A PROGRAM LOGIC FOR AUTOMATED P4 VERIFICATION

Nate Foster
Barefoot Networks/
Cornell University

Cole Schlesinger
Barefoot Networks

Robert Soulé
Barefoot Networks/
University della Svizzera italiana

Han Wang
Barefoot Networks
<table>
<thead>
<tr>
<th>General P4 safety properties</th>
<th>Program-specific properties</th>
</tr>
</thead>
</table>

General P4 safety properties

- Don’t read uninitialized metadata/invalid headers

Program-specific properties
General P4 safety properties

- Don’t read uninitialized metadata/invalid headers
- Avoid unexpected arithmetic overflow/truncation

Program-specific properties
General P4 safety properties

- Don’t read uninitialized metadata/invalid headers
- Avoid unexpected arithmetic overflow/truncation
- Catch all parser exceptions

Program-specific properties
General P4 safety properties
- Don’t read uninitialized metadata/invalid headers
- Avoid unexpected arithmetic overflow/truncation
- Catch all parser exceptions
- Always explicitly handle every packet

Program-specific properties
General P4 safety properties
- Don’t read uninitialized metadata/invalid headers
- Avoid unexpected arithmetic overflow/truncation
- Catch all parser exceptions
- Always explicitly handle every packet

Program-specific properties
- The ACL blocks SSH traffic
General P4 safety properties
- Don’t read uninitialized metadata/invalid headers
- Avoid unexpected arithmetic overflow/truncation
- Catch all parser exceptions
- Always explicitly handle every packet

Program-specific properties
- The ACL blocks SSH traffic
- If an IP packet is not dropped, the TTL is decremented by one
General P4 safety properties
- Don’t read uninitialized metadata/invalid headers
- Avoid unexpected arithmetic overflow/truncation
- Catch all parser exceptions
- Always explicitly handle every packet

Program-specific properties
- The ACL blocks SSH traffic
- If an IP packet is not dropped, the TTL is decremented by one
- NAT and multicast are never applied to the same packet
P4 MODEL

Control Plane

P4 pipeline

ACL

NAT

Forward
The specific behavior of a P4 program depends on the control plane.
Packet-processing pipeline

```
action forward(p) { ... }
table T {
  reads {
    tcp.dstPort;
    eth.src;}
  actions {
    drop;
    forward; }
}
```
Packet-processing pipeline

Desired property:

*If tcp.dstPort is 22, packet has been dropped.*

```c
action forward(p) { ... }
table T {
  reads {
    tcp.dstPort;
    eth.src;
  }
  actions {
    drop;
    forward;
  }
}
```
Desired property:
If tcp.dstPort is 22, packet has been dropped.

Under what condition is the desired property guaranteed to be true?

If the packet has already been dropped.

action forward(p) {
    table T {
        reads {
            tcp.dstPort;
            eth.src;
        }
        actions {
            drop;
            forward;
        }
    }
}

Packet-processing pipeline
Under what condition is the desired property guaranteed to be true?

If the packet has already been dropped.

Desired property:
If tcp.dstPort is 22, packet has been dropped.

```
action forward(p) { ... }

table T {
  reads {
    tcp.dstPort;
    eth.src;
  }
  actions {
    @pragma(true)
    drop;
    @pragma(tcp.dstPort != 22)
    forward;
  }
}```
Packet-processing pipeline

```
action forward(p) { ... }
table T {
  reads {
    tcp.dstPort;
    eth.src;
  }
  actions {
    @pragma(true)
    drop;
    @pragma(tcp.dstPort != 22)
    forward;
  }
}
```

Under what condition is the desired property guaranteed to be true?

*If the packet has already been dropped.*

*Always.*

Desired property:

*If tcp.dstPort is 22, packet has been dropped.*
Under what condition is the desired property guaranteed to be true?

Desired property:

No packet is sent to the control port (say, 512).

