HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

A P2P-Framework for Social Networks

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

www.p2pframework.com



A P2P-Framework for Social Networks

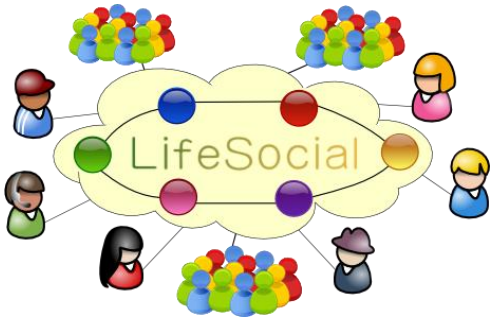
Overview – A quick 101

LifeSocial – A Secure P2P-based OSN Platform

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

History

- Developed since 2007
  - > 55 students worked on it
  - See you at CeBIT 2013
- Aiming at applicable results in p2p research



Goal

- Facebook-like user experience
  - Basis functionality extendible through plugins
  - Data-centric (profiles) and user-to-user (chat,video) interaction
  - BUT: security guarantees
- Operator view
  - Completely p2p-based
  - BUT: with quality of service control and guarantees
- Research
  - New application leads to new requirements
  - New requirements to new results

Main Ideas in LifeSocial

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

General p2p platform

- Combining a wide set of useful modules
  - Storage, messaging, security, caching, app-hosting, multicast, pub/sub …
  - Distributed data structures, monitoring, automated self-control
- Social network on top of platform
  - Build through “plugins” (apps), using platform and each other
  - Extendable, configurable GUI supports app growth

Security goals

- Access controlled secure storage
- Secure (encrypted, authenticated, integer) communication
- No trust assumptions → anybody may be bad

Functional goals

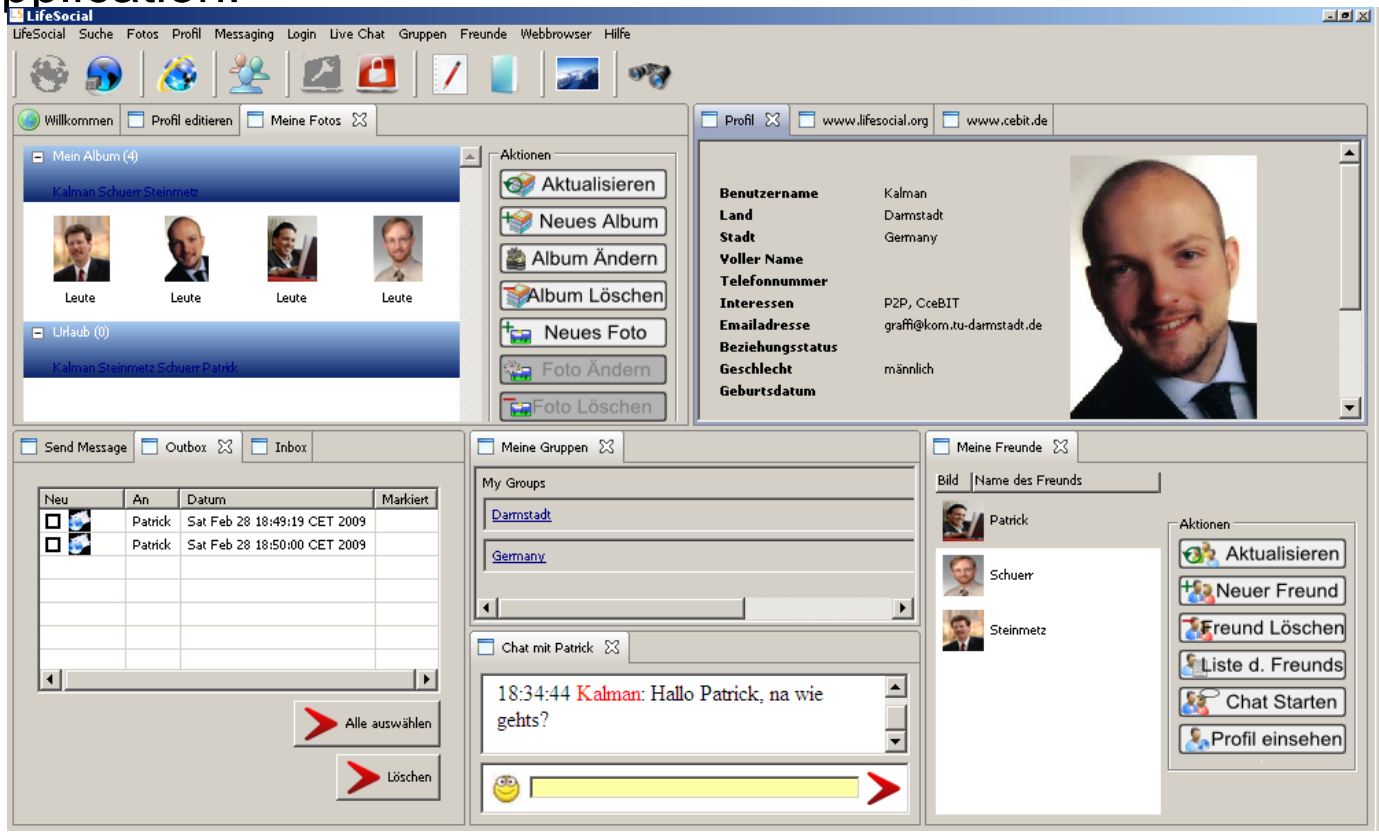
- For the users: Facebook-like
- Also thinking of providers: quality monitoring and control plane

User View: Rich Functionality

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

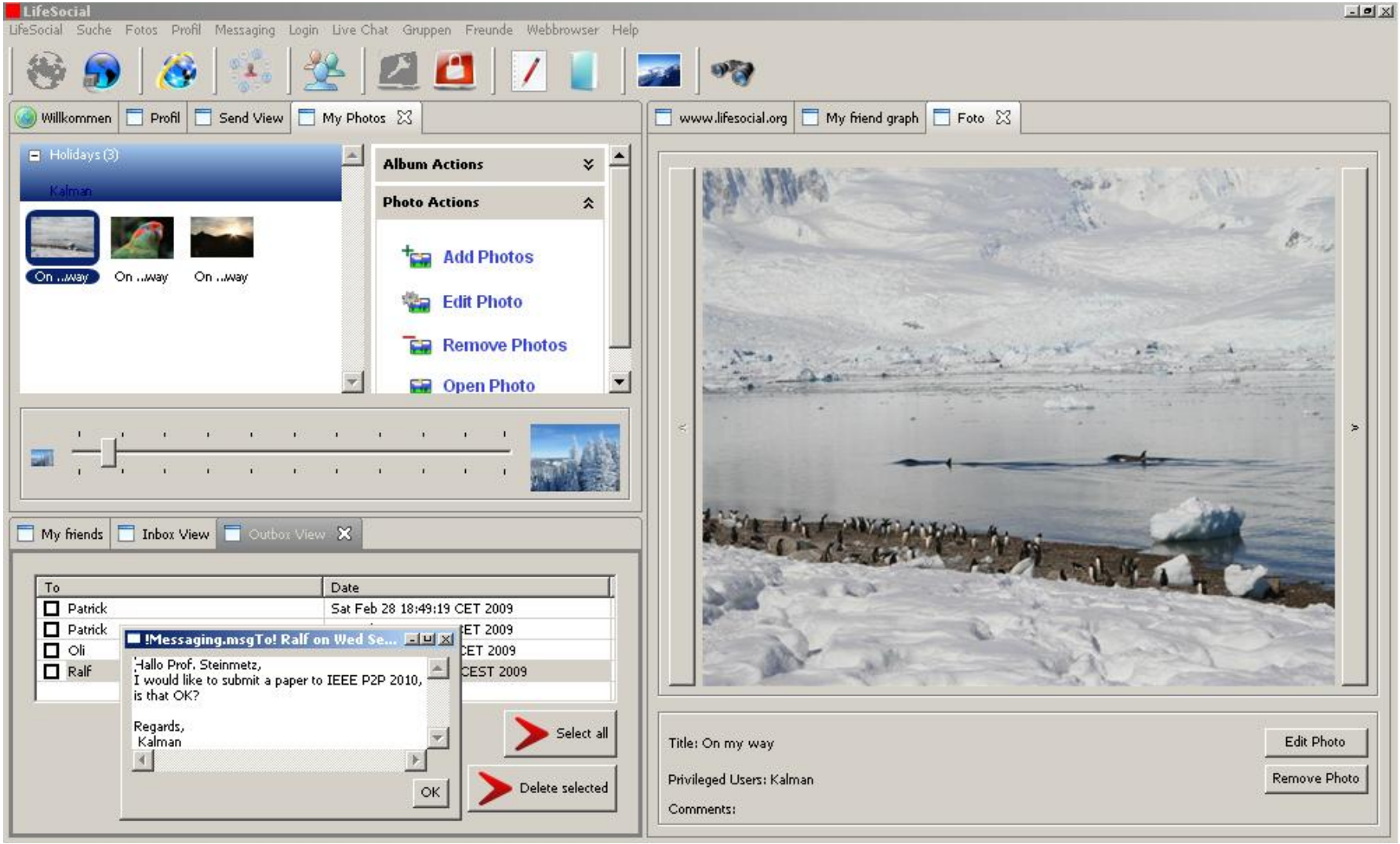
Wide set of functionality

- GUI-Framework like in Eclipse
- Fast and user-friendly performance
- Plugin-based application:
  - Profile
  - Login
  - Friends
  - Groups
  - Mails
  - Photos
  - Chat
  - …

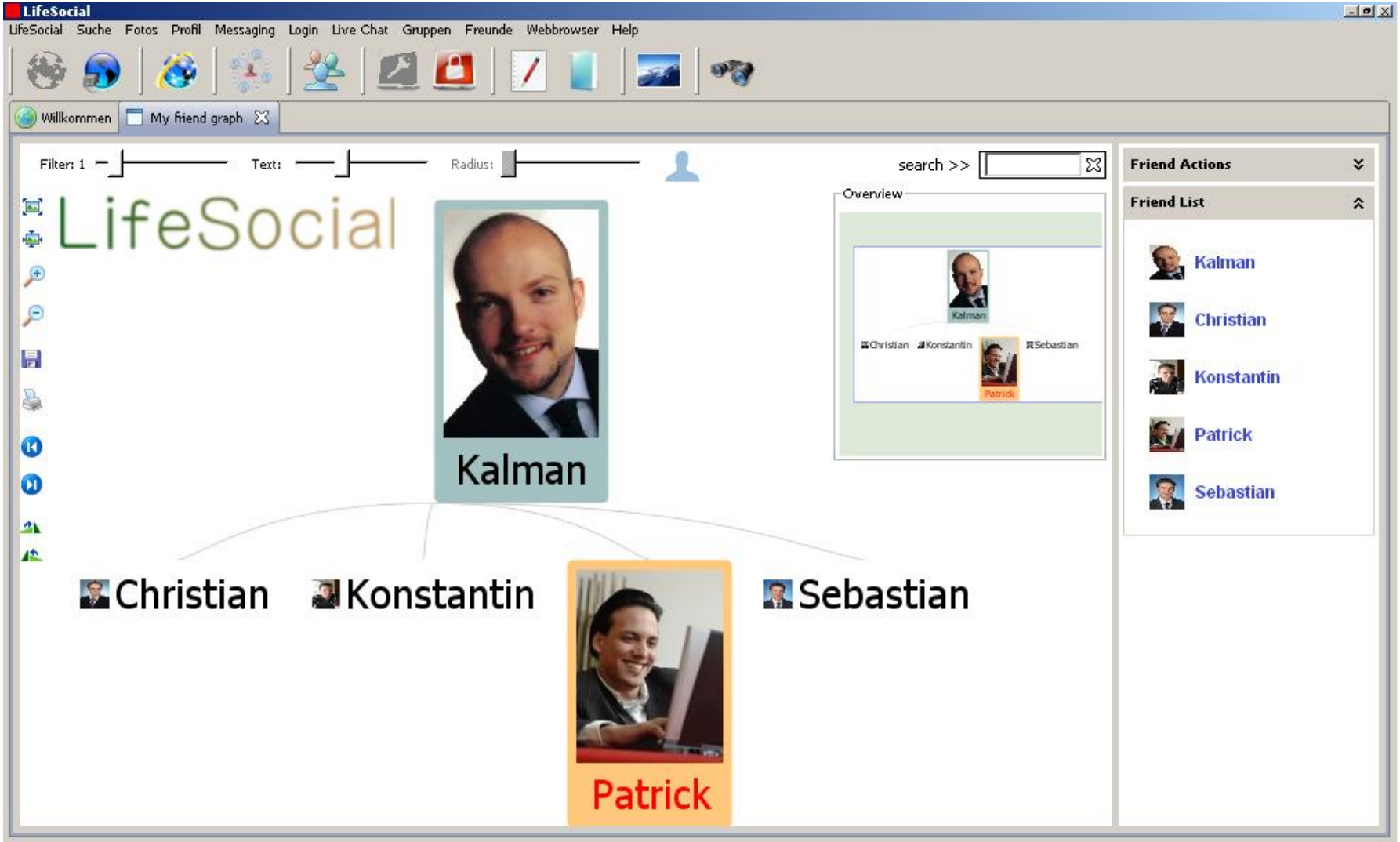


K. Graffi et al., "LifeSocial.KOM: A P2P-based Platform for Secure Online Social Networks", In: IEEE P2P'10

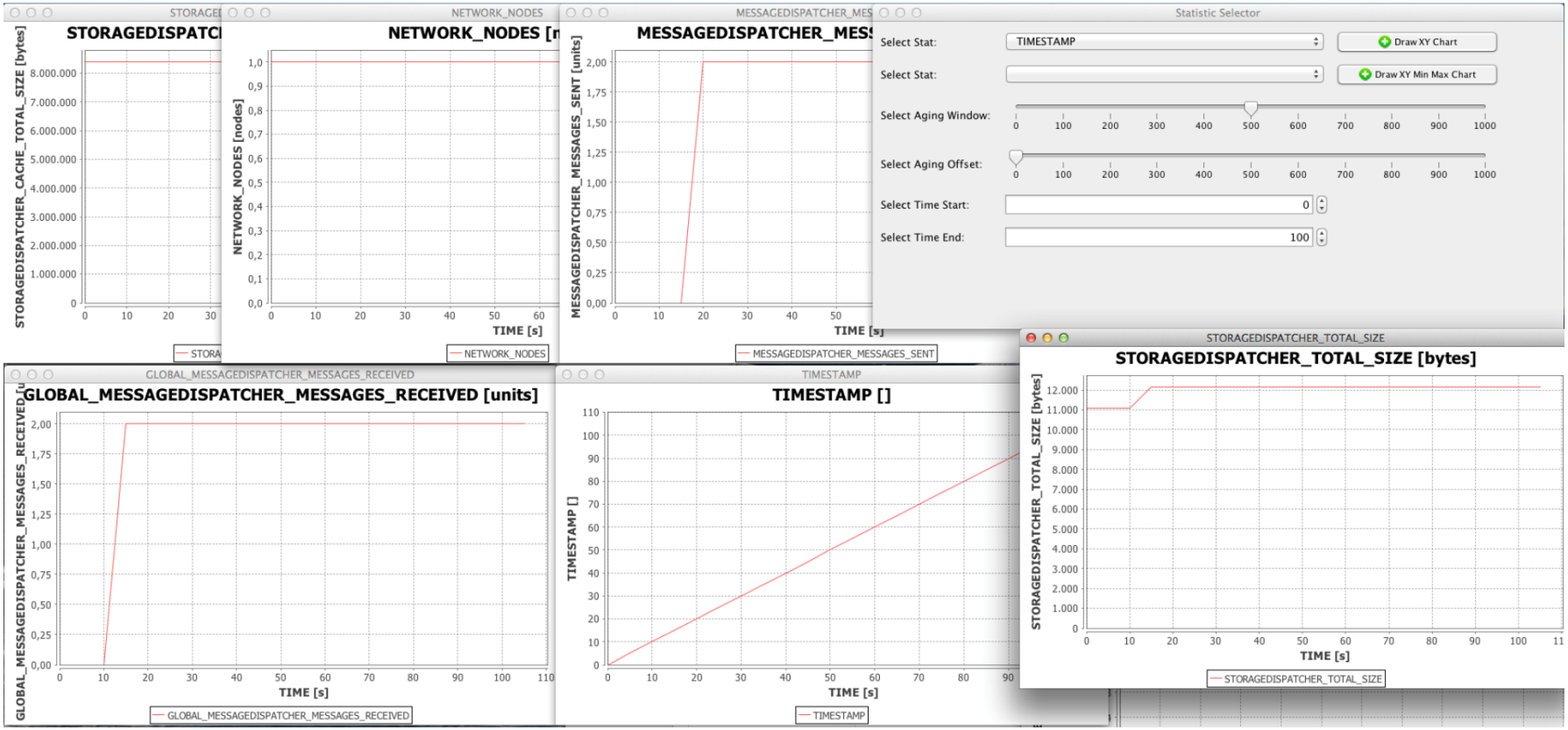
User View


LifeSocial
LifeSocial Suche Fotos Profil Messaging Login Live Chat Gruppen Freunde Webbrowser Help
Willkommen
Profil
Send View
My Photos
Holidays (3)
Kalman
On ...way
On ...way
On ...way
Album Actions
Photo Actions
Add Photos
Edit Photo
Remove Photos
Open Photo
www.lifesocial.org
My friend graph
Foto
My friends
Inbox View
Outbox View
To
Date
Patrick
Sat Feb 28 18:49:19 CET 2009
Patrick
Oli
Ralf
!Messaging.msgTo! Ralf on Wed Se...
Hallo Prof. Steinmetz,
I would like to submit a paper to IEEE P2P 2010,
is that OK?
Regards,
Kalman
OK
Select all
Delete selected
Title: On my way
Privileged Users: Kalman
Comments:
Edit Photo
Remove Photo

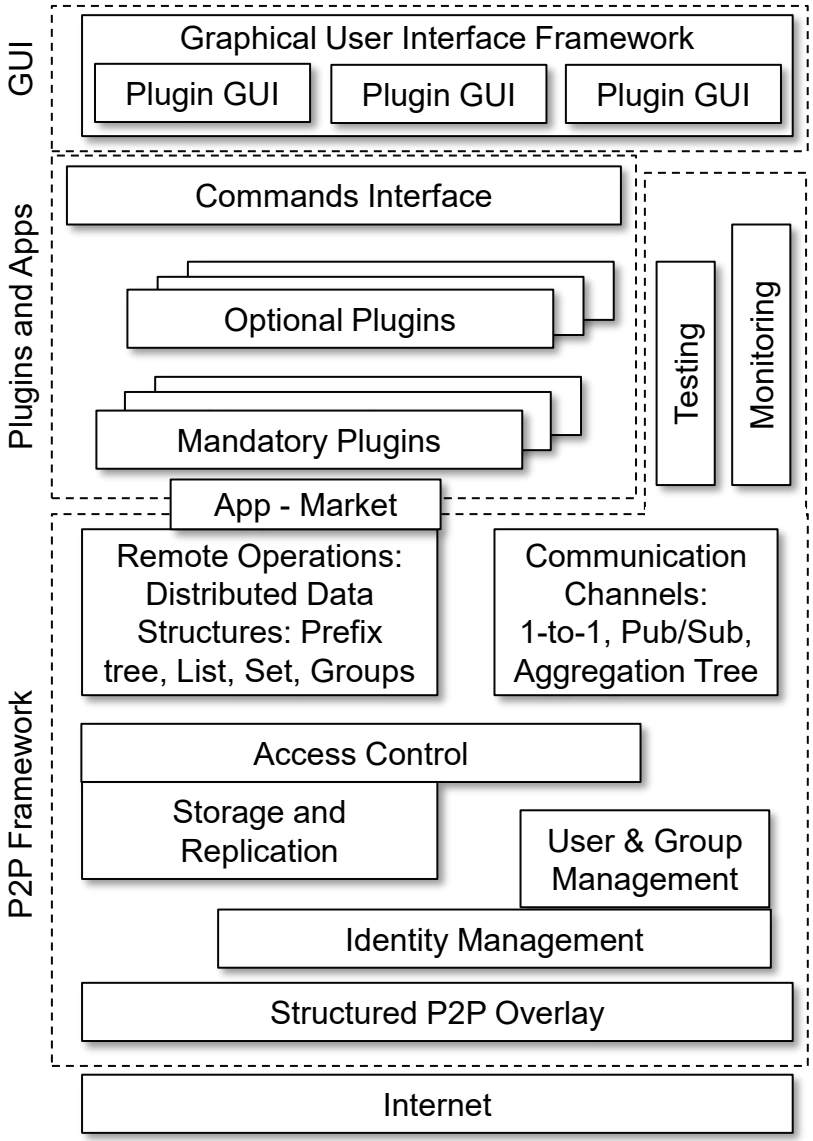
User View


LifeSocial
LifeSocial Suche Fotos Profil Messaging Login Live Chat Gruppen Freunde Webbrowser Help
Willkommen
My friend graph
Filter: 1
Text:
Radius:
search >>
LifeSocial
Overview
Kalman
Christian
Konstantin
Patrick
Sebastian
Friend Actions
Friend List
Kalman
Christian
Konstantin
Patrick
Sebastian

Management View



P2P Framework for Social Networks



P2P Framework - Overlay

Structured p2p overlay

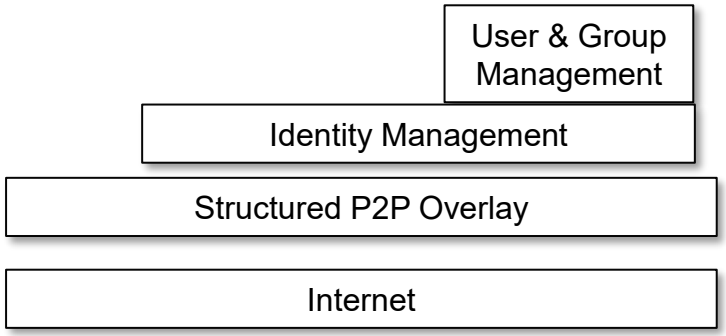
- FreePastry
- For security reasons
  - Buckets as routing entries
  - Redundant routing
  - Iterative routing

Identity management

- To identify users uniquely:
- Create private & public key
- Use public key as node ID
  - Integrates public key infrastructure
  - Enables signing & encryption of communication

User & group management

- Group keys
- Group member management
- Forming of nested groups (subsets) and group key inheritance



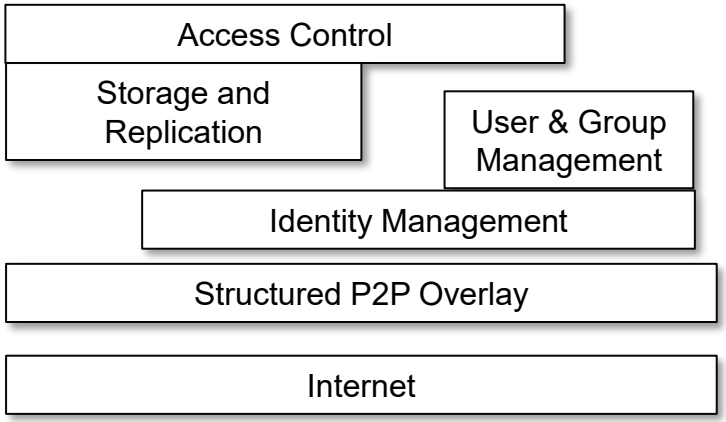
P2P Framework - Storage

PAST

- Storage and replication
- Verification of authorship
- Modified to
  - allow in place updates of data
  - serve data from previous downloaders (load balancing)

Access control

- Sign and encrypt content
- Access control based on
  - User lists
  - Groups



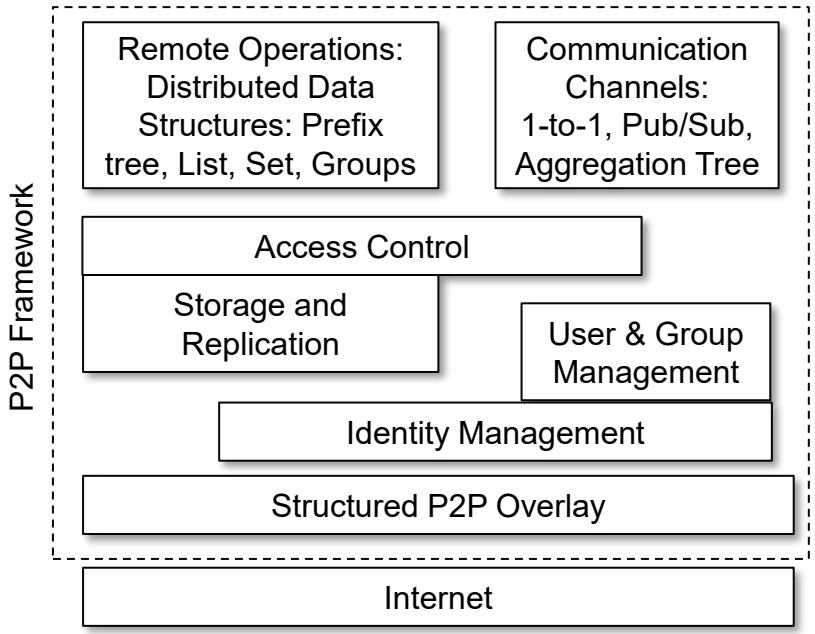
P2P Framework – Communication and Data Structures

