Fix with P6: Verifying Programmable Switches at Runtime

Apoorv Shukla
(Huawei Munich Research Center)

with K. N. Hudemann (SAP), Z. Vági (Swisscom), L. Hügerich (TU Berlin), G. Smaragdakis (TU Berlin/MPI), A. Hecker (Huawei MRC), S. Schmid (Vienna Uni.), and A. Feldmann (MPI)

Programming Protocol-Independent Packet Processors (P4[2]) Lang.

- High-level language: data plane programming
- P4 programs dictate: what packets and how packets processed
- Allows user-defined custom protocols/target independence
- P4 programs: compiled to run on P4 switches

[2] P. Bosshart, D. Daly, G. Gibby, M. Izzardy, N. McKeown, J. Rexford, C. Schlesinger, D. Talaycoy, A. Vahdat, G. Varghese, D. Walker. P4: Programming Protocol-Independent Packet Processors. SIGCOMM' 14.

Unfortunately, Runtime Issues happen

Runtime issues occur under diverse input workloads:

- Checksum and ECMP/hash-calculation
- Platform-dependent bugs

Need: Runtime verification to check switch/network behavior

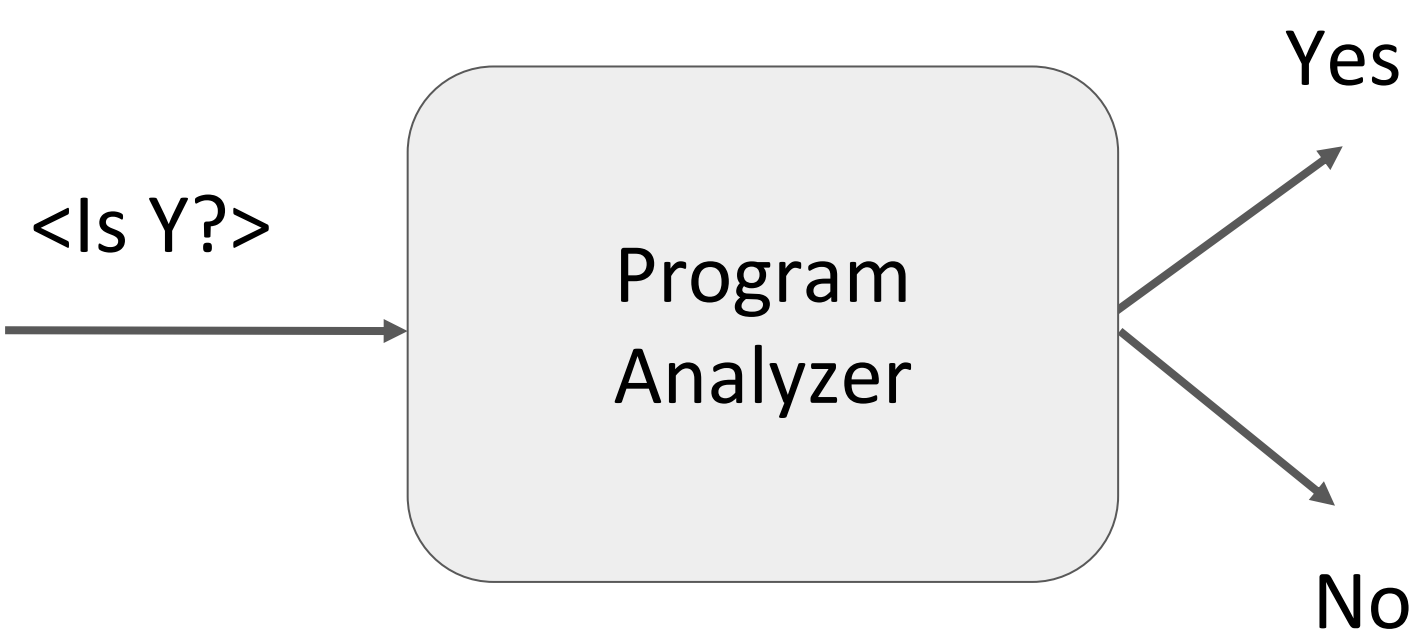
Runtime Bug Detection in Single P4 Switch: **Hard**

