lecture 21:
NetPlumber
5590: software defined networking

anduo wang, Temple University

TTLMAN 401B, R 17:30-20:00

HSA

HSA

pro

- protocol agnostic, not restricted to existing protocols
- statically verify reachability properties

HSA

pro

- protocol agnostic, not restricted to existing protocols
- statically verify reachability properties

cons

- verify a snapshot of the network state
  - assumes external mechanism for collecting the "state" from the entire network
  - checking
- *but network state is constantly changing, and compliance checking needs to be realtime*

HSA

pro

- protocol agnostic, not restricted to existing protocols
- statically verify reachability properties

cons

- verify a snapshot of the network state
  - assumes external mechanism for collecting the "state" from the entire network
  - checking
- *but network state is constantly changing, and compliance checking needs to be realtime*

remedy

- SDN + NetPlumber

SDN presents an opportunity

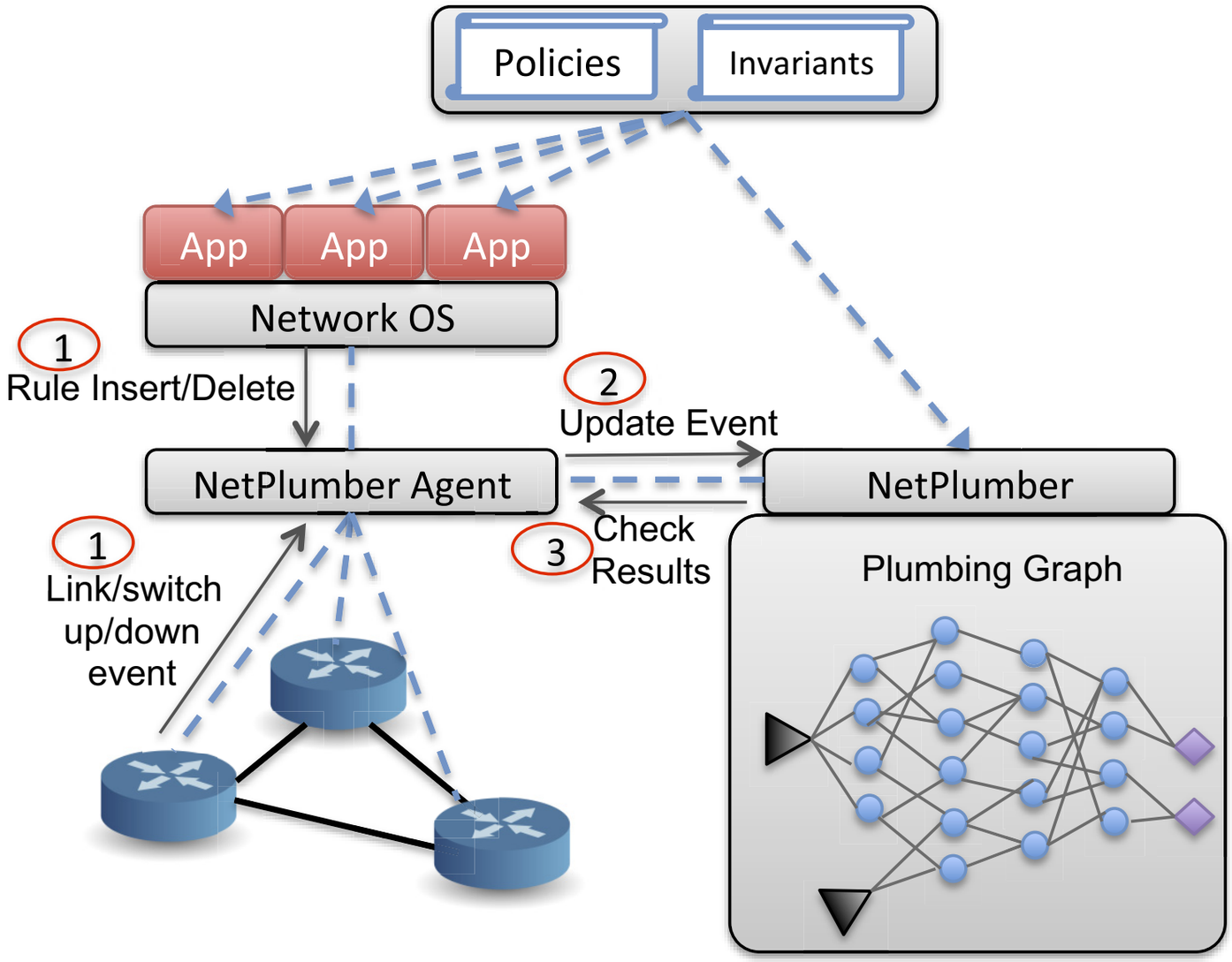
SDN controller

- observes and controls the network state as the single creator

presents an opportunity for fast automatic verification

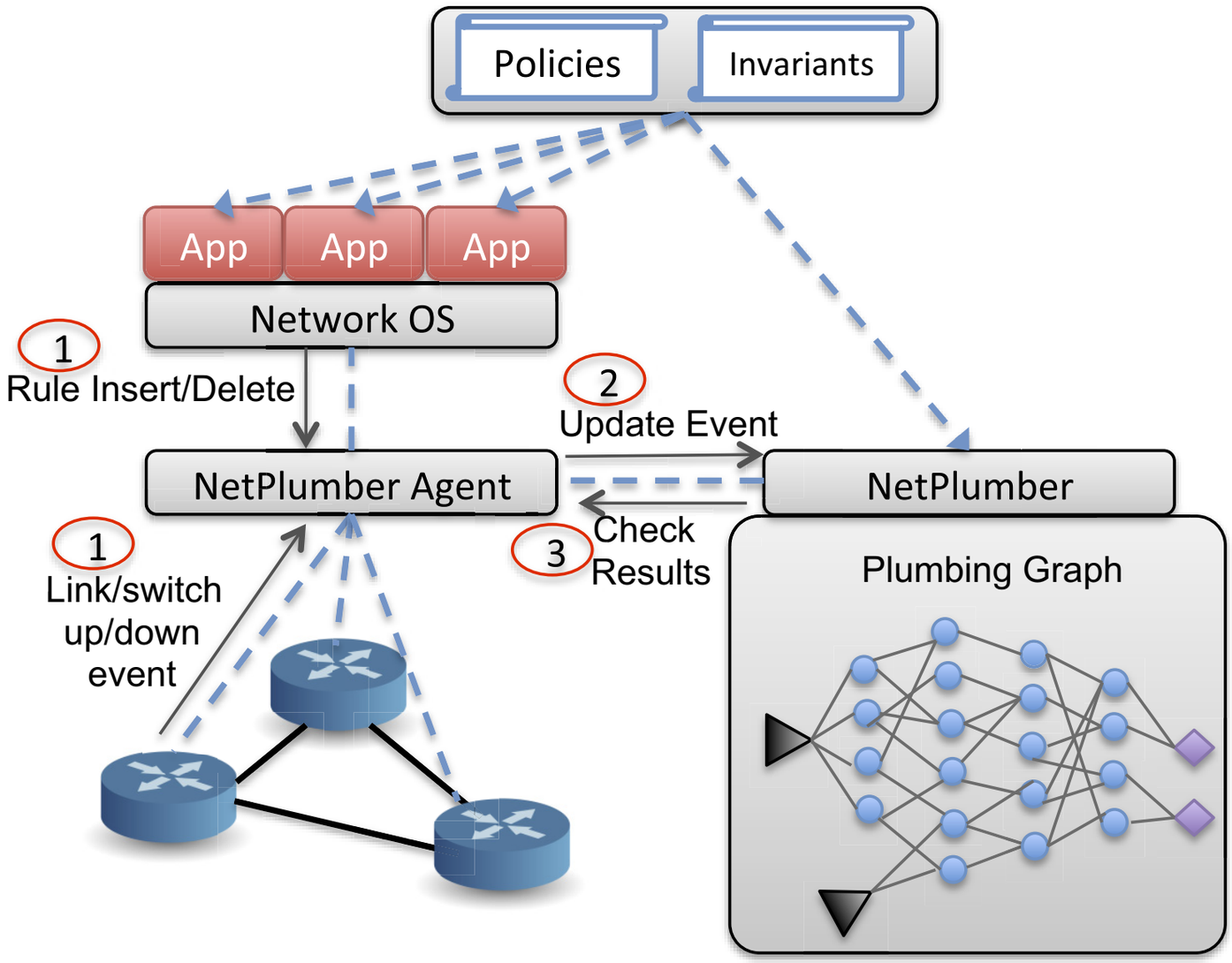
- analyze the network state — forwarding state
- either as the state is written to switches, or after it is written

NetPlumber



real-time checks at update time

NetPlumber



- incremental HSA checks, leveraging Plumbing graph
- policy query language, avoids writing ad hoc checking code

incremental checking

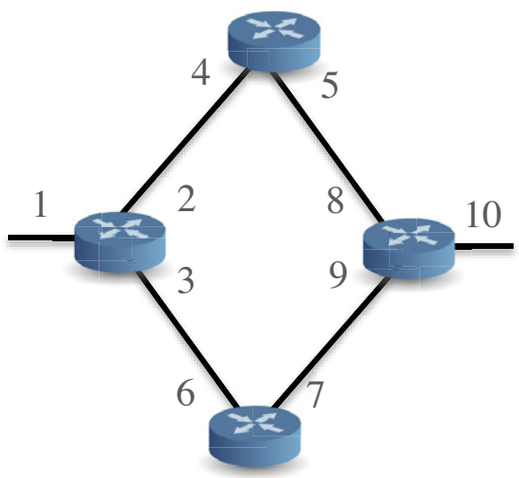
incrementally updates the transfer functions affected by a network change

