Sharing digital objects using NDN: PID interoperability, planning and scaling
SeaDataCloud

- SeaDataCloud is a distributed marine data infrastructure network in different geographical domains
  - 8 institutes with over 100 data centers
  - Aiming to make research data available to scientists
- Sharing large data sets becomes a challenge
  - Congestion
  - Interoperability
Figure 1: Current SeaData cloud setup
Figure 2: Potential solution
Research question

- How to make the Persistent Identifier (PID) and NDN (Named Data Networking) namespace interoperable?
  - How to support different PID types?
  - How to incorporate extensibility for future PID schemes?
- How to plan and scale an NDN network?
  - Which NDN scaling problems are known?
  - Which method can be used to plan an NDN network?
  - How to deploy an NDN network in a scalable way?
Outline

● Short introduction about NDN and PID
● Related work
● System architecture and virtualized NDN functions
  ○ PID interoperability
  ○ Virtual NDN planning, automation and scaling
● Experiment results
● Conclusion and future work
Why NDN?

- NDN is the most mature variation of ICN
  - ICN = Information Centric Networking
  - ndn-cxx solution was used in our proof of concept
- Forwarding based on name prefixes rather than IP
  - No end-to-end connections needed
  - Data cached on intermediary hops

![Figure 3: IP versus NDN](image)
## PID types

<table>
<thead>
<tr>
<th>PID Types</th>
<th>PID Type Identifier</th>
<th>Delimiter</th>
<th>Authority</th>
<th>Delimiter</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>URN</td>
<td>urn</td>
<td>:</td>
<td>&lt;NID&gt;</td>
<td>:</td>
<td>&lt;NSS&gt;</td>
</tr>
<tr>
<td>HANDLE</td>
<td>handle</td>
<td>:</td>
<td>&lt;Handle Naming Authority&gt;</td>
<td>/</td>
<td>&lt;Handle Local Name&gt;</td>
</tr>
<tr>
<td>DOI</td>
<td>doi</td>
<td>:</td>
<td>10.&lt;Naming Authority&gt;</td>
<td>/</td>
<td>&lt;doi name syntax&gt;</td>
</tr>
<tr>
<td>ARK</td>
<td>ark</td>
<td>:</td>
<td>&lt;NAAN&gt;</td>
<td>/</td>
<td>&lt;Name&gt;[&lt;Qualifier&gt;]</td>
</tr>
</tbody>
</table>
Related work

- Rahaf Mousa
  - Focused on DOI > NDN
  - Concluded that PID > NDN is possible
  - Most optimal caching strategy in NDN

- Andreas Karakannas
  - For every PID type a PID > NDN mapping server
  - States:
    - "PID > NDN mapping will be highly depended on the clients NDN browser which will need to be updated every time new rule would be appeared or changed"

- Spiros Koulouzis et al.
  - NaaS4PID
    - Supports one PID type
PID → NDN namespace interoperability

- Translation is transparent to the user
- Support for multiple PID types
- Extensible with future PID types with different naming schemes

Handle: [http://hdl.handle.net/]20/5000/481/objects/example_object
NDN: /ndn/handle/20/5000/481/objects/example_object

NDN: /ndn/urn/anp/1938/10/01/2/mpeg21
**PID → NDN model**

**Figure 4**

1. Open socket
2. Send PID request to Gateway

6a. Send NDN name (cont. step 7a) or PID link to client (cont. step 7c)

7b. Insert PID object in NDN

7a. Request NDN name

8. Retrieve object from NDN cache

9. Send NDN object to router nearby client

10. Send NDN object to client

**GW:**

3. Translates PID to NDN name

4. Checks if object is available in NDN

5a. If available, send NDN name to client (step 6a)

5b. If not available send request to PID server (step 6b) and client (step 6a)

**PIPID model**

Green: initial path
Blue: path if object is available in NDN
Red: path if object is not available in NDN
Proof of concept
How to make NDN scalable and software definable?

