Shared Resources and Distributed Conferencing in RELOAD: Concept Update and Implementation Report

Alexander Knauf, Gabriel Hege
Thomas Schmidt, Matthias Wählisch

alexander.knauf@haw-hamburg.de, hege@fhtw-berlin.de,
{t.schmidt,waehlisch}@ieee.org

Outline

- **Introduction**
- **Update on Shared Resources in RELOAD**
- **Update on Distributed Conferencing in RELOAD**
- **RELOAD Implementation**

Problem Statements

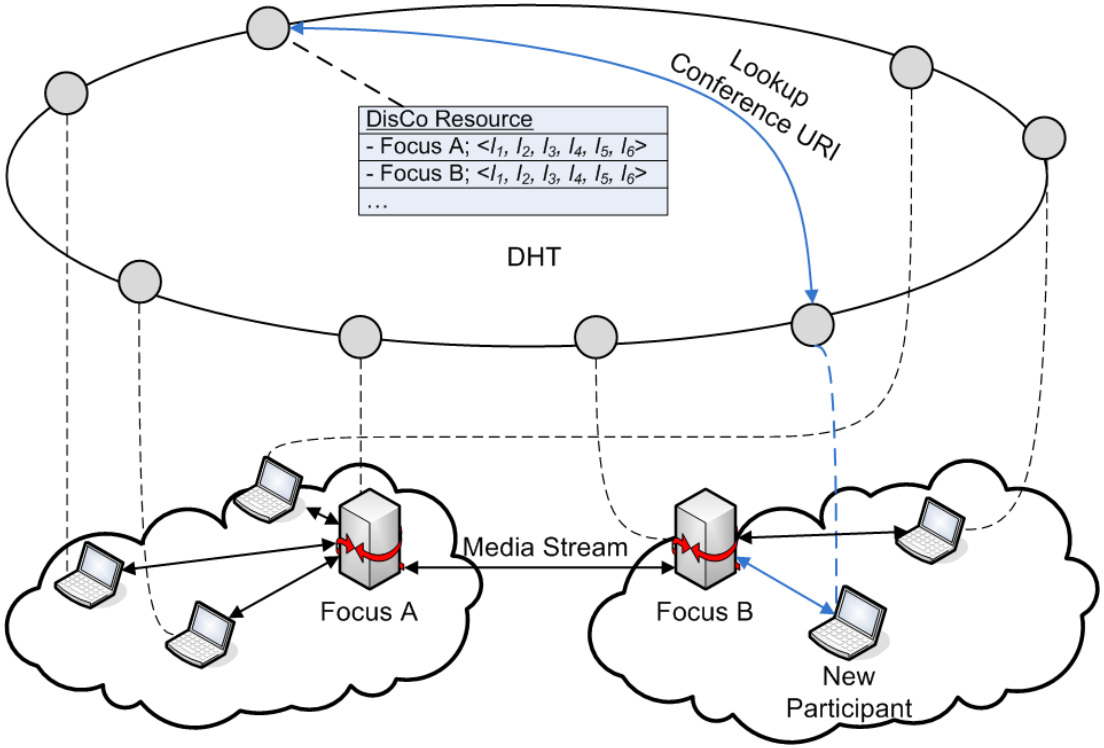
for a P2P Conferencing Approach

- Tightly coupled conferences are managed by a *single* entity called *Focus*:
  - Maintains signaling and media parameter negotiation
  - May perform media mixing functions
- **Problem (1)**: The Conference URI
  - *Identifies* the multiparty session, but
  - *locates* the conference focus
    - Single point of failure
- **Problem (2):** No dedicated server architecture in P2PSIP
  - Media mixing performed at the end-user devices
    - Scaling problem within large conferences
  - Conference must be registered and globally accessible
    - Demands a registrar or conference factory

Distributed Conference Control

an Overview

- A Distributed Conference (**DisCo**) is a multiparty session in a tightly coupled model that is controlled by several independent entities called **Focus Peers**



Separating ID and Locator

of a Conference URI

- Conference URI stored in a RELOAD overlay as *key* to several *Focus Peers* that manage a single conference
- Interested users resolve URI using RELOAD fetch:
  - Returns several contact addresses of focus peers and the relative network coordinates
  - User application chooses the closest focus to join the conference
- **Focus Peers …**
  - … participate of the conference they manage
  - … synchronize conference state via an XML document
  - … perform load balancing by transferring participants

