University of Zurich
Department of Informatics

David Hausheer
Thomas Bocek
Fabio Victora Hecht
Cristian Morariu
Gregor Schaffrath
Burkhard Stiller
(Eds.)

P2P Challenge Task 2008

TECHNICAL REPORT – No. ifi-2008.07

July 2008

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

D. Hausheer, T. Bocek, F. V. Hecht, C. Morariu, G. Schaffrath, B. Stiller (Eds.):
Technical Report No. ifi-2008.07, July 2008
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch/

P2P Challenge Task 2008

Introduction

The Department of Informatics, IFI runs a lecture with exercises, termed "Peer-to-peer Systems and Applications". This combination allows for the introduction of major concepts in the P2P domain and at the same time its practical use in terms of design as well as coding steps, which are flanked by a number of theoretical exercises. Thus, this report documents the practical task description as well as the set of different solutions developed by those four groups of students in the spring term of 2008.

Peer-to-peer (P2P) overlay networking concepts have been investigated for the last couple of years and are now becoming mature enough to be used in a wide range of applications. Besides the traditional file sharing systems, for which P2P overlay concepts have initially become popular, new applications like P2P streaming, distributed storage, and many other applications start to adopt P2P networking principles in order to become more scalable and more robust against failures and changes in the network.

The EU-ICT project SmoothIT currently investigates how the overall benefit of P2P overlay users and Internet service providers (ISPs) can be maximized, by improving the performance of P2P overlay applications while minimizing the costs incurred by the ISP. One of the investigated scenarios in SmoothIT is the streaming of multimedia content over a P2P overlay network. Under the assumption that a peer may select its streaming source among multiple peers, some of these peers may be more beneficial for the ISP, *e.g.*, if they are located within the same domain. At the same time, such a choice may also be beneficial for the requesting peer, because it may get a better performance from a close peer.

Content

The “P2P challenge task 2008” is the third edition of a practical student exercise carried out as a competition among several groups of students in the scope of the lecture on “P2P Systems and Applications”. The main goal of the challenge task is that the students get hands-on experience in applying P2P overlay concepts presented in the lecture. In this year’s challenge task the students were asked to design and prototypically implement a P2P streaming application as introduced above. The solution had to provide time-shift functionality for a live video stream. The time-shift functionality allows a live stream viewer to pause the reception of the stream and to replay particular parts of the stream. The solution had to enable the user to access the stream in realtime and to switch dynamically from live streaming to time shifted streaming and back. Besides the use of P2P mechanisms to establish a fully decentralized structured P2P overlay network, the solution to be developed had to enable each peer to serve multiple clients at the same time. Moreover, the solution had to allow for the storage of at least two continuous hours of time shifting and had to be robust against node or link failure during timeshift.

The challenge task lasted eleven weeks. The students worked on the task in a network laboratory once a week for about two hours, during which they were supported by their supervisors. They were given an application template and could use several libraries and tools such as the FreePas-

try P2P overlay technology and the VideoLAN client (VLC) to play the multimedia stream. Other tools than the ones given, were not allowed and the application had to be implemented in Java.

The students have chosen quite different approaches to solve the problem they were given. Group 1 applied an active distribution concept, while the others used passive distribution. With an active distribution, one peer watching and receiving the stream, forwards it actively to those peers that are supposed to store it. The benefit of this approach is a better balance of the distribution for unpopular channels. However, the drawback is an increased bandwidth usage. In the passive distribution mode, a stream is only recorded if watched. Thus, only few peers record unpopular channels.

The approach taken by group 2 is based on the selective storage of five seconds video chunks on the peers. Whenever a peer wants to watch a replay of the stream, it calculates the chunk-IDs of the required video chunks based on the desired time frame. Based on those IDs the video chunks are retrieved from the P2P network and replayed locally.

Group 3 implemented the timeshift functionality by using a distributed index stored in a PAST network. A respective index entry is created for every buffered slice, to which nodes buffering the respective content append their host information. Lookups happen in PAST and the responsible nodes are queried directly for content.

Finally, group 4 took an approach similar to group 2. Each peer receiving the video stream divides it into fixed-size segments. A segment is stored by a peer if the hash of the segment identifier - its starting timestamp - lies between the node's own NodeID and the NodeID of the Nth neighbor, in the FreePastry ID space. N is a parameter with a common value of 2. After recording a segment, a peer adds a new entry in the FreePastry DHT to announce its availability.

At the end of the challenge task, the solution had to be presented and demonstrated. A testbed infrastructure was available which included 10 Linux PCs. During the demonstration an arbitrary node was disconnected from the network, in order to challenge the robustness of the solutions. After the demo each group was evaluated by the other groups based on several criterias as laid out in the requirements of the application. Based on these evaluation results the winner group was determined. The "P2P Challenge Champion 2008" award was finally given to Group 2 with Marco Kessler, Stefan Christiani, Roger Peyer, and Konstantin Benz. Congratulations!

The code and further documentation in terms of slides can be downloaded from
http://www.csg.uzh.ch/publications/software/p2p-timeshift

Zürich, July 2008

Contents

(This page is left blank intentionally.)

University of Zurich
Department of Informatics

ifi

P2P Challenge Task 2008

Task Description

Authors:

Thomas Bocek, Fabio Hecht, Cristian Morariu,
Gregor Schaffrath, David Hausheer

Spring Term 2008

P2P Challenge Task 2008

Thomas Bocek, Fabio Hecht, Cristian Morariu, Gregor Schaffrath, David Hausheer

Spring Term 2008

Task Description

The P2P Challenge Task for the FS08 semester is to design and prototypically implement time-shift functionality for a live video stream using P2P concepts. The time-shift functionality shall allow a live stream viewer to pause the reception of the stream and shall also allow the replay of particular parts of the stream.

The nodes in the P2P network represent individual users watching a particular TV Channel (video stream). Storing locally the whole data received by each peer individually is an inefficient way of designing a time-shift function, as each peer will have to store the whole content. Another disadvantage of a "local" solution for time-shift is that a user may only replay parts that were broadcast while his application was running (e.g. if a user would start the application just after a team scored during a football game he would not be able to replay the goal). A P2P-based approach addresses these issues by distributing the task of recording and storage of the video stream across multiple nodes. Each node is responsible of recording and storage for particular parts of the video stream. In this case a replay task consists of finding the peers responsible with the recording for the requested timeslots and getting the video stream from them.

The main goal of the challenge task is to design and implement a P2P-based recording, storage, and replay application for live video streams. Each node in the P2P network shall receive the video stream as UDP packets (the live stream transmission is provided). A VideoLan Client (VLC) shall be used as a video player on each node. Your time-shift application should be able to do the following:

- receive the incoming UDP stream from the video server
- buffer the live stream and send it to VLC via UDP
- store parts of the live stream for 7200 seconds (2 hours)
- request the P2P network for video replay for maximum 7200 seconds (2 hours)

Application Requirements

- Live/Timeshift switch: The solution shall enable the user to switch dynamically from live streaming to time shifted streaming and back.

- Realtime access: Stream buffering and access for time shifted streams shall happen in real-time.
- Timeshift period: The solution shall allow for the storage of at least two continuous hours of time shifting.
- Robustness: The solution shall be robust against node or link failure during timeshift.
- Multiple Client Serving: The solution shall enable each peer to serve multiple clients at the same time, if it has sufficient bandwidth to do so.
- P2P mechanisms: The solution shall be based on Peer-to-Peer mechanisms using a structured overlay network.
- Decentralization: The solution shall not contain any central elements.
- Compatibility: The solution shall be executable on the provided test setup machines (Linux, Java 1.6).
- Library and tools: The solution shall be based on libraries and tools which allow for publishing under GPL or another comparable open software license.
- Application report: The application report shall document the application. The report shall have 5-10 pages.
- Note: Depending on the development of the challenge task, further requirements and/or tools may be added, if necessary.

Organization

- The groups shall be balanced. Every group shall have at least one Java expert. During the challenge task, the group shall meet every week during exercise hours to work on the task and discuss the next steps.
- Distribute the workload (P2P load balancing) so that each peer gets a fair load of work.
- You can bring your own laptop and/or use computers in room BIN 1.D.12.

Milestones

The challenge task includes three milestones on which specific deliverables have to be handed-in to the supervisors:

- Milestone 1: Detailed workplan including task distribution among group members, ToC of the report, Meeting with the supervisor
- Milestone 2: Draft code (Step 1 + 2, see below) and draft report
- Milestone 3: Final code (Step 3, see below) and final report (5-10 pages, including design, structure of code, and installation requirements)

Table 1 shows the timeplan for the challenge task with the different milestones.

Table 1: Timeplan

Challenge Task Milestones	Dates
Grouping, Task Presentation	13.03.2008
Work	20.03.2008
Easter Holidays	27.03.2008
Milestone 1	03.04.2008
Work	10.04.2008
Work	17.04.2008
Milestone 2	24.04.2008
Work	08.05.2008
Work	15.05.2008
Milestone 3	22.05.2008
Presentations and Demos	29.05.2008

Support

During the challenge task each group will be able to ask questions and get support for the task from their supervisor:

- Group 1: Thomas Bocek
- Group 2: Cristian Morariu
- Group 3: Gregor Schaffrath
- Group 4: Fabio Hecht

Libraries and Tools

- Use the Java as programming language. You can use the latest version, as this has usually more useful features than older versions. The J2SE Software Development Kit (SDK) can be downloaded under http://java.sun.com/javase/downloads/.
- The assistants do know Eclipse well and the use of Eclipse is highly recommended. The most recent release of Eclipse can be downloaded at http://www.eclipse.org/downloads/. Eclipse is available for Windows XP, Mac OSX, Linux and further platforms. If you prefer other IDEs you may use them but without support from our side.
- The use of FreePastry (Version 2.0_03) as P2P overlay technology is highly recommended. FreePastry is an open source implementation of Pastry. The most recent Binary (JAR) version of FreePastry can be downloaded under http://freepastry.org/FreePastry/FreePastry-2.0_03.jar. Alternatively, you may choose other overlay networks. Thomas Bocek has implemented TomP2P, a variant of the Kademlia protocol. The key benefit of using TomP2P is the

support he can provide. For more information, do not hesitate to contact him if you plan to use this software.

- Use of the Template which serves as a basis for the implementation of the P2P application.
- The use of VLC as a media player is recommended due to its flexibility and availability for Linux and Windows (http://djproject.sourceforge.net/ns/).
- Additionally, the example application shown in the lecture is available as well.

Steps

In order to successfully achieve your goal, the following steps are recommended.

- Start by understanding how FreePastry works, what P2P methodology is used, how nodes are accessed, and how nodes can be found.
- Implement a simple example. This example does not need to be part of the solution. The goal is to get used to the API and see how the Framework behaves.
- Understand how VLC works and how you can use it in this particular case.

We recommend the following steps during the implementation task:

- Step 1: Design your application: specify use case(s), functional requirements, modules. Plan how nodes will communicate. Draw sequence diagrams for communication among nodes and inside nodes, among components. Build GUI prototype.
- Step 2: Handle video content. At a peer level, implement receiving of video stream from a fixed server, buffering of contents in memory or disk (if necessary). Design protocol(s) for communication between peers using the P2P Network - in order to find peers to download/ upload data to/from and to establish connection for video.
- Step 3: Deal with the network. Implement protocols designed in step 2. Test and debug your application.

The tools to be used are generally well documented. Find below a few links which provide a good overview:

- Java Tutorial (http://java.sun.com/docs/books/tutorial/)
- Java ist auch eine Insel (http://www.galileocomputing.de/openbook/javainsel6/)
- Java API (http://java.sun.com/j2se/1.4.2/docs/api/)
- Eclipse Dokumentation (http://help.eclipse.org/help31/index.jsp)
- FreePastry Javadocs (http://freepastry.org/FreePastry/javadoc/index.html)
- FreePastry Readme (http://freepastry.org/FreePastry/README-1.4.4.html)
- Pastry Overview (http://freepastry.org/PAST/overview.pdf)
- Overlay Weaver (http://overlayweaver.sourceforge.net/doc/)

The following steps are necessary to setup the FreePastry application template in Eclipse:

- Install Java SDK.

- Extract Eclipse Installation-ZIP File and start the eclipse application.
- Create a new Java project. In the Java settings click on the Libraries tab.
- Select Add External JARs... and select the file FreePastry-2.0.jar.
- Copy the application template into the project folder. Double-click the template to edit the Java file.
- To run the main method, select “Run...” in the main menu. Select Java Application and click on New. Select the class ApplicationTemplate as the Main class and click on Run. Now the application should be executed in the console frame. To run another instance of the application click on Run once again.

Testbed Infrastructure

During the challenge task and for the final demo 10 Linux PCs (Pentium 4, 2.8 GHz, 500 MB RAM) will be available during exercise hours in BIN 1.D.12. The application itself shall be tested and demonstrated on the EmanicsLab testbed infrastructure (http://www.csg.uzh.ch/services/emanicslab/). For this purpose an EmanicsLab account will be created for each group. The account details will be provided by the group supervisors.

Each EmanicsLab node participating in the Challenge Task receives the stream on the following ports:

- 5001 - for group 1
- 5002 - for group 2
- 5003 - for group 3
- 5004 - for group 4

The following EmanicsLab nodes participate in the Challenge Task:

- emanicslab1.csg.uzh.ch
- emanicslab2.csg.uzh.ch

Presentation and Evaluation

The challenge task will end on the 29.05.2008. On this date the different groups will present and demonstrate their results which will be evaluated by all participants.

- Each group will have 25 minutes for presentation, setup, and demonstration of their solution. The P2P application shall be brought along as source code and binary (JAR) file on a USB memory stick or CD.
- At least a minimal visualization mechanism shall be provided for viewing the participating nodes. (messages of the following form are considered enough: "Node A: received chunk X; origin B; dest C; next hop D;"). This could be achieved using a log file.
- After the demo each group will be evaluated by the other groups. The following criteria shall be taken into consideration:
    - Fulfillment of requirements

    - Clarity of design
    - Performance of the solution

- Each group will have one vote per group. The lecturers and assistants together will also have one vote per group. The results of the votes will be collected and disclosed after the last presentation. In case that two groups achieve the same result, the winner group will be determined by the vote of the lecturers and assistants or by a quiz about topics of the lecture.
- Finally, the winner group will receive a symbolic prize and the "P2P Challenge Champion 2008" award.

(This page is left blank intentionally.)

University of Zurich
Department of Informatics

ifi

P2P Challenge Task 2008

Solution of Group 1

Authors:

Anthony Lymer, Tobias Moser,
Amruth Kumar Juturu, Dmitri Karpovich

Spring Term 2008

Project overview

Motivation

Nowadays there are many different content providers that provide different audio and video content (Zatoo, Joost etc.). Almost all of them have center-oriented architecture (e.g. client-server) which by definition has a single point of failure (central server) and therefore, generally speaking, is not so robust. The peer-to-peer approach can solve this problem, so the peer-to-peer caching and streaming of online context is very promising. The possible examples of streams that can be cached in peer-to-peer network are: sport events, latest news, TV programs, etc.

Goals

The Goals of challenge task can be defined as learning how FreePastry and P2P in general work and developing time-shift application. The developed application should have the following functionality:

- receive the incoming UDP stream from the video server,
- buffer the live stream and send it to VLC via UDP,
- store parts of the live stream for 7200 seconds (2 hours),
- request the P2P network for video replay for maximum 7200 seconds (2 hours).

Application Requirements:

- Live/timeshift switch: the solution shall enable the user to switch dynamically from live streaming to time shifted streaming and back.
- Realtime access: stream buffering and access for time shifted streams shall happen in realtime.
- Timeshift period: the solution shall allow for the storage of at least two continuous hours of time shifting.
- Robustness: the solution shall be robust against node or link failure during timeshift.
- Multiple clients serving: the solution shall enable each peer to serve multiple clients at the same time, if it has sufficient bandwidth to do so.
- P2P mechanisms: the solution shall be based on Peer-to-Peer mechanisms using a structured overlay network.
- Decentralization: the solution shall not contain any central elements.
- Compatibility: the solution shall be executable on the provided test setup machines (Linux, Java 1.6).

Project details

Installation requirements

- VideoLan Client (VLC) installed;
- Java 1.6;
- Linux/Windows operating systems.

Use cases

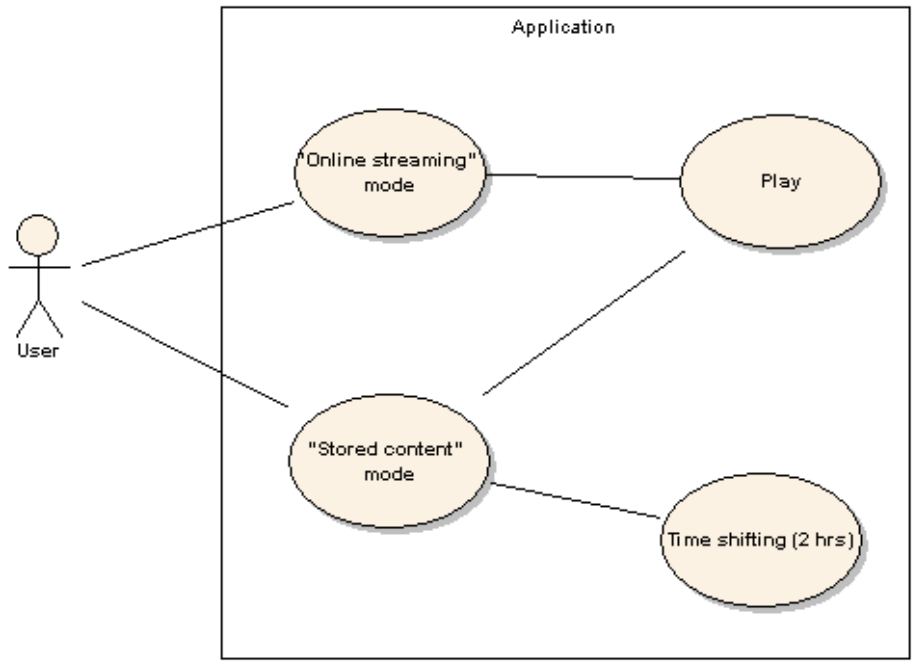


Figure 1. Use cases.

