μNF: A Disaggregated Packet Processing Architecture

Shihabur Chowdhury, Anthony, Haibo Bian, Tim Bai, and Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo

Transition from middleboxes to VNFs

NFs are run as *Virtual Network Functions (VNFs)* on pool of (virtual) resources

Commodity computing, storage & switching equipment
Transition from middleboxes to VNFs

Purpose-built hardware middlebox

Commodity computing, storage & switching equipment

 NFs are run as Virtual Network Functions (VNFs) on pool of (virtual) resources

Current practice
One-to-one substitution of middleboxes with monolithic VNFs
Monolithic VNF Limitations

Functional decomposition of commonly found NFs in Data Centers

Monolithic VNF Limitations

Redundant development of common tasks

Coarse-grained resource allocation & scaling

Functional decomposition of commonly found NFs in Data Centers\(^1\)

Monolithic VNF Limitations

1. Redundant development of common tasks
2. Coarse-grained resource allocation & scaling
3. Wasted CPU cycles when VNFs are chained

Service Function Chain

Functional decomposition of commonly found NFs in Data Centers

---

Monolithic VNFs: Impact on CPU usage

Edge Fw. → Monitoring → App. Fw

(C1) Click-based monolithic VNFs chained with veth pairs

Traffic
HTTP trace derived from a web-service (~15k hits/mo)

(C2) Optimized Click pipeline
Monolithic VNFs: Impact on CPU usage

<table>
<thead>
<tr>
<th>Click Element</th>
<th>CPU Cycles/packet saved in C2</th>
<th>Element weight in C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>FromDevice</td>
<td>71.9%</td>
<td>0.22%</td>
</tr>
<tr>
<td>ToDevice</td>
<td>67.1%</td>
<td>0.25%</td>
</tr>
<tr>
<td>CheckIPHeader</td>
<td>65.1%</td>
<td>0.44%</td>
</tr>
<tr>
<td>HTTPClassifier</td>
<td>48.28%</td>
<td>47.8%</td>
</tr>
<tr>
<td>Overall</td>
<td><strong>29.5%</strong></td>
<td>-</td>
</tr>
</tbody>
</table>
How can we engineer VNFs to better consolidate functions on the same hardware, enabling finer-grained resource allocation while maintaining the same level of performance as the state-of-the-art approaches?
How can we engineer VNFs to better consolidate functions on the same hardware, enabling finer-grained resource allocation while maintaining the same level of performance as the state-of-the-art approaches?

*Microservices approach:* Decompose VNFs into independently deployable and loosely-coupled packet processing entities.
Micro Network Functions (µNFs)

µNF Processing Graph:
Pipelined execution of µNFs

µNFs are:
reusable, loosely-coupled, independently deployable
Micro Network Functions (µNFs)

VNF templates (µNF Processing Graph): Pipelined execution of µNFs

µNFs are: reusable, loosely-coupled, independently deployable

Disaggregate

SFC

Repeated µNFs removed
Similar µNFs consolidated
µNF processing graphs merged

Optimized µNF Processing Graph
System Overview

![Diagram showing System Overview]

- **μNF Orchestrator**
  - Southbound API (e.g., DeployChain)

- **Mgmt NIC**
  - NIC(s)

- **Orchestration Agent**
  - Rx Service
  - Tx Service
  - μNF-0
  - μNF-1
  - ...
  - μNF-n

- **SFC + VNF templates + μNF configuration generators + μNF descriptors**
µNF Components

- **µNF Orchestrator**
  - Southbound API (e.g., DeployChain)

- **Mgmt NIC**
  - NIC(s)
  - Rx Service
  - Tx Service

- **Orchestration Agent**
  - µNF-0
  - µNF-1
  - …
  - µNF-n

- **SFC + VNF templates + µNF configuration generators + µNF descriptors**

- **Ctrl/Mgmt. API**
  - iport-0
  - iport-1
  - …
  - iport-k
  - PacketProcessor
  - iport to eport mapping table

- **Egress Ports**
  - eport-0
  - eport-1
  - …
  - eport-m

- **Ingress Ports**
  - …
Implementation

**Primary DPDK process.** Responsible for bootstrapping (initialize NIC, pre-allocate objects in memory, *etc.*)

Implemented using DPDK Poll Mode Driver to bypass kernel. Implements packet classifier to distribute packets to µNFs.

