Hydra: Effective Runtime Network Verification

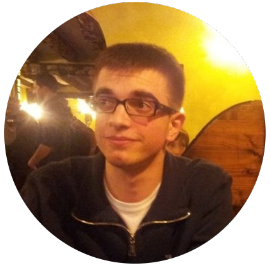
Sundararajan Renganathan

Stanford University

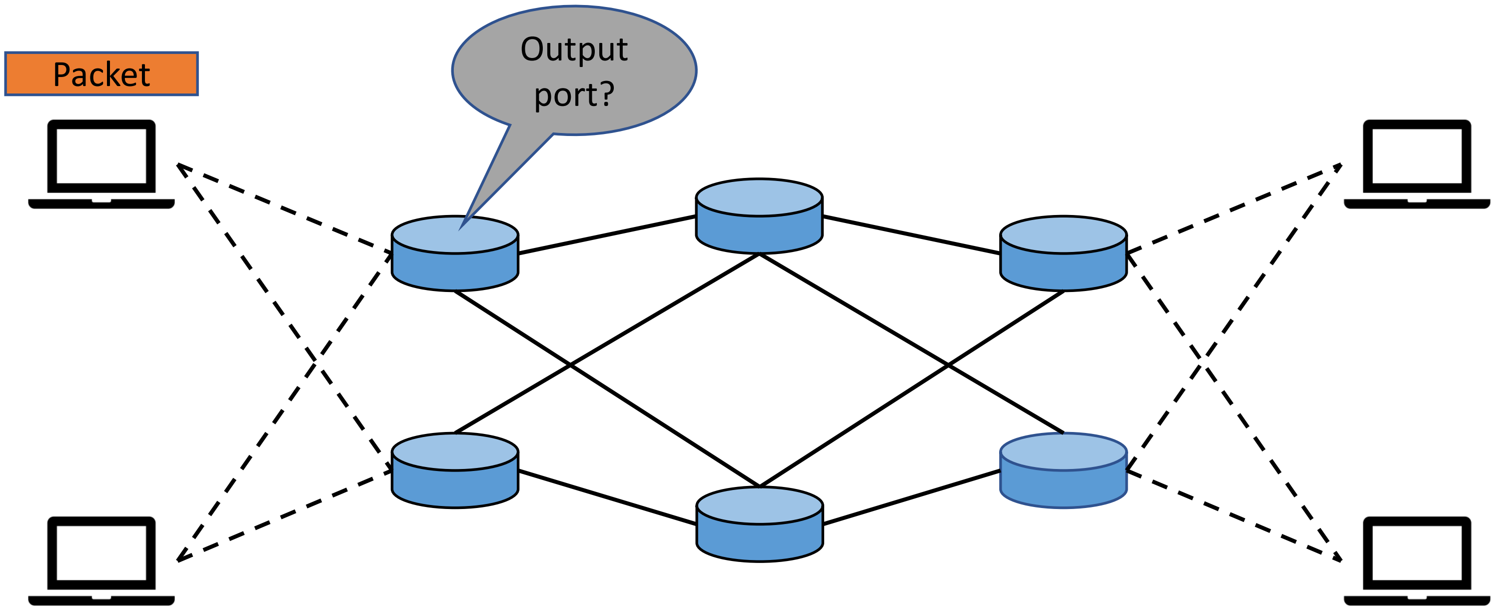
Research Team





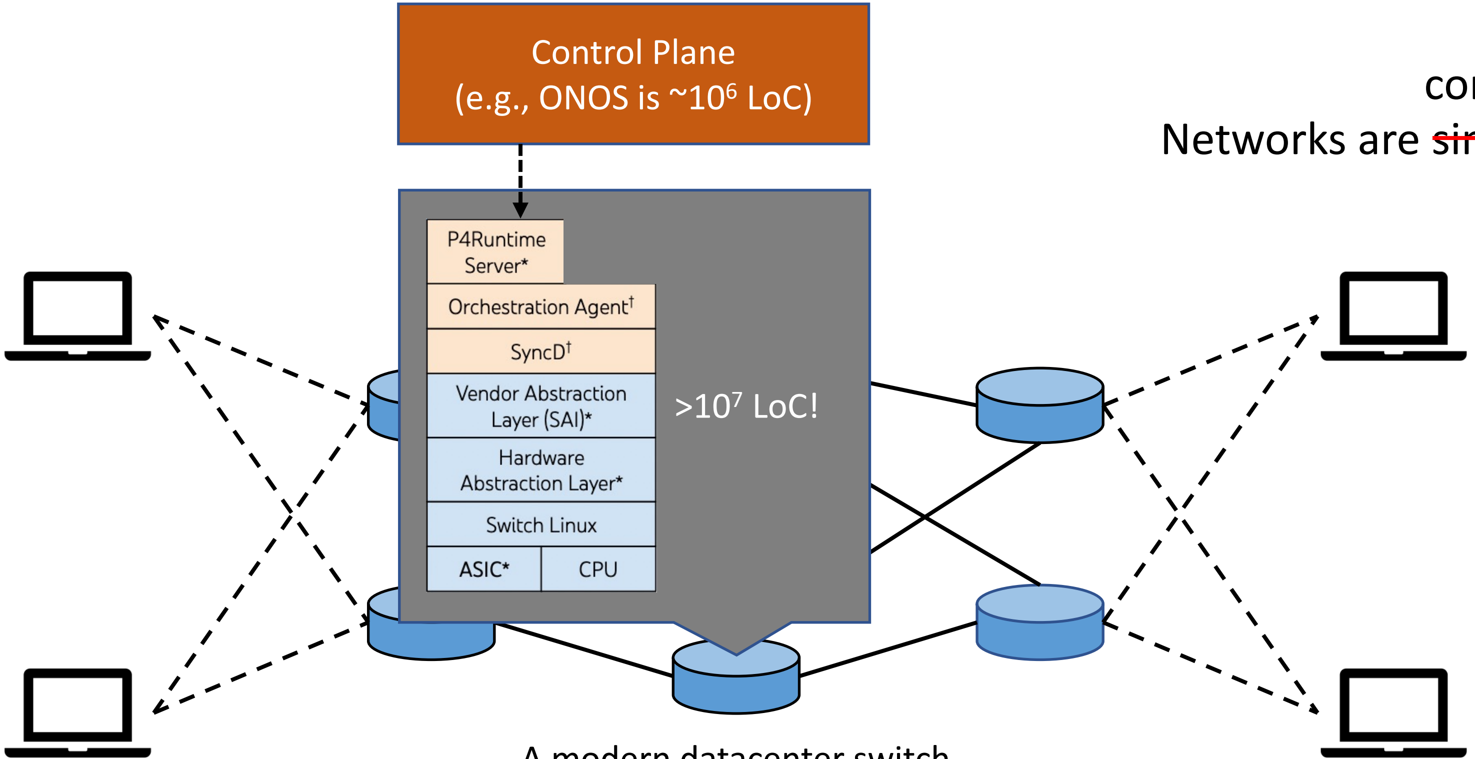