Distributed data structures

- Normal approach:
  - Get data, change locally, upload
- Remote operations:
  - Send only information to change
  - E.g. insert “alice” in list
- Supported structures:
  - Set, list, groups, prefix tree

Communication channels

- 1-to-1 (based on ID)
  - E.g. to copy/stream file
- Topic-based pub/sub
  - Using Scribe
- Streaming
  - Using Splitstream
- Multi-criterion indexing and search
  - E.g. Person name, age, location …
- Aggregation tree
  - Sum, min, max, avg
  - For individual topics



P2P Framework - Services

Monitoring

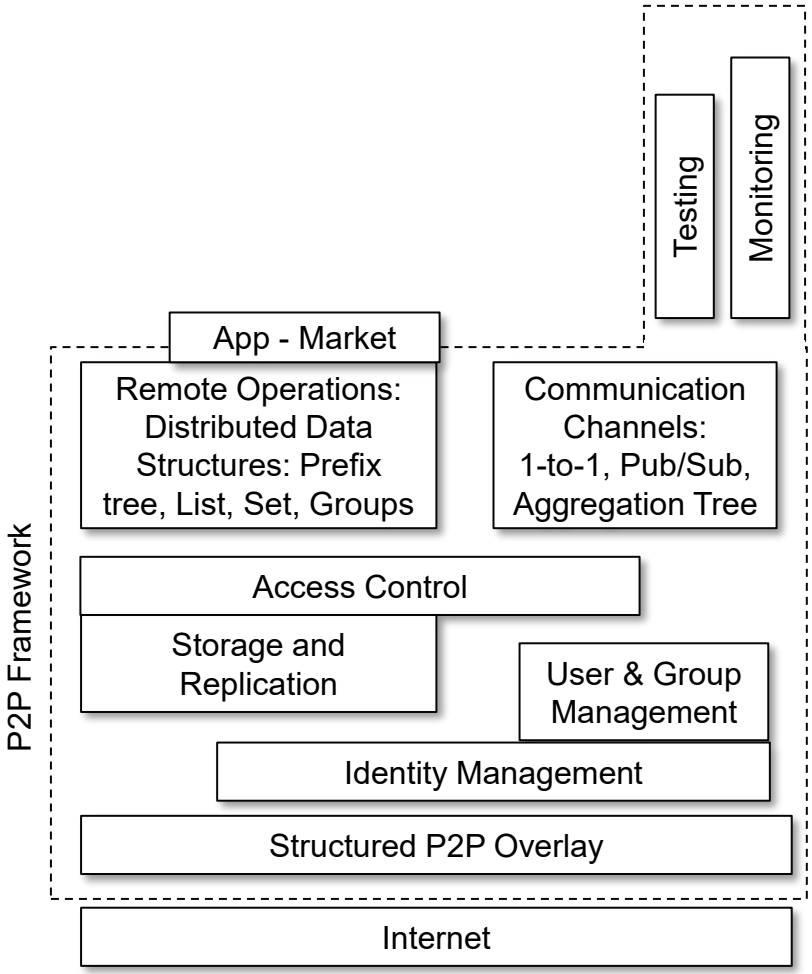
- Captures events from plugins and in framework
- Uses aggregation tree to gather statistics on system

Testing

- Enables to initiate remote commands for testing
- E.g. Initiates in a 50 nodes network defined behavior set (based on measurements)

App – Market

- All Plugins are OSGi – bundles
- Can be loaded, installed at runtime
- Define dependencies
- App market:
  - Host, search and install plugins
  - Determine and get dependencies



LifeSocial: Plugins implementing a Social Network

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Plugins:

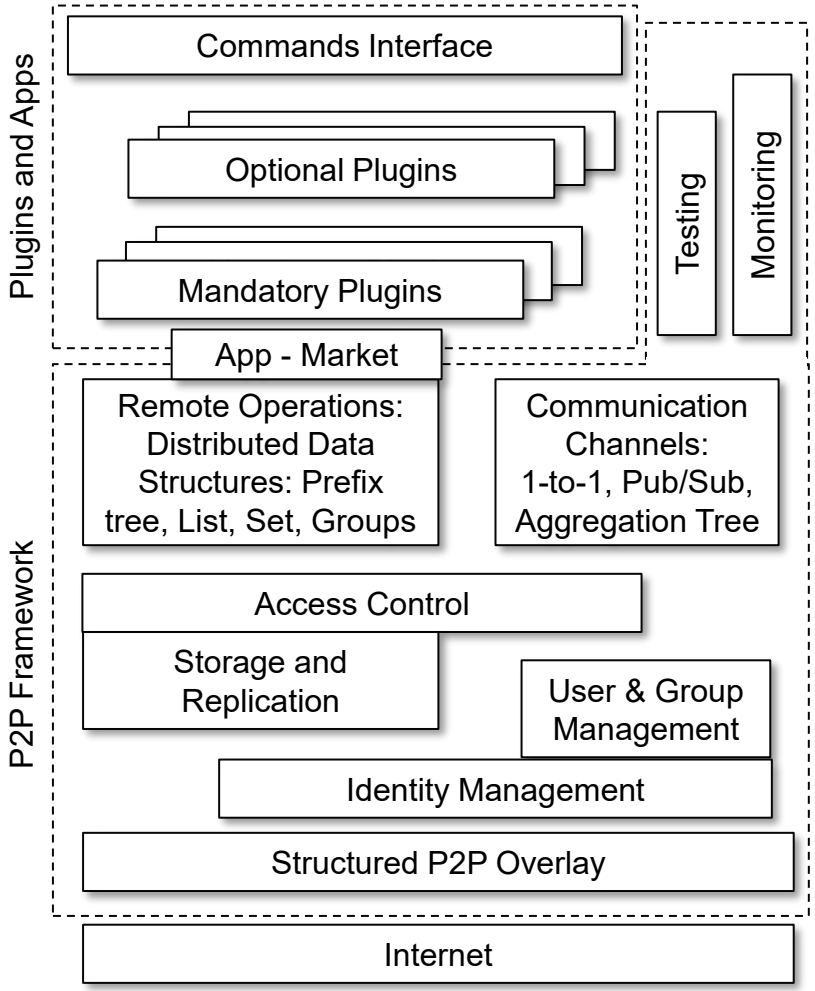
- Using the P2P Framework
- Written in Java, OSGi
  - Open Services Gateway initiative
  - Supports component model (bundle/service)
- Functionality of social networks
- Easy Plugin-to-Plugin communication
  - Over shared storage
    - E.g. photos
  - Over Plugin ID based messaging
    - E.g. Chat-app to chat-app

Mandatory Plugins

- Login, Profile, Friends, Wall, Groups, Photos, Chat, Messaging, Filetransfer (1-to-1), App-Market

Optional Plugins

- Voting, Multi-Chat, One-Click-Filehosting, Forum



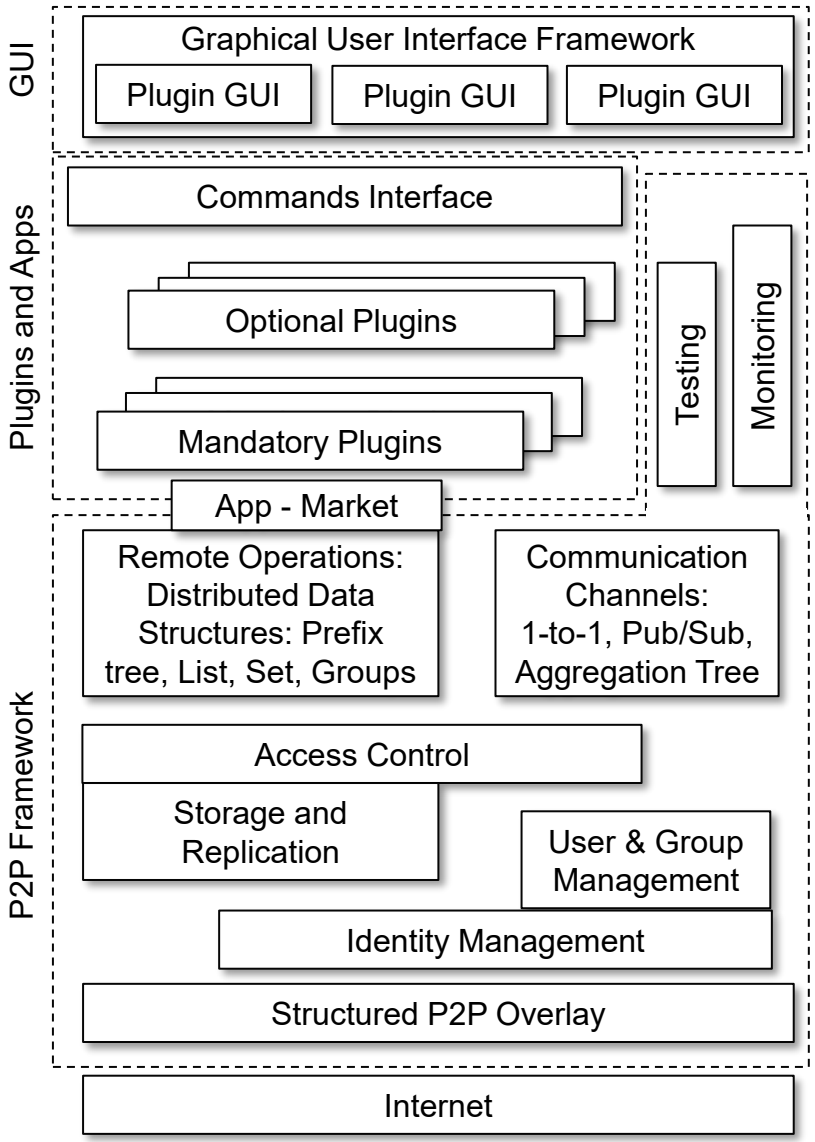
LifeSocial: Graphical User Interface

Graphical User Interface

- GUI Framework able to host individual Plugin Views
- Allows to arrange views
  - Save / load arrangements

Views

- Each Plugin comes with 1+ views
- Some views use several Plugins
- E.g. friends selector
- E.g. message views
  - inbox, outbox, compose



A P2P-Framework for Social Networks

P2P Overlay – Pastry

- Root of Trust
- Modified Routing – Buckets
- Iterative Routing
- Parallel Routing
- Handling Weak Nodes

P2P Framework - Overlay

Structured p2p overlay

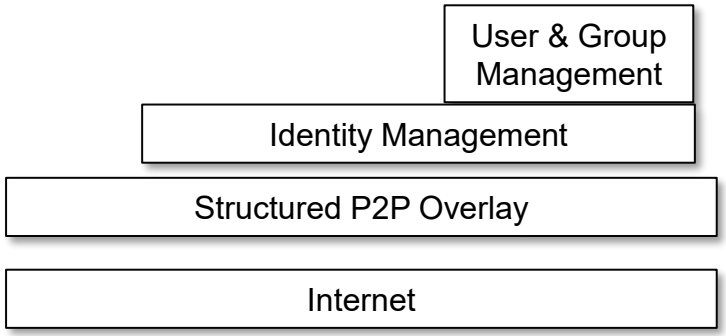
- FreePastry
- For security reasons
  - Buckets as routing entries
  - Redundant routing
  - Iterative routing

Identity management

- To identify users uniquely:
- Create private & public key
- Use public key as node ID
  - Integrates public key infrastructure
  - Enables signing & encryption of communication

User & group management

- Group keys
- Group member management
- Forming of nested groups (subsets) and group key inheritance



Pastry / FreePastry – Introduction

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Pastry: P2P overlay

- Providing DHT functionality and Key-based Routing interface

Two metrics

- ID distance
- Physical distance (“Proximity”)

128-bit-IDs, arranged in a circle

- Variable b defines the size of the routing steps = 2^b; usual value = 4
- Tradeoff between routing table size and maximum number of hops
- Lookup hops scale with $O(log_{2^b}(N))$

FreePastry: prototypical implementation of Pastry

- Current version 2.1: released on 13.3.2009
- Java based, Sun JDK version 1.5.0
- NodeID: 160 bits, 20 byte: 10 hexadecimal number

Custom modification (2012)

- Java package is now part of p2p framework code
- Allows modification

State Information in FreePastry

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

ID Space: [0 , 2^160[

- Randomly assigned while joining
- Base b (2 in example)
- b=4 → hexadecimal in FreePastry

Routing table

- Used for prefix-based routing
- Typical size:
  - log_(2^b) (N) rows
  - 2^b – 1 entries per row
- Row nr. i contains only nodeIDs sharing a prefix of length i with current node

Leaf set

- |L| closest node IDs
- Typical size: L = 2^b or 2x2^b

Neighborhood set

- M entries (typically M = 2x2^b)
- Contains the nodeIDs and IP addresses of locally closest nodes

Leaf Set

greater	10233121	10233122	10233131	10233133
smaller	10233033	10233021	10233001	10233000

Routing table

	0	1	2	3
0	-0-2212102	1	-2-2301203	-3-1203203
1	0	1-1-301233	1-2-230203	1-3-021022
2	10-0-31203	10-1-32102	2	10-3-23302
3	102-0-0230	102-1-1302	102-2-2302	3
4	1023-0-322	1023-1-000	1023-2-121	3
5	10233-0-01	1	10233-2-32	
6	0		102331-2-0	
7			2	

Neighborhood set

	13021022	10200230	11301234	31301235
	02212103	22301201	31203201	33213321

Routing state of node 10233102, base 4

Pastry – Original Routing Procedure

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

In each routing step:

- Prefix-based forwarding:
- A node forwards a message to another node
  - whose ID shares with the target key a prefix
  - that is at least one digit (= b bits) longer
  - than the prefix that is shared with the current node's ID

If no such node is found:

- Numerical distance based forwarding:
- the message is forwarded to a node
  - with the same shared prefix length which is numerically closer

Routing Protocol

Message for key K arrives at node X

- Let X= 10233102, b=2

1. Check if K in scope of Leaf Set
   - E.g. K = 10233030
   - Direct forwarding to 10233033
2. If not (1) use Routing Table
   - Let l:= prefix length of K and X
   - E.g. K = 10320102, l=2
   - Check level 3, prefix 103 → 10-3-23302
3. If not (1) and no routing table entry
   - E.g. K = 10233300
   - Pick closest peer from routing table: 10233-2-32, as closer than 10233102
4. If X is closest to K than any node in Leaf Set (and Routing Table)
   - X is responsible for K, routing ends

Leaf Set

greater	10233121	10233122	10233131	10233133
smaller	10233033	10233021	10233001	10233000

Routing table

	0	1	2	3
0	-0-2212102	1	-2-2301203	-3-1203203
1	0	1-1-301233	1-2-230203	1-3-021022
2	10-0-31203	10-1-32102	2	10-3-23302
3	102-0-0230	102-1-1302	102-2-2302	3
4	1023-0-322	1023-1-000	1023-2-121	3
5	10233-0-01	1	10233-2-32	
6	0		102331-2-0	
7			2	

Neighborhood set

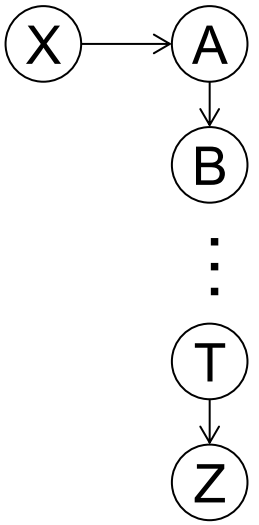
	13021022	10200230	11301234	31301235
	02212103	22301201	31203201	33213321

Routing state of node 10233102, base 2

Pastry – Node join

New node X wants to join

- A is assumed to be physically close to X
- Z is assumed to be responsible for the key “X”



Join protocol

- X asks existing node A to route JOIN message to key X
- JOIN message will be routed to node Z which is closest to key X
- A, Z and all nodes on the route send their state tables to X

X uses following sets as basis for its routing sets

- A’s neighborhood set
- Z’s leaf set

The n-th row of the routing table is copied from the n-th node encountered during the JOIN message routing process

- n = 0: A’s row 0
- n = 1: B’s row 1…

Finally, X sends a copy of its state tables to all nodes contained in them so that those nodes can update their state tables

Modification of Pastry's Routing Table: Buckets

Introduction of Buckets

- Each routing table entry contains k many contacts
- Allows for
  - Parallel routing
  - Node replacement upon node failure

Contact 1
Contact 2
Contact 3
…
Contact k

Coping with node failures

- Nodes leave unexpectedly (fail)
- For detection:
  - Periodic checks of table entries
  - Keep-alive messages
- If node does not answer: failed
  - Failure in Leaf Set:
    - Update entry with leaf set of furthest node
  - Failure in Routing Table:
    - Ask nodes in same row as failed node
    - If all in row failed: as nodes in higher row

Leaf Set

greater	10233121	10233122	10233131	10233133
smaller	10233033	10233021	10233001	10233000

Routing table

	0	1	2	3
0	-0-2212102	1	-2-2301203	-3-1203203
1	0	1-1-301233	1-2-230203	1-3-021022
2	10-0-31203	10-1-32102	2	10-3-23302
3	102-0-0230	102-1-1302	102-2-2302	3
4	1023-0-322	1023-1-000	1023-2-121	3
5	10233-0-01	1	10233-2-32	
6	0		102331-2-0	
7			2	

Neighborhood set

	13021022	10200230	11301234	31301235
	02212103	22301201	31203201	33213321

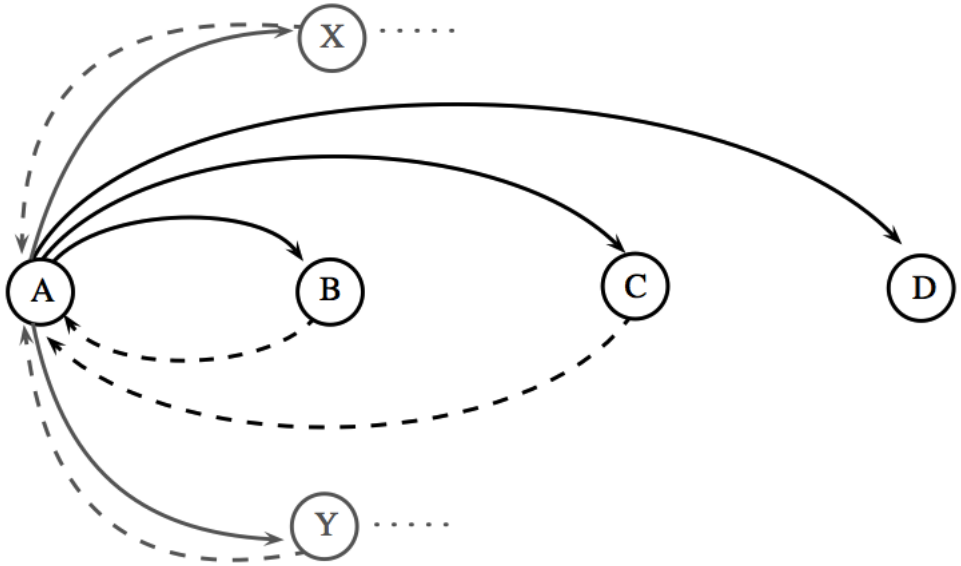
Pastry – Modified Routing Procedure

Prefix-based, parallel and iterative routing

- Loop until no closer node to target ID is revealed
  - Lookup initiator sends out alpha parallel lookups
    - To the alpha nodes sharing the longest prefix in his routing table with target ID
  - Contacted nodes answer alpha contacts closest to the target ID
- Closest found node is responsible

Protocol details

- Alpha answers arrive randomly
  - Process their results only if their sent contacts are better than previous contacts
- Joining protocol requires adaptation



Modification of Pastry's Routing Table: Weak Nodes

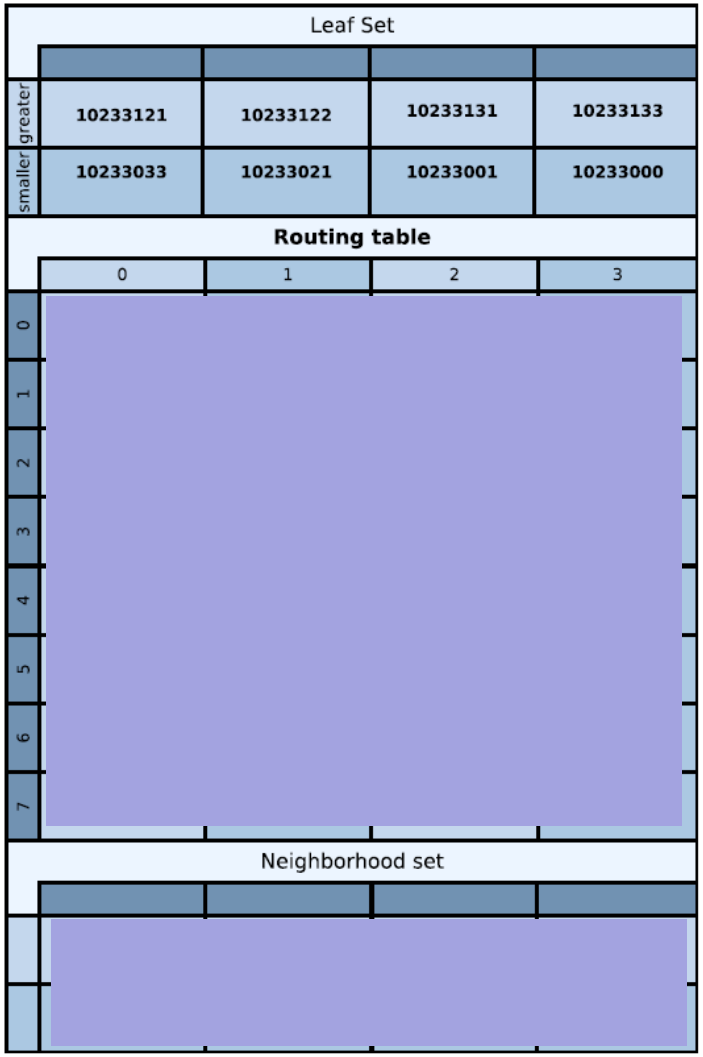
HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Handling of weak nodes

- Weak nodes should not participate in
  - Routing, routing maintenance
  - Storage
- Simply marking weak nodes
  - Odd port: strong node
  - Even port: weak node
  - Port information part of contact info
  - Other marking solution requires additional marking information (+data structures in code) or signaling protocols

Routing to/from weak nodes

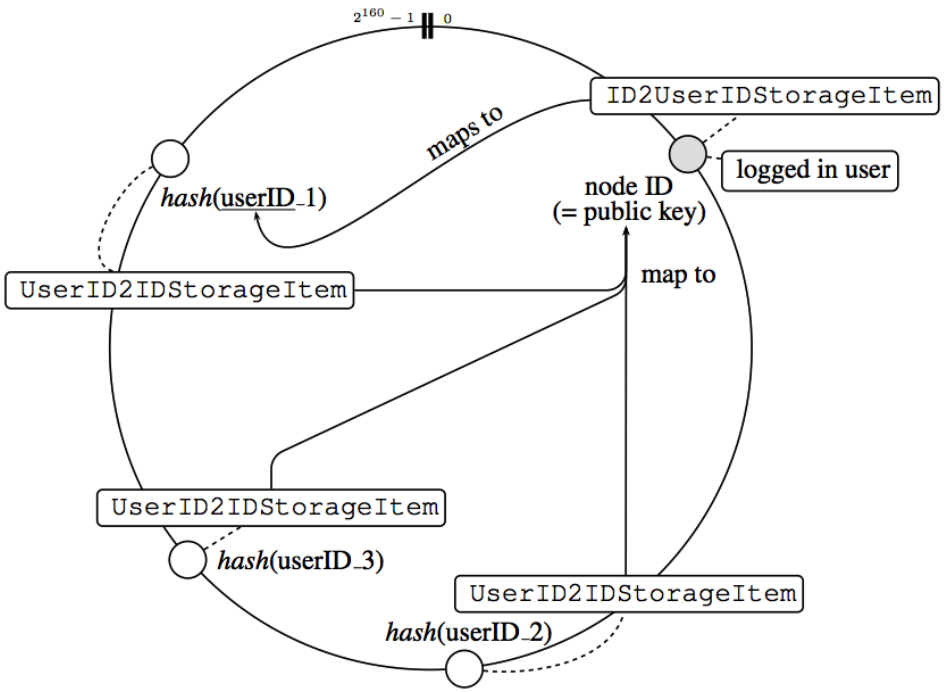
- Weak nodes are inserted only in the leaf set (of close nodes)
- Their routing table also contains only the leaf set entries



Root of Trust: Authenticated Node IDs

Goals

- Users should be authenticated
- Messages confidentially, integer and authenticated sent/received
- Login from any device possible, credentials in the network



Approach

- Immutable UserID
  - Identifying the user throughout the system/application
  - To be used by the plugins: in friendlists, groups …
- Mutable NodeID = PublicKey
  - PublicKey uses 160 bit Elliptic Curve
  - Username and Password are hashed, used as basis for the Private Key
  - Private Key is used for calculation of Public Key
- Mapping from the NodeID to the UserID stored at NodeID
  - At IDs hash(UserID_i) (with i=1,…) store link to NodeID
  - Initial link at new user registration
    - Requires no proof on user
    - Is signed with new NodeID/PubKey
  - Further changes of UserID→NodeID link
    - Are only allowed for previous signer
    - Single Items only to be modified by original author (signature check)

Root of Trust: Effects

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Integrated Publiy Key Infrastructure