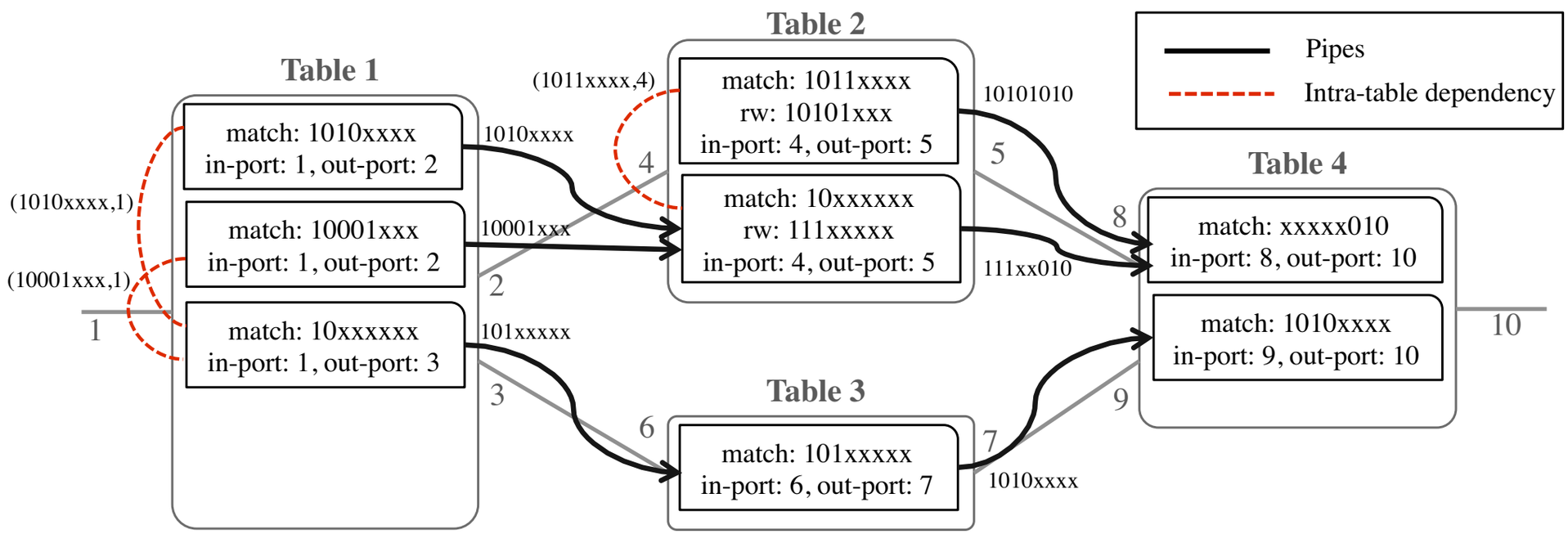
- plumbing graph — the full forwarding state
  - captures all possible paths of flows in the network

static checking

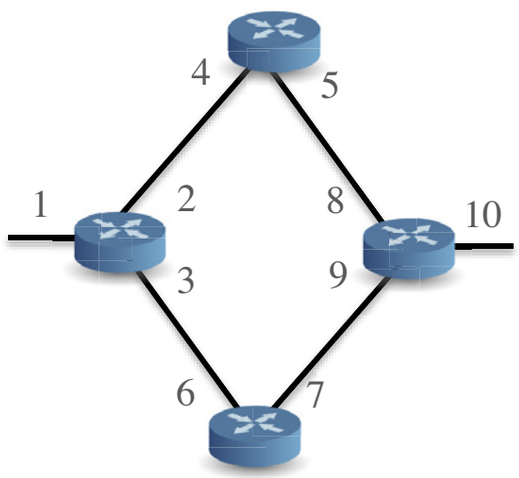
- HSA analysis, but with a (wrapper) policy language