```p4
action forward(p) { ... }

table T {
  reads {
    tcp.dstPort;
    eth.src; }
  actions {
    @pragma(true)
    drop;
    @pragma(tcp.dstPort != 22)
    forward; } }
```
Under what condition is the desired property guaranteed to be true?

*Always.*

```
action forward(p) { ... }
table T {    
  reads {    
    tcp.dstPort;    
    ... }    
  actions {    
    @pragma(true)    
    drop;    
    @pragma(tcp.dstPort != 22)    
    (0 <= p < 48)    
    forward; } } }
```

Desired property:

*No packet is sent to the control port (say, 512).*
OUR APPROACH...

Translating P4 to IMP + tables

Defining a program logic for P4-IMP

Annotating P4-IMP to check specific properties

Reducing to Z3
OUR APPROACH...

Defining a program logic for P4-IMP

Translating P4 to IMP + tables

Annotating P4-IMP to check specific properties

Reducing to Z3
IMP + HOARE LOGIC

Assignments, if statements, and table applications
Axioms describing what is true before and after a command executes.

\[ \vdash \{ P \} \ c \ \{ Q \} \]

*If P holds and c executes, then Q holds.*

**IF**
\[ \vdash \{ P \land b \} \ c_1 \ \{ Q \} \]

**AND**
\[ \vdash \{ P \land \neg b \} \ c_2 \ \{ Q \} \]

**THEN**
\[ \vdash \{ P \} \ \text{if } b \ \text{then } c_1 \ \text{else } c_2 \ \{ Q \} \]
Hoare logic + table constraints

Given a table T:

```
table T {
  reads { ... }
  actions {
    @pragma(true)
    drop;
    @pragma(tcp.dstPort != 22)
    (0 <= p < 48)
    forward; }
}
```

Then P (plus the table constraints) is sufficient to establish that Q holds after applying T, written

\[ \vdash \{ P \land \text{true} \land \text{true} \} \quad \text{drop} \quad \{ Q \} \]

\[ \vdash \left\{ P \land \text{tcp.dstPort} \neq 22 \land (0 \leq p < 48) \right\} \quad \text{forward}(p) \quad \{ Q \} \]

\[ \models R1 \lor \ldots \lor Rn \]

\[ \vdash \{ P \} \quad T() \quad \{ Q \} \]
Given a table $T$:

```
    table T {
        reads { … }
        actions {
            @pragma(R1)
            (S1)
            a1;
            @pragma(R2)
            (S2)
            a2; }
    }
```

If for all actions $a_i$,

\[
\vdash \{ P \land R_i \land S_i(x_i) \} a_i(x_i) \{ Q \}
\]

And

\[
\models R_1 \lor \ldots \lor R_n
\]

Then $P$ (plus the table constraints) is sufficient to establish that $Q$ holds after applying $T$, written

\[
\vdash \{ P \} T() \{ Q \}
\]
Given a command $c$ and a post-condition $Q$, 
$\text{wp}(c, Q) = P$ such that $\vdash \{ P \} \ c \ \{ Q \}$
Given a command $c$ and a post-condition $Q$, 

$$wp(c, Q) = P \quad \text{such that} \quad \vdash \{ P \} \ c \ \{ Q \}$$

**P4 PROGRAM LOGIC**

Hoare logic + table constraints

**P4 PROGRAM**

$$wp(IMP \ PROGRAM, Q)$$
Given a command $c$ and a post-condition $Q$, 

$$wp(c, Q) = P \text{ such that } \vdash \{ P \} \ c \ \{ Q \}$$

eg. if tcp.dstPort is 22, packet has been dropped.
Given a command $c$ and a post-condition $Q$, 

$$ \text{wp}(c, Q) = P \text{ such that } |- \{ P \} c \{ Q \} $$
Given a command \( c \) and a post-condition \( Q \),
\[
wp(c, Q) = P \quad \text{such that} \quad \vdash \{ P \} \ c \ \{ Q \}
\]

eg. if tcp.dstPort is 22, packet has been dropped.
NEXT STEPS

1. Automatically annotate P4-IMP programs to check safety properties.
2. Explore table constraints that hold across multiple tables.
3. Build a control plane run-time monitor.
4. Use table constraints to drive compiler optimizations.
THANK YOU

QUESTIONS?
Standard Hoare logic axioms for commands:

Predicates P, Q ::=  
- true  
- false  
- equals expr expr  
- less expr expr  
- not P  
- P ∧ Q  
- P ∨ Q  
- P → Q  
- forall X. P  
- exists X. P

First order logic with comparisons over program expressions.

Assignments, if statements, and table applications

Axioms describing what is true before and after a command executes.

How to handle tables?
parser start {
    return parse_ethernet; } 

def parser parse_ethernet {
    extract(ethernet);
    return select(ethernet.typ) {
        ETH_IPV4 : parse_ipv4;
        default: ingress; } } 

def parser parse_ipv4 {
    extract(ipv4);
    return ingress; } 

def control ingress {
    if(valid(ipv4) and ipv4.ttl > 0) {
        apply(ipv4_lpm);
        apply(forward); } }
parser start {
    return parse_ethernet; }

parser parse_ethernet {
    extract(ethernet);  
    return select(ethernet.ttyp) {
        ETH_IPV4 : parse_ipv4;
        default: ingress; }
}

parser parse_ipv4 {
    extract(ipv4);  
    return ingress; }

control ingress {
    if(valid(ipv4) and ipv4.ttl > 0) {
        apply(ipv4_lpm);
        apply(forward); }
}
Assignments, if statements, and table applications

```p4
parser start {
    return parse_ethernet;
}

parser parse_ethernet {
    extract(ethernet);
    return select(ethernet.typ) {
        ETH_IPV4 : parse_ipv4;
        default: ingress;
    }
}

parser parse_ipv4 {
    extract(ipv4);
    return ingress;
}

c control ingress {
    if(valid(ipv4) and ipv4.ttl > 0) {
        apply(ipv4_lpm);
        apply(forward);
    }
} pane
```
```plaintext
parser start {
    return parse_ethernet;
}

parser parse_ethernet {
    extract(ethernet);
    return select(ethernet.typ) {
        ETH_IPV4 : parse_ipv4;
        default: ingress;
    }
}

parser parse_ipv4 {
    extract(ipv4);
    return ingress;
}

control ingress {
    if(valid(ipv4) and ipv4.ttl > 0) {
        apply(ipv4_lpm);
        apply(forward);
    }
}
```