- If UserID known, easy to obtain PublicKey
- PublicKey can be stored

Secure communication

- Confidential, integer, authenticated
- Sending messages
  - Message is signed with Public Key (=PubKey) of sender
  - Message is encrypted with PubKey of addressed peer
- Receiving messages
  - Decrypt message using Private Key (=PrivKey) → auth., confidentiality
  - Receiver checks signature of sender → integrity
- For ease:
  - User-User specific symmetric shared key might be exchanged securely before
    - E.g. using Diffie-Hellman

Secure Storage

- Items are signed when stored, can only be changed by original author

Properties of Pastry / FreePastry

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Advantages

Well documented, clear APIs

- Modular, extendable software
- Large user base, still maintained

Basic functionality

- Routing, DHT (key-value mapping)
- Distributed storage

Disadvantages → solved

Now: support for heterogeneity

- All nodes are treated equally
- Strong, long-living peers should do more
- See at storage part

Now: built-in security mechanisms

- Identity attacks, routing attacks
- Sensitive to malicious nodes

Limited API → P2P Framework

- “Only DHT”
- Also requires sufficient replication, additional services

A P2P-Framework for Social Networks

Storage and Replication

- Replication
- Access Control
- Load-balancing

P2P Framework - Storage

PAST

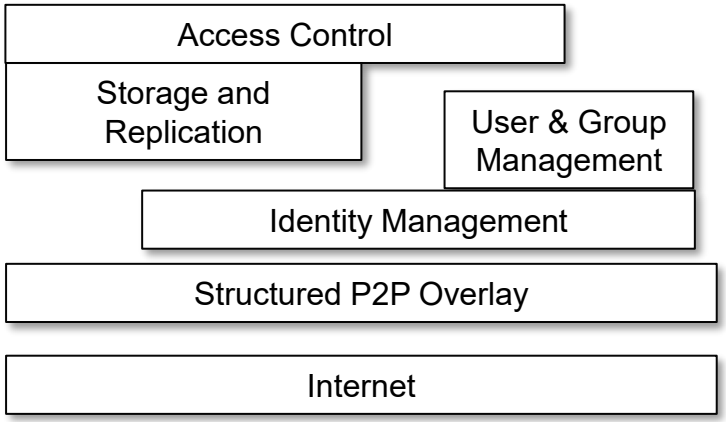
- Storage and replication
- Verification of authorship
- Modified to
  - allow in place updates of data
  - serve data from previous downloaders (load balancing)

Access control

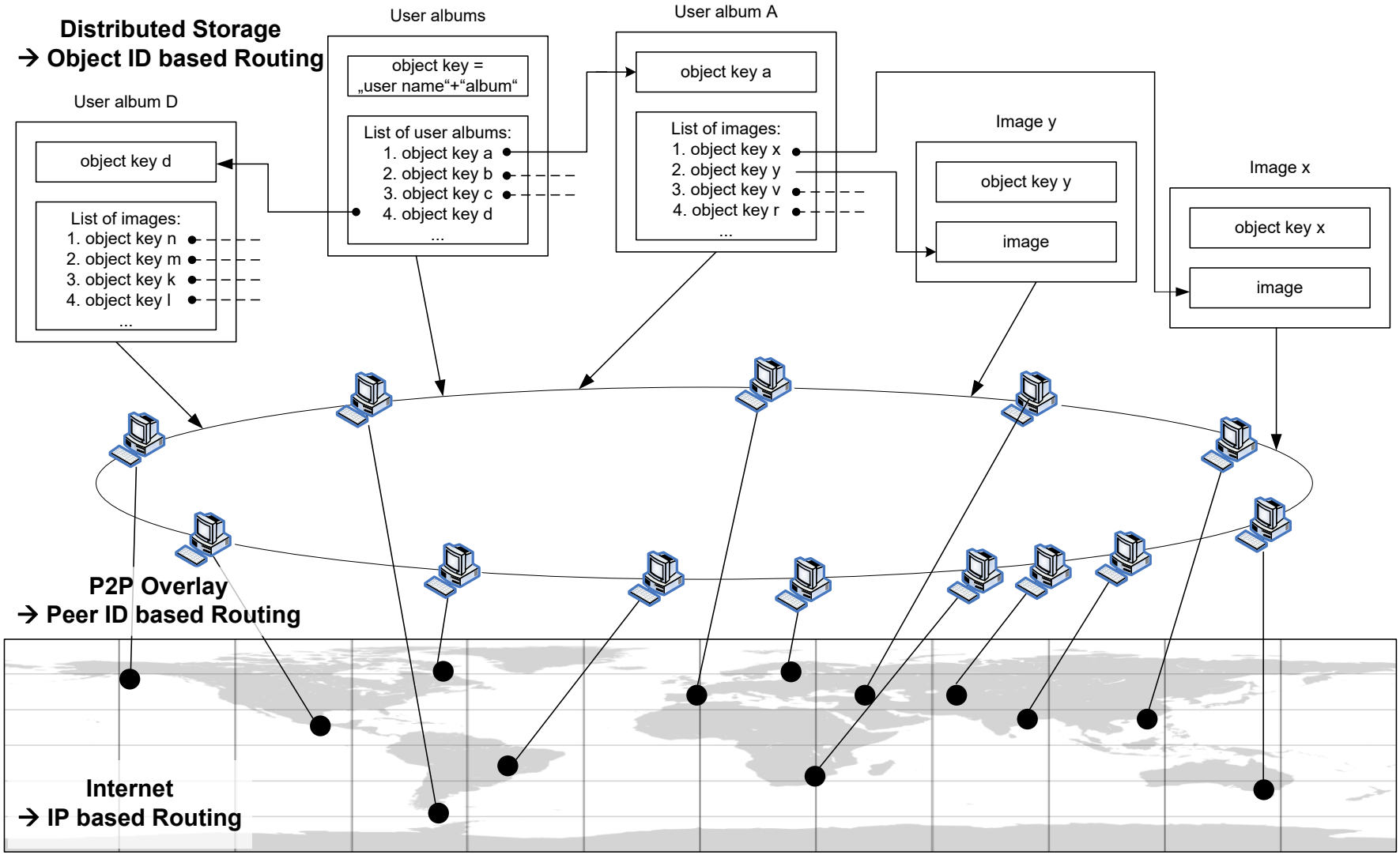
- Sign and encrypt content
- Access control based on
  - User lists
  - Groups

Keys

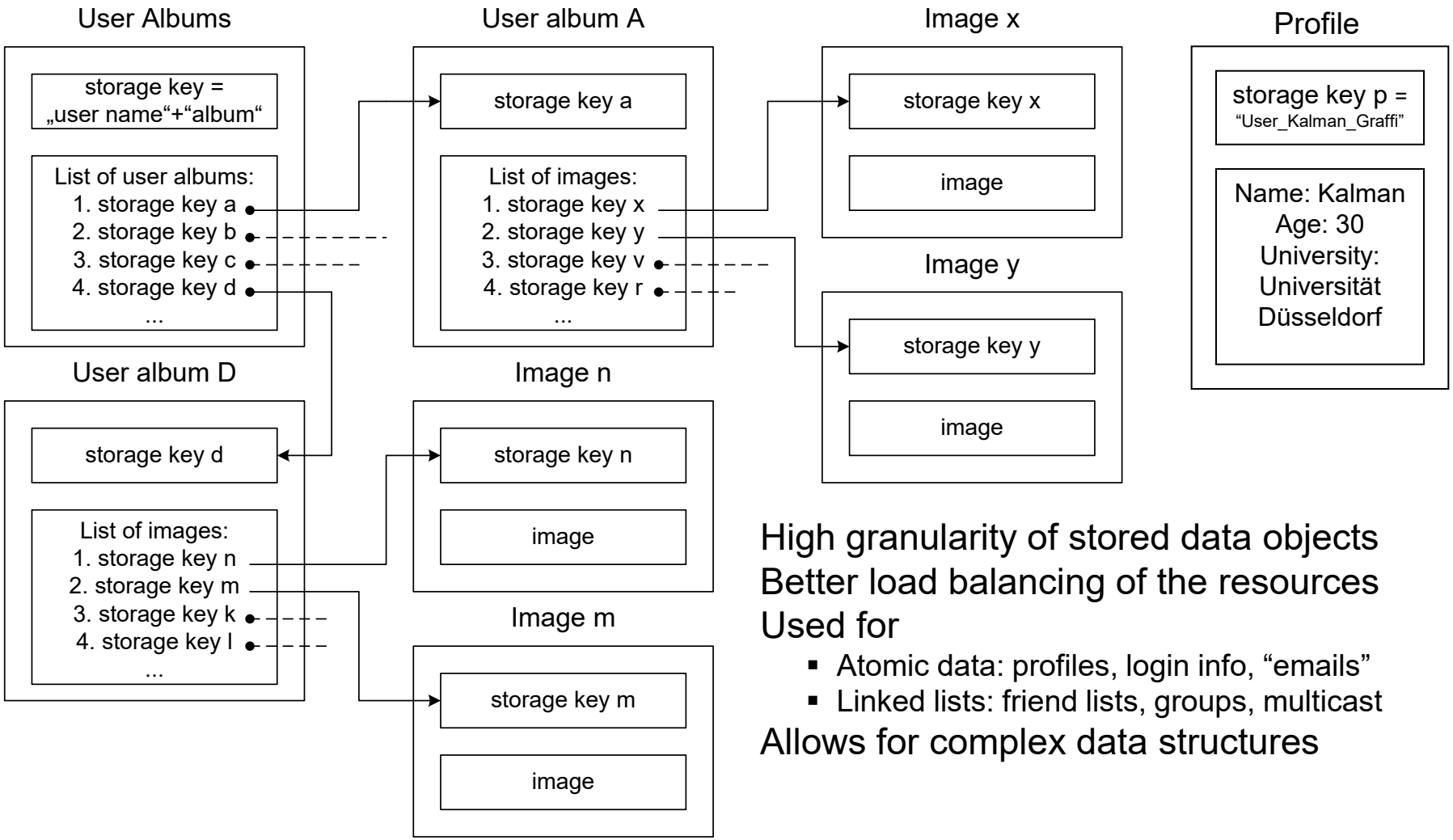
- 160 bit elliptic curve keys
- 128 bit AES symmetric keys



Positioning in the Network



Document Types, Obvious Storage Keys



High granularity of stored data objects

Better load balancing of the resources

Used for

- Atomic data: profiles, login info, “emails”
- Linked lists: friend lists, groups, multicast

Allows for complex data structures

Atomic Single Item Operations

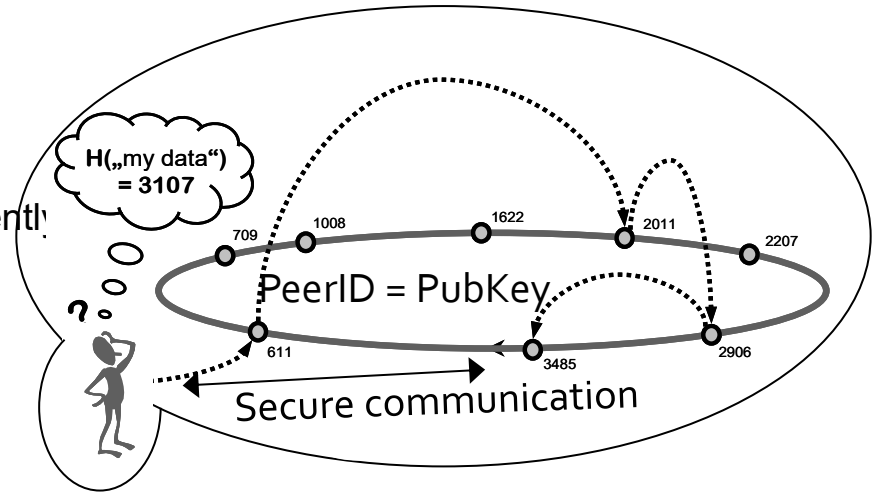
Basis for security must be
- Easy to deploy, fast and reliable

Requirements for access control and item security
- All sensible information is stored in DHT (no trust assumed)
- Integrity / authentication
  - Author is authenticated through signatures
  - Signature allow to detect modifications
  - Only one author for one data item
  - Content integrity can be checked, false authorship can be rejected
- Confidentiality: only authorized users should be able to read content
  - The storing node is typically not allowed
  - Data items can/should be encrypted

Data object must be atomic
- Must contain all relevant security information
- Replication
  - Multiple storage of the same content
  - Find 1 copy to read, all copies to write consistently
  - Replication independent of security

Support for inplace modification

Support for group-based rights



Replication Extension of Pastry: PAST

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Idea

- Store objects also on k further nodes in leaf set
- Remember Key-based Routing (KBR):
  - nodehandle [] replicaSet (key → k, int → max rank)
    - Returns an ordered set of peers of magnitude (max rank) on which replicas of the object with key k can be stored
    - The nodes which become roots for the key k when the local node fails
  - update(nodehandle →n, bool →joined)
    - Upcall: informs that node n has either joined or left the local neighbor set
- If responsible peer fails
  - New responsible node (is in replicaSet)
    - Is informed of peer leave by update upcall
    - Answers to lookups for new objects it is responsible for
    - Calls new replicaSet and deployes replicas (if needed)

PAST Evaluation

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Good

- ID related replication: 1 lookup sufficient to find object
- Replication ratio flexible (might depend on object / peer properties)
- Failed replica nodes are detected by overlay: easy to react

Drawback

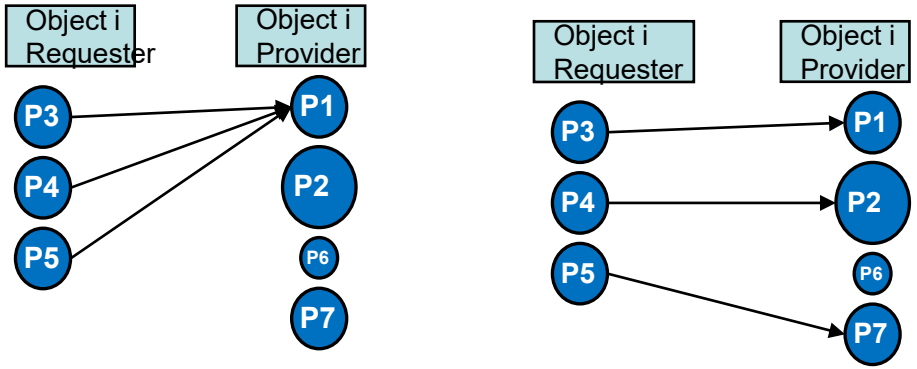
- Replication not peer heterogeneity aware
  - Weak nodes might be overloaded by replication task
  - See next slide how to handle that in the P2P Framework
- Security
  - Replicas all in one ID area: easier to attack
  - Still open

Modification: Heterogeneity Aware Storage

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

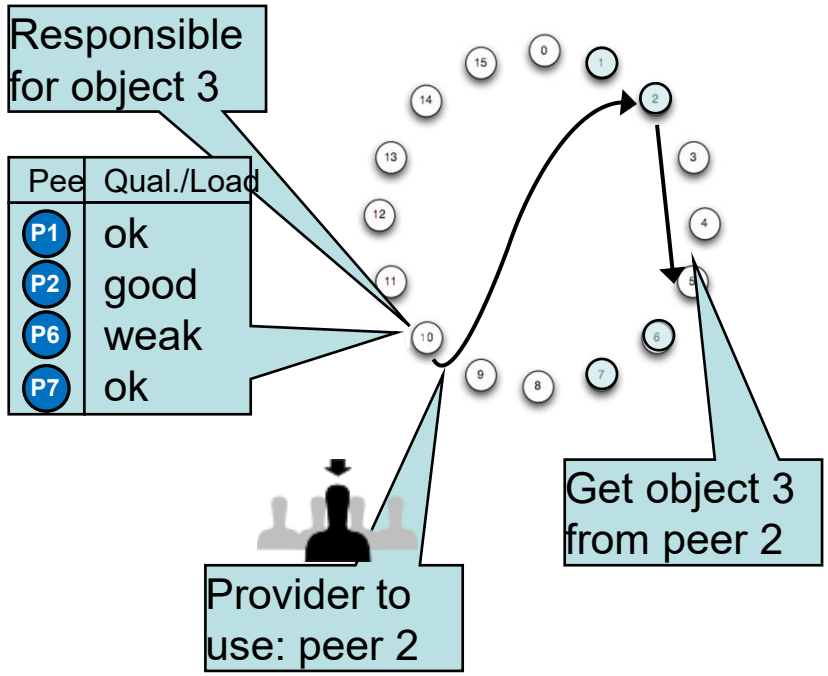
Best strategy to match nodes having and wanting a file?

- Considering:
  - Memory, storage and bandwidth
- Leading to
  - Balanced load regarding the load provision in the system



Approach

- Responsible node remembers downloading nodes (providers)
  - Maintains a provider list
  - Periodically checks their bandwidth
    - Through message size and transmission time
  - Forwards download query (1 hop) to them based on load
  - High capacity → high probability to be chosen



Discussion: Heterogeneity Aware Storage

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Heterogeneity aware storage

- In addition to replication
- Local solution
  - If responsible node wants to share load, he may, but does not have to
  - Source of file irrelevant for requester
    - As files are atomically authenticated

Periodic capacity checks

- Required some overhead
- Are propagated back, so that nodes are aware of their capacities
- Can be used for further purpose

Evaluation: Heterogeneity Aware Storage

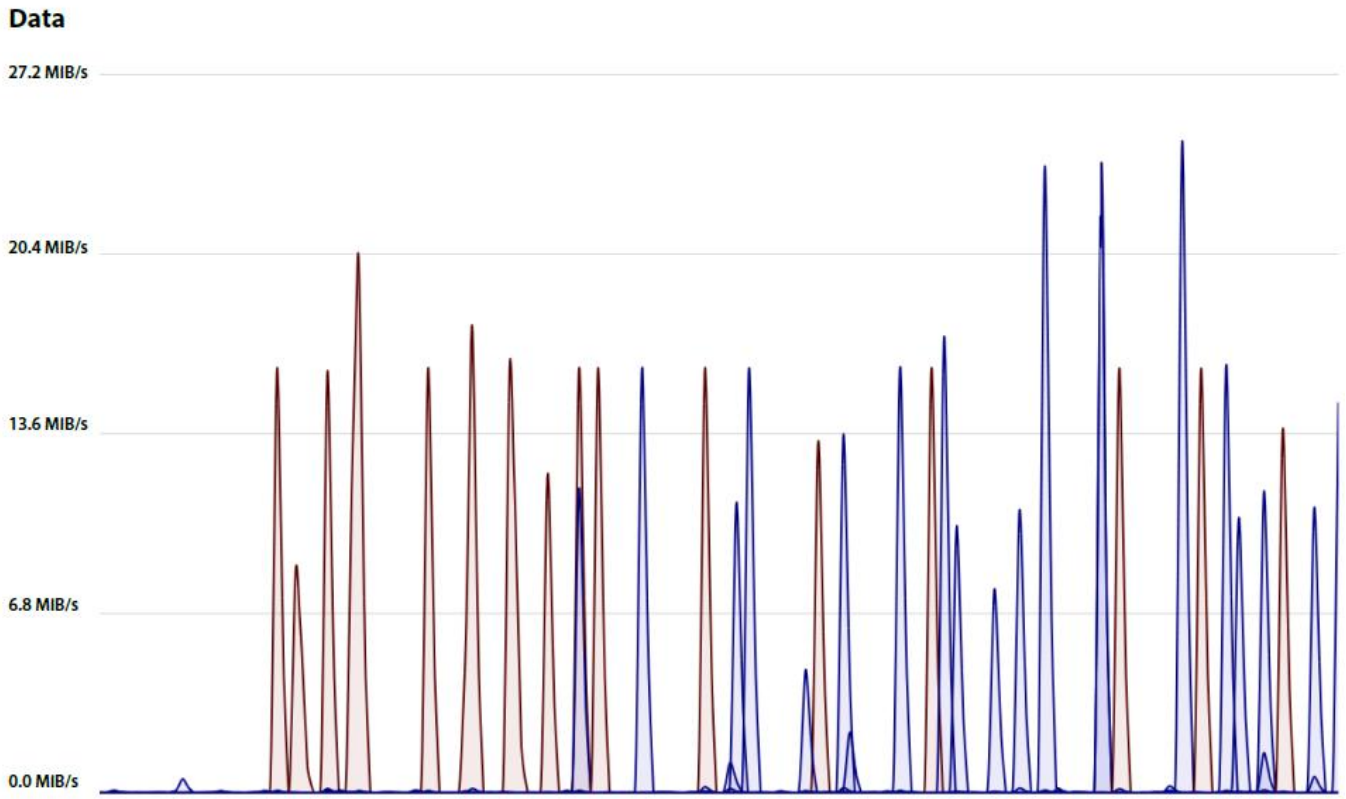
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Setup:

- 10 Machines, 3x TestCaseLoadPhoto in 12 secodns

Evaluation:

- Red: responsible node, blue: a chosen provider



Simple Distributed Access Control for Single Objects

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

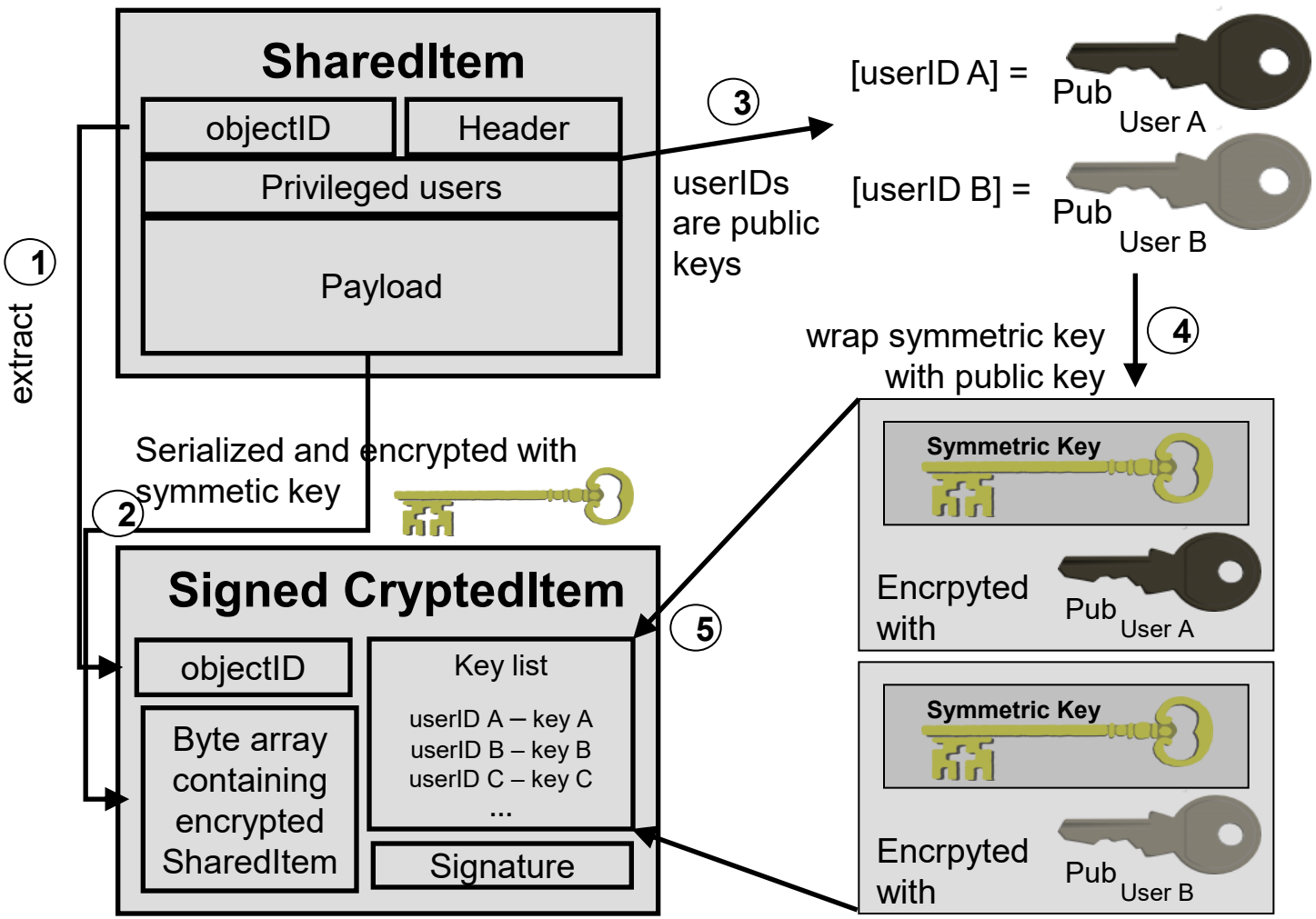
Goals:

- Authentification of hosts (original author)
  - Allows „Write“ access control
- „Read“ access control on individual data items

Idea:

- Remember established public key infrastructure
- Write access control
  - Initial write is free, data item is signed, authors Public Key is added
  - Overwrite on existing Object ID only allowed to original author
  - Protection against replay attacks:
    - Random nonce is stored with item; nonce must also be signed with new item
  - Storing nodes verifies signature
  - ToDo: Replication nodes must also verify signature and verify correct behavior of storing node
- Read access control:
  - Encrypt all stored data with unique symmetric key: Key_S
  - Encrypt the symmetric key for all privileged reader: (Key_S)enc(PubKey_i)
  - Attach the encrypted symmetric key to the encrypted data

