

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

A Peer-to-Peer Framework for Social Networks

Project Group 2012/2013

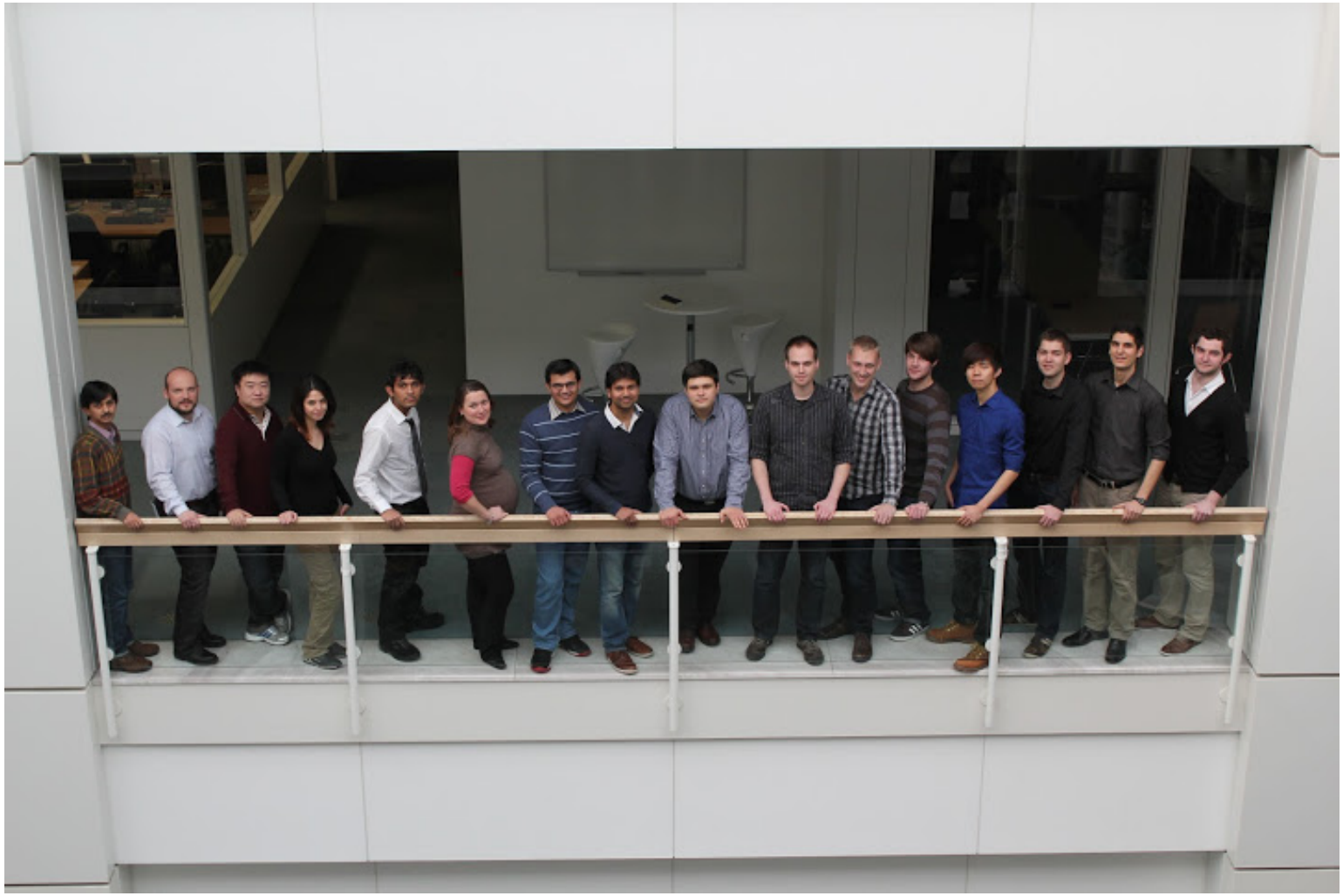
Jun.-Prof. Dr. Kalman Graffi

Institut für Informatik
Fakultät Elektrotechnik, Informatik und Mathematik
Universität Paderborn

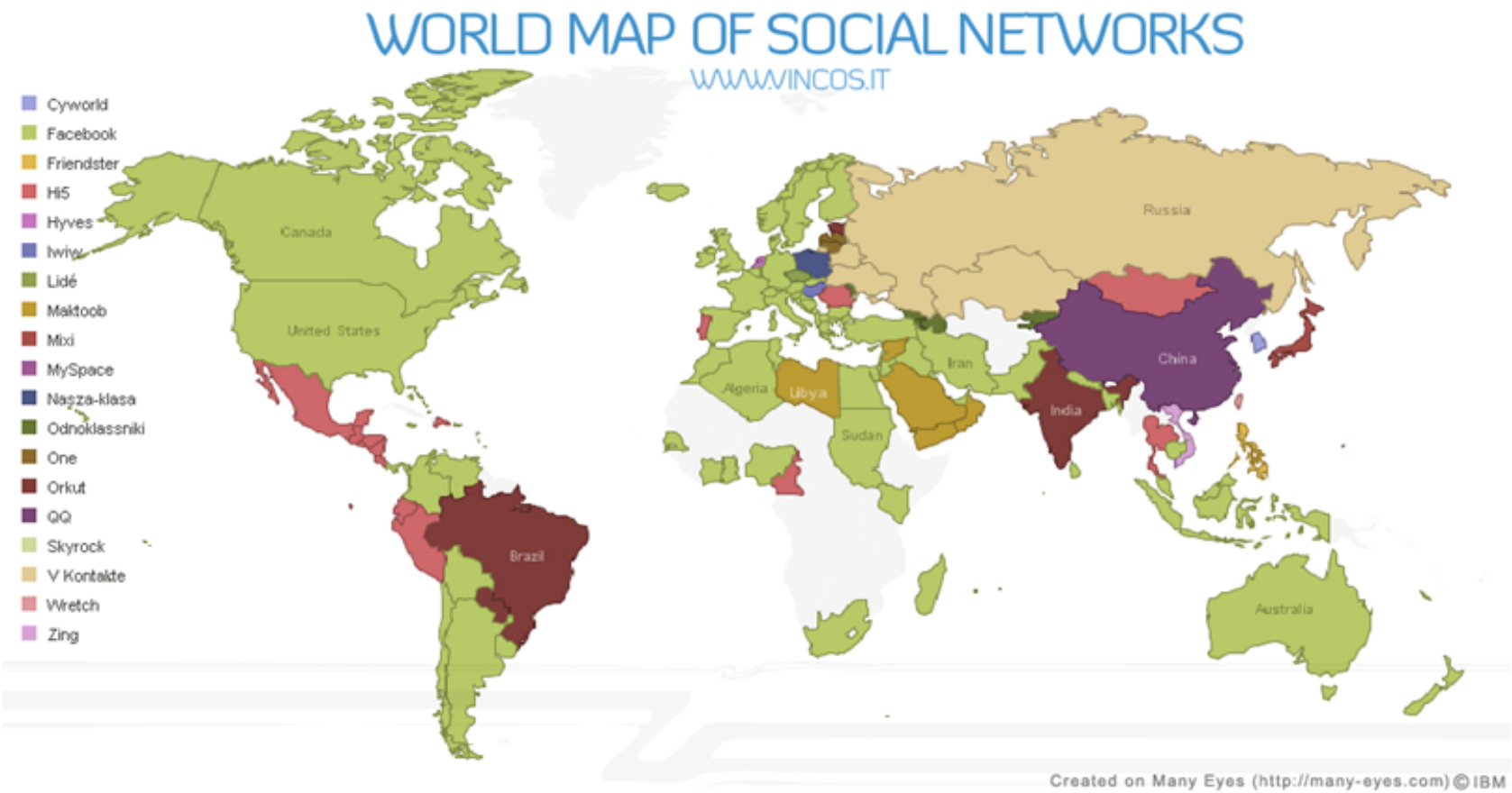
26. März 2013



PG 2012



Internet 2.0


WORLD MAP OF SOCIAL NETWORKS
WWW.VINCOS.IT
Cyworld
Facebook
Friendster
Hi5
Hyves
Iwiw
Lidé
Maktoob
Mixi
MySpace
Nasza-klasa
Odnoklassniki
One
Orkut
QQ
Skyrock
V Kontakte
Wretch
Zing
Canada
United States
Brazil
Russia
China
India
Iran
Algeria
Libya
Sudan
Australia
Created on Many Eyes (http://many-eyes.com) © IBM

Social Networks

Social Communities

- Facebook, MySpace ...

