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

IEEE NetSoft 2019, Paris, France, June 26, 2019

Transition from middleboxes to VNFs

Purpose-built hardware middlebox

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

Commodity computing, storage & switching equipment

Transition from middleboxes to VNFs

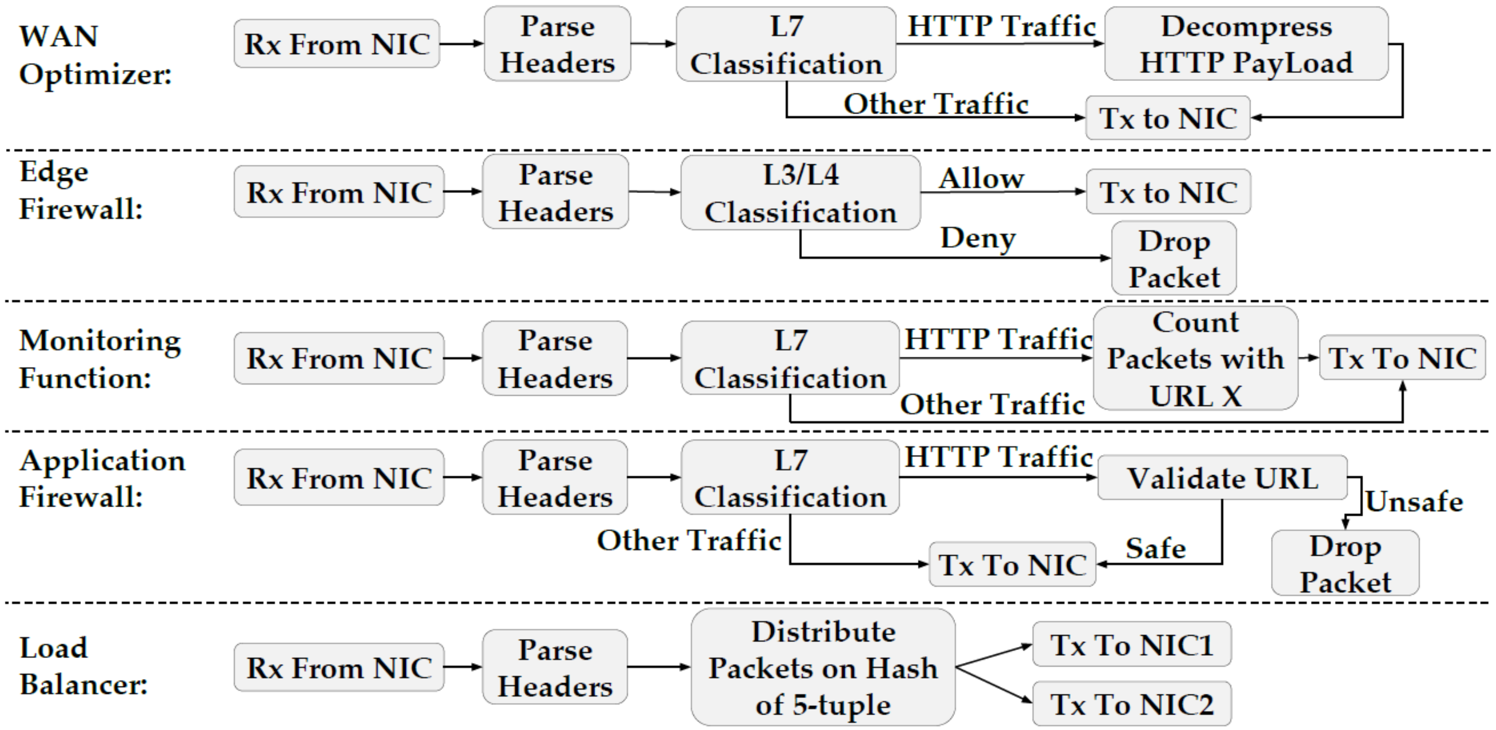
Purpose-built hardware middlebox

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

Commodity computing, storage & switching equipment

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

Monolithic VNF Limitations



Functional decomposition of commonly found NFs in Data Centers[1]

[1]S.R. Chowdhury, *et al.* Re-architecting NFV Ecosystem with Microservices: State-of-the-art and Research Challenges. IEEE Network, 33(3): 168-176, May 2019

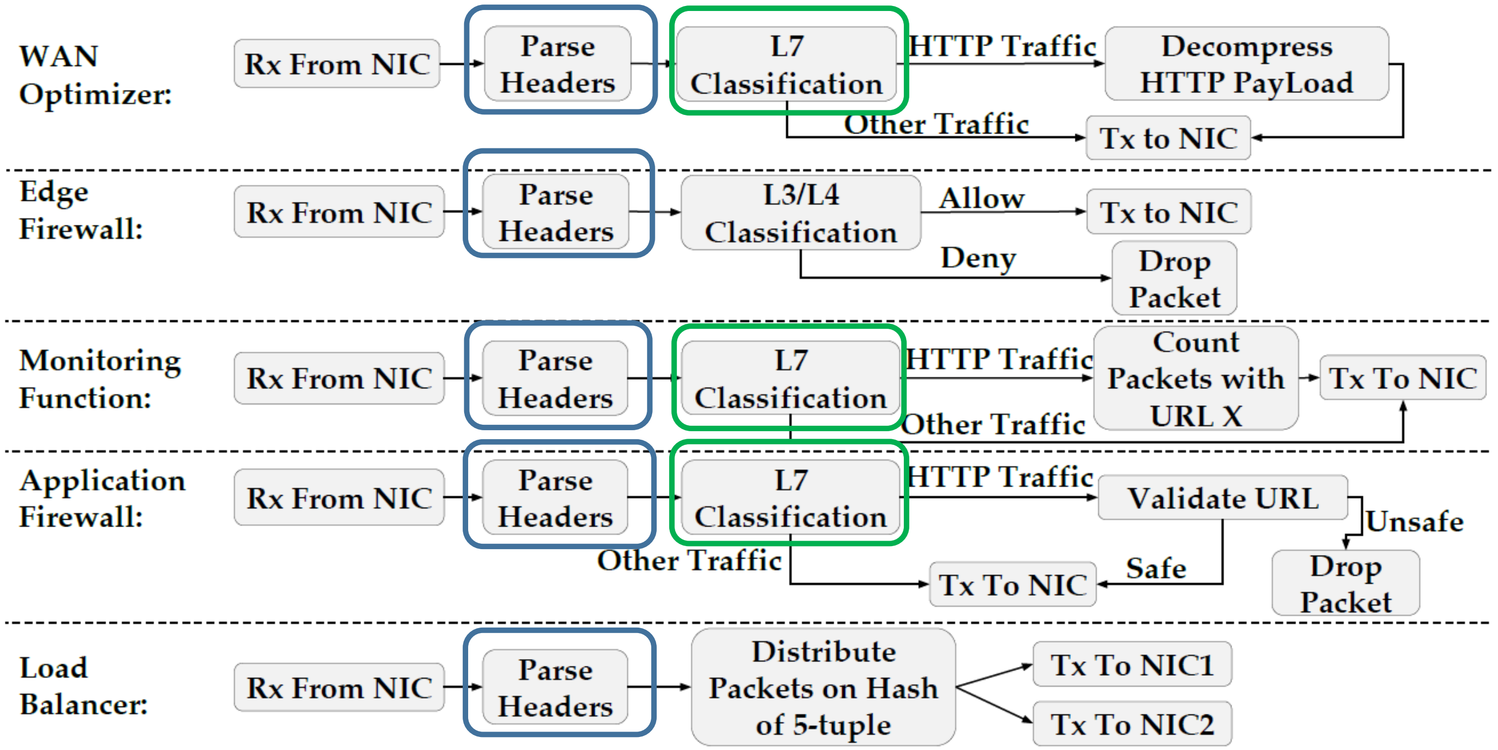
Monolithic VNF Limitations

Redundant development of common tasks

1

Coarse-grained resource allocation & scaling

2



Functional decomposition of commonly found NFs in Data Centers[1]

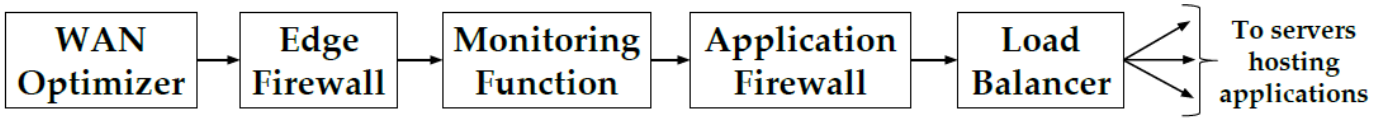
[1] S.R. Chowdhury, *et al.* Re-architecting NFV Ecosystem with Microservices: State-of-the-art and Research Challenges. IEEE Network, 33(3): 168-176, May 2019

