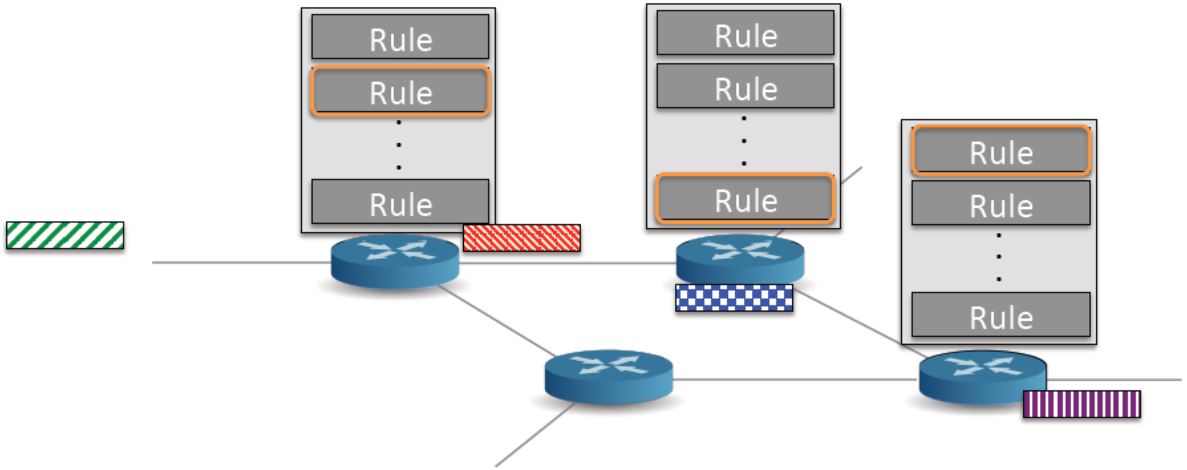
Real Time Network Policy Checking using Header Space Analysis

Peyman Kazemian, Michael Chang, Hongyi Zeng,
George Varghese, Nick McKeown, Scott Whyte

Presenter: Chang Chen
December 18, 2013

Network Debugging is Hard!

- Forwarding state is hard to analyze

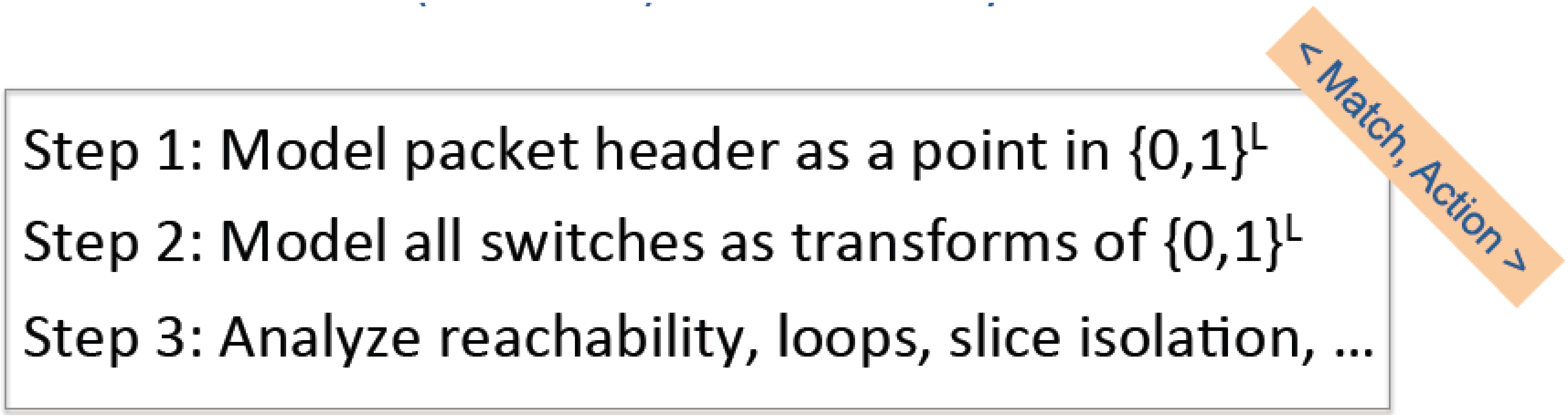


Verification

- □ Distributed across multiple tables and boxes
- □ Written to network by multiple independent writers
- □ Presented in different formats by vendors

Prior Work: Header Space Analysis

- In today’s networks, simple questions are hard.
  - Can A talk to B?
  - What are all the packet headers from A that can reach B?
  - Loops? Isolation?



Step 1: Model packet header as a point in $\{0,1\}^L$

Step 2: Model all switches as transforms of $\{0,1\}^L$

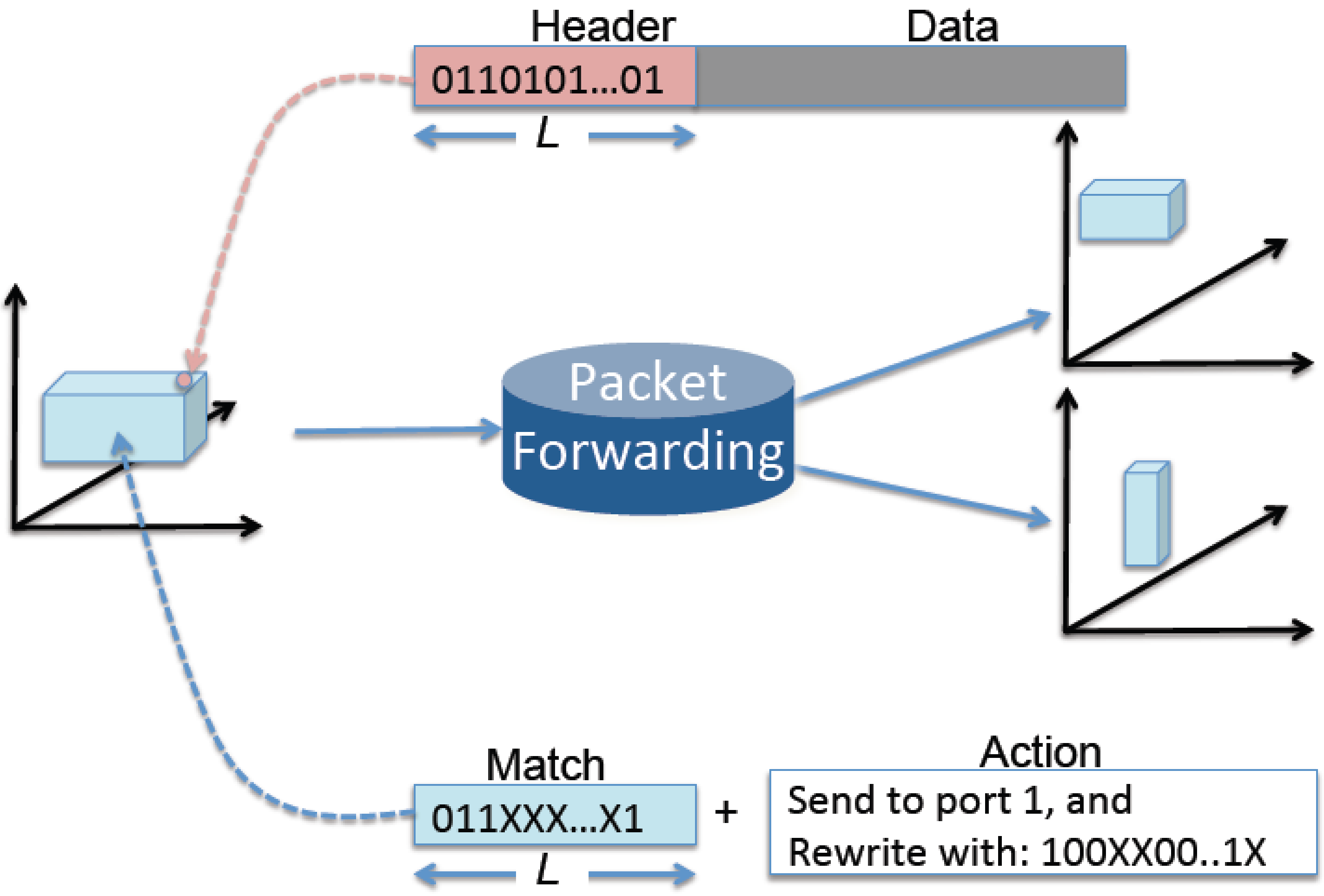
Step 3: Analyze reachability, loops, slice isolation, …

< Match, Action >

- Protocol independent, general.

