Sailor: Efficient P2P Design Using In-Network Data Lockers

Richard Alimi (Yale)  Y. Richard Yang (Yale)
Hongqiang Liu (Tsinghua)  David Zhang (PPLive)
Erran Li (Alcatel-Lucent)  Roger Zhou (China Telecom)
Relation to DECADE

- We present Sailor as just one possible solution architecture for the DECADE problem statement
Basic Idea

- Use *application-agnostic*, in-network *data lockers*
  - Data locker storage provided to users
  - Data lockers accessed by a standard protocol

- Decouple P2P data and control plane
Overall Operational Model

- **Service Provider** provides multiple *data locker servers*

- Data locker server hosts multiple *data locker accounts*
  - Data locker account referred to simply as *data locker*

- **User** gets data locker account(s) on data locker servers
  - User may be an end user or a content publisher

- Users' *P2P applications* retrieve/store objects (chunks) using data lockers
**Simple Example**

**Native BitTorrent Clients**

Client A \(\rightarrow\) P2P Control \(\rightarrow\) Client B

Client A \(\leftarrow\) P2P Data \(\leftarrow\) Client B

**Sailor-enabled BitTorrent Clients**

Client A \(\leftrightarrow\) Locker La \(\leftrightarrow\) Locker Lb \(\leftrightarrow\) Client B

Client A \(\rightarrow\) Locker Data \(\rightarrow\) Client B

Client A \(\leftarrow\) Locker Control \(\leftarrow\) Client B

Client A \(\leftarrow\) P2P Control \(\leftarrow\) Client B

Client A \(\leftarrow\) P2P Data \(\leftarrow\) Client B
Scenario I: P2P Clients have Lockers

Client A

Client B

Client C

Source

P2P Control/Data

Locker Control/Data

Scenario I: P2P Clients have Lockers

(1) Source to Locker La

(2) Source to Locker Lc

(3) Client A to Locker La

(4) Client A to Client B

(5) Client A to Client C

(6) Locker La to Client B

(7) Locker La to Client C

(8) Locker Lc to Client C
Scenario 2: Content Publishers have Lockers

Client A
Client B
Client C

P2P Control/Data
Locker Control/Data
Scenario 3: Content Publishers and Clients have Lockers
Sailor Architecture

Data Locker Server
- Efficient Data Storage
- Data Lockers

Resource Model

Data Locker Access Protocol

Application/Sailor Integration

Client

Data Locker

Data Locker

Data Locker
Data Locker Resource Model

- Hierarchical, weighted partitioning
  - Each user assigned a weight by data locker provider
  - User configures weight assigned to each application
  - Application controls the partition of resources among open connections (if applicable)

- Resources
  - Bandwidth, storage, open network connections
Locker Access Protocol (LAP)

General Approach
- Data Locker Server simplicity
  - Scale to many users
- Reduce resource management messaging

Components
- Data Interface
  - Get, store, inter-locker communication
- Management Interface
  - Manage resources in own locker
LAP: Requirements

- **End-to-end Control**
  - Data locker users decide (independently) when to use locker
  - Explicit authorization for each item

- **Concurrent transfers**
  - Upload/download to/from multiple peers

- **Low delay for data transmission**
  - Reduce delay due to passing data through lockers
LAP: Capability Tokens

- Capability tokens encode
  - Authorization
  - Resource allocation

- Generated and managed by clients
  - Shared key with own data locker
  - Tokens passed via P2P application protocol
LAP: Data Interface

- **slr-store**
  - Store object in data locker
  - **In:** AppID, ObjID, ObjData, Token
  - **Out:** ErrCode

- **slr-get**
  - Retrieve object from data locker
  - **In:** AppID, ObjID, Token
  - **Out:** ObjData, ErrCode
LAP: Data Interface (cont'd)

- **slr-get** (overloaded)
  - Retrieve object from remote data locker and into own locker
  - **In:** AppID, ObjID, Token, RemoteAppID, RemoteToken
  - **Out:** ObjData, ErrCode
Preliminary Evaluation: Bittorrent

All clients inside an ISP have locker accounts

(a) Access Supply
Preliminary Evaluation: PPLive
Thank you!
Backup Slides
Efficient Locker Data Storage

store(obj) – store obj, if duplicate, store only a link

$H$ is a hash table indexed by the hash of each existing object

01. if (fetch from same locker server) then
02. store only a link to existing obj
03. return
04. else
05. h = hash(obj)
06. if (h == h1 ∈ H) then
07. obj1 = object with hash h1
08. if (obj1 == obj) then
09. store only a link to obj1
10. return
11. endif
12. endif
13. endif
14. store obj
Data Locker/P4P(ALTO) Integration

- Client $a$ with locker $L_a$ needs to select peers

- Consider peer $b$
  - Let $C_{a,b}^0$ be the cost from $a$ to $b$

- Three cases
  - If $b$ is a legacy peer
  - else if ($b$ supports DL but no locker account)
  - else // $b$ supports DL and has locker $L_b$

- $C_{a,b}$  
- $C_{a,b}^0$ 
- $C_{ab}$  
- $C_{ab}^0$ 
- $C_{ab}$  
- $C_{ab}^0$ 
- $C_{ab}$  
- $C_{ab}^0$
Preliminary Evaluation: Bittorrent