University of Zurich[UZH]

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

P2P Challenge Task 2007

TECHNICAL REPORT – No. 0

2007

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

ifi

David Hausheer, Thomas Bocek, Cristian Morariu, Gregor Schaffrath, Burkhard Stiller (Eds.)
P2P Challenge Task 2007
Technical Report No. 0
Communication Systems
Department of Informatics (IFI)
University of Zurich
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
http://www.csg.uzh.ch/

University of Zurich
Department of Informatics

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

P2P Challenge Task 2007

July 2007

TECHNICAL REPORT – No. ifi-2007.10

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

ifi

D. Hausheer, T. Bocek, C. Morariu, G. Schaffrath, B. Stiller (Eds.):
Technical Report No. ifi-2007.10, July 2007
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 2007

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 five groups of students in the summer term of 2007.

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-IST project EC-GIN currently investigates how P2P networking principles can be used to improve the network performance for Grid applications running on top of the Internet. One of the investigated scenarios in EC-GIN is the fast transfer of large files from one Grid node to another. Under the assumption that between the two nodes there are multiple paths which do not share a common bottleneck, a large file transfer application may benefit from a higher throughput if transporting the file over multiple paths in parallel. In case that the network does not provide any support for source routing or multi-path routing, the only way to achieve a multi-path file transfer is by relaying the data transmission over different intermediate Grid nodes at the edge of the network.

Content

The “P2P challenge task 2007” is the second 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 investigate the problem of distributing a large file over multiple paths as introduced above. The solution had to split the file to be transferred into several parts and transmit them over different paths via multiple intermediate peers. After transmission the file had to be fully intact again. Besides the use of P2P mechanisms to establish a fully decentralized structured P2P overlay network, the solution to be developed had to achieve a higher throughput than with a direct connection and needed to be robust against node or link failures during the transfer. Moreover, the solution had to enable the parallel transfer of multiple files at the same time.

The challenge task lasted nine 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 with some helper classes and could use several libraries and tools such as the FreePastry P2P overlay technology and the Iperf tool to measure the bandwidth.

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. The solutions they developed differ not only in the way routing decisions for the individual file chunks are taken, but also in the information these decisions are based upon and how this information is collected. While groups 1 and 4 applied a source routing concept, all the other groups followed the approach to determine the routes in a hop-by-hop manner. Independent thereof, a majority of groups decided to select the next hop in a deterministic manner. Only group 2 applied a purely heuristic decision concept, while group 5 found a good trade-off by chosing the routes in a weighted heuristic fashion, where weighting factors can, *e.g.*, be the local link bandwidth.

With respect to the information required to determine the next hop, however, almost all groups selected a different approach. It is obvious that the purely heuristic concept of group 2 requires no information at all. At the same time, the solutions of groups 3 and 5 consider only local bandwidth measurements, while group 4 decided to collect partial information about the routes between the sender and the receiver. The solution of group 1 even establishes full knowledge about all the link capacities. For collecting that information, group 4 applied a flooding-based approach, while group 1 selected to use a link-state routing approach. Finally, some load balancing mechanisms were developed. Here, group 1 based their solution on a global view of the topology, while group 2 selected a local view only. All other groups developed a feedback mechanisms to take that decision.

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. IP connectivity between the nodes was non-uniformely limited to simulate IP traffic limitations. Moreover, 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 2007” award was finally given to Group 2 with Martin Hochstrasser, Marc Haemmig, Philipp Kräutli, and Matthias Alder. Congratulations!

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

Zürich, July 2007

Contents

P2P Challenge Task 2007

David Hausheer, Thomas Bocek, Cristian Morariu, Gregor Schaffrath, Burkhard Stiller

Summer Term 2007

Task Description

This semester's task is the design and implementation of a Peer-to-Peer application that is able to transfer a given large file the fastest way possible from one host to another. Consider the scenario depicted in Figure 1.

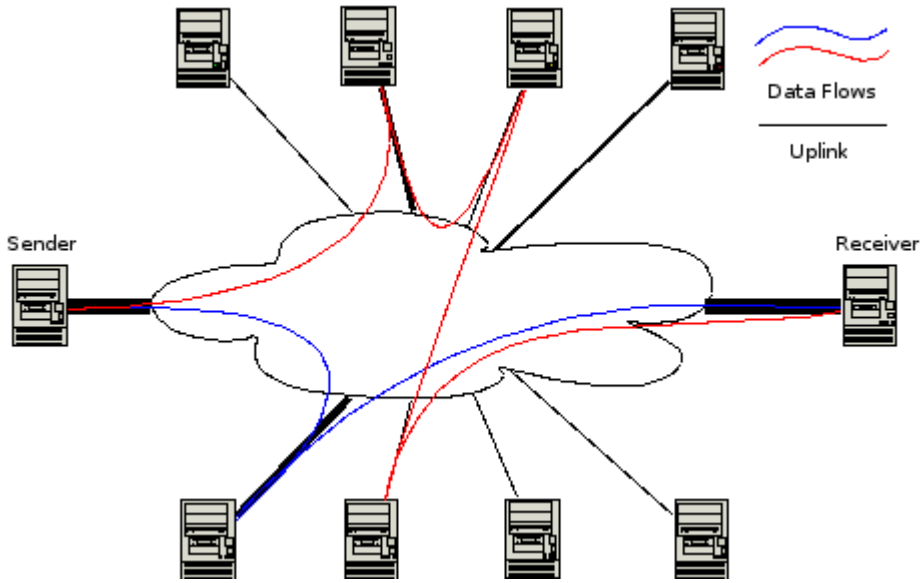


Figure 1: Challenge Scenario

There will be ten nodes interconnected via a network. Only the source and the destination will be free of artificial uplink bandwidth constraints, but the direct connection between them will be very slow. The other eight nodes will cover various parts of the network in their respective leaf-sets (up to a maximum of 4 nodes per leafset), but will have uplink bandwidth constraints.

The task, as mentioned will be to deliver a large file the fastest way possible from source to destination via the other nodes. In order to solve this problem, the underlying topology will have to be considered and files have to be split up in parallel partial transfers.

Application Requirements

- File splitting: The solution shall split the file to be transferred and transmit parts on separate application level streams via different sets of nodes.
- File integrity: The transmitted file shall be fully intact after transmission.
- P2P mechanisms: The solution shall be based on Peer-to-Peer mechanisms using a structured overlay network.
- Compatibility: The solution shall be executable on the provided test setup machines (Debian Linux, Java 1.5).
- High throughput: The solution to transfer a file shall be faster than with a direct connection.
- Robustness: The solution shall be robust against node or link failure during the transfer.
- Multiple file transfers: The solution shall enable the parallel transfer of multiple files at the same time.
- Decentralization: The solution shall not contain any central elements.
- 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 (CM, TB, GS, DH)	19.04.2007
Work (Room BIN 1.D.12 closed)	26.04.2007
Milestone 1 (CM, TB, GS)	03.05.2007
Work	10.05.2007
Ascension Day	17.05.2007
Milestone 2 (CM, TB, GS)	24.05.2007
Work	31.05.2007
Work	07.06.2007
Milestone 3 (CM, TB, GS)	14.06.2007
Presentations and Demos (CM, TB, GS, DH, BS)	21.06.2007

Support

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

- Groups 1 and 2: Thomas Bocek
- Groups 3 and 4: Cristian Morariu
- Groups 5: Gregor Schaffrath

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) 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.jar. Alternatively, you may choose Overlay Weaver (http://overlayweaver.sourceforge.net/), which implements multiple routing algorithms such as Chord, Pastry, or Tapestry. However, FreePastry is the only tool that will be supported for this task.

- Use a large file (> 2GB) for the file transfer. This could be a Linux distribution (http://mirror.switch.ch/ftp/mirror/debian-cd/3.1_r5/i386/iso-dvd/) or Wikipedia articles (http://download.wikimedia.org/enwiki/20070206/)
- Use of the Template which serves as a basis for the implementation of the P2P application. The source code of this template contains helper classes to split files and to measure bandwidth.
- For measuring the bandwidth, the tool Iperf is required. This tool needs to be downloaded, installed, and run on every peer. More details can be found in the template, in the constructor of the class PastryTemplate.
  Command line Iperf example:
  Host1, e.g. (192.168.1.1): iperf -s
  Host2, e.g. (192.168.1.2): iperf -c 192.168.1.1
- Additionally, the example application shown in the lecture is available as well.

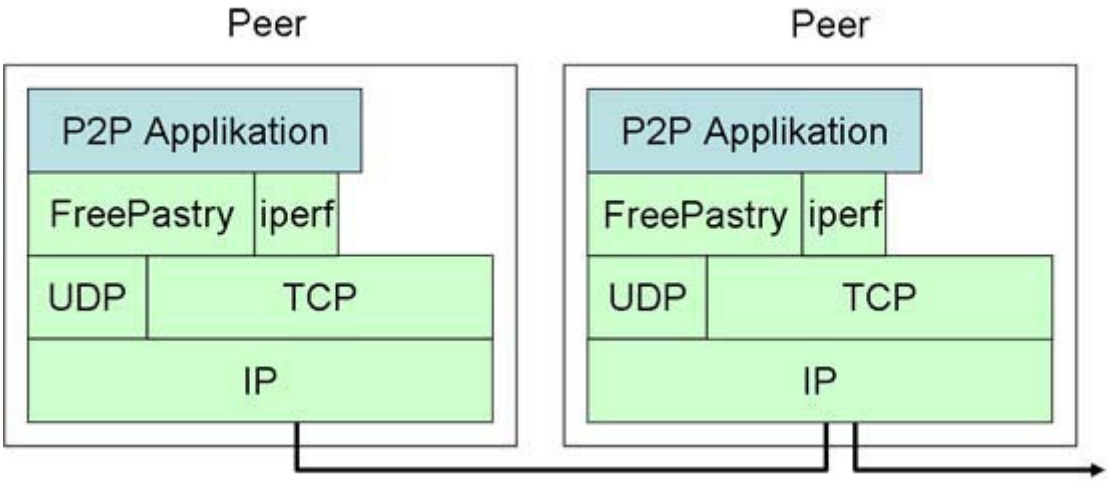


Figure 2: Layered model

Steps

Try to understand 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.

We recommend the following steps during the implementation task:

- Step 1: Design and implement a protocol for splitted file transfer
- Step 2: Design and implement a protocol that measures bandwidth
- Step 3: Find a scheme using relaying nodes to increase throughput, use the measured bandwidth to find suitable nodes. Design and implement the protocol to find relaying nodes. Put all parts together

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.

The following issues shall serve as a hint for the design of the application:

- The direct path is not always the fastest
- A node can join and become a relay
- A node that leaves might break a path
- A bootstrap node has to be present
- To find the fastest path, measurements from other nodes have to be obtained
- Transferring multiple files shall be possible. A unique identifier for the file parts/chunks is necessary
- Close nodes in the overlay address space are not necessarily close in the network

Testbed Infrastructure

During the challenge task and for the final demo the following testbed infrastructure will be available during exercise hours in BIN 1.D.12. A simplified network topology is depicted in Figure 3. During the final demonstration several nodes will be added to the testing network.

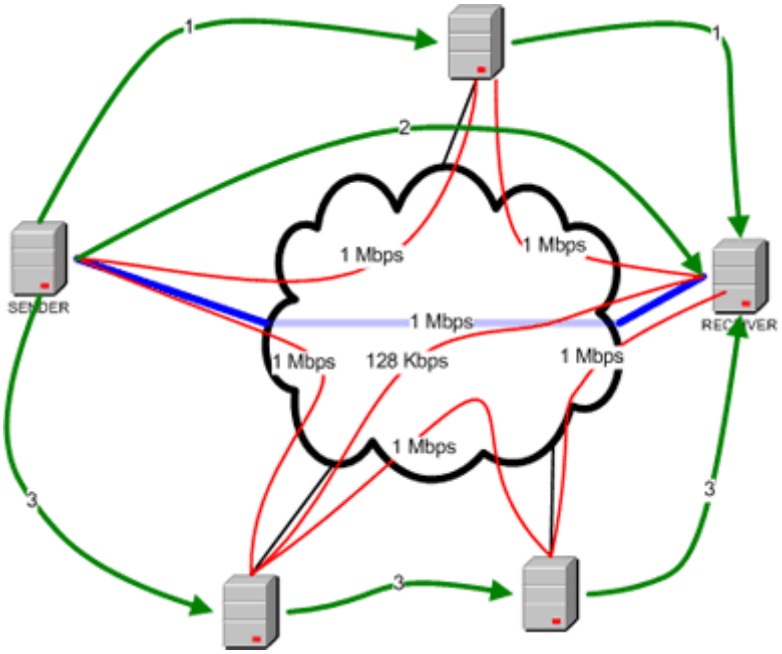


Figure 3: Testbed network topology

- The students shall have access to 10 Linux PCs (Pentium 4, 2.8 GHz, 500 MB RAM)
- IP connectivity between nodes shall be non-uniformly limited. (in the above figure red lines represent IP traffic limitations while the green arrows represent the fastest way to send a file from sender to receiver).
- The network configuration (IP address assignment) might be different in the final demonstration (compared to the one used during testing) so don't try to hard-code the data transfer paths in your application.
- The Layer 3 and the underlying Layer 2 topology should not influence the design of your application. The nodes may or may not share the same subnet of VLAN.
- One of the nodes shall have the following IP address: 192.168.200.10. This node may be used as bootstrap node. It's IP address will remain the same for the demonstration.
- In order to ease working in parallel on the same machine the groups are encouraged to use the following port ranges for their applications:
  - Group 1: 10000 - 10999
  - Group 2: 11000 - 11999
  - Group 3: 12000 - 12999
  - Group 4: 13000 - 13999

During development phase the groups may choose to use only a limited number of nodes for their tests. Students may access the testing testbed on Thursdays between 14:00 - 15:45. Remote access is possible (contact your group supervisor for details).

