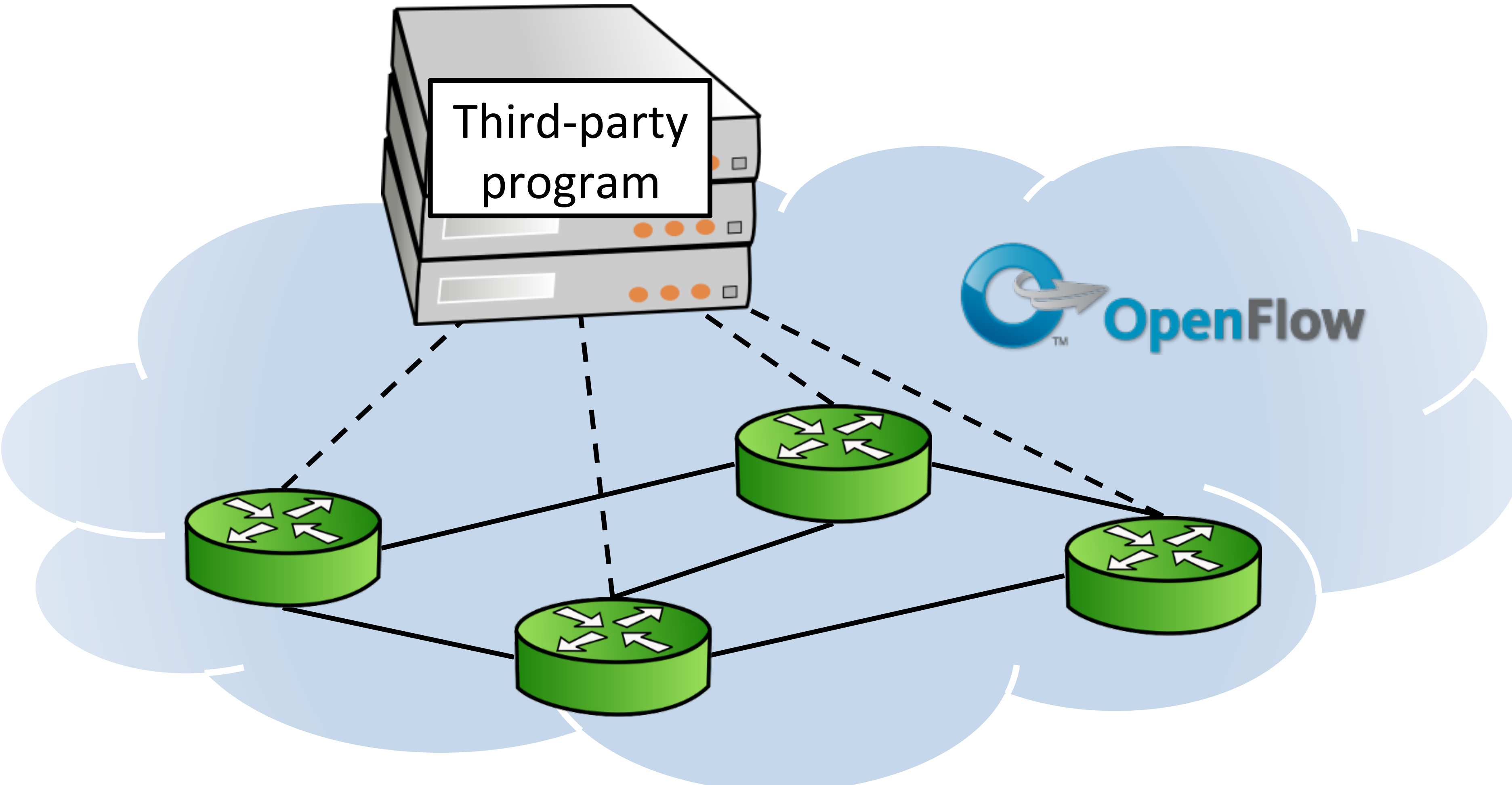
A NICE way to test OpenFlow Applications

Marco Canini, Daniele Venzano,
Peter Perešíni, Dejan Kostić, Jennifer Rexford†

EPFL †Princeton University

25 Apr '12

Software-Defined Networking (SDN)



Enables new functionality through programmability …

… at the risk of bugs

Network Operating System

A fatal exception has occurred at 10.3.0.5/C0011E36 in OF(01) + 00010E36. The current OpenFlow application will be terminated.

* Press any key to terminate the current OpenFlow application
* Press CTRL+ALT+DEL again to restart your network. Your users will lose all network connectivity.

Press any key to continue

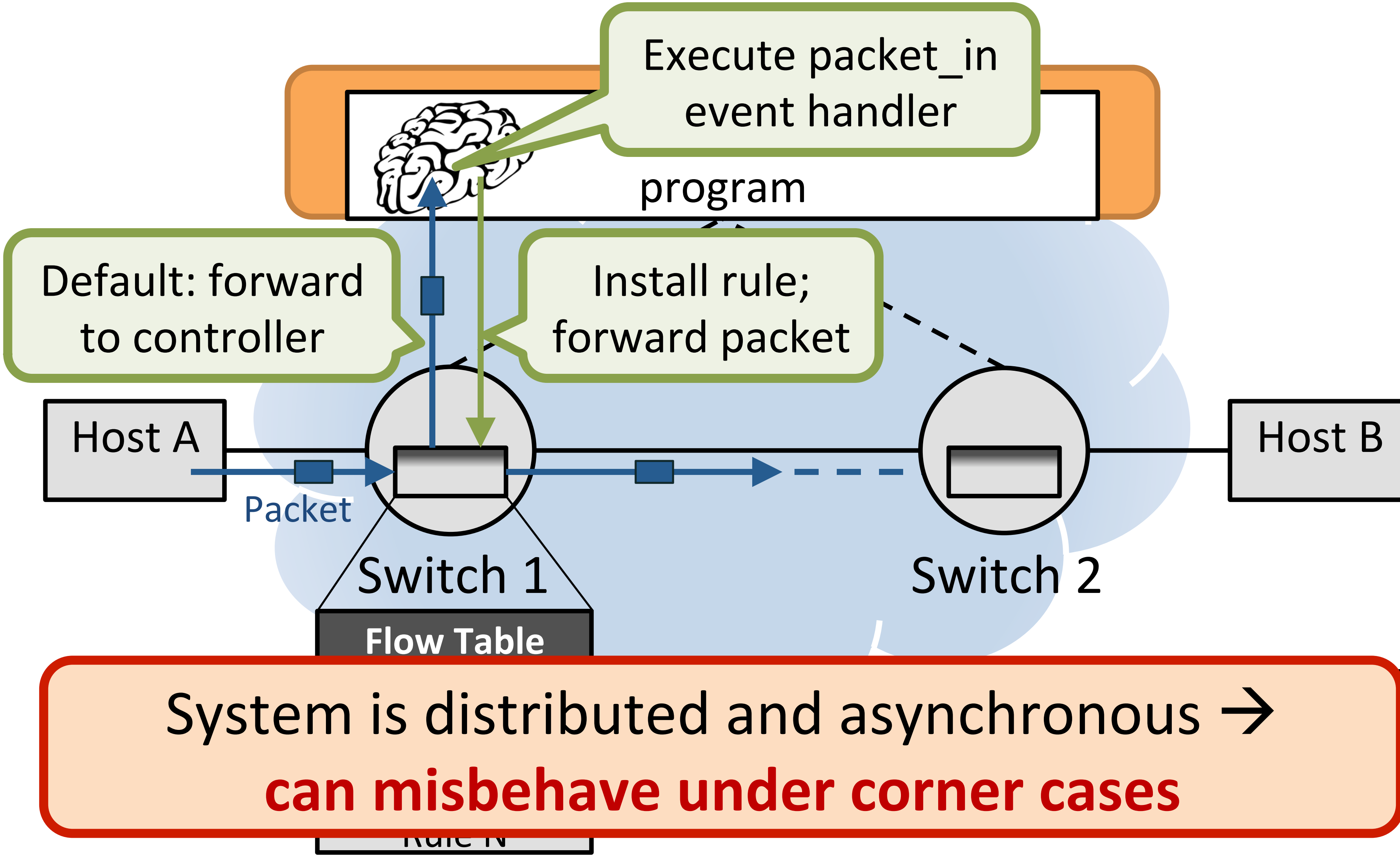
Software Faults

- Will make communication unreliable

- Major hurdle for success of SDN

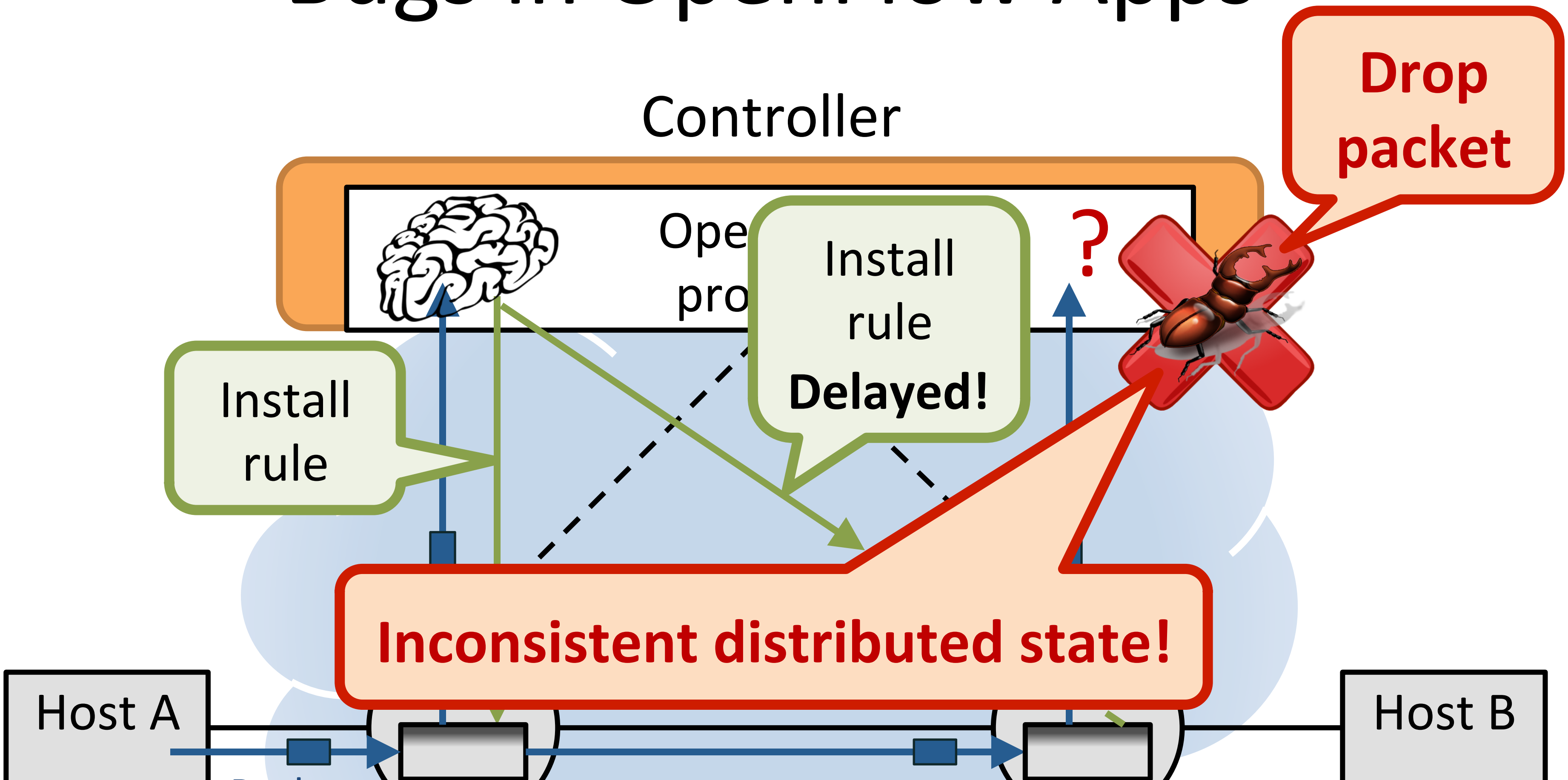
We need effective ways to test SDN networks
This talk: automatically testing OpenFlow Apps

