Motivation

- Increasing content availability
  - Both user generated and publisher generated contents

- Increasing consumption of content
  - E.g., Akamai estimates: 1,296 Tbps [Akami]
Challenge

- Potential bottlenecks at
  - the uploader last (first) mile and/or
  - the middle mile

- Solution
  - Replication/service capability inside the networks
Existing Approaches and their Problems

- IP multicast
  - Lacking in global deployment

- CDN
  - Closed systems with limited scopes

- P2P cache
  - Application specific
What are Data Lockers?

In-network storage/BW accounts, offered by multiple providers, accessible using a standard protocol, under application explicit control, to provide efficient content distribution.
Who Provide Data Lockers?

- A provider of data lockers is called a Content Delivery Storage Provider (CSP)
- An CSP can be an ISP, or a third party (e.g., cloud storage provider)
What should Data Lockers Provide?
Functions of a Content Delivery System

- **Control**
  - Content search/index & composition/authorization
  - Replication and request routing
    - Scaling, Efficiency (e.g., Proximity), Load Balancing and Reliability

- **Data**
  - Storage/transport
“Weak” Coupling: Separation of Control and Data.

“Strong” Coupling: Tight integration of Control functions into in-network storage.

A potential extreme is virtual machines. Not considered for now.
The Decoupled Design Principle

- The data-locker decouples control functions out of data functions as much as possible
  - Decoupled Control and Data Enroute (DeCaDE)

- Control functions implemented either by applications or separate control protocols
Overview

- Reduce production complexity and provide open access
- Integrate with application policies
Use Case: End-User Based Control

Native P2P Clients

Client A  

P2P Control
P2P Data

Client B

Data Locker-enabled P2P Clients

Client A

Storage $S_a$

P2P Control
P2P Data (alternative)

Client B

Storage $S_b$
Use Case: End-User Based Control

Client A

(4)

(3)

Storage $S_a$

(5)

Client B

(2)

Source

(1)

Data

Control

(7)

Storage $S_c$

(8)

Client C
Can Data Lockers Address Challenges of P2P Based Control?

- Low network efficiency, in particular last mile
  - Upload from data locker account of a client, instead of client last mile

- Instability due to high churns
  - Upload from data locker account of a client, even when client goes offline
Use Case: Publisher Based Control

Content Publisher

Storage $S_a$

Storage $S_c$

Client A

Client B

Client C

Control

Data
Potential Benefits of a Decoupled Architecture

- Separates the storage/bandwidth intensive (data) functions from the processing intensive (control) functions
  - Flexible/open/evolvable control platforms
  - Shared data infrastructure

- Decoupled architecture is not new, e.g.,
  - openflow
  - Google File System (GFS)
Design Details: What Does the Data Path Look Like?
Data Naming and Storage Model

- Data naming is an important problem
  "There are only two hard problems in Computer Science: cache invalidation and naming things." — Phil Karlton

- There are many naming models,
  ◦ E.g, Filename, URL, attribute, DONA

- Key assumption
  ◦ Content distribution deals with immutable data

- For immutable data, no need to separate identifier and content

- Design:
  ◦ Each account at a server provides a key-value store with self-certifying keys
  ◦ ID=Key: <Hash_of_DataBlock>
  ◦ Value: <DataBlock>
Read/Write Model: The Basic *Distribution* Primitive

- A basic data command primitive is to indicate a data path
The data path primitive from Client C to server S specifies

- `<data id>`,
- a `<src>`,
- an account on S `<S:account>`, and
- a `<dst>`.

**Interpretation**
- If `<src>` is null: it is a pure read to transfer data from `<S:account>` to `<dst>`
- If `<dst>` is null: it is a pure write to store data from `<src>` to `<S:account>`
- Otherwise, it is a distribution pipeline from `<src>` to `<S:account>` to `<dst>`
Example: Endpoint Controlled Data Flow

- Client P1 has account on A
- Client P2 has account on B

1. P1 puts D₁ in A
2. P2 learns that P1 has D₁
3. Request data D₁
4. Response w/ Token T₁ allowing accessing Data D₁
5. Distribution Request:
   data: D₁
   pivot account: B:P2
   src: Token T₁
   dst: P2
6. Get Object D₁ using Token T₁
7. Return Data D₁
8. Return Data D₁

Storage Server A

Storage Server B
if (no data across the server for <data id>)
server S gets data from <src>;
else
    if (<src> is an account on same S)
        // local deduplication
        link data, and ack succ else
    else
        // remote BW deduplication
        sends challenge for verification

Note: Could move dedup out to app, but then fully implementing dedup requires cross application/session synchronization. Also, hood for content checking. <src> can be protected, can also be chained.
Resource Model
Two Major Architectural Components of **Multipath** Data-Oriented Content Distribution

- **Topology Management**: Who connects to whom?

