Design and Implementation of a Strong Representation System for Network Policies

Fangping Lan, Sanchari Biswas, Bin Gui, Jie Wu and Anduo Wang
Temple University

ICCCN 2022 July 25 - July 28, 2022

Motivation

- Managing networking policies remains hard:
  - One has to fully understand the policy
  - Understanding SDN is difficult for someone not involved in the coding process

Motivation

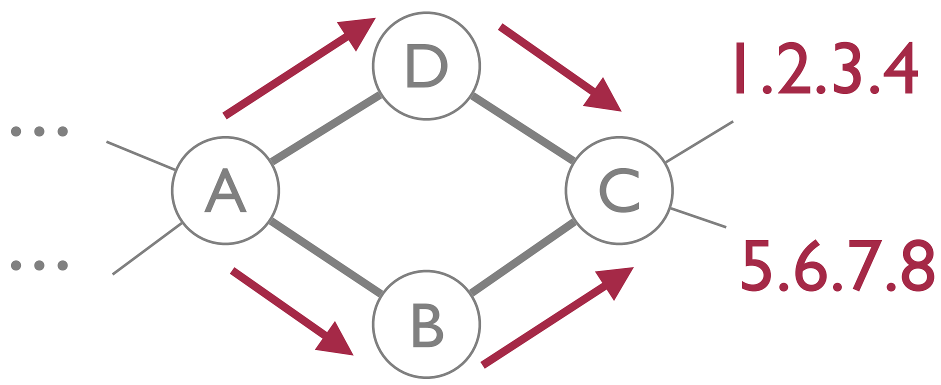
- Managing networking policies remains hard:
  - One has to fully understand the policy
  - Understanding SDN is difficult for someone not involved in the coding process
- Managing relational database is easier:
  - Self-explaining and has a common understanding
  - Little expertise required for non-programmers

Motivation

- Managing networking policies remains hard:
  - One has to fully understand the policy
  - Understanding SDN is difficult for someone not involved in the coding process
- Managing relational database is easier:
  - Self-explaining and has a common understanding
  - Little expertise required for non-programmers

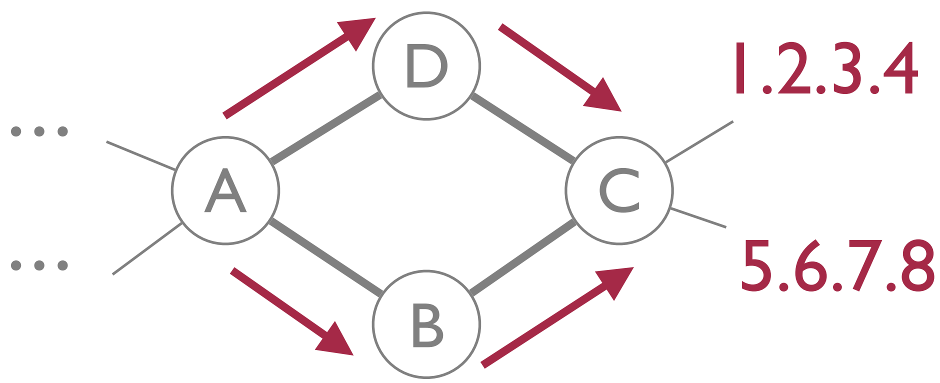
Can we provide a network policy management experience comparable to that on a database?

Network Policy Representation



Two prefixes 1.2.3.4 and 5.6.7.8 over two alternative paths [ABC] and [ADC]

Network Policy Representation

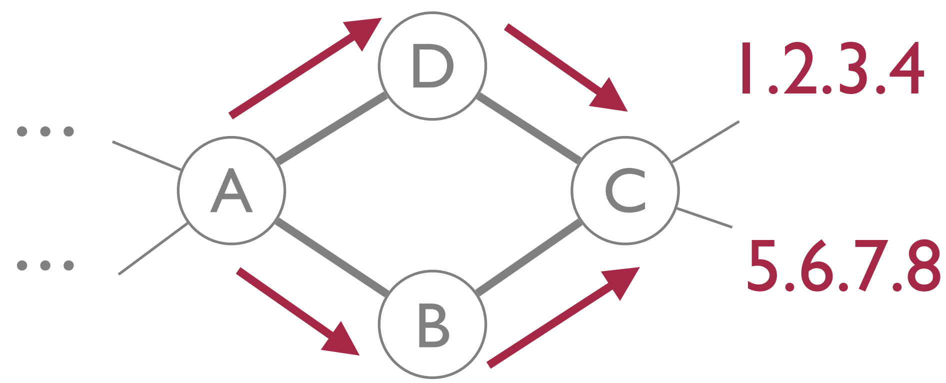


Two prefixes 1.2.3.4 and 5.6.7.8 over two alternative paths [ABC] and [ADC]

P_R	dest	path	condition
	1.2.3.4	x	x=[ABC]∨x=[ADC]
	5.6.7.8	y	y=[ABC]∨y=[ADC]

The destination of the path

Network Policy Representation

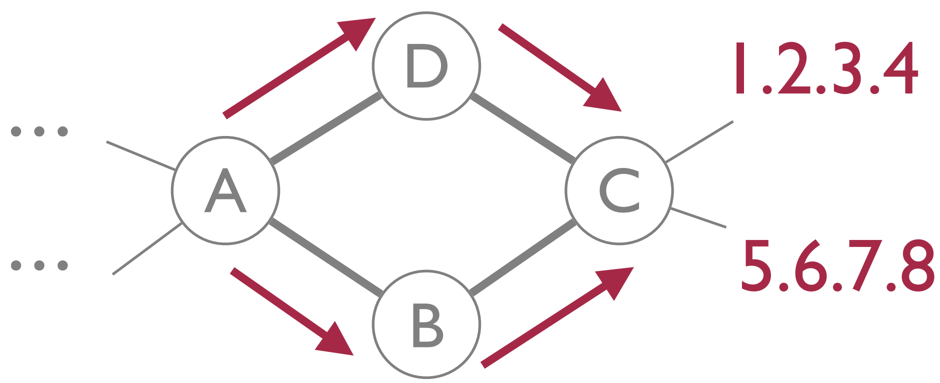


Two prefixes 1.2.3.4 and 5.6.7.8 over two alternative paths [ABC] and [ADC]

P_R	dest	path	condition
	1.2.3.4	x	x=[ABC]∨x=[ADC]
	5.6.7.8	y	y=[ABC]∨y=[ADC]

The path to its destination

Network Policy Representation

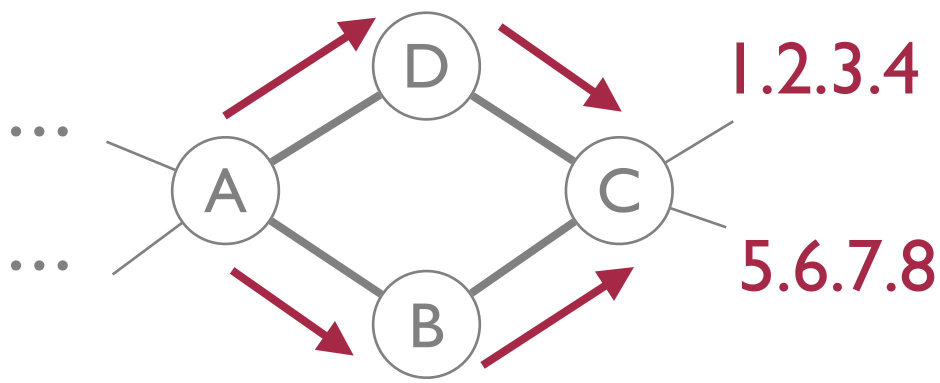


Two prefixes 1.2.3.4 and 5.6.7.8 over two alternative paths [ABC] and [ADC]

P_R	dest	path	condition
	1.2.3.4	x	x=[ABC]∨x=[ADC]
	5.6.7.8	y	y=[ABC]∨y=[ADC]

The constraints over variables

Network Policy Representation



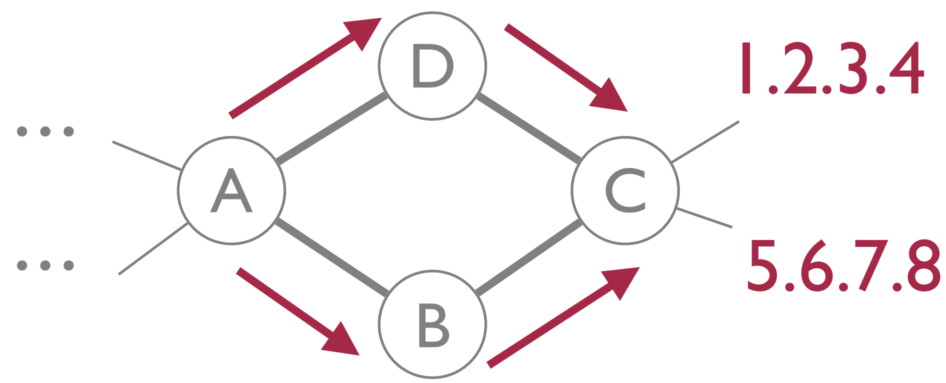
Two prefixes 1.2.3.4 and 5.6.7.8 over two alternative paths [ABC] and [ADC]

P_R	dest	path	condition
	1.2.3.4	x	x=[ABC]∨x=[ADC]
	5.6.7.8	y	y=[ABC]∨y=[ADC]

→ Represent

I	dest	path
	1.2.3.4	[ABC]
	1.2.3.4	[ADC]
	5.6.7.8	[ABC]
	5.6.7.8	[ADC]

Network Policy Representation



Two prefixes 1.2.3.4 and 5.6.7.8 over two alternative paths [ABC] and [ADC]

P_R	dest	path	condition
	1.2.3.4	x	x=[ABC]∨x=[ADC]
	5.6.7.8	y	y=[ABC]∨y=[ADC]

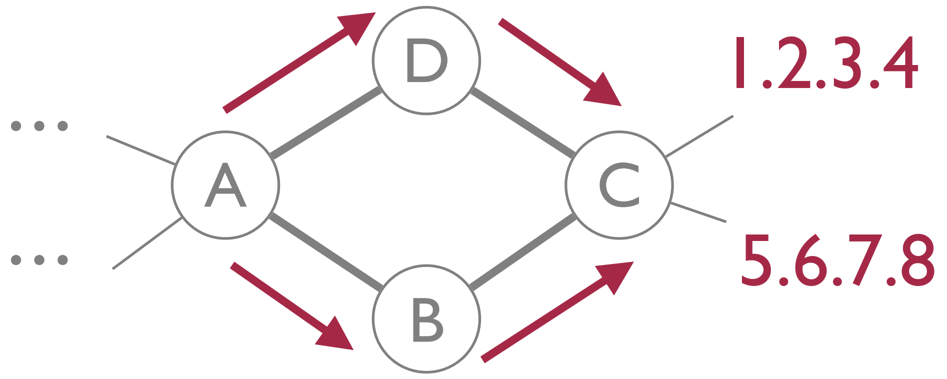
Conditional Table

Network Policy Representation

- Traffic balance policy

Balance traffic to 1.2.3.4 and 5.6.7.8

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

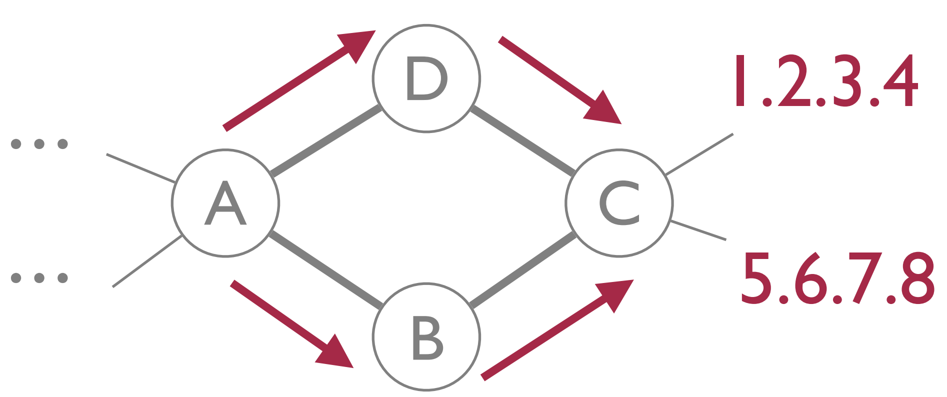


Network Policy Representation

- Traffic balance policy

Balance traffic to 1.2.3.4 and 5.6.7.8

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1



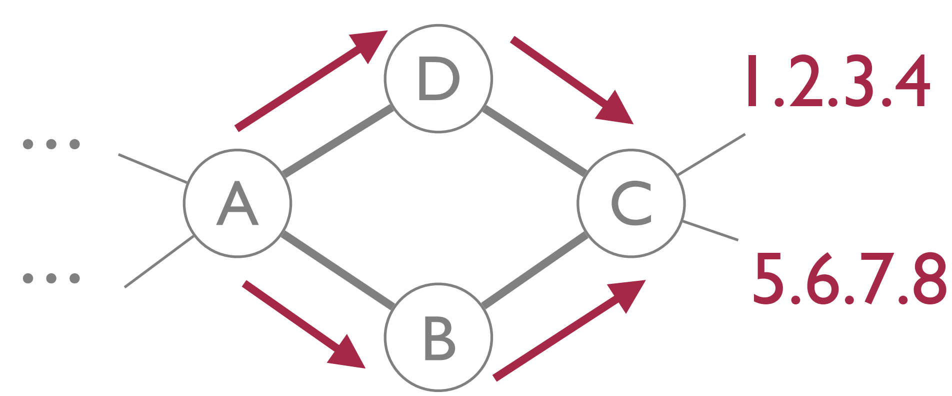
Network Policy Representation

- Traffic balance policy

Balance traffic to 1.2.3.4 and 5.6.7.8

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Cannot assign path [ABC] to 1.2.3.4 and 5.6.7.8 simultaneously.



Network Policy Representation

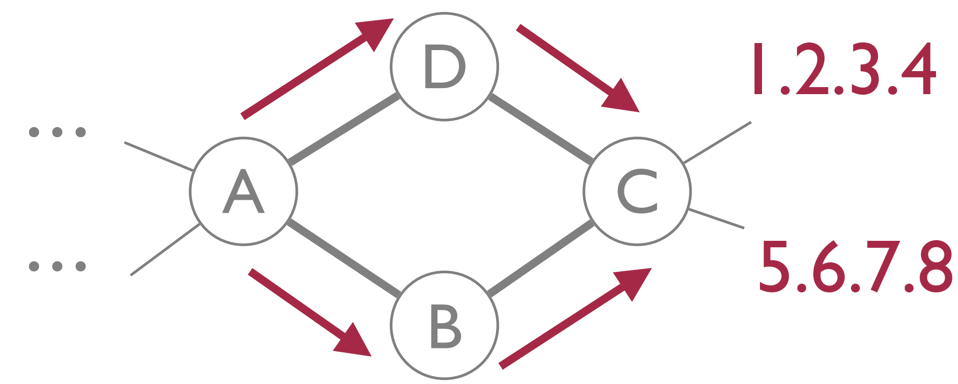
- Traffic balance policy

Balance traffic to 1.2.3.4 and 5.6.7.8

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Cannot assign path [ABC] to 1.2.3.4 and 5.6.7.8 simultaneously.

Cannot assign path [ADC] to 1.2.3.4 and 5.6.7.8 simultaneously.



Network Policy Representation

- Static route and filter policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5

Network Policy Representation

- Static route and filter policy

Assign a static route [ABC] to destination 1.2.3.4

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5

Network Policy Representation

- Static route and filter policy

Assign a static route [ABC] to destination 1.2.3.4

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5

Do not allow traffic to destination 1.2.3.5

Manipulating Network Policies

- Manipulate relational database table:
  - relational algebra : selection, projection, union, join, …
  - implement SQL query on regular table

Manipulating Network Policies

- Manipulate relational database table:
  - relational algebra : selection, projection, union, join, …
  - implement SQL query on regular table
- Our contribution:

Manipulating network policies by simply implementing SQL query on conditional table

Combine Policies using Join Operation

- Static route and filter policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5∧y≠1.2.3.4

- Traffic balance policy

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Combine Policies using Join Operation

- Static route and filter policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5∧y≠1.2.3.4

Can we generate a new policy that satisfies P_1 and P_3 simultaneously?

P_1 join P_3

- Traffic balance policy

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Combine Policies using Join Operation

- Static route and filter policy
- Traffic balance policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5∧y≠1.2.3.4

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Can we generate a new policy that satisfies P_1 and P_3 simultaneously?

P_1 join P_3

$P_1 \bowtie P_3$	dest	path	flag	
	1.2.3.4	x	u	x=[ABC] ∧ u=1
	y	z	u	y=5.6.7.8∧z=[ABC]∧u≠1
	y	z	v	y=5.6.7.8∧z=[ADC]∧v≠1

Combine Policies using Join Operation

- Static route and filter policy
- Traffic balance policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5∧y≠1.2.3.4

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Can we generate a new policy that satisfies P_1 and P_3 simultaneously?