Quick OpenFlow 101



System is distributed and asynchronous →
can misbehave under corner cases

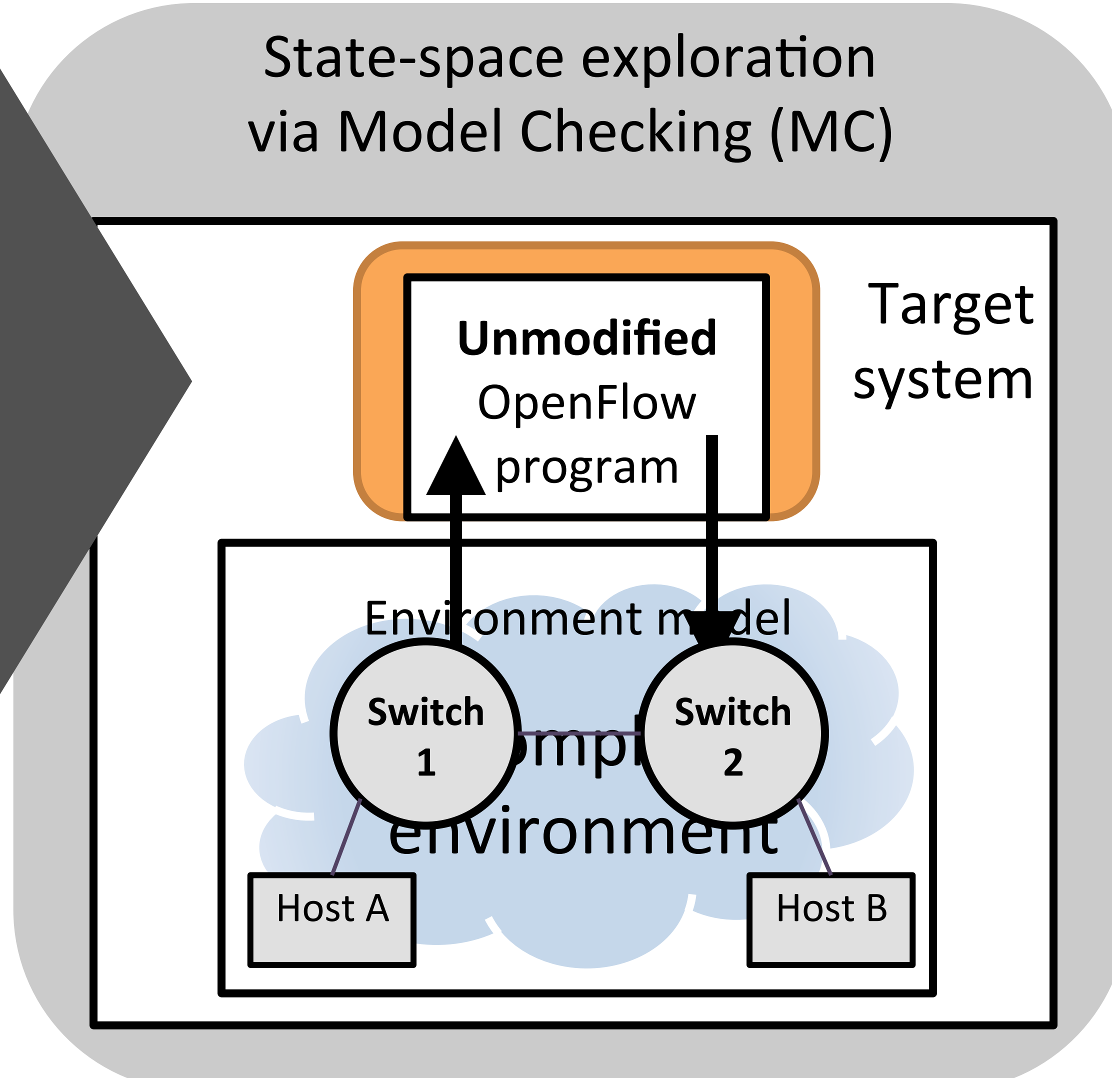
Bugs in OpenFlow Apps



Goal: systematically test possible behaviors to detect bugs

Systematically Testing OpenFlow Apps

- Carefully-crafted streams of packets
- Many orderings of packet arrivals and events



