Formal methods in computer networks

E.P. Stepanov

References:

Qadir, J.; Hasan, O.,

Applying Formal Methods to Networking: Theory, Techniques, and Applications,

Communications Surveys & Tutorials, IEEE , vol.17, no.1, pp.256-291, 2015

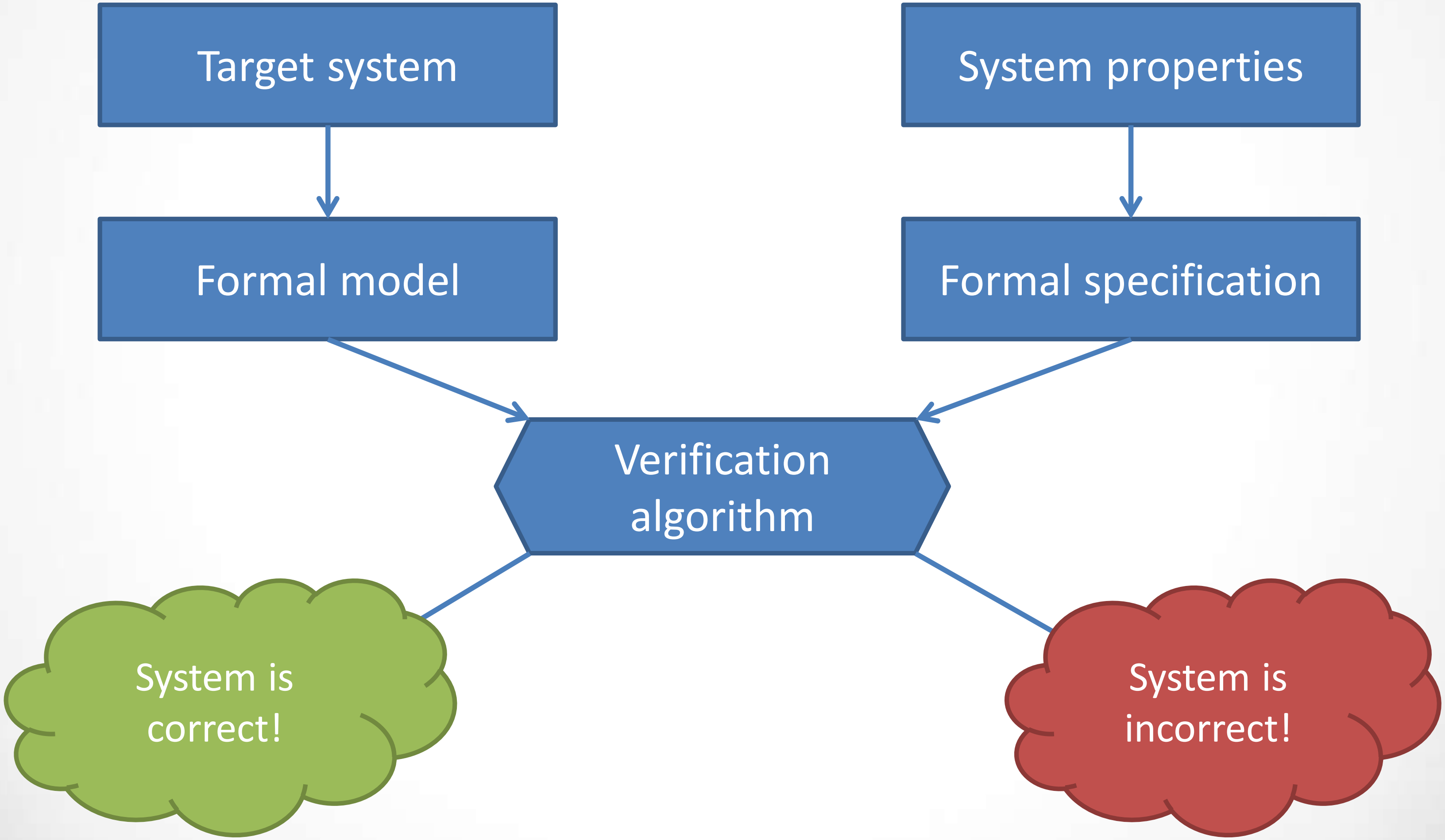
Formal methods

- ***Formal methods*** – such methods of specification, design and analysis of the system properties that have a rigorous mathematical justification
- ***Verification*** – checking the conformity of system properties to a given specification
- ***Synthesis -*** building a system whose properties satisfy a given specification

Components of the formal verification method

- ***Model description language -*** a method of mathematical description of the device and characteristics of the analyzed system
- ***Specification language -*** a way to formally describe the properties of an analyzed system
- ***Verification method -*** an algorithm for checking the conformity of a system model with a given specification

Components of the formal verification method



Peterson's Algorithm(1981)

Input

Global variables:

bool want1, want2; // process $x \in \{1,2\}$ works with critical section

int turn; // process «turn» has priority access to the critical section

// The change in the value of each variable occurs atomically

want1 = false, want2 = false, turn = 1; // initial value

Process №1:

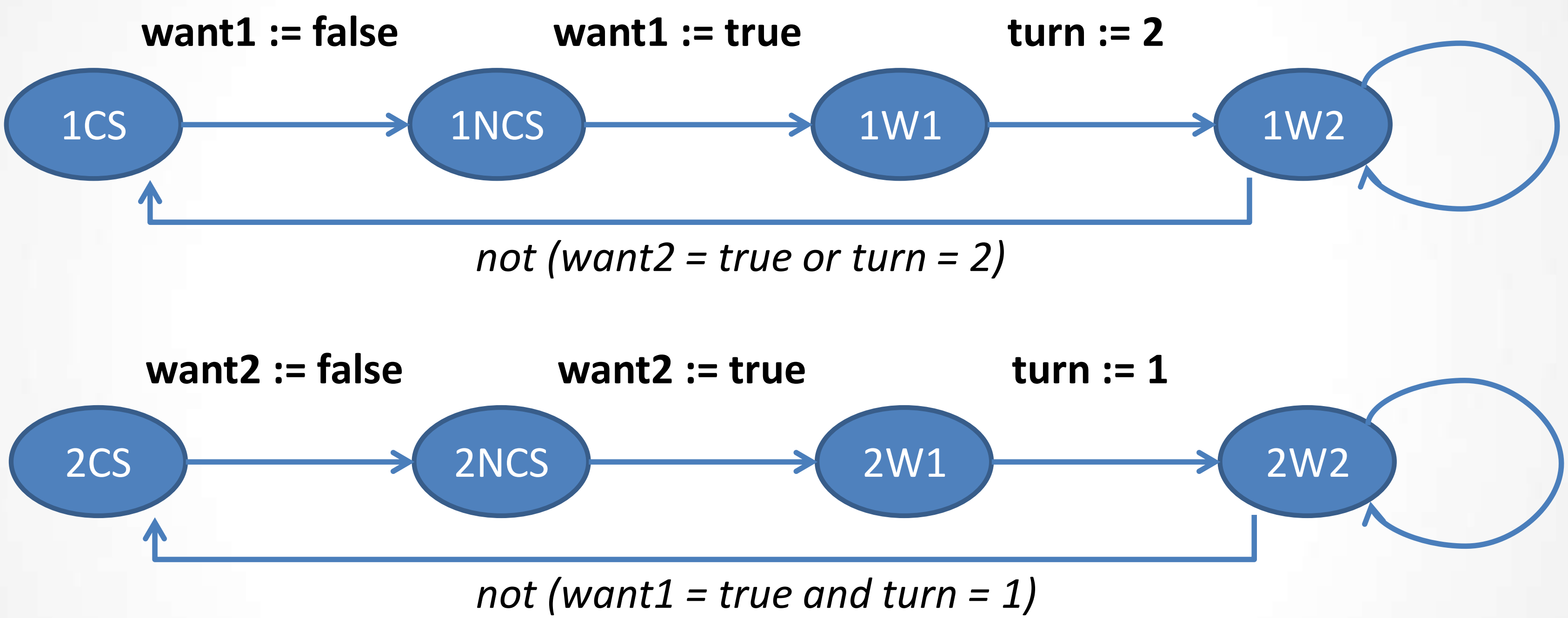
```
while (true) {
   <noncritical section>
   want1 := true;
   turn := 2; // give way to 2
   while (want2 and turn == 2)
     do { /* busy wait */ };
   <critical section>
   want1 := false;
}
```

Process №2:

```
while (true) {
   <noncritical section>
   want2 := true;
   turn := 1; // give way to 1
   while (want1 and turn == 1)
     do { /* busy wait */ };
   <critical section>
   want2 := false;
}
```

Peterson's Algorithm (1981)

Formal model



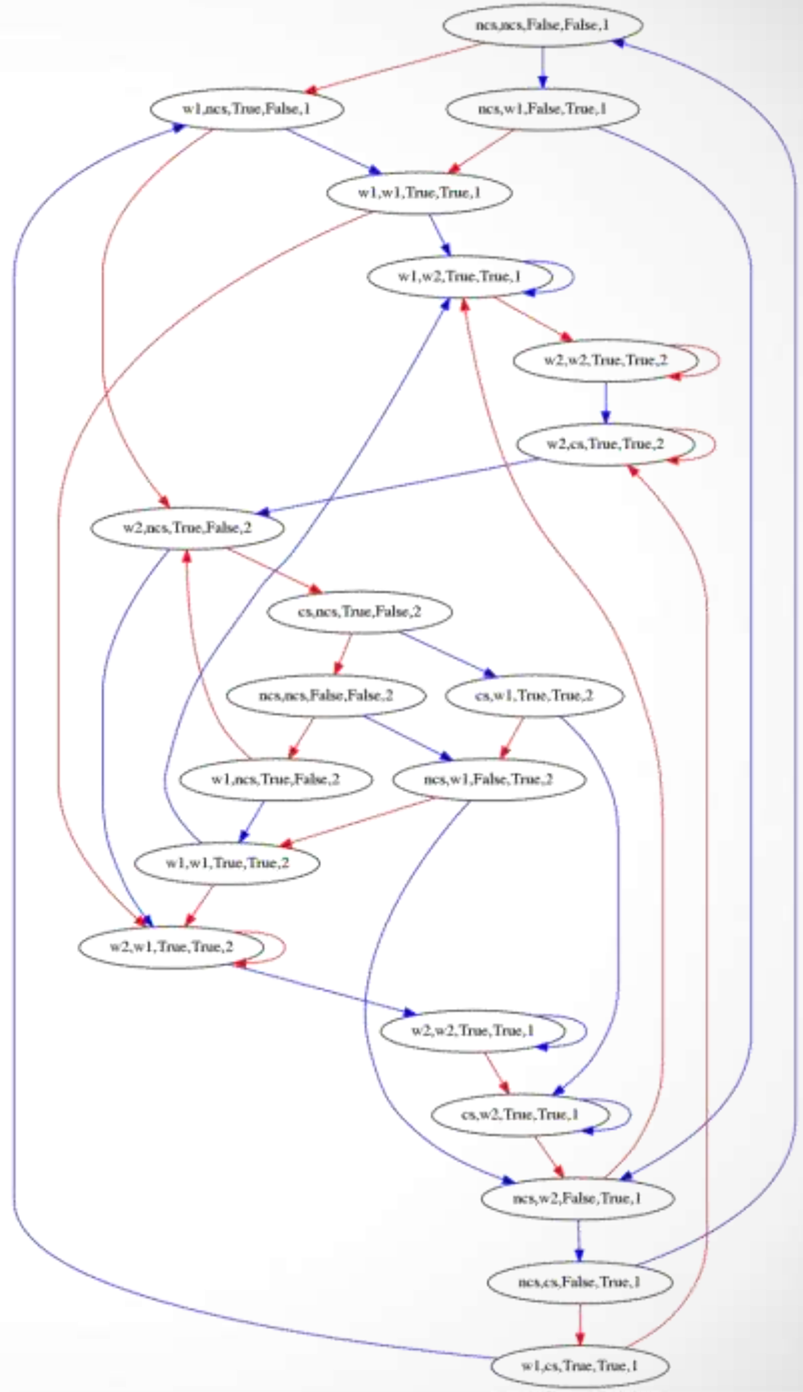
** An inner loop condition can be represented by two intermediate states in accordance with the rules for checking logical expressions (short-circuit evaluation)*

Peterson's Algorithm (1981)

Formal model

- The system is described by a finite state machine obtained by a superposition of automata for each of the processes
- System state - a set of shared variable values and states of each process
  - What is the total number of states?