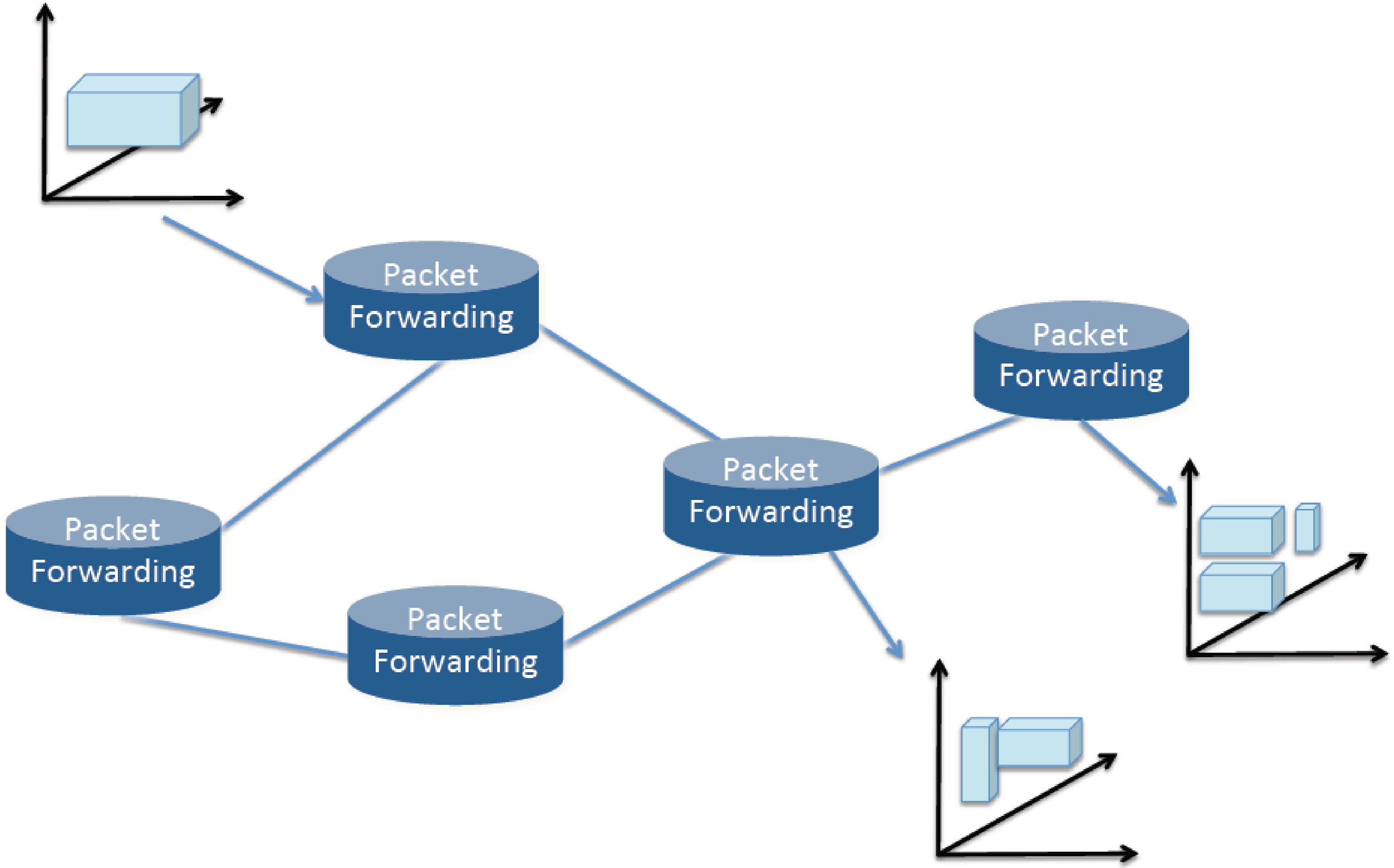
Prior Work: Header Space Analysis

- Basic Model



Prior Work: Header Space Analysis

- Network Transfer Function

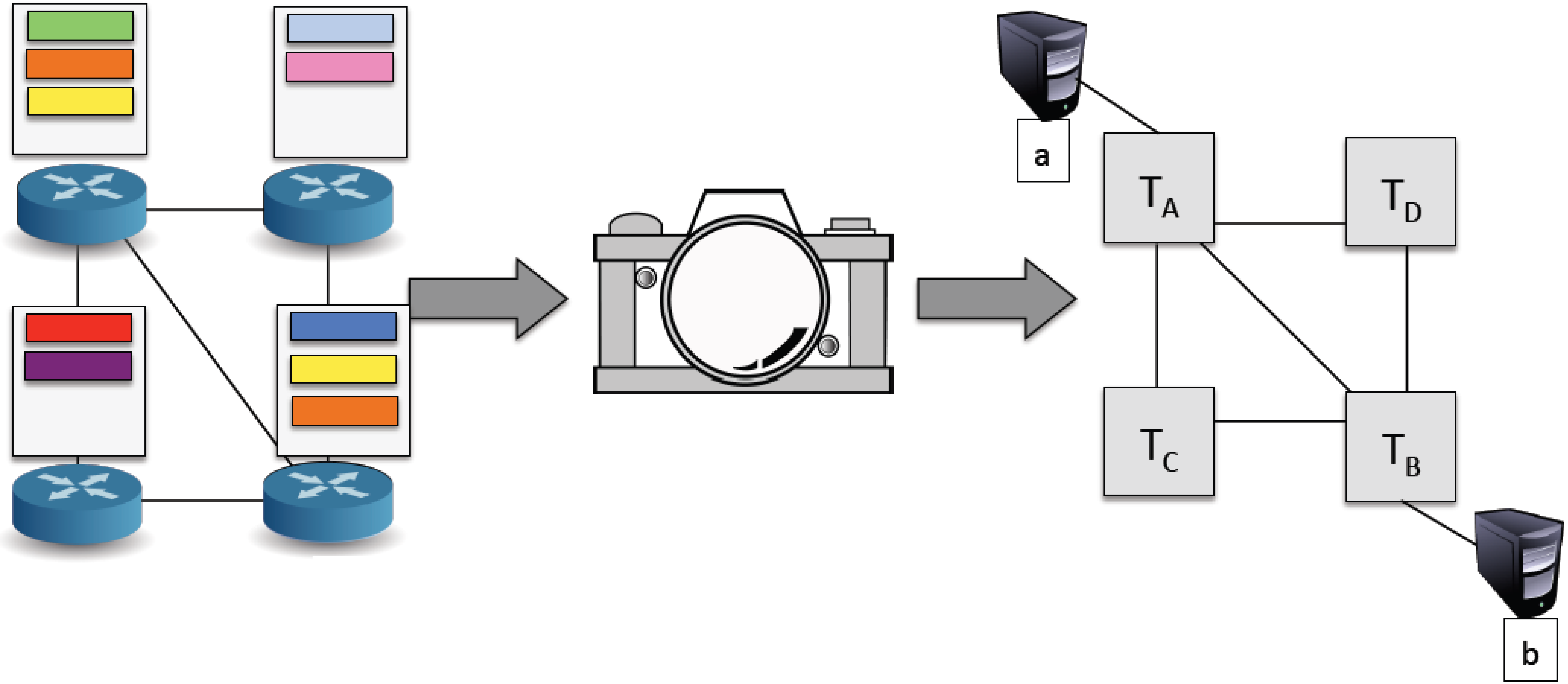


Prior Work: Header Space Analysis

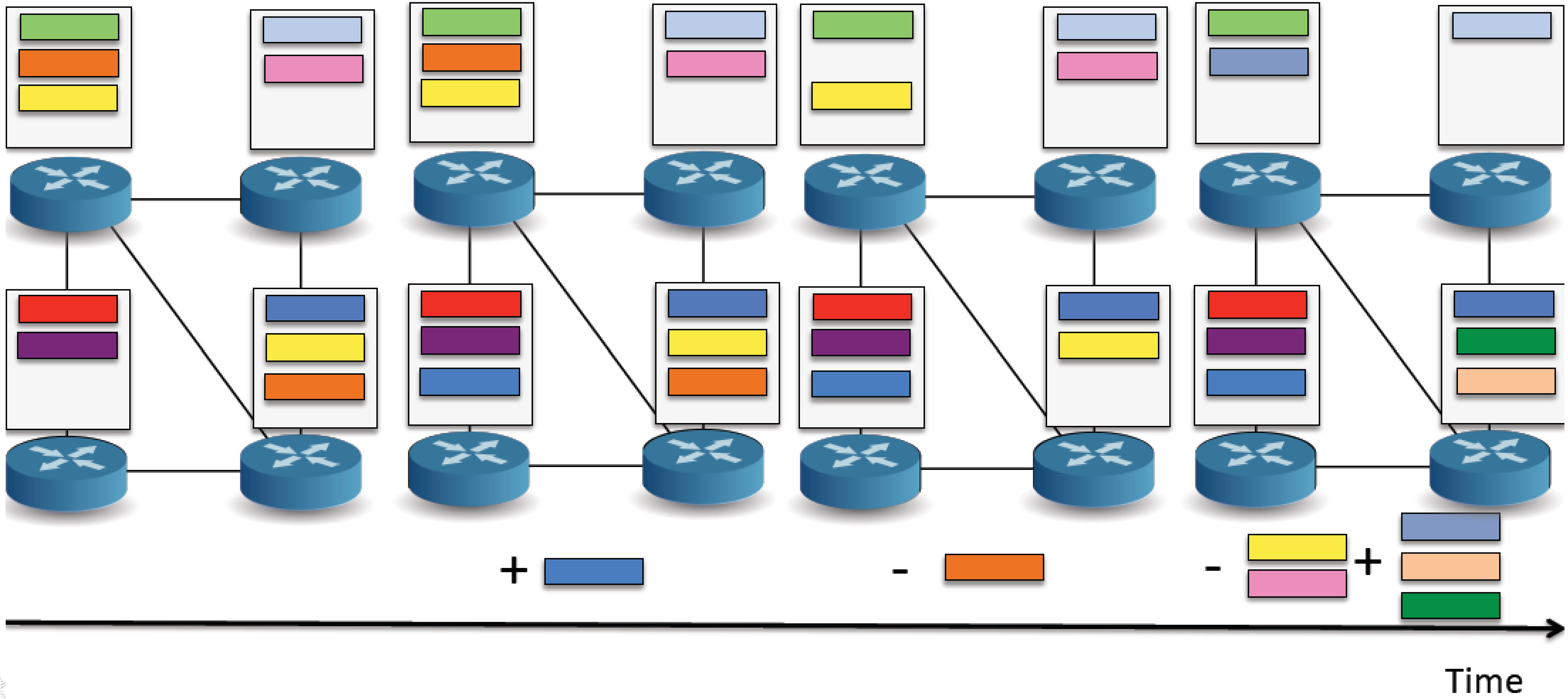
- Properties
  - Network Transfer Function: set of Boolean expressions
  - Only relies on <Match, Action>
    - Subsumes Ethernet, IPv4, firewalls, NAT, …
  - Can prove reachability, isolation and find loops
  - Used to find faults in real networks
    - e.g. Analyzed Stanford backbone in 10mins
  - Code publicly available

