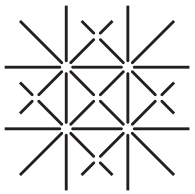
Named Functions

for Media Delivery Orchestration

PV2013 Packet Video Workshop
San Jose, CA USA, Dec 12+13, 2013



Christian Tschudin and Manolis Sifalakis, University of Basel

Abstract

[. . .]
In this paper we introduce a framework for orchestrating media distribution tasks in the spirit of [Content Centric Networking] by means of *named functions*.

[. . .]
The aim of named function networking (NFN) is to serve as a redirection mechanism that allows the network to allocate memory and computation resources in the most optimal way (as CDNs do for content replicas).

Raison d'être of Named Function Networking

Often, **having data** is **not** what you want

—

Using data is what you want!

Use (!) cases:
media consumption, sensor feeds, data analytics

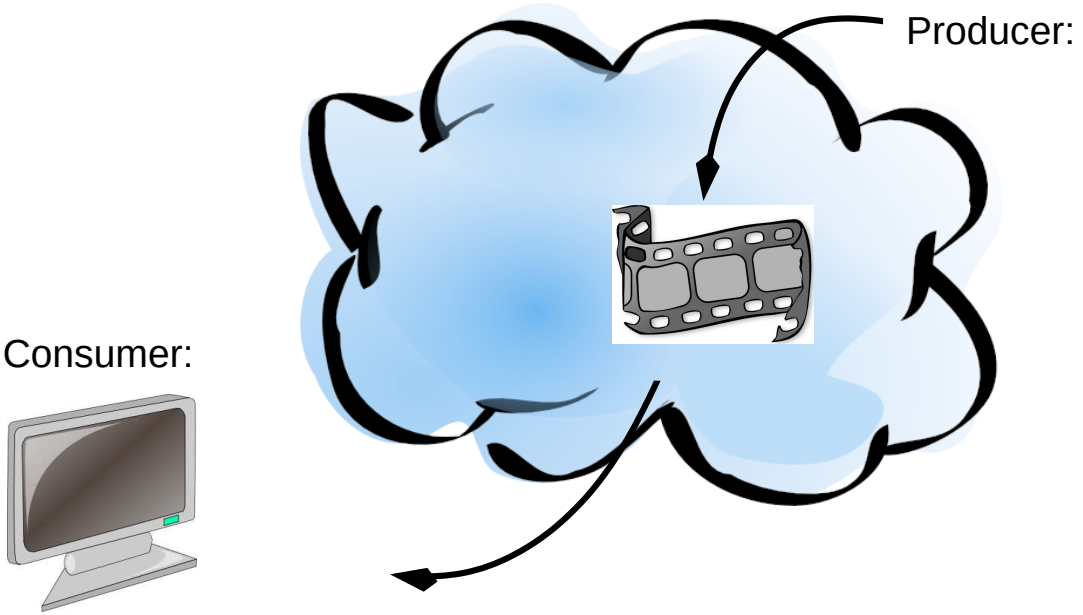
Content Centric Networking is wrong to think that the network is all about transferring **raw data** – it must deliver **cooked data**.

Overview

1. Introduction
   - named data networking (NDN)
   - named function networking (NFN)
   - programs as names
2. Four application scenarios
3. Implementing NFN with the λ-Calculus
4. Views on NFN, Conclusions

1.a) What is Named Data Networking?

