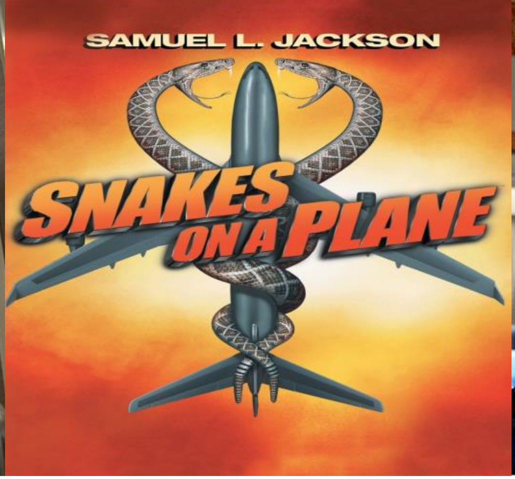


States on a (Data) Plane

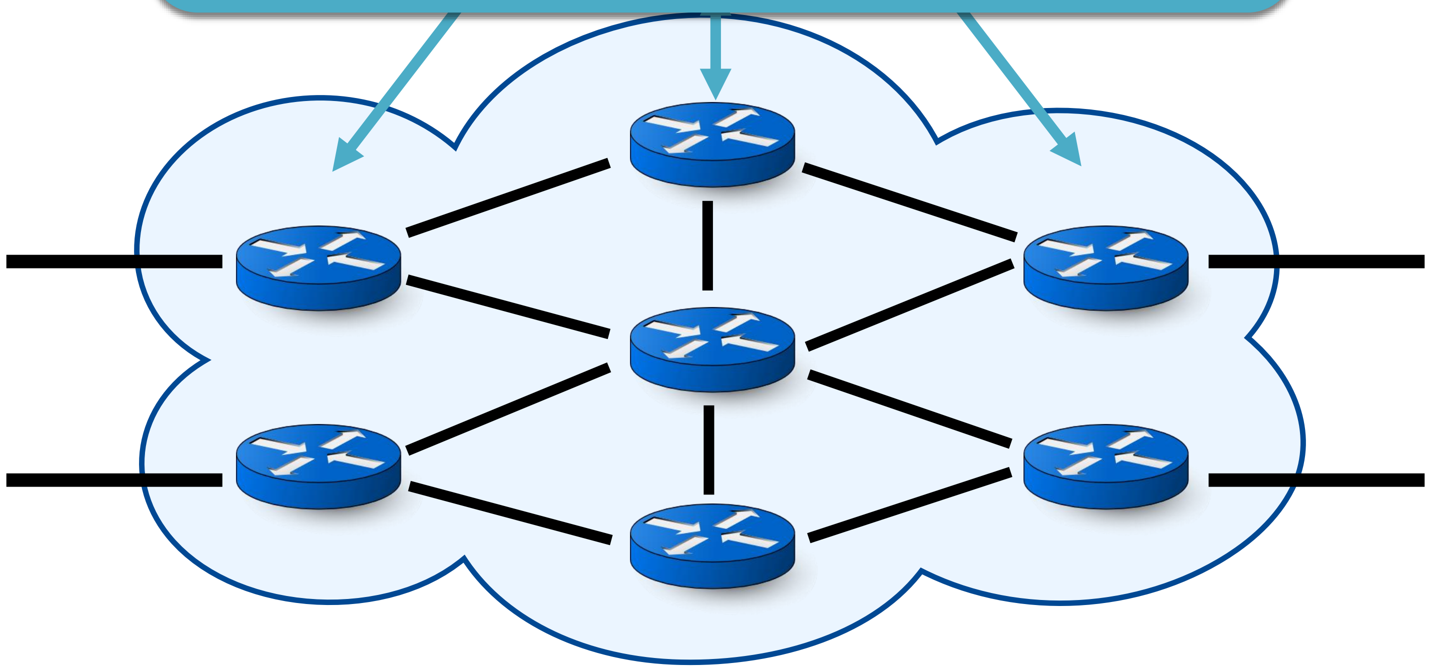
Jennifer Rexford

PRINCETON UNIVERSITY

Traditional data planes are stateless

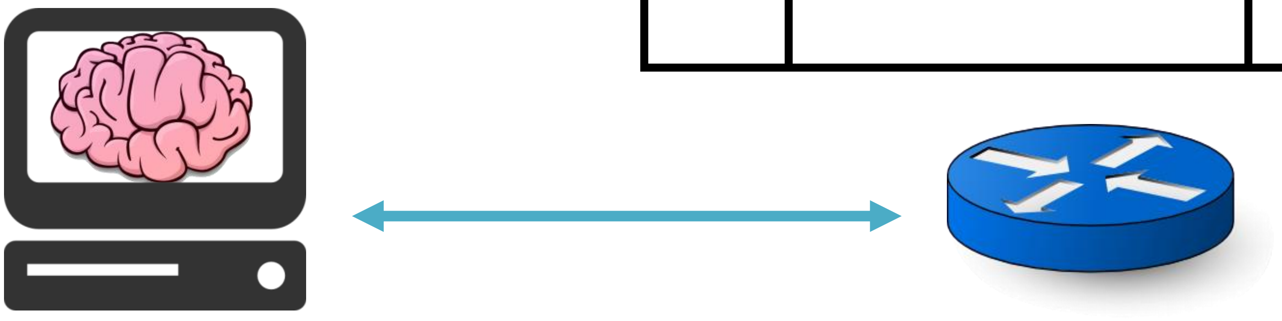
Software Defined Networks (SDN)

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



OpenFlow Rule Tables

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



Two-Tiered Programming Model

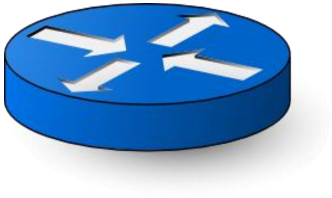
- **Stateless** data-plane rules
  - Process each packet independently
  - State updates are limited to traffic counters
- **Stateful** control-plane program
  - Store and update state in the controller application
  - Adapt by installing new rules in the switches

Forces packets to go to the controller…
or greatly limits the set of applications

Emerging switches have stateful data planes

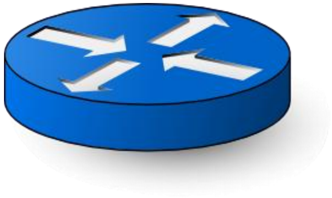
Local State on Data Plane

Key	Value
H2	5
H1	99
⋮	⋮



Local State on Data Plane

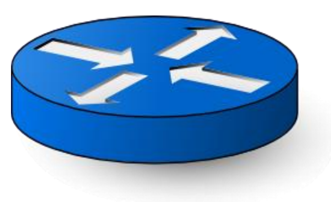
Key	Value
H2	5
H1	100
⋮	⋮



Local State on Data Plane

Key	Value
H2	5
H1	100
⋮	⋮

match	action
value = 100	drop
⋮	⋮



Local State on Data Plane

- Programmatic control over local state
  – P4, POF, OpenState, Open vSwitch
- Plus other important features
  – Programmable packet parsing
  – Simple arithmetic and boolean operations
  – Traffic statistics (delays, queue lengths, etc.)

- Simple stateful network functions can be offloaded to the data plane!