Motivation for Shared Resources

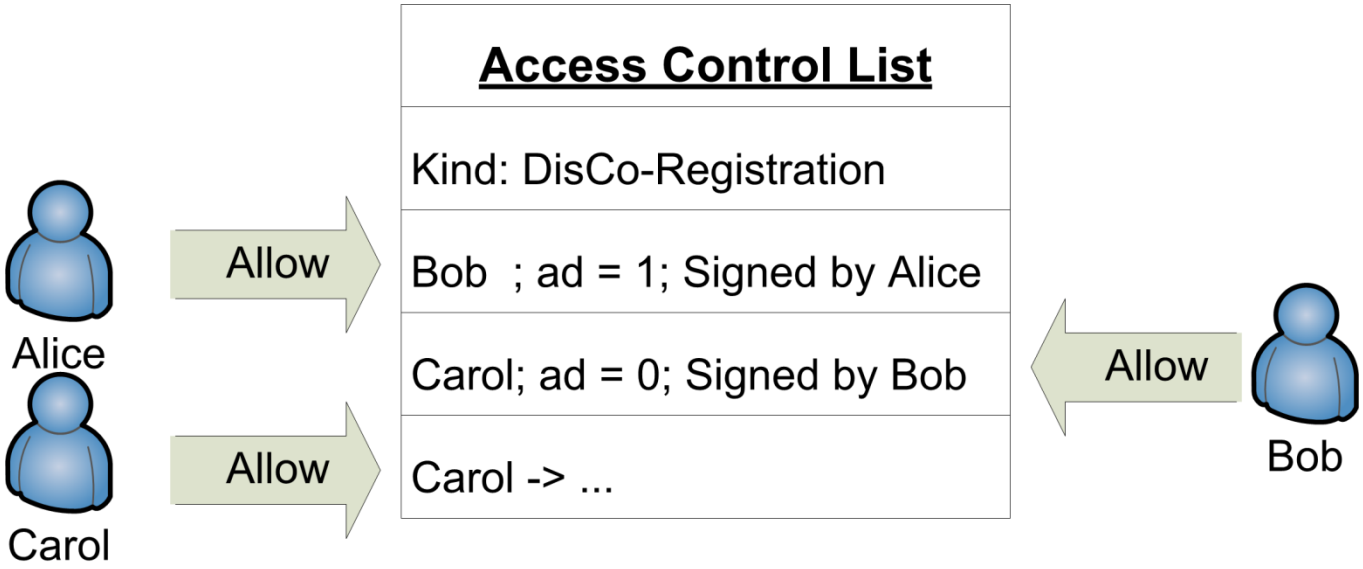
in RELOAD

- **Initial Problem:** Restrictive access control in RELOAD
  - Users have exclusive write access to few overlay locations related to their public key certificate
  - No mechanism to allow a third party write access to a overlay resource
- **Wanted:** A generic mechanism to share resources
  - Applicable for a variety of usages
  - Access control and revocation mechanisms
  - Optional: Flexible naming of overlay resources

Access Control Lists

to share writing permissions

- **Proposal:** Sharing resources by Access Control Lists
  - A resource to be shared is stored along with an additional Access Control List (ACL)
  - ACL contains a list of overlay users explicitly allowed to store data in the shared resource



Update on

Shared Resources in RELOAD

Isolated Data Storage

avoiding race conditions

- **Problem:** Concurrent store requests on Shared Resources can cause race conditions
- **Proposal since -01:** Mechanism for isolating stored data
  - **Case 1:** Shared Resource uses dictionary data model
    - Dictionary key MUST be equal to signers Node-ID
  - **Case 2:** Shared Resource uses array data model
    - Array indexes are a concatenation of the least significant 24 bits of the signers Node-ID + an 8 bit individual short
    - Technique related to SSRC identifier generation in RTP (RFC3550)
  - **Case 3:** Shared Resource is a single value
    - Not allowed

Plain Resource Name in Kinds (1)

Needed to validate variable resource names

- **Initial Problem:** Resource names not available for receiver of a stored data
  - But needed for validating *variable* resource names
- **First Solution in -00:** Preceding resource name field
  - Kinds using ShaRe's USER-CHAIN-ACL access control policy MUST contain the resource name

```
struct {
    opaque resource_name<0..2^16-1>
    /* Kind data */
} AnyKind
```

