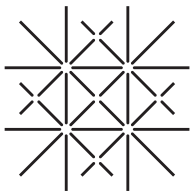
Named Functions

and Cached Computations

CCNx Community Meeting, Xerox PARC, Palo Alto, Sep 2013



Christian Tschudin and Manolis Sifalakis, University of Basel

Abstract

Starting from the observation that CCN is *resolving names to content*, we have embedded this behavior in a more general approach that we call “Named Function Networking”: In NFN, the network’s task is to **resolve names to computations** by reducing lambda expressions.

In the talk we will present an architecture framework and report on first experiences with our implementation of a call-by-name lambda calculus resolver for CCN style “memory access”.

We will demonstrate its resolution power beyond “content-pull” with applications like “mobile-code-drag” and “computation-push”, all using CCNx’s interest+content exchange pattern.

Finally, we also discuss how function resolution can be used to replace some of CCNx’s protocol features, leading to a leaner spec.

Overview

1. Interest$\hookleftarrow$Content = variable lookup

   CCN substrate as a memory plane

2. From NDN to **Named Function** Networking (NFN)

   applying named functions to named data

3. From "name resolution" to "resolving λ-expressions"

   putting the Lambda calculus inside CCN

4. Call-by-name: Routing and Resolution

   teaching Krivine's machine "to do CCN"

1. Interest↩Content = variable lookup

Observation: **CCN is a "name resolution engine"**

- CCN (and pubsub) as a superset of several resolution systems
  – DNS case (hostname-to-ip, maildomain-to-server)
  – hyperlink metaphor (link-to-html), and more
- **Caching** is a nice-to-have optimization (and selling argument), but **not the essence** of CCN.
- CCN's core is: **variable lookup**
  give a variable's name or address, get the assoc. memory content

When memory is added to the network,
the network becomes the memory.

2.a From NDN to Named Function Networking (NFN)

A simple use case of Named Function Networking:

Retrieve a video V in a format produced by transcoder T

- In NDN: download the parts, then apply

```
rawvideo = resolve( "name_of_V" );
transcoder = resolve( "name_of_T" );
cookedvideo = transcoder( rawvideo );
```

- In NFN: download final result

```
cookedvideo = resolve( "name_of_T( name_of_V )" );
```

i.e., leave it to the network to best “resolve” (= search or compute) this request.

2.b From NDN to NFN (continued): behind the façade

- NDN lookup optimizes for locating the variable's bits
  - – find the set of caches that keep the variable's bits
  - – pick fastest access (= usually closest cache)
- In NFN we have even more optimization potential:

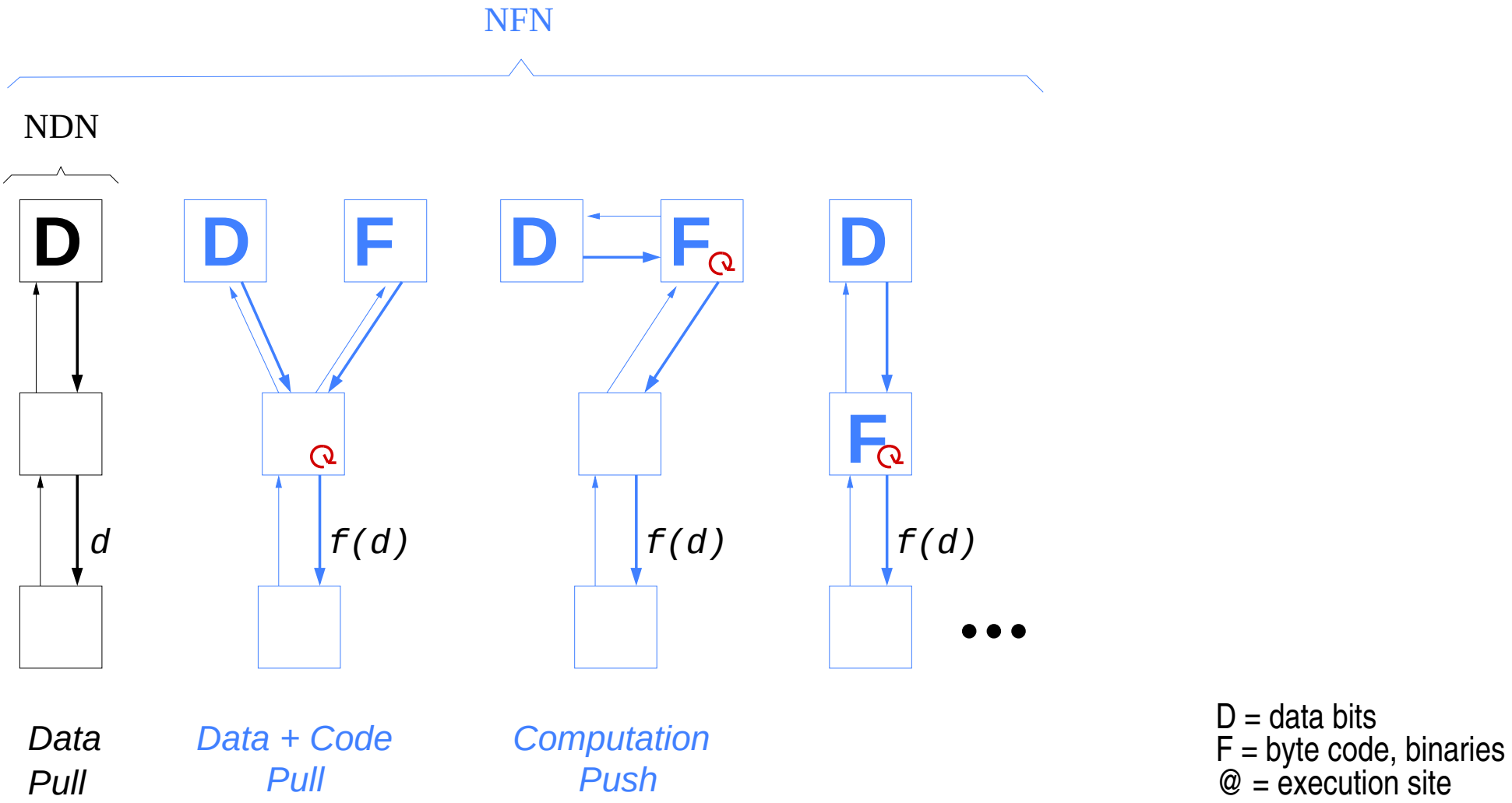
  a) **find** the set of caches that keep the (final or partial) result