After the application was started user can choose the application video mode: “online streaming” (default mode) or “stored content”. If “online streaming” mode is selected, then application can just start transferring online-stream to local VLC player “as is”. If “stored content” mode is selected, then application can not only just play the cached content, but also do time-shifting of played stream.

Architecture

Two different modes of application work (in sense of network behavior) were designed: bootstrap node and client node. Usually only the first peer that starts the network works as the bootstrap node.

Client node

The client node has the following functionality:

- Connect to the network (bootstrapping to the bootstrap node);
- Receive video chunks;
- Send video chunks to requesting peers via UDP protocol;
- Create queries/messages to lookup a specific part of the video in the network;
- Route queries from other peers;
- Stream video stream to local VLC Player.

Bootstrap node

In contrast to the functionality of a client node (as described above), the bootstrap node provides additional functions:

- Connecting to streaming server;
- Receiving UDP packages from streaming server;
- Removing timestamp from incoming UDP packages;
- Merging packages for storage (original UDP is less than a single frame);
- Building hash function of timestamp;
- Distributing video chunks among the peers in the network over the TCP protocol.

Bootstrap-to-client exchange diagram (content distribution)

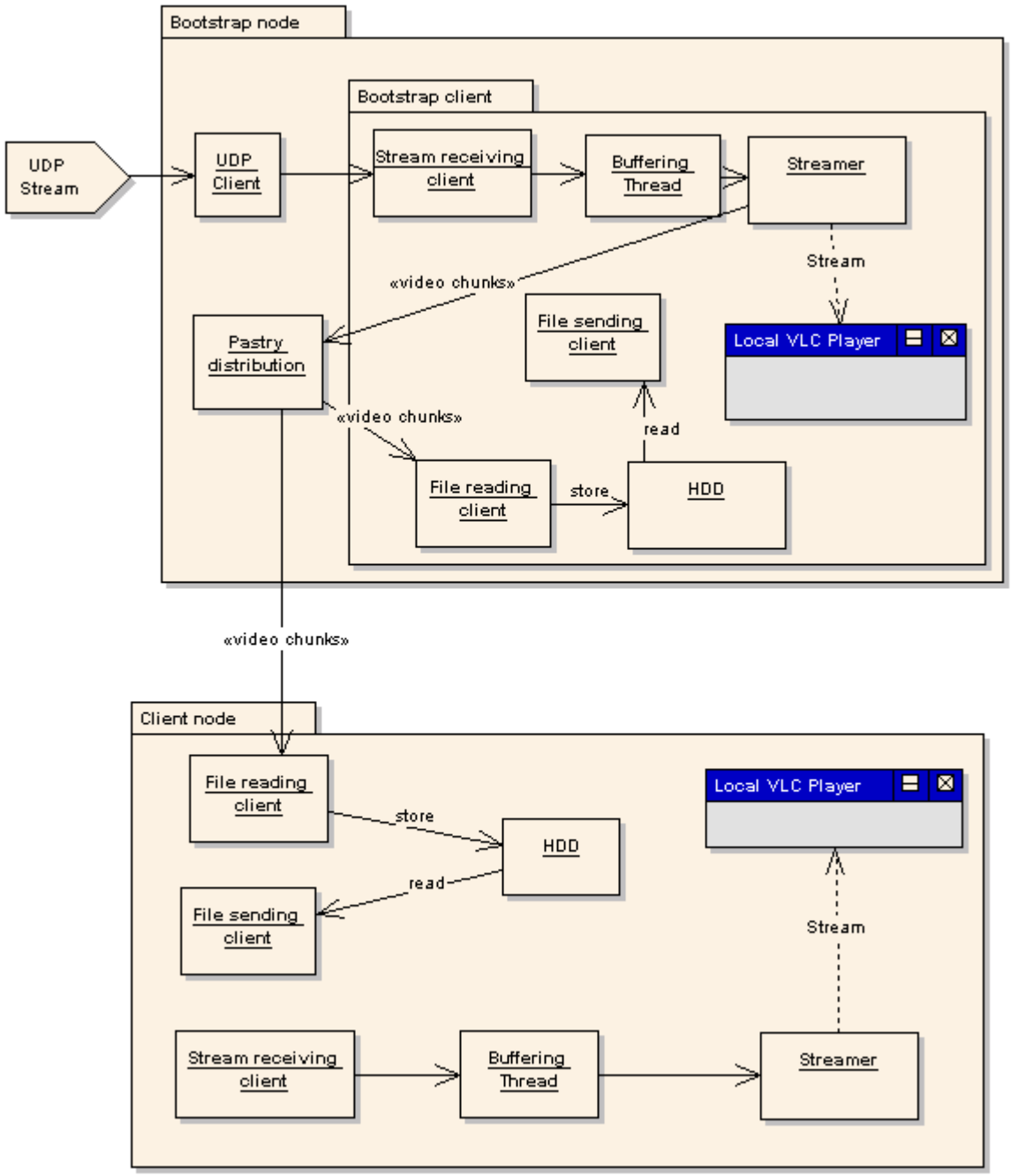


Figure 2. Bootstrap-to-client exchange diagram.

Components of the diagram:

- UDP client:
  - listening to a port,
  - receiving UDP traffic from streaming server;
- Buffer Thread:
  - buffering incoming UDP packets into queue,
  - queue management;
- Streamer:
  - creating files - “video chunks” (in bootstrap node only);
  - removing time stamps from UDP packets,
  - streaming UDP packets to localhost;

- Pastry distribution:
  - calculating IDs of video chunks,
  - looking for a node responsible for storage,
  - sending video chunks to store;
- File reading client:
  - receiving video chunks (TCP),
  - storing video chunks (TCP);
- File sending client:
  - locating and loading local video chunks,
  - sending local video chinks as a UDP stream;
- Stream receiving client:
  - receiving video chunks,
  - streaming them as UDP packets.

Client-to-client exchange diagram ("stored content" video mode)

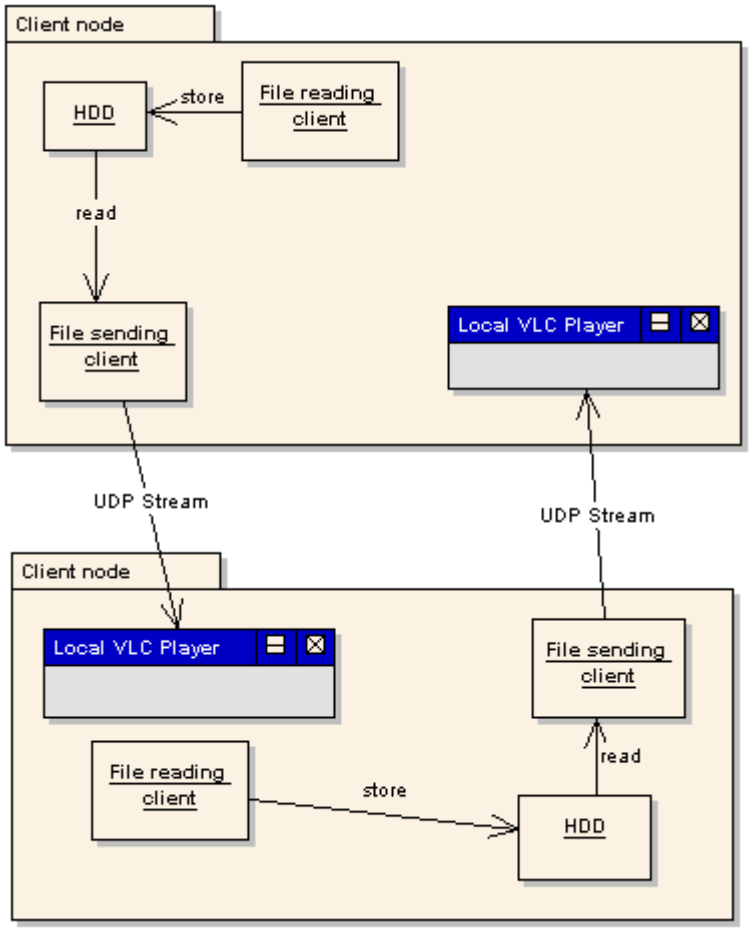


Figure 3. Client-to-client exchange diagram in "stored content" video mode.

When one client node needs to send the stored part of the stream ("video chunk"), to another client node, the "video chunk" is being transferred directly to the VLC player of demanding client.

Client-to-client exchange diagram (“online streaming” video mode)

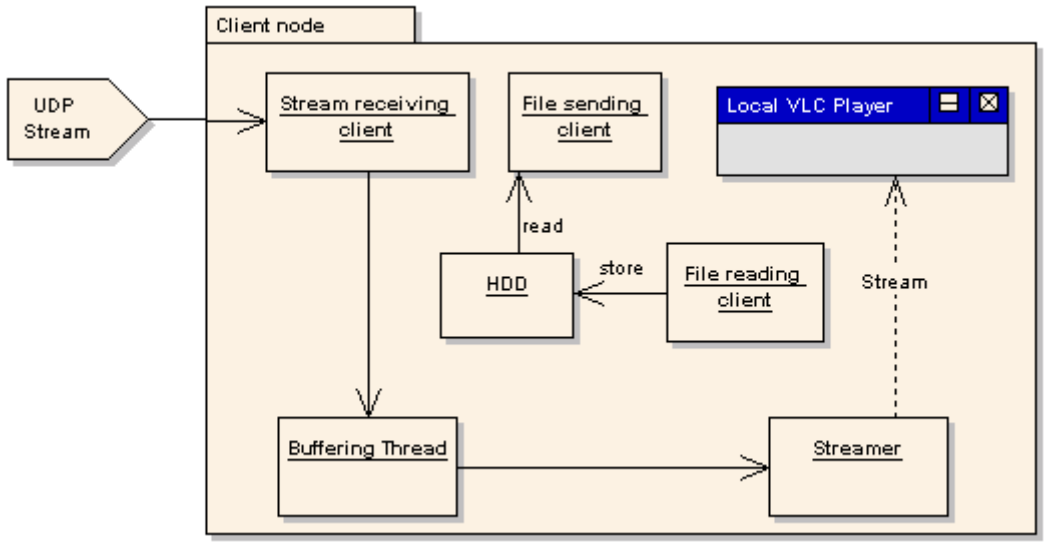


Figure 4. Client-to-client exchange diagram in “online streaming” video mode.

Network handling

Distributing the content

The content is being distributed in several steps. Firstly, the bootstrap node connects to the streaming server and splits the stream into chunks. Secondly, with some time information bootstrap node generates a hash, looks up the destination peer in FreePastry and sends a message to a peer. Thirdly, the peer which received the message sends the bootstrap node its ID, IP address and port where he wants to receive a video stream. Fourthly, the bootstrap node streams the video chunk to the peer via TCP. Lastly, the peer which receives the video chunk saves it on the HDD with the hash (timestamp) as the filename. The steps described above are repeated as the incoming UDP stream exists.

Video streaming

Splitting

Only the bootstrap node splits the stream into video chunks. The incoming stream is being split in files called “video chunks”. Each file represents a video stream, which has exactly one-minute length. The length of the video can easily be changed by adapting a parameter.

Merging

Merging of video chunks into stream does a client node. The merging is being done just by putting incoming chunks into a queue. If the queue isn’t empty the bytes get pulled out of the queue and streamed to localhost.

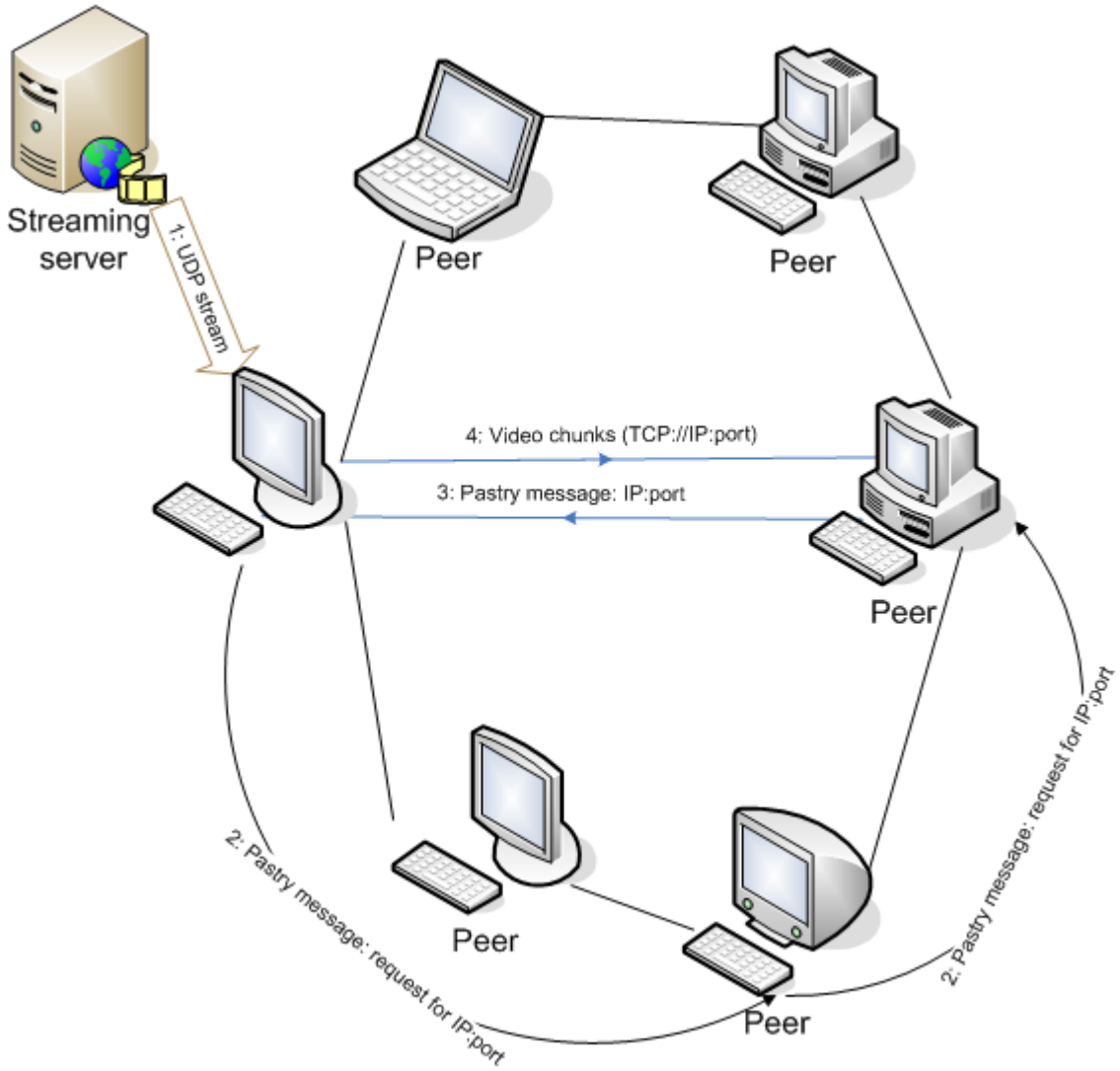


Figure 5. Distributing the content.

GUI description

The GUI is kept very simple. A user has two possible modes: “online streaming” and “stored content”.

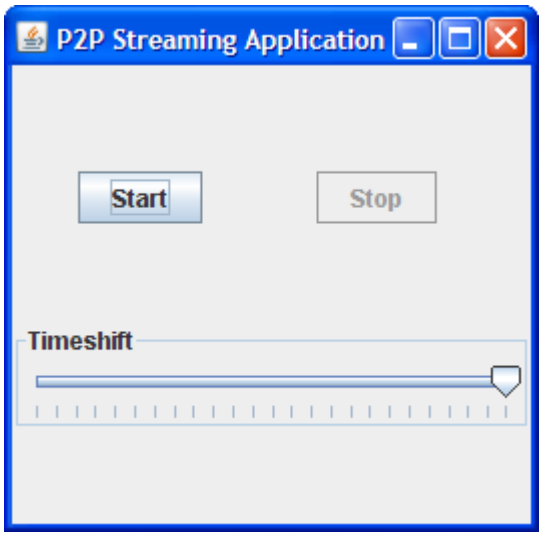


Figure 6. GUI.

If the time-shifting bar is in the utmost right position, the application is in “online streaming” mode, therefore after “Start” is pressed the stream is being showed “as is”. If time shifting isn’t in the utmost right position, then player is in “stored content” mode, and therefore after “Start” is pressed the cached stream is being played.

State automaton

The state automaton diagram below was simplified due to readability. The “Exit” state can be reached from any of the states just by closing the application.

