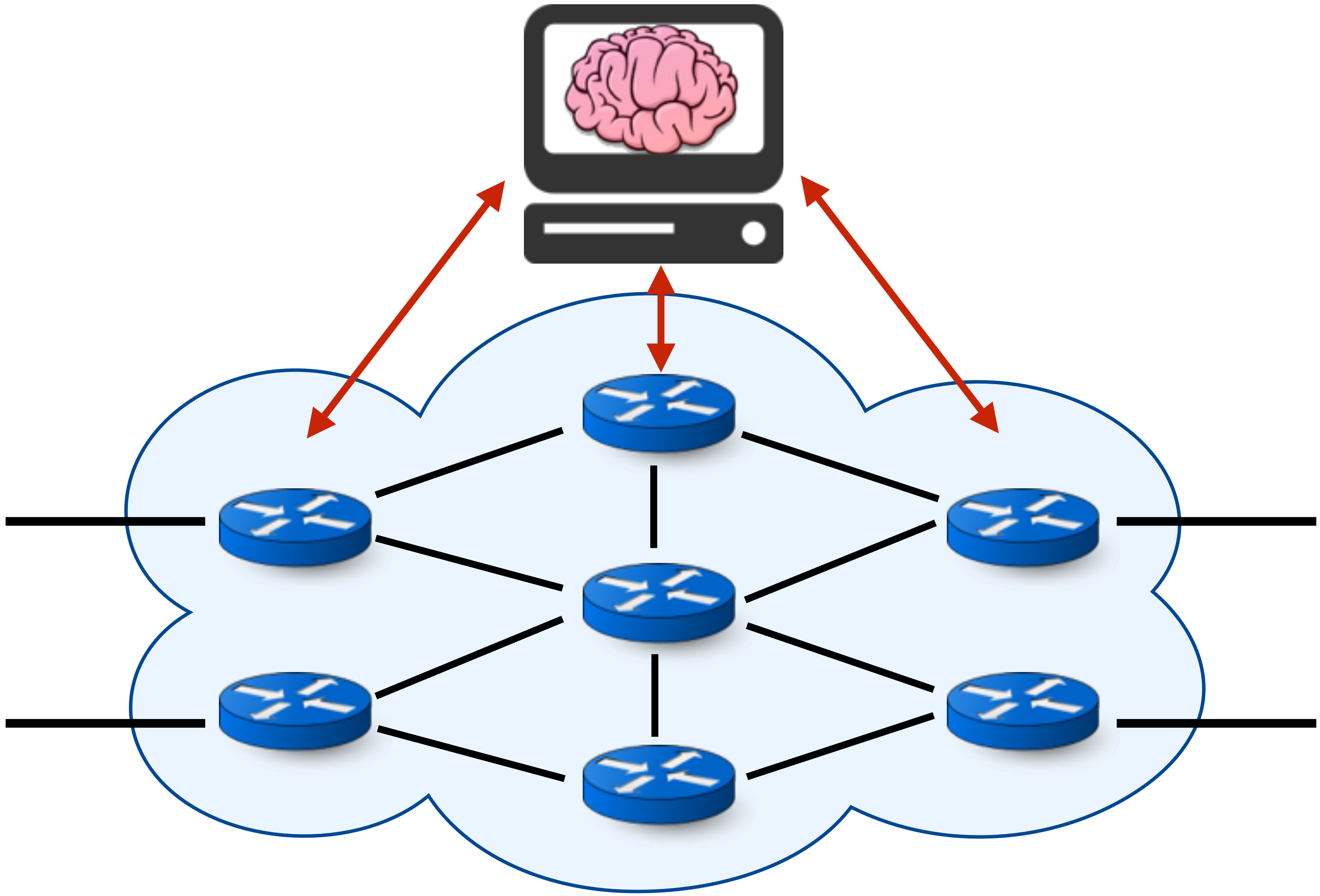
SNAP: Stateful Network-Wide Abstractions for Packet Processing

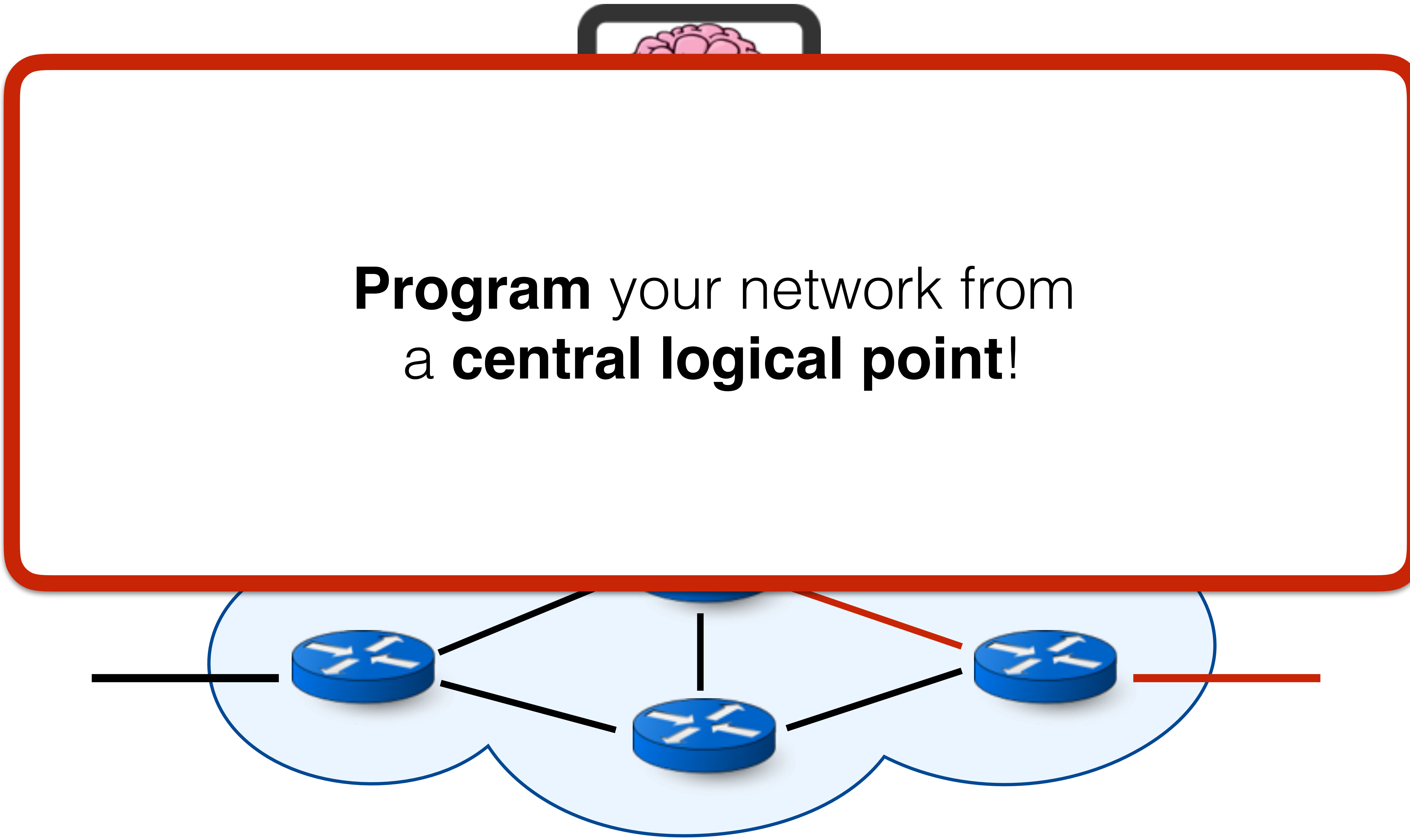
Mina Tahmasbi Arashloo[1], Yaron Koral[1], Michael Greenberg[2], Jennifer Rexford[1], and David Walker[1]

[1] Princeton University, [2] Pomona College

Software Defined Networks (SDN) - Centralized Control



Software Defined Networks (SDN) - Centralized Control



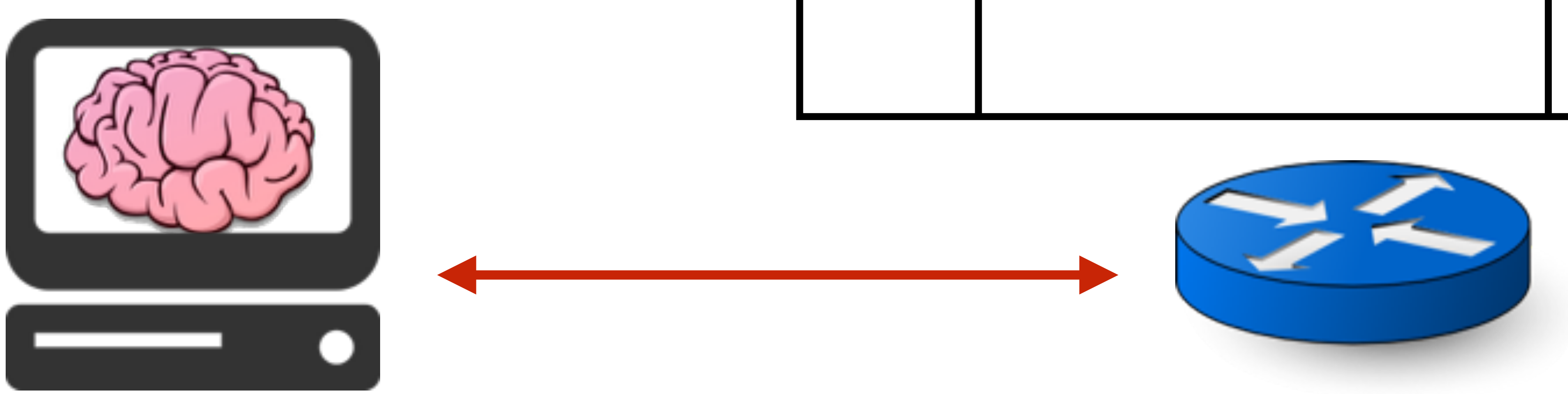
Program your network from a **central logical point**!

OpenFlow - Abstractions for SDN

Each Rule can

- Match on header fields
- modify/forward/drop packets

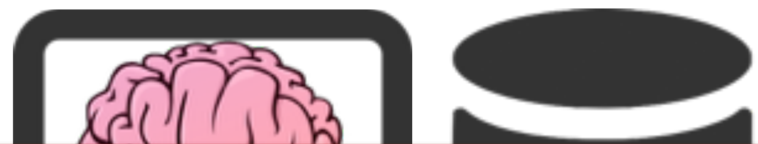
Prio	match	action
1	dstip = 10.0.0.1	outport ← 1
2	dstip = 10.0.0.2	drop
⋮	⋮	⋮



Is OpenFlow Enough?

