The Nuts and Bolts of Building FlowLens

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

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

P4

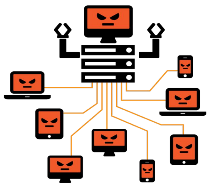
Securing High-Speed Networks

- **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]

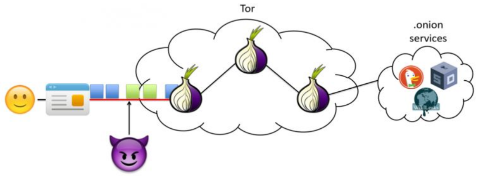
Line-speed packet processing
Highly efficient

Fine-tuned for specific application domain

There are Other Prominent ML-based Security Applications

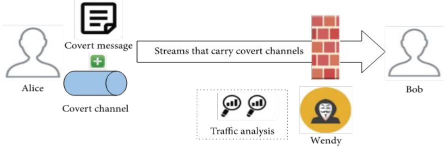


Botnet Detection



Website Fingerprinting

IoT Behavioral Analysis



Detection of Covert Channels

Statistical Traffic Analysis

- Packet lengths
- Packets inter-arrival time

+

ML-based classifier

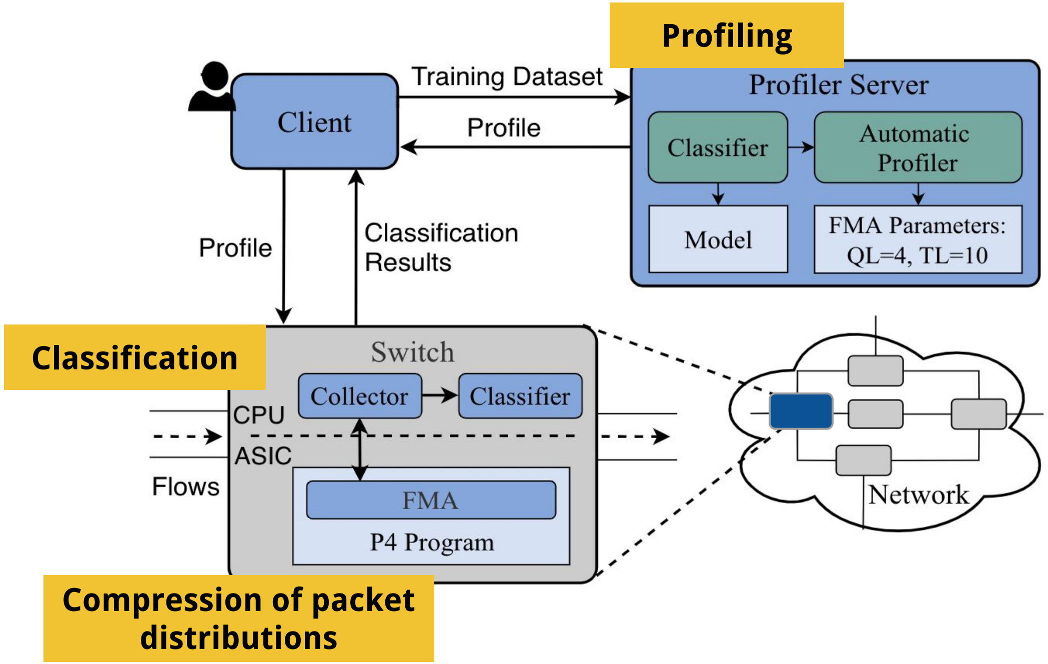
Generic approach towards detecting multiple attacks

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

RQ: Can we collect packet distributions within programmable switches in an efficient way to support generic ML-based security tasks?

FlowLens Architecture



FlowLens: a Flow Classification System for Generic ML-based Security Applications

- **Flow markers**: Compact representation of packet distributions
- **Flow marker accumulator**: HW implementation of flow marker collection
- **Automatic profiling**: Application-tailored configuration of flow markers
- **Evaluation**: Tested for 3 different security tasks

Efficient | **Generic**

Implementation and Evaluation Challenges

1. **Mismatch between software emulator testbed and hardware**
2. **Standardization of heterogeneous ML-based security tasks**
3. **Shortage of convenient means for testing traffic analysis frameworks**