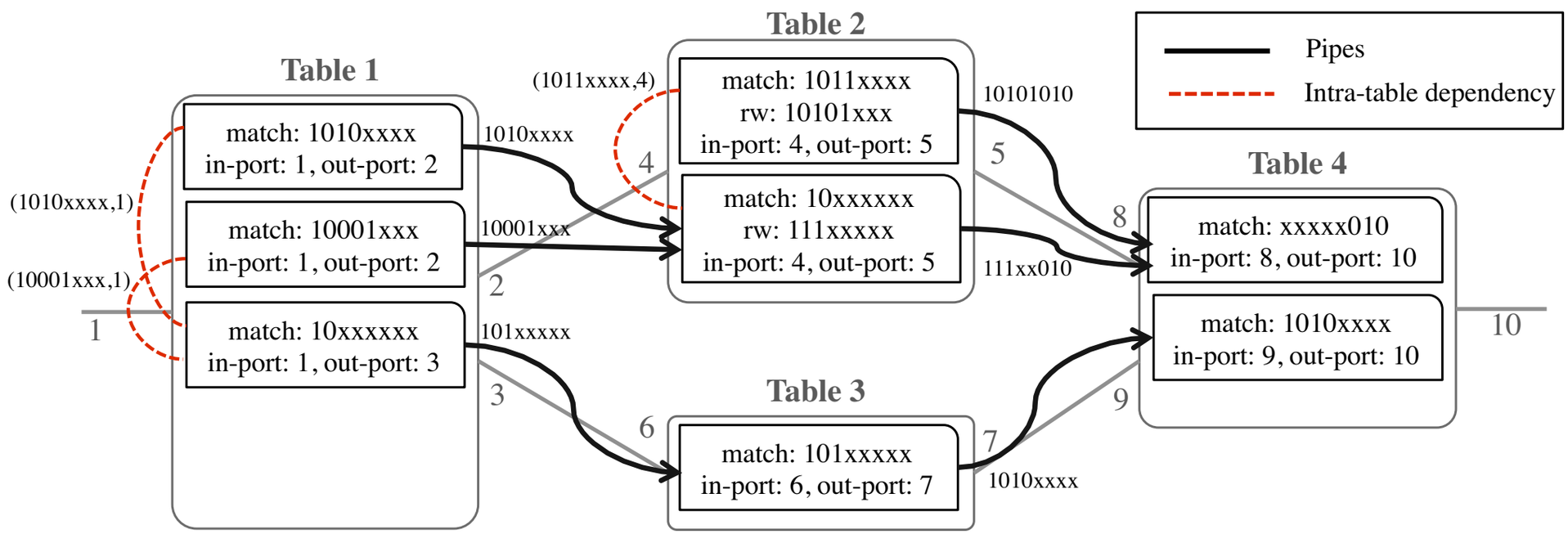
plumbing graph overview





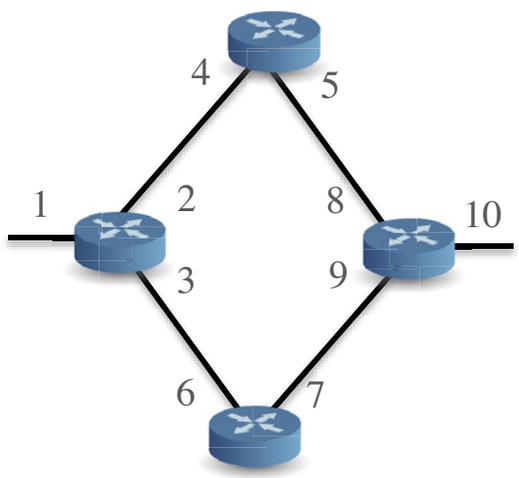
plumbing graph overview

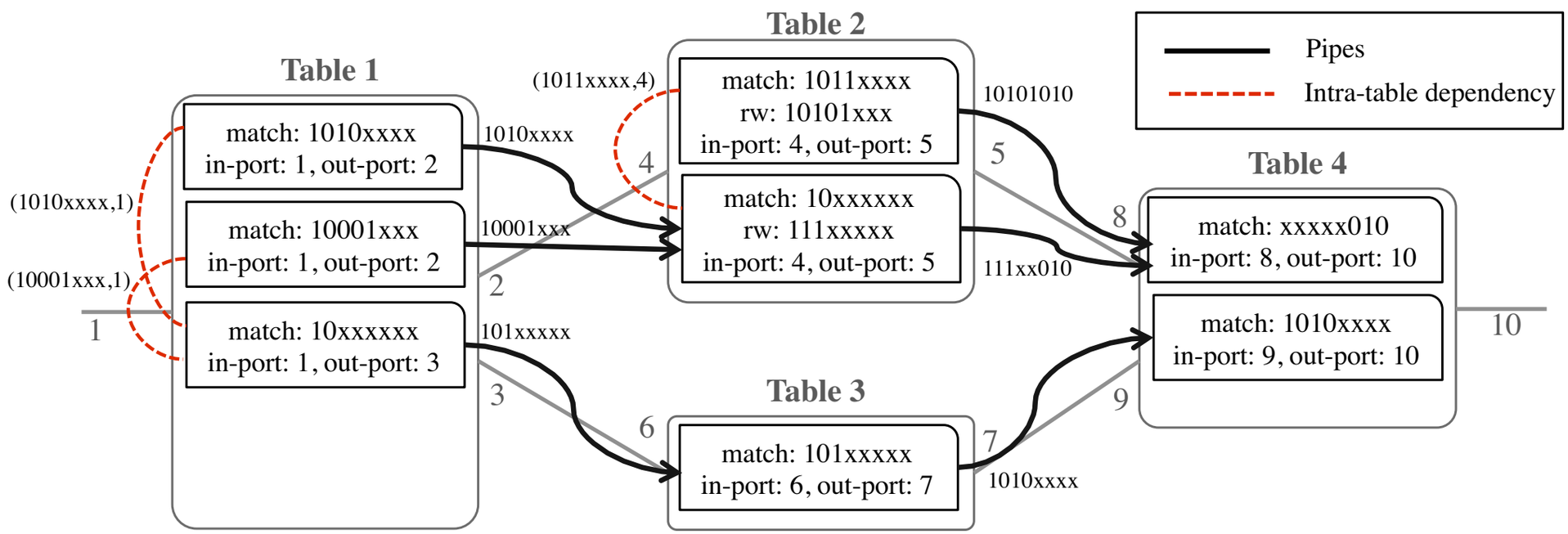




node: OF-like rule <match, action>

plumbing graph overview