Cooperation Portals

- Google Docs

Communication Tools

- Skype

Issues

- Centralized
- Security
- Availability



Solution: Decentralization

Peer-To-Peer Technology

- No server needed
- Self-organizing system

Security

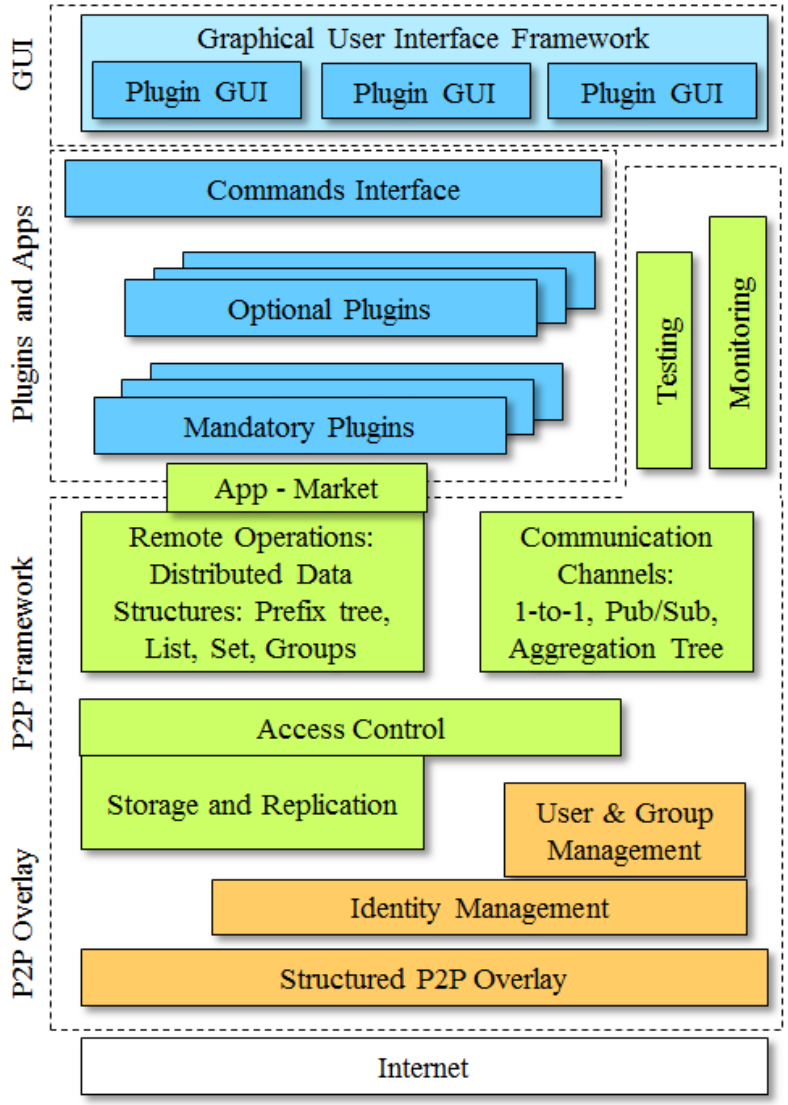
- Control the data access

Availability

- Social network is down. What about the data?



P2P Framework



Agenda

Introduction

Overlay

Framework

LifeSocial

Demonstration

Testing

Summary



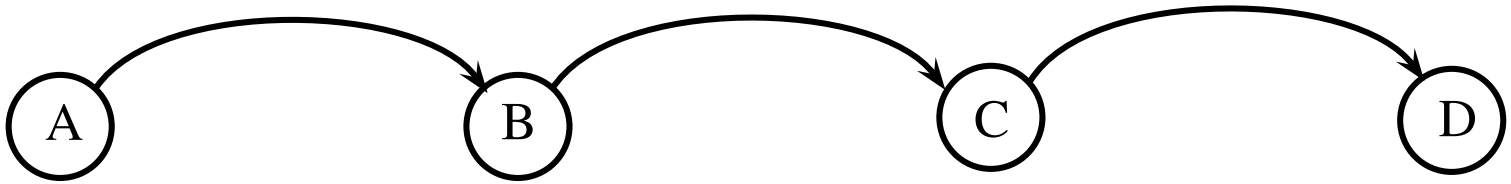
Overlay

- FreePastry
  - Version 2.1
  - PG 2011 (.jar) -> Currently (Source files)
- Security
  - Attacks
  - Parallel-Iterative Routing
  - Optimization
- Capacity Awareness
  - Heterogeneity
  - Mobile/Weak Nodes



Security - Introduction

- Problems:
  - Always there exists malicious and evil nodes
  - Network is target of different types of attacks:
    - Sybil attack
    - Eclipse attack
    - Routing attack
- Default functionality: Freepastry (recursive routing)





Security - Our approach

- Motivation:
  - No replies
  - Misguiding replies or
  - Reply with false answers
  - No way to verify the validity of respond
  - A malicious node could create a malicious loop
- Parallel - Iterative Lookup -> decrease probability of routing attack

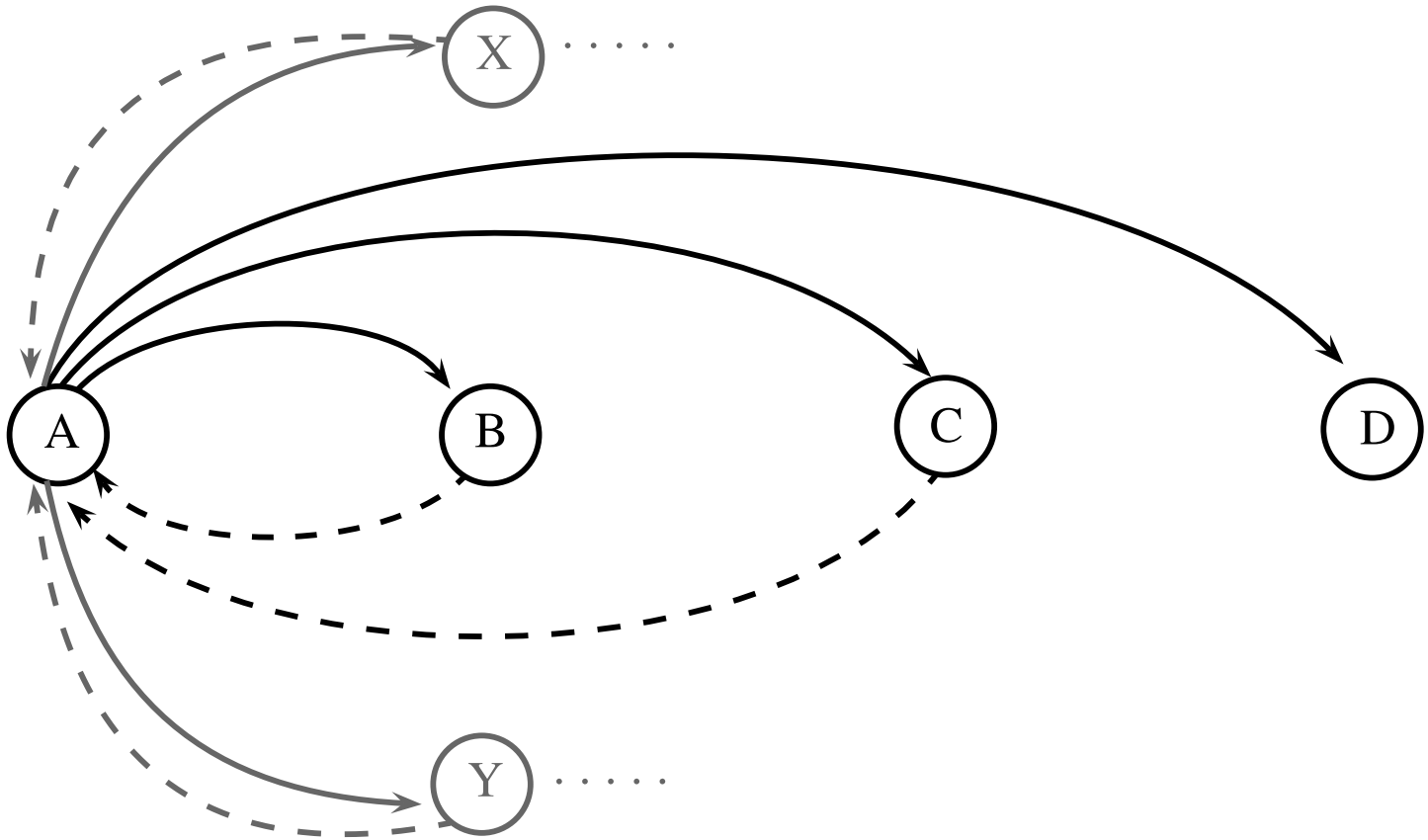


Parallel-Iterative Routing

- Parallel Routing concept:
  - Routing table with K bucket
  - Sends message to m other nodes (parallel routing)
- Iterative Routing concept:
  - Introduction of source node
  - Every node replies back to the source node
  - Source node has control over lookup
  - Middle hop cannot mislead the lookup



Parallel-Iterative routing





Security - Optimization

- Source node must stop solved queries
  - Introducing queryNumber
- Sorted list : source node keeps the last k best answers (k=3)
  - Not send messages to the same/worse distant
- Avoid redundant hops
  - Maintain a list of contacted nodes
  - Not to send messages to the same node
- Configurable functionality



Security - Future works

- Implementation challenges:
  - Finding promising approach -> not easy
  - Overlay implementation : effects main functionality
- Sybil attack
- Eclipse attack
- Anonymity



Heterogeneity - Motivation

Situation before our PG

- FreePastry balances the workload evenly in the network
- However nodes' capacities in the network is not even $\Rightarrow$ Nodes with little capacity may be overloaded!