P_1 join P_3

$P_1 \bowtie P_3$	dest	path	flag	
	1.2.3.4	x	u	x=[ABC] ∧ u=1
	y	z	u	y=5.6.7.8∧z=[ABC]∧u≠1
	y	z	v	y=5.6.7.8∧z=[ADC]∧v≠1

Valid forwarding state

I	dest	path
	1.2.3.4	[ABC]
	5.6.7.8	[ADC]

Combine Policies using Join Operation

- Static route and filter policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	$y \neq 1.2.3.5 \wedge y \neq 1.2.3.4$

Can we generate a new policy that satisfies P_1 and P_3 simultaneously?

P_1 join P_3

- Traffic balance policy

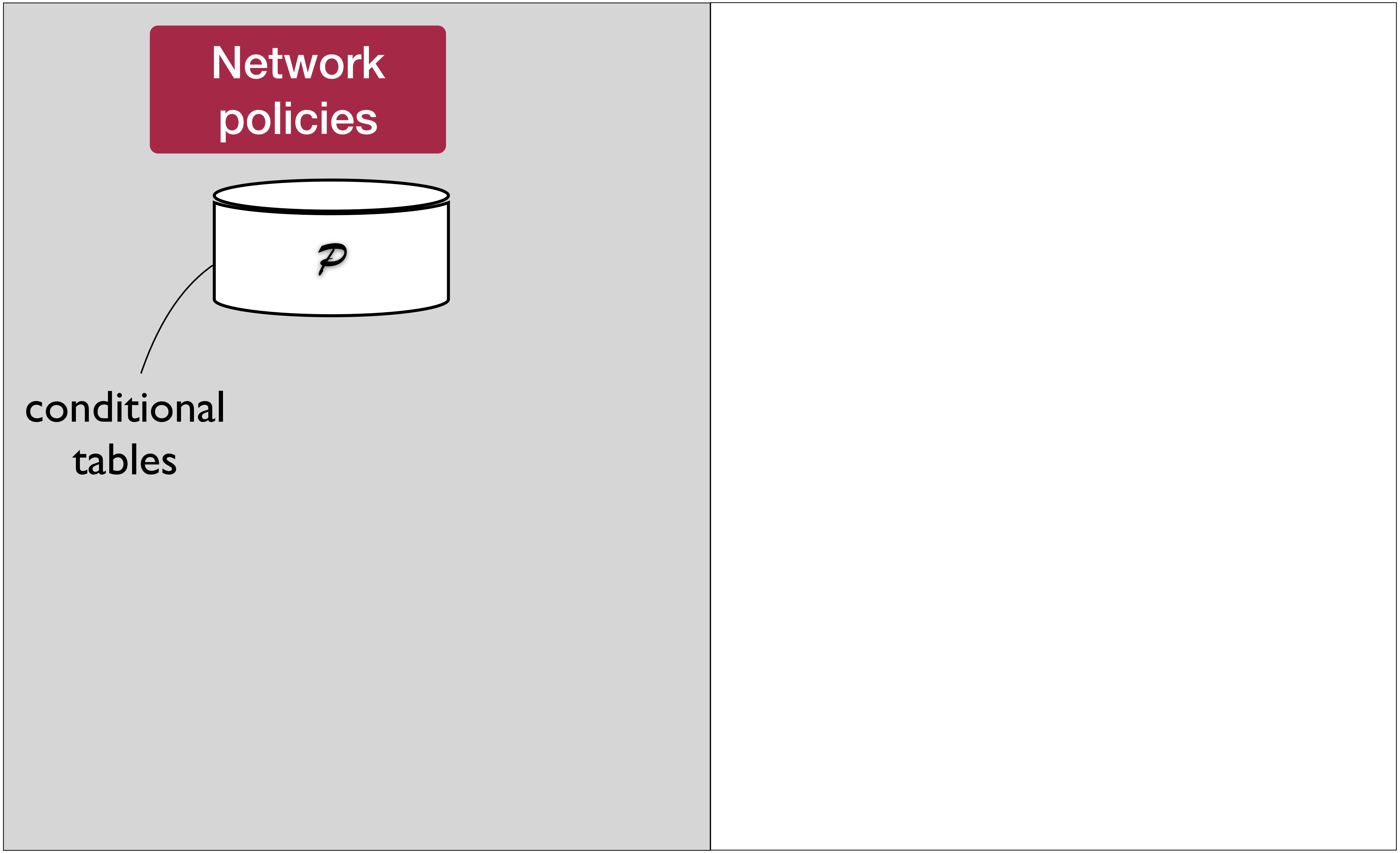
P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	$u \neq 1$
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	$v \neq 1$

$P_1 \bowtie P_3$	dest	path	flag	
	1.2.3.4	x	u	$x=[ABC] \wedge u=1$
	y	z	u	$y=5.6.7.8 \wedge z=[ABC] \wedge u \neq 1$
	y	z	v	$y=5.6.7.8 \wedge z=[ADC] \wedge v \neq 1$

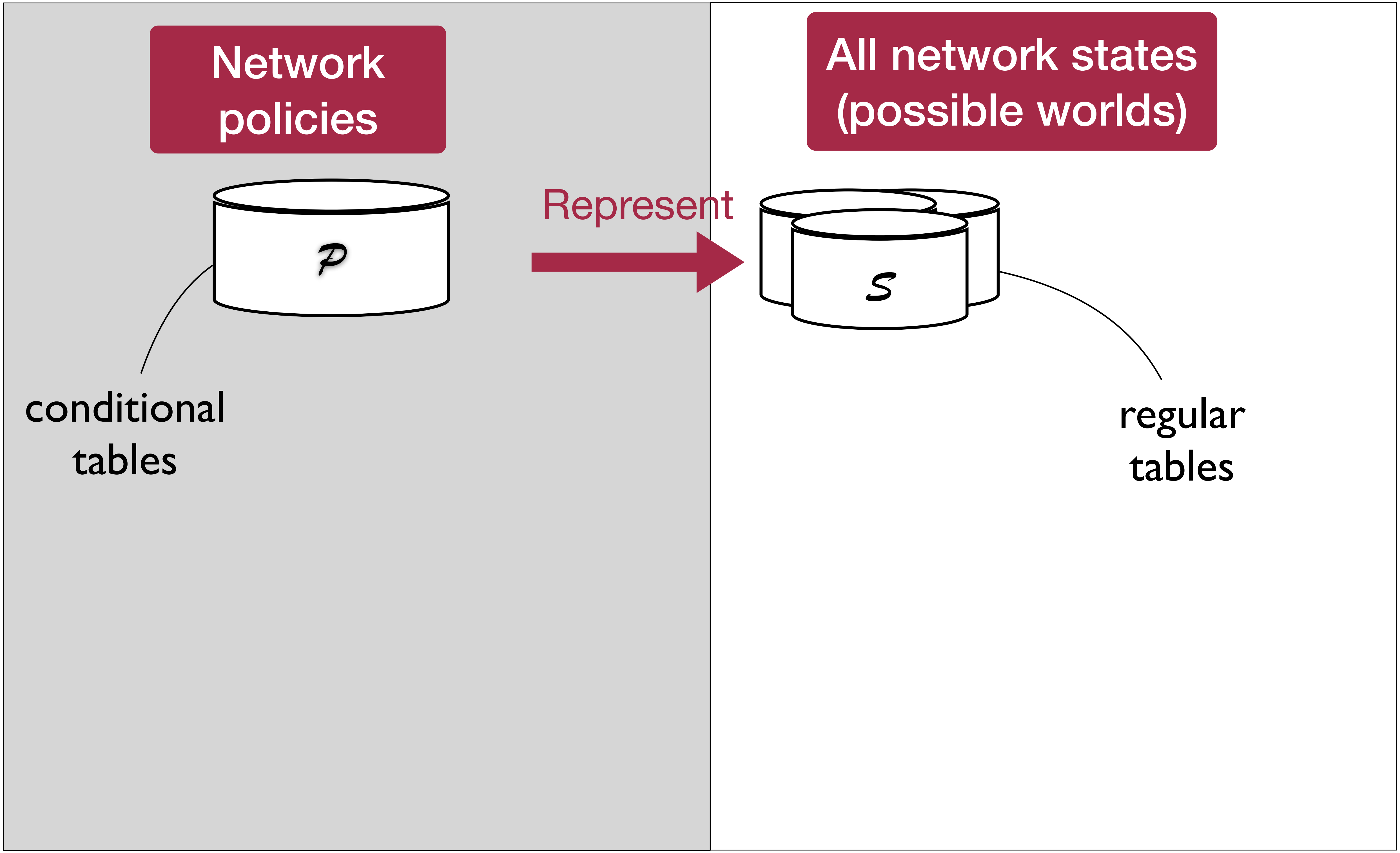
Valid forwarding state

I	dest	path
	5.6.7.8	[ABC]

