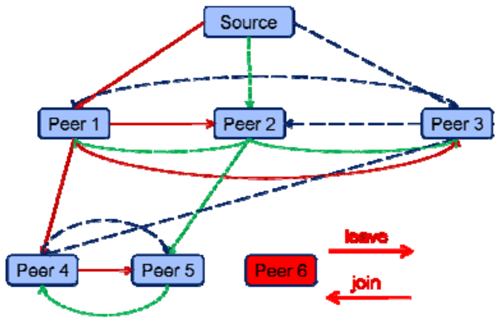


Figure 10: Incomplete Subset

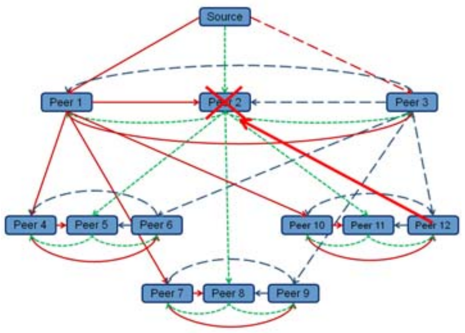


Figure 11: Overlay Maintenance and Peer Substitution

The churn of inner nodes results in a higher complexity to repair the overlay since these nodes have a higher connectivity. Thereby the replacement of a missing inner node in the overlay with the last insert node in the overlay is performed. The chapter evaluation proves the assumption mathematically that inner nodes have a higher connectivity than leaf nodes. A replacement of a missing inner node with a leaf node results in a minimal restructuring of the established paths. Figure 11 denotes the movement of *Peer 12* to the overlay position of the leaving Peer 2.

Overlay Management

The creation and maintenance of an overlay graph was explained so far. The current sub chapter covers the architecture to manage one or multiple overlays in parallel used to enable users the participation in one or multiple overlays at the same time.

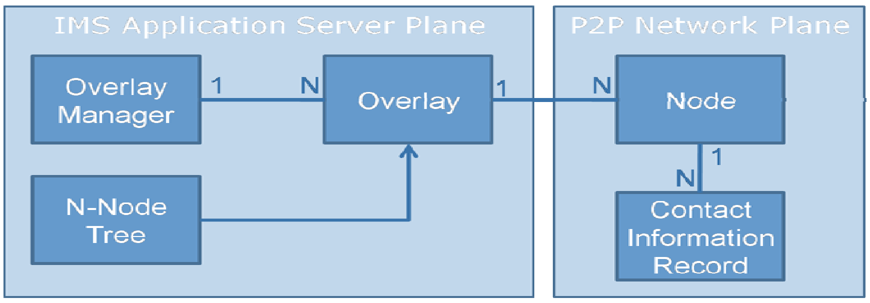


Figure 12: Overlay Management Architecture

The design depicted in Figure 12 illustrates the overall design of the management instance in an UML notation.

An *OverlayManagement* instance handles multiple overlays in parallel. Overlays are identified by a unique ID.

An overlay represents the generalization for an orchestration of nodes. The *NNodeTree* represents a specialized overlay in form of a tree with an arbitrary degree. A homogeneous network is designed.

Other specializations of the overlay are possible, but are not realized at the moment. Nodes could participate technically in each network.

A *ContactInformationRecord* contains the following parameters:

- IP: The IP address of the neighbor.
- Port: The port of the neighbor to address the stream.
- Part: The part of the stream that is exchanged.
- Forward: A flag indicating dataflow direction (forwarding, receiving).

A node manages its neighbors each in one *ContactInformationRecord*. An example XML representation is depicted in Table 1 that is used by an application server to address overlay positions to a peer. The XML document consists of three "node"-tags, which contain IP and port information of other peers of the same overlay, which send data to the local peer. The third tag contains an additional "forwarding"-tag that contains two peer IP and port combinations, to which the received data is forwarded.

```
<?xml version="1.0"?>
<overlayPositioning>
<node>
  <ip>192.168.1.4</ip><port>7000</port>
</node>
<node>
  <ip>192.168.1.5</ip><port>7003</port>
</node>
<node>
  <ip>192.168.1.3</ip><port>7009</port>
<forwarding>
<node>
  <ip>192.168.1.4</ip><port>7009</port>
</node>
<node>
  <ip>192.168.1.5</ip><port>7009</port>
  </node>
</forwarding>
</node>
</overlayPositioning>
```

Table 1: Overlay Positioning Information

The application server assigns overlay positions, when the overlay is filled with the minimum of participants. This number is equal to the degree of the nodes, to minimize the changes at the source.

4. IMPLEMENTATION OF NNodeTree

Existing P2P Streaming solutions are available under open source licenses; others are commercial but free of charge. None of the current solutions is suitable to coexist in combination to an IMS or seems to be adaptable for an IMS application server centric overlay planning.

4.1. Multimedia Frameworks

The currently most promising and widely used open source multimedia frameworks are:

- SUN Java Media Framework (JMF)
- GStreamer project
- VLC media player

Table 2 points out the differences between the frameworks, working on time-based media. They all work in the same manner, that a processing workflow is regarded as a chain of processes or a pipeline of arbitrary length.