Challenges

- No central workload distribution control
- Workload has to be distributed dynamically based on node's capacity



Heterogeneity - Mechanism

Idea: The approach is based on a multi-provider solution

Paper: *Load Balancing for Multimedia Streaming in Heterogeneous Peer-to-Peer Systems - Graffi et al.*

- **Responsible Node:**
  A node that holds the file.
- **Requester:**
  A node that wants to retrieve a file.
- **Provider:**
  A node that was a requester and successfully retrieved the item.



Heterogeneity - Capacity

Current Metrics

- Memory
- Storage
- Bandwidth

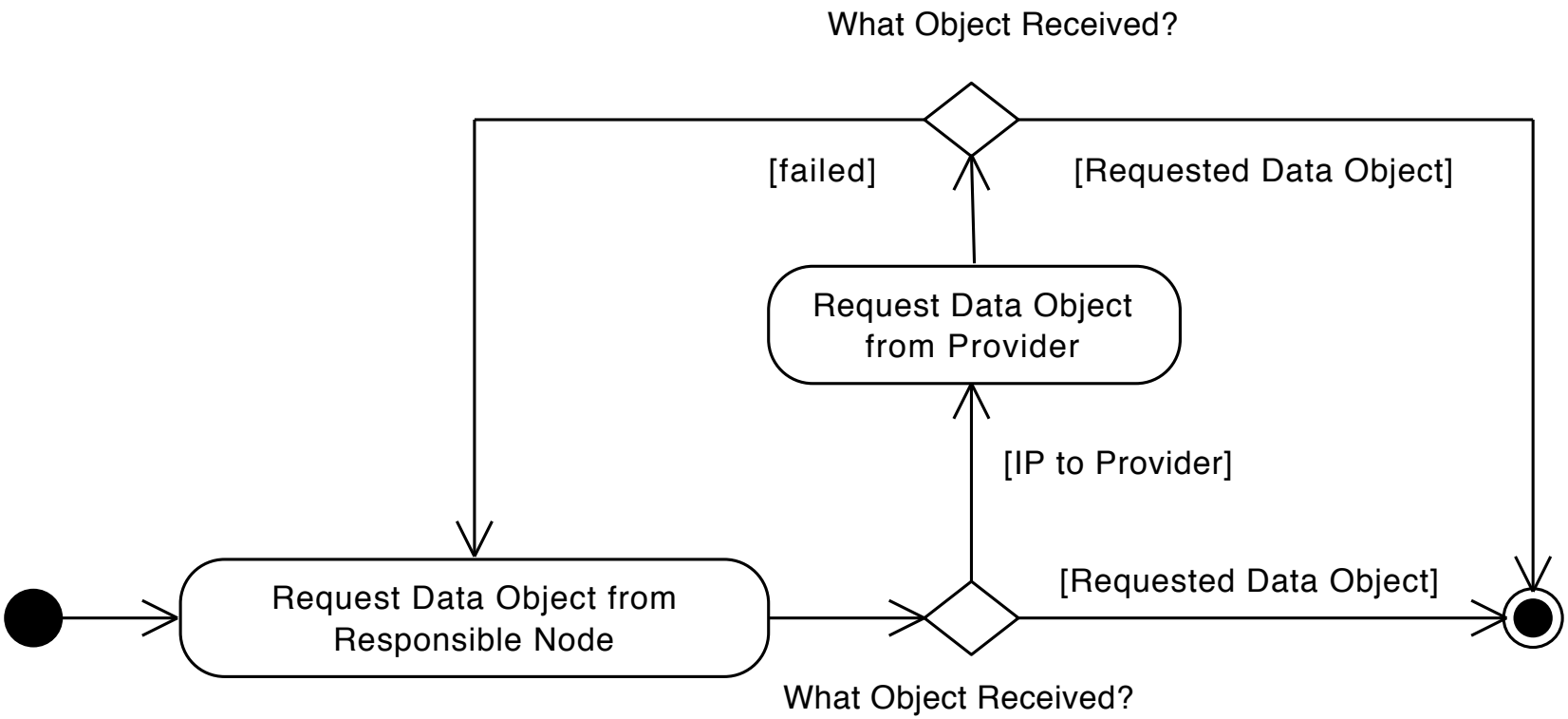
Bandwidth is calculated within the network

- By message's size and message's round trip time
- Calculated during the request process

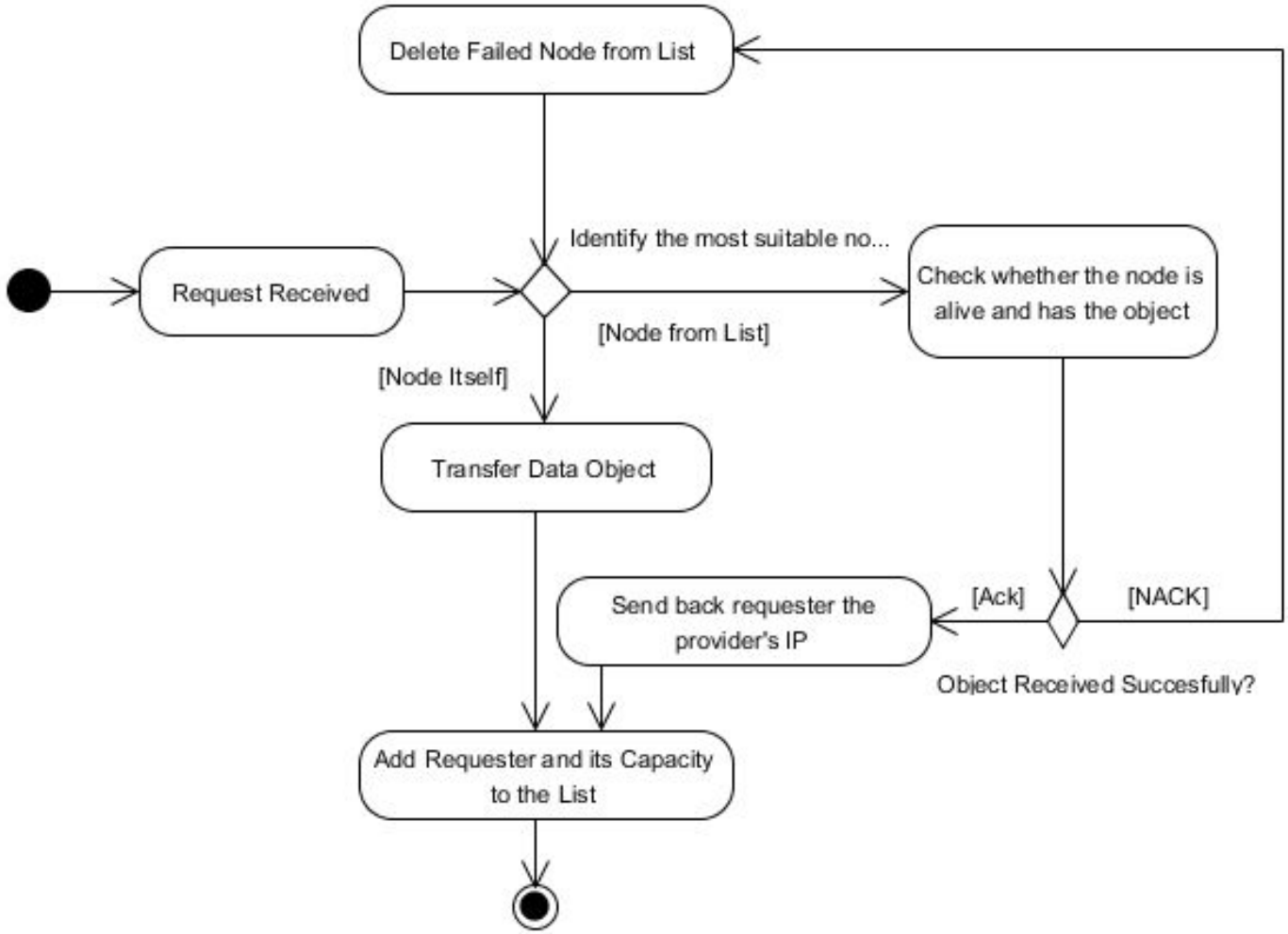
High capacity $\Rightarrow$ High probability to be chosen as provider.