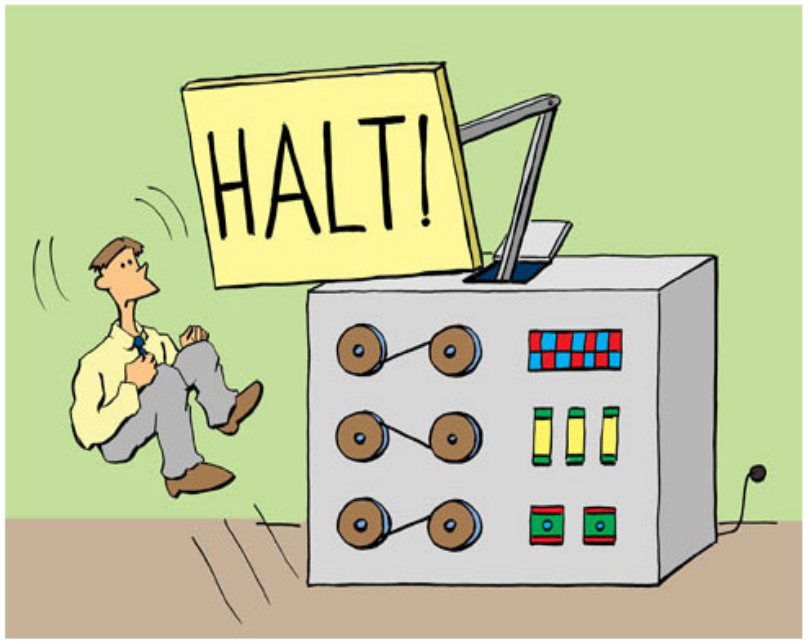
- P4 is *partial* program: **rules populated at runtime**
- Generate packets of interest: **hard**
- No runtime exceptions in P4: **hard to catch**

- Dense conditions, domain-specific constructs: **hard to localize**

Static analysis detects memory safety violations not runtime bugs

Challenge: Undecidability [Rice53]





Alan designed the perfect computer

Does Property Y occur or not?

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

Problem Statement

Can we detect "persistent" runtime bugs and mitigate them in a single P4 switch?

Progress in P4 Verification

P4NoD [MSR Tech. Report' 16]
P4K [P4 Workshop' 17]
Vera [SIGCOMM' 18]
p4v [SIGCOMM' 18]
ASSERT-P4 [SOSR'18, CoNEXT'18]