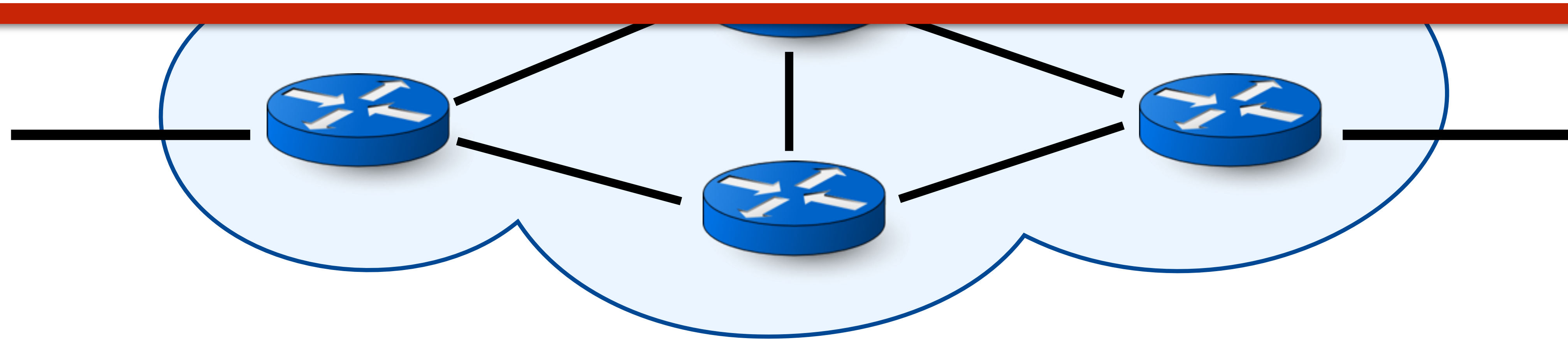
- OpenFlow rules are **“stateless”**
  - Rule tables process each packet independently from the rest
- Algorithms almost always need **“stateful”** processing
  - i.e., decide what to do with the packet based on packets seen so far!

Option #1 - All the state on the controller



Centralized control **but** not efficient!

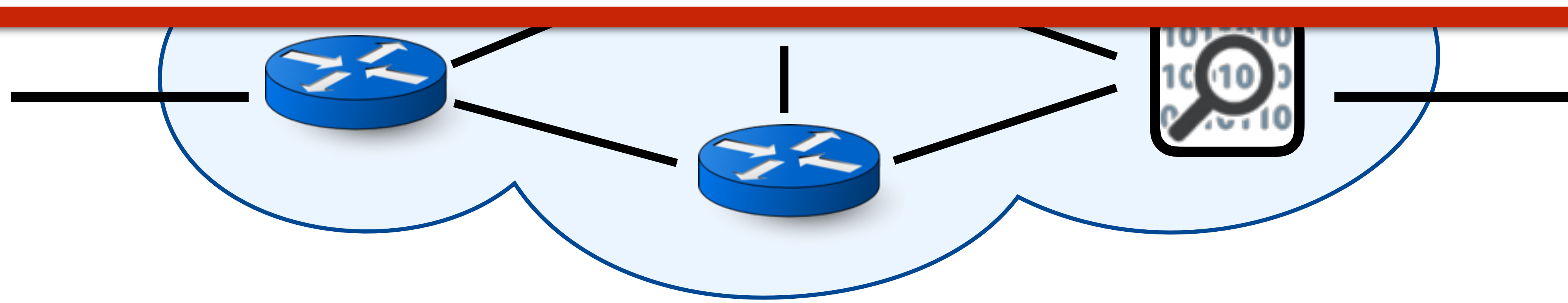
- Switches process packets at **ns** scale
- Going through the controller, each update could take from **ms** to **a few seconds**



Option #2 - Middleboxes (MBs)

Efficient **but** we lose centralized control!

- MBs are ad-hoc blackboxes
- They make it hard to reason about network's behavior



Our Goal

Stateful packet processing

with **centralized control**

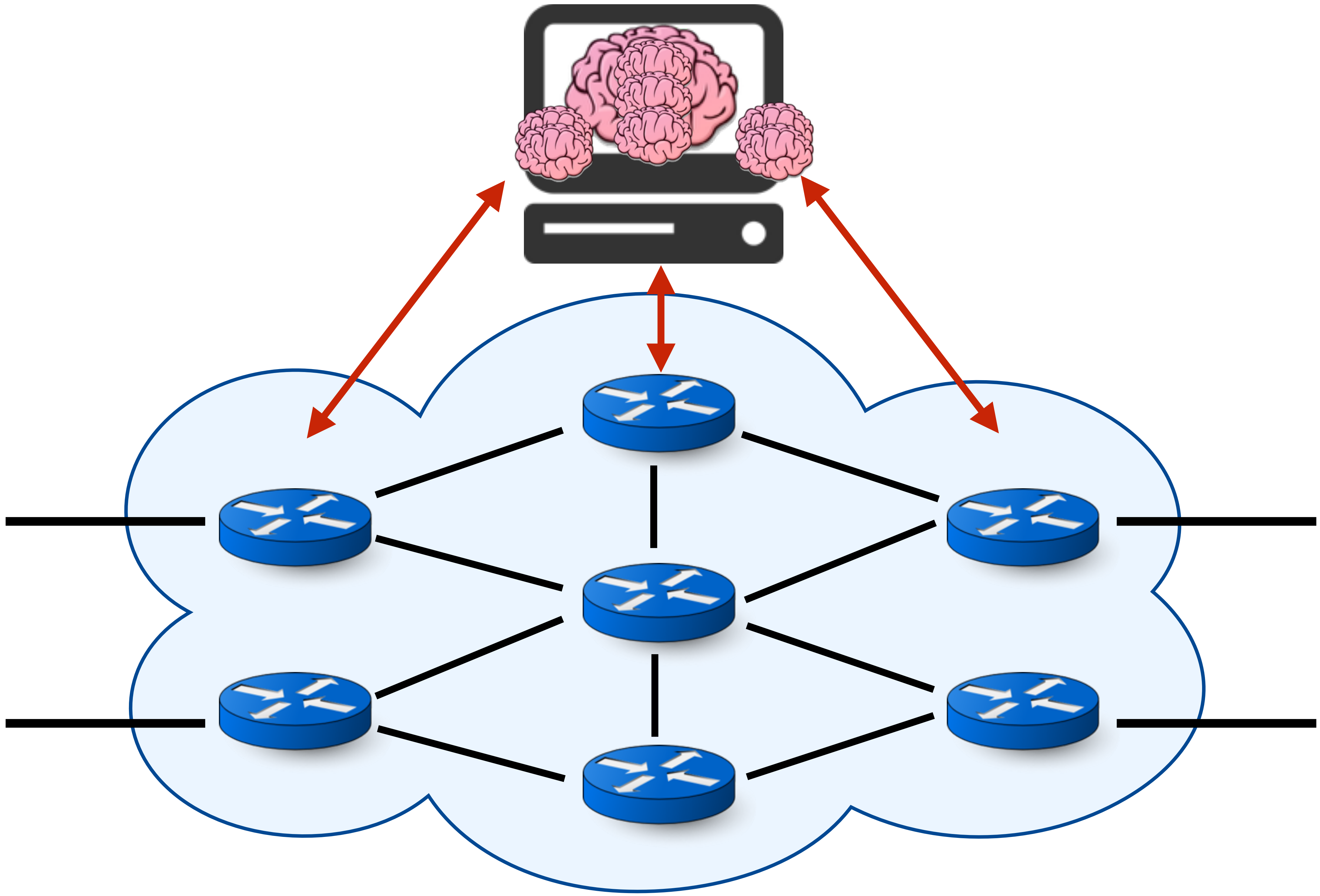
without compromising on **efficiency**

Insight

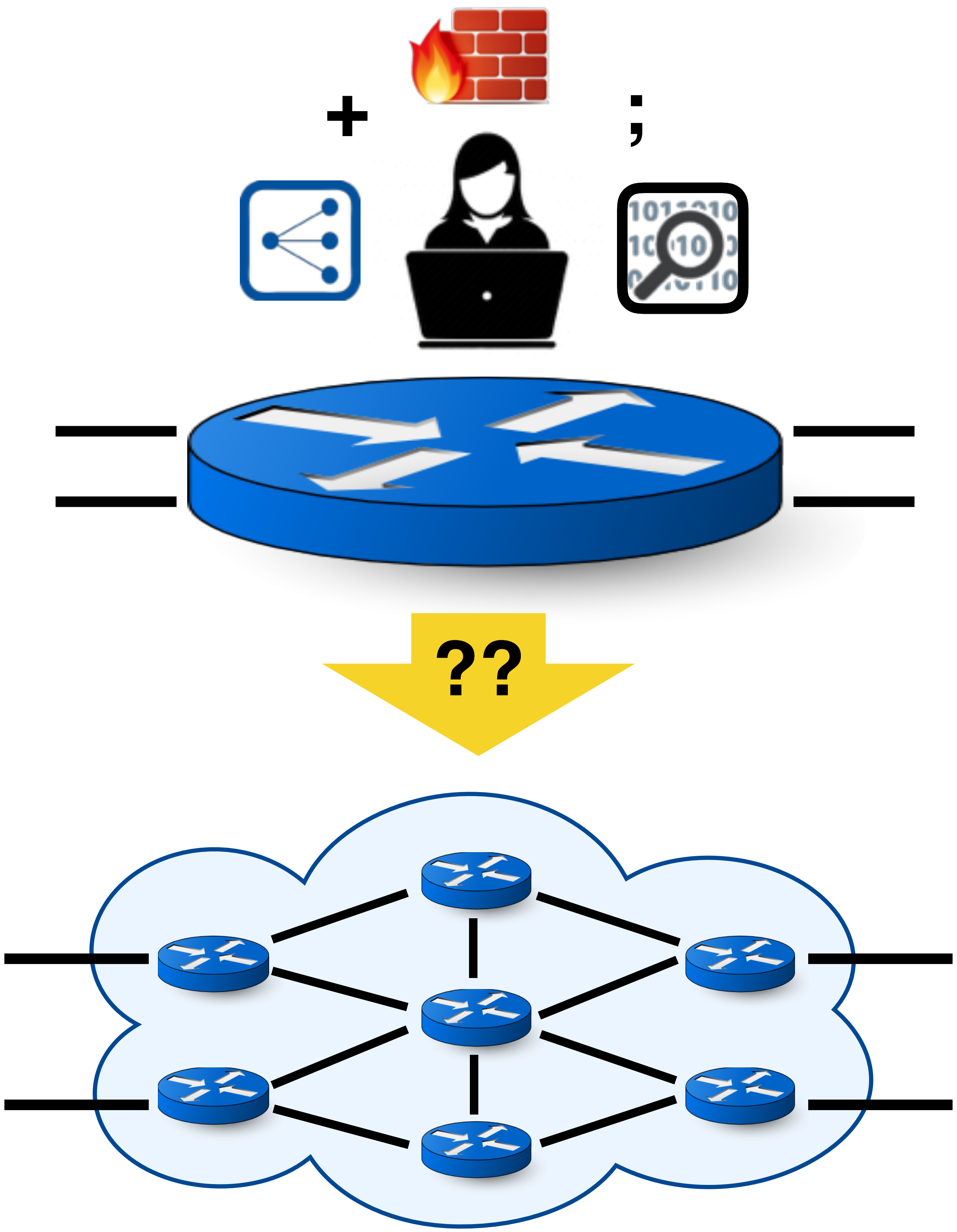
- New switches offer more sophisticated **stateful** packet processing functionality

  - The switch has **local state**
  - Rules can match on/modify local state

Let's push stateful processing to switches!