Prior Work: Header Space Analysis

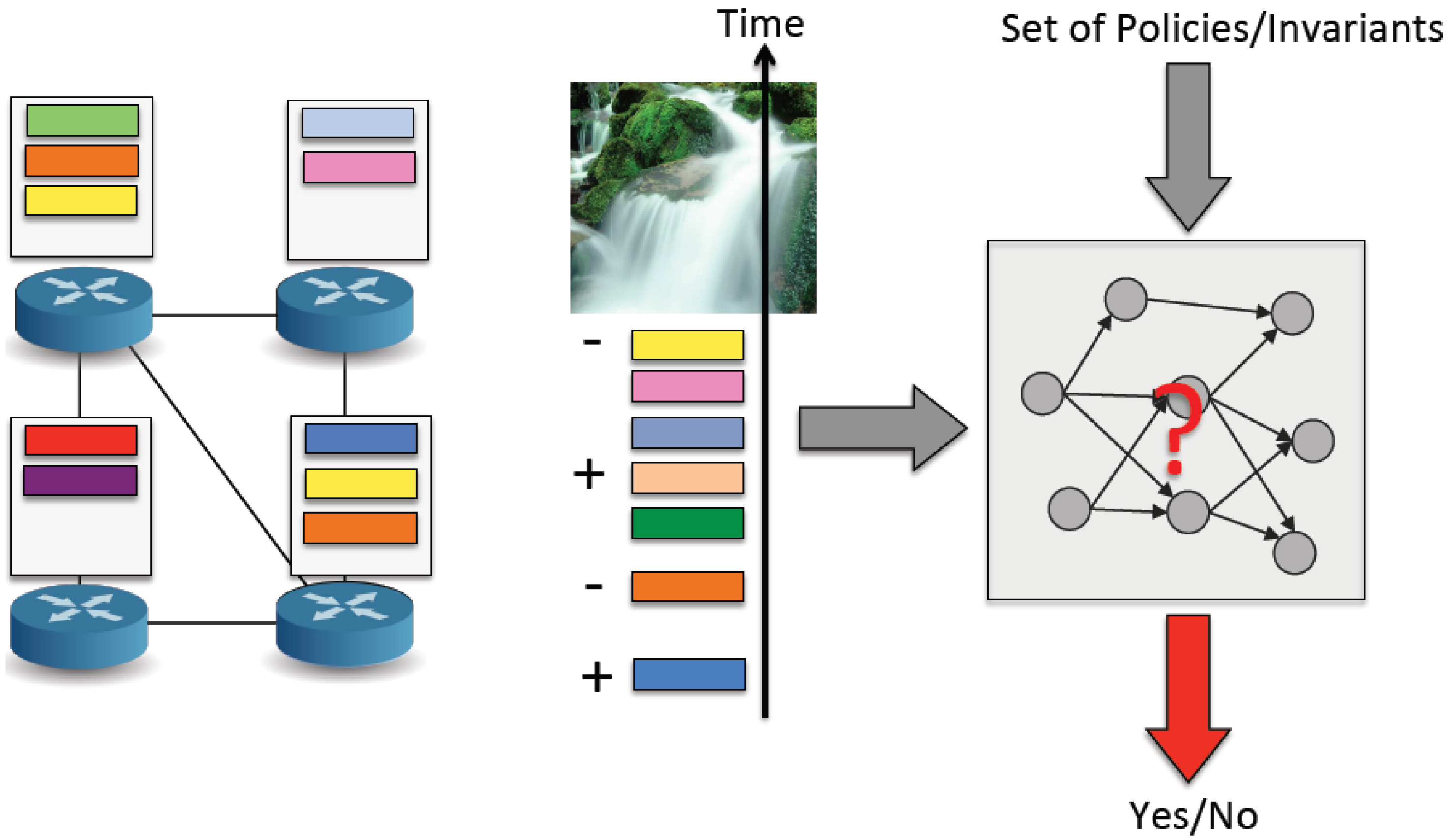
- Snapshot-based Checking



Real Time Policy Checking?



Real Time Policy Checking?



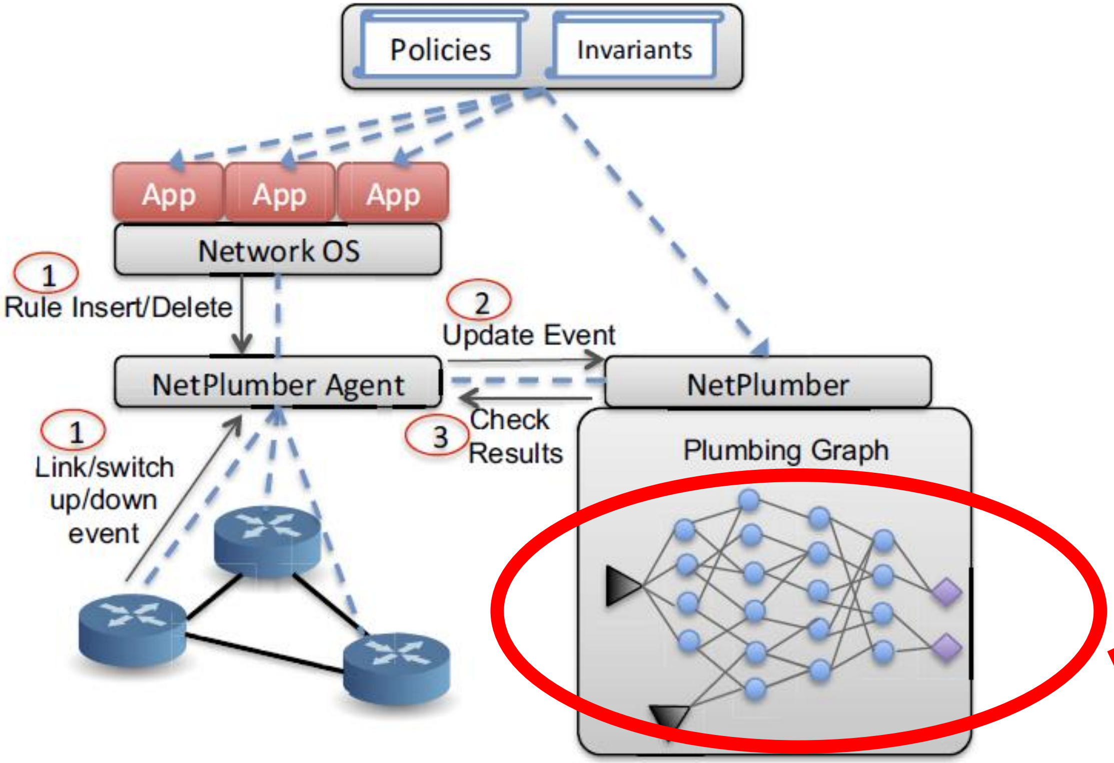
- Prevent errors before they hit network.
- Report a violation as soon as it happens.