  - But redundant if a Kind is stored under the AoR of a peer

Plain Resource Name in Kinds (2)

The ResouceNameExtension field

- **Proposal in -02:** Optional *ResourceNameExtension* struct
  - Extendable structure containing the resource name
  - Precedes Kind data only if indicated in corresponding <kind-block> in configuration document

```
struct {
 ResourceNameType type;
 uint16           length;
   select(type) {
     case pattern:
     opaque resource_name<0..2^16-1>

     /* Types can be extended */
   }
}ResourceNameExtension
```

```
<kind-block>
<!-- other elements -->
 <share:variable-resource-names
  enable="true">
  <pattern>
    $USER-[0-9]@$DOMAIN
  </pattern>
 </share:variable-resource-names>
 </kind>
</kind-block>
```

Changes – ShaRe Kind

according to P2PSIP WG Feedback

- **Initial Approach:** Shared resources contained the username their creator in Kind data structure
  - Used to validate if a storing peer is listed in the corresponding ACL to a shared resource
- **Removed in -02:** Originator identified in Signature object

```
struct {
  /* res_name_ext is optional, see documentation */
  ResourceNameExtension  res_name_ext;
  opaque                 to_user<0..2^16-1>;
  KindId                 kind;
  Boolean                allow_delegation;
} AccessControlListItem;
```

Update on

Distributed Conference Control

DisCo Kind Changes

following new ShaRe requirements

- Removed redundant `user_name` field
- Simplified and Updated `DisCo-Registration` struct according to ShaRe requirements

```
struct {
   /* This field is optional, see documentation */
  ResourceNameExtension res_name_ext;
  opaque coordinate<0..2^16-1>;
  NodeId node_id;
} DisCoRegistration;
```

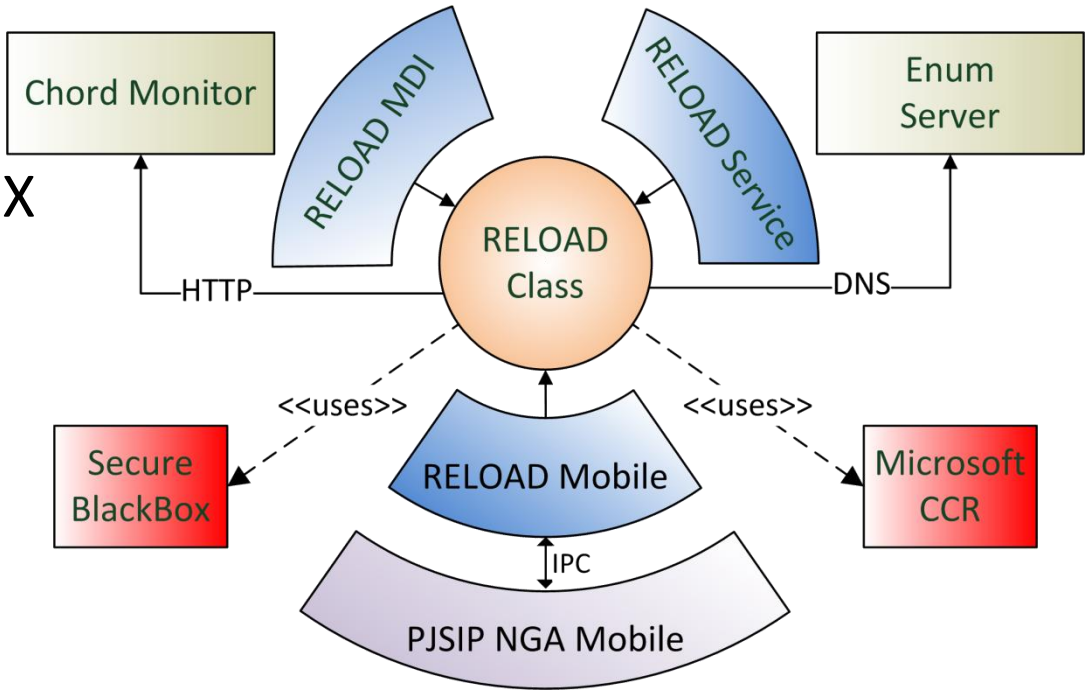
Report and Measurements on

RELOAD Implementation

Implementation

of RELOAD

- A .Net Project
- Run on:
  - Windows PC
  - Windows Mobile 6.X
  - Linux (on MONO)
- Provides:
  - Emulation
  - Monitoring
  - TCP or TLS
  - SIP calls and conferencing