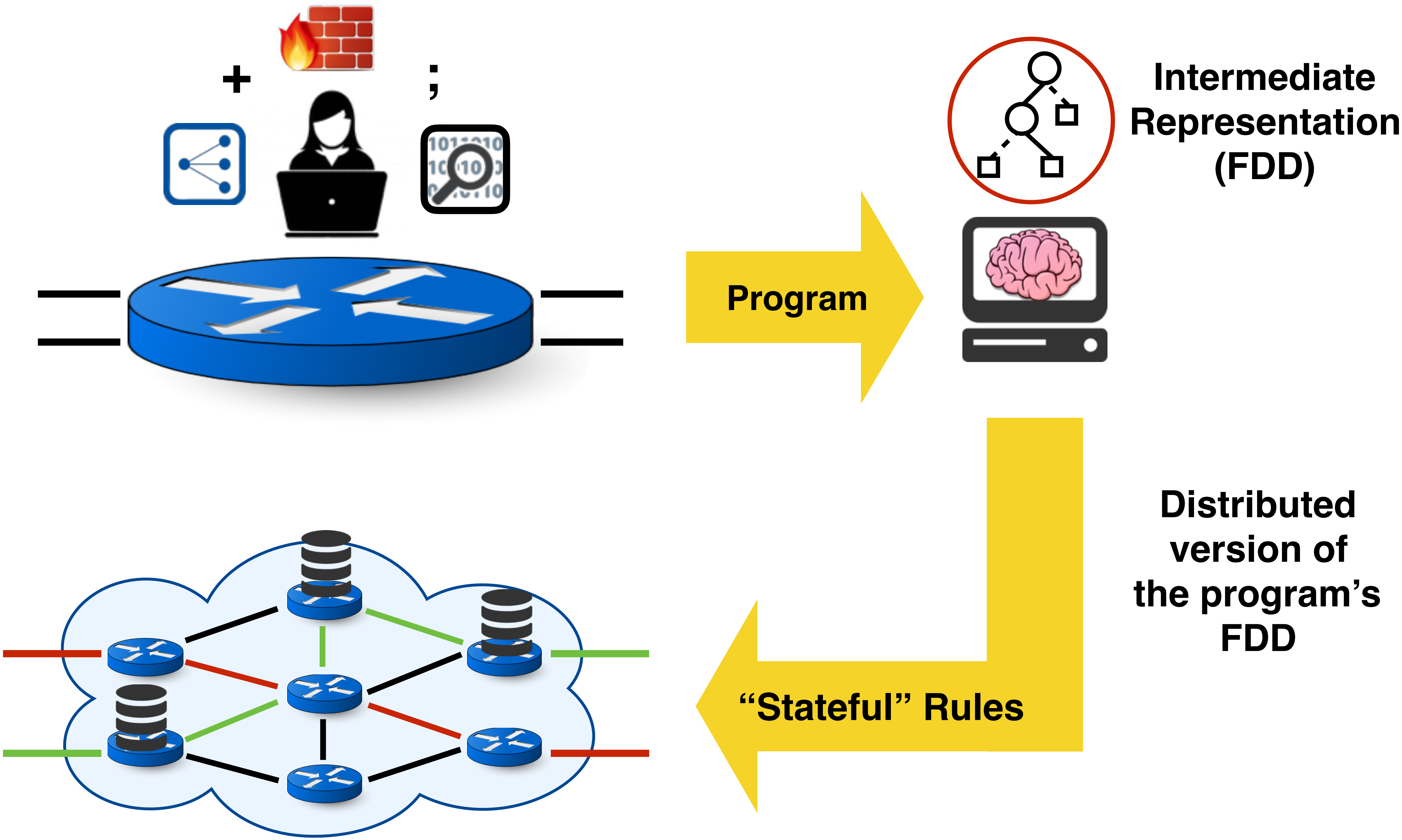


SNAP - Language and Compiler Overview

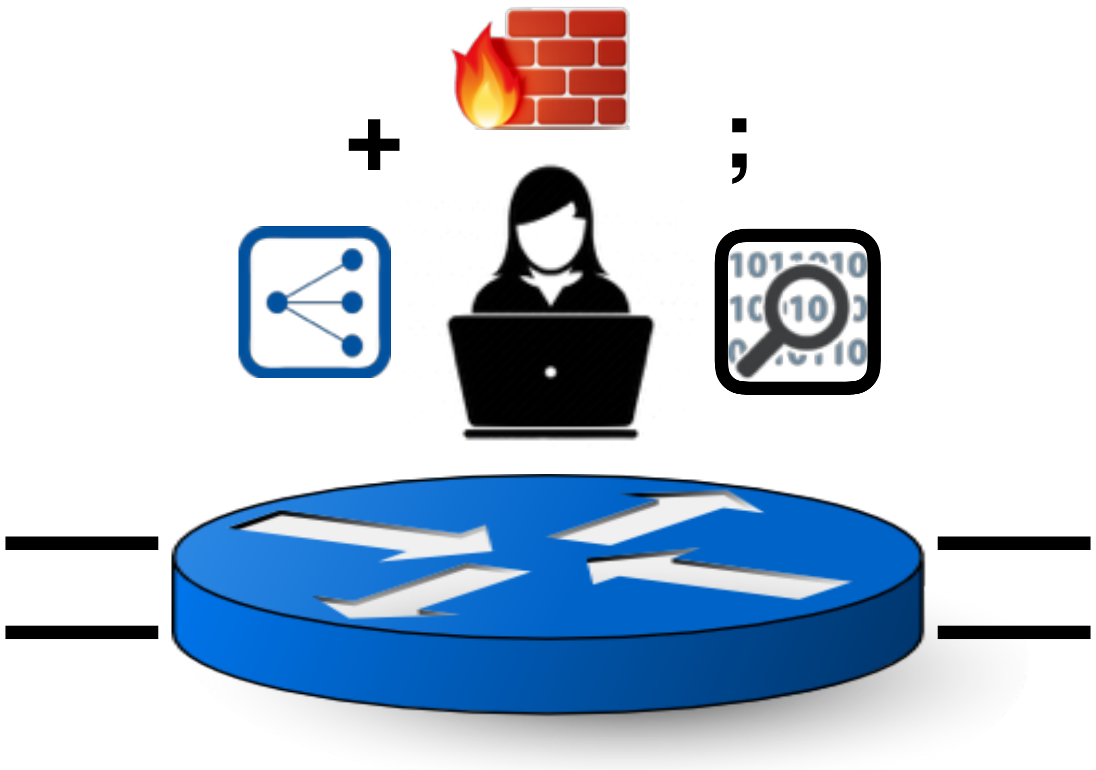


- The stateful program is written on top of one big switch
- The actual network has collections of switches
- How should we realize the program collectively on the network of switches?

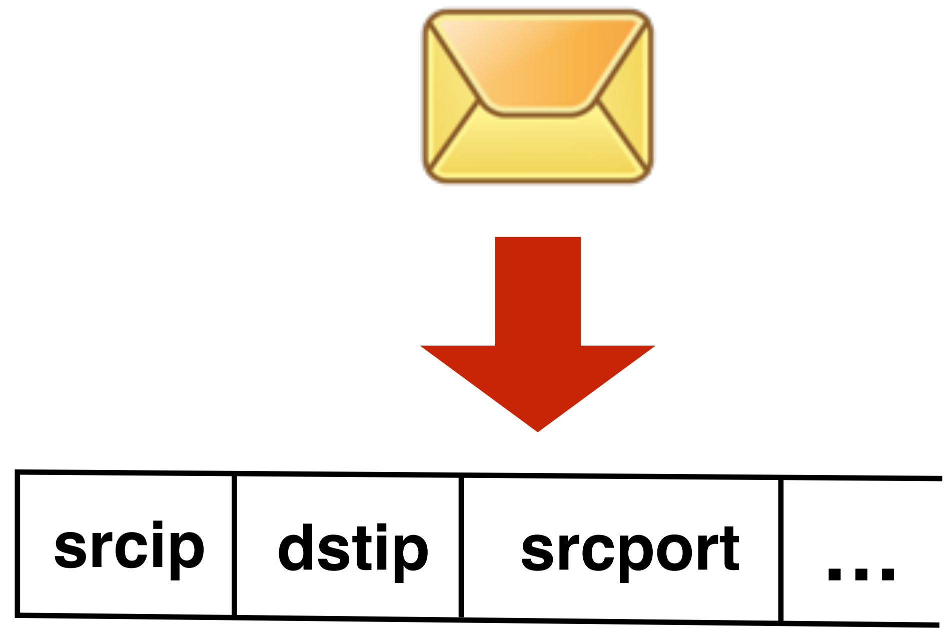
SNAP - Language and Compiler Overview



SNAP - Language



Packets!



Programming Model

- SNAP's expressions are **functions**

current state

input packet

updated state

set of packets

Reads/Modifies state
Reads/Duplicate/Modifies packet

Running Example - Detecting Malicious Domains

- Domains that change TTL frequently are suspected to be malicious

TTL Change Tracking in SNAP

```
if dstip = CS_ip & srcport = DNS then
   if ~seen[dns.domain] then
      seen[dns.domain] ← True;
      last_ttl[dns.domain] ← dns.ttl;
      ttl_change[dns.domain] ← 0
   else
      if dns.ttl = last_ttl[dns.domain] then
         id
      else
         last_ttl[dns.domain] ← dns.ttl;
         ttl_change[dns.domain]++
else id
```

TTL Change Tracking in SNAP

```
if dstip = CS_ip & srcport = DNS then
    if ~seen[dns.domain] then
        seen[dns.domain] ← True;
        last_ttl[dns.domain] ← dns.ttl;
        ttl_change[dns.domain] ← 0
    else
        if dns.ttl = last_ttl[dns.domain] then
            id
        else
            last_ttl[dns.domain] ← dns.ttl;
            ttl_change[dns.domain]++
else id
```

TTL Change Tracking in SNAP

```
if dstip = CS_ip & srcport = DNS then
    if ~seen[dns.domain] then
        seen[dns.domain] ← True;
        last_ttl[dns.domain] ← dns.ttl;
        ttl_change[dns.domain] ← 0
    else
        if dns.ttl = last_ttl[dns.domain] then
            id
        else
            last_ttl[dns.domain] ← dns.ttl;
            ttl_change[dns.domain]++
else id
```

State variable is a key-value dictionary

TTL Change Tracking in SNAP

```
if dstip = CS_ip & srcport = DNS then
    if ~seen[dns.domain] then
        seen[dns.domain] ← True;
        last_ttl[dns.domain] ← dns.ttl;
        ttl_change[dns.domain] ← 0
    else
        if dns.ttl = last_ttl[dns.domain] then
            id
        else
            last_ttl[dns.domain] ← dns.ttl;
            ttl_change[dns.domain]++
else id
```

TTL Change Tracking in SNAP

```
if dstip = CS_ip & srcport = DNS then
    if ~seen[dns.domain] then
        seen[dns.domain] ← True;
        last_ttl[dns.domain] ← dns.ttl;
        ttl_change[dns.domain] ← 0
    else
        if dns.ttl = last_ttl[dns.domain] then
            id
        else
            last_ttl[dns.domain] ← dns.ttl;
            ttl_change[dns.domain]++
else id
```

Adding Forwarding

- Operator wants to specify where packets should be forwarded to

```
forwarding = if dstip = CS_ip then outport ← CS
             else if dstip = EE_ip then outport ← EE
             else if dstip = ISP1_ip then outport ← ISP1
             else if dstip = ISP2_ip then outport ← ISP2
             else drop
```

- Forwarding is composed with TTL change tracking

```
ttl_change ; forwarding
```

SNAP Compiler

Identify State Dependencies ?

Translate to Intermediate Representation (FDD) ?

Identify mapping from packets to state variables ?

Optimally distribute the FDD ?

Generate rules per switch ?

SNAP Compiler

- **Identify State Dependencies** ?
- **Translate to Intermediate Representation (FDD)** ?
- **Identify mapping from packets to state variables** ?
- **Optimally distribute the FDD** ?
- **Generate rules per switch** ?

SNAP Compiler

- **Identify State Dependencies** — **ttl_change → last_ttl → seen**
- **Translate to Intermediate Representation (FDD)** — ?
- **Identify mapping from packets to state variables** — ?
- **Optimally distribute the FDD** — ?
- **Generate rules per switch** — ?

