HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

A real-time Streaming Protocol for large-scale P2P Networks

Bachelor Thesis

by

Christopher Michael Probst

born in
Solingen

submitted to

Technology of Social Networks Lab
Jun.-Prof. Dr.-Ing. Kalman Graffi
Heinrich-Heine-Universität Düsseldorf

October 2014

Supervisor:
Jun.-Prof. Dr.-Ing. Kalman Graffi

Abstract

This thesis concentrates on implementing and evaluating different distribution algorithms for large data transfers in heterogenous peer-to-peer networks, especially for incremental transfers, which yields the possibillity for peer-to-peer video streaming. The main focus is on 1:N scenarios where only one peer has a given data set completely and N peers try to distribute this data set as fast as possible, though other scenarios are implemented and evaluated as well.

The main problem is the choice of the best distribution algorithm, which should be able to use most of the available network bandwidth of all participating peers. In a traditional client/server system the server uploads the data to all clients sequentially, which means that only the server upload bandwidth is used but none of the client upload bandwidths.

Peer-to-peer networks in combination with efficient distribution algorithms can help to solve this problem. Those algorithms are always based to the same technique where all participating peers load specific chunks from the peer with the whole data set and distribute those chunks among themselves. This way the upload bandwidths of the peers are not idle. The difficulty is the choice of the specific chunks the peers download and the tuning of parameters like chunk count and chunk size. Good algorithms are also flexible enough to adjust themselves to changing network conditions.

In the course of this thesis an application was developed which implements a variation of the BitTorrent protocol, I call this variation Chunked-Swarm-Distribution (CSD), which is better suited for the needs of this thesis in terms of performance and time, a logarithmic distribution algorithm and a sequential client/server disitribution algorithm for comparison. The application is implemented in Java using the Netty framework, a high performance network library. This framework offers the possibillity to run simulated network connections which is great for monitoring and evaluating different distribution algorithms since real networks add too much overhead for testing. The application implements a couple of comparing benchmarks to evaluate the quality of the algorithms used.

The benchmark results show that the CSD algorithm is able to almost fully load all participating peers which means that if a transfer from peer A to peer B takes T seconds, the time taken for the complete distribution for N peers also is about T seconds, in practice it is often a bit over T seconds, which is caused by meta data overhead. This means that the number of participating peers is, theoretically, of no importance. In practice more peers means more meta data, but with compression and efficient data formats those influences can be minimised.

Acknowledgments

A lot of people supported me during my work on this thesis to whom I wish to express my gratitude.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 2

Related work

Chapter 3

Theory

Chapter 4

Architecture

This chapter explains the architecture of the software in detail. From the very first beginning it was one of the main goals to achive high modularity so that individual parts can be enhanced or replaced easily. The software is seperated into the framework and the application part which in turn consist of multiple modules. Figure 4.1 shows the architecture in detail.