Example of Distributed Access Control

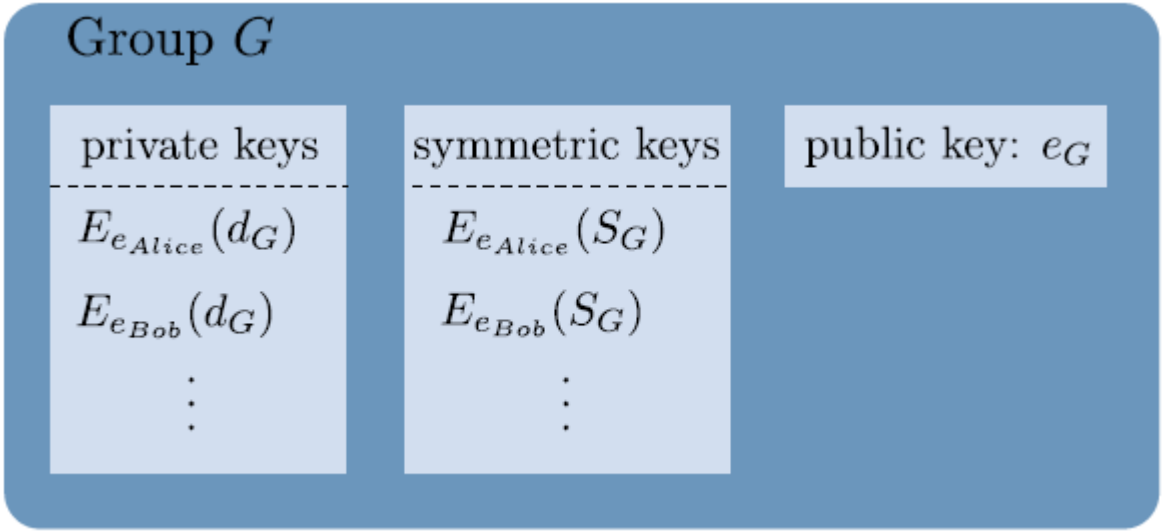


Group-based Access Control

- Groups have own keys → Treated as single user
  - Single data objects encrypted/signed with group keys
  - Only group members can read / write
- Group key is stored in single item by group creator
  - Read access for group members

Group provides for its members

- A symmetric key S_G and an asymmetric key pair (Public: e_G, Private: d_G)



Group-based Access Control

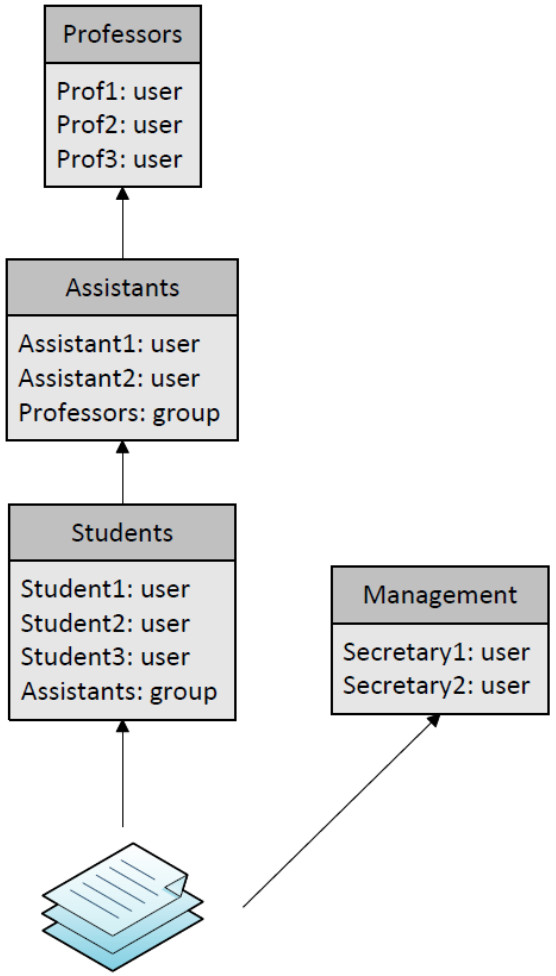
Challenge: Hierarchical structures of groups

- Share content with combination of groups:
  - G1 OR G2: Encrypt with 2 keys, provide both
  - G1 AND G2: Encrypt data with first key, encrypt result with second key
- Hierarchical groups: G1 subgroup of G2

Groups may contain

- User (keys)
- Group (keys)

Encrypted item has a reference to the group used for encryption



A P2P-Framework for Social Networks

Peer-to-Peer Framework

- Storage
  - Distributed Data Structures
    - Sets, Lists, Prefix-Hash Trees
  - Remote Operations
  - Access Control for Distributed Data Structures
- Information Cache and Callbacks
- Communication Channels
  - 1-to-1, Multicast, Publish / Subscribe
  - Search
- App-Store

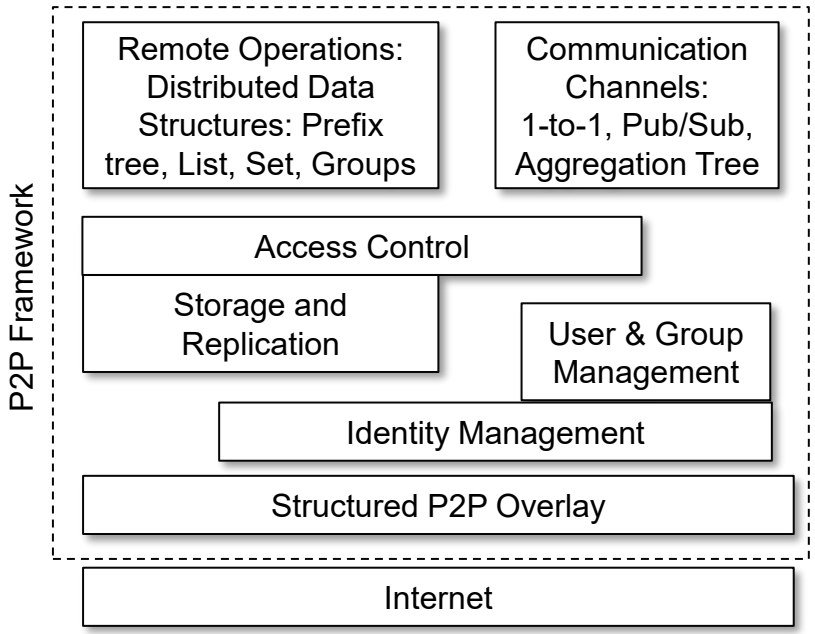
P2P Framework – Communication and Data Structures

Distributed data structures

- Normal approach:
  - Get data, change locally, upload
- Remote operations:
  - Send only information to change
  - E.g. insert “alice” in list
- Supported structures:
  - Set, list, groups, prefix tree

Communication channels

- 1-to-1 (based on ID)
  - E.g. to copy/stream file
- Topic-based pub/sub
  - Using Scribe
- Streaming
  - Using Splitstream
- Multi-criterion indexing and search
  - E.g. Person name, age, location …
- Aggregation tree
  - Sum, min, max, avg
  - For individual topics



Distributed Data Structures (DDS)

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Motivation:

- Put / Get of single data items does not match social network
- Social network data structures:
  - (Unorderd) Set: Friends, Group members, …
  - (Ordered) List: Wall entries, Email posts, Forum entries
  - Prefix Hash Trees: App categories, User categories …

Distributed Data Structures

- Concept: Single data items with
  - Characteristic meta data
  - Payload
  - Pointers to next elements in the „distributed“ data structure

On right: functions of the list

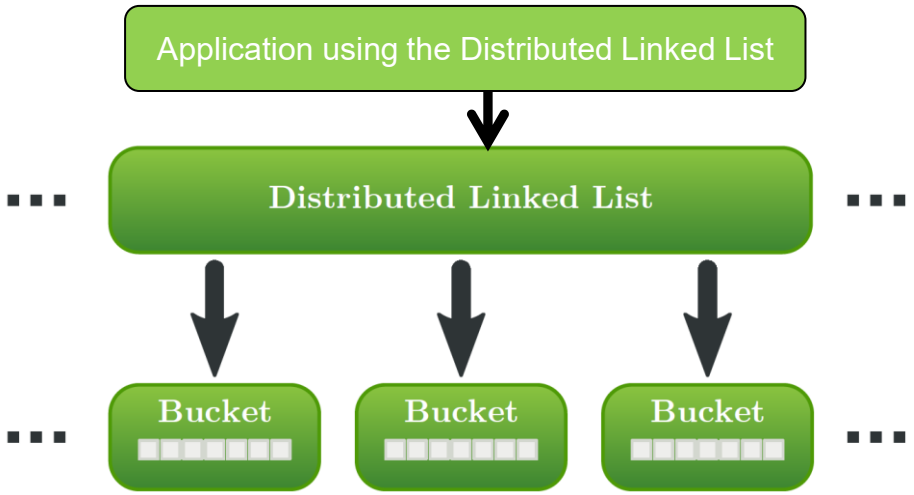
- Location of i-th element
  - Key hash(„listname“ + i/bucketsize)

Operation	Request Method
`add` (*item*)	single message
`addAll` (*items*)	parallel messages
`get` (*i*)	single message
`contains` (*item*)	parallel messages
`set` (*i*)	single message
`indexOf` (*item*)	parallel messages
`lastIndexOf` (*item*)	parallel messages
`remove` (*i*)	single message
`remove` (*item*)	parallel messages
`removeAll` (*items*)	parallel messages
`isEmpty` ()	single message
`size` ()	single message

Basics of Distributed Data Structures

Interfaces

- Common access to a DDS instance for application
- DDS in framework split in several buckets
- Buckets handled individually

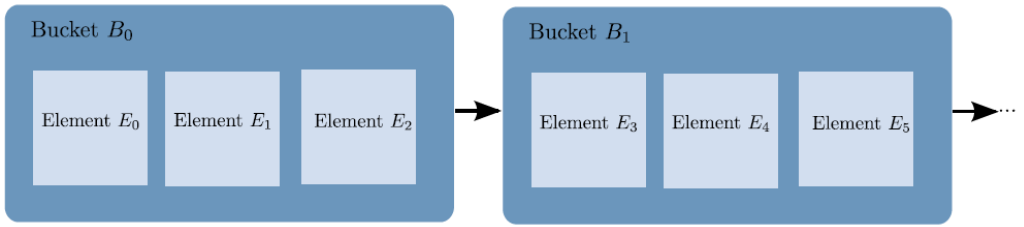


Requirements:

- Only owner allowed to edit/ delete a list entry
- Everyone allowed to add a list entry
- Support groups: only group users allowed to read or add a list entry

Buckets contain:

- Individual entries
  - May belong to different users
  - Number of entries per bucket configurable
- Pointer to next bucket



Distributed Data Structures: Write and Read Access

Add entry to the list

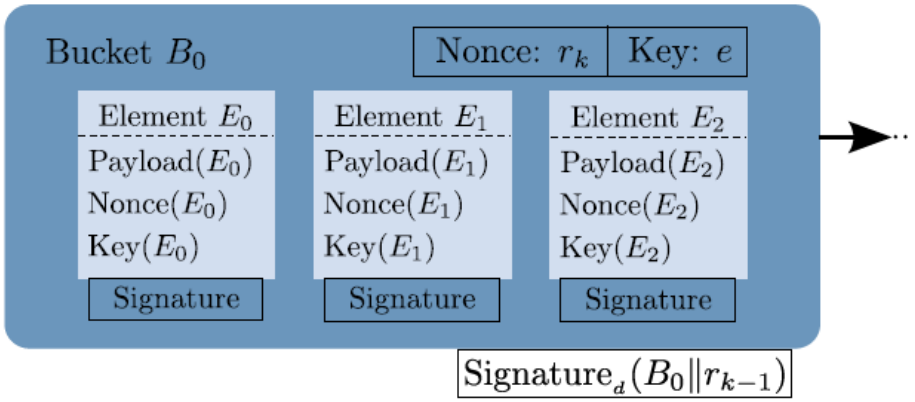
- Free to write:
  - Sign element with private key of the owner
- Write only for group members
  - Sign element with private key of the owner
  - Encrypt each element of the bucket with symmetric group key
  - Sign the bucket with private group key

Read entry

- Find and retrieve entry
- Verify entry and bucket signature

Edit an existing element

- Create and sign new entry
- Sign bucket (all entries of it)
- Storing node recognizes list buckets and verifies signatures of bucket and elements



Distributed Data Structures: Deletion of Entries

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Deleting an existing element

- Deletion = overwrite with empty “Delete”-Entry (deletedItem)
- Create and sign new “Delete”-entry
- Sign bucket (all entries of it)
- Storing node recognizes list buckets and verifies signatures of bucket and elements

Problem

- deletedItems stay in list
- With time: list might contain almost only deleted Items

Solution:

- Algorithm consolidates two buckets
  - deletedItems shifted one bucket to the right
- Called after each write operation if list contains deletedItem
  - Only users with write rights can sign consolidated buckets

Consolidate Protocol

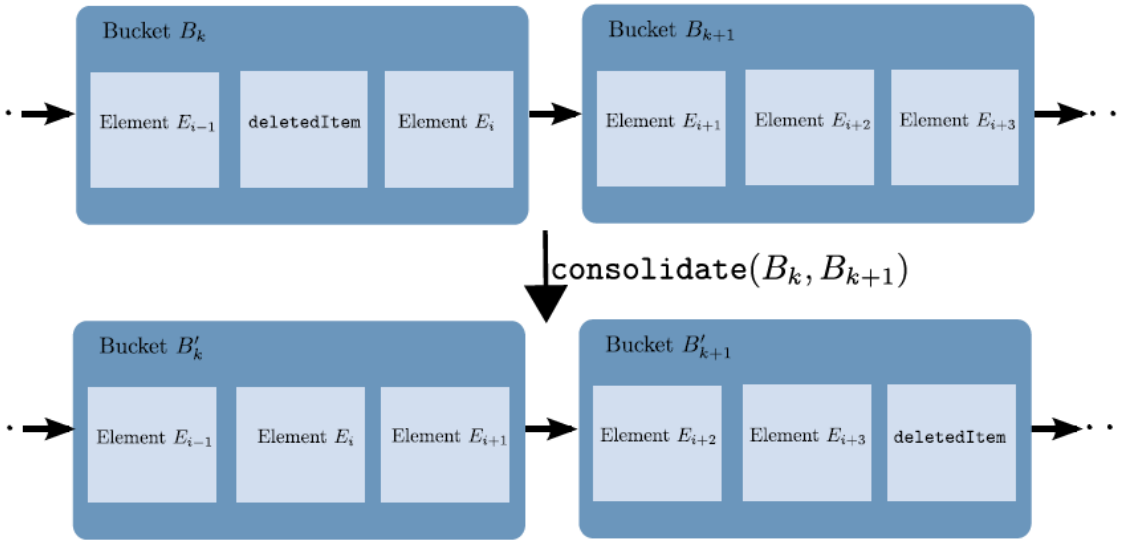
Before

- BK contains a deletedItem
- Consolidate(BK,BK+1) is called

After

- BK → BK‘, BK+1 → BK+1‘
- DeletedItem now on most right position in BK+1‘
- All other elements shifted to left by one position

1) The node initiating the consolidation computes the buckets B'_k and B'_{k+1} and its corresponding signatures $Sig(B'_k)$ and $Sig(B'_{k+1})$. Both signatures are parallely sent to nodes S_k and S_{k+1} which are storing B_k and B_{k+1}, respectively.
2) S_k and S_{k+1} compute B'_k and B'_{k+1} themselves and afterwards verify the corresponding signatures received beforehand. If a node encounters an invalid signature, it cancels its consolidation process.
3) If the signatures are valid, S_{k+1} notifies S_k.
4) If S_k receives the notification of S_{k+1}, it stores B'_k in the network. Afterwards it notifies S_{k+1}.
5) If S_{k+1} receives the notification of S_k, it stores B'_{k+1} after checking that B'_k is stored in the network as expected.



Remote Operations

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

DistributedList 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 needs to be considered
- Storing node performs changes and stores the modified item
- But: storing node is usually not owner of the item
  - Cannot compute valid signature
- User sends delta information, how to sign whole new data item?

Protocol for Remote Operation with Write Access

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Node C wants to modify an object stored at S:

- If C does not have the current object
  - It requests the current version of the file that should be changed from S.
- C has the current object
  - If C already has a version of the file, it sends a hash of it to S.
- S compares the received hash with the hash of its current object version
  - If the hashes differ: S replies with the current version
  - If the hashes match: it acknowledges briefly.
- C performs its changes locally and afterwards computes the signature of the modified file and the hash of the file that should be overwritten
- It sends a request containing the desired changes, the signature and the hash of the file on which C locally performed the changes to S.
- S checks whether its current version and the version on which C has performed its changes are the same by comparing the hashes
  - If they are not equal, step 3 is applied.
  - If the hashes are equal: S performs the desired changes and then inserts the file into the network using the signature received from C.

Evaluation of the Remote Operation and Deletion Consolidation

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Setup: 1000 operations

- Setting A presented

Traffic savings:

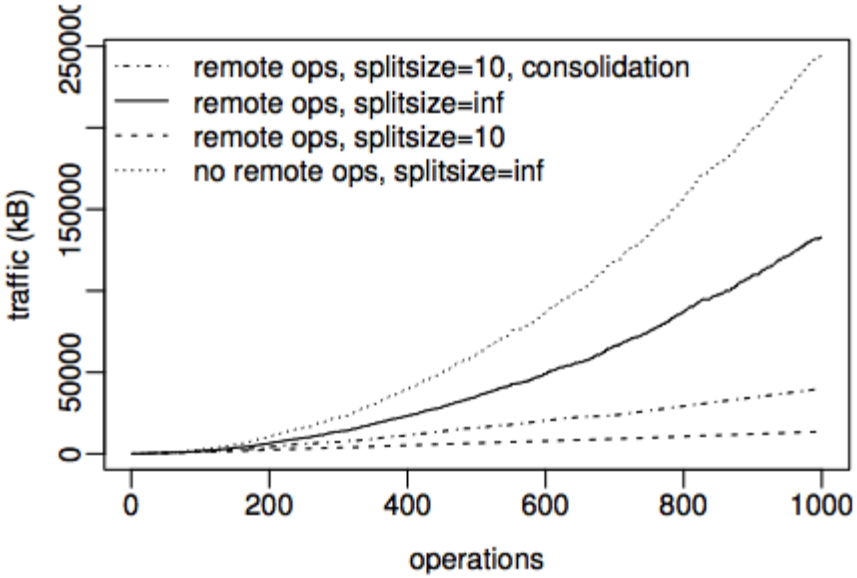
- Through remote operations and buckets with 10 entries
- Consolidation further saves traffic

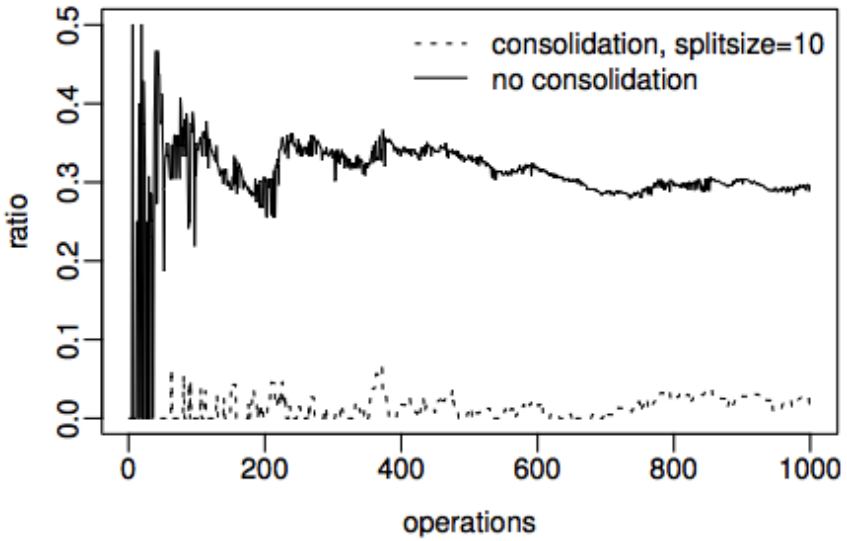
Action distribution:

Operation	Distribution Setting A	Distribution Setting B
`get` (i)	$1/2$	0.93
`add` $(item)$	$3/8$	0.053
`remove` (i)	$1/8$	0.017

Consolidation

- Reduces number of deletedItems in list drastically

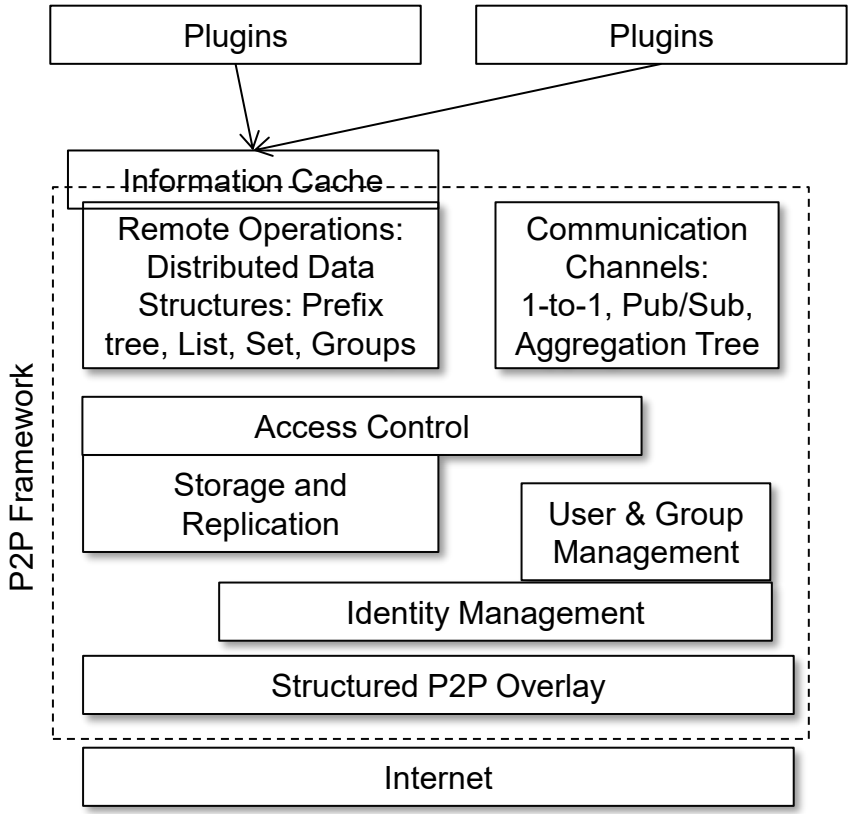




Accessing the Data: Information Cache and Callbacks

Pull approach: Information Cache

- Interface: getData(key)
- Immediate response:
  - Data
    - Lifetime x min, then new lookup
  - Pending
    - While pending: data is retrieved
  - Not available
    - Negative lookup
- Allows data reuse by several plugins – information container

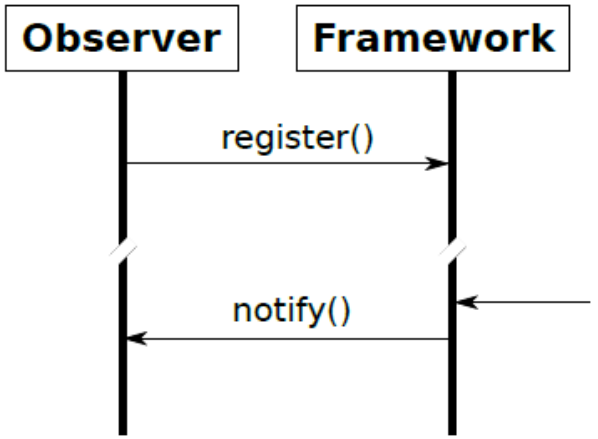


Accessing the Data: Information Cache and Callbacks

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Push Approach: Callbacks / Continuations

- Plugins register for data(key)
- If lookup for it is successfull
  - Data is deliverd to waiting plugin



Framework supports different situations:

- Continuation - Asynchronous operations
  - e.g. storing and continue after it
- Receiver - Register for receiving content
  - e.g. message receiving functions
- Listener - Listen to ongoing operations
- e.g. status reports for file transfers

Communication Structures

1-to-1 (based on ID)

- E.g. to copy/stream file
- For files also 1-click-hosting
  - Store encrypted (Key K) file
  - Under specific ID
  - Retrieve by (ID+Key) (→"link")

Scribe: topic-based pub/sub

- Using Scribe
- Create channel, e.g. for hash("p2phhu")
- Send invitation via 1-to-n to selected users → they join p2phhu channel
- Messages sent to channel p2phhu are received by subscribed users

1-to-n

- Send message 1-to-1 to n users

Multi-criterion indexing and search

- File has metadata: attributes
  - Attributes: E.g. Person name, age, location …
- Attributes have values
- Search for file with attr_i = x_i