Implementation and Evaluation Challenges

1. **Mismatch between software emulator testbed and hardware**

2. **Standardization of heterogeneous ML-based security tasks**

3. **Shortage of convenient means for testing traffic analysis frameworks**

Implementation of the Flow Marker Accumulator

Typical Workflow for a Newbie in P4

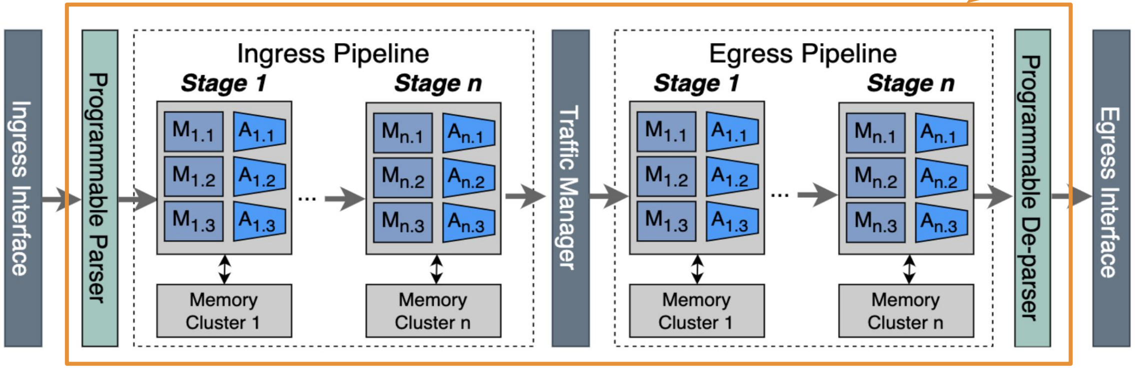
1. **Implementation in a software simulator**
    - **Environment:** bmv2 P4-reference software switch
        - Open-source
        - Very flexible target architecture
        - Perfect for prototyping
    - **Required software:** P4 Tutorial VirtualBox image

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 does a Programmable Switch Look Like?
Protocol-Independent Switch Architecture (PISA)

Flow Marker Accumulator

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



Feed-forward pipeline

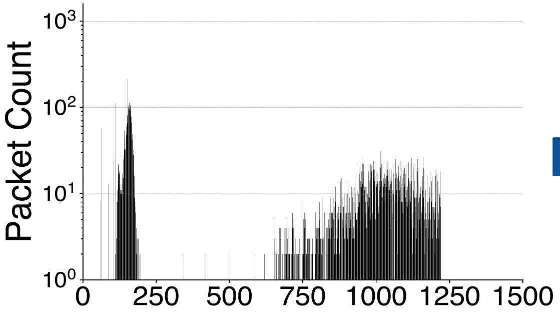
Sequential computations unrolled across stages

Resources are local to each stage

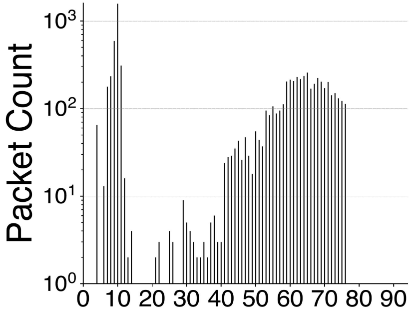
What does it Take to Compress Packet Distributions Efficiently?

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

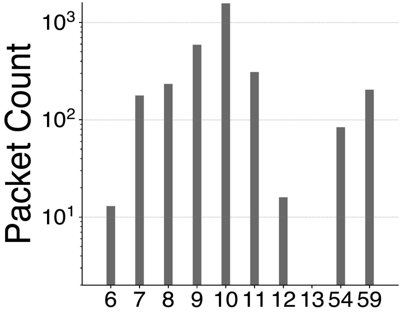
How did we implement these operators?



Raw packet size distribution



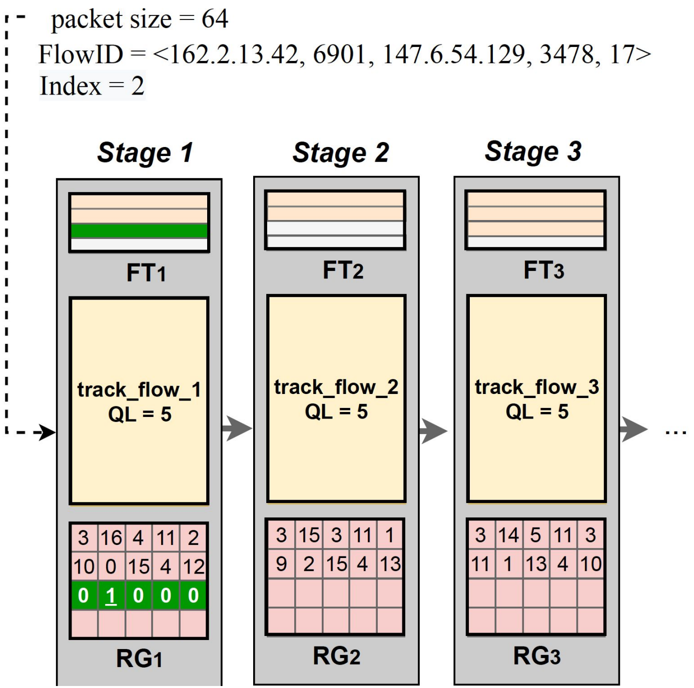
Quantized distribution
QL = 4 (2^4x compression)