- **Chunk (Data) Scheduling**: Who serves whom at what rates? Includes
  - A downloader requests from which uploaders
  - An uploader serves which downloaders at what rates

We can consider both components as conducting resource control on resources, including
- connection slots
- upload/download bandwidth
- storage capability
Why is Resource Control of BW/Connectivity Important?

- Because BW resource control is fundamental for
  - Robustness against selfish behaviors
  - Robustness against attacks
  - Construction of efficient flow distribution patterns (in particular for streaming)
Example Flow Pattern: Live Streaming Feasibility Theorem

- The flow patterns depend on application types and can be the key “secret sauce” of different designers

- For live streaming
  - Assume that each peer $u$ allocates capacity $C_{uv}$ to a connected neighbor $v$
    - We call $C_{uv}$ the link capacity of the link $u$ to $v$
  - Constraints that $\{C_{uv}\}$ should satisfy:
    - Quota: sum of $C_{uv}$ over all neighbors $\{v\}$ of $u$ should be less than the upload capacity of $u$
    - Flow Pattern: For any peer $p$, the maximum flow (minimum cut) from source $s$ to destination peer $p$, under link capacity constraints, should be at least the streaming rate $R$
A hierarchical, weighted resource partitioning scheme
  ◦ Each user is assigned a weight by the data locker provider
  ◦ A user configures weight assigned to each concurrent application
  ◦ Each application controls the partition of resource among open connections
Integration Overview

DECADE Server

DECADE Protocol

DECADE Client

P2P App

DECADE-enabled P2P Client

DECADE Server

DECADE Protocol

DECADE Client

P2P App

DECADE-enabled P2P Client

DECADE Protocol

P2P Protocols
Integration Case 1: P2P Live Streaming
Integration Case 2: P2P File Sharing

Vuze Client
- Download Manager
- Upload Manager
- Peer Manager

Network Access

Client
- Token Generator
- Generate Token

Storage Access
- Read/Write/Distribute Objects Using Distribution Primitive

Plug-in API
- Redirector

DECADE Client
Example: Data Request Flows of Vuze with Storage

Vuze Client 1  Plugin 1  Plugin 2  Vuze Client 2  Data locker 1  Data locker 2

- BT_HandShake
- Vuze HandShake
- BT_BitField
- BT_Request

Intercept the message

Redirect

Distribution (Locker 2 -> Locker 1 -> Client 1)

Read

Data

BT_Piece

Data
Benchmark Setting

- For performance comparison of Native and using Data Lockers, always consider two scenarios with the same amount of total network bandwidth resource.

- Evaluation on both file sharing and live streaming.
Experimental Setting

Storage Server

Storage Server

P2P Client

P2P Client

Source Server

Tracker

Test Controller

Amazon EC2

Planetlab

Yale lab
Evaluation:  
Non-Realtime Content
Integrating into Vuze/BitTorrent: Last Mile Download Traffic

- Note: The same total number of Vuze clients; but some Native Vuze clients could not finish downloading; the total download traffic of Native is lower.
Integrating into Vuze/BitTorrent: Last Mile Upload Traffic

Native Vuze

Data Locker Vuze
Incremental Deployment: Last Mile Upload Traffic

![Bar chart showing normalized access supply for Native, All, and One ISP only categories.](image-url)
Application Performance (Resource Efficiency)

Implication:

Speed up = \frac{(0.88 - 0.65)}{0.65} = 35\%
Why Does Data Locker Gain on Efficiency?

