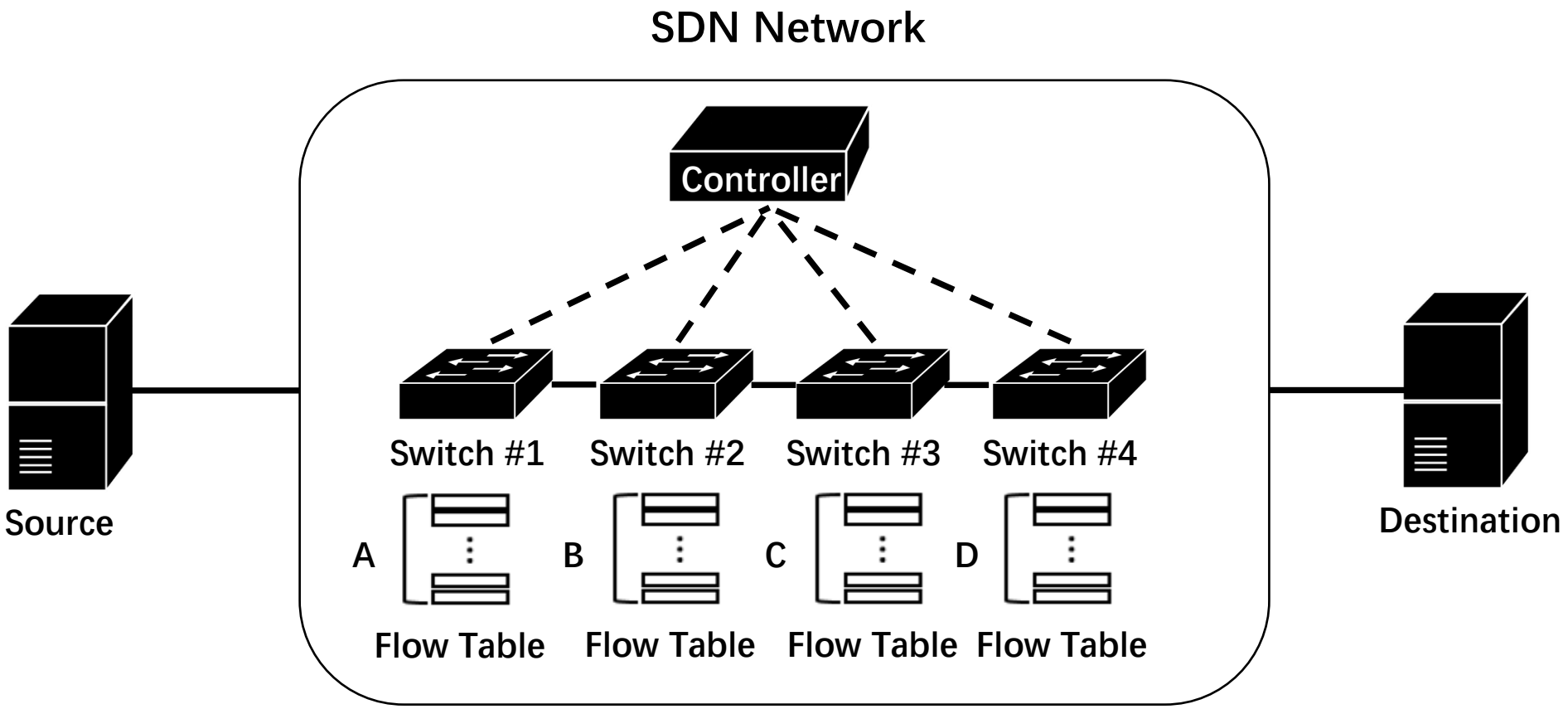
On Temporal Verification of Stateful P4 Programs

