Integrating erasure codes into ICN

**Erasure code benefits**
- Resilience to packet loss
- Reduced latency via downloads over multiple interfaces
- Improved support for mobile clients
- Improved caching performance

**Potential integration overheads**
- Signaling, latency, security, response, storage

**Previous work does not extract the full benefits**
- Each has issues with at least some of these overheads

**Our approach**
- Network architecture design that provides benefits and minimizes overheads
- Seamless integration into NDN
LDN focus within ICN

Object to network data name-mapping
- Object is unit of data that is useful: video segment, image, email, file
- Packet is unit of data that is transported or cached in network
- Mapping is from object name to packet name

Request-response paradigm
- How clients and nodes request packets for an object
- How nodes respond to requests with packets

Security considerations
- End-to-end object verification
- Packet verification to prevent DoS attacks
NDN approach

Object

Source packets

Name-mapping

\[ D \quad D.0 \quad D.1 \quad D.2 \quad D.3 \quad D.4 \]

Request-response

Client explicitly requests \( D.0, D.1, D.2, D.3, D.4 \) to recover \( D \)

Security

Packet verification: Publishers sign packets, nodes and clients validate signed packets
Object verification: clients accept an object if all its source packets are valid
Erasure codes in ICN

object

source packets

repair packets


Coordination strategies

**Additive response**
- Same client receives different packets from different nodes

**Common response**
- Different clients receive same packets from same node
Prior request strategies

**Specific data**
Client asks for specific packets of data by name

**Random data**
Client requests an amount of data packets
Response is randomly generated encoded data

**Useful data**
Client specifies data it has already received
Response is additional data that will be useful

How to enable common response?
How to distinguish between common and additive response?
Issue with additive response
**LDN enables coordination**

**SOPI** $P = (A, B)$, where $A \in \{0, \ldots, N - 1\}, B \in \{1, \ldots, N - 1\}$

Defines permutation $\{A, A + B, A + 2 \cdot B, \ldots, A + (N - 1) \cdot B\}$, where each term is modulo prime $N$

---

**object $D$**

$K = 5$ source packets in size

$N = 11$ packets

$P = (3,5)$

**stream object $D \cdot P$**
LDN request-response paradigm

SOPI assigned to each node
Client requests prefix of stream object associated with SOPI

Additive response

Common response
LDN desirable properties

Support large $K$
- Small $K$ forces splintering objects into many source blocks, causes large response overheads

Support $N > K^2$
- Minimal prefix overlap of stream objects with different SOPIs
- Can choose SOPIs randomly
- Can deterministically design SOPIs

Erasure code properties
- Linear coding complexity
- Even when coding repair packets only

Erasure code choices
- RLNC, Reed-Solomon – significant complexity/response overhead tradeoffs
- BAT fountain code has ok tradeoffs
- RaptorQ fountain code has near optimal tradeoffs

LDN turkey network example

Publisher of $D$ node

Prefix $D.P_0, \frac{K}{2}$

Prefix $D.P_1, \frac{K}{2}$

Prefix $D.P_2, \frac{K}{2}$

node

node

node

node

node

node

P0

P1

P2

client

client
Publisher of $D$ node

Prefix($D.P_0, \frac{K}{2}$) coalesced Prefix($D.P_1, \frac{K}{2}$)

Node $P_0$ Node $P_1$ Node $P_2$

Client

LDN turkey network example
Publisher of $D$

Prefix\( (D.P_0, \frac{K}{3}) \)  Prefix\( (D.P_1, \frac{K}{3}) \)  Prefix\( (D.P_2, \frac{K}{3}) \) coalesced
LDN turkey network example

Publisher of $D$

Prefix $D.P_0, \frac{K}{2}$

Publisher of $D$

Prefix $D.P_1, \frac{K}{2}$

Publisher of $D$

Prefix $D.P_2, \frac{K}{2}$

Client

Prefix $D.P_0, \frac{K}{2}$

Prefix $D.P_1, \frac{K}{2}$

Prefix $D.P_2, \frac{K}{2}$

Client

Prefix $D.P_1, \frac{K}{2}$

Prefix $D.P_2, \frac{K}{2}$

Prefix $D.P_1, \frac{K}{2}$

Prefix $D.P_2, \frac{K}{2}$
LDN load balancing example

Publisher of $D$

- Prefix($D. P_0, \frac{K}{5}$)
- Prefix($D. P_1, \frac{K}{5}$)
- Prefix($D. P_2, \frac{K}{5}$)
- Prefix($D. P_3, \frac{K}{5}$)
- Prefix($D. P_4, \frac{K}{5}$)

- $P_0$
- $P_1$
- $P_2$
- $P_3$
- $P_4$

- switch

- Each client requests $K/5$ packets of $D$ from each node
- Saves 5x on storage overhead versus conventional LB
- Easy to maintain, transparent to end-users
<table>
<thead>
<tr>
<th><strong>LDN security</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>originator</strong></td>
</tr>
<tr>
<td>Original generator of object</td>
</tr>
<tr>
<td>Preferable if network independent</td>
</tr>
<tr>
<td><strong>Object verification</strong></td>
</tr>
<tr>
<td>Originators sign objects</td>
</tr>
<tr>
<td>Clients verify objects</td>
</tr>
<tr>
<td>Ensures end-to-end integrity</td>
</tr>
<tr>
<td><strong>publisher</strong></td>
</tr>
<tr>
<td>Where object is available in network</td>
</tr>
<tr>
<td>• Node where object is injected into network</td>
</tr>
<tr>
<td>• Node that generates object from packets received</td>
</tr>
<tr>
<td><strong>Packet verification</strong></td>
</tr>
<tr>
<td>Publishers verify objects</td>
</tr>
<tr>
<td>Publishers sign packets+object creds</td>
</tr>
<tr>
<td>Nodes &amp; clients verify packets</td>
</tr>
<tr>
<td>• Misbehaving publishers can be identified and blackballed</td>
</tr>
</tbody>
</table>
Clients are in control

- Clients make explicit requests – avoid response overheads

Choosing and assigning SOPIs

- Different SOPIs assigned to nodes from which same client downloads
- Small number of SOPIs overall may be possible

Security

- Distinction between originator and publisher seems important

LDN is an architectural extension of NDN

- Interest message/response protocols can be the same as in NDN
- NDN naming and other features can be directly leveraged
- Benefits of erasure codes achieved, and overheads avoided
Thanks!