Outline

- Background
- NetPlumber: Real time policy checking tool
  - □ How it works?
  - □ How to check policy?
  - □ How to parallelize?
- Evaluation
- Conclusion

NetPlumber

- NetPlumber: Real time policy checking tool



Deploying NetPlumber as a policy checker in SDNs

1. Observe state changes (installation or removal of rules, link up or down events)
2. Update event (NetPlumber in turn updates internal model of the network)
3. Check polices

Heart of NetPlumber

Plumbing Graph — Nodes and Edges

- *Plumbing graph* captures all possible paths of flows through the network.
- **Nodes**: forwarding rules in the network.
  - Rule: OpenFlow-like `<match, action>` tuple where the action can be `forward`, `drop`, `rewrite`, `encapsulate`, `decapsulate`, etc.
- **Directed Edges**: next hop dependency of rules.

pipes

  - Rule A has a next hop dependency to rule B if

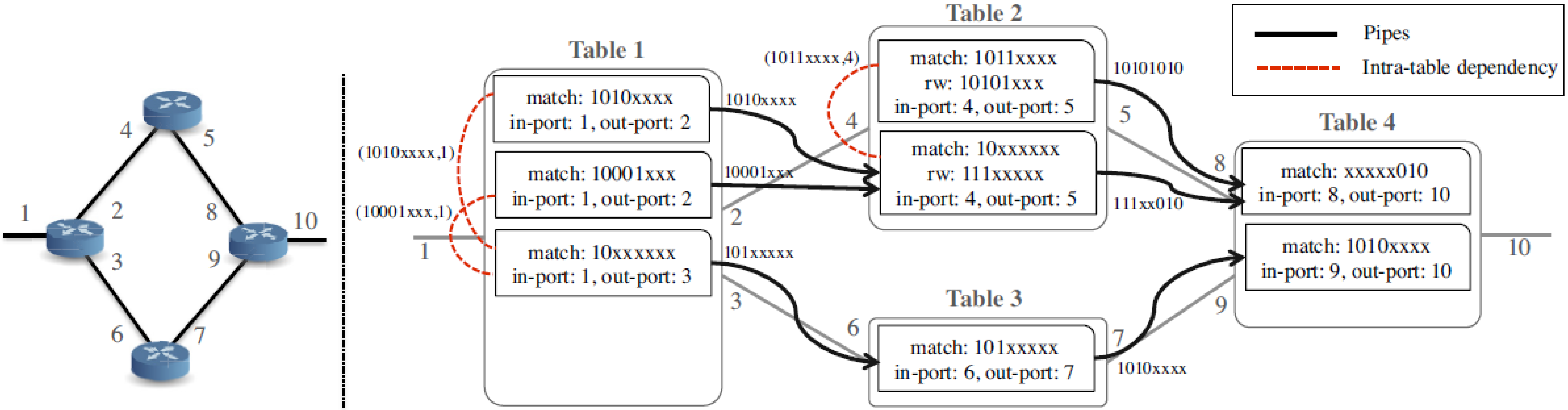
    1) there is a physical link from A’s box to B’s box; *and*

    2) A.range has an intersection with B.domain.

Plumbing Graph — Intra-table Dependency

- *Intra-table dependency* of rules
  - Plumbing graph needs to consider rule priorities.
  - Each rule node keeps track of higher priority rules in the same table.
  - It calculates the domain of each higher priority rule, subtracting it from its own domain.

Plumbing Graph

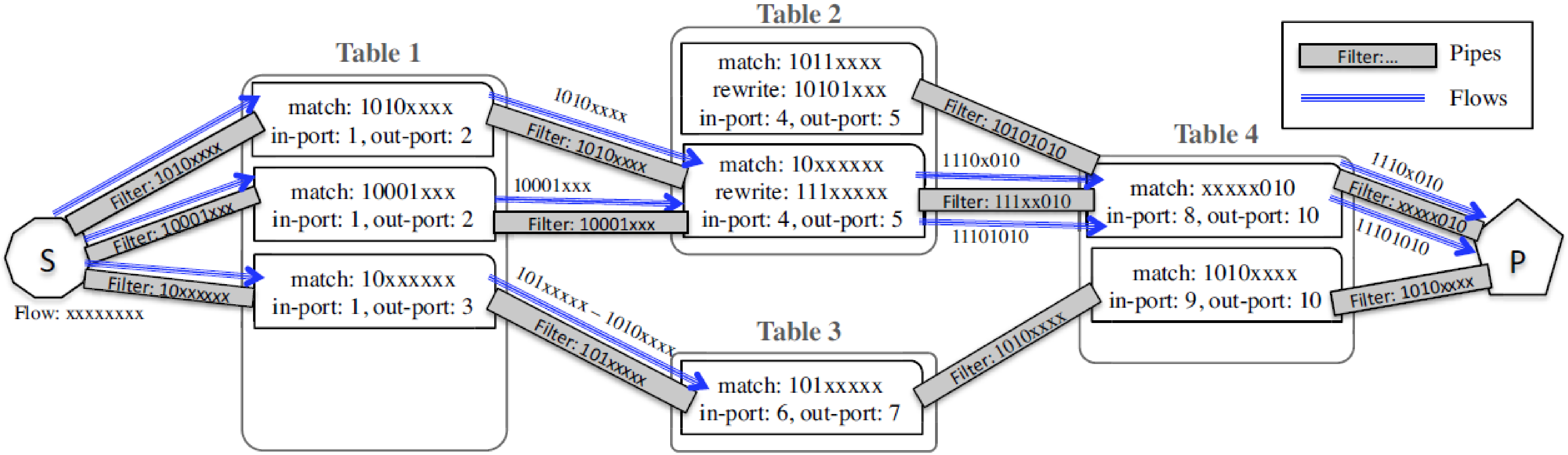


Plumbing graph of a simple network consisting of 4 switches each with one table.

NetPlumber — Source and Probe nodes

- To compute reachability, NetPlumber inserts flow from the source port into the plumbing graph and propagates it towards the destination.
- **Source Node**: “flow generator”, all-wildcard headers.
  - Sink Nodes: the dual of source nodes.
- **Probe Node**: “flow monitor”.
  - can be attached to appropriate locations of the plumbing graph.

NetPlumber — Computing Reachability

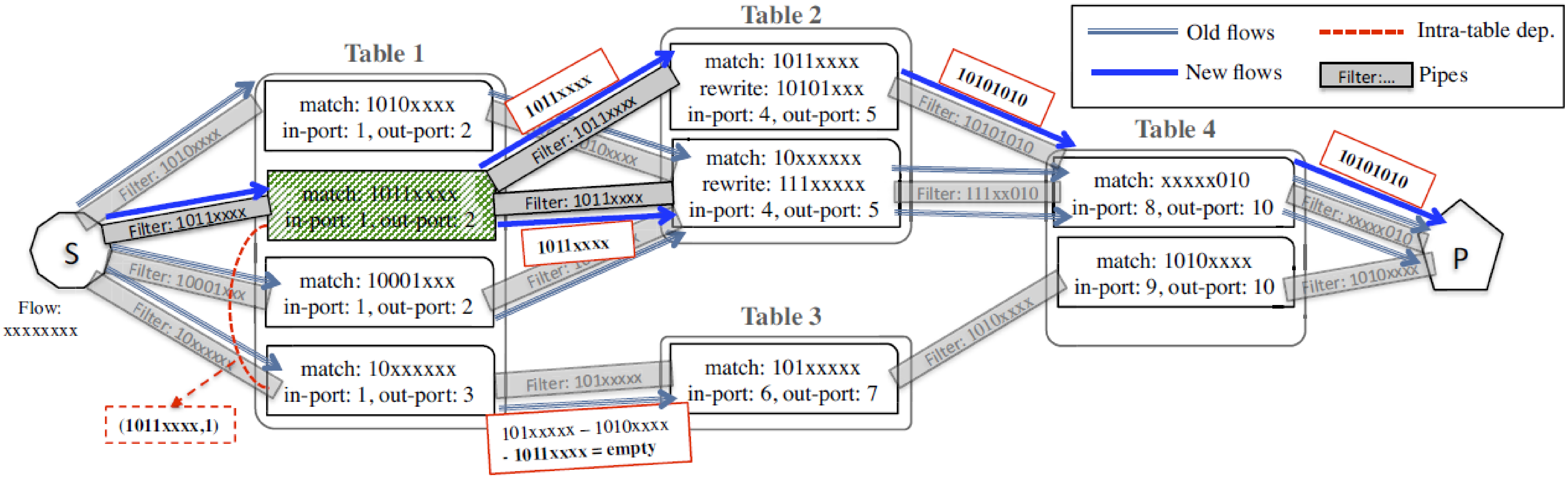


Finding reachability between S and P.

NetPlumber — Updating State

- As events occur in the network, NetPlumber needs to update its plumbing graph and re-route the flows.
- Such events include:
  - Adding new rules
  - Deleting rules
  - Link up
  - Link down
  - Adding new tables
  - Deleting tables

NetPlumber — Updating State

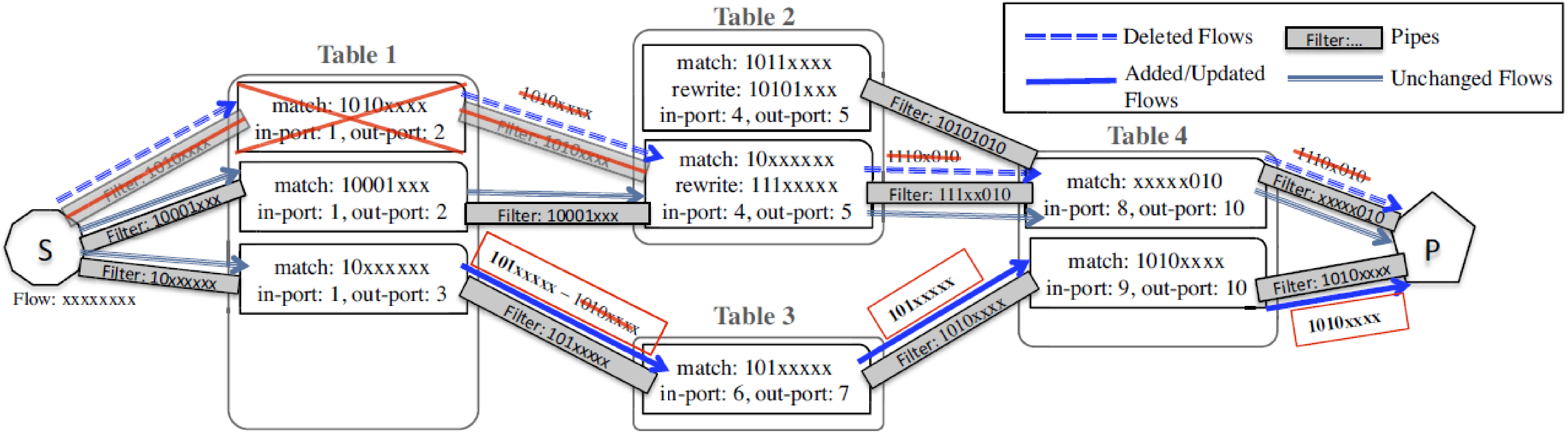


Adding rule 1.2 (shaded in green) to table 1.