Monolithic VNF Limitations

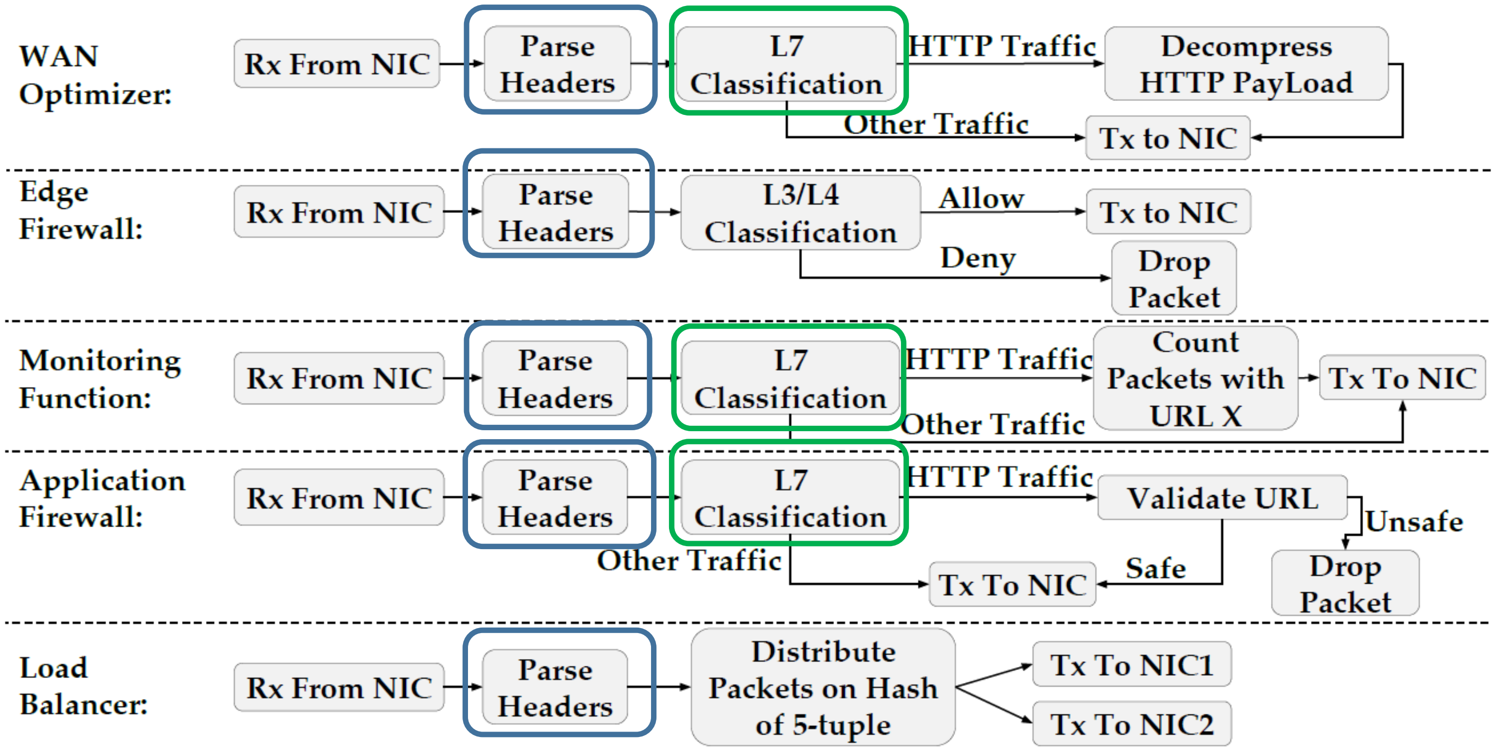
Redundant development of common tasks (1)

Coarse-grained resource allocation & scaling (2)

Wasted CPU cycles when VNFs are chained (3)



Service Function Chain



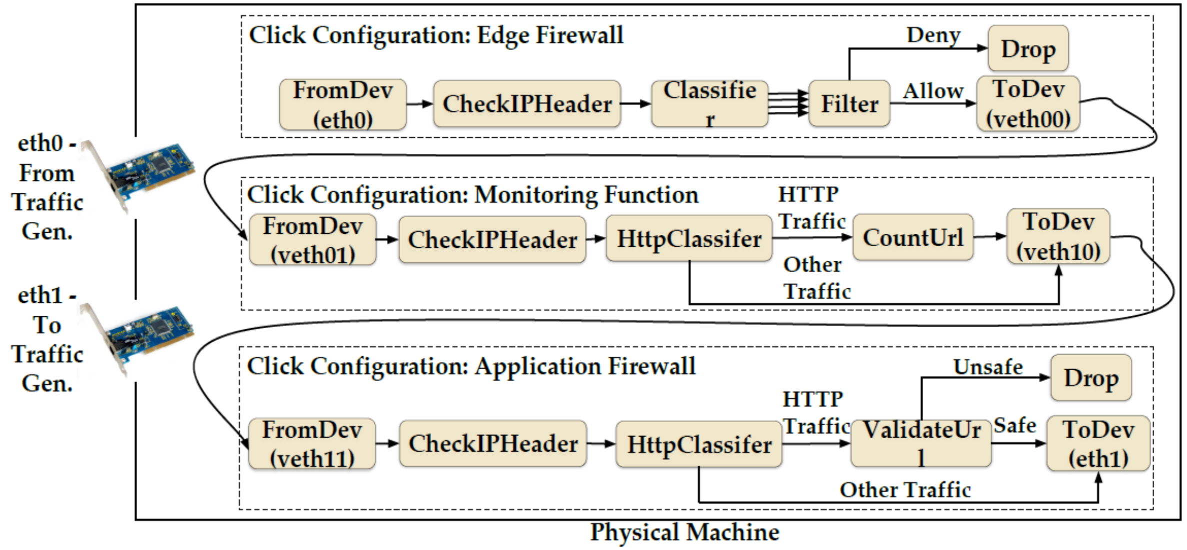
Functional decomposition of commonly found NFs in Data Centers[1]

[1] S.R. Chowdhury, *et al.* Re-architecting NFV Ecosystem with Microservices: State-of-the-art and Research Challenges. IEEE Network, 33(3): 168-176, May 2019

Monolithic VNFs: Impact on CPU usage

Edge Fw. →Monitoring → App. Fw

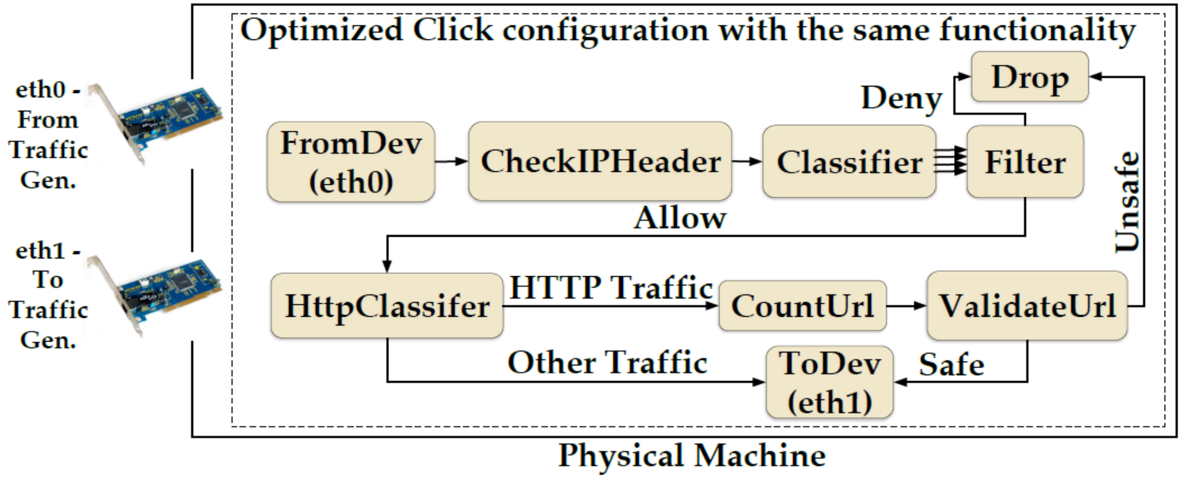
(C1) Click-based monolithic VNFs chained with veth pairs



Traffic

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

(C2) Optimized Click pipeline



Monolithic VNFs: Impact on CPU usage

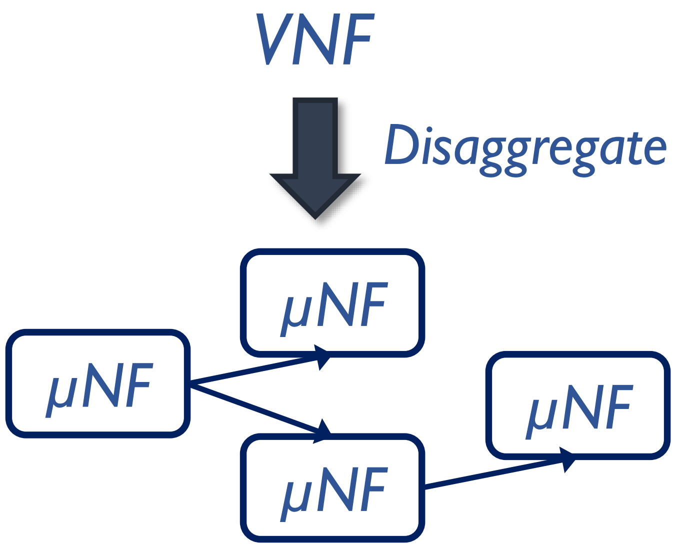
Click Element	CPU Cycles/packet saved in C2	Element weight in C1
FromDevice	71.9%	0.22%
ToDevice	67.1%	0.25%
CheckIPHeader	65.1%	0.44%
HTTPClassifier	48.28%	47.8%
Overall	**29.5%**	-

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.