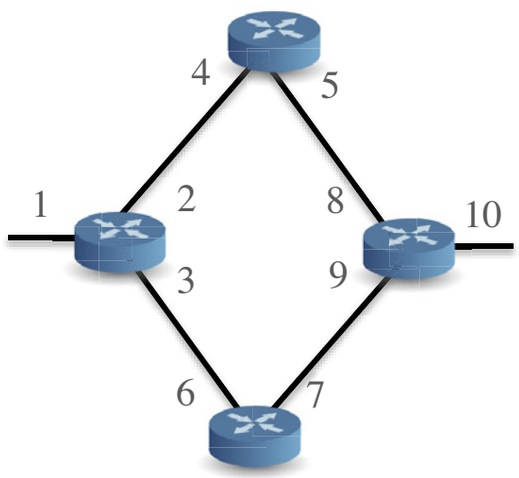


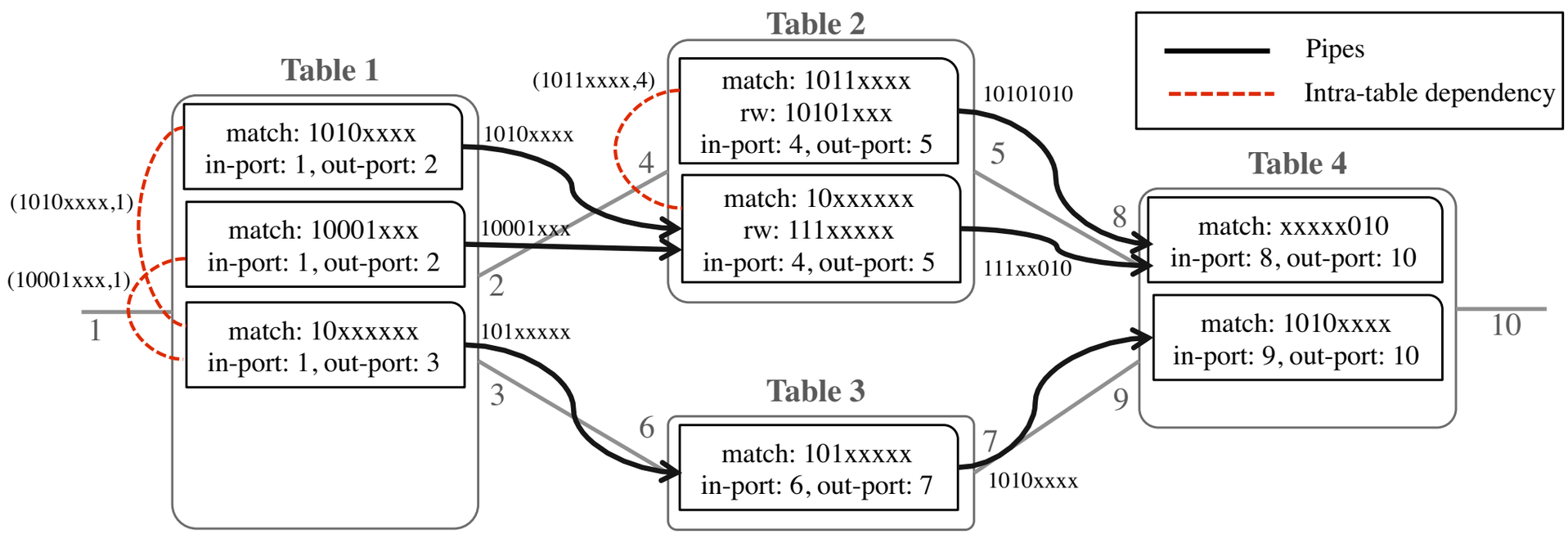
node: OF-like rule <match, action>

directed edges: next-hop dependency

- also called pipe, a pipe from a to b has
  - pipe filter is the intersection of a range and b