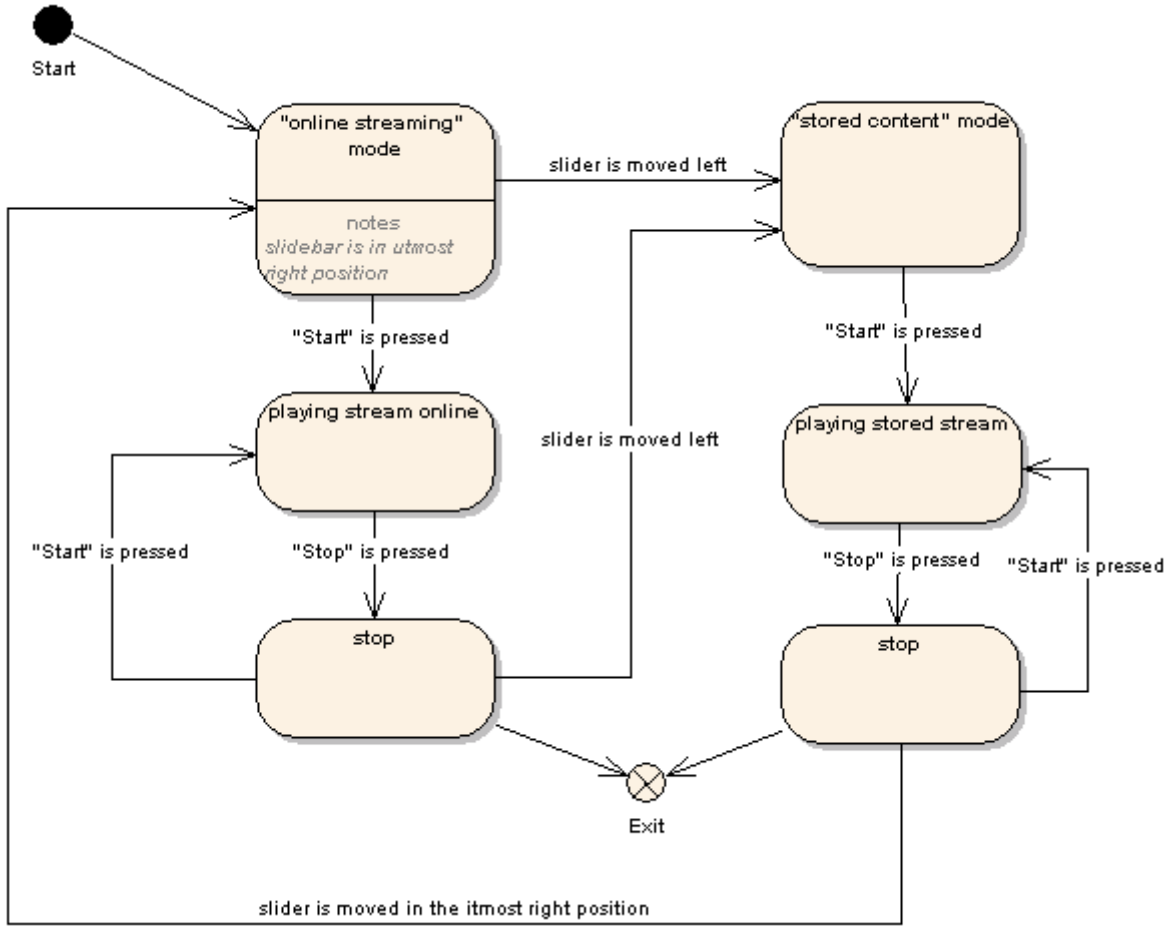


Figure 7. State automaton.

“Online streaming” mode

This mode is used for displaying real-time stream “as is”. The stream can easily be stopped and resumed.

“Stored content” mode

In this mode an additional slider is available, that provides time-shifting functionality. As it follows from the requirements not more than two hours of video content can be stored in the network, therefore the “length” of the slider equals 120 minutes.

Conclusion

Nowadays the peer-to-peer approach is becoming more popular from day to day. Many tasks and problems are being solved with the help peer-to-peer networks. Therefore, it can be concluded, that any peer-to-peer researches and peer-to-peer based solutions are actual today. The whole above outlined project looks quite promising. Nonetheless it can of course be expanded and improved.

Possible future improvements

Redundancy

Redundancy is a very important factor of any P2P network, since, generally speaking, appearance/removal of the nodes is chaotic and unpredictable. Therefore FreePastry managing should take into consideration these factors. We think that higher redundancy of the whole network can be reached in following way:

> The peer *N* that received the video chunk from the bootstrap node sends this chunk to his two neighbors (in sense of FreePastry neighboring) *M* and *K*.

Redundancy would also extend the possibilities for load balancing in our application, because three nodes (instead of one) could serve the requests for a sought-after video part.

References

The components of application that were used:

- Java 1.6: (http://www.sun.com/java/)
- NativeSwing library for Java (http://djproject.sourceforge.net/ns/)
- VLC Media Player (http://www.videolan.org/vlc/)

(This page is left blank intentionally.)

University of Zurich
Department of Informatics

ifi

P2P Challenge Task 2008

Solution of Group 2

Authors:

Marco Kessler, Stefan Christiani,
Roger Peyer, Konstantin Benz

Spring Term 2008

1 Introduction

1.1 Purpose

The purpose of this document is to specify the requirements for the p2p project done as a group work. This document is addressed to any person involved in the development of the project application, as well as the people responsible for evaluating it.

1.2 Scope

The specified project is defined on the website of the lecture and consists in the design and implement a P2P-based recording, storage, and replay application for live video streams.
As was recomended (but not mandated) on the aforementioned mentioned site, the project will be developed in java and will be based on the freePastry engine which has the task to handle the underlying P2P substrate. The displayal of the video streams will be outsourced to VLC.

1.3 Definitions, Acronyms and Abbreviations

- Assistant = responsible for defining the assignment and evaluating the final product.
- Overlay network = The P2P network onto which each TurboTV instance distributes parts of the video stream.
- P2P = Peer to peer.
- Part = A small chunk which represents a limited part of the stream.
- Past stream = A stream which was prerecorded (and thus contains a version of the stream in the past) and thus cannot be considered real-time.
- Pastry = Abbreviation of FreePastry.
- Segment = A sum of one or more sequential parts of a stream.
- Server = The server responsible for multicasting the stream.
- Stream = Audio-Video real-time stream provided by server.

1.4 References

Given definition of the challenge task requisites –
http://www.csg.uzh.ch/teaching/fs08/p2p/challenge/

FreePastry – http://www.freepastry.org/
VLC – http://www.videolan.org/

1.5 Overview

This Document is structured as follows:

Section 1 introduces the reader to the document and provides a coarse overview of the product (TurboTV) to implement. It is strongly recommended to read this section as it illustrates all the important definitions and abbreviations used throughout the document. It also introduces the current tasks of the team members as they were specified in the project wiki (were they are up to date).

Section 2 offers a more detailed description of TurboTV, comprising an illustration of the product perspective, of the functions provided by the product, of the user characteristics (if any), of the costraints imposed on the product and finally of the assumptions that have been taken into account.

Section 3 lists the specific technical requirements involved in the design and implementation of TurboTV and also defines the interfaces through which the components interact.

Section 4 provides some additional information intended to help the reader familiarize with the products and its functions. This information will be represented as Use Cases and Scenarios.

1.6 Contact

1.6.1 Project group wiki

http://challengetask08.wikispaces.com/

2 Overall Description

2.1 Product Perspective

The program that results from this project depends heavily on a clearly defined environment. There are interactions with other instances of the same program and there is an interaction to a main video server. The interfaces used to comunicate between the other clients running the same program described in this report, rely on the pastry substrate while the comunication with the server takes place via an interface defined in a later section of this document (yet to be defined). The functions of TurboTV are described in 2.2.

2.2 Product Functions

The following is a list of functions intentionally offered or neglected by TurboTV. This list is to be continuously updated.

2.2.1 Supported

- TurboTV is able to receive a video stream sent from a given server.
- TurboTV is able to store parts of the received video stream for up to two hours.
- The TurboTV is able to obtain parts of the video stream were broadcast up to two hours in the past.
- The Turbo-TV connects to a P2P network based on pastry.
- The Turbo-TV is able to exchange parts of the video stream between other clients.

2.2.2 Unsupported

- TurboTV cannot in uence the video server.
- TurboTV cannot work with video streams other than those supported by VLC.
- TurboTV cannot obtain past parts of a video stream if no other instance of TurboTV is running and stored it.

2.3 User Characteristics

2.3.1 Everyone:

Read the FreePastry Tutorial

2.3.2 Marco:

- Set up SVN repository and accounts
- Use Cases

2.3.3 Stefan:

- Functional Requirements
- ToC Report (including functional Requirements, Modules, Use Cases, VLC)

2.3.4 Roger:

- Organize Wiki
- Modules

2.3.5 Konstantin:

VLC Integration GUI Prototype

2.4 Assumptions, Constraints and Dependencies

- The assignemnt provides a video server that streams the video.
- Live/Timeshift switch: The solution shall enable the user to switch dynamically from live streaming to time shifted streaming and back.
- Realtime access: Stream buffering and access for time shifted streams shall happen in realtime.
- Timeshift period: The solution shall allow for the storage of at least two continuous hours of time shifting.
- Robustness: The solution shall be robust against node or link failure during timeshift.

- Library and tools: The solution shall be based on libraries and tools which allow for publishing under GPL or another comparable open software license.
- Application report: The application report shall document the application. The report shall have 5-10 pages.
- The auditing process is not defined or requested yet.
- Reliability aspects have not been defined or requested yet.
- The criticality of the component has not been defined yet.
- The safety and security aspects of the component have not been defined yet.

3 Specific Requirements

3.1 Functionality

3.1.1 Viewer watches TV using TurboTV

Description TurboTV receives a stream from either the server or the overlay network and displays it in a window. At the same time, it shares known parts of the stream to other peers, using the overlay network.

Priority 1

Risk 1

3.1.2 TurboTV connects

Group Connection establishment - summary

Description TurboTV connects to both the server and the overlay network. If there is no overlay network, it is created by pastry.

Priority 1

Risk 1 (Connection is essential for TurboTV)

3.1.3 TurboTV handles stream

Group Stream handling - summary
Description TurboTV handles the incoming stream in that it is stored for later usage (time-shifting or sharing).
Priority 2
Risk 3

3.1.4 TurboTV displays stream

Group Content display - subfunction
Description The content requested by the viewer (stream or past stream) is shown by TurboTV in an apposite GUI element embedding VLC.
Priority 1
Risk 1 (The main crucial feature of TurboTV)

3.1.5 Portrayal of past stream

Group Content handling - function
Description TurboTV displays a past stream to the user who, interactively, choses a time in the past at which he would like to observe the stream.
Priority 2 (Stream is more important than the past stream)
Risk 3

3.1.6 Viewers wants to watch a past stream

Group Content handling - subfunction
Description The user choses a time which lies up to two hours in his past. This might either mean to switch from the stream to a past stream, or means to move to another time inside the past stream.
Priority 2
Risk 2

3.1.7 TurboTV requests missing parts

Group Content handling - subfunction
Description TurboTV requests parts that are not avilable through neither the server nor in the own memory. Other peers in the overlay network are expected to answer to this request.
Priority 3
Risk 3

3.1.8 TurboTV receives missing parts

Group Content handling - subfunction
Description TurboTV receives missing parts and assembles them in a yet to be defined manner in memory, so that they might be displayed at a later time.
Priority 2
Risk 2

3.1.9 Overlay network requests part from TurboTV

Group Content request - function
Description Some peer asks the current instance of TurboTV for a part. TurboTV sends them the part, should it be available.
Priority 2
Risk 1

3.1.10 TurboTV checks if part is available

Group Content request - subfunction
Description Some peer asks the current instance of TurboTV for a part. TurboTV checks if the part is available in the current memory.
Priority 2
Risk 1

3.1.11 TurboTV sends part

Group Content request - subfunction
Description TurboTV sends a requested part to the requesting peer in the overlay network.
Priority 2
Risk 1 (the whole time-shift tv part depends on the correct functioning of this subfunction)

3.2 Usability

The usability has to be such that an inexperienced user can use the TurboTV without reading any manual. Some instructions must be given due to the different OSs where TurboTV will be running.

3.5 Maintainability

The system is not prepared for maintance therefore no regression test suite is provided.

3.6 Design Constraints

Since the systems on which TurboTV shall run are given, they impose the design constraints. Please see the project page for further details.

4 Use Cases

4.1 Viewer watches TV using TurboTV

Primary Actor: Viewer
Goal in Context: The viewer feels like watching some tv-stream using the TurboTV application.
Scope: Content-Delivery – Receiving and displaying an audio-video stream in TurboTV.
Level: Very-Summary
Stakeholders and Interests: Viewer: want to watch an audio-video stream.
Server: provides the real-time stream.
Overlay-network: provides the past stream.
TurboTV: handles the input/output of the stream.
Precondition: The user has installed TurboTV and has full inter-/intranet connectivity.
Minimal Guarantees: An error message is provided to understand failure.
Success Guarantees: The Viewer successfully watches TV.
Trigger: The Viewer opens TurboTV.
Main Success Scenario:

1. TurboTV connects to the networks (real-time stream server and overlay network).
2. Viewer watches real-time stream.
3. TurboTV provides requested stream parts to the overlay-network.

Extensions:

1a. TurboTV fails to connect to the stream server. An error message is displayed and the application closed.

1b. TurboTV fails to connect to the overlay-network. TurboTV creates a new overlay-network.

2a. The viewer switches to Time-Shift mode. See !OPEN!

4.1.1 TurboTV connects to the networks

Primary Actor: TurboTV
Goal in Context: TurboTV connects to both the stream server and the overlay-network.
Scope: Connection Establishment – Establishing a connection to the server and to the overlay-network.
Level: Summary
Stakeholders and Interests: Server: provides the stream.
Overlay-network: provides the past stream.
TurboTV: establishes connections.
Precondition: The stream server is available and listening for new connections.
Minimal Guarantees: An error message is provided to understand failure.
Success Guarantees: A connection to the stream server is established and an overlay-network is available.
Trigger: -
Main Success Scenario:

1. TurboTV connects to the stream server.
    a) TurboTV sends an UDP request to the stream server.
    b) The stream server acknowledges by sending the stream.
2. TurboTV connects to the overlay-networks.
    a) TurboTV contacts the bootstrap addresses.
    b) TurboTV joins the over-lay network.

Extensions:

1.2.a. The stream server does not acknowledge within a certain time interval. Go back to 1.1.

2.1.a All bootstrap nodes are unreachable. Pastry automatically creates a new overlay-network.

4.1.2 Viewer watches real-time stream

Primary Actor: Viewer
Goal in Context: the viewer wants to see a real-time stream.
Scope: Content-delivery – TurboTV obtains the required stream and displays it.
Level: Summary
Stakeholders and Interests: Viewer: wants to see a live stream.
Server: provides the live stream.
TurboTV: connects to the server.
Precondition: -
Minimal Guarantees: upon errors a message is shown explaining the issue.
Success Guarantees: the stream is successfully displayed.
Trigger: -
Main Success Scenario:

1. TurboTV handles the incoming stream.
2. TurboTV displays the stream.

4.1.3 TurboTV handles the incoming stream

Primary Actor: TurboTV
Goal in Context: TurboTV receives the stream and stores it on disk following a certain strategy.
Scope: Stream Handling – receiving and storing the incoming stream piecewise.
Level: Summary
Stakeholders and Interests: Server: provides the stream.
Overlay-network: provides the past stream.
TurboTV: handles the stream.
Precondition: There is an incoming stream.
Minimal Guarantees: upon errors a message dialog is shown explaining the issue.
Success Guarantees: the stream is received.
Trigger: the connections have been established.
Main Success Scenario:

1. TurboTV receives the stream.
2. TurboTV saves the stream following a yet to be defined strategy (different strategies for real-time and time-shift?)

Extensions:

2.a. Alternatively, if the streams is from the past, a different storing strategy is used. (?)

4.1.4 The stream is displayed with VLC

Primary Actor: TurboTV

Goal in Context: the stream is passed to VLC where it gets displayed to the viewer.

Scope: Stream Handling – the TurboTV GUI displays the requested stream.

Level: Summary

Stakeholders and Interests: Viewer: enjoys the displayed stream.
TurboTV: displays the stream through VLC.

Precondition: the stream is available in memory and not corrupt.

Minimal Guarantees: upon errors a message is shown explaining the issue.

Success Guarantees: the stream is successfully displayed.

Trigger: there is enough stream buffered to be shown.

Main Success Scenario:

1. The stream is channelled into VLC.
2. VLC renders the stream.

4.2 Viewer watches past streams with time-shift

Primary Actor: Viewer
Goal in Context: the viewer wants to see parts of a past stream.
Scope: Content-delivery – TurboTV obtains the required stream and displays it.
Level: Summary
Stakeholders and Interests: Viewer: wants to see some past event.
TurboTV: needs to obtains the desired stream pieces.
Precondition: -
Minimal Guarantees: upon errors a message is shown explaining the issue.
Success Guarantees: the stream is successfully displayed.
Trigger: the viewer selects a time in the past using the TurboTV GUI (slider).
Main Success Scenario:

1. The viewer selects a particular time in the past using the GUI slider.
2. TurboTV requests the missing stream parts.
3. TurboTV obtains the missing parts and handles them.
4. TurboTV displays the stream. See !OPEN!

4.2.1 TurboTV requests missing parts

Primary Actor: TurboTV
Goal in Context: TurboTV needs a past stream.
Scope: Content-delivery – TurboTV looks up the past stream parts that are missing.
Level: Summary
Stakeholders and Interests: Overlay-network: network of peers that contains the past stream, split in parts.
TurboTV: needs to obtain the desired stream pieces.
Precondition: At least one peer has to posses the requested missing parts.
Minimal Guarantees: upon errors or impossibility to find the parts, a message is shown explaining the issue.
Success Guarantees: the peers holding the requested parts are identified.
Trigger: -
Main Success Scenario:

1. TurboTV checks if missing parts are available on the server.

2. TurboTV resends missing parts.

4.2.2 TurboTV receives the missing parts

Primary Actor: TurboTV
Goal in Context: TurboTV needs a past stream.
Scope: Content-delivery – TurboTV received or is receiving past stream parts.
Level: Summary
Stakeholders and Interests: Overlay-network: network of peers that contains the past stream, split in parts.
TurboTV: needs to obtain the desired stream pieces.
Precondition: -
Minimal Guarantees: upon errors a message is shown explaining the issue.
Success Guarantees: the requested stream, or shorter segments of it, are reconstructed and ready to be shown.
Trigger: -
Main Success Scenario:

1. TurboTV receives the missing stream.

4.3 Overlay-network requests parts from local TurboTV

Primary Actor: TurboTV
Goal in Context: the overlay-network asks TurboTV whether it has some particular stream parts ready to send.
Scope: Stream Handling – sending requested stream parts.
Level: Summary
Stakeholders and Interests: Overlay-network: asks TurboTV for specific stream parts.
TurboTV: checks if parts are available and sends them if it is so..
Precondition: TurboTV has joined the same overlay-network that requests the parts.
Minimal Guarantees:
Success Guarantees: the missing part(s) are sent if available, otherwise, the overlay-network is ignored.
Trigger: another instance of TurboTV in the same overlay-network requires past stream parts.
Main Success Scenario:

1. TurboTV checks if the part is available locally.
2. The part is available and TurboTV sends it back through the overlay-network.