Scalability Challenges

Data-plane driven

Huge space of possible **packets**

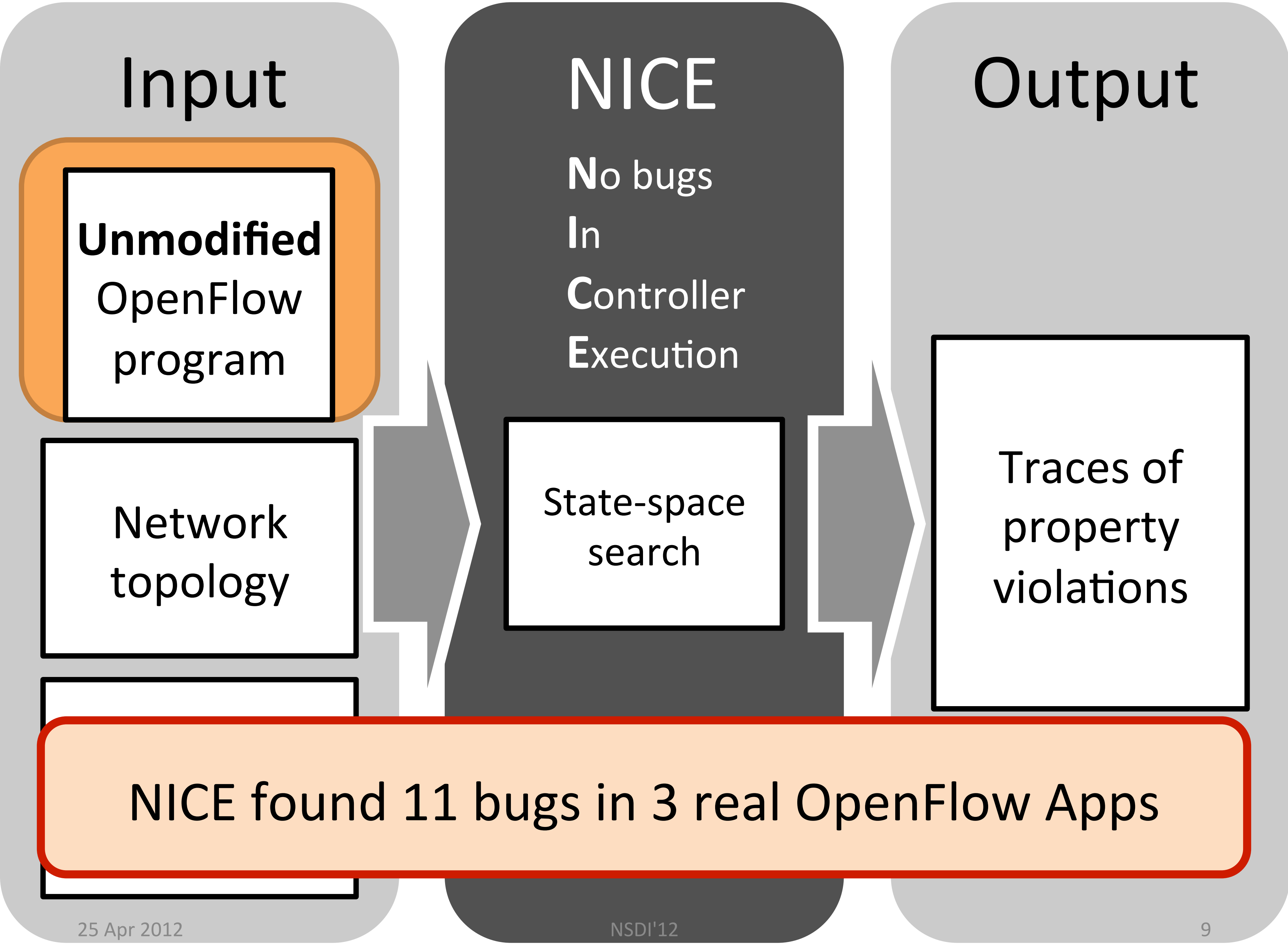
Equivalence classes of packets

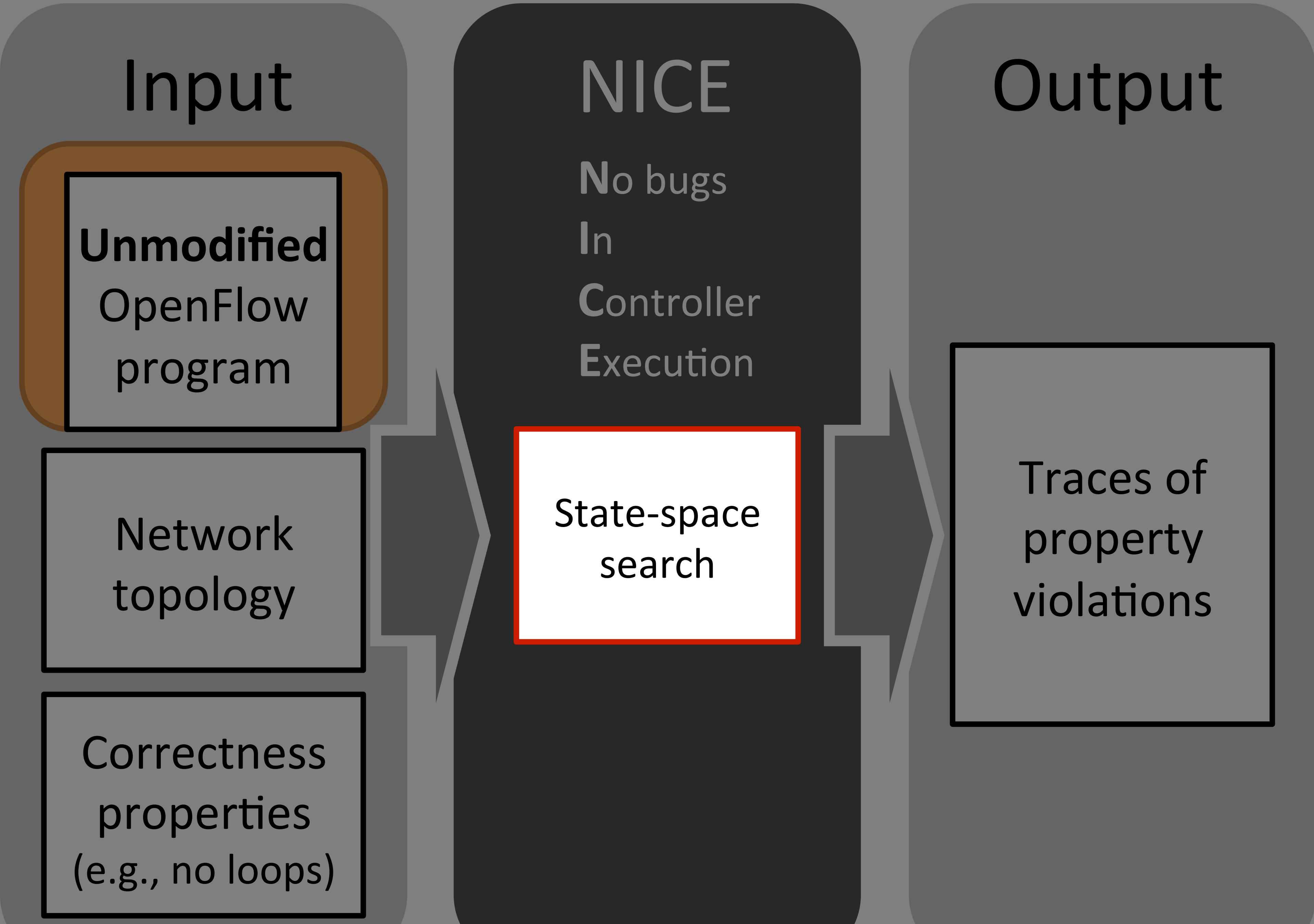
Complex network behavior

Huge space of possible **event orderings**

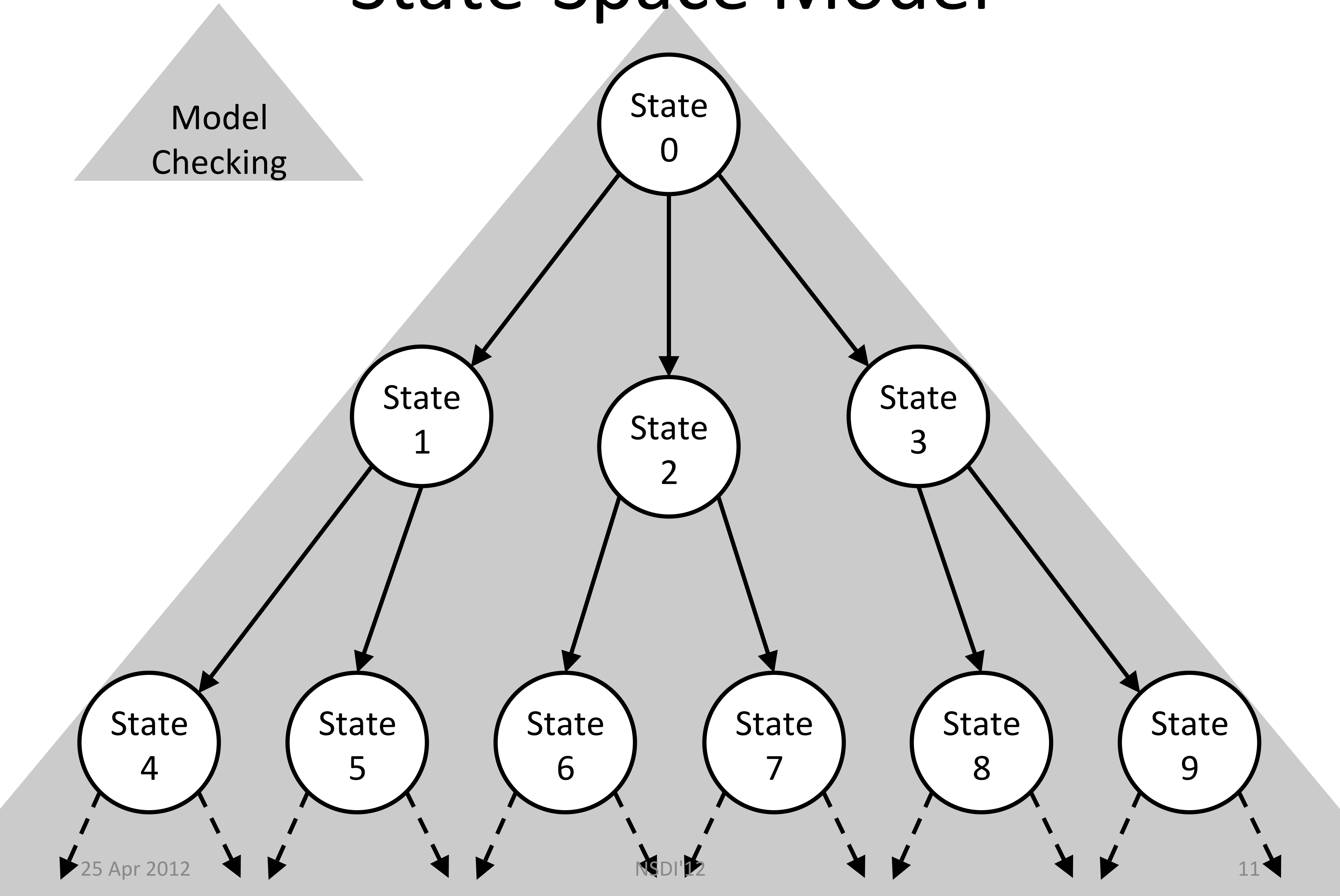
Domain-specific search strategies

Enumerating all inputs and event orderings is intractable


Input
Unmodified
OpenFlow
program
Network
topology
NICE
No bugs
In
Controller
Execution
State-space
search
Output
Traces of
property
violations
NICE found 11 bugs in 3 real OpenFlow Apps


Input
Unmodified OpenFlow program
Network topology
Correctness properties (e.g., no loops)
NICE
No bugs
In
Controller
Execution
State-space search
Output
Traces of property violations

State-Space Model



System State

State

Controller (global variables)

Environment:

Switches (flow table, OpenFlow agent)

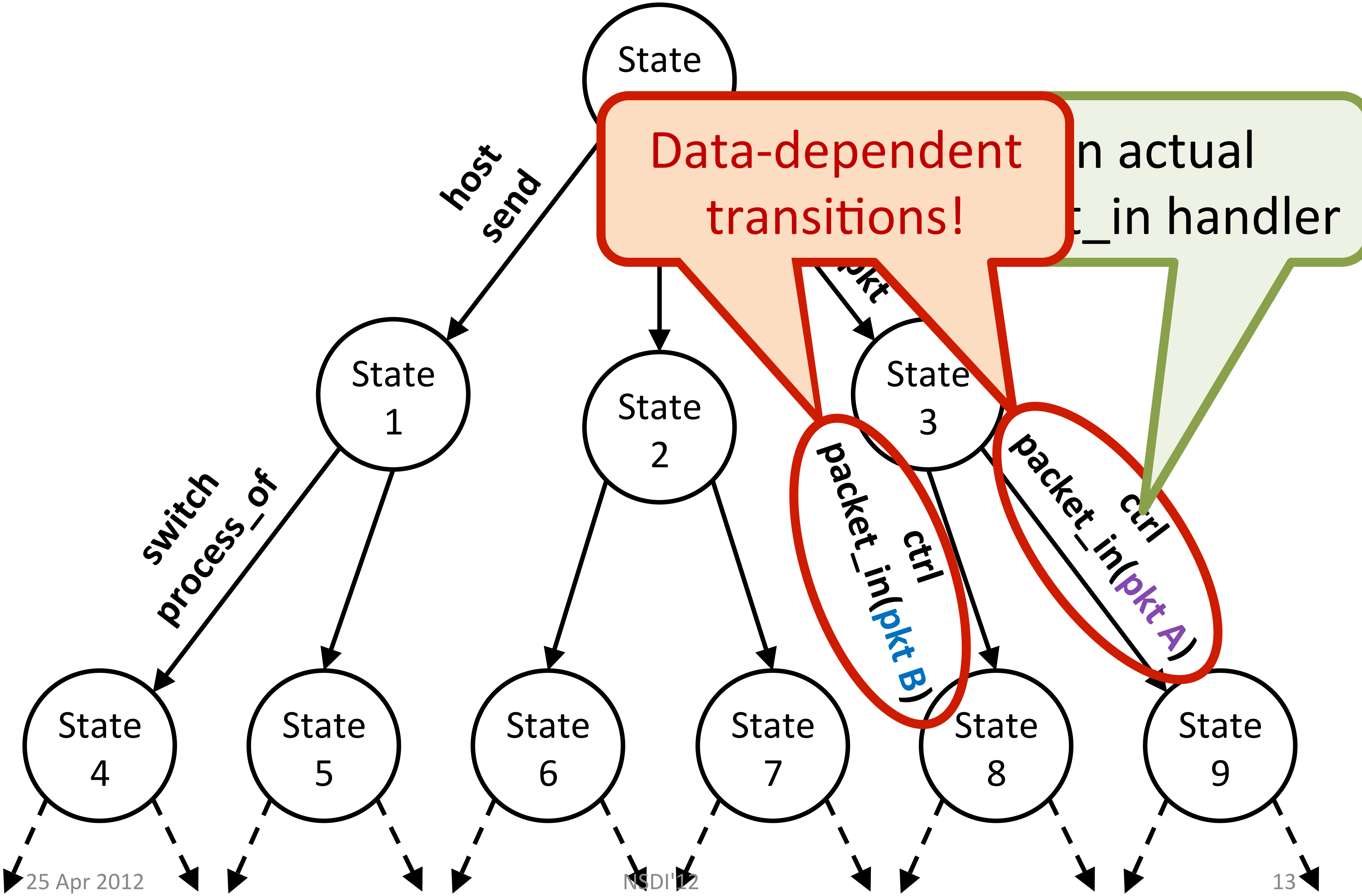
Simplified switch model

End-hosts (network stack)