FRAMEWORK	JMF	GESTREAMER	VLC
Language	Java	C	C/C++
Integration	Direct	Java bindings	
State	Outdated	Under development	
Licence	Open Source		

Table 2: Multimedia Frameworks Overview

These listed three major multimedia frameworks are all powerful frameworks. The Gstreamer and VLC projects are currently under development whereas JMF is outdated since 2003 with its current version 2.1.1.

JMF supports the video codec H.263_1998, which was designed as a low-bit rate compressed format for video conferencing with sparse movement of the acting persons. GStreamer and VLC instead support H.264, which is the most promising and established standard for video compression today, and is equivalent to MPEG-4 Part 10, or MPEG-4 AVC (for Advanced Video Coding). H.264 enables high data compression and allows additionally the use of a Scalable Video. The main idea behind H.264/SVC [13][14] is a codec that splits a media file into several layers with varying importance. A base layer is mandatory for low quality playback, which can be enhanced in combination with the other optionally layers to receive a higher playback quality.

A typical GStreamer installation offers a base set of essential plug-ins. GStreamer offers a broad variety of codec's. GStreamer codec's are grouped due to licenses and their stability into five categories: base, good, ugly, bad and ffmpeg.

All three presented frameworks work platform independent and needed to be installed and integrated into the system to be used. The gstreamer-java project [15] offers Java bindings to include GStreamer in a Java project.

The VLC project also provides Java bindings [16], supports a large number of multimedia formats and includes a free open source cross-platform media player.
The concept of a generic multimedia processing pipeline is depicted in Figure 13.

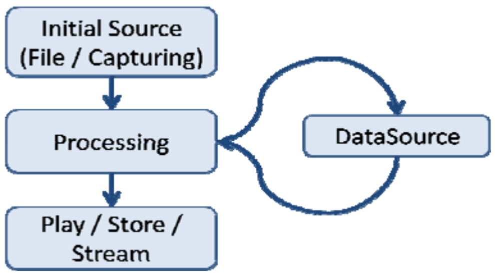


Figure 13: Concept of Multimedia Processing

An initial source, which is either a loaded file out of the local file system or a generated multimedia object (photo, sound or video) from a live capturing device like a webcam or microphone, is selected to be processed in the second stage. The overall term processing represents an instance to transform the chosen media object into another representation. For example a live video stream is transformed into an RTP Stream or a codec is applied to each frame of a stream. The stage of processing may be recursive, in case of the appliance of multiple codec's. Varying codec's, different filter, layers, formats or renderer can be applied to the media object in a single step. Finally the media object can be played out in a local media player, stored in the local file system or streamed to one or multiple destinations after all processing steps.

4.2. Client Side Live Streaming

This subsection aims at the implementation of the IMS/P2P client side. First the selection of the multimedia framework is explained, before the messaging and management are covered. Finally the implementation of the overlay positioning and the playback are discussed.

Multimedia Framework for Live Streaming

The JMF framework is selected as multimedia framework instead of the GStreamer or VLC framework, to meet a seamless integration into the MONSTER client, which is 100% pure Java and runs fluently on a Windows operating system. The components of JMF are extensible and allow flexible changes using generalization or by writing adapter classes.
A webcam live stream is captured with the class *VideoSourceRTPStream* which first creates a *processor* to process video data in a proper format and afterwards creates a *transmitter* to distribute video data via RTP.

Processor

The *DeviceManager* of JMF instantiates a *MediaLocator* for a plugged webcam. A *DataSource* is created out of the *MediaLocator* to individually push or pull data. The singleton *Manager* of JMF creates a *Processor* for the *DataSource* on which format parameters are applied (Encoding, dimension, data rate, etc.). The state full processor is realized and provides a *DataSource* output.

Transmitter

This *DataSource* is used to create an RTP packet stream which is addressed to the *SourceDemultiplexer*. An *RTPStreamManager* instantiates a *Session* containing the local address and data port together with the address and port of the *SourceMultiplexer*. The *RTPStreamManager* creates an object *SendStream* out of the *DataSource*. The *SendStream* controls the RTP packet stream with the methods *start()* and *stop()*.

The *SourceDemultiplexer* is initialized with overlay positions generated by an application server and dispatches the incoming RTP stream to multiple targets. The *SourceDemultiplexer* extends the interface *Thread* and runs non-blocking in parallel to other threads on the system. *ContactInformationRecords* are maintained in a synchronized *Vector* guaranteeing interruption and concurrency free read and write access for updating and changes.

RTPStreamManager

The *RTPStreamManager* represents the main instance on the live streaming plane at the client receiver side. It has to manage the receiving, storage, forwarding of incoming RTP packages at the same time. The local playback engine is provided with RTP packages through the *RTPStreamManager*, too.