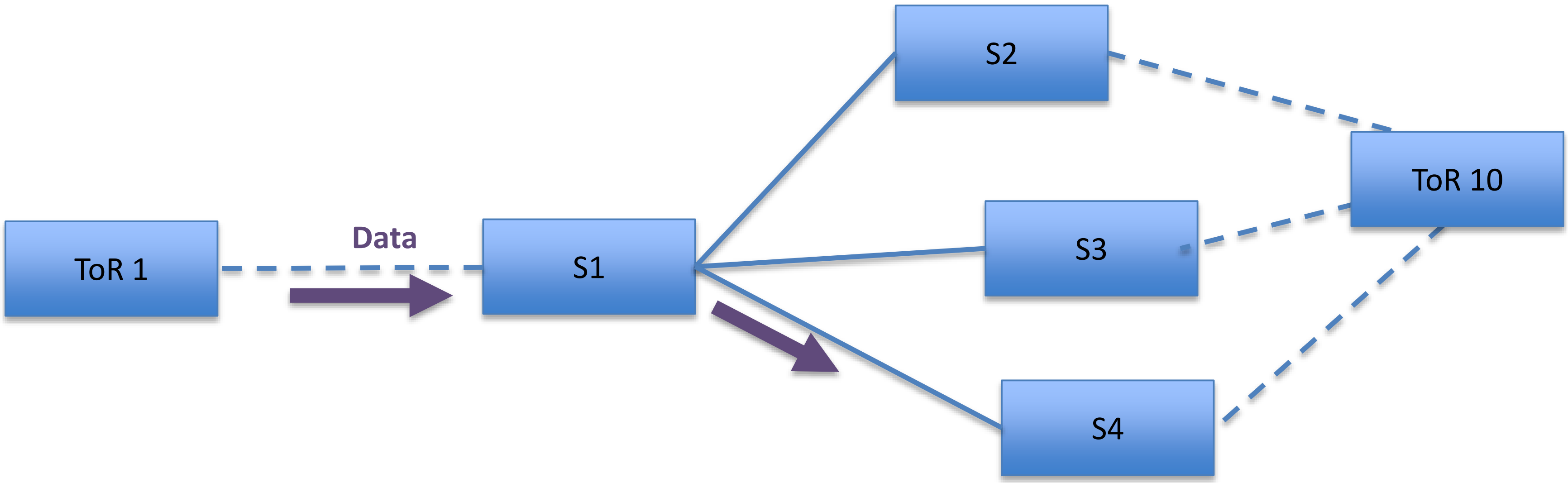
HULA

Hop-by-Hop Utilization-aware Load-balancing Architecture

Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford

http://conferences.sigcomm.org/sosr/2016/papers/sosr_paper67.pdf

HULA Multipath Load Balancing

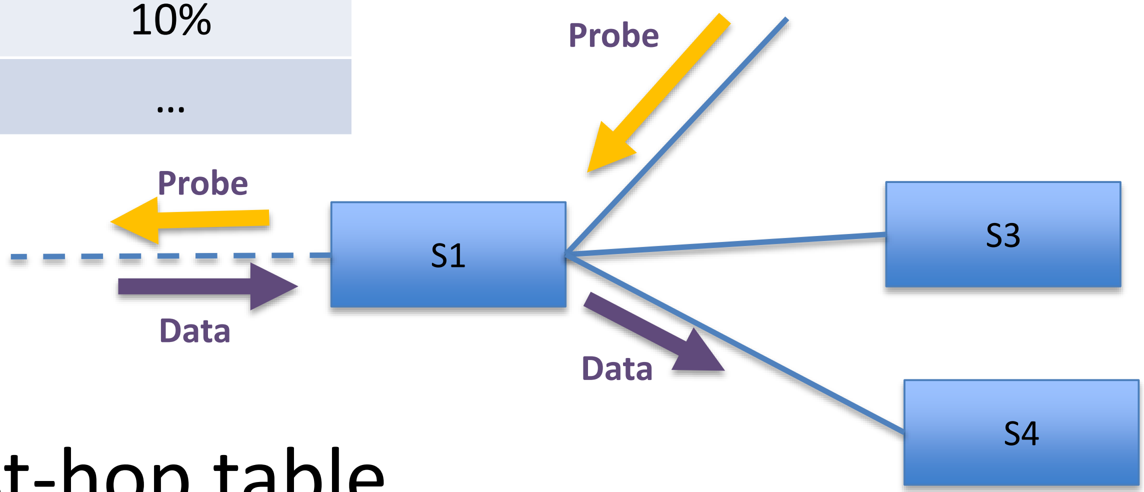


- Load balancing *entirely* in the data plane
  - Collect real-time, path-level performance statistics
  - Group packets into "flowlets" based on time & headers
  - Direct each new flowlet over the current best path

Path Performance Statistics

Best-hop table

Dest ToR	Best Next-Hop	Path Utilization
0	S3	50%
1	S4	10%
…	…	…

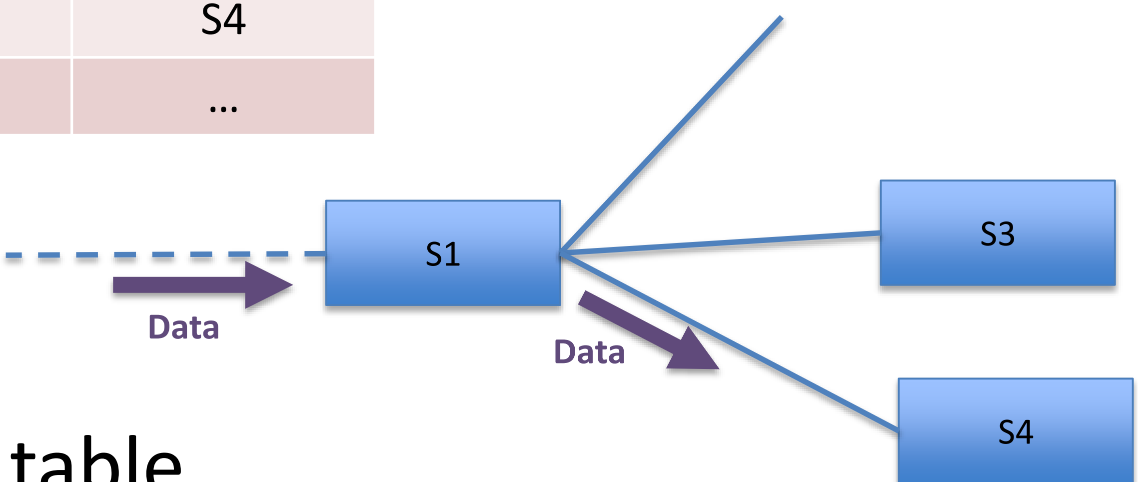


- Using the best-hop table
  - *Update* the best next-hop upon new probes
  - *Assign* a new flowlet to the best next-hop

