
Diogo Barradas        Nuno Santos        Luís Rodrigues
Fernando Ramos        André Madeira

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

Salvatore Signorello

LASIGE, Faculdade de Ciências, Universidade de Lisboa
Diogo Barradas - coinrg (IETF 110)
Performance Breakthroughs with Programmable Switches

- Line-speed packet processing at Tbps
- Fully programmable in the P4 language
- Recent focus of HW manufacturers

New opportunities for network security
Securing High-Speed Networks

- Programmable switches are used to:
  - Obfuscate Network Topologies [NetHide, SEC’18]
  - Filter spoofed IP traffic [NetHCF, ICNP’19]
  - Mitigate DDoS attacks [Poseidon, NDSS’20]
  - Thwart network covert channels [NetWarden, SEC’20]

Line-speed packet processing
Highly efficient

Fine-tuned for specific application domain
There are Other Prominent ML-based Security Applications

- Botnet Detection
- Website Fingerprinting
- IoT Behavioral Analysis
- Detection of Covert Channels

Statistical Traffic Analysis
- Packet lengths
- Packets inter-arrival time
+ ML-based classifier

Generic approach towards detecting multiple attacks

Diogo Barradas - coinrg (IETF 110)
Collecting Packet Distributions in Programmable Switches is Hard

- **Stateful memory is severely limited**
  - ~100 MB SRAM
  - No memory for storing many flows

- **Packets must be processed at line speed (< a few tens of ns)**
  - Limited number of operations
  - Reduced [domain-specific] instruction set

It does not seem feasible to obtain packet distributions in programmable switches at scale
Research Question

- Can we collect packet distributions within programmable switches?

Efficient  Generic
Solutions for Collecting Packet Distributions Have a Few Drawbacks

*Flow, USENIX ATC’18

**Generic**
- Large Bandwidth Costs

Netwarden, USENIX SEC’20

**Efficient**
- Application-tailored
FlowLens: a flow classification system for generic ML-based security tasks

- **Flow markers**: Compact representation of packet distributions in prog. switches
- **Flow marker accumulator**: Implementation of flow marker collection in switching hardware
- **Automatic profiling**: Application-tailored configuration of flow markers
- **Evaluation**: Tested in 3 different security tasks
FlowLens Architecture

- **Distributed Deployment**
  - Scale # of measured flows
  - Ensure network visibility
- **Coordinated Operation**
  - Multiple ML applications

---

Diogo Barradas - coinrg (IETF 110)
What does it Take to Compress Packet Distributions Efficiently?

- **Produce** flow markers with two operators
  - Quantization
  - Truncation

---

![Raw packet size distribution](image1)

![Quantized distribution](image2)

QL = 4  ($2^4$ x compression)

![Truncated distribution](image3)

Top-10 bins

Up to 150x size reduction

Diogo Barradas - coinrg (IETF 110)
Implementation of the Flow Marker Accumulator
Typical Workflow for a Newbie in P4

1. Implementation in a software simulator
   ○ Environment: bmv2 P4-reference software switch
     ■ Open-source
     ■ Very flexible target architecture
     ■ Perfect for prototyping
   ○ Required software: P4 Tutorial VirtualBox image

2. Implementation in physical switching hardware
   ○ Environment: Barefoot Tofino ASIC
     ■ Proprietary SDE and documentation
     ■ Target-specific constraints
     ■ Real production networks
   ○ Required software: Intel P4 Studio SDE

Diogo Barradas - coinrg (IETF 110)
How are Flow Markers Collected in the Switch?

- **Programmable packet parsing**
- **Match-action tables**
  - Arranged in stages
  - Match some packet field
  - Change packet headers or metadata

**Feed-forward pipeline**
Sequential computations unrolled across stages
Resources are local to each stage

Diogo Barradas - coinrg (IETF 110)
Performing Quantization in the P4 bmv2 Behavioral Simulator

**Goal:** Leverage as much memory as possible to store flow markers

Develop single action to:
- a) Quantize packet size;
- b) Compute reg. grid index;
- c) Increment register cell

Unfortunately...

This *does not work* in hardware!