Extensions:

2.a. If the part is not available locally, the overlay-network is ignored.

4.3.1 TurboTV checks if part is available locally

Primary Actor: TurboTV

Goal in Context: the overlay-network asks TurboTV whether it has some particular stream parts ready to send.

Scope: Content discovery – TurboTV looks for a local copy of the requested part.

Level: Sub-function

Stakeholders and Interests: Overlay-network: asks TurboTV for specific stream parts.
TurboTV: checks if part is available.

Precondition: TurboTV has joined the same overlay-network that requests the parts.

Minimal Guarantees:

Success Guarantees: the missing part(s) are sent if available, otherwise, the overlay-network is ignored.

Trigger: another instance of TurboTV in the same overlay-network requires past stream parts.

Main Success Scenario:

1. TurboTV detects parts .

4.3.2 TurboTV sends requested parts

Primary Actor: TurboTV

Goal in Context: the overlay-network asks TurboTV whether it has some particular stream parts ready to send.

Scope: Transfer – available parts are sent over the overlay-network.

Level: Sub-function

Stakeholders and Interests: Overlay-network: asks TurboTV for specific stream parts.
TurboTV: sends available parts.

Precondition: TurboTV has joined the same overlay-network that requests the parts.

Minimal Guarantees:

Success Guarantees: the available parts are sent.

Trigger: another instance of TurboTV in the same overlay-network requires past stream parts.

Main Success Scenario:

1. TurboTV detects parts.
2. TurboTV sends parts to the client.

(This page is left blank intentionally.)

University of Zurich
Department of Informatics

ifi

P2P Challenge Task 2008

Solution of Group 3

Authors:

Linard Moll, Philip Schaffner,
Franziska Schait, Rocky Lonigro

Spring Term 2008

1. Architektur

Die drei zentralen Teile der Applikation sind:

1) P2P Underlay:
   - Einbindung von Pastry und PAST zum Aufbau und Unterhalt des P2P-Netzwerkes: Methoden fürs Abspeichern und Auffinden der Dateien im Netzwerk.
   - Housekeeper Methoden (~P2P GC)
   - Indirektion der benötigten Netzwerkfunktionalitäten für Zugriff durch Scheduler bereitstellen (Design: Façade, Event- und Message-based)

2) Scheduler
   - Algorithmen zur Definition und Sortierung der Time-Slices (Aufnahme, Platzieren/Finden im Netzwerk, Wiedergabe): Welche Slices unter welchem Namen aufgenommen werden, wo sie abgespeichert werden, welche Slices angefordert werden etc.
   - TimeShift: Beschaffung und Verwaltung der Slices, die für einen bestimmten TimeShift benötigt werden
   - Aufbauend auf den P2P Underlay ein End2End Kommunikationsprotokoll erstellen (z.B.: SLICE_NEEDED, SLICE_RECORDED, SLICE_UPLOADED etc.)

3) GUI
   - Konfigurationsfenster für Verbindungsaufbau zum Netzwerk
   - Abspielen des Streams
   - Slider für TimeShift-Funktion und Bereichsauswahl
   - Events/EventDispatcher für gesamte Applikation

2. Aufgabe

Die Aufgabe dieser P2P Challenge Aufgabe ist, ein P2P Video Streaming System zu entwerfen und prototypisch zu implementieren. Das System soll eine Time-Shift Funktionalität enthalten, damit der Live-Stream Zuschauer jederzeit die Wiedergabe von wichtigen oder interessanten Sequenzen wiederholen kann. Dabei soll bis 2h des Streams im P2P Netzwerk gespeichert und abrufbar sein. Jeder Zuschauer speichert Teile des Video-Streams auf seiner Festplatte. Dabei muss er nicht alles speichern, sondern kann für die Time-Shift Wiedergabe auf einfernte Teil zugreifen.

3. Zugrunde liegende Ideen

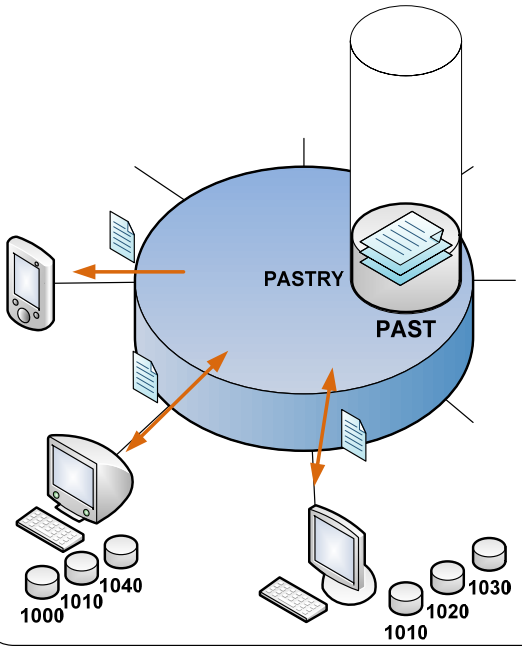
Die Systemzeit spielt neuerdings keine Rolle mehr, da der Datenstrom eine Zeitabgabe enthält. Somit fällt die Synchronisation aller Nodes weg und es kann

der hinzugefügte Timestamp der ankommenden UDP Packete nutzen.
Aufgrund eines vordefinierten Musters (beispielsweise in Schritten von einiger Minuten) wird der Stream in Slices aufgeteilt, die anhand des Zeitpunkts, in dem sie aufgenommen wurden, benannt und gespeichert werden. Die Verteilung und Speicherung der Metainformationen wird, wie in Illustration 1 und 2 gezeigt, durch Pastry und PAST übernommen. Dabei wird eine Versionierung umgesetzt und zusammen mit einem Locking Mechanismus (Illustrationen 3 bis 5). Nach der Speicherung eines neuen Slice veröffentlicht der Peer Informationen dazu im P2P Netzwerk. Dazu wird die ID des dazugehörenden Metadaten Dokument (Peerliste) im Pastry Netzwerk gesucht. Als Schlüssel für die Suche dient der Zeitpunkt der Aufnahme. Sollte kein Wert zu diesem Anfrageschlüssel passen, so wird jetzt ein neues Dokument erstellt. Sollte ein Wert aus der Pastry Anfrage zurückgegeben werden, so wird die erhaltene Dokumenten-ID im PAST Netzwerk nachgefragt und das Dokument beschafft. Das Dokument enthält einen oder mehrere Node-Einträge, wie in der Illustration 2 abgebildet wurde. Diese Liste wird nun um die eigene Node-ID ergänzt, die Dokumenten-ID neu errechnet und im PAST Netzwerk unter der neuen ID veröffentlicht. Anschliessend wird im Pastry Netzwerk der vorhandene Schlüssel (= Zeitpunkt der Aufnahme) mit der neuen Dokumenten-ID versehen.

Pastry & PAST Übersicht

Pastry: Verwaltet P2P Nodes und erlaubt die Suche nach TimeSlots

PAST: Verwaltet die redundante Speicherung von Hostlisten



Pastry: Erstellt P2P Netzwerk, verwaltet Nodes

PAST: Dient als Storage Netzwerk für Host-Listen

Idee:
Das Pastry Netzwerk enthält als Key alle gespeicherten TimeSlots (zb: 1000, 1010, 1020 etc). Als Value wird die ID eines Dokumentes gespeichert – einer Hostliste. Die Hostliste enthält alle Hosts, welche den TimeSlot aufgezeichnet haben. Sollte nun ein weiterer Host den TimeSlot aufgezeichent haben, so beschafft er sich die Liste, fügt seinen Eintrag rein und speichert sie mit neuem Hashwert in das Pastry Netzwerk rein. Somit wird immer die aktuellste Liste abgerufen und aktualisiert.
Beim Timeshift wird pro TimeSlot die Liste beschafft und das Datenpaket geladen.

Vorteile:
- Die Nodes müssen die Datenslices erst beim Timeshift übers Netzwerk senden. Netzwerk-Load sinkt.
- Publizieren der Daten geht somit schneller. Nachdem der Slice auf der HD gespeichert wurde, kann bereits mit dem nächsten begonnen werden. Der redundante Upload an PAST-Nodes entfällt.
-PAST speichert die Hostlisten redundant (einfacher für uns)

Nachteile:
Die veralteten Listenversionen müssen gelöscht werden, sonst gibt's Memory Leaks.

Aktuell:
Einsatz von Housekeeper und PAST-GC um Memory wieder freizugeben.

Illustration 1: Pastry & PAST

In der grafischen-Oberfläche (GUI) wird anhand des Sliders für die TimeShift-Funktion entweder der Live-Stream abgespielt, oder aber es werden aufgrund der Netzwerkzeit und der auf dem Slider eingegebenen Zeitdifferenz die richtigen Slices berechnet, gesucht und angezeigt. Dazu wird ein Scheduler erstellt der diese Koordinationsaufgabe pro Peer übernimmt. Dies ist in der Illustration 6 abgebildet.

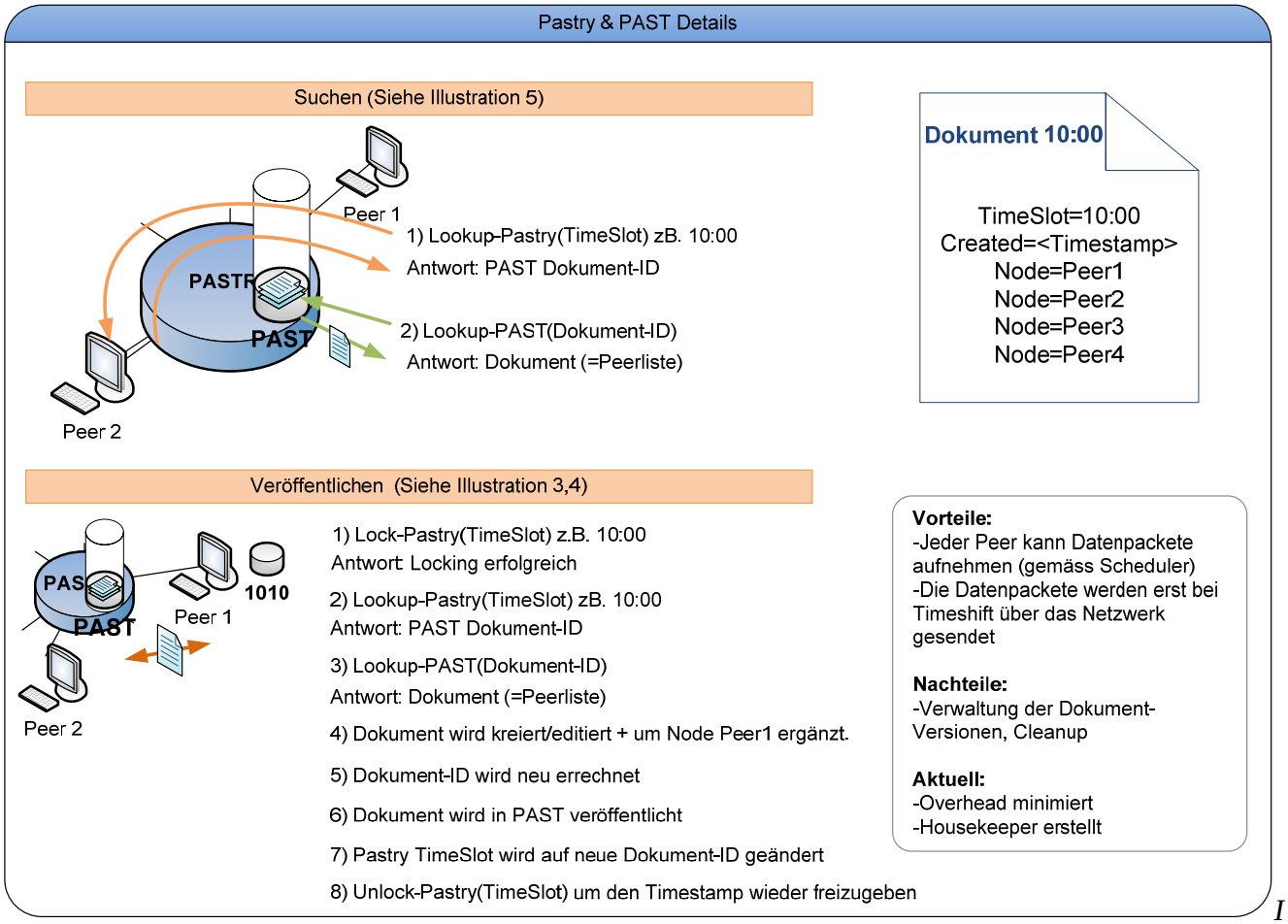


Illustration 2: Funktionsweise Suchen & Veröffentlichen von Datenpaketen

4. Umsetzung

GUI: Rocky Lonigro
Use Cases, Scheduler: Franziska Schait, Philip Schaffner
Kommunikationsaspekte, restliche Applikation: Linard Moll

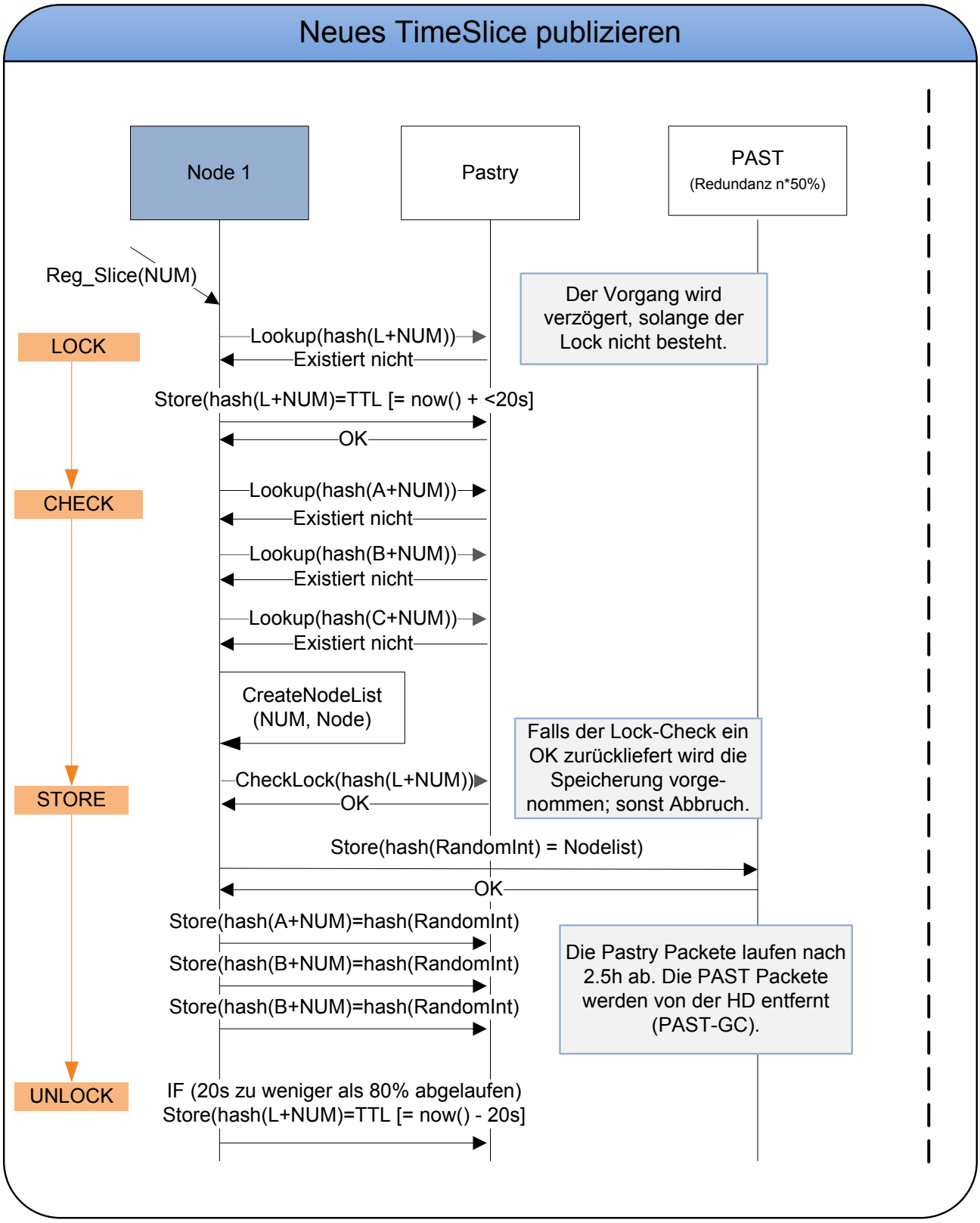


Illustration 3: Versionisierung / Locking Mechanismus: TimeSlice abspeichern.