Transition graph for a system of two processes:
total 20 states

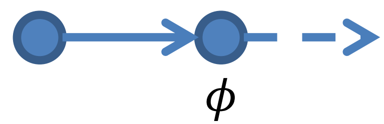


Peterson's Algorithm (1981)

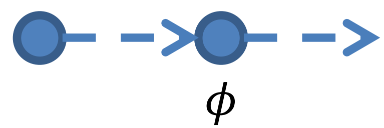
Properties specification

- The properties of a system, expressed by the relationships between its states, can be defined by the temporal logic formulas LTL :

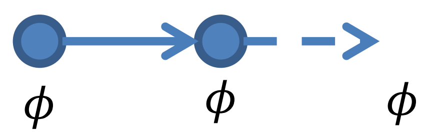
- $X\phi$ (ne**X**t)



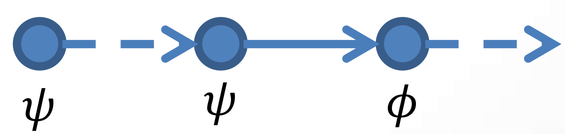
- $F\phi$ (**F**uture)



- $G\phi$ (**G**lobally)



- $\psi U\phi$ (**U**ntil)



Peterson's Algorithm (1981)

Properties specification

- ***Safety***

  There will never be a situation in which both processes are simultaneously inside the critical section

  Verification == Checking reachability

  $$\mathbf{G}\neg(1CS \wedge 2CS)$$

- ***Liveness***

  The process who wants to get into the critical section sooner or later will get there

  Verification == Cycle search

  $$\boldsymbol{G}(1NCS \rightarrow \boldsymbol{F}1CS) \wedge \mathbf{G}(2NCS \rightarrow \mathbf{F}2CS)$$

Formal verification vs. Testing

Problem	Method Efficiency	
	Formal Verification	Testing
Error search	medium	high
Correctness proof	high	low
Usage complexity	high	low

Error type	Frequent	Rare
Harmless	Testing	---
Critical	Testing, Formal verification	**Formal verification**

Target system types

- ***Transforming systems***
  - Transform input to output
  - Can be represented by a table or formula
  - Switch with static rules
- ***Responsive systems***
  - Behavior depends on impact history
  - Can be represented by a state machine
  - Controller Reactive Program

The main properties of mathematical models

- ***Abstractness***

  The model should be a simplification of the system

- ***Adequacy***

  The model should reflect the properties of the system.

- ***Compactness***

  The size and structural complexity of the model affects the complexity of solving the verification problem.

  Symbolic Methods - *the model does not explicitly lists all the system state*

Some examples of formal models

- Tables
- Graph representations
- Boolean formulas
- Binary Decision Diagrams
- Labeled Transition Systems
- Time automata
- Petri nets

The main properties of specification languages

- ***Expressive power***

  The language should cover the studied properties

- ***Problem solving complexity***

  Formula equivalence check

  Checking consistency of formulas

  Formal verification

- ***Compatible with model***

  The complexity of verification depends on the *consistency* of the model with the specification language

Some examples of specification languages

- Propositional logic
- First-Order Predicate Logic
- Higher Order Logic
- Logic of Hoar
- Modal logic
- Temporal logic

Formal Verification Methods

- Non-automatic

  Manual Proof

- Semi-automatic

  Theorem proving

  Semisolvability

- Automatic

  Model checking

  combinatorial explosion

The historical path of computer network development

"Internet has become a victim of its success"

- Internet value is too high
- Metcalf's Empirical Law

Value for formal methods:

- Trial and error development
- Engineering practice is valued above theoretical research

Founding Fathers about Formal Methods

- Formal methods have not yielded results commensurate with the effort to use them. They are overblown, verbose, hard to use, hard to understand.

Vint Cerf

- We reject: kings, presidents and voting.
  We believe in: rough consensus and running code.

David Clark

Principles of computer network design

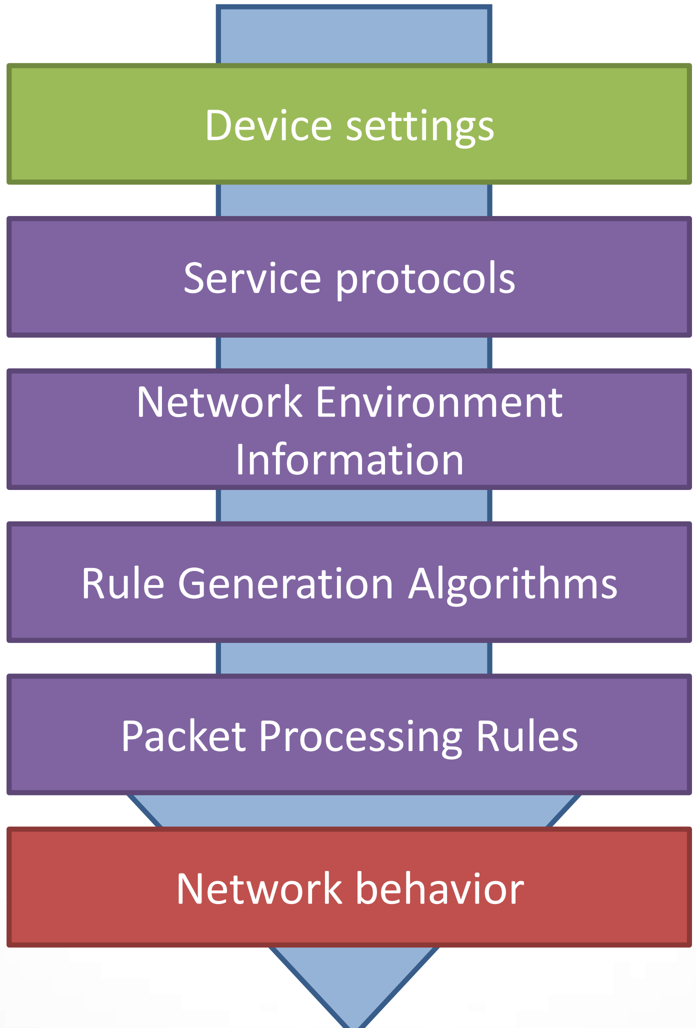
David Clark

The Design Philosophy of the DARPA Internet Protocols

SIGCOMM ’88. — ACM, 1988. — Pp. 106–114.

- Fault tolerance
- Variety of data transfer services
- Support for a wide range of networks
- Distributed resource management
- Profitability
- Extensibility
- Accounting for used network resources

Prediction of network behavior



Network Protocol Verification

Investigated Properties :

- ***Deadlocks*** - waiting for conditions that will never be met
- ***Livelocks*** – execution of protocol instructions does not bring it closer to the goal
- ***Improper termination*** – the protocol finishes its work without reaching the goal

The number of states is potentially infinite - the method of exhaustive search is not applicable!

Verification of static network configuration

Investigated Properties :

- ***Black Holes*** – silent packet dropping
- ***Fowrarding Loops*** – packet looping
- ***Reachability*** – will the packet reach the destination point?
- Restrictions on the route length
- Flow isolation

You need an expressive specification language!

How to restore network behavior?

Network Security

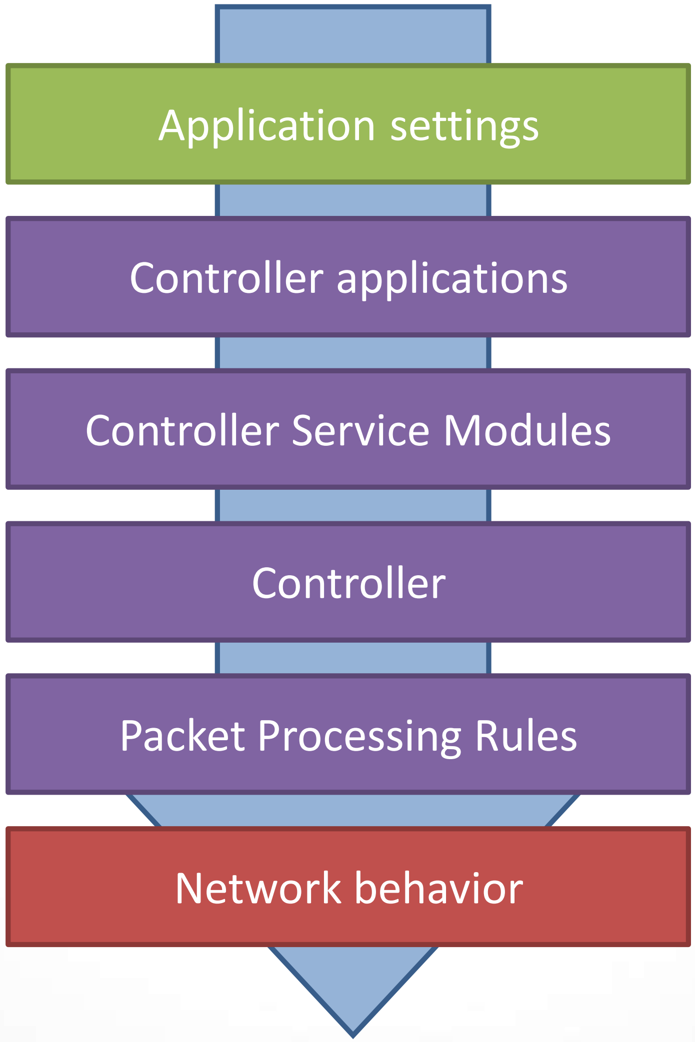
Verification and synthesis of firewall configurations:

- Equivalence check
- Redundancy Check
- Synthesis of a configuration consisting of a minimum number of rules

Formal methods and SDN

- New network equipment control protocols provide access to up-to-date packet processing rules
- Centralization of the network made it possible to collect information about the network configuration at a single point and track its change
- Now you can adapt the developments from other software development fields

New networks - new challenges



Programming SDN w/ OpenFlow

- The Good
  - Network-wide visibility
  - Direct control over the switches
  - Simple data-plane abstraction

- The Bad
  - Low-level programming interface
  - Functionality tied to hardware
  - Explicit resource control

- The Ugly
  - Non-modular, non-compositional
  - Challenging distributed programming