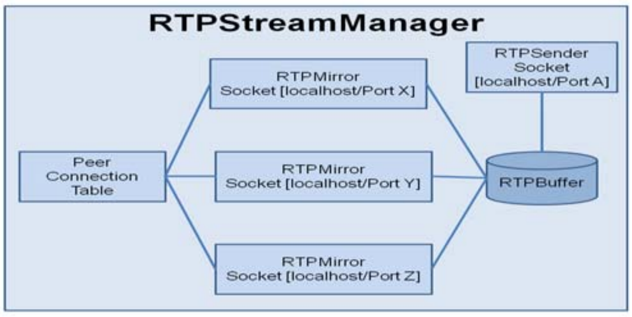

RTPStreamManager
RTPMirror Socket [localhost/Port X]
RTPSender Socket [localhost/Port A]
Peer Connection Table
RTPMirror Socket [localhost/Port Y]
RTPBuffer
RTPMirror Socket [localhost/Port Z]

Figure 14: Components of the RTPStreamManager

Figure 14 illustrates the architecture of this manager. It consists of several RTPMirrors, which collect and forward individual parts of the stream. The RTPMirror is later presented more in detail. All RTPMirrors are handled by the *RTPStreamManager* in a synchronized vector, which supports concurrent changes. The manager is implemented as *Thread* object to support seamless and non blocking execution in parallel with the other components. At the same time all RTPMirrors are linked with a *Peer Connection Table*, which is updated in case of an overlay position change. The *Peer Connection Table* lists all neighbors of the peer in the overlay. The vector is chosen as the adequate data type to store changing connection entries to manage the access in a concurrent environment. The *RTPBuffer* is realized as a synchronized *TreeMap* to order the scattered incoming packets on the one hand and to handle the garbage collection manually by removing played and forwarded RTPPackets to avoid running in out of memory exceptions on the other hand.

RTPMirror

The task of the *RTPMirror* is to receive incoming RTPPackets, to store them in the buffer and to forward them on to other peers. The *RTPMirror* is realized as *Thread* to work seamless and in parallel to other software components.

Each *RTPMirror* maintains a single socket at a given port on the local machine. A datagram socket is created to receive the RTPPackets in form of datagrams.

Another socket for outgoing packets is created to serve other remote peers in the overlay. Received packets are parsed to determine the *RTP Sequence Number*, which is used as key to store the packet in the buffer. The sequence number is located at byte 2 and 3 in the body of the datagram representing an RTPPacket. Each peer is responsible to distribute at least one part of the stream on to a set of other peers. Thereby at least a single *RTPMirror* forwards all incoming RTP packets on to this set. The *Peer Connection Table* maintains a list of endpoints, which needs to be served. The target IP address of the incoming datagram packet is modified and changed to the new receivers address. The changed packet is sent on the socket, which is opened up to serve other peers.

RTPSender

The main functionality of the *RTPSender* is the collection of RTP packets in a continuous order out of the RTP buffer, to send them to a local port, on which an RTP player engine listens.

The *RTPSender* is realized as a threaded object. The *RTPSender* is linked to the *RTPBuffer* and fetches sequentially RTP packets out of the *RTPBuffer*. The instance starts working at the point, at which a fixed number of RTP packets are present in the buffer. This blocking is required, to buffer packets and to ensure a fluent playback.

The transformation of the buffered RTP packets into an RTP stream, with the original time lag between the packets is solved by the *RTPSender*. Each RTP packet is parsed and the *Timestamp* is extracted. This parameter is located at the bytes 4, 5, 6 and 7 of the body in a datagram packet in the header of the included RTP packet. The current timestamp is compared with the previous timestamp and a possible time difference is waited, by letting the thread sleep.

After this pause – which may be null – the local IP address and a fixed port are written into the head of the packet. After this modification the packet is sent on to a local port, on which the player engine listens for incoming RTP packets.
The original RTP stream is reconstructed.

SIPMessageManager

SIP messages are used to transport information between peers and the AS which are stored in the SIP body or as parameter in additional header fields. The parameter *WorkerType* is used to indicate the task which has to be performed on the receiver side. The *Worker* is aligned on the software technique design pattern *COMMAND*. A *SIPMessageManager* handles incoming requests, parses the required header fields, instantiates the specific *Worker* and runs its execution method. Finally the *sendResponse()* method is called on the *Worker*, in case of a generated response object.

Reflections are used to provide a flexible and extensible design and to add new functionality at on a later point by adding only a new *WorkerType*. An abstract basis class *Worker* is created to provide the most important functions to all *Worker* objects to create a response message and to provide message attributes in the correct format.

Overlay Position Management

The *OverlayPositionWorker* extends the *Worker* and is instantiated and executed by the *SIPMessageManager* (0).

SIP message bodies containing XML overlay positions are parsed with the package *javax.xml* and *org.w3c.com*.

The *OverlayPositionWorker* has the two faces of a streamer and a receiver, which is determined through the header field *MarkedAsSource*, which is either marked as *TRUE* or *FALSE*.

A streamer (source) starts a *SourceDemultiplexer* with the new received overlay positions and starts the *VideoSourceRTPStream* (0).

A receiver starts a new or updates an already instantiated *RTPStreamManager* (0) with the new received overlay positions.

Playback

The playback is coordinated by the *Manager* object, which is listening on a defined local port for the incoming assembled RTP stream. The *AVReceive2* (0) class implements the *ReceiveStreamListener* and the *SessionListener* to handle events related to the RTP stream.

4.3. Server Side Live Streaming

This sub section covers the implementation of the server plane and the IMS configuration.