Networks are simple



However…

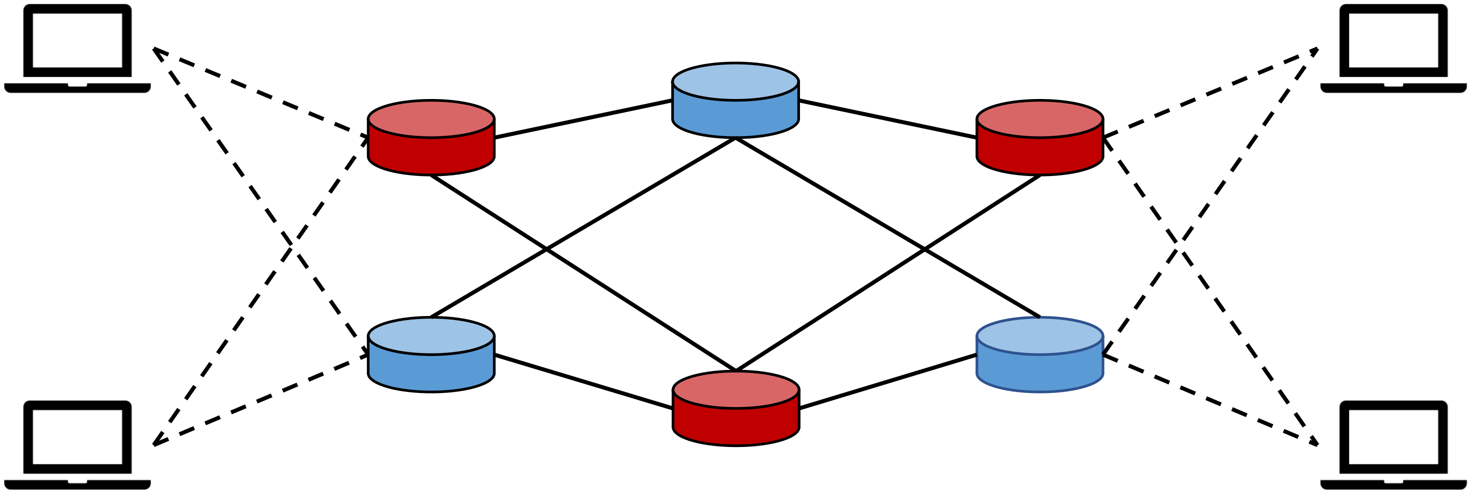
complex 😭
Networks are ~~simple 😀~~



A modern datacenter switch
(“**SwitchV**”, SIGCOMM ‘22)

How do we build trustworthy networks?