SDN Programming

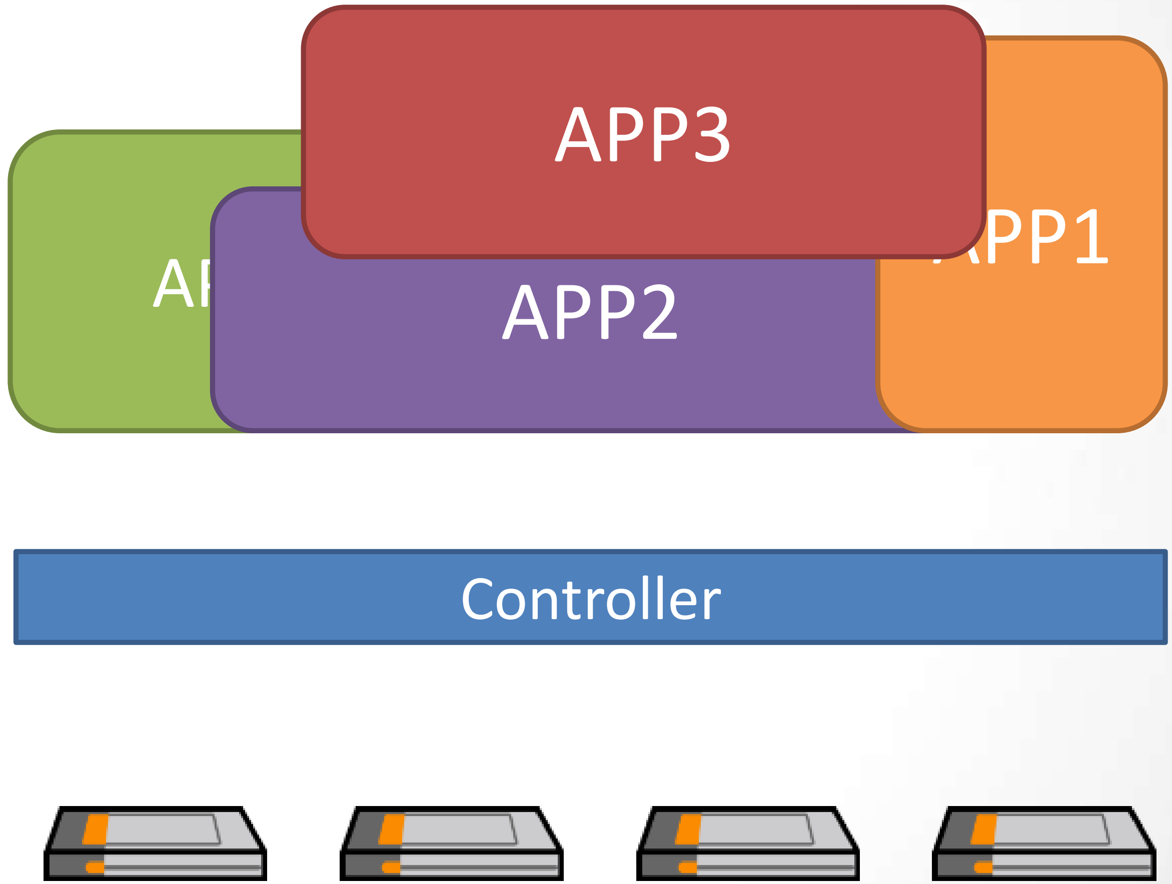
Applications

Controller API

Platform

Switch API

Switches



SDN Programming

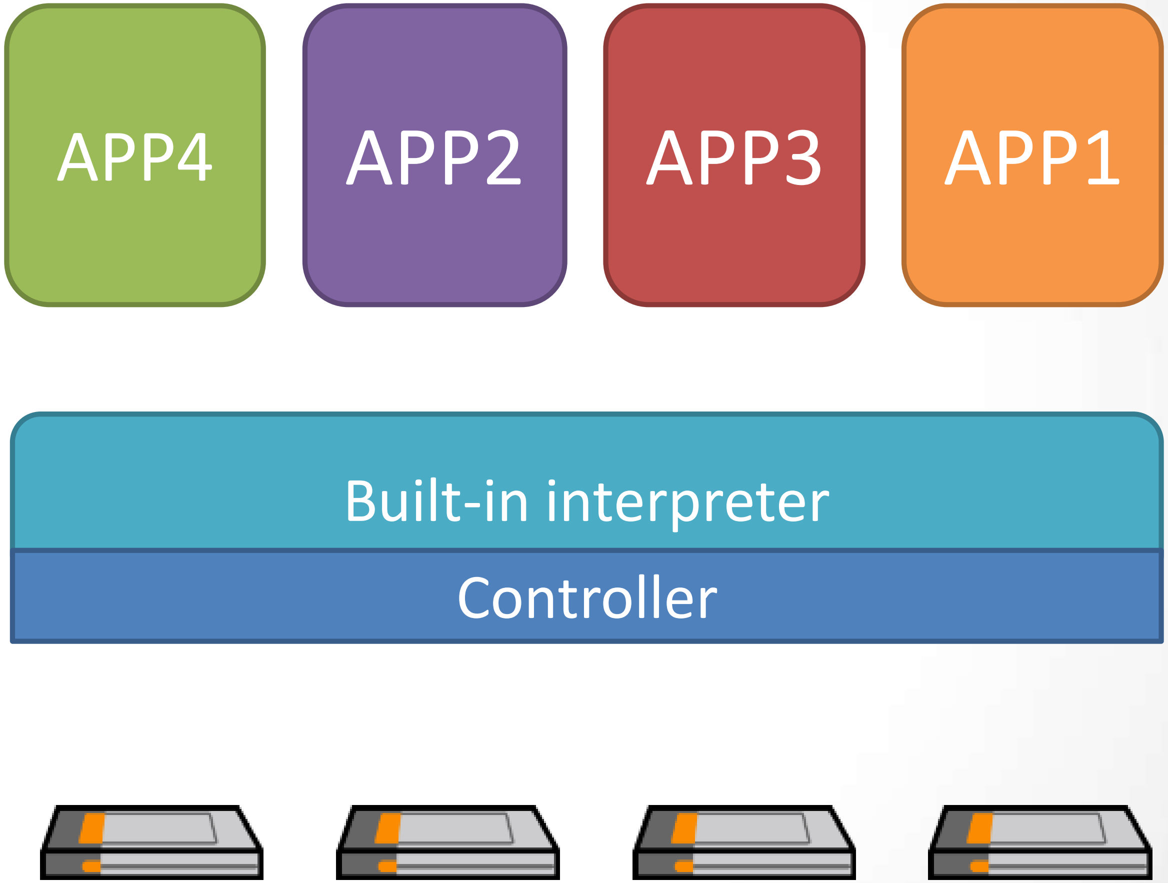
Applications

Controller API

Platform

Switch API

Switches



SDN Programming

```
from pyretic.lib.corelib import *
# send packets to the all ports
def main():
        return flood()
# block the host with IP 10.0.0.1
def access_control():
        return ~(match(srcip='10.0.0.1') |
                 match(dstip='10.0.0.1'))
```

Control Plane Verification

Marco Canini, Daniele Venzano et. all

A NICE way to test OpenFlow applications

A controller program execution model taking into account the state of the entire SDN

Thomas Ball, Nikolaj Biorner et all.

VeriCon: Towards Verifying Controller Programs in Software-Defined Networks

Verifies the program for all network models

Data Plane Verification

- Less assumptions about control applications on the controller
- Checks the execution of policies at the level of rule checking - the method is insensitive to *induced errors* inside the controller
- A natural interpretation of the routing policy concept

Applying Vermont tool to dataplane verification

E.P. Stepanov

Routing policies

Requirements for computer network behavior

- External flows do not reach the mail server
- Outbound flows pass through DPI
- There is a route between each pair of hosts in the office
- Different department networks are isolated
- All routes within the network are shorter than six hop
- Packets do not form routing loops
- Host A cannot connect to host B until host B tries to connect to host A
- The connection throughput is not less than R, and the data transfer delay does not exceed T

VERMÓNT
VERifying MONiTor

VERMONT checks packet processing rules in switch tables for compliance with formal routing policy specifications

Necessary

Express network behavior requirements using our specification language

One-time job

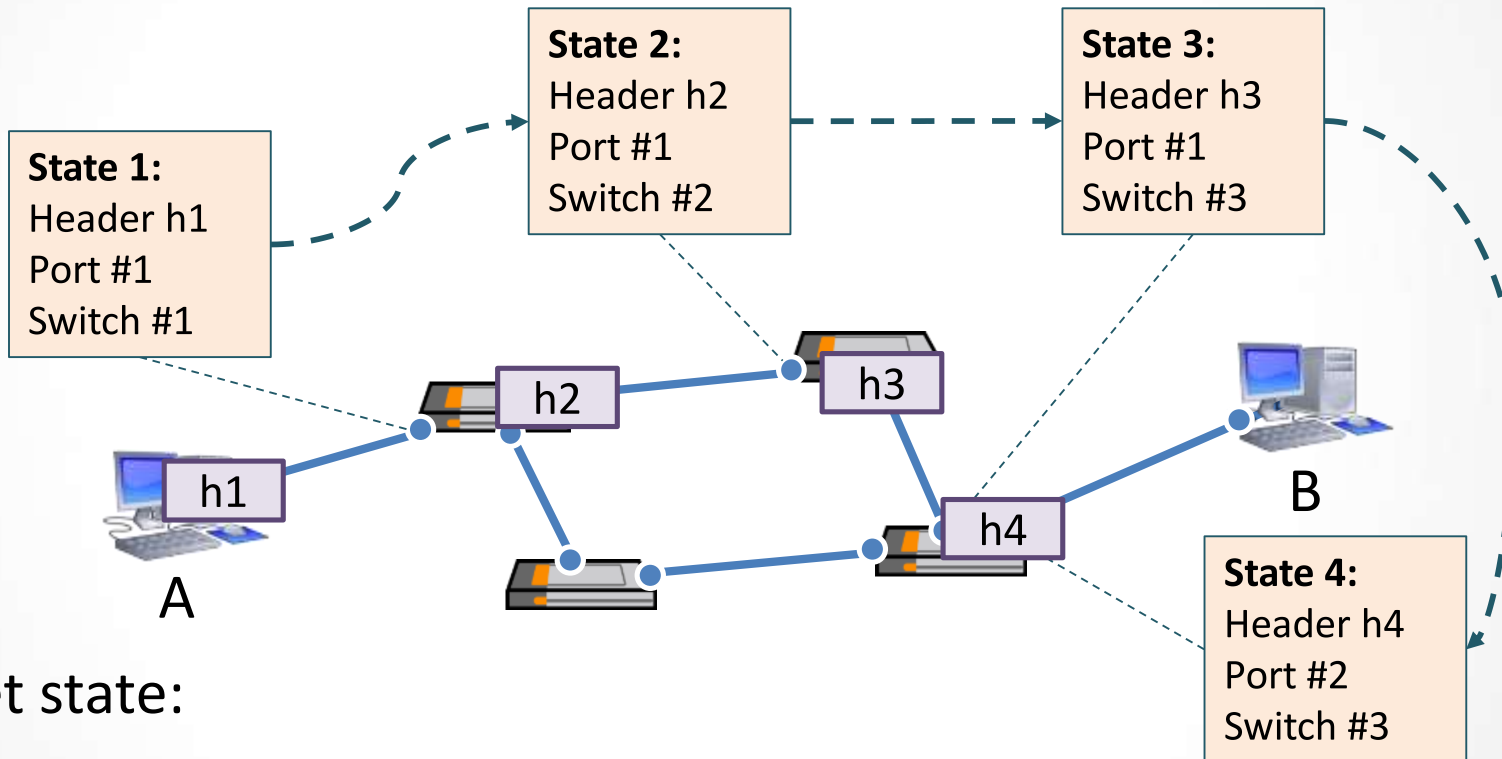
Provide topology and configuration files for switch devices

