Secure and Resilient Peer-to-Peer Email: Design and Implementation

Jussi Kangasharju
TU Darmstadt

Keith W. Ross
Brooklyn Polytechnic

David A. Turner
CSU San Bernardino

Contribution

- Architecture for peer-to-peer email
  - Eliminates need to rely on single server
  - Boost resilience of email against attacks
  - Provides confidential communications
- Reliability analysis of P2P storage
  - Causes of unavailability in DHT storage
  - Degree of replication
- Prototype implementation

What is Wrong with Current Email?

- Email is mission critical for many institutions/people
  - Reliable servers expensive
- Single-server architecture vulnerable
  - Distributed clusters → expensive, still vulnerable
- Server-centric has other problems
  - Storage stress (big attachments)
  - Additional processing (spam & viruses)
- Neighborhood communities?

- Peer-to-peer architecture alleviates problems

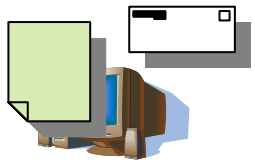
Assumptions

- Community of peers (nodes)
  - Peers up/down, peers independent
- What we assume to have:
  - Distributed Hash Table (DHT) is available (e.g., Chord)
  - DHT gives *k* closest nodes to given key (currently up)
  - No P2P storage layer (e.g., CFS, PAST)
- Store-and-forward email architecture
  - Only message delivery; permanent storage locally
  - Assume user has dedicated computer (revisit later)
- Analyze requirements of P2P email architecture

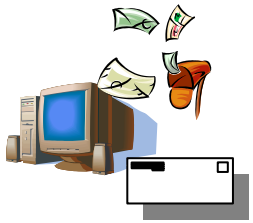
Entities

- Peers in DHT-space
- System nodes store data
- User agents access data
  - UA and SN can be same or different
- Users:
  - Address certificate
  - Inbox
- Messages
  - Headers stored in inbox
  - Message bodies separately
  - Message-ID header as key
- Similar to POP-email

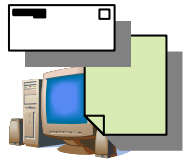
bob-cert



bob-inbox



alice-inbox



alice-cert

Service Primitives

- Store
  - Stores objects on *k* system nodes closest to object's identifier
  - List of authorized persons
- Fetch
  - Retrieves objects from any or all of *k* system nodes
- Delete
  - Remove object from nodes
  - Check authority
  - Deletion not guaranteed
    → garbage collection

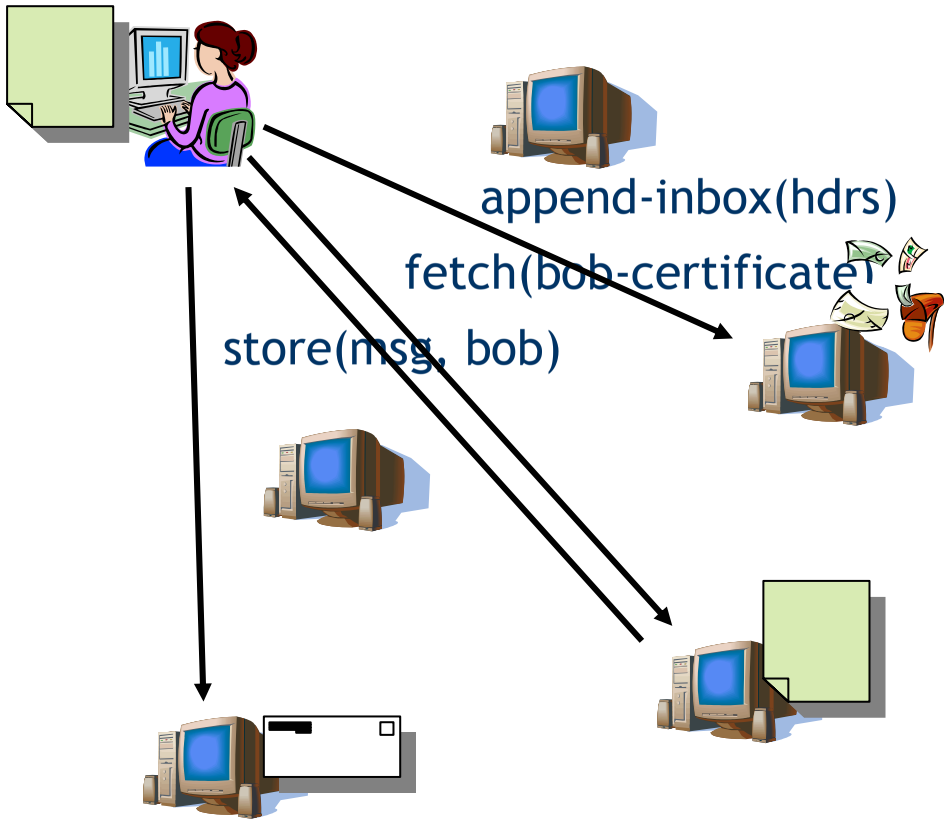
- Append-inbox
  - Append headers to inbox
  - Inboxes not consistent