Heterogeneity - Requester's view

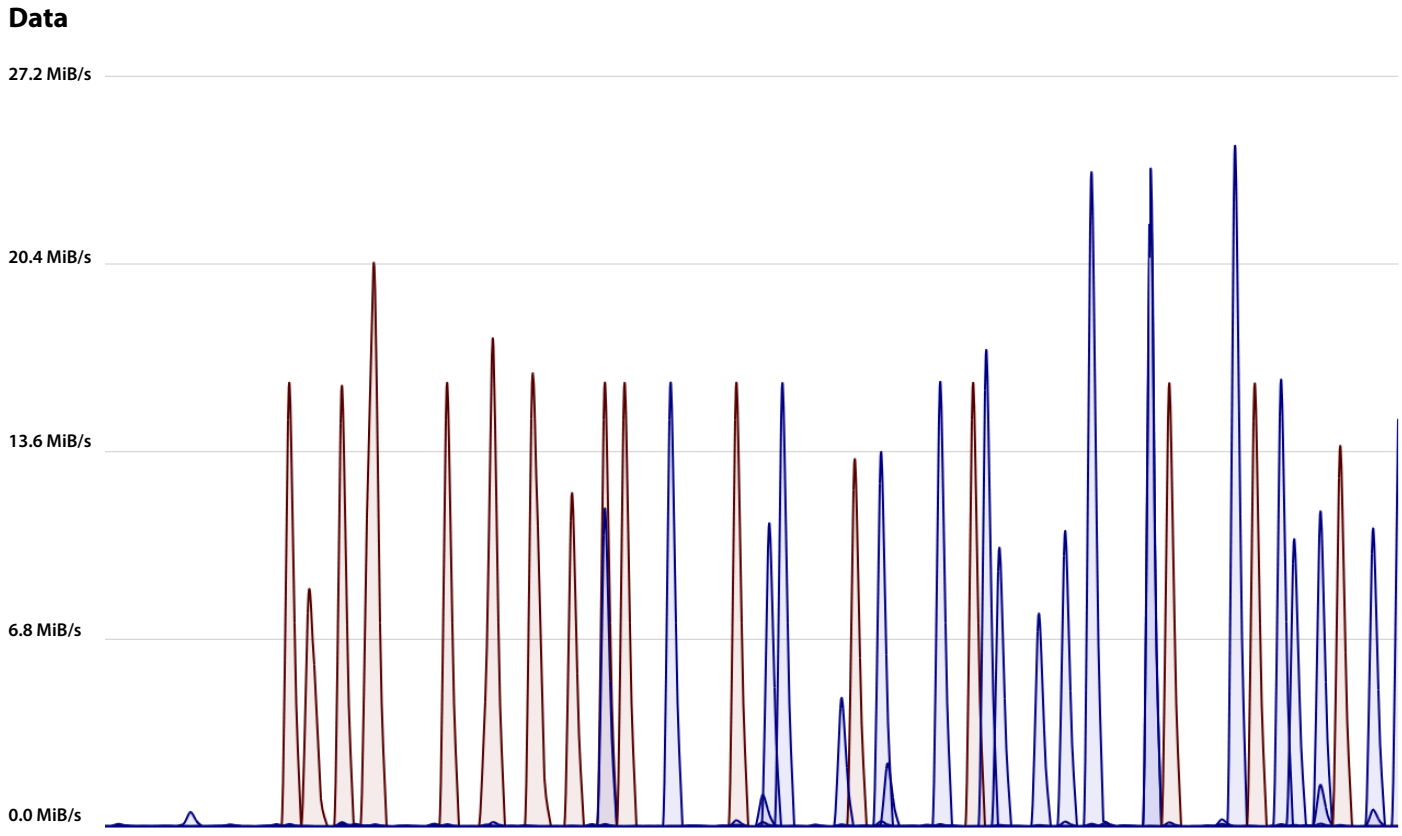


Heterogeneity - Responsible node's view



Heterogeneity - Test result

10 Machines, 3x TestCaseLoadPhoto in 0.2m
red: responsible node, blue: a certain provider



Mobile/Weak nodes handling - Motivation

Situation before our PG

- FreePastry does not consider mobile nodes

Challenges

- Mobile nodes have relatively little storage space and an unreliable network connection
- It may join and leave the network frequently $\Rightarrow$ Reduces Performance!
- How to identify a mobile node



Mobile/Weak nodes handling - Basic solution

Idea: make strong nodes nearby work as relay nodes!

What is implemented in our solution:

- Mobile nodes are not responsible for routing and storing
- Nearby strong nodes forward messages and store data
- Mobile nodes are identified by ports
  - odd $\rightarrow$ strong, even $\rightarrow$ weak

Required enhancement in future:

- Consider Replication.



Agenda

Introduction

Overlay

Framework

LifeSocial

Demonstration

Testing

Summary



Access Control - Motivation

Situation before our PG

- read access control: encryption of single items (single users, groups)
- write access control: none!

Challenge

- no central access control possible
- completely based on cryptographic methods
- every node can be malicious



Read Access - Requirements

Users want to share contents only with a certain group of other users/ friends

Several requirements should be considered

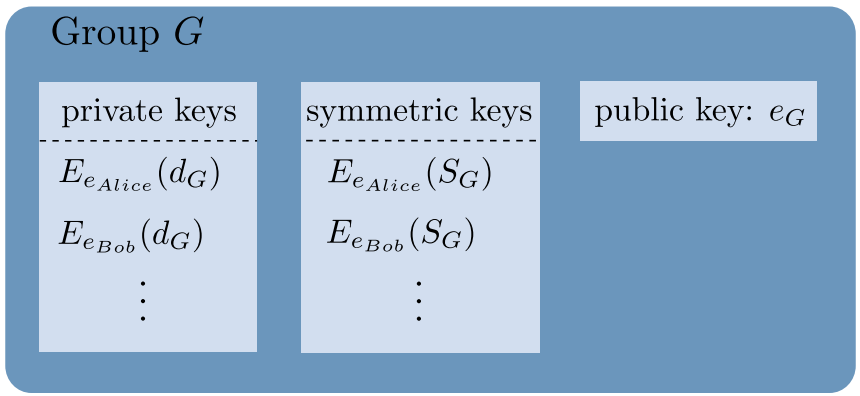
- only defined set of users can read content
- group of authorized users may vary from item to item
- possibility of adding and removing authorized users
- hierarchy and combination of such access groups



Read Access - Basic Solution

Idea: items are stored encrypted; make key only available to authorized users

`Group` provides a symmetric key s and an asymmetric key pair (e, d) for its members



Reference to group is stored together with item

Read Access - Enhancements

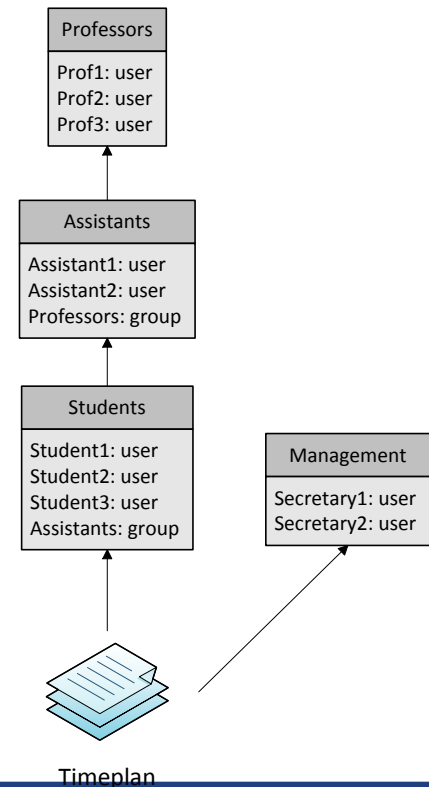
Challenge: Allow more complex and hierarchical structures of groups