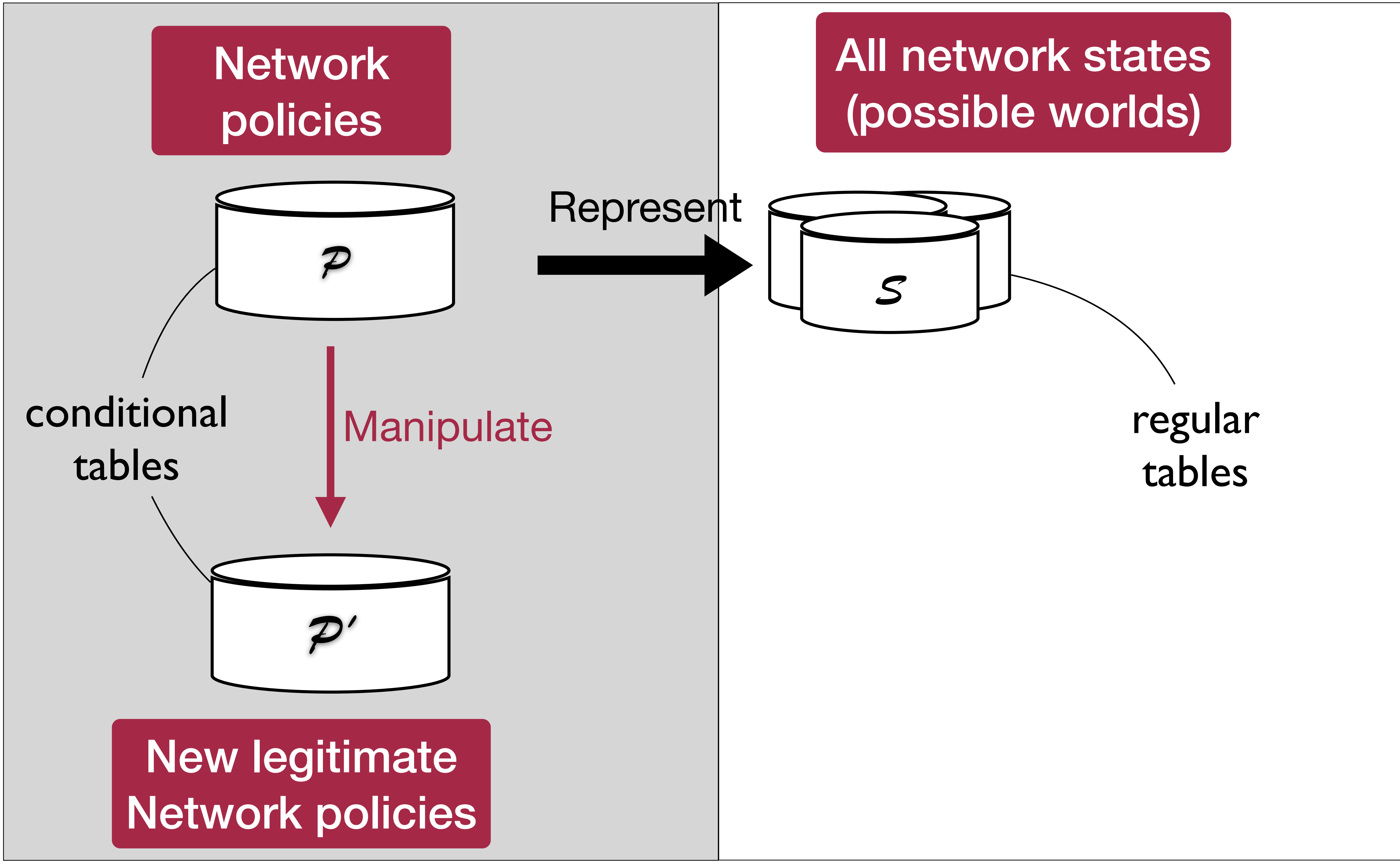
A Strong Representation System



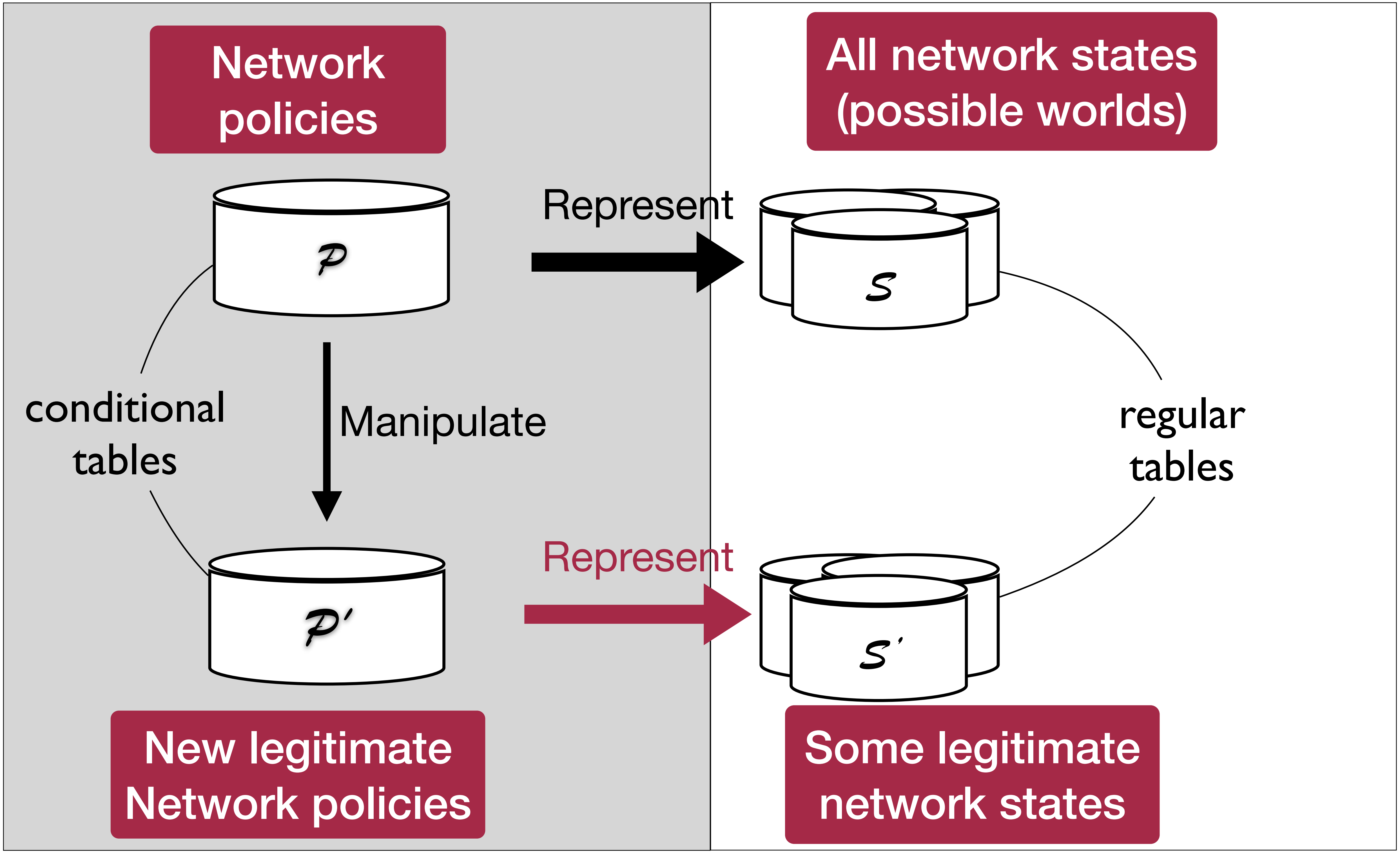
A Strong Representation System



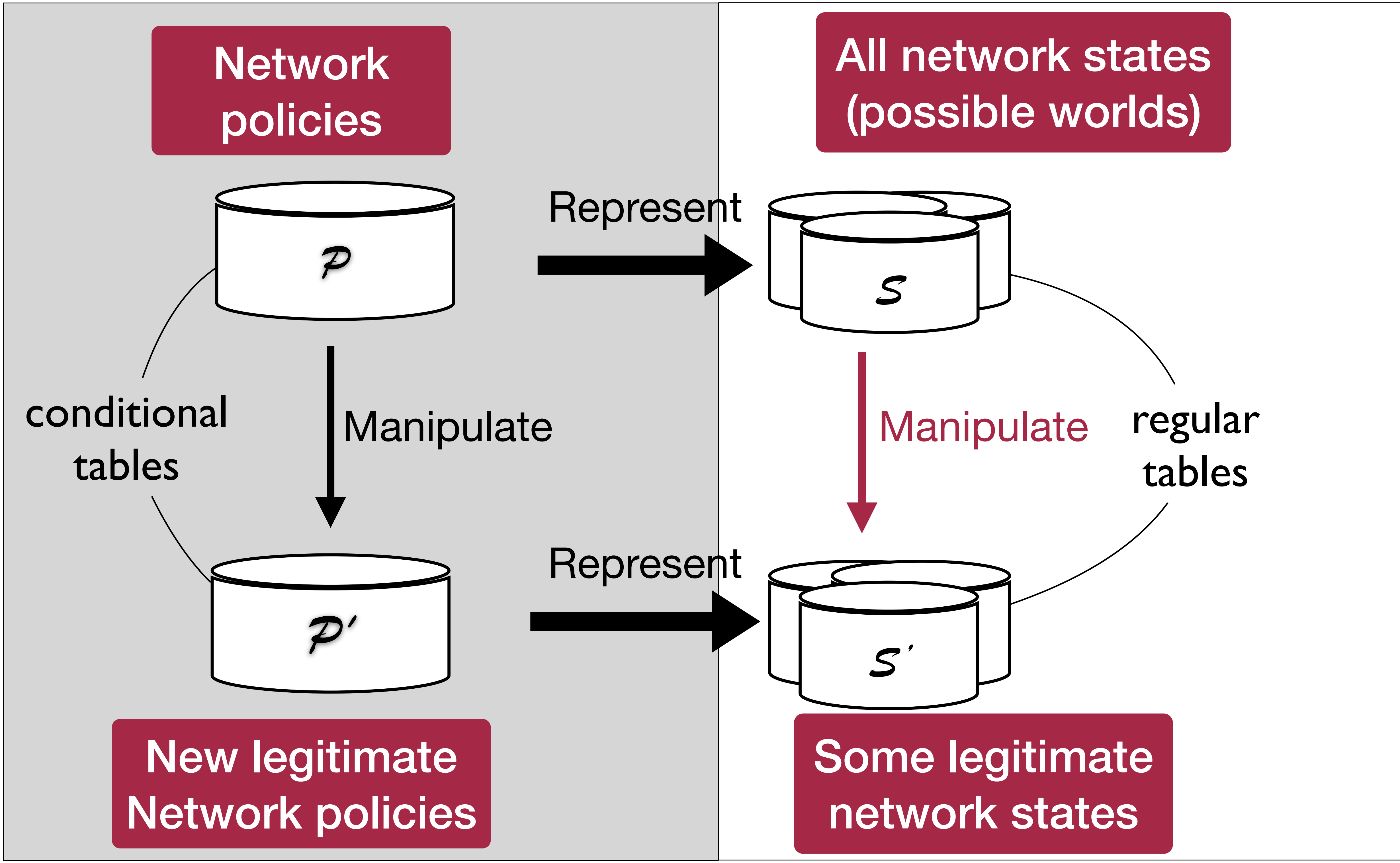
A Strong Representation System



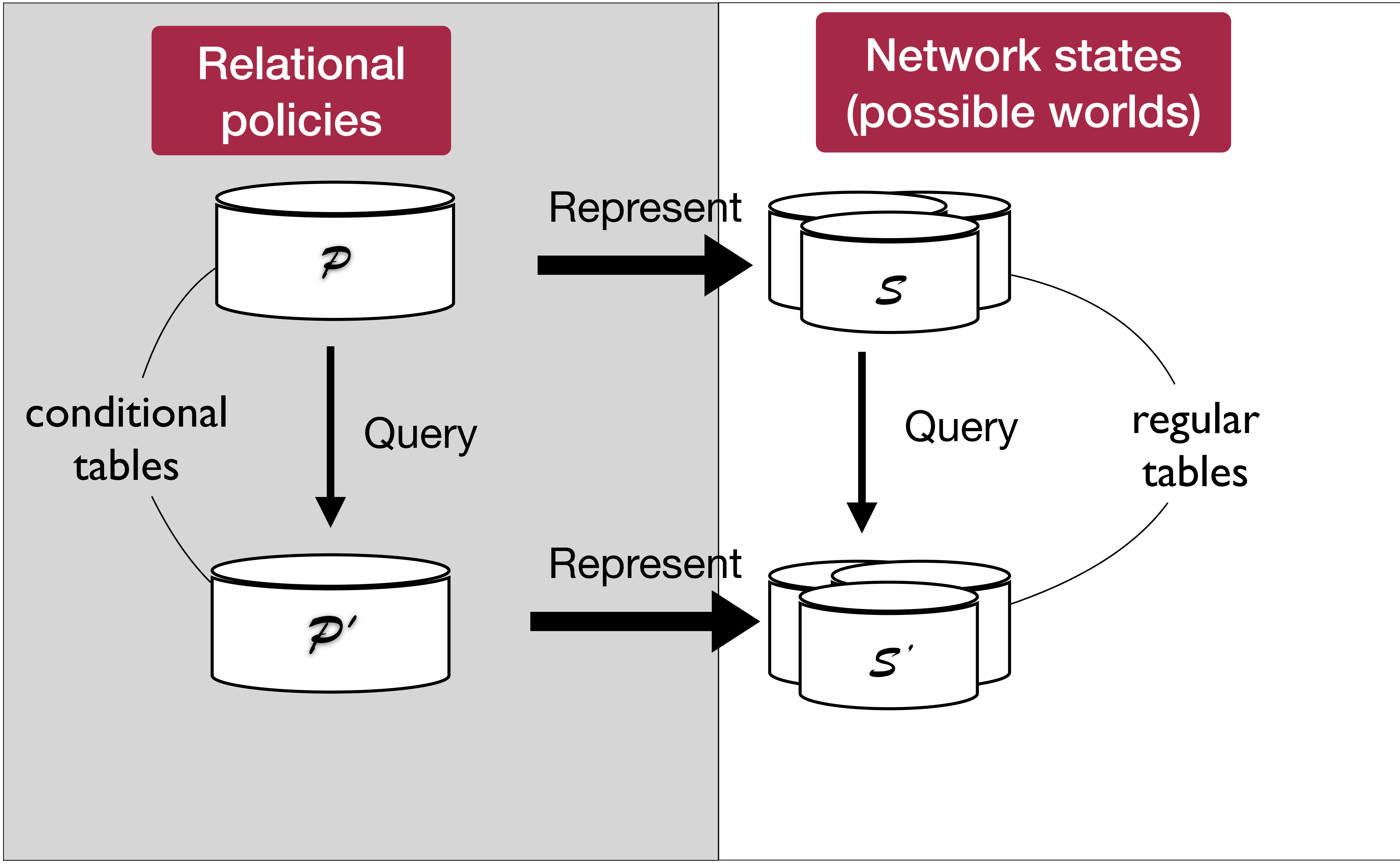
A Strong Representation System



A Strong Representation System



A Strong Representation System



Support more complex policies

- The conventional conditional table is still restricted for the rich semantics of network policies

Using user-defined functions and aggregates

Support more complex policies

- The conventional conditional table is still restricted for the rich semantics of network policies

Using user-defined functions and aggregates

P_2	dest	path	s(path)	
	x	y	z	l(y) ≤ z

Support more complex policies

- The conventional conditional table is still restricted for the rich semantics of network policies

Using user-defined functions and aggregates

P_2	dest	path	s(path)	
	x	y	z	l(y) ≤ z

Function s: return the lowest hop counts among all alternative paths

Support more complex policies

- The conventional conditional table is still restricted for the rich semantics of network policies

Using user-defined functions and aggregates

P_2	dest	path	s(path)	
	x	y	z	$l(y) \leq z$

Function s: return the lowest hop counts among all alternative paths

Function l: return hops of the path

Support Network Addressing

- Network addressing is a basic but critical feature to network area
- Allow sets in conditional table
  - i.e. accommodate variables and conditions

Support Network Addressing

- Network addressing is a basic but critical feature to network area
- Allow sets in conditional table
  - i.e. accommodate variables and conditions
- Two methods:
  - Naive support for sets
  - Leverage SMT(Satisfiability Modulo Theories) solver for sets

Application in inter-domain routing

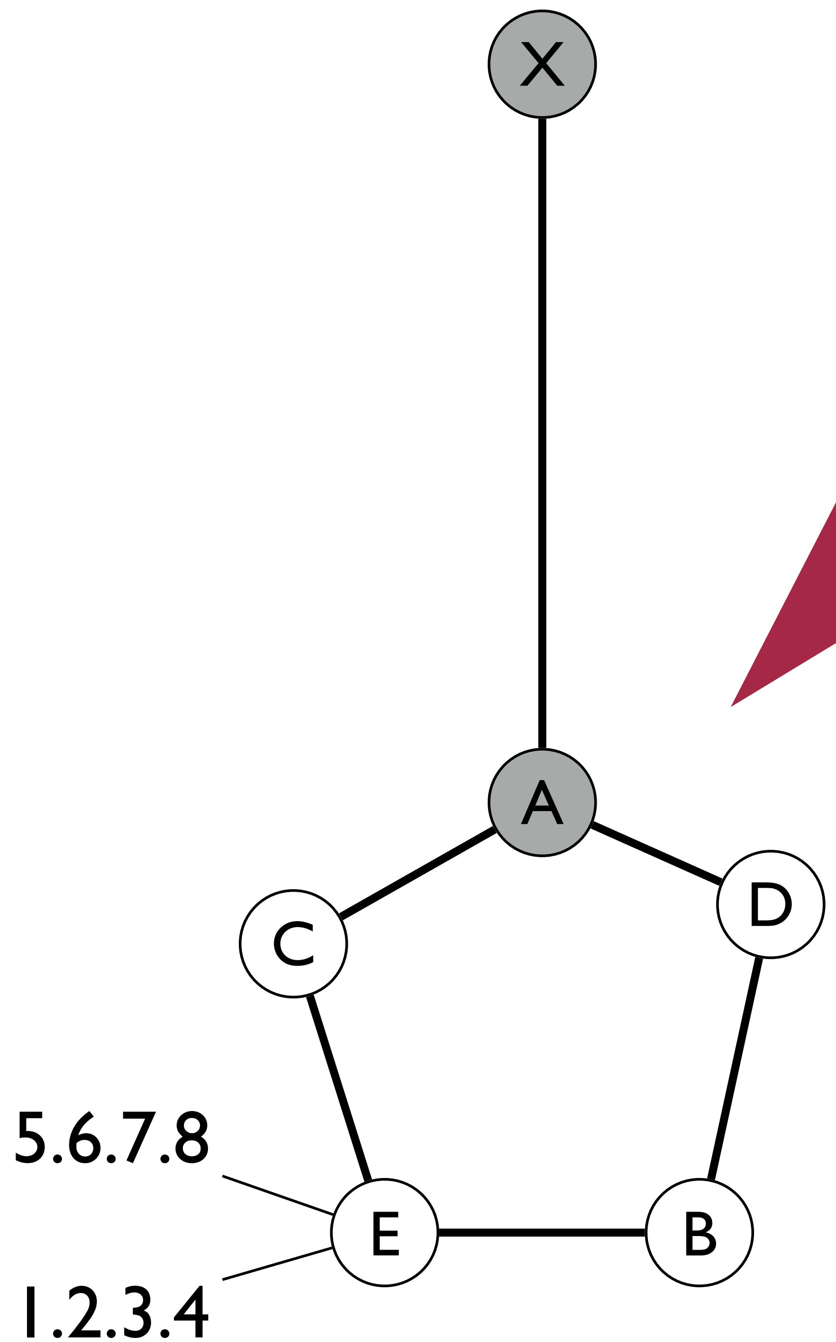
- We show the possibility of coordinating knowledge-driven policies from a single central point

- What about in realistic network scenarios like inter-domain routing?

We show an application -

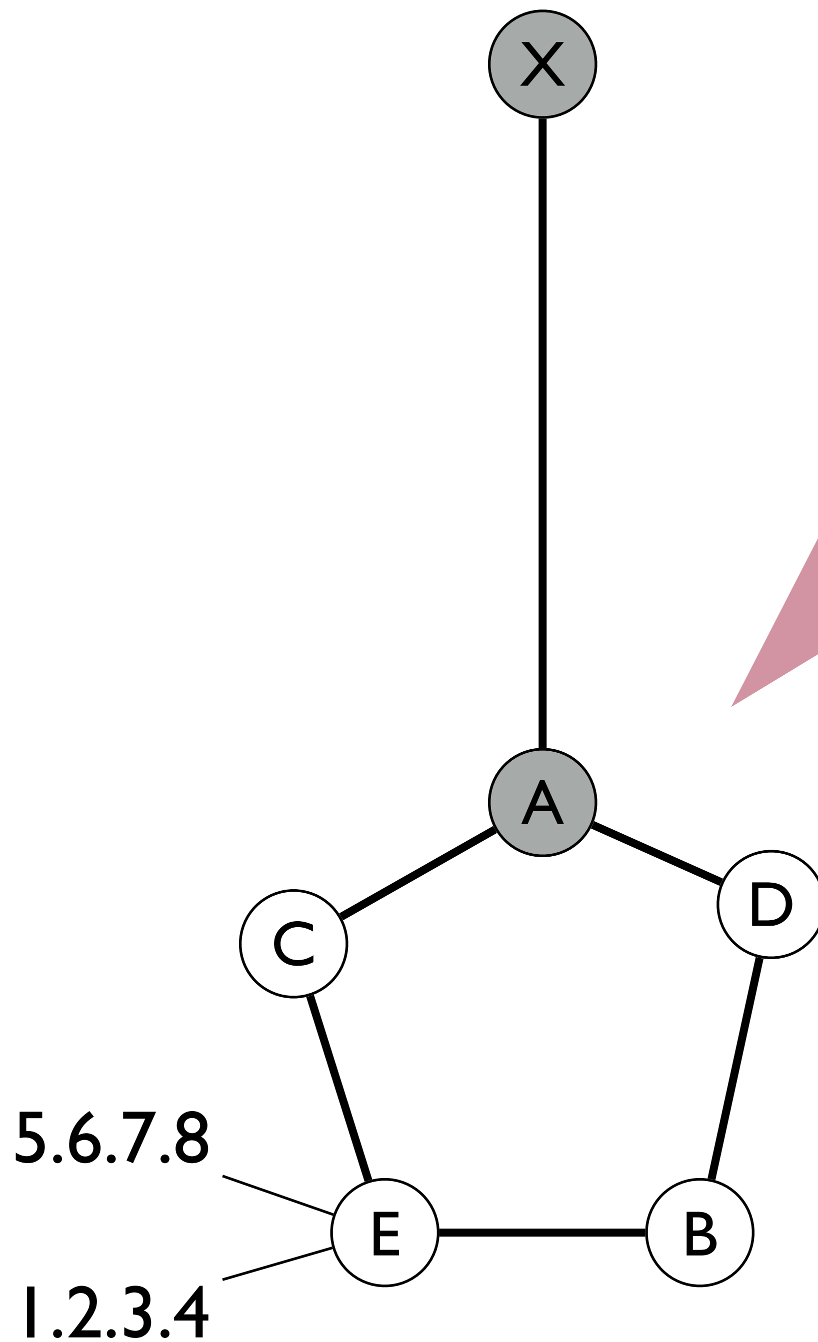
Policy Exchange

A Brief Picture of BGP



Autonomous Systems (ASs) are networks managed by a single enterprise or service provider