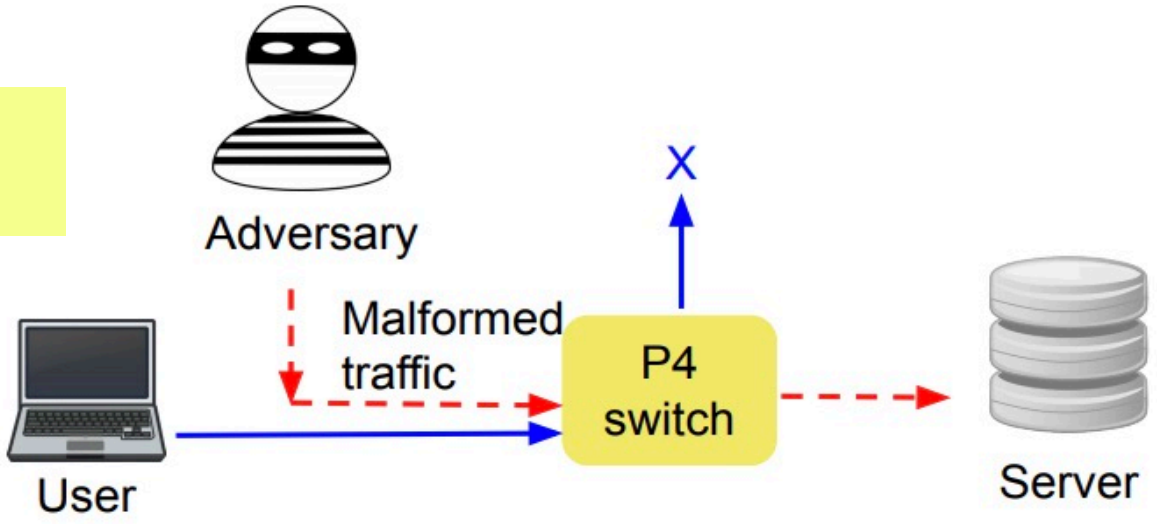
STATIC ANALYSIS

? DYNAMIC ANALYSIS / RUNTIME VERIFICATION

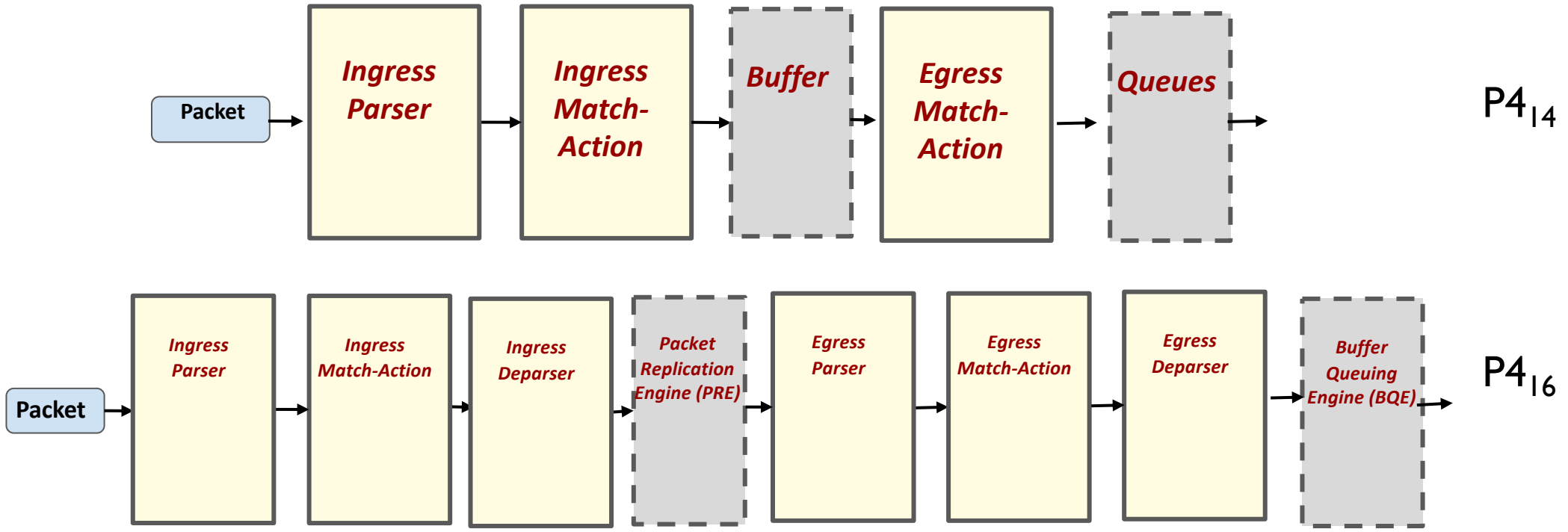
P4 Switch: Challenge- Platform-Independent Bug Example

- Packets with wrong IPv4 checksum is accepted (or generated)
- Why: Checksum calculation (or update) is not done on P4 switch

More Bug Examples in Paper



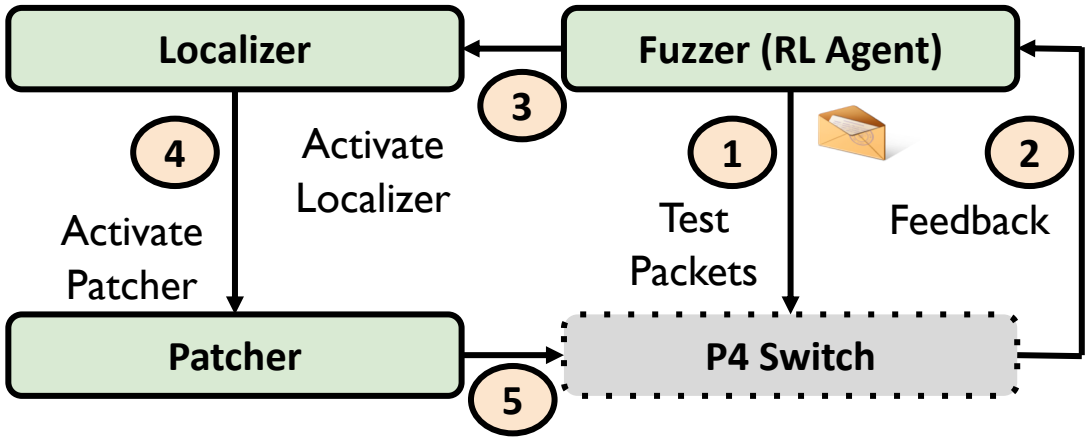
Opportunities: Evolution of P4 Program Structure



Programmable blocks double from P4$_{14}$ to P4$_{16}$ are patchable!

P6: P4 with Runtime Program Patching

- Get expected behavior from:
  1. p4q queries (test cases)
  2. Control plane configs (forwarding rules)
  3. Static analysis (reduce input space)



(1) Generate packets: RL-guided fuzzing

(2) Rewards if bug detected via query violation

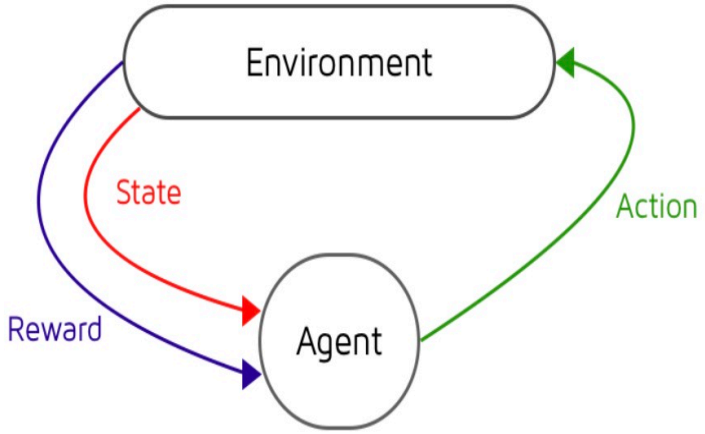
(3) Localize

(4) Patch

(5) Re-test

P6: Reinforcement Learning (RL) Agent for Guiding Fuzzing

- States: Byte-sequence in packet header
- Actions: Add/modify/delete bytes
- Rewards: $\begin{cases} 1, \text{ if packet triggered } \textbf{bug} \text{ via query violation} \\ 0, \text{ otherwise} \end{cases}$