Overlay Management Service

This section highlights implementation details and functionalities of the main components of the overlay application server.

A P2P Support AS manages the data flow between peers.

The AS provides a SIP servlet that implements the Java Specification Request (JSR) 116. A SIP message with the request type MESSAGE is chosen to transport data along the signaling path bidirectional between the AS and the individual UE.
The Open Source SailFin [18] converged SIP and HTTP servlet container is installed in the IMS plane to deploy SIP servlets. A sip.xml enables a mapping of incoming requests to servlets. First servlet names are mapped to servlet class names and paths. A servlet mapping filters incoming requests with a pattern. In principle all requests are passed to each servlet. To minimize the effort and to filter only the required request types, a white list is created. This white list contains a pattern which in turn contains all allowed request types with expressed through Boolean algebra.

The servlet instance handles the incoming message with the specified doX() method. The *P2PSupportASSipServlet* extends the SipServlet and overwrites the doMessage() method to handle incoming messages.

The main task of the method doMessage() are:

- Parse the IP address.
- Parse the SIP-URI.
- Parse the target overlay identifier (ID).
- Create a new Node out of this information.
- Perform action at the overlay demanded by the UE.

P2PSupportASSipServlet

The *P2PSupportASSipServlet* instantiates an *OverlayManager* on startup, which handles several overlays in parallel. One initial test overlay is created on startup, too. The overlay creation by a remote peer is part of the future work.
A new arriving or a leaving peer in the overlay changes the state of the overlay and forces a reorganization of the overlay and its peers. The servlet iterates all peers of the overlay and re-assigns the overlay positions given by the overlay instance.

OverlayManager

The OverlayManager manages multiple overlays. They are distinguished by a unique id. Overlays can be removed and added. The manager provides the functionality to add to or remove from a particular overlay and to clarify the participation of an individual peer in an overlay.

Node

A *Node* is the internal logical representation of a UE or a peer by the overlay. A node is created by the *P2PSupportASSipServlet*, at the point at which a user sends a SIP request to the server. Such a node is identified by its SIP-URI and IP. Each node maintains two lists (in and out) of *ContactInformationRecords*. One list contains all neighbors that serve the peer with data and the other list contains neighbors in the overlay, which are served by the peer. These lists are maintained by the Overlay.

Overlay

The interface *Overlay* represents the super class, which has to be implemented by all other overlay types. It provides function to create a new overlay, add and remove a node, lists all participating nodes and determines the size of the overlay measured in the number of nodes.

NNodeTree

The idea of this new data type is, to enable the deployment of overlay tree with a variable degree, which is fixed in the initialization phase.

The class *NNodeTree* contains the logic to build up the overlay presented in section 3.3. It implements the super class *Overlay* and maintains all peers in an array. The source is stored at the first position, followed by all other peers. Array positions are used to compute neighborhood relations with the following equation:

$$parent(i) = \frac{i}{degree}$$

$$first\ child(i) = i * degree + 1$$
$$last\ child(i) = i * degree * i$$

The computation of the neighbor peers in the overlay is computed with the equation above. The parameter *i* represents the index of the peer in the array, which neighbors are computed.

The part of the stream is represented by the used port and is taken into account at the computation. These information are stored in *ContactInformationRecords*, which allow a dynamically modification of the in and out lists.

An inner node is removed out of the overlay and the nodes old position is filled with the last (leaf) peer of the overlay. This idea is motivated in the section 3.3, by pointing out that inner nodes have a higher connectivity in the overlay instead of leaf peers.

XMLTransformer

The *XMLTransformer* is used at the server side to create and on the client side to interpret XML overlay positions. The server side transforms a *ContactInformationRecord* into XML depicted in Table 1, which is sent to the peer by the servlet.

IMS Configuration

The AS needs to be addressed via SIP by a UE and therefore needed to be announced in the IMS to be addressable. The *service profile* named in the *public user identity* of the IMS lists *initial filter criteria*. Such filter criteria are mapping *trigger points* to application servers. A trigger point describes either with the conjunctive or with the disjunctive normal form a set of trigger points. The *P2PSupportASSipServlet* only requires the SIP message type MESSAGE.

5. VALIDATION OF NNodeTree

The first part of this section analysis the overlay planning algorithm mathematically. The live streaming scenario and test environment is presented in the second part of this section. The third part of this section presents measurements of a live streaming event.

5.1. Live Streaming Overlay Management Analysis

Two mathematical models of the presented overlay planning algorithm are presented in this section. On the one hand there is the degree of a node, which will be analyzed in the following sub section and on the other hand there is a bandwidth analysis, which can be derived out of the first model.

Degree Analysis

The analysis of the degree of inner and leaf nodes is mathematically modeled in a formula and additionally described in the following.

The previous section 3.3 already introduces the term degree as the number of outgoing connections from the source to peers of the first level. Whereas the source has a fixed out degree, inner and leaf nodes may have varying in and out degrees, which are located between a fixed minimum and maximum.