Remote Connectivity

For accessing the testbed remotely lo on to the testbed gateway (130.60.157.140) via ssh using the username and password provided.

```
ssh -CXY username@130.60.157.140
```

From here you may log on to one of the testing machines: pc51, pc52, pc71, pc72, pc81, pc82 using ssh (you don't have to retype the username):

```
ssh -CXY pc81
```

For starting eclipse open a terminal and type:

```
eclipse <ENTER>
```

Presentation and Evaluation

The challenge task will end on the 21.06.2007. 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. During the demo a large file shall be transfered from S to R over multiple paths. After transfer an MD5 function shall be applied on the file for consistency check.
- At least a minimal visualization mechanism shall be provided for view-ing the participating nodes. (messages of the following form are considered enough: "Node A: received chunk X; origin B; dest C; next hop D;")
- 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 (number of multiple paths, time required until the paths were discovered, total transfer time)
- 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 2007" award.

PEER-2-PEER SYSTEMS AND APPLICATIONS

Gruppe 1

Sommersemester 2007

Jan Bielik

Fabian Hensel

Franziska Wirz

Christian Vonesch

Inhaltsverzeichnis

Einleitung

In der Vorlesung Peer-2-Peer Systeme und Applikationen wird ein Grundwissen über den Aufbau und die Funktionsweise solcher Netzwerke vermittelt. Dieses gilt es nun anzuwenden, was im Rahmen einer Challenge Task gemacht werden soll. Die Aufgabe der Challenge Task beinhaltet ein Design einer Peer-2-Peer Applikation zu finden, mit der es möglich sein soll, grosse Datenmengen schnellstmöglich von einem Peer zum nächsten transportieren zu können.
Die Programmiersprache soll Java sein und als Programmierumgebung wurde Eclipse empfohlen. Auch soll diese Aufgabe mit Hilfe von Freepastry gelöst werden, welches helfen soll ein Overlay Netzwerk aufzubauen.
In dieser kurzen Dokumentation wird die Entwicklung der Aufgabe beschrieben, die Schwierigkeiten und Lösungsmöglichkeiten dargestellt und aufgezeigt, wie verschiedenste Faktoren einen Einfluss ausüben. Es gilt auf einige dieser Faktoren einzugehen wie beispielsweise file splitting, file integrity, Kompatibilität, Durchsatz, Robustheit und Dezentralisation. Aufgrund der Aufgabenstellung und der Schwierigkeit eine sehr grosse Datei über ein Peer-2-Peer Netzwerk zu verschicken, muss die Datei zuerst unterteilt werden. Es muss jedoch noch ermittelt werden in wieviele Teile die Datei gestückelt werden soll und wie diese Teile wieder aneinander gefügt werden, so dass die Datei beim Empfänger als Ganzes ankommt und die Integrität nicht verletzt. Dieser ganze Prozesse soll auch möglichst schnell ablaufen, was mit dem Durchsatz gemessen wird. Ausserdem darf kein zentraler Server benutzt werden, die Lösung soll dezentral sein. Als zusätzliche Herausforderung werden einzelne Knoten aus dem Netzwerk aussteigen, doch die Übertragung soll trotzdem gewährleistet sein.
Wie diese verschiedenen Anforderungen gemeistert werden, wird in diesem Report erläutert.

Herausforderungen

In diesem Kapitel werden die Herausforderungen, mit welchen wir uns konfrontiert sahen, genauer beschrieben und unsere Lösungsansätze erläutert. Zuerst wird erklärt, wie mittels eines File-Splitters ein grosses File in kleinere Stücke aufgeteilt wird, um zu ermöglichen, dass die Einzelteile unter Umständen über verschiedene Wege verschickt werden. Am Schluss steht ein Joiner, der die verschiedenen Stücke wieder zusammenfügt, so dass die File Integrität gewährleistet ist. Im zweiten Teil wird erklärt, wie mit Hilfe von Iperf die Bandbreiten gemessen werden, so dass es uns möglich ist, über den schnellsten Weg zu routen. Um den schnellsten Weg auch tatsächlich zu finden, wird im dritten Teil beschrieben wie der Routing Algorithmus genau aussieht, den wir verwenden. Über das Bootstrapping mussten wir uns nicht mehr gross kümmern, da es uns zur Verfügung gestellt wurde. Das Problem von unerwarteten Knotenausfälle wird im letzten Teil behandelt.

File Splitting

Eine Klasse um die Files zu splitten hatten wir am Anfang erhalten. Es galt nun, diese Klasse zu bearbeiten und den Splitter zum Laufen zu bringen. Sobald ein File erkannt wird, startet der Splitter in einem separaten Thread. Er splittet die Files nun und speichert sie auf dem Workspace ab. Die gesplitteten Stücke werden fortlaufend nummeriert. Das erste Teilfile kriegt den Namen splitfile.1. Ein zweiter Thread wartet, bis er dieses erste File erkennt und startet dann die Übertragung der einzelnen Teile. Diese Übertragung sollte möglichst schnell ablaufen, das heisst es wird der schnellste Weg gesucht.
Um die verschiedenen Pakete wieder zusammenzusetzen, haben wir eine Klasse Joiner geschrieben. Seine Aufgabe ist, wie der Name schon sagt, die Pakete abzuwarten und in der richtigen Reihenfolge zusammenzufügen. Dies funktioniert folgendermassen: Auch diese Klasse läuft in einem Thread, der wartet ob ein Paket ankommt. Diese Pakete werden von Splitter nummeriert in der Reihenfolge wie sie aufeinander passen. Der Joiner speichert diese gesplitteten Files in der Reihenfolge wie sie zusammengehören. Im Falle, dass ein File verzögert ankommt, wartet der Thread so lange, bis er auch dieses File erhalten hat. Erst dann fährt er fort und fügt die Files zu einem Ganzen zusammen.

Bandbreiten-Messung

Die Bandbreiten-Messung wurde mit Hilfe von Iperf ausgeführt. Jeder Knoten weiss seine eigene IP-Adresse. Er schickt nun an allen Knoten aus seinem Leafset eine Nachricht, die seine IP enthält. Jeder Knoten der diese Nachricht erhält, macht nun eine Bandbreiten-Messung für 2 Sekunden, wo er feststellt, wie hoch der Throughput zum Sender-Knoten ist. Nachdem diese Messung ausgeführt wurde, schickt der Knoten der die Messung gemacht hat eine Nachricht an den TrackerNode. Den haben wir so benannt, weil er alle IP Adressen des Systems kennt und auch alle Resultate der Bandbreiten-Messung

speichert. Wenn nun ein Knoten ein Paket an einen anderen Knoten schicken möchte, holt er beim TrackerNode den Vektor, wo alle Resultate drin gespeichert sind. Mit Hilfe des Routing Algorithmus wird der schnellste Weg dann anhand des Vektors berechnet. Dieser TrackerNode ist nun eine zentrale Instanz der über alle Knoten im Netz Bescheid weiss. In dem unglücklichen Falle, dass nun genau dieser Knoten aus dem Netz aussteigen würde, gibt es einen Mechanismus von Pastry, den wir uns zu Nutzen machen. Für den TrackerNode benutzen wir einen fixen Key, der dann nach der Anwendung der Hashfunktion auf einen bestimmten Punkt im Pastryadressraum zeigt. Ändert nun der für diesen Punkt verantwortliche Knoten, also der Tracker, fällt dieser Bereich automatisch in den Verantwortungsbereich eines anderen Knoten. Die Iperf-Ergebnisse der einzelnen Nodes werden nun aufgrund des fixen Keys automatisch an diesen Knoten gesendet und nach einer kurzen Zeit besitzt dieser Knoten die ganze Information über das Netzwerk.

Die Bandbreiten-Messung ist periodisch, das heisst es wird alle 20 Sekunden gemessen, so dass wir immer auf sehr aktuelle Werte zurückgreifen können. Zusätzlich merkt das Programm auch mittels einer Exception, wenn ein Knoten unerwarteterweise ausfällt.

Routing Algorithmus

Der Routing Algorithmus basiert auch einer modifizierten Version des Dijkstra/Prim Algorithmus. Dieser Algorithmus gibt im Allgemeinen den kürzesten Weg an. Da wir nun den schnellstmöglichen Weg suchen und benutzen möchten, haben wir die ursprüngliche Idee leicht abgewandelt. Die Informationen über die Geschwindigkeit der einzelnen Verbindungen sind uns bekannt und werden in dem TackerNode gespeichert. Es gilt nun, die Verbindung herauszusuchen, über die am schnellsten geroutet werden kann. Um dies zu bewerkstelligen werden die Informationen über die Geschwindigkeiten beim TrackerNode abgeholt. Der Sender füllt nun einen Vektor mit allen Knoten die am Netzwerk beteiligt sind. In einem zweiten Schritt muss der Sender herausfinden welche Verbindungen von ihm aus weggehen. Bei den Knoten, die von ihm aus erreicht werden können, wird einen Eintrag gemacht, mit welcher Geschwindigkeit und über welche Knoten sie erreicht werden. Da dies die ersten Knoten sind, die überhaupt vom Sender erreicht werden, wird bei denen nur der Sender eingetragen. Danach wird beim Sender der Wert des Flags markiert. Dieses Flag gibt an ob der Knoten besucht wurde und ob alle ausgehenden Verbindungen desselben bereits berücksichtigt wurden.
Von den Einträgen die man auf diese Weise erhalten hat, wird nun die schnellste Verbindung herausgesucht. Das gleiche Prozedere wird nun beim nächsten Knoten wieder angewendet, jedoch wird die maximale Geschwindigkeit der gesamten Verbindung dem maximal möglichen Wert des Flaschenhalses angepasst. Mit anderen Worten, wenn die erste Verbindung beispielsweise 100 Mbit/s transportieren kann und die zweite Verbindung nur noch 60 Mbit/s, dann wird die Geschwindigkeit der gesamten Verbindung zum zweiten Knoten auf 60 Mbit/s gesetzt. Gleichzeitig wird überprüft ob der zweite Knoten schon einmal erreicht wurde und mit welcher Geschwindigkeit. In dem Falle, dass der Knoten auf einem anderen Weg schneller erreicht werden könnte, wird der alte Wert mit dem neuen, maximal möglichen, Wert überschrieben. Dies wird für das gesamte Netz durchgeführt, so dass man am Schluss für jeden Knoten einen Eintrag hat, der einem Informationen darüber liefert, wie schnell der Knoten vom Sender aus erreicht werden kann und über welche anderen Knoten geroutet werden muss.
Der eingetragene Weg vom Sender zum Empfänger, welcher auch der schnellstmögliche ist, wird gespeichert. Da man ja immer noch freie Kapazitäten im Netzwerk hat, die man

ausnutzen möchte, wird die gespeicherte Geschwindigkeit abgezogen, was einem nun die „Rest-Geschwindigkeit“ des Netzwerkes gibt. Natürlich wird die Geschwindigkeit nur bei den Verbindungen abgezogen über welche auch geroutet wird. Mit der Rest-Geschwindigkeit wird gleich verfahren wie am Anfang , also wieder der Routingalgorithmus ausgeführt, und zwar so lange, bis die mögliche Geschwindigkeit zum Empfänger-Knoten null beträgt. Dies stellt auch gleich die Abbruchbedingung dar. Alle schnellsten Wege werden in einem Vektor gespeichert und können nun für den Datentransfer benutzt werden.

Bootstrapping

Das Bootstrapping musste von uns nicht selber programmiert werden, da es im FreePastry schon implementiert ist. Es funktioniert nun so, dass man die IP des Bootstrap-Knoten kennen muss. Diese IP des Bootstrap-Knoten wird nun beim Programm Start in der Konsole eingegeben. Das erste Mal wird die IP nicht gefunden und der Knoten startet einen neuen Ring. Die nachfolgenden Knoten die am Netzwerk teilnehmen möchten, können nun die IP finden und beitreten, da der erste Knoten zum Bootstrap-Knoten für die anderen wird.

Unerwartete Knotenausfälle

In dem unverhofften Falle, dass ein Knoten ausfallen würde, haben wie die folgende Strategie gewählt: sobald die Socket-Verbindung zwischen den Knoten nicht mehr besteht, wird eine Exception geworfen. Diese Exception ruft eine Methode auf, die an den Sender den Namen des Files zurückschickt, welches nicht angekommen ist. Dies wird mittels myMessage von Pastry implementiert. Es könnte auch sein, dass das File zwischen dem Sender und Empfänger-Konten steckenbleibt. Dies muss erkannt werden und an den Sender-Knoten zurückgeschickt werden. Im Routing Vektor steht der Name des Files als zweitletzter Eintrag und der Sender-Knoten als letzter Eintrag. So kann der Knoten ganz einfach den Namen des Files und den Sender-Knoten herausfinden und die Informationen schicken.
Diese Verbindung muss gelöscht und die Route neu berechnet werden. Damit dieses Teilstück dennoch möglichst schnell beim Ziel ankommt, wird es in der Queue zuvorderst platziert.
Wie wir während des Testens herausgefunden haben, funktioniert diese Methode leider nur in der Theorie. Im Moment ist es so, dass das Programm zwar weiterhin die Bandbreiten-Messungen durchführt, jedoch sonst abstürzt.