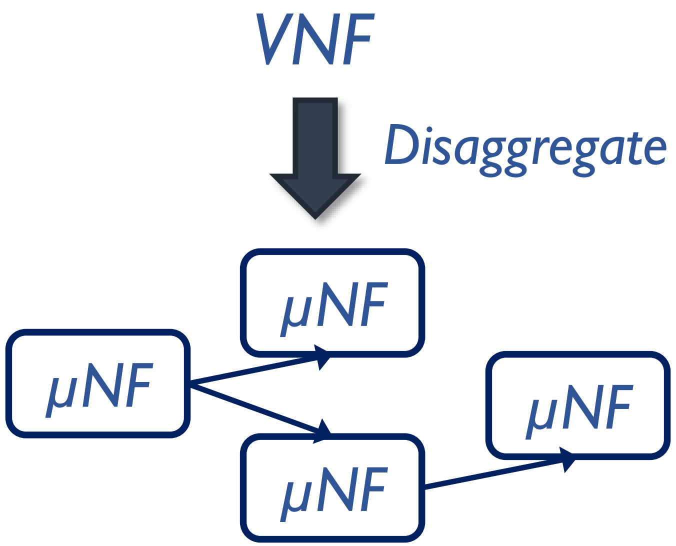
Micro Network Functions (µNFs)



µNF Processing Graph:
Pipelined execution of µNFs

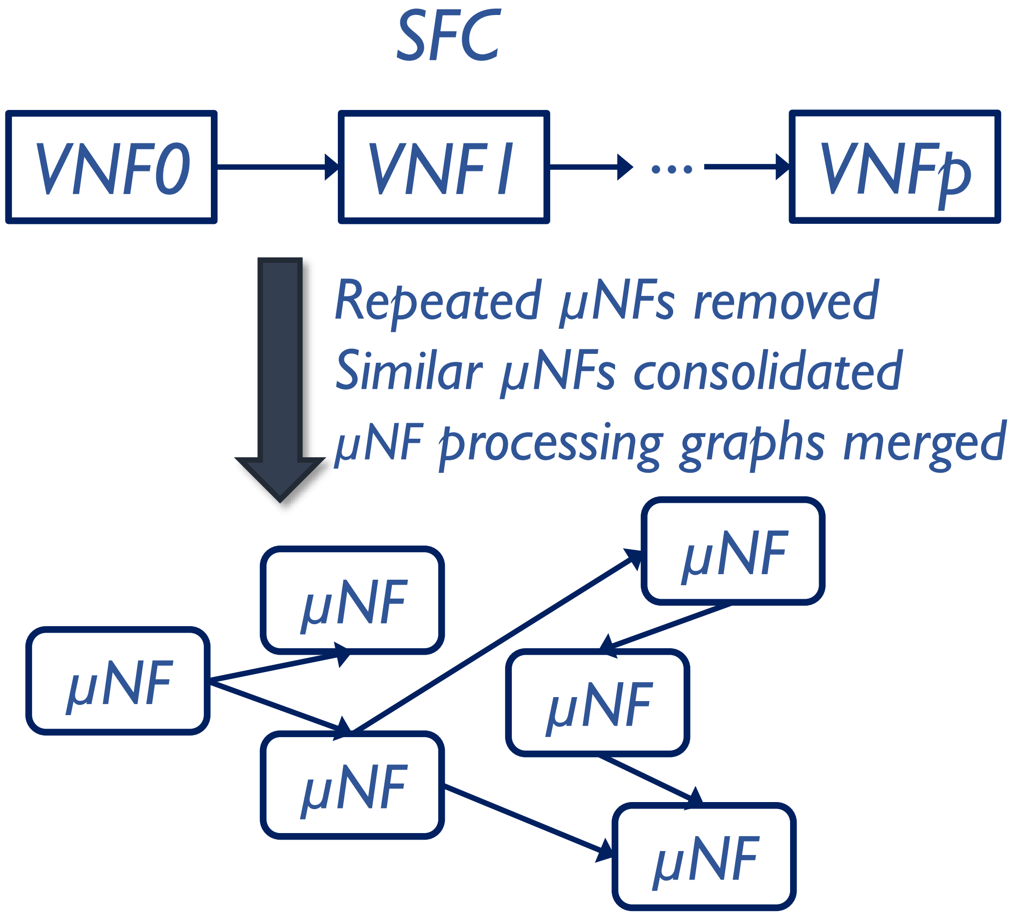
µNFs are:
reusable, loosely-coupled,
independently deployable

Micro Network Functions (μNFs)



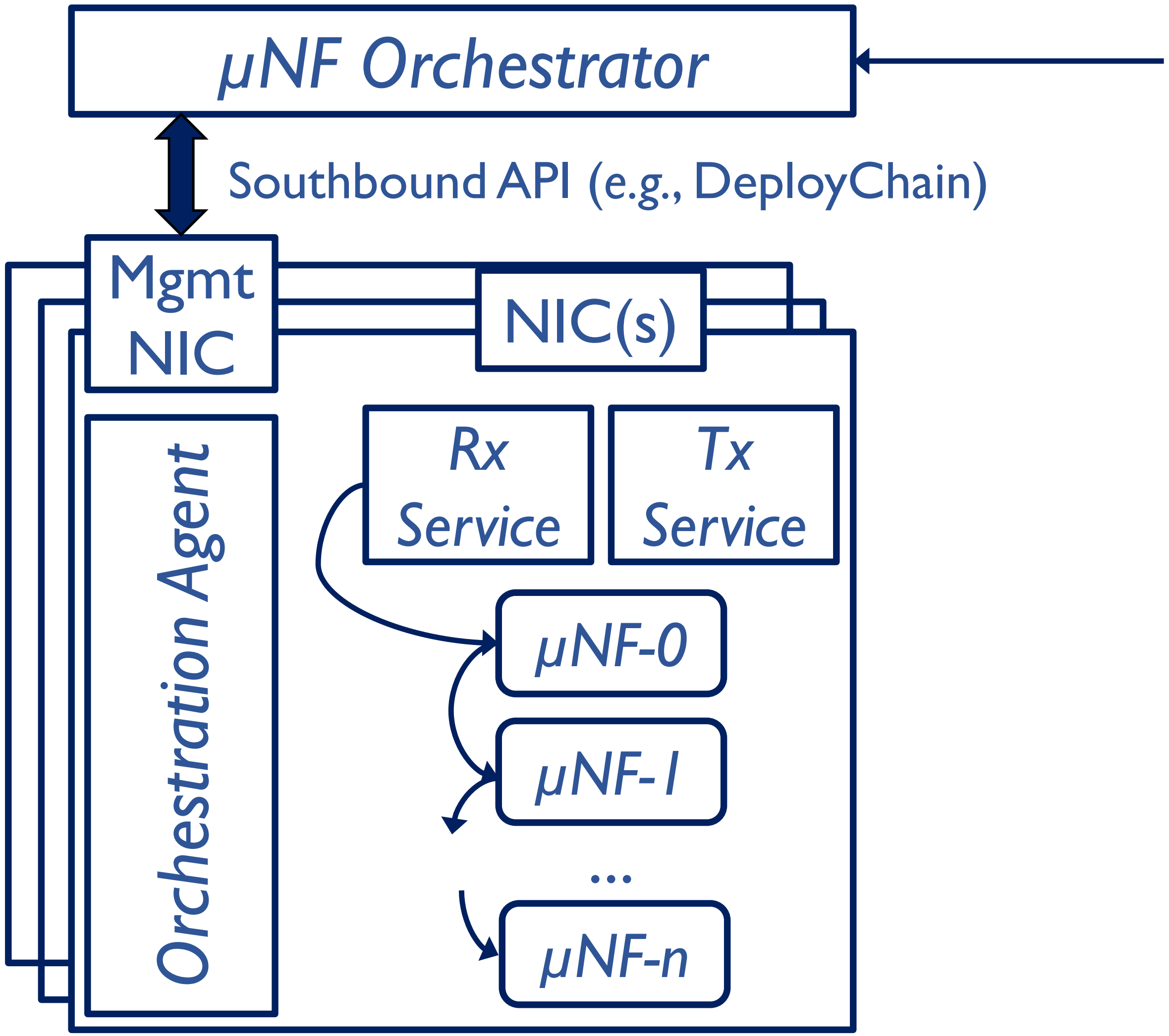
VNF templates (μNF Processing Graph):
Pipelined execution of μNFs