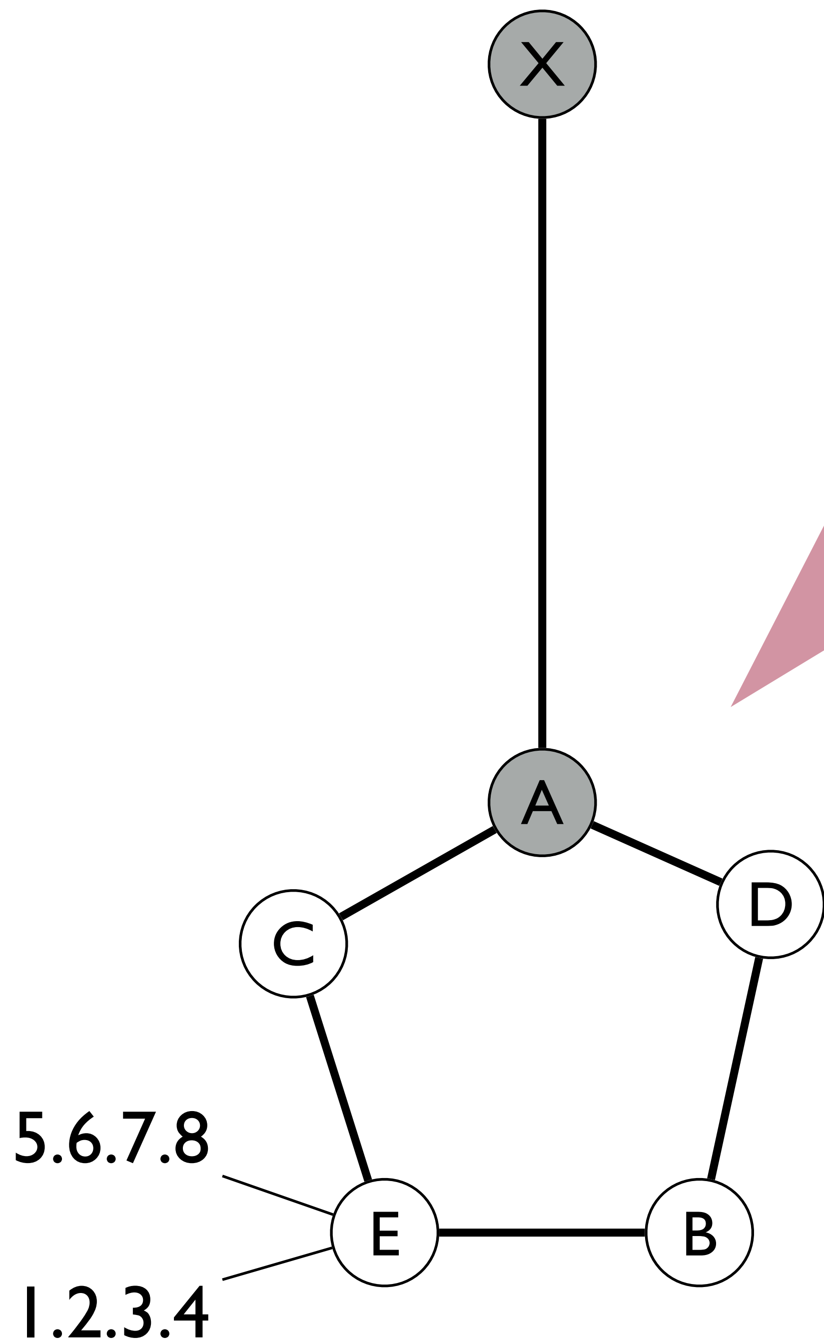
A Brief Picture of BGP



Autonomous Systems (ASs) are networks managed by a single enterprise or service provider

BGP is a shortest path routing protocol

A Brief Picture of BGP



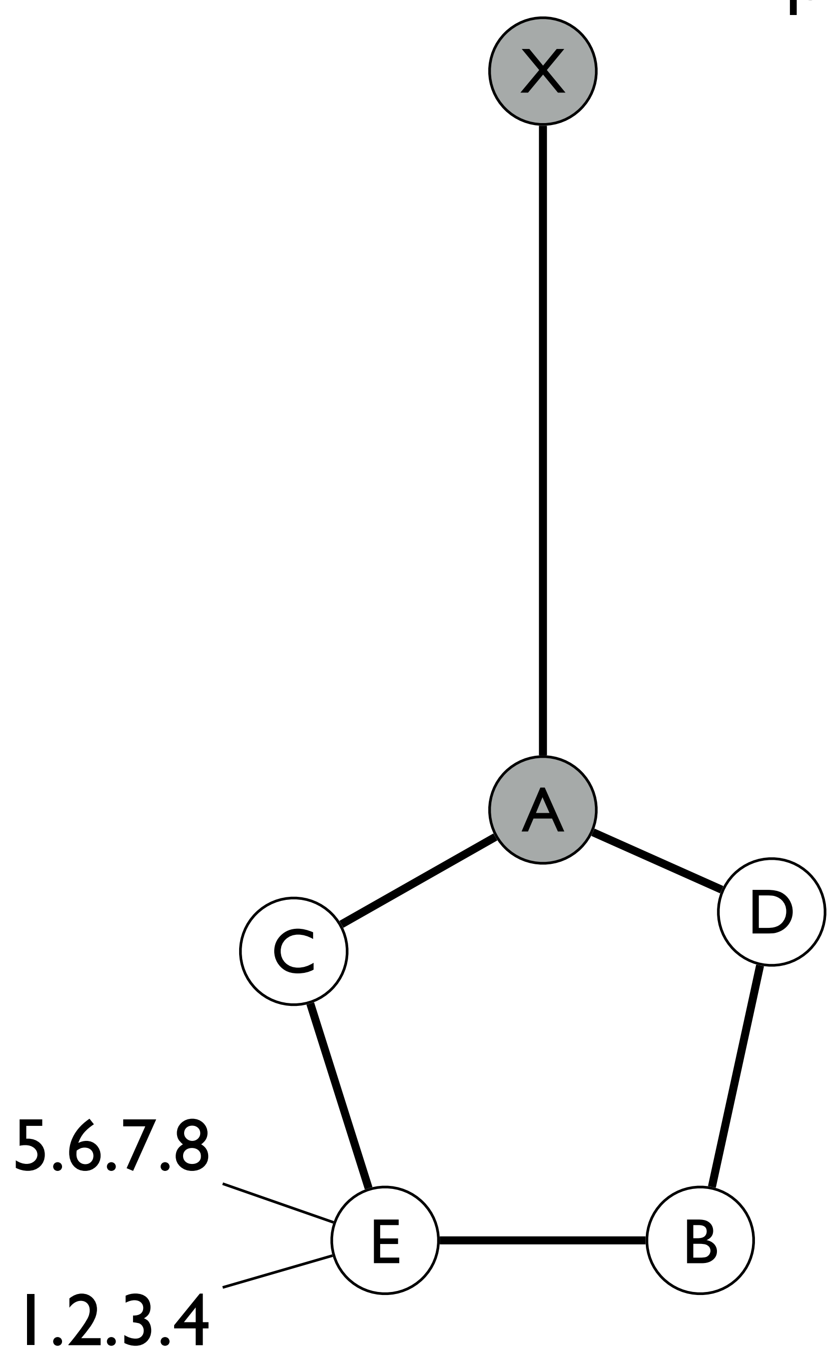
Autonomous Systems (ASs) are networks managed by a single enterprise or service provider

BGP is a shortest path routing protocol

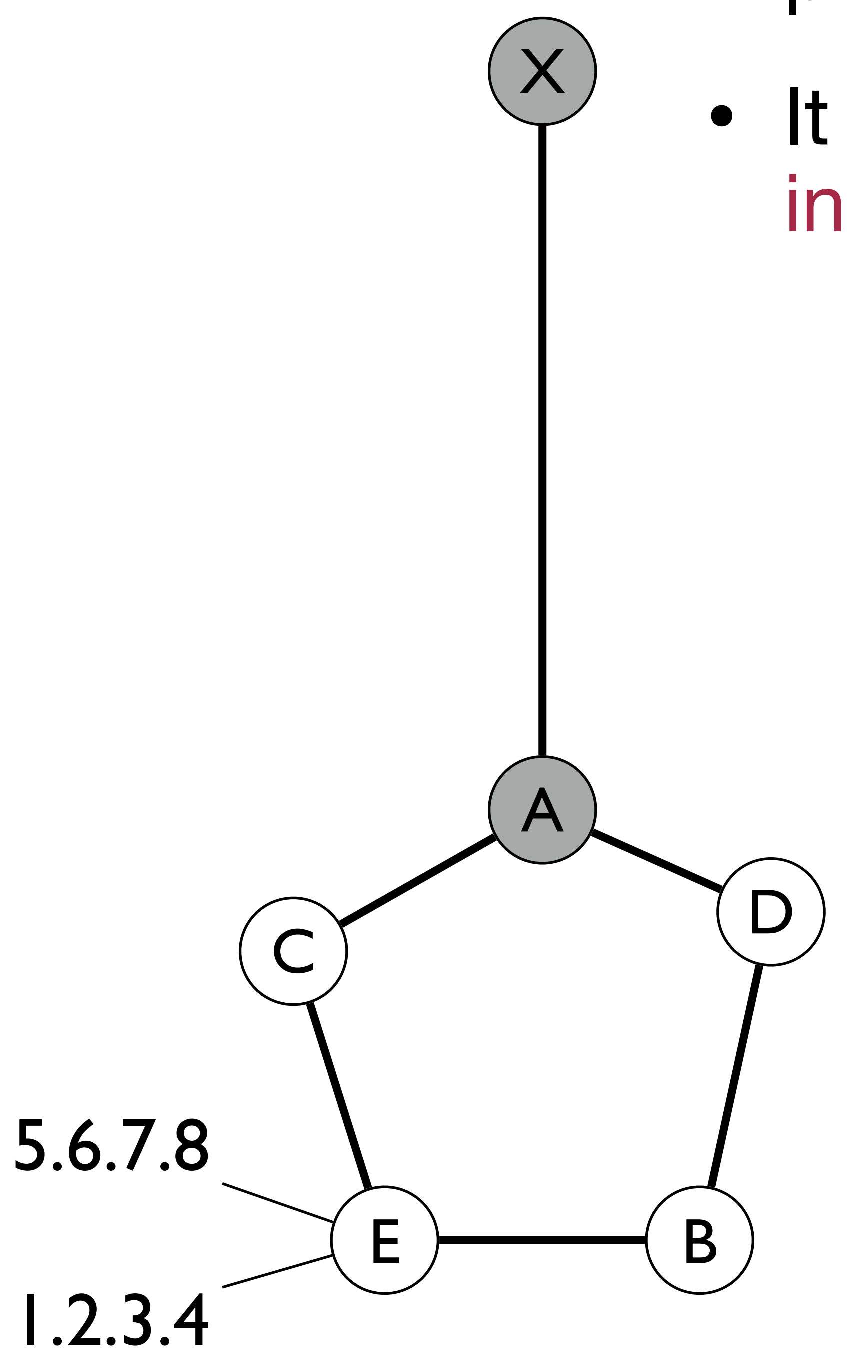
BGP makes routing decision favoring autonomy

Why policy exchange?

- BGP favors autonomy; Policies are made based on local preferences



Why policy exchange?



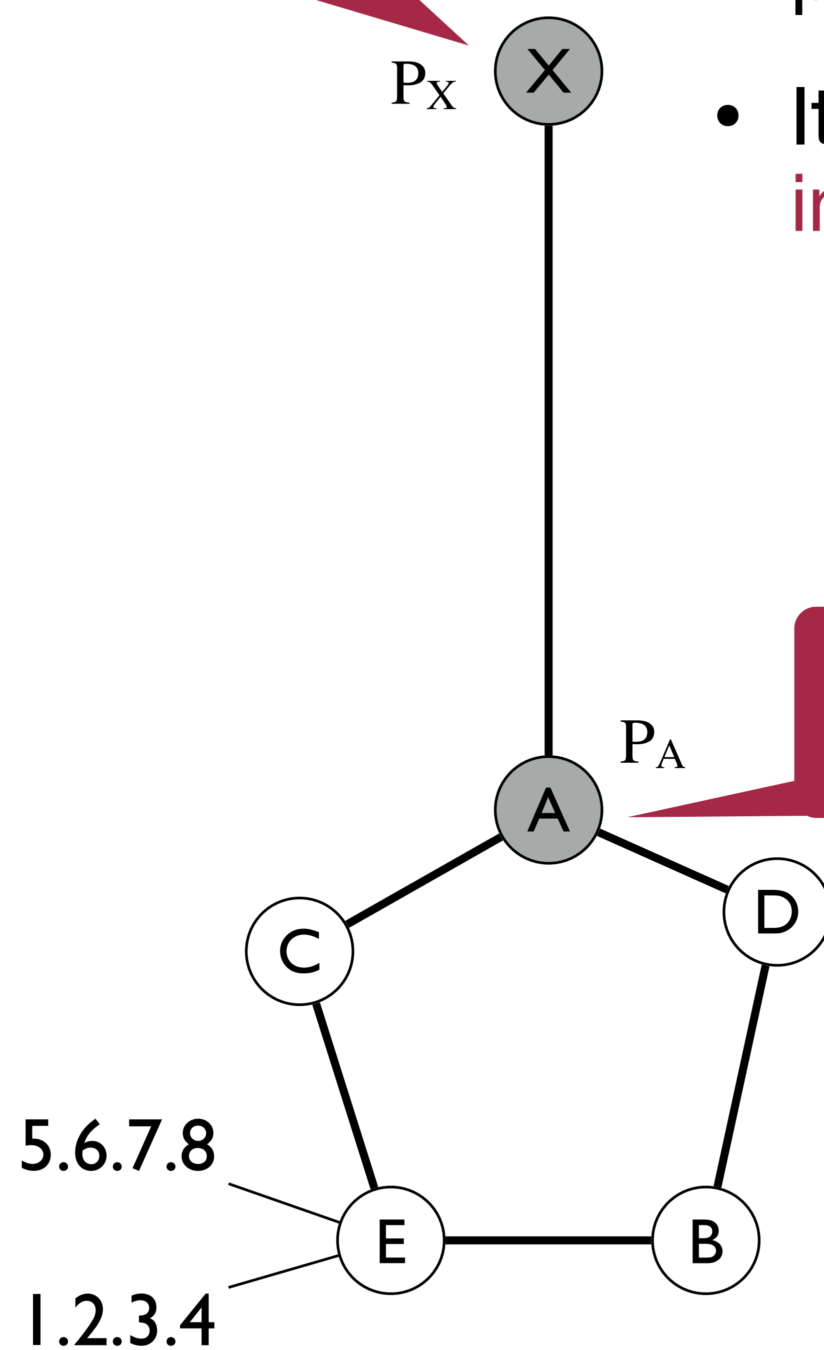
- BGP favors autonomy; Policies are made based on local preferences
- It may cause unwanted interaction between policies

Policy exchange - counter example

X: requires routes no more than 3 hops to 1.2.3.4

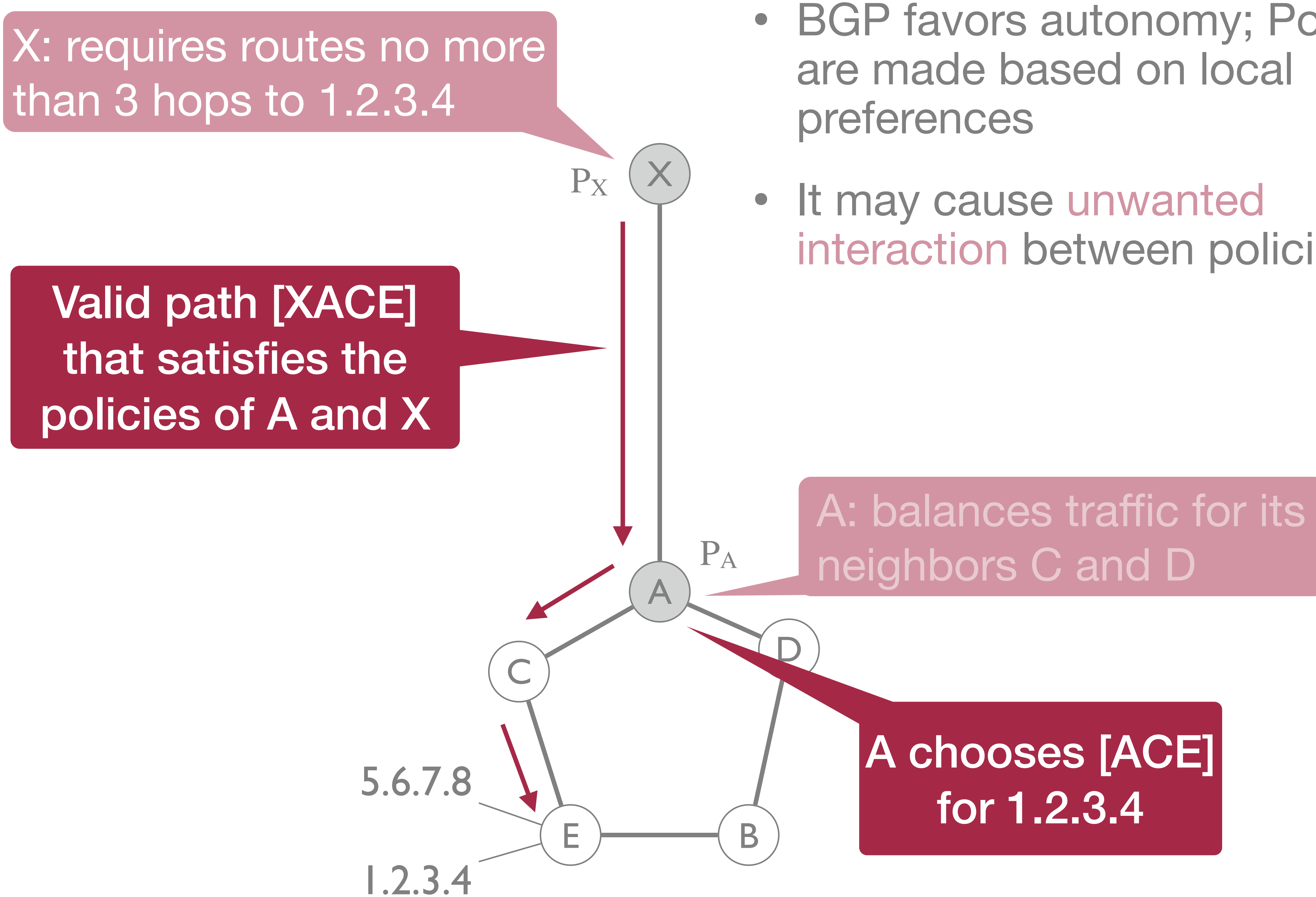
- BGP favors autonomy; Policies are made based on local preferences
- It may cause unwanted interaction between policies