Results:

a) 3 new pipes
b) 1 new intra-table dependency
c) New flows added (highlighted in bold)
d) Flows deleted

NetPlumber — Updating State



Deleting rule 1.1

- Complexity for the addition or deletion of a single rule: O(r+spd)

r: entry # in each table, s: source node #, p: pipe # connected, d: network diameter

NetPlumber

- Each flow at any point in the plumbing graph, carries its complete history.

- By traversing backward, we can examine

  a) the entire history of the flow;

  b) all the rules that have processed this flow along the path.

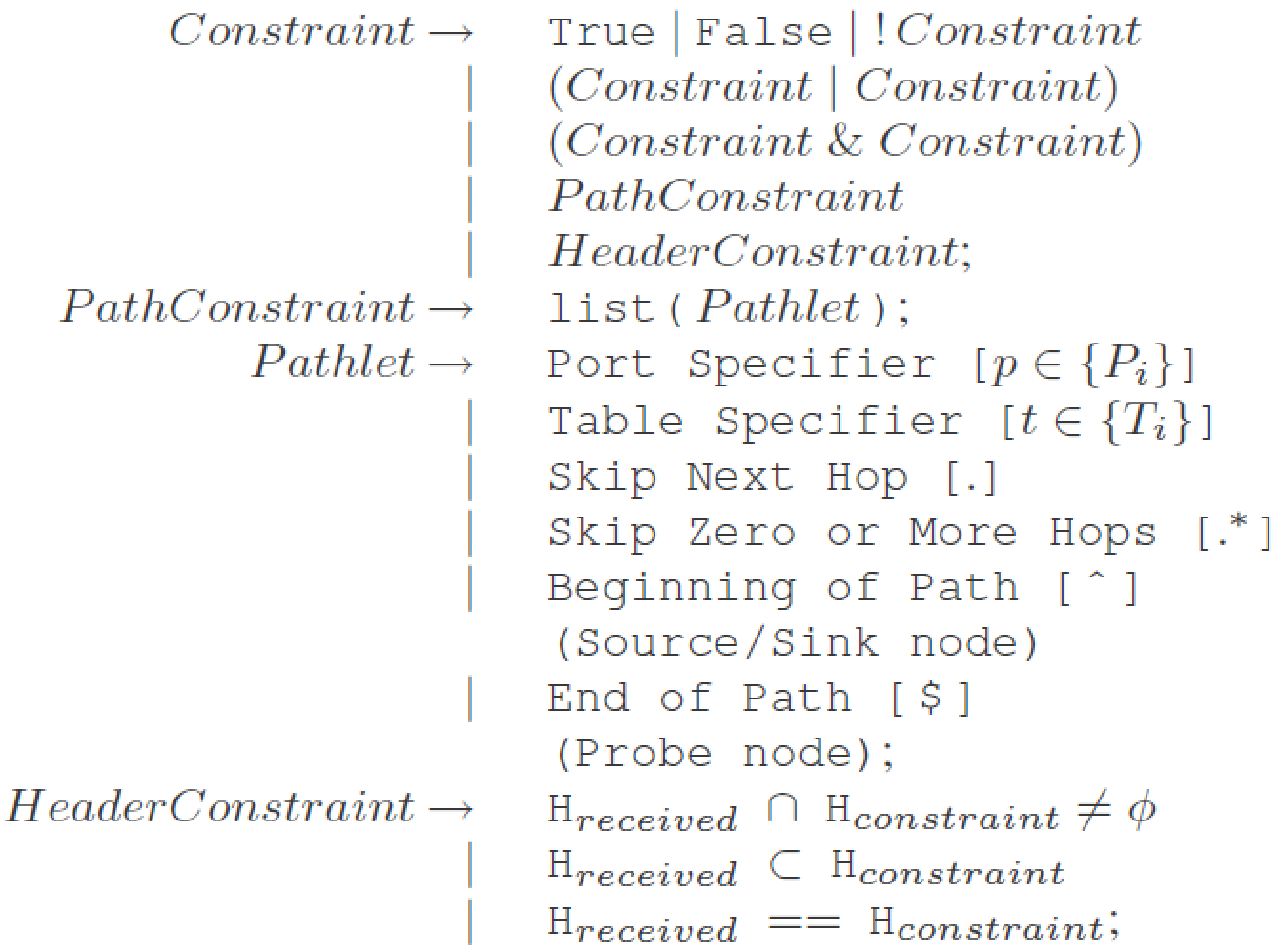
NetPlumber — Checking Policies

- Each probe node is configured with:
  - a *filter* flowexp,

    which constrains the set of flows that should be examined by the probe node, and
  - a *test* flowexp,

    which is the constraint that is checked on the matching flows.
- $\forall\{f|f\sim filter\}: f\sim test$
  - All flows which satisfy the *filter* exp, satisfy the *test* exp as well.
- $\exists\{f|f\sim filter\}: f\sim test$
  - There exist a flow that satisfies both the filter and test exps.

NetPlumber — Checking Policies

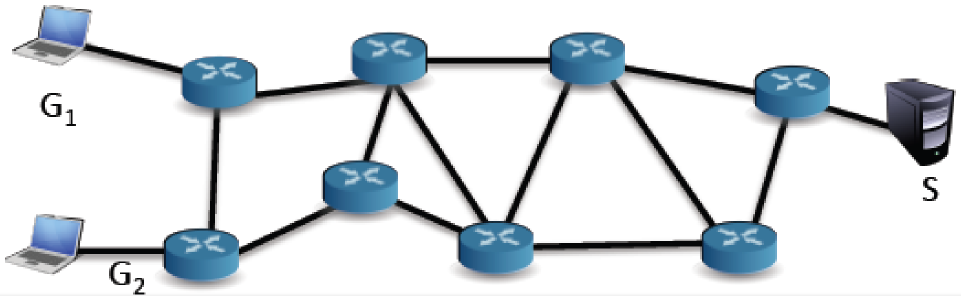


$$
\begin{array}{rcl}
Constraint & \rightarrow & \texttt{True} \mid \texttt{False} \mid\, !Constraint \\
 & \mid & (Constraint \mid Constraint) \\
 & \mid & (Constraint \;\&\; Constraint) \\
 & \mid & PathConstraint \\
 & \mid & HeaderConstraint; \\
PathConstraint & \rightarrow & \texttt{list(}\,Pathlet\,\texttt{);} \\
Pathlet & \rightarrow & \texttt{Port Specifier [}p \in \{P_i\}\texttt{]} \\
 & \mid & \texttt{Table Specifier [}t \in \{T_i\}\texttt{]} \\
 & \mid & \texttt{Skip Next Hop [.]} \\
 & \mid & \texttt{Skip Zero or More Hops [.}^{*}\texttt{]} \\
 & \mid & \texttt{Beginning of Path [}\,\hat{}\,\texttt{]} \\
 & & \texttt{(Source/Sink node)} \\
 & \mid & \texttt{End of Path [\$]} \\
 & & \texttt{(Probe node);} \\
HeaderConstraint & \rightarrow & \mathtt{H}_{received} \cap \mathtt{H}_{constraint} \neq \phi \\
 & \mid & \mathtt{H}_{received} \subset \mathtt{H}_{constraint} \\
 & \mid & \mathtt{H}_{received} == \mathtt{H}_{constraint};
\end{array}
$$