- share content with combination of groups: $G_1 \cup G_2$, $G_1 \cap G_2$
- hierarchical groups: G_1 inherits access rights from G_2

Idea: members of a group can be in turn groups

Encrypted item has a reference to the group used for encryption

User is not direct member of group $\Rightarrow$ apply a `Depth-first search` to get decryption key



Write Access - Requirements

Situation before our PG

- no write access control at all $\Rightarrow$ everybody can (over-) write everything

Items have a dedicated owner $\Rightarrow$ only owner can (over-)write and delete an item

Owner can either be a single user or a set of users ($\rightarrow$ use groups)

Challenge: secure authentication of the owner based on cryptographic methods



Write Access - Basic Solution

Insert an item

- store the item together with public key of owner

Write an existing item

- sign new item with private key corresponding to public key stored along with the existing item
- storing node verifies the signature and possibly denies the write access

Protection against replay attacks

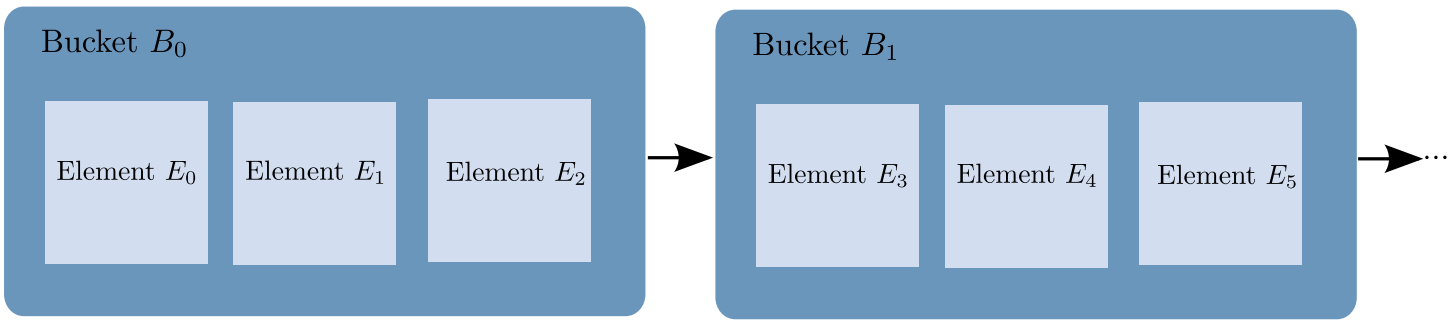
- random nonce is stored with item; nonce must be signed together with new item



DistributedList - Basics

Idea: list is split into buckets and stored on different nodes

- unload a single node
- retrieve and sign a small bucket instead of the whole list



Requirements:

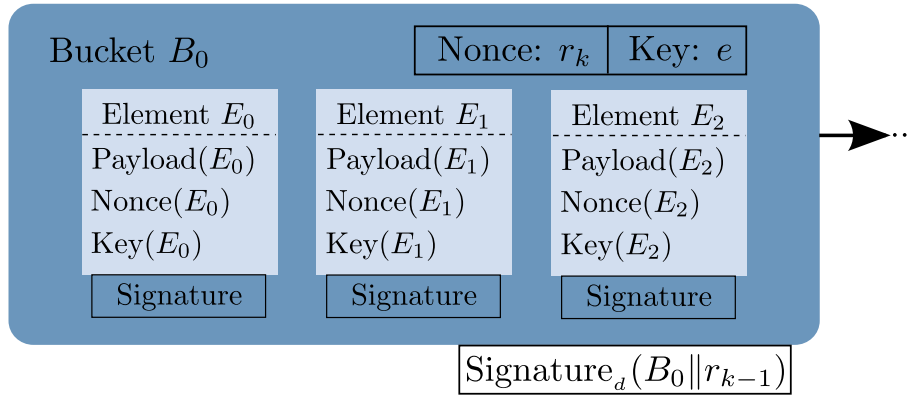
- only owner allowed to edit/ delete a list entry
- everyone allowed to add a list entry
- support groups $\rightarrow$ only group users allowed to read or add a list entry



DistributedList - Write Access

Add entry to the list

- sign element with private key of the owner
- encrypt each element of the bucket with symmetric group key
- sign the bucket with private group key



Edit/ delete an existing element

- sign new entry / a `deletedItem` as usual
- storing node recognizes list buckets and verifies signatures of bucket+elements

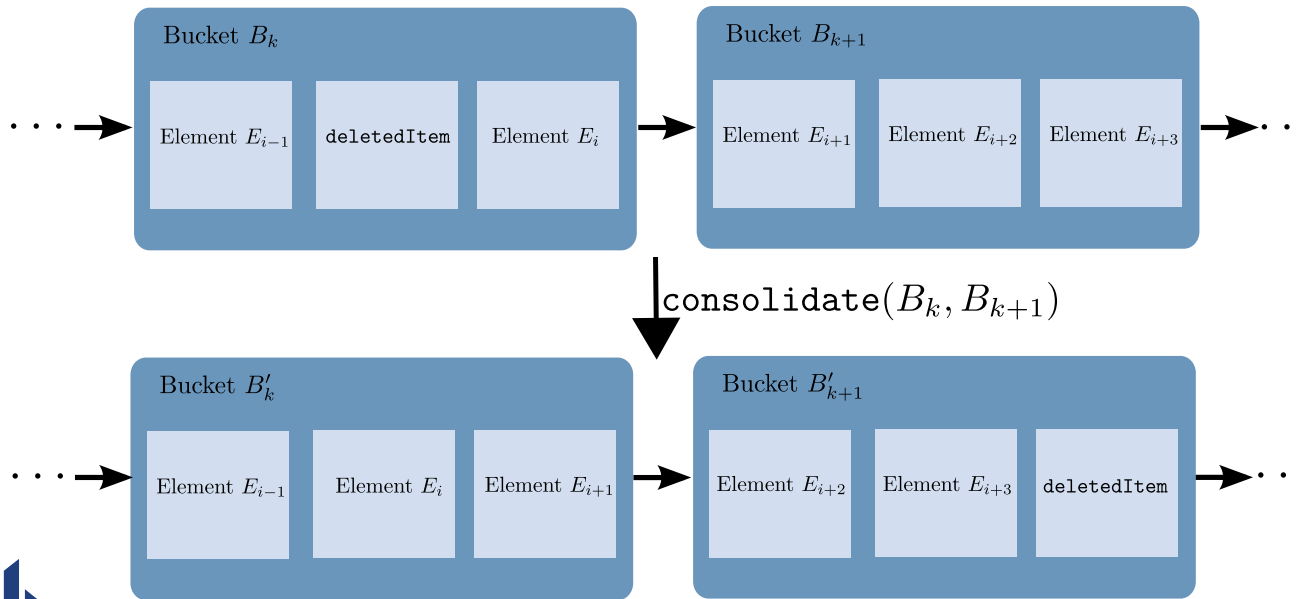


DistributedList - Consolidation

Problem: deletedItems stay in list; list might contain almost only deletedItems

Solution:

- algorithm consolidates two buckets → deletedItems shifted right
- called after each write operation if list contains deletedItem



over time list contains only a small amount of deletedItems

RemoteOperation

DistributedList (and Groups) use the idea of remote operations

- user wants to change a stored item
- user sends request for change to storing node
- storing node performs change and stores the file