. . . also called
Information Centric Networking
or Content Centric Networking

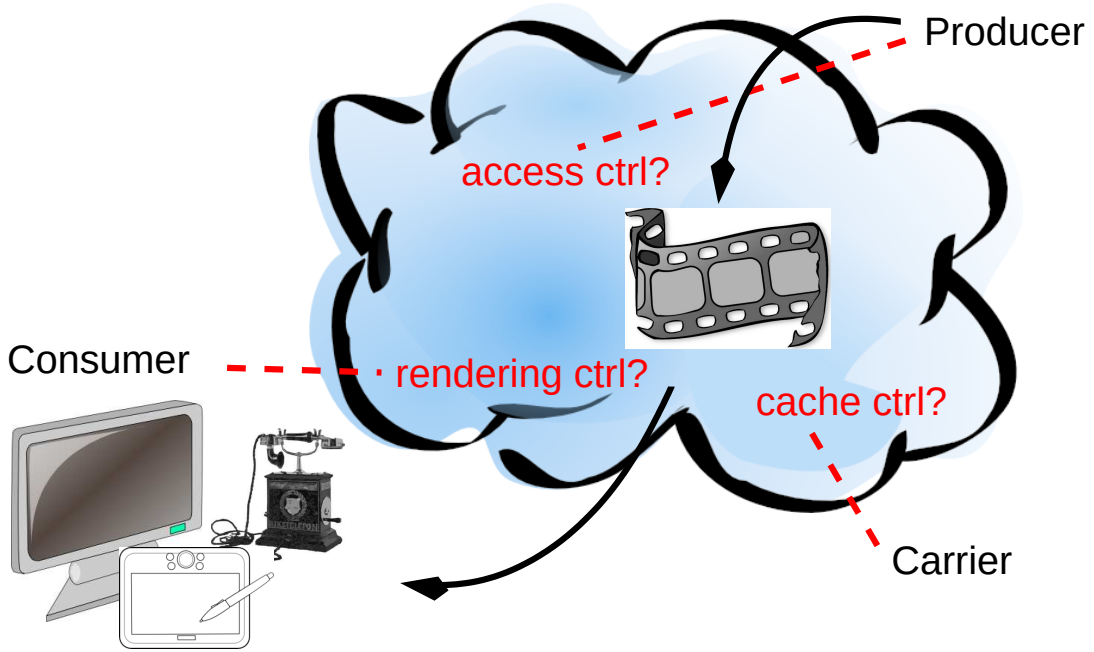


NDN tenets:

- Name the data, not the server
- The network transparently (pre-) caches data, recognizes names
- Network-plus-memory: **The network becomes the memory**

1.b) What is the problem of Named Data Networking?

Many places where customized control would be desirable.



With pure named-data-delivery:

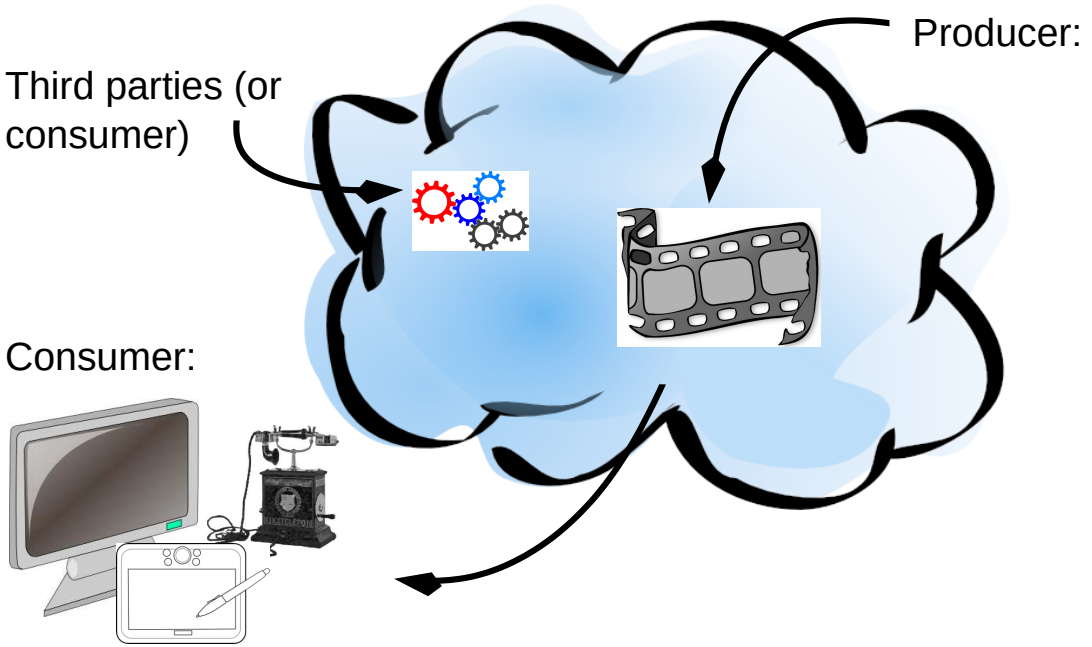
. . . media publishers cannot impose access control,
would have to generate and insert all possible media formats