Automation is possible

Benefit

- Guarantee that network operates according our expectations
- Identify erroneous network configurations
- Information on the reasons for the policy violation

Analysis of routing relationship properties



Packet state:

1. Switch name	Switch	$\{0, 1\}^m$
2. Port number	Port	$\{0, 1\}^l$
3. Packet header	Header	$\{0, 1\}^k$

Relational network model

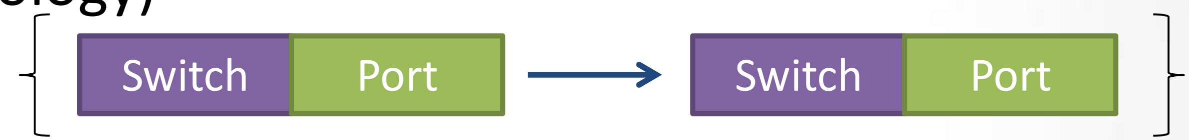
A network is defined by a set of relationships

Input/output ports

(links, network topology)

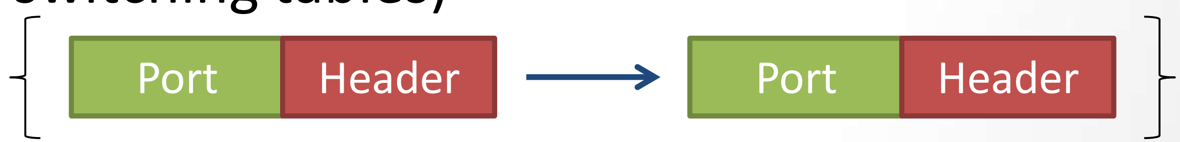
Transmission relation

(links, network topology)



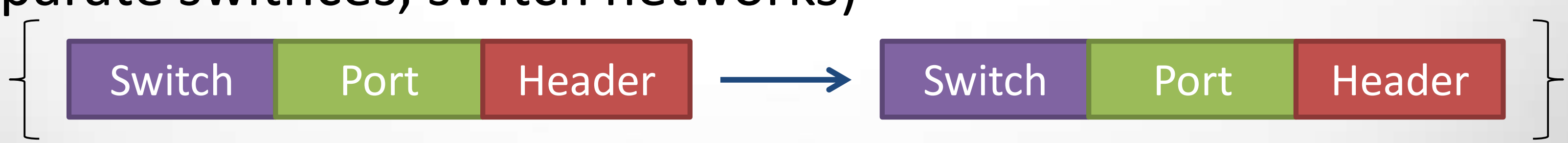
Switching relation on the node

(separate rules and switching tables)



One-step routing relation

(separate swithces, switch networks)



Relational network model

A network is defined by a set of relationships

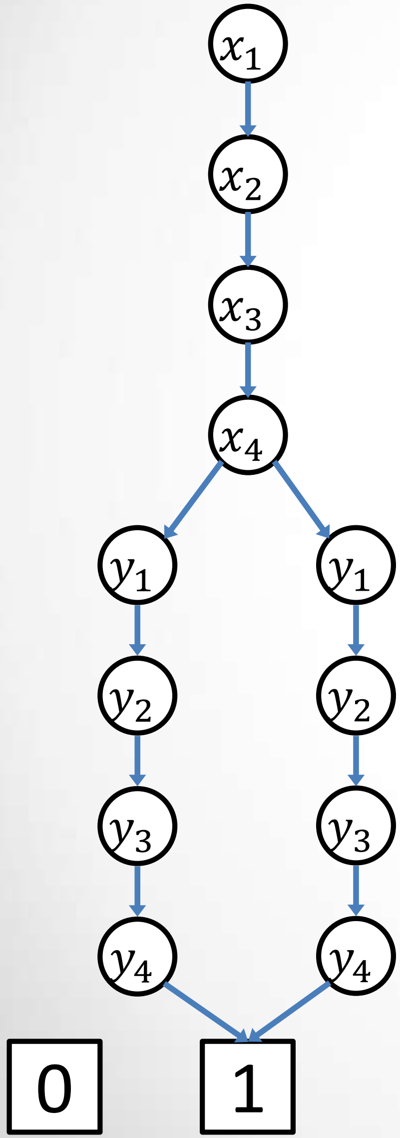
OpenFlow rule is modeled by Pattern and the set of Rewrite patterns

* - the packet falls under the Pattern, independent of bit value

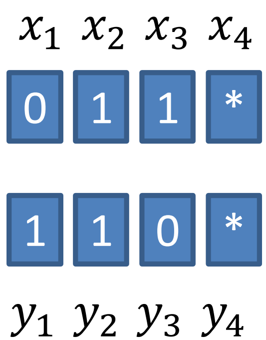
	P[0]	H[0]	H[1]	H[2]	H[3]
		x1 x2 x3 x4			
Pattern	0 0 0 1	0 1 1 *	0 0 0 1	* * * *	0 0 0 1
Rewrite	0 0 0 1	1 1 0 *	0 0 0 1	* * * *	0 0 0 1
		y1 y2 y3 y4			

* - Rewrite pattern does not change bit value

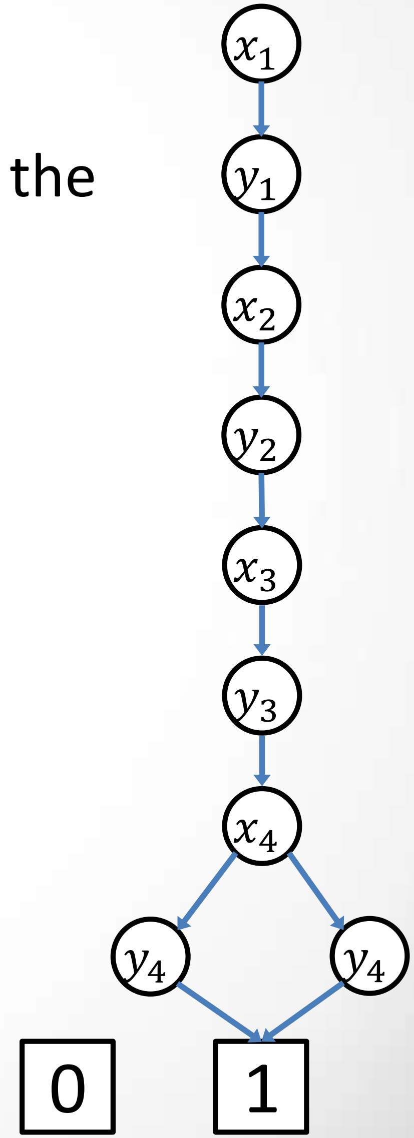
Binary Decision Diagram (ROBDD)



The size of the BDD depends on the choice of variable order



$$(\overline{x_1}y_1)(x_2y_2)$$
$$(x_3\overline{y_3})(x_4y_4 \vee \overline{x_4}\overline{y_4})$$



BBD generation by a network state

- Generate BDD for each pair $\langle Pattern, Rewrite \rangle$
- Generate BDD for each rule of FlowTable
- Generate BDD for each switch
- Generate BDD for all switches S
- Generate BDD for the topology T
- Generate the composition of S and T
- Generate the relation R
- Generate a transitive closure R^*

Specificatio language

Equalities	x[field] = const		x[field] = y[field]
Bundles	$\boldsymbol{\varphi} \wedge \boldsymbol{\varphi}$	$\boldsymbol{\varphi} \vee \boldsymbol{\varphi}$	$\neg\boldsymbol{\varphi}$
Quantifiers	$\forall \boldsymbol{x}.\,\boldsymbol{\varphi}$		$\exists \boldsymbol{x}.\,\boldsymbol{\varphi}$
Closures	$\boldsymbol{\psi}^{+}$		$\boldsymbol{\psi}^{[\boldsymbol{i},\boldsymbol{j}]}$

Rules to calculate the closure

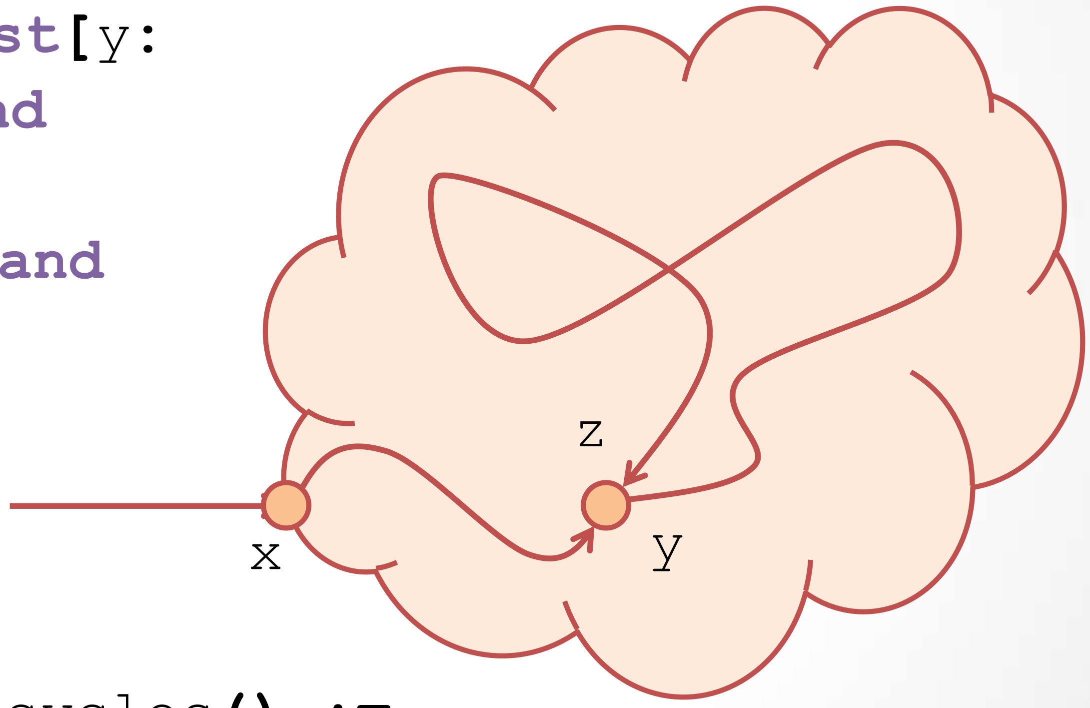
$$\psi^1(x,y) = \psi(x,y)$$
$$\psi^n(x,y) = \exists z\colon\ \psi^{n-1}(x,z) \wedge \psi(z,y)$$
$$\psi^{[\mathrm{i,j}]}(x,y) = \psi^i(x,y) \vee \cdots \vee \psi^j(x,y)$$
$$\psi^+(x,y) = \psi^{[1,\infty]}(x,y)$$

Relations that determine the network behavior

Input	$In(x)$
Output	$Out(x)$
Switching step	$R(x,y)$
Reachability relation	$R^+(x,y)$

