

FlowLens: Enabling Efficient Traffic Analysis for Security Applications Using Programmable Switches

Diogo Barradas **Nuno Santos** **Luís
Rodrigues**

Fernando Ramos **André Madeira**

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

Salvatore Signorello

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**



STRATAXGS[®]
TOMAHAWK



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]

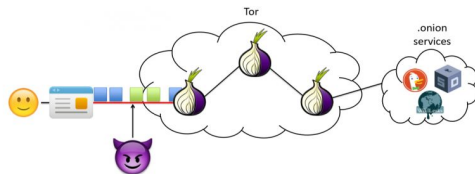
Line-speed packet processing
Highly efficient

Fine-tuned for specific
application domain

There are Other Prominent ML-based Security Applications



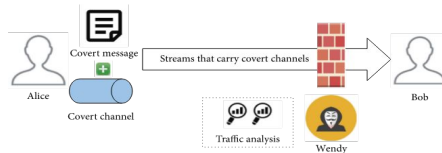
Botnet Detection



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

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

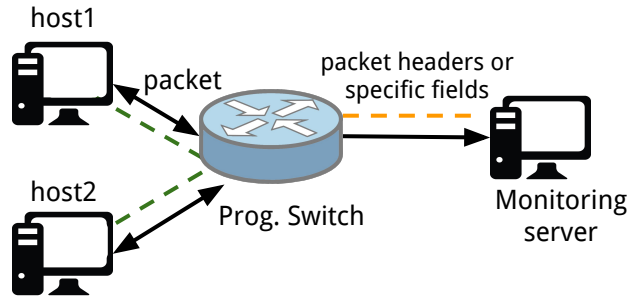
Research Question

- **Can we collect packet distributions within programmable switches?**

Efficient

Generic

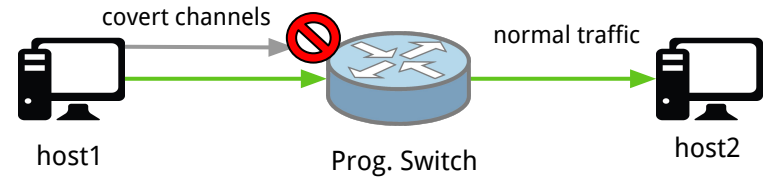
Solutions for Collecting Packet Distributions Have a Few Drawbacks



