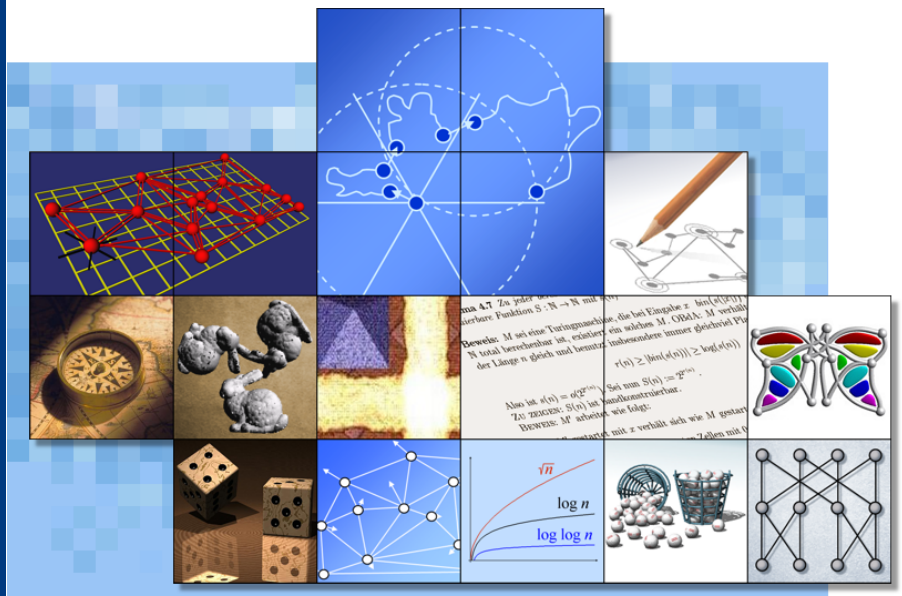
HEINZ NIXDORF INSTITUTE
University of Paderborn

Secure Distributed Data Structures for Peer-to-Peer-based Social Networks

P2PCS as part of CTS 2014



May 21, 2014

Jens Janiuk
Alexander Mäcker
Kalman Graffi

1 Introduction

2 Distributed List Concept

3 Access Control in Distributed List
- Read and Write Access
- Key Distribution

4 Evaluation

Online Social Networks (OSNs)

HEINZ NIXDORF INSTITUTE
University of Paderborn

Facebook, Google+, Twitter, . . . heavily used nowadays

- (semi-) public user profiles
- communication
- collaboration: sharing and searching user generated content

Online Social Networks (OSNs)

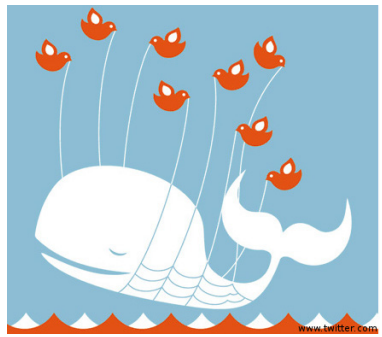
HEINZ NIXDORF INSTITUTE
University of Paderborn

Facebook, Google+, Twitter, . . . heavily used nowadays

- (semi-) public user profiles
- communication
- collaboration: sharing and searching user generated content

Current approaches are centralized

- full access to data: massive data-mining
- censorship
- single point of failure, scalability
- high operational costs



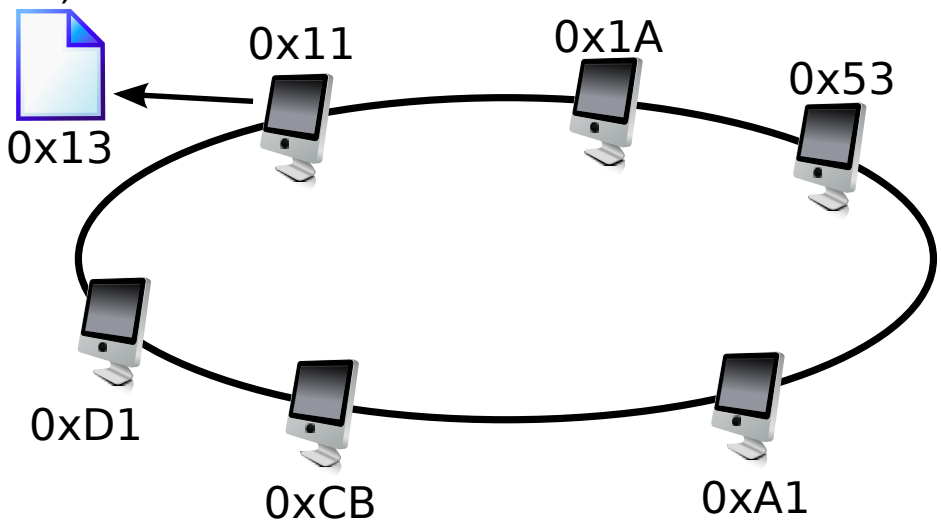
Decentralized Approaches For OSNs

HEINZ NIXDORF INSTITUTE
University of Paderborn

Peer-to-Peer based OSNs address drawbacks

DHT-based solution (e.g. PeerSoN, LifeSocial)

- users build a structured P2P overlay
- key-based routing
- DHT: get, put
- joining and leaving
- replication



1 Introduction

2 Distributed List Concept

3 Access Control in Distributed List
- Read and Write Access
- Key Distribution

4 Evaluation

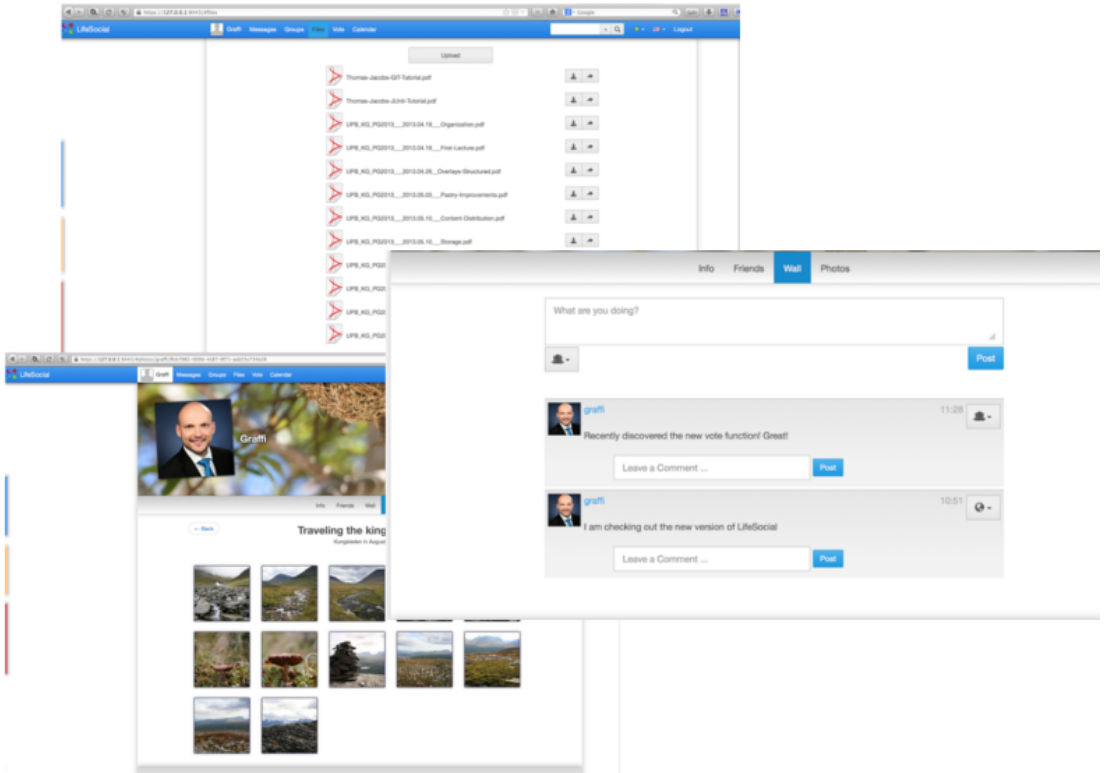
Motivation For Distributed Data Structures

