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

1. Introduction
2. Microservices
3. Manager
4. Experiments
5. Conclusion
# Outline

1. Introduction
2. Microservices
3. Manager
4. Experiments
5. Conclusion
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

1. Introduction
2. Microservices
3. Manager
4. Experiments
5. Conclusion
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): \( \sim \) FIB
- Backward Router (BR): \( \sim \) PIT
- Packet Dispatcher (PD)

Support functions (on-path services):
- Content Store (CS)
- Strategy Forwarder (SF)
- Signature Verifier (SV)
- Name Filter (NF)
The microservices

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Oriented</th>
<th>Ingress/Egress cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name Router</td>
<td>Route <em>Interest</em> packets</td>
<td>Yes</td>
<td>1/N</td>
</tr>
<tr>
<td>Backward Router</td>
<td>Route back <em>Data</em> packets</td>
<td>Yes</td>
<td>N/1</td>
</tr>
<tr>
<td>Packet Dispatcher</td>
<td>Split <em>Interest/Data</em> traffic</td>
<td>No</td>
<td>N/N</td>
</tr>
<tr>
<td>Content Store</td>
<td>Cache <em>Data</em> packets</td>
<td>No</td>
<td>1/1</td>
</tr>
<tr>
<td>Strategy Forwarder</td>
<td>Forward <em>Interest</em> packets</td>
<td>No</td>
<td>1/1 or N</td>
</tr>
<tr>
<td>Signature Verifier</td>
<td>Verify packets’ signature</td>
<td>No</td>
<td>1/1</td>
</tr>
<tr>
<td>Name Filter</td>
<td>Filter on packets’ name</td>
<td>No</td>
<td>1/1</td>
</tr>
</tbody>
</table>

"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

1. Introduction
2. Microservices
3. Manager
4. Experiments
5. Conclusion
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name Router</td>
<td>Route statistics</td>
</tr>
<tr>
<td>Backward Router</td>
<td>Unsolicited <em>Data</em> packets</td>
</tr>
<tr>
<td></td>
<td>Retransmitted <em>Interest</em> packets</td>
</tr>
<tr>
<td>Packet Dispatcher</td>
<td>User traffic statistics</td>
</tr>
<tr>
<td>Content Store</td>
<td>Hit/Miss count</td>
</tr>
<tr>
<td>Signature Verifier</td>
<td>Name of failed packets</td>
</tr>
<tr>
<td>Name Filter</td>
<td>Drop count</td>
</tr>
</tbody>
</table>

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

1. Introduction
2. Microservices
3. Manager
4. Experiments
5. Conclusion
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

<table>
<thead>
<tr>
<th>Module</th>
<th>Throughput (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bare-Metal</td>
</tr>
<tr>
<td>Name Router</td>
<td>1,820</td>
</tr>
<tr>
<td>Backward Router</td>
<td>1,304</td>
</tr>
<tr>
<td>Packet Dispatcher</td>
<td>1,761</td>
</tr>
<tr>
<td>Content Store (freshness = 0)</td>
<td>1,760</td>
</tr>
<tr>
<td>Content Store (freshness &gt; 0)</td>
<td>1,031</td>
</tr>
<tr>
<td>Content Store (from cache)</td>
<td>2,447</td>
</tr>
<tr>
<td>Strategy Forwarder</td>
<td>1,756</td>
</tr>
<tr>
<td>Signature Verifier (RSA2048)</td>
<td>515</td>
</tr>
<tr>
<td>Signature Verifier (ECDSA256)</td>
<td>122</td>
</tr>
<tr>
<td>Name Filter</td>
<td>1,804</td>
</tr>
</tbody>
</table>

- 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**

<table>
<thead>
<tr>
<th>Microservices</th>
<th>NFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD</td>
<td>CS</td>
</tr>
<tr>
<td>%CPU core usage</td>
<td>100</td>
</tr>
<tr>
<td>Throughput (in Mbps)</td>
<td>776</td>
</tr>
<tr>
<td>Latency (in ms)</td>
<td>2.63</td>
</tr>
</tbody>
</table>

If Packet Dispatcher is not a bottleneck → 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

1. Introduction
2. Microservices
3. Manager
4. Experiments
5. Conclusion
Conclusion

\(\mu\text{NDN}\) successfully achieved our goal to enhance NDN with NFV properties thanks to orchestrated microservices. \(\mu\text{NDN}\) 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