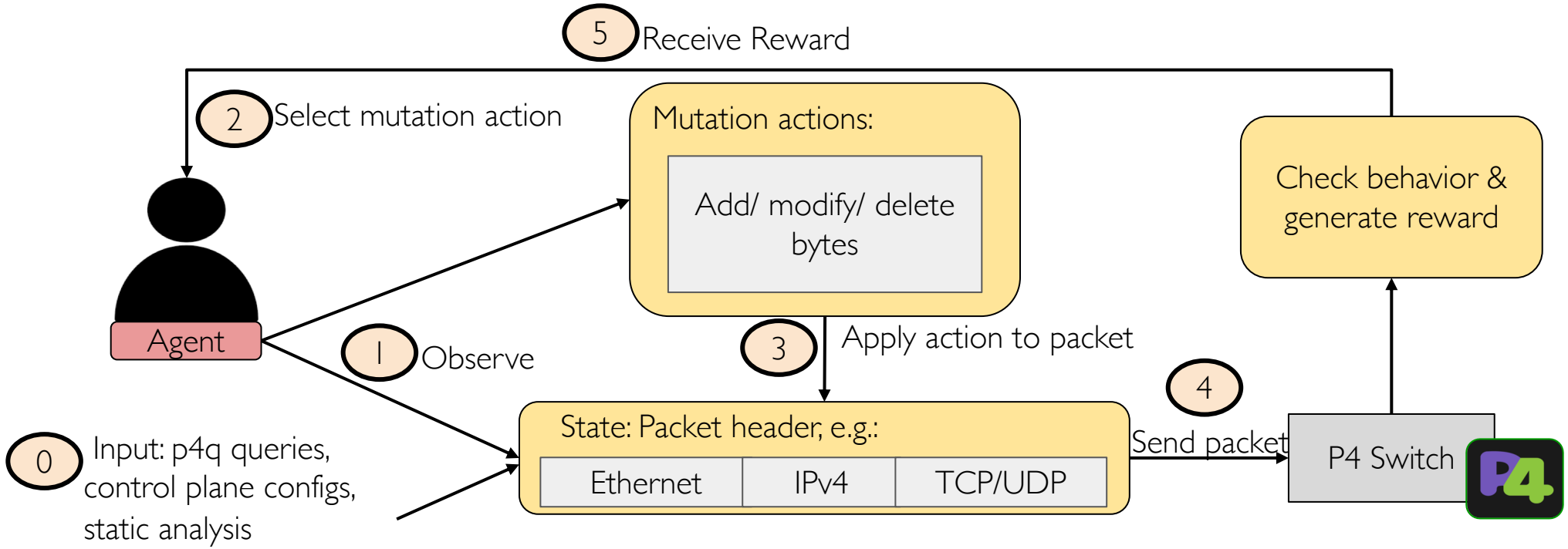
RL tries to maximise the cumulative reward (bugs)

Query Language: p4q

- Goal: Specify expected P4 switch behavior
- If- conditions to be fulfilled at ingress, then- conditions at egress, else- alternative conditions at egress
- Query violation signals bug

(ing.hdr.ipv4 & ing.hdr.ipv4.version !=4,
egr.egress_port == False,)

P6: Fuzzer (P6 Agent with RL-guided Fuzzing & Reward System)



RL tries to select appropriate action in a state to maximize cumulative reward

P6: P4Tarantula (Localize Detected Bugs in P4 Program)

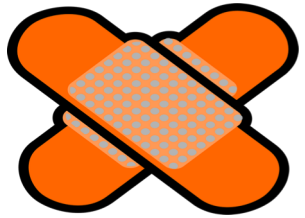
- Tailor Tarantula for P4 programs
- Code traversal based on statements executed for an input
- Assign suspiciousness score (*s*) based on faulty statements
- Rank the statements as per *s*

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }

}
```

P6: Patcher (Patch, if Needed and Available)

Pre-generated library:

- Patches code lines without patches **and** high suspiciousness score violating p4q query



- Few lines, e.g., adding checks in parser

- Pass the sanity/regression tests

P6: In Action

P4 Source Code

```
.....
state parse_ipv4 {
    packet.extract(hdr.ipv4);
            transition accept;
}
.....
```

P4 Source Code with Bugs

P4 Source Code

```
.....
state parse_ipv4 {
    packet.extract(hdr.ipv4);
            transition accept;
}
.....
```

Localized by P4Tarantula

P4 Source Code