HEINZ NIXDORF INSTITUTE
University of Paderborn

DHT operations (get, put) on single items do not match OSN applications

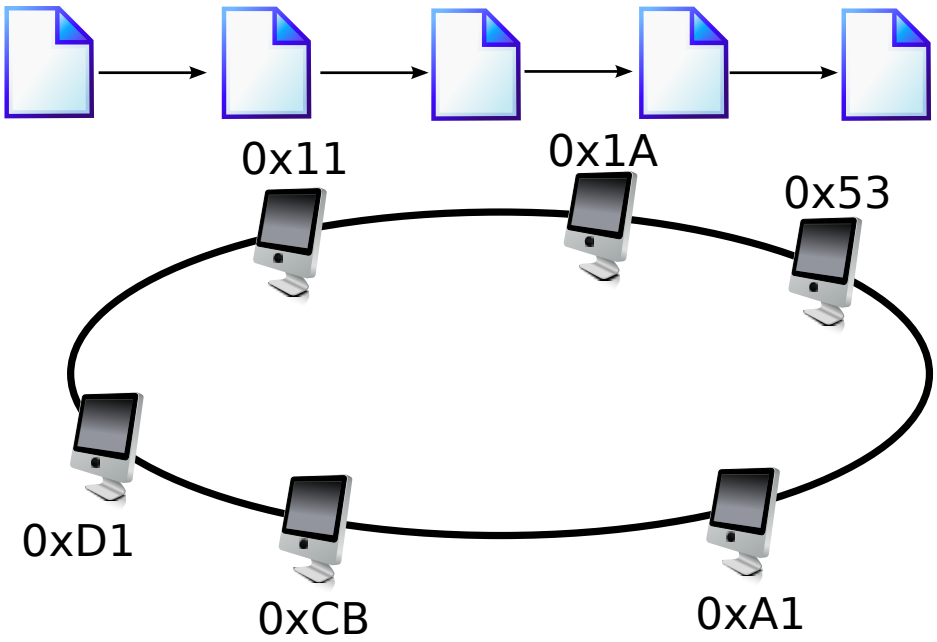
Functionalities operate on collection of items

- guestbooks/ wall entries
- photo albums
- message history, . . .



Storage Organization

HEINZ NIXDORF INSTITUTE
University of Paderborn

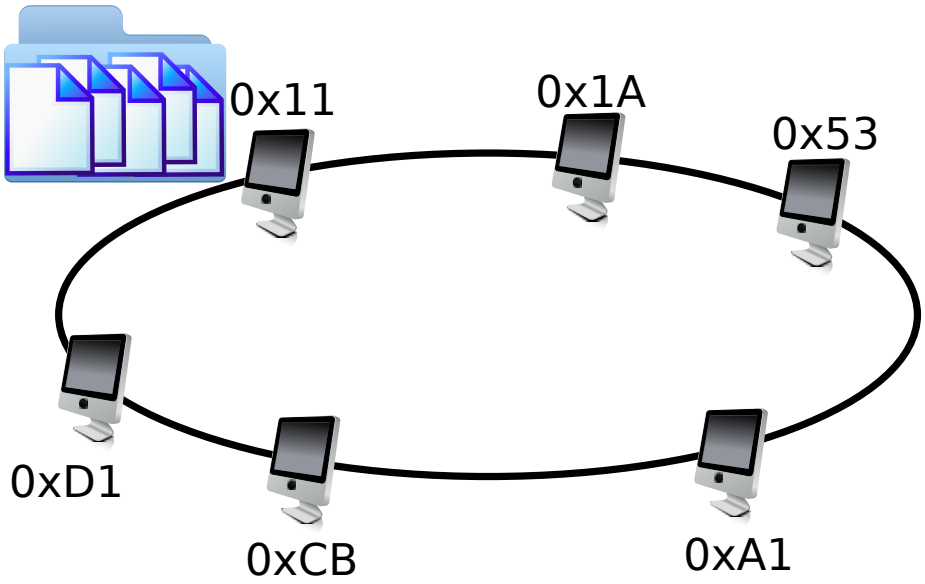


Storage Organization

HEINZ NIXDORF INSTITUTE
University of Paderborn

A) List ≙ DHT item

- \+ single `get` to retrieve list
- \- overloaded peers
- \- no parallelization

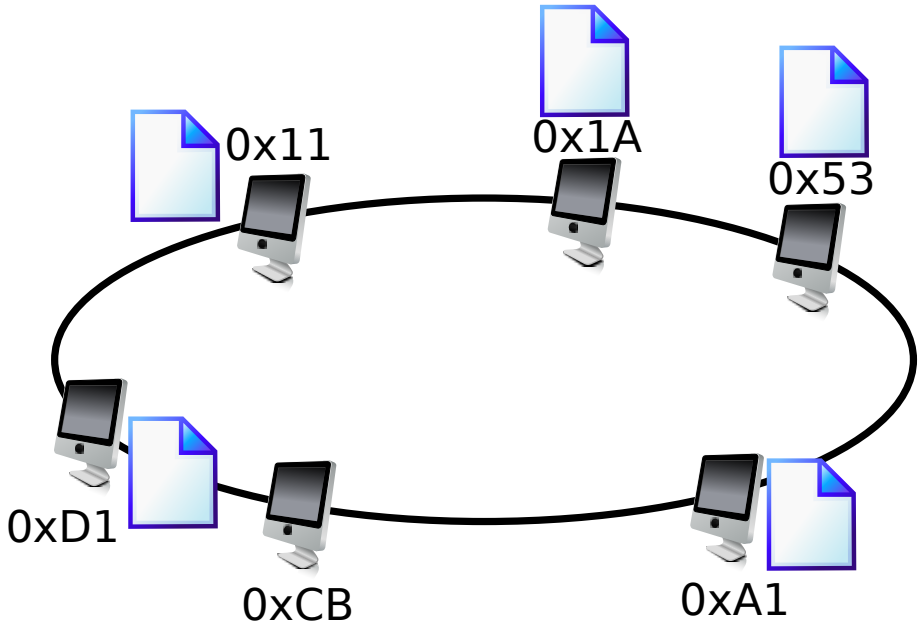


Storage Organization

HEINZ NIXDORF INSTITUTE
University of Paderborn

A) List ≙ DHT item
   - \+ single `get` to retrieve list
   - \- overloaded peers
   - \- no parallelization

B) List element ≙ DHT item
   - \+ parallelization
   - \+ overloading less probable
   - \- many messages

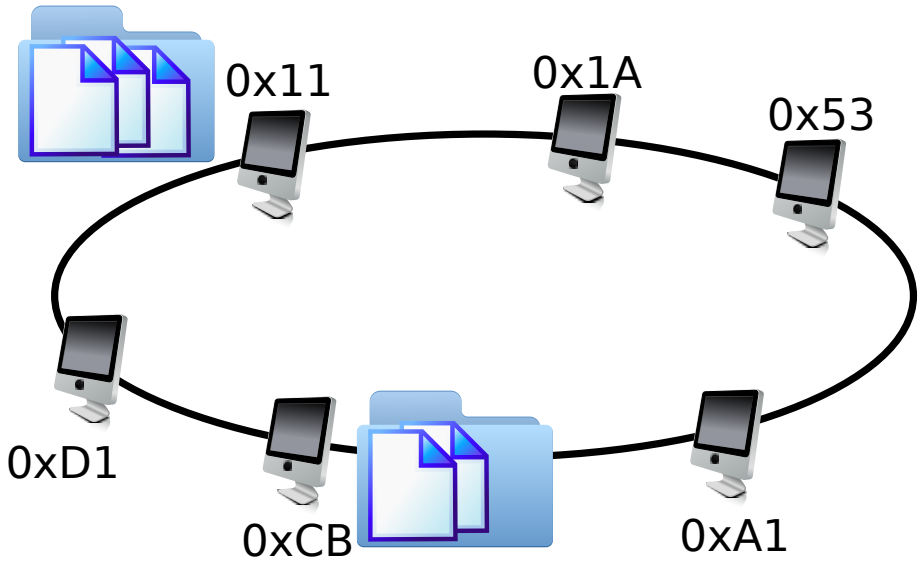


Storage Organization

HEINZ NIXDORF INSTITUTE
University of Paderborn

A) List ≙ DHT item
- \+ single `get` to retrieve list
- \- overloaded peers
- \- no parallelization

B) List element ≙ DHT item
- \+ parallelization
- \+ overloading less probable
- \- many messages



C) Partition list into buckets, store buckets in DHT
- \+ splitsize gives tradeoff between A) and B)
- i-th element has id=hash(listname + $\lfloor i/splitsize \rfloor$)

Remote Operations

Operations on list by put/get functionalities inefficient

- whole buckets are sent through network, e.g.,
- `set(i)`: retrieve bucket, send back bucket
- `contains(item)`: retrieve (several) bucket(s)

Remote Operations

Operations on list by put/get functionalities inefficient

- whole buckets are sent through network, e.g.,
- `set(i)`: retrieve bucket, send back bucket
- `contains(item)`: retrieve (several) bucket(s)

Introduce *Remote Operations*

- use `lookup` function to issue commands
- message contains request, possibly some data
- + less traffic