Delong Zhang*, Chong Ye*, Fei He

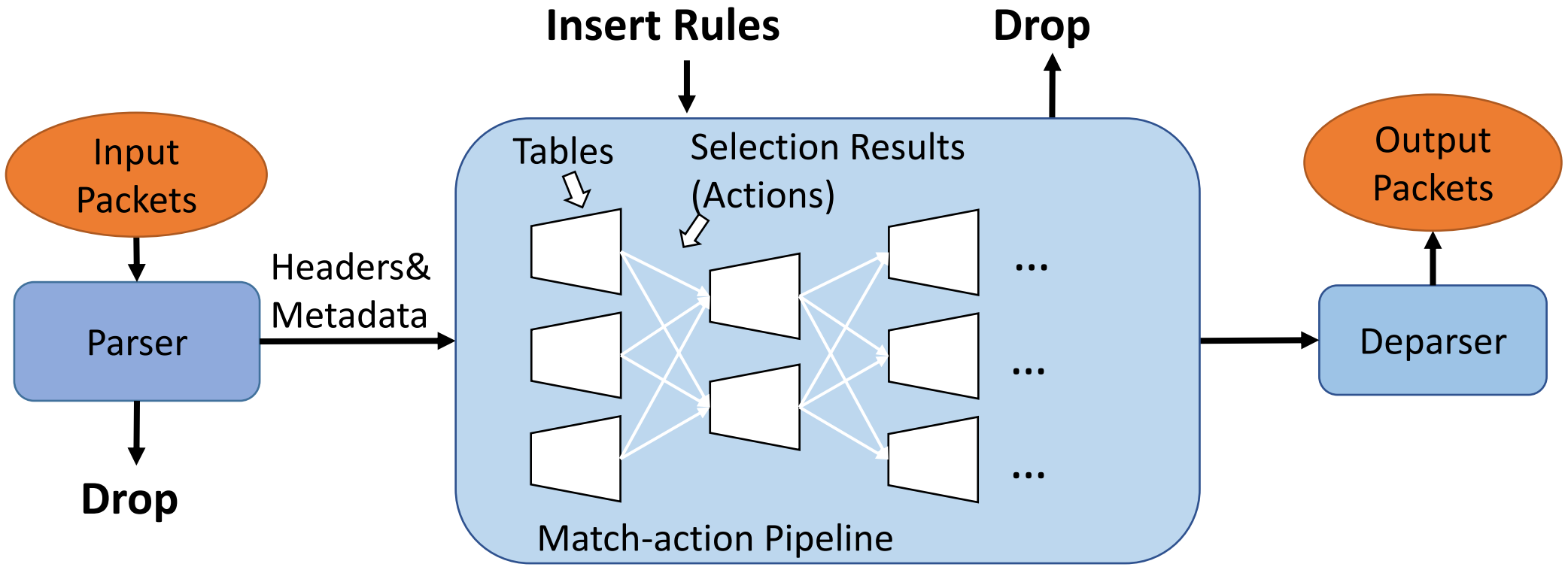
School of Software, Tsinghua University

SDN offers centralized control and remarkable flexibility.



Programming Protocol-Independent Packet Processors

P4 is a DSL for data plane programming.



Patrick Bosshart, et al. P4: Programming Protocol-Independent Packet Processors. HotSDN 2014.

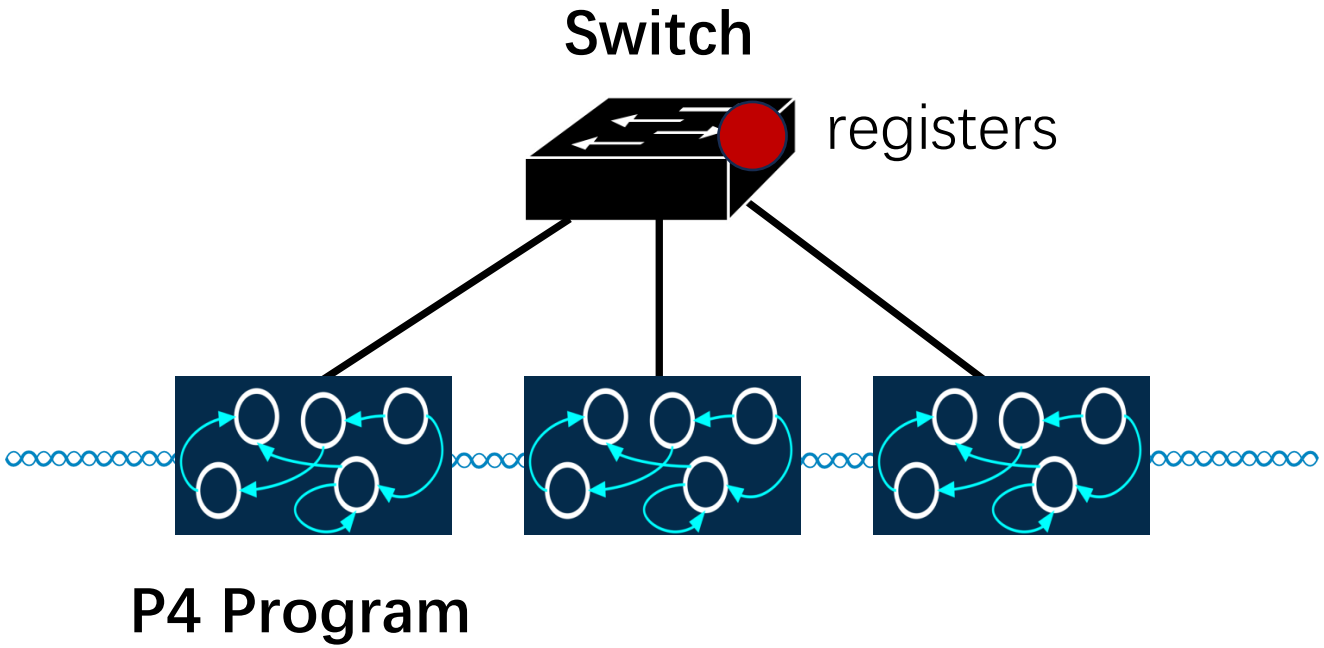
A P4 Program Example

```
header ethernet_t {
  bit <48 > dstAddr ;
  bit <16 > etherType ;
}
struct headers { … }
parser MyParser ( … ) {
    state start {
      …
      transition …
    }
}
table ipv4_lpm {
  key = { hdr.ipv4.dstAddr : lpm ; }
  actions = { my_forward ; drop ; }
  default_action = drop () ;
}
control MyIngress ( … ) {
   …
}
```

Main components:

- **Headers**: the potential format of incoming packets' headers
- **Parser**: extract information from the incoming packets
- **Match-Action Table**: declare the format of match-action rules
- **Control Logic**: specify the execution logic of the match-action pipeline
- **Deparser**: encapsulate the information to the output packet

P4 Programming Language



- P4 program is executed for each incoming packet.
- A special stateful memory, called **register**, is used to store state information across multiple packet processing.