μNFs are:
reusable, loosely-coupled,
independently deployable



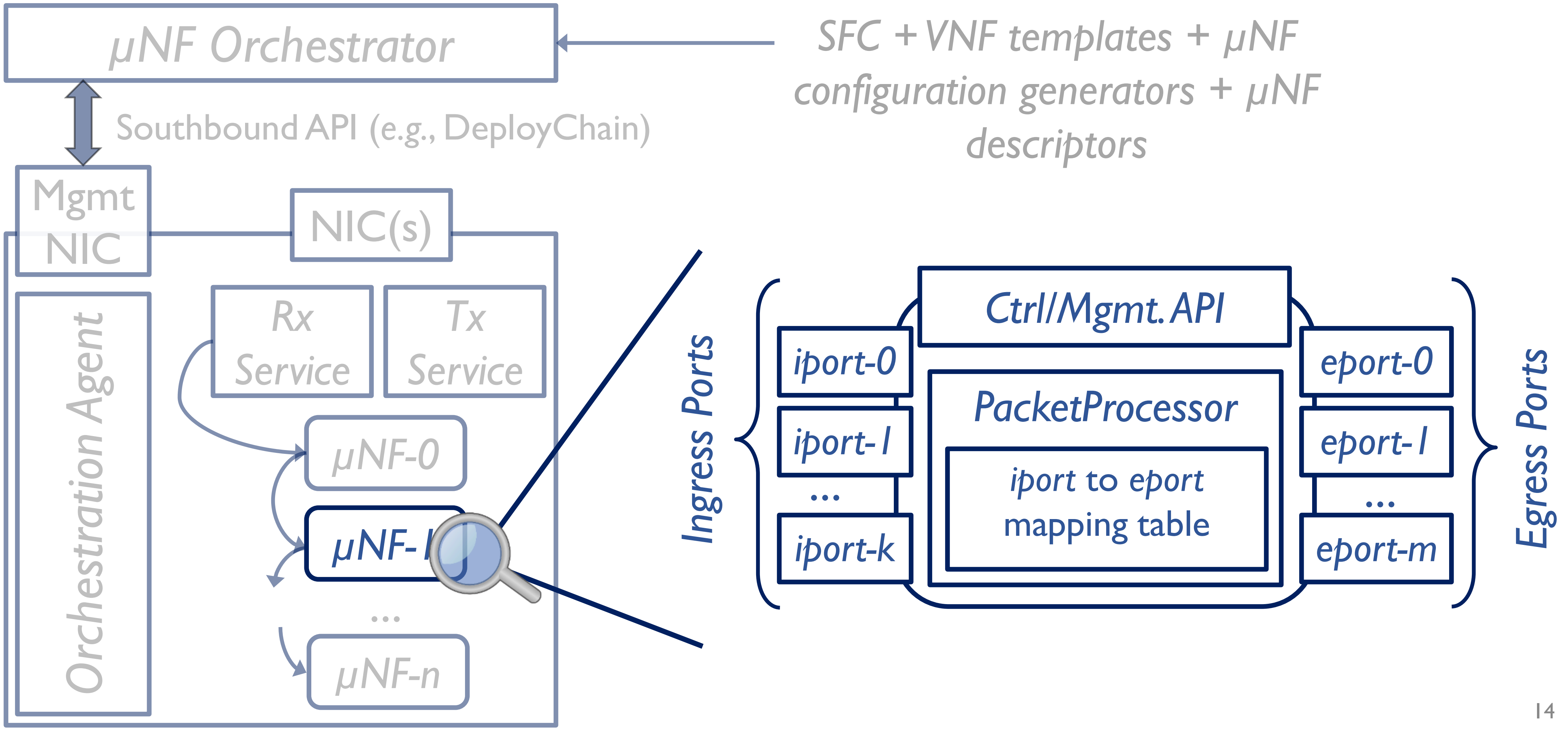
Optimized μNF Processing Graph

System Overview



SFC + VNF templates + µNF configuration generators + µNF descriptors

µNF Components



Implementation

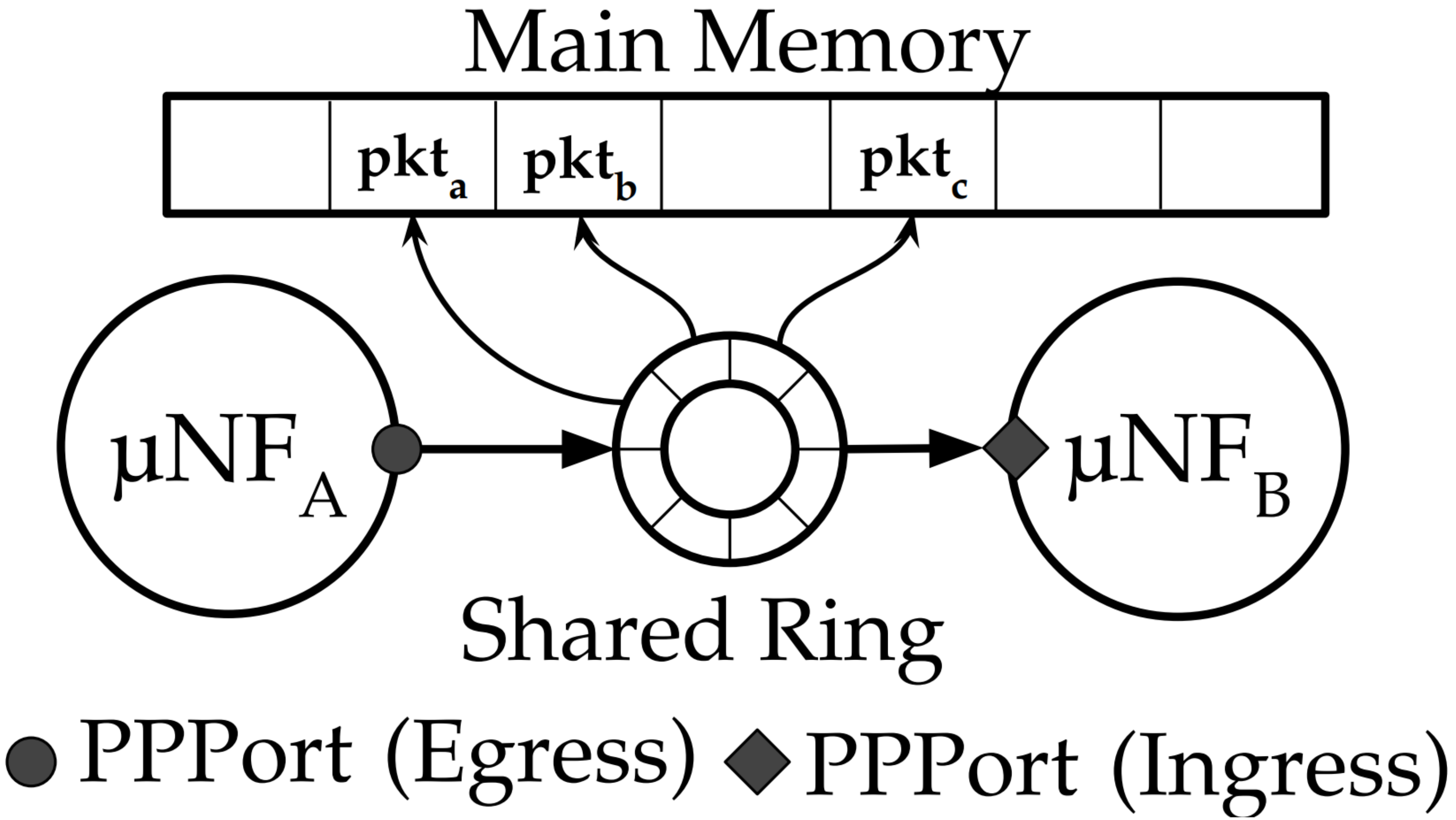
Agent — **Primary DPDK process**. Responsible for bootstrapping (initialize NIC, pre-allocate objects in memory, *etc.*)

Rx/Tx — Implemented using **DPDK Poll Mode Driver** to bypass kernel. Implements packet classifier to distribute packets to µNFs

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

Port — Implemented using **lockless multi-producer multi-consumer circular queue**. Holds packet references for **zero-copy packet exchange**.

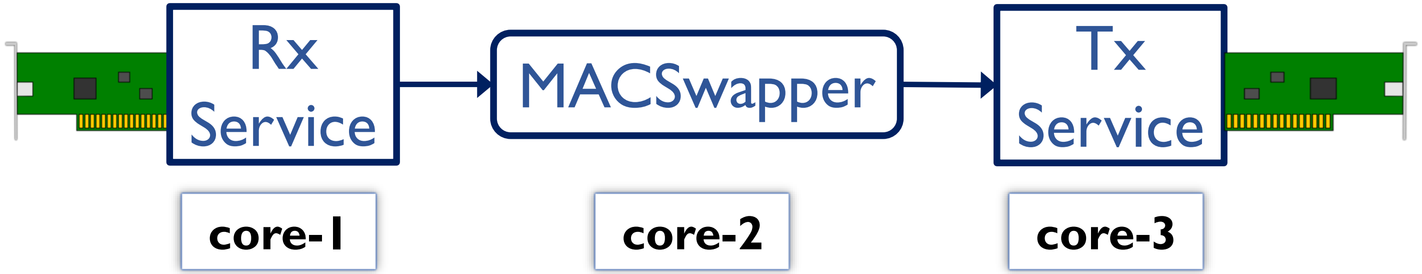
Point-to-Point Ingress/Egress Ports

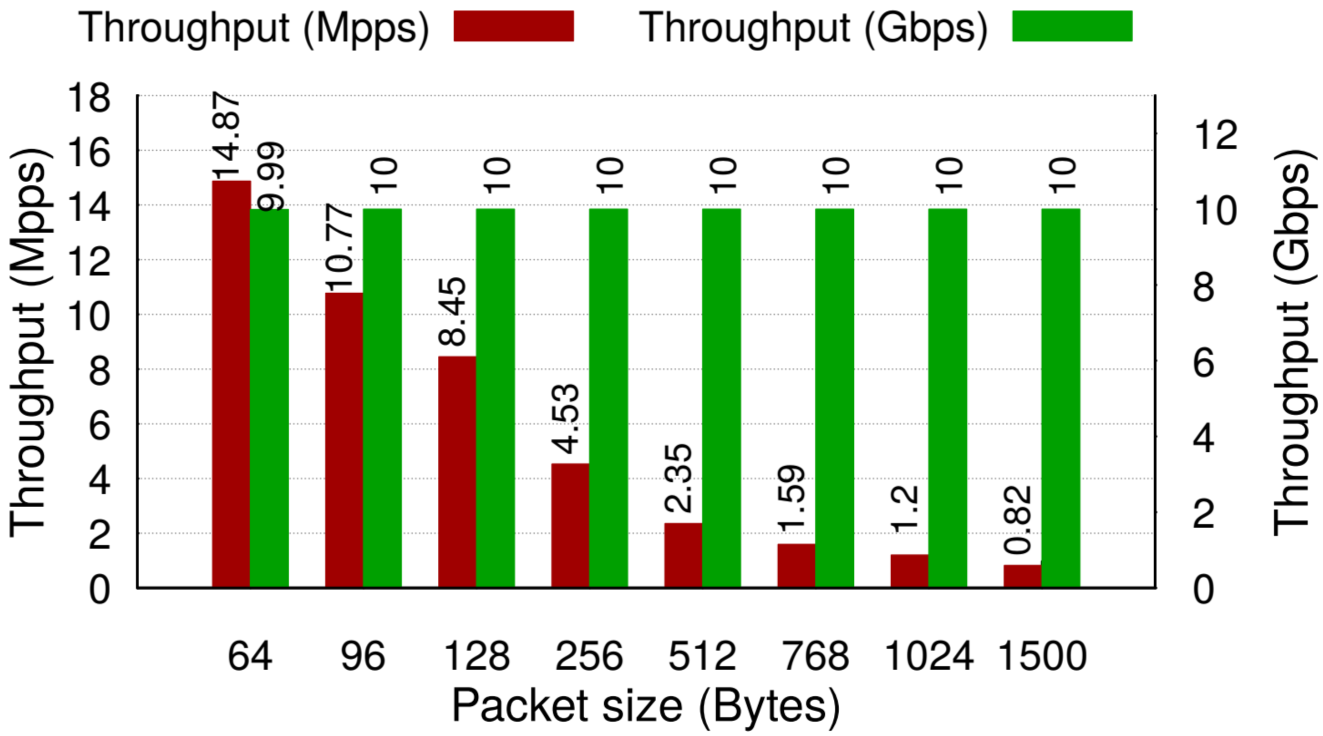


Experiment Setup

- 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)