Motivating example: Color isolation



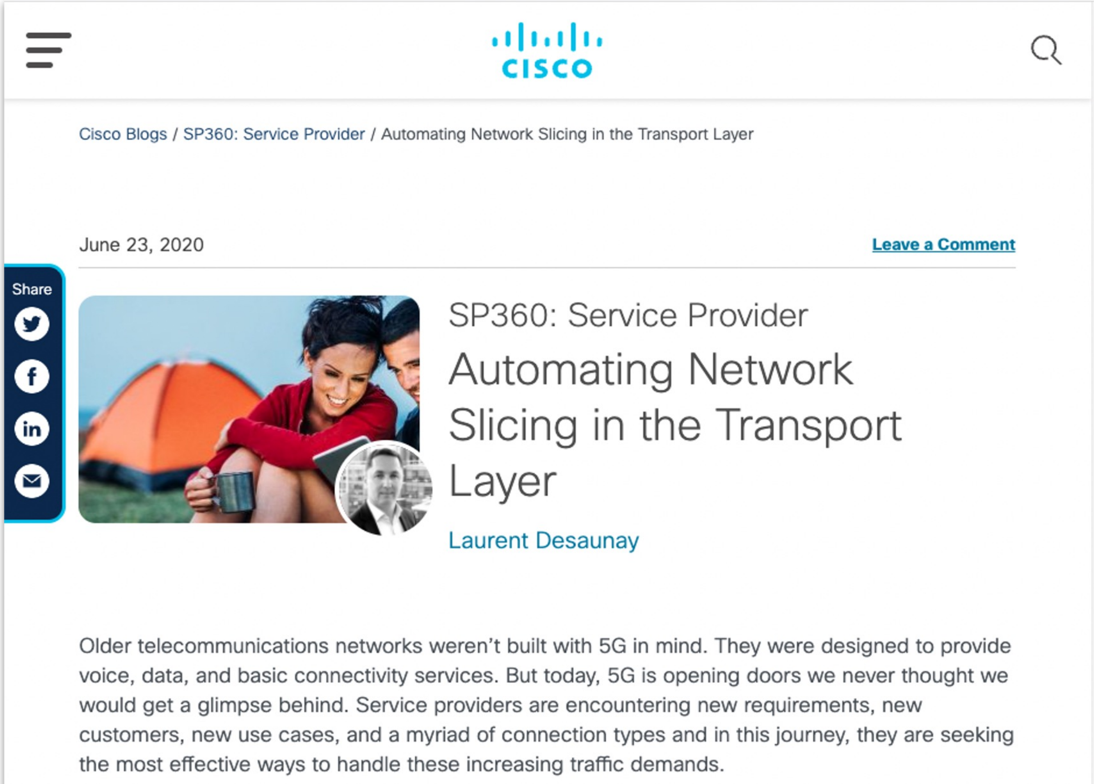
Policy: every packet should only traverse switches of a single color

"Bluebird: High- performance SDN for bare-metal cloud services", NSDI 2022.

How do we build trustworthy networks?

Axiomatic

"I trust my Cisco routers to implement the desired policy correctly."



Cisco Blogs / SP360: Service Provider / Automating Network Slicing in the Transport Layer

June 23, 2020 **Leave a Comment**

Share

SP360: Service Provider

Automating Network Slicing in the Transport Layer

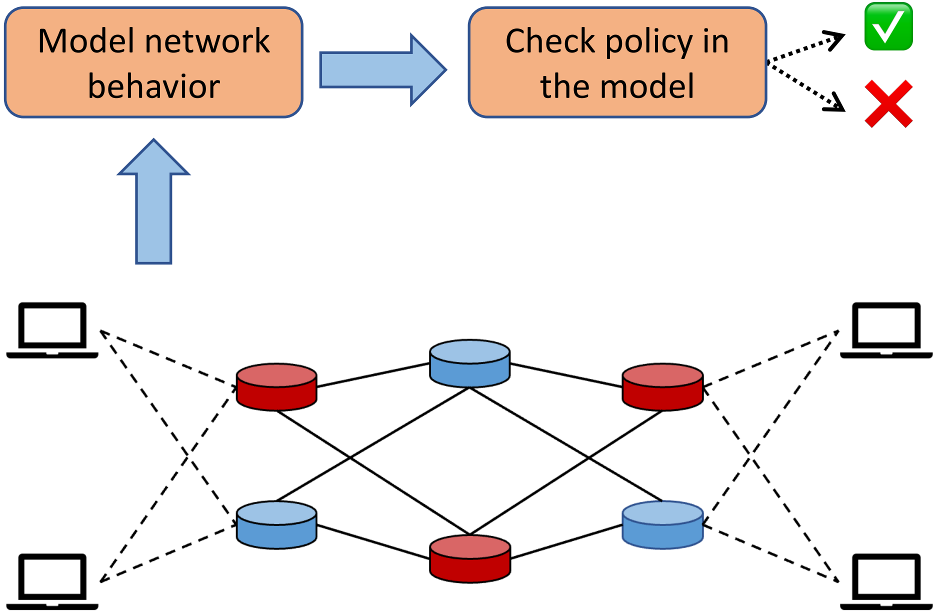
Laurent Desaunay

Older telecommunications networks weren't built with 5G in mind. They were designed to provide voice, data, and basic connectivity services. But today, 5G is opening doors we never thought we would get a glimpse behind. Service providers are encountering new requirements, new customers, new use cases, and a myriad of connection types and in this journey, they are seeking the most effective ways to handle these increasing traffic demands.

How do we build trustworthy networks?

Analytic

E.g., HSA, Veriflow, Anteater, Batfish, Minesweeper, etc.