Schwierigkeiten im Verlaufe der Arbeit

In diesem Teil des Berichts werden wir auf die Schwierigkeiten eingehen, mit denen wir uns im Verlaufe der Arbeit konfrontiert sahen. Was wir als sehr nervenaufreibend empfunden hatten, war die ganze Installation zum Laufen zu bringen. Da wir alle unterschiedliche Versionen von Java und Eclipse besassen, mussten wir uns am Anfang erst einmal einigen welche Versionen wir benutzen wollten. Alle hatten die neuste Version von Eclipse installiert, jedoch konnten wir nicht die neuste Version von Java verwenden, da diese Probleme mit dem Pastry verursachte. Erst als das alles geklärt war, konnte es losgehen mit dem Programmieren. Ziemlich schnell haben wir dann den File Splitter zum Laufen gebracht, jedoch nahmen das Verstehen von FreePastry und das Umstellen auf Linux sehr viel Zeit in Anspruch. Vor allem das Herausfinden der lokalen IP Adresse war problematisch, da die Methode um die IP Adresse zu erhalten, nicht auf Linux funktionierte.

Das Ausdenken eines effektiven Routing Algorithmus war sicher eine der Hauptaufgaben der Challenge Task. Die Idee die wir dann auch weiterverfolgten war, basierend auf Dijkstra/Prim, einen Algorithmus zu finden, der uns mit Hilfe der Bandbreiten-Messung den schnellsten Weg angibt. Das Problem war, dass wir Verbindungen die zurückgehen nicht einberechnet haben. Dies führte zu Loops, auf die wir erst beim Testen aufmerksam wurden und dann korrigiert haben. Dies haben wir bewerkstelligt indem wir den Weg der zu einem Knoten führte nicht mehr in die weitere Berechnung seiner ausgehenden Verbindungen einbezogen haben.
Ein weiteres Problem auf das wir stiessen war, dass wir die Hardware-Topologie nicht in Betracht gezogen haben. Wenn nun irgendwo ein shared link auftritt und die Geschwindigkeiten einzeln gemessen werden, dann stimmt die Messung nicht. Dies ist jedoch meist nur der Fall in einem kleinen LAN, wo die verschiedenen Knoten über Hubs miteinander verbunden sind. Sobald die verschiedenen Knoten nicht mehr im selben Netzwerksegment sind, wird das Problem nicht mehr auftreten.

Wie im Kapitel „unerwartete Knotenausfälle“ schon erwähnt, haben wir grosse Problem wenn ein Knoten plötzlich das Netzwerk verlässt. Dieses Problem konnten wir leider nicht beheben, denn je nach dem in welchem Stadium des Programms der Knoten ausfällt, werden andere Exceptions geworfen. Theoretisch hätten wir einen Lösungsansatz, den wir oben beschrieben haben, leider funktioniert der jedoch nicht in der Praxis.

Zusammenfassung

Unsere Strategie lässt sich kurz so beschreiben: Wir haben einen Knoten der alles weiss. Sobald ein Knoten ins Netzwerk kommt, misst er seine Bandbreite und meldet diese dem Superknoten, der alles in einen Vektor speichert. Wenn nun ein Knoten ein File schicken will, fragt er den Superknoten an, der ihm den Vektor schickt. So kann der Sender den schnellsten Weg herausfinden, welcher auf Grund eines modifizierten Dijkstra/Prim-Algorithmus basiert. Das File wird nun in Einzelteile zerlegt und diese werden über das Netzwerk geschickt. Mittels einer Klasse, die wir Joiner nannten, werden die einzelnen Teile wieder zusammengesetzt. Um dies zu bewerkstelligen, schaut der Joiner die Endungen der Files an, welche nummeriert sind, und setzt sie so zusammen.

Unser Ansatz basiert auf der Annahme eines kleinen Netzwerkes, so wie wir es in der Aufgabenstellung hatten. Ansonsten wäre der Superknoten-Ansatz wahrscheinlich nicht effizient durchführbar.

P2P Challenge Task

Gruppe 2

P2P - Sommersemester 2007

Verfasser: Matthias Alder, Marc Hämmig, Martin Hochstrasser, Philipp Kräutli
Betreuer: Thomas Bocek

Inhaltsverzeichnis

1. Einleitung

In dieser Challenge Task muss eine P2P Applikation geschrieben werden, die in der Lage ist eine grosse (~2 GB) Datei über den schnellsten Weg vom Sender zum Empfänger zu übertragen.
Es gibt 10 Knoten, die im Netzwerk miteinander verbunden sind und die einzelnen Verbindungen werden mit unterschiedlichen Bandbreiten künstlich angelegt. Nur der Quell- und Zielort sind frei von Uplink-Bandbreiten-Einschränkungen. Es ist deshalb notwendig, die grosse Datei in viele, kleinere Dateien aufzuspalten und parallel über verschiedene Hosts zu übertragen.
Für die Bewältigung dieses Problems gibt es verschiedene Lösungsansätze, die zu unterschiedlichen Resultaten führen werden. Im Kapitel 4 wird eine mögliche Lösung von uns aufgestellt, welche die Anforderungen so gut wie möglich erfüllen.
Des Weiteren sind im Dokument die Meilensteine beschrieben und eine Aufgabenverteilung mit einem Zeitplan aufgeführt.

2. Meilensteine

Es wurden für diese Aufgabe drei Meilensteine bestimmt.

- Meilenstein 1: Ein detaillierter Arbeitsplan mit der Aufgabenverteilung und ein Inhaltsverzeichnis mit einem Treffen mit dem Betreuer.
- Meilenstein 2: Codeentwurf und Reportentwurf
- Meilenstein 3: Produktiver Code und finaler Report (5-10 Seiten mit Design, Codestruktur und Installationsanforderungen)

3. Aufgabenverteilung

Software Components		
Component Name	Description	
File Handler	Splits/Reassembles Files into Chunks / File Hash Check	Philipp
Transmission Handler	Handles Hop-to-Hop Transmissions of Chunks	Mattias
Message System	Allows sending of Status information / Handles incoming Messages	Martin
Pastry Wrapper	Wraps the Pastry API / Handles Bootstrapping	Martin
Transmission Planer	Scheduling of Chunk Transfer, Chunk Overview, Chunk Routing	Mattias
Measurement System	Bandwidth Measurement Tool (Iperf Wrapper)	Marc
Status System	Status-Data persistency, recovery, output (probably WebInterface)	Philipp
Testing		All
Paper		
Coordination		Marc
Writing		All
Administrative		
SVN Setup		Mattias
Checking Remote Access Possibilities		Mattias

4. Routing Algorithmus

Die wichtigste Architektur-Eigenschaft unseres Lösungsansatzes ist der Routing Algorithmus. Er beinhaltet zwei Grundfunktionen:

- Forward-Flooding
- Load-Balancing

4.1 Forward-Flooding

Der Routing-Algorithmus basiert auf einem Flut-Verfahren, d.h. die Datei-Teile (Chunks) werden an alle Knoten des Netzwerks gesendet, und somit über das gesamte Netzwerk geflutet.
Der ursprüngliche Sender der Datei teilt diese zuerst in Chunks auf und sendet diese dann an alle Knoten, die er aus seinem Leafset kennt. Ein Knoten, der einen Chunk erhält, wählt dann wiederum einen Knoten aus seinem Leafset aus, an den er den Chunk weiterschicken wird. Der Weg welcher ein Chunk zurückgelegt hat, wird dabei mitgeführt. Dies verhindert ein Zurücksenden des Chunks an einen Knoten an dem er schon einmal gesendet wurde, indem die mitgeführten Weg-Knoten nicht als Ziel für die Weiterleitung berücksichtigt werden. Ein Chunk wird also im Peer-to-Peer-Netzwerk nur „vorwärts“ gesendet.

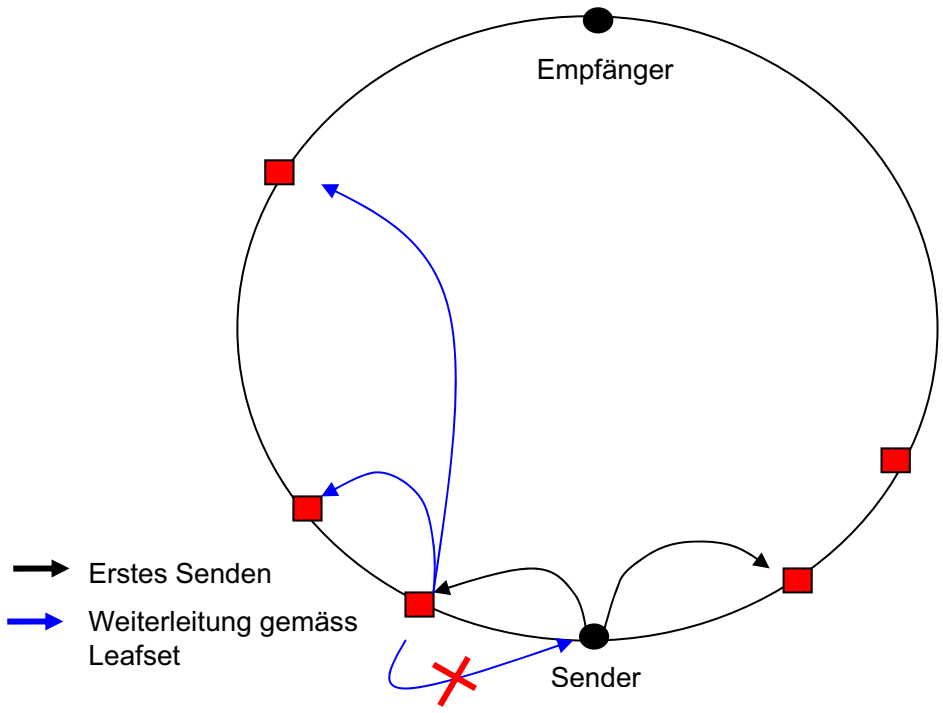


Abbildung 1: Forwarding im Pastry Ring

4.2 Load-Balanicng

Eine zweite wichtige Funktion im Routing der Chunks besteht in einem Load-Balancing. Jeder Knoten im Netzwerk sendet die Größe seines Puffers für ausgehende (noch zu sendende) Chunks an alle Knoten im seinem Leafset. Dies ermöglicht jedem Knoten eine Übersicht der Auslastung der Knoten in seinem Leafset zu bilden. Aufgrund dieser Angaben erfolgt eine zufällige Auswahl eines möglichen Zielknotens für ein Chunk, welche umgekehrt proportional zur Auslastung der Zielknoten gewichtet ist. Das heißt, je geringer eine Auslastung eines möglichen Zielknotens im Vergleich der Auslastung der anderen möglichen Zielknoten ist, umso höher ist die Wahrscheinlichkeit, das dieser Knoten als Ziel für die Weiterleitung des Chunks ausgewählt wird.

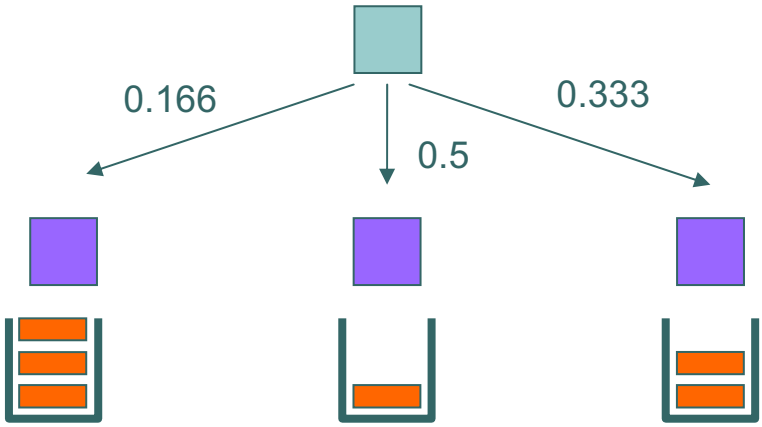


Abbildung 2: Load Balancing

5. Software - Architektur

Unser Lösungsansatz beinhaltet die folgenden Teile:

- File Handler
- Transmission Handler
- Message System
- Pastry Wrapper
- Transmission Planer
- Measurement System
- Status System

Die Funktionsweise dieser Komponenten werden im folgenden Kapitel genauer erklärt.
In Abbildung 3 der werden die Komponenten in einem Schichten-Modell als übersicht dargestellt:

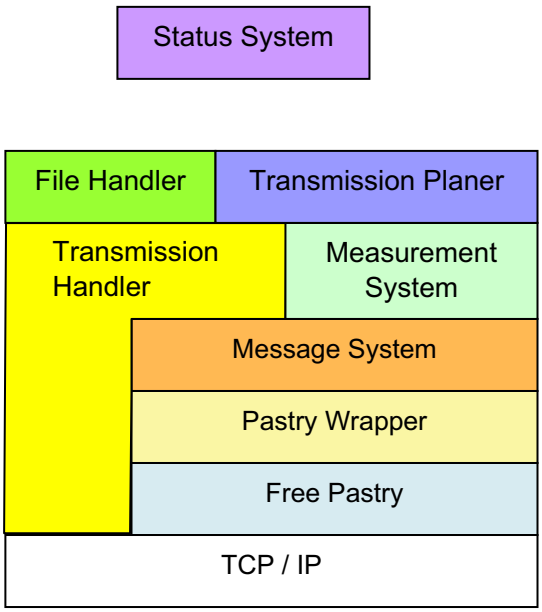


Abbildung 3: Schichtenmodell

5.1 File Handler

Der Filehandler wurde bewusst einfach gehalten, um grösstmögliche Verlässlichkeit und Performance zu ermöglichen. Dank der fixen Chunkgrösse kann der Filehandler anhand der Chunk-Nummer die Position des Chunks in der zu übertragenden Datei gemäss folgender Formel berechnen:

chunkNumber: fortlaufende Zahl, beginnend bei 0. chunkSize: Grösse eines Chunks in Bytes, **offset = chunkNumber * chunkSize**

Ein Chunk wird dann per RandomAccessFile am berechneten Offset ausgelesen bzw. eingefügt.