plumbing graph overview





node: OF-like rule <match, action>

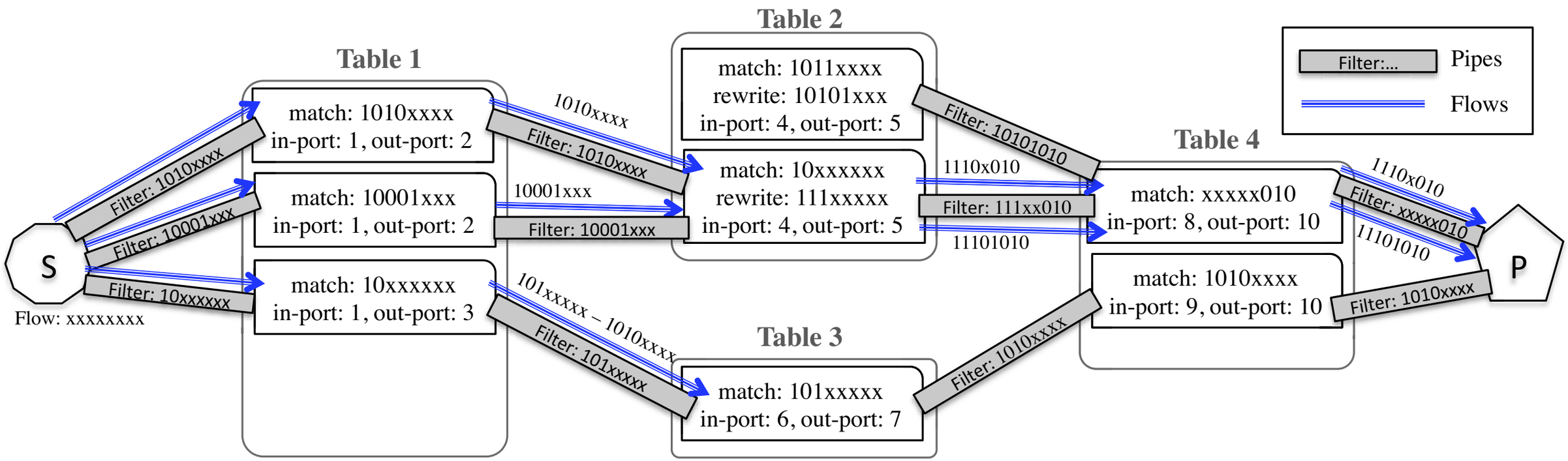
directed edges: next-hop dependency

- also called pipe, a pipe from a to b has
  - pipe filter is the intersection of a range and b

dashed edge: intra-table dependency

- subtracting domain of higher-priority rule in the same table

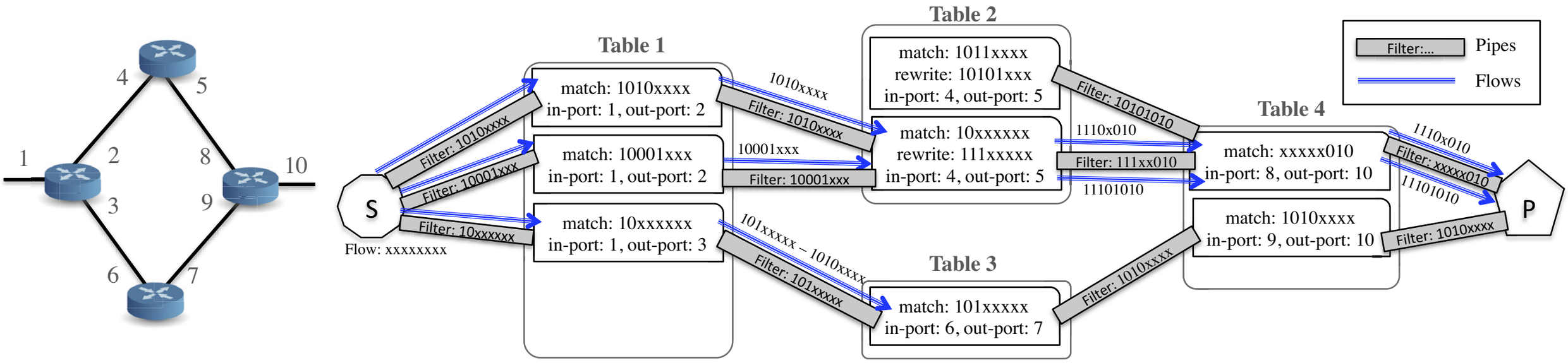
compute reachability



policy checking = reachability computation

- flow generator
  - *source node*: insert flow from the source port and propagates it towards the destination
  - *sink node: generates "sink flow" that traverses backwards*
    - at each hope, processed by the inverse of the rule
- checking policy — probe node

compute reachability



- check policy "port 1 and 10 can only talk using packets matching xxxxx010"
- place a source node (S) at port 1
- place a probe node (P) at port 10, configure P to check whether all flows from S match xxxxx010

maintaining plumbing graph

incrementally update the portion of the graph which is affected by a network change

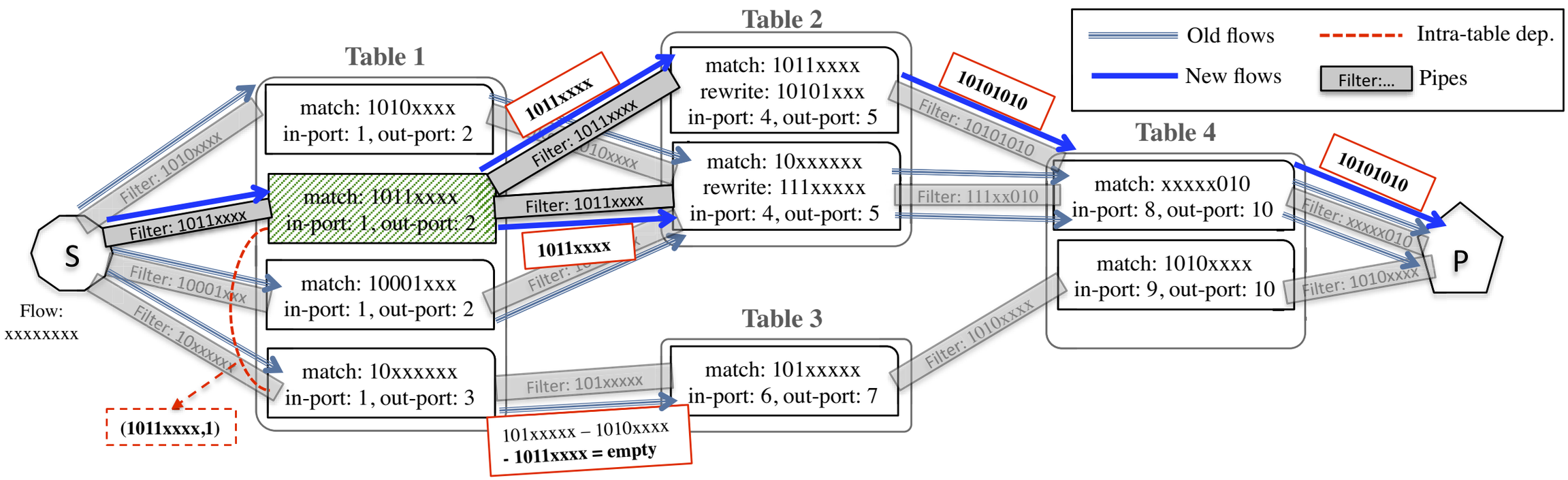
- add new rules
- delete rules
- link up
- link down
- add new tables
- delete tables