Truncated distribution
Top-10 bins

Performing Quantization in the P4 bmv2 Behavioral Simulator

Flow Marker Accumulator



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

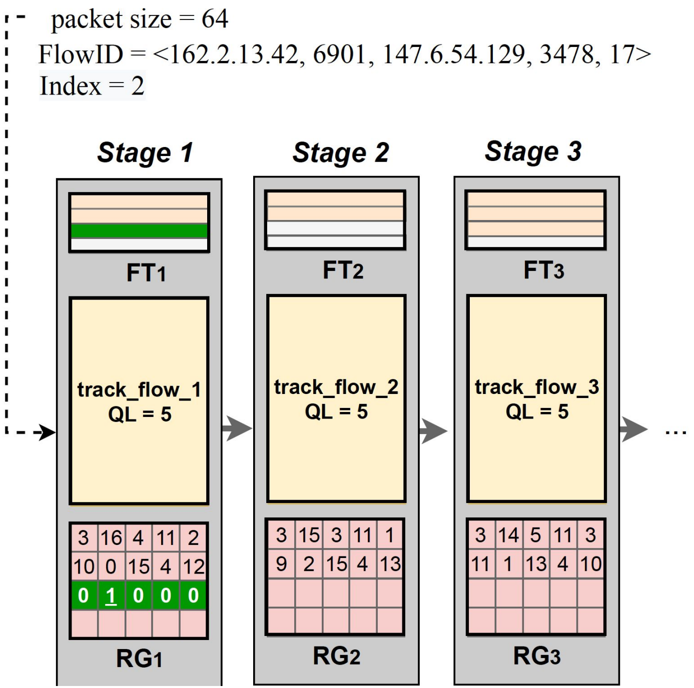
Develop single action to:
a) Quantize packet size;
b) Compute reg. grid index;
c) Increment register cell

Unfortunately...

This does not work in hardware!

This action includes too much complexity for one stage

Performing Quantization in the P4 bmv2 Behavioral Simulator



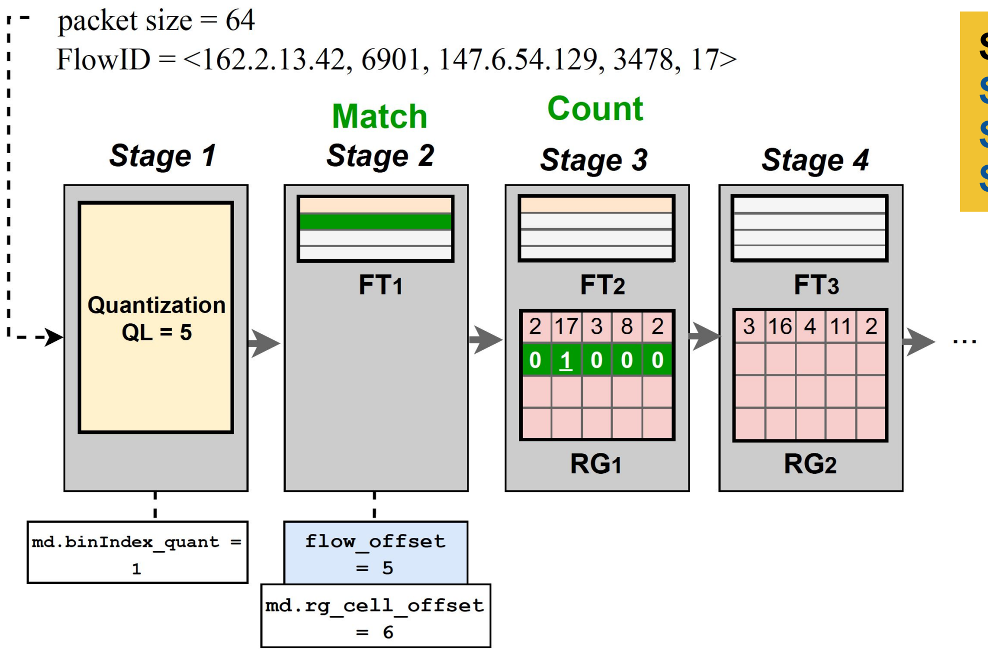
Goal: Leverage as much memory as possible to store flow markers