. . . consumers at the mercy of publishers, have to do redundant rendering

. . . difficult to coordinate cached content across carrier boundaries

1.c) What is Named Function Networking?

Media content as well as function content



NFN tenets:

- Names for data **and** functions: name the result
- The network caches results, recognizes named results
- Network-plus-memory-plus-cpu: **The network is the computer**

1.d) Programs as Names

"name the result . . . the network recognizes named results" **– how?**

- Express the desired result as a chain of function invocations:
  – parameters: named data
  – named functions
  – procedures: permit on-the-fly definition of anon. functions
- Use **"functional expressions" as first-class (content) names**

Later in the slide set: λ-Calculus to our help

From Wikipedia: *"Lambda calculus is a formal system [. . .] for expressing computation[s]"*

2. Four Application Scenarios

Introducing NFN with media delivery tasks:

(a) Media transcoding

(b) Access control

(c) Dynamic (active) content

(d) Cache Control with MapReduce

2.a) Media Transcoding à la carte

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) Conditional Access to Media

Instead of plain `/the/media`, this "access name" is given to clients:

```
( define playmedia(x) ( ifelse (/ca/auth x) (/codec/mpeg x) nil )
  playmedia( /the/media )
)
```

Partial sequence of reduction steps:

→ λx.((ifelse (/ca/auth x) (/codec/mpeg4 x) nil)) /the/media

→ (ifelse (/ca/auth /the/media) (/codec/mpeg /the/media) nil)

→ (/ca/auth /the/media) (/codec/mpeg /the/media) nil

→ (λxy.x) or (λxy.y) *which stands for True or False*

Function /ca/auth can be made to execute on trusted places only.

2.c) Dynamic (active) Content

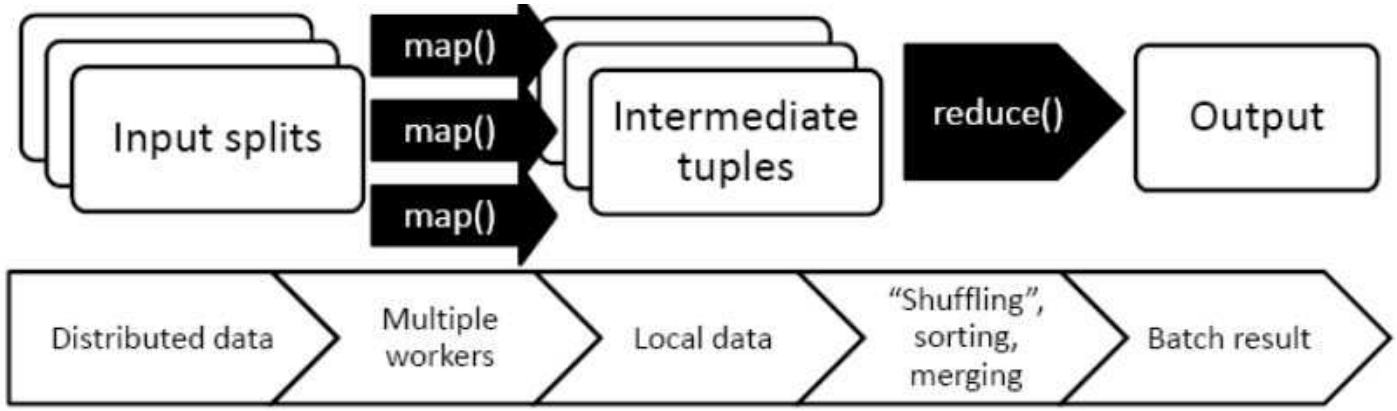
Data carried with prog is handed out except if a newer version exists

```
( define newer_ver (x) (
     ifelse (rightSiblingExists x)
       (rightSibling x)
       tailreturn
   )
   newer_ver( /the/media/ )
) PAYLOAD_GOES_HERE
```