- Statistical multiplexing gain

- Server deduplication to change the location of bottleneck

- Decoupled faster control cycle to speed up distribution
  - Not implemented by current prototype
## Summary: P2P File Sharing

<table>
<thead>
<tr>
<th></th>
<th>Improvement with In-network Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client upload volume</td>
<td>430 MB → 12 MB</td>
</tr>
<tr>
<td>System resource efficiency*</td>
<td>65% → 88%</td>
</tr>
<tr>
<td></td>
<td>(35% speedup)</td>
</tr>
</tbody>
</table>

*System resource efficiency: fraction of total available upload capacity used*
Evaluation:
Live Streaming Content
## Results: P2P Live Streaming

<table>
<thead>
<tr>
<th></th>
<th>Improvement with In-network Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup delay</td>
<td>At 80-percentile: reduced to 1/3 when no storage</td>
</tr>
<tr>
<td>Piece lost rate</td>
<td>About the same, at $\leq 0.02%$</td>
</tr>
<tr>
<td>Average # of freezes</td>
<td>Reduced to 2/3 when no storage</td>
</tr>
</tbody>
</table>
Summary: Current Design Choices

- **Storage Model**
  - key-value store with self-certifying keys

- **Read/Write**
  - [RWDirect] with deduplication

- **Resource Management**
  - Work-conserving proportional allocation
Next Steps

- Presented preliminary design
- Pursuing the direction in IETF DECADE Working Group
- Participation welcome!
Backup Slides
Poor documentation, ongoing protocol changes and rapid introduction of new features make P2P protocol support in caching system a constantly moving target.

-- PeerApp
Why Data Lockers?

- Initially motivated by P2P CDN
  - Advantages
    - Highly-scalable
    - Robust
    - Space for innovation
      - Many novel techniques
      - Many players with novel ideas
  - Problems
    - Low network efficiency
    - High churns
**Additional Use Cases**

- **[Use Case II: Global CDN by Aggregation; CDI]** Can an aggregator build an Akamai–like global CDN utilizing multiple CSPs?

  ![Diagram of CSPs and Aggregator](image)

- **[Use Case III Video conference (i.e., UGC)]**
  - Can a video conferencing application (e.g., iPhone video) utilize CSPs to distribute video from one participant to multiple other participants?

  ![Diagram of CSPs and Participant](image)
Q: How to write into CSP?

[DirectWrite]
- Client writes into specific CSP server (cluster) store
  - Still allow DNS to direct to preferred server by CSP
  - Clients provide replication/request routing

[IndirectWrite]
- [RW–IndirectPull]
  - Client maintains a source
    - Publishes source location to CSP
    - CSP provides internal caching, replication and request routing among internal caches
    - CSP pulls from source when source data first requested

- [RW–IndirectStaging]
  - Variation of [RW–Pull], client uploads to a staging service
Account holder can list keys in its own account, delete keys; keys have expiration time

Not provided: listing of content at aggregation levels (management can have so)
  ◦ Distributed indexing implement by Application
  ◦ [Considering the possibility of Special account (e.g., public account)]
Why Resource Control: Robust Against Selfish Behaviors

- P2P systems depend on user contributions

- Non-contributing users can be a serious problem
  - 70% of Gnutella users share no files and nearly 50% of all responses are returned by the top 1% of sharing hosts

- BW resource control is a major mechanism to design incentives and handle selfish behaviors
  - BitTorrent Tit-for-Tat
    - Attacked by BitTyrant
  - Provable Proportional Sharing [STOC’07; SIGCOMM’08]
A recent study [IMC’08] showed how to attack the Akamai streaming servers due to sharing of server bandwidth but no isolation.

“We demonstrate that it is possible to impact arbitrary customers’ streams in arbitrary network regions …” [IMC’08]
Remote controller cannot control Vuze clients to seed, so we use one Vuze client at Yale for seeding.

Tracker: Vuze client provides tracker capability, so we don’t deploy our own tracker.
DECADE/Vuze Settings/Metrics

- **Settings**
  - 70 peers
  - Native: Peer Upload Capacity: 40 KBps
  - Data locker: Server Capacity: 40 KBps * 70

- **Performance metrics**
  - Client upload bandwidth
  - System resource efficiency: fraction of total network BW used
## DECADE/Vuze Hardware and Software Specification

<table>
<thead>
<tr>
<th></th>
<th>Hardware Environment</th>
<th>Software Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECADE server</td>
<td>Server in EC2</td>
<td>Ubuntu</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DECADE server software version</td>
</tr>
<tr>
<td>HTTP server</td>
<td>Server at Yale</td>
<td>Windows 2003 Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tomcat</td>
</tr>
<tr>
<td>Tracker server</td>
<td>Server at Yale</td>
<td>Windows 2003 Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vuze client</td>
</tr>
<tr>
<td>Vuze client</td>
<td>Download clients: Virtual machine at Planetlab</td>
<td>Vuze client for Windows &amp; Linux</td>
</tr>
<tr>
<td>with DECADE plugin</td>
<td>Seed client: at Yale</td>
<td>JRE 1.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DECADE plugin software version</td>
</tr>
<tr>
<td>Remote controller</td>
<td>Server at Yale</td>
<td>Windows 2003 server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JRE 1.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remote controller software version</td>
</tr>
</tbody>
</table>
Test Steps: Native Vuze