```
.....
state parse_ipv4 {
  packet.extract(hdr.ipv4);
   verify(hdr.ipv4.version == 4, error.BadHeader);
   verify(hdr.ipv4.ihl == 5, error.BadHeader);
   verify(hdr.ipv4.len >= 20, error.BadHeader);
   verify(hdr.ipv4.ttl >= 2, error.BadHeader);
   transition accept; }
.....
apply {
   if (standard_metadata.parser_error != error.NoError) {
   mark_to_drop();
                    return;
                    }
.....
```

Patched by Patcher

P6 detects, localizes and patches bugs in a P4 program non-intrusively;
Important foray into self-driving networks

P6 performs Runtime Verification of P4 Switches

P4NoD [MSR Tech. Report' 16]
P4K [P4 Workshop' 17]
Vera [SIGCOMM' 18]
p4v [SIGCOMM' 18]
ASSERT-P4 [SOSR'18, CoNEXT'18]

STATIC ANALYSIS

P6 DYNAMIC ANALYSIS / RUNTIME VERIFICATION

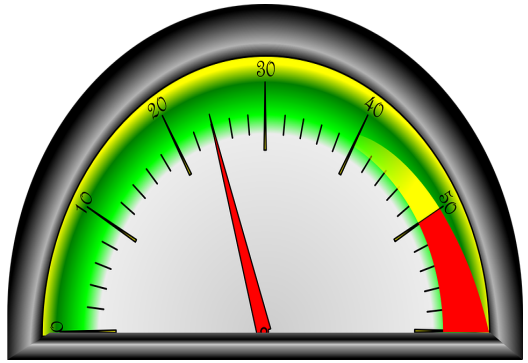
