DRAGON: Distributed Route AGgregatiON

Joint work with: João Luís Sobrinoh, Franck Le and Jennifer Rexford

Laurent Vanbever

Princeton University

ETH Zürich

March, 17 2014

Scalable routing systems maintain

- detailed information about nearby destination
- coarse-grained information about far-away destination

BGP maintains detailed information about every destination (*i.e.*, network)

Sign Post Forest, Watson Lake, Yukon

The problem is that the number of devices connected to the Internet increases rapidly

mobile

Internet of things

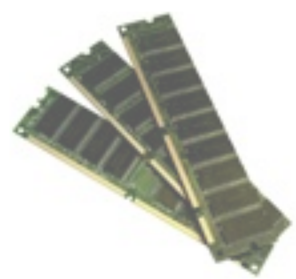
sensors

virtual machines

BGP routers must also maintain routes for
IPv6 networks in addition of IPv4 networks

IPv6 ramping up could easily double
the size of the Internet routing table

The growth of the number of destinations has serious consequences for the Internet

memory  routing and forwarding table size

time  convergence time after a failure

boot time for a router, session, …

security  cost of signing & verifying BGP route

DRAGON: Distributed Route AGgregatiON

1 **Background**
Route aggregation 101

2 **Distributed filtering**
preserving consistency

3 **Performance**
up to 80% of filtering efficiency

DRAGON: Distributed Route AGgregatiON

1 Background

Route aggregation 101

Distributed filtering

preserving consistency

Performance

up to 80% of filtering efficiency

How do you maintain less

routing and/or forwarding information?

You make use of the IP prefix hierarchy to remove redundant information

Routing Table

IP prefix	Output Interface
…	
129.0.0.0/8	IF#2
129.132.1.0/24	IF#2
129.132.2.0/24	IF#2
129.133.0.0/16	IF#3
…	

An IP prefix identifies a set of IP addresses

Routing Table

IP prefix	Output Interface
...	
129.0.0.0/8	IF#2
129.132.1.0/24	IF#2
129.132.2.0/24	IF#2
129.133.0.0/16	IF#3
...	

prefix length

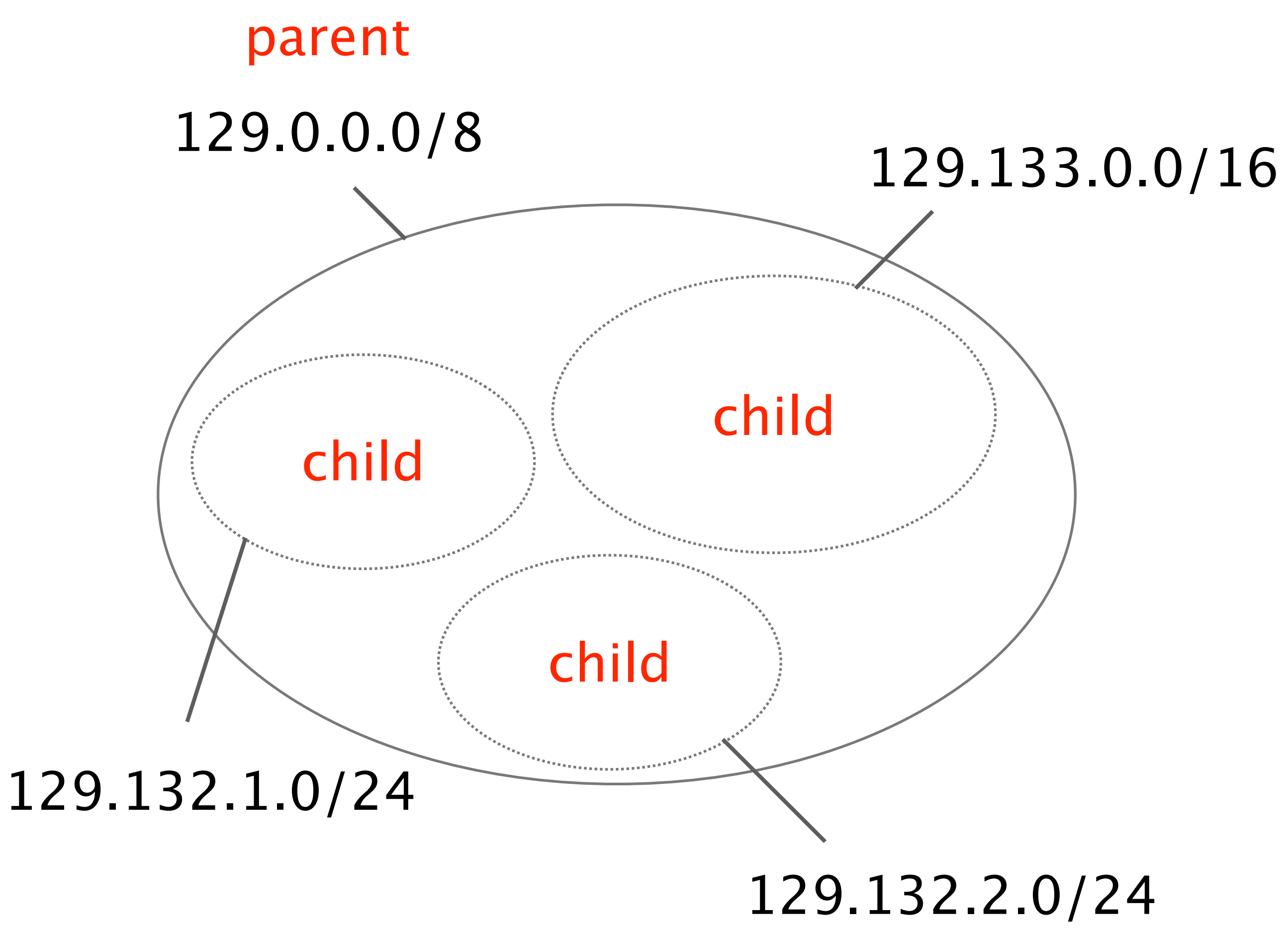
129.0.0.0/8

$2^{(32-8)}$ IP addresses

An IP prefix identifies a set of IP addresses
which can be included into another one

Routing Table

IP prefix	Output Interface
…	
129.0.0.0/8	IF#2
129.132.1.0/24	IF#2
129.132.2.0/24	IF#2
129.133.0.0/16	IF#3
…	