SNAP Compiler

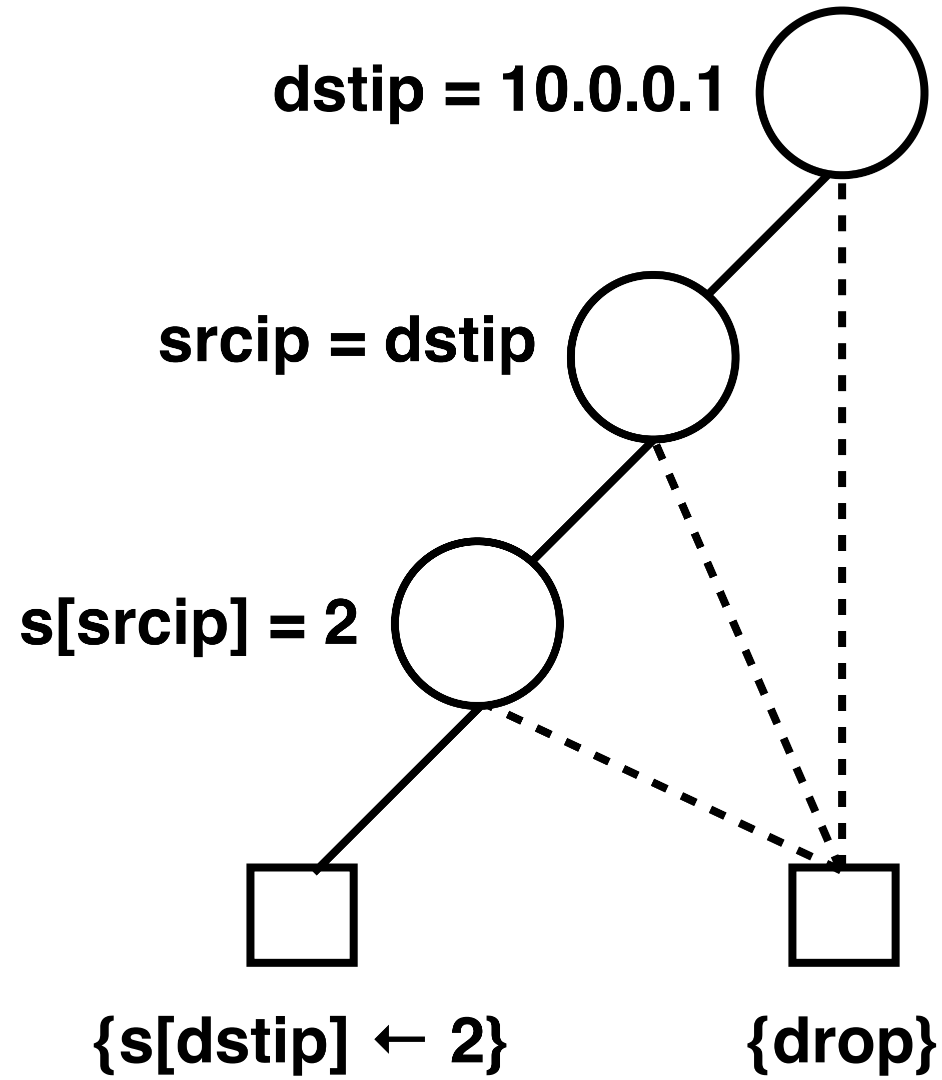
Identify State Dependencies	**ttl_change → last_ttl → seen**
Translate to Intermediate Representation (FDD)	**?**
Identify mapping from packets to state variables	**?**
Optimally distribute the FDD	**?**
Generate rules per switch	**?**

Why Forwarding Decision Diagrams (FDDs)?

- Efficient
  - in terms of number of generated rules
  - for extraction of mapping from packets to state variables (next phase)

Forwarding Decision Diagrams (FDDs)

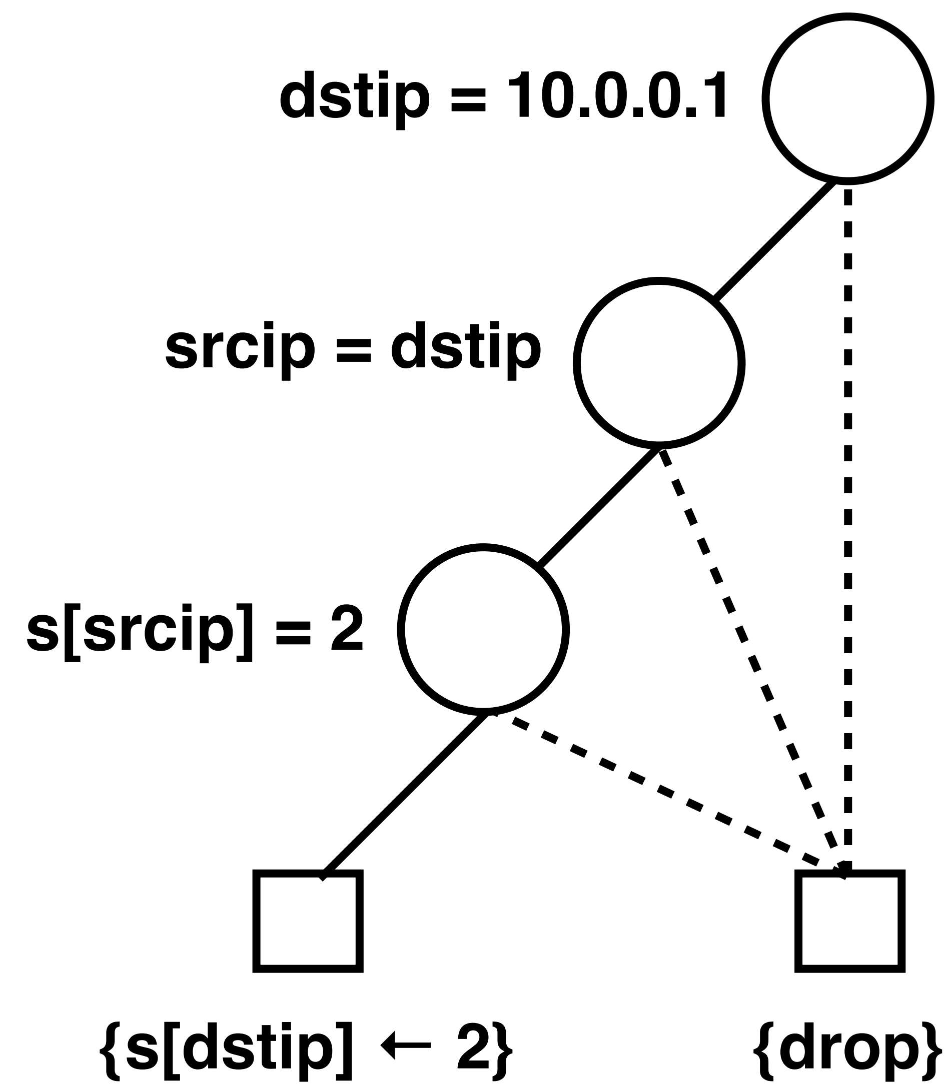
- Generalization of binary decision diagrams [1]
- Intermediate node : test on header fields and state
- Leaf : set of action sequences



[1] Fast NetKAT Compiler, Smolka et.al, SIGPLAN 2015

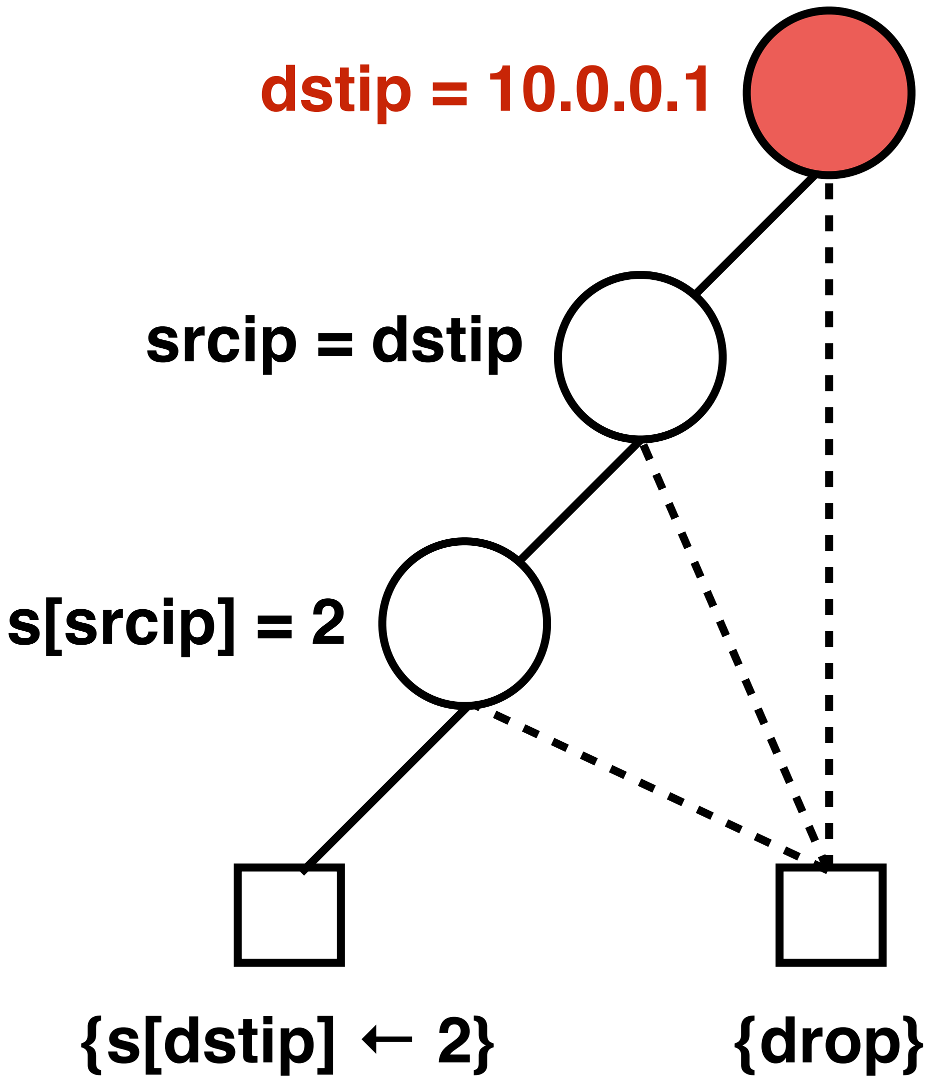
Forwarding Decision Diagrams (FDDs)

- Three types of tests
  - *field = value*
  - $field_1 = field_2$
  - $state\_var[e_1] = e_2$