Verifying Correctness of P4

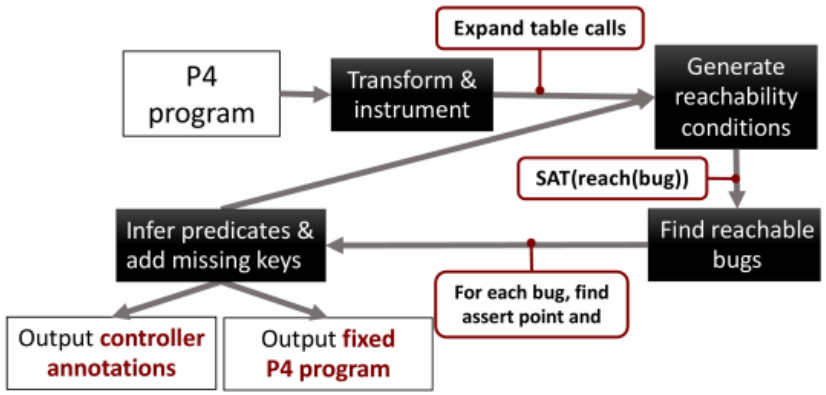


Figure. bf4 system architecture

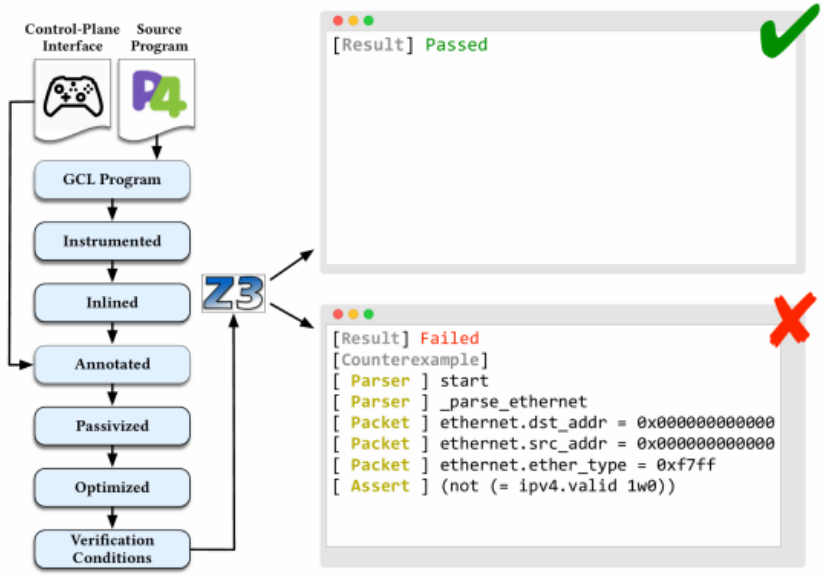


Figure. p4v system architecture

Numerous P4 program verifiers have recently been developed, including ASSERT-P4 , Aquila , bf4 , p4v , P4-NOD, and Vera.

State Modeling

Existing P4 verifiers overapproximate the register variables as **fully non-deterministic**.

```
procedure main() {
    nondet_register_variables();
    call P4Program();
}
```

Pros

Reduce reasoning of infinite packet processing to a single processing.

Cons

Precision lost! Unable to track the register states.

A simplified snippet of P4NIS [G. Liu, et al.]

```
count.read(status,0); // send packages circularly
if(status == 1)
    egress_spec = status = 2;
if(status == 2)
    egress_spec = status = 3;
if(status == 3)
    egress_spec = status = 1;
count.write(status,0); // record the port status
```

Safety Property: the forwarding port should always remain within the valid range, i.e., $1 \leq egress\_spec \leq 3$.

The property is **safe** if *count* is initially configured in the valid range [1, 3].

However, priors will roughly report **unsafe**, as all registers' values being assumed **fully non-deterministic**!

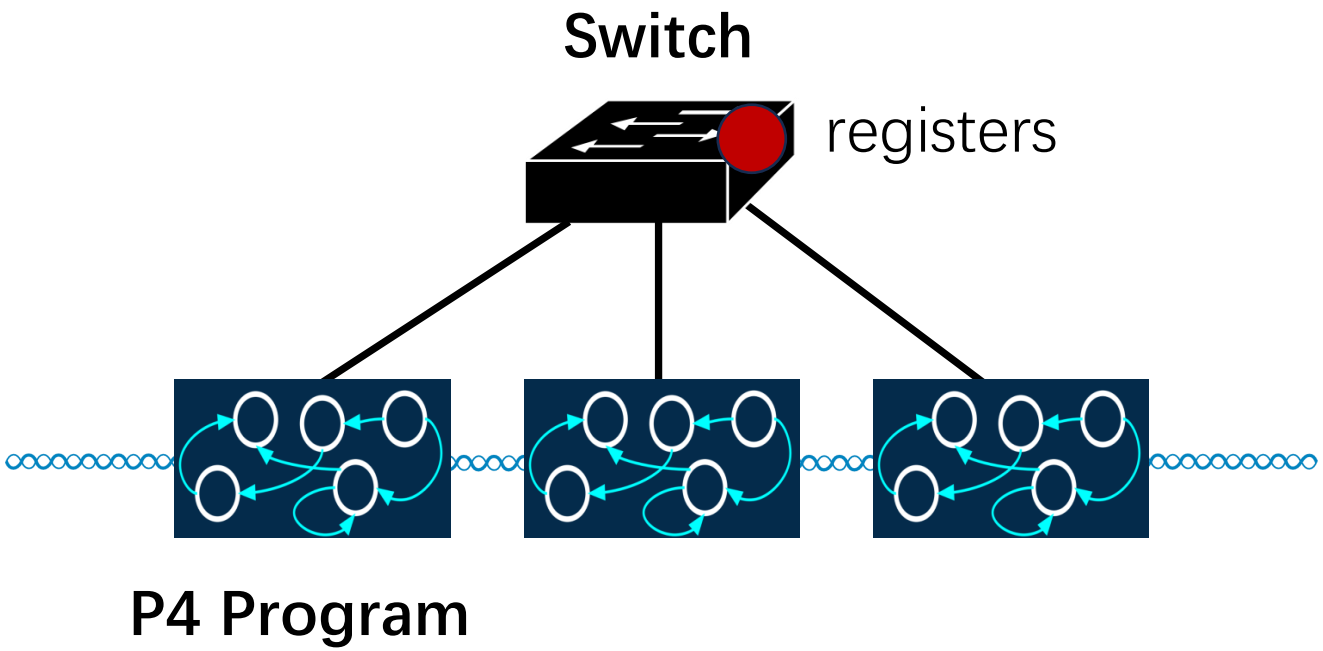
A simplified snippet of P4NIS

```
count.read(status,0); // send packages circularly
if(status == 1)
    egress_spec = status = 2;
if(status == 2)
    egress_spec = status = 3;
if(status == 3)
    egress_spec = status = 1;
count.write(status,0); // record the port status
```