`rightSiblingExists()` could be “pinned down” and the publisher to never register the result, which effectively disables caching.

2.d) MapReduce (and Cache Control)

MR computation pattern known since long (in LISP: Map and Fold)



Mapper and reducer fcts can be passed by name, network applies the MR pattern.

Example: Learning the regional top-ten content; a) mappers ask all neighbors for top-ten names and frequency, b) reducing step sorts results, selects top-ten. Action: Tell neighbors to keep only this list.

3.a) How NFN works: λ-Calculus

λ-calculus recap slide, also for functional programming novices (LISP, Haskel, 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 $\stackrel{\text{def}}{=}$ λx.e — a function defined by expr e with parameter x

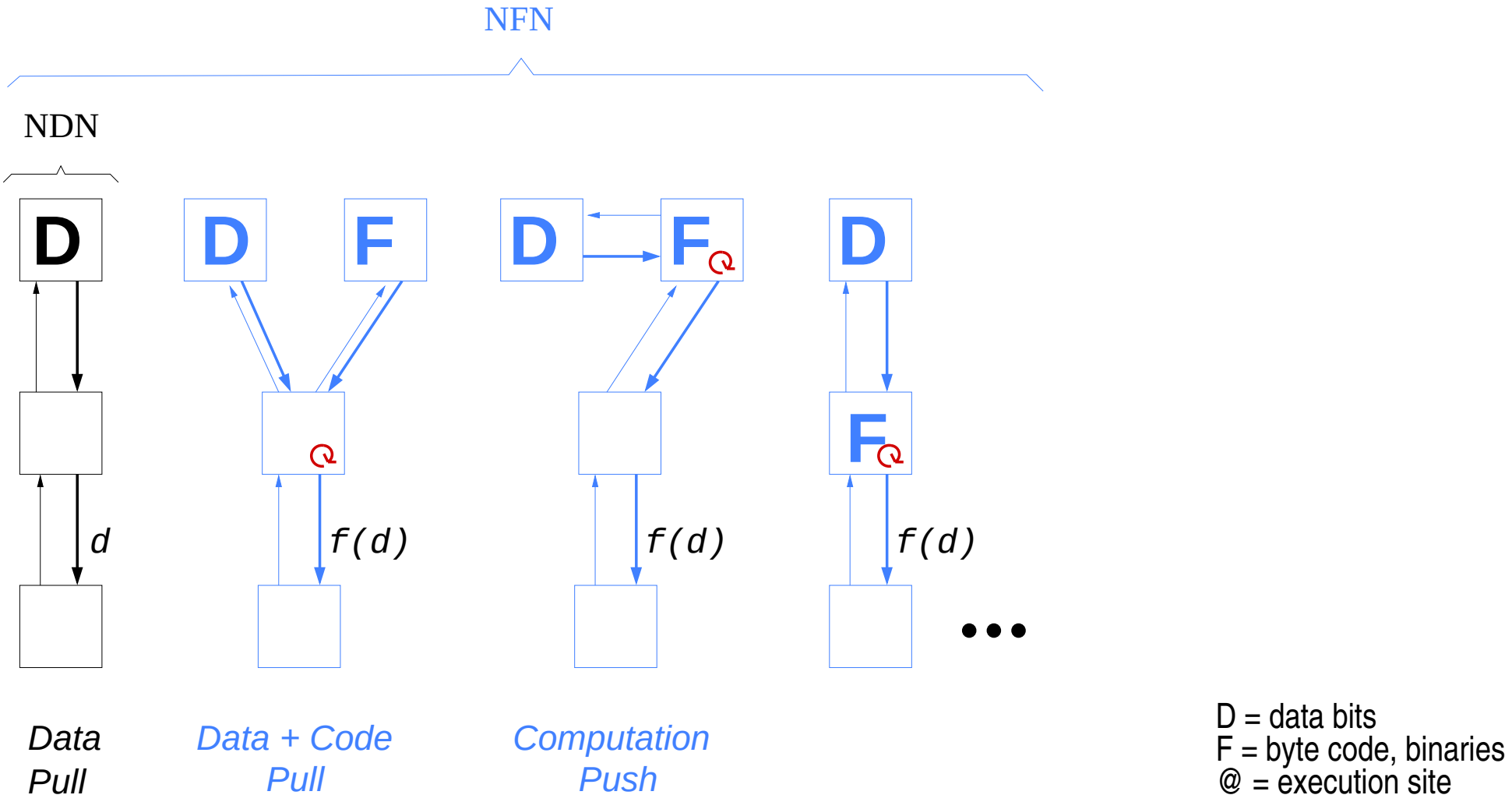
- NDN is doing a "variable lookup" (case 1):
  fetch the content that is bound to a given name
- Our NFN approach adds cases #2 and #3 to NDN