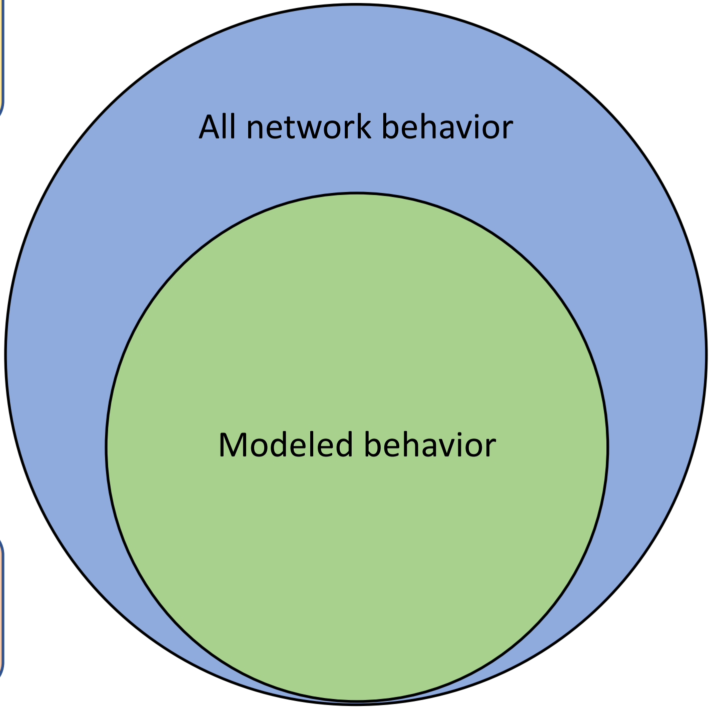


Are we done? ☺

Limitations of Analytic Trust

Limitation 1: Bugs may be in unmodeled behavior

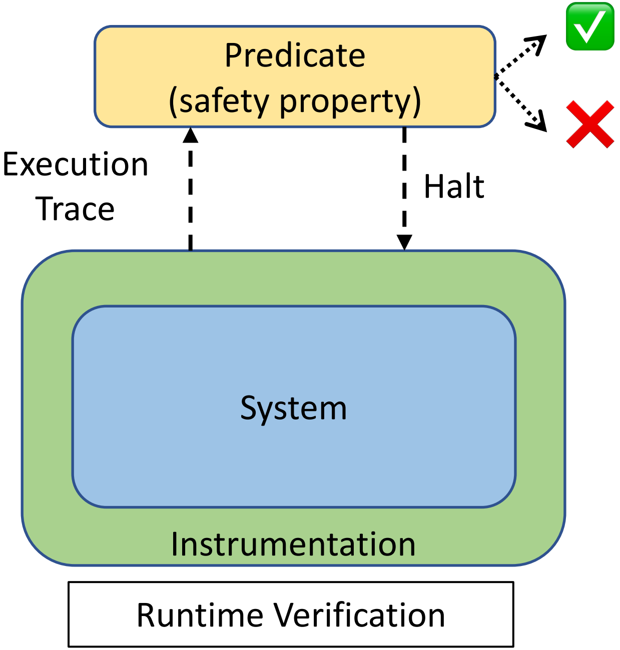
Limitation 2: Model may not reflect reality

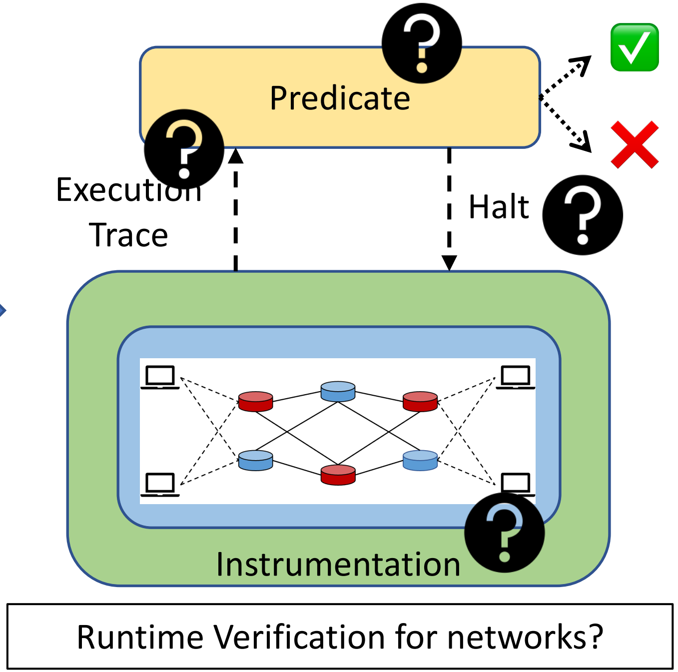


Limitation 3: Model may be tedious to construct

Limitation 4: Model checking may not scale

Can we build a network that checks itself?





- *"Eraser: A Dynamic Data Race Detector for Multithreaded Programs"*, ToCS 1997.
- *"Efficient formal verification for the Linux kernel",* ICSE 2019.

Comparison to Analytic Trust

Bugs may be in unmodeled behavior

Model may not reflect reality

→ RV verifies a running system

Model checking may not scale

Model may be tedious to construct

→ RV scales better since it verifies one or a few execution traces

How do we build trustworthy networks?

Axiomatic

"I trust my Cisco routers to implement the desired policy correctly."