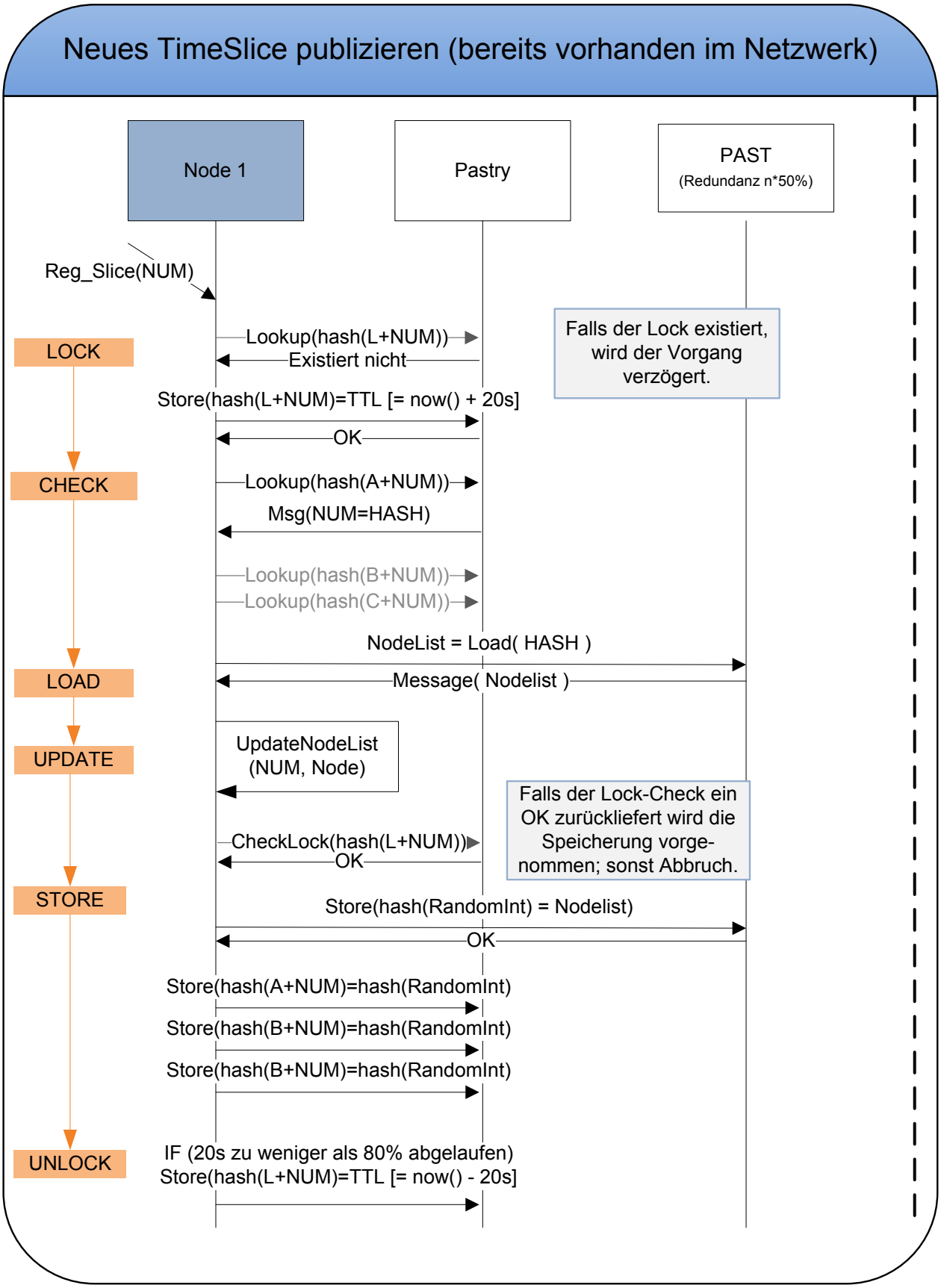


Illustration 4: Versionisierung / Locking Mechanismus: Vorhandenes TimeSlice abspeichern.

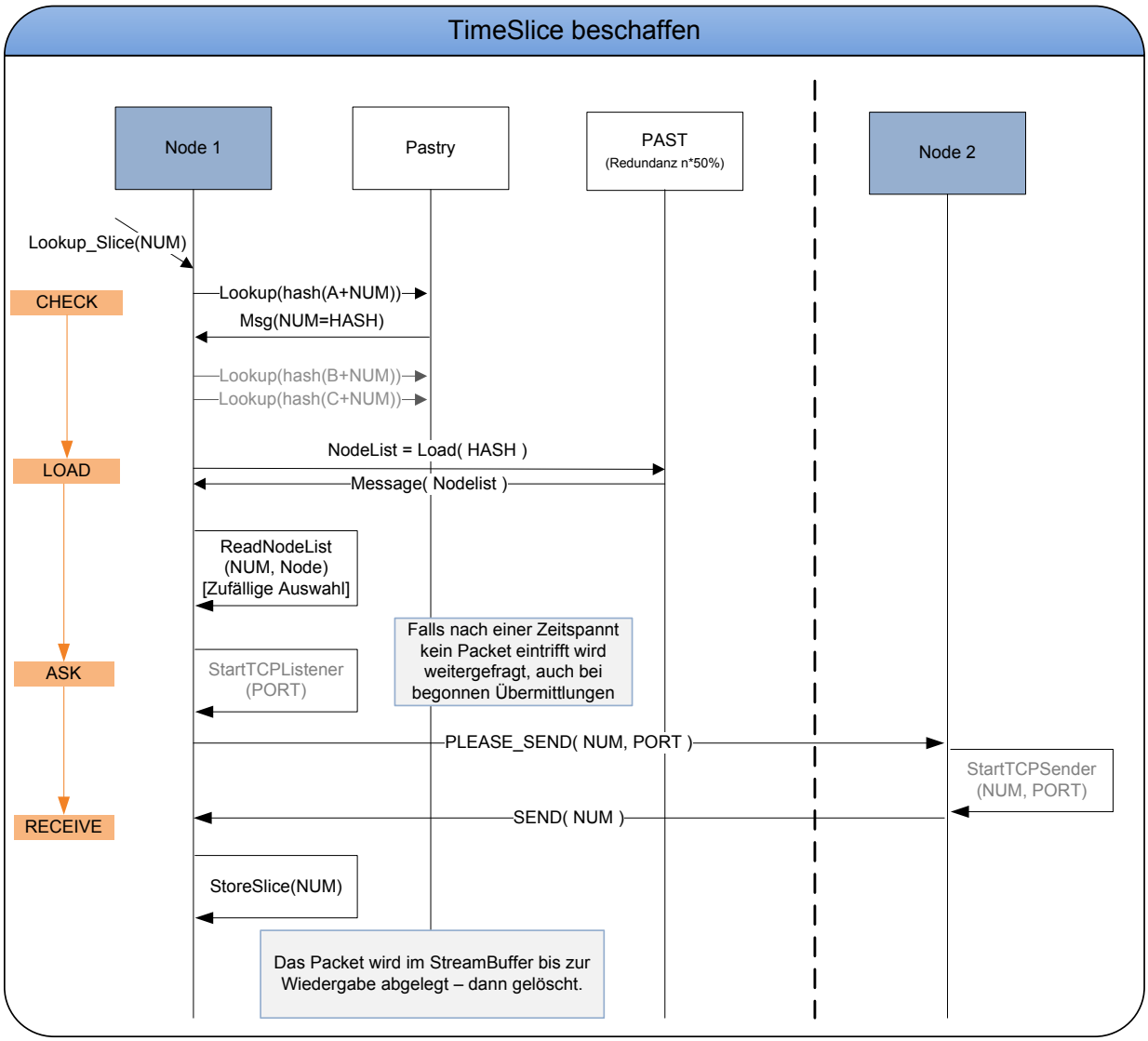


Illustration 5: Versionisierung / Locking Mechanismus: TimeSlice beschaffen

Die Sekunden Angaben sind jeweils zur Illustration. Momentan beleibt ein Lock 5 Sekunden bestehen und wird dann ungültig.

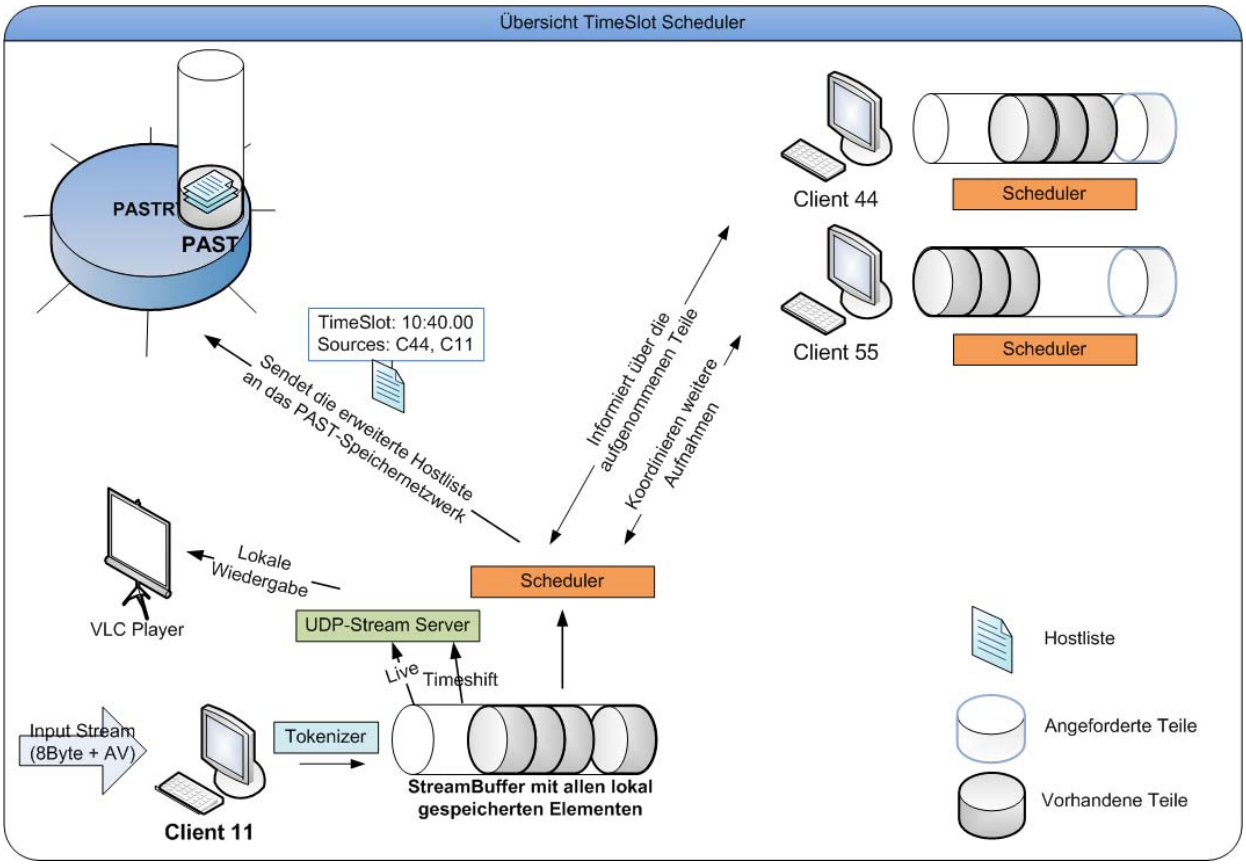


Illustration 6: Funktionsweise der Scheduler Komponente.

5. Ausblick

Gemäss Aufgabenstellung musste ein Prototyp erstellt werden. Dies ist uns gelungen. Folgende Features könnten noch verbessert und ergänzt werden, bzw. wurde aus zeitlichen und personellen Gründen nicht angepasst:

1) UDP anstatt TCP Transfer der Slice Packete. TCP eignet sich nicht sonderlich gut für Livestreaming und kann bei instabilen Verbindungsverhältnissen zu unterbrücken führen. Da diese Applikation unter Laborbedingungen entwickelt und getestet wurde, war dieser Umstand nicht besonders relevant.
2) Das Locking Protokoll muss überarbeitet werden, damit jede Art von Race-Condition ausgeschlossen werden kann. Momentan wird nach erfolgreichem Locking ein zweites Mal nachgefragt, um dann bei bestehendem Lock den Link zur PAST-Datei (Peerliste) zu überschreiben.

Ansonsten war es eine interessant Erfahrung, diese P2P Libraries verwenden zu können und Kommunikationsalgorithmen damit aufzubauen.

6. Technischer Hintergrund

Klassen & Interface Hierarchie

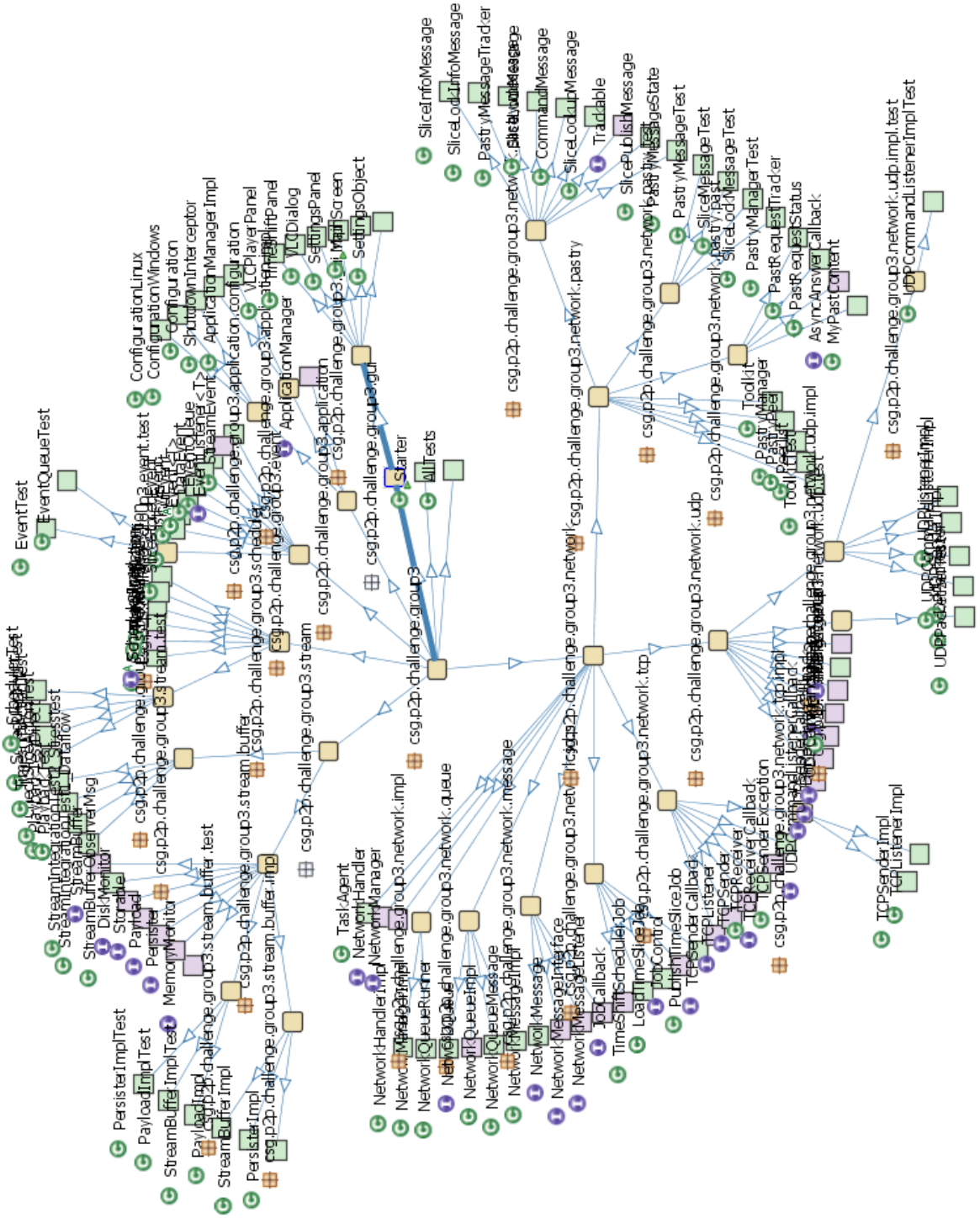


Illustration 7: Klassen & Interface Hierarchie (radial), Testklassen eingeblendet.

