NDN applications need libraries that offer capabilities beyond basic Interest / Data exchange.

But what application patterns to support?
NDN as an Evolution of Communication Abstraction

Telephone Network:
Focused on building the wires

Internet Protocol (RFC791):
Focused on delivering packets to destination node

NDN:
Focusing on retrieving data
Abstracting away the notion of “node”
Superset of node-to-node communication model
<table>
<thead>
<tr>
<th>Distributed Applications</th>
<th>IP Challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaboratively Developed</td>
<td>Limited application semantics represented in the network architecture</td>
</tr>
<tr>
<td>Iteratively Deployed</td>
<td>IP networks can be brittle to change or require NAT, etc. to scale</td>
</tr>
<tr>
<td>Dynamically Assembled</td>
<td>Connection- / session-oriented models, as well as address assignment requirement</td>
</tr>
<tr>
<td>Physically Integrated</td>
<td>IPv6 provides “room” but doesn’t aid application development</td>
</tr>
<tr>
<td>Asynchronously Experienced</td>
<td>Mobility and multicast not well-supported</td>
</tr>
<tr>
<td>Globally Accesssed / Integrated</td>
<td>Perimeter- and channel-based security model presents challenges</td>
</tr>
</tbody>
</table>
# Authoring Distributed Applications

<table>
<thead>
<tr>
<th>Distributed Applications</th>
<th>NDN Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaboratively Developed</td>
<td>Express application semantics in network naming, down to packet level</td>
</tr>
<tr>
<td>Iteratively Deployed</td>
<td>Reduce network brittleness to configuration change</td>
</tr>
<tr>
<td>Dynamically Assembled</td>
<td>Connection-less, session-less communication, leveraging storage in the network</td>
</tr>
<tr>
<td>Physically Integrated</td>
<td>Consistent, meaningful addressing for virtual and physical components</td>
</tr>
<tr>
<td>Asynchronously Experienced</td>
<td>Disruption tolerant and multipath-friendly</td>
</tr>
<tr>
<td>Globally Accessed / Integrated</td>
<td>Data-centric rather than perimeter-centric security</td>
</tr>
</tbody>
</table>
The Premise: Our Typical Application Design Goals

1. **Support common data access pattern(s) in the namespace:**
   Let the naming do as much “work” as possible.

2. **Avoid connection, session and host semantics:**
   Decouple production from consumption when feasible.

3. **Leverage / anticipate in-network caching and persistent storage:**
   Consider the network a distributed data store.

4. **Use signatures for provenance and encryption for access control:**
   Do not rely on channel- or perimeter-based security.

5. **Write for realistic conditions and the end-to-end principle:**
   Handle retry of dropped Interest and Data packets, etc.
   (We’re won’t cover this here, but library support is being considered.)
Assumption

Basic awareness of the NDN protocol and conventions, including fields of Interests & Data packets (e.g., Child Selectors) and naming conventions for versioning and segmenting.

(At least, very quickly, go look at

http://tinyurl.com/ndn-interest
http://tinyurl.com/ndn-data
http://tinyurl.com/ndn-conventions

while I am talking.)
Basics of the API

PRODUCER

Announce name prefix(es).
Name and sign Data packets.
Answer Interests in prefix(es).

REPOSITORY

(Optionally) provide persistence.

CONSUMER

Express Interest packets for data by name.
Receive Data, verify signature, decrypt if necessary.

ROUTER

Route directly based on (hierarchical) prefixes, inherently multicast.
Remember where Interests came from, return data along that path.
Verify packet signatures along the way.
Application Pattern Examples
(from implementations)

Recent patterns
• Fetching “latest data” via exclusions, child selectors, freshness.
• Rely on local repositories to answer network requests.
• Use index namespaces that match data common query types.
• Control / RPC Style commands with authenticated interests.
• Namespace overloading: data description, routing, trust.
• Namespace sync for many-to-many sharing.