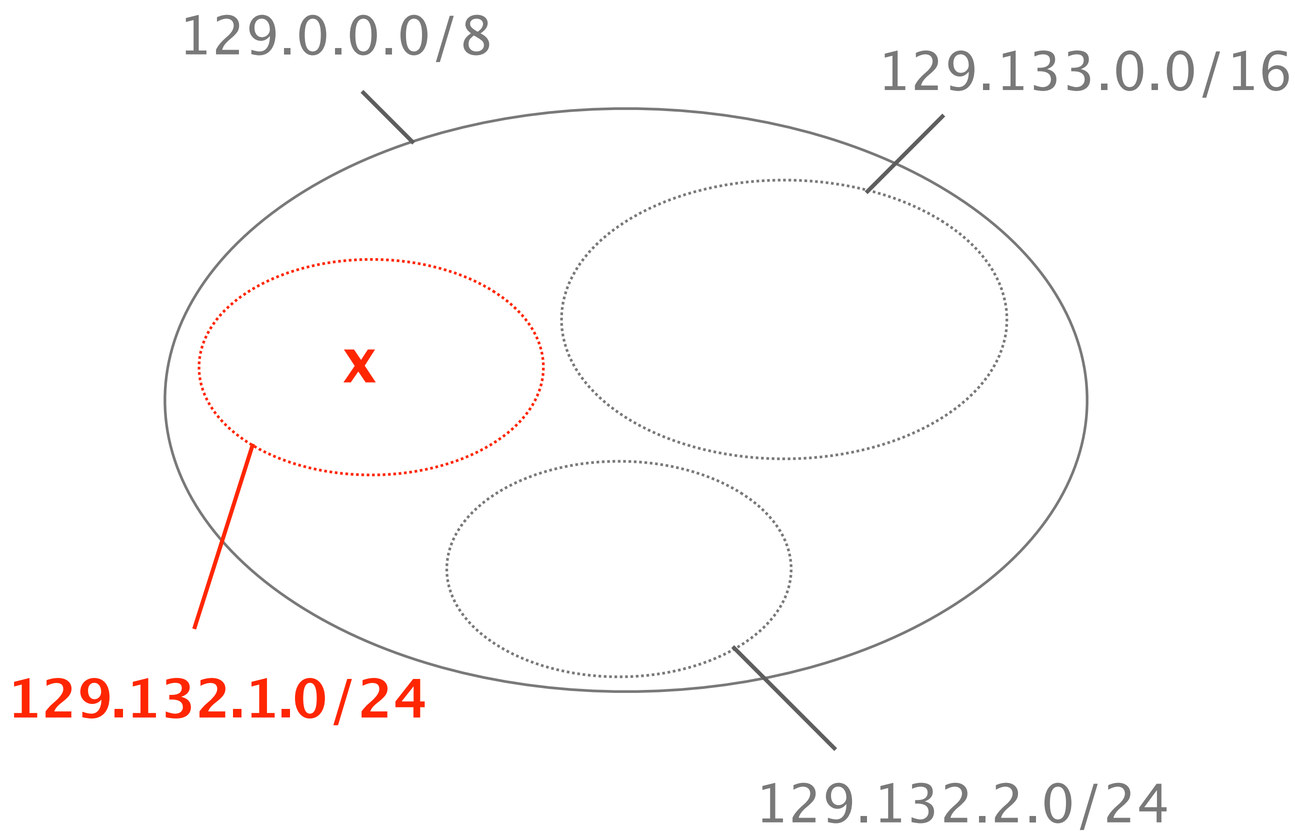


Forwarding is done along the most specific prefix, *i.e.*, the smallest set containing the IP address

Routing Table

IP prefix	Output Interface
…	
129.0.0.0/8	IF#2
129.132.1.0/24	**IF#2**
129.132.2.0/24	IF#2
129.133.0.0/16	IF#3
…	

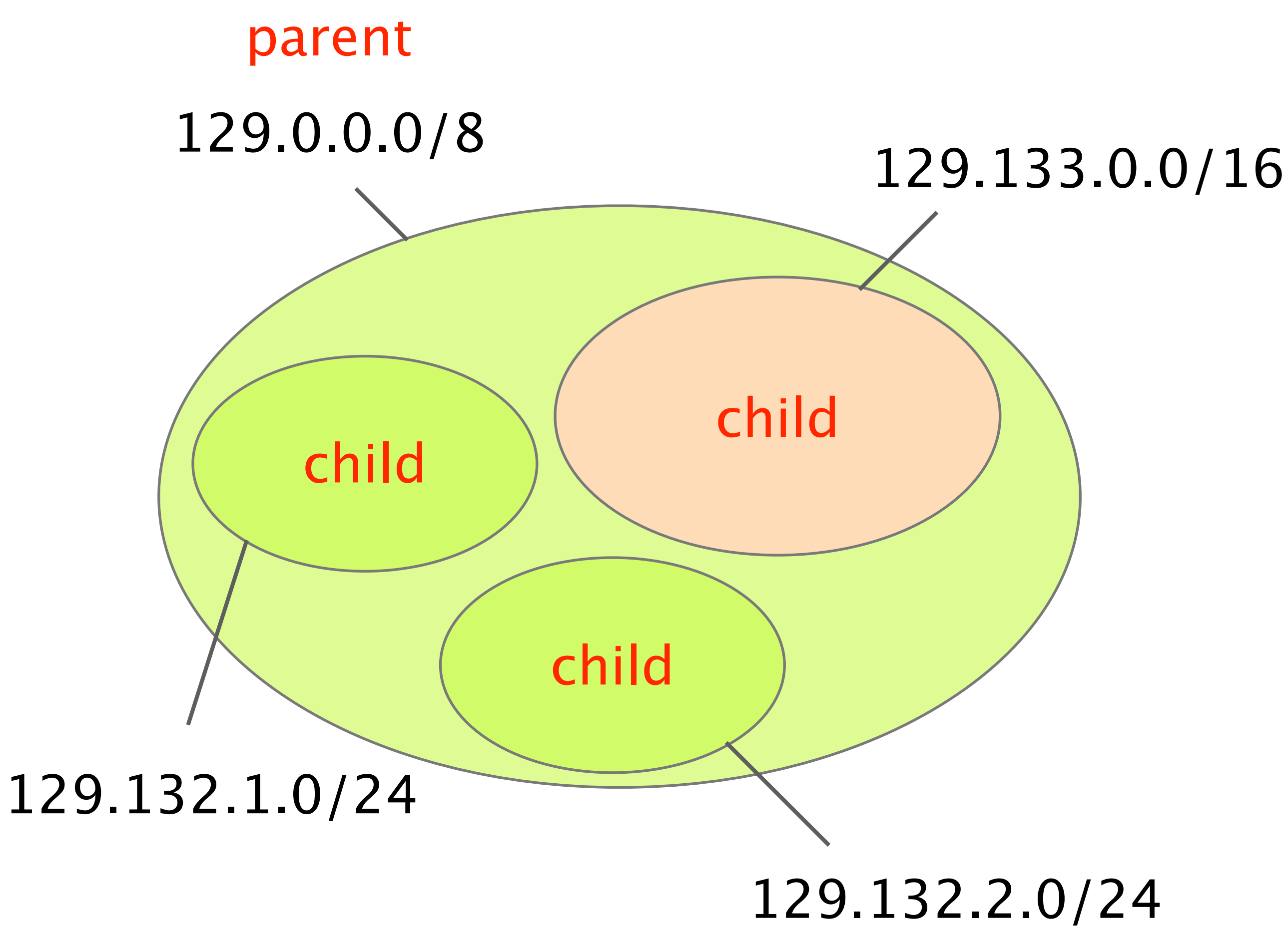
Input packet: 129.132.1.1



A child prefix can be filtered whenever it shares the same output interface as its parent

Routing Table

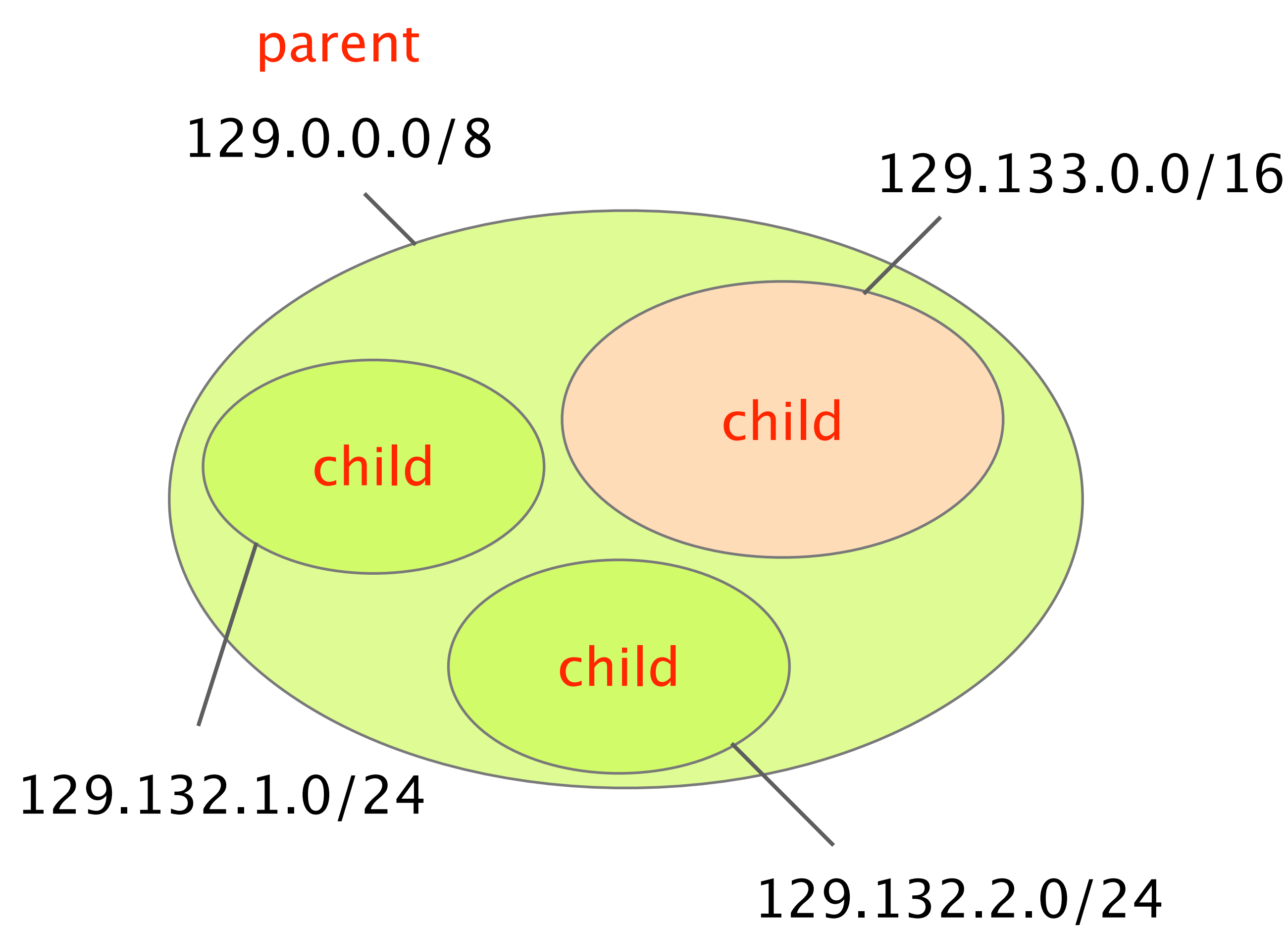
IP prefix	Output Interface
...	
129.0.0.0/8	IF#2
129.132.1.0/24	IF#2
129.132.2.0/24	IF#2
129.133.0.0/16	IF#3
...	