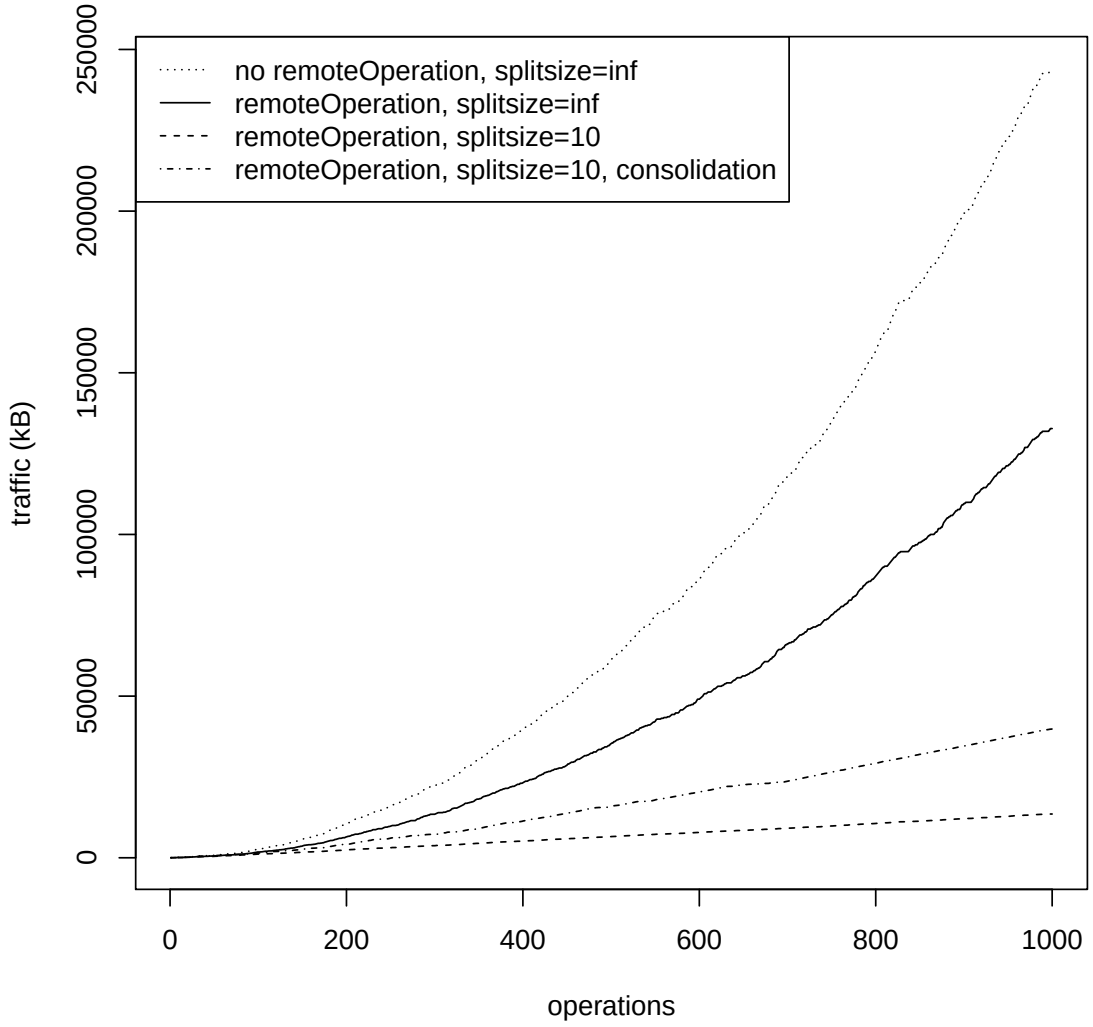
Advantage: user need not transfer whole file but just the changes

Problem: write protection (as described before) does not work

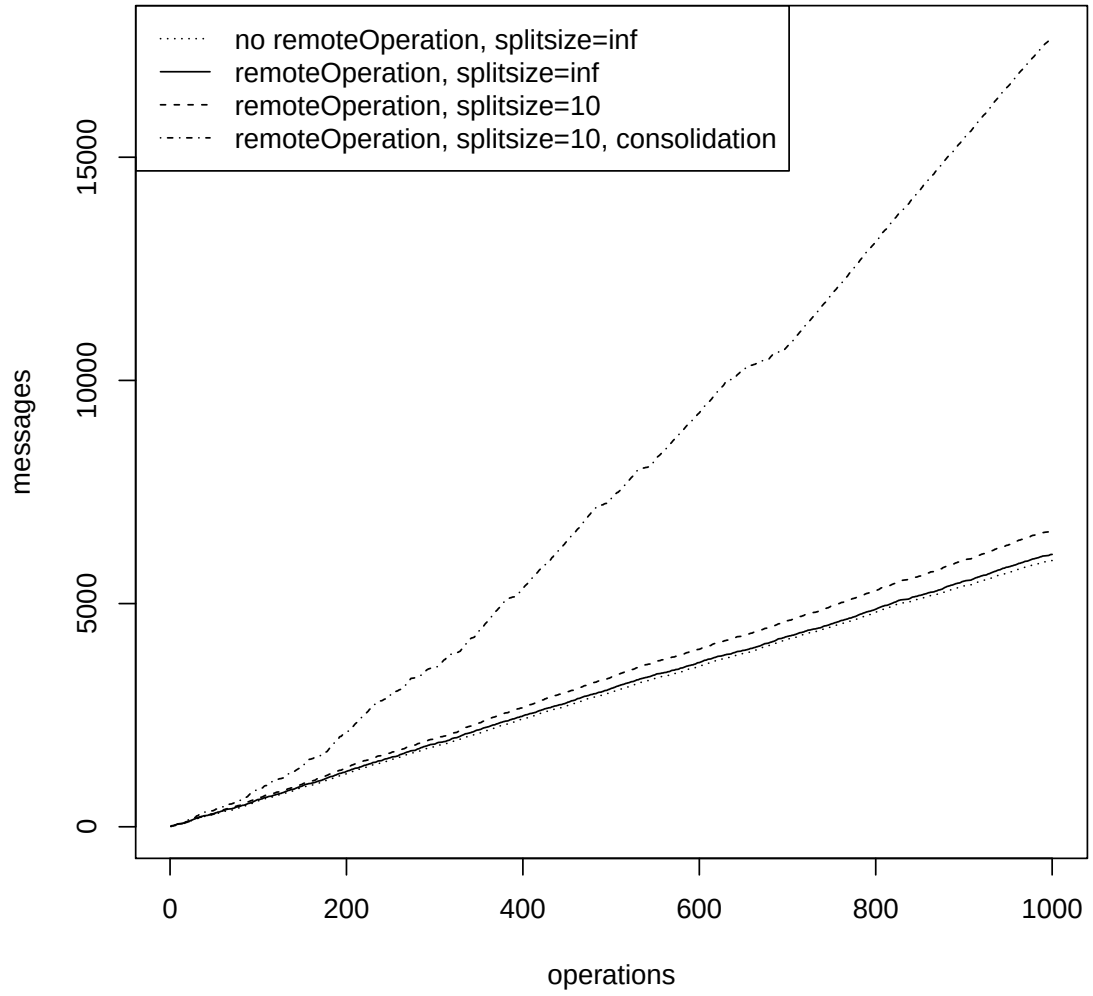
- storing node performs changes and stores the modified item
- **but:** storing node is usually not owner of the item $\Rightarrow$ cannot compute valid signature
- we developed a protocol to address this problem



DistributedList - Evaluation

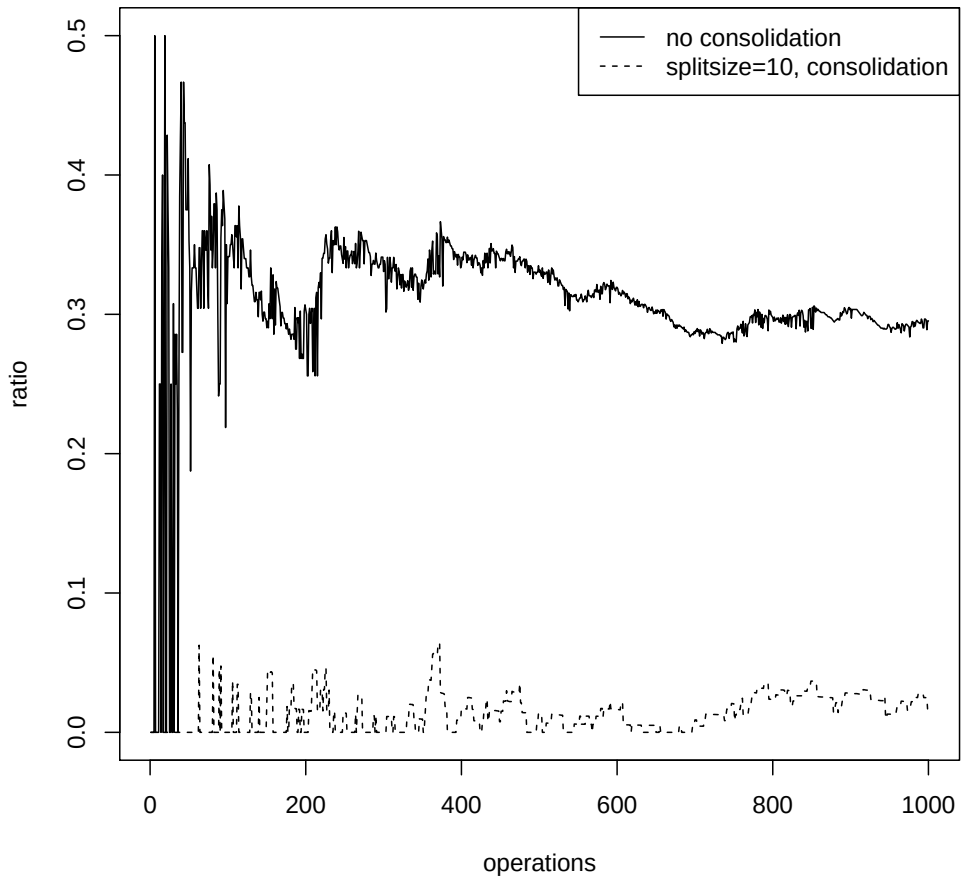


DistributedList - Evaluation



DistributedList - Evaluation

ratio := #deletedItems/ (#deletedItems + #real items)