Klassen Hierarchie

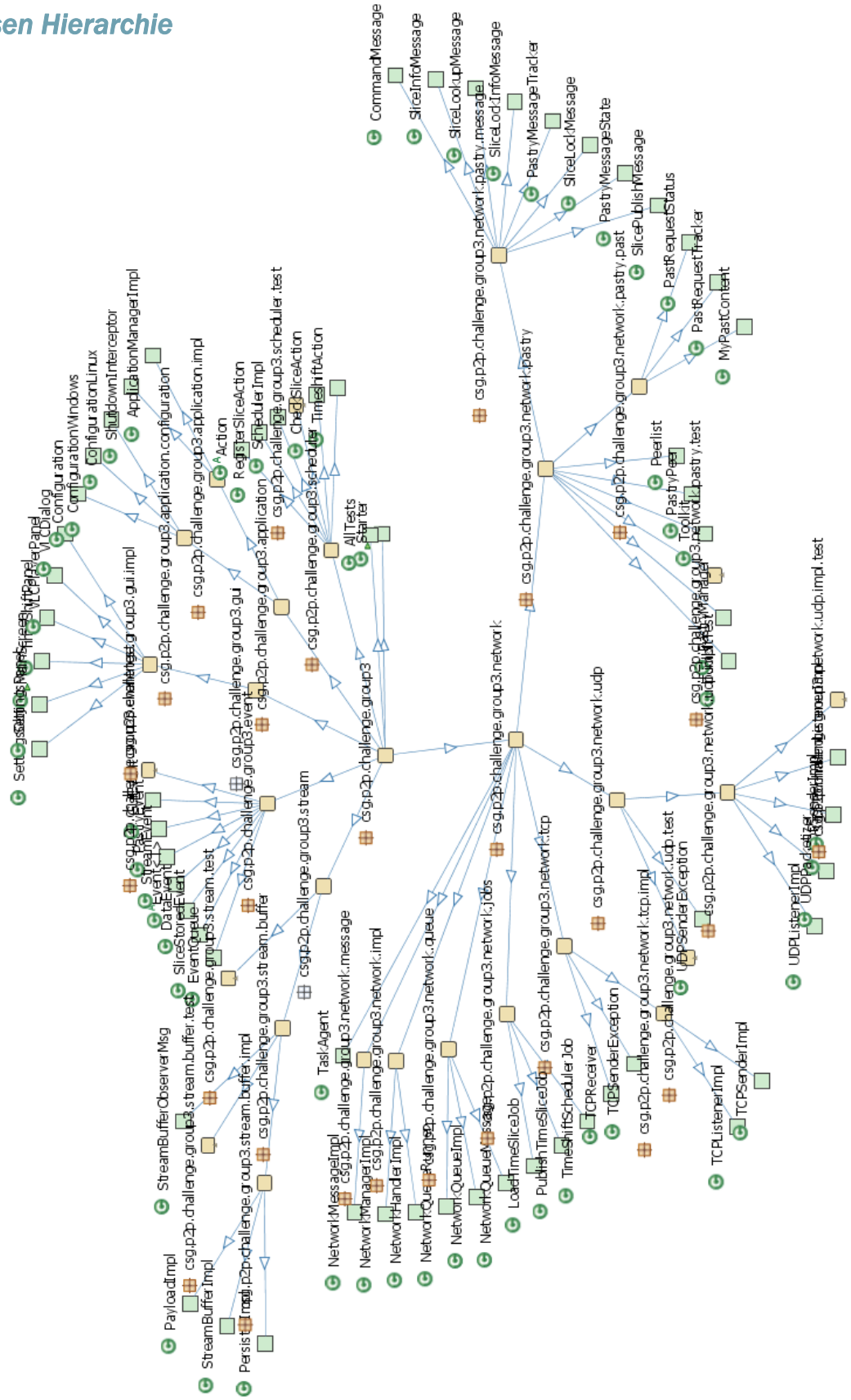


Illustration 8: Klassen Hierarchie (radial), Testklassen ausgeblendet.

Interface Hierarchie

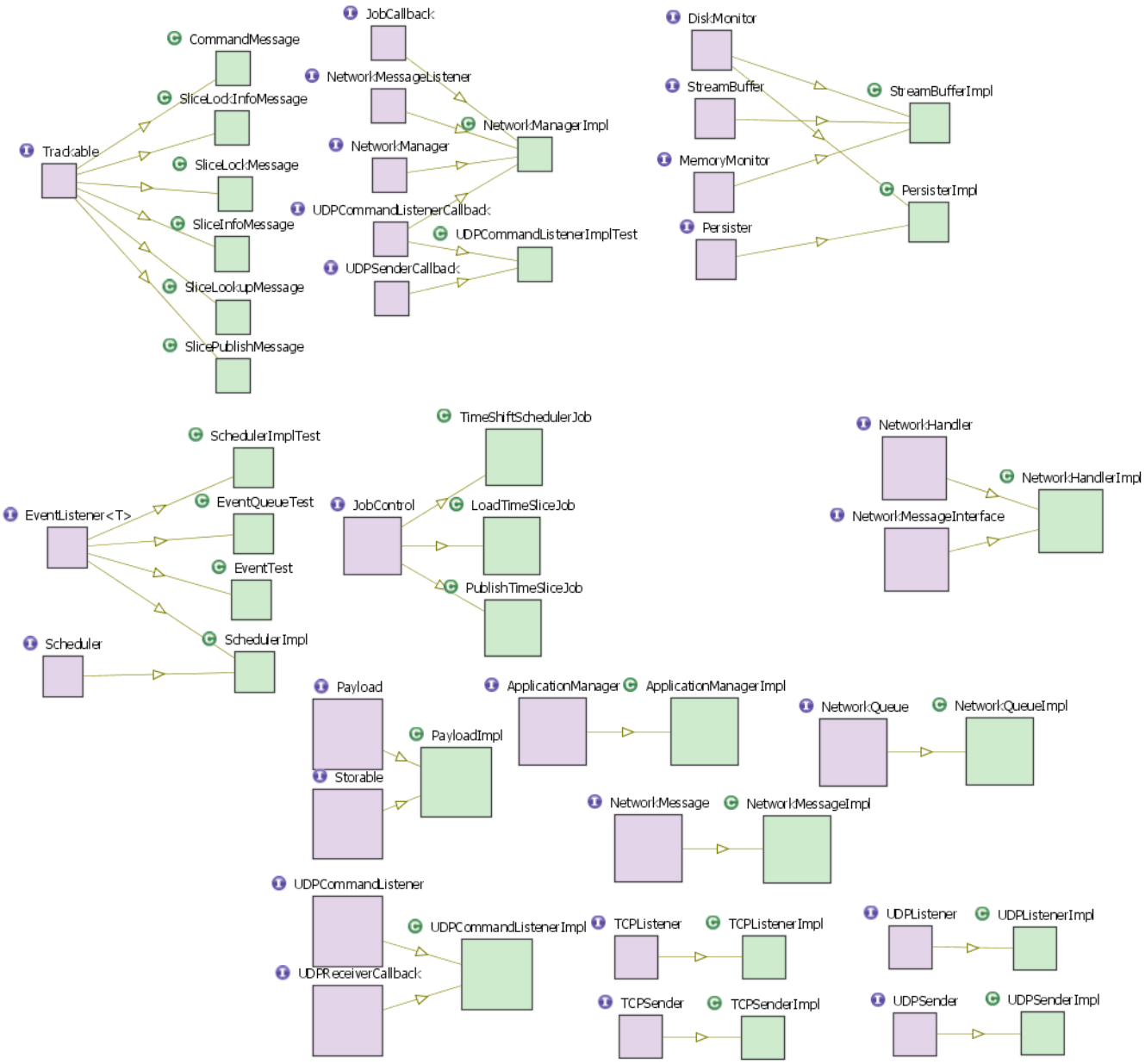


Illustration 9: Interface Hierarchie.

Abhängigkeiten auf Ebene Package

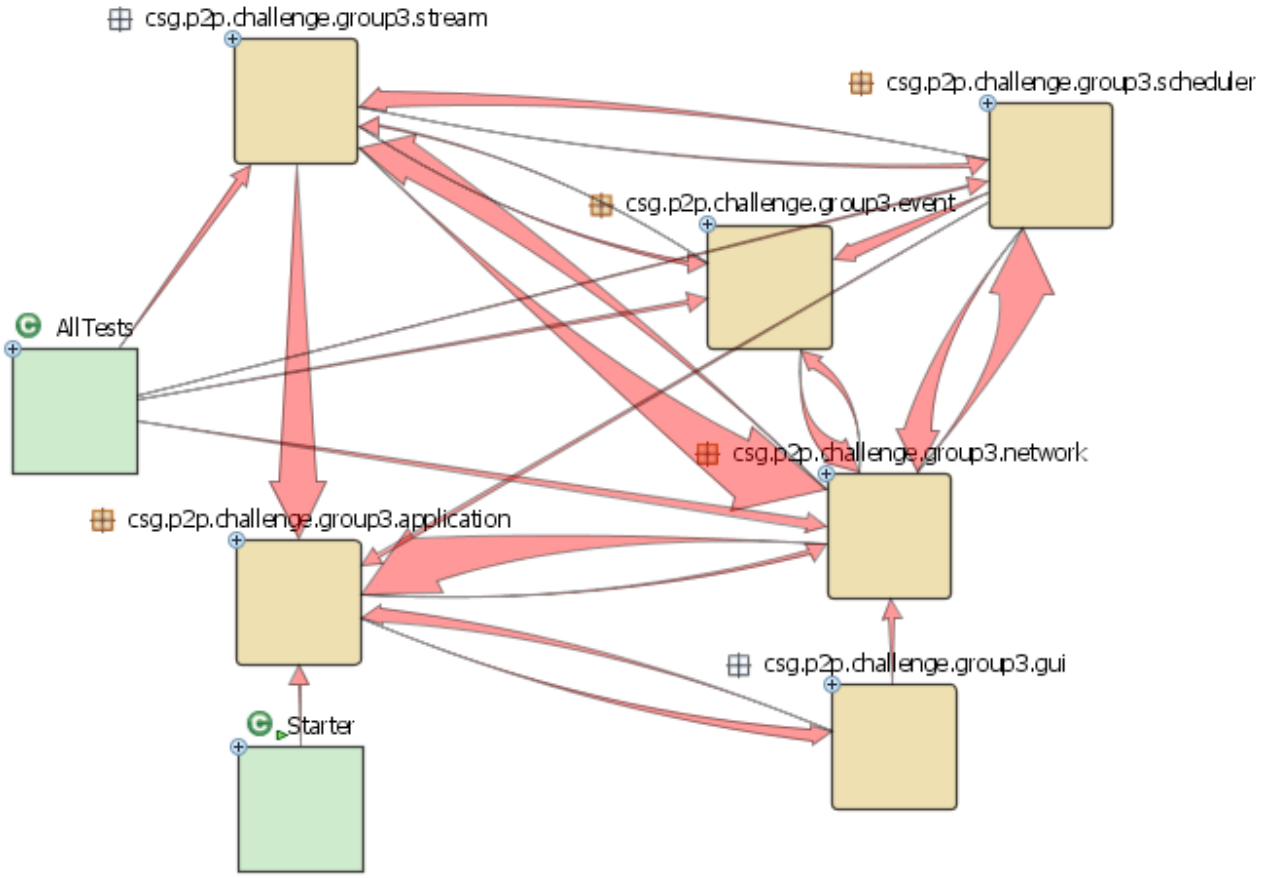
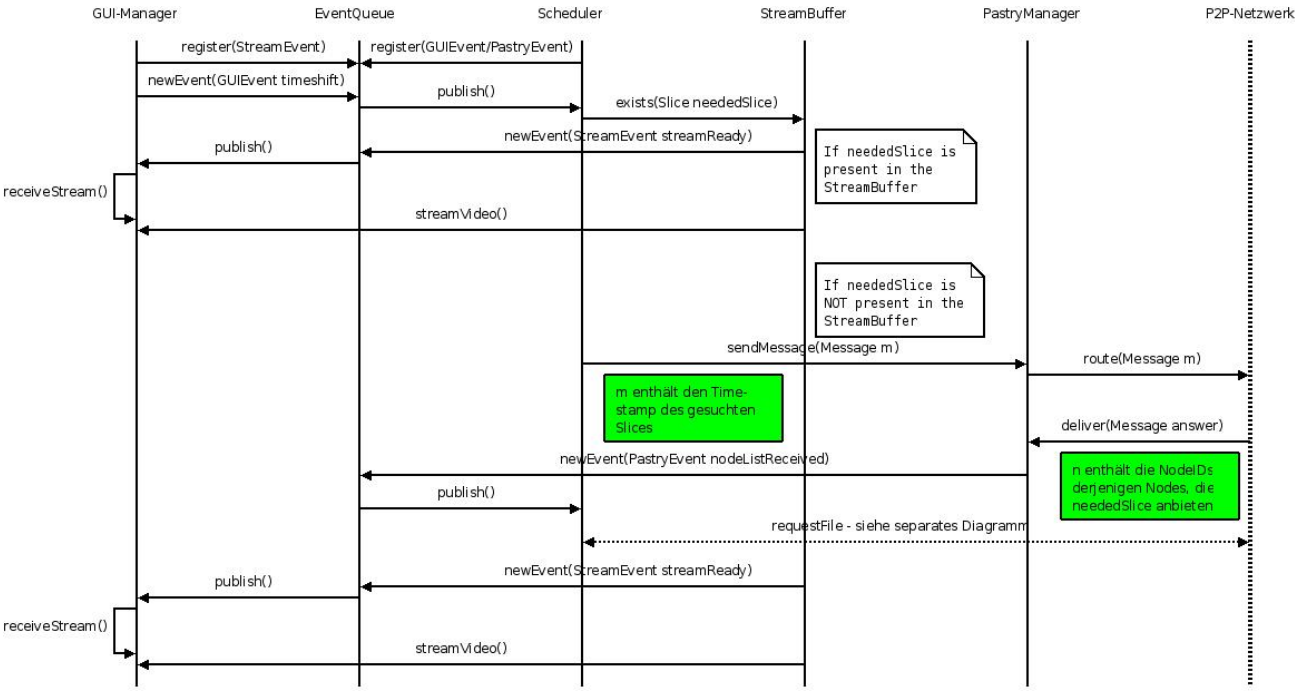


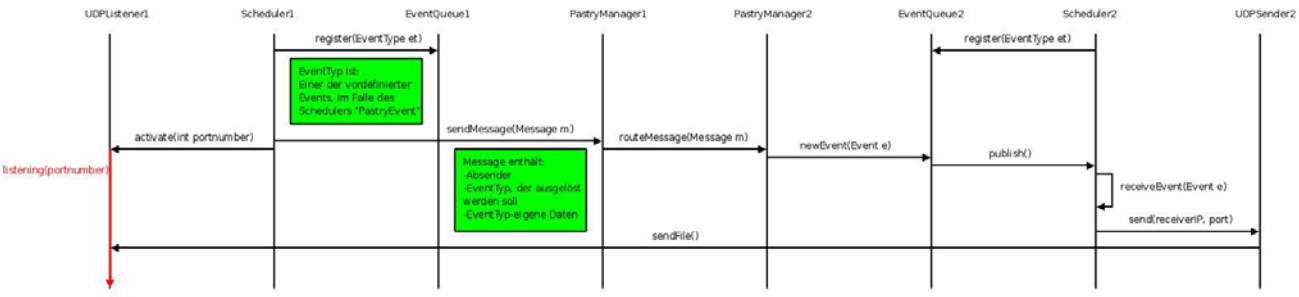
Illustration 10: Abhängigkeiten auf Ebene Package (Methoden Aufrufe & Zugriff auf Felder).

Sequenzdiagramme



Sequenzdiagramm 1: Von dem GUI via Netzwerk zum Videostream.

Die Registrierung von Scheduler und GUI-Manager beim EventQueue ist eigentlich nicht Teil von, aber essentiell für den Ablauf. Die eigentliche Sequenz beginnt mit dem GUIEvent „Timeshift“, der vom GUI-Manager ausgelöst wird.



Sequenzdiagramm 2: Beschaffung eines Timeslices via Scheduler über das Netzwerk.

Wiederum sind die Registrierungen nicht Teil der Sequenz, sondern nur der Vollständigkeit halber eingezeichnet. Scheduler1 beginnt die Abfolge einerseits mit einem Slice-Request und andererseits dem Starten eines Listener-Jobs.

Use Cases

Use-Cases für die drei Szenarien, wenn ein Node einen Slice

- Aufnehmen will
- Aufgenommen hat
- Anfordern will

Variabeln:
1. Slice-Grösse s [in Sekunden]
2. Stream-Zeit t [in Sekunden?]
3. Slice-ID sID [Timestamp Aufnahmestart, auf Sekunden genau]
4. Eigene Node-ID nID [128bit-Zahl]
5. Neighbourhood-Liste nList [Pastry erstellt das in jedem Knoten: Liste der nächsten Knoten nach Node-ID]
6. Netzwerkgrösse nSize [Schätzung über die ungefähre Anzahl Knoten]
7. nNodeList [Wieviele Knoten-Listen geführt werden sollten]
8. Wartezeit, wenn eine Kollision auftritt tWait [in Millisekunden]

1. Aufnahme (Illustration 1)

Benötigte Variabeln:
Slice-Grösse s
Stream-Zeit t
Node-ID id
Netzwerkgrösse nSize
Neighbourhood-Liste nList

Jeder Knoten, der den Stream empfängt, prüft den Timestamp des Stream. Ist (t % s == 0), wird mit der Aufnahme begonnen wenn:
Der Timestamp in umgekehrter Reihenfolge (least significant bit wird zum most significant bit) zwischen nID und der des nächsten Nachbars auf der Neighbourhood-Liste liegt.
Der Grund für die umgekehrte Reihenfolge ist eine bessere Verteilung auf alle Knoten. Dadurch, dass das am schnellsten ändernde Bit die wichtigste Stelle ist, wandert „die Zuständigkeit“ schnell durch den gesamten Adressraum.
Beginnt ein Node mit der Aufnahme, schickt er eine Message an die nächsten (logn(nSize)+1) Knoten in nList, damit diese den Slice ebenfalls aufnehmen.
n sollte irgendwo im einstelligen Bereich liegen.

Dadurch wird sichergestellt, dass
Jeder Slice (mehrfach) aufgezeichnet wird
Kein Knoten über längere Zeit aufzeichnen muss

2. Registrierung aufgenommener Slices (Illustrationen 2, 3 und 4)

Benötigte Variabeln:
Stream-Zeit t
Eigene Node-ID id

Der Knoten versieht den gerade aufgenommenen Slice mit einem Dateinamen nID, der t entspricht.
Danach stellt er eine Anfrage an das Pastry-Netzwerk nach dem Key nID.
Existiert dieser dieser Key schon, erhält er als Antwort eine Referenz auf ein Datei-Objekte (für eine Knoten-Datei) im PAST-Netzwerk sowie einen Boolean objectLocked.
Ist der Key noch nicht vorhanden, wird er erstellt. Dabei ist objectLocked true.
Sind objectLocked false, darf das Objekt beschrieben werden. In diesem Fall wird als erstes objectLocked für diesen Key auf true gesetzt, um spätere Schreibzugriffe auf das Objekt zu verhindern.
Ist objectLocked true, darf das Objekt momentan nicht beschrieben werden. Die Anfrage wird für tWait suspendiert und dann wiederholt. Dies geschieht so lange, bis einer der Fälle 1 oder 2 eintritt.