5.2 Message System

Das Message System bietet Methoden für die Übertragung von Nachrichten an. In der Abbildung 4 wird das Schichtenmodell dargestellt, wo sich das das Message System oberhalb des Pastry Wrappers befindet

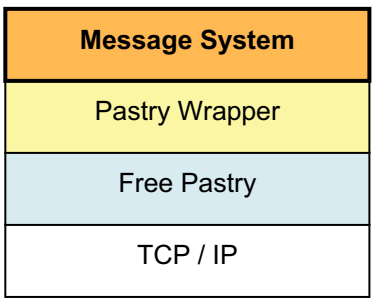


Abbildung 4: Schichtenmodell für MessageSystem

Das Message System bedient sich der Methoden für die Pastry Message-Übermittlung des Pastry Wrappers und bietet seinerseits Methoden für die Senden und Empfangen von Messages an die

anderen Applikationsteile an. Das Message System definiert dabei eine eigene Message-Klasse welche beliebig erweiterbar ist.
Neben dem einfachen Senden einer Nachricht über Pastry bietet das Message System auch Methoden für das direkte Senden einer Nachricht an einen Pastry-Knoten an. Die Ausführung dieser Methode geschieht nur dann erfolgreich, wenn der Zielknoten im LeafSet des lokalen Pastry-Knotens liegt, sonst ist kein direktes Senden möglich und es wird eine Exception geworfen.
Es ist auch möglich eine spezielle SafeMessage zu senden. Beim Empfang einer SafeMessage wird dem Sender Message System eine Benachrichtigung zurückgesendet. Das ausgangs Message System kann dann einen Interessenten innerhalb der Applikation über die erfolgreiche Auslieferung der Message benachrichtigen.
Das Message System verarbeitet auch alle Eingehenden Nachrichten. Eingehende Nachrichten werden dabei zuerst in eine Inbox-Queue gelegt und von einem separaten Thread abgearbeitet. Applikationsteile, welche sich für einen speziellen Typ von eingehenden Messages interessieren, können sich durch die Implementierung eines MessageEnpoint-Interfaces beim Message System als Endpunkt registrieren.

5.3 Pastry Wrapper

Der Pastry Wrapper ist die unterste Schicht unserer Applikation. Die Aufgabe des Wrappers ist es, die Pastry-spezifischen Aufgaben zu übernehmen und gegenüber den anderen Teilen der Applikation zu verbergen. Der Pastry-Wrapper wiederum bietet dann eine Schnittstelle für die anderen Applikationsteile an, welche vor allem durch das Message System verwendet wird.

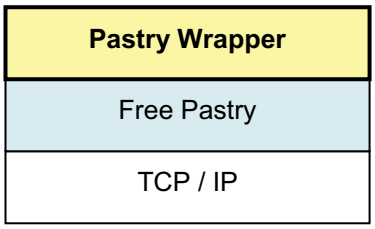


Abbildung 5: Schichtenmodell für PastryWrapper

Bei der Instanzierung des Wrappers wird versucht eine Instanz einen Pastry Node zu erzeugen und dem Netzwerk beizutreten, wird kein geeigneter Bootstrap Node gefunden, wird ein neuer Pastry Ring erzeugt.
Die Schnittstelle für die anderen Applikationsteile beinhaltet Methoden für das Senden einer Pastry Message und für die Abfrage eines Sets von Nachbar Node Handles welche im Leafset des Knotens bekannt sind.

5.4 Transmission Planer

Der Transmission Planer ist für das Routing der Chunks zuständig. Er benutzt dazu Daten die vom Transmission Handler und vom Measurement System zur Verfügung gestellt werden. Er ist auch für das Load-Balancing zuständig.

5.5 Measurement System

Die Komponente ist misst die Auslastung der Peers im Leafset. Das Measurement System liest die Größe der Liste der Ausgehenden Chunks aus dem Transmission Handler aus, und sendet diese Information periodisch an alle Knoten aus dem Leafset mithilfe des Message Systems.

Gleichzeitig horcht es auf eingehende Messages von anderen Peers welche ihre Auslastungsinformation senden. Diese gesammelten Daten werden dem Transmission Planer für die Routing Entscheidung zur Verfügung gestellt.

5.6 Status System

Das Status System ist für die Ausgabe der Statusinformation des Knotens zuständig. Es erlaubt verschiedene Stufen für Statusmeldungen für die Ausgabe:

6. Installation und Betrieb

Um die Applikation zu starten, muss die ausführbare Datei p2p.jar mit folgenden Parametern ausgeführt werden:

Start eines Receivers / Relay:
`java –jar p2p.jar <bootsratp-ip> <bootsrap-port> <local port> receive <output folder> (<verbosity-level>)`

- `bootstrap-ip`: die IP-Adresse des Knotens über den das bootsrapping probiert werden soll, oder „local“ um ein bootsrapping mit der localhost Adresse zu versuchen.
- `Bootstrap-port`: Der Port des Bootstrapknotens
- `local port`: Der lokale Port auf welchem Pastry horchen soll. (Für die Socketübertragung wird automatisch ein Port der um 1 höher liegt verwendet)
- „`receive`“ gibt an das der Knoten kein Sender sein soll.
- `output folder`: Der Pfad an dem eingehende Dateien gespeichert werden sollen
- `verbosity-level`: Gibt an wie detailliert der output sein soll. Mögliche Werte: „fatal“, „error“, „warning“, „notice“, „debug“, „trace“ (Standarteinstellung ist „notice“)

Um einen Sender zu starten wird folgendes Kommando verwendet:
`java –jar p2p.jar <bootsratp-ip> <bootsrap-port> <local port> send <datei> (<verbosity-level>)`

- `bootstrap-ip`: Die IP des Knotens mit dem das Bootsrapping erfolgen soll. Dies ist gleichzeitig die Adresse des Knotens an den die Datei gesendet wird.
- `datei`: Gibt die Datei an die gesendet werden soll (vollständiger oder relativer Pfad)

Die restlichen Parameter sind gleich wie beim Receiver/Relay.

Wenn der angegebene Boostrap-Knoten nicht gefunden wird oder kein Bootsrapping an dieser Adresse möglich ist, wird automatisch ein neuer Ring generiert.

Beispiele:

```
java –jar p2p.jar 192.168.1.15 20000 20000 receive /home/user4
```

Dies startet einen Sender, der ein Bootsrap mit dem Knoten auf 192.168.1.15:20000 macht, und eingehende Dateiübertragungen in den Ordner „`/home/user4`“ speichert

```
java –jar p2p.jar 192.168.1.15 20000 20000 send /home/user2/big_file.big
debug
```

Dies startet einen Sender, der ein Bootsrap mit dem Knoten auf 192.168.1.15:20000 macht, und dann an diesen Knoten die datei „/home/user2/big_file.big“ überträgt. Debug-Nachrichten werden ausgegeben.

7. Bewertung

In diesem Kapitel werden die Anforderungspunkte aus der Aufgabenstellung mit unserer Lösung verglichen. Folgendes Resultat ist dabei herausgekommen:

- Das Splitten einer Datei wurde mit Chunks realisiert, deren Grösse über eine statische Variable festgelegt werden kann. Chunksize ist defaultmässig auf 512Kb eingestellt.
- Integrität der Datei wird gewährleistet. Die versendete Datei kommt beim Empfänger korrekt und integer an.
- Die Applikation ist auf den zur Verfügung gestellten Maschinen (Debian Linux, Java 1.5) lauffähig.
- Anhand eines Tests konnte festgestellt werden, dass unsere Lösung eine höhere Durchsatzrate hatte, als der Dateitransfer über die direkte Verbindung.
- Im Falle eines Knotenausfalls verfolgte die Applikation ohne Absturz ihr Ziel. Die verloren gegangenen Pakete werden ohne Probleme nochmals angefordert und erneut an den Empfänger verschickt
- Paralleler Datentransfer ist möglich. Alle vorhandenen Knoten sind am Transfer beteiligt und werden anhand eines Load-Balancing Algorithmus optimal ausgelastet.
- Unsere Lösung basiert auf einem dezentralen System. Es gibt keine Komponente, die den Vorgang von einer zentralen Instanz aus steuert.

Alle Anforderungen an das System werden von unserer P2P-Applikation erfüllt.
Auch wenn sie eine simple Lösung darstellt, meistert sie die Aufgabe und der Datentransfer über verschiedene Knoten ist jedenfalls schneller als der Datentransfer mit der direkten Verbindung.

UNIVERSITÄT ZÜRICH

Challenge Task: FastP2P

P2P Systems and Applications

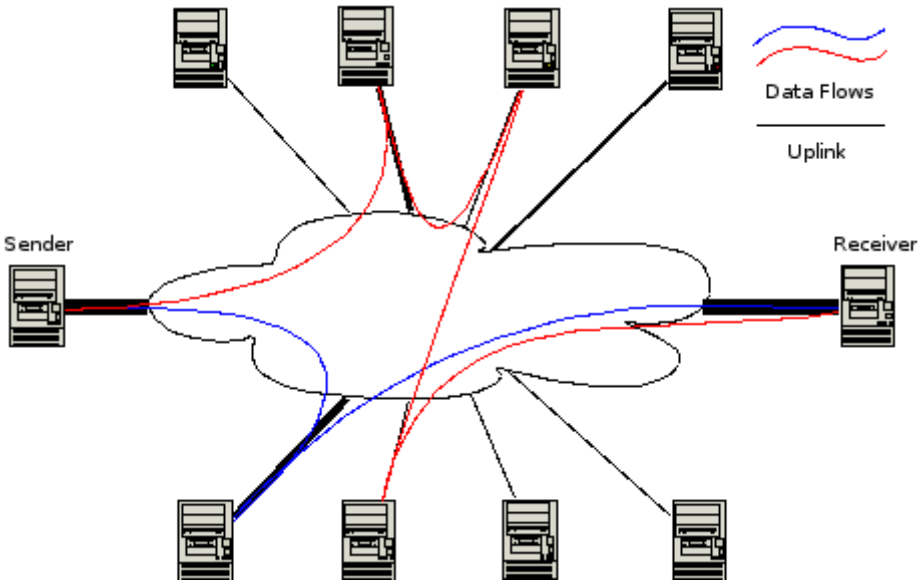
Thierry Kramis
Marcel Schönbächler
Alex Bucher
Stefan Bösch

Group 3

Inhalt

1. Task Description

This semester's task is the design and implementation of a Peer-to-Peer application that is able to transfer a given large file the fastest way possible from one host to another. Consider the following scenario:



Challenge Scenario

There will be ten nodes interconnected via a network. Only the source and the destination will be free of artificial uplink bandwidth constraints, but the direct connection between them will be very slow. The other eight nodes will cover various parts of the network in their respective leafsets (up to a maximum of 4 nodes per leafset), but will have uplink bandwidth constraints.

The task, as mentioned will be to deliver a large file the fastest way possible from source to destination via the other nodes. In order to solve this problem, the underlying topology will have to be considered and files have to be split up in parallel partial transfers. (Source: P2P Challenge Task Homepage)

2. Workplan (Milestones)

2.1. Milestones

In order to enable a well-regulated work progress, we divided the project into different working steps and milestones.