Storage Abstraction

Stored objets often have similar requirements

- unqiue identifier
- associations between each other (to-one, to-many)
- dependencies to other objects
  - e.g. ForumPost cannot exist without a ForumThread

AbstractStorageItem

- common way for identifying objects (using UUID)
- allows specification, storage and lazy-loading of related objects
  - identified by a keyword (e.g. „posts“)
  - to-many: distributed data structures
- automatically delete dependent objects



Storage Abstraction

Stored objets often have similar requirements

- unqiue identifier
- associations between each other (to-one, to-many)
- dependencies to other objects
  - e.g. ForumPost cannot exist without a ForumThread

AbstractStorageItem

- common way for identifying objects (using UUID)
- allows specification, storage and lazy-loading of related objects
  - identified by a keyword (e.g. „posts“)
  - to-many: distributed data structures
- automatically delete dependent objects



Agenda

Introduction

Overlay

Framework

LifeSocial

Demonstration

Testing

Summary



LifeSocial



Example Application

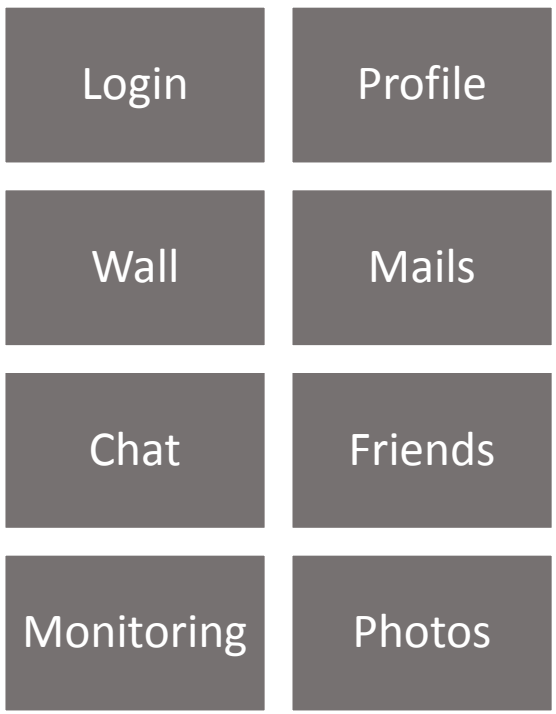


Social Network



Monitoring & Testing

Previous Work





Goals

E-Learning	Collaborative Work	Social Networking
• Share Files • Groups • Wikis • Forums	• Collaborative Editors • Calendar • Video/Audio Chat	• Friend Graph • Activity Stream



What did we do?

Login	Profile	Groups	Forum
Wall	Mails	Browser	App Store
Chat	Friends	Voting	File Hosting
Monitoring	Photos	File Transfer	Multi-Chat



Extensibility



Login	Profile	Groups	Forum
Wall	Mails	Browser	App Store
Chat	Friends	Voting	File Hosting
Monitoring	Photos	File Transfer	Multi-Chat





Appstore - Requirements

Appstore makes the expanding of available functionalities as well as the creating of new features possible.

Some Requirements:

- every user may act as 2 roles: plugin provider and plugin user
- plugin provider can pubilish new plugins, release new versions, edit or delete them
- plugin user can search for the plugins, download and install them, the management of installed plugins (start/stop/update/uninstall) should also be avaliable
- related GUI for the above mentioned functionalities:
  - Publisher View
  - Search View
  - Installed Plugins View





Appstore - Solutions

Model

- divide the plugins into 2 parts:
  - `plugin object`
  - `metadata`
    - `plugin meta`
    - `version meta`
- `private repository` contains the information of:
  - `published plugins`
  - `installed plugins`



Appstore - Solutions

Behaviors

Publisher:

- Actions: publish, remove, release new version, edit the metadata
- Plugin object and metadata ⇒ network
- Meta-information of the published plugins ⇒ private repository
- UI: Publisher View

Search:

- Actions: precise search, download and install
- Warning: exact plugin name is required
- Output: list of avaliable versions
- UI: Precise Search View





Appstore - Solutions

Plugin user:

- Actions: start, stop, update, uninstall
- Warning: update only if there is new version available
- UI: Installed Plugins View



Appstore - Enhancement in future

Repository:

- opened- and closed repositories
- related GUI

Dependencies management:

- check for lacking of dependencies
- automatical installation of required plugins



Demonstration



Agenda

Introduction

Overlay

Framework

LifeSocial

Demonstration

Testing

Summary



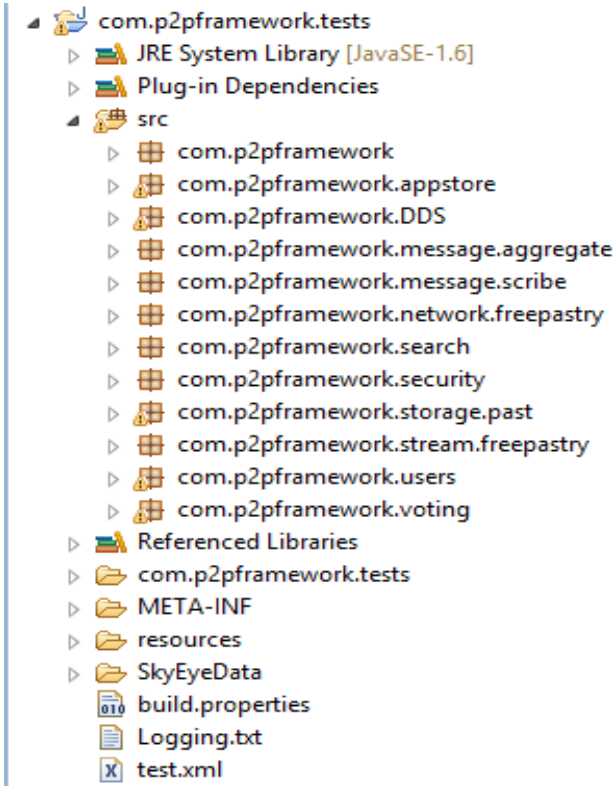
Testing

- Why is it important?
- Monitoring
  - Various metrics (lookup time, hops required...)
  - Performance Analysis
  - Optimization
- Two kinds of test for LifeSocial
  - JUnit test specific modules/projects
  - Large scale test with multiple machine and test plan



JUnit test

New tests included for appstore, framework, voting and many more features





Large scale test

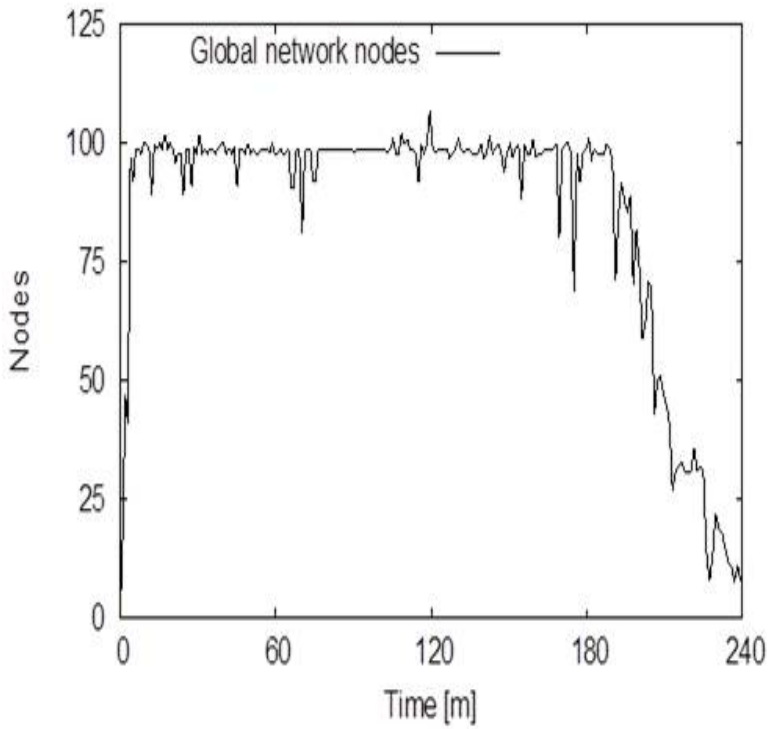
- Idea
  - Running test plan on multiple machine
  - 1 machine acts as a master and all the rest as slaves
  - Masters initiates the test plan on all machine
- Setup Environment
  - Multiple instances on each machine (currently 1)
- Additional Features
  - New test plan developed as a part of thesis (by Pascal)
  - Inclusion of Forums and other new features