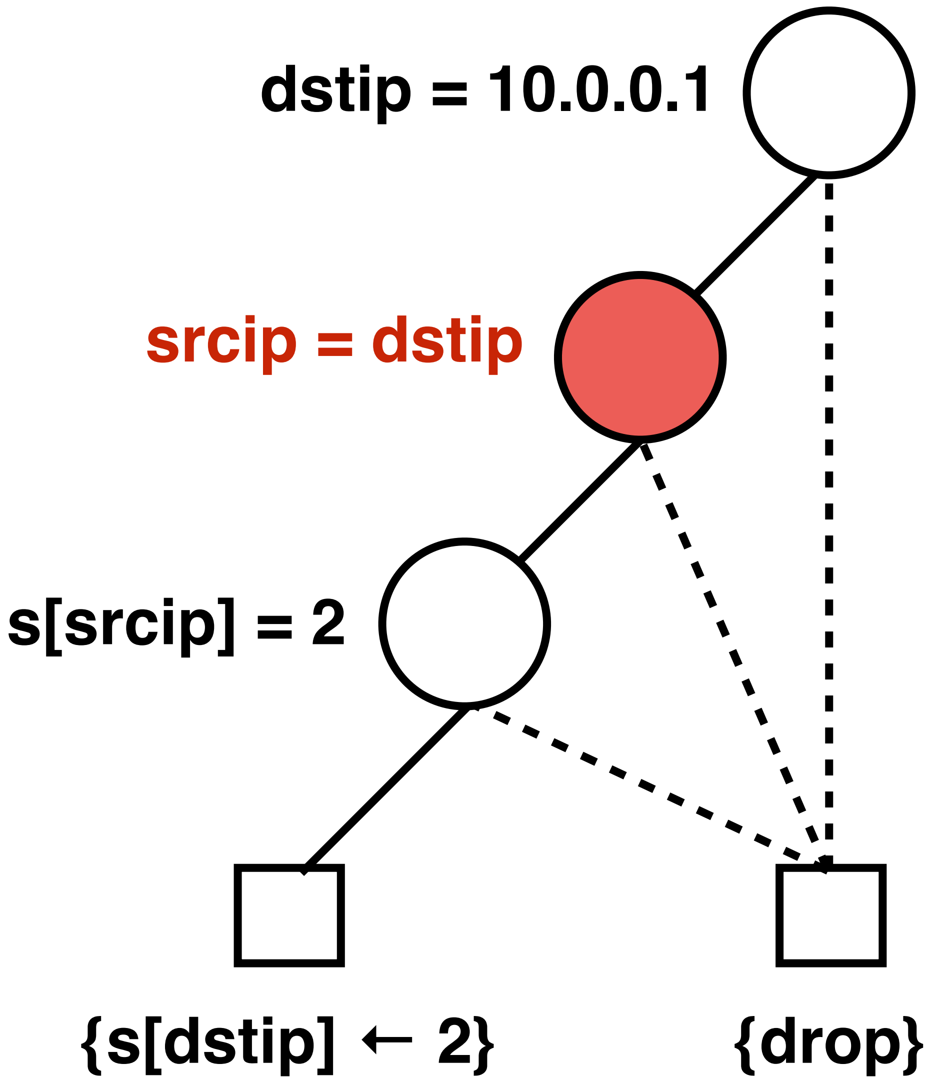
Forwarding Decision Diagrams (FDDs)

- Three types of tests
  - *field = value*
  - *$field_1$ = $field_2$*
  - *state_var[e_1] = e_2*



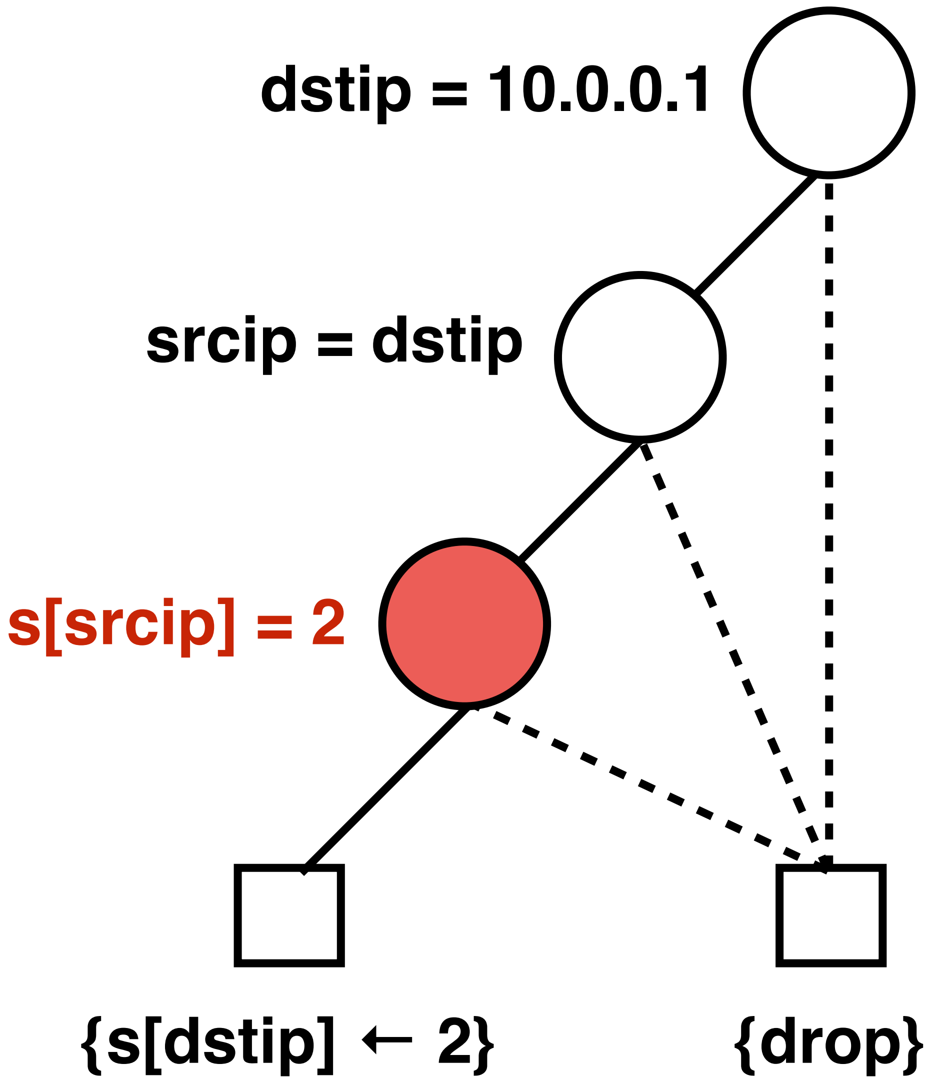
Forwarding Decision Diagrams (FDDs)

- Three types of tests
  - *field = value*
  - *$field_1$ = $field_2$*
  - *state_var[e_1] = e_2*



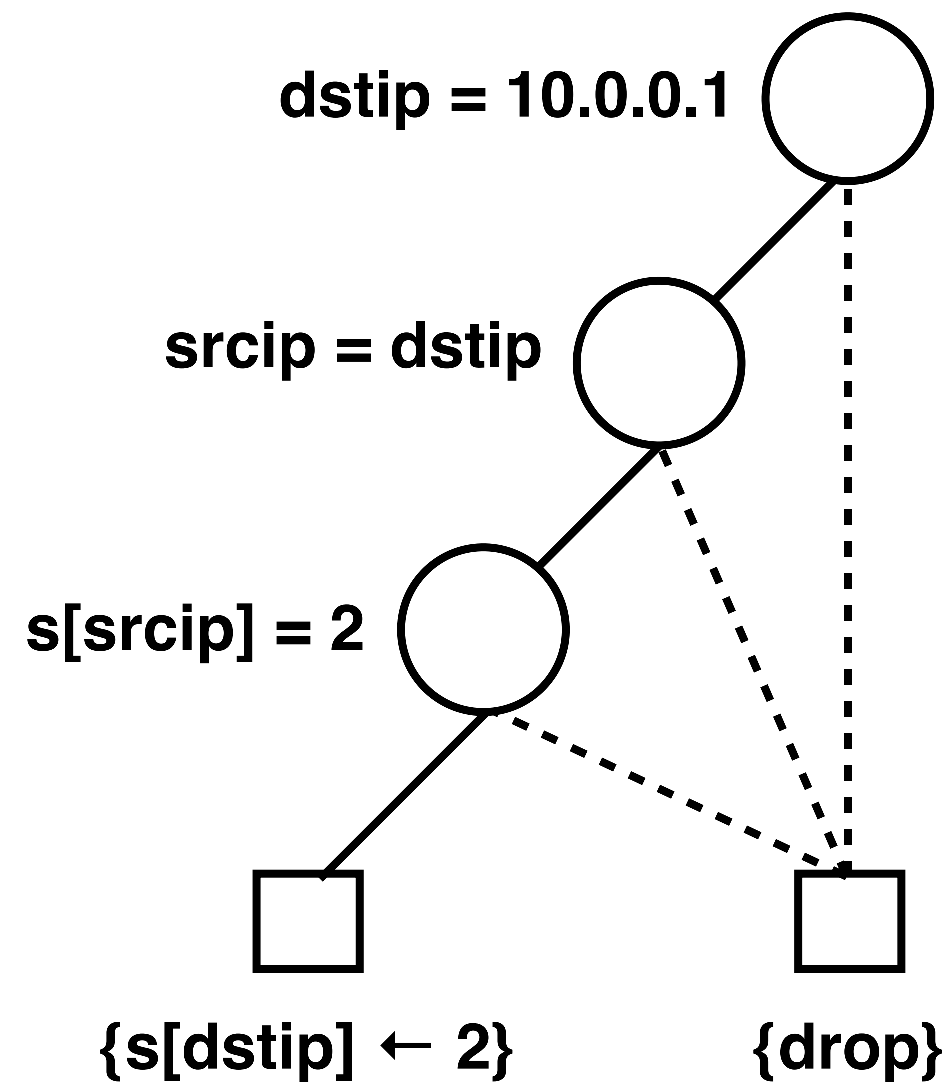
Forwarding Decision Diagrams (FDDs)

- Three types of tests
  - *field = value*
  - $field_1 = field_2$
  - $state\_var[e_1] = e_2$

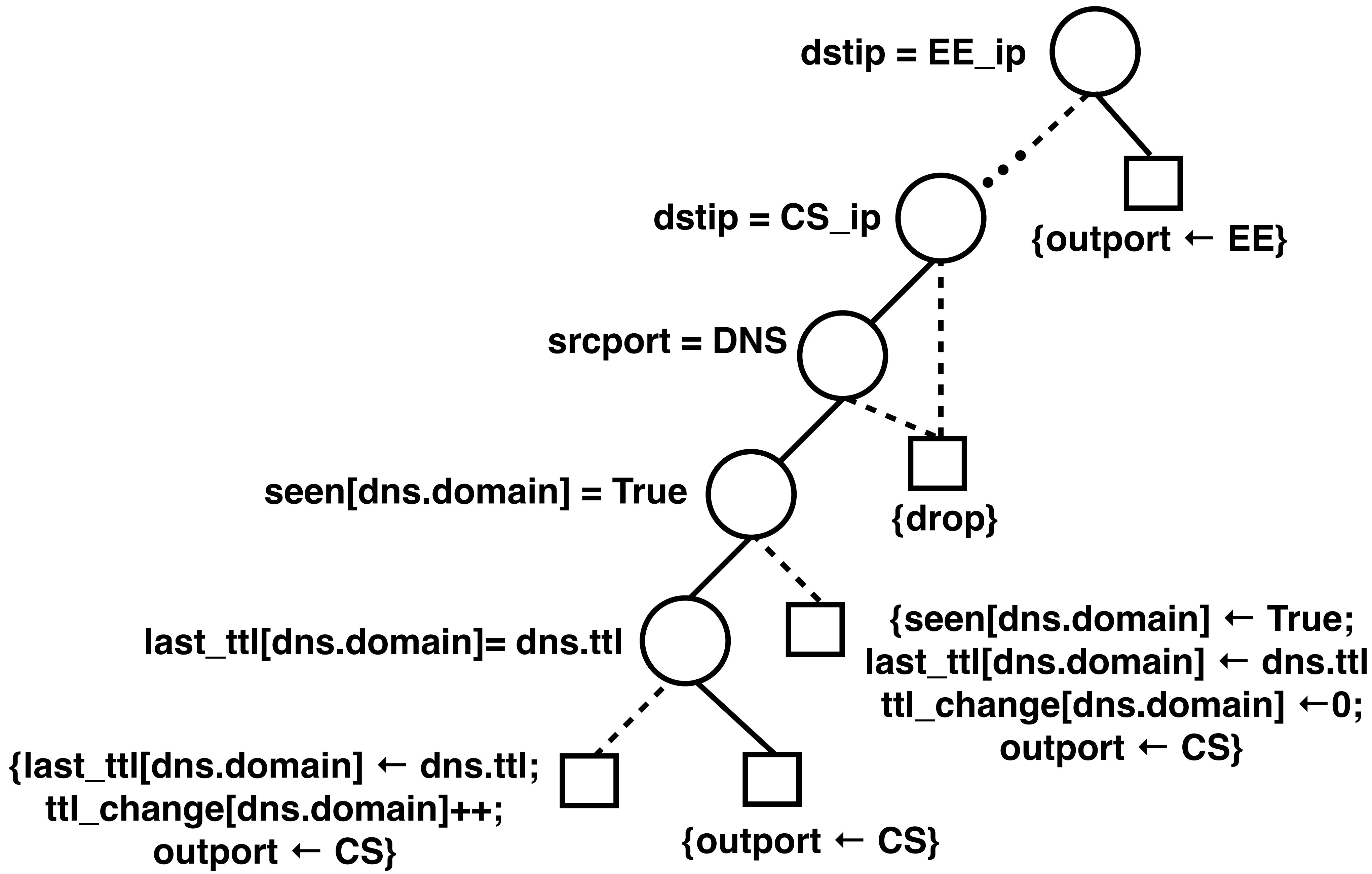


Forwarding Decision Diagrams (FDDs)