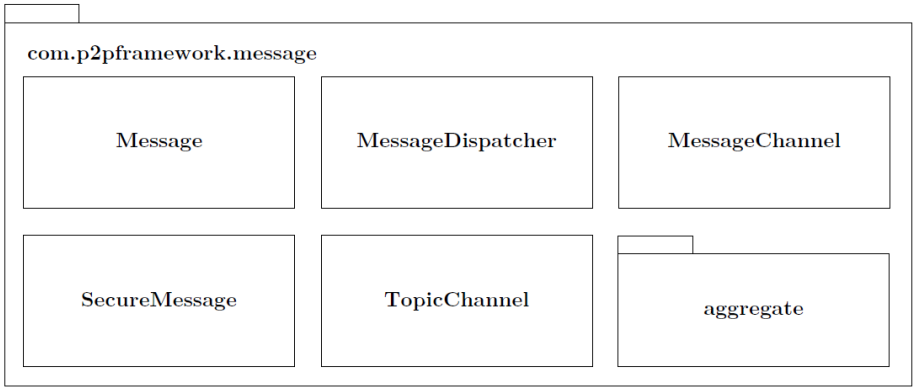
Aggregation tree

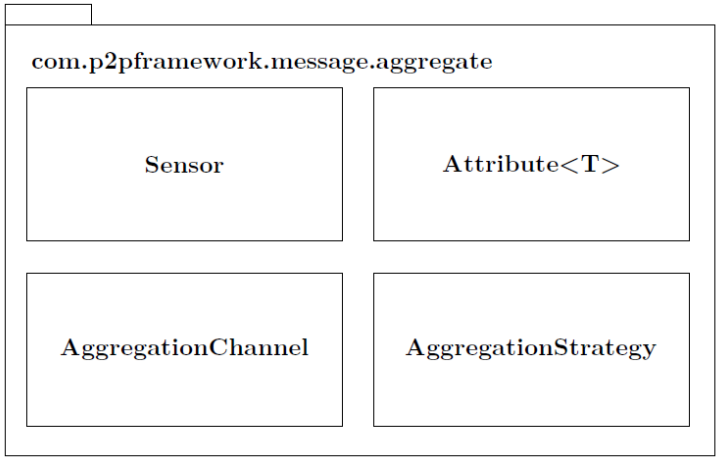
- Sum, min, max, avg, std dev
- For individual topics →
  - One channel per metric

Main Communication Classes

Communication Classes

- 1-1, 1-N: MessageChannel
  - Identified by unique name
  - Both comm. partners need to create this channel
  - 1-to-N: Sender defines list of receivers
- N-to-m: TopicChannel
  - Identified by unique name
  - Participants subscribe at unique name
  - All messages sent to list are forwarded to subscribers
- Aggregate
  - Attribute<T> holds for a single measurement:
    - name, unit, aggregation strategy and the sensor it was measured
  - Aggregation strategies
    - Sum, count, average, min, max…
  - Statistics on all nodes in AggChannel





Multi-Criterion Search

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Query Language:

- Keyword: describes a document through a string
- Attribute/Value: attribute attached with numeric value

Query:

- collection of keywords or attribute/value pairs
- (Keyword | (Attribute = Value))*

Distributed Query Engine:

Indexing

- Takes any object and a collection of keywords
- Afterwards the object can be retrieved throughout the p2p network

Querying

- Takes a query (collection of keywords)
- Returns the collection of objects, where the query-keywords subset of index-keywords of the objects

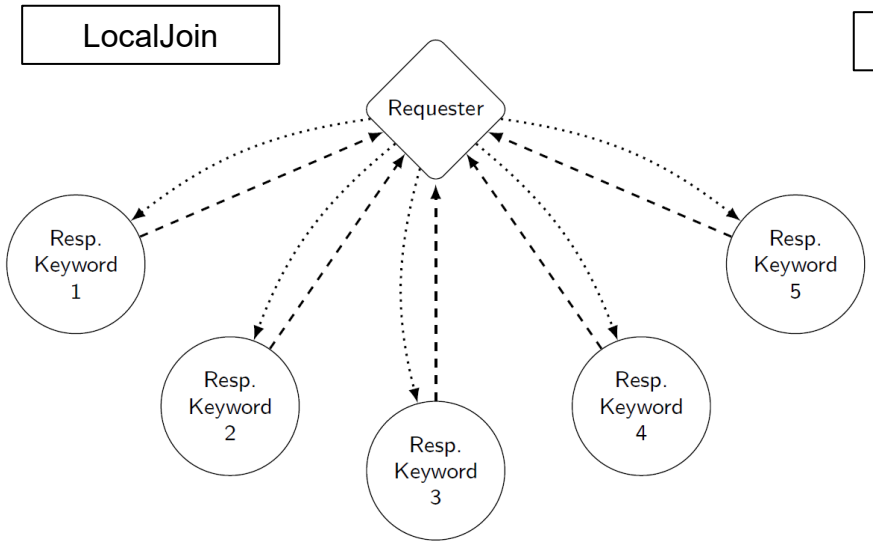
```
void indexItemWithTags ( Serializable
identifier,Collection <String > keywords );

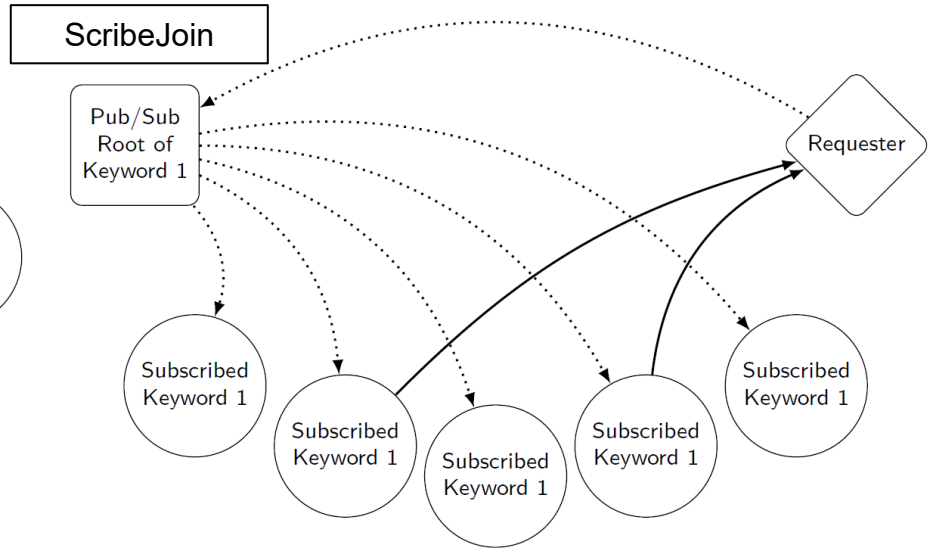
void removeItemIndexWithTags ( Serializable
identifier, Collection <String > keywords );

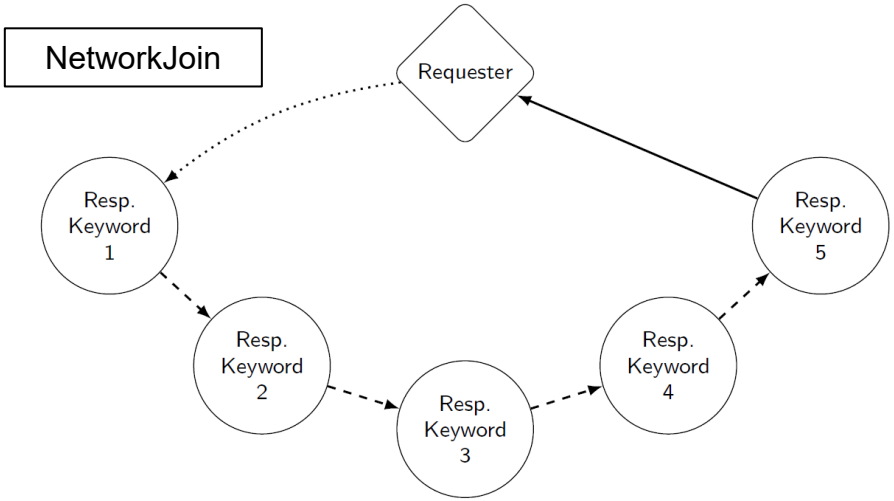
Collection <Serializable>
searchAllItemsWithTags
(Collection <String > keywords );

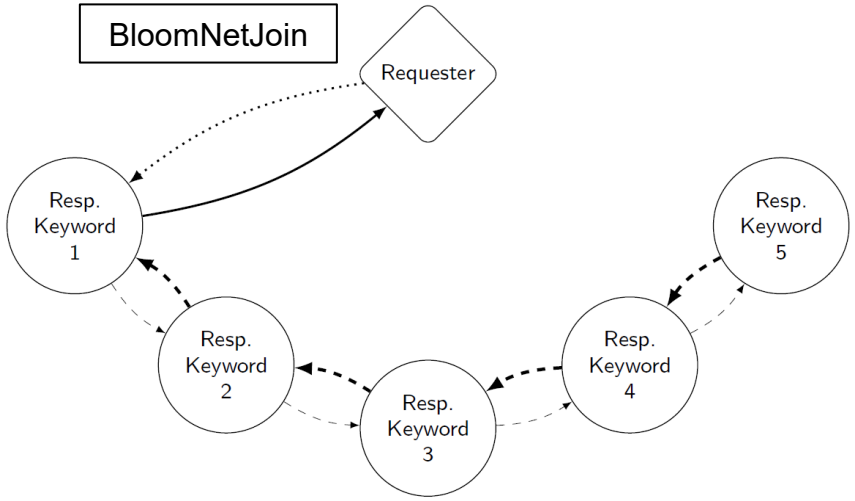
Serializable searchItemWithTags (Collection
<String > keywords);
```

Implemented Search Engines, Common Indexing







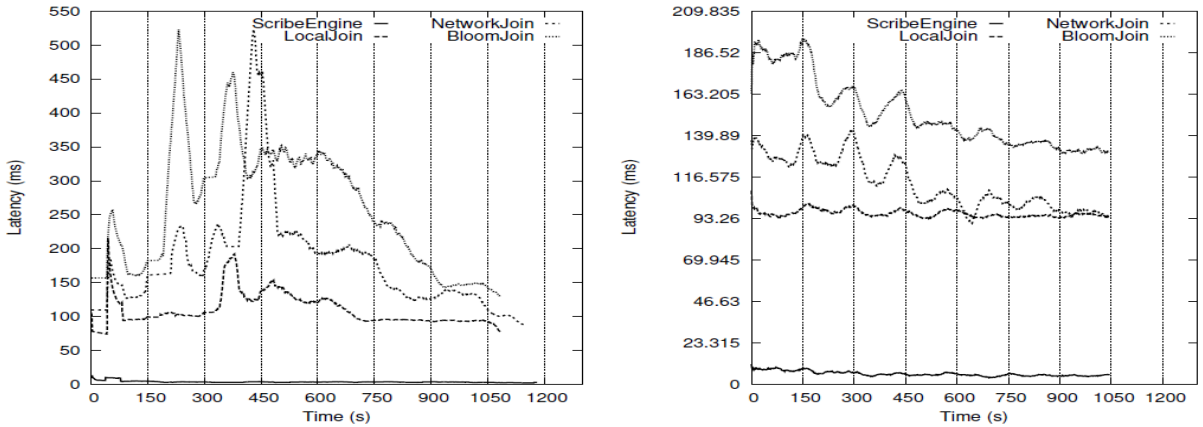


Testing the Query Engines

Setup:	Test	NoChurn
Nodes:	10	10
Nodes per PC:	1	1
Documents:	1000	1000
Unique Keywords:	1000	1000
All Keywords:	20000	20000
Query Scenario:	1st	1st
Queries/Scenario:	10000	10000
Churn:	**5** %	**0** %

Observation:

- Churn disrupts system performance (even at 5%)
- ScribeEngine has best query latency
  - but long reconnect reconnect and indexing latencies
- BloomJoin does not signicantly reduce network costs
- ScribeEngine handles queries with many keywords good
- The others handle queries with very popular keywords good



(a) Churn (b) No Churn

Figure: Churn: Mean Query Latency

A P2P-Framework for Social Networks

Peer-to-Peer Framework - Continued

- Monitoring
  - Tree-based Monitoring
- Testing
  - Test-Plugin

P2P Framework - Services

Testing

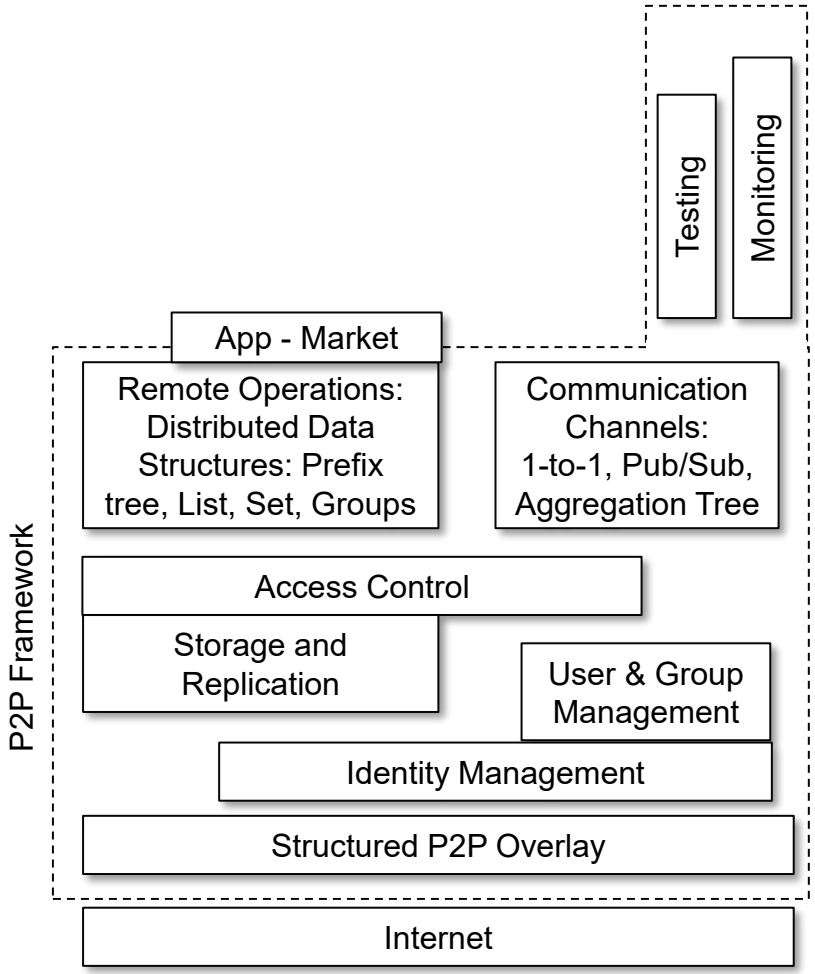
- Enables to initiate remote commands for testing
- E.g. Initiates in a 50 nodes network defined behavior set (based on measurements)

Monitoring

- Captures events from plugins and in framework
- Uses aggregation tree to gather statistics on system

App – Market

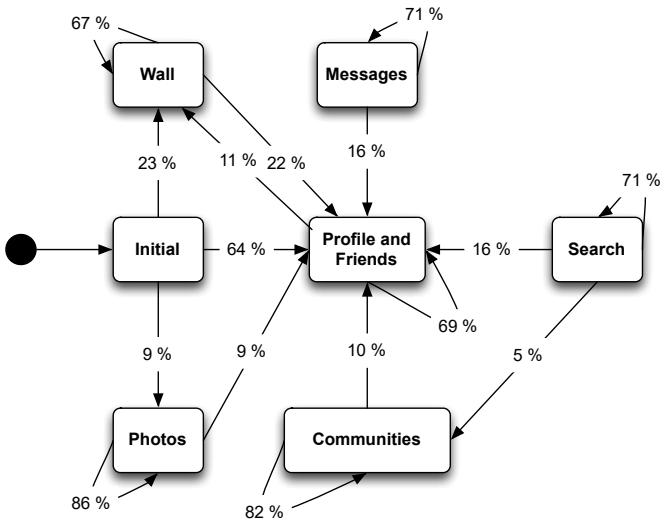
- All Plugins are OSGi – bundles
- Can be loaded, installed at runtime
- Define dependencies
- App market:
  - Host, search and install plugins
  - Determine and get dependencies



Motivation for Testing and Monitoring

Test Plugin for LifeSocial

- Coordinated automated execution & delegation of tasks
- Easy test setup
- Generation of content
- Expandable / Support for new LifeSocial plugins



Monitoring:

- Information on system status can be used for optimized decisions
  - E.g. peer count defines size of time-to-live
  - E.g. churn pattern defines stabilization frequency
- Necessary to identify (bad) quality of mechanisms
  - Too much overhead
  - Too slow routing
  - Efficiency leaks
- Helps in designing better mechanisms

Test Plugin I – Easy Setup & Generation of Content

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Automated Setup using Java RMI

- "First" node creates P2P network and serves as bootstrap node (Master node)
- Other nodes join P2P network and await orders (Slave nodes)

User content is generated without need for user interaction

- Resource folder contains photos and files
- Values for messages, group names, city names, country names, user interests, usernames, …
  - Example: login.txt file contains more than 4000 possible usernames

Test Plugin II – Automated Execution of Activities

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

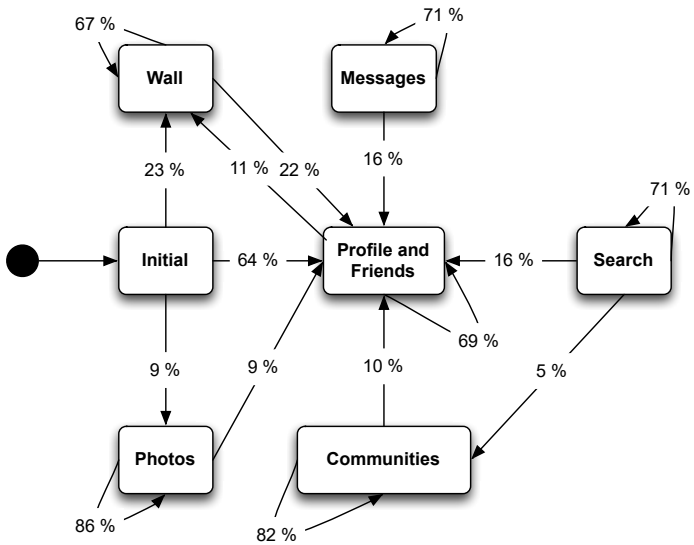
Activity: Execution of a social network function

- Has parameters and preconditions
- Example
  - Activity: Send a chat message to a friend
  - Precondition: Logged in and at least one friendship to another user
  - Parameter: Actual message content and name of the friend
  - If user has no friendship to another user, Test Plugin automatically sends a friend request to a random (online) user

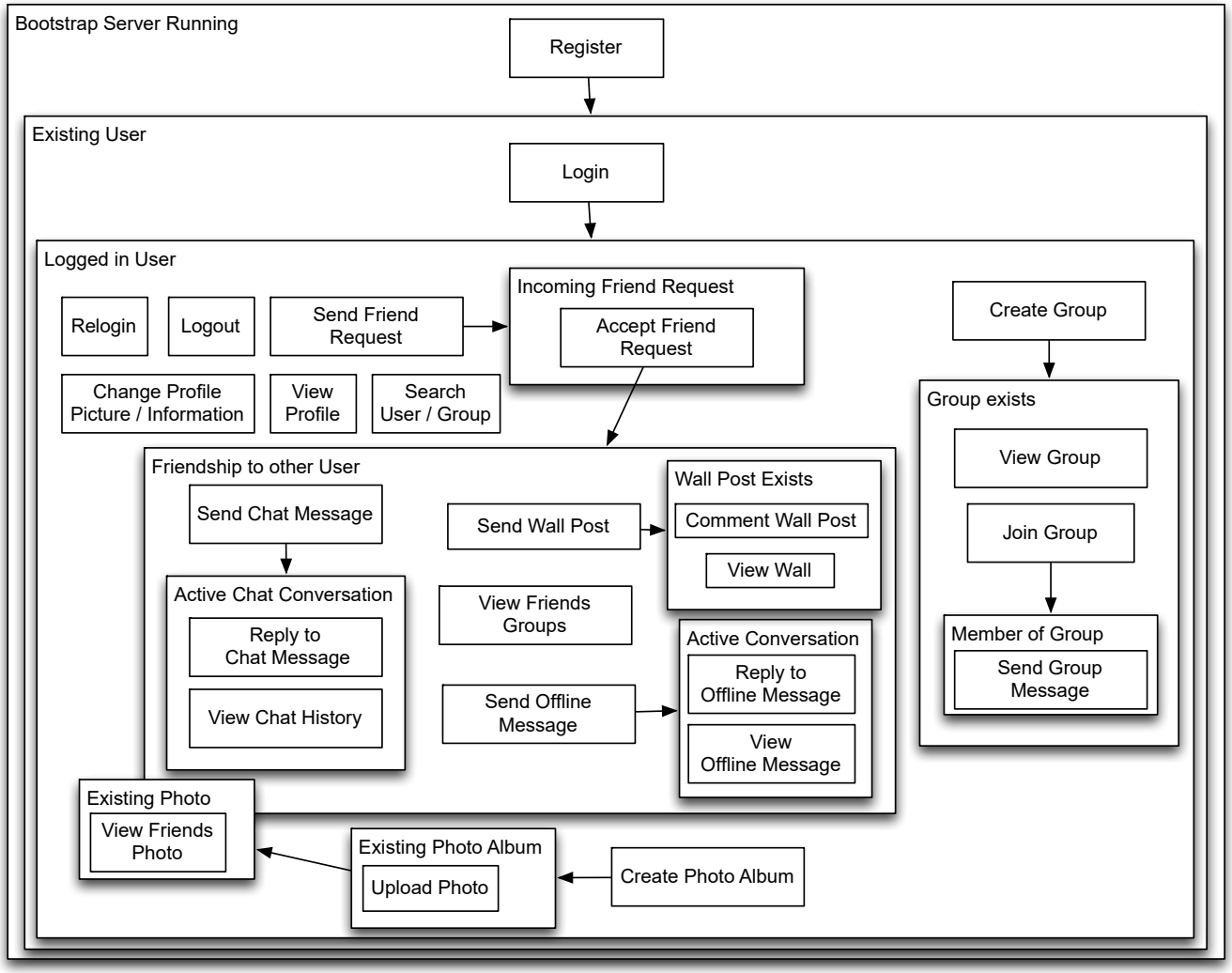
Preconditions are fulfilled if necessary

If activity parameters are missing

- they are chosen randomly from the resource files



Test Plugin II – Automated Execution of Activities



Test Plugin III – Delegation of Tasks

Every test participant can delegate tasks (social network activities) to other test participants

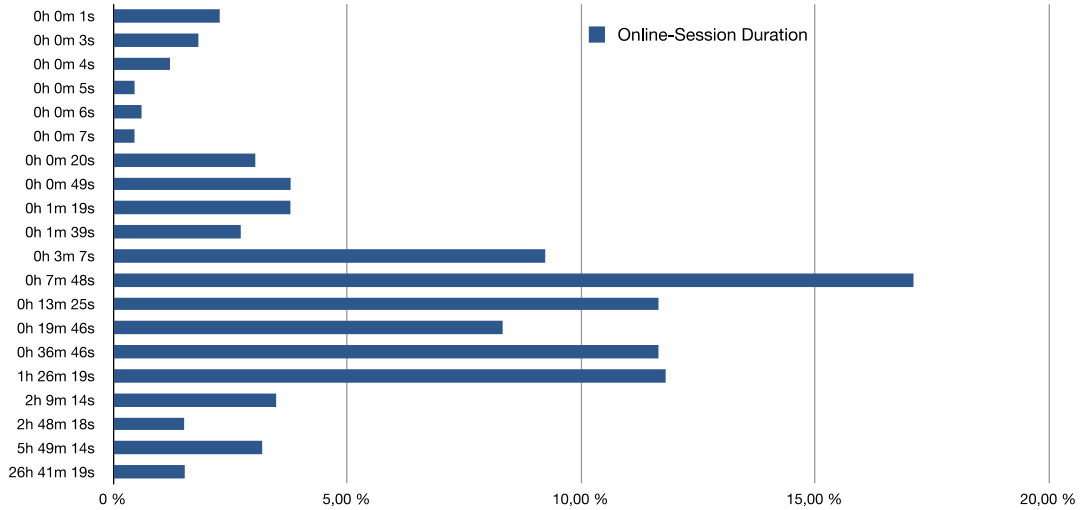
- Using 1-m messaging communication

Queue with remaining tasks to execute

Timed test plans

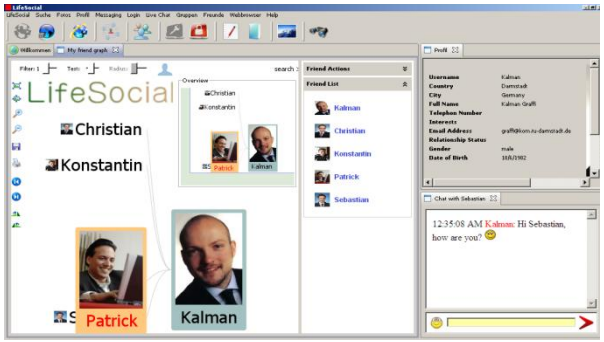


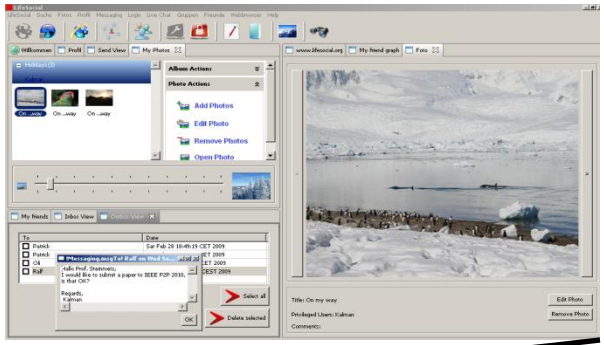
Realisitic churn model

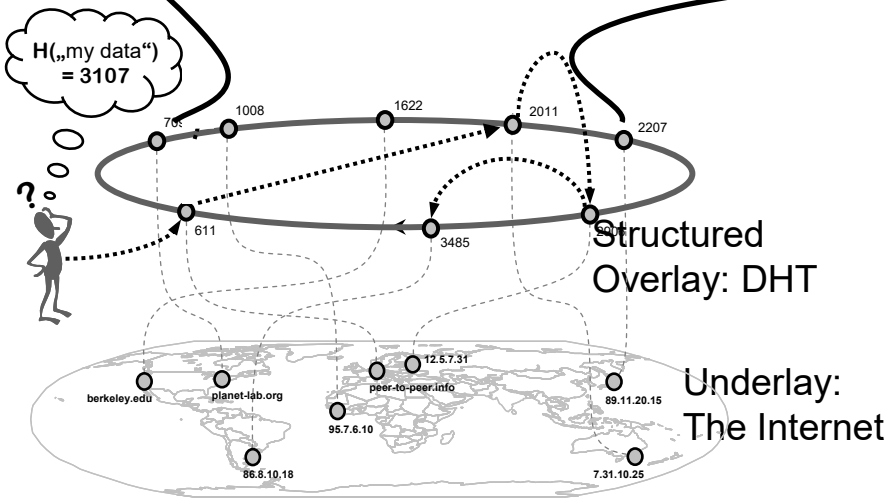


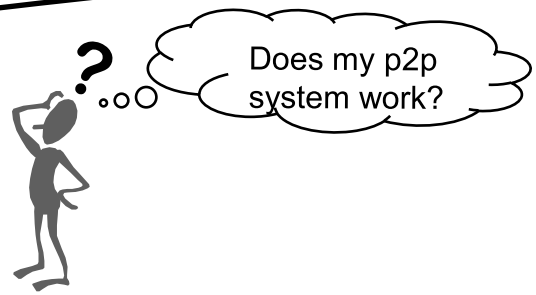
Monitoring











Is everything running fine?
How to debug and to gain insight?
How to improve the running system?

