Decoupling Optimizations and Algorithms in Network Functions

Omid Alipourfard and Minlan Yu
13.2 Mpps

\[
\begin{align*}
&\text{SNF pipeline} \\
&\text{Semantically equivalent NF pipeline}
\end{align*}
\]

4.4 Mpps
A decade ago

Hardware boxes
- Faster
- Slow to update
- More expensive

Today

Software boxes
- Slower
- Fast update cycles
- Cheaper

Network functions are popular
Performance is critical

Innovation and cost benefits only come with **good performance**:

- Lower tail and average **latency**
  - RPC like applications
  - Every µs counts
- Higher **throughput**
  - WAN, ISP, and storage like applications
  - Every additional bytes/cycle counts
Many optimizations for packet processing
Many optimizations for packet processing

Batching packets

Better throughput (amortize static cost)
Higher latency (wait for the batch to finish)
Many optimizations for packet processing

Better throughput (depending on cache availability)
Less latency variability
Many optimizations for packet processing

Data structure tuning, e.g., layout, size, algorithm

Better throughput
Lower latency variability (Cache locality)
Many optimizations for packet processing

Many other optimizations, e.g., fastpath, reorganizing the pipeline, end-to-end optimizations.

Lower latency variability (Cache locality)
Better throughput
Whereas optimizations are well known, applying optimizations requires many trials and errors.
Applying optimizations takes a huge effort, cont.

**Platform:**
- Single core
- 64 byte packets
- 10 Gbps (14.8 Mpps)

**Workload:**
- 10 mil unique flows
- Top 1% has 10% volume
Applying optimizations takes a huge effort, cont.

**Platform:**
- Single core
- 64 byte packets
- 10 Gbps (14.8 Mpps)

**Workload:**
- 10 mil unique flows
- Top 1% has 10% volume

With DPDK & compiler optimizations on:

**4.4 Mpps**
Applying optimizations takes a huge effort, cont.
Applying optimizations takes a huge effort, cont.

Platform:
- Single core
- 64 byte packets
- 10 Gbps (14.8 Mpps)

Workload:
- 10 mil unique flows
- Top 1% has 10% volume

User program

Routing

Measurement

Checksum

Prefetch inst.

Batching 8 pkts

Batching 128 pkts

13.2 Mpps

Batching

$P_{op}$

$M_{op}$

$R_{op}$

$C_{op}$

14
Consider a different workload:

- 100 k unique flows
- Top 1% has 99% traffic volume

Optimizations depend on the **workload**

**Platform:**
- Single core
- 64 byte packets
- 10 Gbps (14.8 Mpps)

**Workload:**

- Checksum
- Routing
- Measurement
- User program

\( p \)
Optimizations depend on the workload

Observation:
Top-5 flows have 70% of traffic.

Idea:
Build a fastpath (cache the computation) of top-5 flows.

Platform:
- Single core
- 64 byte packets
- 10 Gbps (14.8 Mpps)

Workload:
- 100k unique flows
- Top 1% has 99% volume

User program
Checksum
Routing
Measurement

Ideas:
- Checksum
- Routing
- Measurement
- User program
Optimizations depend on the **workload**

**Platform:**
- Single core
- 64 byte packets
- 10 Gbps (14.8 Mpps)

**Workload:**
- 100 k unique flows
- Top 1% has 99% volume

**User program**

```
if !top-5-flows:
    goto
```

```
else:
    goto
```

```
pop - Prev-pipeline
```

```
Cached computation
```

Opt. depend on the **workload** and are (very) **HARD**!

**Platform:**
- Single core
- 64 byte packets
- 10 Gbps (14.8 Mpps)

**Workload:**
- 100 k unique flows
- Top 1% has 99% volume

---

**User program**

- **if** !top-5-flows:
  - **goto**

- **else**:
  - **12.5 Mpps** (< 13.2 Mpps)
  - **goto**

---

**Cached computation**
Optimizations depend on the **workload**

**Platform:**
Single core
64 byte packets
10 Gbps (14.8 Mpps)

**Workload:**
100 k unique flows
Top 1% has 99% volume
Optimizations depend on the **workload**

**Platform:**
- Single core
- 64 byte packets
- 10 Gbps (14.8 Mpps)

**Workload:**
- 100 k unique flows
- Top 1% has 99% volume

**User program**
- **Checksum**
- **Routing**
- **Measurement**

**Steps:**
- **P**
- **Checksum**
- **Routing**
- **Measurement**
- **Fastpath extraction**
- **Cached-computation**

**14.8 Mpps**
Very different pipeline!

Δ of 1.4 Mpps
Optimizations depend on the **platform**

- Offload checksum to the NIC.
- Reoptimize measurement + routing

**Platform:**
- Single core & **NIC** *(checksum offloading)*
- 64 byte packets
- 10 Gbps (14.8 Mpps)

**Workload:**
- 100 k unique flows
- Top 1% has 99% volume
Optimizations depend on the **NF-chain**

Redo all the optimizations for the NF chain.

**Impossible to optimize for every NF-chain, platform, and workload.**
What are the **fundamental challenges** to automatically optimize NFs for commodity servers?
**x86** is not designed for packet processing.

- No packet pipelining (only instructions!)
  - Mapping requires knowledge of limited resources
- Non-determinism:
  - Variable memory access latency
  - Shared resources with other application
Optimizing **compiler** goals are different.

- Compiler goal: minimize **completion time** or **code size**
  - NF goals: minimize **latency** or maximize **throughput**
- **Packet optimizations** could change semantics
  - Reorder packets (keep each TCP conn. still in-order)
Optimizations **impact** each other.

- **Trial and error:**
  - Large batches help / Prefetching help
  - Large batches with prefetching pollutes the cache.
- **Proactively** optimizing the code is impossible
  - Workload/Platform/NF Chain
Solution: Decouple algorithms and optimizations in network function design
Domain Specific Language

**Solution:** Decouple *algorithms* and *optimizations* in network function design

Optimizing Runtime
Domain Specific Language

- Express algorithm on a single packet
Domain Specific Language

- Express algorithm on a single packet
  - Make packets **first class type**
Domain Specific Language

- Express algorithm on a single packet
- Include the abstractions available in today’s hardware
Domain Specific Language

- Express algorithm on a single packet
- Include the abstractions available in today’s hardware
  - Packet processing **keywords**

LPMTTable  Hash  TCPChecksum
Domain Specific Language

- Express algorithm on a single packet
- Include the abstractions available in today’s hardware
- Include “hints” to guide optimization choices
Domain Specific Language

- Express algorithm on a single packet
- Include the abstractions available in today’s hardware
- Include “hints” to guide optimization choices
  - Optimization keywords: pure, commutative, ...

Extern  Pure  Commutative
From **language** to **machine code**

Can we systematically make efficient code?

- Workload variations
- Available Platforms
- NF chain

**No single best optimization strategy!**
Profile guided optimization

- Profile code while executing

- Profile traffic characteristics
Template based optimization

- Abstract syntax tree transformation
- Templates with holes
  - Use well-known opt. templates: batch, prefetch, ...
  - Preserve the packet processing semantic
Summary

- End-to-end NF optimization has meaningful gains.
- Figuring out the right set of optimizations is difficult.
- But NF optimizations are well-known.
- By decoupling algorithms and optimizations, we can automatically optimize NF functions.
“In a given paradigm, ... programs become complicated for technical reasons that have no direct relationship to ... problem ... being solved. This is a sign that there is a new concept waiting to be discovered.”

—Peter Van Roy

Thanks!