A child prefix can be filtered whenever it shares the same output interface as its parent

Routing Table

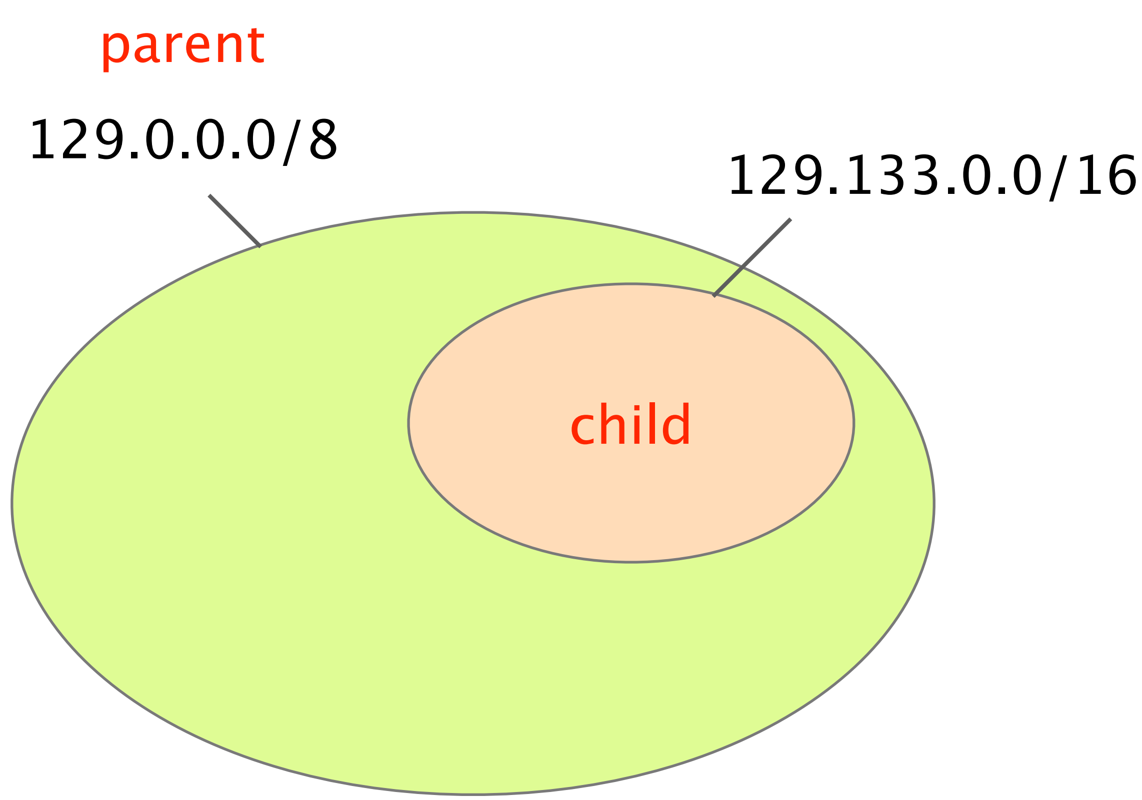
IP prefix	Output Interface
…	
129.0.0.0/8	IF#2
~~129.132.1.0/24~~	~~IF#2~~
~~129.132.2.0/24~~	~~IF#2~~
129.133.0.0/16	IF#3
…	



A child prefix can be filtered whenever it shares the same output interface as its parent

Routing Table

IP prefix	Output Interface
...	
129.0.0.0/8	IF#2
129.133.0.0/16	IF#3
...	



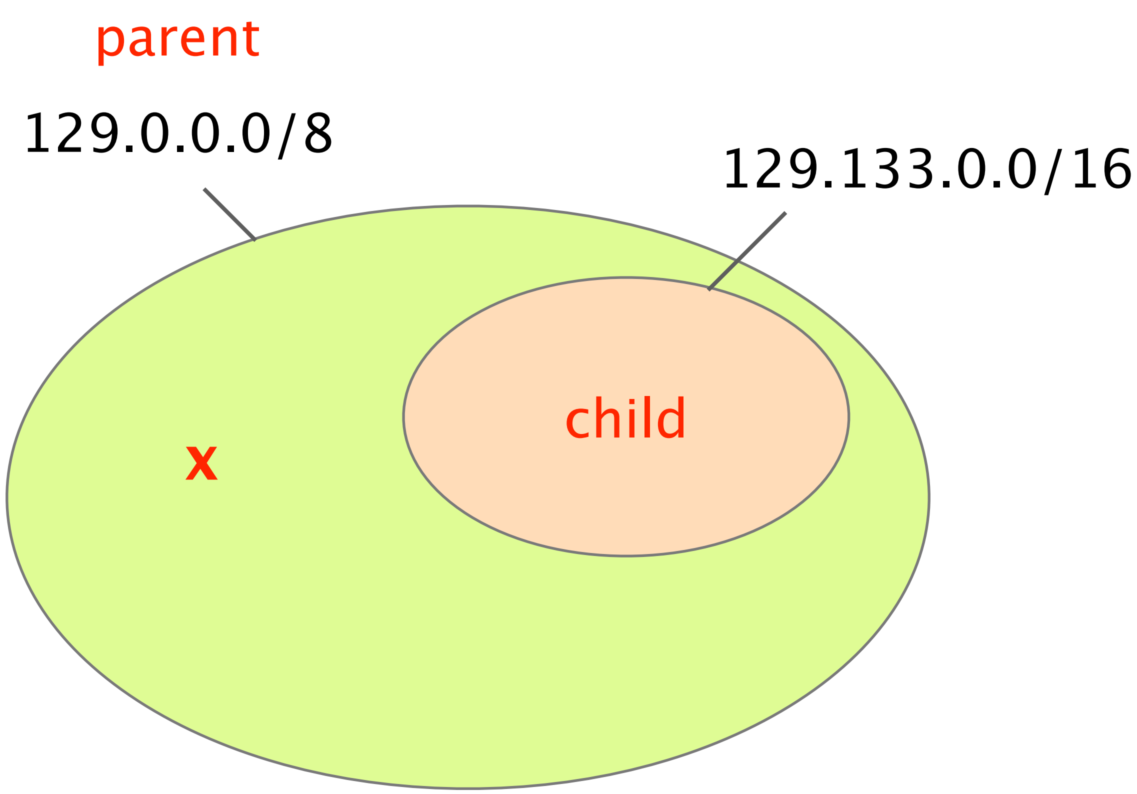
Exactly the same forwarding as before

A child prefix can be filtered whenever it shares the same output interface as its parent

Routing Table

IP prefix	Output Interface
…	
129.0.0.0/8	IF#2
129.133.0.0/16	IF#3
…	

Input packet: 129.132.1.1



Exactly the same forwarding as before

Numerous previous works have studied this problem

2013	(Rétvári, SIGCOMM); (Rottenstreich, INFOCOM)
2012	(Karpilovsky, IEEE TNSM)
2011	(Li, INFOCOM); (Uzmi, CoNEXT)
2010	(Zhao, INFOCOM); (Liu, GLOBECOM)
2009	(Ballani, NDSI)
…	…
1999	(Draves, INFOCOM)

The problem is that they only provide local gain

local gain

router or network

(Rétvári, SIGCOMM); (Rottenstreich, INFOCOM)

(Karpilovsky, IEEE TNSM)