Liveness Property: When valid packets arrive infinitely often, every valid port should be activated infinitely often.

No prior P4 verifiers are capable of checking liveness!

Challenges



1. How to model the stateful nature of P4 programs?
2. How to specify temporal properties of P4 programs at the packet processing level?
3. How to verify temporal properties of P4 programs?

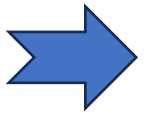
Challenge 1

How to model the stateful nature of P4 programs?

Environment Model

We explicitly model the stateful nature of P4 program.

```
procedure main() {
    nondet_register_variables();
    call P4Program();
}
```



```
procedure main() {
    init_register_variables();
    while (true){
        init_packet();
        call P4Program();
    }
}
```

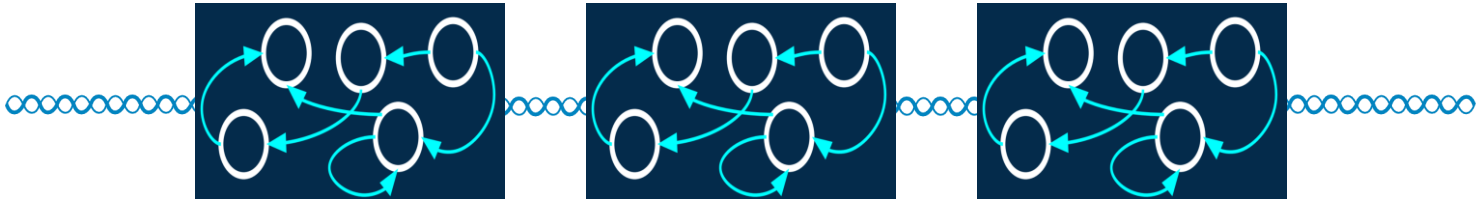
Priors

Ours

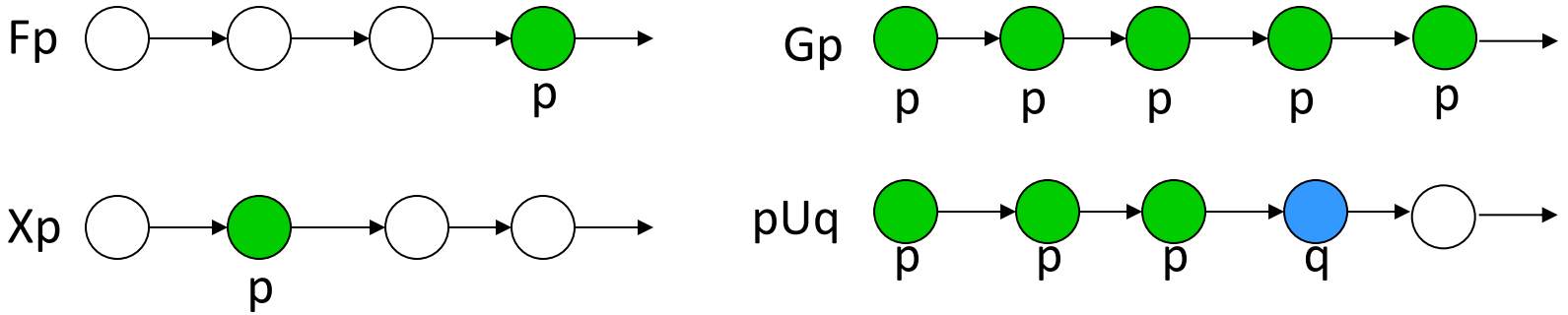
Challenge 2

How to specify temporal properties of P4 programs at the level of packet processing?

Specification Language



Behavior of a P4 program at the packet processing level



Linear Temporal Logic

P4LTL

P4LTL

$$
\begin{aligned}
\text{P4LTL}\,\varphi \quad &\rightarrow \quad \psi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \\
&\quad\; \mid \circ\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi\, \mathcal{U}\, \varphi \mid \varphi\, \mathcal{W}\, \varphi \\
\text{Pred}\,\psi \quad &\rightarrow \quad t\,\text{comp}\,t \mid \text{fwd}(t) \mid \text{valid}(h) \mid \text{hit}(t) \mid \text{drop} \\
&\quad\; \mid \text{apply}(t,a) \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \psi \rightarrow \psi \\
\text{Term}\,t \quad &\rightarrow \quad d \mid h \mid m \mid v \mid r[t] \mid \text{old}(t) \mid \text{key}(t,k) \\
&\quad\; \mid \text{para}(t,a,p) \mid t\ \text{op}\ t
\end{aligned}
$$

Key X	Key Y	Action
1	2	forward(port = 3)
3	4	forward(port = 7)