```
ethernet.valid := 1;
ethernet.src := havoc;
ethernet.dst := havoc;
ethernet.typ := havoc;
if(ethernet.typ == ETH_IPV4)
    ipv4.valid := 1;
    ipv4.src := havoc;
    ipv4.dst := havoc;
    ipv4.ttl := havoc;
    if(valid(ipv4) and ipv4.ttl > 0)
        apply(ipv4_lpm);
        apply(forward);
```
GOAL 1

Automatically detect violations of generic safety properties, including:

- reads of uninitialized values
- unsafe arithmetic operations
- unhandled parser exceptions
- and so on…
GOAL 1

Automatically detect violations of generic safety properties, including:

- reads of uninitialized values
- unsafe arithmetic operations
- unhandled parser exceptions
- and so on...

And enable programmers to describe expected control plane behavior with table constraints.
GOAL 2

Check table constraints in the control plane with run-time monitoring.

Flag or reject updates that violate table constraints.
Automatically generate table constraints necessary to ensure correct behavior.
MORE GOALS

- Automatically generate table constraints necessary to ensure correct behavior.
- Use table constraints to drive compiler optimizations.
Observation: P4 programs are loop-free* table graphs.
Observation: P4 programs are loop-free* table graphs.

Convert parsers/controls to functions with imperative bodies.
Observation: P4 programs are loop-free* table graphs.
Observation: P4 programs are loop-free* table graphs.
Observation: P4 programs are loop-free* table graphs.

- Convert parsers/controls to functions with imperative bodies
- Unroll parser loops and inline function bodies
- Define primitive actions as reads/writes of metadata fields
- Treat tables and externs as uninterpreted functions
- Assignments, if statements, and table applications
parser start {
    extract(eth);
    return ingress; }

control ingress {
    if valid(eth) {
        apply(acl);
        apply(forward); } }
parser start {
    extract(eth);
    return ingress; }

control ingress {
    if valid(eth) {
        apply(acl);
        apply(forward); }
}

def start() =
    extract(eth);
    ingress();
def ingress() =
    if valid(eth) then
        acl();
        forward();
    start();

Convert parsers/controls to functions with imperative bodies

Assignments, if statements, and table applications
parser start {
    extract(eth);
    return ingress;
}

control ingress {
    if valid(eth) {
        apply(acl);
        apply(forward);
    }
}

def start() =
    extract(eth);
    ingress();

def ingress() =
    if valid(eth) then
        acl();
        forward();
    start();

extract(eth);
if valid(eth) then
    acl();
    forward();
def extract(eth) =
    eth.valid = 1;
    eth.srcaddr = havoc;
    eth.dstaddr = havoc;
    eth.ethTyp = havoc;

def valid(h) = h.valid == 1

extract(eth);
if valid(eth) then
    acl();
    forward();
def extract(eth) =
    eth.valid = 1;
    eth.srcaddr = havoc;
    eth.dstaddr = havoc;
    eth.ethTyp = havoc;

def valid(h) = h.valid == 1

extract(eth);
if valid(eth) then
    acl();
    forward();

eth.valid = 1;
eth.srcaddr = havoc;
eth.dstaddr = havoc;
eth.ethTyp = havoc;
if eth.valid == 1 then
    acl();
    forward();
parser start {
    @pragma assert(X == egress_spec);
    extract(eth);
    @pragma assume(eth.ethTyp == 0x806)
    return ingress; }
control ingress {
    if valid(eth) {
        apply(acl);
        apply(forward);
        @pragma assert(egress_spec != X) }
    }

Assertion/assumption annotations
are passed through to IMP

assert(X == egress_spec);
extract(eth);
assume(eth.ethTyp == 0x806);
if valid(eth) then
    acl();
    forward();
    assert(egress_spec != havoc);
table acl { reads = { eth.ethTyp; } 
    actions = { drop; nop; } }

control ingress {
    if valid(eth) {

        apply(acl);
        @pragma assert(eth.ethTyp == 0x86DD ==> egress_spec == DROP)

        apply(forward); }
    }
}
**COMPILE TIME**

```
bit<1> do_monitoring;
table Sample {
    reads = { … }
    actions = { nop; } }
table Monitor { … }
table Forward { … }
control ingress {
    Sample.apply();
    if (do_monitoring) Monitor.apply();
    Forward.apply(); }
```

Metadata for flow sampling

Decides whether Monitor is applied

But is never set...

Packet-processing pipeline
bit<1> do_monitoring;

table Sample {
    reads = { … }
    actions = { sampling; set; nop; } }

table Monitor { … }

table Forward { … }

control ingress {
    Sample.apply();
    if (do_monitoring) Monitor.apply();
    Forward.apply(); }

Packet-processing pipeline

Metadata for flow sampling

Decides whether Monitor is applied

sample() randomly sets do_monitoring

set() explicitly sets do_monitoring
Problems:

1. `nop()` leaves `do_monitoring uninitialized`
2. Table "miss" is `nop()`

Control Plane

- Add:
  `if ipv4_src == 10.12/16 then set(1)`
- Add:
  `if ipv4_src == 10/24 then sample()`
- Add:
  `if ipv4_src == 192/24 then nop()`