(Li, INFOCOM); (Uzmi, CoNEXT)

(Zhao, INFOCOM); (Liu, GLOBECOM)

(Ballani, NDSI)

…

(Draves, INFOCOM)

Others proposed clean-slate approach to improve scalability, but none of them is incrementally deployable

local gain router or network	(Rétvári, SIGCOMM); (Rottenstreich, INFOCOM) (Karpilovsky, IEEE TNSM) (Li, INFOCOM); (Uzmi, CoNEXT) (Zhao, INFOCOM); (Liu, GLOBECOM) (Ballani, NDSI) … (Draves, INFOCOM)
clean-slate hard to deploy	(Godfrey, SIGCOMM), (Andersen, SIGCOMM) (Subramanian, SIGCOMM)

DRAGON provides both *Internet-wide gain* and *incremental deployability*

existing	**DRAGON**
local gain	global gain
router or network	Internet-wide
clean-slate	works with BGP
hard to deploy	incrementally deployable

DRAGON: Distributed Route AGgregatiON

Background

Route aggregation 101

2 Distributed filtering

preserving consistency

Performance

up to 80% of filtering efficiency

DRAGON is distributed route-aggregation technique where routers "think globally, but act locally"

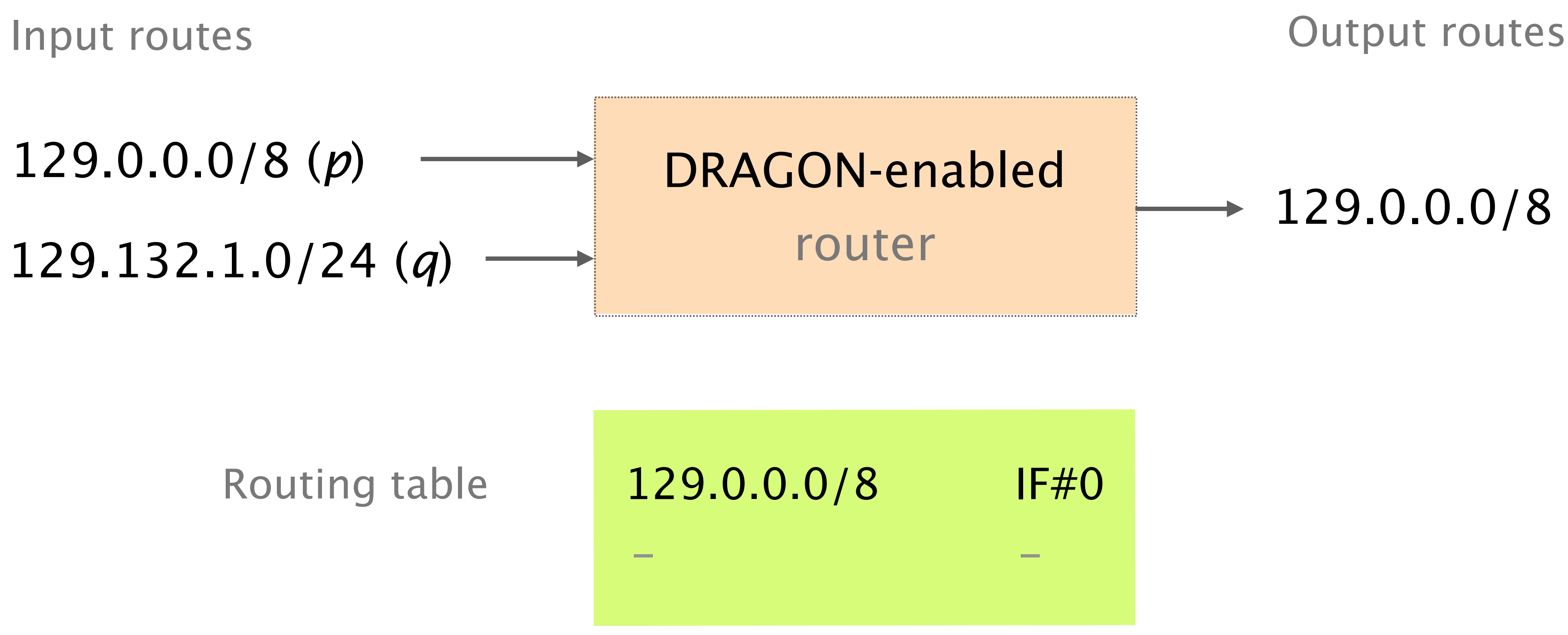
Main result	By comparing routes for different prefixes, a router can locally compute which routes it can filter and not export while preserving routing & forwarding decisions globally

DRAGON is distributed route-aggregation technique where routers “think globally, but act locally”

Main result

By comparing routes for different prefixes, a router can locally compute which routes it can filter and not export while preserving routing & forwarding decisions globally

When a router filters *q*, it does not create any forwarding entry for *q* and does not export *q* to any neighbor



DRAGON is distributed route-aggregation technique where routers "think globally, but act locally"

Main result

By comparing routes for different prefixes, a router can locally compute which routes it can filter and not export while preserving routing & forwarding decisions globally

DRAGON filters routing information,

preserving the flow of data traffic

Somewhere in Belgium…

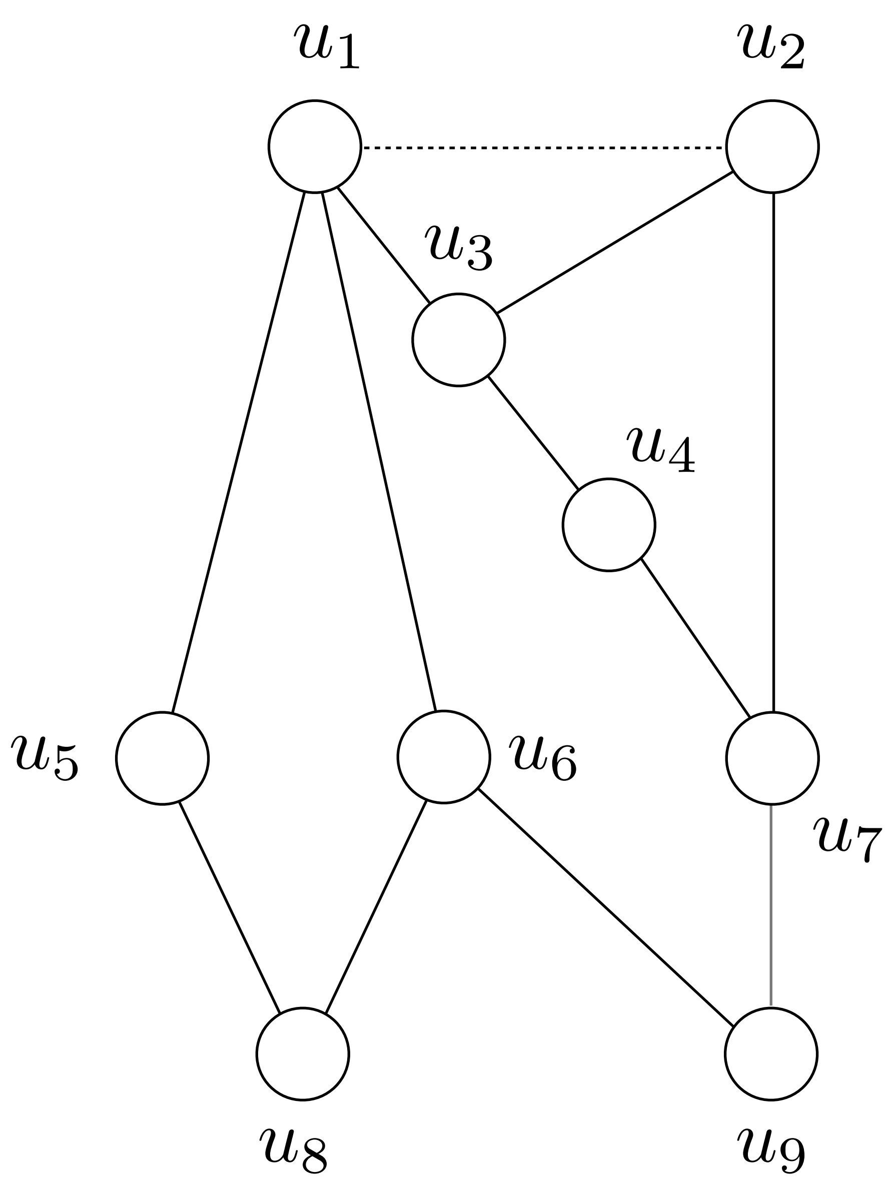
DRAGON guarantees network-wide routing and/or forwarding consistency *post-filtering*