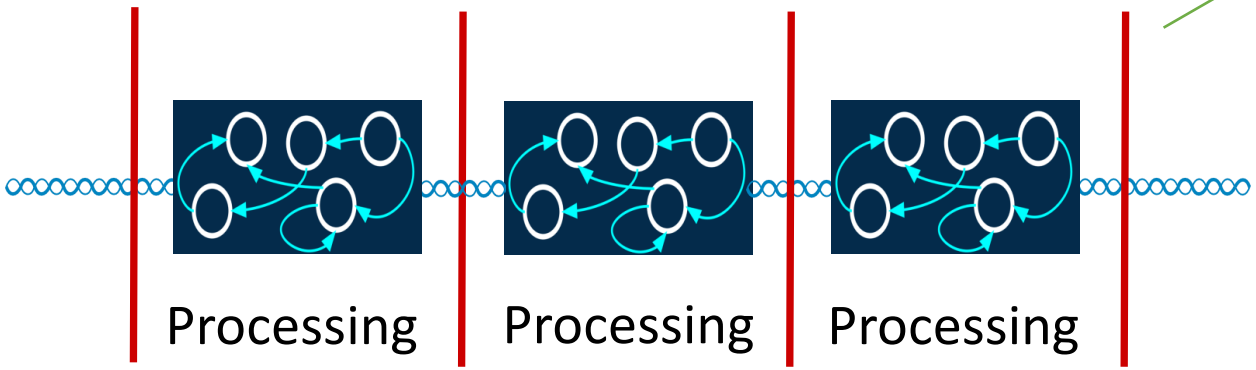
P4LTL

$$
\begin{aligned}
\text{P4LTL}\,\varphi \quad &\rightarrow \quad \psi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \\
&\quad\; \mid \circ\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi\, \mathcal{U}\, \varphi \mid \varphi\, \mathcal{W}\, \varphi \\
\text{Pred}\,\psi \quad &\rightarrow \quad t\,\text{comp}\,t \mid \boxed{\text{fwd}(t)} \mid \boxed{\text{valid}(h)} \mid \boxed{\text{hit}(t)} \mid \boxed{\text{drop}} \\
&\quad\; \mid \boxed{\text{apply}(t,a)} \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \psi \rightarrow \psi \\
\text{Term}\,t \quad &\rightarrow \quad d \mid h \mid m \mid v \mid r[t] \mid \text{old}(t) \mid \text{key}(t,k) \\
&\quad\; \mid \text{para}(t,a,p) \mid t\ \text{op}\ t
\end{aligned}
$$

Key X	Key Y	Action
1	2	forward(port = 3)
3	4	forward(port = 7)

P4LTL

$$
\begin{aligned}
\text{P4LTL}\,\varphi \quad &\rightarrow \quad \psi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \\
&\quad\;\; \mid \circ\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi\, \mathcal{U}\, \varphi \mid \varphi\, \mathcal{W}\, \varphi \\
\text{Pred}\,\psi \quad &\rightarrow \quad t\,\text{comp}\,t \mid \text{fwd}(t) \mid \text{valid}(h) \mid \text{hit}(t) \mid \text{drop} \\
&\quad\;\; \mid \text{apply}(t,a) \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \psi \rightarrow \psi \\
\text{Term}\,t \quad &\rightarrow \quad d \mid h \mid m \mid v \mid r[t] \mid \text{old}(t) \mid \text{key}(t,k) \\
&\quad\;\; \mid \text{para}(t,a,p) \mid t\ \text{op}\ t
\end{aligned}
$$



A **time step** is executed every time the P4 program finishes processing a packet.

Example

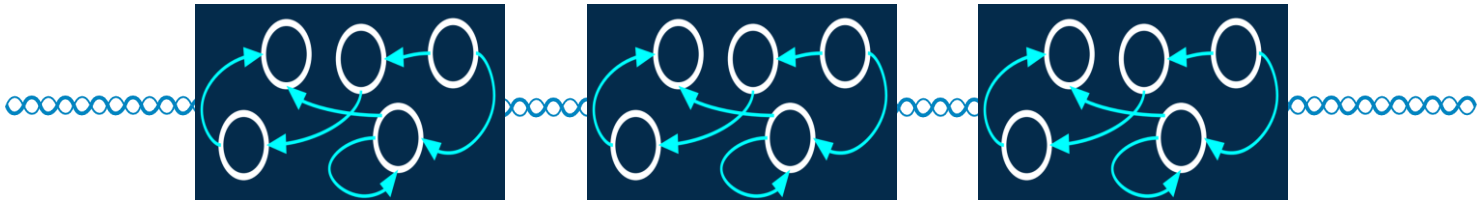
```
count.read(status,0); // send packages circularly
if(status == 1)
    egress_spec = status = 2;
if(status == 2)
    egress_spec = status = 3;
if(status == 3)
    egress_spec = status = 1;
count.write(status,0); // record the port status
```

Liveness Property: When valid packets arrive infinitely often, every valid port should be activated infinitely often.

$$\square\diamond(\text{valid}(\text{hdr.label})) \longrightarrow \left(\bigwedge_{i=1}^{n} \square\diamond \text{fwd}(i) \right)$$

Satisfiablity

A trace:



A trace τ **satisfies** a P4LTL specification ϕ is written as $\tau \vDash \phi$.