RELOAD Usages

running on Stack

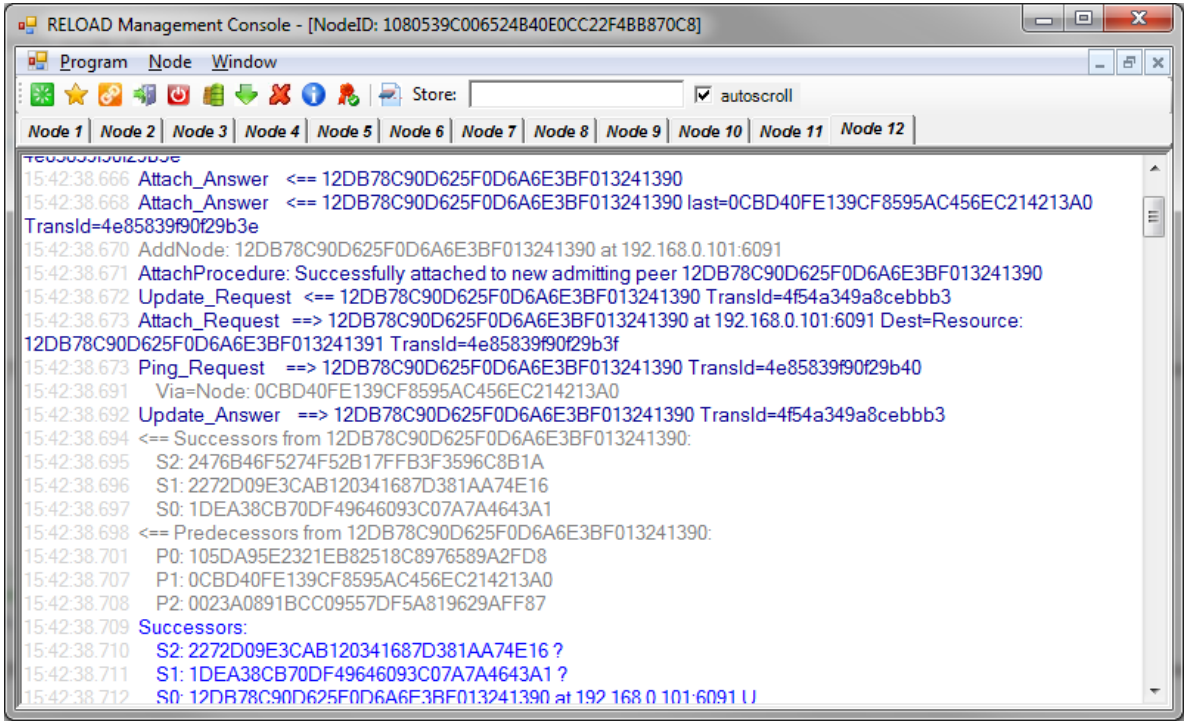
- Supports:
  - `draft-ietf-p2psip-sip`
  - `draft-knauf-p2psip-disco`
  - `draft-knauf-p2psip-share`
- Further Usages can be added to stack binary
  - C# classes need to implement an interface
  - Added to stack by a register method

    +register(usage: Usage): void

Emulation Tool(1)