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

```
action track_flow_1(bit<32> action_index, bit<32> flow_index) {
        bit<32> value;
    a)  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);
    b)  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);
    c)  value = (action_index == 1) ? value+1 : value;
        reg_grid0.write(reg_grid_pos, value);
}
```

Restructuring the Quantization Code for the Physical Hardware



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:

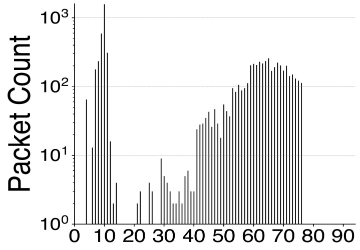
```
action reg_grid0_action() {
      bit<16> value;
      reg_grid0.read(value, meta.rg_cell_offset);
      value = value+1;
      reg_grid0.write(meta.rg_cell_offset, value);
   }
```

Increment register cell

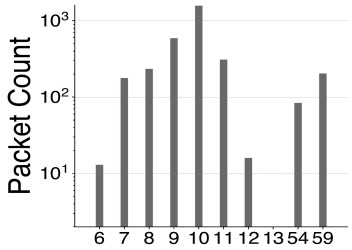
How can we implement truncation?

Bins to Truncate are Selected in an Offline Fashion

Recall…



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



Truncated distribution
Top-10 bins

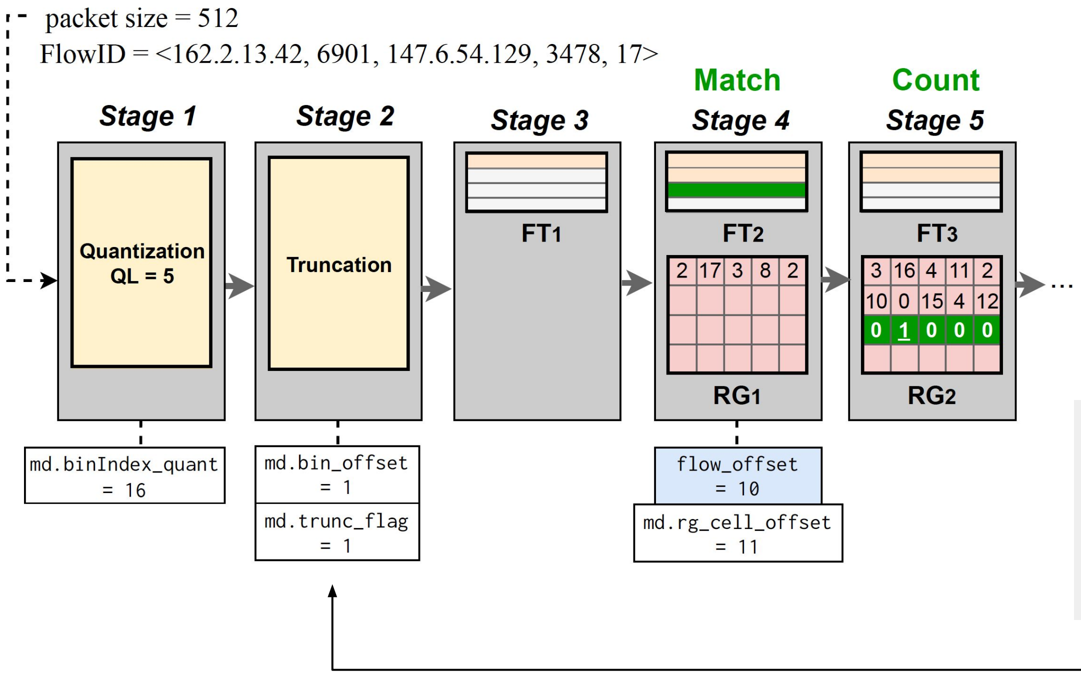
Table defined in the control plane

Match on quantized packets of interest

Table-assisted truncation design

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

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;

Register cell increment is conditional:
Now depends on truncation operation

What did we Learn?