Design decisions in SkyEye.KOM

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

New layer (vs. integrated)
- New layer allows wider applicability
- Set on top of KBR-compatible structured p2p overlays

Proactive (vs. reactive)
- System state information is continuously interesting for all users

Monitoring topology: tree (vs. bus, ring, star, mesh)
- Fixed out and in degree

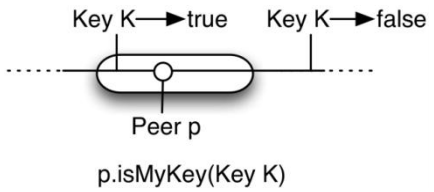
Position assignment: dynamic and deterministic
- Deterministic IDs used in topology, dynamically resolved with DHT

For all structured P2P overlays
- Covered by DHT-function: route(msg, key), lookup(key)

Assumptions

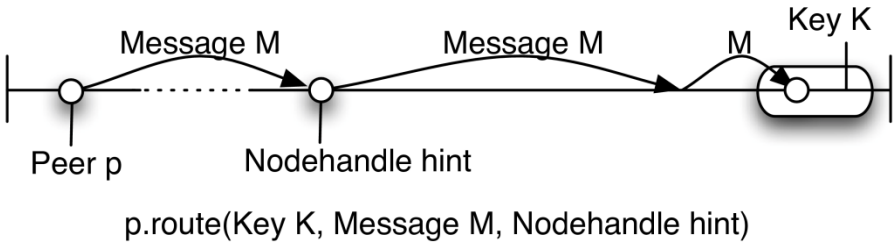
Reliable structured p2p overlay

- "Key-based Routing" – operations
  - boolean isMyKey(Key K)
  - void route(key K, Message M, node hint)



Building a tree topology

- Introduce new overlay layer
  - With own ID space ([0,1[)
- Create tree topology in new overlay
  - Using routing of p2p structured overlay



Concept of new layer

- Decouples from specific p2p overlay
- Unified ID space [0,1]

Tree-based Monitoring Mechanism

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Idea:

- Create (additional) tree topology
- Protocol:
  - Periodically
    - Calculate aggregate of own local view and received from child nodes
    - Send aggregate to parent node
  - Root calculates global view
    - And passes global view to all peers

Used in the p2p framework: SkyEye.KOM

- Assumes structured p2p overlay
- Aims at high precision with low overhead

SkyEye.KOM: Tree Topology

Tree of information domains

- Domain: ID interval
  - E.g. [0, 0.5[or [0.75, 0.875[
  - Largest domain, level 0: [0,1[
- Domain ID: “middle value” in interval
- Domain size split in β parts per level

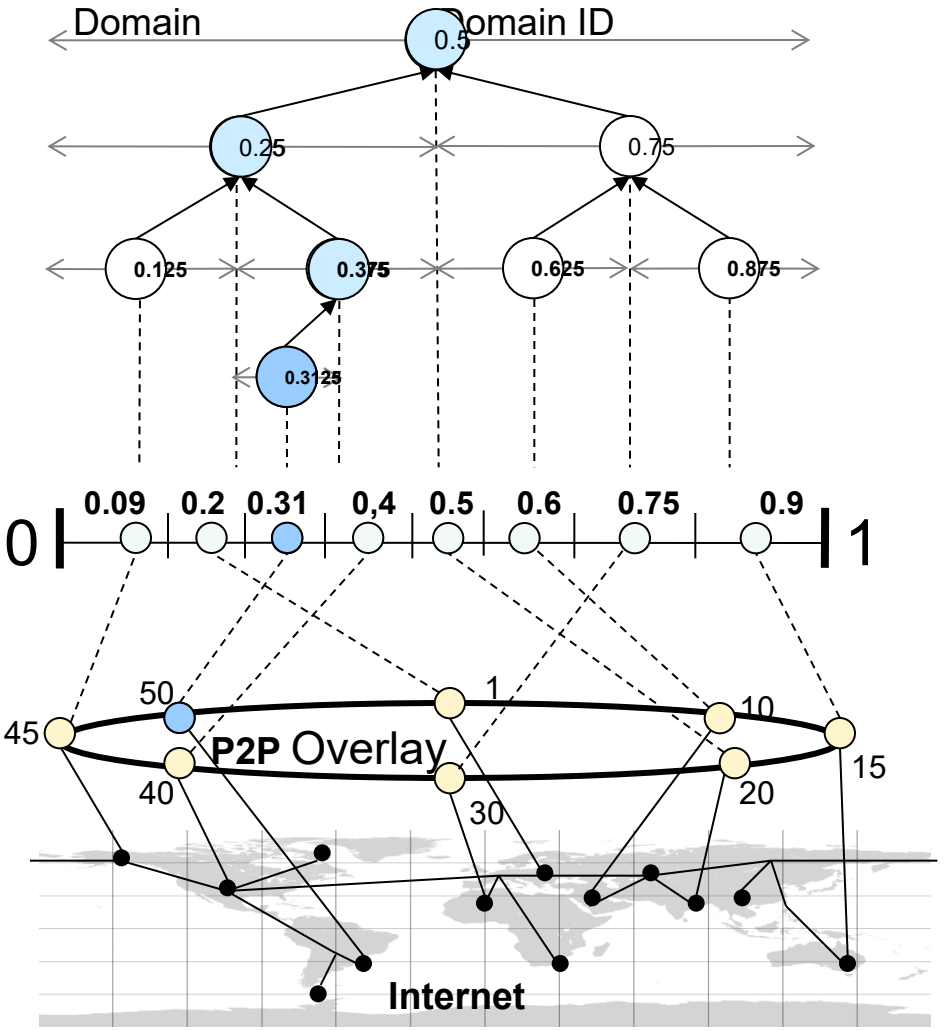
Domain IDs build tree topology

- Node degree: β child nodes
- Tree topology of domains does not change over time!
- Assignment of peers to domains dynamic

Peers to Domain ID assignment

- Peers calculate Domains in which they are located
- For those domains, they calculate the Domain IDs (K_p^l)
- If peer is responsible: position defined

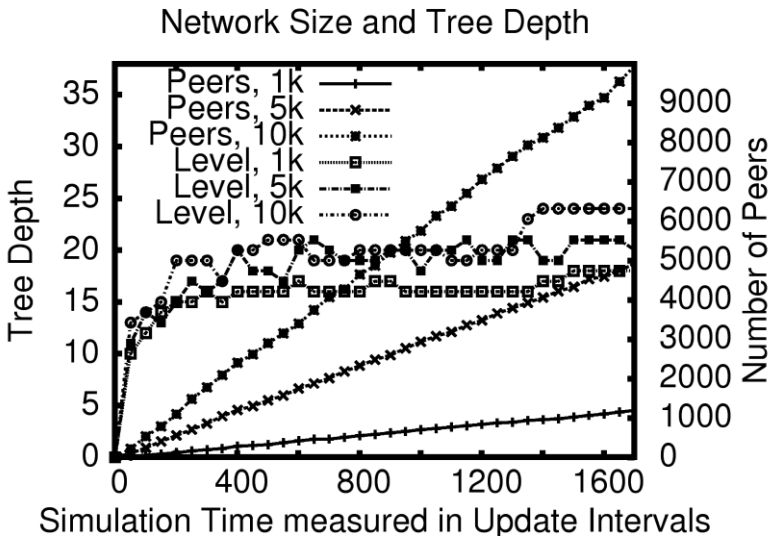
$$K_p^l := \frac{\frac{\lfloor ID_p \cdot \beta^l \rfloor + 1}{\beta^l} + \frac{\lfloor ID_p \cdot \beta^l \rfloor}{\beta^l}}{2}$$



SkyEye.KOM: Communication

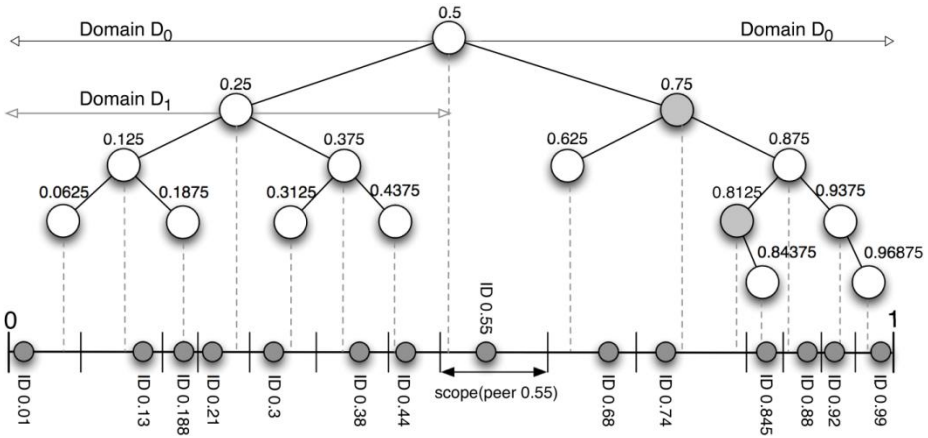
Tree-overlay ←→ p2p overlay

- Reconvert $treeID \in [0,1[$ to $peerID \in \{0, \dots, 2^{160} - 1\}$
- Coordinator:
  - Responsible for Domain ID
  - Check via DHT function
    - isMyKey(Key K)
- For communication in tree
  - Use route-function of overlay
  - route(Msg M,Key K,Node next)



Example tree

- Tree degree (β) = 2
  - Results in logarithmic tree size
- Balanced, if ID space balanced
- Not always β children
  - Peers may be Coordinators at various levels



SkyEye.KOM: Communication Protocol

Gathering global view

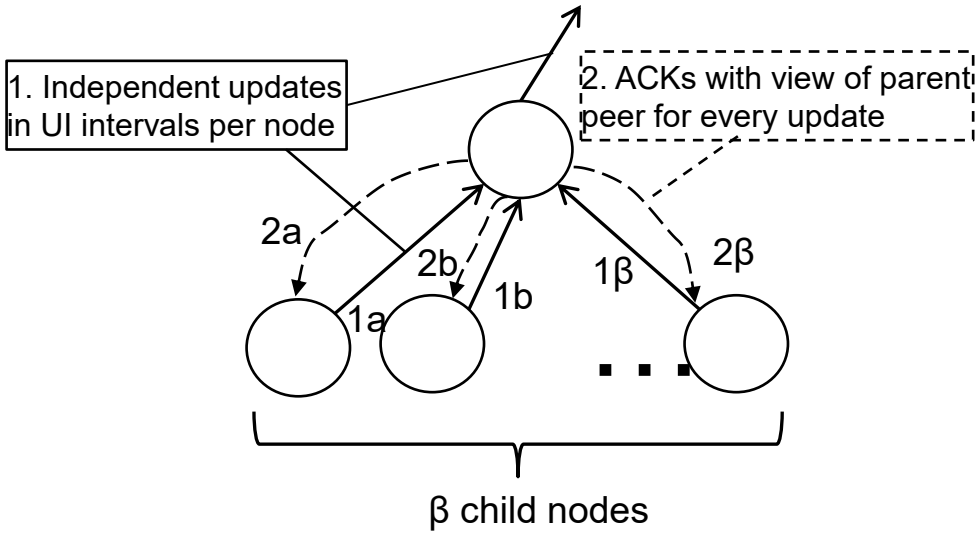
- All peers measure local status x_i
- Periodically sent to parent peer
  - → Update Interval (UI)

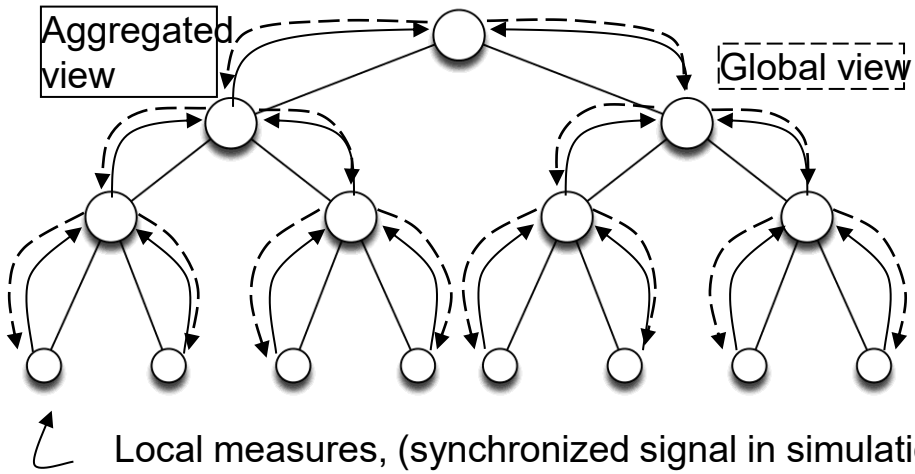
Aggregation

- Direct: count, sum, minimum, maximum, sum of squares
- Derived: mean, variance, std. deviation

Dissemination of global view

- Global view in root
- Every update message is acknowledged
- Contains global view from level above





Aggregation Functions

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Common aggregation functions that can be used on-the-fly

- $count(a_1, \dots, a_n) = n$
- $min(a_1, \dots, a_n) = min(a_1, \min(a_2, \dots min(a_{n-1}, a_n) \dots))$
  - $min(a_1, a_2) = \begin{cases} a_1 \; if \; a_1 \leq a_2 \\ a_2 \; if \; a_2 < a_1 \end{cases}$
- $max(a_1, \dots, a_n) = max(a_1, \max(a_2, \dots max(a_{n-1}, a_n) \dots))$
  - $max(a_1, a_2) = \begin{cases} a_1 \; if \; a_1 \geq a_2 \\ a_2 \; if \; a_2 > a_1 \end{cases}$
- $sum(a_1, \dots, a_n) = \sum_{i=1}^{n} a_i$
- $\text{mean}(a_1, \dots, a_n) = \frac{sum(a_1,\dots,a_n)}{count(a_1,\dots,a_n)}$
- $\text{standard_deviation}(a_1, \dots, a_n) = \sqrt{\text{variance}(a_1, \dots, a_n)}$

- $sum_of_squares(a_1, \dots, a_n) = \sum_{i=1}^{n} {a_i}^2$
- $\text{variance}(a_1, \dots, a_n) = \frac{sum_of_squares(a_1,\dots,a_n)}{count(a_1,\dots,a_n)} - \left(\frac{sum(a_1,\dots,a_n)}{count(a_1,\dots,a_n)}\right)^2$

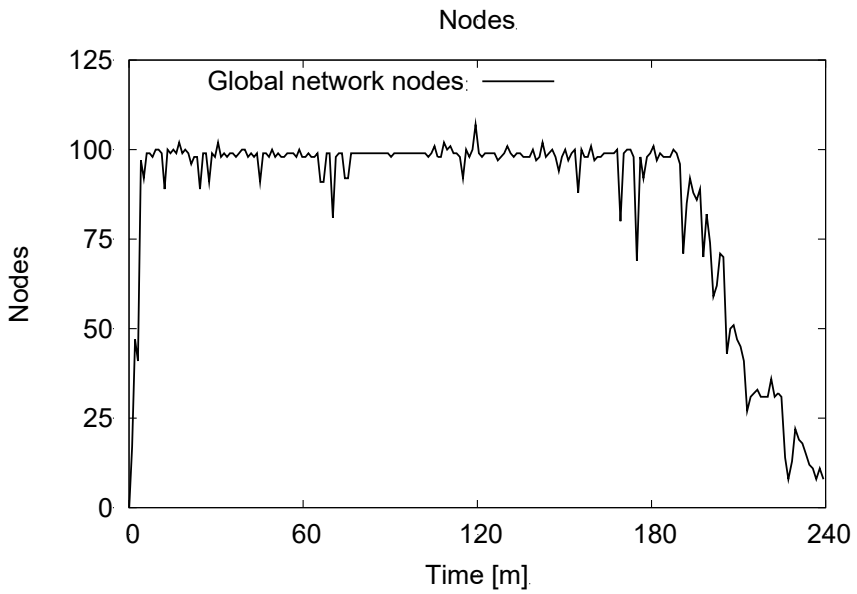
Example: Test Plan

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

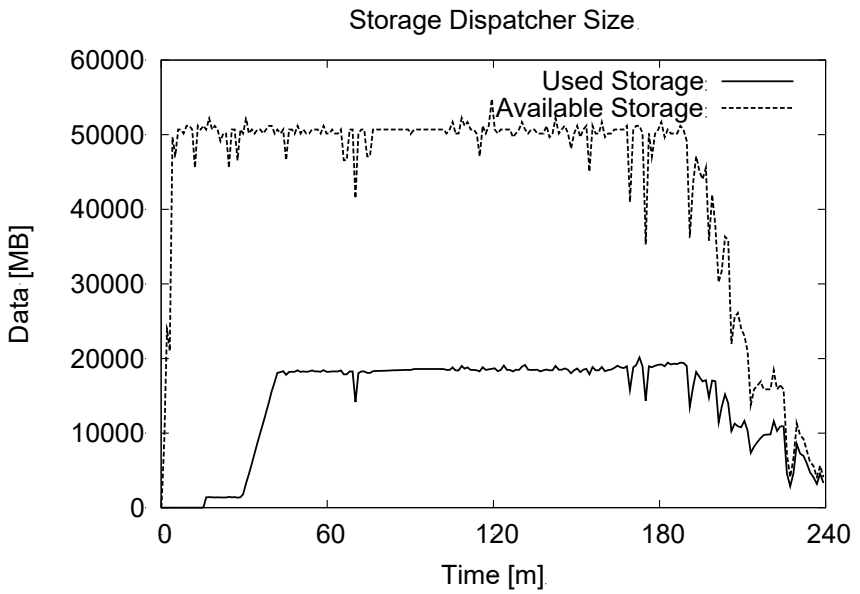
Activity	Amount of Repetitions	Planned Duration (in Minutes)
Login	1	15
Change Profile Picture	1	3
Send Friend Request	10	3
Create Photo Album	10	2
Upload Photo	200	15
View Friend's Photo	100	10
Join Group	30	10
Send Group Message	100	10
View Group Messages	10	5
Send Wall Post	100	10
Comment Wall Post	100	50
View Friend's Wall	5	5
Random Activities	-	30
Logout	1	50