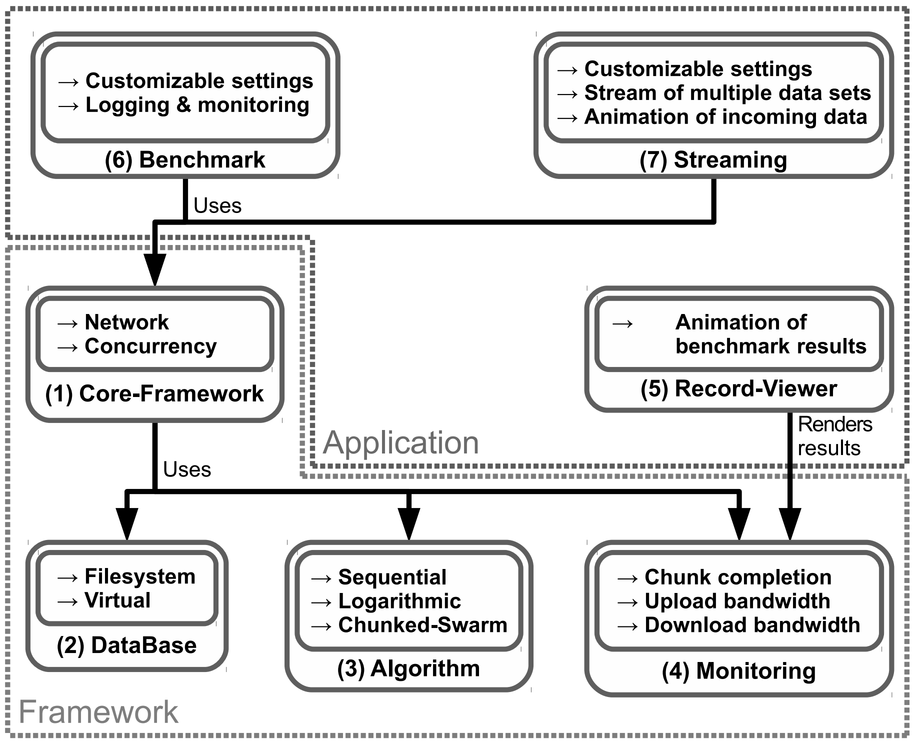

Figure 4.1: Software architecture

The application part depends on the Core-Framework and the Monitoring module. It contains three modules which are Benchmark, Streaming and Record-Viewer.

The Benchmark module is the most important application part because it can help to create, monitor and evaluate different scenarios in terms of performance and efficiency. All measurements were done using this module. The Streaming module is related to future work and demonstrates the possibility for an incremental stream of data sets. The Record-Viewer module can render and animate results taken from the Monitoring module which is part of the framework part and thus only depends on it.

The framework part contains the Core-Framework, DataBase, Algorithm and Monitoring module. While the DataBase, Algorithm and Monitoring module can be replaced or even omitted the Core-Framework module is the most important module. It manages network, concurrency and the communication of the three other modules located in the framework part.

The DataBase module represents an interface for a generic data source which might be the filesystem or even completely virtual. This way the framework is able to transfer any kind of data. The Algorithm module contains the implementations of the concepts presented before which include a sequential, logarithmic and chunked-swarm algorithm. To implement and test new distribution algorithms only this module has to be modified. The Monitoring module records data like current bandwidth usage and chunk completion which are important data points for evaluating algorithms. This module creates csv files which can be plotted with tools like gnuplot and a log file which contains a chronological stream of events happend during the recording which can be rendered using the Record-Viewer module.

The next two chapters the framework and application modules are explained in detail. The order is bottom-up which means the Core-Framework module is explained first in section 5.1 and after this the DataBase, Algorithm and Monitoring module in section 5.2, 5.3 and 5.4 respectively. The application modules are explained in section 6.1, 6.2 and 6.3 referring to the Benchmark, Record-Viewer and Streaming module respectively.

Chapter 5

Framework Modules

5.1 Module: Core-Framework

5.1.1 Network

- explain leecher, seeder, peer, core concepts etc

5.1.1.1 Automatic Connect

Every peer can be both a leecher and a seeder. While every leecher has to manually connect to a seeder for the first time, after this a leecher is able to automatically connect to other seeders. This is because every leecher who is paired with a seeder announces the address of its seeder to all seeders it is connected with. The figure 5.1 show this in detail.

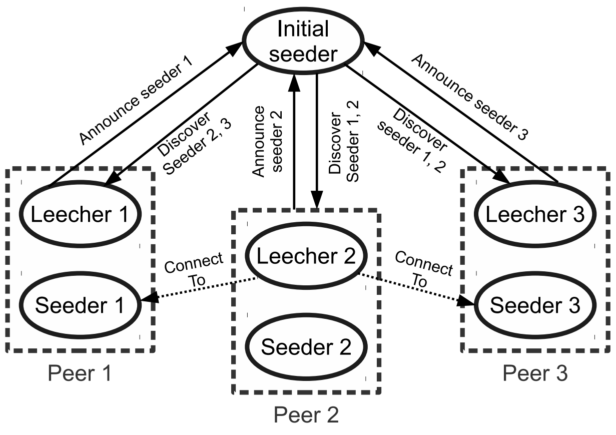

Figure 5.1: Seeder Discovery

Here Leecher 1, 2 and 3 announce the address of Seeder 1, 2 and 3 respectively to the initial Seeder. The initial seeder then broadcasts those addresses to all connected leechers. This way the leechers get to know each and can connect to the remaining seeders. Though in figure5.1 only Leecher 2 is illustrated, Leecher 1 and 3 connect to the remaining seeders, too. Of course this works recursively, so if one of the remaining seeders already has other leechers connected to it those will also be found. So in the end the topology is just a mesh with n^2 connections where n refers to the total number of peers where every leecher of each peer is connected to every seeder of all other peers. The figure 5.2 show this from the point of view of Leecher 1 from Peer 1 assuming there are 7 other peers.