Simple clients/servers

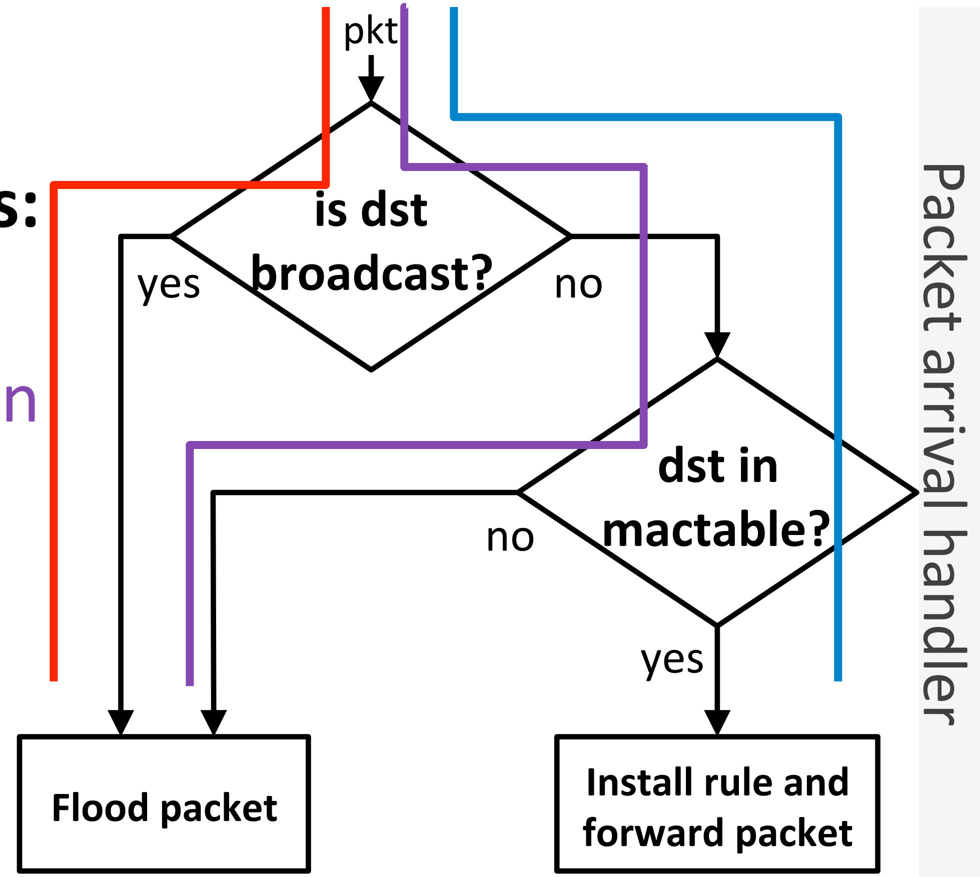
Communication channels (in-flight pkts)

Transition System



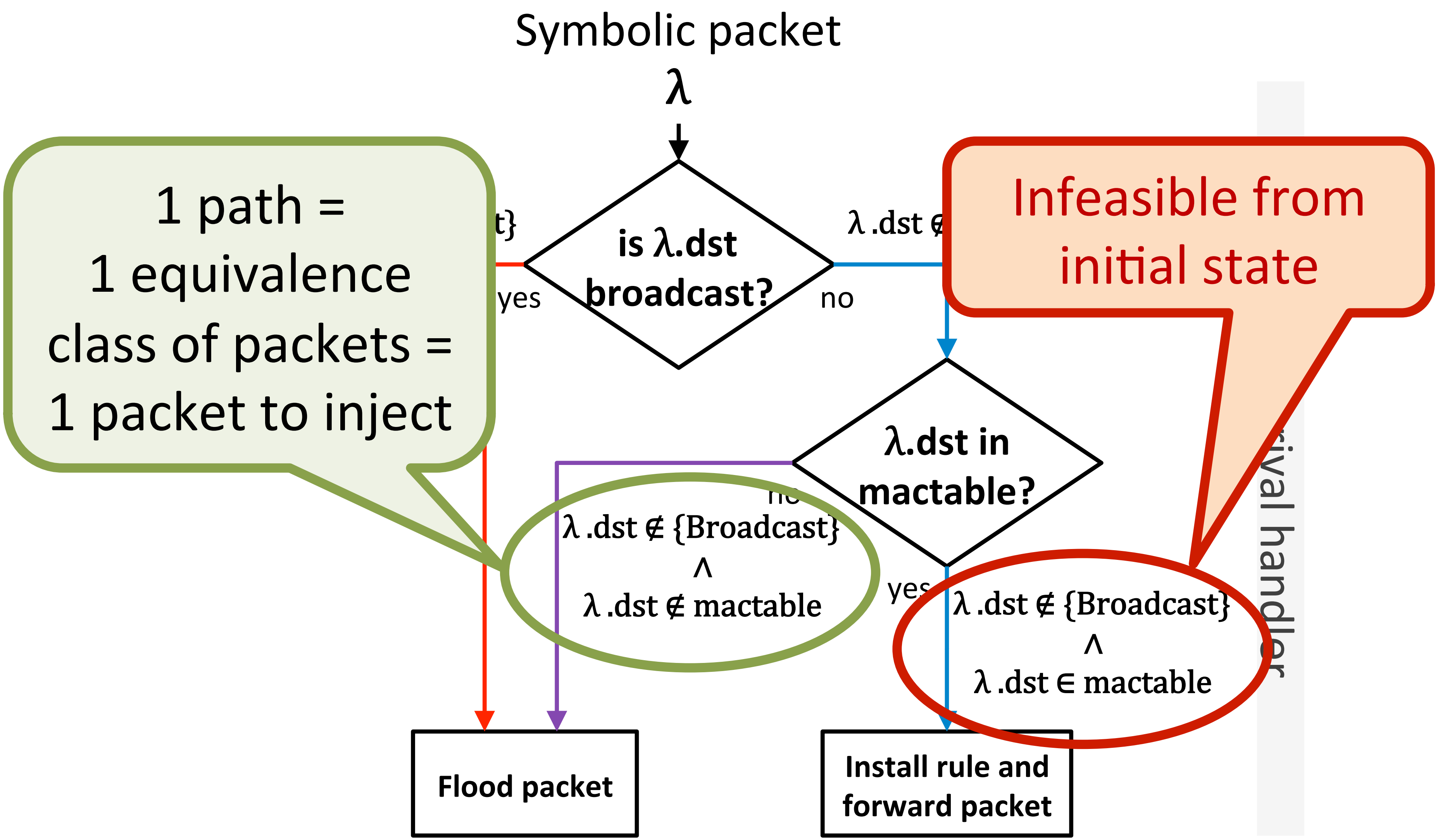
Combating Huge Space of Packets

Equivalence classes of packets:
1. Broadcast destination
2. Unknown unicast destination
3. Known unicast destination

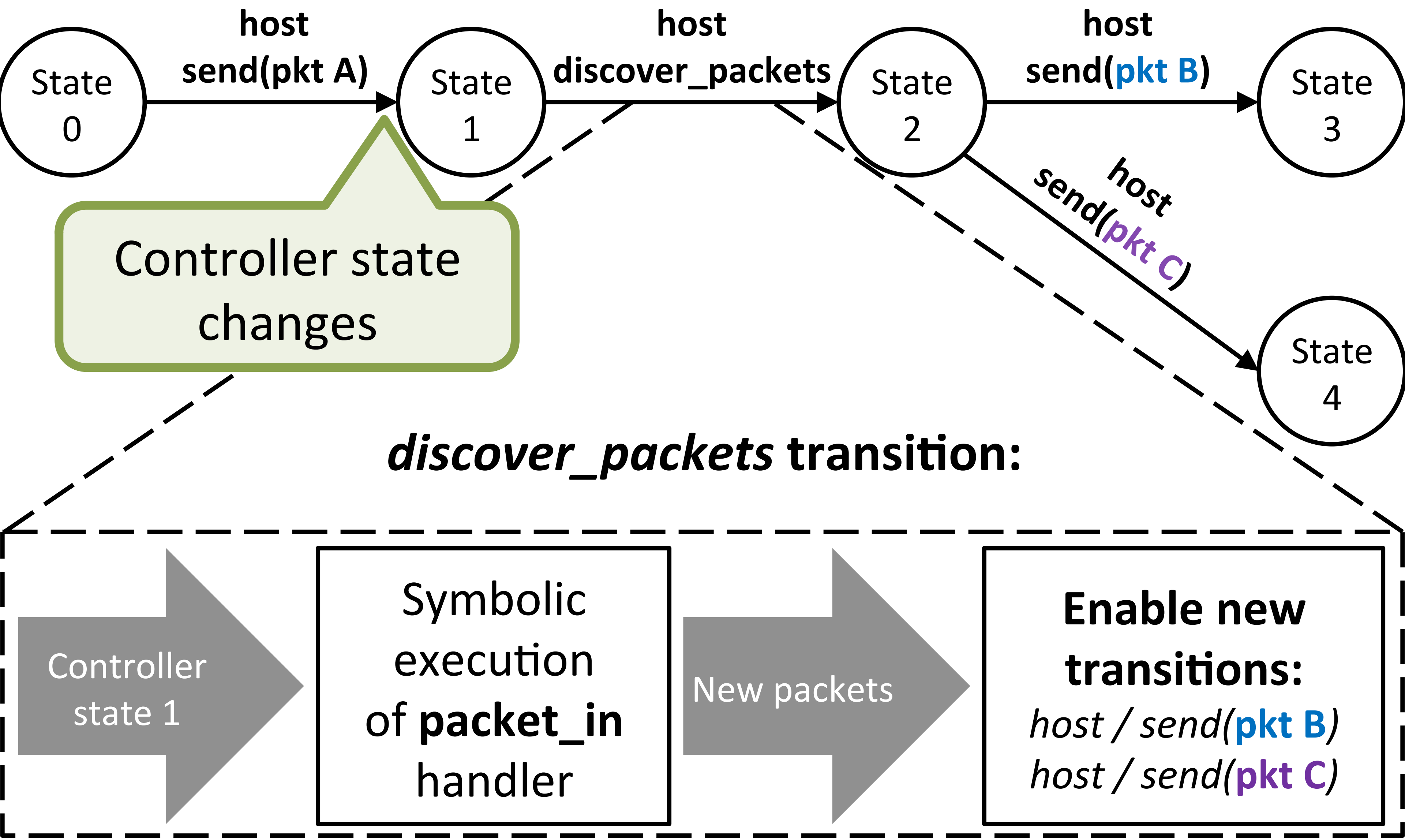


Code itself reveals equivalence classes of packets

Code Analysis: Symbolic Execution (SE)

