Some Technical/Architectural Issues
Overview

• Update and discussion of some ongoing work
  – Packet format, system design, tech memos.
    • LINK, ENCAP, NACK
    • Routing scalability
    • Name discovery, selectors.
    • Variable-length header
    • Hop-by-hop fragmentation/reassembly
    • Implicit digest
    • Naming conventions

• Suggestions on new topics.
LINK

- LINK is a data packet whose payload contains multiple names that point to the same content.

- Example:
  - Files are published under /net/ndnsim,
  - but hosted by ATT, /att/user/alex/net/ndnsim
  - Consumers need to use the latter name to retrieve the content across the Internet.
### LINK

<table>
<thead>
<tr>
<th>Name of the link object (/net/ndnsim/LINK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaInfo: <strong>ContentType=LINK</strong>, ....</td>
</tr>
<tr>
<td>Content:</td>
</tr>
<tr>
<td>alias 1, pref (/att/user/alex/net/ndnsim, 100)</td>
</tr>
<tr>
<td>alias 2, pref</td>
</tr>
<tr>
<td>Signed by the publisher of the LINK</td>
</tr>
</tbody>
</table>

- **LINK** is defined as a new ContentType.
  - Allow multiple aliases.
  - Support preference/weight for each alias.
ENCAP

- A general mechanism to encapsulate one or more packets under a different name.
  - A new ContentType
  - Each enclosed object is a complete packet on its own.
  - The outer signature covers the outer name and signatures of all the enclosed objects.

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaInfo: <strong>ContentType=ENCAP, ...</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Content:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Object 1, Object 2, ...</td>
</tr>
</tbody>
</table>

| Signature |
ENCAP

• Example:
  – Return a chain of certificates in response to a key retrieval request.

• Example:
  – Interest /att/user/alex/net/ndnsim/a.cpp
  – Return an encapsulated packet that contains
    • Original data object (/net/ndnsim/a.cpp), and
    • The LINK object (/net/ndnsim <- /att/user/alex/net/ndnsim)
    • Under an outer name like
      – /att/user/alex/net/ndnsim/a.cpp/encap
Application NACK

• “Content doesn’t exist yet”
  – Published by the content producer
  – A new ContentType
    • Routers process it as a regular data packet.
      – Satisfy PIT, cache it, etc.
      – No need to explore alternative paths.
    • Consumer apps need to handle this NACK.
  – Meant to be used at time scale much longer than regular interest/data exchange.
    • App NACK vs. retx/refresh.
### Application NACK

<table>
<thead>
<tr>
<th>Name of this NACK object</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaInfo: <strong>ContentType=NACK</strong>, ....</td>
</tr>
<tr>
<td><strong>Content:</strong></td>
</tr>
<tr>
<td>• Name (prefix) of non-existent content</td>
</tr>
<tr>
<td>• A code of why the content is not available</td>
</tr>
<tr>
<td>• Expiration time of this NACK</td>
</tr>
<tr>
<td><strong>Signed</strong> by the publisher</td>
</tr>
</tbody>
</table>

- Applications may add/remove what’s in the content part. Need more experimentation.
Application NACK Example

• A NACK is published for a prefix
  – /ndnsim/src
• But an Interest asks for a specific piece of data
  – /ndnsim/src/a.cpp
• Need to encapsulate the NACK object in order to match the interest name.

<table>
<thead>
<tr>
<th>Name</th>
<th>/ndnsim/src/a.cpp/nack</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaInfo:</td>
<td>ContentType=ENCAP,</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Content:</td>
<td></td>
</tr>
<tr>
<td>• NACK</td>
<td>(name=/ndnsim/src/nack, content, sig)</td>
</tr>
<tr>
<td>signature</td>
<td></td>
</tr>
</tbody>
</table>
Network NACK

• Non-authoritative, generated by routers, repos, server replica, etc.
  – “cannot get the content, because of X”.
  – Downstream node should explore other paths upon receiving this NACK.
    • NACK only when exhausted all local options.
  – The reason “X” is important for downstream to react appropriately. For examples:
    • Link failure: don’t send future interests upstream.
    • Congestion: send to upstream with reduced rate.
    • Loop/duplicate: try an interest with different nonce.
Per-packet Network NACK

- Return the Interest packet to the downstream as a NACK
- Include the error code in the shim layer (layer 2.5)
- In-band, fine-grained feedback.

### Layer 2.5: NACK and error code

<table>
<thead>
<tr>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interest Name</td>
</tr>
<tr>
<td>Other fields</td>
</tr>
<tr>
<td>signature</td>
</tr>
</tbody>
</table>
Aggregated Network NACK

• Upstream and downstream neighbors run a control plane app, e.g., /localhop/feedback/...