*Flow, USENIX ATC'18

Generic

Large Bandwidth Costs



Netwarden, USENIX SEC'20

Efficient

Application-tailored

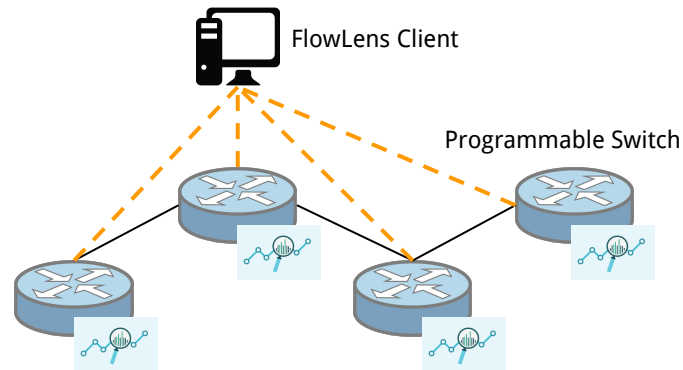
Contributions

FlowLens: a flow classification system for generic ML-based security tasks

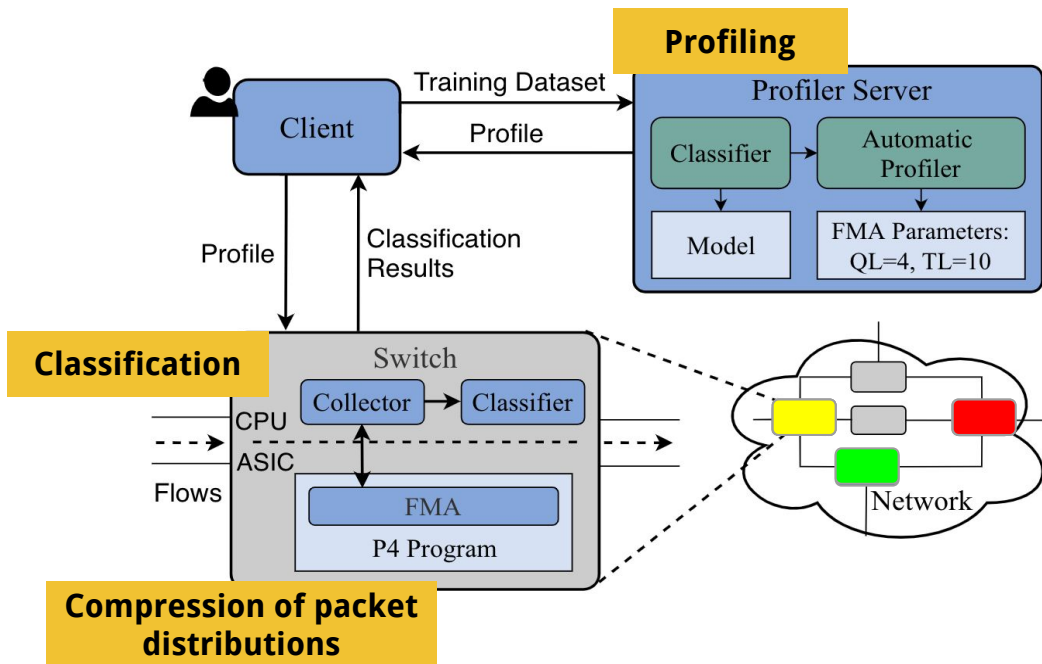
- **Flow markers:** Compact representation of packet distributions in prog. switches
- **Flow marker accumulator:** Implementation of flow marker collection in switching hardware
- **Automatic profiling:** Application-tailored configuration of flow markers
- **Evaluation:** Tested in 3 different security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

Efficient

Generic



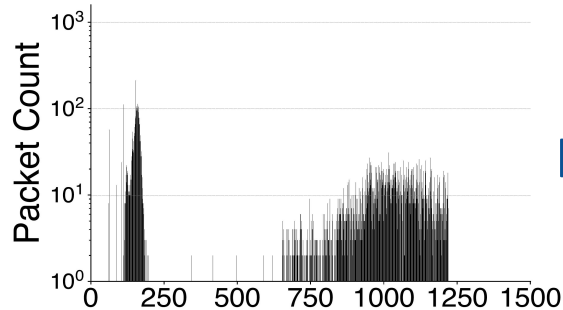
FlowLens Architecture



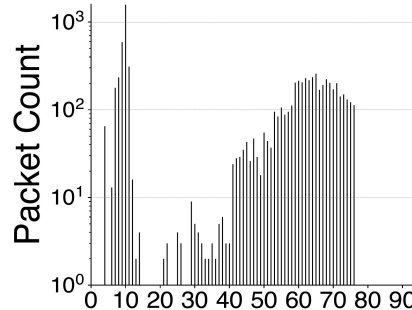
- **Distributed Deployment**
 - Scale # of measured flows
 - Ensure network visibility
- **Coordinated Operation**
 - Multiple ML applications

What does it Take to Compress Packet Distributions Efficiently?

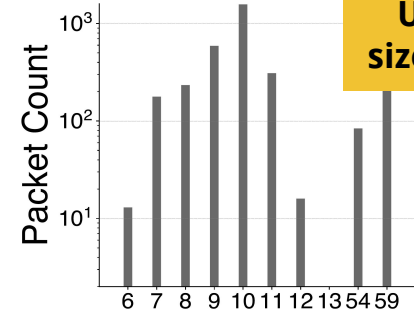
- Produce **flow markers** with two operators
 - Quantization
 - Truncation



Raw packet size distribution



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



Truncated distribution
Top-10 bins

Up to 150x
size reduction

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

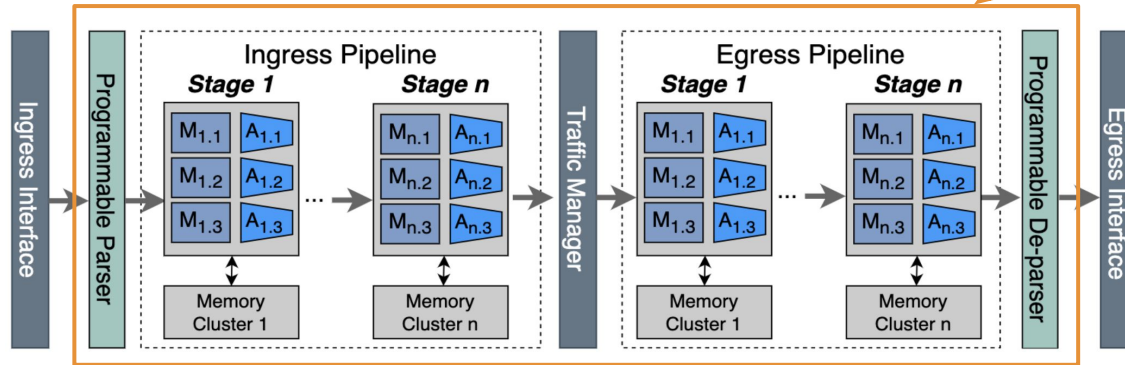
- **Environment:** Barefoot Tofino ASIC
 - Proprietary SDE and documentation
 - Target-specific constraints
 - Real production networks
- **Required software:** Intel P4 Studio SDE