(The hard way)

Flow Marker Accumulator

- **Keep it simple!**
  - Very limited computation per-stage
  - Offload complex computations to the control plane as much as possible
- **Know the layout of your switch's pipeline!**
  - Physical vs logical memory layout (split memory and indexing across stages)

Know the primitives and restrictions of your hardware before you start developing

- Create your system to respect HW primitives & restrictions
  - But don't base your whole design on those
- Abstract system design beyond HW restrictions
  - Your system may be adaptable to multiple HW targets

Discussion

- Have you developed P4 code for other security applications?
- Have you tested your P4 code in other emulators?
- Have you deployed P4 code in the Tofino? What difficulties did you face?
- Besides Tofino, have you developed for any other hardware target?
    - like Smart NICs
- Have you implemented some other kind of ML-based framework in programmable switches?
- What kind of data structures have you implemented?

Implementation and Evaluation Challenges

1. Mismatch between software emulator testbed and hardware
2. **Standardization of heterogeneous ML-based security tasks**
3. Shortage of convenient means for testing traffic analysis frameworks

We Need Multiple ML-based Security Applications to Evaluate our (Purposely) Generic System

ML tasks compatibility

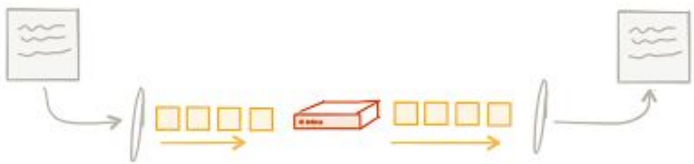
- **Can we find suitable application scenarios?**

- **How can we map the classification workflows of such applications to use flow markers?**

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

Datasets and Classifiers



- **Detection of Covert Channels**
  - Packet traces of Skype flows
  - XGBoost
- **Website Fingerprinting**
  - Packet lengths records of websites browsed over OpenSSH
  - Multinomial naive-Bayes
- **Detection of Botnet Traffic**
  - Packet traces of P2P & botnet traffic (Zeus, Storm, Waledac)
  - Random Forest

Experimental Artifacts

ML tasks compatibility

- **Detection of Covert Channels**
    - Code on GitHub
    - Dataset hosted on authors' webpage
- **Website Fingerprinting**
    - No code, but good guidelines to reproduce testbed
    - Dataset hosted on authors' webpage
- **Detection of Botnet Traffic**
    - Code on GitHub
    - Dataset hosted on authors' webpage

Required Software Packages

ML tasks compatibility

- **Detection of Covert Channels**
  - Python's `sklearn` and `xgboost`
- **Website Fingerprinting**
  - `weka` (+ classifier-specific plugin)
- **Detection of Botnet Traffic**
  - Python's `sklearn`

ML tasks compatibility

How Hard was it to Reproduce the Original Results?

- **Scenario 1** [Covert Channel detection]
  - Easy to replicate
  - Code allowed for obtaining the numbers reported in the paper
- **Scenario 2** [Website Fingerprinting]
  - Easy to replicate
- **Scenario 3** [Botnet Detection]
  - Missing details about exact dataset composition
  - Slight mismatch between obtained numbers vs the paper

What Challenges are Involved in Adapting the Classification Process to Work with FlowLens?

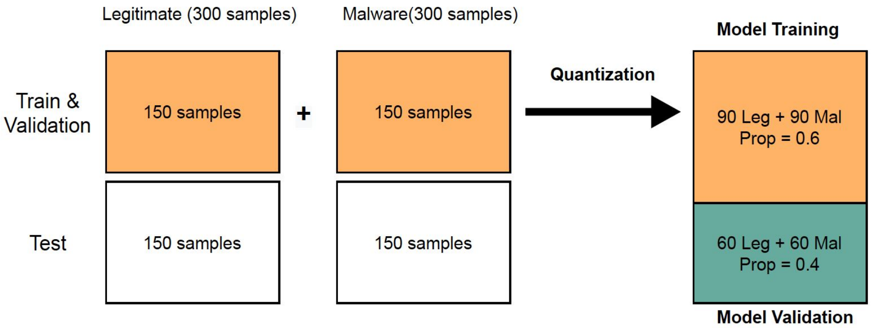
ML tasks compatibility

- **Adaptation of the training and classification workflow**
- **Apples-to-apples comparison with original work**
- **Deal with corner cases**

How can we Adapt the Training and Classification Workflow to Use FlowLens? (I)