	Routing consistency	Forwarding consistency
preserved property at *every node* for each *data packet*	route attribute	forwarding neighbors

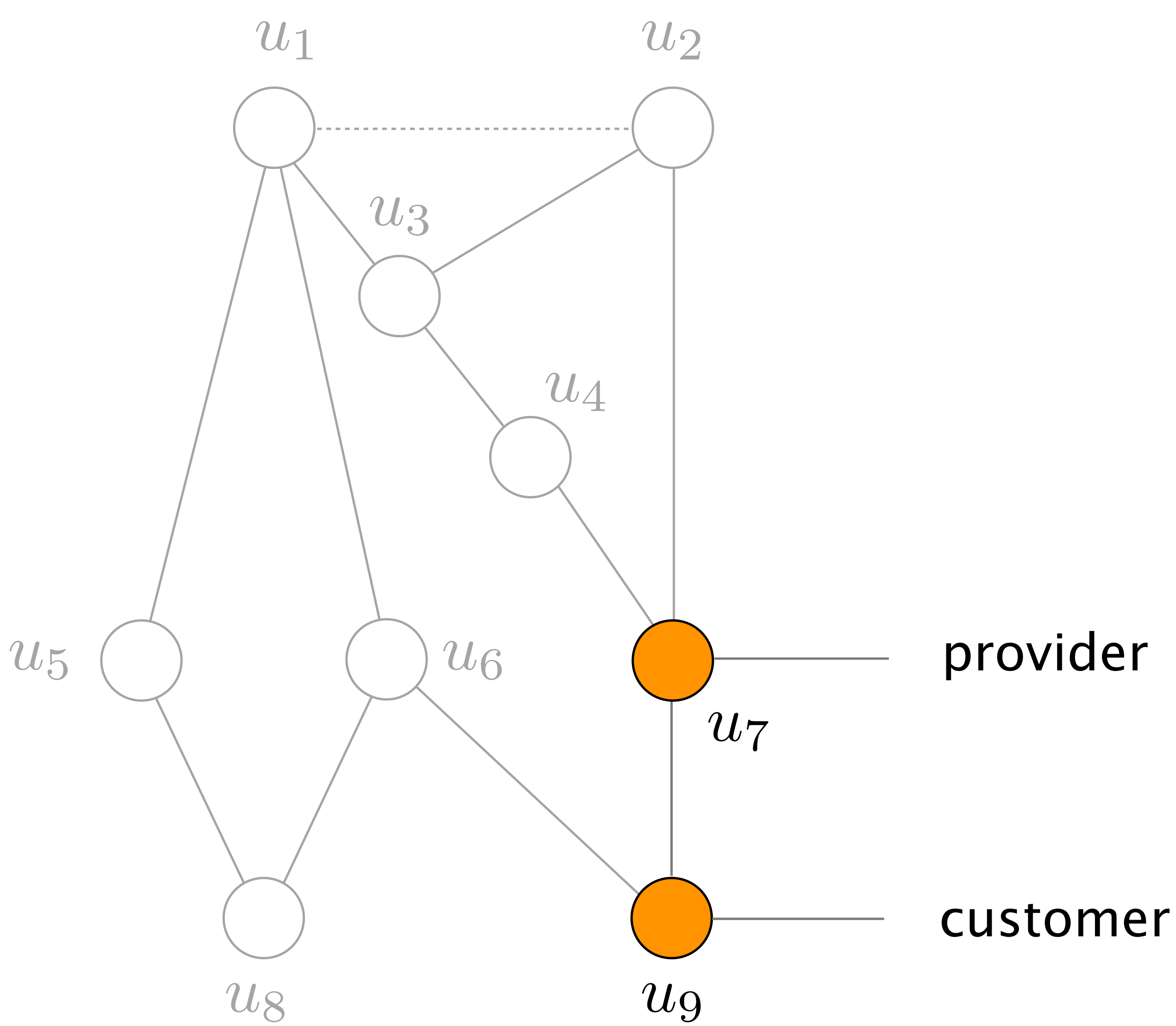
DRAGON guarantees network-wide routing and/or forwarding consistency *post-filtering*

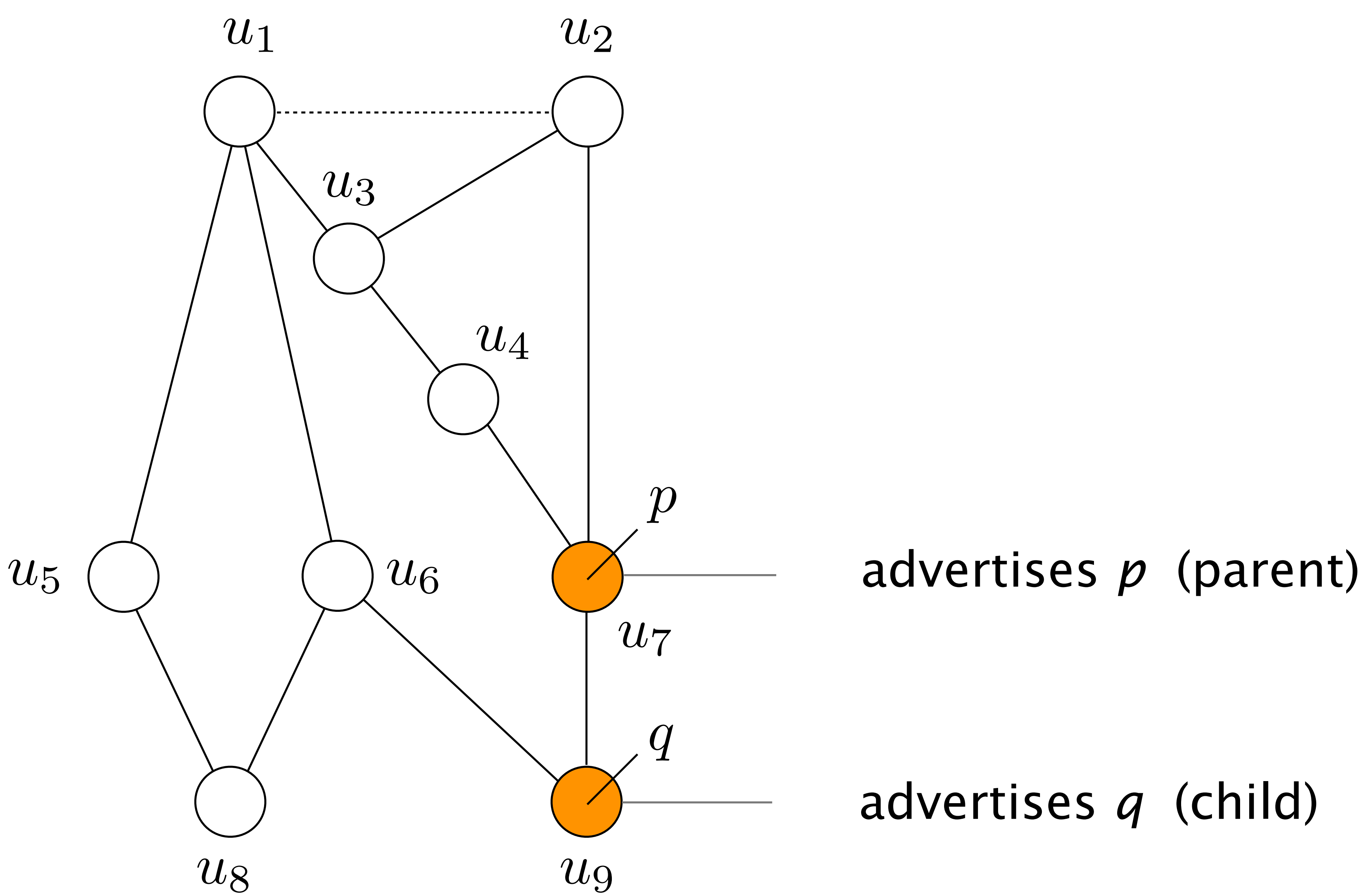
	Routing consistency	Forwarding consistency
preserved property at *every node* for each *data packet*	route attribute	forwarding neighbors
	This talk	