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

P4



Feed-forward pipeline

Sequential computations
unrolled across stages

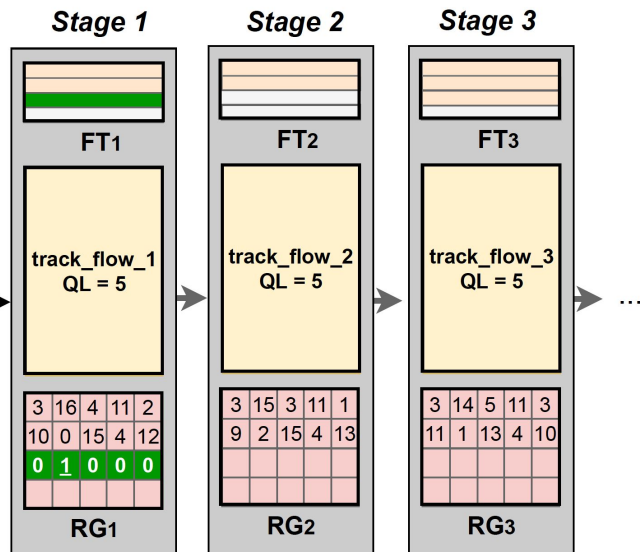
Resources are local
to each stage

Performing Quantization in the P4 bmv2 Behavioral Simulator

packet size = 64

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

Index = 2



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...

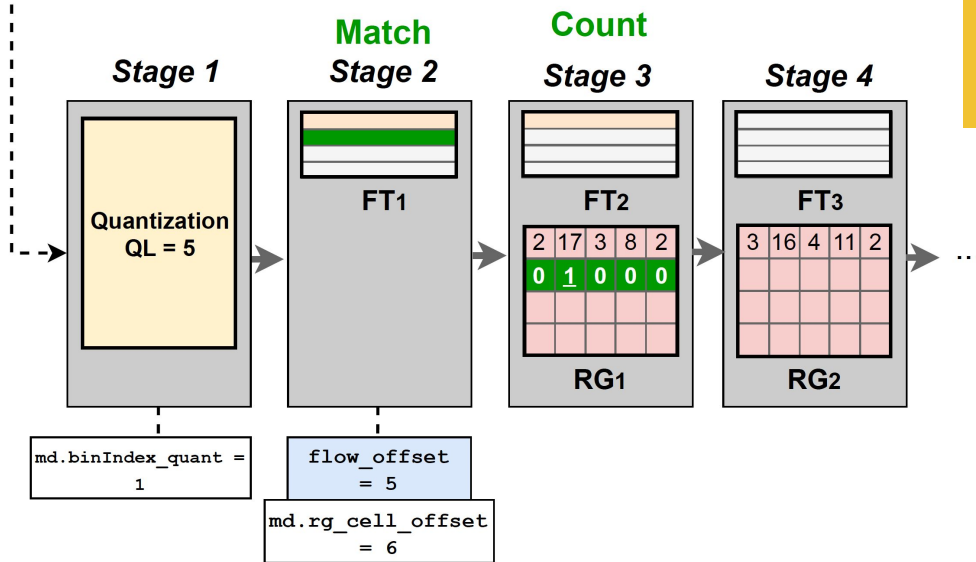
This **does not work** in hardware!

This action includes too much complexity for one stage

Restructuring the Quantization Code for the Physical Hardware

packet size = 64

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



Split computation among different stages:

Stage 1: Quantize packet size;

Stage 2: Compute register grid index;

Stage 3: Increment register cell

Trade-off:

Action complexity vs Usable memory

Dependency on computations leads to some memory waste

New Version of Quantization Performs Only Simple Actions in Each Stage

Stage 1:

```
action quantization_act(){  
    meta.binIndex = (bit<32>)  
        (standard_metadata.packet_length >> BIN_WIDTH_SHIFT);  
}
```

Quantize packet size

Stage x:

```
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:

```
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...