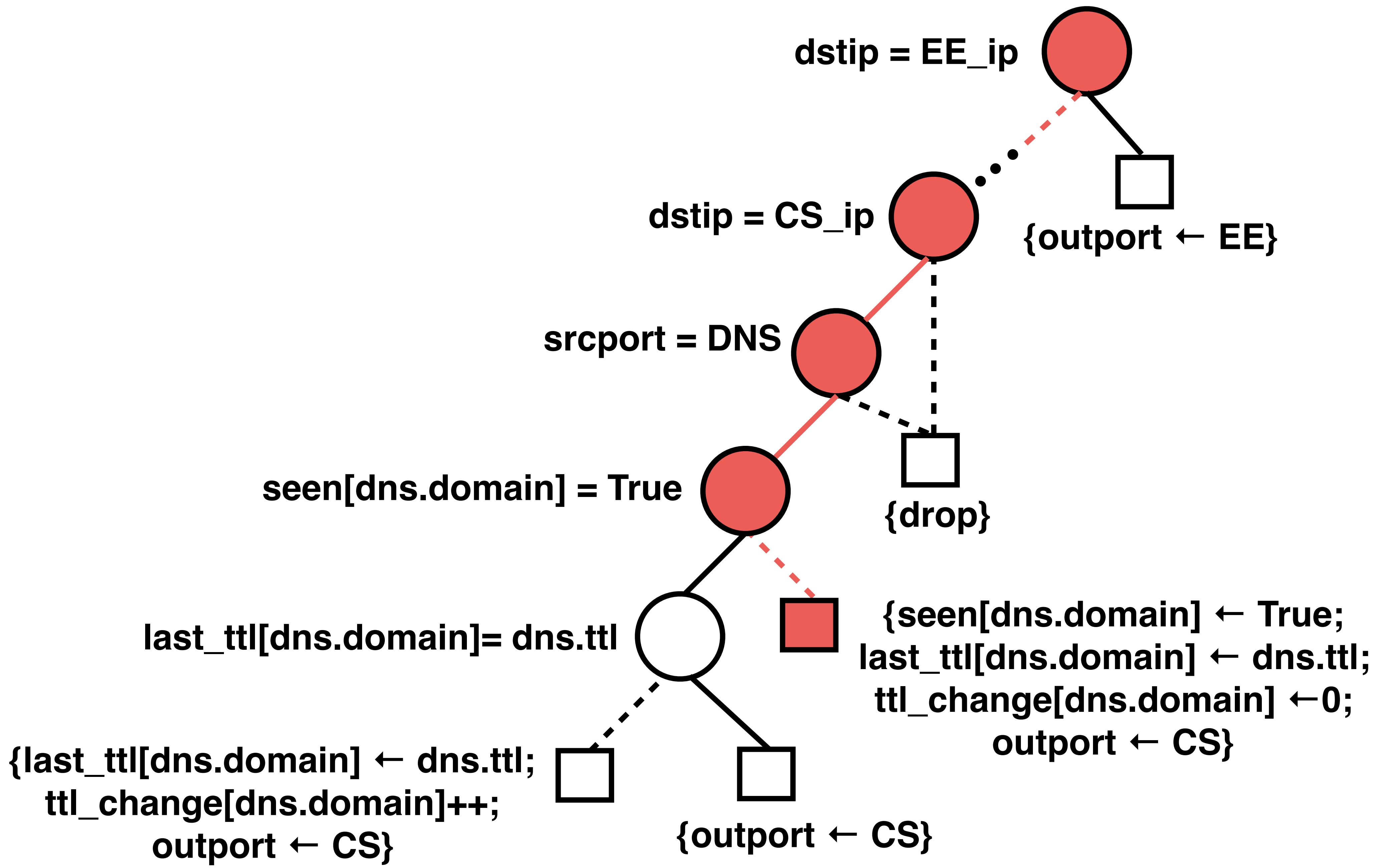
- Three types of tests
  - *field = value*
  - ***$field_1$ = $field_2$***
  - ***state_var[e_1] = e_2***



SNAP Expression to FDD



SNAP Expression to FDD



SNAP Compiler

Identify State Dependencies	ttl_change → last_ttl → seen
Translate to Intermediate Representation (FDD)	✔
Identify mapping from packets to state variables	?
Optimally distribute the FDD	?
Generate rules per switch	?

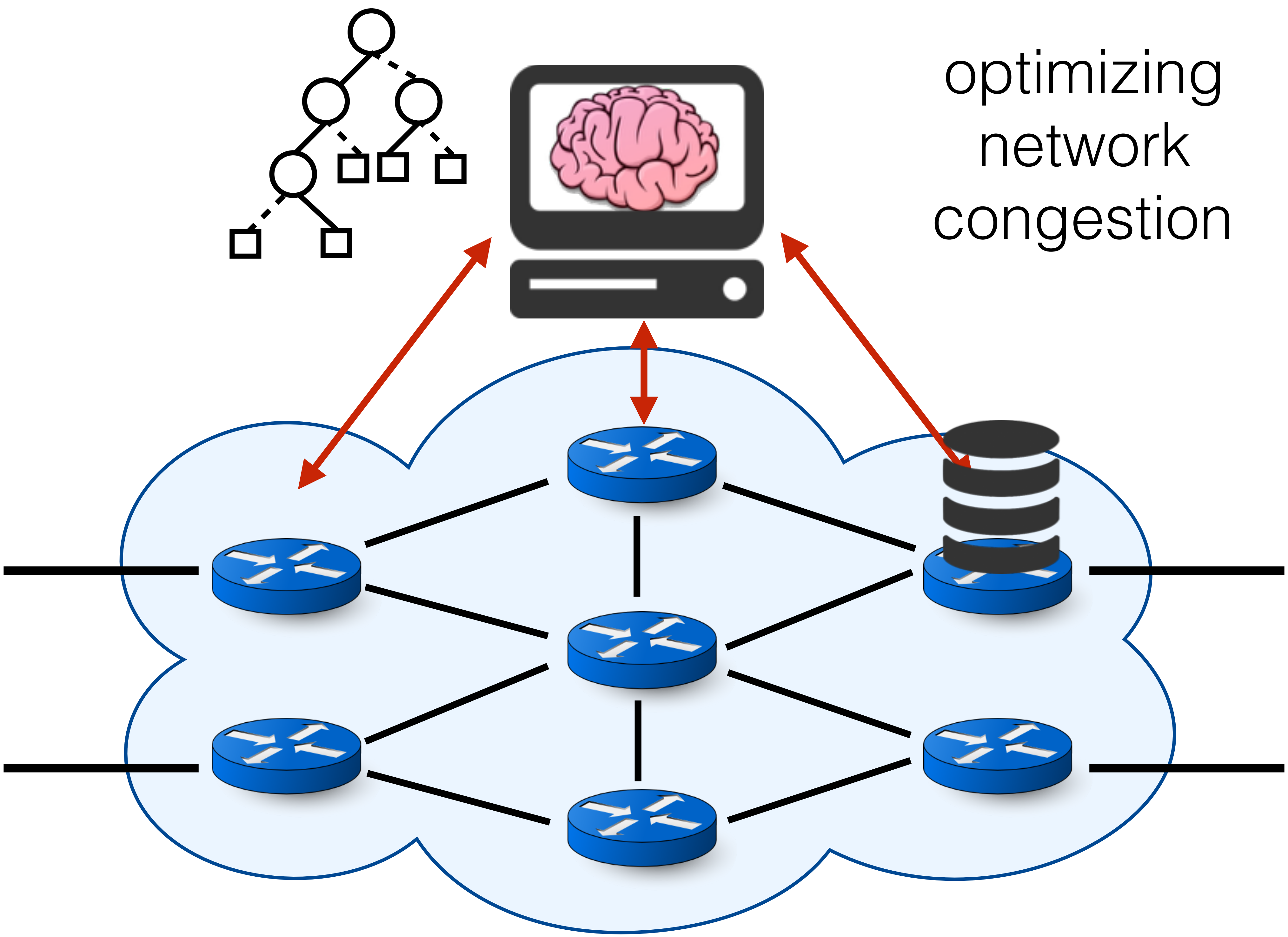
SNAP Compiler

Identify State Dependencies	**ttl_change → last_ttl → seen**
Translate to Intermediate Representation (FDD)	✔
Identify mapping from packets to state variables	**flows to CS need all three state variables**
Optimally distribute the FDD	**?**
Generate rules per switch	**?**