Example: Test 3 – Results

Amount of Nodes

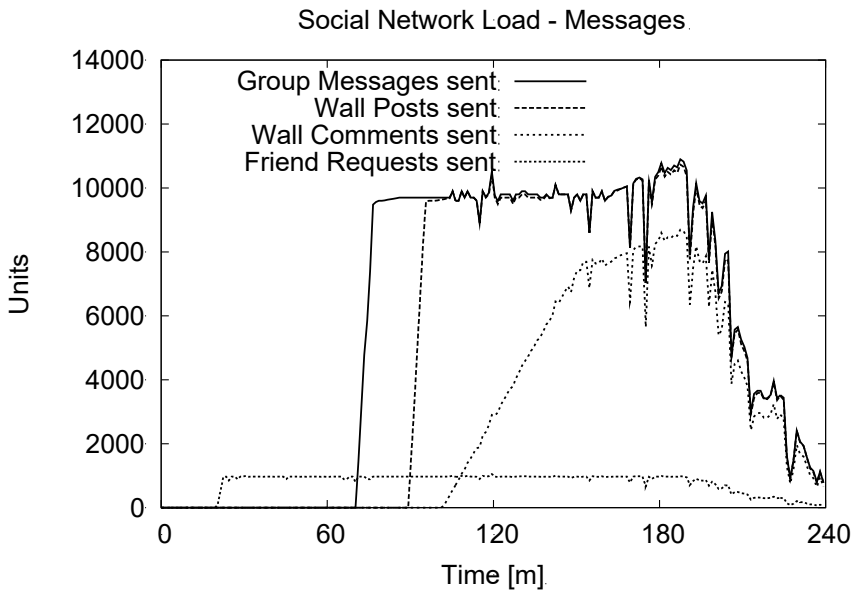


Available and Used Storage

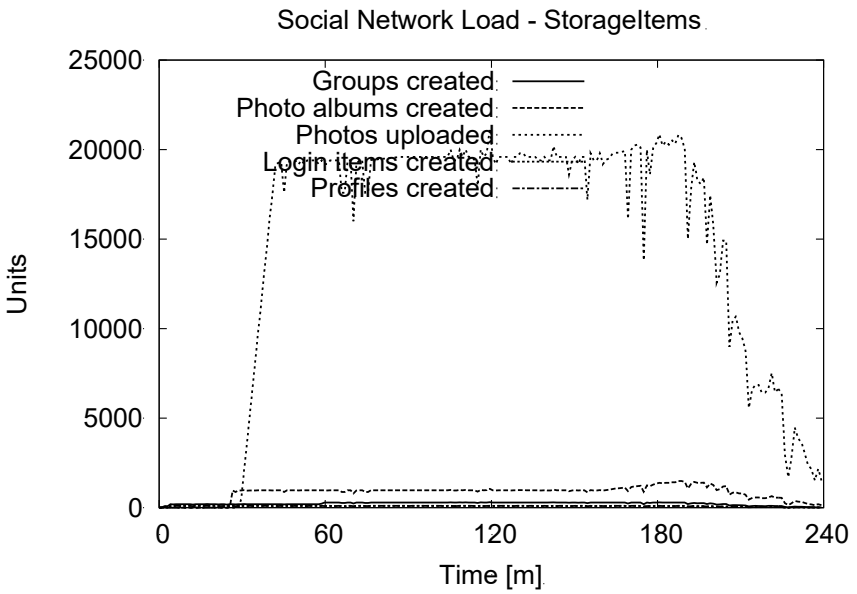


Example: Test 3 – Results

Load – Messages

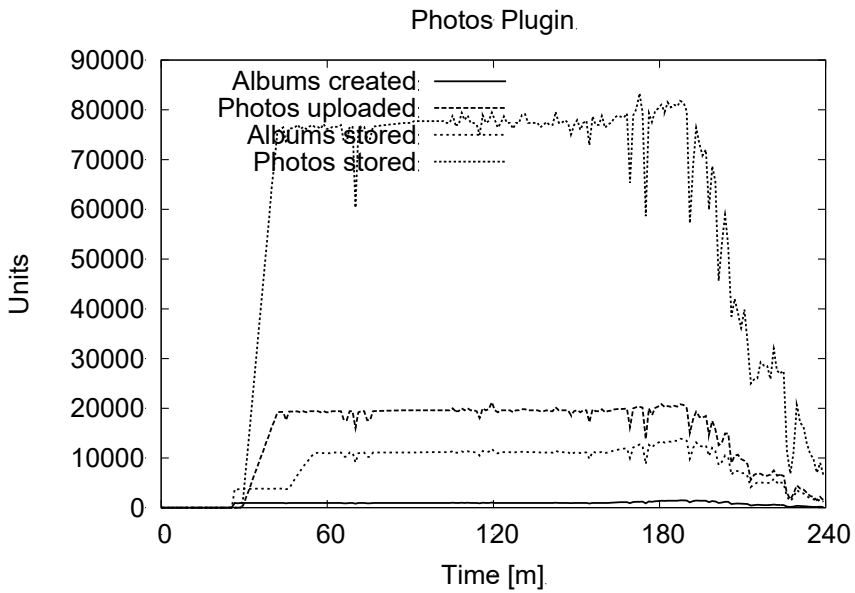


Load – StorageItems

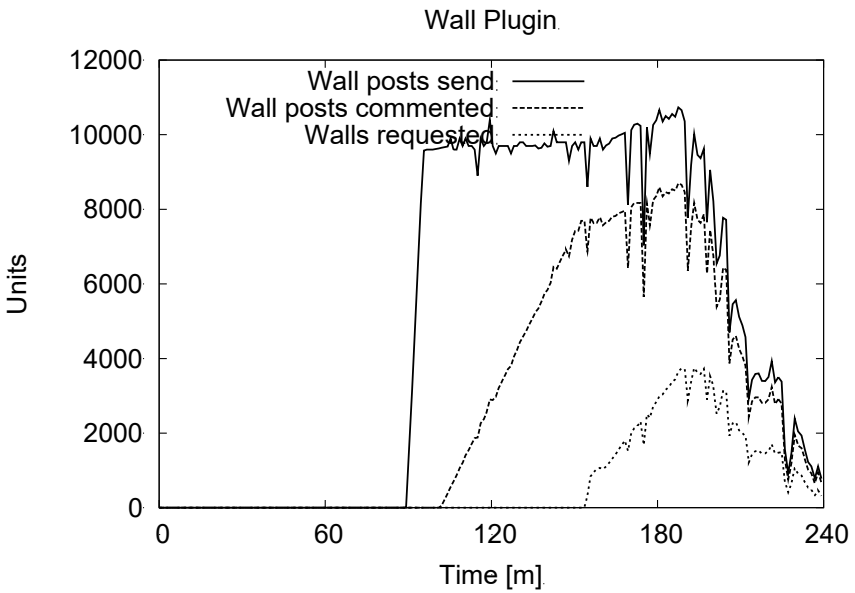


Example: Test 3 – Results

Photos Plugin

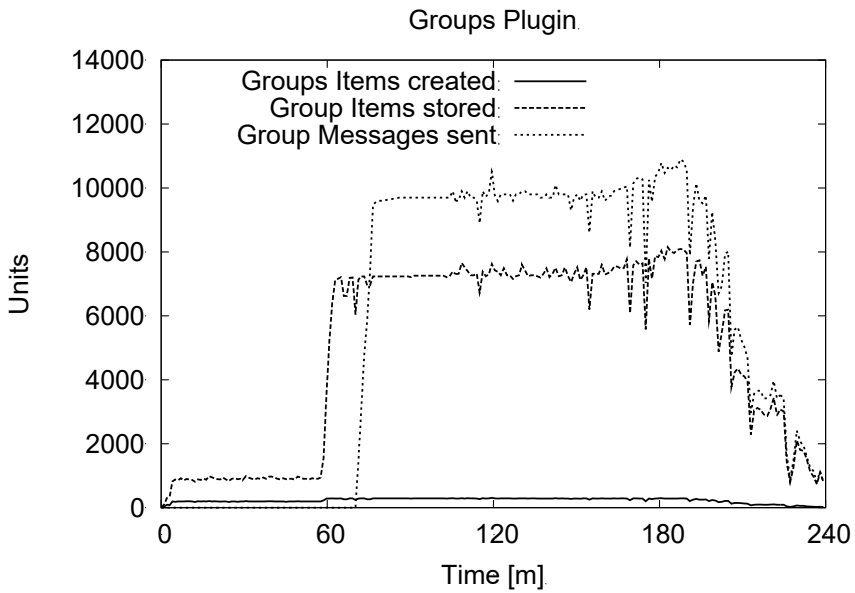


Wall Plugin

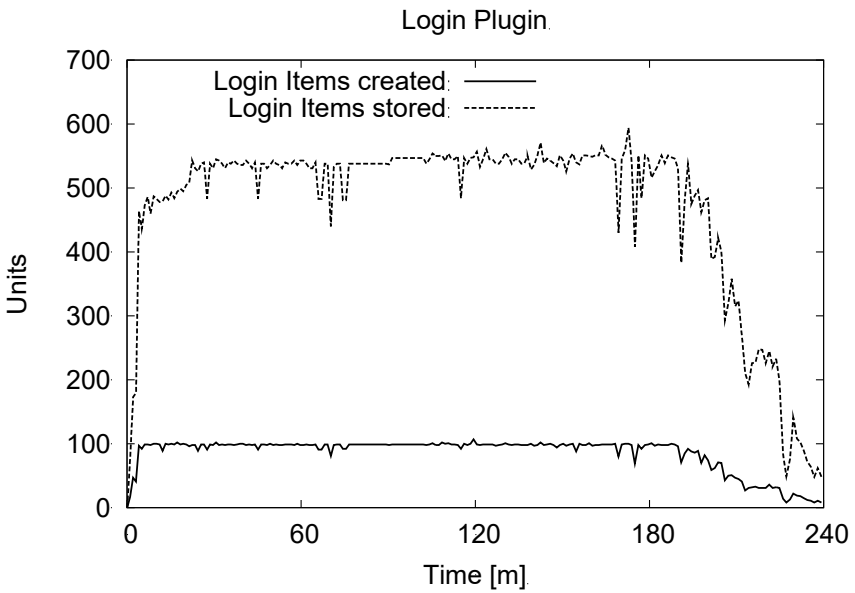


Example: Test 3 – Results

Groups Plugin

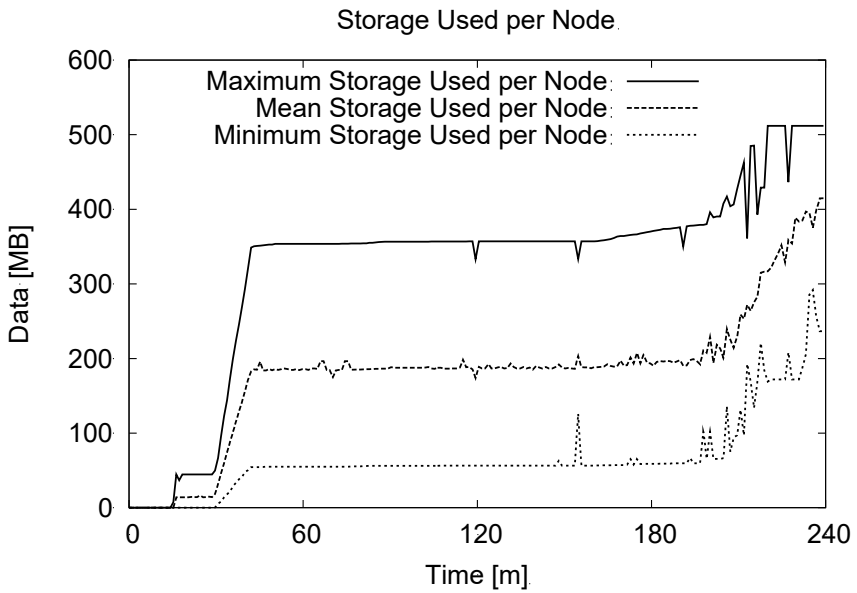


Login Plugin

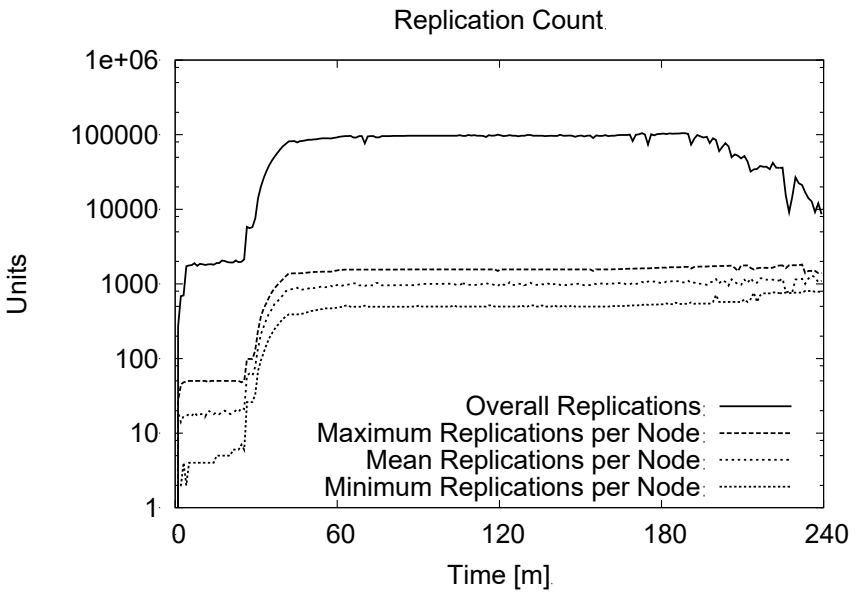


Example: Test 3 – Results

Storage per Node

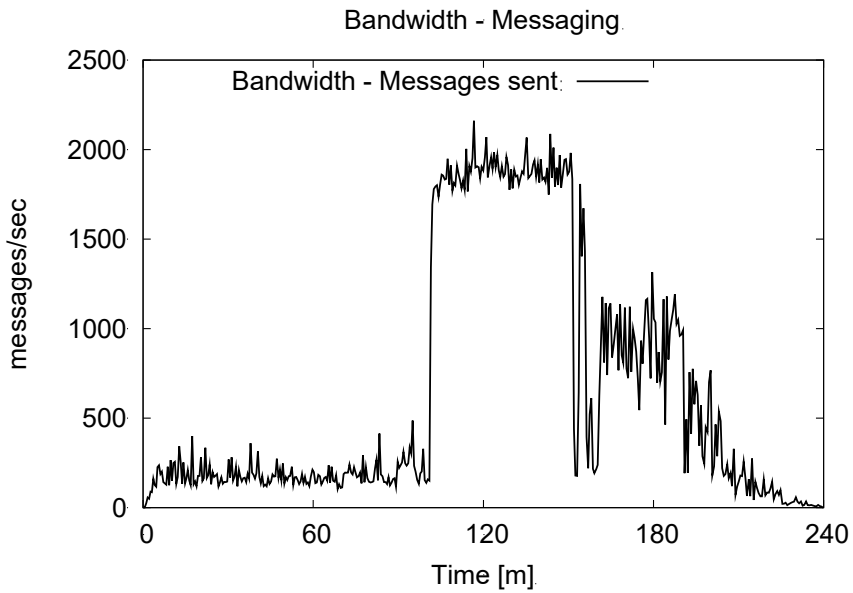


Replication Count

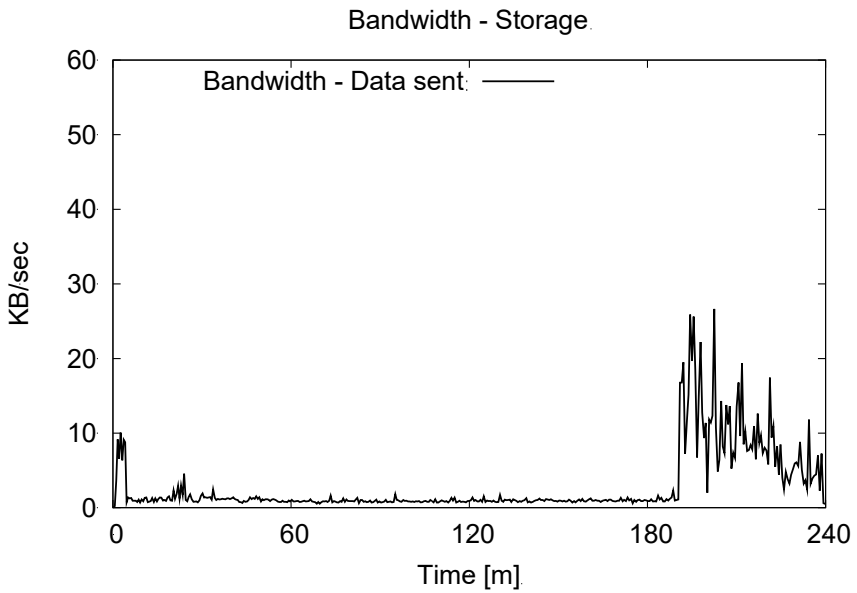


Example: Test 3 – Results

Bandwidth – Messages

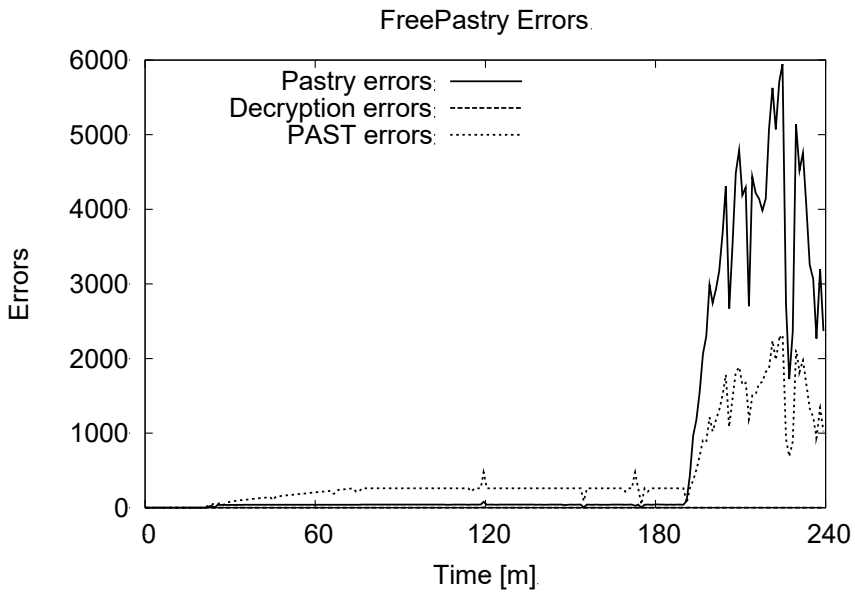


Traffic

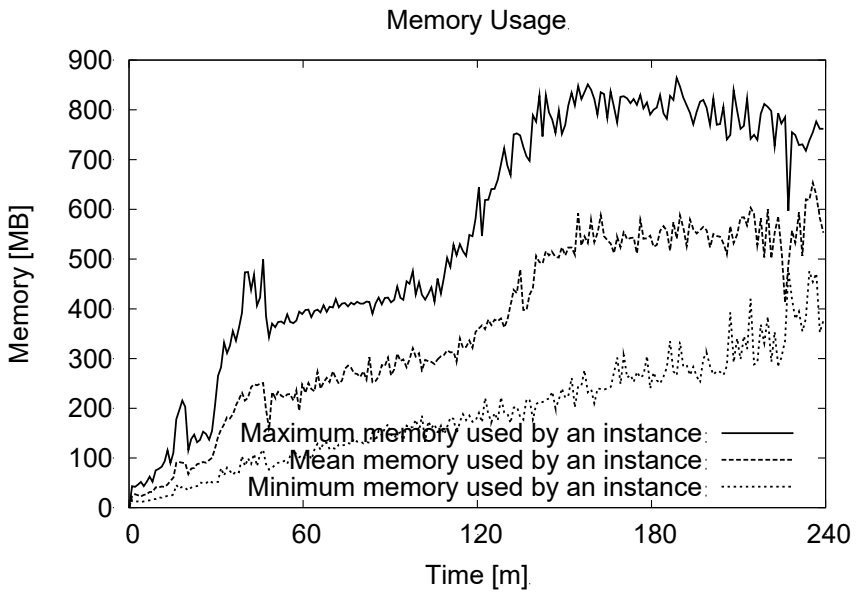


Example: Test 3 – Results

Amount of Errors



Memory Usage



Test Results – Summary I

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Synthetic Behavior of P2P Framework tested with 25, 50 and 100 nodes

- Amount of actual executed activities measured (uploaded photos, commented wall posts, …)
- Measured values match the expected values (according to the test plans)

→ LifeSocial behaves as expected during synthetic tests

Performance

- All activities finished in scheduled time or earlier (except for commenting wall posts)
  - > 19.000 photos in 13 minutes, 9600 group messages in 8 minutes, 9600 wall posts sent in 7 minutes

Bandwidth usage per node

- Always below 30 Kb/s
- In average: around 3 Kb / s

Data Storage

- Each Item replicated at least 2 times (matches defined replication factor)
- An average node contributed around 200 Mb storage space with a max-min load divergence of 350Mb:50Mb

Test Results – Summary II

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Monitoring and Test Plugin work reliable

- Small variances on monitoring results, still precise
- Test Plugin is a powerful tool to coordinate large-scale distributed tests

But: Prior tests revealed errors in LifeSocial

- Memory leak in FreePastry (has been fixed)
- Failure in MessageDispatcher
  - Deadlock leads to unresponsive application
  - Chat Plugin has been disabled for remaining tests
- Error-prone usage of LoginItem to obtain node ID
  - If the application crashes, the system assumes the user is still online

Failures in Search & Wall Plugin occur in unstable test environment (with churn)

- Failure in Search Plugin
  - Leads to endless search operations
- Failure in Wall Plugin
  - ClassCastException leads to application crash

Appstore - Requirements

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Appstore: Offer and retrieve new plugins

Some Requirements:

- Two roles: plugin provider and plugin user
- Plugin provider can publish new plugins, release new versions, edit or delete them
- Plugin user can search for plugins, download, install them and manage installed plugins
- Installed plugins: start/stop/update/uninstall

Related GUI for the above mentioned functionalities:

- Publisher View
- Search View
- Installed Plugins View

Appstore - Details

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Plugin – Model

- Plugin object
- Metadata: Plugin meta, version meta

Private repository (with unique user-related name)

- Contains the information of:
  - Published plugins
    - Offered plugins
  - Installed plugins
    - In order to maintain app-list in the network
    - Allows to install all apps at other device

Publisher actions: publish, remove, release new version, edit metadata

- Storage: Plugin object and metadata → network
- Meta-information of the published plugins → private repository

Search actions: precise search by name, download and install

- Exact plugin name is required
- Output: list of avaliable versions

Plugin user actions: start, stop, update, uninstall

A P2P-Framework for Social Networks

Plugins and Graphical User Interface

- Mandatory Plugins
- Optional Plugins

Graphical User Interface

- GUI Framework
- Perspectives
- Individual Views

LifeSocial: Plugins implementing a Social Network

Plugins:

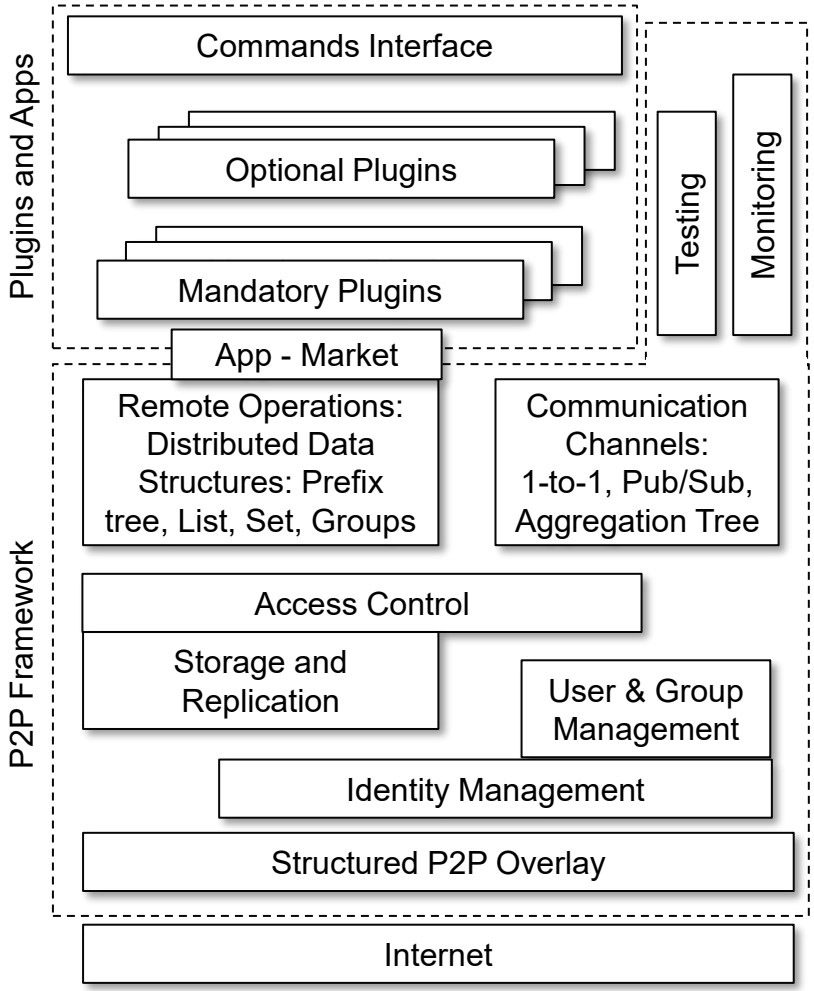
- Using the P2P Framework
- Written in Java, OSGi
  - Open Services Gateway initiative
  - Supports component model (bundle/service)
- Functionality of social networks
- Easy Plugin-to-Plugin communication
  - Over shared storage
    - E.g. photos
  - Over Plugin ID based messaging
    - E.g. Chat-app to chat-app

Mandatory Plugins

- Login, Profile, Friends, Wall, Groups, Photos, Chat, Messaging, Filetransfer (1-to-1), App-Market

Optional Plugins

- Voting, Multi-Chat, One-Click-Filehosting, Forum



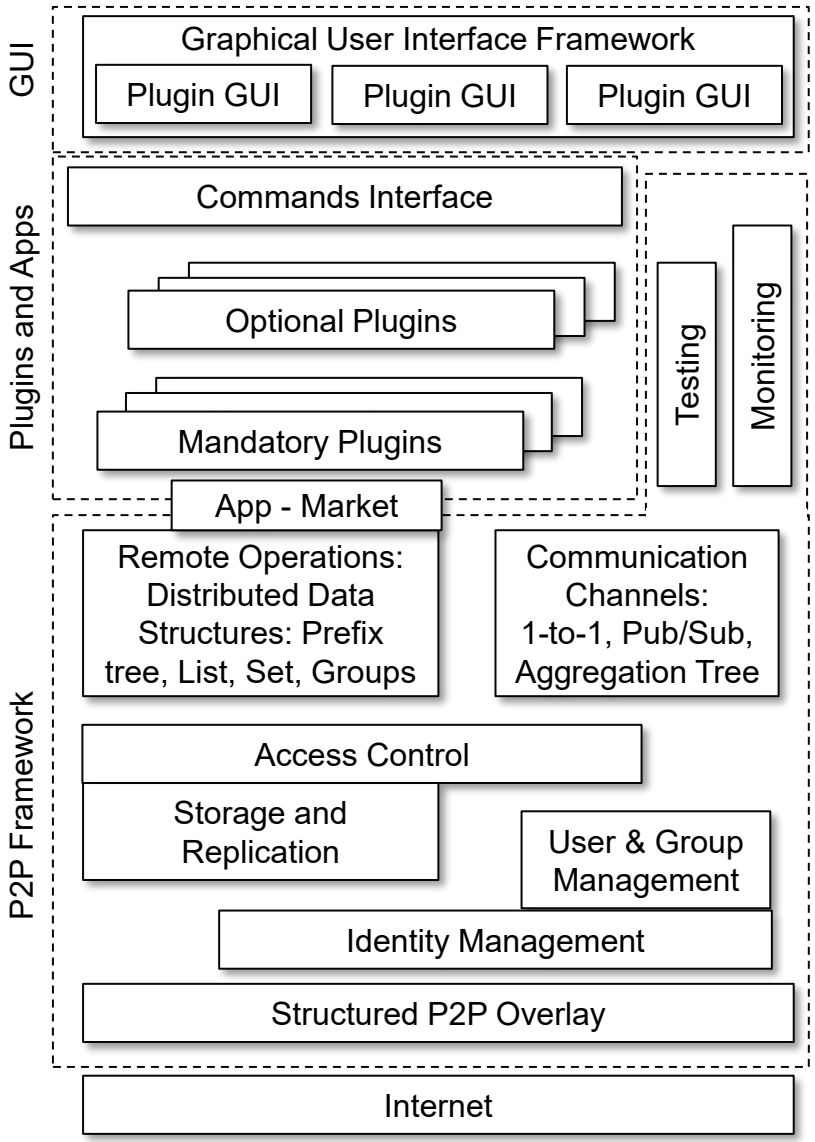
LifeSocial: Graphical User Interface

Graphical User Interface

- GUI Framework able to host individual Plugin Views
- Allows to arrange views
  - Save / load arrangements

Views

- Each Plugin comes with 1+ views
- Some views use several Plugins
- E.g. friends selector
- E.g. message views
  - inbox, outbox, compose



Current Plugins

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Login
- Create and login to user account

Profile
- Create and share personal information

Friends
- Link to friends

Mails
- Send and receive messages

Photos
- Upload, share and watch photos

Wall
- Write public messages, post own status, comment messages

Chat
- Chat with a friend

Multi-Chat
- Create / Join a topic channel and chat

Browser
- Extended views to display formated entries or websites

Groups
- Join groups and use dedicated working environments

Forum (Group element)
- Create threads and topics
- Discuss on topics

Voting
- Create polls, define answers, invite participants for public/private votes