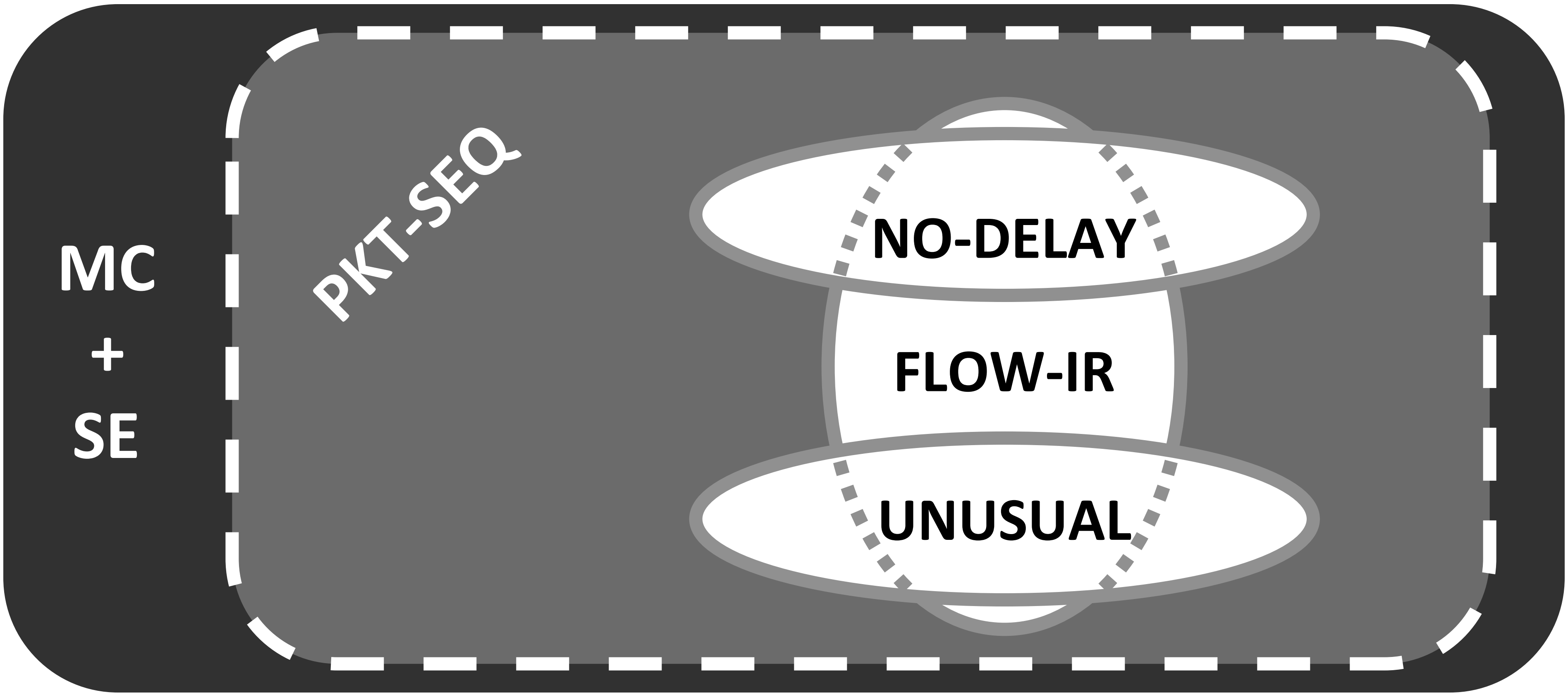


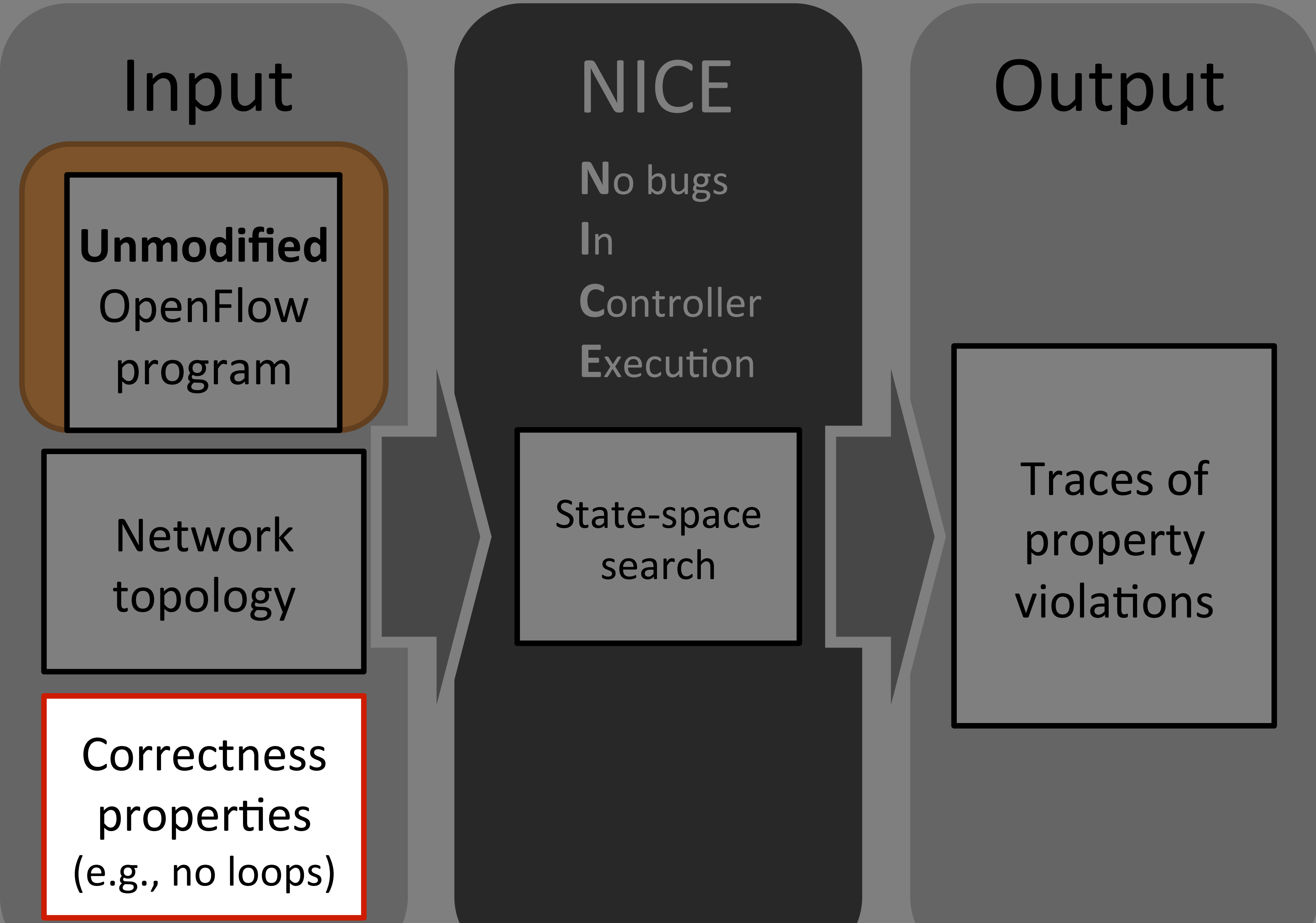
Combining SE with Model Checking



Combating Huge Space of Orderings

OpenFlow-specific search strategies for up to 20x state-space reduction:

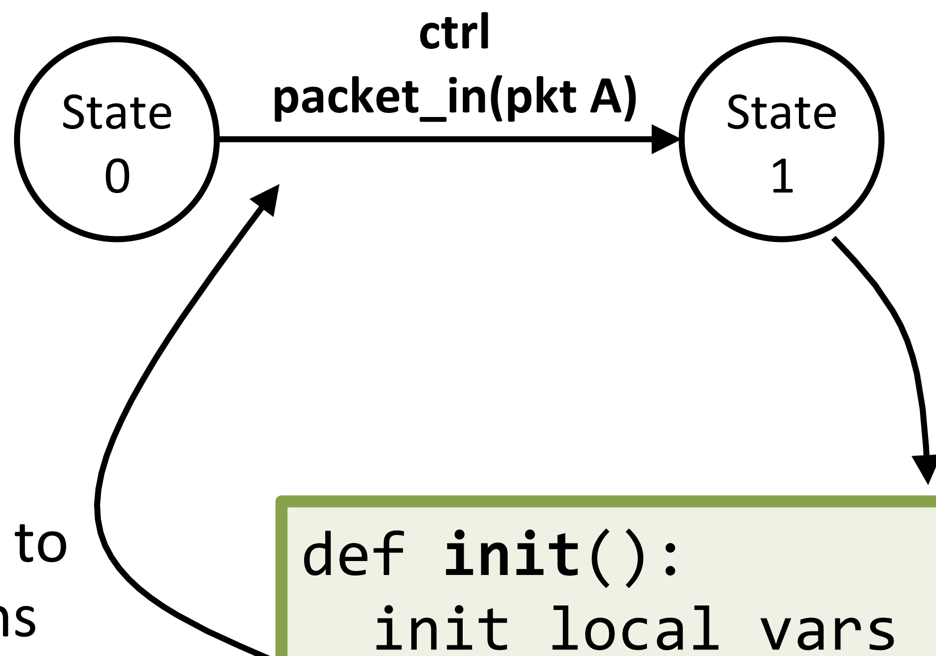



Input
Unmodified
OpenFlow
program
Network
topology
Correctness
properties
(e.g., no loops)
NICE
No bugs
In
Controller
Execution
State-space
search
Output
Traces of
property
violations

Specifying App Correctness

- Library of **common properties**
  - No forwarding loops
  - No black holes
  - Direct paths (no unnecessary flooding)
  - Etc…
- Correctness is app-specific in nature