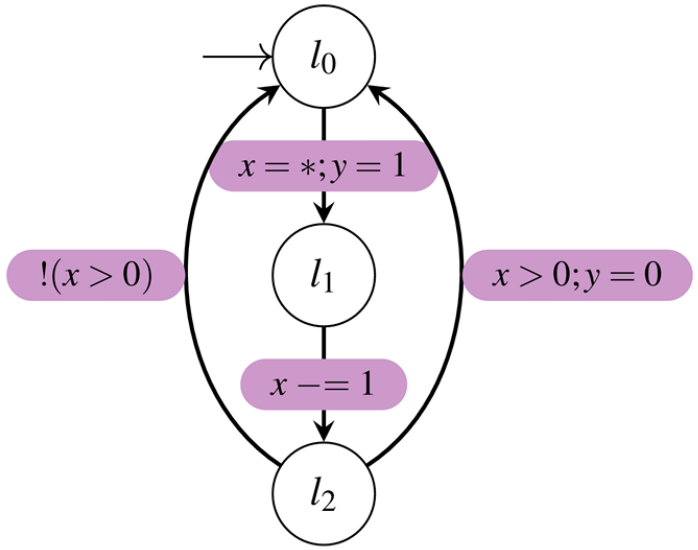
A P4 program P **satisfies** a P4LTL specification ϕ, written as $\mathrm{P} \vDash \phi$, iff for every trace τ of P, $\tau \vDash \phi$ holds.

Challenge 3

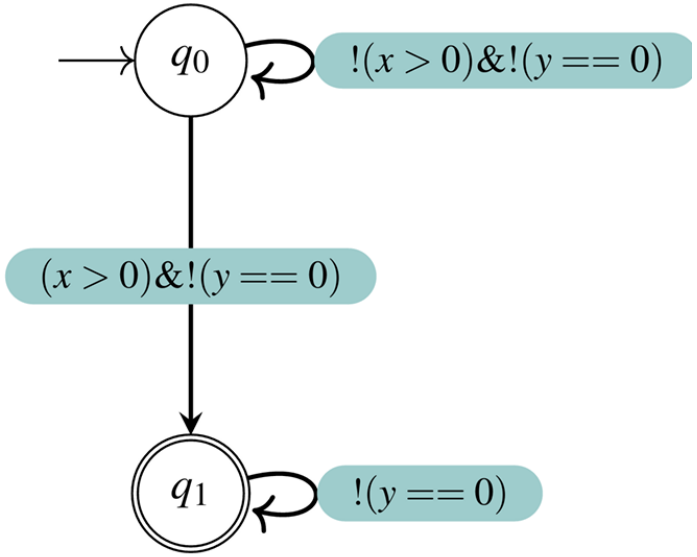
How to verify temporal properties of P4 programs?

Software Model Checking

Program Automaton A_P



Specification automaton $A_{\neg\phi}$



Product

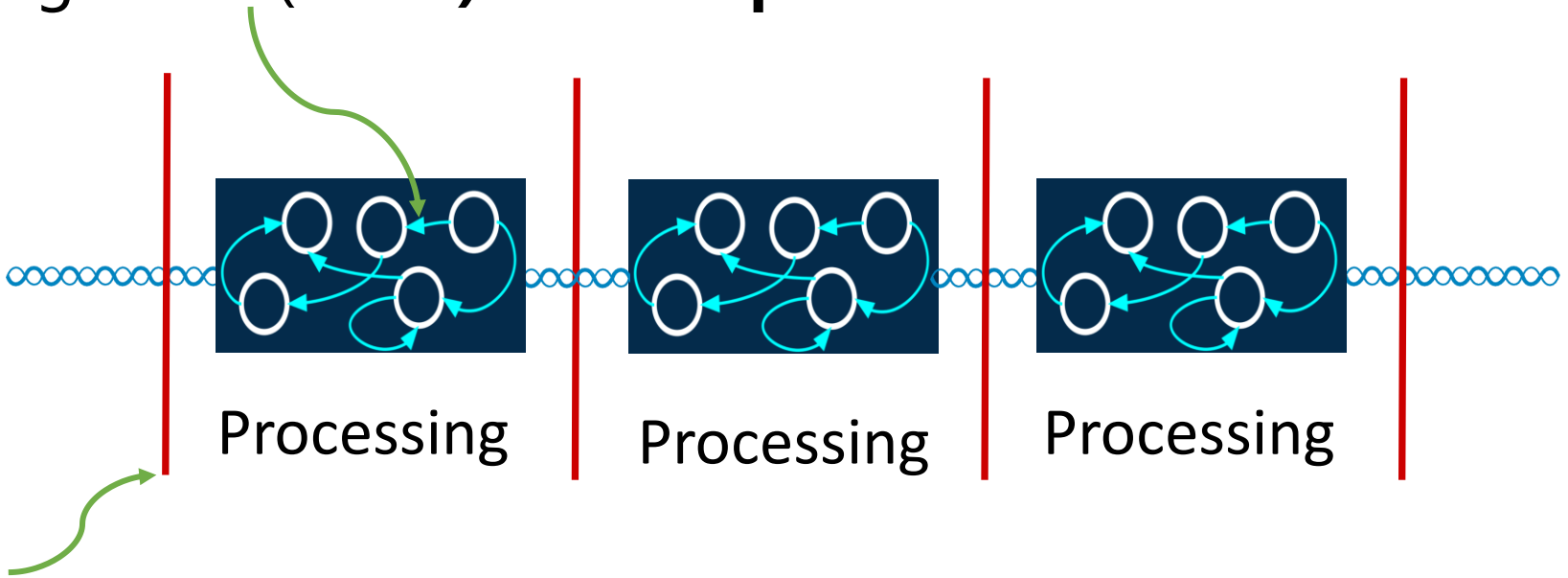
$A_P \times A_{\neg\phi}$

$P \vDash \phi$ iff $A_P \times A_{\neg\phi}$ recognizes an **empty** fair language

Challenge

There are two distinct time steps:

- In program: a (**small) time step** ticks with *each statement* executed.