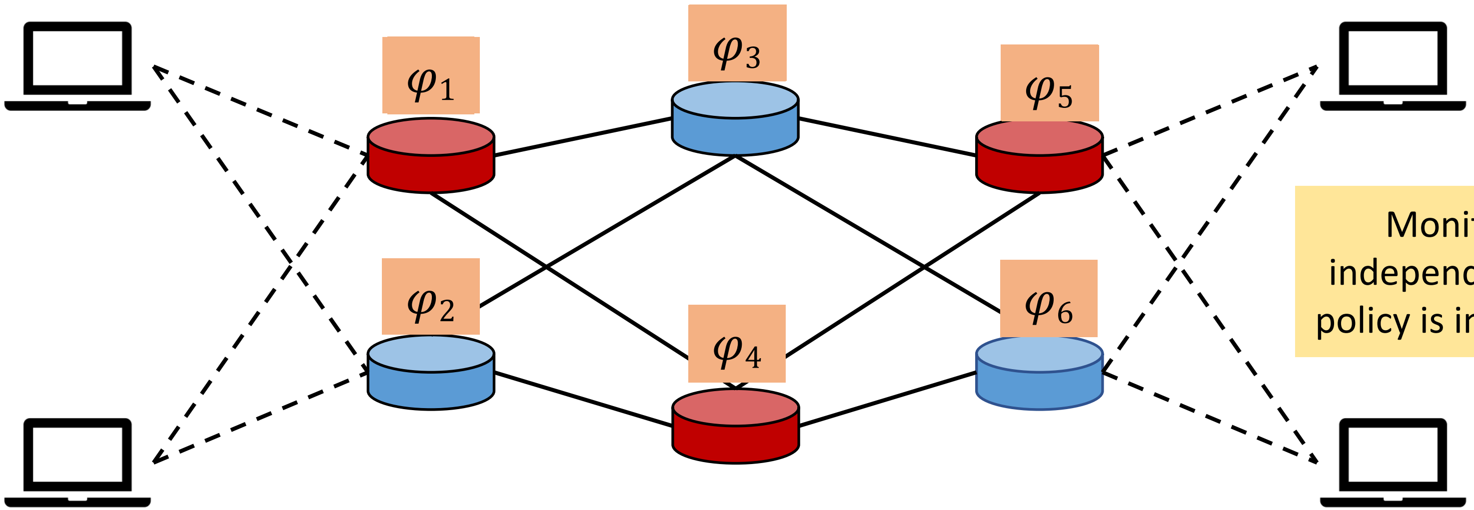
Analytic

"I built a model of my network. The desired policy holds in the model."

Synthetic

"I check that every packet conforms to the desired policy at runtime."

