Performance Interfaces for Network Functions

Rishabh Iyer, Katerina Argyraki, George Candea

EPFL
# Semantic Interfaces

<table>
<thead>
<tr>
<th><code>connect</code></th>
</tr>
</thead>
</table>
| public void connect(SocketAddress endpoint)  
  throws IOException  |

Connects this socket to the server.

**Parameters:**
- `endpoint` - the SocketAddress

**Throws:**
- `IOException` - if an error occurs during the connection
- `IllegalBlockingModeException` - if this socket has an associated channel, and the channel is in non-blocking mode
- `IllegalArgumentException` - if endpoint is null or is a SocketAddress subclass not supported by this socket

**Since:**
- 1.4

https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html
Semantic Interfaces

```c
int aws_array_list_get_at_ptr(
    const struct aws_array_list* list,
    void **val,
    size_t index)
{
    AWS_PRECONDITION(aws_array_list_is_valid(list));
    AWS_PRECONDITION(val != NULL);
    if (aws_array_list_length(list) > index) {
        *val = (void*)((uint8_t*)list->data +
                        (list->item_size * index));
        AWS_POSTCONDITION(aws_array_list_is_valid(list));
        return AWS_OP_SUCCESS;
    }
    AWS_POSTCONDITION(aws_array_list_is_valid(list));
    return aws_raise_error(AWS_ERROR_INVALID_INDEX);
}

Code-Level Model Checking in the Software Development Workflow, Chong et al., ICSE (2020)
An Ideal Interface

- Simple
  - Concise
  - Accessible
- Precise
Can there exist a performance interface?

- Simple
  - Concise
  - Accessible
- Precise
Performance Interfaces for NFs

- Concise: 100-1000x shorter than NF implementations
- Accessible: use similar primitives as semantic specifications
- Precise: predict NF latency with avg. error of 8%

Simple, precise performance interfaces are useful!
- NF developers can identify performance regressions/bugs
- NF operators can identify root cause of performance anomalies
Performance Interfaces for NFs

Performance interfaces summarize NF latency simply and precisely, just like semantic interfaces summarize functionality.
Outline

- What do performance interfaces look like?
- What could one do with performance interfaces?
- How to extract performance interfaces from NF code?
- Evaluation
Outline

● What do performance interfaces look like?
● What could one do with performance interfaces?
● How to extract performance interfaces from NF code?
● Evaluation
def perf_interface_vsignat(pkt):
    # Perf metric: x86 instructions
    # Resolution: 1
    # NF state: flowtable
    # PCVs:
    # s - flowtable.stale_flows
    # t - flowtable.bucket_traversals
    # c - flowtable.hash_collisions
    x = 19*s*t + 40*s + c + 227*s + 123
    if (pkt.port != internal_network):
        if not (pkt.is_IP):
            return x
        else:
            if pkt.is_TCP:
                if flowtable.contains(pkt.flow):
                    return x + 200
                else:
                    return x + 67
            else:
                if not (pkt.is_UDP):
                    return x + 13
                else:
                    if flowtable.contains(pkt.flow):
                        return x + 290
                    else:
                        return x + 69
            else:
                if not (pkt.is_IP):
                    return x
                else:
                    if pkt.is_TCP:
                        if flowtable.contains(pkt.flow):
                            return x + 18*s*t + 30*s + 304
                        else:
                            return x + 31*w*t + 30*c + 546
                    else:
                        if not (pkt.is_UDP):
                            return x + 13
                        else:
                            if flowtable.contains(pkt.flow):
                                return x + 18*s*t + 30*c + 396
                            else:
                                return x + 31*w*t + 30*c + 548
The performance interface of a program $P$ is a program $S_P$. $S_P$ takes the same arguments as $P$ and returns $P$’s performance.
Latency Metrics:

- x86 instructions
- x86 mem-ops
- CPU cycles