1 Introduction

2 Distributed List Concept

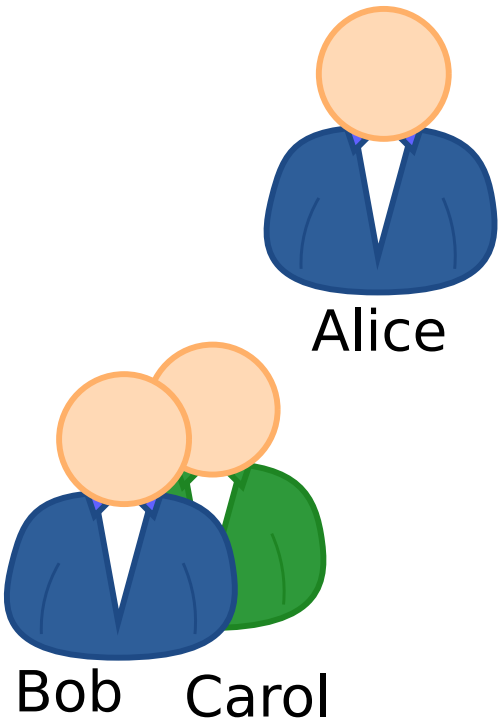
3 Access Control in Distributed List
- Read and Write Access
- Key Distribution

4 Evaluation

Example

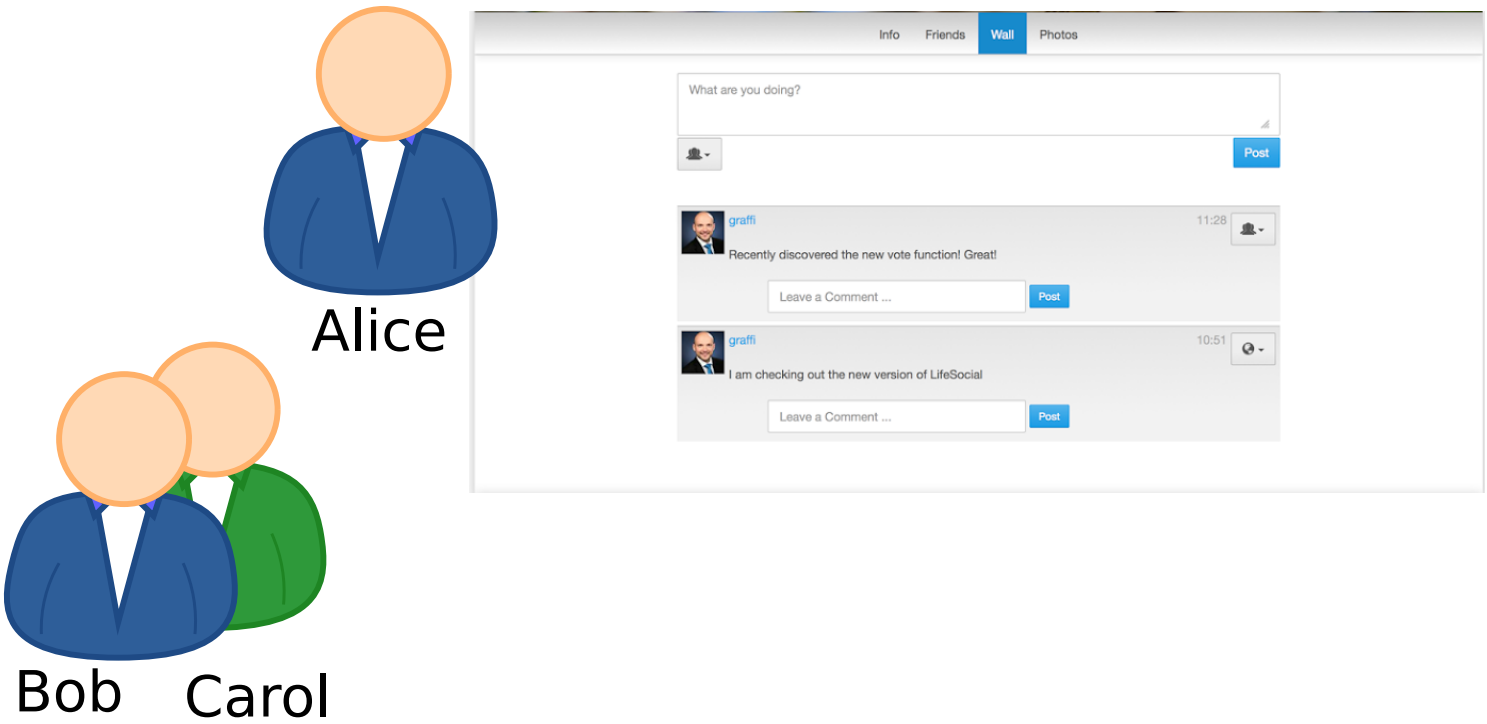
HEINZ NIXDORF INSTITUTE
University of Paderborn

- User Alice has friends Bob and Carol



Example

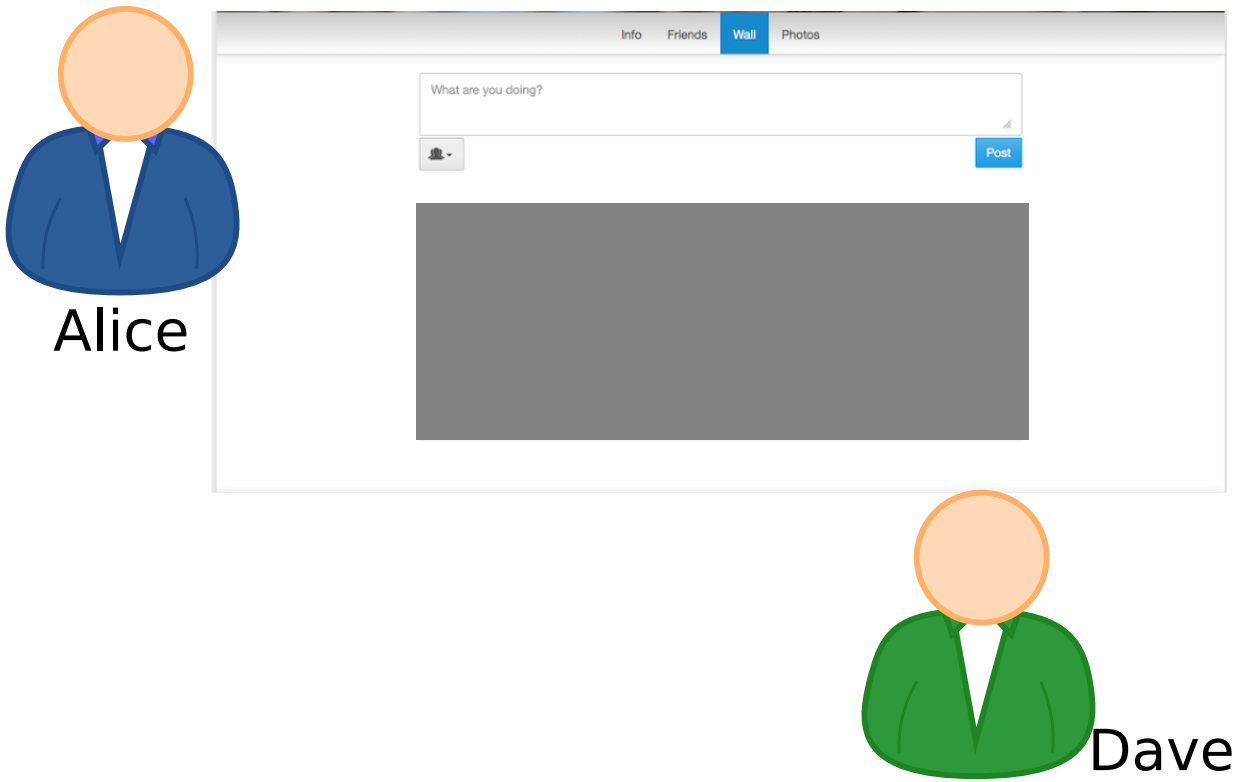
- User Alice has friends Bob and Carol
- Alice has guestbook/ wall for friends (distributed list)