Let's consider a mini-Internet using simplified routing policies

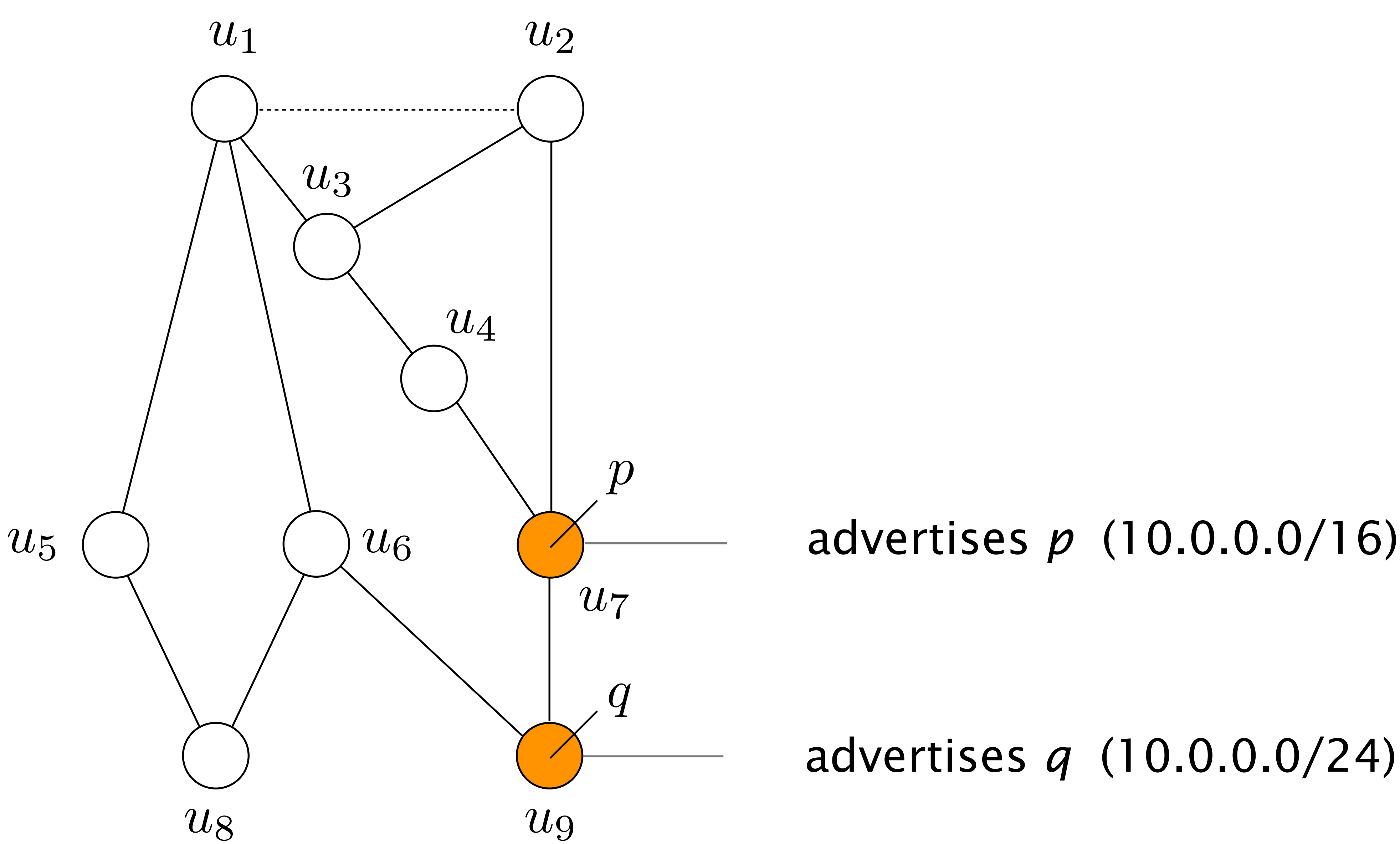


Solid lines join a provider and a customer, with the provider drawn above the customer

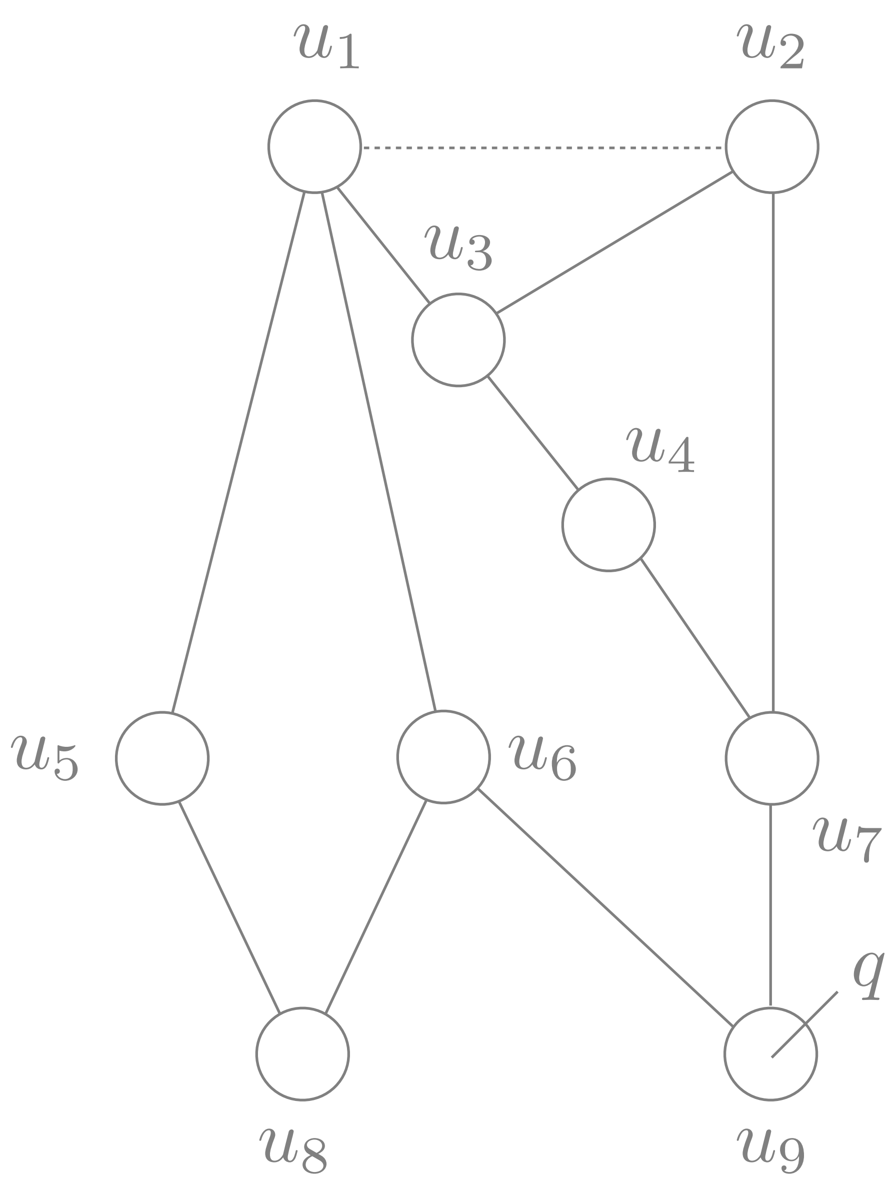




u_1
u_2
u_3
u_4
u_5
u_6
u_7
u_8
u_9
p
q
advertises p (parent)
advertises q (child)



u_1
u_2
u_3
u_4
u_5
u_6
u_7
u_8
u_9
p
q
advertises p (10.0.0.0/16)
advertises q (10.0.0.0/24)