Example: banning loops by packet state

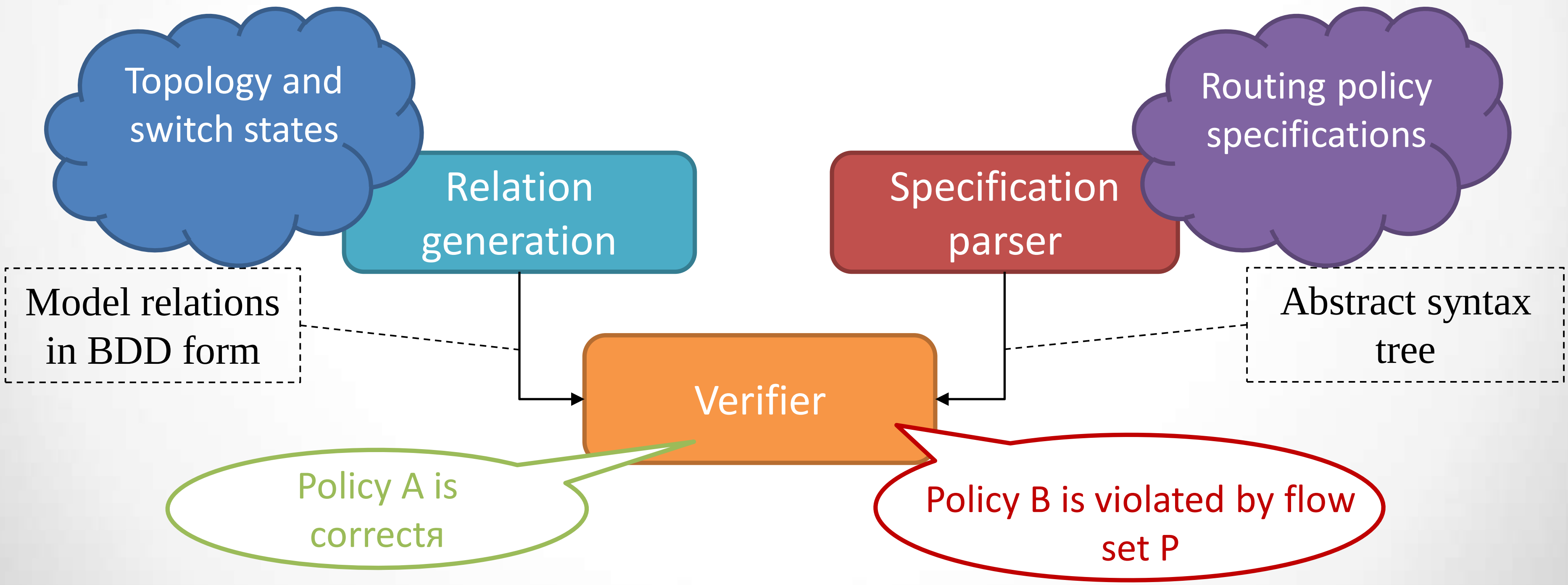
```
aux: lead_to_cycle(x) :=
  In(x) and Exist[y:
    R_tc(x,y) and
    Exist[z:
      R_tc(y,z) and
      y == z
    ]
  ];
```




```
main: no_state_cycles() :=
  Forall[x: not lead_to_cycle(x)];
```

Network configuration verification method

The network satisfies the routing policy <=> the specification formulas of this policy are fulfilled for relations modeling the give network



90 Mb of configuration files

Fat Tree Topology

16 routers

757,000 rules

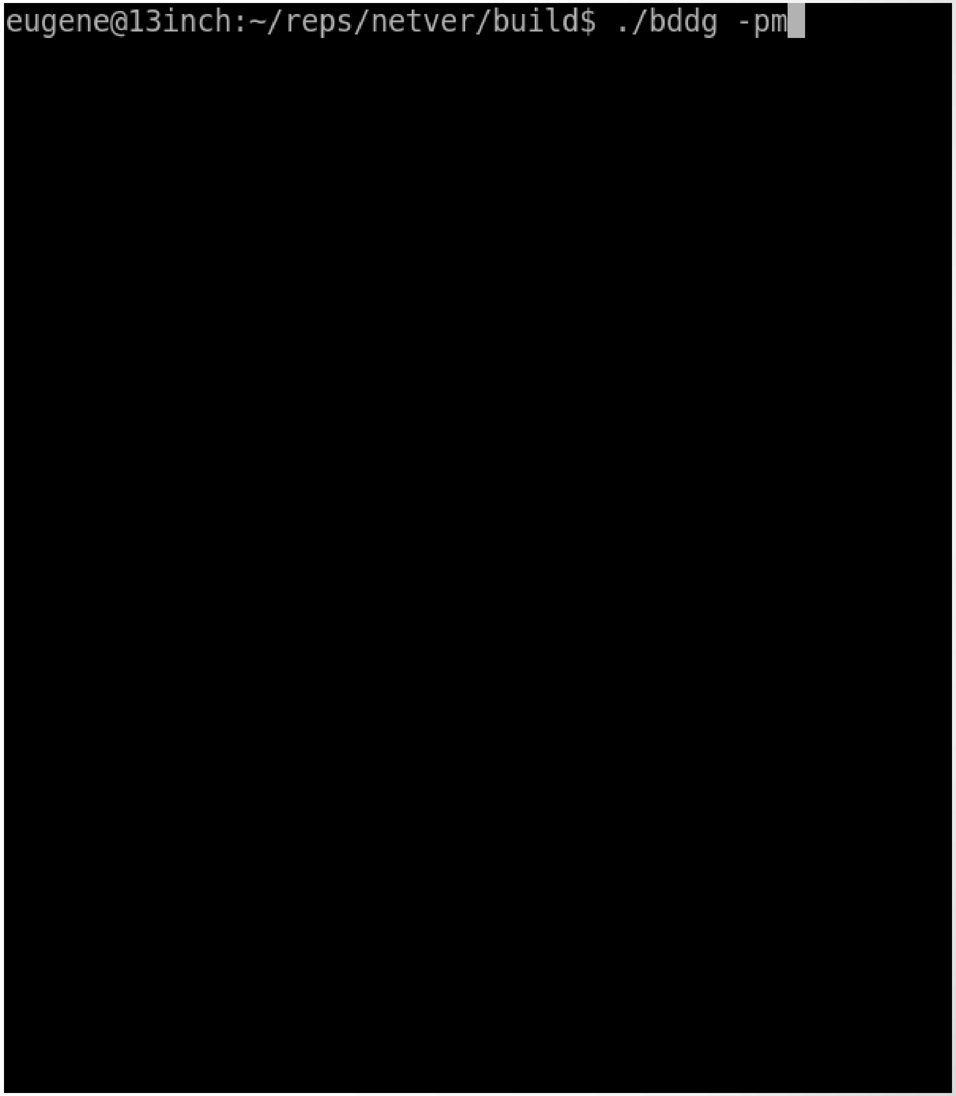
48 tables

Model generation time, seconds			Properties R^{+}_{step}		SDN verification time, seconds				
$One-step\ routing\ relation\ R_{step}$	$Transitive\ closure\ of\ relation\ R_{step}$	Total generation time	The number of OBDD vertices, thous.pcs.	The order of the OBDD path number	State cycles	Topology cycle	Paths longer than one hop	Paths longer than two hops	Paths longer than three hops
1.094	6.294	18.028	642	18	0.043	0.047	0.855	2.013	3.764

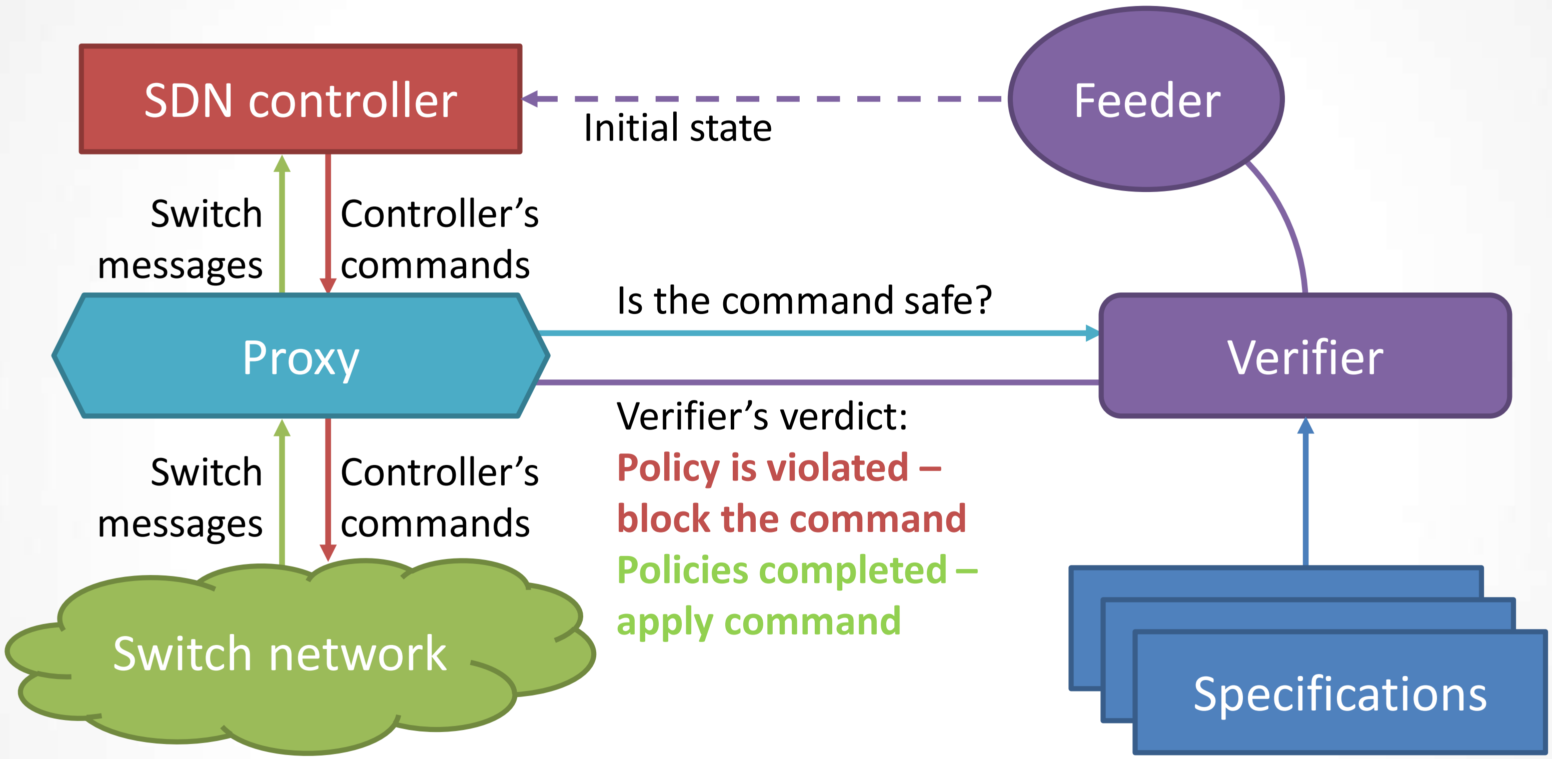
Stanford University Backbone Network

Stanford network verification

Requirement check	Verdict	Time spent (ms)
Model generation	-	3043.687
Transmission cycle	YES	166.191
Black holes	NO	174.845
Route length <= 3 hops	NO	293.522
Route length <= 4 hops	YES	736.015
Rule insert seq. /in parallel	-	100 / 0.3*
Rule remove seq. / in parallel	-	70 / 1*



Deployment Scheme



VERMONT models changes to switch tables after applying a controller command, and blocks the command if it leads to a violation of routing policies