Runtime Verification for Color Isolation

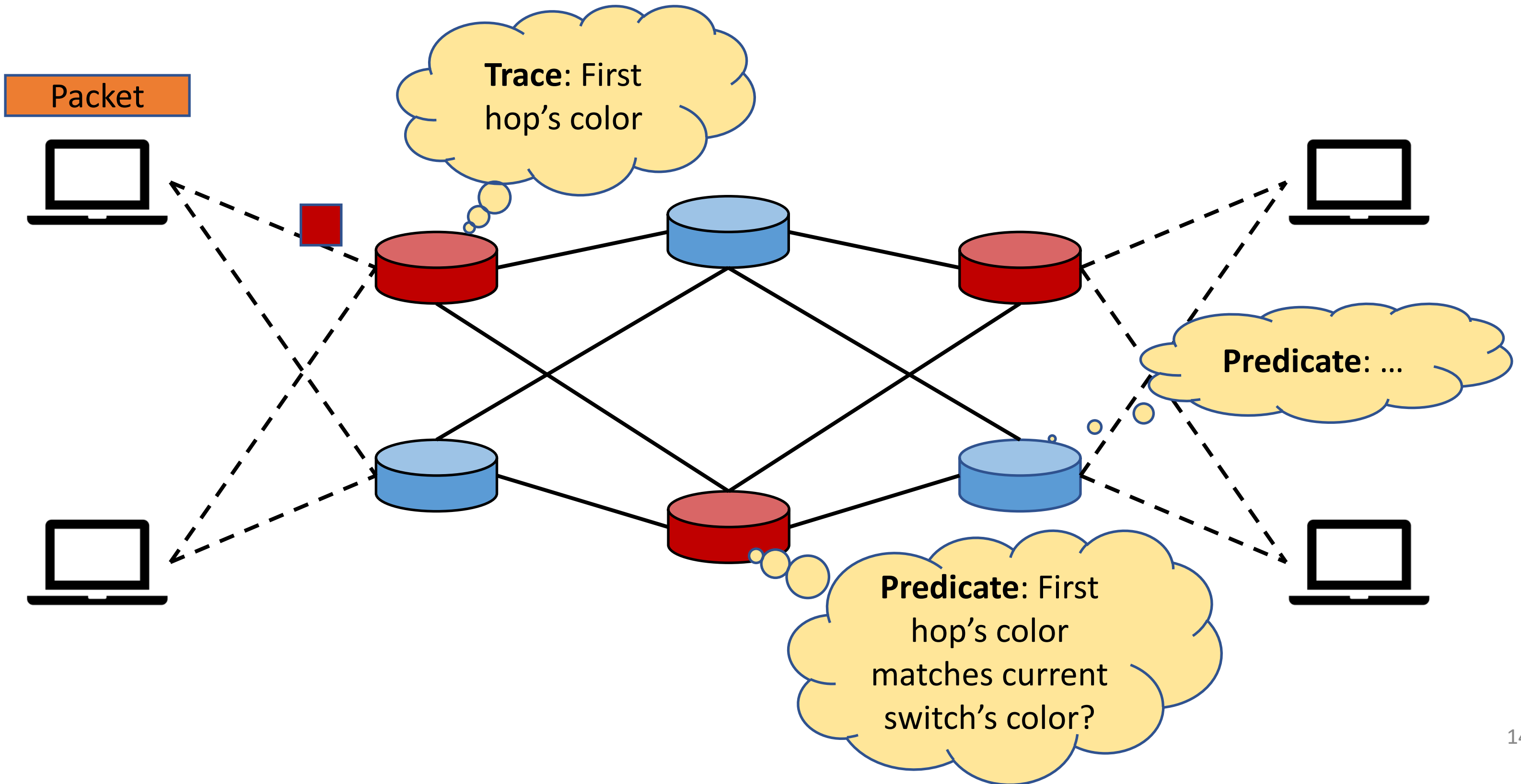


Monitoring is independent of how policy is implemented

Policy: every packet should only traverse switches of a single color

Goal: instrument the network to verify policy compliance at runtime

Runtime Verification for Color Isolation



Hints for Runtime Network Verification

Collect network-wide execution traces on packets

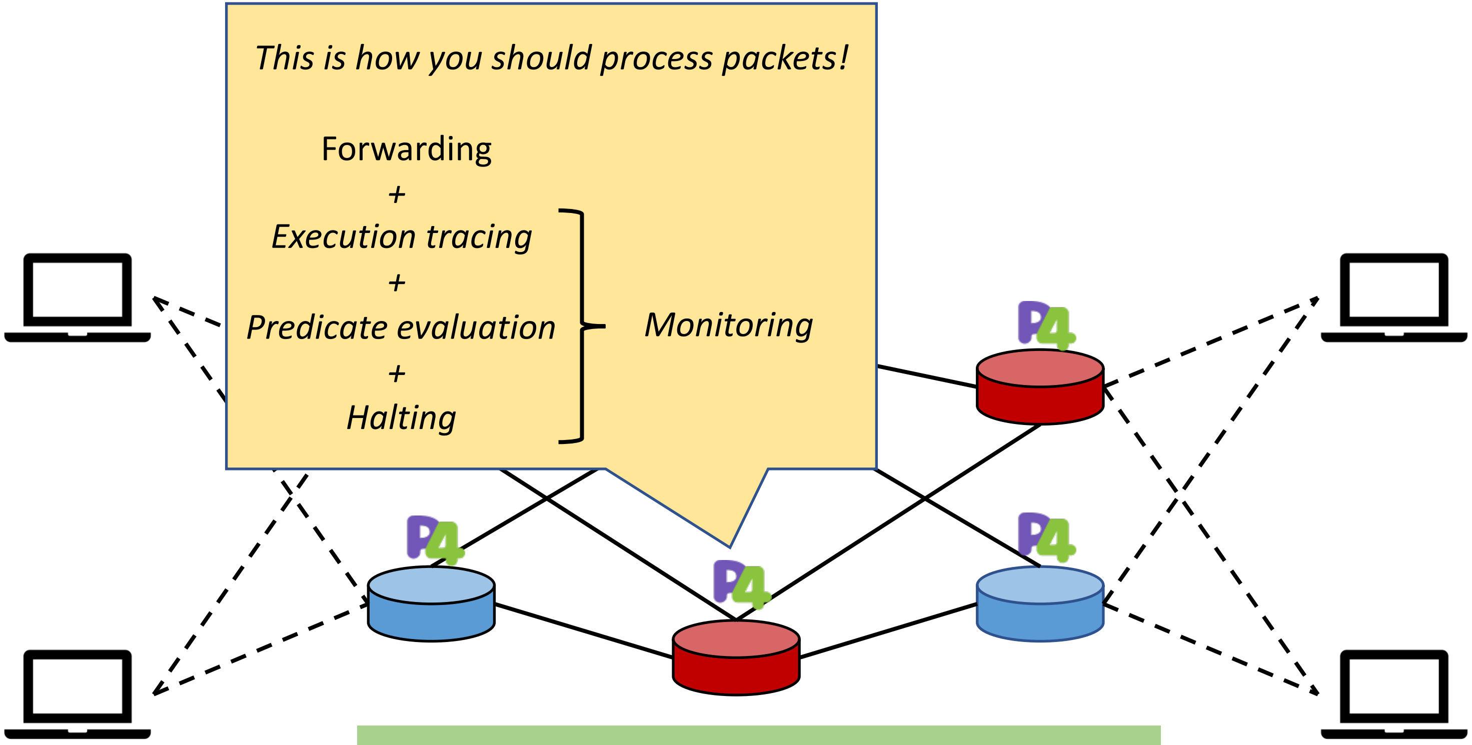
Evaluate predicates per packet (fine-grained events)

If a check fails, then stop packet from making forward progress

Monitoring and forwarding code/state should be independent

How do we realize this design? ☺

Runtime Network Verification using P4



Forwarding + Monitoring = Single P4 Program

Runtime Network Verification using P4

Collect network-wide execution traces on packets	P4 only presents a single-switch abstraction ❓
Evaluate predicates on traces on per-packet basis	✅
If a check fails, then stop packet from making forward progress	✅
Monitoring and forwarding code/state should be independent	Hard to enforce independence when P4 code is used for both ❓