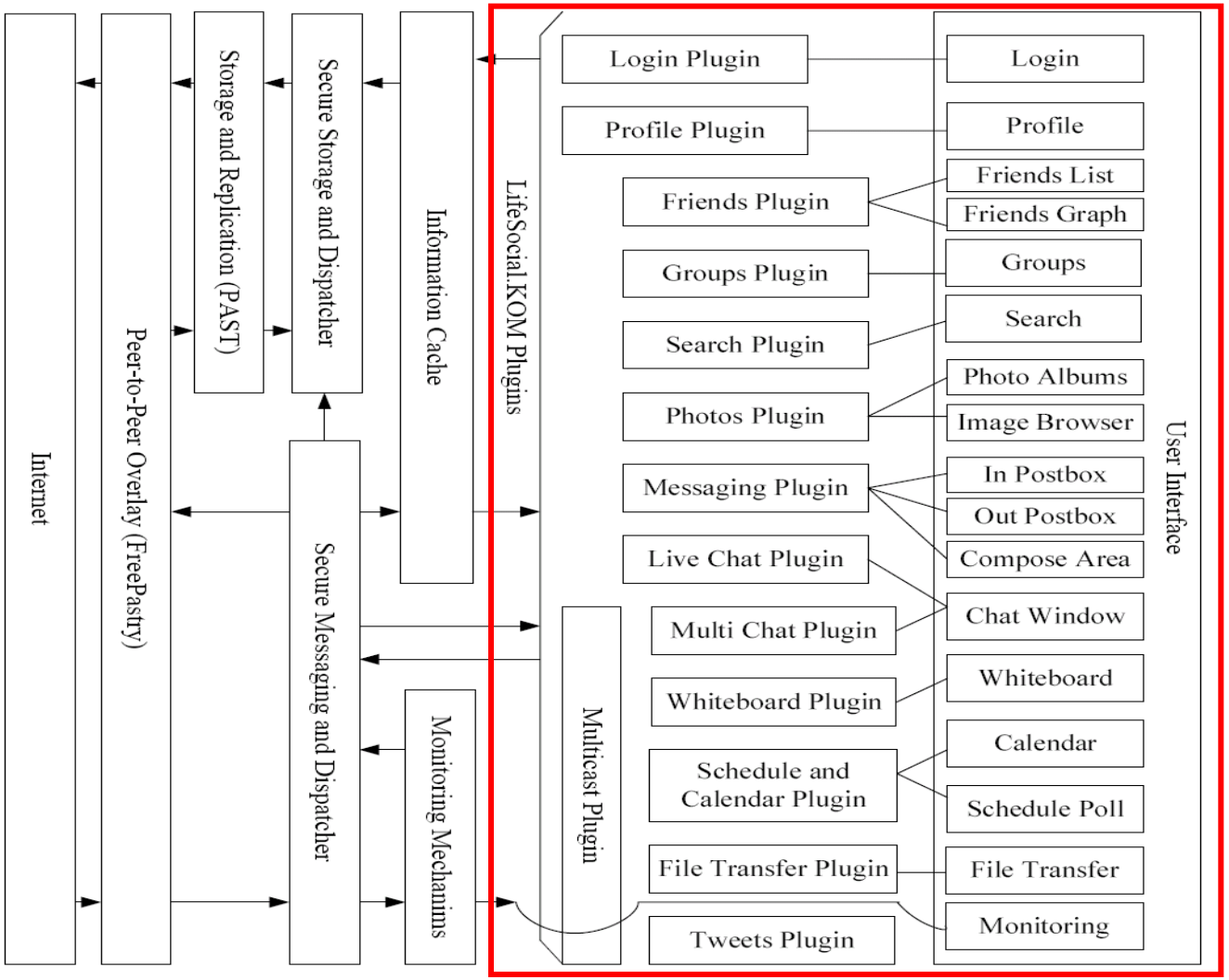
File Storage
- One-click file hosting: upload, get a link, share the link and allow other users to download files

App Store
- Create and share new plugins
- Maintain own plugin list independent of used device

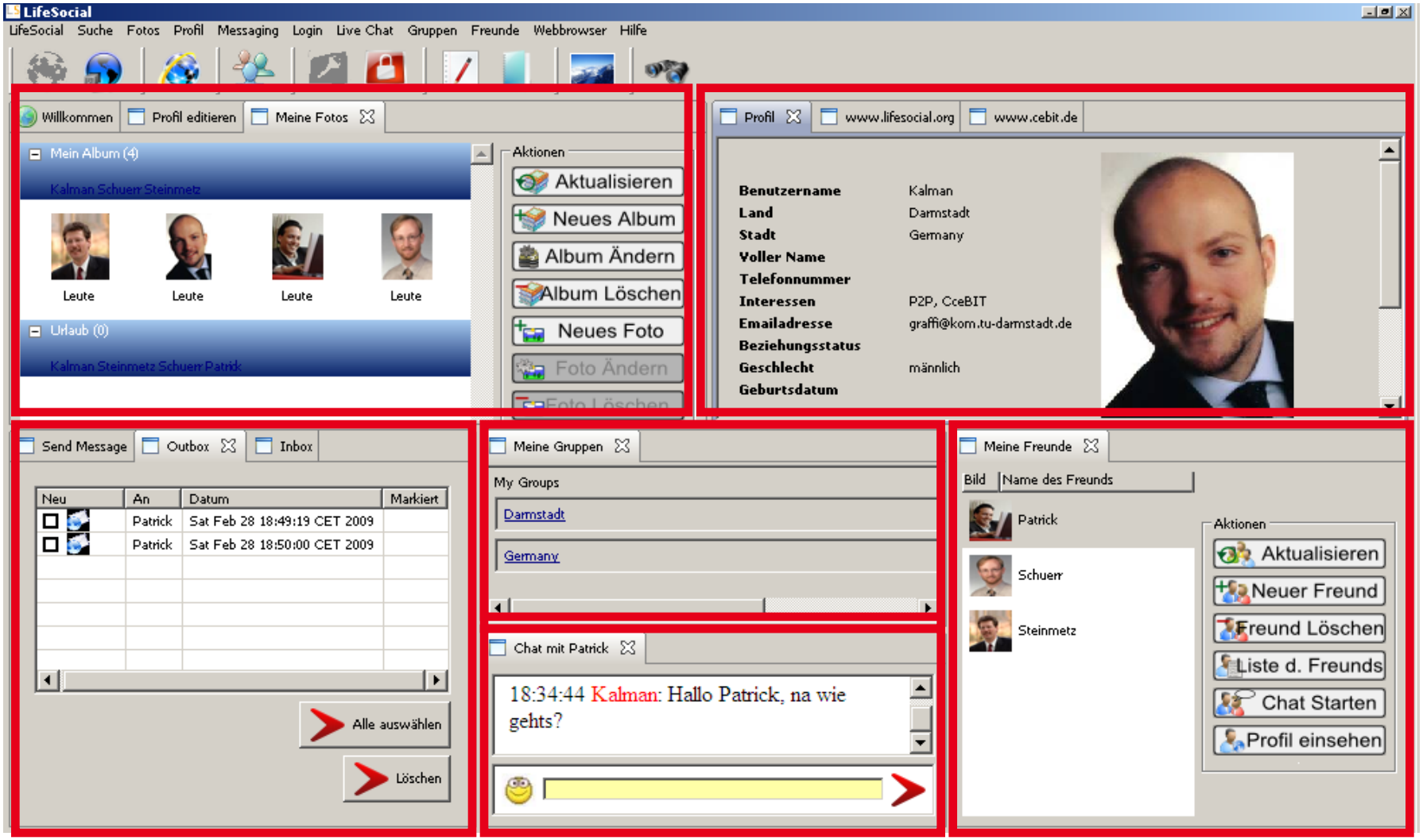
Monitoring
- Monitor usage of the network
- Existing graphical user interface

Test plugin
- Remotely control network peers for automatic test plan execution
- Content generation and real user environment simulation

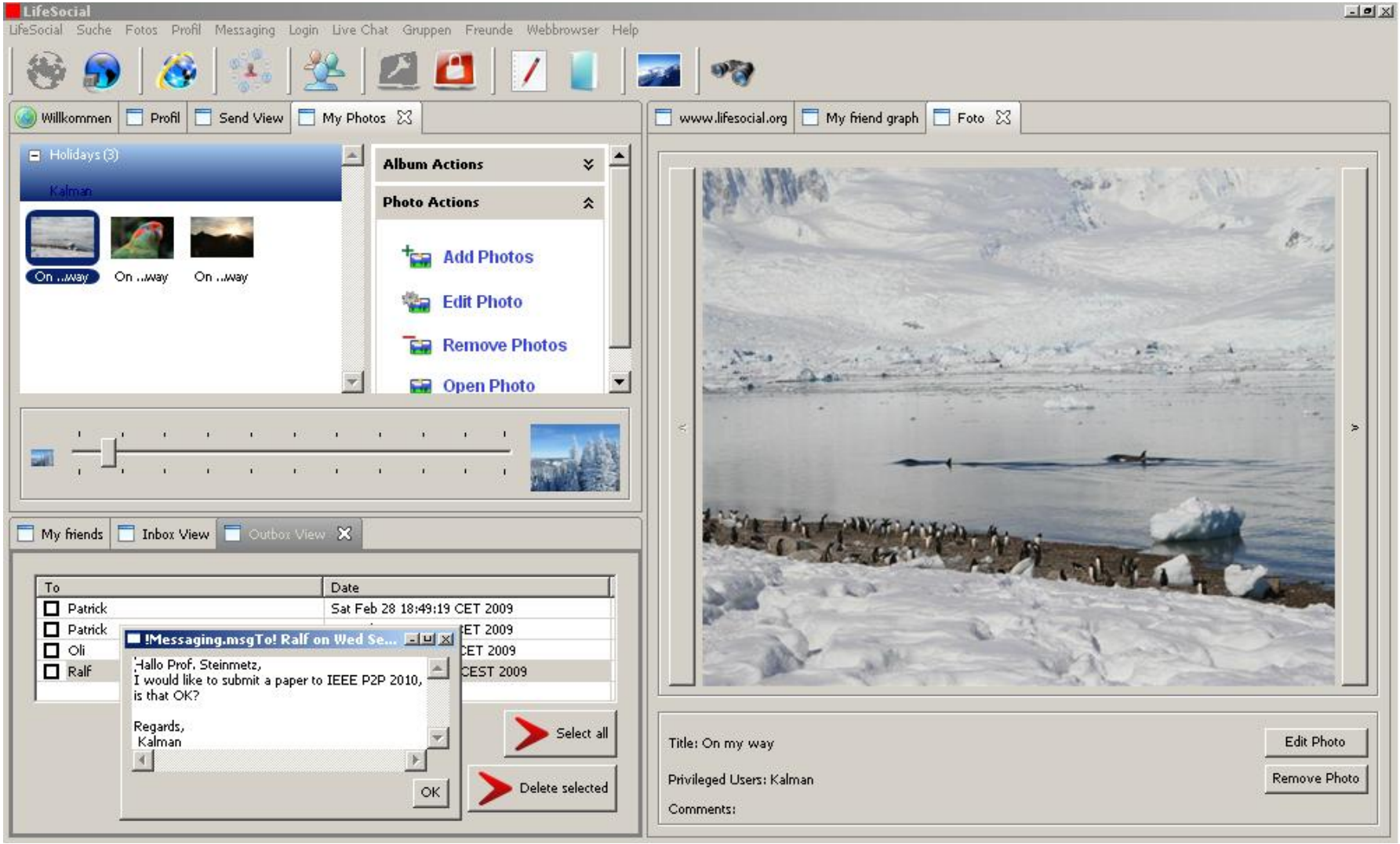
(Outdated) Plugin Architecture Overview



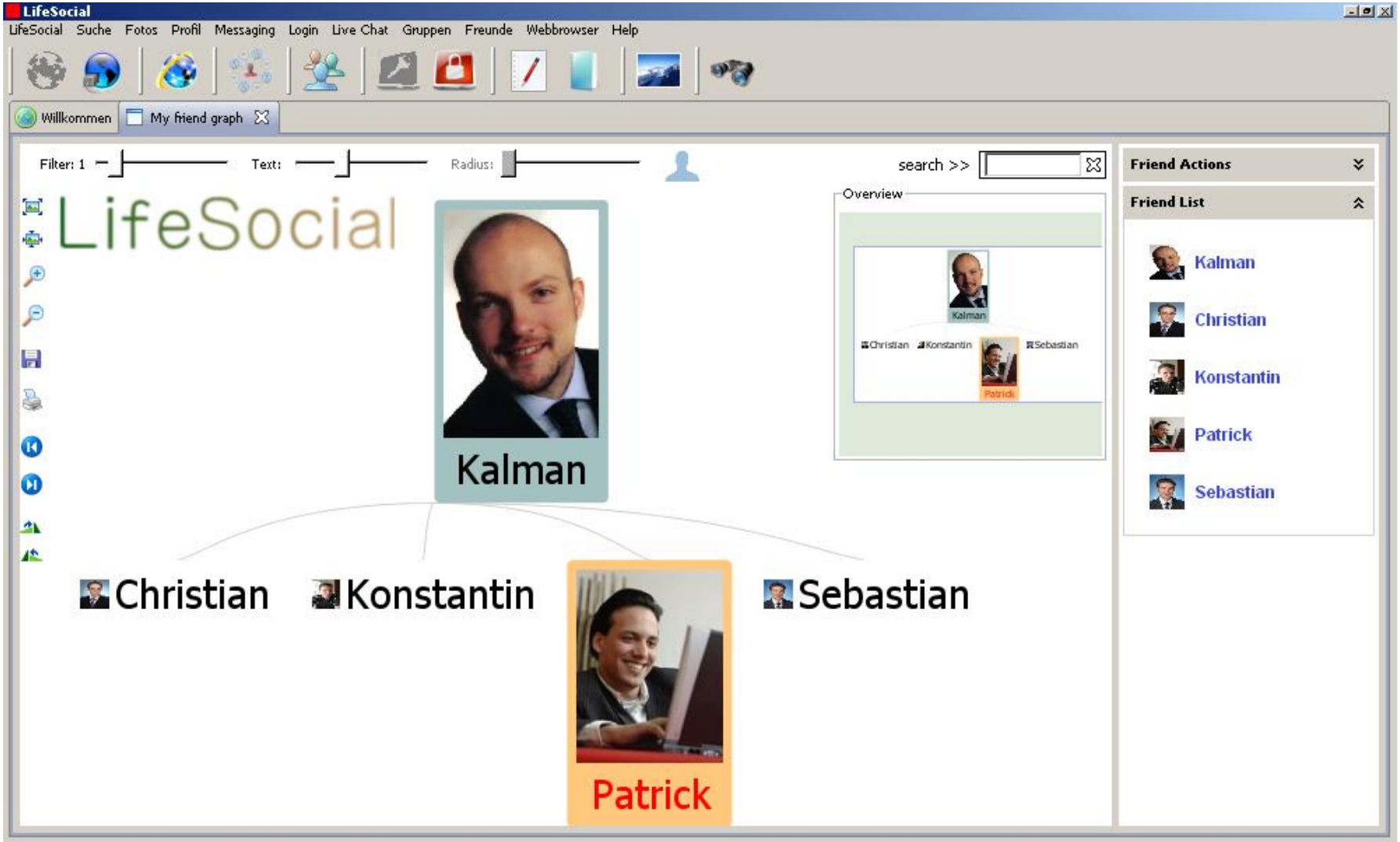
Plugins to GUI Relation



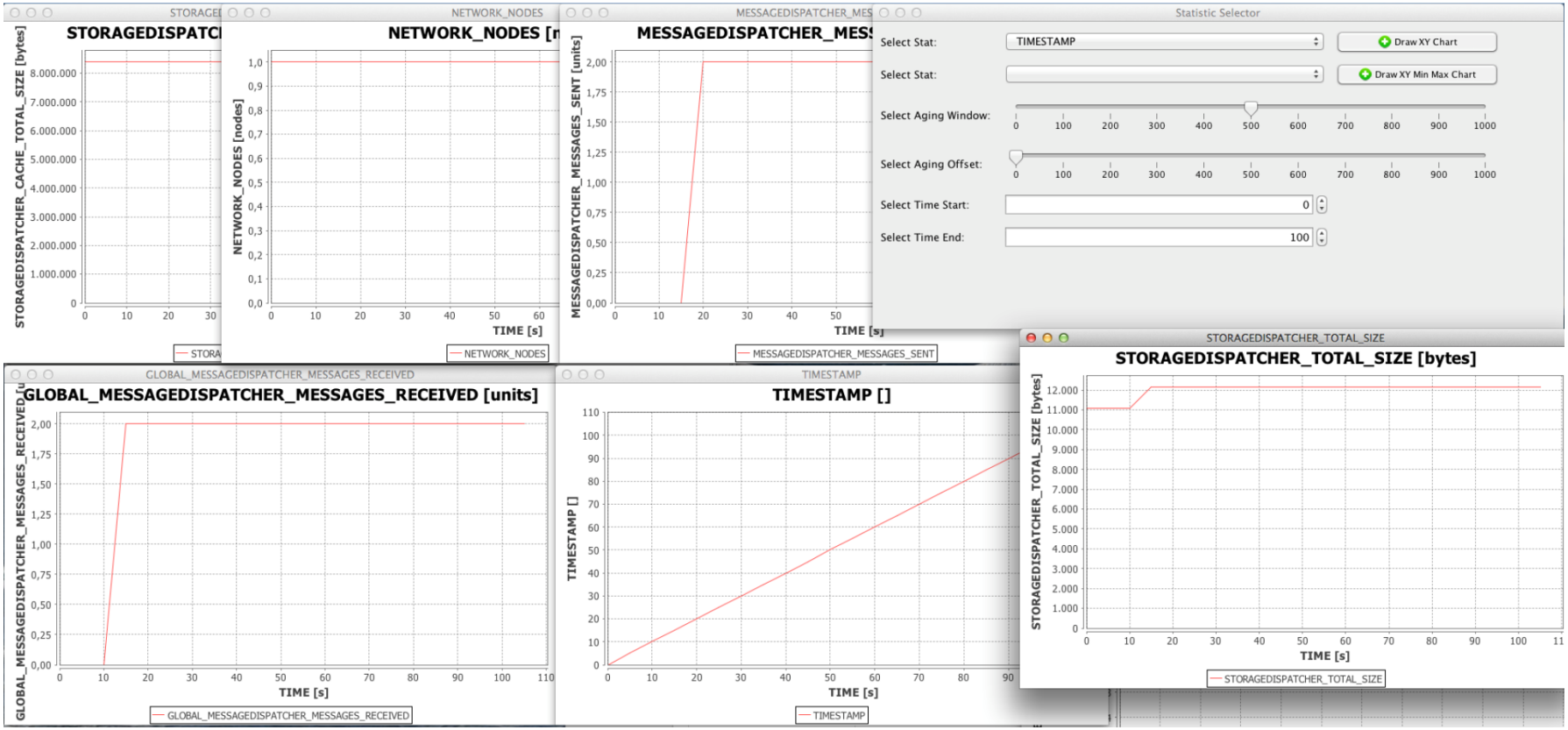
User View


LifeSocial
LifeSocial Suche Fotos Profil Messaging Login Live Chat Gruppen Freunde Webbrowser Help
Willkommen
Profil
Send View
My Photos
Holidays (3)
Kalman
On ...way
On ...way
On ...way
Album Actions
Photo Actions
Add Photos
Edit Photo
Remove Photos
Open Photo
www.lifesocial.org
My friend graph
Foto
My friends
Inbox View
Outbox View
To
Date
Patrick
Sat Feb 28 18:49:19 CET 2009
Patrick
Oli
Ralf
!Messaging.msgTo! Ralf on Wed Se...
Hallo Prof. Steinmetz,
I would like to submit a paper to IEEE P2P 2010,
is that OK?
Regards,
Kalman
OK
Select all
Delete selected
Title: On my way
Privileged Users: Kalman
Comments:
Edit Photo
Remove Photo

User View


LifeSocial
LifeSocial Suche Fotos Profil Messaging Login Live Chat Gruppen Freunde Webbrowser Help
Willkommen
My friend graph
Filter: 1
Text:
Radius:
search >>
LifeSocial
Overview
Kalman
Christian
Konstantin
Patrick
Sebastian
Friend Actions
Friend List
Kalman
Christian
Konstantin
Patrick
Sebastian

Management View



Graphical User Interface Details

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Task of views

- Combine functionality from framework to applications
- Often uses several plugins

Plugins might come with 0-n views

- Email has 2+ views
  - Compose, inbox/outbox
- Some plugins have no views
  - Search plugin only as helper

Perspectives

- View combinations and positions can be stored as perspectives
- Allows for customized allocations
  - E.g. Working perspective, communincation perspective …

A P2P-Framework for Social Networks

Outlook – Future Work

Current Steps in LifeSocial

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Ongoing project group (16 students)

- "A Peer-to-Peer Framework for Social Networks"
- Goal
  - Finalize p2p framework: secure, quality-controlled, …
  - Add further essential functionality to framework
    - Live multimedia streams (chatting, streaming)
  - Add more novel applications

2014 presentation at CeBIT

- Like 2009, 2010, 2012, 2013

Contributions for the community

- New applications induce new research challenges
- Publish p2p framework as open source
- Initiate a community for further developments

Several Open Topics and Application Areas

E-Learning Environment

- Live lectures, tests
- Data repository

Realtime Communication

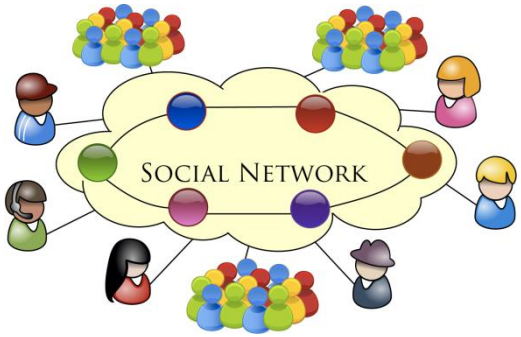
- Voice / video

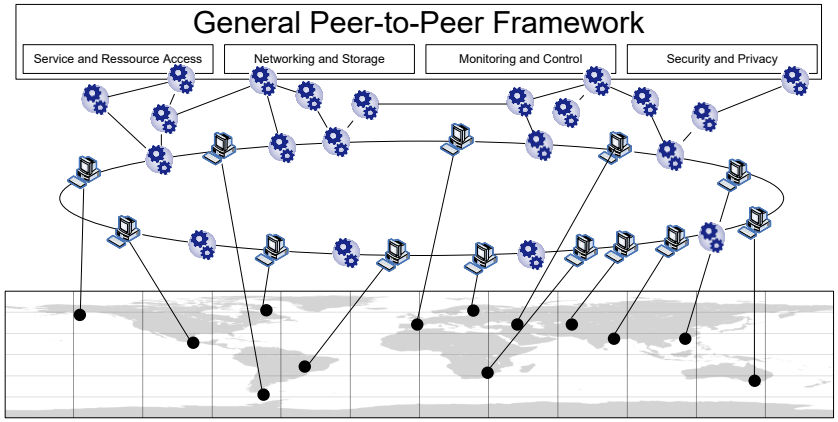
P2P Wiki

- Versioning

P2P Normsetting Environment

- Proposals, Votes
- Decisions





Publications

P2P Framework – LifeSocial Overview

- K. Graffi, C. Groß, D. Stingl, D. Hartung, A. Kovacevic, R. Steinmetz „LifeSocial.KOM: A Secure and P2P-based Solution for Online Social Networks“. In: Proc. of the IEEE Consumer Communications and Networking Conference, (IEEE CCNC'11), Januar 2011
- K. Graffi, C.Groß, P. Mukherjee, A. Kovacevic, R. Steinmetz: „LifeSocial.KOM: A P2P-based Platform for Secure Online Social Networks“. In: Proc. of the 10th IEEE International Conference on Peer-to-Peer Computing, (IEEE P2P'10), August 2010.
- K. Graffi, S. Podrajanski, P. Mukherjee, A. Kovacevic, R. Steinmetz: „A Distributed Platform for Multimedia Communities“. In: Proc. of the IEEE International Symposium on Multimedia (IEEE ISM '08), December 2008.

Security

- K. Graffi, P. Mukherjee, B.Menges, D. Hartung, A. Kovacevic, R. Steinmetz: „Practical Security for P2P-based Social Networks“. In: Proc. of the 9th IEEE International Conference on Local Computing Networks, (IEEE LCN'09), August 2009.

Monitoring

- K. Graffi, D.Stingl, J. Rueckert, A. Kovacevic, R. Steinmetz: „Monitoring and Management of Structured P2P Systems“. In: Proc. of the 9th IEEE International Conference on Peer-to-Peer Computing, (IEEE P2P'09), September 2009.
- K. Graffi, A. Kovacevic, S. Xiao, R. Steinmetz: „SkyEye.KOM: An Information Management Over-Overlay for Getting the Oracle View on Structured P2P Systems“. In: Proc. of the 14th IEEE International Conference on Parallel and Distributed Systems (IEEE ICPADS'08), December 2008.
- K.Graffi, „Monitoring and Management of Peer-to-Peer Systems“, Technische Universität Darmstadt, 2010.

Heterogeneity handling

- K.Graffi, S. Kaune, K.Pussep, A.Kovacevic, R.Steinmetz: "Load Balancing for Multimedia Streaming in Heterogeneous Peer-to-Peer Systems“. In: Proc. of the 8th ACM SIGMM Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (ACM NOSSDAV '08), May 2008.

Related evaluation tools:

- M. Feldotto, K. Graffi: "Comparative Evaluation of Peer-to-Peer Systems using PeerfactSim.KOM ". In Proc. of the IEEE International Conference on High Performance Computing and Simulation (IEEE HPCS '13), 2013
- K. Graffi: "PeerfactSim.KOM: A P2P System Simulator – Experiences and Lessons Learned“. In: Proc. of the IEEE International Conference on Peer-to-Peer Computing, (IEEE P2P'11), August 2011

Related load-balancing / heterogeneity awareness

- L. Bremer, K. Graffi: "Symbiotic Coupling of P2P and Cloud Systems: The Wikipedia Case“. In Proc. of the IEEE International Conference on Communications (IEEE ICC '13), 2013
- P. Wette, K. Graffi: "Adding Capacity Awareness and Load Balancing to Homogeneous Distributed Hash Tables“. In Proc. of the IEEE International Conference on Networked Systems (IEEE NetSys '13), 2013
- K. Graffi, C. Groß, D. Stingl, H.Nguyen, A. Kovacevic, R. Steinmetz: „Towards a P2P Cloud: Reliable Resource Reservations in Unreliable P2P Systems“.In: Proc. of the 16th IEEE International Conference on Parallel and Distributed Systems, (IEEE ICPADS'10), December 2010

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

More information: www.p2pframework.com