  b) **or compute** the result:
    - find the set of caches that keep the variable's bits
    - find the set of caches that keep the function's code bits
    - find the set of suitable execution sites, apply and cache

  Pick the faster variant of a) or b)

→ trade CPU cycles for memory and access time!

2.c From NDN to NFN: behind the façade (continued)



3. From "name resolution" to "resolving λ-expressions"

[Here comes a slide on the Lambda Calculus in case you never attended a course on functional programming languages (LISP, Haskel, ML, etc)]

A λ-calculus expression E has one of three forms:

1. E $\stackrel{\text{def}}{=}$ a — variable a
2. E $\stackrel{\text{def}}{=}$ f(e) — result of function f applied to expr e
3. E $\stackrel{\text{def}}{=}$ λ.x e — a function defined by expr e with parameter x

The λ-calculus has the same expressiveness as a Turing machine.

Our approach adds case #2 and #3 to CCN

(and we relate to "call-by-name resolution" in progr. languages)

4. Call-by-Name: Routing and Resolution

Resolving arbitrary λ-expressions is not trivial:

- Call-by-name result (1980ies) from Jean-Louis Krivine
  today known as "Krivine's (lazy) machine"
- Krivine's machine is expressable in terms of another abstract machine called ZINC (Zinc-Is-Not-OCaml, for running OCaml programs)
- Our NFN resolution engine is based on ZINC, replaces all memory accesses with "Interest$\hookleftarrow$Content" actions:
  – a returning content msg carries on with the computation

We also add decision logic re execution places to the routing strategy.

5. Named Functions and Cached Computations

How can NFN cache intermediate and final results?

- New `FOX` instruction, internal to the NFN abstract machine
- `FOX` = **find-or-execute**
  - before launching resolution of expr E, attempt a lookup `resolve( hash(E) )`
  - if that fails: resolve E, store result under `hash(E)`, and return the `resolve(E)` result to client
- Possible optimizations:
  - do the lookup in parallel with a launch: the fastest wins
  - prepend name of function to `hash(E)`: helps in routing etc.

5.b Named Functions and Cached Computations

How do *clients* compose function and data names? Example:

- `f(g(data))` is mapped to a name with 4 components
  `[ ccnx:nfn | /name/of/data | /name/of/g | /name/of/f ]`
  Note that each name component is itself a CCNx name.
- Conceptually, this term inversion (data first) aligns with CCNx' longest prefix-match philosophy:
  – try to get the full `f(g(data))` result, or if that fails
  – try to get the partial result `g(data)`, or if that fails
  – try to get the raw `data` (and try to fetch `g` alone etc)

5.c Streamlining the CCNx Protocol with NFN

CCNx protocol spec requires filtering of replies:
exclusion, exact match of implicit hash.

- Filtering is an action which can be expressed as a program:

```
define filter(name_of_data, hash_value_to_match) (
   (ifelse (eq (sha256 name_of_data) hash_value_to_match)
      name_of_data
      nil)
)
```

- Let the network find the best place to execute this program
  (instead of each battery powered sensor node having to run this test)

With NFN, filtering can be removed from the CCNx protocol, adds futureproofness (e.g. replace the hash function)

Named Function Networking – Beyond the Data Dump

- NFN turns the net into an ICN cloud:
  - NFN as an *orchestration* language for logical+binary execution
  - in-core "protocol extensibility"
  - in-core information inference!
- Global optimizations:
  - caching data *and* computations (edge solutions can't match this)
  - on-the-fly computation (CPU vs memory), min delay target
- NFN supports two-sided tussles:
  - clients express what they want, information mash up
  - providers get control over their content, can add access logic

Summary

- Named Data was a first step: **name resolution**
  network can optimize on finding closest content store
- Named function is the next logical step: **expression resolution**
  network finds best execution site, it drags/pushes code and parameters, launches the execution

  NFN only *orchestrates* code execution, i.e. controls the code flow and picks the execution location.
- We implemented the NFN resolution engine, it invokes CCN's I and C-primitives, resolves arbitrary λ-expressions.

$\rightarrow$ visit us at the demo booth (CCN-lite, λ resolution engine for CCN)