03/04: Data oriented Transfer

- Allows avoiding of fn. split in traditional n/w.
- Decoupling of functionality from content negotiation from content transfer.
- Generic source design with clearly defined interfaces.

Key motivation/observation:

- Innovation in data transfer techniques is hard.
- Imagine you have a novel technique that you want to deploy and use.
  - Modify HTTP/SNTP; talk to BTF; change apps.
  - Long and painstaking process.

- Why? Applications bundled data transfer with application-specific content negotiation.
  - Naming scheme...
  - URL, directories.
  - Encodings, etc.

Data transfer itself is just the function of moving bits and it is common to many applications.

Tight coupling hinders new services.

Solution: A separate, generic data transfer service that implements the app-independent payload as a separate service.
This enables an extensible transfer architecture

Benefits

- Apps can reuse available transfer techniques
- Easy deployment of new
- Cross-app sharing
- Handles transparent disconnection
  - Multihoming
  - Data oriented nature
- Content delivery
- Data delivery acceleration in a protocol independent fashion
- Use any n/w technology
- Cross-application data processors, such as virus checkers

10,000 ft view

\[
\begin{array}{c}
S \\
\text{put}(x) \\
X.S \\
\text{??} \\
X.S.
\end{array}
\quad \begin{array}{c}
R \\
\text{read}() \\
\text{??} \\
\text{??} \\
\text{??} \\
\text{??}
\end{array}
\]

Q1: How to name data?
- Host and app-independent content name
- CSP
Objects can be further subdivided into chunks.

- `OD` → list of chunk descriptors.
- `chunks` → allow for partial Xfers

**Object Location:**
1. `data` file are accessed, drivers
2. Sender provides hints
3. Receiver selects appropriate location(s) depending on local constraints

→ late binding, flexible adaptation, multipath; disconnection tolerance etc

**A transfer using DOT:**

```
A transfer using DOT:

```

**API and Modular Architecture:**

```
API          DOT - App
             App
DOT          xfer
            plugin
            plugin API
Storage plugin
            plugin API
0 → put, get
1 → get, desc. (hits) → VFI (method) → plugin and params
2 → get, chunks (hits) → priority → order of signup
    weight → prob. of signup when priority is same
03/04: Data-oriented Transfer

allows avoiding of in-split in traditional m/w.

Decoupling of functionality

separation of content negotiation from content transfer

generic source design with clearly defined interfaces.

Key motivation / observation:

- Innovation in data transfer technique is hard
- Imagine you have a novel technique that you want to deploy and use
  - Modify HTTP / SMTP; talk to BETF; change app.
  - Long and painstaking process.

- Why? Applications bundled data transfer with application-specific content negotiation
  - Namng scheme
  - URL, directory
  - Encodings etc.

Data transfer itself is just the function of moving bits and it is common to many applications.

Tight coupling - hinders new services

Solution: A separate, generic data-transfer service that implements the app-independent part as a separate service.

```
  Sender  App-protocol  Receiver  ->  Sender  App-protocol  Receiver
  Data    ↓           ↓           ↑           ↓           ↑
           X5          Data          X5
```