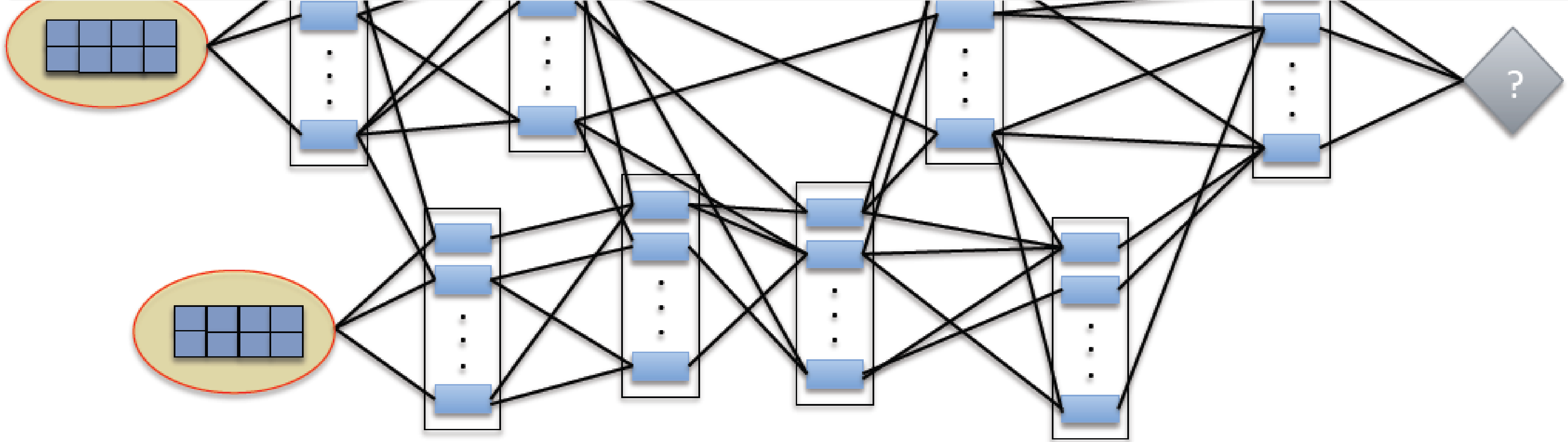
Flowexp language grammar in a standard BNF syntax

NetPlumber — Checking Policies

Policy: Guests can not access server S.

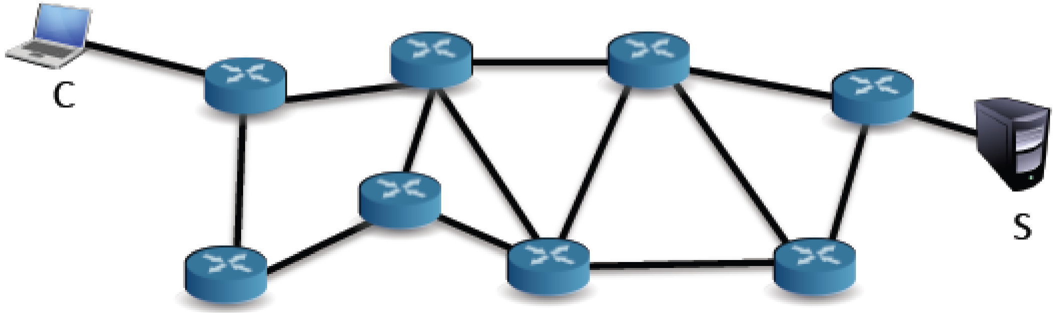


$$\forall f : f.path \sim ![\,\hat{}\,(p = G_1 \mid p = G_2).^{*}]$$

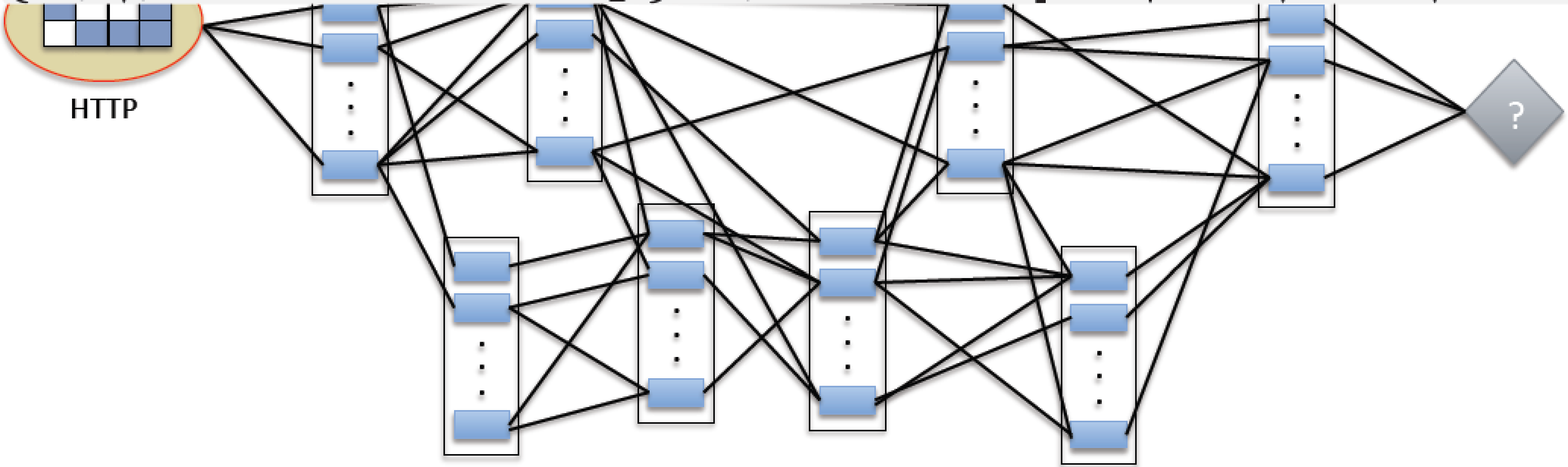


NetPlumber — Checking Policies

Policy: http traffic from client C to server S doesn’t go through more than 4 hops.

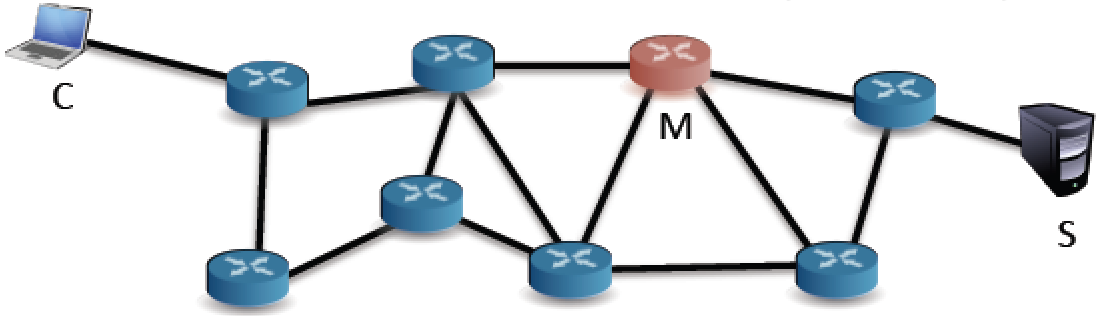


$$\{\forall f \,|\, f.header \sim \mathtt{http}\} : f.path \sim [\hat{}.\$|\hat{}..\$|\hat{}...\$|\hat{}....\$]$$

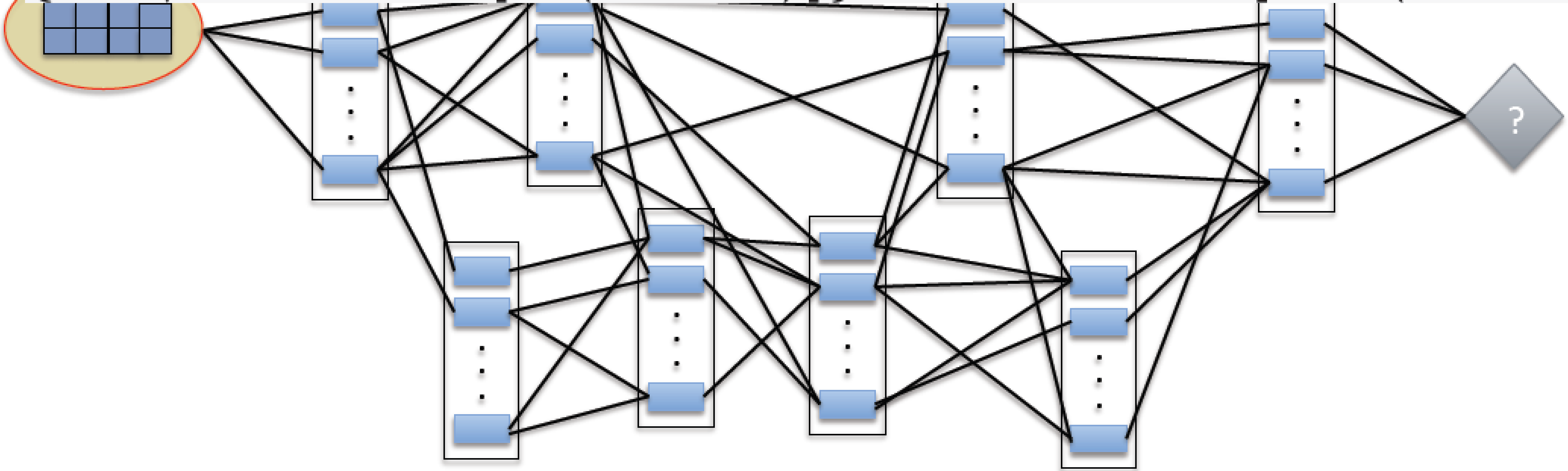


NetPlumber — Checking Policies

Policy: traffic from client C to server S should go through middle box M.



$$\{\forall f \mid f.path \sim [\,\hat{}\,(p = C)]\} \;:\; f.path \sim [\,\hat{}\,.{*}(t = M)]$$



Why the Dependency Graph Helps

- Incremental update
  - Only have to trace through dependency sub-graph affected by an update.
- Flexible policy expression
  - Probe and source nodes are flexible to place and configure.
- Parallelization
  - Can partition dependency graph into clusters to minimize inter-cluster dependences.