Flowlet Routing

Flowlet table

h(flowid)	Dest ToR	Timestamp	Next-Hop
0	ToR 10	1	S2
1	ToR 0	17	S4
…	…	…	…



- Using the flowlet table
  - *Update* the next hop if enough time has elapsed
  - *Update* the timestamp to the current time
- *Forward* the packet to the chosen next hop

Putting it all Together

data packet

Dest ToR	Best Next-Hop	Path Utilization
0	S3	50%
1	S4	10%
…	…	…

current best next-hop S3

h(flowid)	Dest ToR	Timestamp	Next-Hop
0	ToR 10	1	S2
1	ToR 0	17	S4
…	…	…	…

Update next-hop (if enough time elapsed) and time

chosen next-hop

Plenty of Other Applications

- Stateful firewall
- DNS tunnel detection
- SYN flood detection
- Elephant flow detection
- DNS amplification attack detection
- Sidejack detection
- Heavy-hitter detection
- …

But, how to best *write* these stateful apps?

SNAP: Stateful Network-Wide Abstractions for Packet Processing

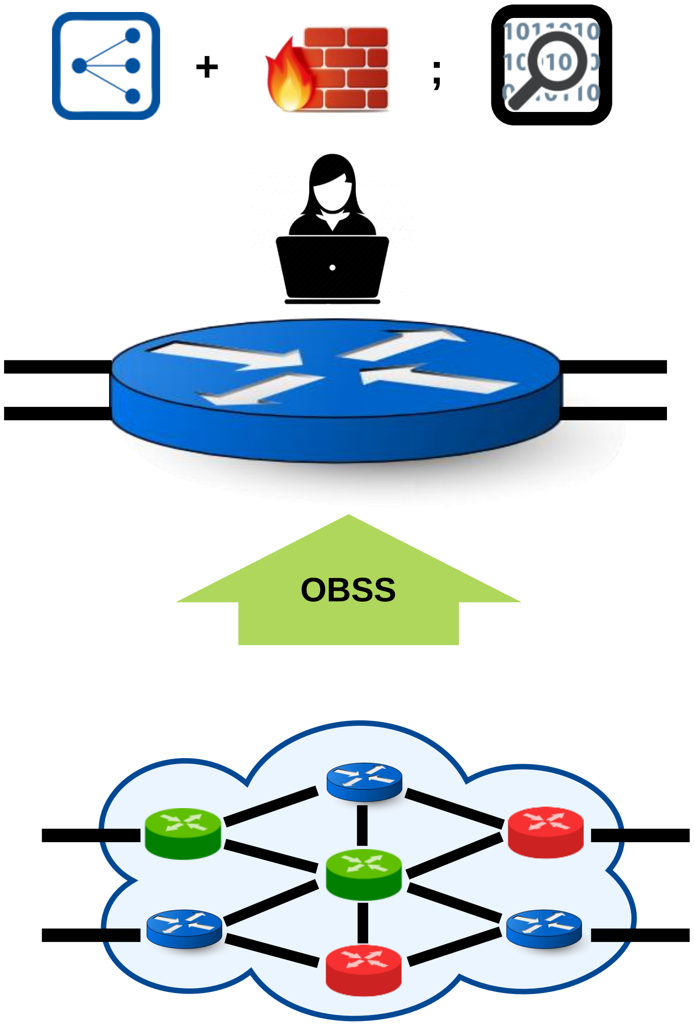
Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker

http://www.cs.princeton.edu/~jrex/papers/snap16.pdf

Writing Stateful Network Apps is Hard

- Low-level switch interface
  - Multiple stages of match-action processing
  - Registers/arrays for maintaining state
- Multiple switches
  - Placing the state
  - Routing traffic through the state
- Multiple applications
  - Combining forwarding, monitoring, etc.