RELOAD running

- GUI RELOAD emulation Tool

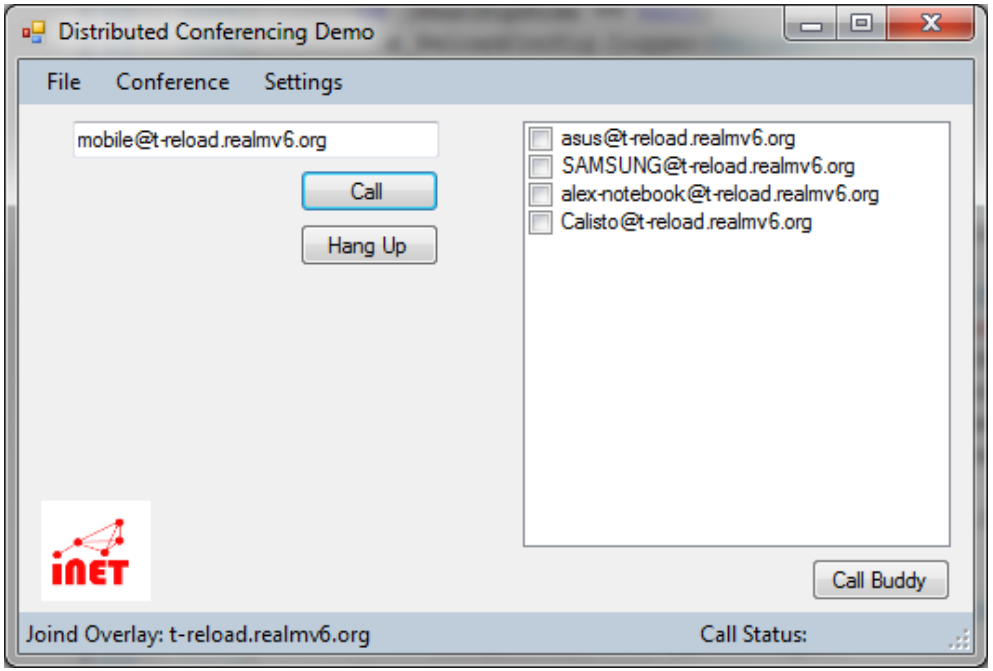


  - Instantiates *Peers* and *Clients* locally
  - Lots of debugging output

Demo Application

RELOAD running

- Simple RELOAD softphone application

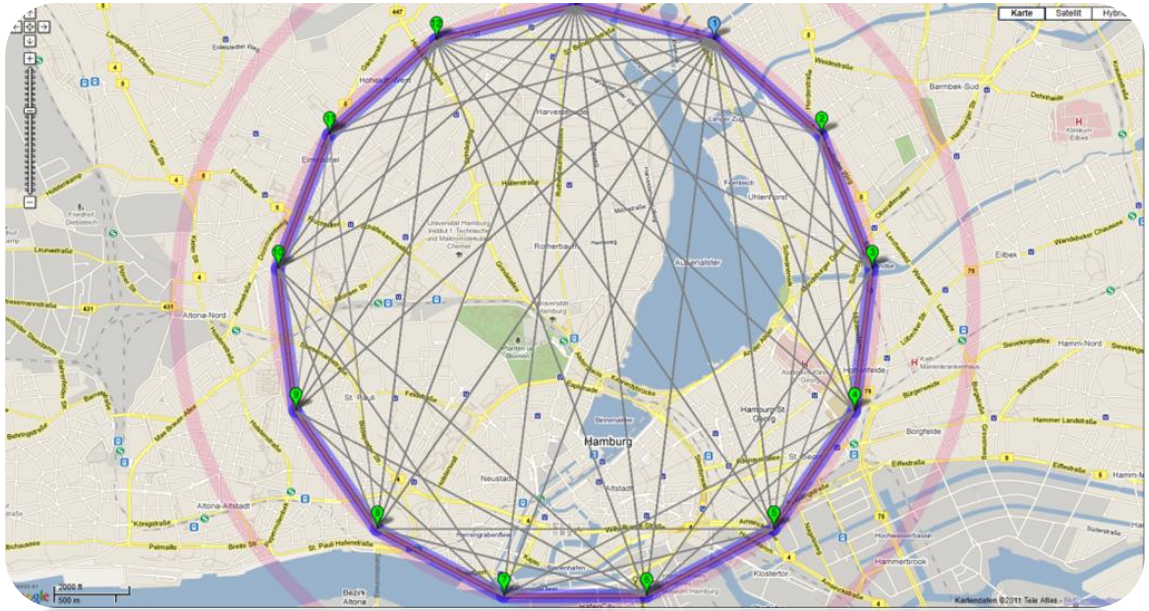


  - Supports VoIP calls and distributed conferencing
  - SIP signaling and media streams based on PJSIP stack

Monitoring

RELOAD running

- Visualizes arrangement of overlay parties
  - Based on Google Maps API
  - Configurable to visualize different aspects