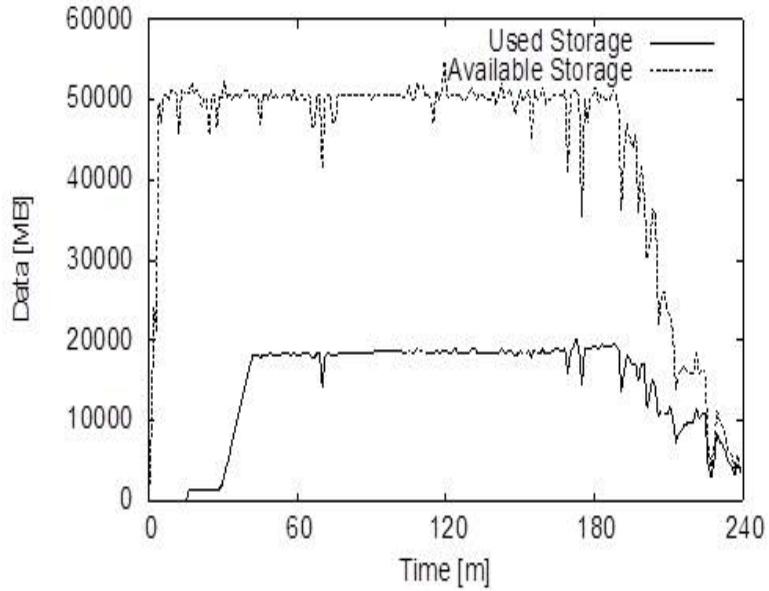


Test results

Amount of nodes

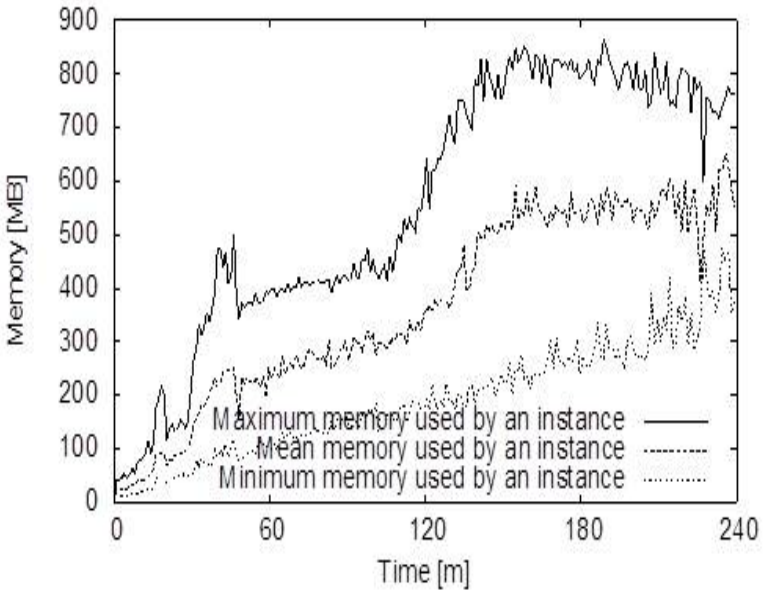


Storage

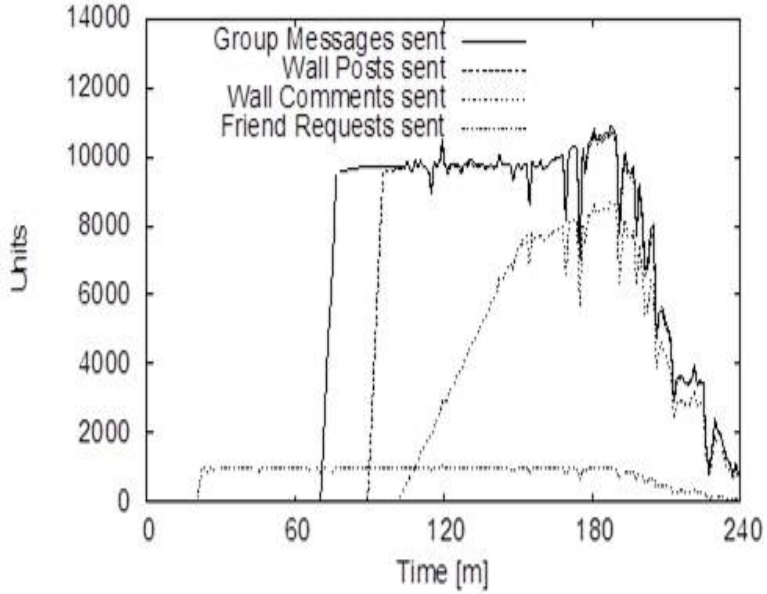


Test results

Memory usage

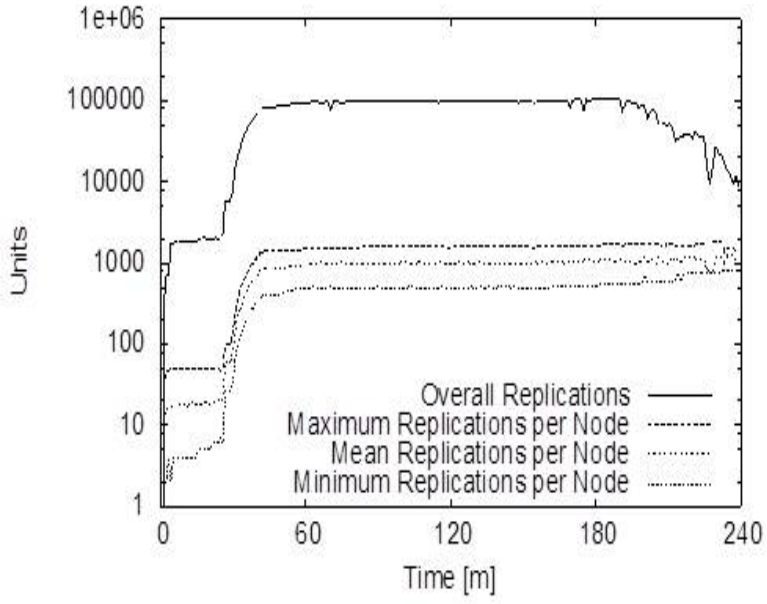


Messages

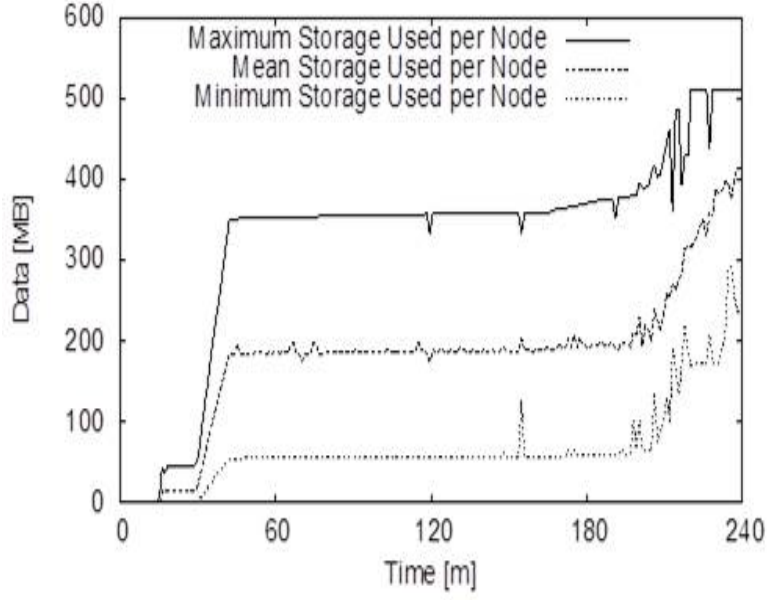


Test results

Replication count



Storage per node



Agenda

Introduction

Overlay

Framework

LifeSocial

Demonstration

Testing

Summary



Summary

P2P Framework

- Messaging
- Storage
- Distributed Datastructures

LifeSocial

- Using the P2P Framework
- Extendable

Overlay

- Routing
- Performance