2 route attributes

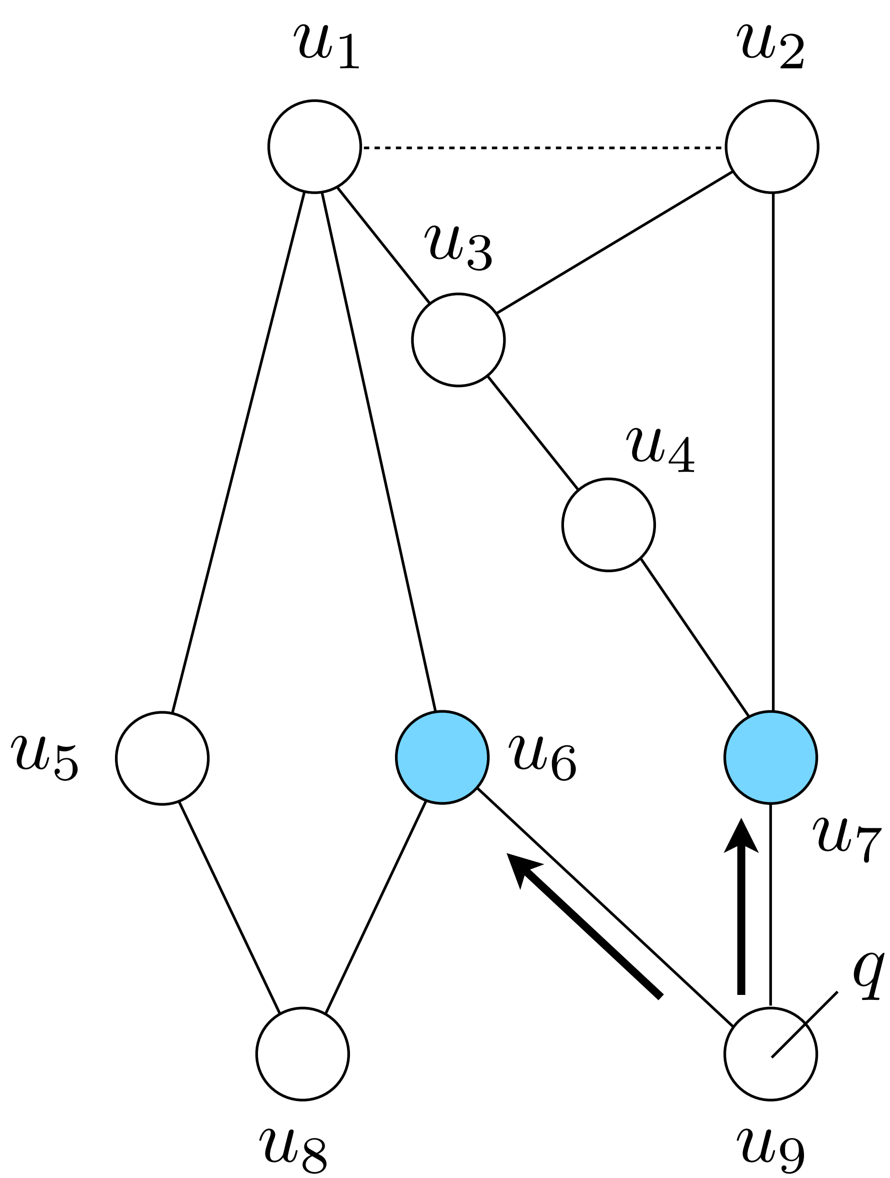
- learned from **customer**
- learned from **provider**

preference

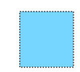
2 exportation rules

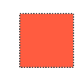
- customer routes to every neighbor
- provider routes to customers