Runtime Network Verification using *Hydra*

We designed *Indus, a* new domain-specific property language based on *network-wide* traces and *predicates*

```
/* Variable declarations */
tele bit<8> first_hop_color;
tele bit<8>[4] hop_colors;
control bit<8> switch_color;
/* Code blocks */
init { /* Executes at first hop */
  first_hop_color = switch_color;
}
telemetry { /* Executes at every hop */
  hop_colors.append(switch_color);
}
checker { /* Executes at the last hop */
  for (hop_color in hop_colors) {
    if (hop_color != first_hop_color) { reject; }
  }
}
```

State

- Packet variables
  - first_hop_color
  - hop_colors
- Static variables
  - switch_color

Color Isolation in Indus

Runtime Network Verification using *Hydra*

We designed *Indus, a* new domain-specific property language based on *network-wide* traces and *predicates*

```
/* Variable declarations */
tele bit<8> first_hop_color;
tele bit<8>[4] hop_colors;
control bit<8> switch_color;
/* Code blocks */
init { /* Executes at first hop */
  first_hop_color = switch_color;
}
telemetry { /* Executes at every hop */
  hop_colors.append(switch_color);
}
checker { /* Executes at the last hop */
  for (hop_color in hop_colors) {
    if (hop_color != first_hop_color) { reject; }
  }
}
```

Semantics

- Initialization happens at first hop
- Telemetry executes at every hop and updates telemetry variables
- Checker executes at last hop and implements the predicate; packets that fail checks are dropped

Color Isolation in Indus

Verifying load balance in Indus

- Invariant to verify
  - Load is balanced across two output ports at every switch in a packet's path
- Indus provides *sensor* variables to aggregate state across packets
  - *Semantics*: sensor variables reside on switches
- Telemetry
  - Carry values of sensors in telemetry variables
- Predicate
  - Check that sensor values for each pair of output ports is approximately equal
  - Send a report to the control plane if they are not

```
/* Sensor variables, located on every switch */
sensor bit<32> left_load = 0;
sensor bit<32> right_load = 0;
/* Control variables, located on every switch */
control left_port;
control right_port;
control thresh;
control dict<bit<8>,bool> is_uplink;
/* Telemetry variables, carried on the packet */
tele bit<32>[15] left_loads;
tele bit<32>[15] right_loads;
init {}
telemetry {
  if (is_uplink[%eg_port]) {
    if (%eg_port == left_port) {
      left_load += %packet_length;
    }
    elsif (eg_port == right_port) {
      right_load += %packet_length;
    }
  }
  left_loads.append(left_load);
  right_loads.append(right_load);
}
checker {
  for (left_load, right_load in left_loads,
       right_loads) {
    if (abs(left_load - right_load) > thresh) {
      report;
    }
  }
}
```

Research Questions

Q. Is the language *powerful* enough to express rich network-wide properties?

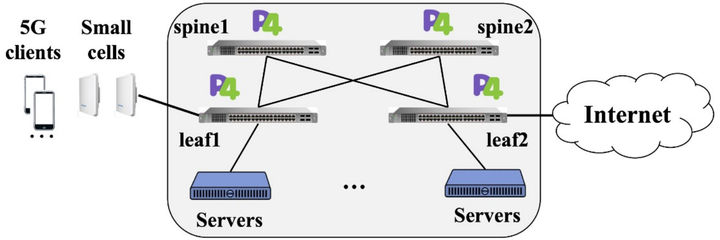
We *prove* that the language can encode any network-wide property written in Linear Temporal Logic, heavily used in RV

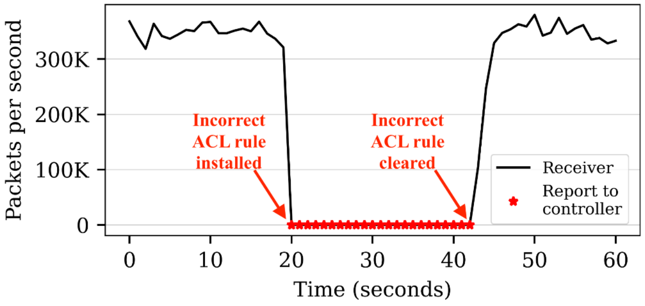
Q. How do we *efficiently* enforce properties thus specified on modern hardware?

We built a *compiler* that compiles and merges an *Indus* program with the forwarding code into a single binary for P4 switches

Hydra in action

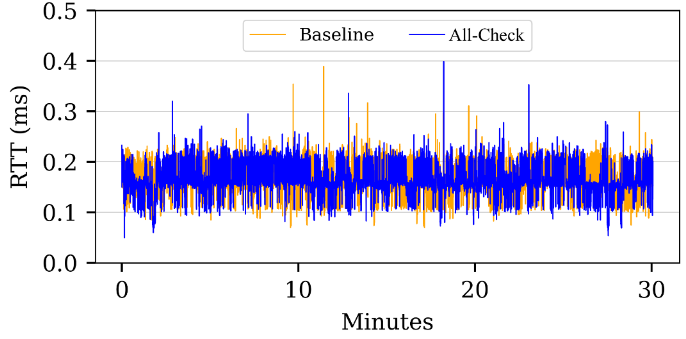
- Developed properties that capture key invariants for Aether, an open-source cellular platform
  - “Aether: Private 4G/5G Connected Edge Platform for Enterprises”
- Properties
  - Loop avoidance, leaf-spine routing, egress port validity, VLAN isolation, ECMP correctness
- Deployed said checkers on the Aether “dogfooding” testbed at Princeton
- Injected faults (buggy forwarding rules) that violate the “Egress Port Validity” property
- Errant packets are immediately detected and reported to controller