API to Define App-Specific Properties



Execute after transitions

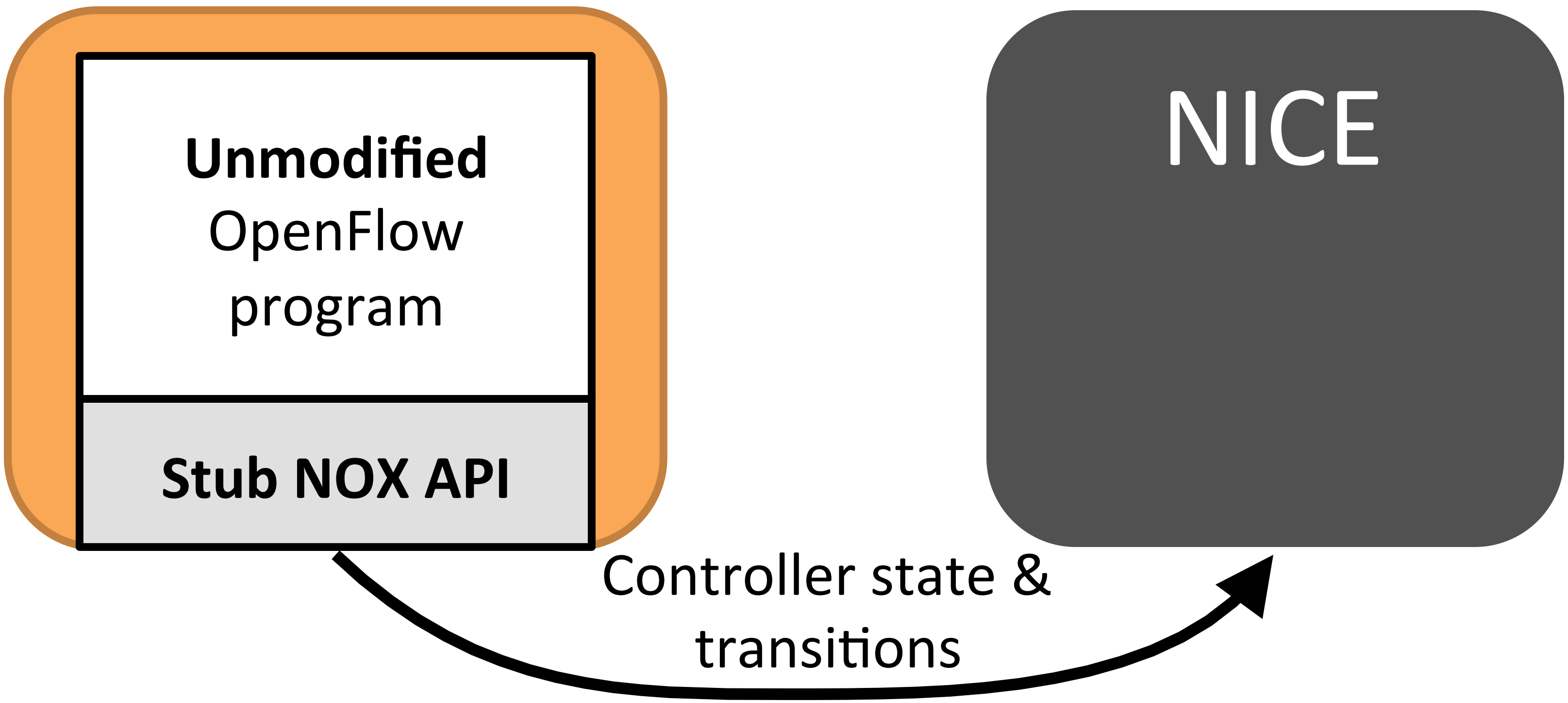
Register callbacks to observe transitions

```
def init():
  init local vars
  register("packet_in")

def on_packet_in():
  check system-wide state
```

Prototype Implementation

- Built a NICE prototype in Python
- Target the Python API of NOX



Experiences

- Tested 3 unmodified NOX OpenFlow Apps
  - MAC-learning switch
  - LB: Web server load balancer [Wang et al., HotICE'11]
  - TE: Energy-aware traffic engineering [CoNEXT'11]
- Setup
  - Iterated with 1, 2 or 3-switch topologies; 1,2,… pkts
  - App-specific properties
    - LB: All packets of same request go to same server replica
    - TE: Use appropriate path based on network load

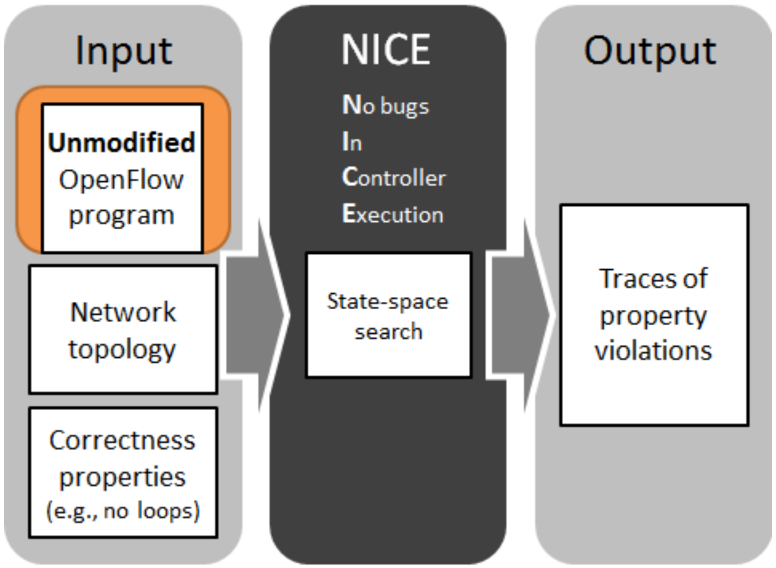
Results

- NICE found 11 property violations → bugs
  - Few secs to find 1st violation of each bug (max 30m)
  - Few simple mistakes (not freeing buffered packets)
  - **3 insidious bugs** due to network race conditions
    - NICE makes corner cases as likely as normal cases

Thank you! Questions?

Conclusions

NICE automates the testing of OpenFlow Apps



http://code.google.com/p/nice-of/

- Explores state-space efficiently
- Tests unmodified NOX applications
- Helps to specify correctness
- Finds bugs in real applications

SDN: a new role for software tool chains to **make networks more dependable**.
NICE is a step in this direction!

Backup slides

Related Work (1/2)

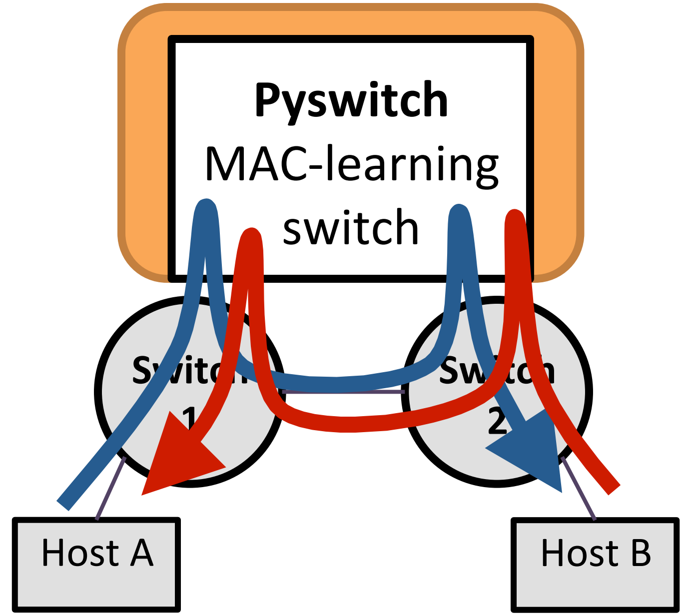
- Model Checking
  - SPIN [Holzmann'04], Verisoft [Godefroid'97], JPF [Visser'03]
  - Musuvathi'04, MaceMC [Killian'07], MODIST [Yang'09]
- Symbolic Execution
  - DART [Godefroid'05], Klee [Cadar'08], Cloud9 [Bucur'11]
- MC+SE: Khurshid'03

Related Work (2/2)

- OpenFlow programming
  - Frenetic [Foster'11], NetCore [Monsanto'12]
- Network testing
  - FlowChecker [Al-Shaer'10]
  - OFRewind [Wundsam'11]
  - Anteater [Mai'11]
  - Header Space Analysis [Kazemian'12]

Micro-benchmark of full state-space search

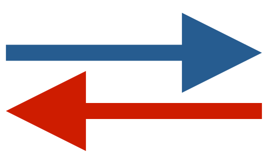
- Single 2.6 GHz core
- 64 GB RAM



Compared with

- SPIN: 7 pings → out of memory
- JPF is 5.5 x slower

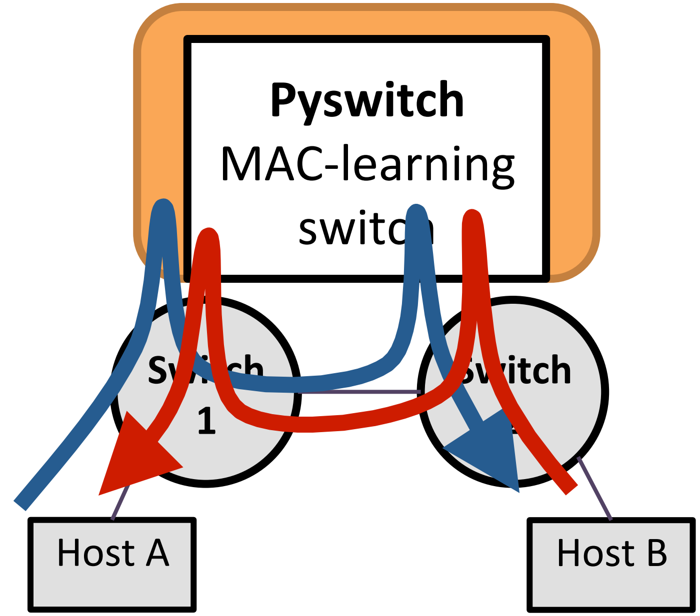
Concurrent "Layer-2 ping"



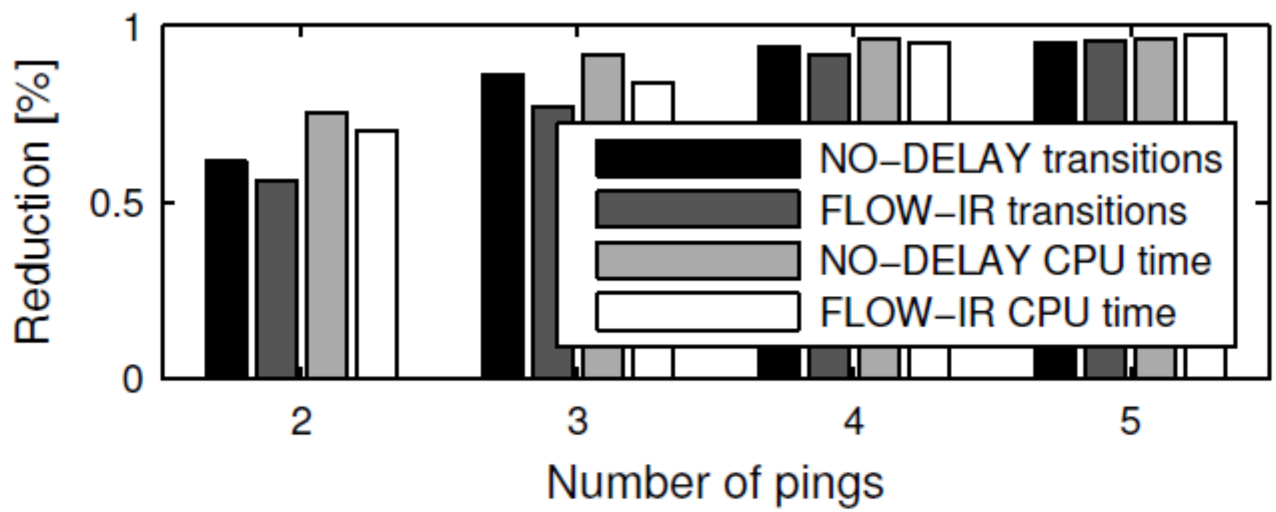
Pings	Transitions	Unique states	Time
2	470	268	0.94 [s]
3	12,801	5,257	47.27 [s]
4	391,091	131,515	36 [m]
5	14,052,853	4,161,335	30 [h]