ML tasks compatibility

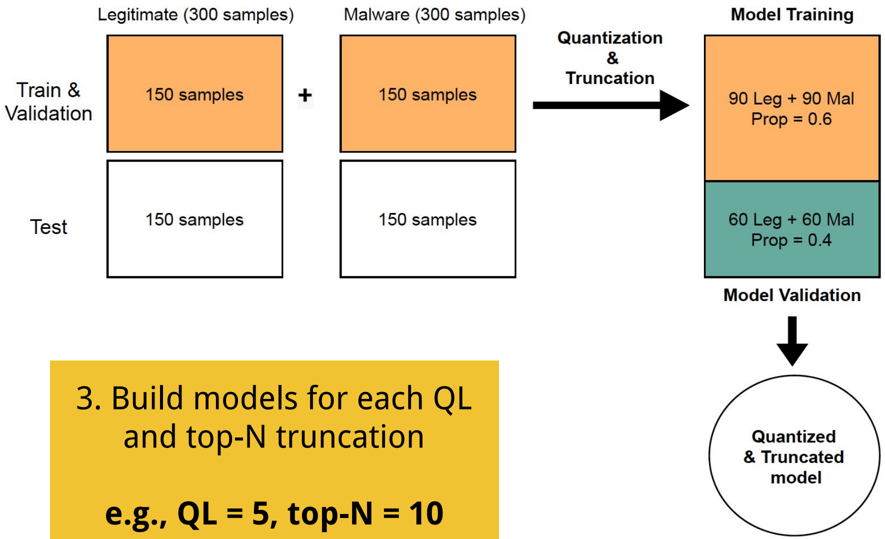
a) Quantization



1. Build models for each quantization level (QL)

2. Check feature importance for each model

b) Truncation

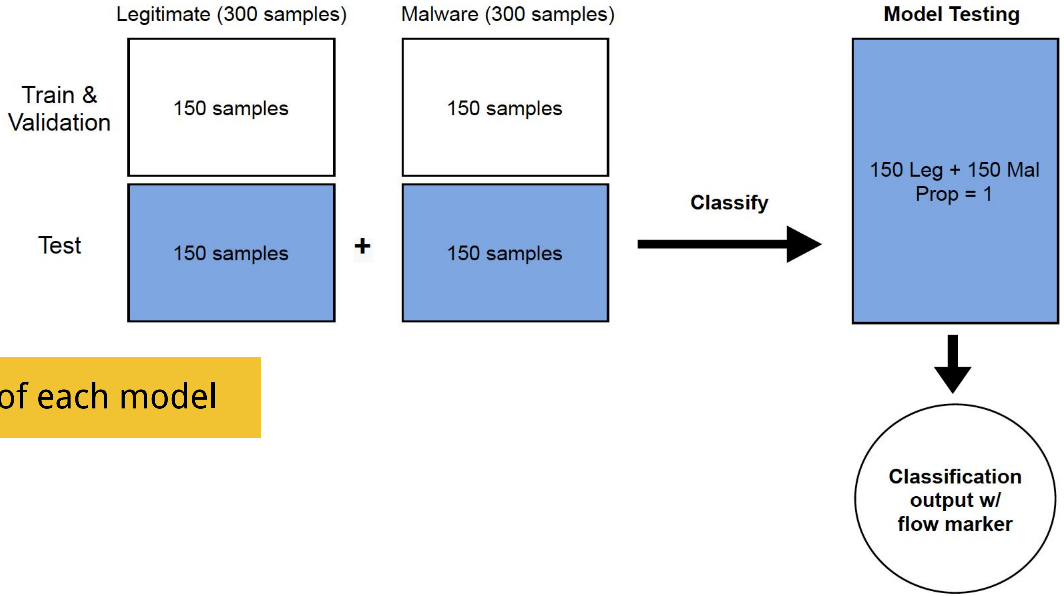


3. Build models for each QL and top-N truncation

e.g., QL = 5, top-N = 10

How can we Adapt the Training and Classification Workflow to Use FlowLens? (II)

c) Test Flow Marker-enabled model



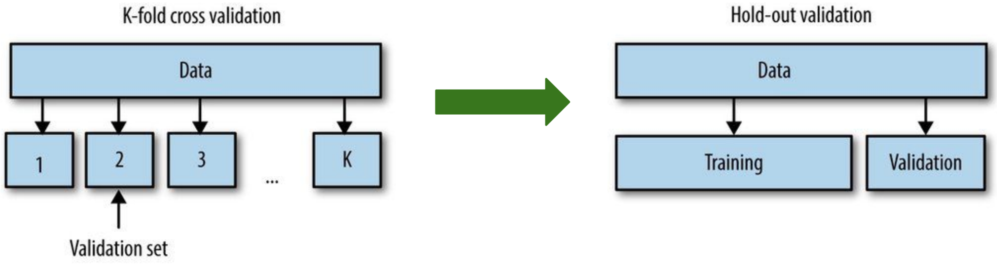
4. Obtain output of each model

We respect original paper's performance metrics

In general, ***Accuracy***

Is it Possible to Perform an Apples-to-apples Comparison with Original Results?