P6: Evaluation Strategy

- 8 public P4 programs from P4.org, NetPaxos across 2 platforms
- Baselines:
  1. Advanced Agent: like P6 Agent, same mutation actions (Intelligent Baseline)
  2. IPv4-Based Agent: IPv4-based packets only
  3. Naïve Agent: Exhaustive generation

P6: Metrics

Gauge P6 performance over:

- # Bugs discovered
- P6 Performance: Detection Time
- P6 vs. Baselines:
    - Learning Performance: Rewards generated



P6 Evaluation: Bugs discovered quickly with few packets only

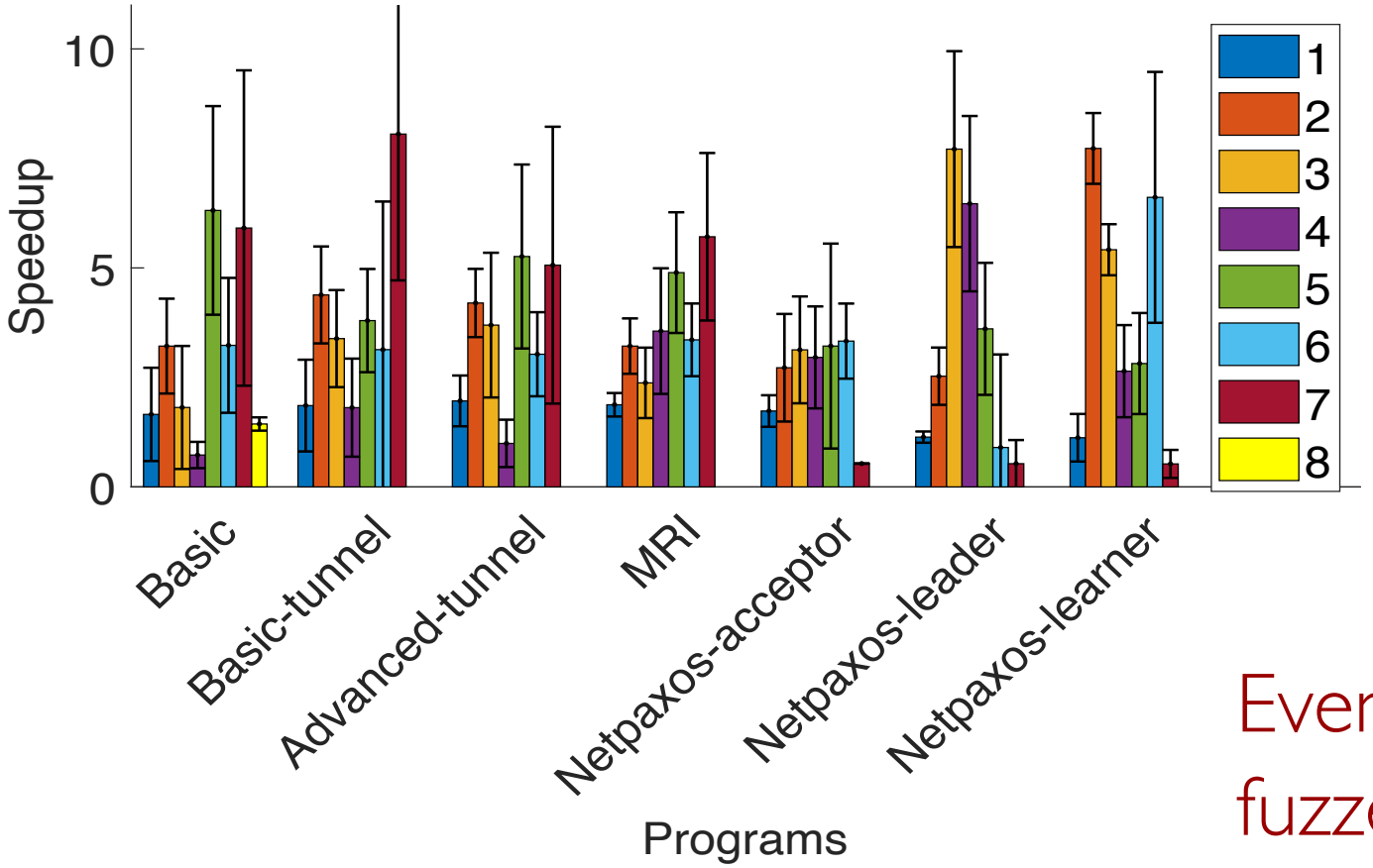
Bug IDs	Bugs
1	Accepted wrong checksum (PI)
2	Generated wrong checksum (PI)
3	Incorrect IP version (PI)
4	IP IHL value out of bounds (PI)
5	IP TotalLen value is too small (PI)
6	TTL 0 or 1 is accepted (PI)
7	TTL not decremented (PI)
8	Clone not dropped (PD)
9	Resubmitted packet not dropped (PD)