This action includes too much complexity for one stage

---

packet size = 64
FlowID = <162.2.13.42, 6901, 147.6.54.129, 3478, 17>
Index = 2

```
action track_flow_1(bit<32> action_index, bit<32> flow_index) {
  bit<32> value;
  bit<32> binIndex = standard_metadata.packet_length >> binWidthShifts;
  bit<32> reg_grid_pos = flow_index << 6;
  reg_grid_pos = reg_grid_pos + (flow_index << 4);
  reg_grid_pos = reg_grid_pos + (flow_index << 3);
  reg_grid_pos = reg_grid_pos + (flow_index << 2);
  reg_grid_pos = reg_grid_pos + (flow_index << 1);
  reg_grid_pos = reg_grid_pos + binIndex;
  reg_grid0.read(value, reg_grid_pos);
  value = (action_index == 1) ? value+1 : value;
  reg_grid0.write(reg_grid_pos, value);
}
```
Restructuring the Quantization Code for the Physical Hardware

Trade-off: Action complexity vs Usable memory

Dependency on computations leads to some memory waste

Diogo Barradas - coinrg (IETF 110)
New Version of Quantization Performs Only Simple Actions in Each Stage

**Stage 1:**
```cpp
action quantization_act(){
    meta.binIndex = (bit<32>)(standard_metadata.packet_length >> BIN_WIDTH_SHIFT);
}
```
*Quantize packet size*

**Stage x:**
```cpp
action set_flow_data(bit<32> flow_offset) {
    meta.rg_cell_offset = flow_offset + meta.binIndex;
}
```
*Compute register grid index to increment*

**Stage x+1:**
```cpp
action reg_grid0_action() {
    bit<16> value;
    reg_grid0.read(value, meta.rg_cell_offset);
    value = value+1;
    reg_grid0.write(meta.rg_cell_offset, value);
}
```
*Increment register cell*

**How can we implement truncation?**
Bins to Truncate are Selected in an Offline Fashion

Recall...

Quantized distribution
QL = 4 (2^4x compression)

Truncated distribution
Top-10 bins

Table-defined in the control plane
Match on quantized packets of interest

Table-assisted truncation design

<table>
<thead>
<tr>
<th>Quant. packet size</th>
<th>Truncated bin offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>54</td>
<td>8</td>
</tr>
<tr>
<td>59</td>
<td>9</td>
</tr>
</tbody>
</table>

Diogo Barradas - coinrg (IETF 110)
Use an additional stage to:

**Stage 2**: Truncate quantized packet size;

Modify further stages to:

**Stage 4**: Compute register grid index;

Truncation Requires Only an Additional Pipeline Stage

Control Plane

Register grids (memory clusters)

sets flow offset in register grid

indexes flow

- low table

FlowID = <162.2.13.42, 6901, 147.6.54.129, 3478, 17>

packet size = 512

Stage 1: Quantization
  QL = 5
  md.binIndex_quant = 16

Stage 2: Truncation
  md.bin_offset = 1
  md.trunc_flag = 1

Stage 3: FT1

Stage 4: Match
  flow_offset = 10
  md.rg_cell_offset = 11

Stage 5: Count
  RG1
  RG2

Stage 4: FT2
  2 17 3 8 2

Stage 4: FT3
  3 16 4 11 2
  10 0 15 4 12
  0 1 0 0 0

Diogo Barradas - coinrg (IETF 110)
How to Automatically Choose Quant/Trunc Parameters?

- Large configuration space
  - Quantization x Truncation

- Leverage Bayesian Optimization

- Automatic Profiler with three criteria
  - Smaller marker for target accuracy
  - Best accuracy given a size constraint
  - Compromise between marker size and accuracy

Saves many hours of testing sub-optimal configurations
Evaluation

- **Scalability** in three ML-based security tasks
  - Covert Channel Detection
  - Website Fingerprinting
  - Botnet Detection

- **Performance** of FlowLens’s profiler

- **Resources consumption**
  - CPU usage (control plane)
  - ASIC usage (data plane)
ML-based Security Tasks

- **Detection of Covert Channels**

- **Website Fingerprinting**
  - *Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier.* Herrmann et al., CCS Workshops, 2009

- **Detection of Botnet Traffic**
## Scalability Gains Overview

- **Scalability** in three use cases
  - Covert Channel Detection
  - Website Fingerprinting
  - Botnet Detection

Check the paper for our comprehensive evaluation!