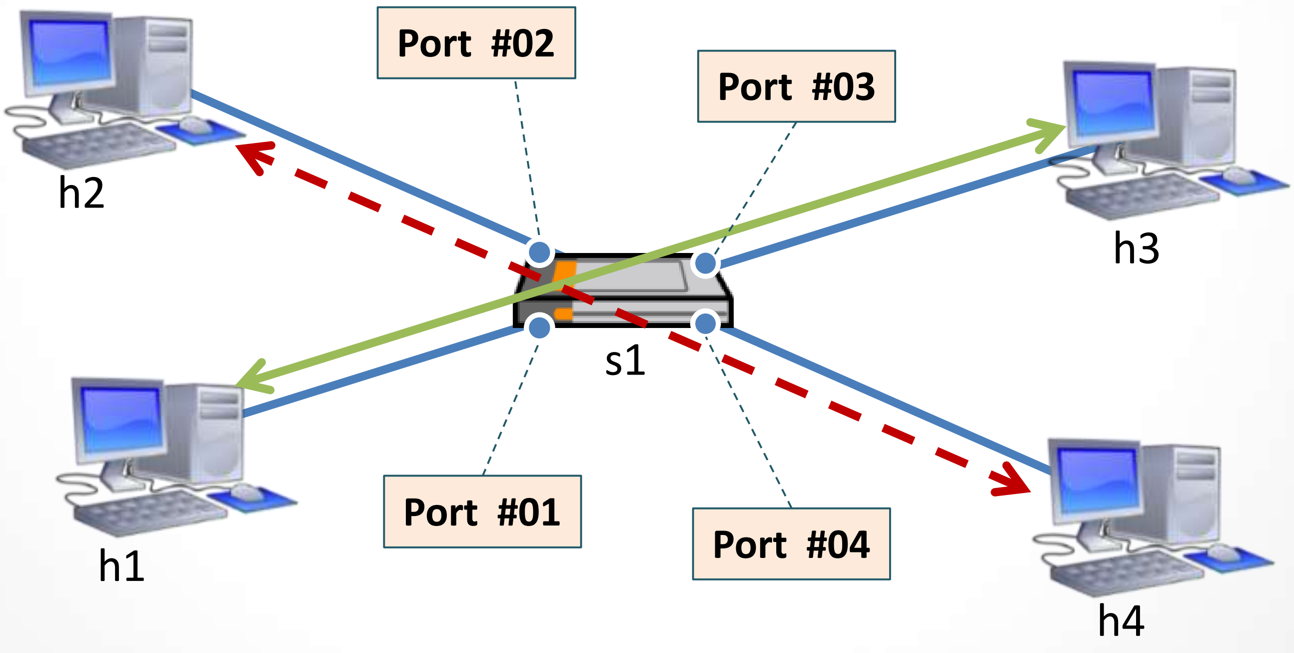
Comparison with other tools

Tool	Model generation (ms)	Model regeneration (ms)	The language power	OpenFlow support
VERMONT 2013	3100	100 - 600	FO[TC]	Full
NetPlumber Stanford University 2013	37000	2 - 1000	CTL	Partial
VeriFlow University of Illinois 2013	>4000	68 - 100	Fixed property set	Minimal
AP Verifier University of Texas 2013	1000	0.1	Fixed property set	Minimal
Anteater University of Illinois 2011	400000	???	Fixed property set	None
FlowChecker University of North Carolina 2010	1200000	350 - 67000	CTL	Full

