μNDN: an Orchestrated Microservice Architecture for Named Data Networking

Xavier MARCHAL, Thibault CHOLEZ, Olivier FESTOR

LORIA, UMR 7503 (University of Lorraine, CNRS, INRIA)
Vandoeuvre-les-Nancy, F-54506, France

September 22, 2018

Outline

Outline

Context

Network Function Virtualization (NFV):

- Common hardware, hosting various software components
- Reduce operational and capital expenditures
- Improve reliability and flexibility

Microservices architecture:

- Split a monolithic software into multiple and simple services
- Easier development and improvement of each service
- Better horizontal scalability
- Tend to use more resources individually
- Need a proper management plane
- Additional deployment complexity

Motivation

Expected benefits from NVF and microservices for ICN:

- Incremental deployment of NDN alongside existing protocols
- More efficient NDN topologies
- Better performance
- Deploy dynamically on-path functions

Challenges:

- Decomposition of a monolithic NDN router
- Linkage and packet processing
- Management of the different services

Outline

The microservices

Five are extracted from NDN router plus two others for security purpose

Can be split in two categories

Core routing functions:

- Name Router (NR): $\simeq$ FIB
- Backward Router (BR): $\simeq$ PIT
- Packet Dispatcher (PD)

Support functions (on-path services):

- Content Store (CS)
- Strategy Forwarder (SF)
- Signature Verifier (SV)
- Name Filter (NF)

The microservices

Name	Function	Oriented	Ingress/Egress cardinality
Name Router	Route *Interest* packets	Yes	1/N
Backward Router	Route back *Data* packets	Yes	N/1
Packet Dispatcher	Split *Interest*/*Data* traffic	No	N/N
Content Store	Cache *Data* packets	No	1/1
Strategy Forwarder	Forward *Interest* packets	No	1/1 or N
Signature Verifier	Verify packets' signature	No	1/1
Name Filter	Filter on packets' name	No	1/1

"Oriented" refers as if a module has specialized *Faces* to handle consumer and producer traffics

Effective cardinality:

- "1" means a modules should be connected to a single other module but can still broadcast traffic if more than one
- "N" means a modules can accept any number of other modules and is able to identify which send and/or to which forward the packets

Outline

The manager

Needed for efficient microservice architecture

Operations to implement for a proper network management:

- Deploy on demand or automatically the microservices
- Dynamically adapt the topology
- Update the microservices' running configuration
- Scale up the bottleneck services accordingly

Microservices must implement a management interface

- Get command from manager
- Send request to the manager
- Periodically report statistics

The manager

Basic metrics from microservices used to dynamically improve QoS

- Identify attacks like content poisoning attack
- Identify bottleneck and useless components

Name	**Values**
Name Router	Route statistics
Backward Router	Unsolicited *Data* packets Retransmitted *Interest* packets
Packet Dispatcher	User traffic statistics
Content Store	Hit/Miss count
Signature Verifier	Name of failed packets
Name Filter	Drop count

Manager can also get resource usages from the orchestrator

The manager

NLSR is not mandatory inside the managed network

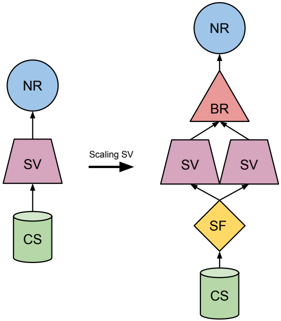
- The manager knows about all the topology
- Can trigger routine(s) and push new configurations like a SDN controller

External routing protocols can be implemented as microservice

- Placed at the edge of the managed network
- Offer protocol agnostic communication

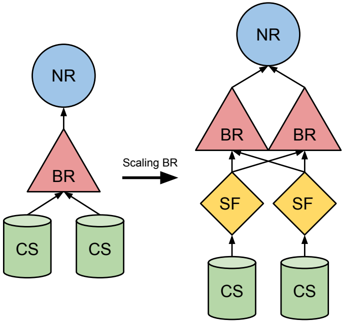
Scaling procedure

Support functions scaling



- Like a box with same properties
- BR may be replaced by a simpler function like another SF for stateless functions

Possible Backward Router scaling



- Adding an upper BR will only move the bottleneck (in most cases)
- Force the next hop to broadcast traffic

Outline

Environment

Plateform:

- 2 Intel Xeons 8 cores 2.4 GHz (E5 2630v3)
- Docker CE 18.03
- ndn-cxx v0.6.1

Microservices are written in C++ and are single-threaded[1]

NDN packets are carried over TCP/IP in the experiments

NDN *Data* packets always carry 8192 octets

Usage of a Docker bridge network when the microservices are in Containers

Producer(s) and consumer(s) are always executed from host

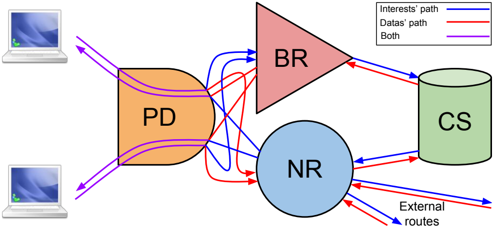
[1] Source code: https://github.com/Kanemochi/NDN-microservices

Performance

Module	Throughput (Mbps)	
	Bare-Metal	Container
Name Router	1,820	1,595
Backward Router	1,304	1,090
Packet Dispatcher	1,761	1,635
Content Store (freshness $= 0$)	1,760	1,538
Content Store (freshness > 0)	1,031	979
Content Store (from cache)	2,447	2,061
Strategy Forwarder	1,756	1,540
Signature Verifier (RSA2048)	515	401
Signature Verifier (ECDSA256)	122	101
Name Filter	1,804	1,593

- Signature verification is a heavy task, throughput can be "improved" with per registered prefix statistical verification
- CS can be slower than BR in some scenarios
- Around 13% throughput penalty from Docker virtualization

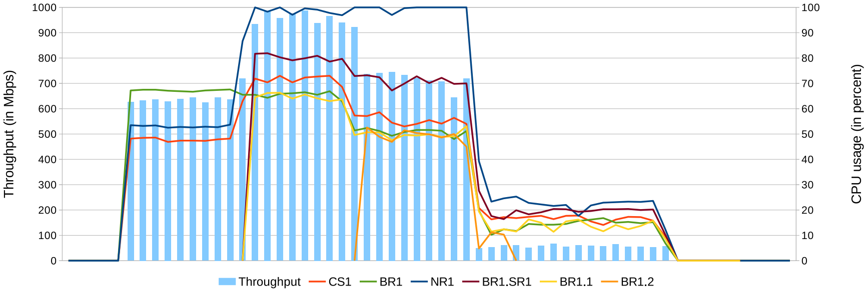
μNDN coupling "equivalent" to NFD



	Microservices				NFD
	PD	CS	BR	NR	
%CPU core usage	100	59	89	64	100
Throughput (in Mbps)	776				527
Latency (in ms)	2,63				3,88

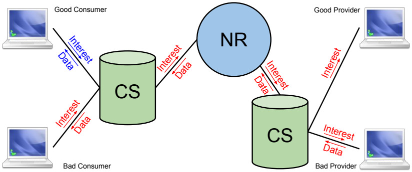
If Packet Dispatcher is not a bottleneck $\rightarrow$ 969 Mbps

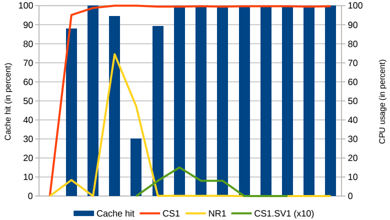
Scaling experiment



- BR is artificially limited to 67%
- Throughput increases from 625 up to 980 Mbps
- The scaling rule is not optimal
- Only get performance of one BR with no limit, huge load increase when broadcasting traffic to BR instances

Security experiment





- Content Poisoning Attack
- If cache hit decreases too much in a short period of time, the manager will insert a signature verifier between left CS and NR
- The manager can incrementally move SV toward the source(s) of bad *Data* packets

Outline

Conclusion

μNDN successfully achieved our goal to enhance NDN with NFV properties thanks to orchestrated microservices.

μNDN is implemented and running, it showed:

- To offer more possibilities when designing the network
- Its ability to dynamically instantiate and chain NDN functions for security and performance, based on predefined rules
- Better throughput than a monolithic forwarder

Main limitation: splitting FIB and PIT resulted in higher complexity (oriented functions, asymmetric 1/N vs N/1 cardinality)

Future works:

- Pursue the development (mainly the management plane)
- Explore further the possibilities offered by adding new functionalities as microservices