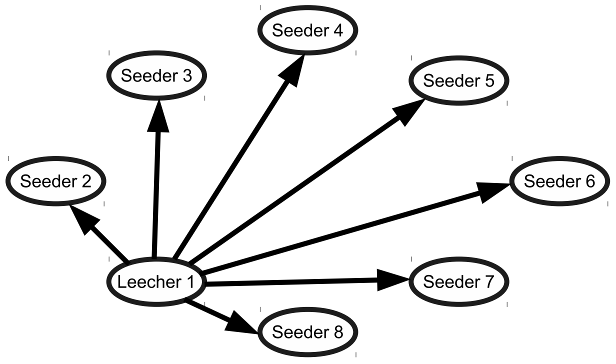

Figure 5.2: Mesh topology

While this topology is really great in terms of knowledge where every peer knows exactly what data sets the other peers have and thus can download data efficiently, it is also quite expensive and would not scale indefinitely. To improve scalability and reduce efficiency the number of connections per peer can be limited which limits the knowledge of each peer and thus the ability to make good decisions what to download from whom. This is explained later in section 6.1 in more detail.

If a connection for what ever reason terminates the framework is able to reconnect if the peer is still online. This will be detected by periodic address broadcasts of each seeder. So if for instance Peer 1 lost the connection to Peer 2 but both Peers are still connected to Peer 3 then Peer 3 will notify both that each peer still exist.

5.1.1.2 Meta-Data Announcements

After a Leecher connects to a Seeder the Leecher does not know anything about the Seeder. As previously explained in section 5.1.1.1 the Leecher discovers new Seeders through the Seeders it is currently connected to. The way the Leecher gets knowlege about what data sets a Seeder offers works a similiar way. A Seeder announces to all connected Leechers periodically what data sets it has to offer. It basically transfers a list of Meta-Data, which is explained further in section 5.2.0.3, so that every Leecher can update its knowledge about the Seeder.

As an optimization the Seeder only transfers new Meta-Data. So if a data set does not change at all, a Seeder will never transfer a second announcement of this data set. Another optimization is that while a Leecher downloads a chunk from a Seeder, the Seeder will also not transfer any announcements during this period, which is explained further in the next section 5.1.1.3

5.1.1.3 Download Requests

5.1.1.4 Nonblocking I/O

5.1.1.5 Transport: Local

5.1.1.6 Transport: TCP

5.1.2 Concurrency

5.1.2.1 Event-Loop

5.1.2.2 Lock-Free-Progamming

5.2 Module: DataBase

The DataBase Module represents a generic interface for any kind of data. While the Core-Framework transfers data between peers the data has to be read from and written to some kind of storage. This storage has the responsibility to verify incoming data in terms of consistency and validation. But to do so the storage has to know about the structure of the data. Because of that the metadata which describes a given data set is stored together with the data as key-value pairs.

5.2.0.3 Meta-Data

The Meta-Data contains information about a given data set. Those information are:

- Name
- Description

- ID
- Size
- Chunks
- Hash
- ChunkHashes

The Name and Description fields are used for human-related tasks. The ID has to be globally unique and is used for streaming purposes which will be explained later. The Size field determines the total size of the data set including all chunks which leads to the next field. The Chunks field is a bit set which simply stores one or zero bits for existing or missing chunks respectively. The length of the bit set determins the number of chunks a data set has. The Hash field contains a check sum generated by a strong hash algorithm like SHA from the whole data set. The ChunkHashes field does the same but for each chunk individually. This way the storage can verify single chunks as well as the data set in total. The reason why there is an extra Hash field for the whole data set is to reduce the chance of hash collision. Two data sets are equal if all of their fields are also equal.

5.2.0.4 Virtual

5.2.0.5 Filesystem

5.3 Module: Algorithm

5.4 Module: Monitoring

Chapter 6

Application Modules

6.1 Module: Benchmark

6.2 Module: Record-Viewer

6.3 Module: Streaming

Chapter 7

Evaluation

Bibliography

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 1.October 2014 Christopher Michael Probst

Please add here

the DVD holding sheet

This DVD contains:

- A *pdf* Version of this bachelor thesis
- All LaTeXand grafic files that have been used, as well as the corresponding scripts
- **[adapt]** The source code of the software that was created during the bachelor thesis
- **[adapt]** The measurment data that was created during the evaluation
- The referenced websites and papers