Example

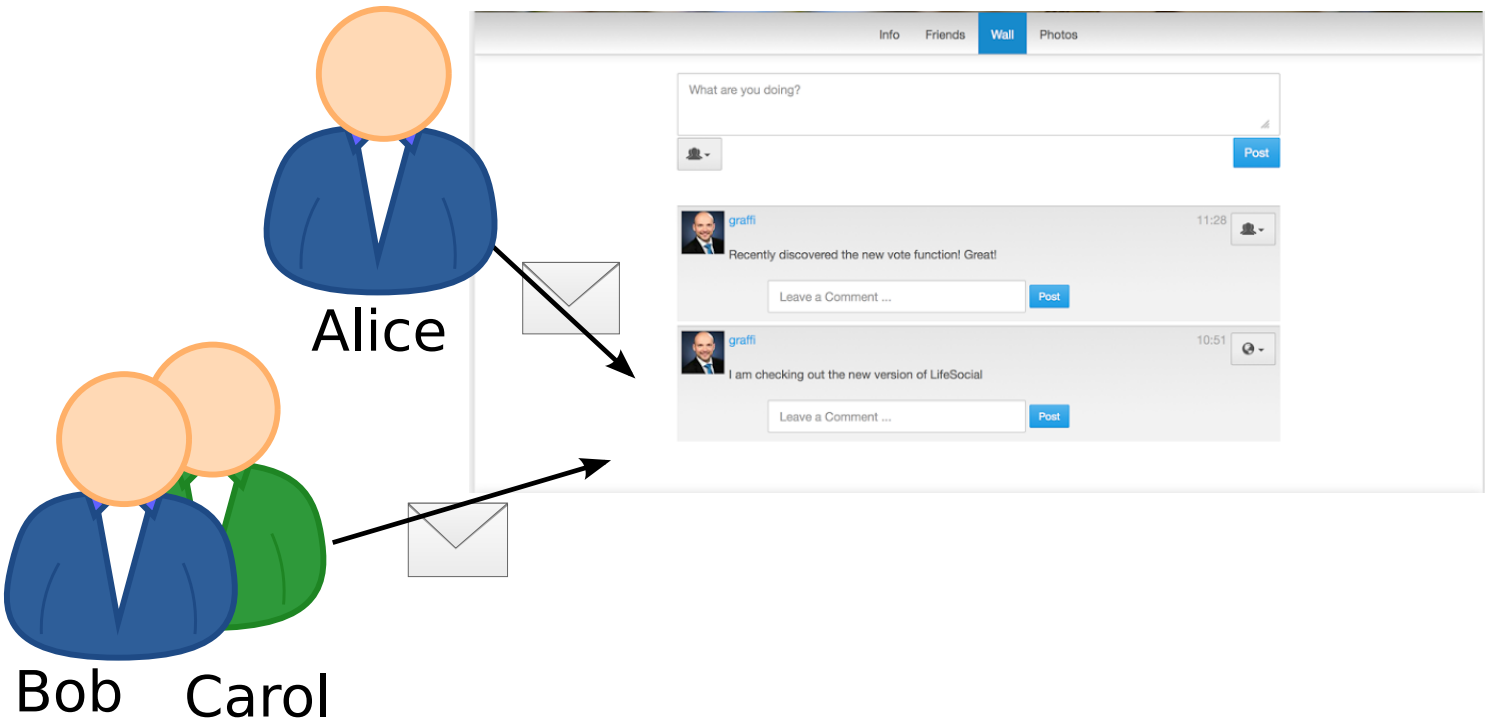
HEINZ NIXDORF INSTITUTE
University of Paderborn

- User Alice has friends Bob and Carol
- Alice has guestbook/ wall for friends (distributed list)
- Only Alice and friends can read wall



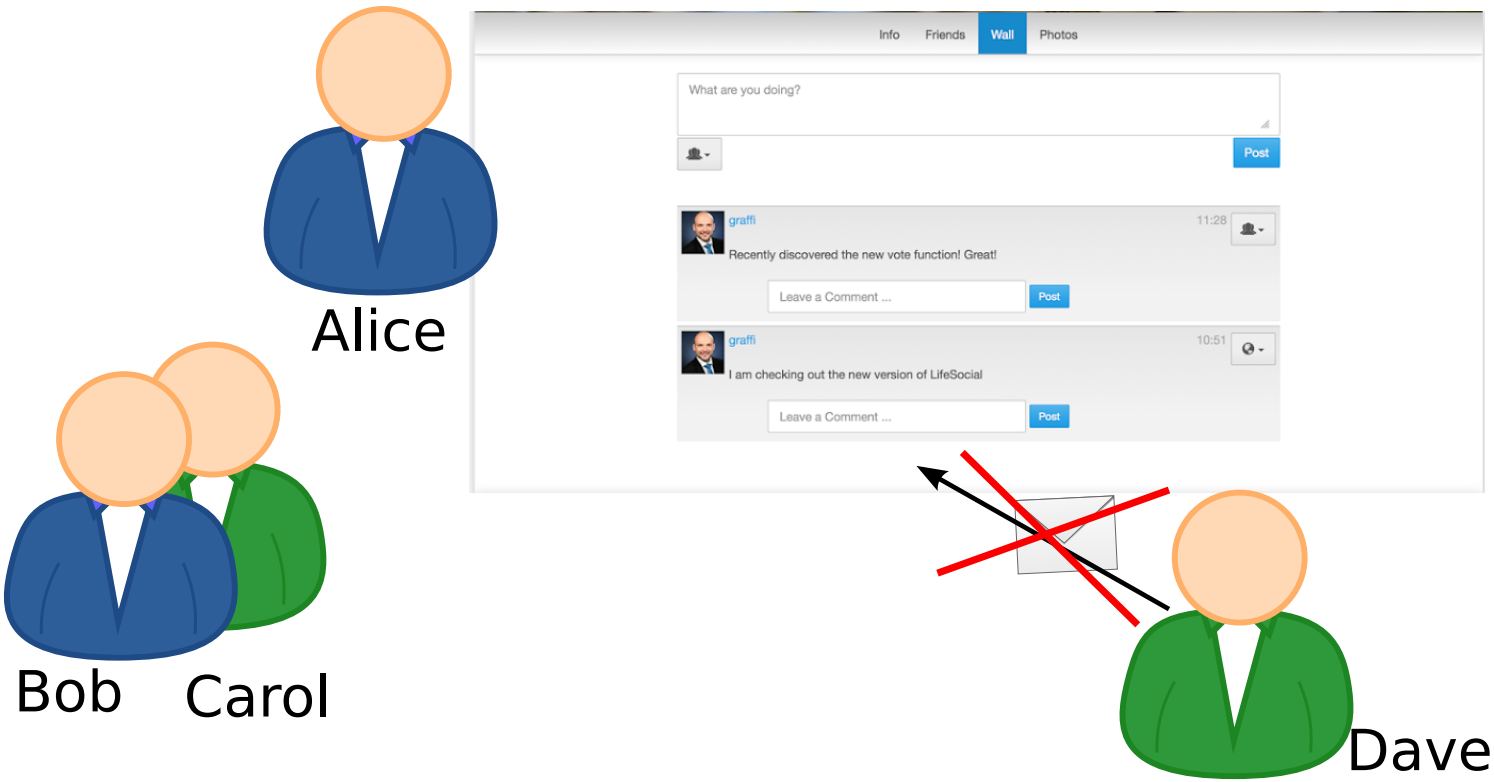
Example

- User Alice has friends Bob and Carol
- Alice has guestbook/ wall for friends (distributed list)
- Only Alice and friends can read wall
- Only Alice and friends may create new entries