Being explored
• Progressive discovery in 3D virtual spaces and beyond.
• Sampling the network for “live” time series data.
• Blending hierarchical and set-based naming.
Naive fetching of latest data in a time series: Occupancy Sensing

1. Express “discovery Interest”
   
   \(<\text{root}>/tv1/occupants\)

   **Returns**

   \(<\text{root}>/tv1/occupants/<\text{person\_id}>/<\text{tracktime}>\)

   \{
   "id": "9561",
   "status": "present",
   "x": "-78.2",
   "y": "-337.1",
   "z": "177.3",
   "tracktime": "1320274208.54",
   "trackcount": "617517"
   \}

   **Signed by public key of tracking system.**

   **Encrypt with a shared key for access control.**

2. Re-express discovery interest with “exclusion filter” to get other occupants.

3. Express interest in specific occupants with “rightmost child” flag to get current status.

4. What about Data freshness?
Handling the data publishing step:
Many apps can “push” to local repo instead of responding directly

Rely on namespace for “queries.”

**Diagram:**
- **ACQUIRE A/V**
- **STORE TO REPO**
- **ENCODE (Future: SVC)**
- **STORE TO REPO**
- **EDGE ROUTER W/CONTENT STORE**

Interest for 00:01:40.20 or Segment 2981
Use index namespaces to support random-access needs

/ndn/ucla.edu/video/<version>/video0/h264-1024k/

Timecode (frame-based) names make random access straightforward.

Very helpful for applications.

Adding synchronized tracks, such as metadata, is easy and backwards compatible.

Exploration: Retrieval of “live” time series data as (Adaptive) Sampling of the Networked Storage
Exploration: Retrieval of “live” time series data as (Adaptive) Sampling of the Networked Storage
Control / RPC-style commands with Authenticated Interests

Authenticated Interests

- Leverage opaqueness of names.
- Asymmetric keys to work directly with pub-key based identities.
- Symmetric keys and HMACs for fast ‘signatures’.
- Protect against replay attacks with state / counter
- Command privacy by encrypting non-routable portion of interest name
- (More efficient techniques in our NOMEN 2013 paper.)

Interest signed by app
UCLA/boelter/3551/lights/fixture/41/rgb-8bit-hex/FAF87F/<state>/<authenticator>

Synchronization of namespaces

Recall step #1 of sensing example – Enumerate /<root>/tv1/occupants

“Sync” - The key idea: separate state-sync from data delivery
• Synchronize everyone’s knowledge about data production by comparing hashes of namespace trees.
• Data can be retrieved in regular NDN ways whenever the app wants.

ChronoChat chatroom example:
• Each app computes a digest of the chatroom dataset as known
• Each app exchanges that digest with others by broadcasting an Interest packet containing its digest
  “if your digest differs from mine, tell me what new data you have”
• If anyone generates new data, reply with new data names.

Namespace synchronization examples

**ChronoShare**

**ChronoChat[[-js]]**

Source for these & other applications (ChronoSync is the library):
github.com/named-data

Good paper to read for motivation:
Sync in Limited Broadcast Domain, Then Fetch
How to name & sync data in a large virtual space?

Challenge: progressive discovery in a P2P gaming environment;
syncing subsets of knowledge...

Naming Option - Hierarchical Spatial Partitioning

Tanin, Harwood, and Samet (2007) describe a distributed quadtree index hosted in a Chord-based P2P network. Could similar principles be applied for spatial data in NDN? e.g., Kumar et al., (2012), in their SIGCOMM CarSpeak paper, empty octree-named data. These cases, however, don’t aim to reconcile peers as much as distribute queries: the former via a DHT, the latter relying on one-hop local communication.

Pau’s approach with Cantor Pairing (last talk) has potential, but we need to easily extract representations for 3D spheres.

So, we continue to consider the case of a moving Area of Interest in virtual space.
Consider data objects that can be “named” (indexed) by their feature vectors or spatial coordinates: opportunities for search, sorting, and subset sync (recall that there is a canonical ordering to NDN name components).