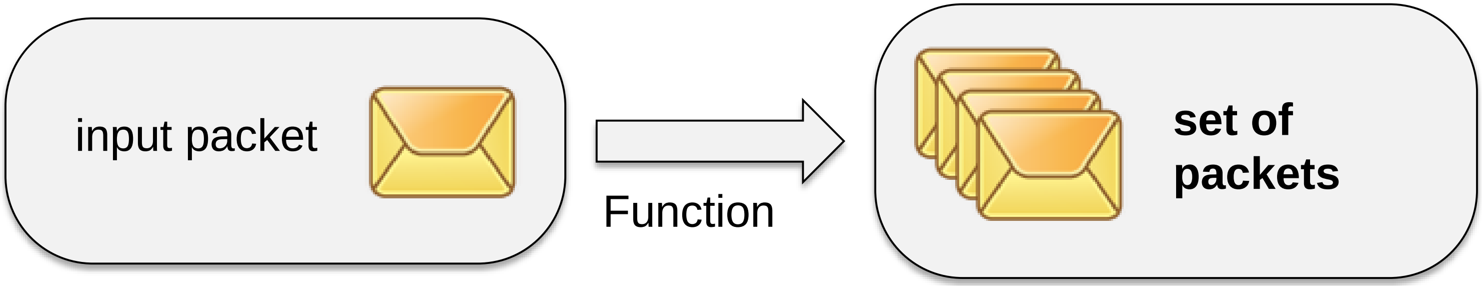
Snap Language



- Hardware independent
- One Big Stateful Switch **(OBSS)**
- Composition

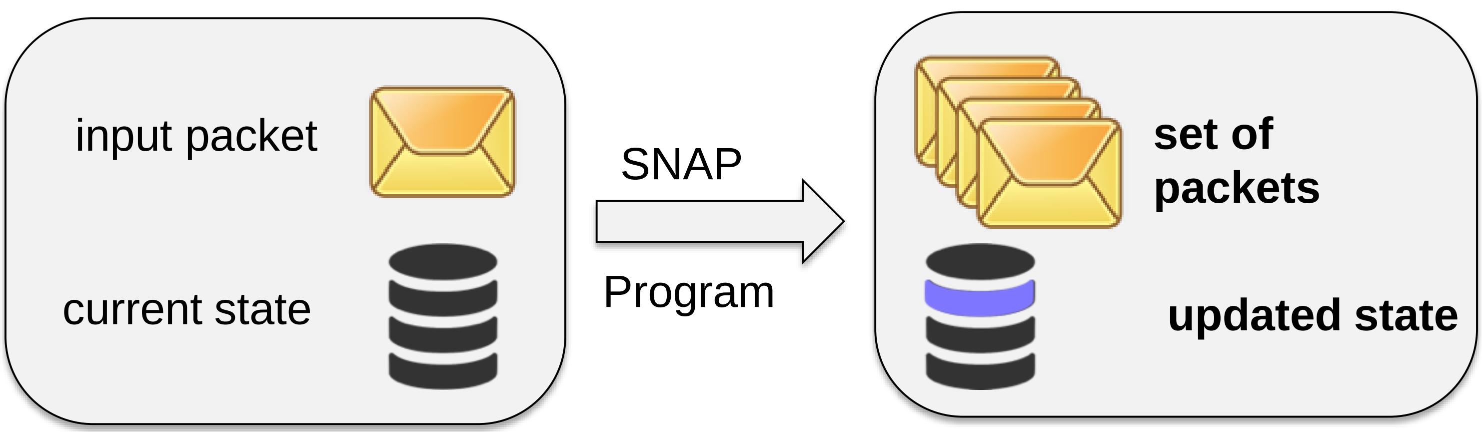
Stateless Packet Processing

- A function that specifies
  - How to process each packet on a one-big-switch
  - Based on its **fields**
- E.g., NetKat



Stateful Packet Processing

- A function that specifies
  - How to process each packet on a one-big-switch
  - Based on its **fields** and the **program state**
  - Where state is an **array** indexed by header fields



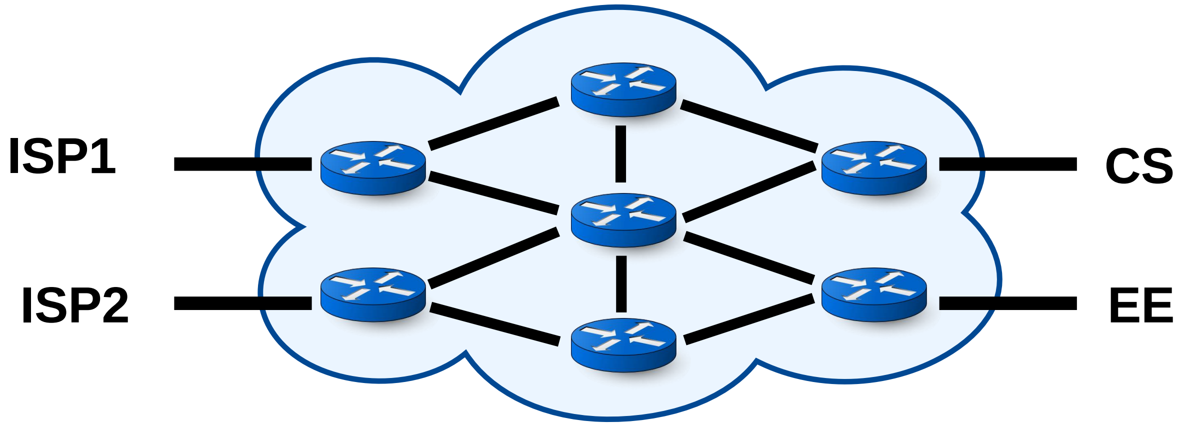
Example Snap App: DNS Reflection

```
if srcip in CSNET & dstport = 53 then
   seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++;
      if unmatched[dstip] = threshold then
         susp[dstip] ← True
   else id
else id
```