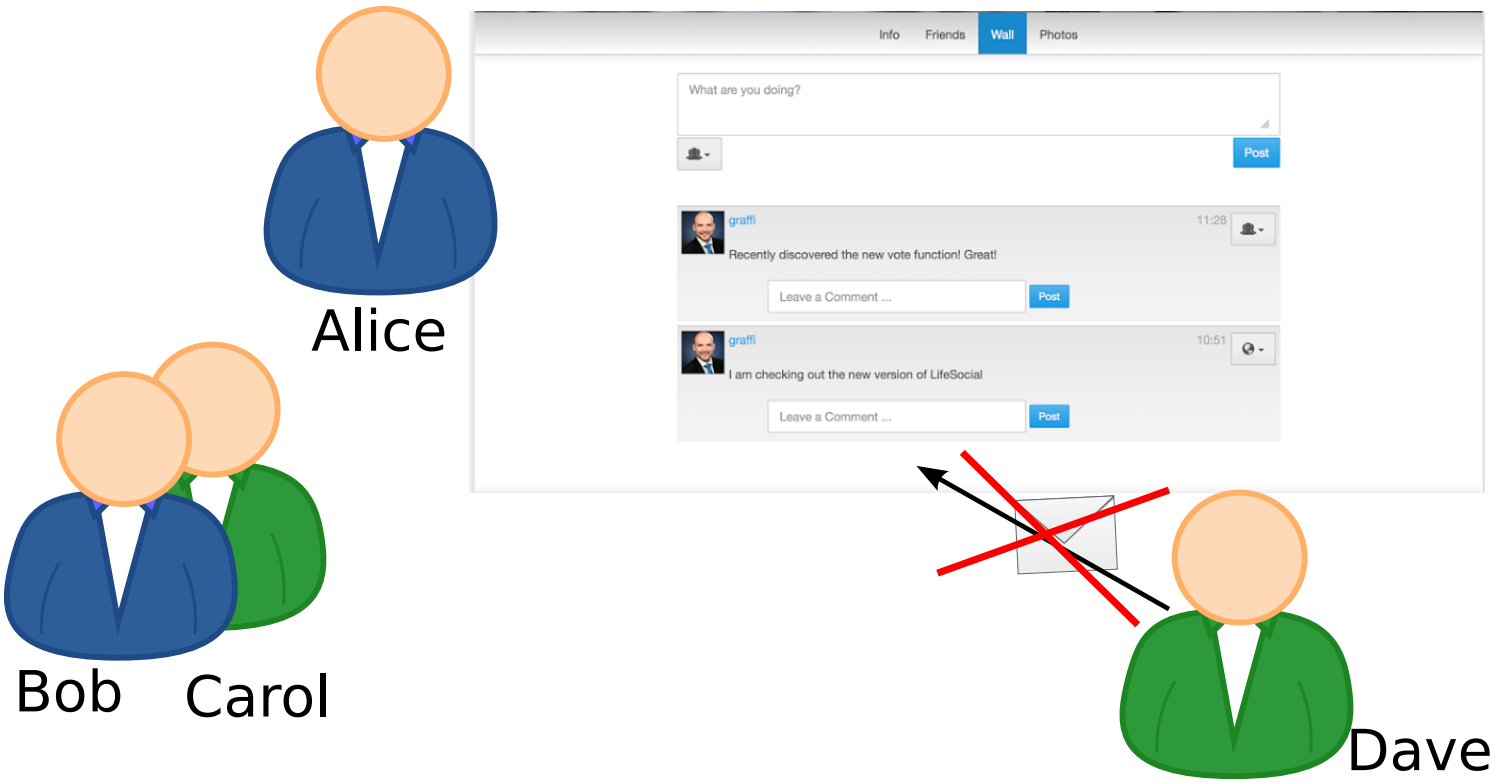
Example

- User Alice has friends Bob and Carol
- Alice has guestbook/ wall for friends (distributed list)
- Only Alice and friends can read wall
- Only Alice and friends may create new entries



Example

- User Alice has friends Bob and Carol
- Alice has guestbook/ wall for friends (distributed list)
- Only Alice and friends can read wall
- Only Alice and friends may create new entries
- Only author can modify existing entry



Using Cryptographic Means



HEINZ NIXDORF INSTITUTE
University of Paderborn

Restrict read access

- encrypt elements with common symmetric key

Restrict changing elements

Using Cryptographic Means

HEINZ NIXDORF INSTITUTE
University of Paderborn

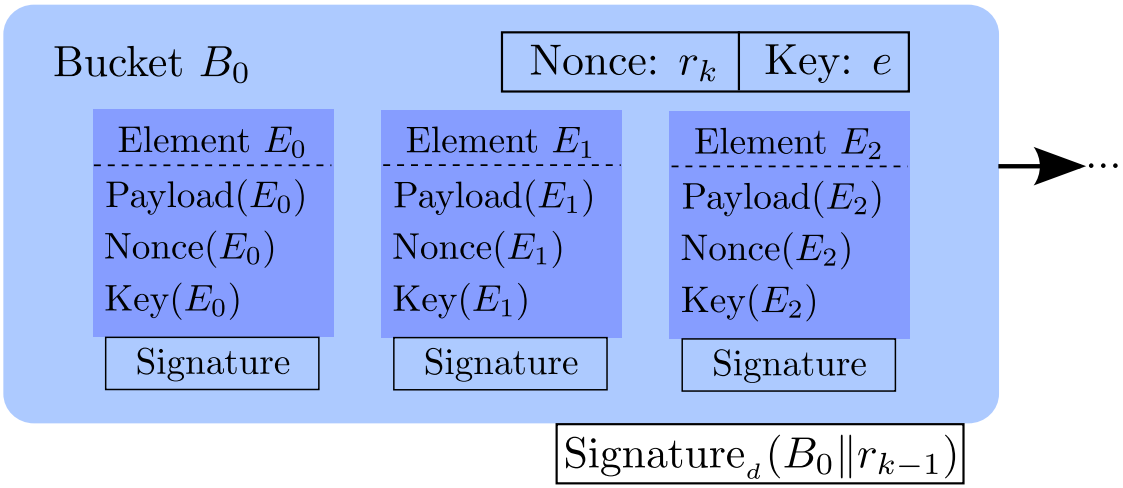
Restrict read access

- encrypt elements with common symmetric key

Restrict changing elements

- sign elements with author's private key (+nonce)
- storing peer verifies signatures

Restrict adding elements



Using Cryptographic Means

HEINZ NIXDORF INSTITUTE
University of Paderborn

Restrict read access

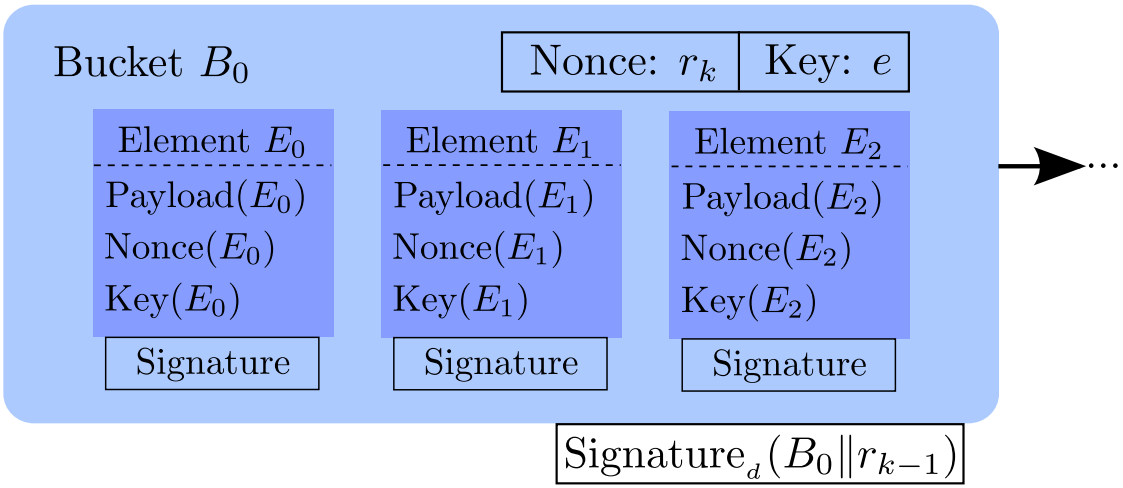
- encrypt elements with common symmetric key

Restrict changing elements

- sign elements with author's private key (+nonce)
- storing peer verifies signatures

Restrict adding elements

- bucket signed with common key of Alice and friends



1 Introduction

2 Distributed List Concept

3 Access Control in Distributed List
- Read and Write Access
- Key Distribution

4 Evaluation

Key Distribution by Groups

Bootstrap security (Graffi et al. IEEE LCN 2009)

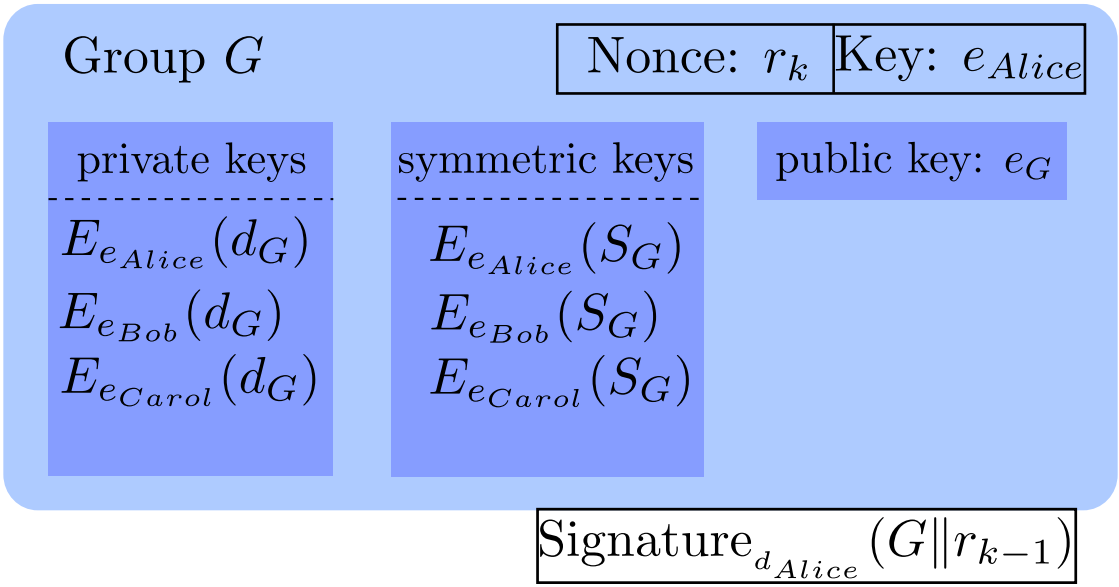
- derive private key from user name and password
- derive public key $=$ identifier of user