A: balances traffic for its neighbors C and D



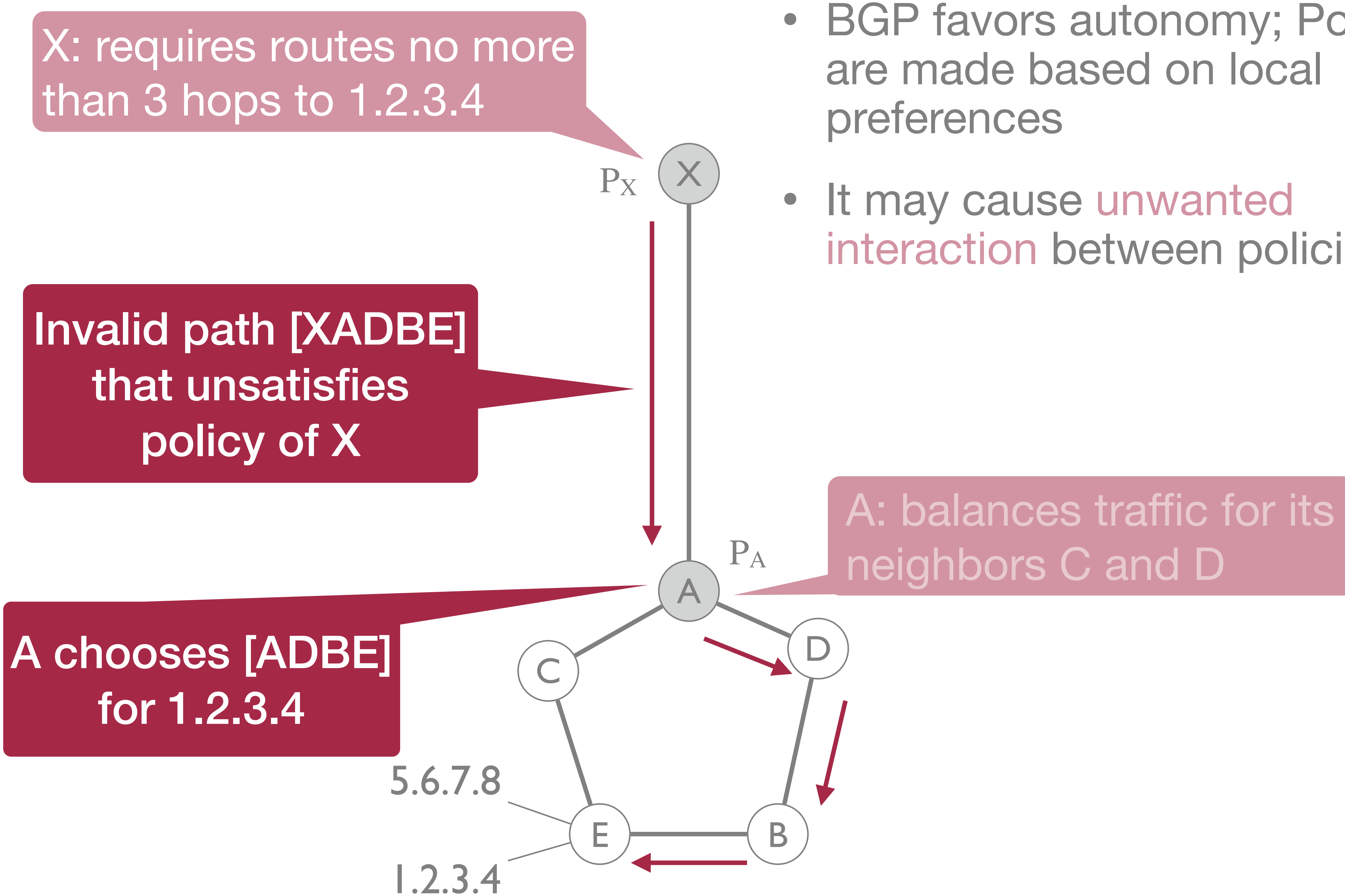
Policy exchange - counter example

- BGP favors autonomy; Policies are made based on local preferences
- It may cause unwanted interaction between policies



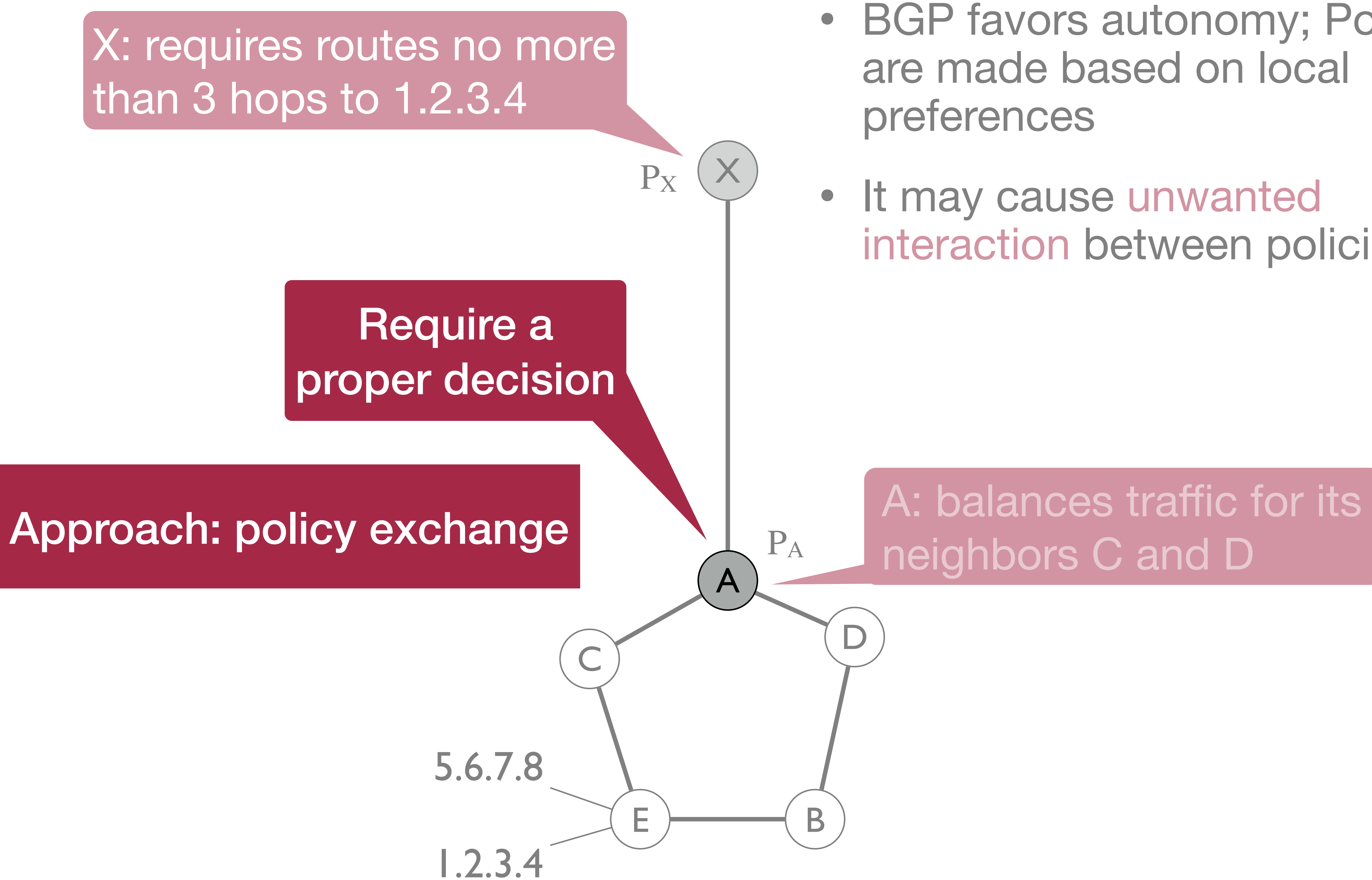
Policy exchange - counter example

- BGP favors autonomy; Policies are made based on local preferences
- It may cause unwanted interaction between policies



Policy exchange - counter example

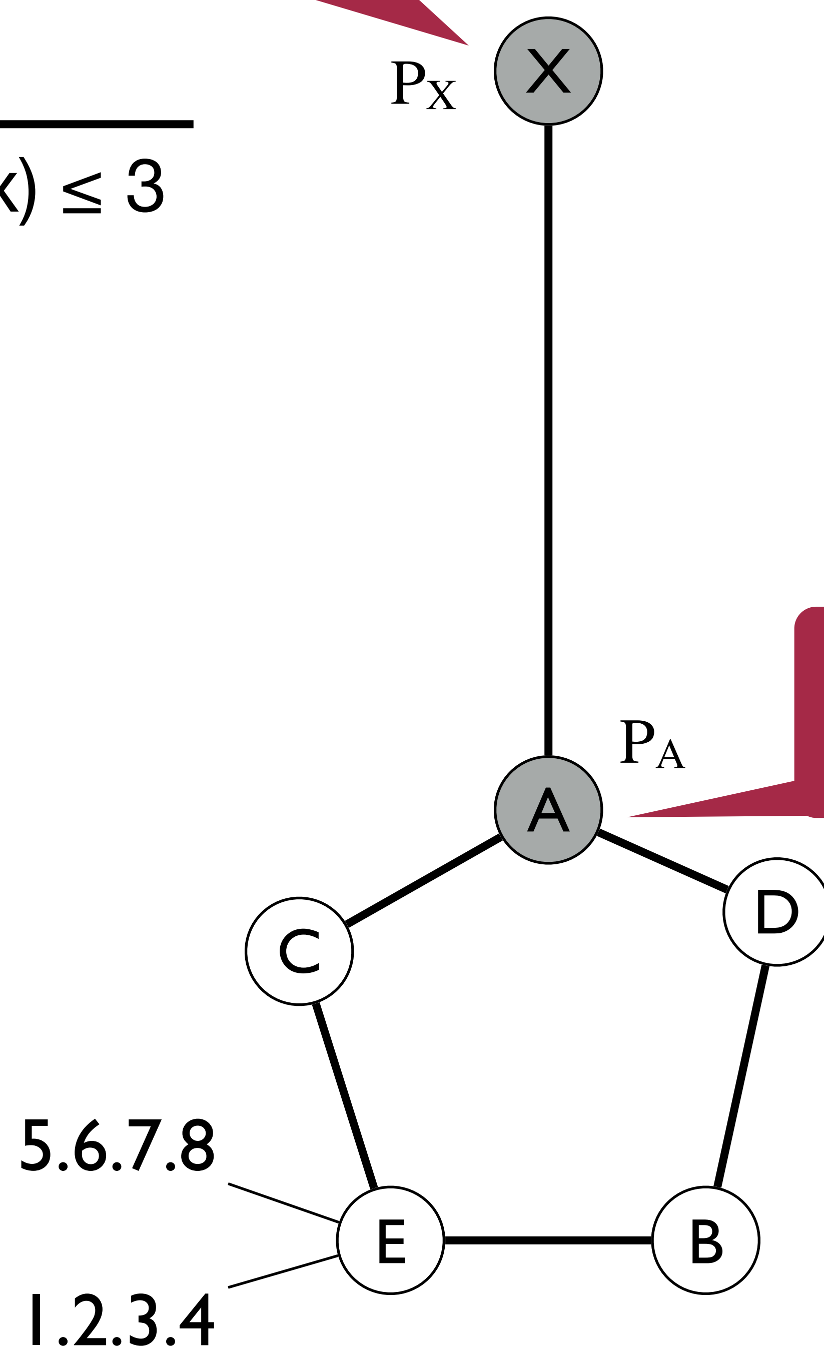
- BGP favors autonomy; Policies are made based on local preferences
- It may cause unwanted interaction between policies



How to do policy exchange?

X: requires routes no more than 3 hops to 1.2.3.4

P_X	dest	path	
	1.2.3.4	x	$l(x) \leq 3$



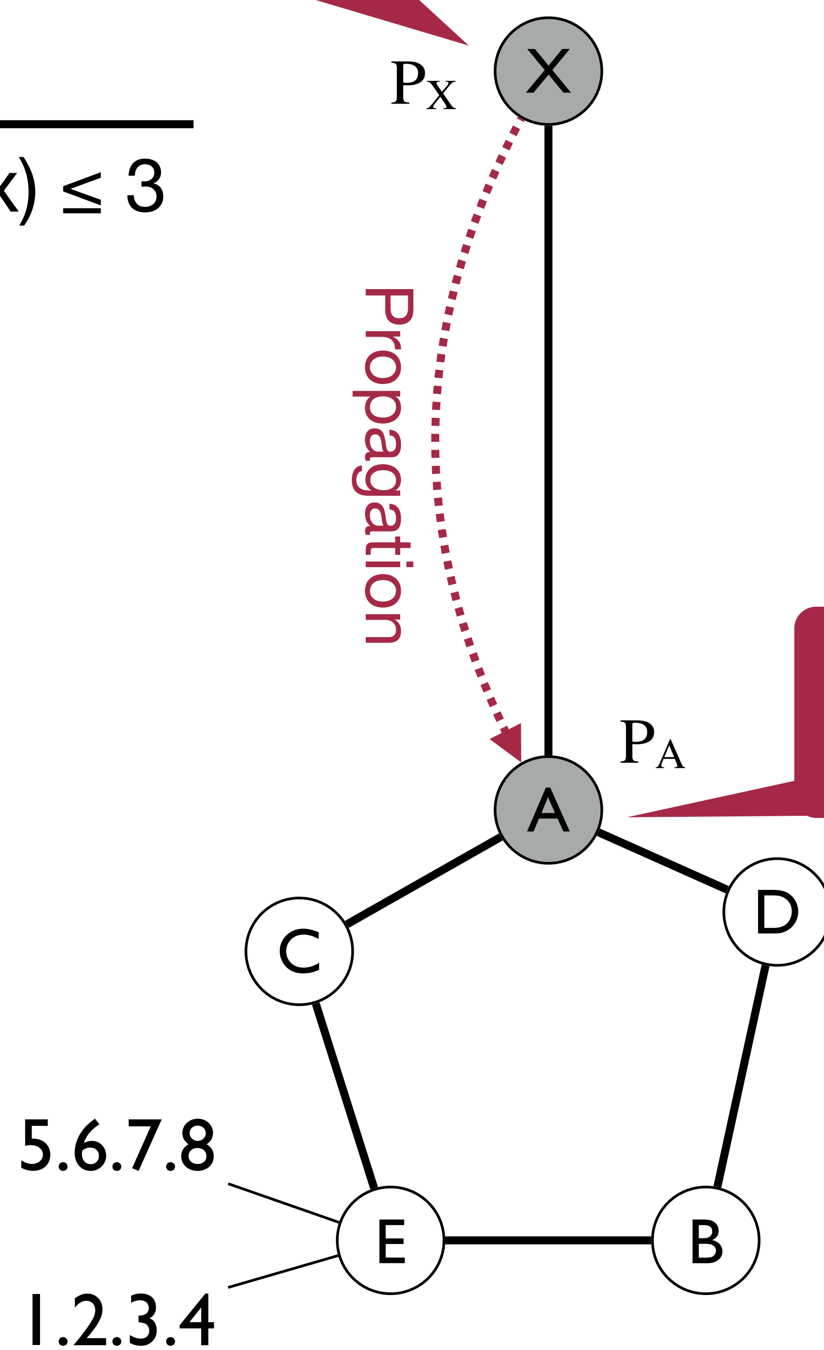
A: balances traffic for its neighbors C and D

P_A	dest	nh	flag	
	1.2.3.4	C	u	u = 1
	1.2.3.4	D	v	v = 1
	5.6.7.8	C	u	u ≠ 1
	5.6.7.8	D	v	v ≠ 1

How to do policy exchange?