VERMONT demonstration

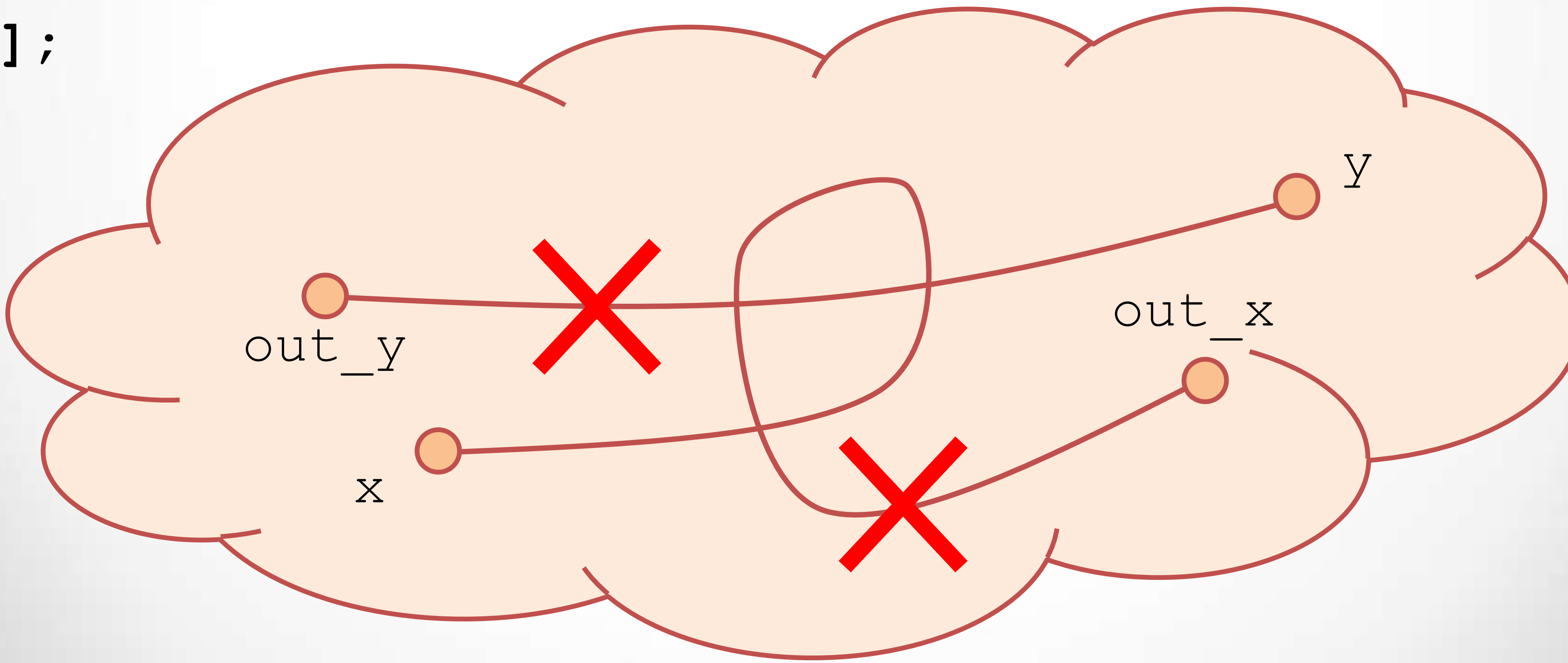
Network disjoint

The switch serves up to two subscribers simultaneously



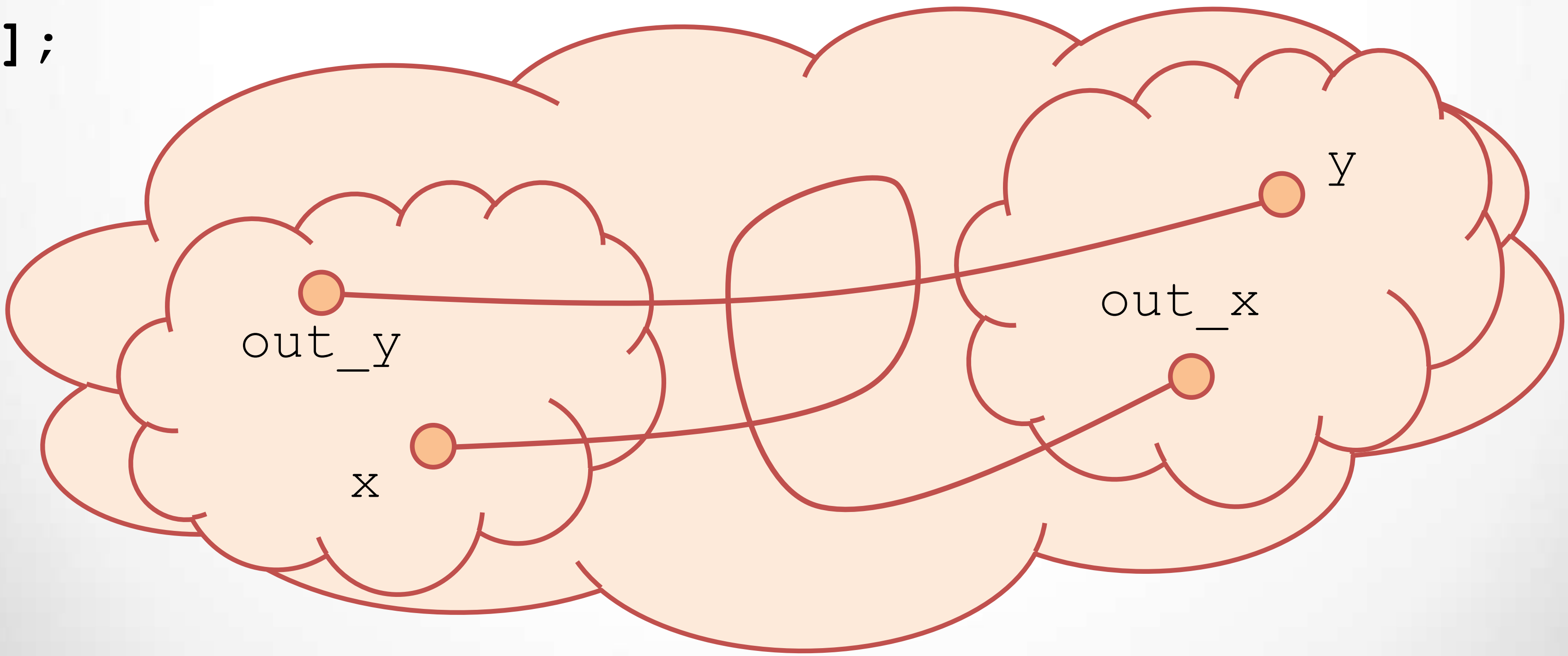
Policy specification

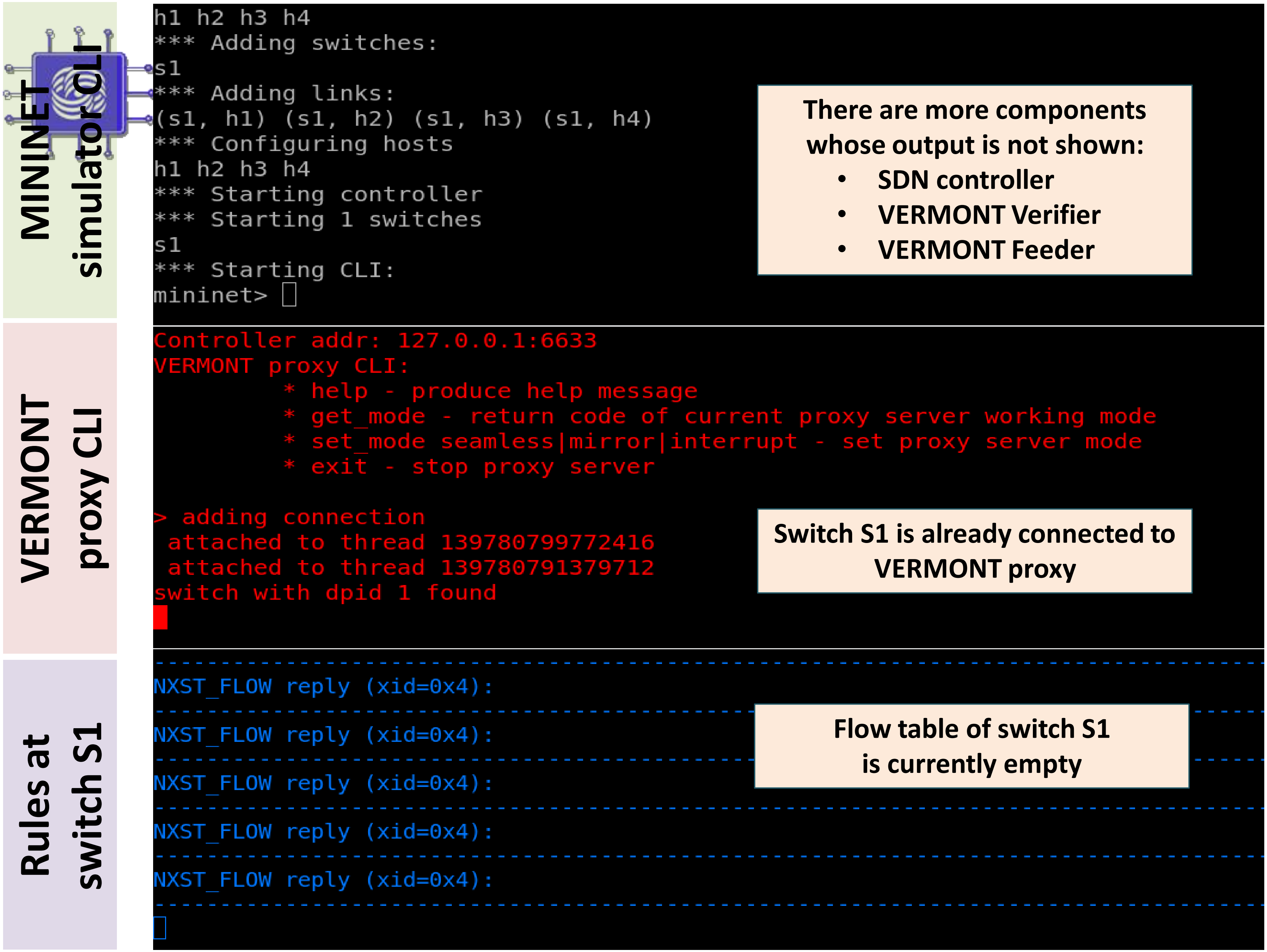
```
main: disjoint() := Forall[x, out_x, y, out_y:
  !R(x, out_x) or !R(y, out_y) or
  x[p] == out_y[p] and out_x[p] == y[p] or
  x[p] == y[p] and out_x[p] == out_y[p]
];
```



Policy specification

```
main: disjoint() := Forall[x, out_x, y, out_y:
  !R(x, out_x) or !R(y, out_y) or
  x[p] == out_y[p] and out_x[p] == y[p] or
  x[p] == y[p] and out_x[p] == out_y[p]
];
```





MININET simulator CLI
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
There are more components whose output is not shown:
• SDN controller
• VERMONT Verifier
• VERMONT Feeder
VERMONT proxy CLI
Controller addr: 127.0.0.1:6633
VERMONT proxy CLI:
* help - produce help message
* get_mode - return code of current proxy server working mode
* set_mode seamless|mirror|interrupt - set proxy server mode
* exit - stop proxy server
> adding connection
attached to thread 139780799772416
attached to thread 139780791379712
switch with dpid 1 found
Switch S1 is already connected to VERMONT proxy
Rules at switch S1
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
Flow table of switch S1 is currently empty

MININET simulator CLI

```
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

VERMONT proxy CLI

```
Controller addr: 127.0.0.1:6633
VERMONT proxy CLI:
        * help - produce help message
        * get_mode - return code of current proxy server working mode
        * set_mode seamless|mirror|interrupt - set proxy server mode
        * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
s
```

Setting VERMONT proxy to interrupt mode

Rules at switch S1

```
------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------------------
```

MININET simulator CLI

```
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Host h1 starts to ping host h3

VERMONT proxy CLI

```
         * get_mode - return code of current proxy server working mode
         * set_mode seamless|mirror|interrupt - set proxy server mode
         * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
set_mode interrupt
connection to verifier established
Connected to verifier (127.0.0.1:3366)
>
```

Controller tries to install the rules to transmit ping packets

Proxy interrupts command from the controller and sends them to Verifier

Rules at switch S1

```
------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------
```

Verifier create an updated model of the data plane and checks them against a set of PFPs

MININET simulator CLI

```
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
```

First packet of the flow uses slow path

VERMONT proxy CLI

```
         * get_mode - return code of current proxy server working mode
         * set_mode seamless|mirror|interrupt - set proxy server mode
         * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
set_mode interrupt
connection to verifier established
Connected to verifier (127.0.0.1:3366)
>
```

Rules at switch S1

```
------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
------------------------------------------------------------------------------
```

Proxy delivers verified commands to the switch

Switch table contains rules to transmit packets between host h1 and host h3

MININET simulator CLI

```
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=85.1 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.132 ms
```

Subsidiary packets use fast path

VERMONT proxy CLI

```
         * get_mode - return code of current proxy server working mode
         * set_mode seamless|mirror|interrupt - set proxy server mode
         * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
set_mode interrupt
connection to verifier established
Connected to verifier (127.0.0.1:3366)
>
```

Rules have an idle timeout and will expire in 5 seconds

Rules at switch S1

```
-----------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
-----------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=0.135s, table=0, n_packets=1, n_bytes=98, idle_tim
eout=5, idle_age=0, priority=0,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl_
dst=00:00:00:00:00:01 actions=output:1
 cookie=0x20000000000000, duration=0.094s, table=0, n_packets=0, n_bytes=0, idle_time
out=5, idle_age=0, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_d
st=00:00:00:00:00:03 actions=output:3
-----------------------------------------------------------------------------
```

MININET simulator CLI

```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=85.1 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.132 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.068 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.068 ms
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.068/17.103/85.177/34.037 ms
mininet>
```

Host h2 starts to ping host h4

VERMONT proxy CLI

```
         * get_mode - return code of current proxy server working mode
         * set_mode seamless|mirror|interrupt - set proxy server mode
         * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
set_mode interrupt
connection to verifier established
Connected to verifier (127.0.0.1:3366)
>
```

Controller tries to install the rules to transmit ping packets

Proxy interrupts command from the controller and sends them to Verifier

Rules at switch S1

```
meout=5, idle_age=0, priority=0,in_port=1,vlan                        :01,dl
_dst=00:00:00:00:00:03 actions=output:3
------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=4.175s, table=0, n_packets=5, n_bytes=490, idle_ti
meout=5, idle_age=0, priority=0,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl
_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x20000000000000, duration=4.134s, table=0, n_packets=4, n_bytes=392, idle_ti
meout=5, idle_age=0, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl
_dst=00:00:00:00:00:03 actions=output:3
------------------------------------------------------------------------------
```

Verifier create an updated model of the data plane and checks them against a set of PFPs

MININET simulator CLI

```
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=85.1 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.132 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.068 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.068 ms
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.068/17.103/85.177/34.037 ms
mininet> h2 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
```

VERMONT proxy CLI

```
         * get_mode - return code of current proxy server working mode
         * set_mode seamless|mirror|interrupt - set proxy server mode
         * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
set_mode interrupt
connection to verifier established
Connected to verifier (127.0.0.1:3366)
>
```

Proxy drops unsafe commands and notifies the controller

Rules at switch S1

```
meout=5, idle_age=0, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl
_dst=00:00:00:00:00:03 actions=output:3
------------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=6.197s, table=0, n_packets=6, n_bytes=532, idle_ti
meout=5, idle_age=1, priority=0,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl
_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x20000000000000, duration=6.156s, table=0, n_packets=5, n_bytes=434, idle_ti
meout=5, idle_age=1, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl
_dst=00:00:00:00:00:03 actions=output:3
------------------------------------------------------------------------------------
```

MININET simulator CLI

```
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=85.1 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.132 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.068 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.068 ms
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.068/17.103/85.177/34.037 ms
mininet> h2 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
```

VERMONT proxy CLI

```
         * exit - stop proxy server

> adding connection
 attached to thread 139780799772416
 attached to thread 139780791379712
switch with dpid 1 found
set_mode interrupt
connection to verifier established
Connected to verifier (127.0.0.1:3366)
> command blocked
command blocked
```

Why does ping work?

Packets are delivered through the control plane

We can block them!
But do we really want to?

Rules at switch S1

```
meout=5, idle_age=0, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl
_dst=00:00:00:00:00:03 actions=output:3
---------------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=6.197s, table=0, n_packets=6, n_bytes=532, idle_ti
meout=5, idle_age=1, priority=0,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl
_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x20000000000000, duration=6.156s, table=0, n_packets=5, n_bytes=434, idle_ti
meout=5, idle_age=1, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl
_dst=00:00:00:00:00:03 actions=output:3
---------------------------------------------------------------------------------------
```

MININET simulator CLI

```
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% pack
rtt min/avg/max/mdev = 0.068/17.103/85.177/
mininet> h2 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=91.5 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=45.1 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=46.5 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=47.5 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=45.5 ms
```

Packets start to use fast path

VERMONT proxy CLI

```
connection to verifier established
Connected to verifier (127.0.0.1:3366)
> command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
```

Old rules have been expired

Controller is allowed to install new rules

Rules at switch S1

```
imeout=5, idle_age=5, priority=0,in_port=1,vla                                   0:01,d
l_dst=00:00:00:00:00:03 actions=output:3
---------------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=0.629s, tab                                   le_tim
eout=5, idle_age=0, priority=0,in_port=4,vlan_tci=0x0000,dl_src=00:00:00:00:00:04,dl_
dst=00:00:00:00:00:02 actions=output:2
 cookie=0x20000000000000, duration=0.633s, table=0, n_packets=1, n_bytes=98, idle_tim
eout=5, idle_age=0, priority=0,in_port=2,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl_
dst=00:00:00:00:00:04 actions=output:4
---------------------------------------------------------------------------------------
```

Switch table contains rules to transmit packets between host h2 and host h4

MININET simulator CLI

```
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=45.5 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=0.159 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=0.037 ms
64 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=0.055 ms
^C
--- 10.0.0.4 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9007ms
rtt min/avg/max/mdev = 0.037/27.672/91.542/30.443 ms

mininet>
```

Packets use slow path

VERMONT proxy CLI

```
connection to verifier established
Connected to verifier (127.0.0.1:3366)
> command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
```

Rules to transmit packets from host h1 to host h3 violate forwarding policy

Rules at switch S1

```
meout=5, idle_age=0, priority=0,in_port=2,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl
_dst=00:00:00:00:00:04 actions=output:4
-----------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=4.671s, table=0, n_packets=6, n_bytes=532, idle_ti
meout=5, idle_age=0, priority=0,in_port=4,vlan_tci=0x0000,dl_src=00:00:00:00:00:04,dl
_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x20000000000000, duration=4.675s, table=0, n_packets=6, n_bytes=532, idle_ti
meout=5, idle_age=0, priority=0,in_port=2,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl
_dst=00:00:00:00:00:04 actions=output:4
-----------------------------------------------------------------------------------
```

MININET simulator CLI

```
--- 10.0.0.4 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9007ms
rtt min/avg/max/mdev = 0.037/27.672/91.542/

mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=45.9 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=50.5 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=47.1 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=41.8 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=45.8 ms
```

Packets start to use fast path

VERMONT proxy CLI

```
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
command blocked
```

Rules at switch S1

```
meout=5, idle_age=4, priority=0,in_port=2,vlan                                          :02,dl
_dst=00:00:00:00:00:04 actions=output:4
--------------------------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x20000000000000, duration=0.570s, table=0, n_packets=1, n_bytes=98, idle_tim
eout=5, idle_age=0, priority=0,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl_
dst=00:00:00:00:00:01 actions=output:1
 cookie=0x20000000000000, duration=0.573s, table=0, n_packets=1, n_bytes=98, idle_tim
eout=5, idle_age=0, priority=0,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_
dst=00:00:00:00:00:03 actions=output:3
--------------------------------------------------------------------------------------------
```

Switch table contains rules to transmit packets between host h1 and host h3

Network configuration consistent update

E.P. Stepanov

Configuration consistent update problem

- ***Network configuration*** C defines binding of rules to switches and network topology
- Relations IN_C and R_C for configuration C determine the $\boldsymbol{path}$ of packet transmission

$$s_0 \in IN_C$$
$$(s_i, s_{i+1}) \in R_C$$
$$path = s_0, s_1, \dots, s_i, s_{i+1}, \dots$$

- $Path(C)$ – the set of all packet transmission paths for configuration C

Configuration consistent update problem

- com – ***network reconfiguration command***

  add, delete or modify a routing rule
- $com(C)$ - configuration, obtained by **applying the *com* command** to the configuration C
- If $\alpha = com_1, \dots, com_k$, then

$$\alpha(C) = com_k(\dots, com_1(C) \dots)$$

Configuration consistent update problem

- **The partial order** $\prec$ is introduced on the set of reconfiguration commands Com :

  if $com' \prec com''$, then com'' is applying only after com' is finished to apply

- ***Reconfiguration package*** - a set of reconfiguration commands, supplemented by a partial order relation $(Com, \prec)$.

