Verification of Software Network Functions with No Verification Expertise

Arseniy Zaostrovnykh, Solal Pirelli, Rishabh Iyer, Matteo Rizzo, Luis Pedrosa, Katerina Argyraki, George Candea

EPFL
Bright Future With Verification
Bright Future With Verification
Vigor enables practical verification of software network functions.

- Virtualized networks
- Critical infrastructure, but unreliable software
- NF operators and devs have no verification expertise
• Traditional verification cost: 5-10x development effort\textsuperscript{†}

• Ultimate goal: zero-cost verification

\textsuperscript{†}seL4 \textsuperscript{[SOSP 2009]}
Ironclad \textsuperscript{[OSDI 2014]}
HACL* \textsuperscript{[CCS 2017]}
EXT_PORT = 2
if received_on_port != EXT_PORT: # Heartbeat
    return ([],[])

else:
    alloc_flow_and_process_packet = True

if alloc_flow_and_process_packet:
    if backend_ip_map.exists_with_idt(new_id, loadbalance_flow_hash(packet_flow)):
        flow_id = backend_ip_map.choose MetroFramework(idt, loadbalance_flow_hash(packet_flow))
        if not flow_id:
            # do nothing
            return (backend_id,)

    backend_id = backend_ip_map.get(id)
    flow_id = flow_id_map[5]
    if flow_id:
        if alloc_flow_and_process_packet:
            flow_id = flow_id_map[5]
            alloc_flow_and_process_packet = False

    flow_id_map[5] = False
    return (backend_id,)

else:
    alloc_flow_and_process_packet = True
Push-button  Pay-as-you-go  Full-stack

Whole SW stack: framework, OS, drivers

NF: 3 KLOC

Framework, driver: 85 KLOC
Whole SW stack: framework, OS, drivers

NF: 3 KLOC

Framework, driver: 85 KLOC

Linux: 23 MLOC
Vigor enables

Push-button ,

Pay-as-you-go , and

Full-stack verification of software NFs.
Outline

- Push-button
- Pay-as-you-go
  - Usage scenario
- Full-stack
- Evaluation [ Practical | No performance overhead ]

vigor.epfl.ch
Network Function (NF) is the core of a network middlebox
NF in a Nutshell

Complex parts
Queue
Hash table

Straightforward parts

Infinite loop

Complex parts

Data structures
NF = NF Logic + Data Structs

- NF developer
  - Implement NF logic

- Library developer
  - Implement data structs

Data structure library
2 Verification Approaches

- **Exhaustive Symbolic Execution (ESE)**
  - (e.g. KLEE, S2E)
- **Theorem Proving (TP)**
  - (as seen in seL4, Ironclad, HACL*)

**Automation** vs **Power**

**NF logic**

**Data structure library**
$SE = \text{Low-Level & Automatic}$

\[ a \leftarrow b + c \]

\[ \Rightarrow b \leftarrow c + d \]
TP = High-Level & Manual

a ← *b

⇒ b ← *(b + 1)
TP = High-Level & Manual

/**< a - head, b - tail */

\[ \text{a <- *b} \]

/**< This is replacing the list head */

\[ \Rightarrow \text{b <- *(b + 1)} \]

/**< This is replacing the list tail */
Verification Steps

1. TP
   - Data structure library

2. ESE
   - NF logic

3. Symbolic traces
   - API contracts
   - Semantic check

Verification Complete
Semantic Check

Symbolic traces

API contracts

destiny
there
possibility

way
will
where
Where there’s a will, there’s a way.
Who Writes The Specs

- NF developer
- NF logic
- Library developer
- Data structure library
- IETF
- IEEE
- In-house design
- NF spec
Specification Abstraction

Where there’s a will, there’s a way.

Vouloir, c’est pouvoir.  Wo ein Wille ist, ist auch ein Weg.

Было бы желание, а способ найдётся.
## Common Abstractions

<table>
<thead>
<tr>
<th>Party</th>
<th>Abstractions used</th>
</tr>
</thead>
<tbody>
<tr>
<td>IETF (NAT, Firewall)</td>
<td>same data structs</td>
</tr>
<tr>
<td>IEEE (bridge)</td>
<td>same data structs</td>
</tr>
<tr>
<td>In-house design (Maglev)</td>
<td>same data structs</td>
</tr>
<tr>
<td>Developers</td>
<td>same data structs</td>
</tr>
</tbody>
</table>

Common abstractions do exist: NF data structures
Insight

NF implementations and specifications use the same set of abstractions
Verification Effort

5-10x
Verification Effort

0.5x (specification effort)
Outline

Push-button

Pay-as-you-go

Usage scenario

Full-stack

Evaluation [ Practical | No performance overhead ]

vigor.epfl.ch
Verification Effort

Variable cost (of property formulation)

Fixed cost of verification

0.5x (specification effort)
Verification Effort

Variable cost (of property formulation)

0.1x (specification effort)
from state import flow_emap, int_devices

EXP_TIME = 10 + 1008
EXT_DEVICE = 1

if a_packet_received:
    flow_emap.expire_all(now - EXP_TIME)

h3 = pop_header(tcpudp, on_mismatch=([[],[]])

h2 = pop_header(ipv4, on_mismatch=([[],[]])

h1 = pop_header(ether, on_mismatch=([[],[]])