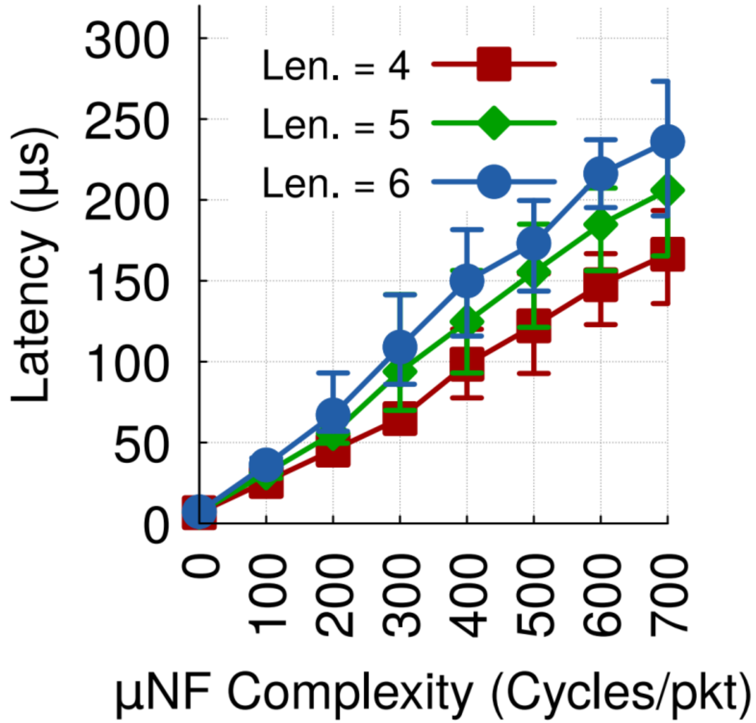
Microbenchmark: Throughput





Microbenchmark: Latency

Longer chain ➔ Higher Latency



Can we improve latency?

Optimization: Parallel execution of μNFs

Parallelize sequential blocks of μNFs if:

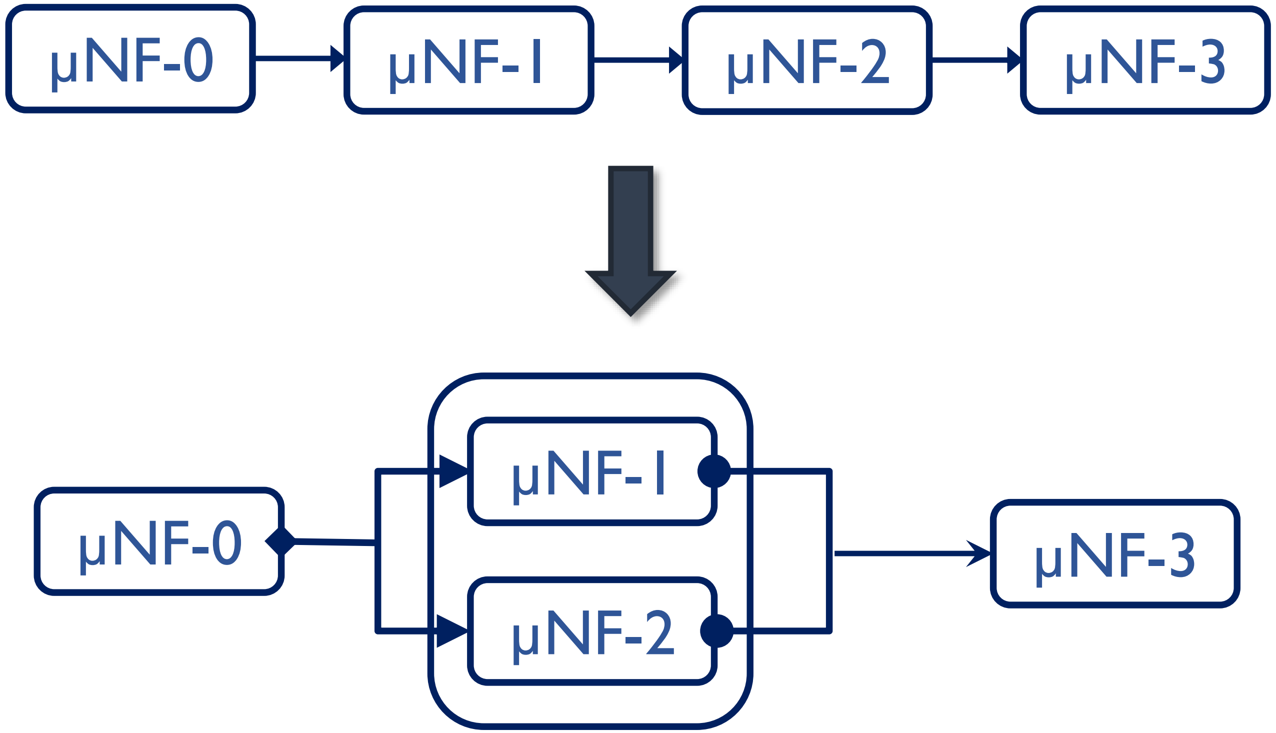
1. The μNFs do not modify the same headers

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

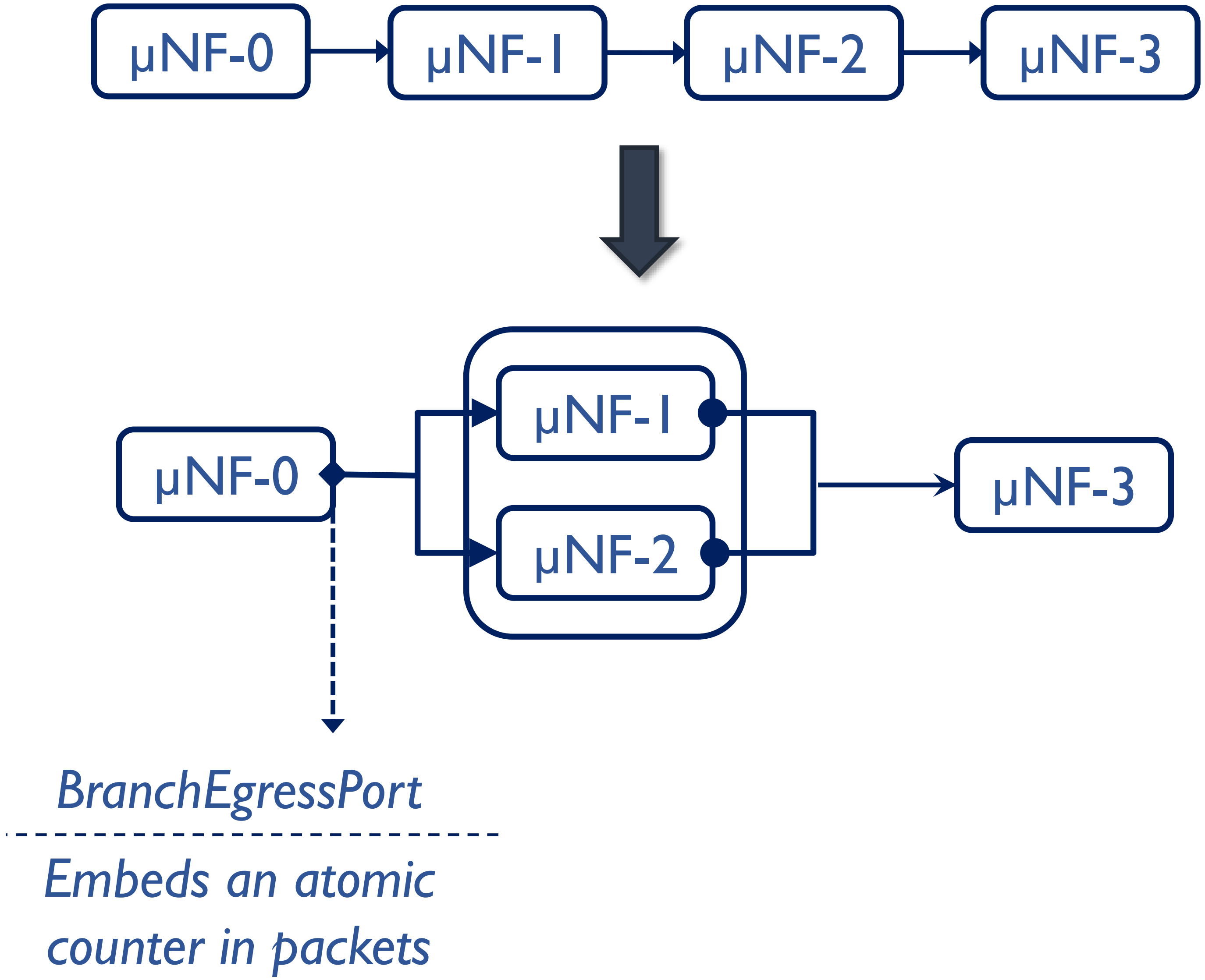


Optimization: Parallel execution of μNFs

Parallelize sequential blocks of μNFs if:

1. The μNFs do not modify the same headers

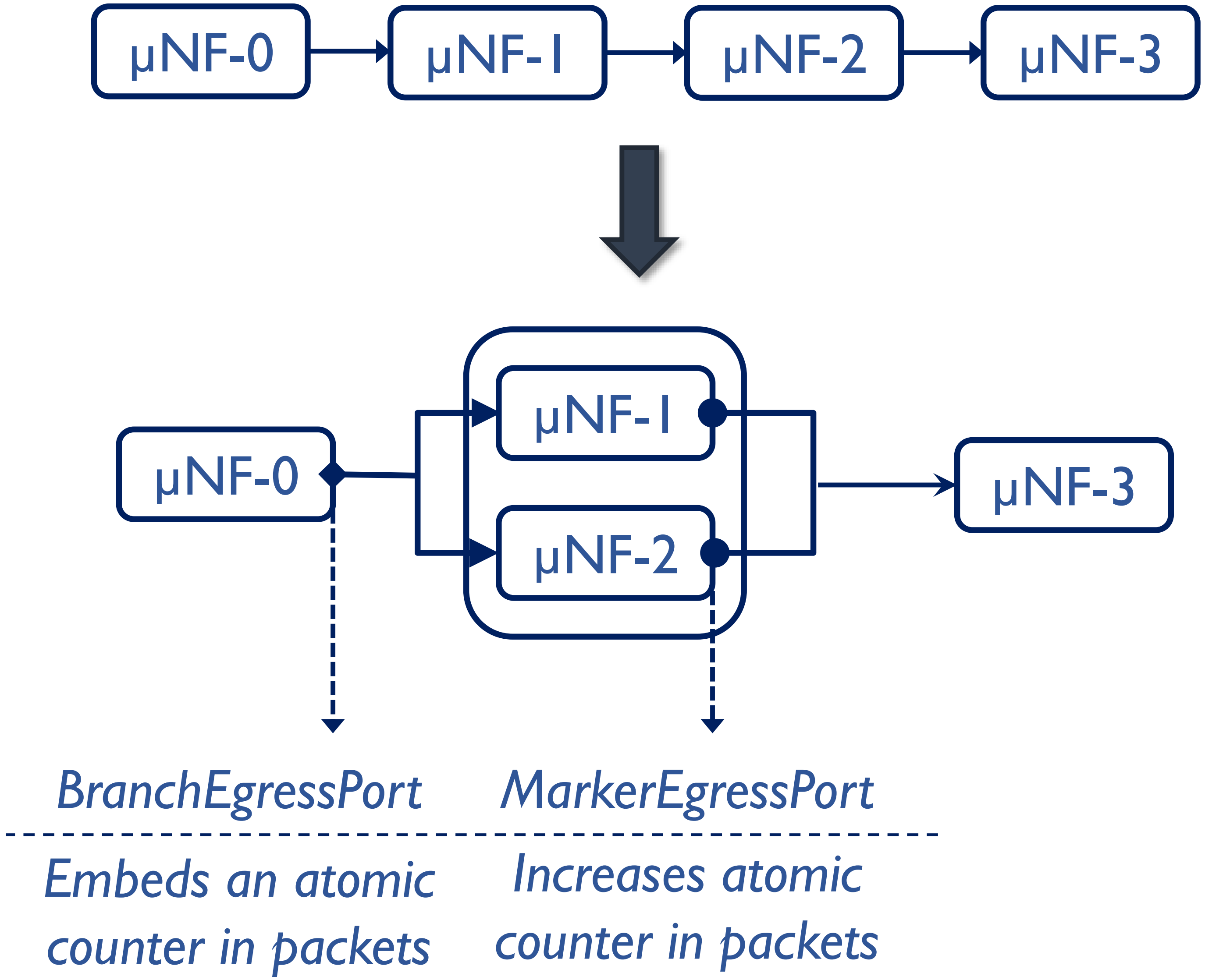
2. The μNFs do not modify the packet stream



Optimization: Parallel execution of μNFs

Parallelize sequential blocks of μNFs if:

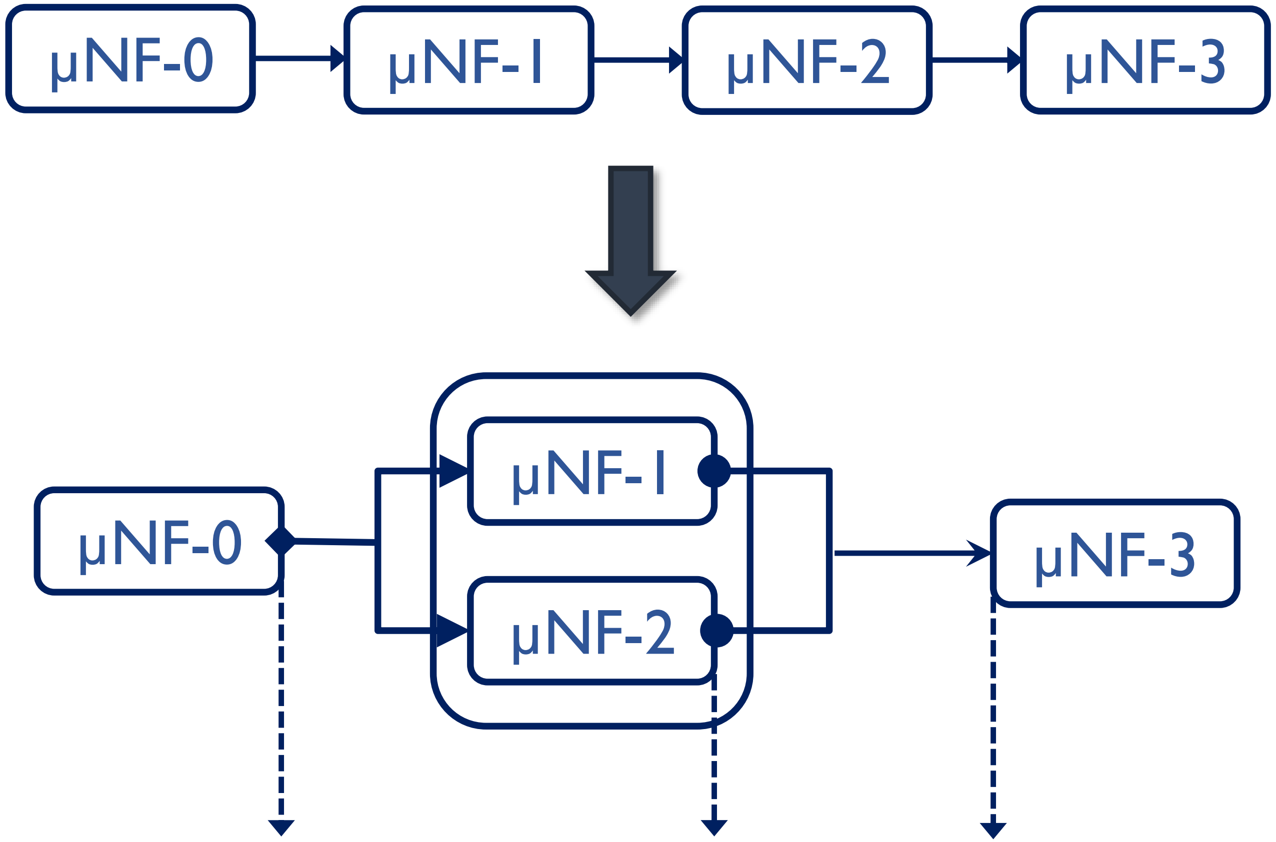
1. The μNFs do not modify the same headers
2. The μNFs do not modify the packet stream



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

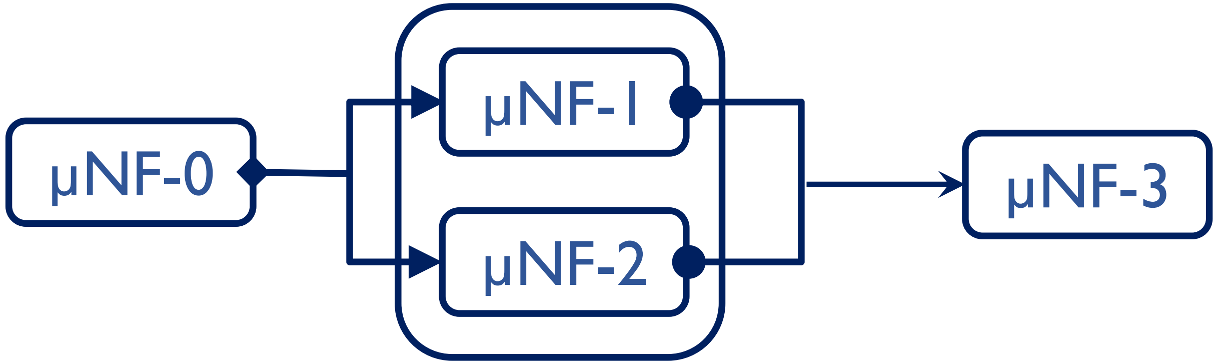


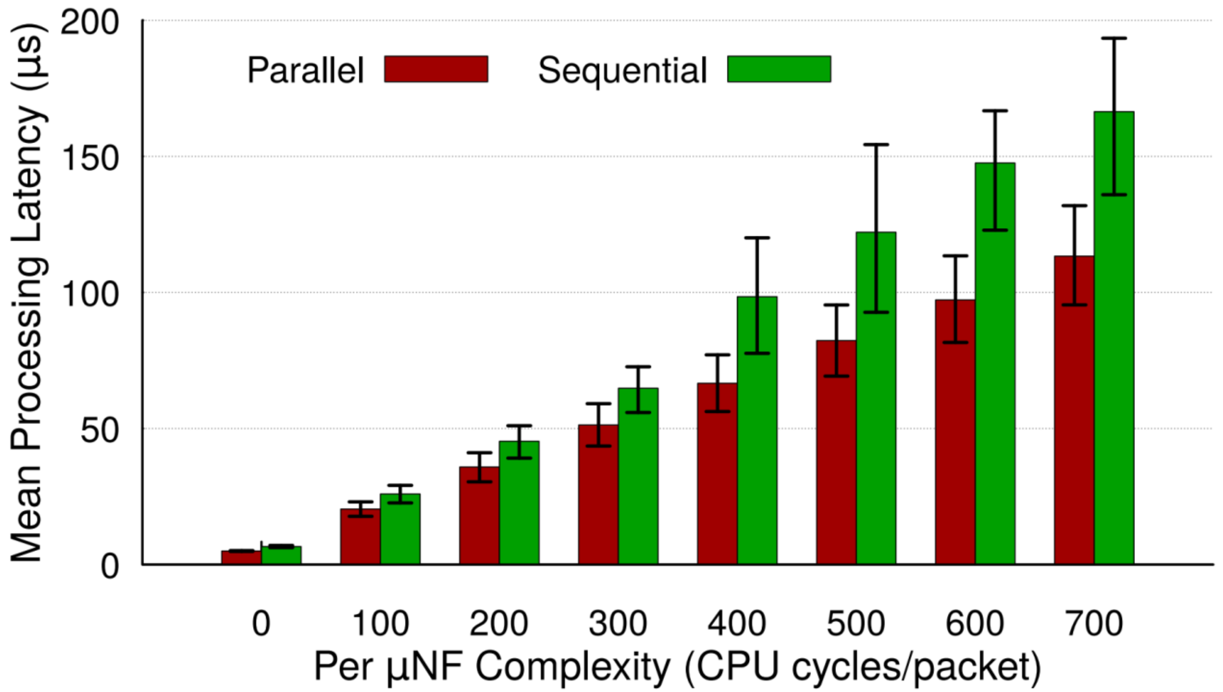
BranchEgressPort	*MarkerEgressPort*	*SyncIngressPort*
Embeds an atomic counter in packets	*Increases atomic counter in packets*	*Releases packets after all the µNFs have incremented the atomic counter*