maintaining plumbing graph

incrementally update the portion of the graph which is affected by a network change

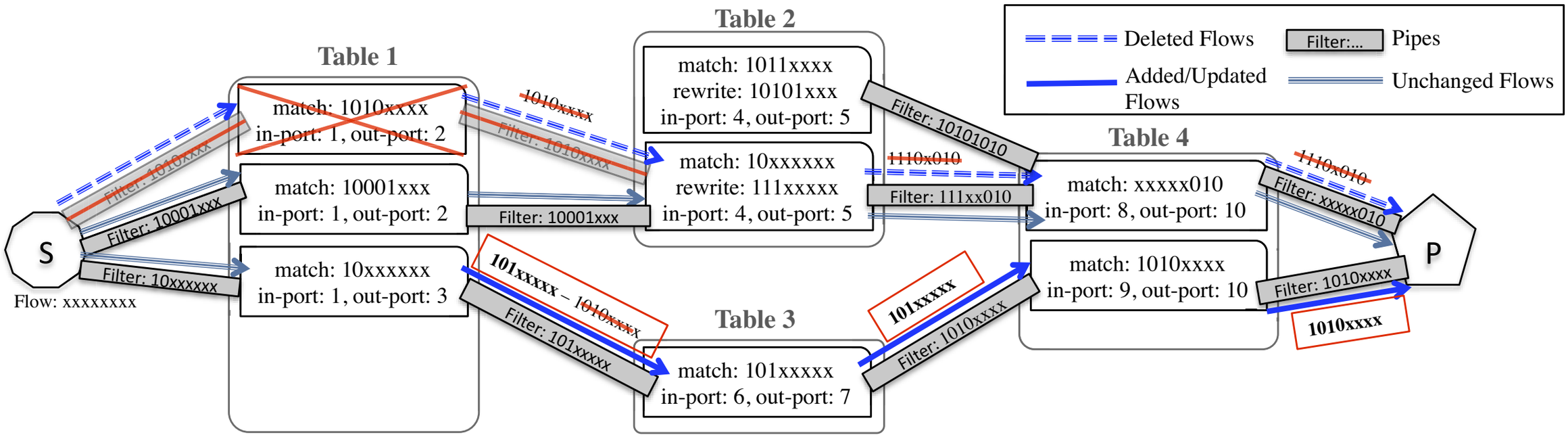
- add new rules
- delete rules
- link up
- link down
- add new tables
- delete tables

maintaining plumbing graph — add rules



- create pipes
  - from new rule to all next-hops
  - from previous hop rules to the new one
- update routing flows
  - adding flows to the newly created pipes
  - subtracting flows passing through lower priority rules

maintaining plumbing graph — delete rules



- remove pipes
- update routing flows
  - delete flows which pass through the rule to be removed
  - adding back flows passing through lower priority rules

checking policy

probe node

- monitor flows received on a set of ports

configure probe node with *flowexp*

- filter exp: constrain flows examined
- test exp: test constraints on the matched flow

flowexp

$$\forall\{f \mid f \sim filter\} : f \sim test$$

$$\exists\{f \mid f \sim filter\} : f \sim test$$

policy language

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

Flowexp

- regular expression
- check constraints on the history of flows

policy language

$Constraint \rightarrow$ `True` | `False` | `!`$Constraint$
| (`(`$Constraint$ | $Constraint$`)`
| (`(`$Constraint$ `&` $Constraint$`)`
| $PathConstraint$
| $HeaderConstraint$;

$PathConstraint \rightarrow$ `list(` $Pathlet$ `);`

$Pathlet \rightarrow$ `Port Specifier [`$p \in \{P_i\}$`]`
| `Table Specifier [`$t \in \{T_i\}$`]`
| `Skip Next Hop [.]`
| `Skip Zero or More Hops [.*]`
| `Beginning of Path [^]`
`(Source/Sink node)`
| `End of Path [$]`
`(Probe node);`

$HeaderConstraint \rightarrow$ $H_{received} \cap H_{constraint} \neq \phi$
| $H_{received} \subset H_{constraint}$
| $H_{received} == H_{constraint}$;

- path constraints, e.g.:

$S \rightarrow A \rightarrow B \rightarrow C \rightarrow P$ → $\hat{}(p = A)$ $(p = A).(p = C)$

- header constraints
  - received header intersects / is a subset / exactly equals a specified header

loops, black holes

each node in plumbing graph

- by default, checks received flows
  - for loops, black holes

reachability properties

idea: attach one or more source (sink) nodes and one or more probe nodes in the plumbing graph