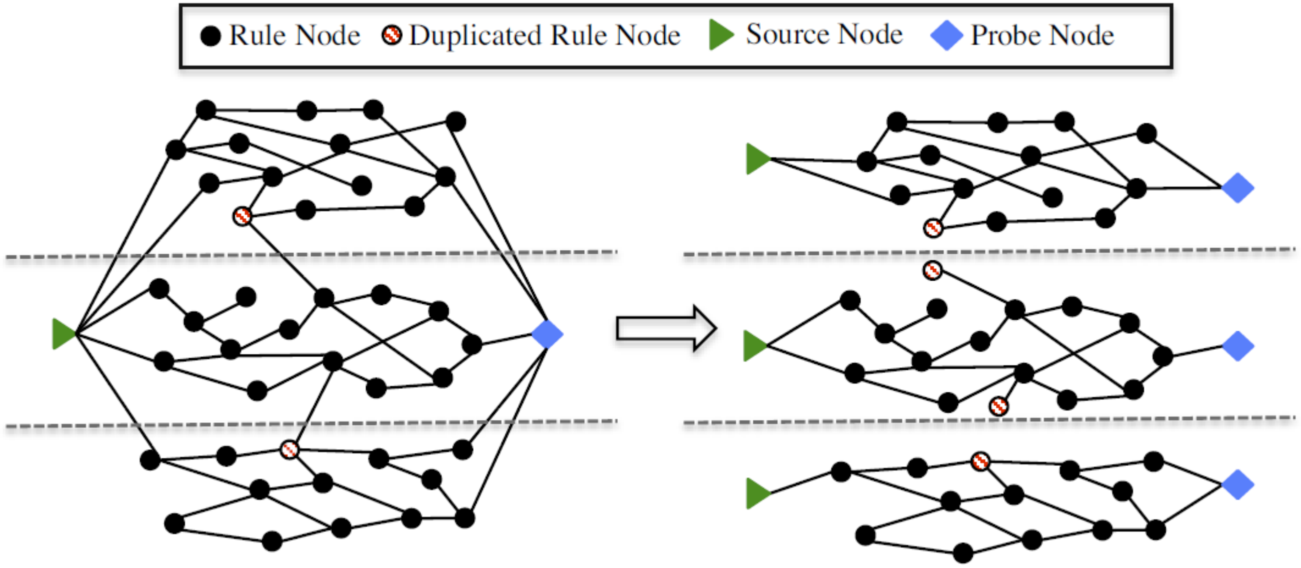
Distributed NetPlumber

- A key observation: there are clusters of highly dependent rules with very few dependencies between rules in different clusters.

- This is caused by forwarding equivalence classes (FECs).

Distributed NetPlumber

- Distributed implementation:
  - Each instance of NetPlumber is responsible for checking a subset of rules that belong to one cluster (i.e. a FEC).
  - Rules that belong to more than one cluster will be replicated on all the instances they interact with.

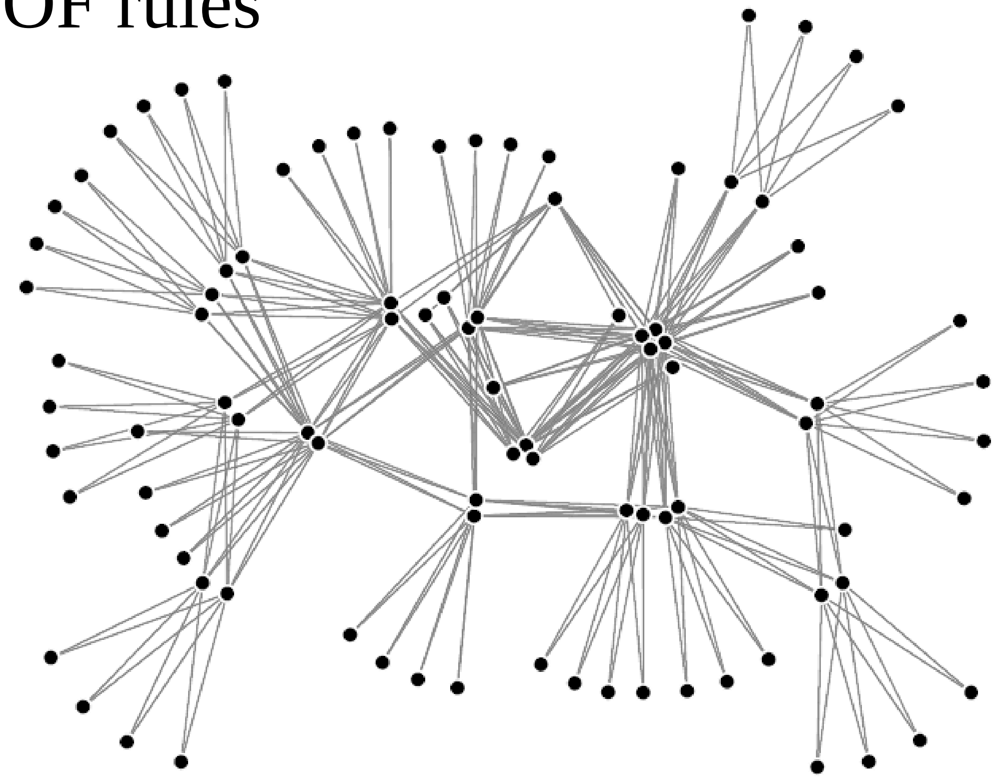


Outline

- Background
- NetPlumber: Real time policy checking tool
  - How it works?
  - How to check policy?
  - How to parallelize?
- Evaluation
- Conclusion

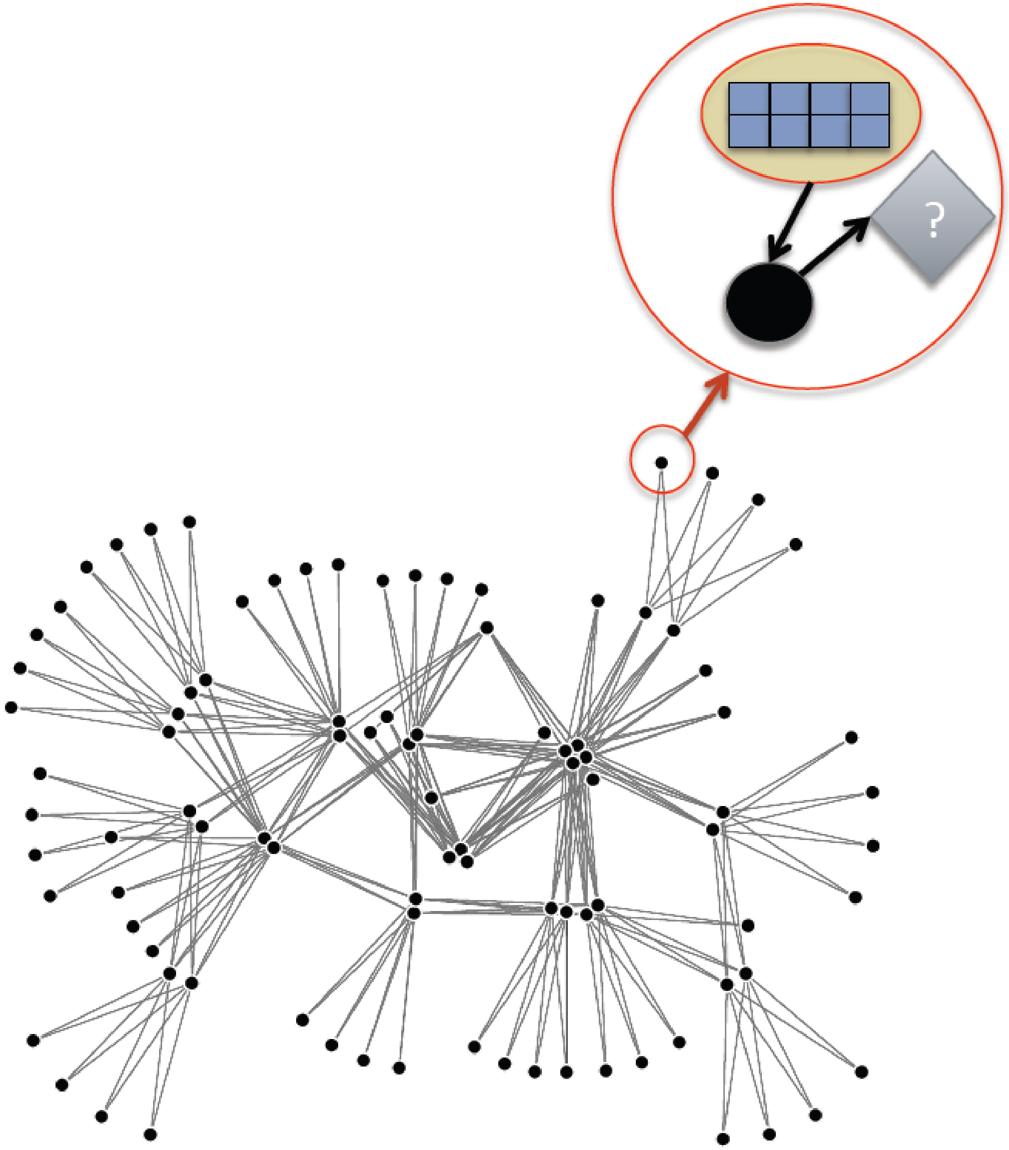
Experiment on Google WAN

- Google Inter-datacenter WAN
  - Largest deployed SDN, running OpenFlow
  - 143,000+ OF rules