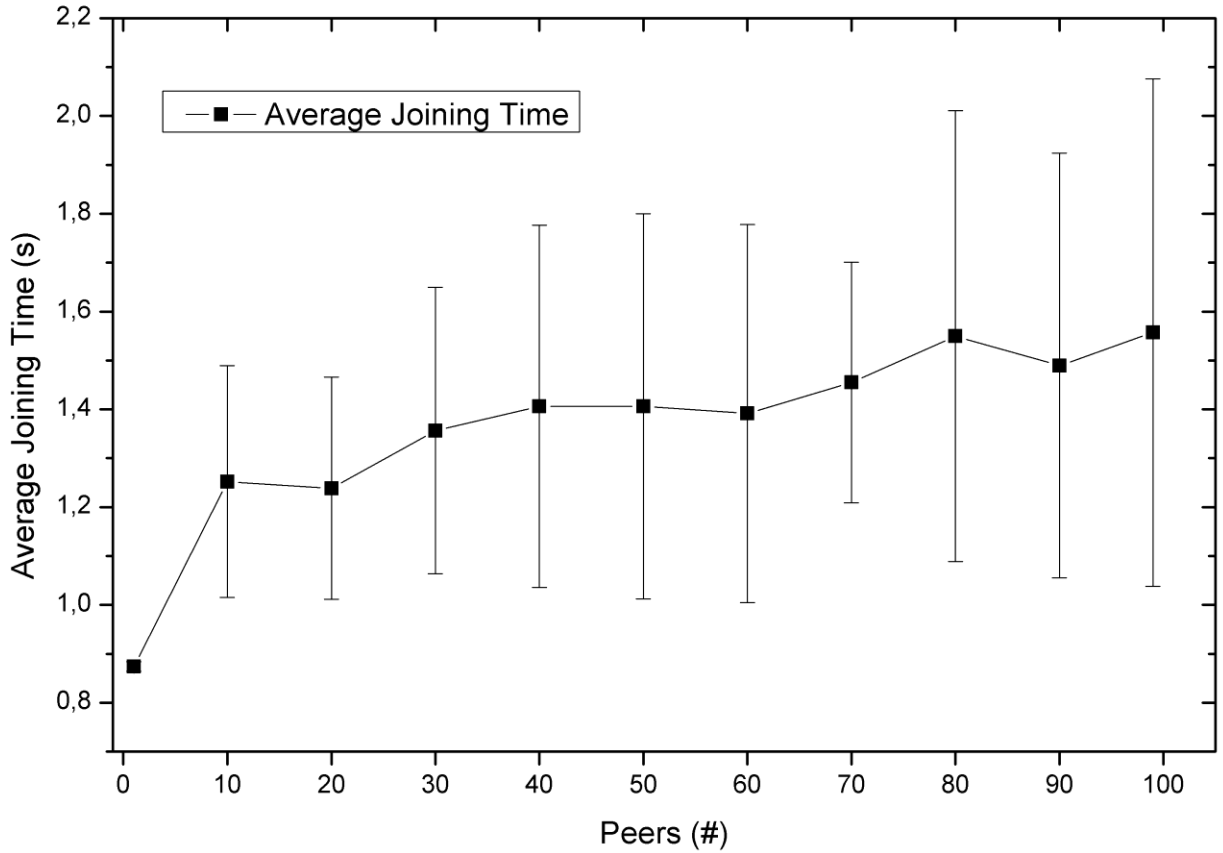


SIP Usage on Mobiles

RELOAD running

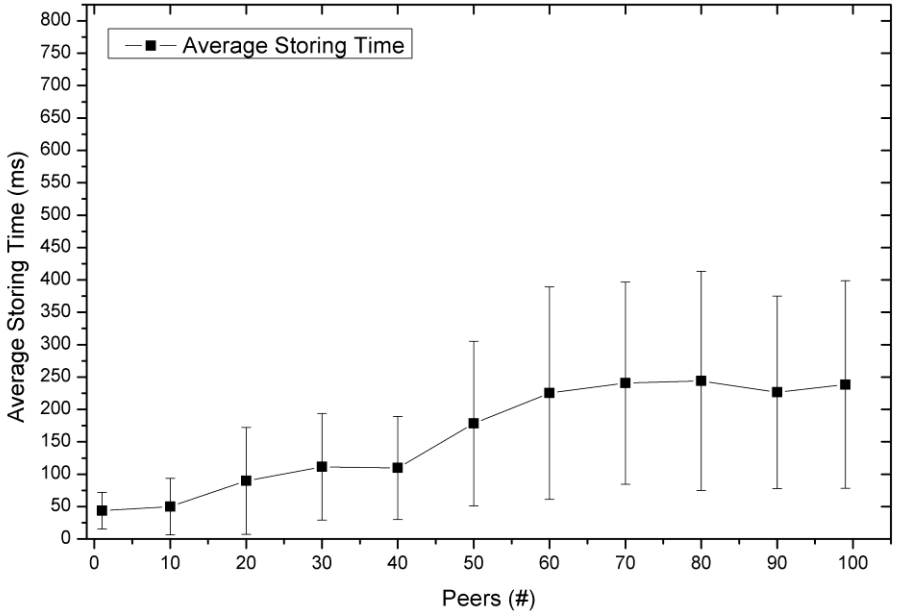
- Limited device capacities
  - ➢ Mobile join overlay as RELOAD *Clients*
- Authentication by SIM card
  - ➢ *International Mobile Subscriber Identity* (IMSI) for authentication
- Registration and lookup of mobile telephone numbers
  - ➢ Resource name = *{telephone_number}@{Domain}*
- **Problem:** Response times
  - ➢ Secure transports on mobiles may costly

Average Joining Delay

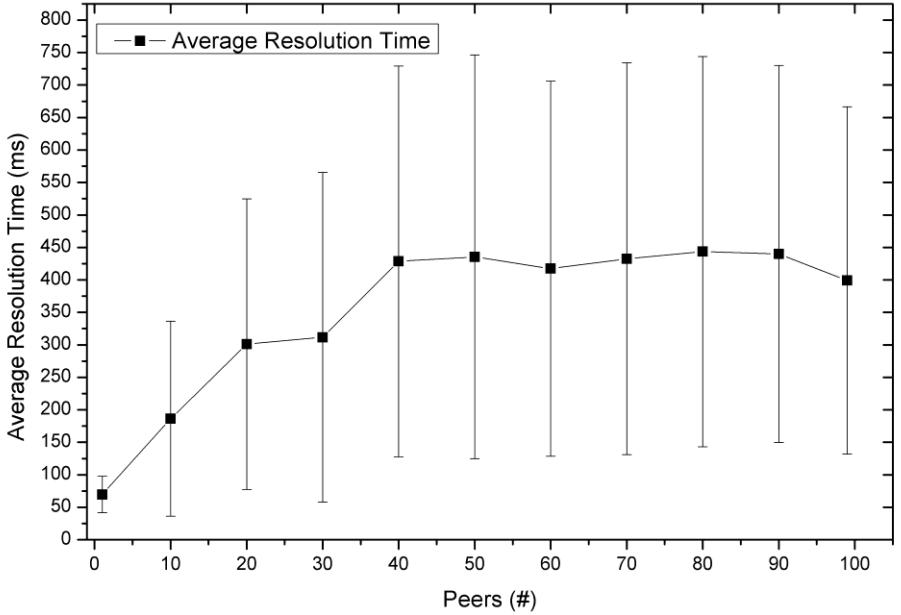


Average joining delay of RELOAD peer (incl. Enrollment)

Average Store Fetch/AppAttach Delay



Average Delay to Store a SIP Record



Average Delay to Fetch and Attach a Destination Node

Measurement Evaluation

- **Desktop Devices:**
  - Logarithmical scaling (expected using Chord)
  - Joining slight more costly
  - Delay resolving AoR to node-id > registering AoR
- **Mobile Devices:**
  - TLS connection establishment very costy
  - TLS stack probably not efficient
  - TCP delay approx. a factor 10-20 times faster

Conclusion & Outlook

- **Conclusion:**
  - Shared Resources providing variable resource names
  - Distributed Conferencing in P2PSIP
  - RELOAD implementation
- **Outlook:**
  - Soon available as Open Source
  - Further Evaluations on PlanetLab
  - DisCo/ShaRe ongoing work in the P2PSIP WG:
    - `draft-knauf-p2psip-disco`
    - `draft-knauf-p2psip-share`

Questions?

Thanks for your attention!

http://inet.cpt.haw-hamburg.de/