- **Not really...**
  - Adaptation to FlowLens changes dataset composition
  - Model training is different (holdout vs cross-validation)

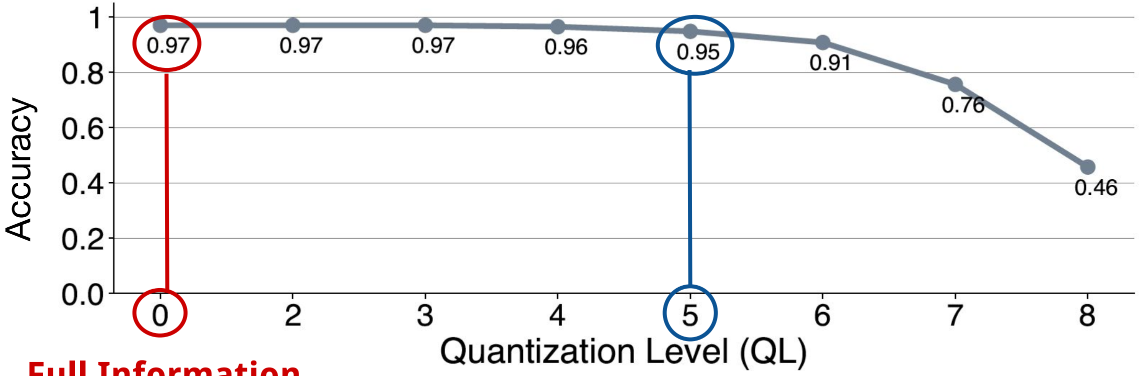


- **But it is fine!**
  - We just want to measure accuracy w.r.t. flow marker size

We can Compare the Accuracy Obtained with Flow Markers vs Full Information

e.g., Website Fingerprinting

# Bins	1500	375	188	94	47	24	12	6
Memory (B)	3000	750	376	188	94	48	24	12



Full Information

Full information = **3000B**
97% accuracy

Quant (QL=5) = **94B**
95% accuracy

What did we Learn?
(The hard way)

ML tasks compatibility

- **Many ML-based security scenarios could use FlowLens!**
  - Widespread dependency on the analysis of packet length distributions
  - Unfortunately, many authors only make available pre-processed features
- **Application scenarios can be adapted despite heterogeneity !**
  - We can reproduce 3rd party results after modification (training / classification)
  - Possible to apply truncation (with slight adjustments in WF)

The heterogeneity of ML scenarios does not prevent experimentation

- ML-based scenarios can be adapted to work in a uniform classification framework like FlowLens, despite relying on different classification processes / datasets
- Make full packet traces available to provide richer datasets

Discussion

- **Do you have a “killer app” for FlowLens that you’d like to share?**
  - Would flow markers make that task harder for any specific reason?

- **Are you involved in some project that requires the adaptation of ML classification frameworks?**
  - What do you adapt? Training process? Dataset composition?
  - Did you adapt something but got bad results?

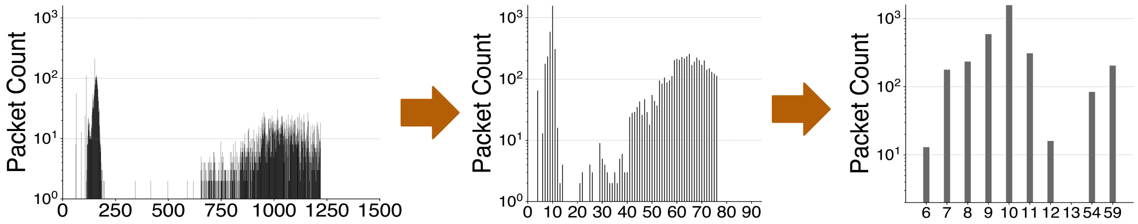
- **Do you have any idea for an alternative profiling step for FlowLens?**

Implementation and Evaluation Challenges

1. Mismatch between software emulator testbed and hardware
2. Standardization of heterogeneous ML-based security tasks
3. **Shortage of convenient means for testing traffic analysis frameworks**

How can we Compare the Scalability Gains Offered by FlowLens with Other Approaches?