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

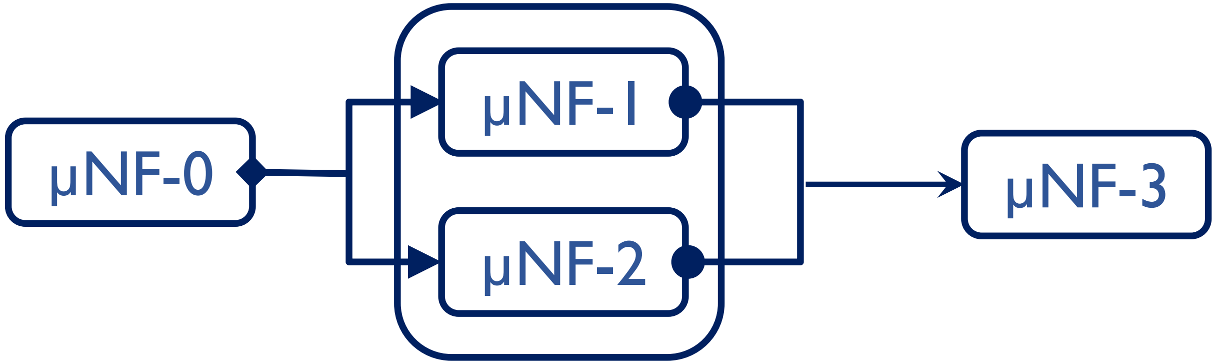


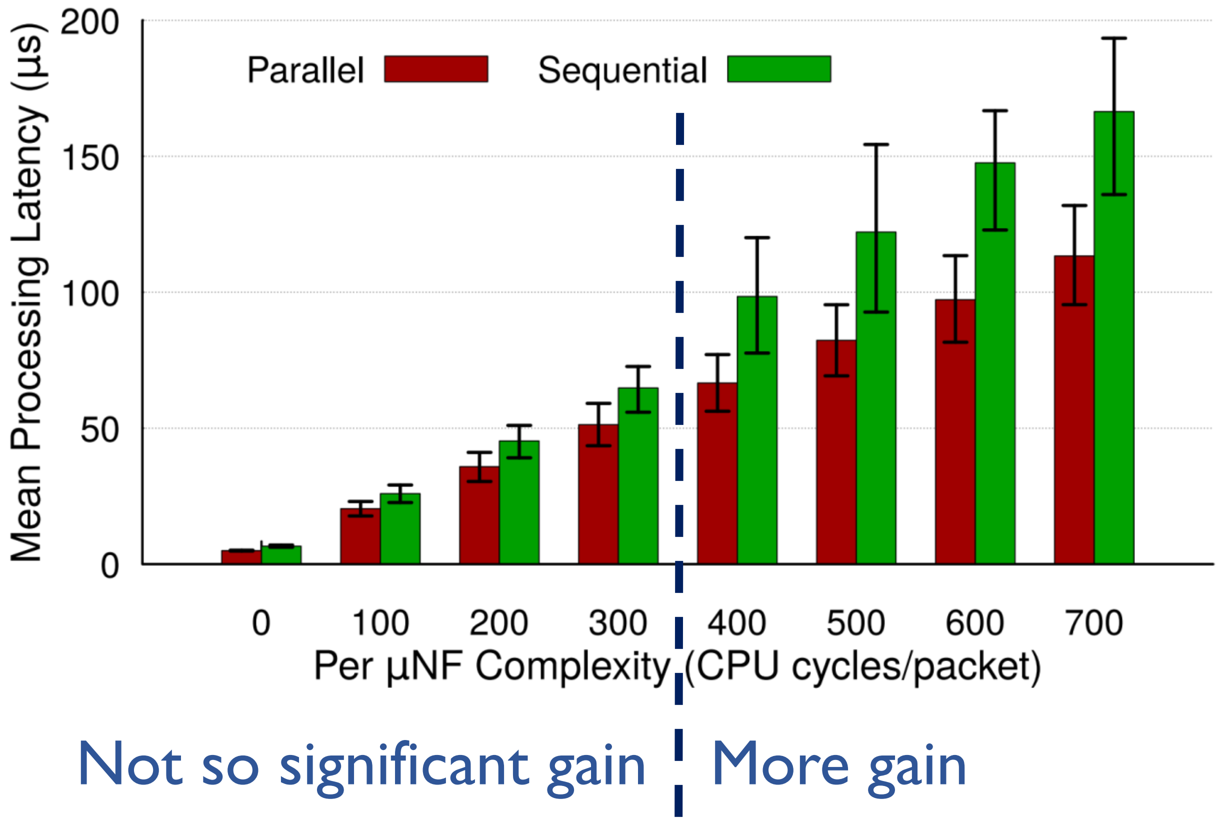


Optimization: Parallel execution of μNFs

Parallelize sequential blocks of μNFs if:

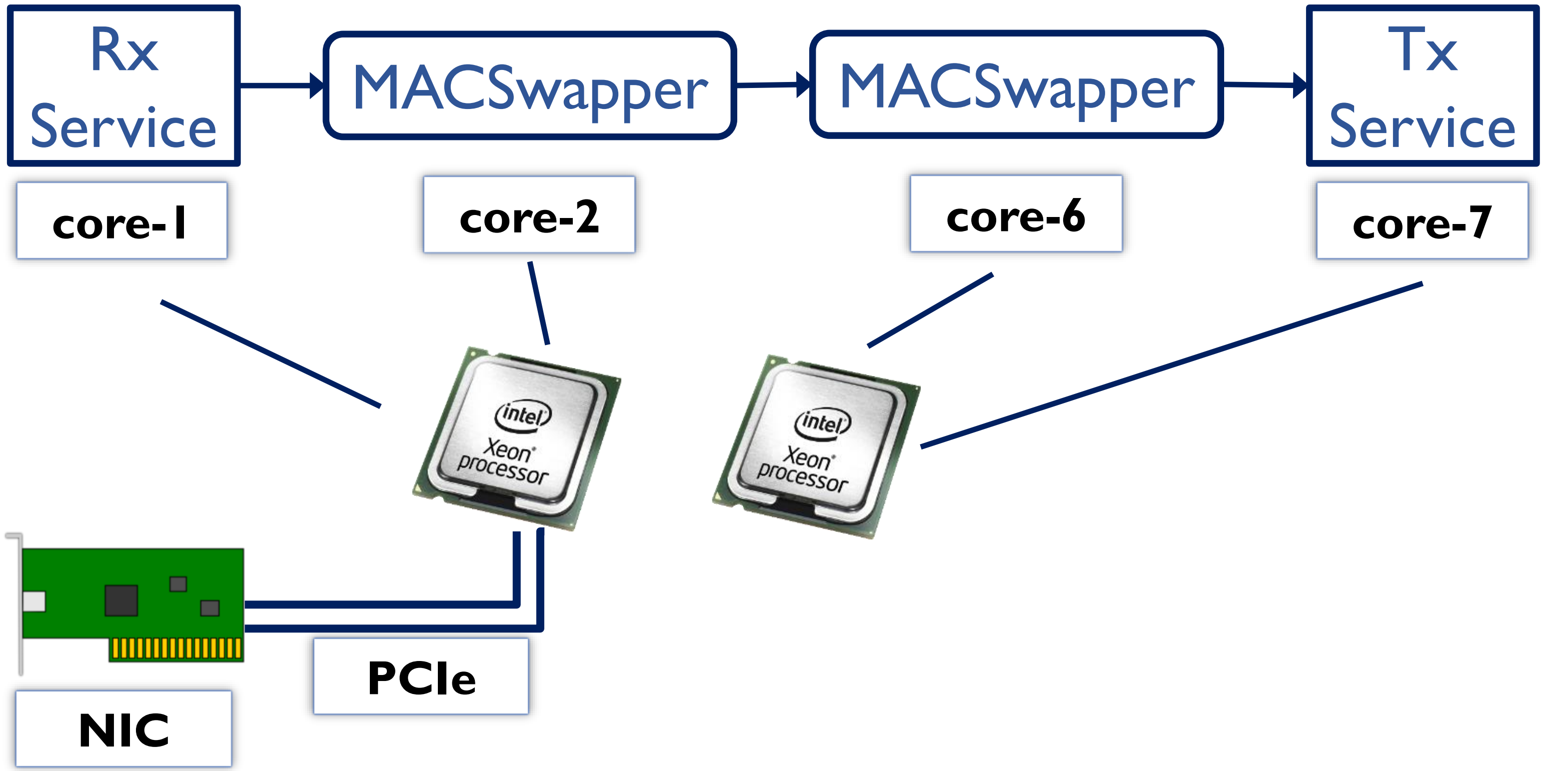
1. The μNFs do not modify the same headers
2. The μNFs do not modify the packet stream