- Seen: Keep track of DNS requests by client and DNS identifier
- Unmatched: Count DNS responses that don't match prior requests
- Susp: Suspected victims receive many unmatched responses

Example Snap App: Stateless Forwarding

```
if dstip = CSNET then outport ← CS
else if dstip = EENET then outport ← EE
else if dstip = ISP1NET then outport ← ISP1
else if dstip = ISP2NET then outport ← ISP2
else drop
```



Composition

```
if srcip in CSNET & dstport = 53 then
   seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++;
      if unmatched[dstip] = threshold then
         susp[dstip] ← True
   else id
else id
```

;

```
if dstip = CSNET then outport ← CS
else if dstip = EENET then outport ← EE
else if dstip = ISP1NET then outport ← ISP1
else if dstip = ISP2NET then outport ← ISP2
else drop
```

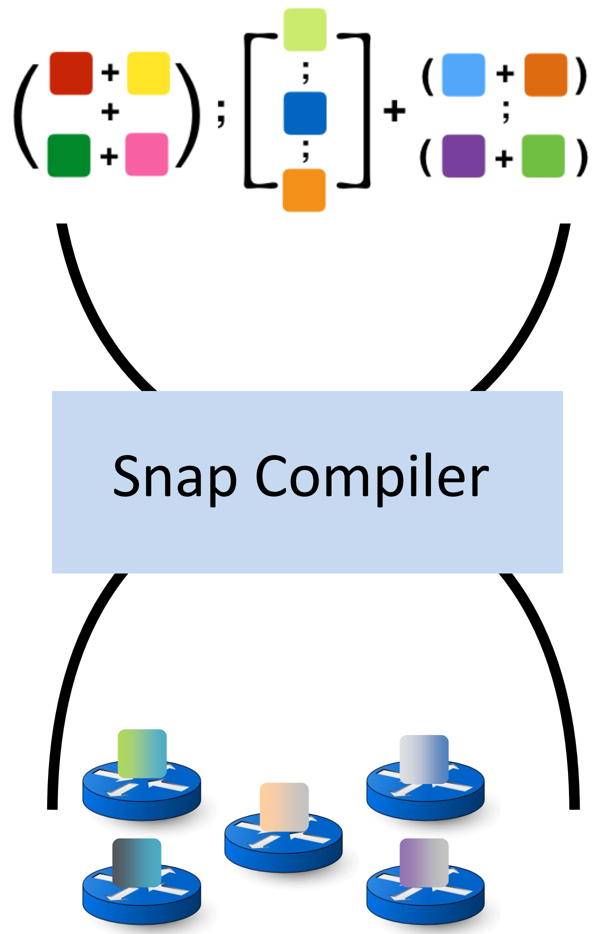
Snap Applications

Source	Application
Chimera (USENIX Security'12)	Number of domains sharing the same IP address Number of distinct IP addresses under the same domain DNS TTL change tracking DNS tunnel detection Sidejack detection Phishing/spam detection
FAST (HotSDN'14)	Stateful firewall FTP monitoring Heavy-hitter detection Super-spreader detection Sampling based on flow size Selective packet dropping (MPEG frames) Connection affinity
Bohatei (USENIX Security'15)	SYN flood detection DNS reflection (and amplification) detection UDP flood mitigation Elephant flows detection
Others	Bump-on-the-wire TCP state machine Snort flowbits

Snap Compiler

Composition of multiple apps

State placement and routing



Snap Compiler

Identify State Dependencies

Translate to Intermediate Representation (xFDD)

Identify mapping from packets to state variables

Optimally distribute the xFDD

Generate rules per switch

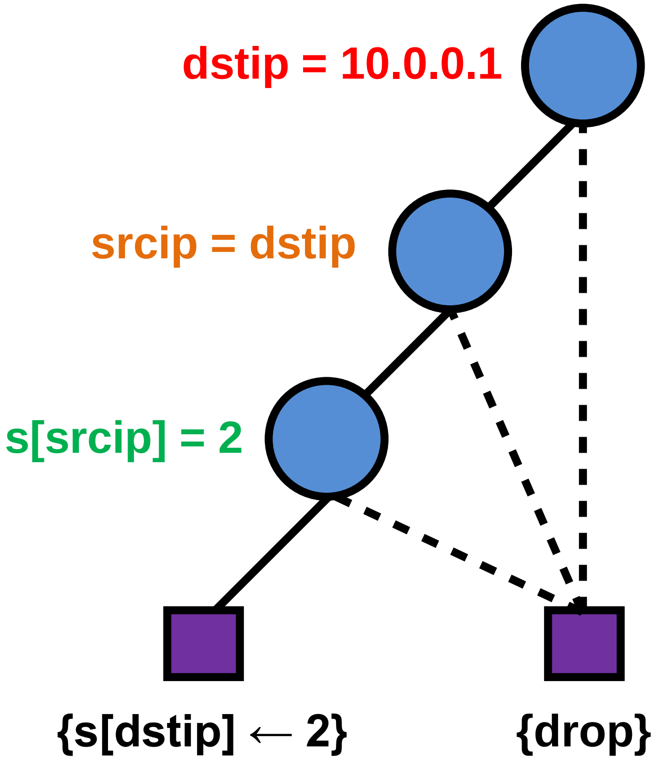
Intermediate Representation: xFDDs

- Canonical representation of a program
- Composable
- Easily partitioned
- Simplify program analysis