DEGREE	LEAF NODE	INNER NIODE	LEAF NODE	INNER NODE
IN-DEGREE	*degree*	*degree*	*degree*	*degree*
OUT-DEGREE	0	*Degree-1+1*	*Degree-1*	2* *degree-1*
	MINIMUM		MAXIMUM	

Table 3: In and Out Degree Evaluation of the Overlay

The row *IN DEGREE* of Table 3 evaluates the in-degree of inner and leaf nodes. Every peer in the overlay needs a complete stream at every point in time. Therefore the in degree has to be equal to the parameter *degree* to meet this requirement.

The row *OUT DEGREE* evaluates the out degree of leaf and inner nodes. A leaf node may have no outgoing connections and thus an out degree of 0, if this node is the only node in a subset. The maximum of degree - 1 of outgoing connections of a leaf node is reached in case of a full subset, which needs to be served.

The minimum out degree of an inner node consists of the outgoing connections to nodes located in the same level of the overlay graph plus the number of outgoing connections to subsets one level below the inner node. At least one subset is required below a node to be an inner node and at most |*degree*| subsets are possible in an overlay.

Bandwidth Segmentation Analysis

The bandwidth utilization caused by a peer depends on its *degree* and on the *number of peers* in the overlay counted in subsets. This analysis affects only inner nodes and not the leaf nodes, since leaf nodes may have a lower connectivity in the overlay graph.

$$b = [m + (n - 1) + (n - 1)] * s$$
$$m = 1 < m < n+1$$

The equation above calculates the expected bandwidth utilization b for a peer in an overlay with the degree of n. The term m describes the numbers of all involved subsets, which is limited and the term s represents the data rate of a single stream.

The required bandwidth consists of the following four terms, which are added.

- The number of outgoing streams to involved subsets (m) in the distribution of the peer. This is minimally the own subset of the peer and maximally the own stream plus *degree* other subsets one level below.
- The number of outgoing streams to neighbor peers in the same subset at the same level.
- The number of outgoing streams to subsets one level below, which are *zero* or at most *degree*.
- The data rate of a single stream, which is equal to all other parts of the stream in this scenario.
-

Degree(n)/ Subset m	PART 1	PART 2	PART n	SUM
m=1	m+(n-1)=3	1	1	m+(n-1)+(n-1)=5
m=2	m+(n-1)=4	1	1	m+(n-1)+(n-1)=6
m=3	m+(n-1)=5	1	1	m+(n-1)+(n-1)=7
m=n+1	m+(n-1)=6	1	1	m+(n-1)+(n-1)=8

Table 4: Bandwidth Segmentation of Different Parts of the Stream

The Table 4 illustrates a mathematical model of the required bandwidth of a single peer. It is exemplarily filled with values referring to a node degree (n) of n=3. Part 1 of the stream is selected to be the part of the stream, which is forwarded to others. Part 2 to Part n are only received and processed by the peer but not forwarded.

5.2. Demo Scenario

The demo environment for live streaming is a wall consists of 16 tablet PCs depicted in Figure 15. All tablets are running with Windows XP, have a Java Media Framework installation and are connected through a centralized Ethernet switch. An IMS core running on a separate server is connected to this switch, too.

All instances are installed in the same local address space.

The Open IMS Core is an open source implementation of IMS Call Session Control Functions (CSCFs) and a light weight HSS developed by Fraunhofer FOKUS and offered to the academia and industry for developing own IMS components or IMS applications.

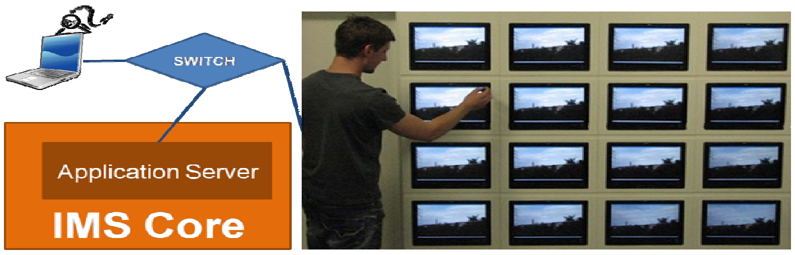


Figure 15: Tablet Wall Demo Setup

The application server was initially first Fraunhofer FOKUS SIPSEE [17] converged servlet container, which was substituted by the SailFin [18] converged servlet container. The SailFin open source project supports the latest IMS servlet specifications and offers further developments and patches. The Java based SailFin project is installed in the application server with the standard setup and without modifications.

5.3. Demo measurements

The measurement in a live streaming scenario is manifold. Measurements may be taken at the source or at endpoints (peers). The measurement is taken exemplarily at one peer of the first level of the overlay graph, to see the maximum of changes in this dynamically changing overlay. The overlay management algorithm presented in chapter 3 explained the overlay construction in detail.

The demo scenario starts with a creation of an empty overlay at the application server, which is instantiated automatically on startup and is controlled through the remote SSH tool Putty. The peers of the overlay have a specific degree, which can be defined in the initialization phase of the overlay. This demo scenario selects a degree of 3, in order to construct an overlay with up to three levels and 17 peers. A laptop with a plugged webcam joins the new created overlay in the role of a live video source.

All of the 16 tablet PCs join the overlay one after another in a random order at the next step of the demo scenario. Finally an overlay consisting 16 tablet PCs and one laptop acting as source stream provider is created. After all 17 peers have successfully joined the overlay; all peers are taken out of the overlay in random order by announcing their leaving individually to the application server.