X: requires routes no more than 3 hops to 1.2.3.4

P_X	dest	path	
	1.2.3.4	x	$l(x) \leq 3$



A: balances traffic for its neighbors C and D

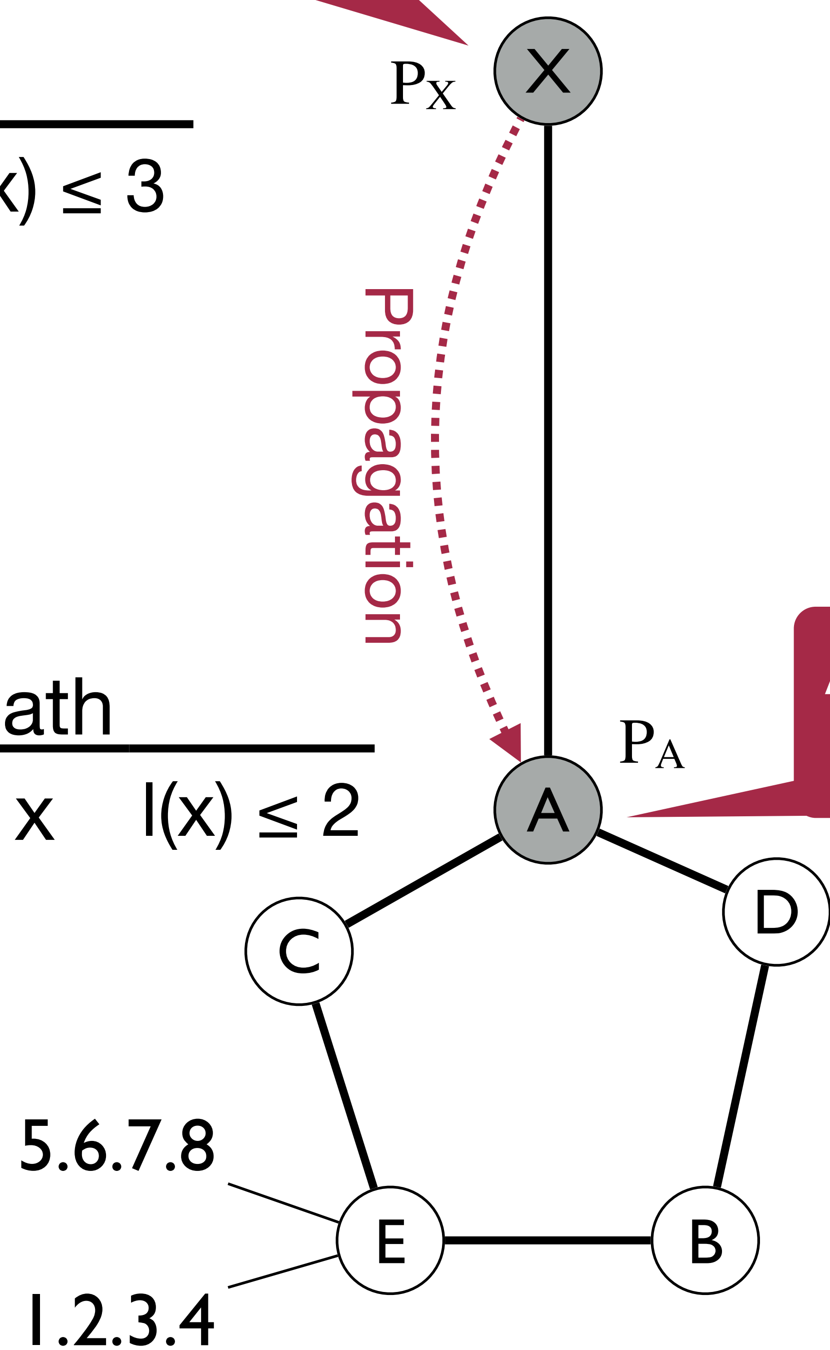
P_A	dest	nh	flag	
	1.2.3.4	C	u	$u = 1$
	1.2.3.4	D	v	$v = 1$
	5.6.7.8	C	u	$u \neq 1$
	5.6.7.8	D	v	$v \neq 1$

How to do policy exchange?

X: requires routes no more than 3 hops to 1.2.3.4

P_X	dest	path	
	1.2.3.4	x	$l(x) \leq 3$

P_X'	dest	nh	flag	path	
	1.2.3.4	-	-	x	$l(x) \leq 2$



A: balances traffic for its neighbors C and D

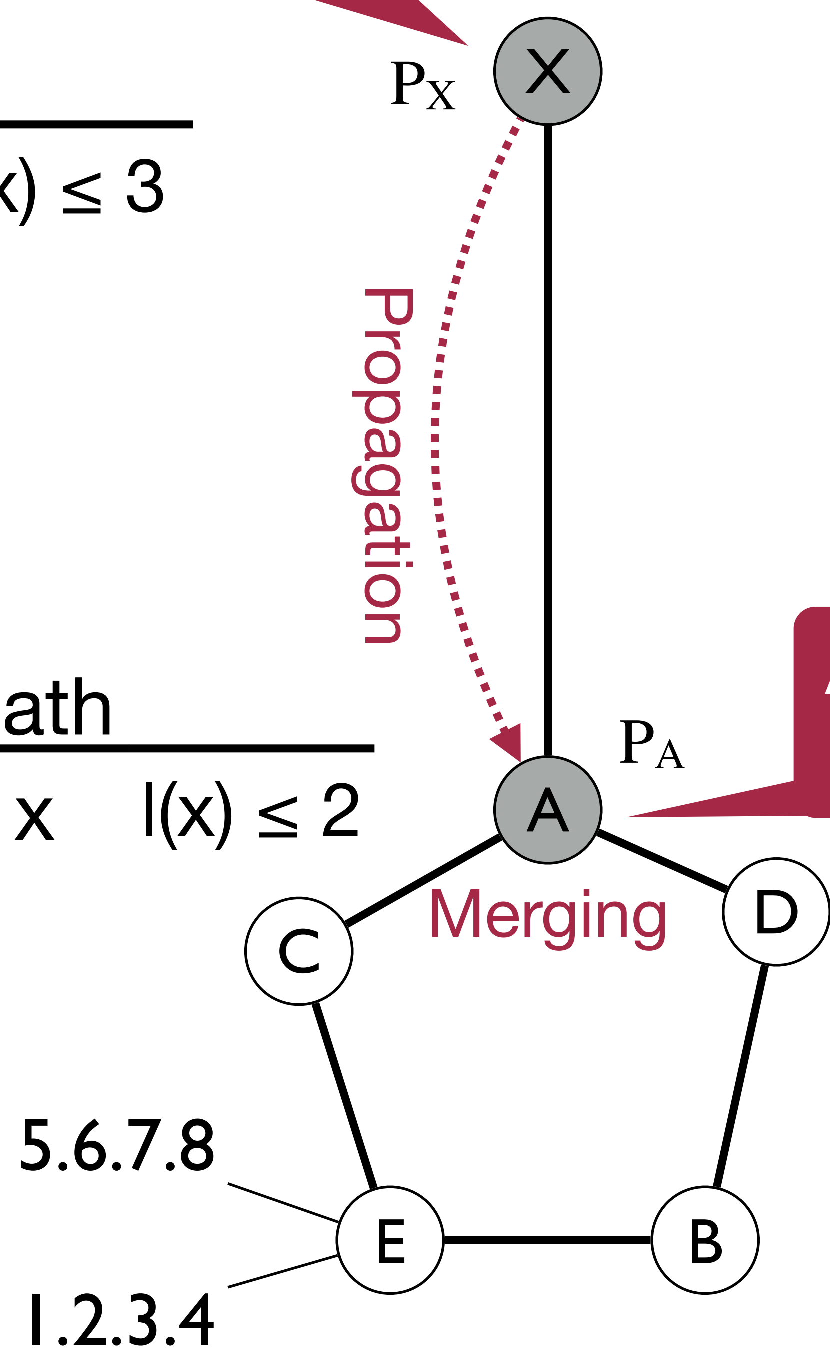
P_A	dest	nh	flag	
	1.2.3.4	C	u	$u = 1$
	1.2.3.4	D	v	$v = 1$
	5.6.7.8	C	u	$u \neq 1$
	5.6.7.8	D	v	$v \neq 1$

How to do policy exchange?

X: requires routes no more than 3 hops to 1.2.3.4

P_X	dest	path	
	1.2.3.4	x	$l(x) \leq 3$

P_X'	dest	nh	flag	path	
	1.2.3.4	-	-	x	$l(x) \leq 2$



A: balances traffic for its neighbors C and D

P_A	dest	nh	flag	
	1.2.3.4	C	u	$u = 1$
	1.2.3.4	D	v	$v = 1$
	5.6.7.8	C	u	$u \neq 1$
	5.6.7.8	D	v	$v \neq 1$

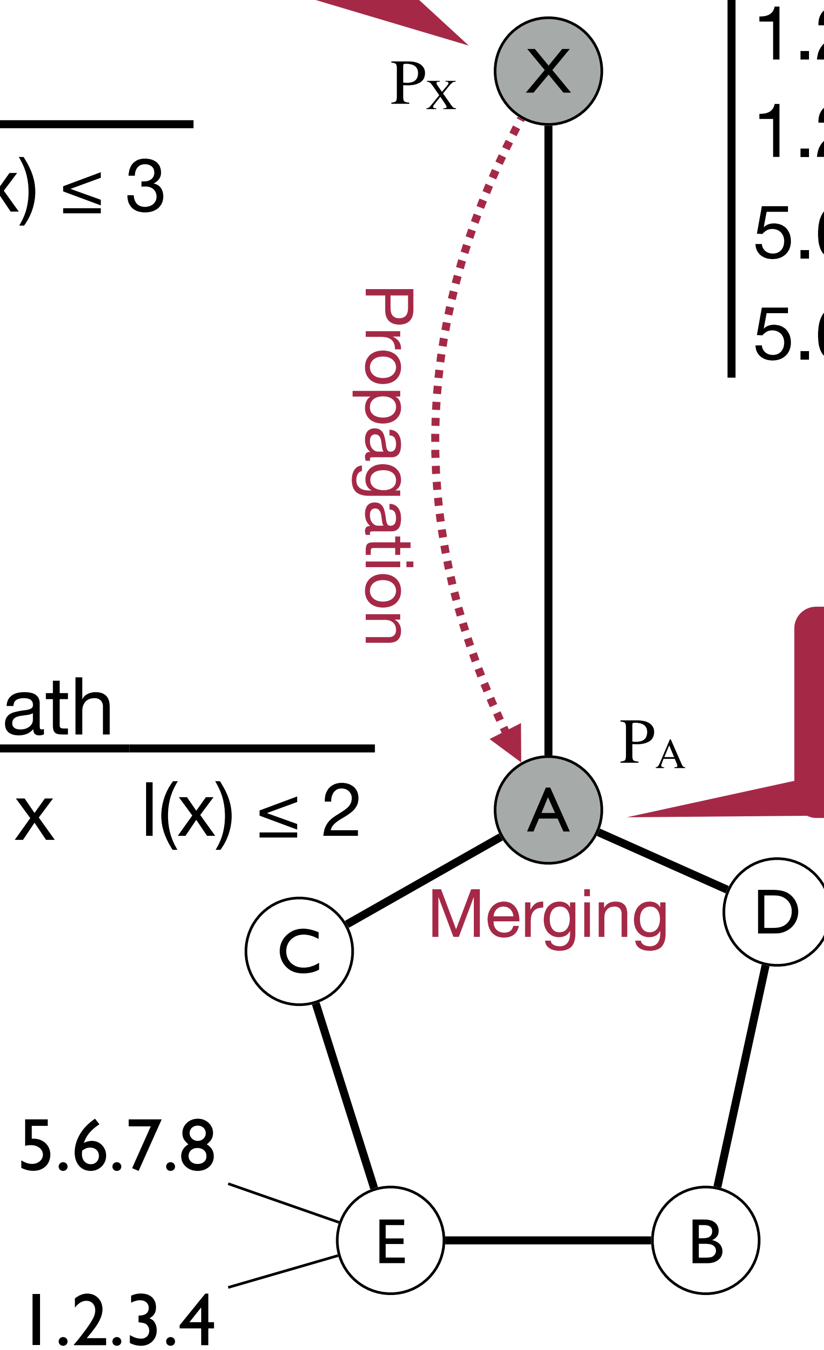
How to do policy exchange?

X: requires routes no more than 3 hops to 1.2.3.4

P_X	dest	path	
	1.2.3.4	x	l(x) ≤ 3

P_X'	dest	nh	flag	path	
	1.2.3.4	-	-	x	l(x) ≤ 2

P_A'	dest	nh	flag	path	
	1.2.3.4	C	u	x	u = 1 ∧ l(x) ≤ 2
	1.2.3.4	D	v	x	v = 1 ∧ l(x) ≤ 2
	5.6.7.8	C	u	-	u ≠ 1
	5.6.7.8	D	v	-	v ≠ 1



A: balances traffic for its neighbors C and D

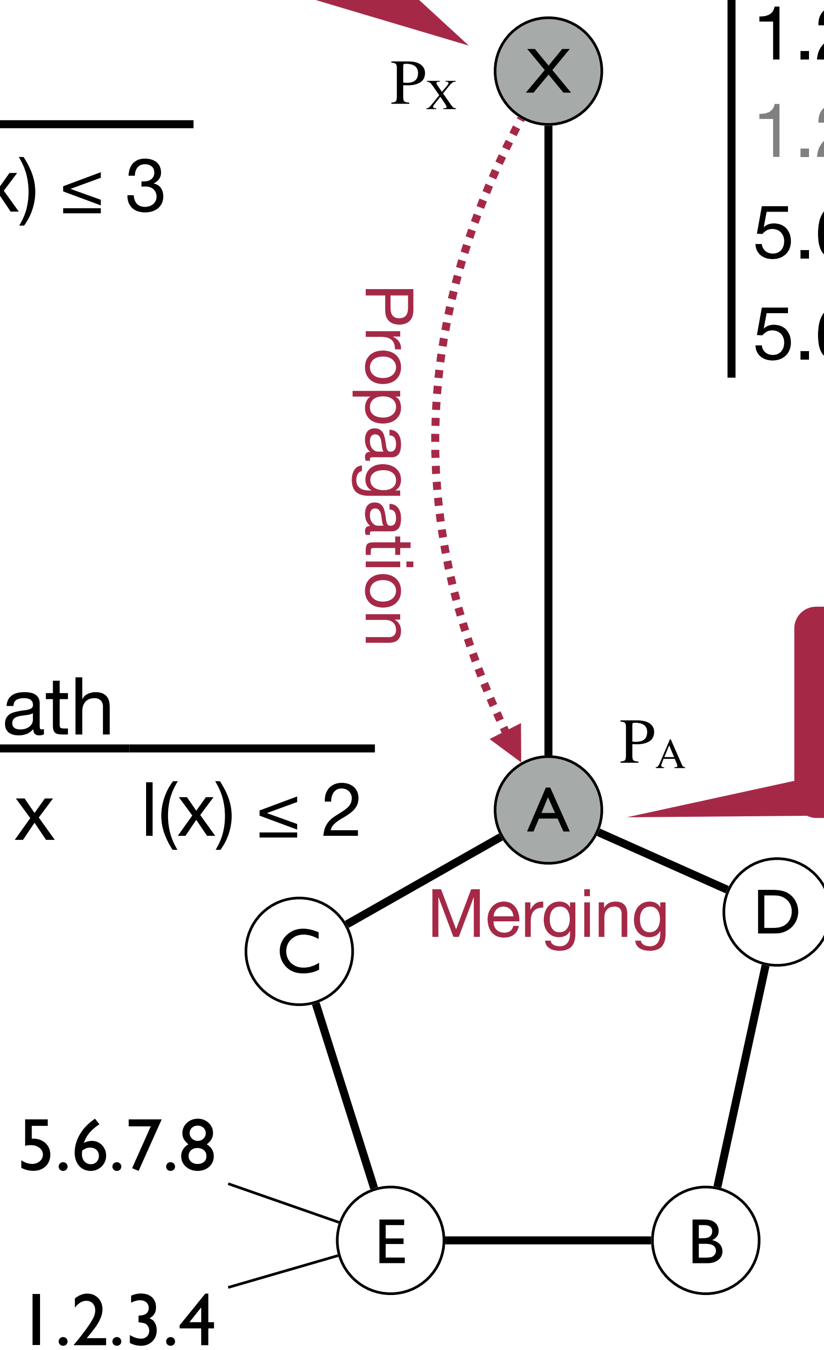
P_A	dest	nh	flag	
	1.2.3.4	C	u	u = 1
	1.2.3.4	D	v	v = 1
	5.6.7.8	C	u	u ≠ 1
	5.6.7.8	D	v	v ≠ 1

How to do policy exchange?