• Send NACK information as regular interest/data exchange between the control processes.

• Provide out-of-band, aggregated feedback.
  – E.g., when the outgoing link at the upstream node fails, it can send this NACK to the downstream node to stop incoming traffic.

• Closely related to routing decisions and forwarding strategy.
Routing Scalability

• The problem: what if core routers can no longer hold all the content prefixes.

• The solution: map-and-encap
  – Only a subset of prefixes are allowed in DFZ routers. They’re globally routable prefixes.
  – A distributed mapping system that given a content prefix will return one or multiple routable prefixes belong to ISPs hosting the content.
  – Interest is sent using the ISP/routable prefix to reach and retrieve the content. Returned data is encapsulated.
Data Encapsulation Approach

- Consumer app: /net/ndnsim/a.cpp
  - Should not be bothered with anything below.
- Consumer library/service/...
  - Look up the mapping system, get a LINK
    - /att/user/alex/net/ndnsim -> /net/ndnsim
  - Send interest with a routable prefix
    - /att/users/alex/net/ndnsim/a.cpp
    - Which prefix to use if multiple? Who makes the decision?
- Producer reply with encapsulated data
  - Need to know the ISP prefixes and register.
  - Think about a corporate network multihomed to several ISPs.
- How would selectors such as Exclude work if we modify the names?
Forwarding Hint Approach

• Keep the name intact: /net/ndnsim/a.cpp
• Consumer library/service ...
  – Look up the mapping system, get a LINK
    • /att/user/alex/net/ndnsim -> /net/ndnsim
  – Send interest with original name
    • Attach the LINK object to the Interest.
• Routers lookup /att prefix if no route to /net/ndnsim.
  – Better routing decision in the network
• Producer reply with original data
  – No change to the logic, no encapsulation.
  – Better caching, multicast, etc.
• Colluded content poisoning?
Name Discovery

• If a consumer supplies the complete name, we only need exact match between interest/data.
• Name discovery problem: how to find out the complete name?
• A complete name usually contains components that need to be dynamically discovered.
  – E.g, Version, local context.
• Can we accomplish the discovery at the app layer rather than the network layer?
  – So the network layer only needs to support exact match.
Example: discovering versions

• When an app doesn’t know the exact version number
  – E.g., the latest version of /nytimes.com/frontpage.
NDN’s approach
• Allow the consumer to ask a vague question, i.e., an incomplete name.
  – E.g., /nytimes.com/frontpage/latest
• Any answer with a longer name will do.
  – E.g., /nytimes.com/frontpage/latest/v6
• Consumer uses selectors to narrow down to the data that it wants.
Alternatives

• Manifest
  – Publish manifest file that contains the complete names of all versions of /nytimes.com/frontpage. Retrieve the manifest first, then request desired page using complete name.
  – However, the manifest itself is just another piece of data, how to discover the latest version of the manifest?
• Can I request /nytimes.com/frontpage/latest and get the current latest page in return with the exact same name?
  – Then the page you got today and yesterday have different contents but share the same name.
• How about each node (cache) runs a service that periodically announces what contents it offers over the network?
  – Need to run this directory service
  – Doesn’t work in wide-area network due to broadcast.
What can we do?

• Apps: minimize the use of name discovery.
  – E.g., limit it to manifest. The bulk retrieval is done using complete names.

• Routers
  – Can core routers ignore selectors?

• Architecture: examine existing selectors
  – Do we need them? Any better way to achieve the functionality?
Can core routers ignore selectors?

• Approach One
  – Core routers skip selector processing, skip CS for these packets, forward *all* interests carrying selectors without interest aggregation.
  – Edge routers and producers will still evaluate selectors.
  – May increase bandwidth use, and consumer delay, but should not impact system correctness.

• It works most of the time, but has a problem under certain conditions.
• C1 and C2 are sending interests with the same name but different selectors.
• C2 could get starved under certain circumstances.
Can core routers ignore selectors?

• Approach Two
  – Consumer appends the hash (H) of the selector field to the interest name (N) to make it /N/H.
  – Router processing has no change.
  – Producer sends data /N/x back by encapsulating it under name /N/H/x.

• No need to change the forwarding behavior, but consumers/producers need to agree on the naming convention.
Next step

• Write these up
• Implement and experiment
• Add a ContentType for encrypted data?
• Mobility support, especially producer mobility.
• The shim layer
  – Hop-by-hop fragmentation/reassembly
  – Detect loss on a link, retransmission.
  – Carry network NACK