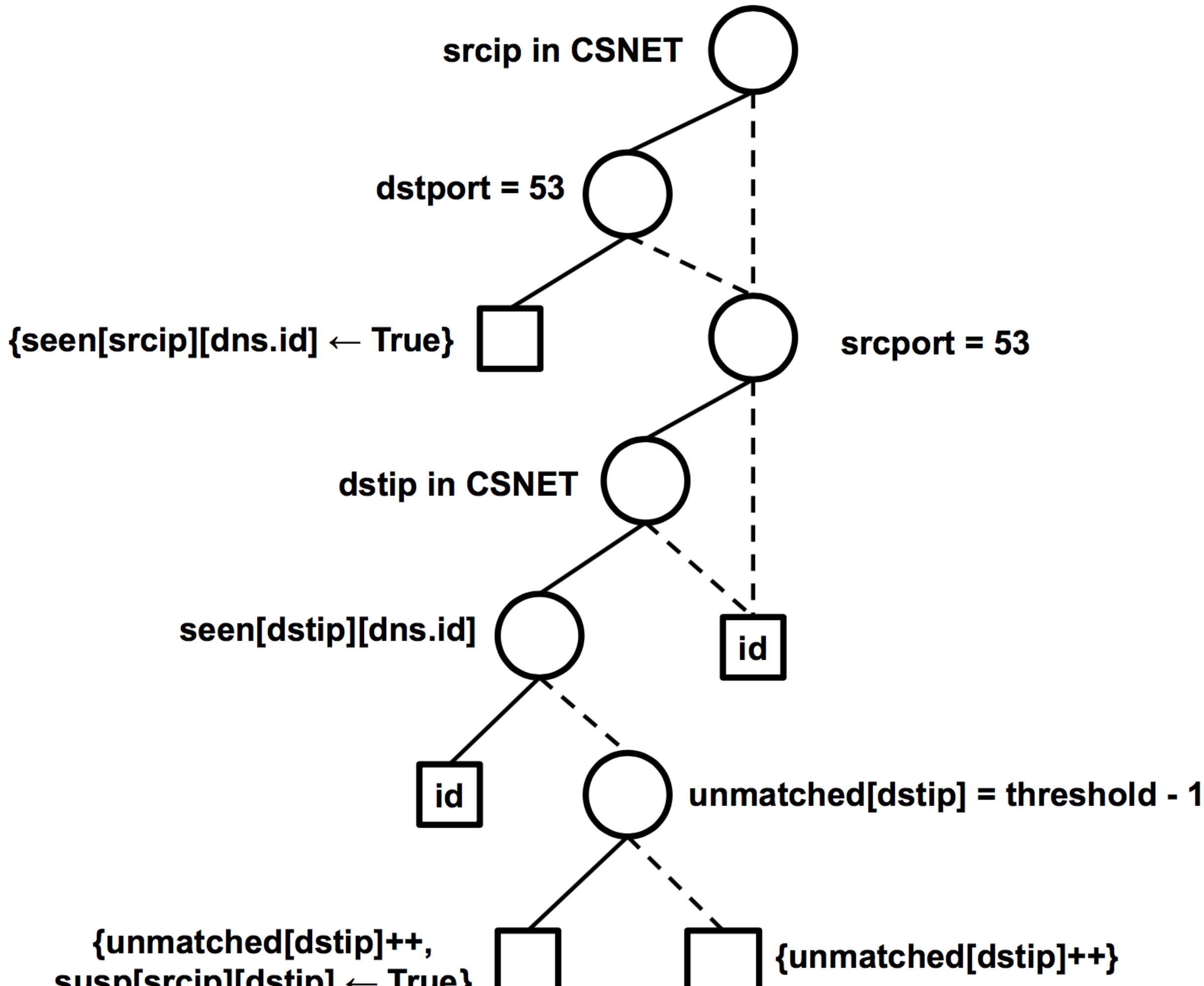
Extended Forwarding Decision Diagrams (xFDDs)

- Intermediate node: test on header fields and state
- Leaf: set of action sequences