if received_on_port == EXT_DEVICE:
    internal_flow = FlowIdc(h3.dst_port, h3.src_port, h2.daddr, h2.saddr, h2.npid)
    if flow_emap.has(internal_flow):
        fl_id = flow_emap.get(internal_flow)
        flow_emap.refresh_idx(fl_id, now)
        out_port = vector_get(int_devices, fl_id)
        return ([out_port], [ether(h1, saddr=..., daddr=...), ipv4(h2, cksum=...), tcpudp(h3)])
    else:
        return ([], [])
else:
    internal_flow = FlowIdc(h3.src_port, h3.dst_port, h2.saddr, h2.daddr, h2.npid)
    if flow_emap.has(internal_flow):
        fl_id = flow_emap.get(internal_flow)
        flow_emap.refresh_idx(fl_id, now)
    else:
        if not flow_emap.full():
            fl_id = the_index_allocated
            flow_emap.add(internal_flow, fl_id, now)
            vector_set(int_devices, fl_id, received_on_port)

        return ([EXT_DEVICE], [ether(h1, saddr=..., daddr=...), ipv4(h2, cksum=...), tcpudp(h3)])
One-Off Property

```
if received_on_port == EXT_DEVICE and flow_emap.has(internal_flow):
    fl_id = flow_emap.get(internal_flow)
    flow_emap.refresh_idx(fl_id, now)
    out_port = vector_get(int_devices, fl_id)
    return ([out_port], [ether(h1, saddr=..., daddr=...), ipv4(h2, cksum=...), tcpudp(h3)])
else:
    pass
```
Pay-as-you-go: ‘pass’

```python
if received_on_port == EXT_DEVICE and flow_emap.has(internal_flow):
    fl_id = flow_emap.get(internal_flow)
    flow_emap.refresh_idx(fl_id, now)
    out_port = vector_get(int_devices, fl_id)
    return ([out_port], [ether(h1, saddr=..., daddr=...), ipv4(h2, cksum=...), tcpudp(h3)])
else:
    pass
```
Pay-as-you-go: ‘…’(ellipsis)

```python
1 if received_on_port == EXT_DEVICE and flow_emap.has(internal_flow):
2     fl_id = flow_emap.get(internal_flow)
3     flow_emap.refresh_idx(fl_id, now)
4     out_port = vector_get(int_devices, fl_id)
5     return ([out_port], [ether(h1, saddr=..., daddr=...), ipv4(h2, cksum=...), tcpudp(h3)])
6 else:
7     pass
```
Outline

Push-button

Pay-as-you-go

Usage scenario

Full-stack

Evaluation [ Practical | No performance overhead ]

vigor.epfl.ch
Usage Scenario

1. **Implement, using Vigor library**

2. Get the (standard) spec

3. ???

4. **VERIFIED**
1. Implement, using Vigor library

2. Get the (standard) spec

3. `vigor$ make verify`

4. VERIFIED

Usage Scenario

Counterexample (execution trace)
Outline

Push-button

Pay-as-you-go

Usage scenario

Full-stack

Evaluation [ Practical | No performance overhead ]

vigor.epfl.ch
Recap: Insights and Design

Observations:

• NFs naturally split into NF logic + data structures
• NF specs and code use common abstractions
• Writing full specs is costly
• Most code in SW stack has no impact on desired properties

How Vigor leverages them:

• Verify different parts with different techniques, then stitch
• Adopt those abstractions for the Vigor library API and contracts
• Support incremental specs
• Exclude irrelevant code and verify the rest
Outline

Push-button

Pay-as-you-go

Usage scenario

Full-stack

Evaluation [Practical | No performance overhead ]

vigor.epfl.ch
Vigor NFs

- NAT
- Load balancer (Maglev)
- Traffic policer
- MAC-learning bridge
- Firewall
Verify On Every Commit

- Verify NF in minutes
- Verify Full-stack in hours
- Embarrassingly parallel
- Found bugs in:
  - DPDK pkt proc framework
  - Intel NIC driver
  - Our NFs

<table>
<thead>
<tr>
<th>NF</th>
<th>NF-only</th>
<th>Full-stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAT</td>
<td>7 sec + 54 x 88 sec</td>
<td>8 min + 488 x 88 sec</td>
</tr>
<tr>
<td>Load Balancer</td>
<td>23 sec + 146 x 219 sec</td>
<td>26 min + 1,336 x 219 sec</td>
</tr>
<tr>
<td>Traffic Policer</td>
<td>14 sec + 37 x 82 sec</td>
<td>6 min + 309 x 82 sec</td>
</tr>
<tr>
<td>Bridge</td>
<td>7 sec + 69 x 80 sec</td>
<td>10 min + 601 x 80 sec</td>
</tr>
<tr>
<td>Firewall</td>
<td>6 sec + 43 x 88 sec</td>
<td>7 min + 369 x 88 sec</td>
</tr>
</tbody>
</table>
Concise and incremental specs

• **1 day** to formulate a full spec

• **30-60 LOC** size of a full spec

• **1-13 LOC** size of a minimal property

• **4-9 LOC** size of a property increment
Outline

Push-button

Pay-as-you-go

Usage scenario

Full-stack

Evaluation [ Practical ] [ No performance overhead ]

vigor.epfl.ch
Latency is Competitive

Latency

- NOOP
- NAT
- Bridge
- Load Balancer
- Traffic Policer
- Vigor NF
- Baseline (Click-based, no batching)

† Moonpol
Throughput is Competitive

Throughput

Throughput

Vigor NF
Baseline (Click-based), no batching

0.0 Mpps
2.3 Mpps
4.5 Mpps
6.8 Mpps
9.0 Mpps

NOOP
NAT
Bridge
Load Balancer
Traffic Policier
Firewall

†Moonpol
Recap: Eval

- Fast: can be done on every commit
- Easy: properties take just a few lines
- No performance overhead (modulo batching support)
Vigor enables NF developers to verify the **Full stack** of their NFs in a **Pay-as-you-go** manner with **no** verification expertise or effort (Push-button)

Vigor enables **practical** NF verification

Get it online! [vigor.epfl.ch](http://vigor.epfl.ch)