Key Distribution by Groups

HEINZ NIXDORF INSTITUTE
University of Paderborn

Bootstrap security (Graffi et al. IEEE LCN 2009)

- derive private key from user name and password
- derive public key = identifier of user



Create and store a group item

- asymmetric key pair (e_G, d_G), symmetric key S_G
- stored encrypted for each member

1 Introduction

2 Distributed List Concept

3 Access Control in Distributed List
- Read and Write Access
- Key Distribution

4 Evaluation

Impact on Traffic

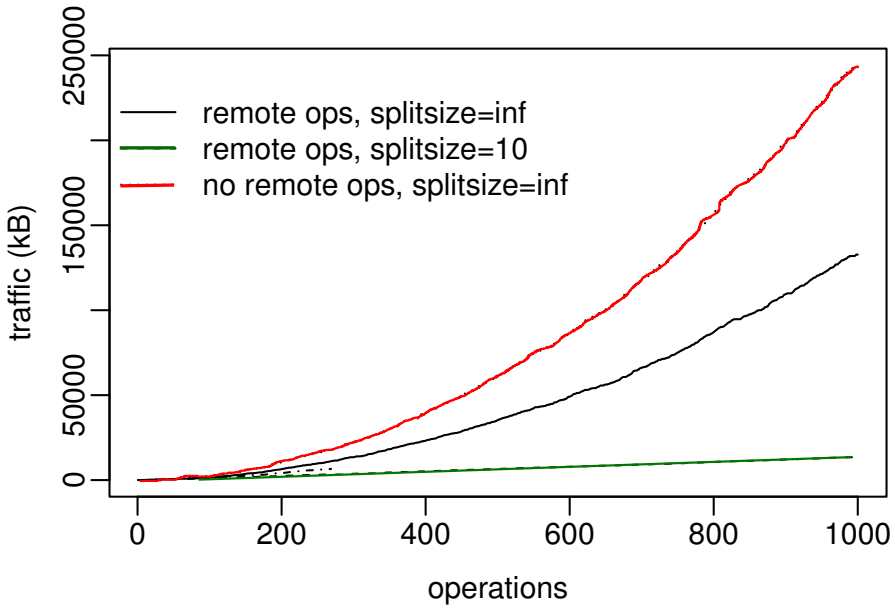
- Simulation of 1000 list operations
- each bucket stored at different node
- elements of size 1 kB

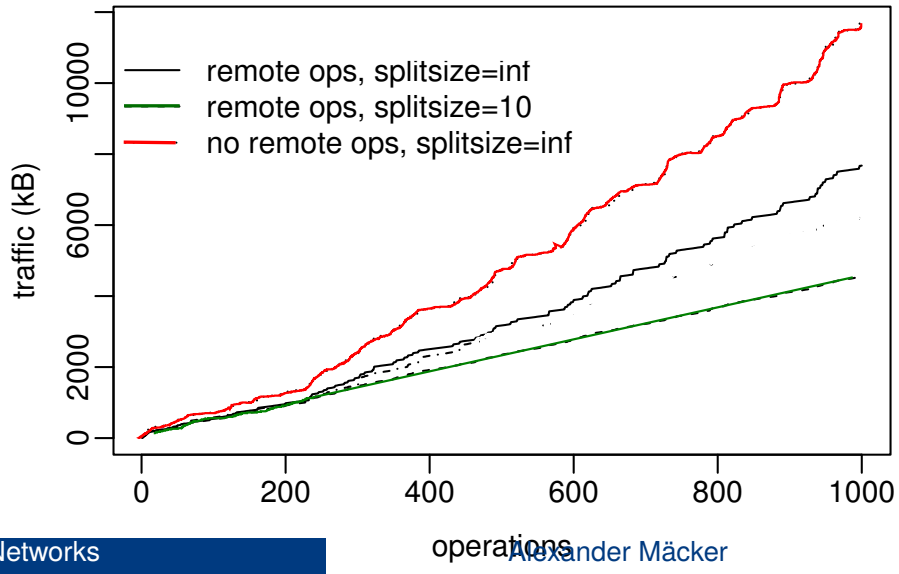
Operation	*A*	*B*
`get(`*i*`)`	0.5	0.93
`add(`*item*`)`	0.375	0.053
`remove(`*i*`)`	0.125	0.017

Impact on Traffic

- Simulation of 1000 list operations
- each bucket stored at different node
- elements of size 1 kB
- traffic considerably reduced

Operation	*A*	*B*
`get(`*i*`)`	0.5	0.93
`add(`*item*`)`	0.375	0.053
`remove(`*i*`)`	0.125	0.017

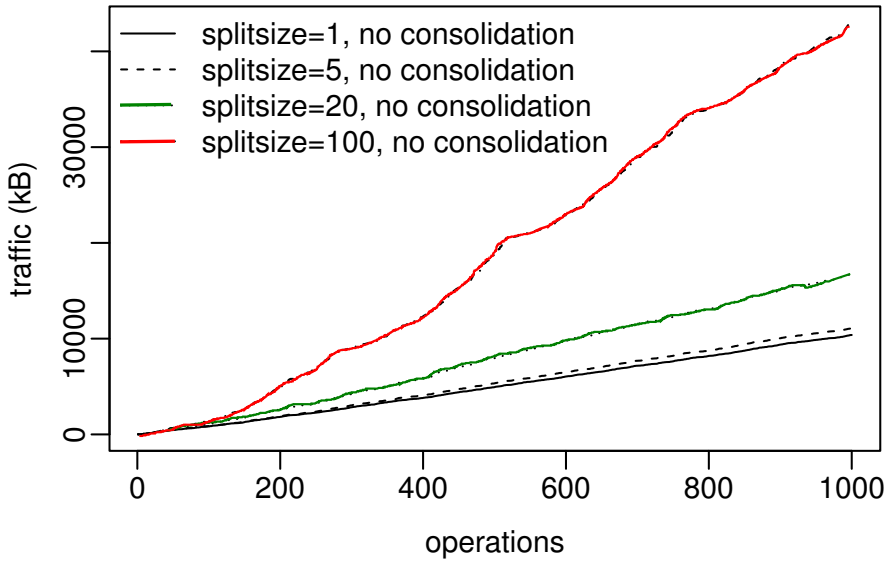


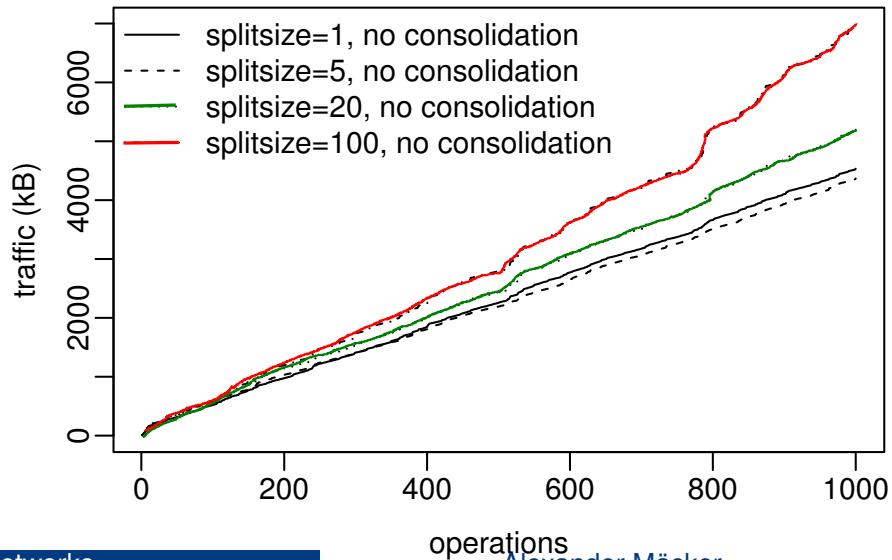


Impact on Traffic

- Simulation of 1000 list operations
- each bucket stored at different node
- elements of size 1 kB
- traffic considerably reduced
- traffic depends on splitsize

Operation	*A*	*B*
`get(`*i*`)`	0.5	0.93
`add(`*item*`)`	0.375	0.053
`remove(`*i*`)`	0.125	0.017