- In specification: a **(big) time step** ticks when the *whole P4 program* executes to an end.

We must **align** the distinct time step in verification!

Büchi Transaction

Definition (Büchi Transaction) Consider a program $\mathcal{P} = (\mathcal{S}, \mathcal{L}, \ell_0, \delta_p)$. We call it $\mathrm{TE}(s)$/ $\mathrm{TR}(s)$ when $s \in \mathcal{S}$ is an entering/return statement to/from the transaction. Given a specification automaton $\mathcal{A}_\varphi = (\Sigma, Q, q_0, \delta, \mathcal{F})$, the Büchi transaction $\mathcal{P} \otimes \mathcal{A}_\varphi$ is a Büchi automaton $\mathcal{B} = \left(\widehat{\Sigma}, \widehat{Q}, \widehat{q_0}, \widehat{\delta}, \widehat{\mathcal{F}}\right)$ where:

- $\widehat{\Sigma} = \{s;\ \mathrm{assume}\ (\phi) \mid s \in \mathcal{S} \wedge \phi \in \Sigma\}$
- $\widehat{Q} = \{(l, q) \mid l \in \mathcal{L} \wedge q \in Q\}$ and $\widehat{q_0} = (\ell_0, q_0)$
- $\widehat{\delta} = \{((l, q), s; \mathrm{assume}\ (\phi), (l', q')) \mid$
  $(\mathrm{TR}(s) \wedge (l, s, l') \in \delta_p \wedge (q, \phi, q') \in \delta) \vee$
  $(\neg \mathrm{TR}(s) \wedge (l, s, l') \in \delta_p \wedge q = q' \wedge \phi = true)\}$
- $\widehat{\mathcal{F}} = \{(\ell, q) \mid q \in \mathcal{F} \wedge (\mathrm{TE}(\ell) \vee \mathrm{TR}(\ell))\}$, where
  - $\mathrm{TE}(\ell) : \exists \ell' \in \mathcal{L}, s \in \mathcal{S}.\ (\ell, s, \ell') \in \delta_p \wedge \mathrm{TE}(s)$,
  - $\mathrm{TR}(\ell) : \exists \ell' \in \mathcal{L}, s \in \mathcal{S}.\ (\ell', s, \ell) \in \delta_p \wedge \mathrm{TR}(s)$.

- **Büchi product**: Every time program automata advances, the specification automata also advances.
- **Büchi Transaction**: The specification automata advances only when the program transaction is complete.

Büchi Transaction

Theorem 1

The P4 program P satisfies the P4LTL specification φ if and only if the Büchi transaction $B = P \otimes A_{\neg\varphi}$ does not have any fair and feasible trace.

[Proofs are detailed in our paper.]

Evaluation

P4TV

We implemented our prototype named **p4tv**, and compared with the state-of-the-art prior tools:

- **bf4**, in SIGCOMM'2020
- **p4v**, in SIGCOMM'2018
- **vera**, in SIGCOMM'2018
- **Aquila**, in SIGCOMM'2021

Temporal Verification

The benchmarks consist of 9 non-trivial P4 programs and 19 stateful verification tasks.

Name	Benchmark Description
Bfs [45]	The fast failover mechanism in a Bfs fashion to reroute packets without control plane interference.
Blink [28]	Detect characteristic failure based on information aggregated with registers and trigger rerouting.
CoDel [36]	An active queue implementation for RFC 8289.
Dfs [45]	A Dfs variation of the Bfs benchmark.
Ndn [46]	A named data networking paradigm where data are retrieved based on names rather than addresses.
P4NIS [38]	Several mechanisms for eavesdropping protection.
P4sp [37]	A sensor failure recovery system.
P4xos-acceptor [19]	P4 implementation of the acceptor in Paxos protocol.
P4xos-learner [19]	P4 implementation of the learner in Paxos protocol.

All prior verifiers failed, while p4tv successfully solves all.

Assertion Verification

Prior P4 verifiers focus on **assertion verification**, which is a special case of P4LTL formulas.

Benchmark	Assertion	Stateful?	p4tv	bf4	p4v
SimpleNat-bug1_6	Whether the header is valid before accessing.	No	✗ (2m)	✗ (3s)	✗ (4s)
SimpleNat-safe1_11			✓ (10s)	✓ (3s)	✓ (4s)
P4NIS-safe1	The egress port should be equal to the register value if tunnel is applied.	Yes	✓ (2m36s)	✓ (3s)	✓ (2s)
P4NIS-safe2	Labeled packets cannot be implicitly dropped.	Yes	✓ (59s)	Δ(3s)	Δ(3s)
P4xos-acceptor-safe	Stored consensus round must be always less than the current valid round.	Yes	✓ (28m21s)	Δ(4s)	Δ(3s)
P4xos-learner-unsafe	The value of vote counter register is always zero.	Yes	✗ (6m54s)	Δ(2s)	Δ(3s)
P4xos-acceptor-unsafe	Stored consensus round must be always equal to the current valid round.	Yes	✗ (3m40s)	Δ(3s)	Δ(2s)

p4tv effectively confirms safe assertions (✓) and reports counterexamples for unsafe ones (x).