Experiment on Google WAN

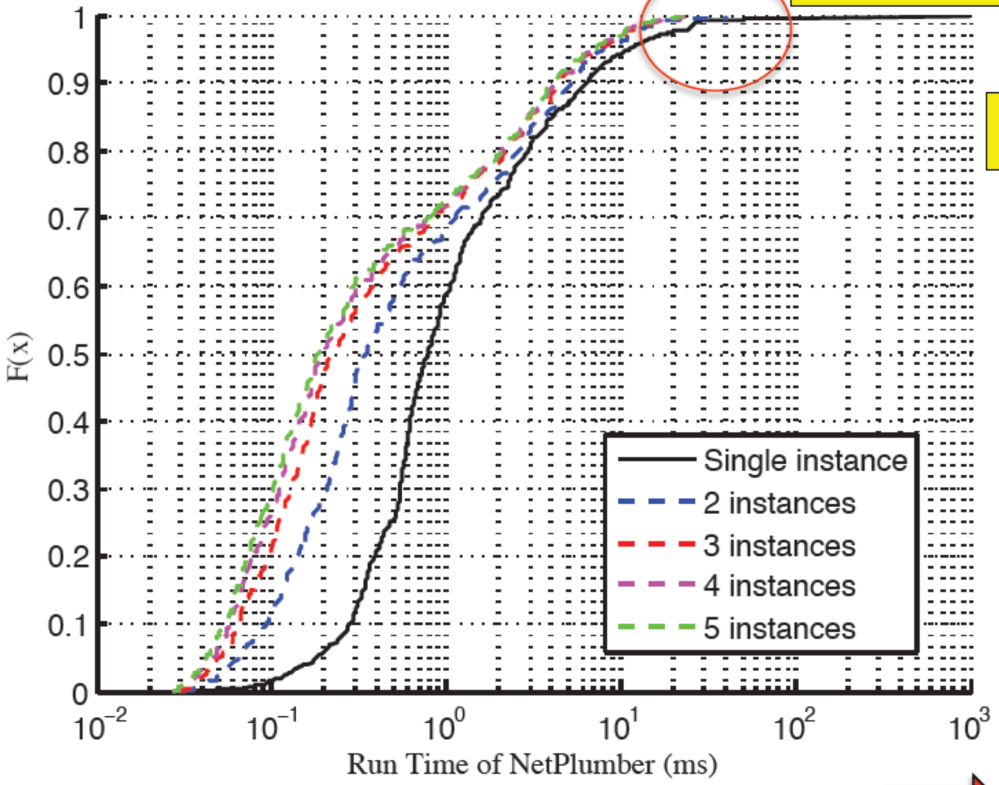
- Policy check: all 52 edge switches can talk to each other.
- 52^2 pairwise reachability check.
- Used two snapshots taken 6 weeks apart.
- Used the first snapshot to create initial NetPlumber state and used the diff as a sequential update.



Experiment on Google WAN

- Incremental updates

CDF of the run time per update

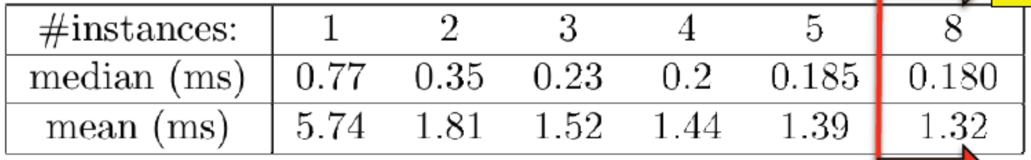


Default/Aggregate Rules.

Run time with Hassel > 100s

Not much more benefit!

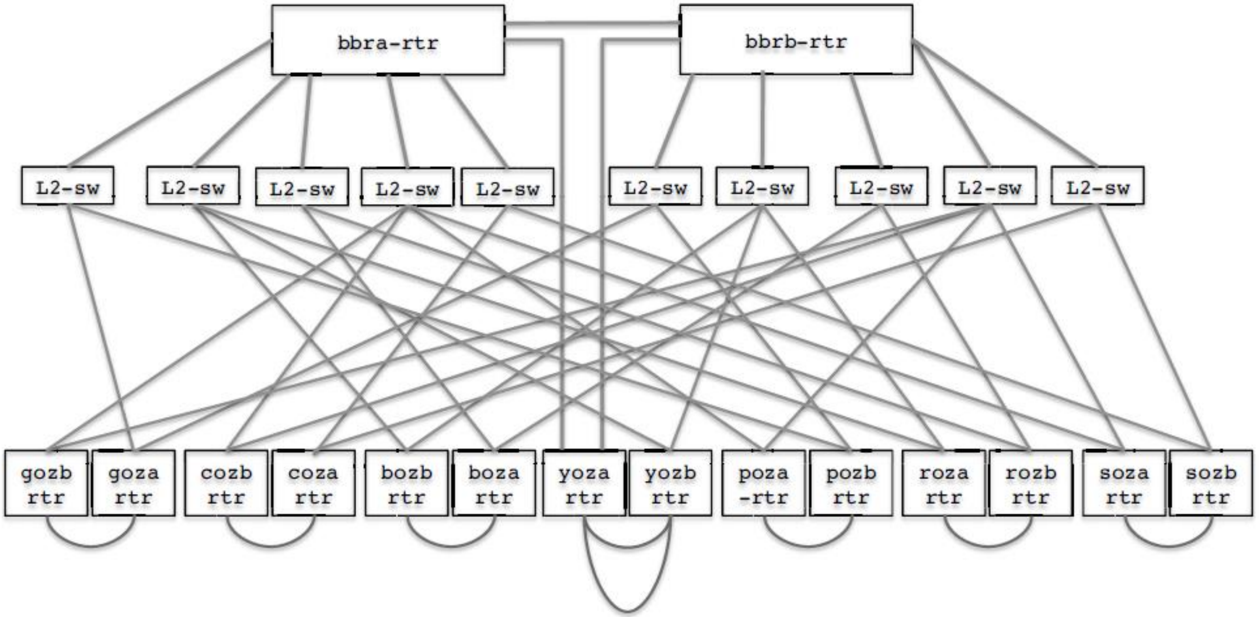
Run time of distributed NetPlumber



#instances:	1	2	3	4	5	8
median (ms)	0.77	0.35	0.23	0.2	0.185	0.180
mean (ms)	5.74	1.81	1.52	1.44	1.39	1.32

Experiment

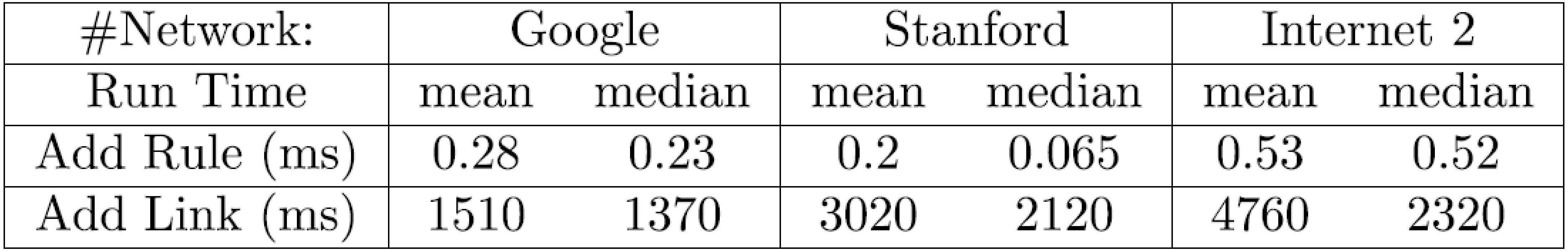
- Stanford network
  - 757,000+ forwarding entries.
  - 100+ VLANs.
  - 1500+ ACL rules.



- Internet 2
  - A nationwide backbone network.
  - ~100,000 IPv4 forwarding rules.

Benchmarking Experiment

- For a single pairwise reachability check.



#Network:	Google		Stanford		Internet 2	
Run Time	mean	median	mean	median	mean	median
Add Rule (ms)	0.28	0.23	0.2	0.065	0.53	0.52
Add Link (ms)	1510	1370	3020	2120	4760	2320

Conclusions

- Designed a protocol-independent system for real time network policy checking.
- Key component: dependency graph of forwarding rule, capturing all flow paths.
  - Incremental update.
  - Flexible policy expressions.
  - Parallelization by clustering.

Thank You!