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Scaling (# flows)</th>
<th>Performance Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covert Channels</td>
<td>150x</td>
<td>-3% accuracy</td>
</tr>
<tr>
<td>Website Fingerprinting</td>
<td>32x</td>
<td>-2% accuracy</td>
</tr>
<tr>
<td>Botnet Detection</td>
<td>34x</td>
<td>-3% recall, -2% precision</td>
</tr>
</tbody>
</table>
FlowLens Scales the Amount of Inspected Flows and Retains Acc.

- **Covert Channel Detection** [Barradas et al.]
  - Legitimate / Modified Skype flows
  - Packet lengths + XGBoost

16x increase in measured flows

<table>
<thead>
<tr>
<th># Bins</th>
<th>1500</th>
<th>375</th>
<th>188</th>
<th>94</th>
<th>47</th>
<th>24</th>
<th>12</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory (B)</td>
<td>3000</td>
<td>750</td>
<td>376</td>
<td>188</td>
<td>94</td>
<td>48</td>
<td>24</td>
<td>12</td>
</tr>
</tbody>
</table>

- Full information = 3000B
  - Detection: 96% accuracy
- Quant (QL=4) = 188B
  - Detection: 92% accuracy

Diogo Barradas - coinrg (IETF 110)
FlowLens Scales the Amount of Inspected Flows and Retains Acc.

- **Covert Channel Detection** [Barradas et al.]
  - Legitimate / Modified Skype flows
  - Packet lengths + XGBoost

- **Quant (QL=4)** + Trunc (top-10) = 20B
  - Detection: 93% accuracy

- **Quant (QL=4)**
  - Full information = 3000B
  - Detection: 96% accuracy

150x increase in measured flows

Diogo Barradas - coinrg (IETF 110)
FlowLens’ Profiler Finds Good Quant. / Trunc. Parameters

- **Automatic profiling (Covert Channel):**
  - 48 valid parameter combinations
  - Set max exploration of 10 combinations

<table>
<thead>
<tr>
<th>Rank (accuracy-wise)</th>
<th>Combination</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>(QL = 2, Top-n = all) = 0.960</td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td>(QL = 3, Top-n = 50) = 0.951</td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td>(QL = 0, Top-n = 30) = 0.947</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>(QL = 3, Top-n = 10) = 0.944</td>
<td></td>
</tr>
</tbody>
</table>

Optimize for a reasonable **Size vs Accuracy** trade-off
FlowLens Imposes a Small Overhead on the Switch

- **CPU usage (ML component):**
  - Botnet detection (our largest model)
  - 140MB out of 32GB RAM
  - 5.6MB storage
  - ~200 μs per prediction

- **ASIC usage (Flow Marker Accumulator):**

<table>
<thead>
<tr>
<th>Computational</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>eMatch xBar</td>
<td>TCAM</td>
</tr>
<tr>
<td>Gateway</td>
<td>SRAM</td>
</tr>
<tr>
<td>VLIW</td>
<td></td>
</tr>
<tr>
<td>8.46%</td>
<td>0.00%</td>
</tr>
<tr>
<td>5.21%</td>
<td>38.54%</td>
</tr>
<tr>
<td>3.39%</td>
<td></td>
</tr>
</tbody>
</table>

Supports flow classification in the control plane

Supports the concurrent execution of other forwarding behaviors
Our Experimentation Artifacts are Publicly Available

- **P4 implementation** of the Flow Marker Accumulator
- **Testbed** for flow marker-enabled classification
  - Includes adaptations for the 3 ML-based tasks covered in this talk

Code available in Github!
https://github.com/dmbb/flowlens
Conclusions

- **FlowLens:** First traffic analysis system for generic ML-based security applications in programmable switches
- **Collects** compressed packet distributions, ensuring:
  - Classification accuracy
  - Small memory footprint
- **Classifies** flows directly on the switch
  - Saves communication, compute, and storage costs

https://web.ist.utl.pt/diogo.barradas
Discussion

- Do you have a “killer app” for FlowLens that you’d like to share?
- Have you deployed P4 code in the Tofino? What difficulties did you face?
- Which data structures have you implemented in switching devices?
- Have you implemented some other kind of ML-based framework in programmable switches?
- Have you tested your own P4 programs in a distributed setting?
  - Like a Tofino-powered PlanetLab?