  → Form superset when reading
- Read-inbox
  - Read from all *k* nodes
  - Return all headers from inbox
  - Clears inbox atomically

- Note: No need to enforce consistency between copies

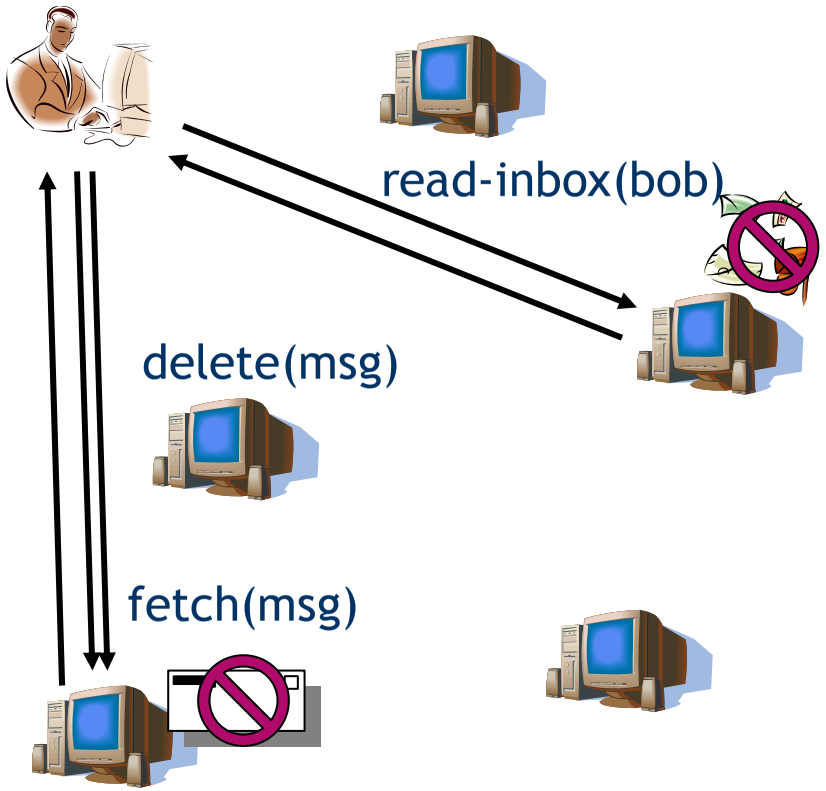
Alice Sends a Message to Bob

- Alice fetches Bob's certificate
- Alice writes message
  - Alice picks session key
  - Encrypt message with session key
  - Encrypt session key with Bob's public key
- Store message
- Append headers to Bob's inbox
  - Headers encrypted
- Same view to user!



Bob Reads His Messages

- Bob fetches his inbox
  - Read from all *k* nodes, form superset
  - Inboxes cleared
- Bob fetches message
- Message deleted
  - Delete from all *k*
  - Garbage collection



Reliability of Data in DHT-Storage

- Storage system using a distributed hash table (DHT)
- Peer *A* wants to store object *O*
  - Create *k* copies on different peers
  - *k* peers determined by DHT for each object (*k* closest)
- Later peer *B* wants to read *O*
  - What can go wrong?