Nach diesem Schritt ist sichergestellt:
Das Objekt gelocked ist.
Der Key existiert

Im Fall 1. wird eine neue Knoten-Datei erstellt, die die folgenden Einträge enthält:
nID

Diese Datei wird als zugehörigen Wert zum neu erstellten Key geschrieben. Anschliessend wird objectLocked auf false gesetzt, damit weitere Knoten die Liste erweitern können.

Im 2. Fall wird die Knoten-Datei, deren Referenz der Knoten erhält, abgerufen.
Diese wird um die id erweitert und wieder im Netzwerk verteilt. Anschliessend wird der Wert des Keys nID auf die neuen Datei umgeschrieben umgeschrieben.
Jetzt wird objectLocked wieder auf false gesetzt.

Nach diesem Schritt ist sichergestellt:
Alle Knoten-Dateien sind auf dem neusten Stand
Der Key ist wieder freigegeben, weitere Knoten können sich eintragen

3. Suchen eines Slices (Illustration 5)

Wird die Timeshift-Funktion aktiviert, sucht der Client im Netzwerk nach dem entsprechenden Slice.

Wird der Key nicht gefunden, wird der entsprechende Slice als „nicht vorhanden“ markiert und der Vorgang abgebrochen.
Ist der Key vorhanden, wird die Datei, die in dem Key angegeben ist, abgerufen. Anschliessend kann der Slice von einem der Knoten in der Liste angefordert werden.

Fehlerbehebung:
Ist eine der Knoten-Dateien nicht vorhanden, könnte der Knoten diese erneut erstellen. Knoten, die in der Liste aufgeführt, aber nicht erreichbar sind, könnten daraus gelöscht werden. Dann könnte z.B. der gerade anfordernde Knoten sich in die Liste eintragen und den Slice fortan auch anbieten, um eine gewisse Redundanz wieder herzustellen.

Netzwerk-Funktionen, die benötigt werden:

-int getNumberOfNodes()
Möglichkeit, die ungefähre Anzahl der Nodes im Netzwerk zu schätzen

-Object getNeighbourhood()
Gibt die Neighbourhood-Liste zurück

-void recordSlice(nodeID, startingTime)
Beauftragt den Knoten nodeID, den Slice beginnend bei startingTime aufzunehmen

-void createKey(keyID)
Erstellt den Key keyID im Pastry-Netzwerk

-void setValue(keyID, value)
Setzt den Wert von keyID auf Value

-Object getValue(keyID)
Sucht den Key mit der ID keyID und gibt den dazugehörigen Wert zurück

-boolean lockKey(keyID) / unlockKey(keyID)
Setzt object/Locked für keyID auf true. Return Value ist, ob die Aktion erfolgreich war, oder ob es zu Überschneidungen kam

-void saveFile(file)
Legt die Datei file im PAST-Netzwerk ab

-getDSice(sliceID, nodeID)
Holt sich den Slice sliceID vom Knoten nodeID.

GUI Prototyp

Das GUI besteht aus einem Hauptbildschirm, der nach Bedarf vergrössert wird und das gewünschte Panel anzeigt. Es existieren drei Panels, das Settings-Panel, das Player-Panel und das TimeShift-Panel.

Im Settings-Panel kann man unteranderem IP und Port bestimmen und dann den VLC-Player starten, welcher im Player Panel angezeigt wird. Im Player Panel kann man den Stream steuern, die Lautstärke regeln zu den Settings und in den TimeShift-Modus wechseln

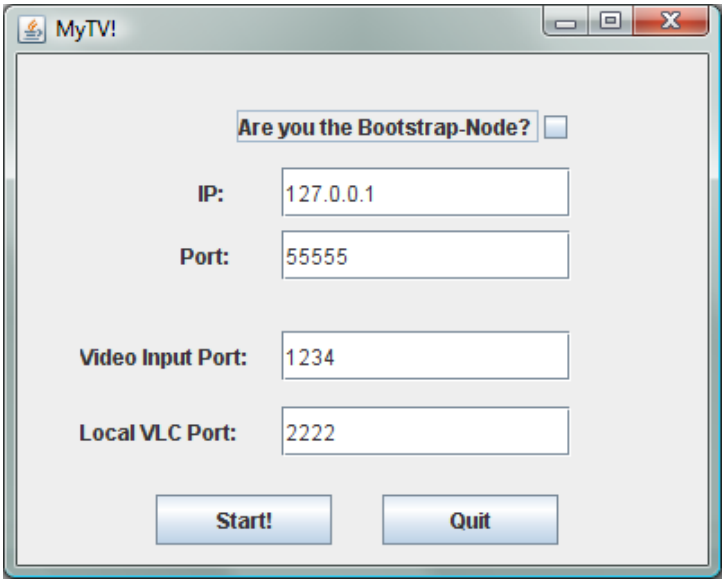


Illustration 11: Einstellungen

Illustration 11 zeigt das Settings-Panel mit der CheckBox für den BootstrapNode. Wenn man nicht der Bootstrap ist muss man IP und Port des Bootstraps angeben. Weiter müssen noch Video Input-Port und Local-VLC Port angegeben werden.
Alle Angaben sind schon mit Standardwerten gesetzt.

Illustration 12 zeigt den Player mit Start, Stop, Pause und Volume Kontrolle. TimeShiftButton um in den TimeShift Modus zu kommen und um dieses wieder zu verlassen.

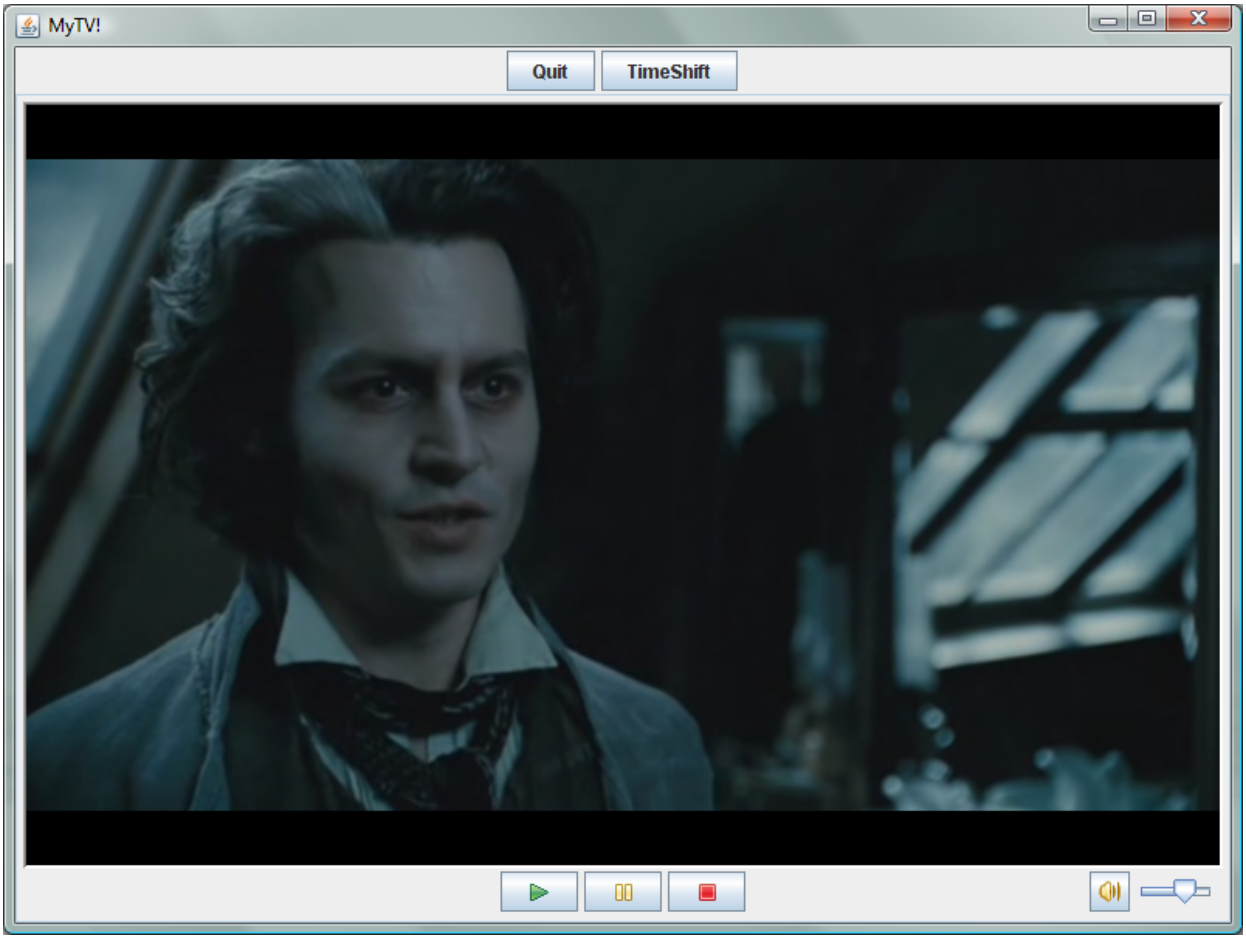


Illustration 12: Hauptdialog

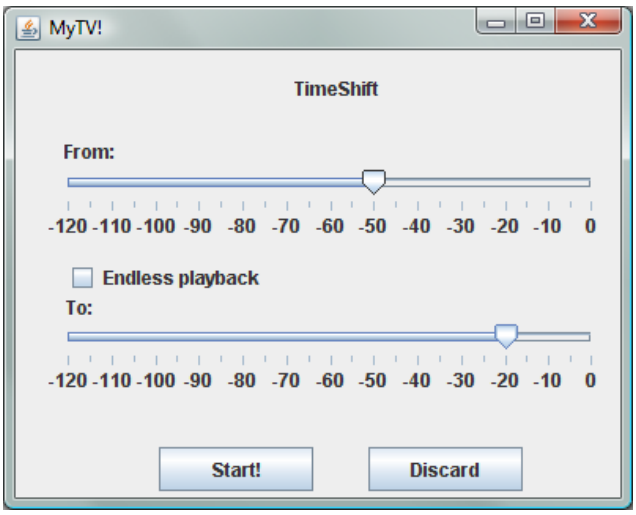


Illustration 13: Time-Shift Dialog

Illustration 13 zeigt das TimeShift Panel. Dieses dient dazu, den Zeitframe auszuwählen, den man anschauen will. Man kann dazu bis zu zwei Stunden alte

Aufnahmen des Streams abspielen sowie mit dem zweiten Slider den Endpunkt setzen. Wenn man die Checkbox aktiviert, wird der zweite Slider deaktiviert und der Stream läuft ab dem gewählten Startpunkt unendlich lange weiter
Falls der Timeshift abgebrochen werden sollte, so kann die From-To Zeit auf Null gesetzt werden, bzw. die endlose Wiedergabe deaktiviert werden.

Housekeeper und PAST-GC

Während dem Betrieb fallen laufend neue Daten an: Pastry/PAST-Nachrichten werden ausgewertet und teilweise gespeichert, Timeslices werden im Memory, dann auf der Festplatte hinterlegt. Diese Datensammlungen sind nur für eine bestimmte Zeit gültig und müssen regelmässig gesäubert werden. Umgesetzt wurde dies durch folgende Mechanismen:

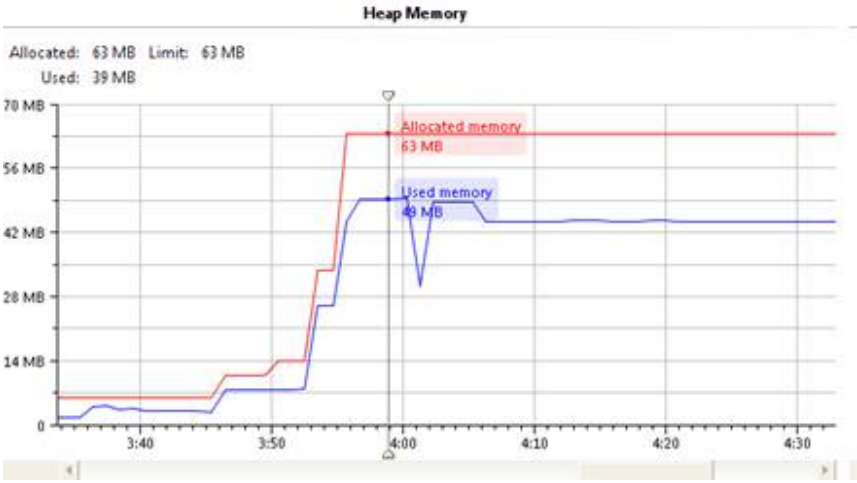
- Pastry Message-Tracker und PAST Message-Tracker: Alle 5 Sekunden werden alte Einträge aus den Tracker-Tabellen gelöscht. So etwa abgelaufene Lock-Messages oder PAST Nachrichten.

- PAST Storage: PAST wurde mit aktivierten Garbage-Collector implementiert (PAST mit PAST-GC Erweiterung). Dieser löscht Daten nach 2.5h aus dem System.

- Gespeicherte Slices: Der Streambuffer besitzt einen Housekeeper-Thread, der alle 60 Sekunden veraltete Slices von HD und Memory entfernt. Slices laufen nach 2.5h ab.

Systembelastung

Da während dem Timeshift konstant drei 18MB Video-Slices im Memory gehalten werden, steigt die Auslastung gleich nach Start der Timeshift Funktion rapide an. Das Niveau bleibt anschliessend sehr stabil bis zum Ende des Timeshifts bestehen.



Da während dem Betrieb ungefähr 5-7 Threads laufen, ist ein Mehrprozessorsystem empfohlen.

Beim Beenden des Programms werden alle gesammelten Daten gelöscht. Im Konsolenmodus wurde ein Shutdown-Hook erstellt. Somit kann die Anwendung mit Ctr-C beendet werden.
Nachdem die 2.5h Aufnahmezeit erreicht sind, bleibt der Speicherbedarf konstant bei maximal folgenden Werten:

```
FreeMemorySize: 11484 KBytes
MaxMemorySize: 133234 KBytes
TotalMemorySize: 35009 KBytes
NumberOfMemoryStoredSlices: 150
FreeDiskSize: 2226561 KBytes
MaxDiskSize: 0 KBytes
TotalDiskSize: 2374506 KBytes
NumberOfDiskStoredSlices: 149
```

(Memory enthält immer das aktuelle Slice, bis ein neues beginnt und das alte auf die Platte geschrieben wird – daher ein Slice Differenz)

Installation

Im Ordner /SVN/dist befindet sich ein Update-Skript. Dieses lädt die aktuellste MyTV.for.you.jar Datei auf die EmanicsLab Server.

Das File MyTV.for.you.jar besteht aus allen Libraries und Class Files.

Gestartet wird das Programm in der Linux Shell folgendermassen:

1) Direkt per Jar: java –jar MyTV.for.you.jar <Parameter>

2) Falls vorgängig „jar xf JAR-FILE“ ausgeführt wurde:

```
[uzh_p2pgroup3@emanicslab2 P2P]$ ~/java csg/p2p/challenge/group3/Starter help
[Starter] Application started.

---------------------------------------
    -->  MyTV! Console Arguments  <--

 java csg.p2p.challenge.group3.Starter [Option, ..]

 Options:
 'linux'        Loads Linux Configuration
 'windows'      Loads Windows Configuration (Default)
 'debug' or 'moredebug'  Guess what..
 'nogui'        Only Shell mode + streaming to the localPlayerPort
 'localport=PORT'       VLC localPlayerPort
 'streamport=PORT'      Input Stream Port
 'tcpport=PORT'         TCP Listener Port
 'isbootstrap'          Activates this node as a bootstrap node
 'bootstrapip=IP'       IP Address (111.222.333.444)
 'bootstrapport=PORT'
 'packetheaderexists'      Guess what..
 'inputpacketsize=SIZE'    Size of the input UDP packets in bytes (Linux: 1324)
 'transferpacketsize=SIZE' Size of the internal UDP packets in bytes (Linux: 1316)
 'resistant'    Activates a more resistant p2p mode
 'help'         This help output

---------------------------------------
```

Als Bootstrap Node mit Debug-Output:

```
~/java -Xmx128m csg.p2p.challenge.group3.Starter isbootstrap streamport=5003
tcpport=6000 inputpacketsize=1324 transferpacketsize=1316 nogui debug
```

Als normaler Node mit Debug-Output:

```
~/java -Xmx128m csg.p2p.challenge.group3.Starter bootstrapip=192.41.135.210
streamport=5003 tcpport=6000 inputpacketsize=1324 transferpacketsize=1316 nogui
debug
```

Gestartet wird das Programm mit der SWT-GUI folgendermassen:

```
~/java csg/p2p/challenge/group3/Starter
```

(This page is left blank intentionally.)

University of Zurich
Department of Informatics

ifi

P2P Challenge Task 2008

Solution of Group 4

Authors:

Kevin Leopold, Clemens Wilding, Marc Körsgen

Spring Term 2008

1. INTRODUCTION

Watching television on a PC is an interesting and enjoyable practice. Applications like Zattoo[1] allow that functionality with a big number of available TV channels to a broad user community. But what if you want to watch one of these channels in a timeshifted fashion? Timeshifted means, that a user can go back in time to watch earlier scenes. For example, if you're watching a football match and you want to see the goal again, you just select how much time you want to go back and the scene appears on the screen. The technical solution behind this concept is to store the recorded video on the hard drive. To save space on the local hard drive and make it possible for users that start the application after the event happened, that they wanted to see, we distribute the recorded video stream on different machines using the peer-to-peer network. Peer-to-peer systems are an ideal solution to share already available resources, like space and bandwidth and therefore present an optimal source for our application.
In the next two chapters you'll find a more detailed description of the task and the needed requirements. Afterwards, we'll explain our concept of our work, how the program is built and how the communication between the peers is organized. The following chapter tells about our work as a group of programmers and the main problems of the task. The last chapter consists of two parts: firstly our conclusions will be presented and secondly some suggestions for future developments are proposed.