- Name objects with a locality sensitive hash of their feature vectors.
- Objects with identical features can be expressed as multiple children of the same parent.
- Use sync-style messages among peers interested in the same area to reconcile $k$ nearest neighbors to a query in the hash namespace.
- Related to Cantor factoring approach taken by Pau but better for kNN.
- Probably our approach will use a combination of hierarchical names and hashing.
Question via the network: **What is the simulated precipitation distribution between January 6 and February 2, 1902?**

**Current approach**
1) Determine where the two files reside
2) Perform conversion from human-readable dates to filenames
3) Extract the relevant range on each server
4) Fetch the extracted subset
5) Assemble into a final data set

**NDN approach**
1) Put the data into NDN-accessible storage as it is generated
2) Name it to support application data access pattern(s)
3) Send packets to fetch data needed

This example was covered by Prof. Papadopoulous, so we will skip it in the talk.
NDN Approach

**Consumer Request**
- Client wants Jan 30 – Feb 02
- NDN routes Jan requests to Server1 and Feb requests to Server2

/\cmmap/precipitation/
  \GCRM/GridZ/
  \horiz_resolution\/<field>\/<date>/<time>\

**Publisher Announcement**
- Server 1 advertises January prefix
- Server2 advertises February prefix
- Routing protocol propagates announcements
- Servers answer at appropriate granularity for the application
Challenge: Names have more than one use

Overloading naming:
  Data description / query
  Routing
  Trust

Conflicts can be handled.
  • Routing/description via forwarding hints (and perhaps eventually blended hierarchical/set-based namespace).
  • Trust/description via cert chain starting in the data object.
  • Routing/trust via enterprise policy.
Blending hierarchical- and set-based naming

Back to sensing example

• Electrical demand monitoring: different groupings for different apps
  
  /ucla.edu/bms/melnitz/1451/electrical/A/I_peak
  /ucla.edu/bms/electrical/melnitz/1451/A/I_peak

• Lighting control: “symbolic links”
  
  /ucla.edu/bms/melnitz/1451/light/04A10B89
  /ucla.edu/ftvdm/tv1/shows/all_my_children/light/west_key/04A10B89

Shang, Marianantoni, Horn, Burke.
Blending hierarchical- and set-based naming

Building Automation & Management

- Electrical demand monitoring: different groupings for different apps
  
  /ucla.edu/bms/melnitz/1451/electrical/A/I_peak
  /ucla.edu/bms/electrical/melnitz/1451/A/I_peak

- Lighting control: “symbolic links”
  
  /ucla.edu/bms/melnitz/1451/light/04A10B89
  /ucla.edu/ftvdm/tv1/shows/all_my_children/light/west_key/04A10B89

Names are both a characterization of a data object when it is created and a query when it is retrieved. Blend hierarchical & set-based naming?

Where will library support go?  NDN Platform.

- **NDN-CCL**: Common Client Libraries
  - NDN-CPP for C++, PyNDN for Python, NDN-JS for Javascript: start with basic, wire-format independent API.
  - Incorporate basic security library for name-based policy;
  - Include over time other advanced features supporting common application patterns.

- **Software router options**
  - NDNx: Fork of the CCNx software router incorporating NDNLP, forwarding speedups, and other enhancements.
  - NDNFD: Wire format compatible router with modular architecture for strategy experimentation.

- **NDNSim**: NDN Simulator.
- Other tools (e.g., dump, ping, traceroute).
- Other platforms (e.g., NDN on Node, NDN in the browser).
- Package installation support on Mac OS X and Ubuntu.
First ‘mature’ platform release targeted for Nov 2013
In the meantime...

Github repository
github.com/named-data

Preliminary platform release v0.1
named-data.net/codebase/platform

For more info:
Jeff Thompson – jefft0@remap.ucla.edu
Jeff Burke – jburke@remap.ucla.edu
Lixia Zhang – lixia@cs.ucla.edu