03.05.2007	**Milestone 1**	Detailed workplan including task distribution among group members, ToC of the report, Meeting with the supervisor
Week 1		Everyone gets familiar with the Code and Systems
Week 2		The different components are developed separately according to the task distribution in 3.
Week 3		The components are put together and tested.
24.05.2007	**Milestone 2**	Draft Code: - Design and implement a protocol for splitted file transfer - Design and implement a protocol that measures bandwidth
Week 4		Optimization of the file transfer
Week 5		Optimization of the routing algorithm
Week 6		Test of the whole system in the lab
14.06.2007	**Milestone 3**	Final Code: Find a scheme using relaying nodes to increase throughput, use the measured bandwidth to find suitable nodes. Design and implement the protocol to find relaying nodes. Put all parts together

3. Task Distribution

The task distribution amongst the team members looks as follows:

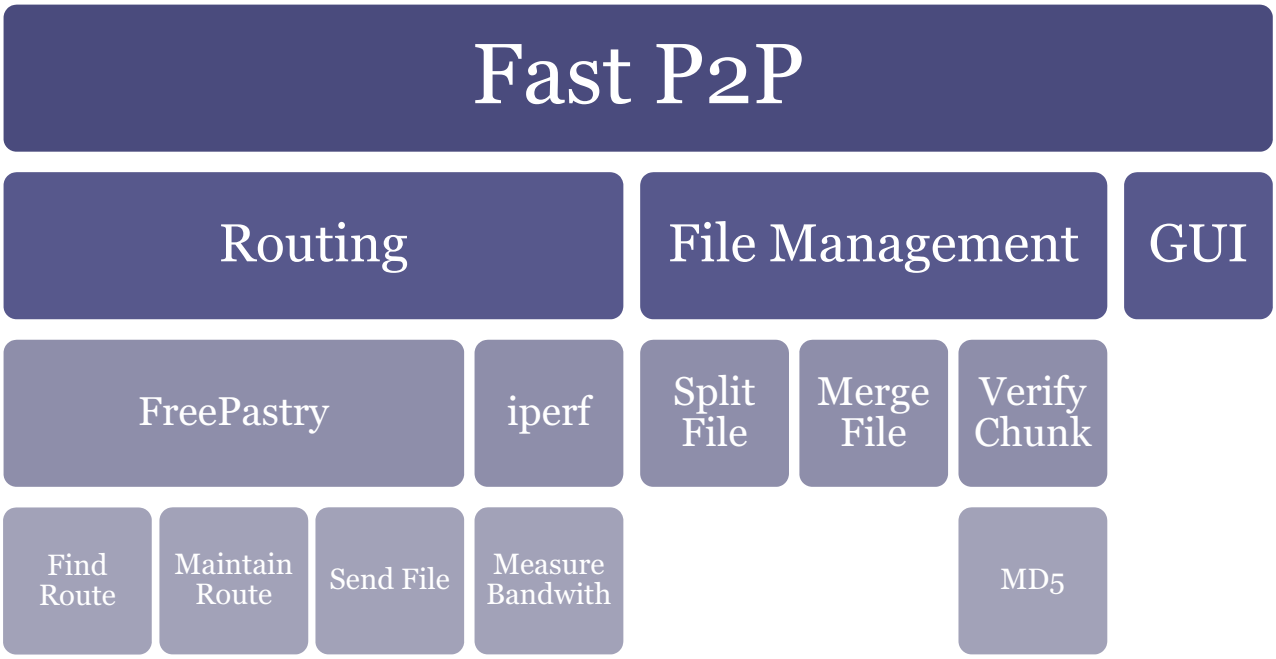
Task	Responsible team member
Get an overview of Pastry	Everyone
File splitting / merging / checking / presentation	Marcel
Routing algorithm / testing	Alex
File transfer / general code	Thierry
File transfer / documentation	Stefan

4. System Overview

4.1. System Requirements

- Java 1.5
- Operating Systems: Windows, OSX, Linux

4.2. System Architecture



FastP2P architecture

The Fast P2P system is divided into three sub modules, providing the needed functionality.

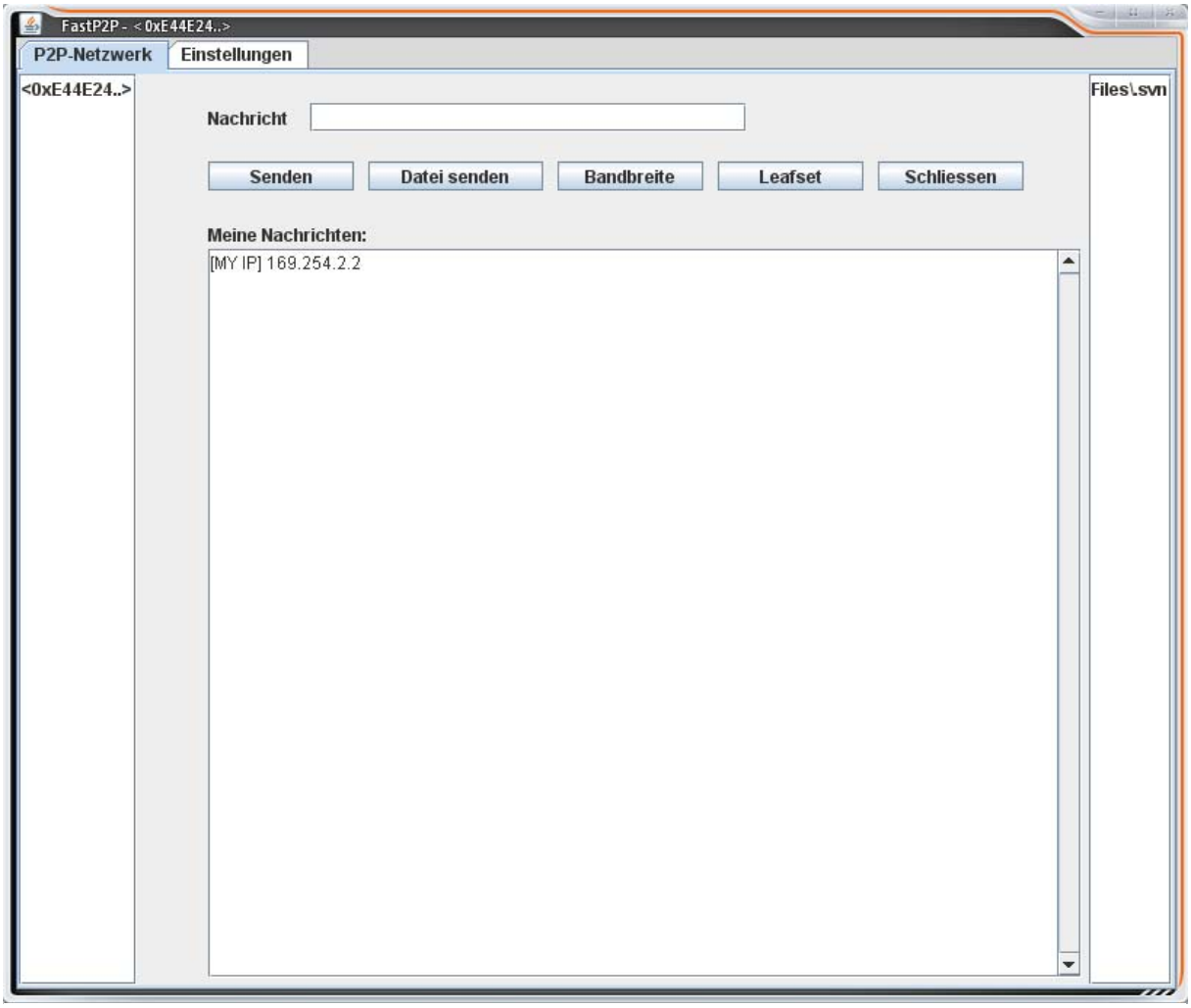
Routing
The routing part is based on FreePastry and provides all functions to find and maintain a route as well as sending a file to the selected node.

A new node gets 5 neighbours from the leafset generating its own pathtable and distributes its IP to the whole pathtable (IPREQUEST).
Each node that receives an IPREQUEST adds the new path with the IP to its own pathtable and calls updateBW(String endnodeID) to check the bandwidth to the new node.

To send a File over FastP2P the startSearch()-method is called. A path to the end node has to be fast and near to the sender. Therefore a search message is sent from node to node to discover the fastest path to the end node. GetFastestLink() checks whether the endpoint appears in the localookupset and if not, takes the fastest connection to the next node. If a loop is detected, the search message is sent back one hop and the second fastest connection will be used to discover a path. Once the end node is found, the searchmessage is sent back over the path discovered during the lookup phase and the file transmission can start.

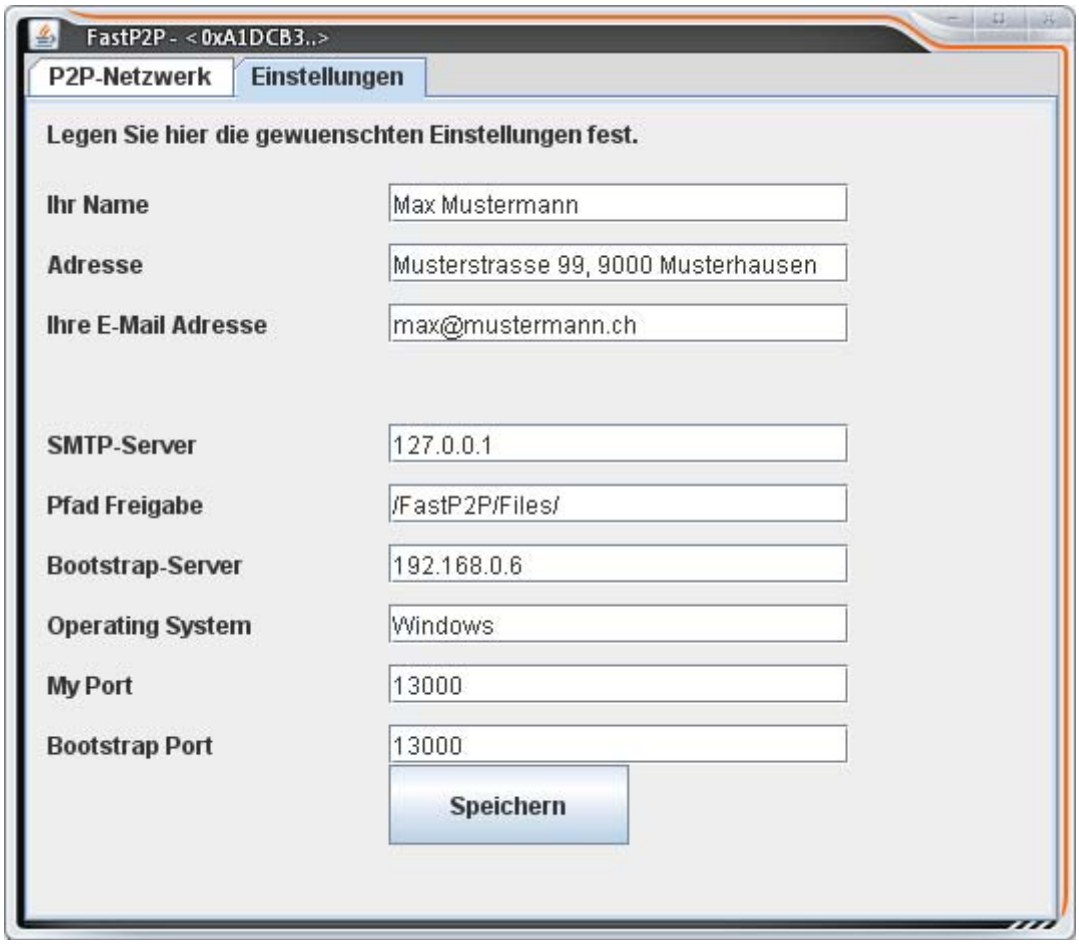
File Management
The file management is responsible for all the tasks regarding the preparation and verification of the transmitted file.
After pressing the “send file” button, the file gets split into several junks and is sent according the routing algorithm. If a node receives a chunk that’s not designated for it, no merging will be done and the chunk is forwarded to the next peer. At the endpoint the chunks are merged frequently after receiving a certain number of consecutive chunks. If a chunk gets lost on its way to the endpoint a NACK message initiates the retransmission of that particular chunk.

GUI
The GUI includes all elements used to control the program.



FastP2P – Main Tab

The leafset row shows the peers visible to the FastP2P client. By selecting a peer in the list, messages and files can be sent directly to the corresponding node.
Incoming messages from other peers as well as log entries are displayed in the messages area.
The file browser displays the content of the shared folder, set in the settings tab (see below). In order to send a file to another node, it can directly be selected in the file list. Incoming files are stored in the shared folder as well and will show up in the list.



FastP2P – Settings Tab

In the settings tab, entries like username, bootstrap node or operating system can be altered and saved in the properties-file.

5. Tools used in the project

- Eclipse 3.2
- Subversion 1.4.4
- Subclipse 1.2.2
- FreePastry 2
- Iperf 1.7.0

P2P challenge task 2007:

Documentation of group 4

Document version:
1.1

Date:
26.06.2007

Authors:
Jonas Zuberbühler
Andreas Siegrist
Remo Welti
Philipp Vontobel

1 Project Plan

1.1 Workplan

We decided to work in the following order:
1. Sending serialized files in form of a message over a pastry ring from A to B
2. Visualising routing tables, leaf sets and eventually routing trees (global net view of a node)
3. Measuring bandwidth with the goal of getting a global view over the ring
4. Developing an appropriate routing algorithm
5. Extending the routing algorithm in terms of robustness
6. Introducing splitting
7. Extending splitting in terms of robustness and integrity (MD5 check)

1.2 Task distribution

The programming parts particularly that of Jonas and Andreas will be done mostly in pair programming style, so a clear task distribution is not necessary and makeable.

- Jonas Zuberbühler: tutorial/reading, SVN host, programming specialist, algorithm discussion
- Andreas Siegrist: tutorial/reading, programmings specialist, algorithm discussion
- Remo Welti: tutorial/reading, testing iperf, FileSplitter/Joiner and Integrity check, algorithm discussion
- Philipp Vontobel: tutorial/reading, reporting, testing iperf, FileSplitter/Joiner and Integrity check, algorithm discussion

2 General Idea

All peers are managed in a Distributed Hash Table (DHT) and can be found using IP adresses. This is realised with the PAST implementation. Splitting and joining of file is made using the JAxe library.
We are using a Java-based MD5-implementation which is about 30% faster than the algorithm implemented in OSX from Apple.
MD5 reference: http://www.twmacinta.com/myjava/fast_md5.php

3 GUI

The GUI currently includes visualization of the leafset of every peer and the possible routes. Additionally it has a notification window to inform the user about the peers status. Buttons within the GUI make it possible to manually execute every action step by step. The purpose of these buttons will be explained in chapter 10.