3.b) Executing λ-Expressions: Reduction Steps

Consider `f( a, g(b) )` (where `a, b, f` and `g` are names)

- Reduce by launching three activities:
  – hunt for `f`
  – hunt for `a`
  – reduce `g(b)` recursively
  such that we can “execute” that expression.
- Note that depending on `f`, some reductions are not needed at all.
  (`f` might be the `if-then` function and `a` a boolean value).
- There are several strategies how to resolve:
  call-by-value, call-by-name, call-by-need, etc

3.c) Named-Data as a Special λ-Calculus Case



3.d) Caching Results by Fingerprinting λ-Expressions

NFN avoids recomputing results, first checks for cooked data:

- requires a “canonical name” for each possible result
- We do a hash on the full “functional name”
  h = hash(`"the expression whose result we are interested in"`)
- NFN has an internal instruction `FOX(expr)` called find-or-execute:

```
h = hash(expr);
result = lookup-by-name(h);
if result != nil then
    return result;
return reduce-by-name(expr);
```

4.a) Different Views on NFN

Information Centric Networking is more than NDN

- Customized “cooked data” dissemination, consumption:
  just express your logical requirement, network to find ways to satisfy the request, optimize with similar & partial requests
- ICN becomes a cloud:
  superset of CDN behavior, plus dynamic content
- Two levels of programmability:
  – λ-Calculus for orchestration
  – (named) binary code for bit-level computations

4.b) Implementation Status

- λ-Expression reduction engine up and running
  – call-by-name reduction strategy (Krivine's machine)
  – re-implemented the ZINC abstract machine (from Caml)
  – replaced all memory access to use CCNx substrate
- Intervowen Reduction and Routing strategies:
  – mapping NFN result names to CCNx' name scheme (see paper)
  – reduction+routing strategy being integrated to CCN-lite

λ-Expression-reduction was demoed at CCNxCon at PARC, Sep 2013

4.c) Conclusions

- Push ICN vision to the next level:
  – results, not raw data, matter
- “Generative media” on demand
- Customizable (by publisher): open set of access functions
- Customizable (by consumer): open set of rendering functions

Name the result – and consider it done!

Appendix 1 - Call-By-Name

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.

Appendix 2 - Extending CCNx' Routing Strategy