Summary

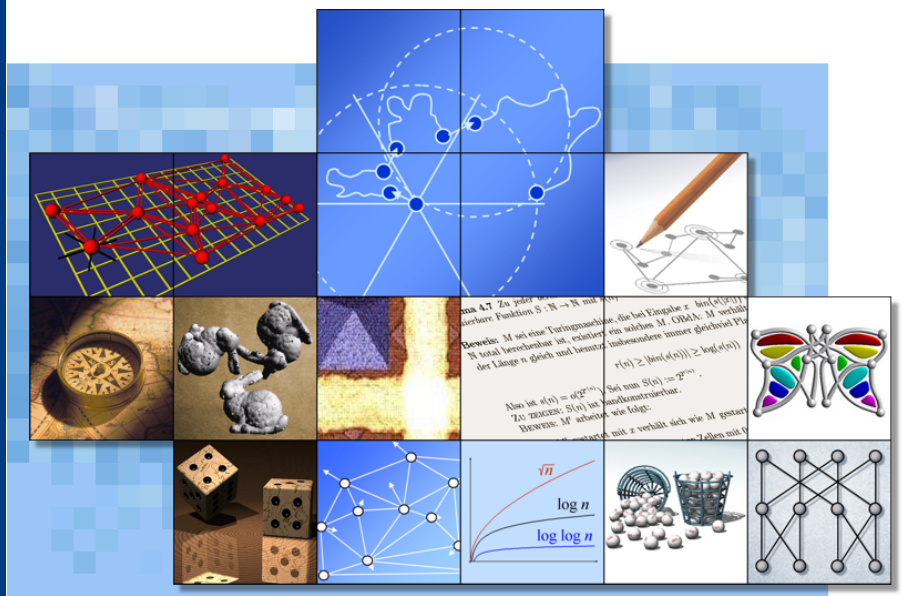
Distributed datastructures are useful for OSNs

Buckets and remote operations allow flexible, efficient list functionalities

Access control fundamental in OSNs

Cryptographic approaches and groups to control access to list

Thank you for your attention!



Alexander Mäcker

Heinz Nixdorf Institute
& Department of Computer Science
University of Paderborn

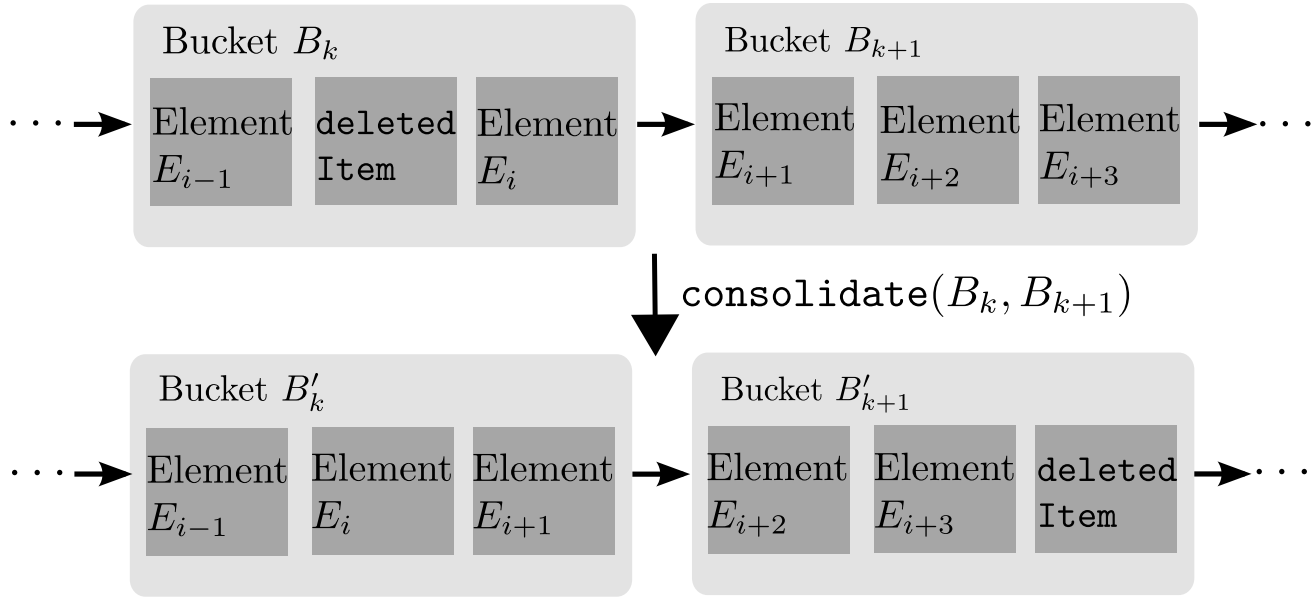
Address: Fürstenallee 11
33102 Paderborn
Germany

E-mail: amaecker@upb.de
Web: http://www.p2pframework.com

Remote Write Access

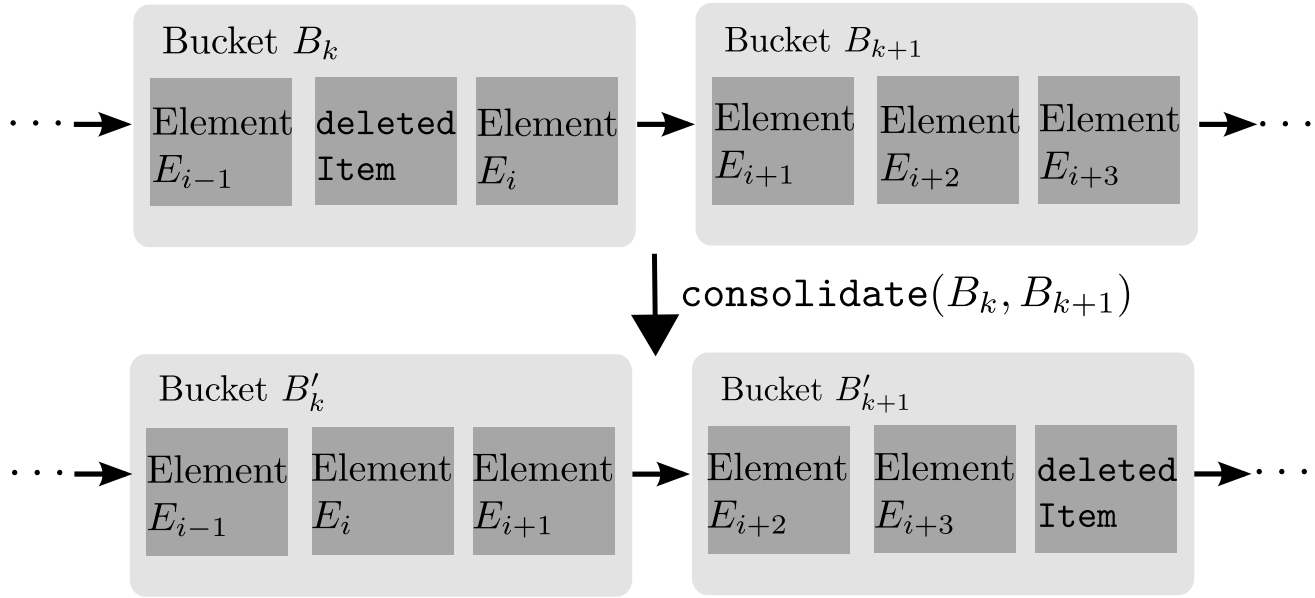
1) *C* requests item from S; possibly sends hash of its own version of item
2) *S* replies with current version of item or ACK
3) *C* performs following steps
   - perform change locally
   - compute signature
   - send back signature, command and old hash
4) *C* compares hashes; either performs changes locally and stores or back to step 2

Consolidation of Distributed List



1) Initiating node computes B'_k and B'_{k+1} and signatures $Sig(B'_k)$, $Sig(B'_{k+1})$. Signatures sent to nodes S_k and S_{k+1} storing B_k and B_{k+1}.