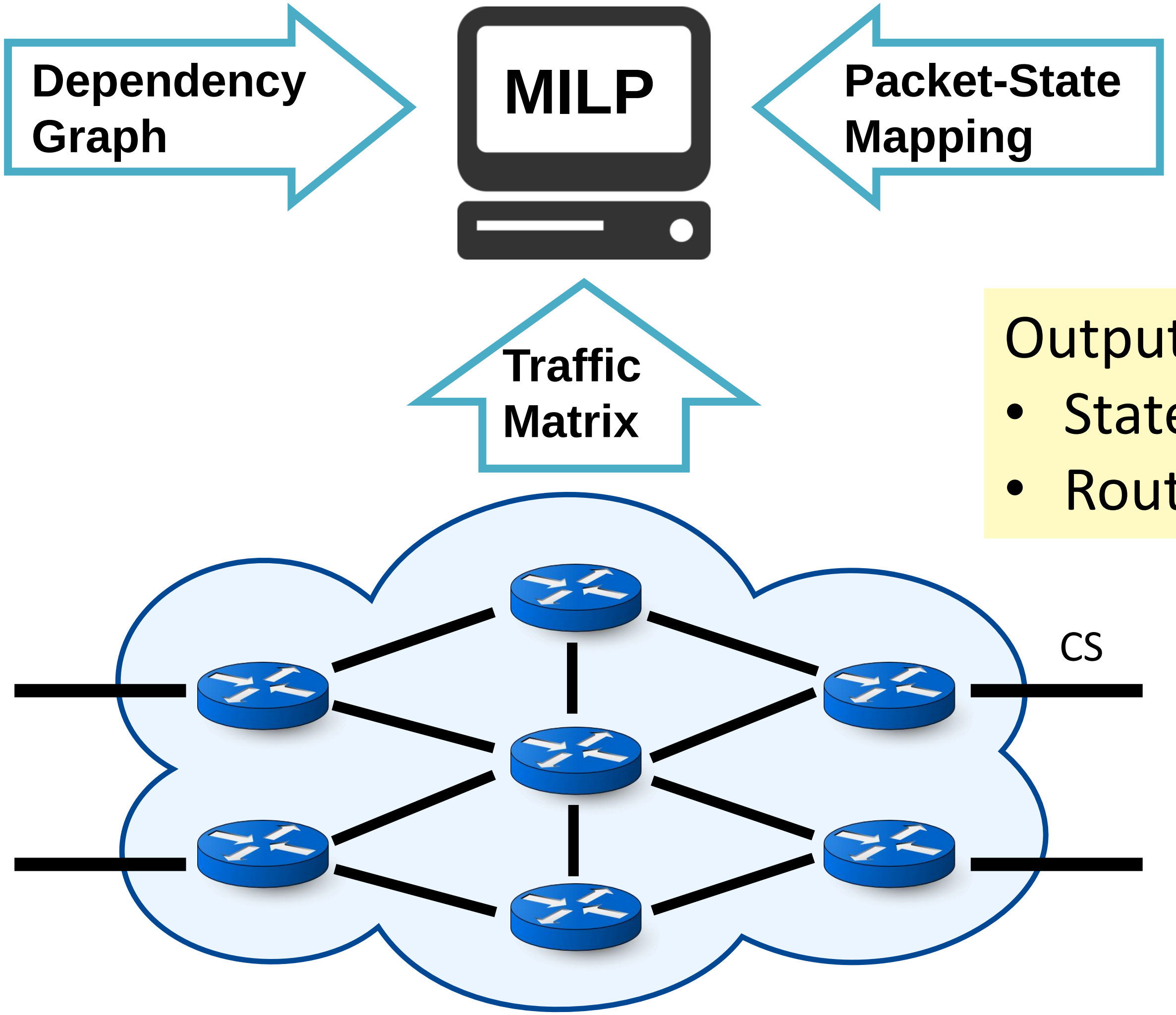
- Three kinds of tests
  - field = value
  - $field_1 = field_2$
  - $state\_var[e_1] = e_2$



xFDD for DNS Reflection Detection



Optimally Distribute the xFDD



Output
- State placement
- Routing

See SIGCOMM'16 paper for prototype, experiments, etc.

http://www.cs.princeton.edu/~jrex/papers/snap16.pdf

More Fun With State

- Extending Snap
  - More operations, e.g., field ← state[index]
  - Sharding and replication of state
  - Faster compilation
- Richer computational model
  - Limits on computation per packet
  - Different memory (array, hash table, key-value store)
  - Hash collisions, delays in adding new keys, etc.
- More stateful applications!

Conclusion

- Emerging switches have stateful data planes
  - Can run simple network functions
  - … within and across switches!
- Standard interfaces
  - E.g., P4 (p4.org)
- Raises many new algorithmic challenges
  - New computational model
  - Compact data structures (e.g., sketches)
  - Working within hardware limitations