Runtime Verification of P4 Switches with Reinforcement Learning

Apoorv Shukla (TU Berlin)

with Kevin Nico Hudemann (TU Berlin), Artur Hecker (Huawei), Stefan Schmid (Vienna Uni.)
P4\cite{1}: Data plane Programming Language

- Domain-specific high-level language for data plane programming
- Support for user-defined custom protocols, target independence, etc.

P4 Pipeline: Complex

PSA Architecture with programmable (yellow) and non-programmable blocks (grey)
P4: Multiple versions and platforms

- Versions: P4_{14} & P4_{16}
- Platforms: bmv2, Tofino, eBPF, XDP
- Platform-specific implementations

Interplay between programmable and non-programmable blocks gets complex!
Bugs happen

- Bugs related to memory safety: buffer overflow, invalid memory accesses (detectable by static analysis)

- Runtime bugs related to checksum, ECMP/hash-calculation, platform-dependent, etc.
Runtime bug detection is hard

- P4 is half a program; forwarding rules populated at runtime
- Static Analysis prone to false positives: insufficient
- Switch does not throw any runtime exceptions: hard to catch

This talk: P4 Runtime bug Detection!
Example: Platform-Independent Bug

- L3 switch parser of P4 language tutorials **does not** validate IPv4 ihl
- Packets with IP options are forwarded with wrong checksum
Motivating Example: Platform-Dependent Bug

- Conflicting forwarding decisions can lead to unexpected behavior
- Dependent on implementation of packet replication engine (PRE)

More bug examples in the paper!
Problem Statement

Is it possible to automatically detect \textit{runtime} bugs in P4 switches?
Goal

- Design a system which automatically detects runtime bugs
- Detects both: platform-dependent and –independent bugs
- Is non-intrusive: no changes to the P4 program or switch
Approach in a nutshell

- Use fuzzing, and guide it through reinforcement learning agent
- Generate +ve rewards if an anomaly is detected in the feedback
- Feedback also guides the agent further
• P4RL Agent – Guides Fuzzing
• p4q – Query Language for expressivity, reducing input search space

Credit: https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html
P4RL Reinforcement Learning

- States: Sequence of bytes forming the packet header
- Actions: Add/modify/delete bytes at position $X$
- Rewards: $\begin{cases} 1, \text{ if the packet triggered a bug} \\ 0, \text{ otherwise} \end{cases}$
Reducing Input Search Space for Fuzzing

• Pre-generated dictionary created using control plane configuration, compiled P4 program and p4q queries
• Compiled P4 program in JSON format aids in knowing accepted header layouts
• Check boundary values first for header fields by queries
Query Language: p4q

- Goal: Specify *expected* P4 switch behavior
- If-then-else conditional statements
- Common boolean expressions & relational operators
  
  \[
  \text{ing.hdr.ipv4} \& \text{ing.hdr.ipv4.version} \neq 4, \quad \text{egr.egress_port} = \text{False},
  \]
P4RL Agent-guided Fuzzing
P4RL DDQN

- Combination of double Q-learning and deep Q networks with a simple form of prioritized experience replay
- Select next action based upon the result of feeding current environment state to neural network
- Two separate neural networks for action selection and evaluation
P4RL Workflow

1. Get control plane config

User written queries

Agent

Reward System

Control Plane

P4 Switch

P4RL

P4 Network

2. Select fuzz action

4. Get Reward

3. Send packets & monitor behaviour

1. Get control plane config

Agent

User written queries

Reward System

P4 Switch

Control Plane
Evaluation Strategy

- Target: Publicly available L3 (basic.p4) switch (simple_switch_grpc) implementation
- Baseline: Simple Agent relying on random action selection
- Metrics:
  - Mean Cumulative Reward (MCR) over 10 runs
  - Bug Detection Time
### Bugs found by P4RL in publicly available programs

<table>
<thead>
<tr>
<th>Bug IDs</th>
<th>Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accepted wrong checksum (PI)</td>
</tr>
<tr>
<td>2</td>
<td>Generated wrong checksum (PI)</td>
</tr>
<tr>
<td>3</td>
<td>Incorrect IP version (PI)</td>
</tr>
<tr>
<td>4</td>
<td>IP IHL value out of bounds (PI)</td>
</tr>
<tr>
<td>5</td>
<td>IP TotalLen value is too small (PI)</td>
</tr>
<tr>
<td>6</td>
<td>TTL 0 or 1 is accepted (PI)</td>
</tr>
<tr>
<td>7</td>
<td>TTL not decremented (PI)</td>
</tr>
<tr>
<td>8</td>
<td>Clone not dropped (PD)</td>
</tr>
</tbody>
</table>

**PI** – Platform-independent  
**PD** – Platform-dependent
Learning Performance: P4RL Agent vs. Baseline

➔ P4RL generates $\sim 3 \times$ rewards
Detection Time Speedup: P4RL Agent vs. Baseline

➔ P4RL up to 4.42× faster
Limitations: Undecidability

P4RL engine

Yes

No

Credit: https://www.coopertoons.com/education/haltingproblem/haltingproblem.html
Conclusion

- P4RL’s machine learning-guided fuzzing enables detection of complex runtime bugs (non-intrusively)

- Identifies platform-dependent and -independent bugs

- Ensure correctness in P4 deployments
Summary

1. Get control plane config
   - P4 Switch
   - P4 Runtime
   - Control Plane
2. Select fuzz action
3. Send packets & monitor behavior
4. Get Reward

Contact: apoorv@inet.tu-berlin.de
Code: gitlab.inet.tu-berlin.de/apoorv/P4ML