X: requires routes no more than 3 hops to 1.2.3.4

P_X	dest	path	
	1.2.3.4	x	$l(x) \leq 3$

P_X'	dest	nh	flag	path	
	1.2.3.4	-	-	x	$l(x) \leq 2$



P_A'	dest	nh	flag	path	
	1.2.3.4	C	u	x	$u = 1 \wedge l(x) \leq 2$
	1.2.3.4	D	v	x	$v = 1 \wedge l(x) \leq 2$
	5.6.7.8	C	u	-	$u \neq 1$
	5.6.7.8	D	v	-	$v \neq 1$

A: balances traffic for its neighbors C and D

P_A	dest	nh	flag	
	1.2.3.4	C	u	$u = 1$
	1.2.3.4	D	v	$v = 1$
	5.6.7.8	C	u	$u \neq 1$
	5.6.7.8	D	v	$v \neq 1$

Preliminary Prototype

- We currently use in-memory implementation, in terms of Python lists

Preliminary Prototype

- We currently use in-memory implementation, in terms of Python lists

- Applications:
  - Policy Exchange
  - BGP simulation

Preliminary Prototype

- We currently use in-memory implementation, in terms of Python lists

- Applications:
  - Policy Exchange
  - BGP simulation

Mimicking the behavior of the BGP speaker as it receives announcement from their neighbors

Preliminary Prototype

- We currently use in-memory implementation, in terms of Python lists

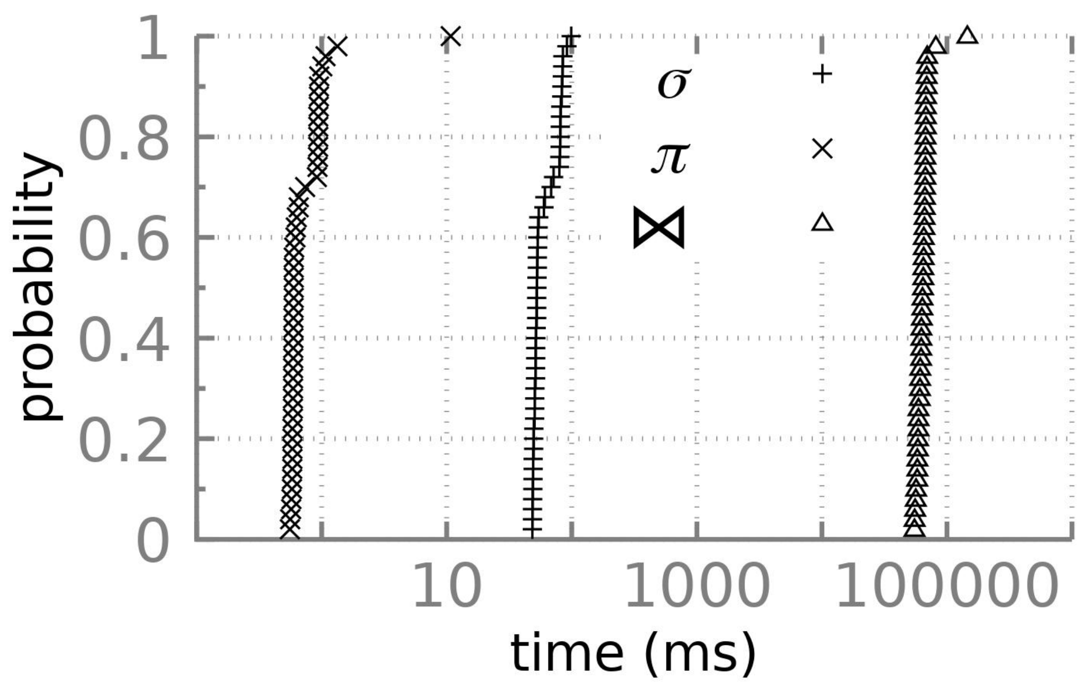
- Applications:
  - Policy Exchange
  - BGP simulation

Benchmark

- Using MRT format RouteView BGP data - RIBs and UPDATEs - from route-view2.oregon-ix.net to generate synthetic policies and realistic topologies

Performance - Relational Operators

Projection Selection ⋈



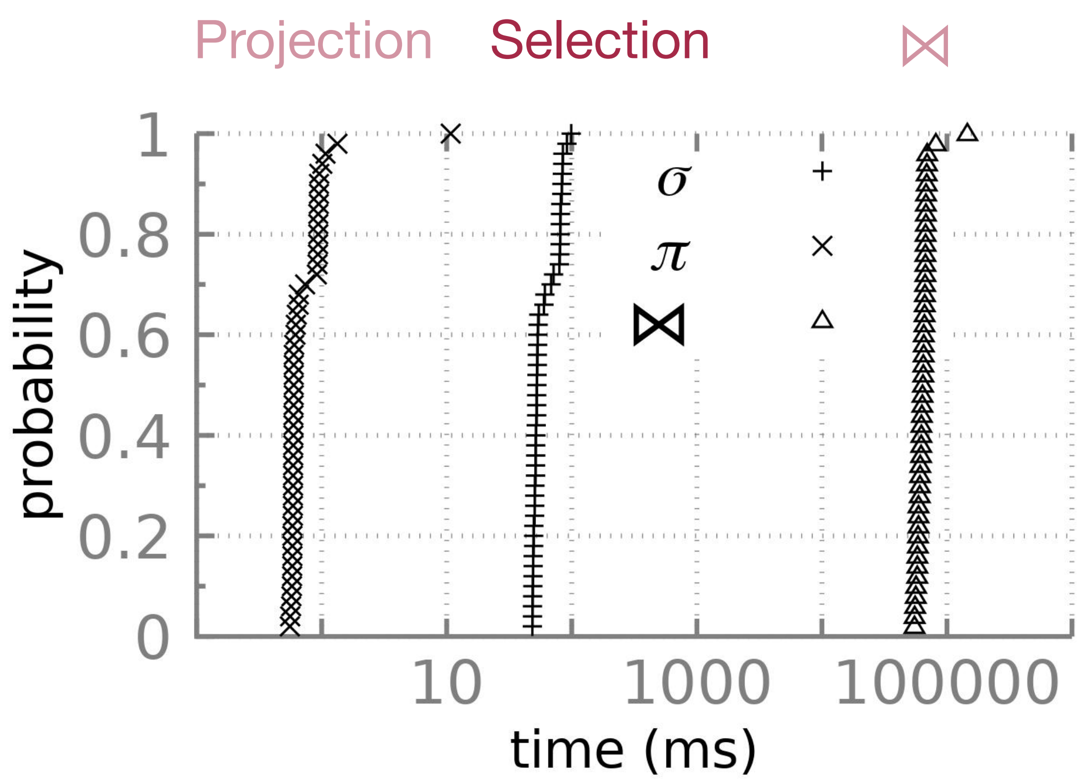
Realistic destination

⋈: $P_1 \bowtie P_3$

P_1: static-route & filter policy

P_3: traffic balance policy

Performance - Relational Operators



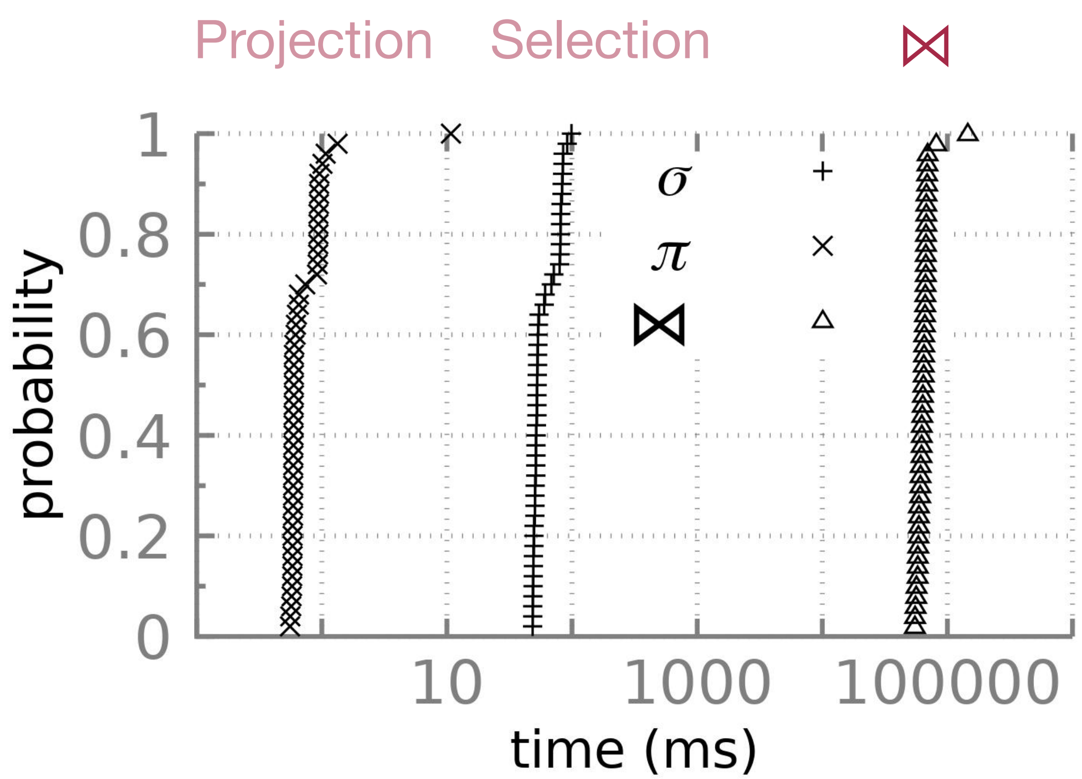
Realistic destination

⋈: $P_1 \bowtie P_3$

P_1: static-route & filter policy

P_3: traffic balance policy

Performance - Relational Operators



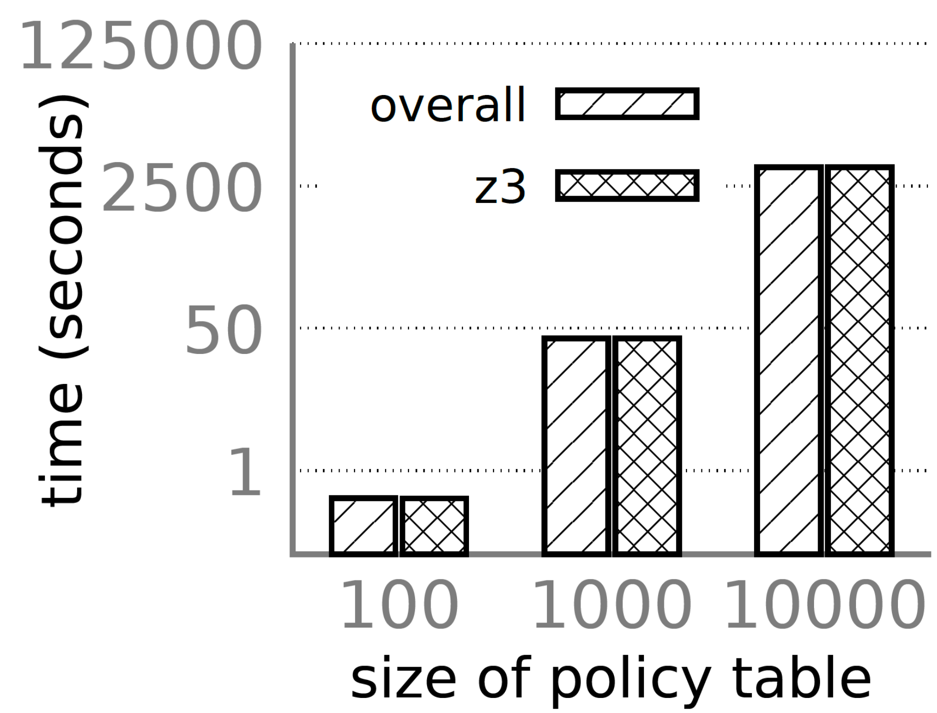
Realistic destination

⋈: $P_1 \bowtie P_3$

P_1: static-route & filter policy

P_3: traffic balance policy

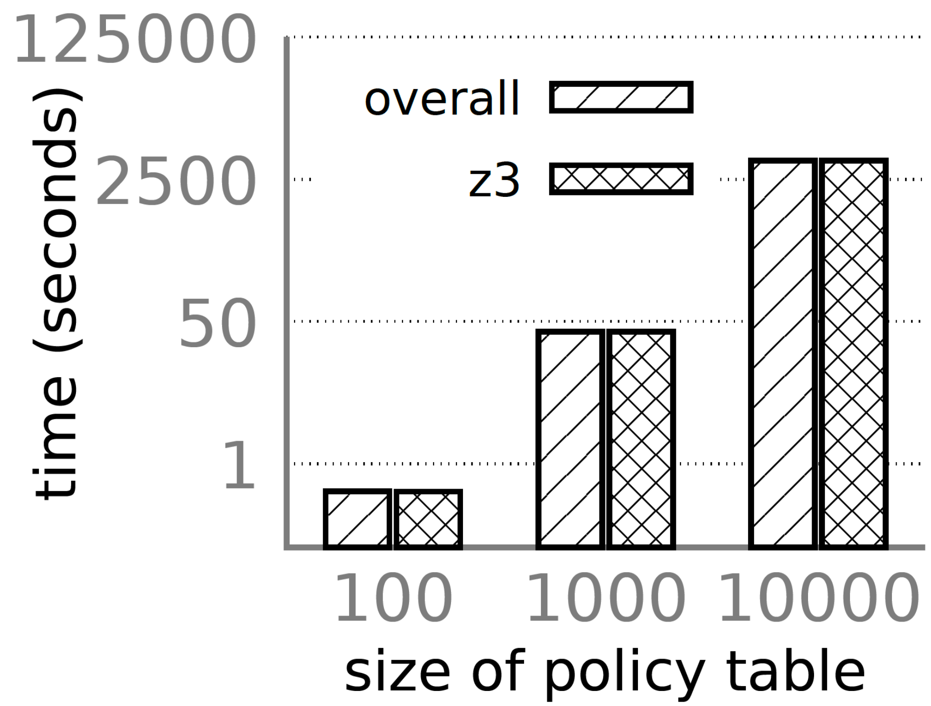
Performance - Scalability



Scalability property of ⋈

- SMT(Z3) takes 96% of the running time
- Our implementation can handle 10,000 policies in ≤ 70 minutes

Performance - Scalability

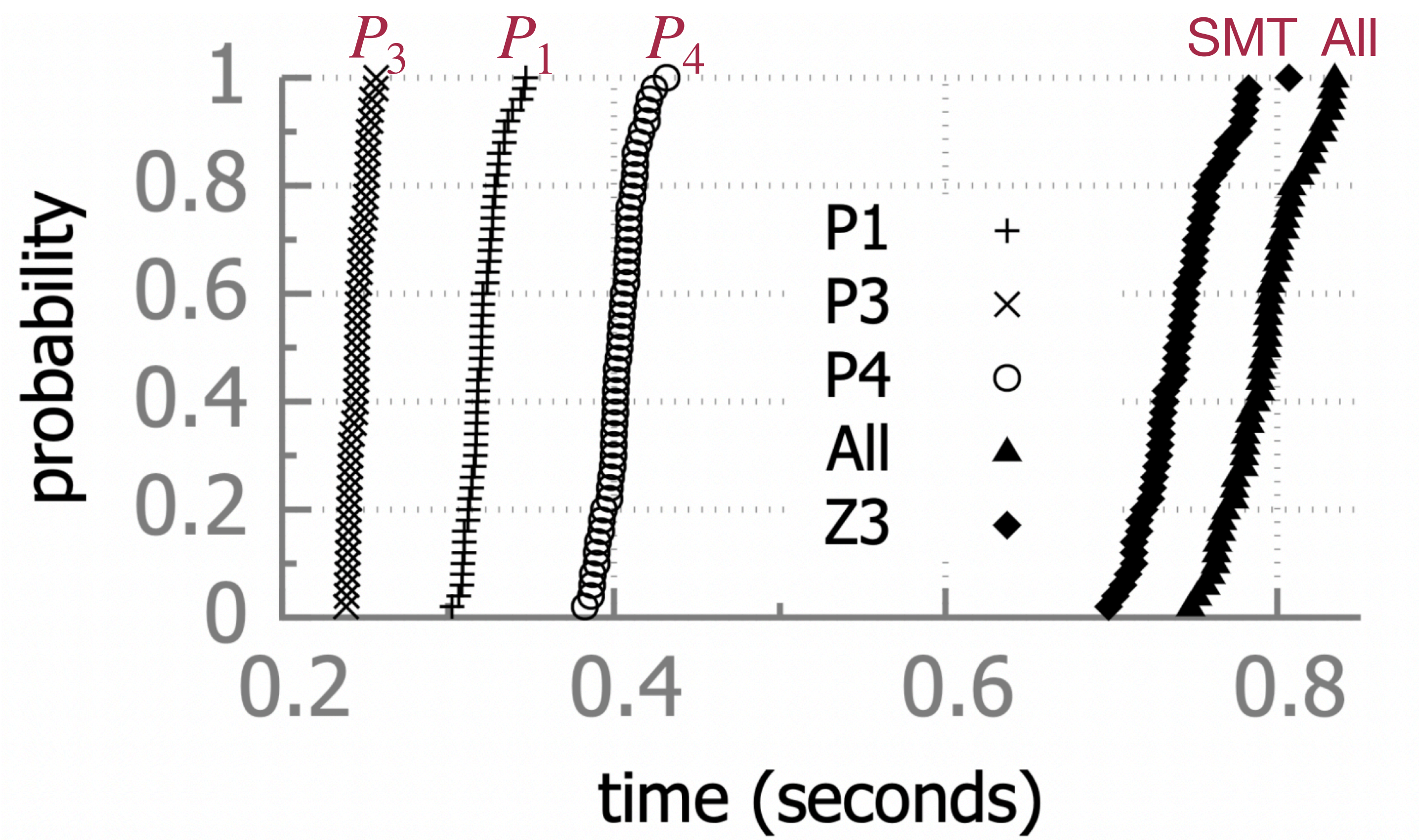


Scalability property of ⋈

Apply ⋈ on three policy sizes

- SMT(Z3) takes 96% of the running time
- Our implementation can handle 10,000 policies in ≤ 70 minutes

