Hydra: Effective Runtime Network Verification

Sundararajan Renganathan
Stanford University
Research Team
Networks are simple
However…

Control Plane (e.g., ONOS is \(\sim 10^6 \) LoC)

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

How do we build trustworthy networks?

Axiomatic

“I trust my Cisco routers to implement the desired policy correctly.”
How do we build trustworthy networks?

Analytic

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

Model network behavior → Check policy in the model

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
- Model checking may not scale
- Model may be tedious to construct

RV verifies a running system

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

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

Goal: instrument the network to verify policy compliance at runtime

Monitoring is independent of how policy is implemented
Runtime Verification for Color Isolation

Packet Trace: First hop’s color

Predicate: First hop’s color matches current switch’s color?
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

This is how you should process packets!

Forwarding
+ Execution tracing
+ Predicate evaluation
+ Halting

Forwarding + Monitoring = Single P4 Program
Runtime Network Verification using P4

Collect network-wide execution traces on packets

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

P4 only presents a single-switch abstraction

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*.

```c
/* Variable declarations */
teletype 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;
}
telemtery {
   /* 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*

```plaintext
/* 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
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

Programmable switching pipeline

Programmable Parser
Packet Header Vector (PHV)

1. Pipeline Stage Usage
2. PHV Usage
3. Parser TCAM Usage

N Pipeline Stages

Programmable Deparser
Hydra Overheads: Tofino Resource Utilization

**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! 😊