NFN – Recent Work on Expression Forwarding

Christopher Scherb · Claudio Marxer · Christian Tschudin

University of Basel, Switzerland

ICNRG Interim Meeting @ IETF 104 Prague, 24 Mar 2019
ICN generalization: Deliver cooked results instead of raw data

Idea:

– User defines computation / workflow
– Network finds execution location
– Similar to Serverless Computing, but offers workflow orchestration
– Implemented using Interest / Named Data Object Transport Layer
PiCN - pure Python Implementation of NFN

- code available at: https://github.com/cn-uofbasel/PiCN
- includes an ICN Forwarder
- executes Python code under network control
  - decomposes complex expressions “in the network”
  - optimizes by migrating either code and/or data
    (more about this in this talk)
- since recently also for native Intel x86 code
- previous Scala (JVM) support could be added again
Example

Request: <lambda expression>

PYTHON

```python
func =
def func(a):
    <code>
```

Client

Input Data 1

Compute

Here

Function

Code

Input Data 2

Fetch

Input Data 1

Fetch

Compute

Here
PiCN (cont.)

- Software contains:
  - NDN and NFN Forwarder
  - client and command line tools
  - repo

- modular, easy to extend

- simple simulation and debugging system
Outline

- Basic NFN Principles (what we want to do)
- Layering Overview
- Core of NFN: expression rewriting
- Example: expression rewriting based on FIB information
- Example: expression rewriting based on mobility patterns
- Example: expression rewriting based on pricing
- Orchestration of "Plans" and creating "Templates"
- More research topics: result authenticity, compute privacy
Architecture / Core Components

- NFN Execution: Execute Function Code
- NFN Rewriting: Decide where to compute
- ICN Transport Layer: Forward Messages
CCN/NDN Layer

- Basic CCNx/NDN Forwarding
- Hierarchical Names
- Forwarding based on Longest Prefix Matching
- Name in Interest Message contains reference to single Named Data Object

Components of a name

<table>
<thead>
<tr>
<th>NFN Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFN Rewriting</td>
</tr>
<tr>
<td>ICN Transport Layer</td>
</tr>
</tbody>
</table>
NFN Rewriting 1

- Workflow - consisting of **Function Calls** on **Input Data**
- Encoding the Workflow in CCN/NDN **Names**
- Using $\lambda$-calculus to describe workflow
- Name in Interest Message contains references to multiple **Named Data Object**
- Magic: **Longest Prefix Matching** and **Name Rewriting**
NFN Rewriting 2: Name Encoding Example

Workflow:
/func/f1 (/data/d1, /func/f1 (/data/d2) )

Names:
/func/f1
/data/d1
/func/f2
/data/d2

Choose One Name to Prepend
/data/d1

NFN - Name Encoding

Name Components:
data
d1
\[ \lambda \ x. \ /func/f1(x, /func/f2( /data/d2 ) ) \]

NFN Execution
NFN Rewriting
ICN Transport Layer
# NFN Execution

- Store **Function Code** in Named Data Objects
- Any **NFN node** can execute **Function Code** by fetching it over ICN
- Requesting missing **Named Data Objects** and **Function Code**
- Requires Safe Execution Environment (Sandboxing)

| NFN Execution | NFN Rewriting | ICN Transport Layer |
NFN Interest Handling
Rewriting Decisions

Interesting Part:

Which Name to prepend in front of the computation (Rewriting)

– Because: Influences where to forward a computation
– Defined by a Rewriting Strategy

  Should a node forward a computation or execute it locally?

– Determines where to compute a result.
– A node can split a computation into subcomputations ⇒ parallel/distributed execution
Rewriting Strategy

- Simple Rewriting Strategy - inspired by Hadoop
- Goal: Reduce load on links
- Forward a computation request to the input data (prepend input data)
- Start computation if the prepend data are available on the node.
- Transport function code to input data
Advanced Expression Rewriting for NFN

- Add additional information to improve the rewriting decision
- Scenario dependent and independent
- Use the FIB to decide which name should be prepended
- Create AST and search for independent subcomputations
- Split the computation if subcomputations are forwarded to different nodes
- Good for Map Reduce and Parallel Execution
NFN-Expression Rewriting based on FIB Information - 2

call /f1

independent calls

call /f2

independent calls

/d1 /d2
call /f3
call /f4

independent call/data

/d3 /d4 /d5
NFN-Expression Rewriting based on Mobility Patterns

- Use information about mobility patterns
  - e.g. Node is **Edge Computing Node**
  - e.g. **Neighbor Node** has no computational capabilities
- Execute even if **prepended data** are not on the node
NFN-Expression Rewriting based on the Price - 1

- Request **Meta Information** about file size, bandwidth or load
- Compute a **Plan** for the **cheapest** execution possibility
- Storing **Plans** in **Named Data Objects**
- **Caching** and **Reusing** of Plans (for subcomputations)
NFN-Expression Rewriting based on the Price - 2

IM: <expression>
Plan
<Name to prepend>
Subplan1
Subplan2

NFN Forwarder

IM: <expression>
Plan
Subplan1
Subplan2

NFN Forwarder

IM: <subexpression1>
Subplan1

IM: Function/Data

IM: <subexpression2>
Subplan2

FWD: prepend Name

Split computation
NFN-Expression Rewriting based on Templates - 1

- Create **Templates** based on the **Planning Process**
- Idea: Instead of creating a new **Rewriting Strategy**, let the network learn from previous situations (or simulation)
- Compare **AST** structure and names within the **AST**
- Introduce **Wildcard** Names into the **Plans**
- Based on Observation and Statistics:
  - $<\text{prefix}>/\text{name}-1 \ldots <\text{prefix}>/\text{name}-n$ → Action $a$
  - if $n > x$ → reduce $<\text{prefix}>/*$
- In future with Machine Learning?
A Template is a Tuple: \(< AST^*, Action >\)

A request is matched against the AST containing the Wildcards

**Request**

- call /func/f1
- call /func/f2
- /data/d1
- /d2
- call /func/f3
- /d3
- call /func/f4
- /d4
- /d5

**Template**

- call /func/*
- call /func/*
- /data/*
- /d2
- call /func/*
- /d3
- call /func/*
- /d4
- /d5
- call /func/*
- /d4
- /d5
Result Authenticity

- Data are secured by *signatures*, what about *results*
- Question: How can you know the result is correct?
- Idea: Signed by execution node.
- Add Signatures of the Input Data and the Function Code
- Item: No Proof, but enables users to find out which nodes are fooling
Data Security

- Question: How to deal with sensible input data?
- Intel SGX may be a way
- Homomorphic Computing (increases runtime)
- Only real solution: Do it local
Software

- PiCN
- https://github.com/cn-uofbasel/PiCN
- ICN and NFN Forwarder, Client Tools, Repository
- Modular, easy to extend
- NFN Rewriting Strategies as Plugins
- Simple Simulation System
References

- https://github.com/cn-uofbasel/PiCN
- An information centric network for computing the distribution of computations (M. Sifalakis, B. Kohler, C. Scherb, C. Tschudin)
- Access-controlled in-network processing of named data (C. Marxer, C. Scherb, C. Tschudin)
- Resolution Strategies for Networking the IoT at the Edge via Named Functions (C. Scherb, D. Grewe, M. Wagner, C. Tschudin)
- Execution State Management in Named Function Networking (C. Scherb, B. Faludi, C. Tschudin)
- A Packet Rewriting Core for Information Centric Networking (C. Scherb, M. Sifalakis, C. Tschudin)
- Smart Execution Strategy Selection for Multi Tier Execution in Named Function Networking (C. Scherb, C. Tschudin)
Question