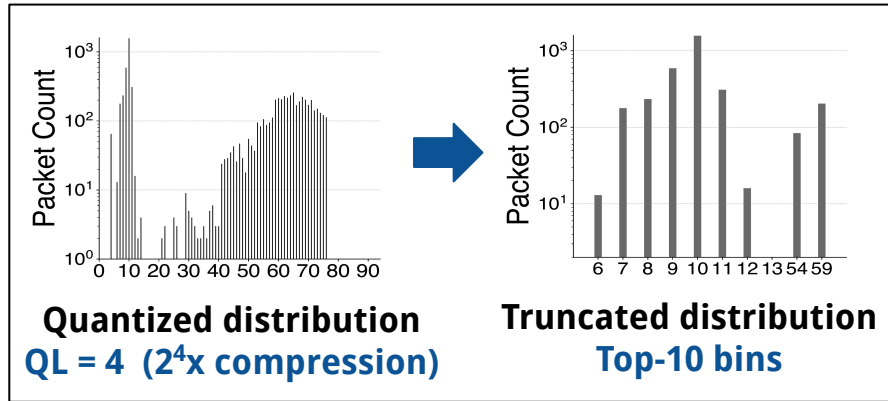


Table-assisted truncation design

Quant. packet size	Truncated bin offset
6	0
7	1
8	2
9	3
10	4
11	5
12	6
13	7
54	8
59	9

Table defined in the control plane

Match on quantized packets of interest

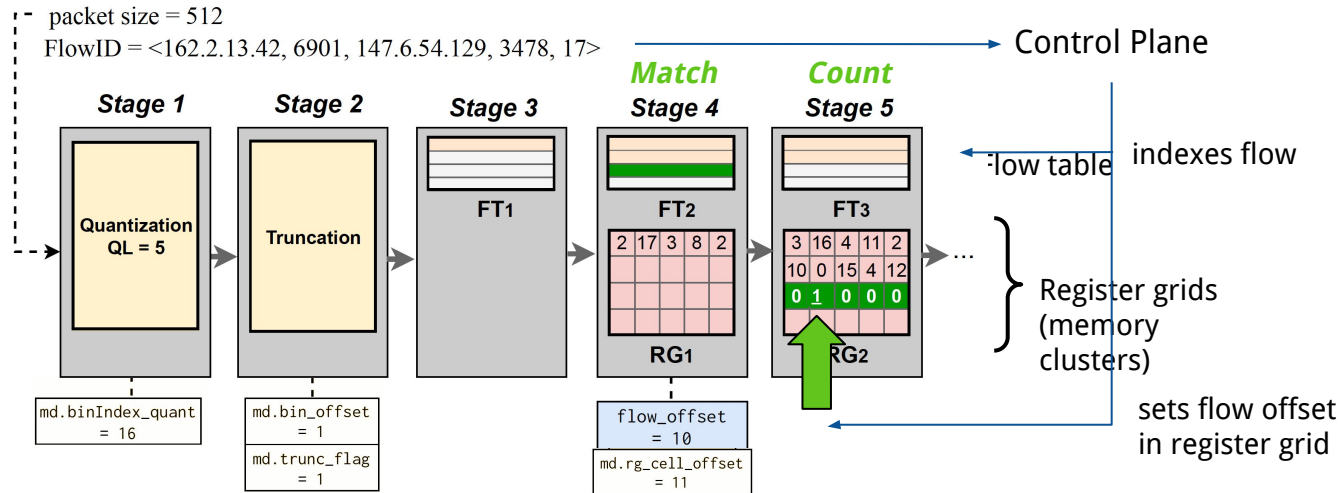
Truncation Requires Only an Additional Pipeline Stage

Use an additional stage to:

Stage 2: Truncate quantized packet size;

Modify further stages to:

Stage 4: Compute register grid index;



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**

- *Effective Detection of Multimedia Protocol Tunneling using Machine Learning.* Barradas et al., USENIX Security, 2018

- **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**

- *PeerShark: flow-clustering and conversation-generation for malicious peer-to-peer traffic identification.* Narang et al., EURASIP Journal on Information Security, 2014