4 Bandwidth measurement

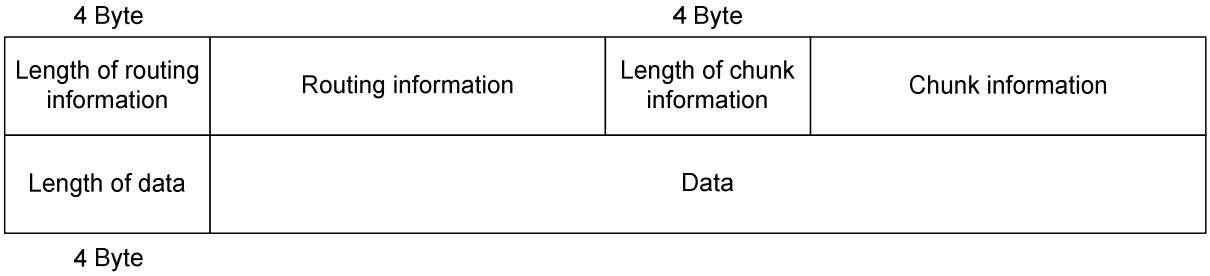
Not yet implemented because of elementary issues to be dealt with first. We couldn't find a java implementation so we will likely use iPerf as proposed.

5 Routing

Actual implementation: The sender sends the receiver a HelloMessage as the „hello“-button is pressed, which is realised by a broadcast to the whole leafset. At every forward, the HelloMessage is expanded by the current path from the sender. As soon as it is reached, the receiver sends back the whole path information to the sender, which in this way gets all possible paths to the receiver.
The paths have to be extended by performance/bandwidth information. This is not implemented yet due to the same reason as mentioned before.

6 Data Transfer

To transmit data, the application level socket interface is used. This allows to transmit great amounts of data, without blocking the overlay network. We use the sockets only to transmit chunks. All other messages and even the meta-file are sent via overlay network based on Pastry.



Picture 1: chunk protocol

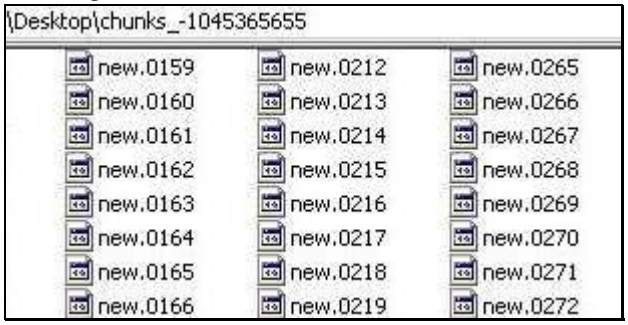
We defined the following protocol for our chunks:
A data package consists of the length of routing information data, the routing information, the length of chunk informations, the chunksinformations, the length of filedata and the filedata itself. The data is sent in form of bytestreams.
On every intermediate/forwarding node, the routing information is read out and the package then gets forwarded.

7 Robustness

Actually, we have problems with intermediate nodes crashing because of high memory usage. Until this issue is handled, we cannot test further our network robustness.

8 File splitting and integrity

First off all, a random SessionID is generated along with a new splitter. The splitter then makes a new directory for his output (the chunks) including the SessionID in the name.



Picture 2: chunk folder

Before the splitting can begin in a new thread, the sender process is registered as an observer. In this way he gets notified if certain numbers of chunks are ready to be transmitted.
The sender starts the transmission of the first chunks before splitting is finished. At every transmission, a hard coded timer is launched which initiates a re-send, if a chunks doesn't get acknowledged in a certain amount of time. Our target here would have been to make this timer dynamically dependent of the network performance, but we haven't realised that yet.
As soon as all chunks are made, a metafile is constructed including the number of chunks, the MD5-checksums of all chunks and that of the original file.
After the construction of the meta-file, the sender gets notified and will send it at least twice to the receiver, using the overlay network. The receiver computes the checksums of the meta-files until he has two matching numbers. The sender then receives a message that he can stop sending meta-files.
The joiner on the receiving side can decide if he wants to consider the checksums or not (so he could join/merge the files without the meta-information).

When chunks arrive at the receiver, they are written to the HDD and the joiner gets notified. Now two states are possible:

1. The receiver has no valid meta-file yet: The joiner notes the chunknumber and computes and saves the checksum.
2. The receiver has two matching meta-files: The joiner additionally compares the computed or stored checksums with those out of the meta-information.

If a comparision has a negative result, a retransmission request will be sent over the overlay network back to the sender. Otherwise the sender gets also informed of the succesfull arrival of the chunk.
As the receiver has all chunks with the rigth checksums, he starts the joining and reconstructs the original file. Another MD5-check of the reconstructed file is then possible if needed. The corresponding comparision value is available in the meta-file.

9 Performance test results

Setup: 8 nodes, 100MB file, limited bandwidth on some routes (no knowledge of detailed limitation)

Test 1: without node failure
130 sec.
MD5 integrity check passed

Test 2: including node failure
220 sec.
MD5 integrity check failed

At test 2, the internal md5 check mechanism was not started because the standard parameters where used. As a result, the resending-routine didn't start as the application was not able to check if the chunk integrity was right.

10 Installation requirements and usage

The following Jar-archives have to be placed in the “lib” directory at the same place as the p2p.jar:

- p2p.jar (Main application)
- lib
  - swing-worker-1.1.jar
  - gnujaxp.jar
  - itext-2.0.1.jar
  - jcommon-1.0.9.jar
  - jfreechart-1.0.5.jar
  - slf4j-api-1.1.0-RC1.jar
  - slf4j-jdk14.jar
  - slf4j-simple-1.1.0-RC1.jar
  - timeframework.jar
  - xpp3-1.1.3.4d_b2.jar
  - FreePastry-2.0.jar
  - xmlpull_1_1_3_4a.jar

The application has to be lauched with the following shell command in order to reserve enough memory:

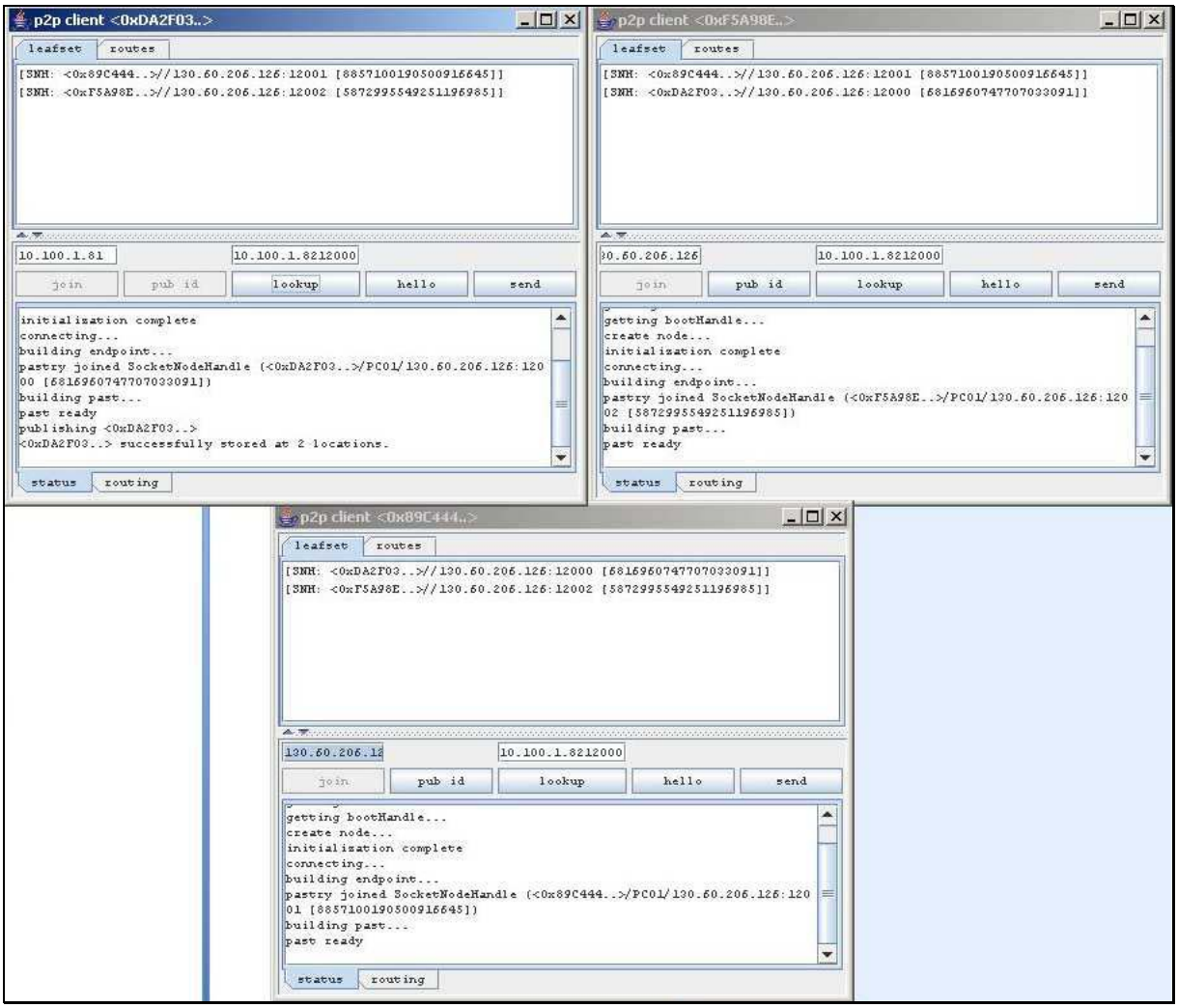
java -Xmx500M -jar p2p.jar

In order to change other parameters the console gives out an explanation. If the launch starts without extra parameters (like above) the standard ones will be loaded.

On the first node, the following buttons have to be pressed:

1. Join (creates a new ring)
2. Pub ID (publishes the own IP in the Past network)

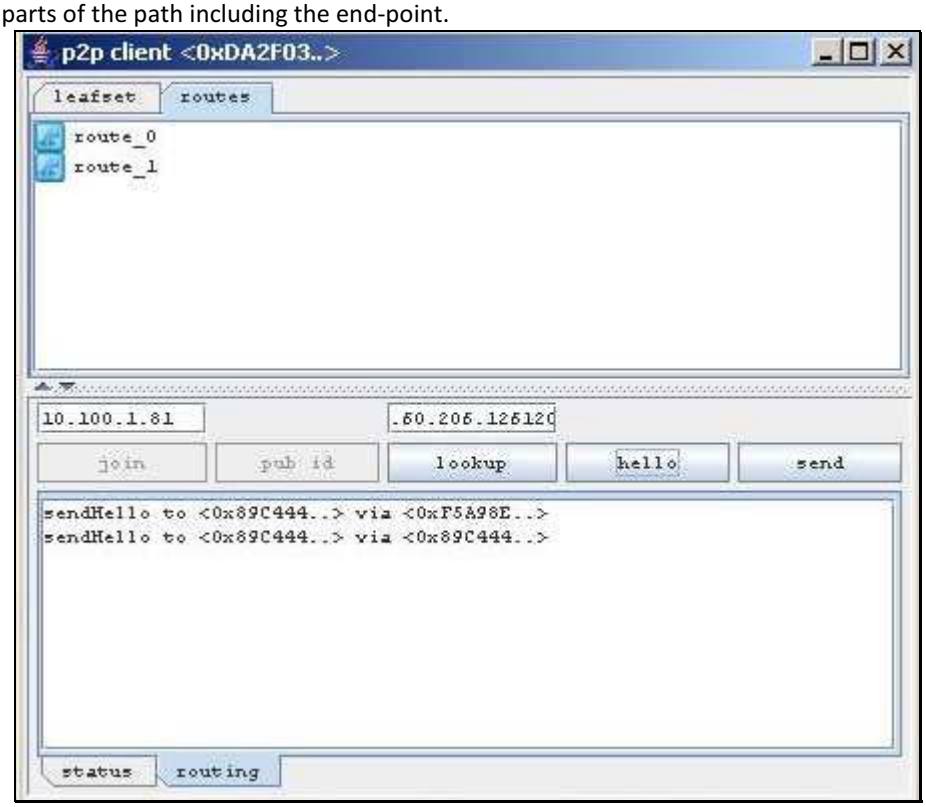
This first node is afterward the bootstrap for the other ones to join, so before pressing any button on them, the IP of the bootstrap has to be right in the textfield above. Respecting this issue, the other nodes can join the ring pressing the "join"-button. As you can see in the picture below, the leafset of the nodes gets actualized, as soon as there are more than one peer in the network.