- Simple storage system: Object created once, read many times, no modifications to object
- Assume *l* peers, peers homogeneously up/down (*p*), uniformly distributed in hash space

3 Causes of Loss

1. All *k* peers are down when *B* reads

$$p_{l1} = (1-p)^k$$

2. Real *k* closest peers were down when *A* wrote and are up when *B* reads

$$p_{l2} \approx \sum_{i=k}^{(1-p)I} \binom{(1-p)I}{i} \left(\frac{p(1-p)}{I}\right)^i$$

3. At least *k* peers join and become new closest peers

$$p_{l3} \approx \sum_{i=k}^{N} \binom{N}{i} \frac{1}{I^i}$$

Results

- First case dominates clearly
  - For cases 2 and 3 applies:
    Search more than *k* nodes

$p_{l_3} \approx$

I		
10^2	10^3	10^4
10^{-10}	10^{-15}	10^{-20}

p	$p_{l_1} \approx$
0.99	10^{-10}
0.9	10^{-8}
0.5	0.03
0.3	0.17

$p_{l_2} \approx$ (for given I and p)

0	10^{-15}	10^{-15}
10^{-8}	10^{-8}	10^{-8}
10^{-4}	10^{-4}	10^{-4}
10^{-3}	10^{-3}	10^{-3}

- How to improve?
  - Maintain storage invariant → *O* always at *k* closest
    - Needs additional coordination
    - Possible if down-events controlled
    - Crash → others need to detect crash (before they crash)
  - Increase *k* → waste storage (maybe not a problem?)

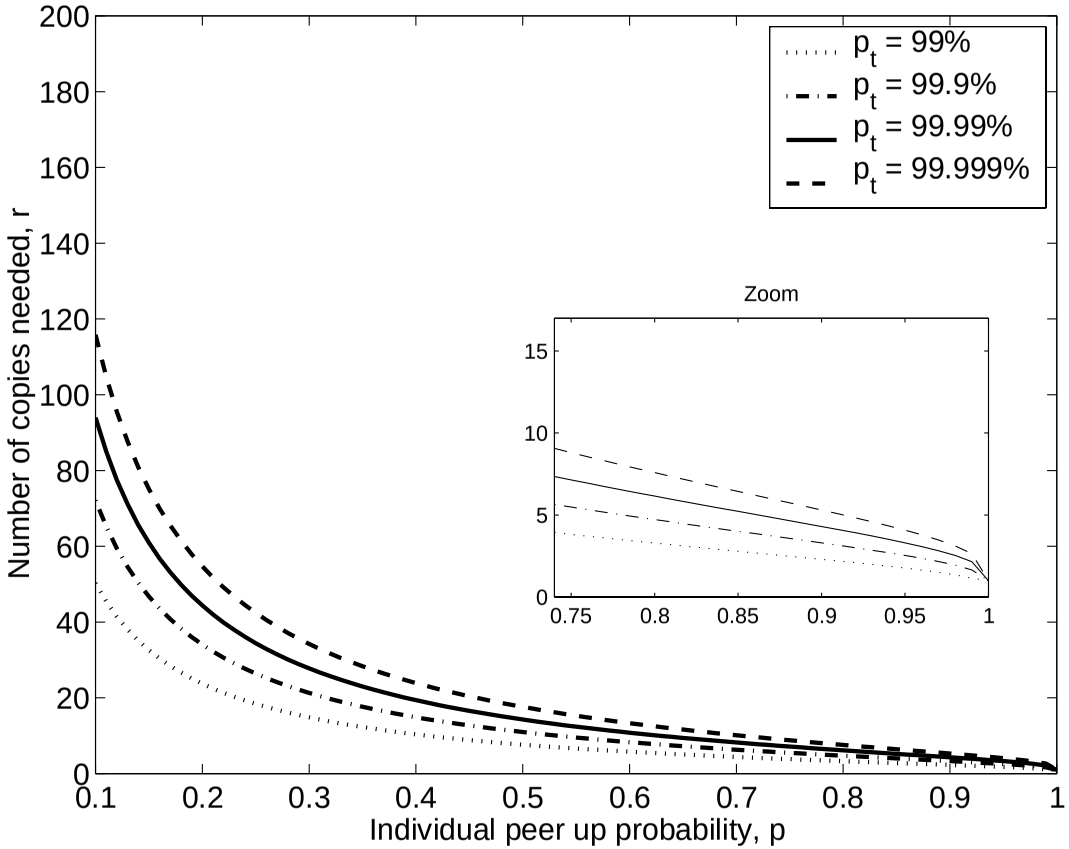
What the User Sees?