10	Multicast packet not dropped (PD)

PI – Platform-independent

PD – Platform-dependent

P6 discovered 10 bugs in 8 publicly available P4 programs;
P6 detects bugs in ~2 secs in 7 programs,
in ~10 secs in switch.p4 (8715 LOC) with 28 packets only

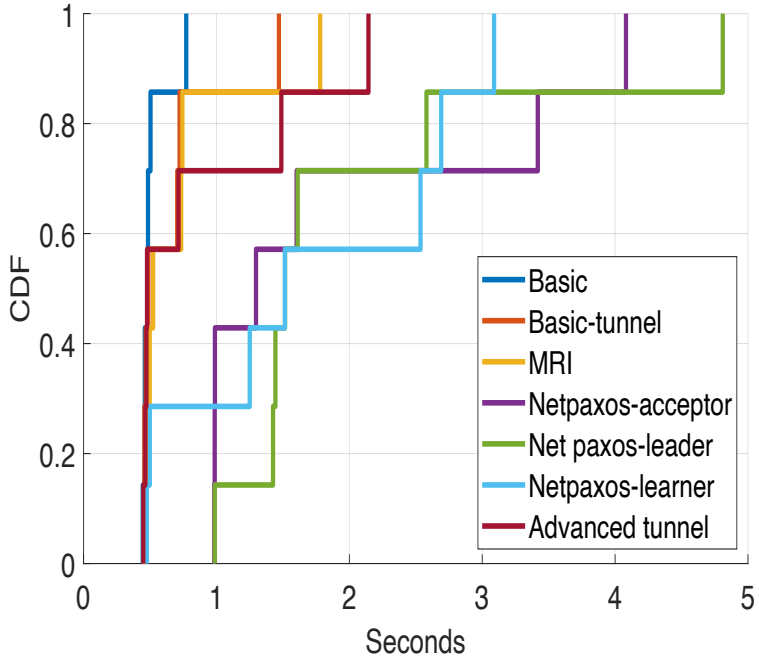
P6 Agent Detection Time Speedup vs. Advanced Agent



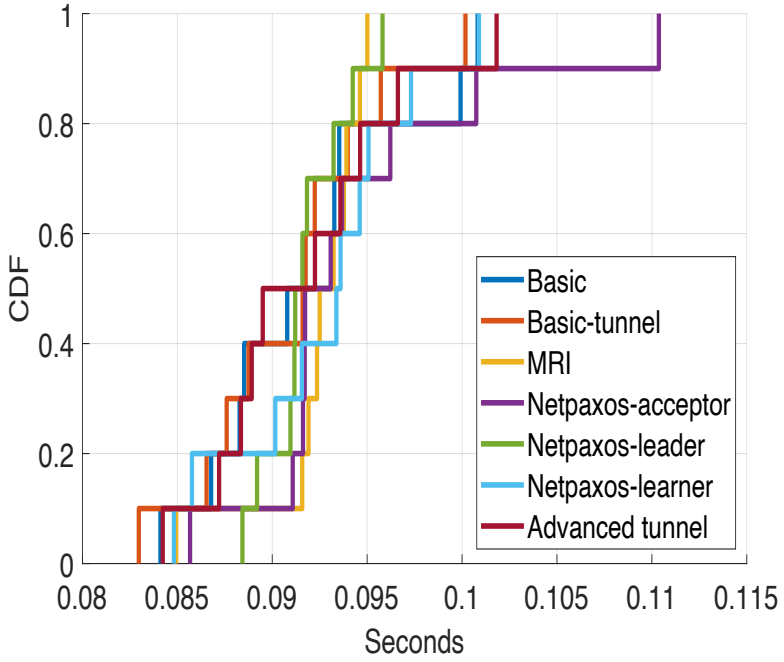
➔ P6 up to 8× faster

Even faster on IPv4-based fuzzer, naïve cannot detect any

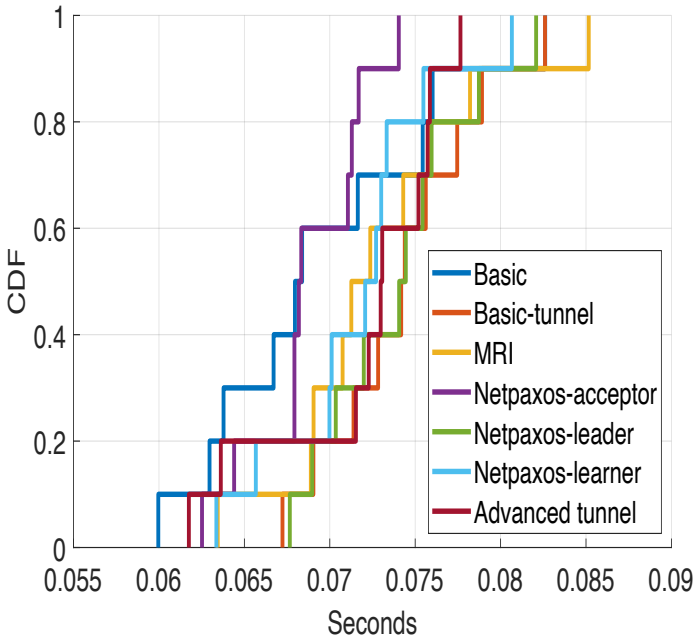
P6: Detection, Patching & Localization Times on Tofino Switch



Detection Time



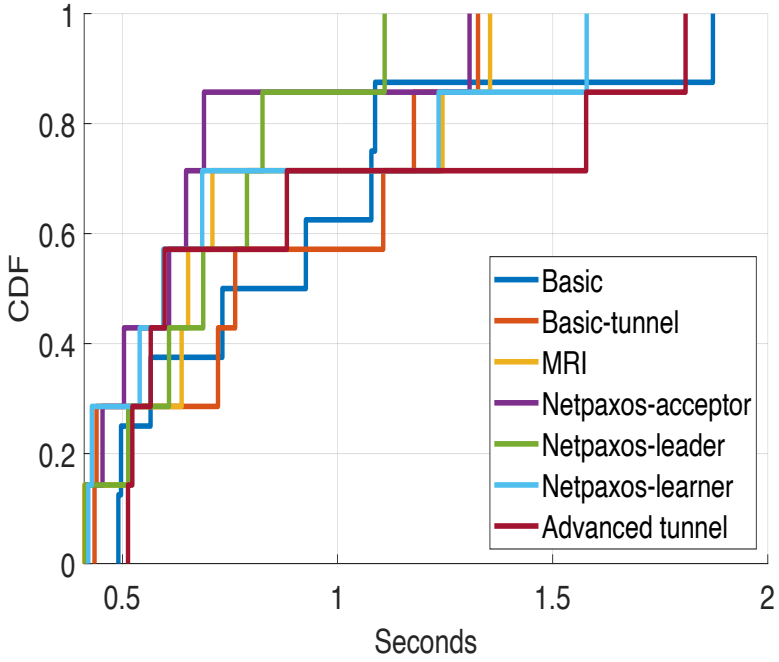
Localization Time



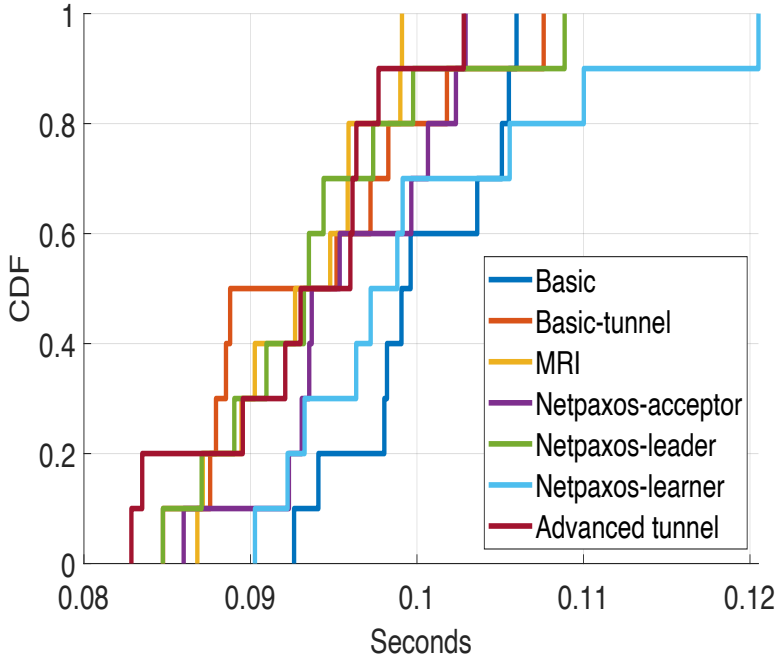
Patching Time

P6 detection time (second-scale), localization and patching time (ms-scale) performance

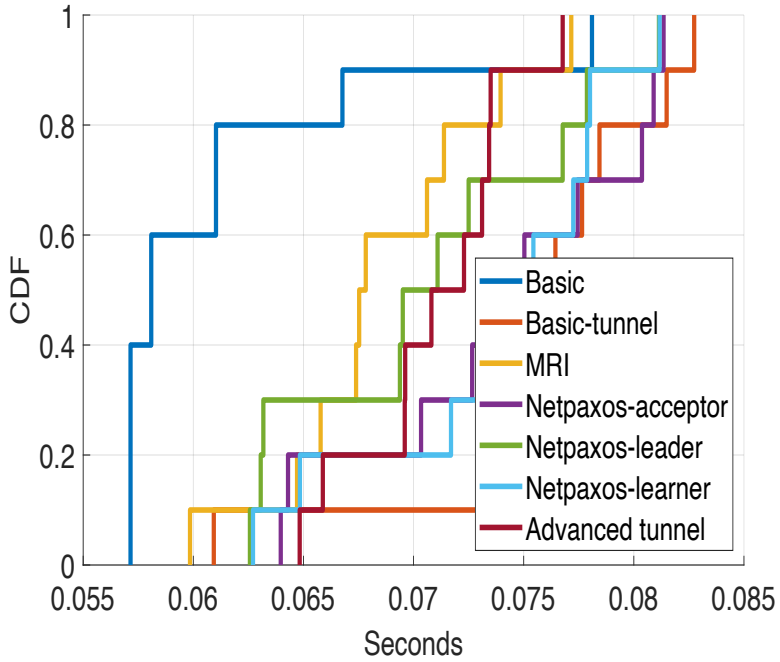