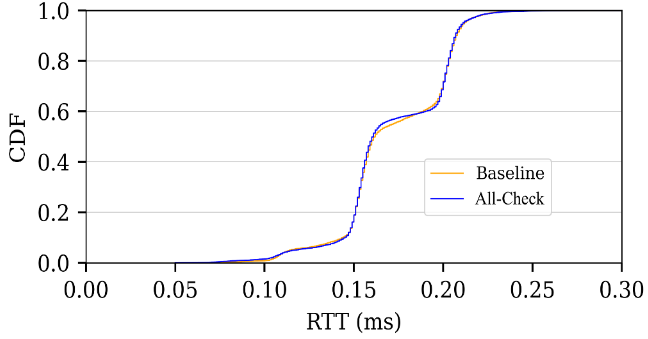




Hydra Overheads: Latency

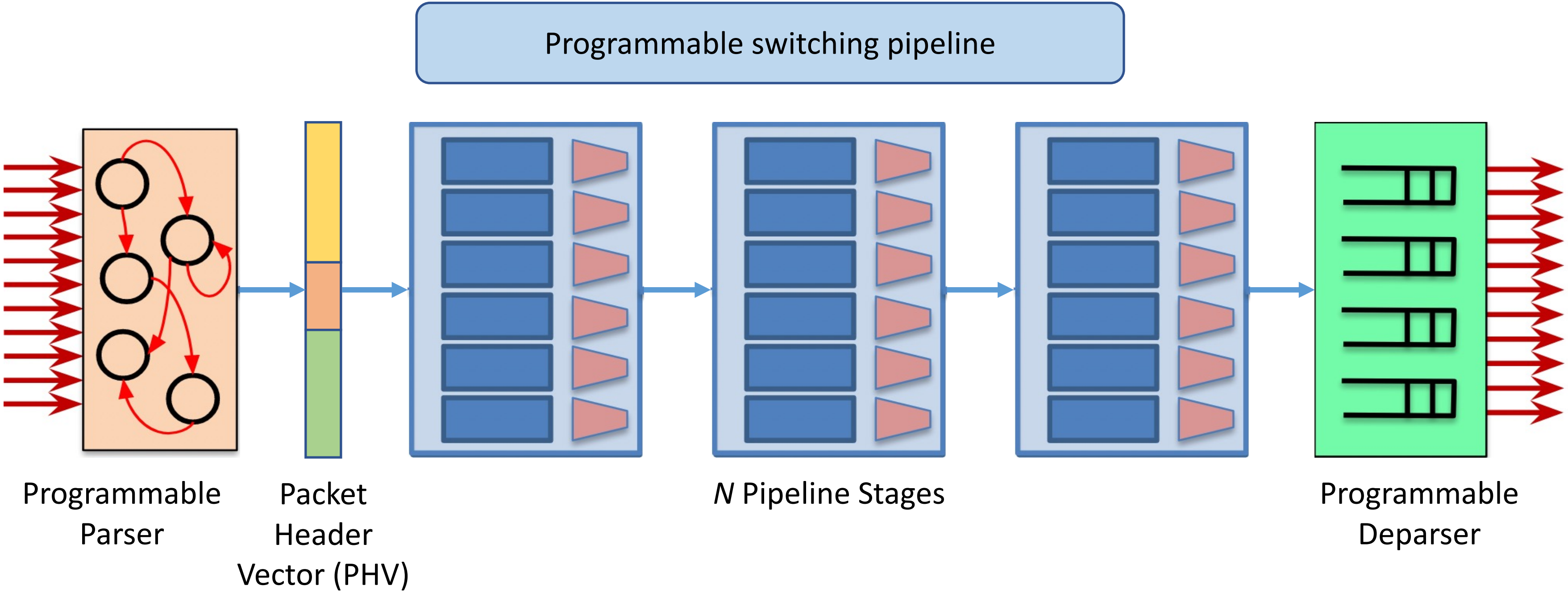
Comparison of packet RTT with and without Hydra





Overhead is negligible ☺

Hydra Overheads: Tofino Resource Utilization



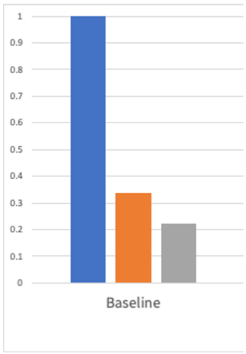
1. Pipeline Stage Usage

2. PHV Usage

3. Parser TCAM Usage

Hydra Overheads: Tofino Resource Utilization

Aether's P4 Program



Finding 1: Number of stages used is the same, despite more usage on existing stages

Finding 2: PHV and Parser TCAM overheads are low

Overheads seem manageable ☺

Future Work

- Incremental Deployment
  - Fixed function switches provide telemetry, check predicates at edge in NICs/eBPF?
- “Root-cause” packets that fail checks instead of simply halting progress
- Closed-loop control
  - Can we actuate the network back to a known good state?
- Probabilistic Verification
- Verifying higher-level service abstractions composed of per-packet checks

Summary

- Networks provide telemetry “for free”
- Hydra: Runtime Monitoring for Networks
  - An underexplored approach to verification!
- Contributions
  - Domain-specific property language *Indus*
  - A compiler to produce P4 code
  - TTE seems to be a killer application! ☺