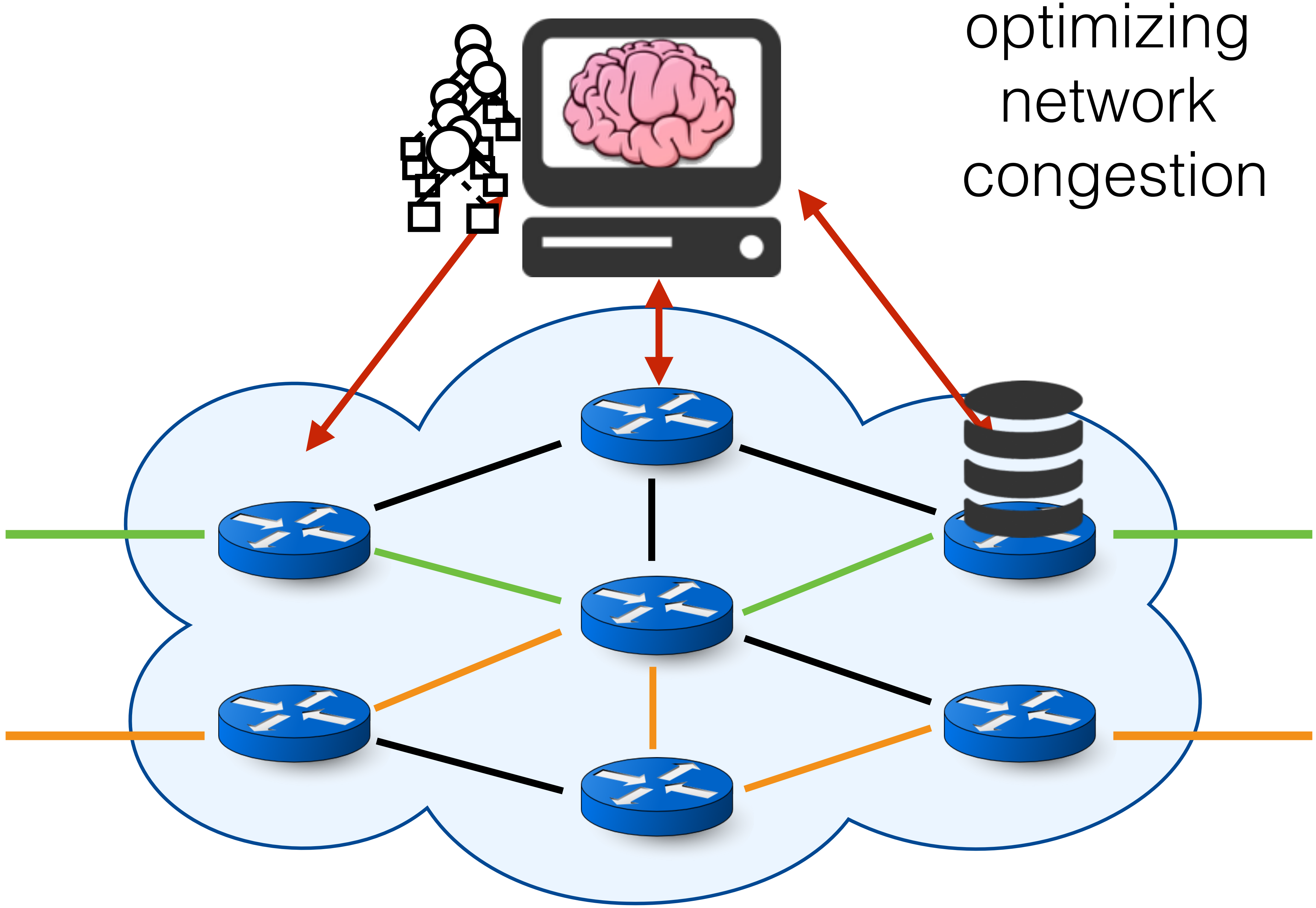
SNAP Compiler

Identify State Dependencies	**ttl_change → last_ttl → seen**
Translate to Intermediate Representation (FDD)	✔
Identify mapping from packets to state variables	**flows to CS need all three state variables**
Optimally distribute the FDD	**?**
Generate rules per switch	**?**

Optimal Distribution of the FDD



Optimal Distribution of the FDD



SNAP Compiler

Identify State Dependencies	ttl_change → last_ttl → seen
Translate to Intermediate Representation (FDD)	✔
Identify mapping from packets to state variables	flows to CS need all three state variables
Optimally distribute the FDD	✔
Generate rules per switch	?

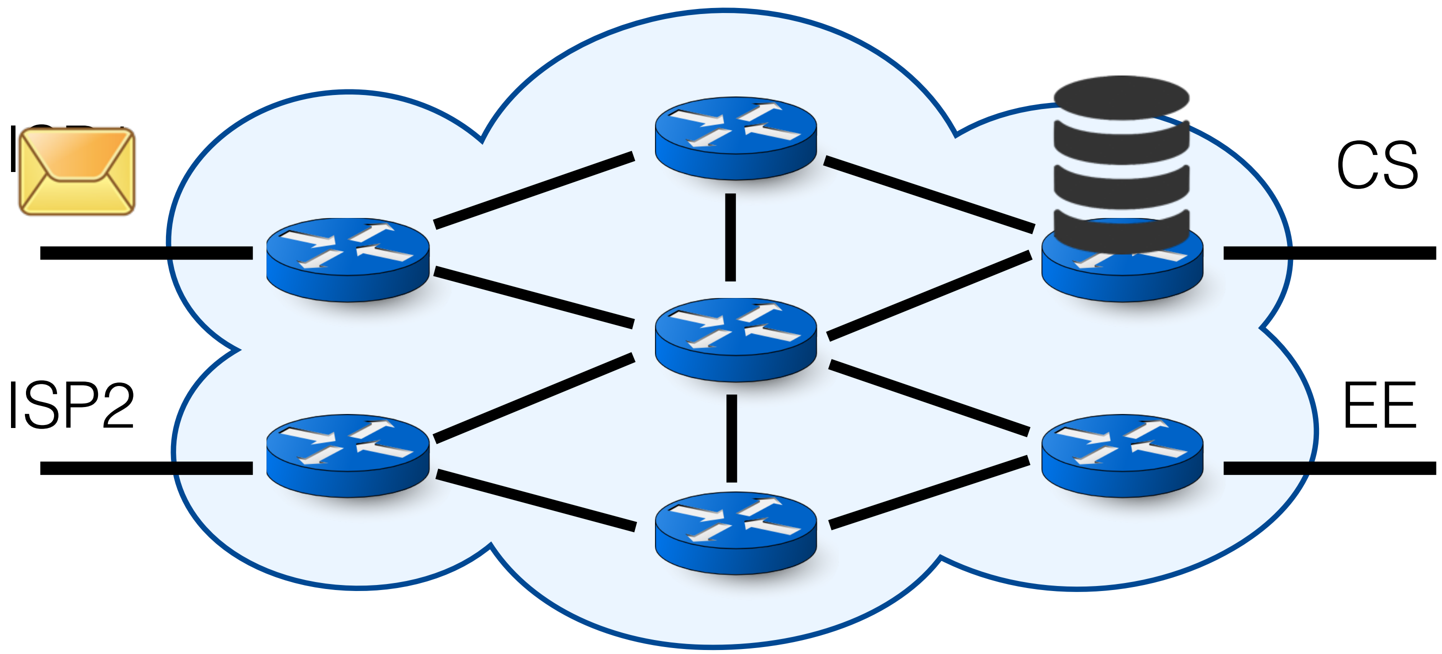
SNAP Compiler

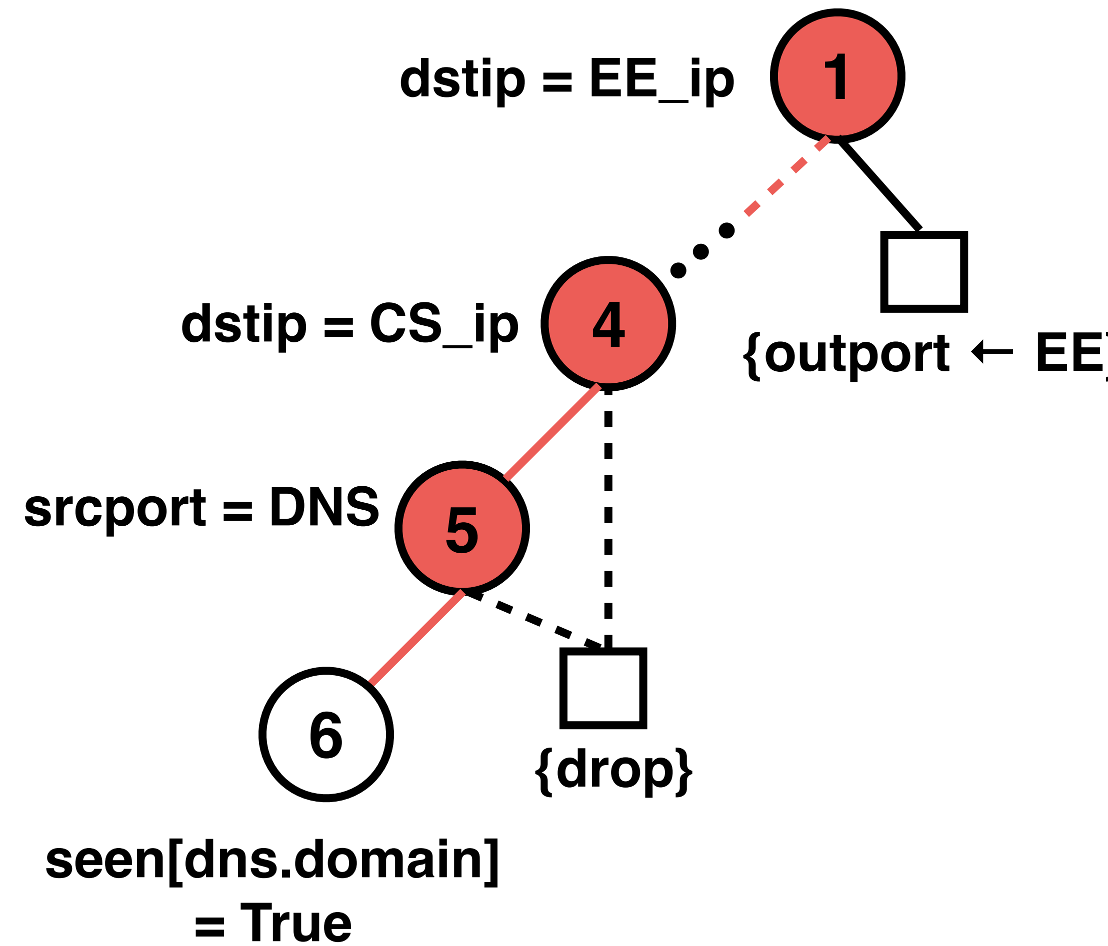
Identify State Dependencies	ttl_change → last_ttl → seen
Translate to Intermediate Representation (FDD)	✔
Identify mapping from packets to state variables	flows to CS need all three state variables
Optimally distribute the FDD	✔
Generate rules per switch	?

SNAP Compiler

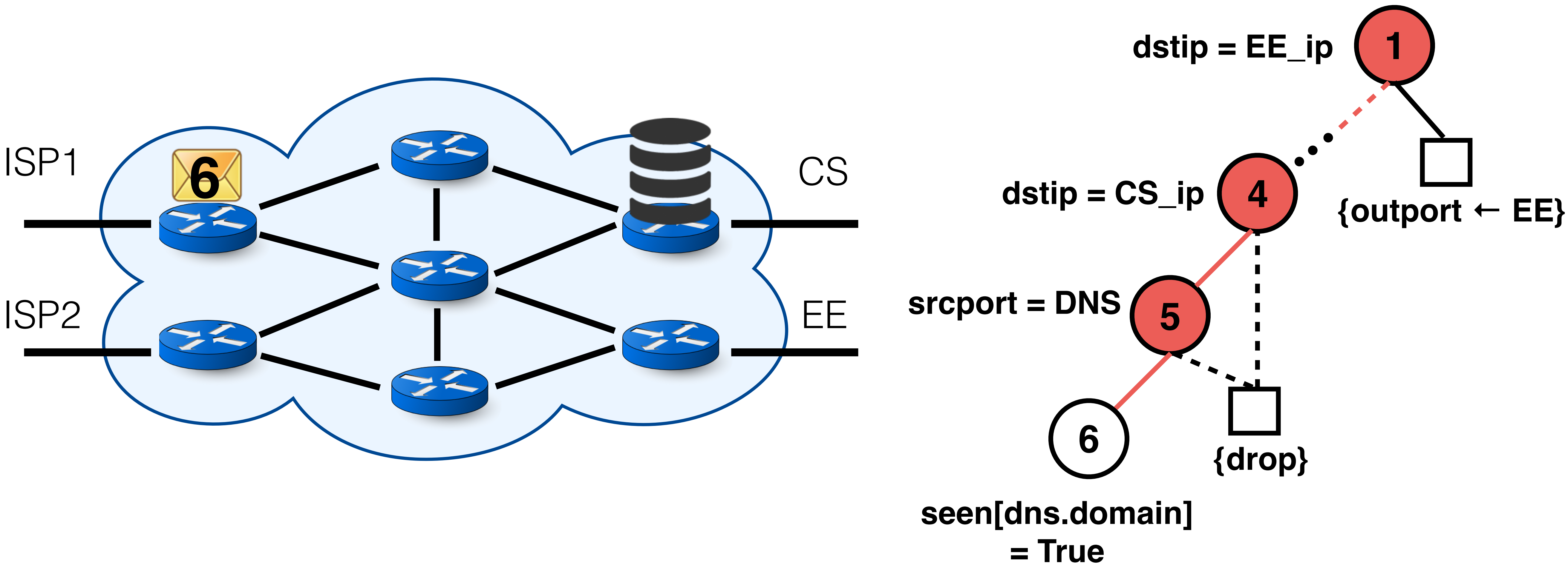
Identify State Dependencies	ttl_change → last_ttl → seen
Translate to Intermediate Representation (FDD)	✔
Identify mapping from packets to state variables	flows to CS need all three state variables
Optimally distribute the FDD	✔
Generate rules per switch	✔

