Meeting SLOs in Cross-Platform NFV

Jane Yen*, Jianfeng Wang*, Sucha Supittayapornpong,
Marcos A. M. Vieira, Ramesh Govindan and Barath Raghavan
Network Functions
Network Function Chain

NF chain: concatenated NFs

Packet

Load Balancer

NAT

Firewall
Our Focus: Network Functions at ISP Edge

ISP customer

Firewall

Load Balancer

NAT

ISP
Heterogeneous Hardware

ISP customer

Load Balancer

NAT

Firewall

Offload NF processing to specialized hardware
NF Chains Can Coexist

ISP customer

ISP customer

NAT

Load Balancer

Firewall

ISP
Service Level Objective Violations

Expected: (10Gbps, 30ms)

ISP customer

Expected: (8Gbps, 25ms)
Service Level Objectives (SLOs)

- Throughput rate
- Maximum burst rate
- Guaranteed minimum rate
ISP Pricing Model

Marginal Throughput: $$$
Traffic rate in excess of the minimum

Throughput rate
Maximum burst rate
Guaranteed minimum rate $
Maximizing ISP Revenue

Maximizing aggregate marginal throughput

Overall marginal throughput
\[ = \Delta_1 + \Delta_2 + \Delta_3 \]
Goal

“How do we automatically place and configure multiple NF chains across heterogeneous hardware so that SLOs are met and aggregate marginal throughput is maximized?”
Key Ideas

1. Careful NF placement to meet SLO objectives

2. Automatic compilation and traffic steering to simplify orchestration
Contributions

1. Careful NF placement to meet SLO objectives
   Placer efficiently compacts NFs into hardware accelerators and scale the slowest NFs with available CPU cores such that the SLOs are met and marginal throughput is highest

2. Automatic compilation and traffic steering to simplify orchestration
   Meta-Compiler automatically generates code for NF chain execution on PISA switch and software switch
Placer

Chain #1: Load Balancer → NAT
Chain #2: Firewall → Load Balancer

Placer

Firewall → Load Balancer → NAT
Placer Design Principle

1. Offload as many NF instances to hardware accelerators

Custom Hardware  smartNIC  NetFPGA  Commodity Server

Performance  Hardware Acceleration  Programmability
Placer Design Principle

2. Coalesce NFs to release CPU cores for scaling heavy NFs

Use one core to process two light NFs

Light NF#1  Light NF#2  Heavy NF
Placer Design Principle

3. Arrange cores to achieve high overall marginal throughput
Other Details

- Checking stage constraints using a PISA compiler
- Using MILP problem for CPU core constraints, link capacity constraints etc.
- Placer complexity discussion
Meta-compiler goal: from placement to deployment

Input 1: NF chains (platform-specific)

Input 2: modular NFs (BESS NFs, P4 NFs, etc.)

Output: * platform-specific NF pipeline codes (p4, bess, C, OpenFlow)
Meta-compiler: code generation tasks

- Unify P4 NFs into a P4 NF pipeline
- Generate BESS vNF pipeline with core optimizations
- Synthesize inter-NF routing
Meta-compiler: code generation for P4 NF chains

- Challenge 1: to unify standalone NFs into a PISA switch pipeline
- Challenge 2: to avoid unnecessary NF dependencies

![Diagram showing P4 NF chain and PISA switch pipeline]
Meta-compiler: code generation for P4 NF chains

Our method respects NF dependencies, but does not create unnecessary dependencies

Step (1) converts a DAG to a tree;

Step (2) traverses the P4 NF tree in Preorder;

A final P4 table pipeline
Evaluation

- Can Lemur persistently achieve a higher marginal throughput?
- Is Lemur extendable and scalable?
- What are benefits and overheads of Lemur meta-compiler
Evaluation methodology

Comparison

- Lemur v.s. Various NF allocation/placement alternatives

Metrics

- Feasibility: can an algorithm satisfy the min rate requirement
- Aggregate marginal throughput: derived from the aggregate throughout

Details in the paper
Testbed Setup: heterogeneous platforms

PISA Switch / Openflow Switch

SmartNIC

Commodity Server

Performance

Programmability

Barefoot Wedge 100BF-32X Tofino / Edgecore AS5712-54X OpenFlow switch
Netronome Agilio CX 40Gbps SmartNIC + Berkeley Extensible Software Switch (BESS)
Testbed Setup: heterogeneous platforms

We mainly conduct the experiments in this setting
Lemur outperforms in both metrics
- Lemur always finds a feasible solution while others do 17%-76% of the time.
- Lemur can obtain a higher marginal throughput ranging from 500 Mbps to 24 Gbps.

Lemur’s Placer utilizes the hardware better
- Unlike many existing work, Lemur allows bounces between platforms
Conclusion

Lemur tackles an important NFV problem: meeting SLOs with various hardware accelerators (PISA switches, NICs, servers)

- Lemur considers practical programmability and resource constraints at heterogeneous platforms
- Lemur targets a deployment to meet SLOs

Lemur’s solution outperforms several other alternatives

- Lemur finds a deployment scheme that maximizes the marginal throughput
- Lemur’s placement computation scales well with parallelism
Meta-compiler: synthesize inter-NF routing

Problem: how to ensure that packets are processed by the correct NF?
Our solution: Network Service Header (NSH)
- Maintain chain execution state within packets across platforms
- De/Encapsulate NSH header at each platform’s ingress/egress

<table>
<thead>
<tr>
<th>State</th>
<th>NF</th>
<th>Operate at...</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Encrypt</td>
<td>Server</td>
</tr>
<tr>
<td>3</td>
<td>NAT</td>
<td>PISA switch</td>
</tr>
</tbody>
</table>
Meta-compiler: code generation for vNF chains

Lemur uses BESS as the NF runtime for vNF chains
- Lemur enforces core optimizations via a load balancer

Coalesced run-to-completion subgroups

NAT → BPF
ACL → Encrypt

Modular BESS vNFs
- Encrypt lib

Lemur’s CPU Core optimization

PortInc -> NSHdecap -> Encrypt -> NSHencap -> PortOut

* BESS pipeline code
Lemur’s Placer works by considering the worst-case profiled NF cycle costs
- NF profiles are stable, with the worst-case cycle cost being within 6.5% of the average cycle cost

NF profiles are necessary for achieving a better resource allocation
- Lemur w/o NF profiles generally has lower marginal throughput, and may fail in some cases
Other experiments (multi-server, multi-hardware)

Lemur can be extended to multiple servers and multiple types of hardware