Figure 16 depicts the complete bandwidth measurement divided into time interval of this demo scenario measured on a peer of the first level.

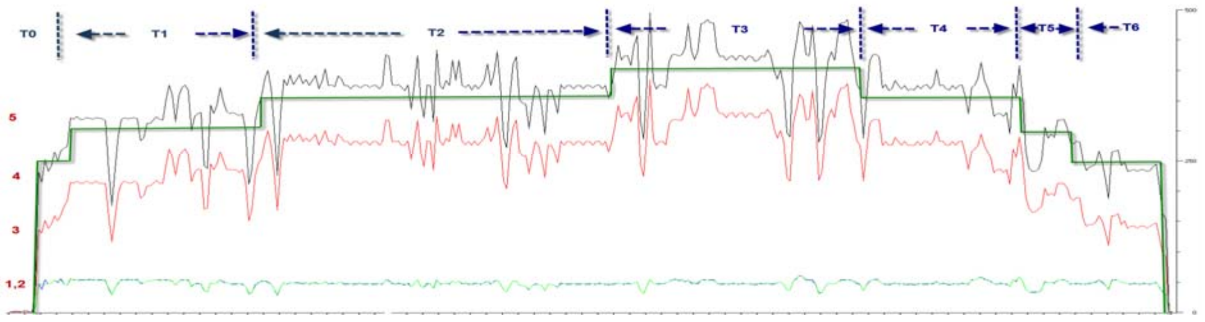


Figure 16: Live Streaming Bandwidth Evaluation

Figure 16 illustrates the measurements of the demo scenario. The figures consist of four graphs, displayed in a coordinate system having the time at the X-axis and the data rate at the Y-axis.

- The green line (1) at the bottom indicates part one of the stream, which is received from a neighbor peer.
- The blue line (2) at the bottom indicates part two of the stream, which is received from a neighbor peer.
- The red line (3) in the middle indicates part three of the stream, which is received from the source and is distributed twice to the neighbor peers.
- The black line (5) at the top indicates the sum of all parts of the incoming and outgoing streams.

The complete demo scenario is divided into time slots T0 to T6. Each of them represents a specific phase of the distribution.

Interval T0

The first time slot T0 indicates the setup phase of the overlay in which the first three participating peers join the overlay and receive the stream. The time 0 to 10 seconds indicates the setup phase. The first three peers joined the overlay and the source starts sending the captured live stream at approximately 10 seconds and marked by the left two arrows. The first three peers receive an individual part of the stream directly and forward the missing parts.

Line 1 and 2 illustrate the two incoming streams of the neighboring nodes, which are only processed locally and not forwarded to other peers in the overlay. Each of them has a data rate of approximately 50 kbit/s, maybe slightly more, what can be metered at the two single streams at line 1 and 2. The data rate indicated by line 3 is three time higher than the one at line 1 and line 2, because it consists of three streams. The first is received from the source (incoming) and two other streams are sent to the neighbors of the same level (outgoing).

The data rate of the original stream is about 150 kbit/s, which is summed up by three different parts of the stream.

Line 5 depicts the total bandwidth, which the peer is exhausting. Three incoming and two outgoing parts of the stream each having a data rate of about 50 kbit/s is summed up to a total data rate of 250 kbit/s, which is indicated at the scale at the right.

The second arrow at line 3 points out the increased bandwidth, caused by the additional forwarding costs of the new joining peers in the overlay, which is needed to be served with part 3 of the stream. The organization and forwarding is established between 10s and 15s and completed afterwards.

The first level of the overlay is filled completely and no peers are present in the second level of the overlay at this moment.

Interval T1

The first one of the next three peers joins the overlay in the beginning of the time slot T1 between 15 and 80 seconds. The first of three subsets of the second level is filled with peers in this period.

One stream is received by the source and the same part is distributed three times by the monitored client at this point, each with approximately 50kbit/s. The part is sent twice to the neighboring peers and once to a subset.

The other two parts of the stream are received from the two neighbor peers by the client. The total bandwidth utilization reaches a point with 300kbit/s in the phase T3, caused by these six parts.

Interval T2

The next three peers join the overlay in the time slot T2 between 80 and 190 seconds. The first and the second of three subsets are filled with peers at the end of this time slot.
The data rate is increased by the new link, which serves the second subset with part 3 of the stream.

One stream is received by the source and the same part is distributed five times by the monitored client at this point, each with approximately 50kbit/s. The part is sent twice to the neighboring peers and to each of the two subsets once.
The other two parts of the stream are received from the two neighbor peers by the client. The total bandwidth utilization reaches a point with 350kbit/s in the phase T2, caused by these seven parts.

Interval T3

The next three peers join the overlay in the time slot T3 between 190 and 270 seconds and allocate the third subset - one after the other. All three subsets are filled with peers at the end of this time slot.

One stream is received by the source and the same part is distributed five times by the monitored client at this point, each with approximately 50kbit/s. The part is sent twice to the neighboring peers and to each of the three subsets once.
The other two parts of the stream are received from the two neighbor peers by the client. The total bandwidth utilization reaches its highest possible point with 400kbit/s in the phase T3, caused by these eight parts.