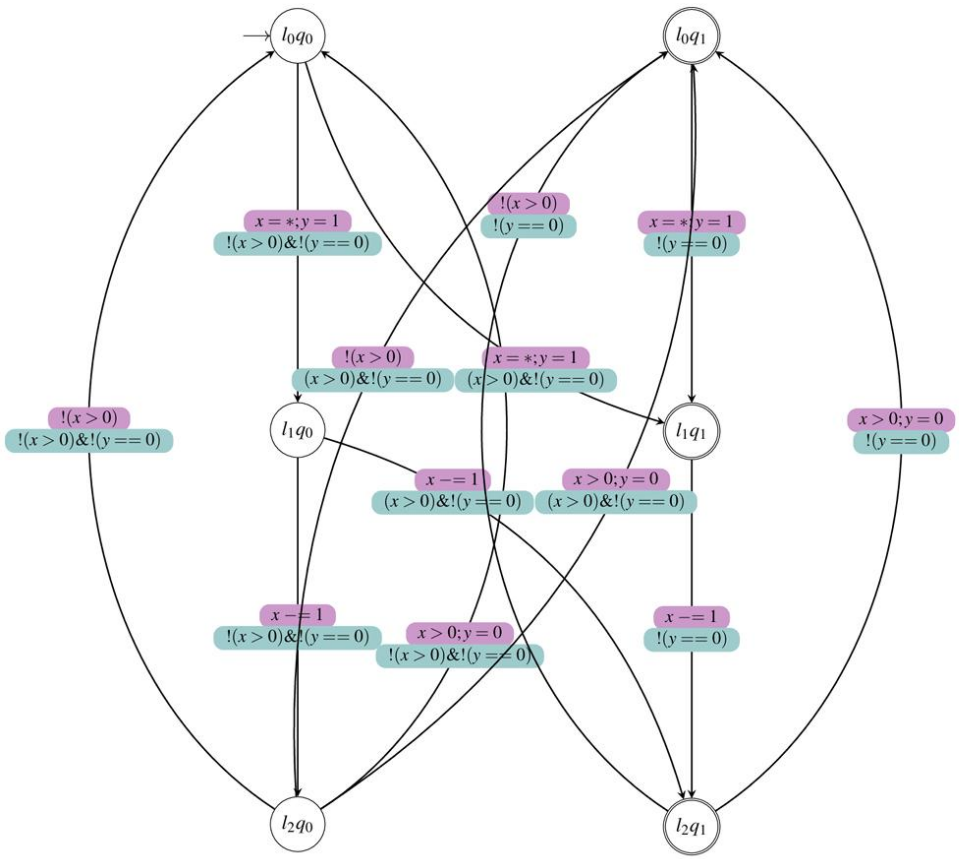
In contrast, prior verifiers often return spurious counterexamples (Δ), which would not occur under the given configurations.

Conclusion

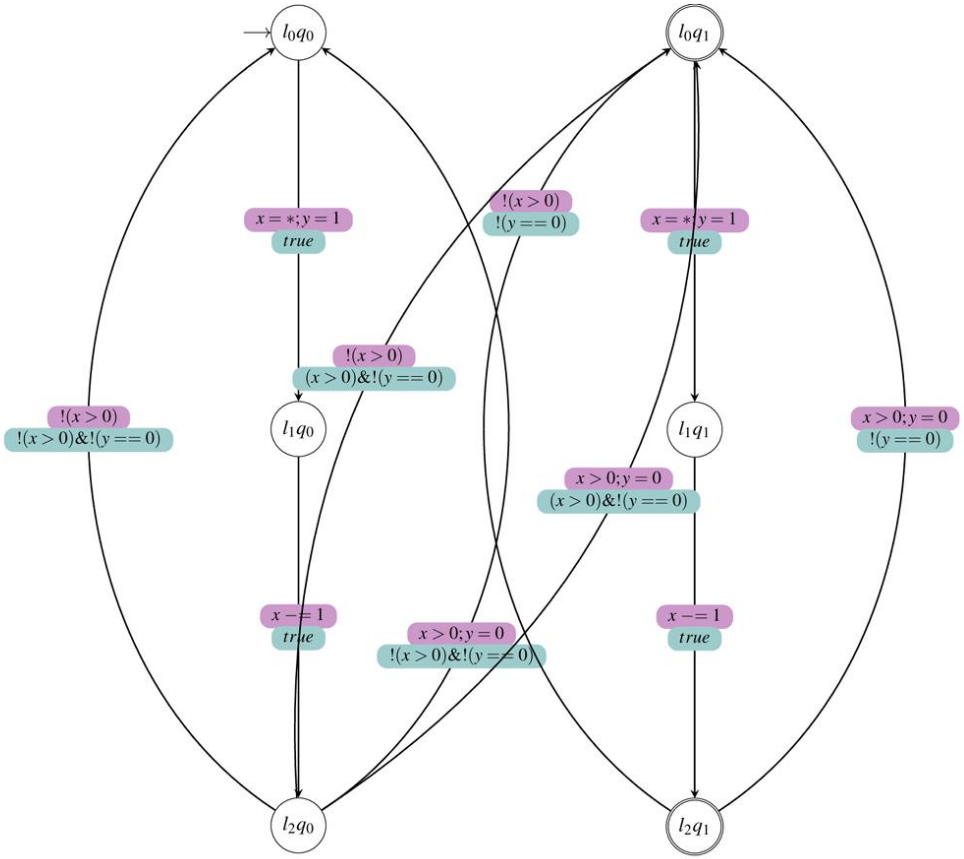
- The **first** approach for temporal verification of P4 programs at packet processing level
- A specification language, **P4LTL**, for describing temporal properties of stateful P4 programs at the packet processing level
- A novel concept, called **Büchi Transaction**, for aligning time steps between P4 programs and P4LTL specifications

- Thanks! For questions, feel free to contact us at hefei@tsinghua.edu.cn

Büchi Transaction



(1) Büchi Program product



(2) Büchi Transaction

Büchi Transaction enables **transaction level temproal verification**, and reduces the number of transitions and accepted traces.