• Kubernetes
  ○ Open-source container-orchestration system
    ■ Deployment
    ■ Scaling
    ■ Management

• SDN-style control
  ○ Centrally deploy and configure containers (NDN functions)
    ■ Add roles (routers)
    ■ Configure routes
    ■ Allocate resources
Architecture drawing - Proof of concept

Figure 5
How to plan the NDN network

- The challenge becomes
  - How to manage/plan/deploy such a diverse infrastructure?
- Single description to plan and deploy needed
  - Is there an open standard available?
How to plan the NDN network (TOSCA)

- - -

● What is TOSCA?
  ○ Topology and Orchestration Specification for Cloud Applications
  ○ Declarative Domain Specific Language (YAML/XML)
  ○ TOSCA descriptions → orchestrator
  ○ Used to describe complete lifecycle
    ■ **Hosts** (bare metal, VM, containers)
    ■ **Software components** (applications, databases, middleware)
    ■ **Network components** (load balancers, gateways, VNF’s)

● TOSCA is agnostic towards orchestrators
  ○ DRIP
  ○ OpenStack
  ○ And gaining popularity
Different types in TOSCA to describe building blocks

- Eight different types to use
  - Node
  - Relationships
  - Artifacts
  - Capabilities
  - Interface
  - Groups
  - Policies
  - Data

- Node
  - Host, container, VM, etc.

- Relationships
  - Connects nodes to each other
  - dependsOn, hostedOn, connectsTo

- Interface
  - Set of hooks
  - Actions to: Create, configure, start, stop or delete
Figure 6: TOSCA diagram
How to make NDN software definable? (Kubernetes)

```
spec:
  hostname: ndn-router-1
  nodeName: mulhouse
containers:
  - image: aqualilte/ndn:router3
    name: ndn-router1
    env:
      - name: gateway
        value: ndn-producer-2
      - name: routes
        value: /ndn/handle /ndn/ark
      - name: protocol
        value: tcp
```
Cluster status

Consumer 1
NDN -> PID -> NDN

Consumer 2
NDN

Copy area

# Handle URL
20.300.481/data/objects/object100M

# Client scripts
python3 ndn_client.py
python3 pid_client.py
Conclusion

---

- **Deployment planning**
  - TOSCA can describe complete lifecycle of infrastructure
- **Easy scaling out to other clouds**
  - VM’s used to allocate/deallocate resources in the cloud
  - Kubernetes used to scale in/out the application (NDN)
  - Bringing data closer to the user decreases latency and chance of congestion
- **Interoperability between different PID types is possible**
  - Adding new PID types is low effort cost
Future work

- TOSCA blueprints are conceptual
  - The VM and Kubernetes was deployed manually
  - Full implementation developed needed with an orchestrator such as e.g. DRIP
- NDN is still experimental
  - Explore performance bottlenecks (benchmarking)
  - Test routing protocols (e.g. OSPFN)
- Extent Kubernetes with intelligence
  - Where to deploy NDN routers (containers)?
- Incorporate the PID → NDN translation into NDN software natively
Questions?
Performance of proof of concept setup

NDN/UDP vs NDN/TCP vs TCP/IP 100MB object

- TODO: Graphs of NDN vs TCP/IP (boxplot or barplot)
- TODO: Explain why the performance differs
Performance of proof of concept setup

NDN/UDP vs NDN/TCP vs TCP/IP 1000MB object

The box plot shows the performance comparison of PID (TCP/IP), NDN (UDP), and NDN (TCP) for a 1000MB object. The Y-axis represents milliseconds, with less is better.
Performance of proof of concept setup

- Difference in percentage
  - 100MB file:
    - NDN (UDP) vs PID (TCP/IP): 27%
    - NDN (TCP) vs PID (TCP/IP): 150%
    - NDN (TCP) vs NDN (UDP): 98%
  - 1000MB file:
    - NDN (UDP) vs PID (TCP/IP): 18%
    - NDN (TCP) vs PID (TCP/IP): 24%
    - NDN (TCP) vs NDN (UDP): 5%
NDN performance bottlenecks

- **Underlay (TCP/IP)**
  - UDP vs TCP
  - MTU sizes
- **Processing problems in software**
  - Slow packet decode functions (35.4% time spend on decoding)
  - Long names can degrade performance
- **Named data forwarding scaling**
  - Routing table sizes
  - Forward strategies
- **Named data caching scaling**
  - Cache strategies + size
    - LCE (Leave Copy Everywhere)
  - Cache replacement strategies