Putting It All Together

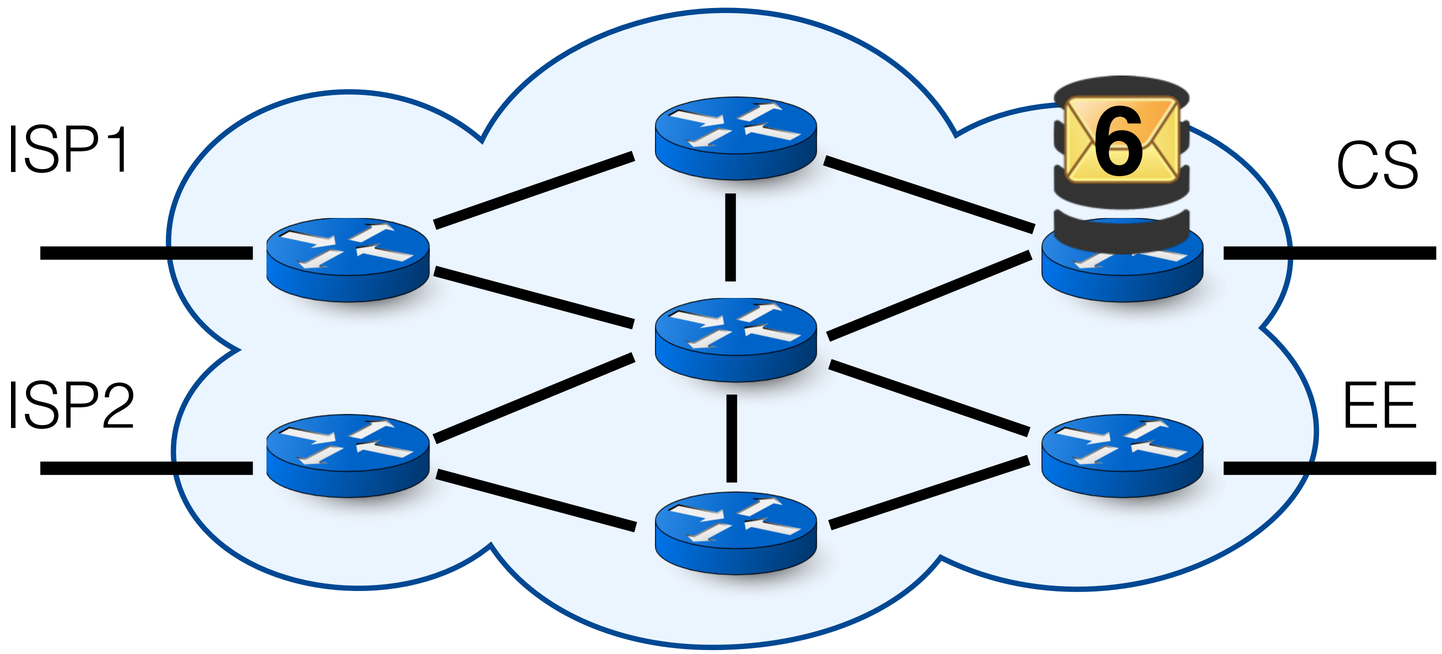


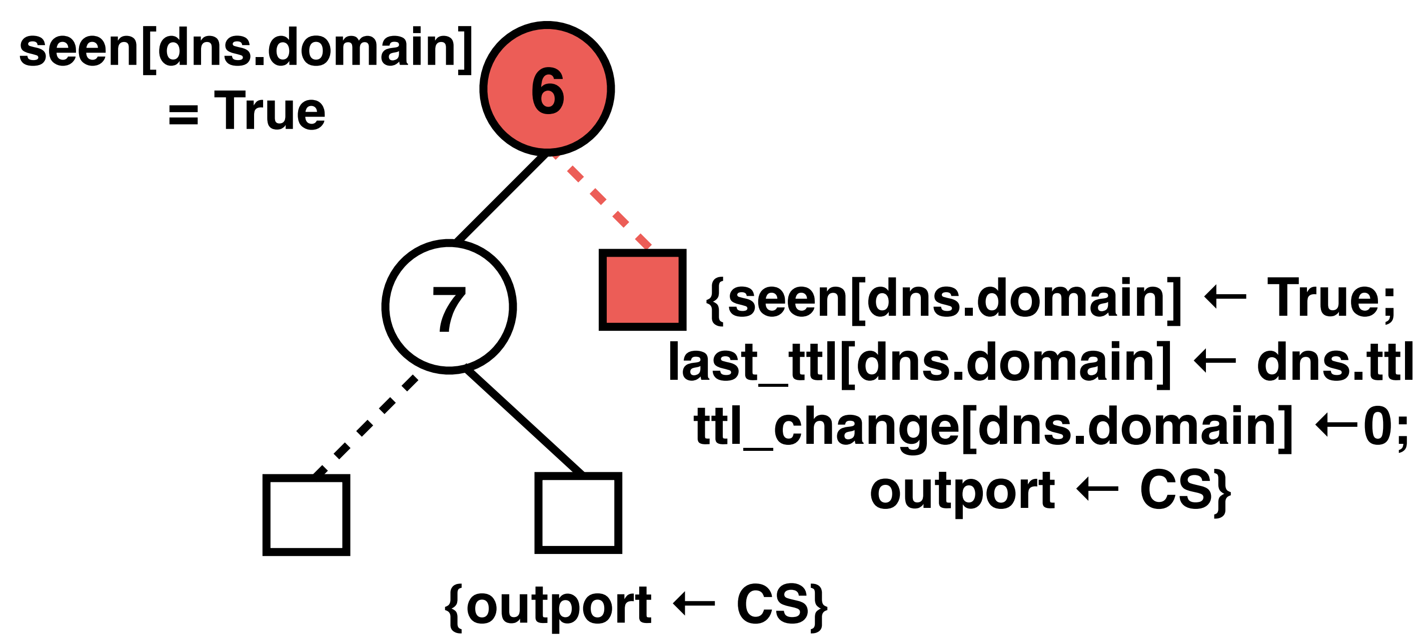


Putting It All Together

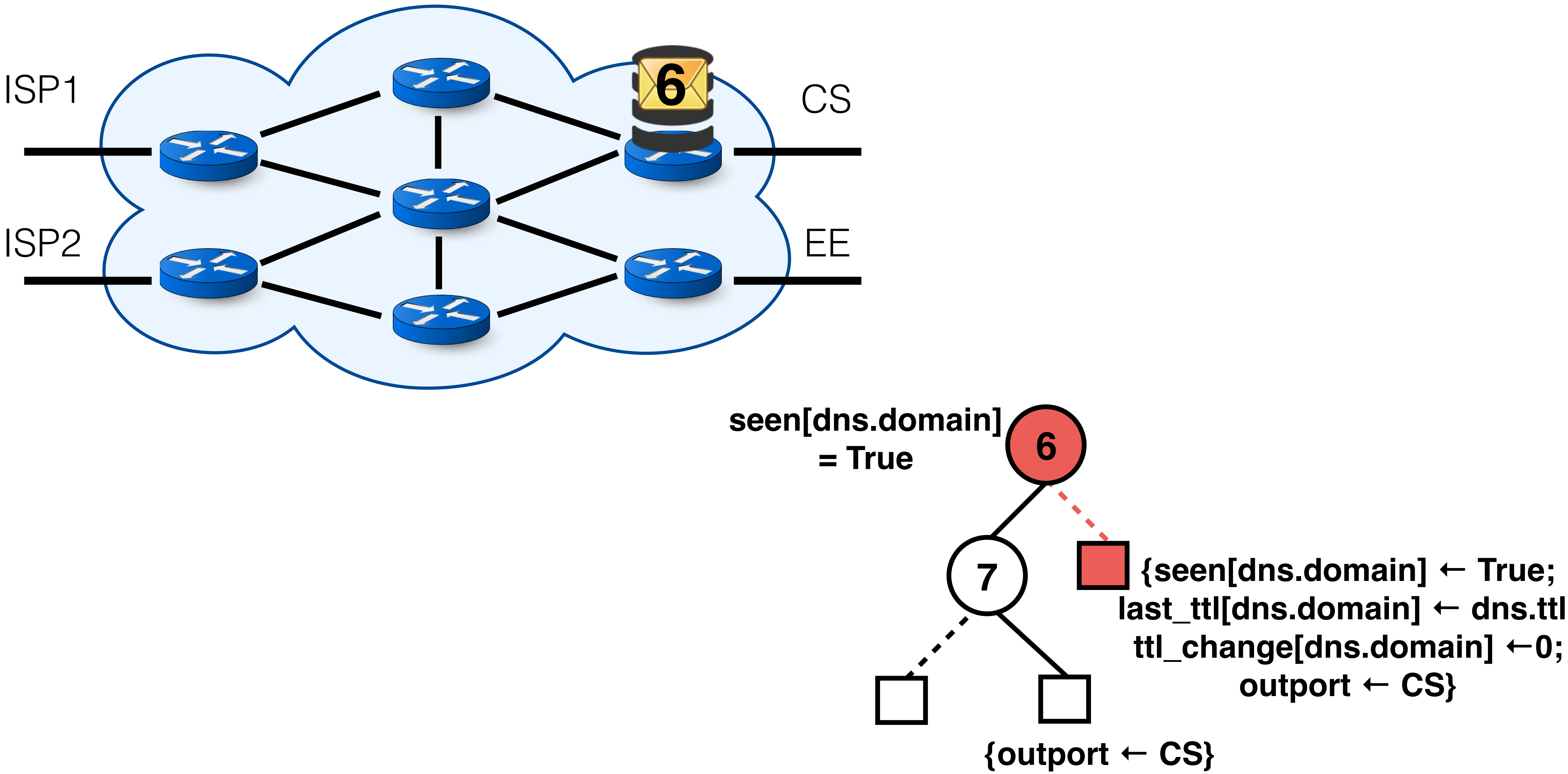


Putting It All Together





Putting It All Together



Evaluation

- Evaluated on three campus networks and four ASs
  - 25-160 switches
  - 100-650 links
- Cold-start compilation takes 35-600 seconds
  - most of the time goes for optimally distributing the FDD
- Re-compilation time can be reduced to under one minute by **fixing** state placement

Related Work

- **NetKAT**
  - inspired basic language constructs
- **Fast NetKAT Compiler**
  - stateless FDDs
- **Stateful NetKAT** (largely concurrent with SNAP)
  - simple registers (vs general dictionaries)
  - formal definition and proof of correctness for updates
  - Different optimization goal (rule space)

Questions?