P6: Detection, Patching & Localization Times on bmv2 Switch



Detection Time



Localization Time



Patching Time

P6 detection time (second-scale), localization and patching time (ms-scale) performance

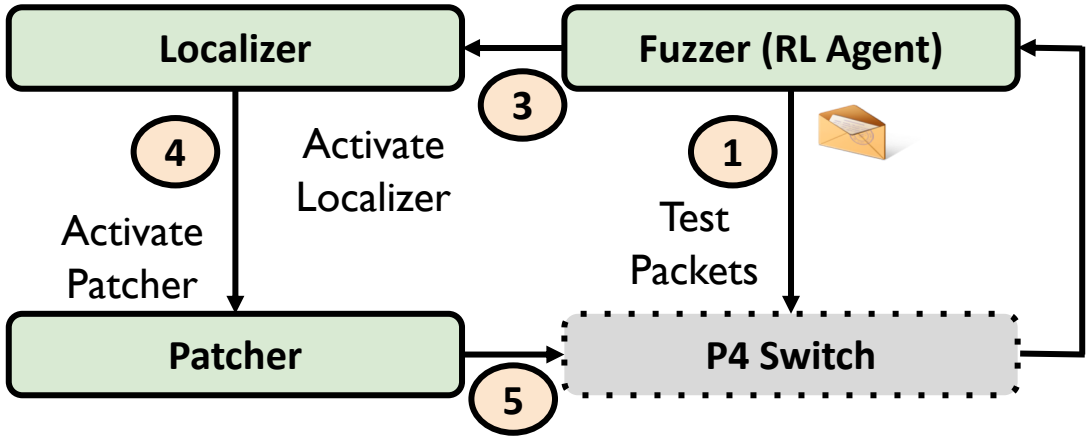
More Results in Paper

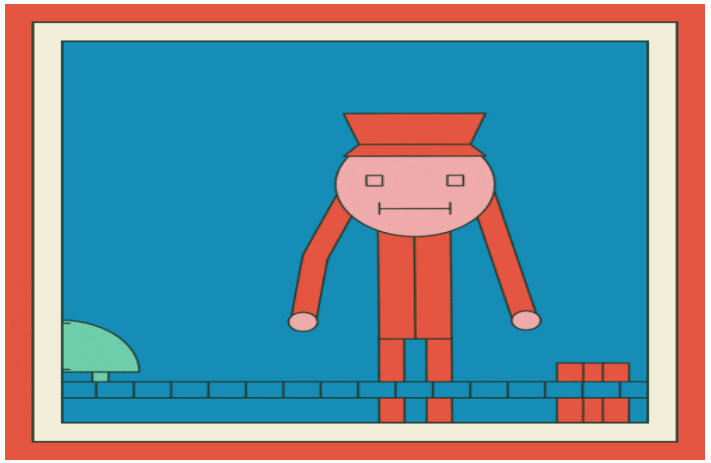
Conclusion

P6:

- enables **runtime verification** of P4 switches non-intrusively
- uses static analysis- and machine learning-guided fuzzing to **detect** multiple runtime bugs. Later, **localizes** and **patches** with minimal human effort
- **significantly outperforms baseline** bug detection approaches to detect platform-independent and -dependent bugs
- foray into **Self-driving** or autonomous networks

THANK YOU!





P6

FUTURE
IS SELF-DRIVING

Contact: Dr. Apoorv Shukla
Email: apoorv.shukla@huawei.com