```python
def perf_interface_vignat(pkt):
    # Perf metric: x86 instructions
    # Resolution: 1
    # NF state: flowtable
    # PCVs:
    # s - flowtable.stale_flows
    # t - flowtable.bucket_traversals
    # c - flowtable.hash_collisions
    x = 19*s*t + 40*s*c + 227*s + 123
    if (pkt.port != internal_network):
        if not (pkt.is_IP):
            return x
        else:
            if pkt.is_TCP:
```
PCVs = Performance Critical Variables

PCVs capture the effect of state on NF latency

```python
def perf_interface_vignat(pkt):
    # Perf metric: x86 instructions
    # Resolution: 1
    # NF state: flowtable
    # PCVs:
    # s - flowtable.stale_flows
    # t - flowtable.bucket_traversals
    # c - flowtable.hash_collisions
    x = 19*s*t + 40*s*c + 227*s + 123
    if (pkt.port != internal_network):
        if not (pkt.is_IP):
            return x
        else:
            if pkt.is_TCP:
```

\( r: S_p \) ’s resolution

\( \mathcal{P}(P(I)) \):

P’s performance given input \( I \)

\(|S_p(I) - \mathcal{P}(p_i(I))| < r \)

def perf_interface_vignat(pkt):
    # Perf metric: x86 instructions
    # Resolution: 1
    # NF state: flowtable
    # PCVs:
    # s - flowtable.stale_flows
    # t - flowtable.bucket_traversals
    # c - flowtable.hash_collisions
    x = 19*s*t + 40*s*c + 227*s + 123
    if (pkt.port != internal_network):
        if not (pkt.is_IP):
            return x
        else:
            if pkt.is_TCP:
def perf_interface_vignat(pkt):
    # Perf metric: x86 instructions
    # Resolution: 1
    # NF state: flowtable
    # PCVs:
    # s - flowtable.stale_flows
    # t - flowtable.bucket_traversals
    # c - flowtable.hash_collisions
    x = 19*s*t + 48*s*c + 227*s + 123
    if (pkt.port != internal_network_port):
        if not (pkt.is_IP):
            return x
        else:
            if pkt.is_TCP:
                if flowtable.contains(pkt.flow):
                    return x + 288
                else:
                    return x + 67
            else:
                return x + 13
            if not (pkt.is_UDP):
                if flowtable.contains(pkt.flow):
                    return x + 298
                else:
                    return x + 69
            else:
                return x
    else:
        if not (pkt.is_IP):
            return x
        else:
            if pkt.is_TCP:
                if flowtable.contains(pkt.flow):
                    return x + 18*t + 30*c + 394
                else:
                    return x + 31*t + 30*c + 546
            else:
                return x + 13
            if not (pkt.is_UDP):
                if flowtable.contains(pkt.flow):
                    return x + 18*t + 30*c + 395
                else:
                    return x + 31*t + 30*c + 548

    x = 19*s*t + 48*s*c + 227*s + 123
    if not (pkt.is_IP) or not(pkt.is_TCP or pkt.is_UDP):
        return x + 7
    else:
        if pkt.port != internal_network_port:
            if flowtable.contains(pkt.flow):
                return x + 289
            else:
                return x + 68
        else:
            if flowtable.contains(pkt.flow):
                return x + 18*t + 30*c + 395
            else:
                return x + 31*t + 30*c + 547
def perf_interface_vignat(pkt):
    # Perf metric: x86 instructions
    # Resolution: 10
    # NF state: flowtable
    # PCVs:
    #   s - flowtable.stale_flows
    #   t - flowtable.bucket_traversals
    #   c - flowtable.hash_collisions

    x = 19*s*t + 40*s*c + 227*s + 123

    if not (pkt.is_IP) or not(pkt.is_TCP or pkt.is_UDP):
        return x + 7
    else:
        if pkt.port != internal_network_port:
            if flowtable.contains(pkt.flow):
                return x + 289
            else:
                return x + 68
        else:
            if flowtable.contains(pkt.flow):
                return x + 18*t + 30*c + 395
            else:
                return x + 31*t + 30*c + 547
General-case interfaces

- Precise ✓
General-case interfaces

- Precise ✓
- Simple
  - Concise ✓