Configuration consistent update problem

Input:

- Initial network configuration C_0
- Correctness and safety requirements

  post-condition Φ, ***invariant*** Ψ.

Output:

- Find such a reconfiguration package $(U, \prec)$, that for each any linearization of which α_U it is satisfied:

1. $\alpha_U(C_0) \vDash \Phi$

2. $\forall \alpha' \, (\, \alpha_U = \alpha' \alpha'' \; \Rightarrow \alpha'(C_0) \vDash \Psi \,)$

1. Synthesis of a given network configuration

Generate such configuration C, that satisfies the given post-condition Φ

A. Noyes, T. Warszawski, P. Cernyand, N. Foster. *Toward Synthesis of Network Updates.* 2-nd Workshop on Synthesis (CAV-2013), 2013, Saint Petersburg, Russia

2. Global consistent network update

- Post-condition $\Phi(\mathrm{X})$: $X = C$
- Invariant $\Psi(\mathrm{X})$: $\forall\ s\colon IN(s) \rightarrow$
  $\Big(\big(Path(X,s) \subseteq\ Path(C)\big) \vee\ \big(Path(X,s)$

M. Reitblatt, N. Foster, J. Rexford, D. Walker. *Consistent updates for software-defined networks: change you can believe in! HotNets, v. 7, 2011.*

3. Local consistent network update

- Post-condition Φ(X):

$$Path(X) = (Path(C_0) \setminus \{path_0\}) \cup \{path_1\}$$

- Invariant Ψ(X) :

$$Path(C_0) \setminus \{path_0\} \subseteq Path(X) \subseteq Path(C_0) \cup \{path_1\}$$

S. Raza, Y. Zhu, C.-N. Chu S. Raza, Y. Zhu, C.-N. Chuah.
Graceful network state migrations.
IEEE/ACM Transactions on Networking, 2011.

4. Network recovery

- Post-condition $\Phi(X)$:

$$X = C$$

- Invariant $\Psi(X)$:

$$\big(Path(C_0) \cap Path(C) \subseteq Path(X)\big) \wedge$$
$$\big(Path(X) \subseteq Path(C_0) \cup Path(C)\big)$$

- Configuration C was translated to the configuration C_0 as a result of deleting part of the rules
- The goal is to restore C from the configuration C_0

5. Flow table optimization

- Post-condition $\Phi(X)$:

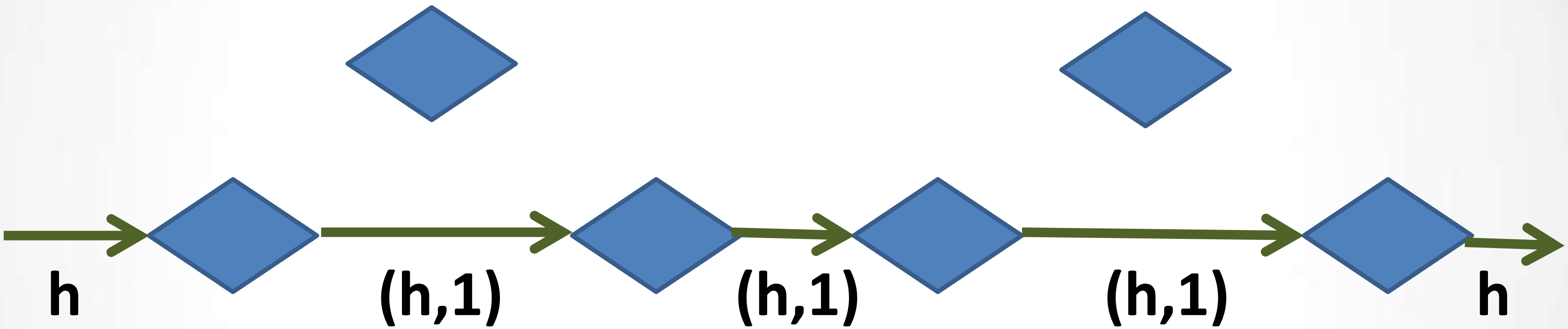
$$(Path(X) = Path(C)) \wedge$$
$$\forall Y\left(\left(Path(X) = Path(C)\right) \rightarrow \left(f(X) \leq f(Y)\right)\right)$$

- Invariant $\Psi(X)$:

$$Path(X) = Path(C)$$

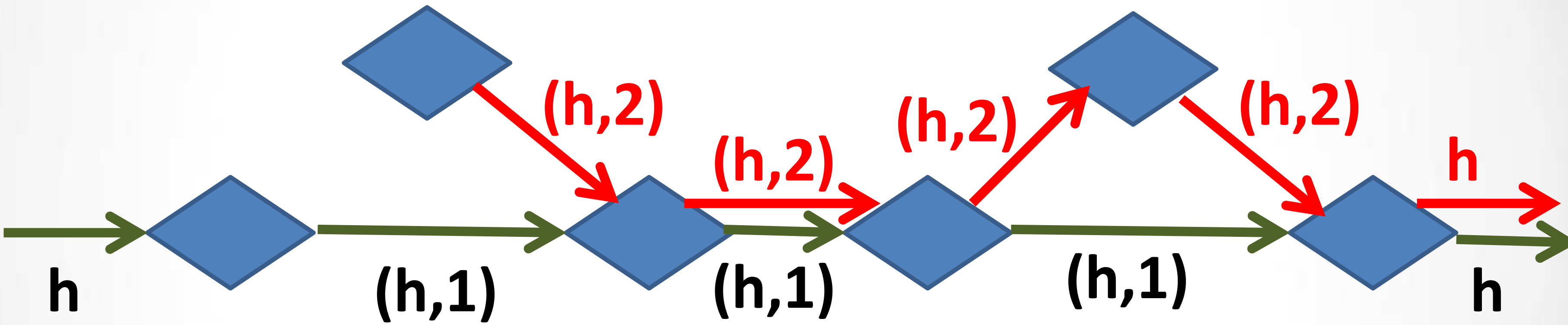
- K. Kogan, S. Nikolenko, W. Culhane, P. Eugster, E. Ruan. *Towards efficient implementation of packet classifiers.* Proc. of the 2-d Workshop on Hot Topics in SDN, 2013.

Network Update Using Tags



C_0 sets the initial flow path
C_1 adds two intermediate nodes to it

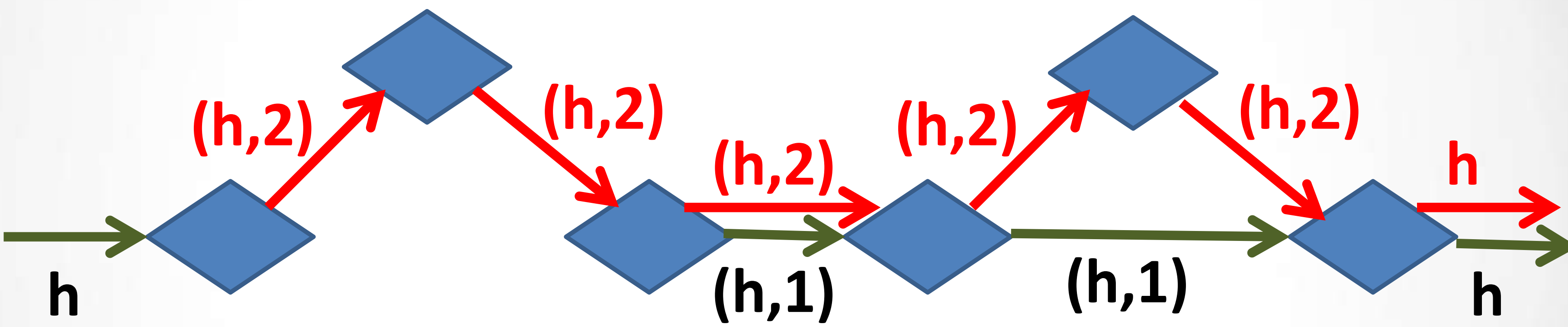
Network Update Using Tags



3-phase update algorithm

1: adding new rules

Network Update Using Tags

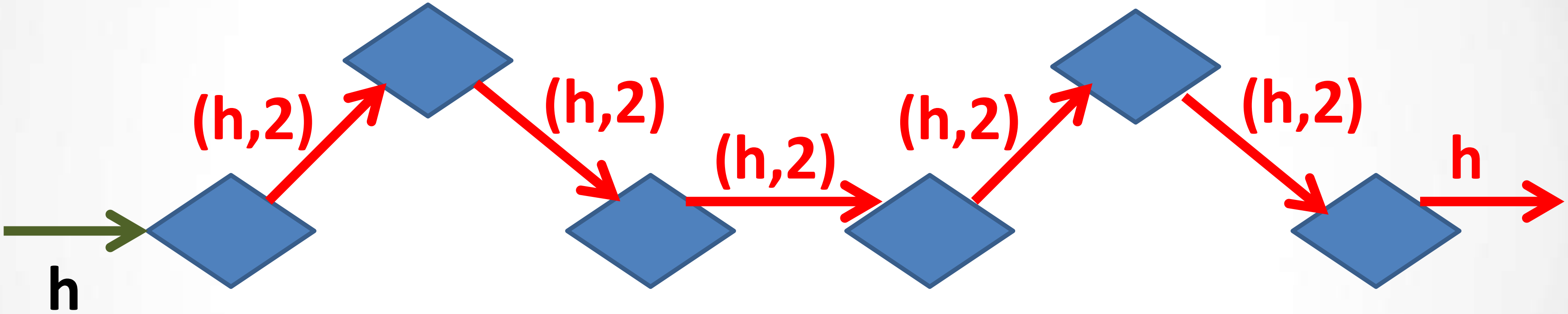


3-phase update algorithm

1: adding new rules

2: switching route

Network Update Using Tags



3-phase update algorithm

1: adding new rules

2: switching routes

3: deleting obsolete rules

Network Update Using Tags

- Packet headers must have an additional field that will be used exclusively during configuration updates
- In special cases the update problem can be solved without the use of tagging, in the general case this problem is unsolvable