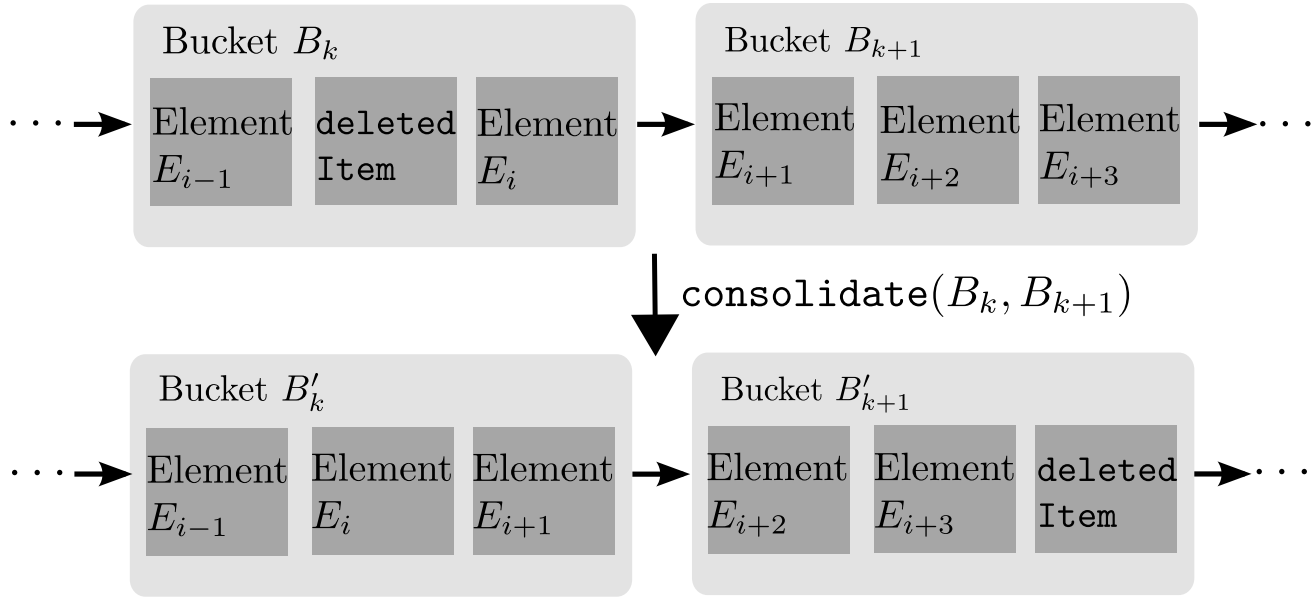
Consolidation of Distributed List



1) Initiating node computes B'_k and B'_{k+1} and signatures $Sig(B'_k)$, $Sig(B'_{k+1})$. Signatures sent to nodes S_k and S_{k+1} storing B_k and B_{k+1}.

2) S_k and S_{k+1} compute B'_k and B'_{k+1}. Cancel if differ from received ones.

Consolidation of Distributed List



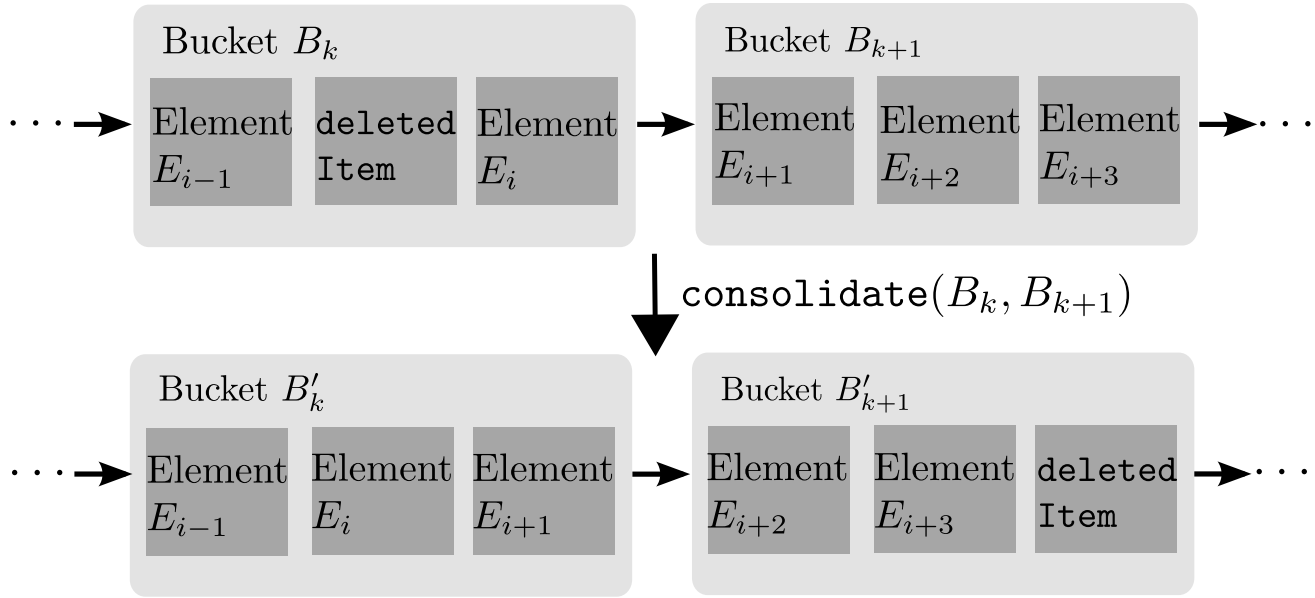
1) Initiating node computes B'_k and B'_{k+1} and signatures $Sig(B'_k)$, $Sig(B'_{k+1})$. Signatures sent to nodes S_k and S_{k+1} storing B_k and B_{k+1}.

2) S_k and S_{k+1} compute B'_k and B'_{k+1}. Cancel if differ from received ones.

3) S_{k+1} notifies S_k.

Consolidation of Distributed List

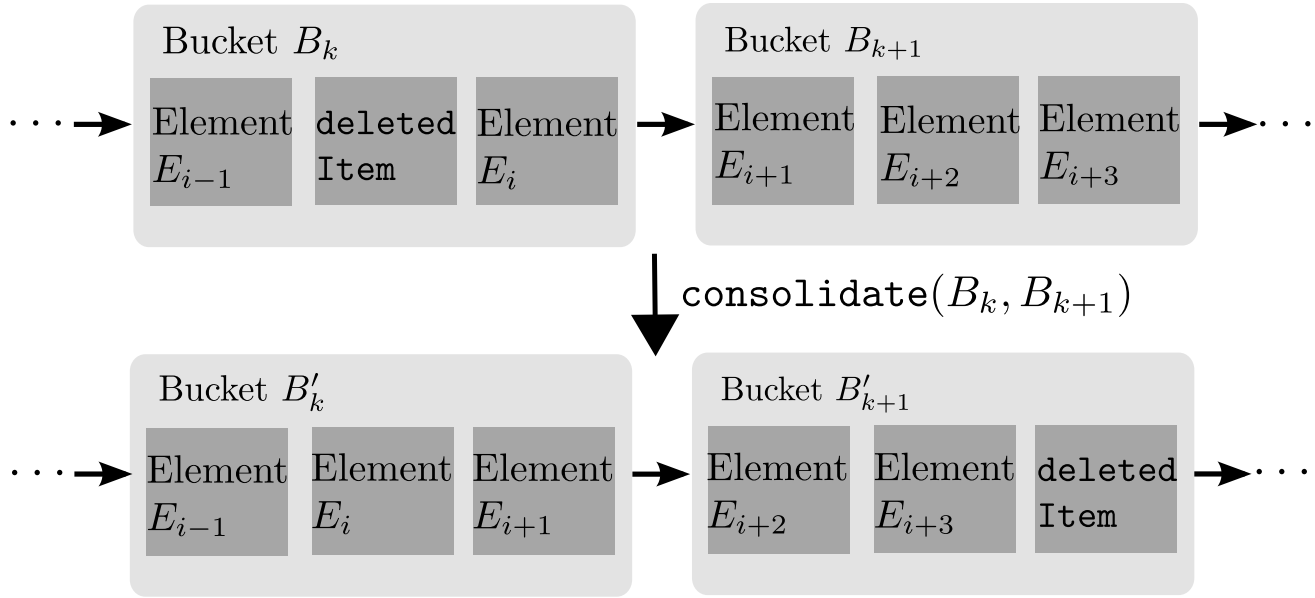
HEINZ NIXDORF INSTITUTE
University of Paderborn



1) Initiating node computes B'_k and B'_{k+1} and signatures $Sig(B'_k)$, $Sig(B'_{k+1})$. Signatures sent to nodes S_k and S_{k+1} storing B_k and B_{k+1}.
2) S_k and S_{k+1} compute B'_k and B'_{k+1}. Cancel if differ from received ones.
3) S_{k+1} notifies S_k.
4) On reception of notification, S_{k+1} stores B'_k and notifies S_{k+1}.

Consolidation of Distributed List

HEINZ NIXDORF INSTITUTE
University of Paderborn



1) Initiating node computes B'_k and B'_{k+1} and signatures $Sig(B'_k)$, $Sig(B'_{k+1})$. Signatures sent to nodes S_k and S_{k+1} storing B_k and B_{k+1}.
2) S_k and S_{k+1} compute B'_k and B'_{k+1}. Cancel if differ from received ones.
3) S_{k+1} notifies S_k.
4) On reception of notification, S_{k+1} stores B'_k and notifies S_{k+1}.
5) On reception of notification, S_k stores B'_{k+1} after checking that B'_k is stored as expected.