- Every user's action needs to access several objects
- For each access: $p_s = 1 - p_{l1} = 1 - (1 - p)^k$
- Reading and sending need 2 objects
- Success for user: $p_t = (1 - (1 - p)^k)^2$

- Solving for k:

$$k = \frac{\log(1 - \sqrt{p_t})}{\log(1 - p)}$$

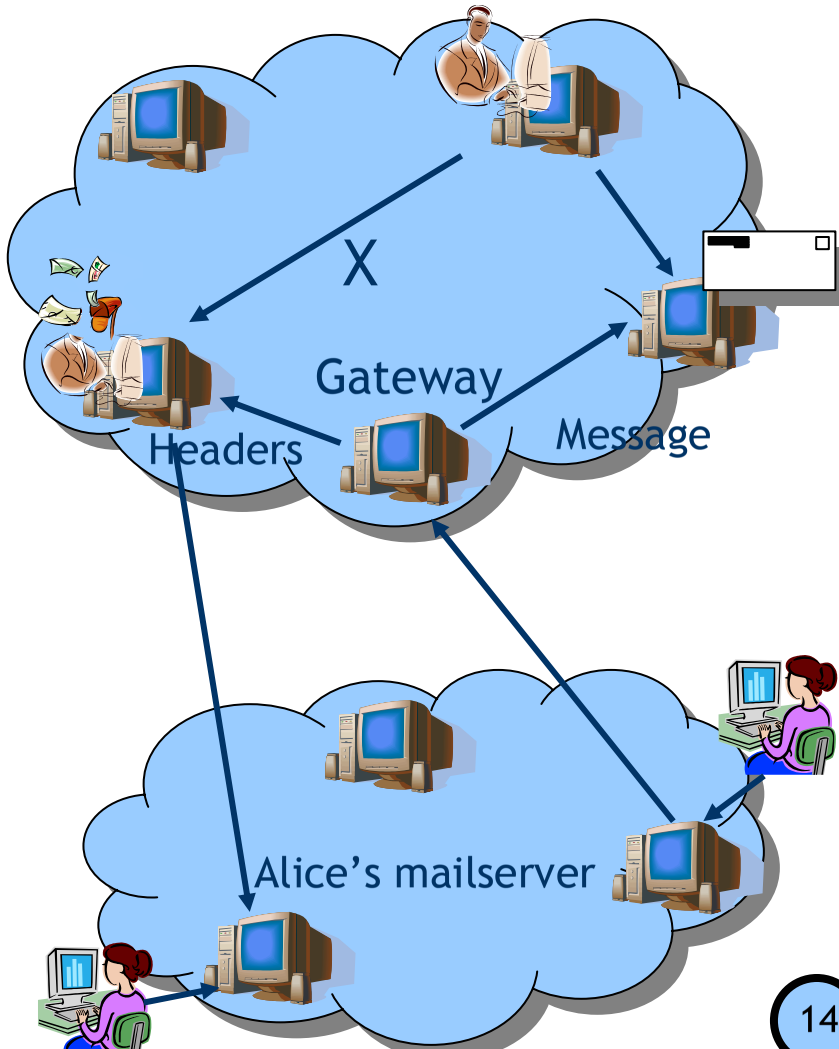
How Large Should *k* Be?