Scalability Gains Overview

- **Scalability in three use cases**

- Covert Channel Detection
- Website Fingerprinting
- Botnet Detection



**Check the paper for our
comprehensive evaluation!**

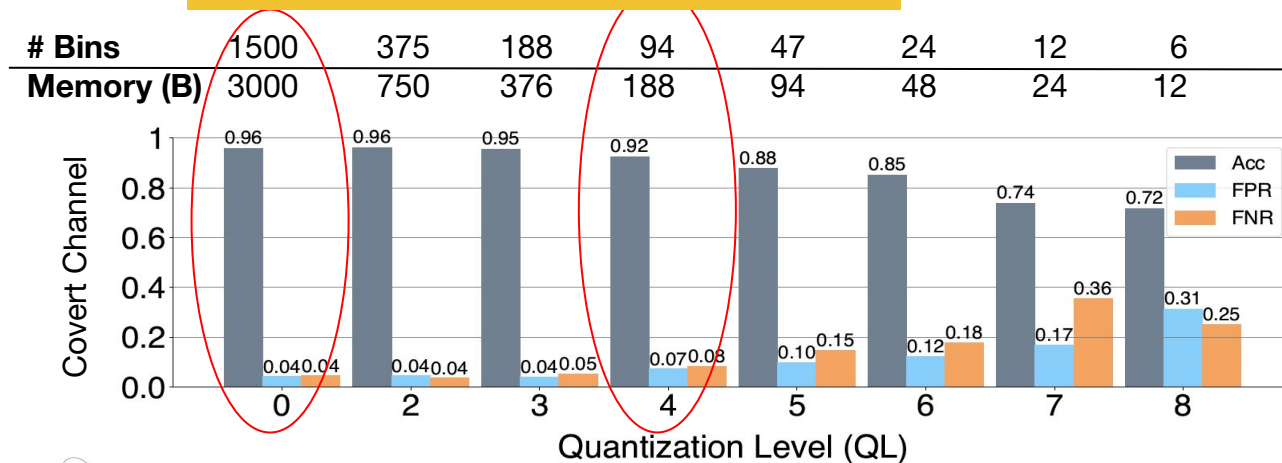
Use Case	Scaling (# flows)	Performance Loss
Covert Channels	150x	-3% accuracy
Website Fingerprinting	32x	-2% accuracy
Botnet Detection	34x	-3% recall -2% precision

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



Full information = **3000B**
Detection: **96%** accuracy



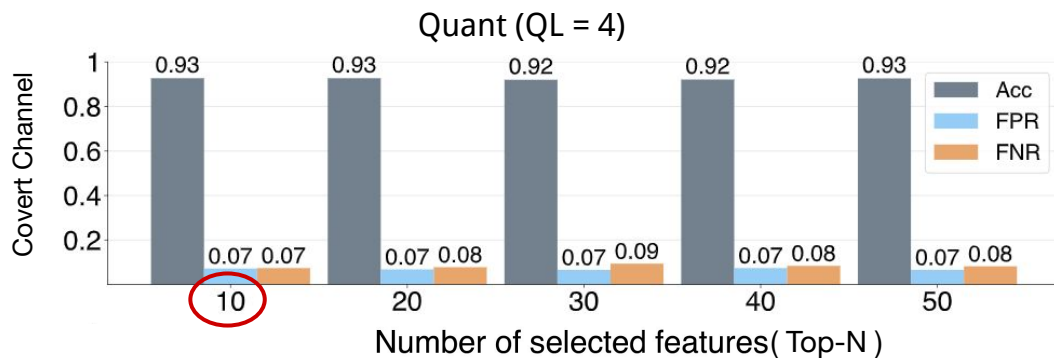
Quant (QL=4) = **188B**
Detection: **92%** accuracy

FlowLens Scales the Amount of Inspected Flows and Retains Acc.

- **Covert Channel Detection** [Barradas et al.]

- Legitimate / Modified Skype flows
- Packet lengths + XGBoost

150x increase in measured flows



Full information = **3000B**
Detection: 96% accuracy



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

FlowLens' Profiler Finds Good Quant. / Trunc. Parameters

- **Automatic profiling (Covert Channel):**

- **48 valid** parameter combinations
- Set max exploration of **10** combinations

Rank (accuracy-wise)	Combination
#1	(QL = 2, Top-n = all) = 0.960
#2	(QL = 3, Top-n = 50) = 0.951
#3	(QL = 0, Top-n = 30) = 0.947
Output	(QL = 3, Top-n = 10) = 0.944

**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 μ s per prediction

Supports flow classification in the control plane

Supports the concurrent execution of other forwarding behaviors

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

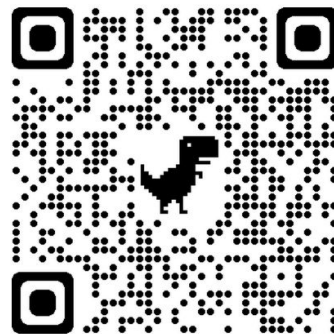
Computational			Memory	
eMatch xBar	Gateway	VLIW	TCAM	SRAM
8.46%	5.21%	3.39%	0.00%	38.54%

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

Thank You!

<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?