Picture 3: building up the ring

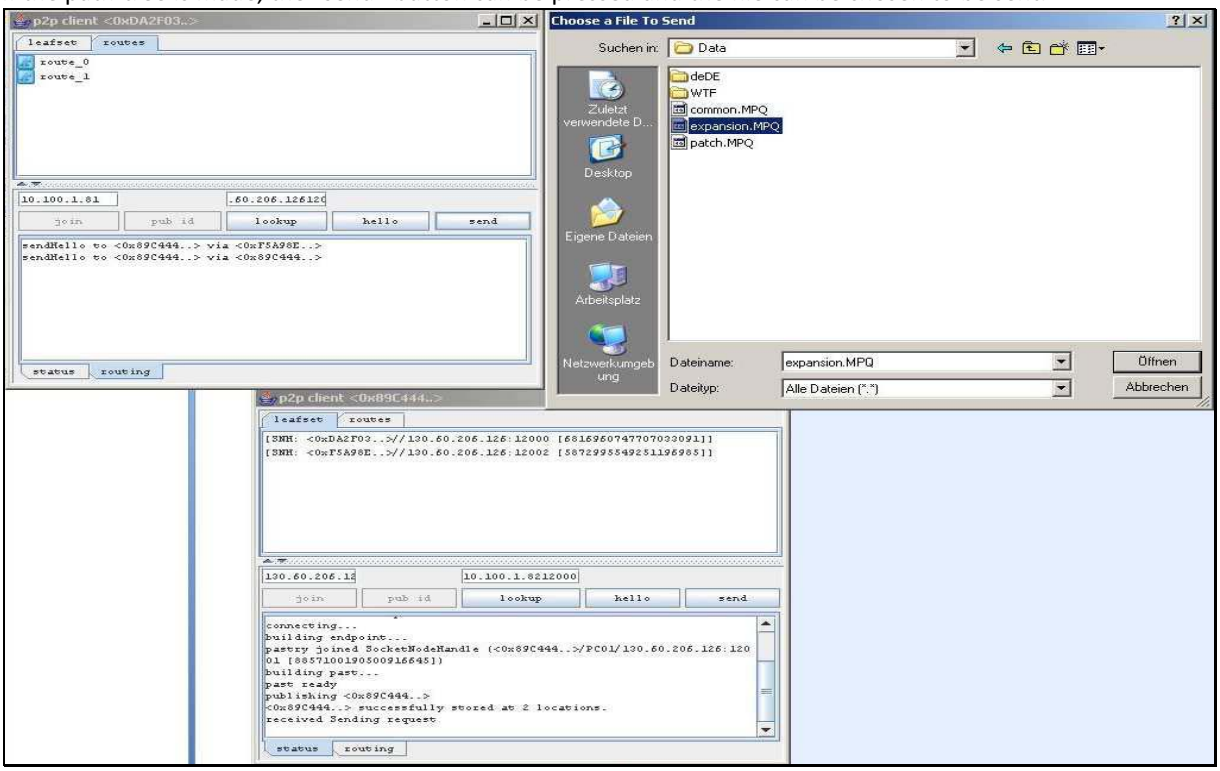
Sending a file:

On the sender, the receiver-IP has to be entered in the right textfield including the port number (without ":"). Afterwards the "hello"-button has to be pressed which constructs a list of all possible paths to the receiver. The following picture shows the evaluated routes, for an example with three nodes. A double click on a route visualizes all parts of the path including the end-point.



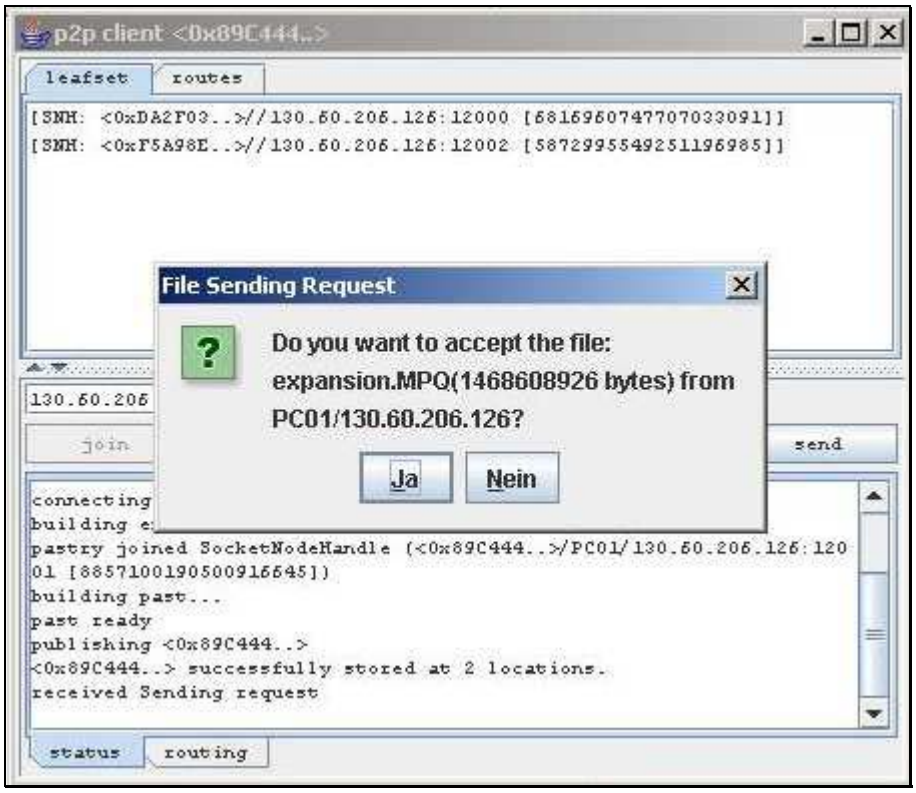
Picture 4: possible routes

If the path-tree is made, the "send"-button can be pressed and the file can be chosen to be sent.



Picture 5: filechooser

The receiver gets a message, where he can accept the file and choose a destination for it and notifies the sender. Splitting (and all other further things) on the sender starts as soon as he gets the overlay message from the receiver.



Picture 6: acceptance at the receiver

P2P Challenge Task für Peer-to-Peer Systems and Applications

Gruppe 5:

Tobias Bannwart
tbannwart@access.unizh.ch

Stefan Zehnder
stefanzehnder@gmx.net

Alessandro Vagliardo
vagliardo@tele2.ch

Dalibor Peric
dalibor@access.uzh.ch

Milomir Krstic
krsticm@access.uzh.ch

Inhaltsverzeichnis:

1. Einführung

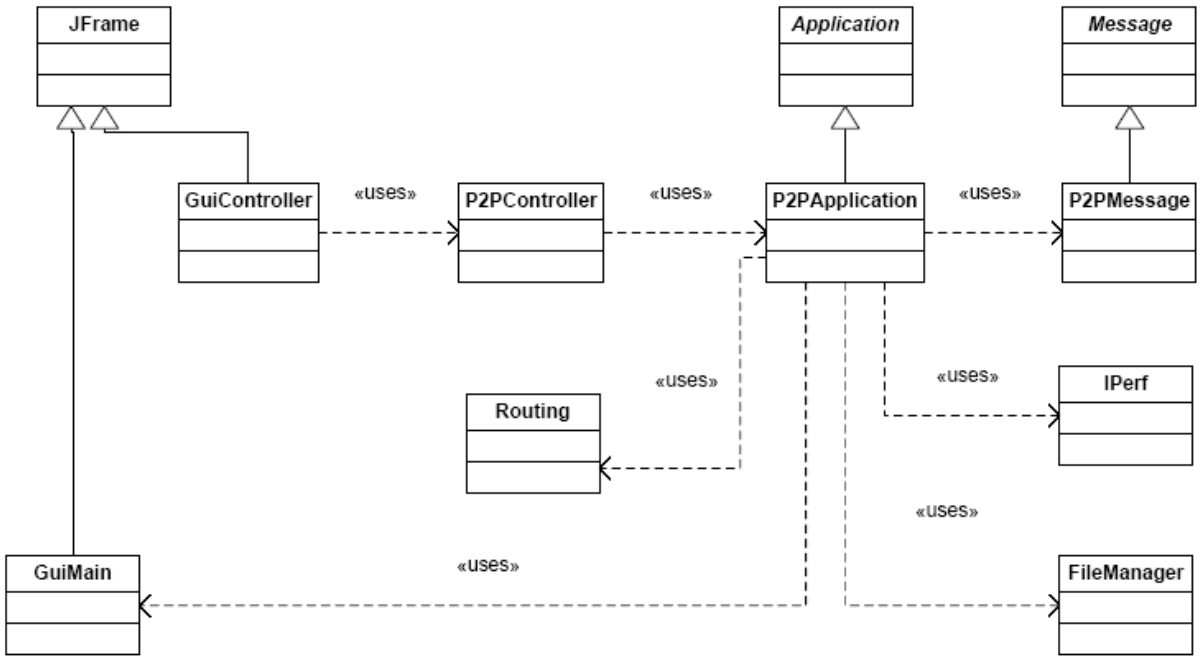
Am 19. April startete die P2P Challenge. Wir mussten eine P2P Applikation schreiben, die über eine gewisse Anzahl Knoten ein grosses File schicken kann, dabei sollte diese File möglichst schnell transferiert werden. Die Gruppe bestand aus dem Team Tobias Bannwart, Stefan Zehnder, Dalibor Peric, Milomir Krstic und Alessandro Vagliardo. Wir arbeiteten über einen CVS Server, damit jeder immer auf den neusten Stand war.

In diesem Endbericht werden wir zuerst etwas über den Aufbau unserer Applikation schreiben, danach etwas über den gewählten Routingalgorithmus und über die Probleme, die wir während dem Projekt hatten diskutieren.

2. Unsere P2P Applikation

2.1 Aufbau des Projekts

Über ein kleines Loginfenster, den GuiController, wird der P2PController aufgerufen, der für den Aufbau und das Einloggen zum Bootstrapknoten zuständig ist. Falls kein Knoten gefunden wird, wird ein neuer Ring geschaffen. Die Klasse MainGui ist die Schnittstelle des Users zur Applikation selber. Man kann hier die Nachrichten schicken oder ein File wählen, dass dann gesendet werden soll. Die beiden GUI Frames wurden mittels den JFormDesigner erstellt.
Um die Files oder Nachrichten zu schicken, wurde das Interface von Message verwendet und erweitert. Durch einen eigenen Filesplitter werden grosse Files auf gesplittet. Nachdem die einzelnen Chunks angekommen sind, werden diese dann auch wieder Zusammengesetzt. Die P2PApplication Klasse implementiert Applications und hat eine Instanz der Routingklasse. Diese ist für die Bandbreitenmessung und für die Wahl des schnellsten Weges verantwortlich.
Ein besseren Überblick wie die Klassen miteinander zusammenhängen, zeigt das folgende Klassendiagramm:



Die Routingklasse hat eine Instanz des Interfaces RoutingAlgo, so können wir verschiedene Algorithmen für das Routing testen. Wird beim Transfer der Chunks festgestellt, das welche fehlen wird eine Nachricht an den Startknoten gesendet und die fehlenden Chunks werden nochmals gesendet.

2.2 Charakteristiken

Keine globale Sicht des Netzwerks, sondern nur lokale Entscheidungen werden getroffen. Der Sender sendet immer eine Endnachricht an den Empfänger, wenn dieser diese nicht bestätigt, wird weiter Chunks gesendet. Falls beim Empfänger Chunks fehlen, wird eine Nachricht an den Sender geschickt mit den fehlenden Chunks, die dann nochmals geschickt werden.
Die Nachrichten, die die Chunks tragen, enthalten Informationen über den aktuellen zurückgelegten Weg, somit werden Loops verhindert. Wenn ein solches Chunk erhalten wird, wird es automatisch gelöscht.

2.3 Klassenbeschreibung

GuiController und GuiMain
Diese Klassen sind für den Start unserer Applikation verantwortlich. Mittels dem GuiController wird ein Knoten gestartet. Weitere Erklärung wie man einen Knoten startet, wird im Kapitel Manual beschrieben.
Das GuiMain ist die graphische Schnittstelle für den User. Es gibt die eingegeben Informationen an die P2PApplication. Zuerst wird eine P2PMessage generiert. Diese Nachricht kann entweder ein File selber sein oder eine Textnachricht.

P2PController
Dieser ist für die Erschaffung eines neuen Rings oder für das Hinzufügen eines Knotens in einem schon vorhandenem Netzwerks verantwortlich.

P2PApplication
Sie ist eine Implementierung des Interfaces Application von Freepastry. Diese Klasse besitzt eine Instanz von der Klasse Routing. Weiter werden hier die Messages geroutet, weitergeleitet oder an den jeweiligen Zielknoten ausgeliefert. Wenn eine Nachricht angekommen ist, wird das GuiMain aktualisiert und der User sieht das er etwas bekommen hat.

Routing
Durch einen Routingalgorithmus werden die Chunks über mehrere Wege an den Zielknoten gesendet. Weitere Erklärungen folgen in einem separaten Kapitel.

FileManager
Die FileManagerklasse ist für die Zerstückelung der Files zuständig. Die grossen Dateien werden in vielen Chunks verkleinert. Wie gross ein solcher Chunk sein wird, hängt vom Routingalgorithmus ab, der die grösse der jeweiligen Chunks bestimmt.

P2PMessage
Diese Klasse implementiert das Interface Message. So können wir unsere eigene Nachrichten kreieren mit den Informationen, die wir benötigen. So können wir das senden mittels Freepastry selber benutzen, da nur Nachrichten der Klasse Message verschickt werden können.