2. TASK DESCRIPTION

The P2P Challenge Task for the FS08 semester is to design and prototypically implement time-shift functionality for a live video stream using P2P concepts. The time-shift functionality shall allow a live stream viewer to pause the reception of the stream and shall also allow the replay of particular parts of the stream.

The nodes in the P2P network represent individual users watching a particular TV Channel (video stream). Storing locally the whole data received by each peer individually is an inefficient way of designing a time-shift function, as each peer will have to store the whole content. Another disadvantage of a "local" solution for time-shift is that a user may only replay parts that were broadcast while his application was running (e.g. if a user would start the application just after a team scored during a football game he would not be able to replay the goal). A P2P-based approach addresses these issues by distributing the task of recording and storage of the video stream across multiple nodes. Each node is responsible of recording and storage for particular parts of the video stream. In this case a replay task consists of finding the peers responsible with the recording for the requested timeslots and getting the video stream from them.

The main goal of the challenge task is to design and prototypically implement a P2P-based recording, storage, and replay application for live video streams. Each node in the P2P network shall receive the video stream as UDP packets (the live stream transmission is provided). A VideoLan Client (VLC) shall be used as a video player on each node. Your time-shift application should be able to do the following:

[1] Zattoo is an application for watching television on a PC using a peer-to-peer network. http://www.zattoo.ch

- receive the incoming UDP stream from the video server
- buffer the live stream and send it to VLC via UDP
- store parts of the live stream for 7200 seconds (2 hours)
- request the P2P network for video replay for maximum 7200 seconds (2 hours)

3. REQUIREMENTS

The following requirements are supposed to be met by the application.

- Live/Timeshift switch: The solution shall enable the user to switch dynamically from live streaming to time shifted streaming and back.
- Realtime access: Stream buffering and access for time shifted streams shall happen in realtime.
- Timeshift period: The solution shall allow for the storage of at least two continuous hours of time shifting.
- Robustness: The solution shall be robust against node or link failure during timeshift.
- Multiple Client Serving: The solution shall enable each peer to serve multiple clients at the same time, if it has sufficient bandwidth to do so.
- P2P mechanisms: The solution shall be based on Peer-to-Peer mechanisms using a structured overlay network.
- Decentralization: The solution shall not contain any central elements.
- Compatibility: The solution shall be executable on the provided test setup machines (Linux, Java 1.6).
- Library and tools: The solution shall be based on libraries and tools which allow for publishing under GPL or another comparable open software license.

4. CONCEPT

The basic concept of our application will be presented in this chapter including the underlying model and design for the prototype.

4.1. MODELING OF THE STRUCTURE

4.1.1. THE PLANNED MODEL

The planned model (see Fig. 1) failed our requirements in some points, but the main model was acceptable. There were obsolete classes like the MessageManager or the LoadBalancingManager, but some classes like the FileProvider or the StreamPlaybackManager are missing. There are also some naming conflicts, for example the GUI consists of three instead of two classes with more useful names.

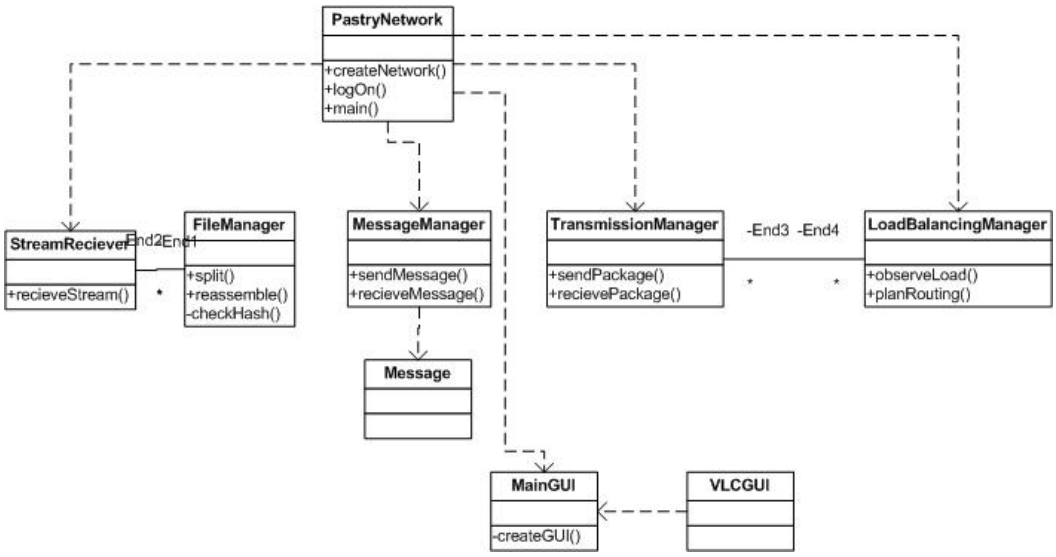


Fig. 1: The planned model

4.1.2. THE ACTUAL MODEL

Our source code consists of twelve classes (see Fig. 2). The main class is the P2PTimeshift, which contains instances of several classes. The most important of these is the implementation of the P2PManager, which manages all peer-to-peer-related actions, like creating the pastry network or providing methods to retrieve video chunks.

The other classes, which are instanced by the P2PTimeshift are the FileProviderServer, which continuously runs and checks if someone wants to retrieve a file, of course an instance of the GUI, more precisely of the StreamPanel, an instance of the StreamRecordManager, to record the video stream, save it to data-files and extract the timestamp information and an instance of the StreamPlaybackManager, to play received data-files or the live video stream in VLC.

The peer package consists of seven classes, four of which were already explained in the previous paragraphs. The others are the StreamData class, which implements the FreePastry class ContentHashPastContent and contains the association between video chunk and peer that saved it (given by address and port), the FileProvider, with its help the data-files can be sent through a TCP connection from one peer to another and the P2PApplication, which represents, as the name already says, a timeshift application.

The GUI-package contains three classes: the main class is the StreamPanel, which extends the JFrame class. The other two classes in the package each create a JPanel in the StreamPanel. The StartPanel creates the View panel, in which the lifestream or the timeshifted stream can be watched. The SettingsPanel creates the Settings panel, which provides possibilities to play a file from the hard disk or from a UDP-stream from the Internet.

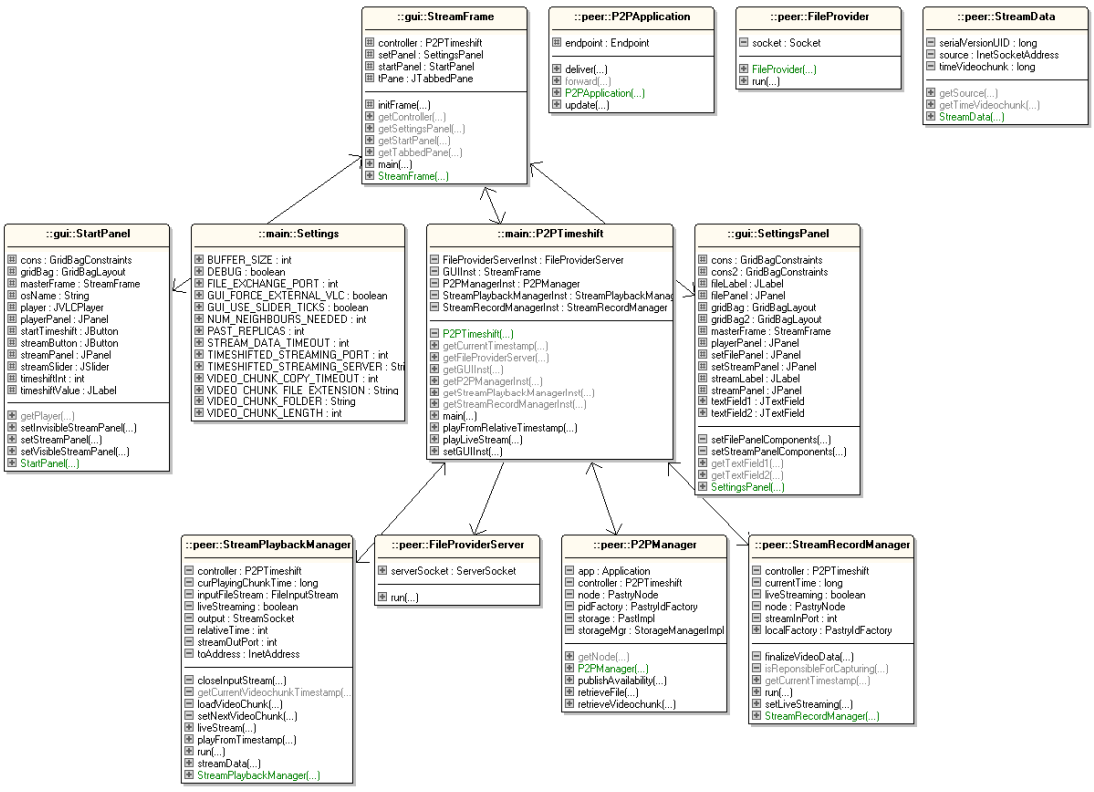


Fig. 2: The actual model as it was implemented

4.2. COMMUNICATIONS PROTOCOL

Our solution distinguishes between the following two roles. Every peer in the network can take multiple roles at a time.

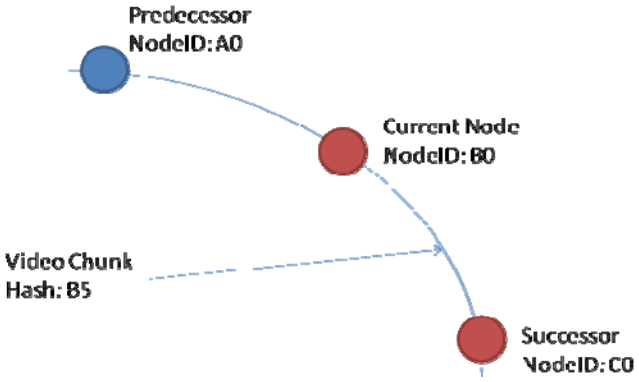


Fig. 3: Responsibility for a node to save a given video chunk

Content Provider (CP). The content provider is responsible for the reception and the monitoring of the video stream and locally saving those parts of the video that the current peer is responsible for. All CPs in the network therefore allow a content consumer to use the timeshift function by

making all needed video chunks available to all peers in the network. The mainly CP fullfills the folowing tasks:

1. *Analysis of the incoming video stream*
   The CP continuously monitors the incoming UDP video stream. It parses the timestamp information from each packet and checks whether the received time represents the start of a new video chunk. If that is the case, the CP calculates the hash of that timestamp (*H*) and checks if *H* lies between the peer's own NodeID and the NodeID of the *N*th neighbour (in the FreePastry space). *N* is a well-defined parameter with a common value of 2. If the latter check validates, the node procceeds with the buffering of the video data (see task 2). The task of checking whether a node is responsible for recording a video chunk is shown in Fig. 3.
2. *Buffering the video data for a given length*
   The CP analyses all packages received, starting from a given timestamp *T*. It then separates all following timestamps from the video data and discards those timestamps. The video data instead, is buffered in the local memory until a new timestamp T + video chunk length is reached.
3. *Saving the buffered data / video chunk to the disk*
   The previously buffered video data is saved the the hard drive using specific PAST functionality.
4. *Publish the respective video chunk ownership information to the network*
   The CP adds a new entry into the FreePastry-DHT containing its own NodeID/IP and the hash of timestamp of the previously saved video chunk.
5. *Transfer video chunk to any requesting content consumer*
   As soon as a request for a given video chunk is received by the CP, it responds with the respective video chunk (provided it was previously saved the the hard drive).
6. *Delete local video chunk*
   A previously saved video chunk is deleted from the hard drive. This task is periodically executed to make sure the node does not run out of memory and the network will only save the video data for the requested amount of time (see requirements).
7. *"Unpublish" video chunk ownership information*
   If a local video chunk was deleted, the FreePastry ownership entry (created in task 4) will be deleted, as well.

Content Consumer (CC). A node takes the role of the content consumer as soon as it starts timeshifting. The content consumer localizes and retreives a video chunk from the P2P network to be able to show it to the user. The CC has the following main tasks:

1. *Determining the node (CP) which owns the requested video chunk*
   The CC uses FreePastry to retrieve the information of which peer (which CP) owns the video chunk the CC currently needs.
2. *Receive the requested video chunk from the CP*
   The CC contacts the previously determined CP and requests the video data file from that CP. It receives the requested video chunk file and saves it to the hard drive.
3. *Locally stream the video data from the video chunk to VLC*
   The received video chunk is streamed locally to a VLC instance for the user to be able to see the time-shifted video.

The described protocols rely on the usage of video chunks, which shall be described in more detail.

Video Chunk
A video chunk contains the following information:

- *Timestamp* of the time the video chunk began
- *Video data.* All video chunks have a well-defined length, with common value of 60 seconds. The responsibility of saving a video chunk lies on those nodes who have a NodeID near to the hash of the video chunk's timestamp (see section "Content Provider, task 1" for detailed information on video chunk responsibility).

5. IMPLEMENTATION

5.1. DIVISION OF THE TASKS

We divided the tasks more or less evenly to the team members, as visible in the table.

Name	Tasks
Clemens	Modeling Implementation of the file transfer Leading writer of the report
Kevin	Communications protocol Implementation of: main architecture, stream recorder, stream player, P2P overlay connection (FreePastry), file transfer Expert programmer and response point for questions
Marc	Modeling of the GUI Implementation of the GUI
all	Project plan Bugfixing and Testing Writing the report and presentation

5.2. TIMELINE

The following table represents an overview of the given timeline.

Date	Milestone #	Tasks
03.04.08	1	Project Planning Table of Contents for the Report
24.04.08	2	Requirements Design

		Prototype
22.05.08	3	Final Code Report
06.06.08	4	Presentation Revised Report

5.3. PROBLEMS OF THE IMPLEMENTATION

One of the main problems was the usage of different operating systems. Two of us, Clemens and Kevin, used Windows to implement to write the code and to test the application. On Mac OS X, which the other developer, Marc, used, he had severe problems with the GUI, namely with the DJNativeSwing and JVLC, so he had to switch on Windows too. The behavior on the Linux machines however, was again different from the behavior of the program on the windows machines, so we had to debug for several hours to make everything work. The GUI does still not work perfectly on the Debian machines, so we switched to the external VLC player, but not on the Windows machines. This problem couldn't be solved by using either the DJNativeSwing or by using JVLC.

The last problem was the file transfer (video chunks) from one peer to another. The straightforward solution we first thought of was to use PAST (based on FreePastry) to store the files on the overlay network in a distributed manner. Unfortunately, the file transfer could then not be done efficiently, since it would have caused a lot of overhead (e.g. storage of video data in objects and serialization of objects). We later decided to implement the file transfer via TCP. Another solution would have been to use UDP to stream the chunks directly from peer to peer, which would have been a little more efficient, since for video streaming it is not so important that all packets reach its destination. However, we started creating a TCP file transfer prototype, and succeeded pretty fast. But more complicated was integrating it into our almost complete program, which worked already locally. However, finally we also managed that, by creating a new class FileProviderServer, which runs all the time and listens for incoming requests for files and sends those files back to the requesting peers.

6. CONCLUSIONS

6.1. CONFIDENCE IN OUR SOLUTION

All in all we're content with the application. It meets pretty much all of the requirements and works very stable on Windows. The application also works on Linux, but the playback is sometimes a little fast and choppy. The program is more or less light-weight, which means it doesn't consume so many resources and looks pretty cool.

6.2. IMPROVEMENTS AND FUTURE DEVELOPMENTS

The application could be further improved by compressing the video chunk files before transferring them across the network. That would cause less bandwith usage and faster reception

of files. However the files would have to be compressed and uncompressed, which needs time, as well. Another improvement considering file transfer is also possible by checking whether a file was fully received or not. If one of two nodes break down during a file transfer, the file will never reach the receiver and the latter has to wait for the next video chunk. That procedure could be further improved, by requesting the file from another peer, if the file transfer failed. The deletion of old video chunks would also improve the application. Chunks that are older than a given threshold (e.g. two hours) could be deleted and therefore free (hard drive) resources.

A future development could be the reception of several streams and switching between the streams to watch several channels (like Zattoo). The problem with this solution is the massive use of bandwidth, but it would be a really great feature and would probably meet the approval of many users. Other "nice-to-have" features would be different skins for the GUI or the ability to make screenshots directly from within the program.

Not really a feature, but still a future development would it be, if we could manage to run not only the external VLC player on Linux, but the built in VLC player in our software. This task could be done together with the reworking of the GUI and the addition of the skins.

Not a development, but a suggestion, if the program would be distributed commercially. More tests would be useful, to stretch the program to its limit. Bandwidth tests and stress tests could help to detect weaknesses in the program and would lead to less resource consuming solutions.

7. INSTALLATION INSTRUCTIONS

The code can be run from a terminal application using the following commands (provided java is installed correctly and the path to the java bin directory is included in the PATH-variable of the environment):

java P2PTimeshift [streamInPort] [bootAddress] [bootPort] [localBindPort] [noOfInstances] [showGUI]

The arguments for the application are described below:
streamInPort. The local port on which the input video data is streamed to.
bootAddress. The address of a FreePastry bootstrap node.
bootPort. The port, which should be used to contact the same FreePastry bootstrap node.
localBindPort. The local port to use for FreePastry communication.
noOfInstances. The number of Instances to start locally.
showGUI. Whether to show a graphical user interface or not (e.g. if a peer is only used for serving video data)