- Define target p_t
  - This is what user sees
  - Failures temporary
- When peers mostly up, *k* small
- Increase in p_t → small increase in *k*

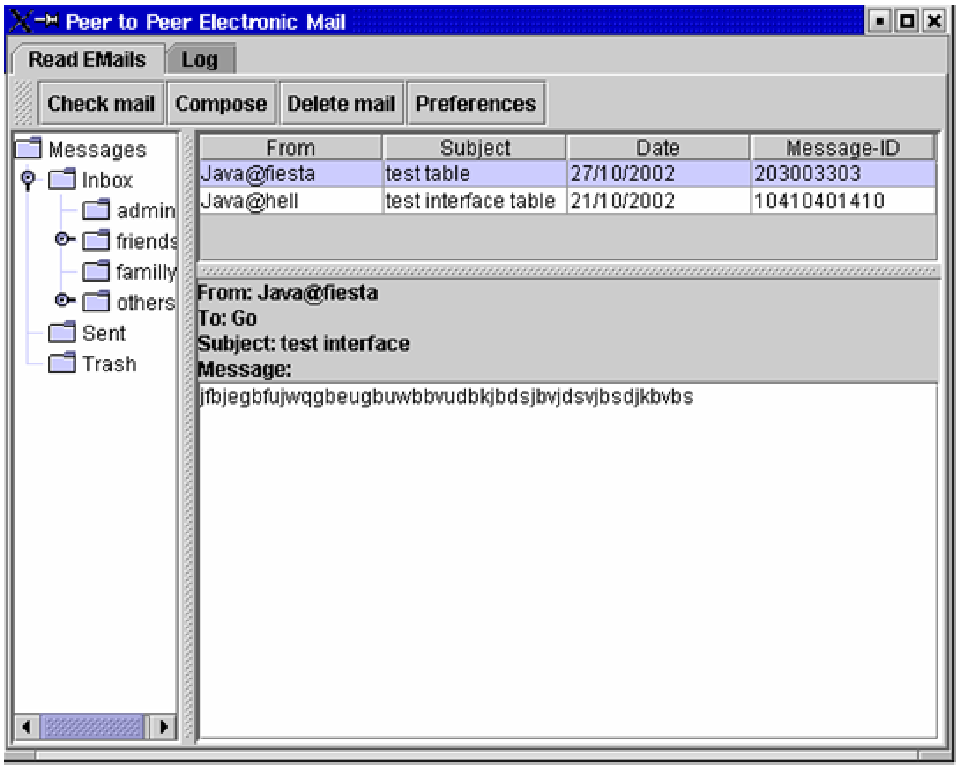
Interoperability and Migration

- Organization X replaces old email with ours
- How to talk to others?
- Designate gateways
  - Outgoing and incoming
- Outgoing gateway can be sending peer
- Incoming gateway's address must be in DNS (MX-query)
  - Gateway splits messages
- MX-support needed for others to send mail to org. X



Java Prototype

- Two components:
  - System node
  - User Agent
- Parameters for tuning communications
  - maxOps
  - maxConns
- Object transfer over HTTP/1.1
  - Some new headers



Discussion and Future

Requirements:

- DHT substrate
- No need for storage system
- “No need” for consistency

No permanent storage

- Turn off garbage collection?
- Enforce replication in UA?
- Still, need 10-20 times storage of central server
- Storage on peers is free!

How about mobility?

- Information about folders and read messages?
- Store email metadata (folders, etc.) on peers
- Offer permanent storage

  → Full access on the move

- Problem: Need to access private key often...

  → Mobility = Trusted access

Related Work

- POST by A. Mislove et al.
- Independent work in parallel
- POST relies on Pastry DHT, PAST storage layer, and Scribe multicast system (notifications)
- Main differences:
  - POST uses convergent encryption (some weaknesses)
  - POST stores inbox on user's computer

  → Mobility requires keeping own computer on

  → Still applies: mobility = trusted access to own computer
  - Notification if user is online vs. periodic polling

Conclusion

- Design and implementation of P2P email
- Store-and-forward architecture
  - Can be extended
- Analyze requirements
  - DHT, ability to store objects, no consistency
- Reliability analysis of P2P storage
  - How many copies needed for given target quality
- Java prototype implementation