2.4 Robustness

Bei einem Knotenausfall werden diejenigen benachrichtigt, die diesen Knoten im Leafset enthalten und somit werden keine Nachrichten mehr an diesem gesendet.
Jedes mal wenn neue Knoten hinzukommen, werden diese auch involviert. Das gesendete File wird am Schluss auf die Richtigkeit geprüft uns somit auf Integrität getestet.

2.5 Statistiken

Typ	Anzahl
Klassen	33
Methoden	199
Package	7
Zeilencode	2488 Zeilen
Längste Methode	195 Zeilen
Methode mit meisten Parameter	5

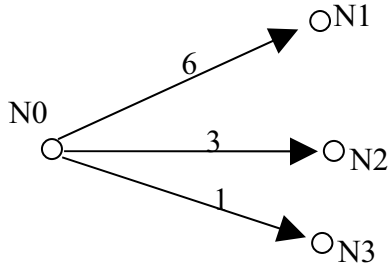
3. Routing

3.1 Bandwidth measurements and buffer

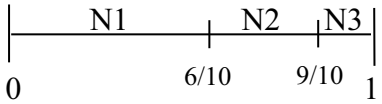
Every node of our network has a Router class. The class Router contains a bandwidth table and also a buffer. The bandwidth table contains routing elements (nodes of the leaf-set). The Router class contains methods for the initialisation and for the update of the table. A bandwidth table is an instance of the BandwidthTable class, which allows access to the routingElement entries and contains bandwidth from the current node to the specific nodes and collects their output capacities. The output capacity of a node is the sum of the bandwidths from this node to all other nodes in his leafset. For the concrete bandwidth measurements between the network nodes we use the Iperf utility. The buffer is an instance of the MessageBuffer class. This class extends a ConcurrentLinkedQueue (FIFO). The current size of the buffer represents the sum of sizes of all chunks, and the maximum buffer size can be arbitrarily defined (setMaxBufferSize(int)). A condition of overflow should never occur (currentBufferSize <= maxBufferSize).

3.2 Routing Algorithm

First we have to normalize the bandwidth measures to each node of the leafset. Then we split the range between 0 and 1 into subranges. The length of these subranges corresponds to the normalized bandwidth measures. The sum of the normalized units is equal to 1. In the next step we calculate a random number (between 0 and 1). This random unit will be part of one of the created subranges. There is a high probability that the random number will occur in the bigger range than in a smaller one and that the chunk will be forwarded to the node relative to the subrange. In the long run the majority of the chunks will be forwarded to nodes with high bandwidth, but also the nodes with smaller bandwidth will be involved in the sending process.



Subranges:



Ex.: if Random unit = **0.2**
→ send chunk to **N1**

4. Wahl des Design

Wir haben uns für Freepastry entschieden, damit wir die Kommunikation und deren Aufbau Freepastry überlassen können. Weiter wollten wir uns nicht um die Eigenschaften der Knoten kümmern. Somit versuchten wir die gute Infrastruktur von Freepastry ausnutzen. Den Aufbau des P2P Netzwerkes übernimmt so die Opensource Software.

Weiter wollten wir für jeden Knoten selber eine zentrale Stelle haben, die für jeden einzelnen Knoten die Nachrichten verwaltet und weiterleitet. Diese Klasse startet auch das Routing und somit werden die Chunks über einen möglichst schnellen Weg gesendet.

5. Probleme

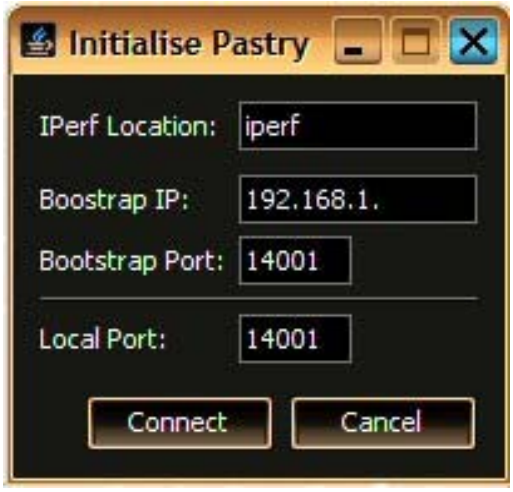
Das Hauptproblem für dieses Projekt ist die sehr schlechte Dokumentation von Freepastry. Man brauchte und verbrauchte so ziemlich viel Zeit für die Einarbeitung. Natürlich dachten wir, das vielleicht einige Beispiele von Freepastry zu finden wären. Dem war aber nicht so, was die Arbeit sehr erschwerte.

Ein weiteres Problem, das mit der Zeit auf uns kam, war das Routing. Wir hatten ziemlich viel Schwierigkeiten einen laufenden Algorithmus zu entwickeln. Gründe dafür ist sicherlich die schlechte Dokumentation der Methoden und dass wir kein simples Beispiel für einen Routingalgorithmus finden konnten, das mit Freepastry arbeitet.

Weiter hätten wir mehr Testmöglichkeiten in den Netzwerkräumen haben sollen, um unser Projekt besser voranzubringen.

6. Manual

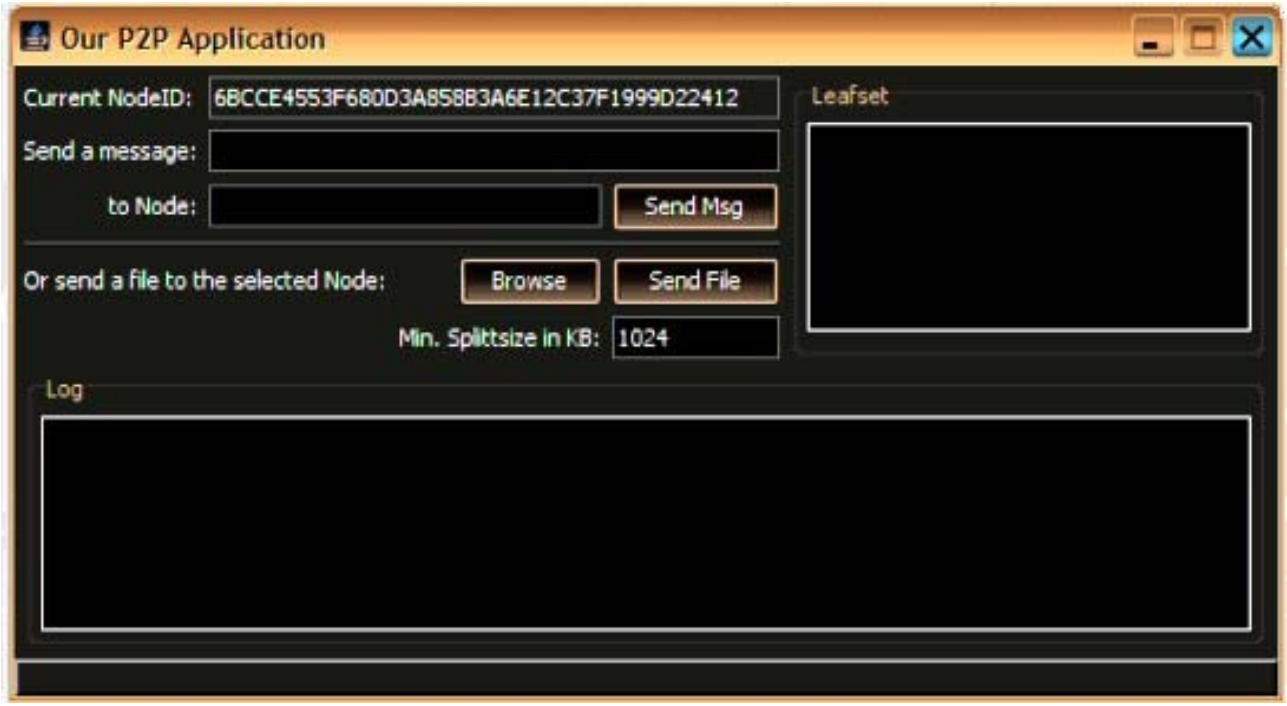
Connection Window: window used for the connection of the nodes to the network.



Input-fields:

- **IPerf Location**: Location of the Iperf utility (bandwidth measurement).
- **Bootstrap IP**: IP address of the bootstrap node. The first node initialise the pastry network (no need of IP input).
- **Bootstrap Port**: Bootstrap Port used.
- **Local Port**: Local Port.

Node Window:

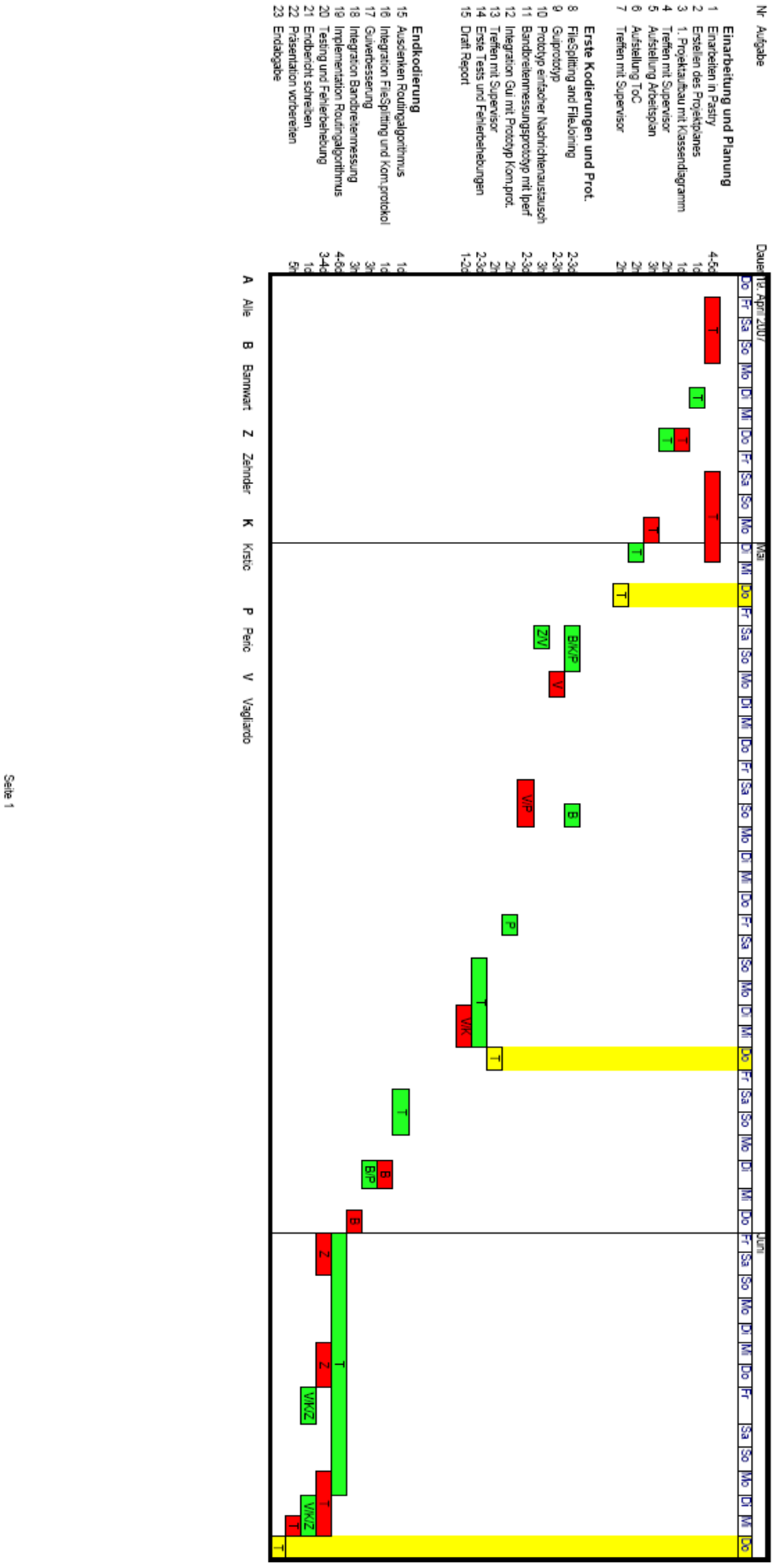


- After the connection to the network every node will be represented by such a window. The node window contains information fields and also fields used to send files and messages.
- **Information fields:**

  - **Current NodeID**: Contains the ID of the current node.
  - **Leafset**: This field contains ID's of the nodes which are in the leafset of the current node.
  - **Log**: The Log-field will show important activities and events concerning the node (examples of shown events: leafset node failure, received chunks, files or messages from other nodes-origin, destination and next hop,…)

- **Send a Message…to Node:** We can send simple messages to a node by selecting it in the leafset field.
- **Send a File…to Node:** Used to send files to nodes also by selecting them in the leafset field.

8. Projektplan



9. Schlussreflektion

Die ganze Idee einer Implementation einer P2P Applikation war sicher eine spannende Sache. Leider war die Wahl von Freepastry nicht die Beste, da wir mehr Problemen hatte, als es uns nützte. Die grösste Herausforderung war sicherlich einen funktionierenden Routingalgorithmus zu finden, dies haben wir mit mehr oder weniger Erfolg erreicht. Die Robustness war auch ein weiterer Knackpunkt, den wir ebenfalls fast erreicht haben. Dafür funktioniert das Forwarding und man kann mehrere Files gleichzeitig senden.