Introduction to Named Function Networking

Claudio Marxer
<claudio.marxer@unibas.ch>

Christopher Scherb
<christopher.scherb@unibas.ch>

Christian Tschudin
<christian.tschudin@unibas.ch>

Computer Networks Group · University of Basel · Switzerland

Tutorial: Running IoT Applications over ICN · September 26, 2017
Users want Results, not Data!

**Trick:** “Name the Result” by specifying how to obtain it.
Users want Results, not Data!

**Trick:** “Name the Result” by specifying how to obtain it.

...and let the network orchestrates the result generation:
- Name Rewriting
- Disassembly into Sub-Computations
- Moving Code
- Executing Code
...
Example 1, Client’s Perspective for “get duration”

**Named Data Networking:** Distribution of named content (published)

lookup: /joe/NYmarathon/track.gpx

**Named Function Networking (NFN):** Generation of named content (on-demand)

lookup: /get/duration( /joe/NYmarathon/track.gpx )
Example 1, Client’s Perspective for “get duration”

**Named Data Networking:** Distribution of named content (published)

lookup: /joe/NYmarathon/track.gpx

**Named Function Networking (NFN):** Generation of named content (on-demand)

lookup: /get/duration( /joe/NYmarathon/track.gpx )
  
- **named function**
- **named content**
Example 1. Client’s Perspective for “get duration”

**Named Data Networking**: Distribution of named content (published)

lookup: /joe/NYmarathon/track.gpx

INTEREST[ name ]

---

**Named Function Networking (NFN)**: Generation of named content (on-demand)

lookup: `/get/duration( /joe/NYmarathon/track.gpx )`

- **named function**
- **named content**

INTEREST[ expression ]
Example 2, Network’s Perspective for “rank two runners”

Special NFN-capable nodes dissect the interest’s NFN name and orchestrate the result derivation.
Example 2, Network’s Perspective for “rank two runners”

Special NFN-capable nodes dissect the interest’s NFN name and orchestrate the result derivation.
From Name-Resolution (NDN) to Expression-Reduction (NFN)

<table>
<thead>
<tr>
<th>Realm</th>
<th>Instances</th>
<th>Network Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Named Data</strong></td>
<td>“classic” ICN, key–value store, DNS</td>
<td>“name resolution” (= lookup)</td>
</tr>
<tr>
<td>(access to data)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Named Functions</strong></td>
<td>“new” ICN</td>
<td>“expression reduction” (= processing)</td>
</tr>
<tr>
<td>(access to results)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Three Tasks for doing “reduction in the network”:
Locate (data, fct and exec place) / Run / Collect

e.g. Find a server close to the DB (instead of downloading tera bytes), execute there.

Network does not execute: NFN only orchestrates the computation
What is Really new in NFN?

– User-Formulated Content Names
  Building Blocks: Named-Functions, Content (static or dynamic)
  Glue: Extended $\lambda$-Calculus

– Provides On-Demand / Dynamic / Derived Content

– Requires NFN-capable nodes (doing the expression reduction):

  ![Diagram](NDN NDN NDN NDN)

  Purpose: Derivation of derived content

– Caching of computation results (since done in-network)
NFN’s reduction task, at a glance

\[ f( a, g(b) ) \]

a, b are content objects; f, g are named functions

Reduced by launching three activities:

– hunt for f
– hunt for a
– reduce \( g(b) \) recursively
A λ-calculus expression $E$ has one of three forms:

1. $E \overset{\text{def}}{=} a$  
   variable $a$

2. $E \overset{\text{def}}{=} f(e)$  
   result of function $f$ applied to expr $e$

3. $E \overset{\text{def}}{=} \lambda x.e$  
   a function defined by expr $e$ with parameter $x$

The last case will be important for the NFN tutorial track:
It permits to move a name (inside the expression) in front of that expression,
therefore influence the routing (as we will see).
Outline

– NFN Mindset

– Client’s Interface to NFN

– Implementation: nfn-scala
How to Map a NFN Expressions to one NDN Name.

Building Blocks:
- Named Content: Data to derive from
- Named Functions: Derivation procedure inside a content object
  - Side-effect-free mapping: Content object(s) $\rightarrow$ Content object
  - Bytecode of a JVM function.

Glue:
- $\lambda$-Calculus: Formal language to describe computations
- Additional call operator: To express a named function invocation.
The `call` Operator

call <num> <fct> <arg1> <arg2> ...

Examples:
call 2 /fct/wordCount "hello world"
call 2 /fct/wordCount /nice/poem.txt
call 2 /fct/wordCount (call 2 /fct/firstVerse /nice/poem.txt)
The call Operator

call <num> <fct> <arg1> <arg2> ...

Examples:
call 2 /fct/wordCount "hello world"
call 2 /fct/wordCount /nice/poem.txt
call 2 /fct/wordCount (call 2 /fct/firstVerse /nice/poem.txt)
NFN Expressions = Structured Names

Recipe to get a network name:
1. Reformulate: Prepend one of the network names (λ abstraction and application)
2. Split into Name Components
NFN Expressions = Structured Names

Recipe to get a network name:

1. Reformulate: Prepend one of the network names (\(\lambda\) abstraction and application)
2. Split into Name Components

Example (One Expression: Different Mappings \(\rightarrow\) Different Forwarding Behavior)

- Prepended function name:
  1. \(/fct/wordCount\ @x\ call\ 2\ x\ /nice/poem.txt\) (\(\lambda\)-expression)

- Prepended argument name:
  1. \(/nice/poem.txt\ @x\ call\ 2\ /fct/wordCount\ x\) (\(\lambda\)-expression)
NFN Expressions = Structured Names

Recipe to get a network name:

1. Reformulate: Prepend one of the network names ($\lambda$ abstraction and application)
2. Split into Name Components

Example (One Expression: Different Mappings $\rightarrow$ Different Forwarding Behavior)

- Prepended function name:
  1. `/fct/wordCount @x call 2 x /nice/poem.txt` ($\lambda$-expression)
  2. `/fct/wordCount/@x call 2 x /nice/poem.txt/NFN` (network name)

- Prepended argument name:
  1. `/nice/poem.txt @x call 2 /fct/wordCount x` ($\lambda$-expression)
  2. `/nice/poem.txt/@x call 2 /fct/wordCount x/NFN` (network name)
How to Implement a Named Function?

In our current system, named functions are implemented in Scala and published as JVM Bytecode (but could also be written in Python...)

```scala
class WordCount() extends NFNService {
  override def function(interestName: CCNName, args: Seq[NFNValue], ccnApi: ActorRef): NFNValue = {
    def splitString(s: String) = s.split(" ").size
    NFNIntValue(
      args.map{
        case doc: NFNContentObjectValue => splitString(new String(doc.data))
        case NFNStringValue(s) => splitString(s)
        case NFNIntValue(i) => 1
        case _ =>
          throw new NFNServiceArgumentException(s"No words to count!")
      }).sum
  }
}
```
Write your first named function in the following hands-on track!
Q & A