```python
def perf_interface_vignat(pkt):
    # Perf metric: x86 instructions
    # Resolution: 10
    # NF state: flowtable
    # PCVs:
    # s - flowtable.stale_flows
    # t - flowtable.bucket_traversals
    # c - flowtable.hash_collisions
    x = 19*s*t + 40*s*c + 227*s + 123

    if not (pkt.is_IP) or not(pkt.is_TCP or pkt.is_UDP):
        return x + 7
    else:
        if pkt.port != internal_network_port:
            if flowtable.contains(pkt.flow):
                return x + 289
            else:
                return x + 68
        else:
            if flowtable.contains(pkt.flow):
                return x + 18*t + 30*c + 395
            else:
                return x + 31*t + 30*c + 547
```
General-case interfaces

- Precise ✓
- Simple
  - Concise ✓
  - Accessible ?

PCVs hard to understand for those who didn’t write the code

```python
def perf_interface_vignat(pkt):
    # Perf metric: x86 instructions
    # Resolution: 10
    # NF state: flowtable
    # PCVs:
    #  s - flowtable.stale_flows
    #  t - flowtable.bucket_traversals
    #  c - flowtable.hash_collisions
    x = 19*s*t + 40*s*c + 227*s + 123

    if not (pkt.is_IP) or not(pkt.is_TCP or pkt.is_UDP):
        return x + 7
    else:
        if pkt.port != internal_network_port:
            if flowtable.contains(pkt.flow):
                return x + 289
            else:
                return x + 68
        else:
            if flowtable.contains(pkt.flow):
                return x + 18*t + 30*c + 395
            else:
                return x + 31*t + 30*c + 547
```
Deployment-specific interfaces

- get joint PCV distribution from given deployment
- replace PCV formulas with desired statistic