- **We compress packet distributions through the creation of flow markers**



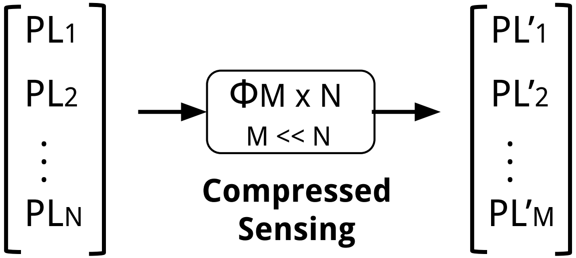
- **How good are flow markers vs. other compression approaches?**

- **How good is FlowLens vs. packet aggregation approaches?**

Comparing the Use of Flow Markers with Other Packet Distribution Compression Approaches

- **Online Sketching**
  - *Online sketching of network flows for real-time stepping-stone detection*. Coskun et al., ACSAC, 2009

- **Compressive Traffic Analysis**
  - *Compressive Traffic Analysis: A New Paradigm for Scalable Traffic Analysis*. Nasr et al., ACM CCS 2017



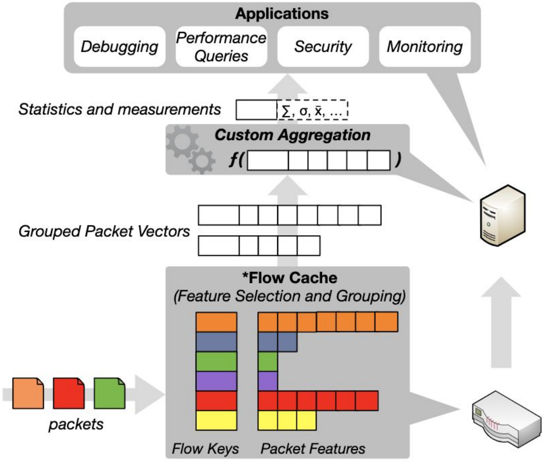
No source code available…

… even upon request

Re-implementation

Comparing FlowLens with Alternative Packet Aggregation Approaches

- ***Flow**
  - *Scaling Hardware Accelerated Network Monitoring to Concurrent and Dynamic Queries With *Flow*. Sonchack et al., USENIX ATC 2018



Opaque target-specific instructions

No end-to-end testbed at the "press of a button"

Analytical Estimation of Required Bandwidth

What did we Learn?
(The hard way)

- **Re-implementation is oftentimes required!**
  - Represents extra effort
  - May fail to respect original implementation decisions (not always obvious)
- **Traffic analysis tooling can be difficult to test!**
  - Hard to experiment with P4 traffic analysis frameworks
  - Programmable switching testbeds are expensive ($$$$)

There is a lack of reproducible experimental testbeds

- Make your code available and provide documentation
- Provide convenient end-to-end experimental testbeds

RW Comparison

Discussion

- **What hurdles did you face when using 3rd party experimental testbeds?**
- **Have you tested your own P4 programs in a distributed setting?**
    - Did you experiment on emulators only?
    - Did you experiment in software switches?
    - Did you experiment in a distributed physical switch infrastructure?

Like a Tofino-powered PlanetLab?

Our Experimentation Artifacts are Publicly Available

- **P4 implementation of 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`

Next Steps and Plans for Workshop Paper

- **Design end-to-end FlowLens test platform in P4 Tutorial VM**
  - **How cool would it be to replay your own traces through FlowLens?**
- **Look towards future developments on distributed testbeds**
  - **What if you could experiment with multiple vantage points enabled with programmable switching devices?**
- **Compile lessons learned - bmv2 vs Hardware**
  - **We hit our own heads on the wall so you don't have to :)**

Thank You!

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

Can we Truncate when Classifiers do not Output Feature Importance?

- **We found a corner case**
  - The multinomial Naive-Bayes classifier does not output feature importance
  - Can we still apply truncation?
- **Insight**
  - Accesses to different websites generate different packet length signatures
  - Weed out bins that never show up on the distribution of a target website

For `amazon.com`:

# Bins	1500	375	188	94	47	24	12	6
#Bins (After Truncation)	159	159	156	87	46	23	12	6

Significant space savings for QL={0,2,3}