Precondition:
- Start remote controller and all Vuze clients
- Vuze clients register with remote controller; remote controller assigns the IP address of decade server

Test steps:
1. The Vuze client at Yale seeds
2. Manually upload the BitTorrent file to HTTP server
3. Remote controller starts up Vuze clients in PlanetLab
4. Vuze clients at PlanetLab fetch BitTorrent file from HTTP server
5. Vuze clients at PlanetLab receives peer list from tracker server
6. Vuze clients at PlanetLab network send BT_Request to peers and get BT_Piece message from peers
7. All Vuze clients report statistics to remote controller
Test Steps: DECADE/Vuze

Test steps:
1. The Vuze client at Yale seeds
2. Manually upload the BitTorrent file to HTTP server
3. Remote controller starts up Vuze clients at PlanetLab
4. Vuze clients at Planetlab fetch BitTorrent file from HTTP server
5. Vuze clients at PlanetLab receives peer list from tracker server
6. Vuze clients at PlanetLab send BT_request to peers and get Redirect messages
7. Vuze clients at PlanetLab download objects from DECADE servers
8. All Vuze clients report statistics to remote controller

Precondition:
- Create DECADE server instances at EC2
- The rest is the same as Vuze/Native
DECADE/PPNG Trial Setting

Google App Engine
- PPNG Manager

DECADE Server

EC2

DECADE Server

Planetlab
- PPNG Client

Yale lab
- PPNG Client

Streaming Files

Source

Tracker

Trial Controller

Browser To Display Result

11/18/2010
Streaming Rate: 40 KBps
Source Capacity: 200 KBps
[Native] Peer upload capacity: 64 KBps
[Data locker] Servers
- 5 servers at different Amazon EC2 locations
- Each server has capacity: 51.2 Mbps
- P4P/ALTO Map to assign clients to close-by Data Locker servers
## DECADE/PPNG Hardware and Software Specification

<table>
<thead>
<tr>
<th>Components</th>
<th>Platform</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decade Server</td>
<td>• Run in EC2, (US East, US West, EU, and Asia Pacific) • EC2 images</td>
<td>• OS: Ubuntu 10.04 • 3-party lib • EC2 control scripts</td>
</tr>
<tr>
<td>PPNG Client</td>
<td>• Run at PlanetLab</td>
<td>• PPNG Client • DECADE Client lib • 3-party lib</td>
</tr>
<tr>
<td>Tracker</td>
<td>• Run at Yale</td>
<td>• PPNG original tracker</td>
</tr>
<tr>
<td>Source</td>
<td>• Run at Yale</td>
<td>• PPNG original source • DECADE integration</td>
</tr>
<tr>
<td>Trial Controller</td>
<td>• Run at Yale</td>
<td>• Experiment control scripts • Log collecting scripts</td>
</tr>
<tr>
<td>Media Player Server</td>
<td>Run in PPNG Client</td>
<td>• Integrated in PPNG Client</td>
</tr>
<tr>
<td>GoogleMap Webpage</td>
<td>Run at Yale</td>
<td>• Google Map API • Runtime scripts</td>
</tr>
<tr>
<td>Online Statistic</td>
<td>Run at GoogleApp Engine</td>
<td>• Client log reporter • Log server</td>
</tr>
</tbody>
</table>
DECADE/PPNG Write/Put Block

Current version, only source explicitly put data into Data Locker

- is legacy PPNG blocks
- is DECADE blocks

cdnjs-source → http → cdpn-add-piece

dec-put
If the message is a legacy subpiece request, and if dst is a DECADE peer change it to DECADE request.
If is a DECADE request

peer-decode → dec-peer-decode → dec-response-constructor → dec-peer-encode → peer-encode

If is a DECADE response

dec-get
Get data from Data Locker → recv-subpiece

Otherwise

Legacy PPNG Blocks
Client $a$ with locker $L_a$ needs to select peers

Consider peer $b$

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

Three cases

- If $b$ is a legacy peer
  - $C_{a,b} = C_{0,a,b}^0$
- else if ($b$ supports DL but no locker)
  - $C_{ab} = C_{L_a,b}^0$
- else // $b$ supports DL and has locker $L_b$
  - $C_{ab} = C_{L_a,L_b}^0$
Resource Contention