- basic reachability
  - a server port S is not reachable from guest ports $\{G_1,\ldots,G_k\}$
    - place source nodes at each guest port
    - probe node at S, and configure it with $\forall f : f.path \sim ![\hat{}(p \in \{G_1, ...G_k\})]$
  - S reachable from $\{G_1,\ldots,G_k\}$ $\exists f : f.path \sim [\hat{}(p \in \{G_1, ...G_k\})]$
  - dual solution with
    - place sink node at S, configure probe at guests

$$\forall f : f.path \sim [\hat{}(p \in \{S\})]$$

reachability properties

idea: attach one or more source (sink) nodes and one or more probe nodes in the plumbing graph

- waypoint: traffic from C to S must pass through M
- solution
  - place source at C, probe at S
  - configure probe $\forall\{f \mid f.path \sim [\hat{}(p \in \{C\})]\} \;:\; f.path \sim [\hat{}.^*(t = M)]$

policy translator

```
guest(sam).
guest(michael).
server(webserver).
waypoint(HostSrc, HostDst, firewall):-
    guest(HostSrc),
    server(HostDst).
```

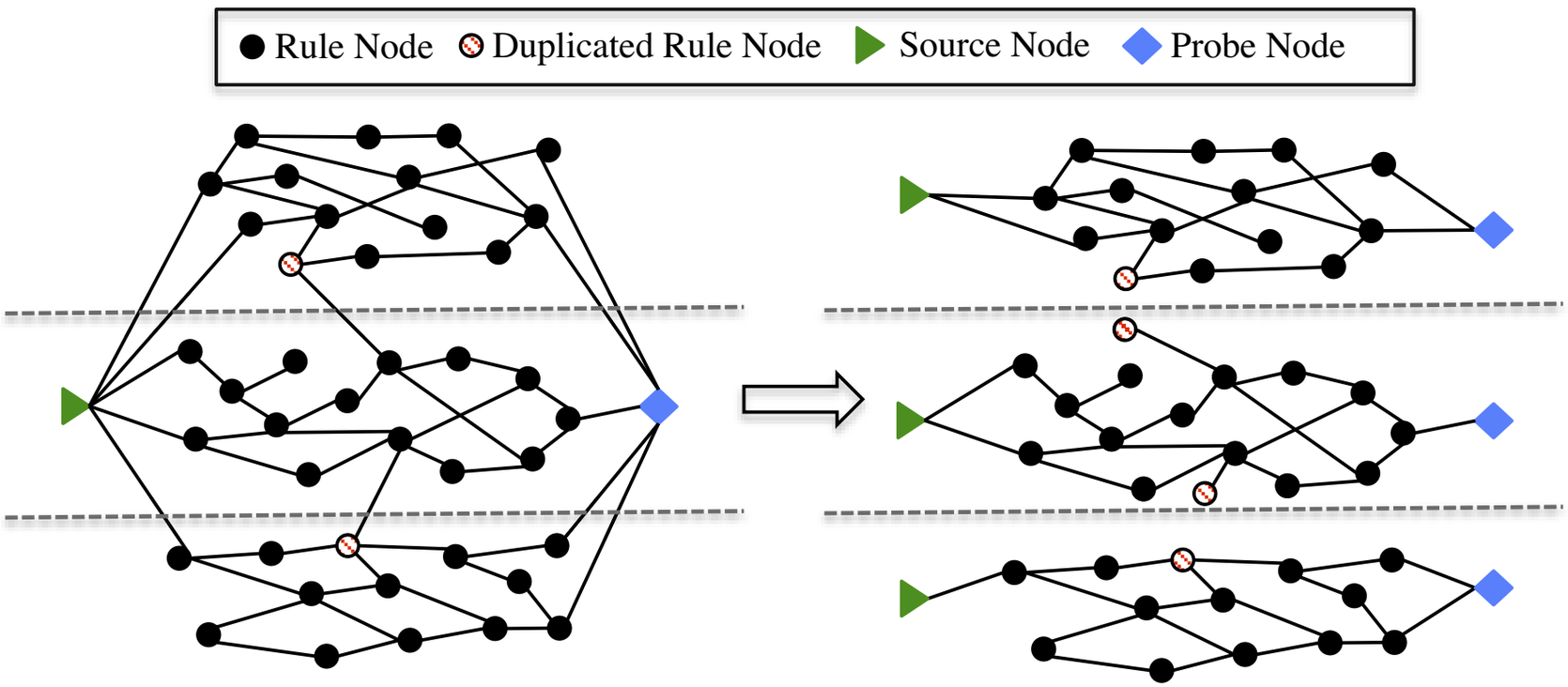
Prolog (FML)-like frontend language

- declare binding (group)
- specify which groups can communicate

NetPlumber translator generates

- placement of source node
- placement of probe node, configure the probe node with filter and test expression

distributed NetPlumber



run parallel instances of NetPlumber on each cluster

- cluster: highly dependent rules
  - (forwarding equivalence classes), e.g., 10.1.0.0/16 subnet traffic be a FEC
- very few dependency across clusters
  - very few rules outside the range of 10.1.0.0/16