State space reduction by heuristics

- Single 2.6 GHz core
- 64 GB RAM



Compared to base model checking

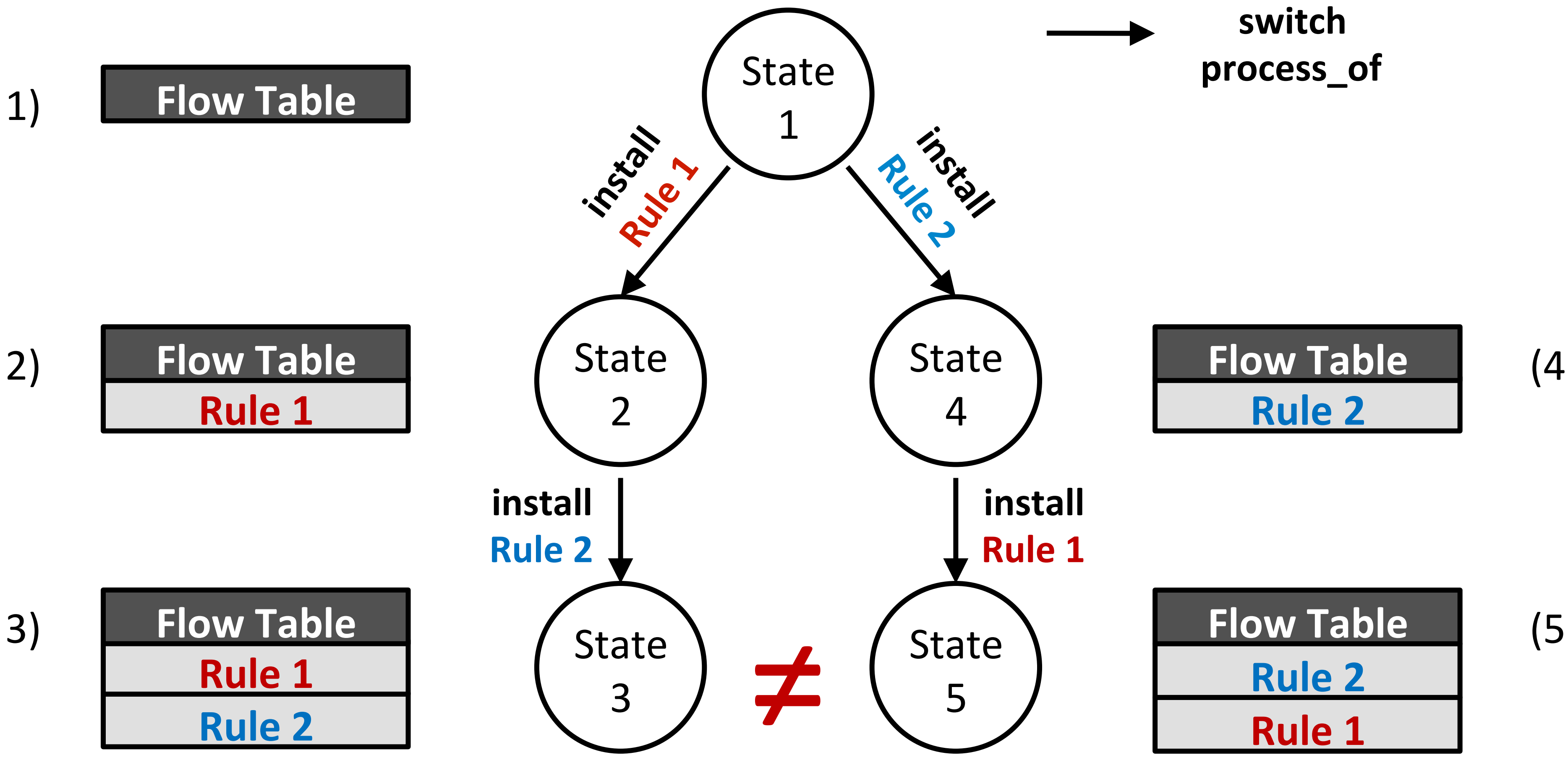


Transitions # / run time [s] to 1st property violation of each bug

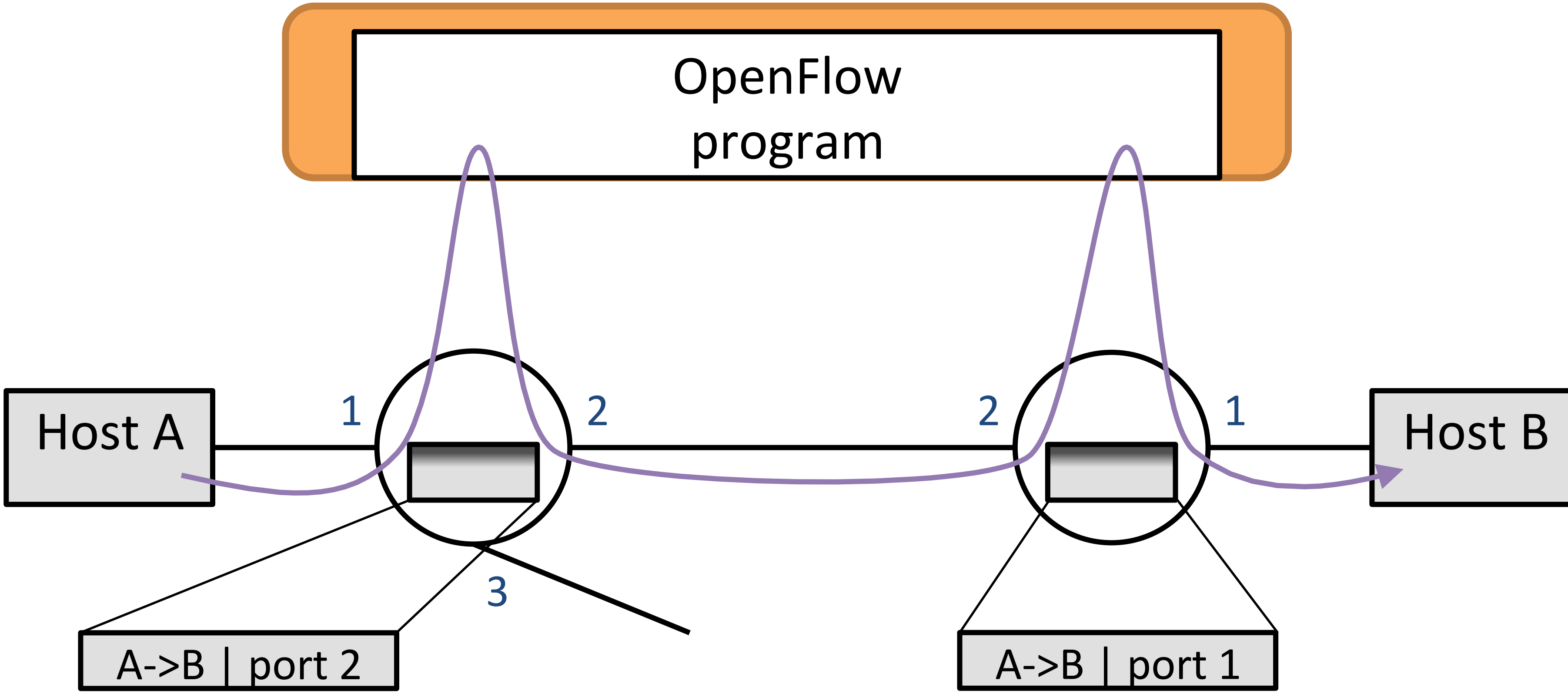
BUG	PKT-SEQ only	NO-DELAY	FLOW-IR	UNUSUAL
I	23 / 0.02	23 / 0.02	23 / 0.02	23 / 0.02
II	18 / 0.01	18 / 0.01	18 / 0.01	18 / 0.01
III	11 / 0.01	16 / 0.01	11 / 0.01	11 / 0.01
IV	386 / 3.41	1661 / 9.66	321 / 1.1	64 / 0.19
V	22 / 0.05	Missed	21 / 0.02	60 / 0.18
VI	48 / 0.05	48 / 0.06	31 / 0.04	49 / 0.07
VII	297k / 1h	191k / 39m	Missed	26.5k / 5m
VIII	23 / 0.03	22 / 0.02	23 / 0.03	23 / 0.02
IX	21 / 0.03	17 / 0.02	21 / 0.03	21 / 0.02
X	2893 / 35.2	Missed	2893 / 35.2	2367 / 25.6
XI	98 / 0.67	Missed	98 / 0.67	25 / 0.03