**Secondary DPDK processes.** Obtains pre-allocated memory objects from the agent; works in polling mode.

Point-to-Point Ingress/Egress Ports

Main Memory

\[ \text{pkt}_a \quad \text{pkt}_b \quad \text{pkt}_c \]

\[ \mu\text{NF}_A \quad \mu\text{NF}_B \]

Shared Ring

\[ \bullet \text{PPPort (Egress)} \quad \diamond \text{PPPort (Ingress)} \]
Experiment Setup

- Two machines connected back-to-back without a switch
- 2x6 core 2.1 GHz Intel Xeon E5 CPUs, 32GB RAM, Intel 10G NIC
- Hyper-threading disabled; All but cpu-0 isolated from kernel scheduler; μNFs pinned to CPU cores
- Traffic generators: *pktgen-dpdk* (throughput) and *Moongen* (latency)
Microbenchmark: Throughput

Throughput (Mpps)  |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Core-1</td>
<td>Core-2</td>
</tr>
<tr>
<td>14.87</td>
<td>10.77</td>
</tr>
<tr>
<td>3.99</td>
<td>8.45</td>
</tr>
<tr>
<td>5.43</td>
<td>2.35</td>
</tr>
<tr>
<td>1.59</td>
<td>1.2</td>
</tr>
<tr>
<td>0.82</td>
<td>-</td>
</tr>
</tbody>
</table>

Packet size (Bytes)  |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
</tr>
</tbody>
</table>

Throughput (Gbps)  |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
</tr>
</tbody>
</table>
Microbenchmark: Latency

 Longer chain ➔ Higher Latency

Can we improve latency?
Optimization: Parallel execution of µNFs

Parallelize sequential blocks of µNFs if:

1. The µNFs do not modify the same headers
2. The µNFs do not modify the packet stream
Optimization: Parallel execution of µNFs

Parallelize sequential blocks of µNFs if:

1. The µNFs do not modify the same headers
2. The µNFs do not modify the packet stream
Optimization: Parallel execution of µNFs

Parallelize sequential blocks of µNFs if:

1. The µNFs do not modify the same headers
2. The µNFs do not modify the packet stream

BranchEgressPort

Embeds an atomic counter in packets
Optimization: Parallel execution of µNFs

Parallelize sequential blocks of µNFs if:

1. The µNFs do not modify the same headers
2. The µNFs do not modify the packet stream

BranchEgressPort
Embeds an atomic counter in packets

MarkerEgressPort
Increases atomic counter in packets
Optimization: Parallel execution of µNFs

Parallelize sequential blocks of µNFs if:

1. The µNFs do not modify the same headers
2. The µNFs do not modify the packet stream

µNF-0 → µNF-1 → µNF-2 → µNF-3

BranchEgressPort
MarkerEgressPort
SyncIngressPort

Embeds an atomic counter in packets
Increases atomic counter in packets
Releases packets after all the µNFs have incremented the atomic counter

24
Optimization: Parallel execution of µNFs

Parallelize sequential blocks of µNFs if:

1. The µNFs do not modify the same headers
2. The µNFs do not modify the packet stream
Optimization: Parallel execution of μNFs

Parallelize sequential blocks of μNFs if:

1. The μNFs do not modify the same headers
2. The μNFs do not modify the packet stream
Impact of NUMA configuration
Impact of NUMA configuration

~3x drop in throughput
Optimization: Pipelined Cache-prefetching
Optimization: Pipelined Cache-prefetching

Before processing starts:
Prefetch a cacheline from first $k$ packets
Optimization: Pipelined Cache-prefetching

Before processing starts: 1
Prefetch a cacheline from first $k$ packets

While processing packet $i$: 2
Prefetch a cacheline from packet $(i + k)$
Optimization: Pipelined Cache-prefetching

Before processing starts:
Prefetch a cacheline from first $k$ packets

While processing packet $i$:
Prefetch a cacheline from packet $(i + k)$

Prefetching $\sim 20\%$ packets in a batch improves throughput by $\sim 3x$
Performance of µNF-based SFC

Edge Fw. → Monitoring → App. Fw

- RxService
- CheckIPHeader
- L3L4Filter
  - Deny
  - Allow
- HTTPClassifier
- CountUrl
- ValidateUrl
  - Safe
  - Other
- RxService

<table>
<thead>
<tr>
<th>Click Element</th>
<th>Saved cycles/packet</th>
<th>Element weight in CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckIPHeader</td>
<td>27.8%</td>
<td>0.44%</td>
</tr>
<tr>
<td>HTTPClassifier</td>
<td>28.9%</td>
<td>47.8%</td>
</tr>
<tr>
<td>Overall</td>
<td>16.8%</td>
<td>-</td>
</tr>
</tbody>
</table>
What’s Next?

- Disaggregated & pipelined-packet processing for 25/40/100G line rate
- End-to-end aspects of the system: e.g., optimized μNF processing pipeline deployment with specific SLOs
Questions?