At the end of this phase, all 16 tablet PCs and the source are now participating in the overlay. The overlay consists of the following subsets:

- The source is on top of the overlay.
- The first level contains one subsets with three nodes (Peer 1 to Peer 3).

- The second level contains three subset each with three nodes (Peer 4 to Peer 12).
- The third level contains two subsets. The first subset with three nodes (Peer 13 to Peer 15) and a second with only one node (Peer 16).

Interval T4

The phase T4 starts at 270 and ends at 320 seconds. The Peers are not leaving in the same order as they have joined the overlay, what leads to a reorganization of the overlay and changes the communication partners. Such reorganization phases are in the first part of phase T4 at 245, 255 and 270 seconds, in which the data flow shortly collapses and recovers quickly.

The bandwidth utilization of 350 kbit/s can be measured, which is equal to phase T2 and has also the same number of peers and the same number of subsets in the overlay.

Interval T5

The phase T5 starts at 320 and ends at 355 seconds and contains only two full subsets with six Peers.

Interval T6

The phase T6 starts at 355 and ends at 365 seconds and contains only one full subset with three Peers in it. The stream is stopped at this point and the bandwidth utilization decreases to zero.

5.4. Evaluation

This section covers the live streaming evaluation first in a theoretically mathematical model and later in a practical demo scenario. The presented formula in Table 4 is applicable in every part of the measurements T0 to T6.

The bandwidth requirement of the peer at e.g. T3 was measured with approximately 400 kbit/s. At this point three subsets of the second level are filled. The formula in Table 4 with the values m=4 and n=3 calculates a factor 8, which needs to be multiplied with the data rate of a single part = 50kbit/s. The resulting total bandwidth is calculated with 400kbits/s.
A total bandwidth of 400 kbit/s was calculated and approximately measured.

Figure 16 illustrates the differences of the expected total bandwidth (green solid line) and the total measured bandwidth (black line). The expected line has accurately defined steps, where the black has variations in it. The variations of the curves of these two figures might have several reasons.

The source is connected through WLAN with the network of the demo scenario. WLAN is a shared medium and differ in the available bandwidth, in case of multiple sending participants.

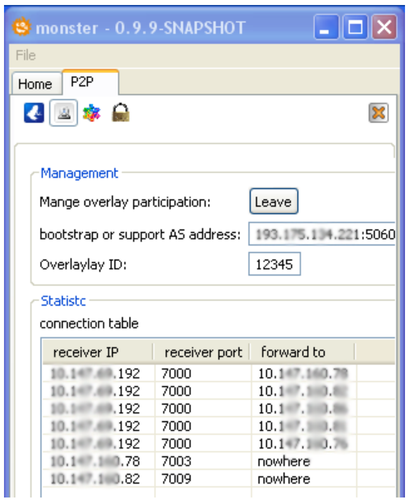
All 16 tablet PCs and the source are connected through one switch in the background. This bottleneck might be responsible for bandwidth variations at the peers, too.
The application server (AS) located in the IMS plane forces reorganizations at all peers. SIP messages are sent to all of the peers sequentially, what leads to minimal variations in the time of reception. Changes in the overlay result in changes of the synchronized internal routing

tables in the *RTP Streaming Manager*, what causes short and radical variations of the data flow.

The mpeg RTP codec of JMF produces small bandwidth variations while processing the web cam stream and providing the RTP stream.

The small gaps (at 30s, 75s and 85s) in the lines of stream part one and two at the bottom can be found as big gaps in the line of the total bandwidth. The line of the total bandwidth is an amplification of one incoming stream part with the number of the forwarding stream parts. Bottlenecks in the WLAN connection between the IMS and the source laptop are most likely for the variations.

The Figure 17 shows the IMS/P2P client after joining the overlay. The IP address and Port of the application server are stated in their proper input fields. A mouse click on the button *leave* lets the client exit the overlay. The AS assigned an overlay position to the Peer. A connection state table lists outgoing and incoming connections on different Ports. The depicted Peer received on Port 7000 a part of the stream by a Peer with the IP 10.147.69.169 address and forwards the part on to five other Peers (X.Y.Z.[78, 82,86,81 and 76]). The Peers X.Y.Z.[78 and 82] are sending different parts of the stream to different Ports at the Peer.



Management

Mange overlay participation: Leave

bootstrap or support AS address: 193.1[redacted].221:5060

Overlaylay ID: 12345

Statistc

connection table

receiver IP	receiver port	forward to
10.1[redacted].192	7000	10.1[redacted]
10.1[redacted].192	7000	10.1[redacted]
10.1[redacted].192	7000	10.1[redacted]
10.1[redacted].192	7000	10.1[redacted]
10.1[redacted].192	7000	10.1[redacted]
10.1[redacted].78	7003	nowhere
10.1[redacted].82	7009	nowhere

Figure 17: MONSTER addon-P2P Live Streaming

The demo scenario points out the scalability and efficiency of the *NNodeTree* algorithm. The maximum bandwidth utilization for the monitored peer was reached in the interval T3 and was not growing further even in a growing overlay, because the maximal out degree of the peer was reached.

