A Simpler and Faster NIC Driver Model for Network Functions

Solal Pirelli, George Candea

EPFL
Designing for verification can help with performance!
Network future

Bridge  Router  Firewall

↑ Flexibility
↓ Dependability
↓ Performance
Ethernet speeds (Mb/s)

- 1980: 67 μs/pkt
- 1990: 67 ns/pkt
- 2000: <2 ns/pkt
- 2010: <2 ns/pkt

S. Pirelli & G. Candea
A Simpler and Faster NIC Driver Model for Network Functions
Verifying fast stacks

Network Function
NAT, IP router, firewall, bridge, ...

I/O framework
DPDK, Netmap, Click, ...

Network Driver

Network Card

p4v (SIGCOMM’18)  Alembic (NSDI’19)
Vigor (SOSP’19)    Gravel (NSDI’20)
...

Complexity bottleneck!

Out of scope
The driver bottleneck

Simple

Verified
(Vigor\textsuperscript{[1]})

Unverified
(DPDK)

Fast

[1]: A. Zaostrovnykh et al., Verifying software network functions with no verification expertise, SOSP’19
Process packets 1 by 1, in order

*also: NAT, load balancer, ... non-TCP functions*
Core networking can be fast and simple (and verified)
Key results

25% more throughput than full DPDK
160% more than Vigor

8x faster to verify

Pure C implementation
How?

New driver model

Efficient use of NIC

From-scratch implementation
Outline

Intro

Design

Implementation

Evaluation
Classic model (e.g., BSD)

recv() — Copy — Receive

send() — Copy — Transmit
Closed model (e.g., DPDK)
Our model: “TinyNF”
Outline

Intro

Design

Implementation

Evaluation
Separate rings

Receive

No overlap

Transmit
Merged rings

Transmit

Receive

Process

A Simpler and Faster NIC Driver Model for Network Functions
A Simpler and Faster NIC Driver Model for Network Functions

Packet path

Read input metadata

Process

Write output metadata

Maybe

Flush NIC state

Maybe

Recycle buffers

No pointer changes
No pool operations
No explicit batches
Flushing NIC state

Expensive operation

DPDK “batching” estimates network load

We flush when idle or every N packets
Code

Driver: 550 lines

Environment abstraction: 300 lines
(endianness, memory, PCI, time)

100% user mode
Writing drivers

Not as complex as one would think

Publicly available data sheet

Many interpretations; most are trivial
Outline

Intro

Design

Implementation

Evaluation
Evaluation

NAT, Bridge, Policer, Firewall, Load Balancer

Throughput, latency, complexity

Baseline: DPDK
### Complexity

<table>
<thead>
<tr>
<th></th>
<th>Lines of code</th>
<th>Number of paths</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DPDK</td>
<td>TinyNF</td>
</tr>
<tr>
<td><strong>Init</strong></td>
<td>3204</td>
<td>245</td>
</tr>
<tr>
<td><strong>Receive</strong></td>
<td>136</td>
<td>17</td>
</tr>
<tr>
<td><strong>Transmit</strong></td>
<td>122</td>
<td>29</td>
</tr>
</tbody>
</table>

$L = \#\text{links}$

- 7x fewer paths in real NFs
- 8x lower verification time
## Complexity

<table>
<thead>
<tr>
<th></th>
<th>Lines of code</th>
<th>Number of paths</th>
<th>L = #links</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ixy([1])</td>
<td>TinyNF</td>
<td>Ixy([1])</td>
</tr>
<tr>
<td>Init</td>
<td>3204</td>
<td>245</td>
<td>-</td>
</tr>
<tr>
<td>Receive</td>
<td>136</td>
<td>17</td>
<td>1 + A_F + A_S</td>
</tr>
<tr>
<td>Transmit</td>
<td>122</td>
<td>29</td>
<td>14(_L)</td>
</tr>
</tbody>
</table>

\([1]\): P. Emmerich et al., *User Space Network Drivers, ANCS’19*
Performance

1.25x vs. unverified

2.6x throughput vs. verified

Much more in the paper

Virtualization, low-level metrics...

dslab.epfl.ch/research/tinynf
Performance

Worse in “no-op”

CPU frequency reduced to avoid NIC bottleneck
Low-level performance

35 instructions to receive + transmit
100 for DPDK

Lower cache footprint

➡️ Less interference
Conclusion

Designing for verification can help performance!

Core network functions can be both fast and verified

dslab.epfl.ch/research/tinynf