Performance - More Policy Examples

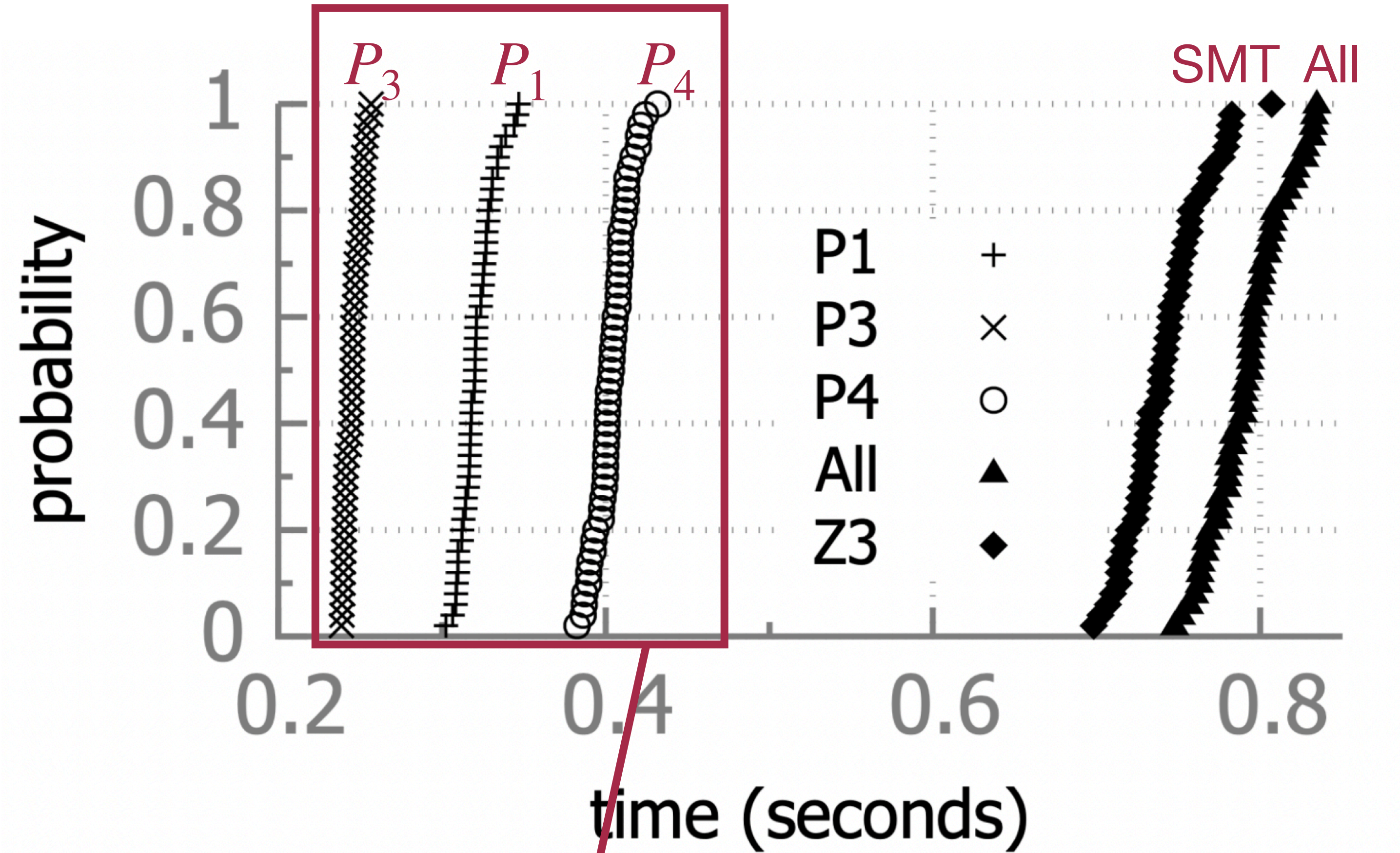


P_1: static route policy
P_3: traffic balance policy
P_4: filter policy
All: total time
SMT: reasoning time

Processing time breakdown of poss(s, p)

- Primitive that validates a relational policy (p) on a given network state (s)

Performance - More Policy Examples

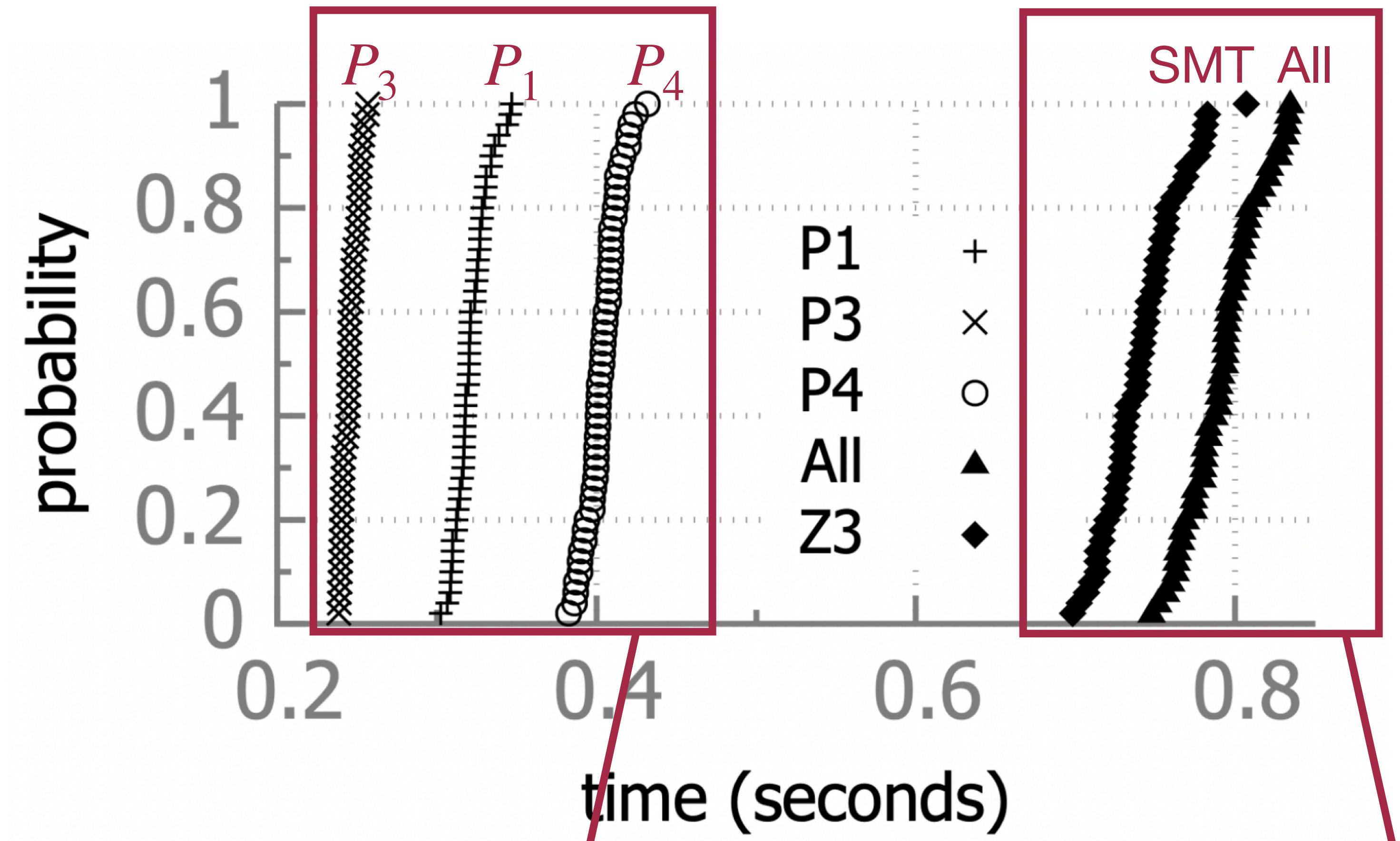


Processing time breakdown of poss(s, p)

P_1: static route policy
P_3: traffic balance policy
P_4: filter policy
All: total time
SMT: reasoning time

- Primitive that validates a relational policy (p) on a given network state (s)
- Running time: P3 < P1 < P4

Performance - More Policy Examples



Processing time breakdown of poss(s, p)

P_1: static route policy
P_3: traffic balance policy
P_4: filter policy
All: total time
SMT: reasoning time

- Primitive that validates a relational policy (p) on a given network state (s)

- Running time: P3 < P1 < P4
- SMT dominates the source of delay

Related Work

- Inter-domain Routing Protocols and Architectures
  - D-BGP and Trotsky: partial deployments of protocols, requires their co-existence on the global internet

Related Work

- Inter-domain Routing Protocols and Architectures
  - D-BGP and Trotsky: partial deployments of protocols, requires their co-existence on the global internet
- Declarative Networking:
  - We share database usage in networking
  - We introduce and implement a novel use of conditional tables

Thank you

http://ravel-net.org/

Other Relational Operations

- Support full set of relational operators

Union: $\cup$

Selection: σ

Difference: $-$

Projection: π

Rename: ρ

Other Relational Operations

- Support full set of relational operators

Union: $\cup$

e.g. Assign a new path [AEC] to 1.2.3.4

Selection: σ

Difference: $-$

Projection: π

Rename: ρ

Other Relational Operations

- Support full set of relational operators

Union: $\cup$

e.g. Assign a new path [AEC] to 1.2.3.4

Selection: σ

e.g. Select which path=[ABC] from P_R

Difference: $-$

Projection: π

Rename: ρ

Other Relational Operations

- Support full set of relational operators

Union: $\cup$

e.g. Assign a new path [AEC] to 1.2.3.4

Selection: σ

e.g. Select which path=[ABC] from P_R

Difference: $-$

e.g. Difference between two policies

Projection: π

Rename: ρ

Other Relational Operations

- Support full set of relational operators

Union: ∪

e.g. Assign a new path [AEC] to 1.2.3.4

Selection: σ

e.g. Select which path=[ABC] from P_R

Difference: $-$

e.g. Difference between two policies

Projection: π

e.g. Select attribute dest from P_R

Rename: ρ

Other Relational Operations

- Support full set of relational operators

Union: $\cup$ — e.g. Assign a new path [AEC] to 1.2.3.4

Selection: σ — e.g. Select which path=[ABC] from P_R

Difference: $-$ — e.g. Difference between two policies

Projection: π — e.g. Select attribute dest from P_R

Rename: ρ — e.g. Rename column dest to prefix

Combine Policies using Join Operation

- Static route and filter policy
- Traffic balance policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5∧y≠1.2.3.4

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Can we generate a new policy that satisfies P_1 and P_3 simultaneously?

P_1 join P_3

$P_1 \bowtie P_3$	dest	path	flag	
	1.2.3.4	x	u	x=[ABC] ∧ u=1
	1.2.3.4	x	v	x=[ABC] ∧ x=[ADC] ∧ v=1
	y	z	u	y≠1.2.3.5∧y≠1.2.3.4∧y=1.2.3.4∧z=[ABC]∧u=1
	y	z	u	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ABC]∧u≠1
	y	z	v	y≠1.2.3.5∧y≠1.2.3.4∧y=1.2.3.4∧z=[ADC]∧v=1
	y	z	v	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ADC]∧v≠1

Combine Policies using Join Operation

- Static route and filter policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5∧y≠1.2.3.4

- Traffic balance policy

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Can we generate a new policy that satisfies P_1 and P_3 simultaneously?

P_1 join P_3

$P_1 \bowtie P_3$	dest	path	flag	
	1.2.3.4	x	u	x=[ABC] ∧ u=1
	1.2.3.4	x	v	x=[ABC] ∧ x=[ADC] ∧ v=1
	y	z	u	y≠1.2.3.5∧y≠1.2.3.4∧y=1.2.3.4∧z=[ABC]∧u=1
	y	z	u	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ABC]∧u≠1
	y	z	v	y≠1.2.3.5∧y≠1.2.3.4∧y=1.2.3.4∧z=[ADC]∧v=1
	y	z	v	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ADC]∧v≠1

Combine Policies using Join Operation

- Static route and filter policy
- Traffic balance policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5∧y≠1.2.3.4

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Can we generate a new policy that satisfies P_1 and P_3 simultaneously?

P_1 join P_3

$P_1 \bowtie P_3$	dest	path	flag	
	1.2.3.4	x	u	x=[ABC] ∧ u=1
	1.2.3.4	x	v	x=[ABC] ∧ x=[ADC] ∧ v=1
	y	z	u	y≠1.2.3.5∧y≠1.2.3.4∧y=1.2.3.4∧z=[ABC]∧u=1
	y	z	u	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ABC]∧u≠1
	y	z	v	y≠1.2.3.5∧y≠1.2.3.4∧y=1.2.3.4∧z=[ADC]∧v=1
	y	z	v	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ADC]∧v≠1

Combine Policies using Join Operation

- Static route and filter policy
- Traffic balance policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5∧y≠1.2.3.4

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Can we generate a new policy that satisfies P_1 and P_3 simultaneously?

P_1 join P_3

Contradictory

$P_1 \bowtie P_3$	dest	path	flag	
	1.2.3.4	x	u	x=[ABC] ∧ u=1
	1.2.3.4	x	v	x=[ABC] ∧ x=[ADC] ∧ v=1
	y	z	u	y≠1.2.3.5∧y≠1.2.3.4∧y=1.2.3.4∧z=[ABC]∧u=1
	y	z	u	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ABC]∧u≠1
	y	z	v	y≠1.2.3.5∧y≠1.2.3.4∧y=1.2.3.4∧z=[ADC]∧v=1
	y	z	v	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ADC]∧v≠1

Combine Policies using Join Operation

- Static route and filter policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5∧y≠1.2.3.4

- Traffic balance policy

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Can we generate a new policy that satisfies P_1 and P_3 simultaneously?

P_1 join P_3

Contradictory

$P_1 \bowtie P_3$	dest	path	flag	
	1.2.3.4	x	u	x=[ABC] ∧ u=1
	~~1.2.3.4~~	~~x~~	~~v~~	~~x=[ABC] ∧ x=[ADC] ∧ v=1~~
	~~y~~	~~z~~	~~u~~	~~y≠1.2.3.5∧y≠1.2.3.4∧y=1.2.3.4∧z=[ABC]∧u=1~~
	y	z	u	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ABC]∧u≠1
	~~y~~	~~z~~	~~v~~	~~y≠1.2.3.5∧y≠1.2.3.4∧y=1.2.3.4∧z=[ADC]∧v=1~~
	y	z	v	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ADC]∧v≠1

Combine Policies using Join Operation

- Static route and filter policy
- Traffic balance policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5∧y≠1.2.3.4

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Can we generate a new policy that satisfies P_1 and P_3 simultaneously?

P_1 join P_3

$P_1 \bowtie P_3$	dest	path	flag	
	1.2.3.4	x	u	x=[ABC] ∧ u=1
	y	z	u	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ABC]∧u≠1
	y	z	v	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ADC]∧v≠1

Combine Policies using Join Operation

- Static route and filter policy
- Traffic balance policy

P_1	dest	path	
	1.2.3.4	x	x=[ABC]
	y	z	y≠1.2.3.5∧y≠1.2.3.4

P_3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u ≠ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v ≠ 1

Can we generate a new policy that satisfies P_1 and P_3 simultaneously?

P_1 join P_3

$P_1 \bowtie P_3$	dest	path	flag	
	1.2.3.4	x	u	x=[ABC] ∧ u=1
	y	z	u	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ABC]∧u≠1
	y	z	v	y≠1.2.3.5∧y≠1.2.3.4∧y=5.6.7.8∧z=[ADC]∧v≠1

Redundancy