```python
def perf_interface_vignat_ds(pkt):
    # Perf metric: x86 instructions
    # Resolution: 10
    # Statistic: 99th percentile
    # NF state: flowtable
    if not (pkt.is_IP) or not(pkt.is_TCP or pkt.is_UDF):
        return 492
    else:
        if pkt.port != internal_network_port:
            if flowtable.contains(pkt.flow):
                return 774
            else:
                return 553
        else:
            if flowtable.contains(pkt.flow):
                return 1000
            else:
                return 1117
```
What does a performance interface look like?

- Program with same inputs that returns the latency
- Resolution: granularity at which interface specifies performance

- General-case interfaces express latency as a function of PCVs
- Deployment-specific interfaces express latency as concrete statistics
Outline

● What do performance interfaces look like?

● **What could one do with performance interfaces?**

● How to extract performance interfaces from NF code?

● Evaluation
## Developer: Identify latency regressions

<table>
<thead>
<tr>
<th>Commit ID</th>
<th>Perf before [# of instrns]</th>
<th>Perf after [# of instrns]</th>
<th>Performance regression [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orig commit</td>
<td>-</td>
<td>1771</td>
<td>-</td>
</tr>
<tr>
<td>873d0501695c</td>
<td>1765</td>
<td>1896</td>
<td>7.42%</td>
</tr>
<tr>
<td>39e58b530a8a</td>
<td>1896</td>
<td>1914</td>
<td>0.95%</td>
</tr>
<tr>
<td>458aa0907b68</td>
<td>1914</td>
<td>1933</td>
<td>0.99%</td>
</tr>
<tr>
<td>15f81d0e7ec6</td>
<td>1930</td>
<td>1946</td>
<td>0.83%</td>
</tr>
<tr>
<td>74c3338c2f7e</td>
<td>1952</td>
<td>1983</td>
<td>1.59%</td>
</tr>
<tr>
<td>d0790d3a3823</td>
<td>1983</td>
<td>2030</td>
<td>2.37%</td>
</tr>
<tr>
<td>All commits</td>
<td>1771</td>
<td>2030</td>
<td>14.62%</td>
</tr>
</tbody>
</table>

Maximum packet processing latency in Katran
# Operator: Root-cause diagnosis

<table>
<thead>
<tr>
<th>Bug</th>
<th>Root cause</th>
<th>Identified as most-likely cause?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spike in median latency of Bridge for uniform random workload</td>
<td>hash-collisions</td>
<td>Yes</td>
</tr>
<tr>
<td>Spike in tail latency of VigNAT due to high churn</td>
<td>expired-flows (batched)</td>
<td>Yes</td>
</tr>
<tr>
<td>Spike in median latency of Maglev on a particular x86 server</td>
<td>active-flowtable-size</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Outline

- What do performance interfaces look like?
- What could one do with performance interfaces?
- How to extract performance interfaces from NF code?
- Evaluation
Performance Interface eXtractor (PIX)

- Input: NF source code in C. Output: Python performance interfaces

- Limitations:
  - Relies on Exhaustive Symbolic Execution (ESE)
    - Single-threaded, static loop bounds, cleanly separated state
  - Interfaces do not account for performance interference
  - Interfaces do not reason about queueing latency
PIX Overview

- Consists of 2 parts: a back-end and front-end

- PIX back-end run by NF developers
  - Input: NF source in C.
  - Output: General-case (GC) interfaces

- PIX front-end run by NF operators
  - Inputs: NF binary, GC interface, packet trace.
  - Output: Deployment-specific interfaces
PIX Back-end

1. NF source
2. Bolt (ESE)
3. NF HW model
4. Py Translation
5. Resolution-based merge
6. GC interface
Step 1: Bolt

- Exhaustively symbolizes the NF code
- Plugs-in contracts for pre-analyzed data structures
- Output:
  - # of x86 instructions, mem-ops per execution path
PIX Back-end

Step 2: NF Hardware Model

- LLC misses are primary cause of increased latency
- Taint analysis to identify potential miss sites
PIX Back-end

Step 3: Python translation

- SMT queries → human-readable python expressions
Step 4: Resolution-based merging

- Eliminates implementation details irrelevant at a given resolution
Outline

● What do performance interfaces look like?
● What could one do with performance interfaces?
● How to extract performance interfaces from NF code?
● Evaluation
Evaluation

- Extracted interfaces for 12 NFs written using DPDK and eBPF XDP
  - 3 NFs used in production (Katran LB, Natasha NAT, Cilium filter)
- Eval questions:
  - Accuracy of PIX-extracted interfaces
  - Time required to extract interfaces
  - Simplicity of PIX-extracted interfaces
    - 100-1000x simpler than NF implementations
Prediction accuracy across deployments

- Evaluated accuracy for 4 deployments
  - 2 workloads (typical, adversarial) x 2 servers (Intel Sandy Bridge, AMD EPYC)
  - Absolute NF latency varies by up to 3x

PIX-extracted interfaces correctly adapt to different deployments
Prediction accuracy across latency percentiles

PIX-extracted interfaces are accurate until the 99th percentile.
Extracting performance interfaces can be part of the regular NF development cycle
Performance Interfaces for NFs

Performance interfaces summarize NF latency simply and precisely, just like semantic interfaces summarize functionality.

Paper and code available at: https://dslab.epfl.ch/research/pix
Backup Slides
Backup: Complexity (Katran Load Balancer)
Backup: PIX Prediction error

![Graph showing prediction error for typical and adversarial traffic over latency percentile. The graph indicates a minimal error for both types of traffic, with a slight decrease towards higher percentiles.]
Backup: Bolt Prediction Error