OpenFlow Switch Model

Example: adding Rule 1 and Rule 2

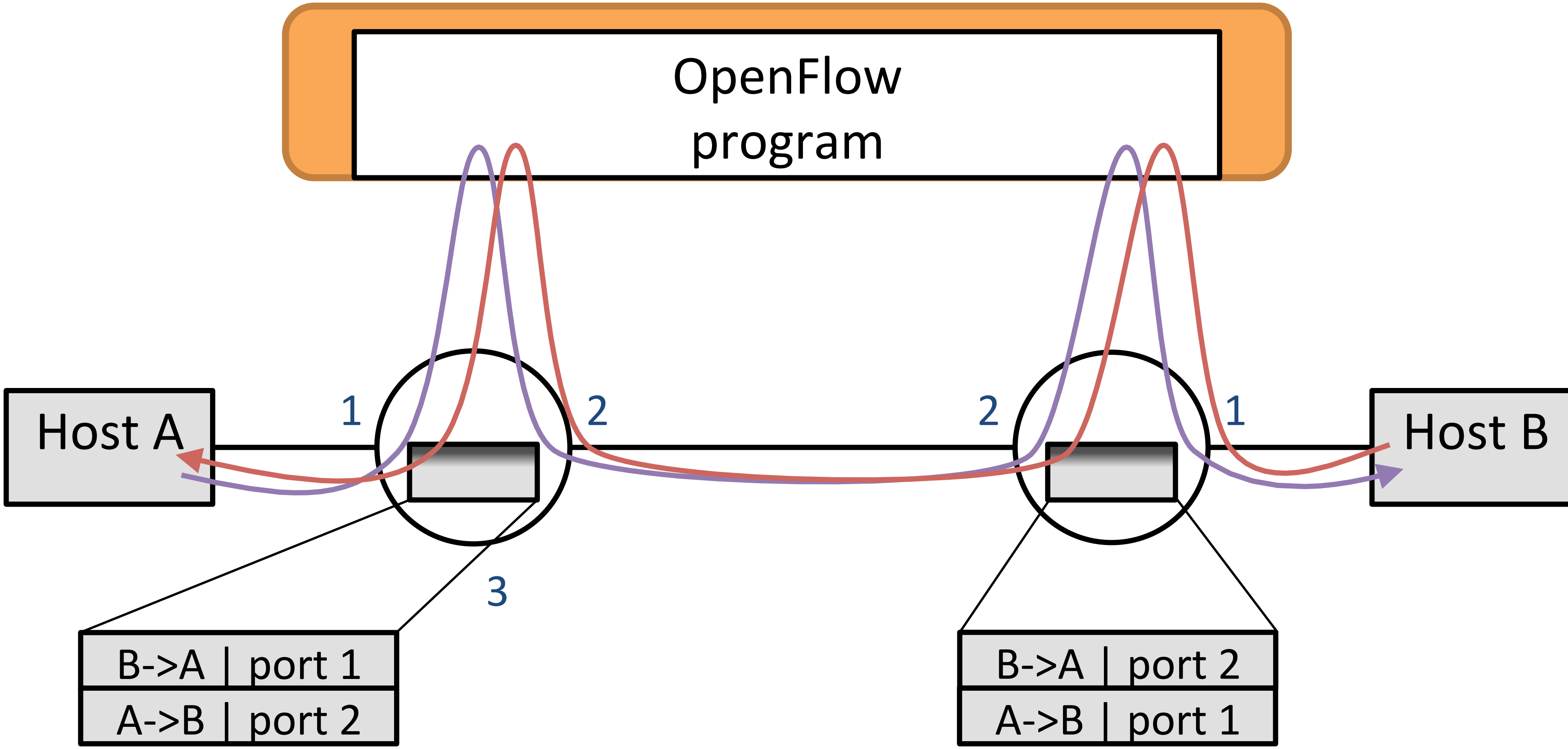


MAC-learning switch (3 bugs)



BUG-I: Host unreachable after moving

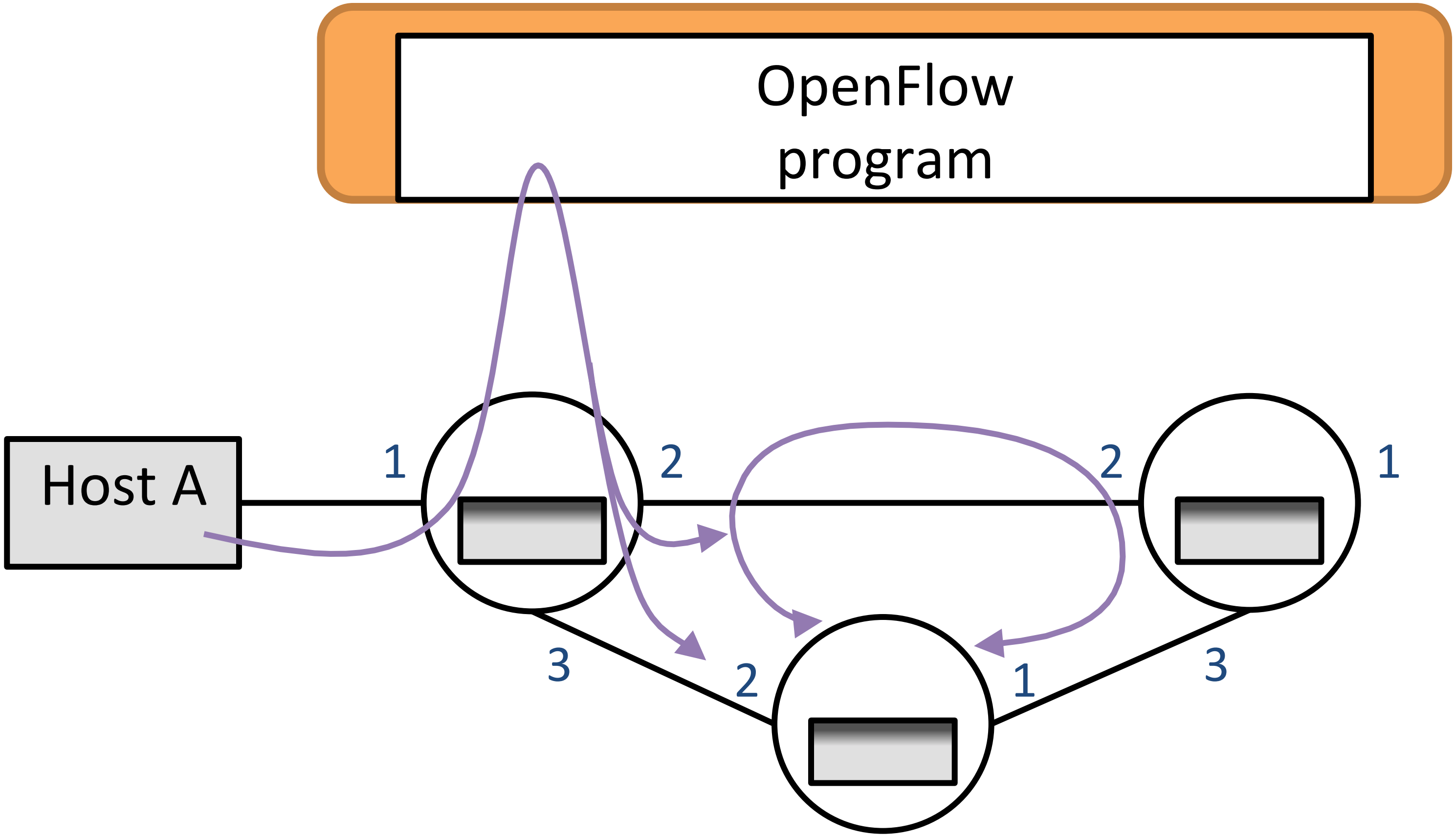
MAC-learning switch (3 bugs)



BUG-I: Host unreachable after moving

BUG-II: Delayed direct path

MAC-learning switch (3 bugs)

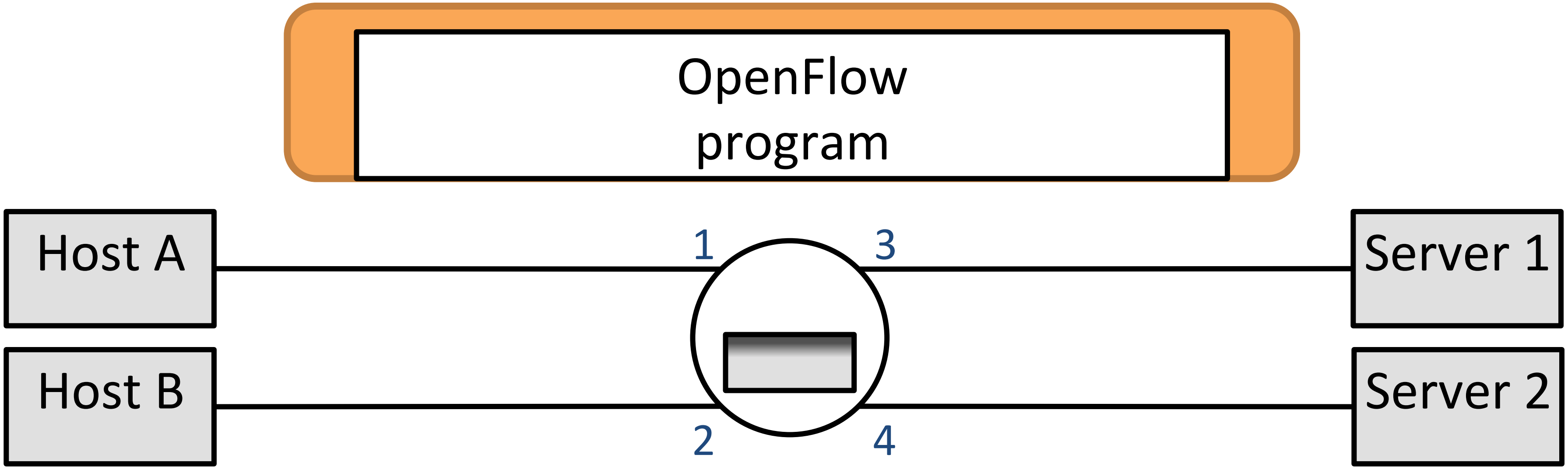


BUG-I: Host unreachable after moving

BUG-II: Delayed direct path

BUG-III: Excess flooding

Web Server Load Balancer (4 bugs)



Custom property: all packets of same request go to same server replica

BUG-IV: Next TCP packet always dropped after reconfiguration

BUG-V: Some TCP packets dropped after reconfiguration

BUG-VI: ARP packets forgotten during address resolution

BUG-VII: Duplicate SYN packets during transitions

Energy-Efficient TE (4 bugs)

- Precompute 2 paths per <origin,dest.>
  - Always-on and on-demand
- Make online decision:
  - Use the smallest subset of network elements that satisfies current demand

BUG-VIII: The first packet of a new flow is dropped

BUG-IX: The first few packets of a new flow can be dropped

BUG-X: Only on-demand routes used under high load

BUG-XI: Packets can be dropped when the load reduces

Results

- Why were mistakes easy to make?
  - Centralized programming model only an abstraction
- Why the programmer could not detect them?
  - Bugs don't always manifest
  - TCP masks transient packet loss
  - Platform lacks runtime checks
- Why NICE easily found them?
  - Makes corner cases as likely as normal cases

Example: MAC-learning Switch

```
ctrl_state = {} # State of the controller is a global variable (a hashtable)
def packet_in(sw_id, inport, pkt, bufid): # Handles packet arrivals
  mactable = ctrl_state[sw_id]
  is_bcast_src = pkt.src[0] & 1
  is_bcast_dst = pkt.dst[0] & 1
  if not is_bcast_src:
    mactable[pkt.src] = inport
  if (not is_bcast_dst) and (mactable.has_key(pkt.dst)):
    outport = mactable[pkt.dst]
    if outport != inport:
      match = {DL SRC: pkt.src, DL DST: pkt.dst, DL TYPE: pkt.type, IN PORT: inport}
      actions = [OUTPUT, outport]
      install_rule(sw_id, match, actions, soft_timer=5, hard_timer=PERMANENT)
      send_packet_out(sw_id, pkt, bufid)
      return
  flood_packet(sw_id, pkt, bufid)
```

Causes of Corner Cases (Examples)

- Multiple packets of a flow reach controller
- No atomic update across multiple switches
- Previously-installed rules limit visibility
- Composing functions that affect same packets
- Assumptions about end-host protocols & SW