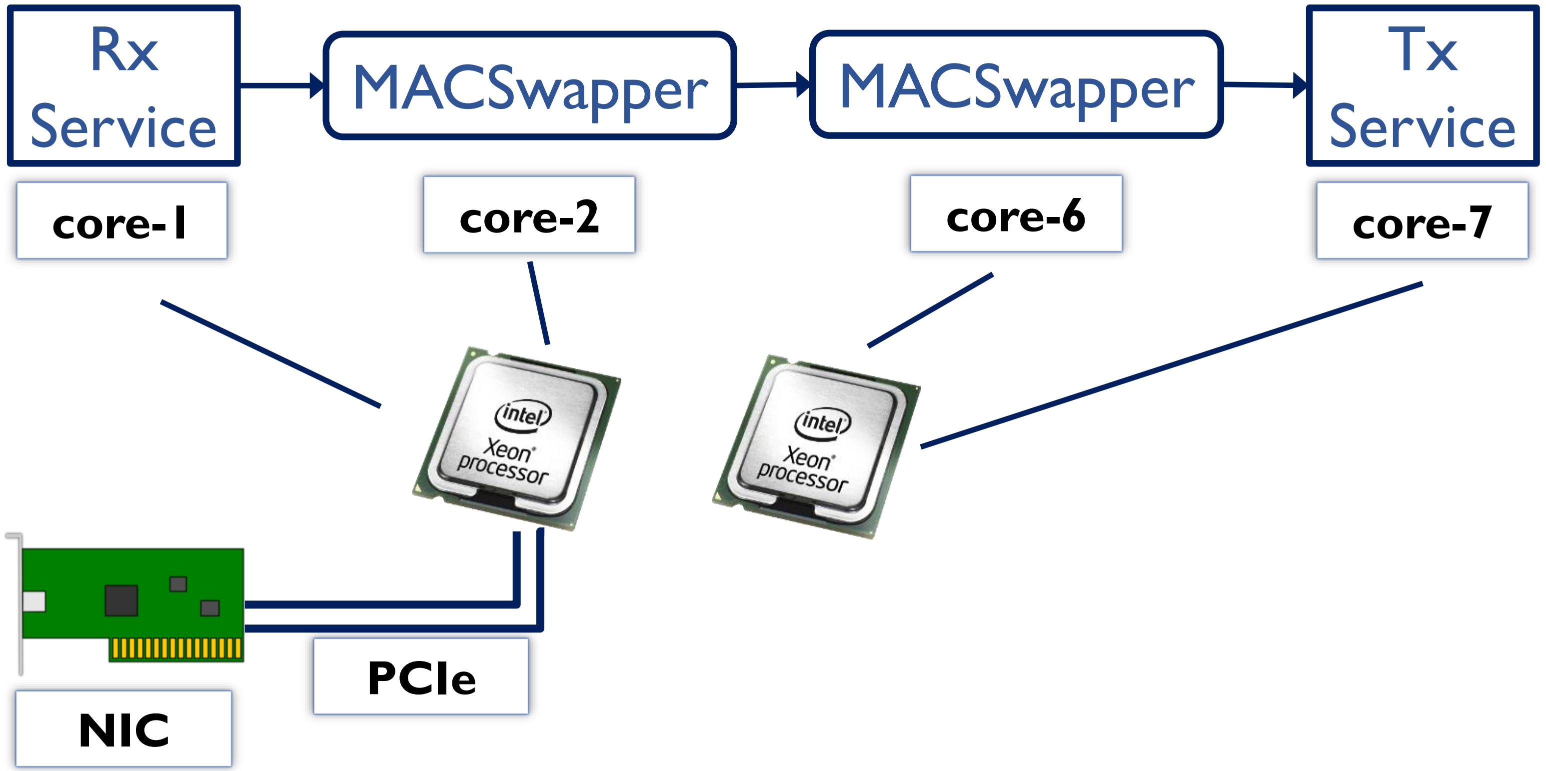


Not so significant gain | More gain

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

1

Optimization: Pipelined Cache-prefetching

1

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

2

While processing packet *i*:

Prefetch a cacheline from packet *(i + k)*

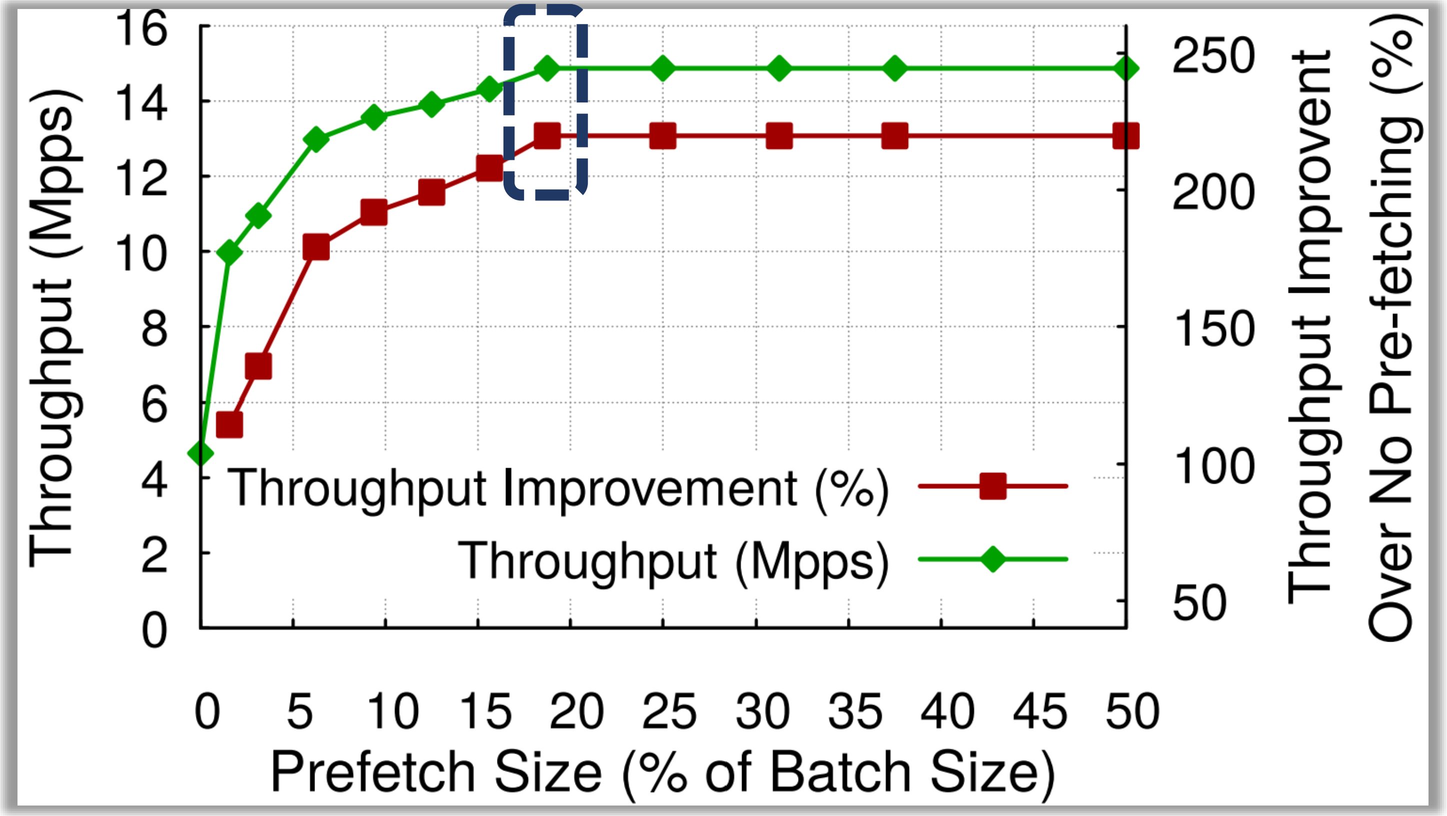
Optimization: Pipelined Cache-prefetching

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

1

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

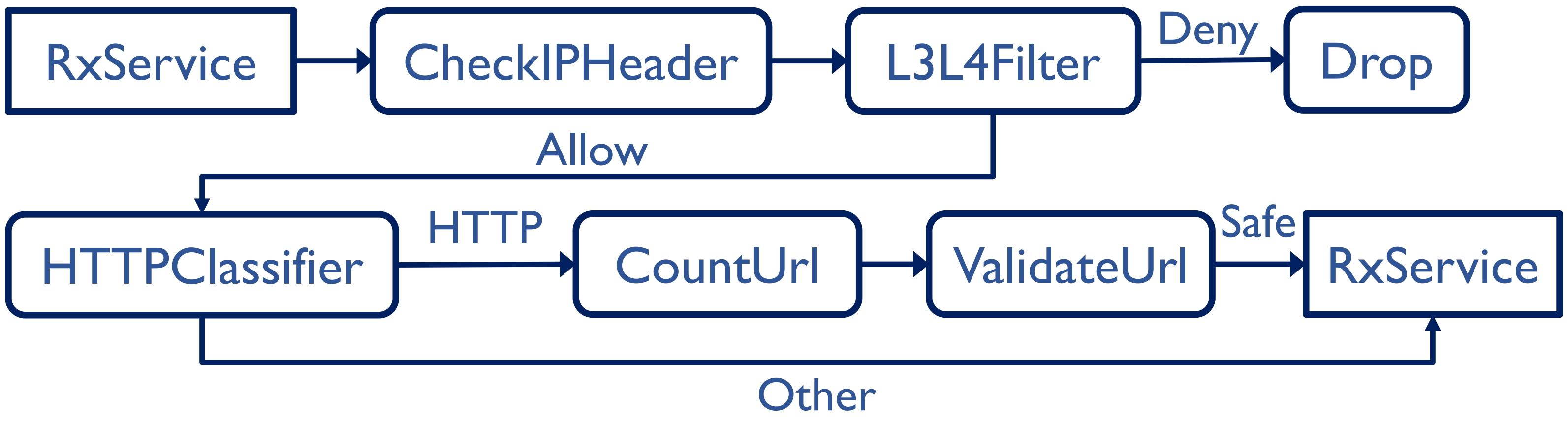
2



Prefetching **~20%** packets in a batch improves throughput by **~3x**

Performance of µNF-based SFC

Edge Fw. → Monitoring → App. Fw



Click Element	Saved cycles/packet	Element weight in C1
CheckIPHeader	27.8%	0.44%
HTTPClassifier	28.9%	47.8%
Overall	**16.8%**	**-**

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

https://github.com/micronf

Questions?