6. CONCLUSION

The work described in this paper presents a tree based P2P live streaming overlay architecture, which enables scalable live streaming with hybrid IMS/P2P clients . It combines IMS with P2P and takes advantages out of one system to eliminate disadvantages of the other system.

We realized our live stream overlay architecture in form of a tree, to ensure the integration of the envisaged additional tasks QoS, location awareness and DRM.

The paper first presents the design, implementation and evaluation of a tree based overlay architecture. The theoretically approach is practically implemented in a hybrid IMS/P2P client with live streaming functionalities. Such a hybrid IMS/P2P client is supported through new developed P2P Support Application Server located in the IMS, which creates and manages optimized overlay networks. An extensive testing validates and proofs our new developed software prototype in different service scenarios.

7. FUTURE WORK

This section presents an outlook and future work of our paper. The future work consists of the fields Quality of Service, DRM and topology awareness.
Topology awareness of peers in the overlay has not been taken into account in this thesis. The design of the P2P support application server supports new overlay types, which enable topology awareness. Existing solutions for creating a topology aware overlay are applied on the application server.

The presented content distribution does not include Digital Rights Management (DRM) at this moment. A next step is the introduction of existing DRM solutions. The performance of the system will be decreased, when dealing with encrypted content, because of the decryption process on the source side and the encryption process on the receiver side. Different existing DRM systems will be applied to measure and evaluate the stream performance.

The application server will be extended to collect statistics of the peers, which can be used for charging.

The charging of peers will be enabled with the introduction of DRM, what enables new business models and increases the field of potential customers.

An aspiring improvement would be to add Quality of Service (QoS) to the P2P network. SIP signaling can be used to assign a fixed and guarantied bandwidth between two peers in a desired direction.

Acknowledgement

Many thanks go to the MONSTER developer team at the group for Next Generation Networks Infrastructures (NGNI) at the Fraunhofer FOKUS Institute in Berlin, Germany for inspiring discussions.

The work presented in this paper is related to the ongoing EU project VITAL++. VITAL++ is a Specific Targeted Research Project (STREP) supported by the European 7th Framework Programme, Contract number ICT-2-1.6-224287, Project starting date 1st June 2008 (duration 30 months).

References

[1] Klaus Mochalski, Hendrik Schulze. Ipoque internet study 2008 2009

[2] TS 23.228, "IP multimedia subsystem (IMS)", 3GPP, 2006.

[3] J. Fiedler, T. Magedanz, A. Menendez: "IMS secured content delivery over peer-to-peer networks", Proceedings of SIGMAP 2007, Spain, July 28-31, 2007, INSTICC Press, Portugal, p. 5-12, ISBN 978-989-8111-13-5Tavel, P. 2007.

[4] Julius Müller J. Fiedler, T. Magedanz. Extending an ims client with peer-to-peer content delivery. Second ICST International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications - MOBILWARE 2009, Berlin, Germany, April 29, 2009, www.mobilware.org, pages 197–207, 2009.

[5] Julius Mueller, Diploma Thesis, "Design & Implementation of a Scalable End-to-End Multimedia Distribution Architecture for Next-Generation-Networks", Fraunhofer FOKUS, Oct. 2009

[6] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment", SOSP'03, Lake Bolton, New York, October, 2003.

[7] Antony Rowstron, Peter Druschel,"Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems", Middleware 2001 : IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001. Proceedings (2001), pp. 329-350.

[8] P2p-next website, http://www.p2p-next.org/, Last visited in March, 2009

[9] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu and Keith Ross,"A Measurement Study of a Large-Scale P2P IPTV System", in IEEE Transactions on Multimedia, Volume 9, Number 8, December, 2007

[10] Donet/coolstreaming: A data-driven overlay network for live media streaming – Zhang, Liu, et al. – 2005

[11] Will IPTV ride the peer-to-peer stream? [Peer-to-Peer Multimedia Streaming],A Sentinelli, G Marfia, M Gerla, L Kleinrock, S Tewari, Communications Magazine, IEEE, Vol. 45, No. 6. (2007), pp. 86-92.

[12] myMONSTER - Telco Communicator Suite, www.monster-the-client.org

[13] R Kurceren M Karczewicz. The SP- and SI-frames design for h.264/avc. Circuits and Systems for Video Technology, IEEE Transactions, 7:637–644, 2003.

[14] Thomas Wiegand Heiko Schwarz, Detlev Marpe. Overview of the scalable video coding extension of the h.264/avc standard. IEEE Trans. Circuits Syst. Video Techn. 17(9): 1103-1120 (2007).

[15] GStreamer Framework. http://www.gstreamer.net/.

[16] Java VLC. http://trac.videolan.org/jvlc/.

[17] T.Magedanz, E.Fasel, K.Knuettel. The fokus open sip as - aservice platform for ngn. Kommunikation in Verteilten Systemen (KiVS) : Kurzbeiträge und Workshop der 14. GI/ITGFachtagung, 28. Februar - 3. März 2005 in Kaiserslautern Bonn: GI, 2005 GIEdition ISBN: 3-88579-390-3 S.49-56.

[18] Project SailFin. https://sailfin.dev.java.net/.