Leo: Online ML-based Traffic Classification at Multi-Terabit Line Rate

Syed Usman Jafri                Sanjay Rao                Vishal Shrivastav

Mohit Tawarmalani

PURDUE UNIVERSITY
Why ML-based traffic classification?

- Detecting traffic anomalies, IoT device classification and application classification
- Capture complex patterns w/o peeking into payload
- Learn behavioral patterns from network flow statistics
  ➜ Applicable to encrypted traffic
Limitations of current practice

- Traffic sent off-path for analysis
- Asynchronous $\rightarrow$ slow reaction time
- Challenging to analyze traffic at line rate
Limitations of current practice

- Traffic sent off-path for analysis
- Asynchronous \( \rightarrow \) slow reaction time
- Challenging to analyze traffic at line rate

Can we do classification synchronously at line rate?
Opportunity: In-network compute

Programmable switches offer new opportunities

- Ability to define custom packet processing logic
- Multi-terabit execution of user programs

Goal: Classify packets in real-time!
Opportunity: In-network compute

Programmable switches offer new opportunities

- Ability to define custom packet processing logic
- Multi-terabit execution of user programs
Challenges

Run-time programmable
• Allow model updates with no switch downtime!

Resource-efficient
• Switch HW resources are reserved at compile time
• Worst-case bounds on resources are important!

Programmable switches have limited expressivity
Our focus: Decision Trees

- Programmable switches sufficiently expressive for decision tree operations
- Easily interpretable compared to “black box” models
- Competitive accuracy [1, 2]


Leo contributions

Support a **class** of decision trees in a runtime programmable fashion
- Can support **any** tree within a (depth, leaves, features) class
  - Emphasis on supporting a **class** of trees, **not a specific tree**
- Update the model with **no switch downtime**

**Scalable**
- Resource-efficient (memory, ALUs and pipeline stages)
- Scales to large depths (**2x** of prior work)
- 1 million flows (using 56 stateful feature bits per flow)

**High accuracy**
- Comparable F1 score to control plane (SRAM : 93%, TCAM : 98%)
Prior attempts to support decision trees in data plane

Follow natural tree dependency

Break tree dependency

Memory Stages

pForest [1]
SwitchTree [2]
Infocom [3]

Bottleneck: switch stages

Question:
How well does it perform?
Our analysis of IIsy

- To support all trees with depth $\leq D$, a subset of $N$ features taking values $[0..K]$ in a run-time programmable manner:
  
  - **Proposition 1**: The total SRAM to provision with IIsy is exponential in number of features
  
  - **Proposition 2**: There exist a family of trees with polynomial # of leaves w.r.t $K$ but requires at least an exponential # of TCAM rules

- See paper for full analysis!
Leo – Overview

**Workflow**

1. User specifies a tree class: (Depth, Leaves, Feature set)

2. Chooses a representative tree structure

3. Provisions resources for the representative tree in the switch data plane

4. At runtime, switch control plane can configure any tree in the (D, L, F) class into the data plane
Leo – Programmable Tree Node

**Challenge:** To implement different trees in the (D, L, F) class: Node **features** and **constraints** need to be runtime programmable.

Leo provides a Multiplexed ALU abstraction:

Now to build a runtime programmable tree:
- Reserve a Mux ALU for every node in tree
- **Prohibitively expensive ALU requirement!**
Leo – Programmable Tree Node

Challenge: To implement different trees in the (D, L, F) class: Node features and constraints need to be runtime programmable.

Leo provides a Multiplexed ALU abstraction:

Now to build a runtime programmable tree:
• Reserve a Mux ALU for every node in tree
• Prohibitively expensive ALU requirement!

Question: How to make resource requirements tractable?
Leo – Node Multiplexing

**Key Observation:** At runtime, only 1 node per level is accessed

- Allocate resources for only **one node per level**
- At runtime, **multiplex** the feature comparisons at each node

Results in resource (ALU) efficiency!
Node Multiplexing – Limitation

→ Depth of decision tree limited by # of switch stages
  • A switch with D match-action stages can support a decision tree of depth at most D!

Switch
Stage 1
A < 0

Switch
Stage 2
B < 1
C < 2

Switch
Stage 3
D < 3
E < 4
F < 5
G < 7

Switch
Stage 4
H
I
J
K
L
M
N
O

Limited number of stages
Node Multiplexing – Limitation

→ Depth of decision tree limited by # of switch stages
  • A switch with D match-action stages can support a decision tree of depth at most D!

Question: How to scale tree depth?
→ Key insight: trees have a common substructure → Subtrees
- Flatten the subtree
- Multiplex between subtrees

1. Identify subtrees
A decision tree may be represented as a binary table.
3. Multiplex subtrees

Need half as many stages! (For subtree size = 3)

In general:
Reduce # of stages by factor of \([\log K]\)

\(K = \) subtree size
Leo – Implementation Optimizations

- **Handling stateful features while making efficient use of pipeline stages**
- **Handling transient states correctly during tree updates**
- **Optimization: TCAM stage reduction**
- **See paper for details**
Leo – Worst-case bounds

Leo has acceptable upper bound on the resource requirement:

• For subtrees with $k = 3$ and $I$ internal nodes
  \[
  \text{SRAM entries} = 8I + 3
  \]

• IIIsy number of entries is exponential in number of features $N$:
  \[
  \text{SRAM entries} = \left(\frac{I}{N} + 1\right)^N
  \]

See paper for more analysis (TCAM entries, general subtrees, …)
Evaluation: setup

**Compare Leo with:**
- IIsy, pForest and SwitchTree

**Methodology:**
- Deploy Leo and related work on Intel Tofino switch to find supported tree classes
- Train decision trees to find highest accuracy tree in supported class

**Two intrusion detection datasets:**
- UNSW-NB15 – as a binary classification problem
- CICIDS-2017 – as a multi-class classification

**Metrics:**
- Evaluate on both SRAM and TCAM memory
  - (i) Number of table entries, (ii) Number of switch stages, (iii) Mean F1 score
  - (iv) Num. flows supported and (iv) Scaling depth by introducing leaf limits
Evaluation: resource utilization

SRAM utilization of **complete** tree classes (D, D^2, |F|=10)

pForest/SwitchTree: Limited tree depth

Exponential increase in resource
Evaluation: resource utilization

SRAM utilization of **complete** tree classes \((D, D^2, |F| = 10)\)

![Graph showing SRAM entries required vs. stages required for different tree classes.]

- **Leo** supports trees with 2x larger depth (using same # stages).
- **Leo**: order of magnitude fewer entries.
Evaluation: accuracy

Leo achieves accuracies close to the control plane

Leo: accuracy much higher than prior work

More evaluation in the paper:

- With TCAM, Leo can support:
  - Complete trees of depth 13
  - Depth 22 with 1024 leaves

- Impact of per-flow state on number of flows
- TCAM classification accuracy results

Baseline (Server-based)
Large Tree + All Features
Conclusion

**Support a class of decision trees in a runtime programmable fashion**
- Can support any tree within a (depth, leaves, features) class

**Scalable**
- To large depths (2x of prior work)
- 1 million flows (using 56 stateful feature bits per flow)

**High accuracy**
- Comparable F1 score to control plane (SRAM : 93%, TCAM : 98%)

We released Leo source code
https://github.com/Purdue-ISL/Leo