Current routing state for ***q***



2 route attributes

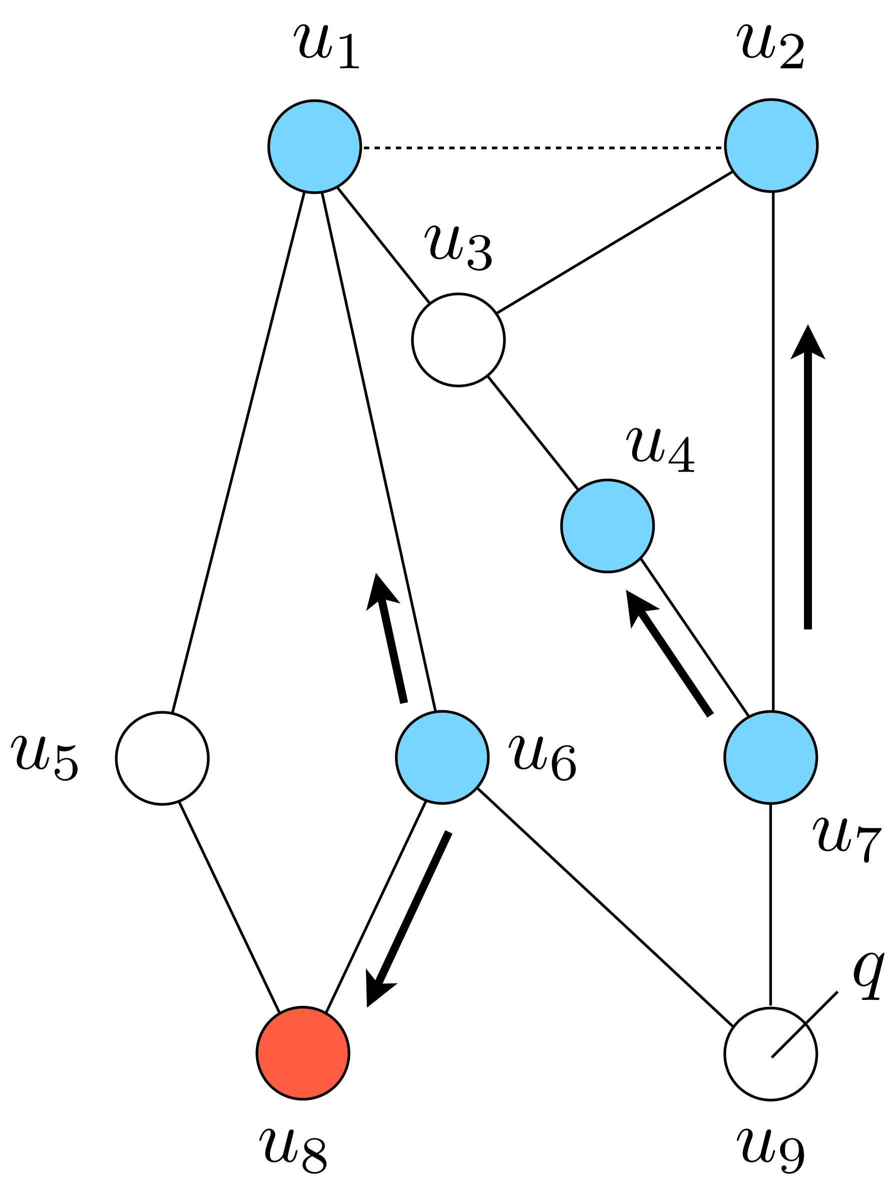

learned from customer


learned from provider

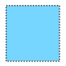
2 exportation rules

- customer routes to every neighbor
- provider routes to customers

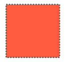
Current routing state for q



2 route attributes



learned from customer

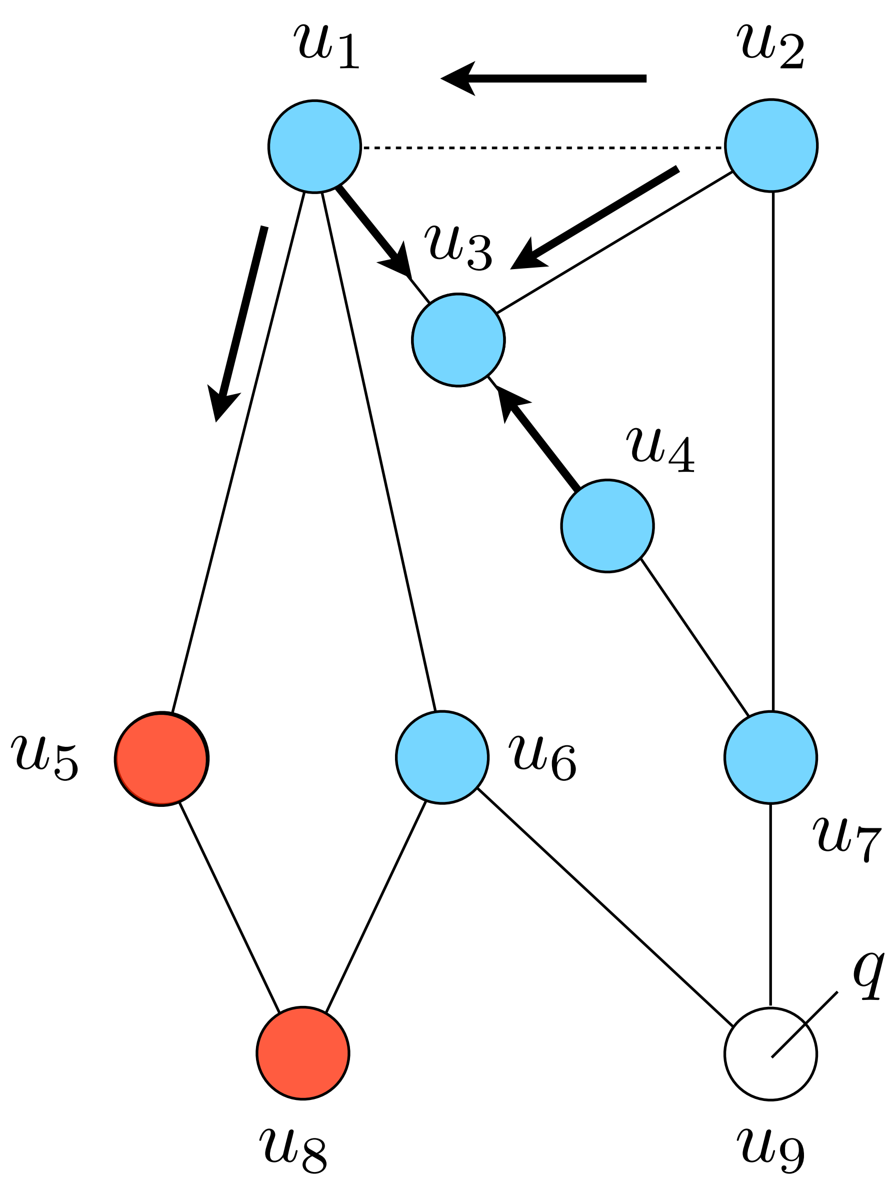


learned from provider

2 exportation rules

- customer routes to every neighbor
- provider routes to customers

Current routing state for q



2 route attributes

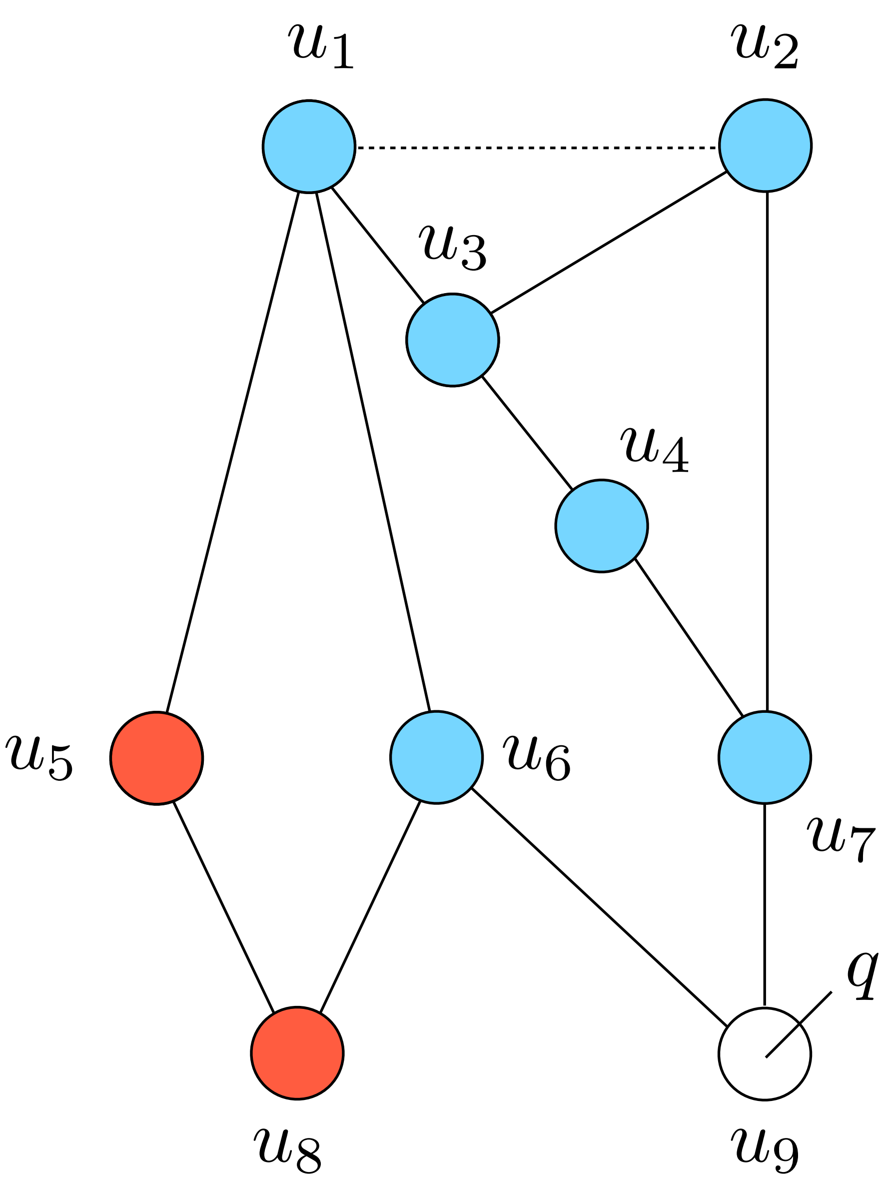
learned from customer

learned from provider

2 exportation rules

- customer routes to every neighbor
- provider routes to customers

Final routing state for *q*



2 route attributes

learned from customer

learned from provider

2 exportation rules

- customer routes to every neighbor
- provider routes to customers

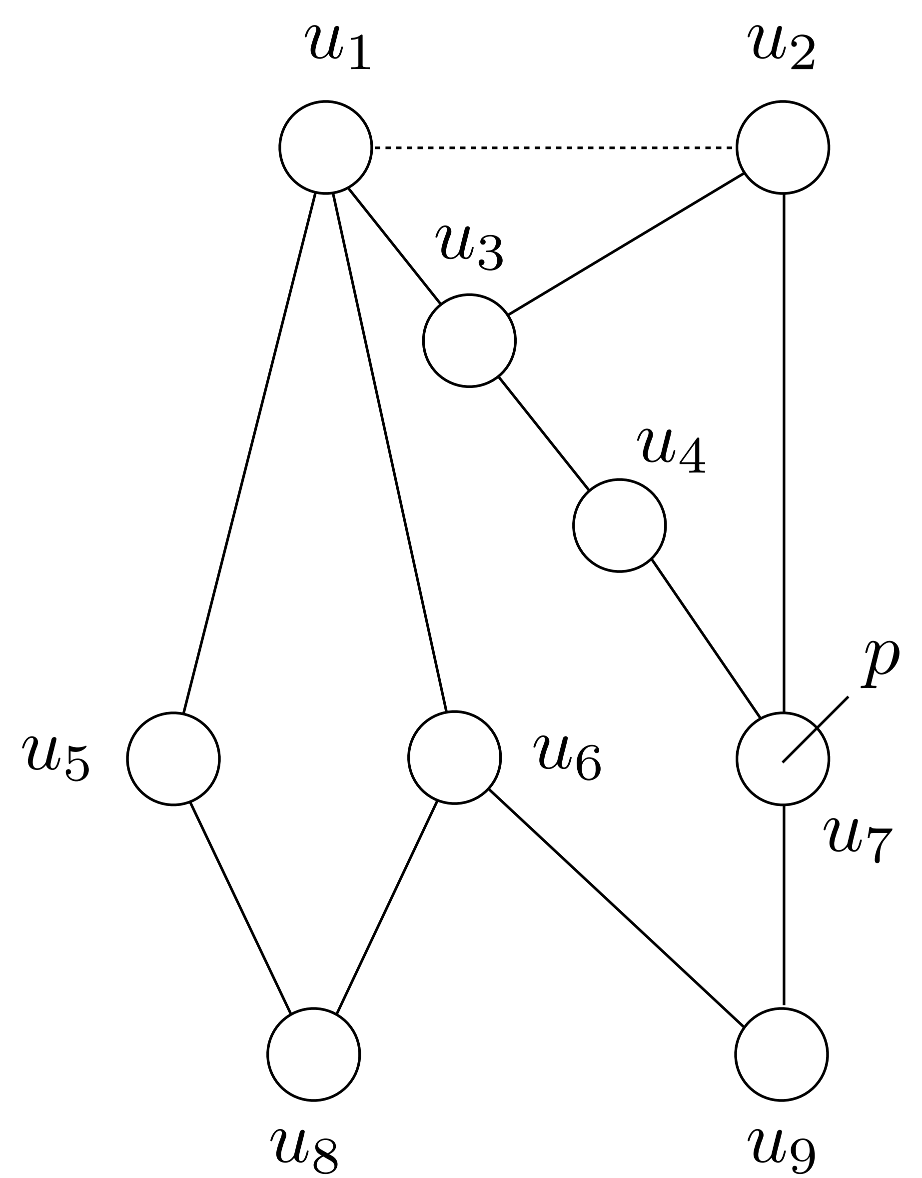
2 route attributes

- learned from customer
- learned from provider

2 exportation rules

- customer routes to every neighbor
- provider routes to customers

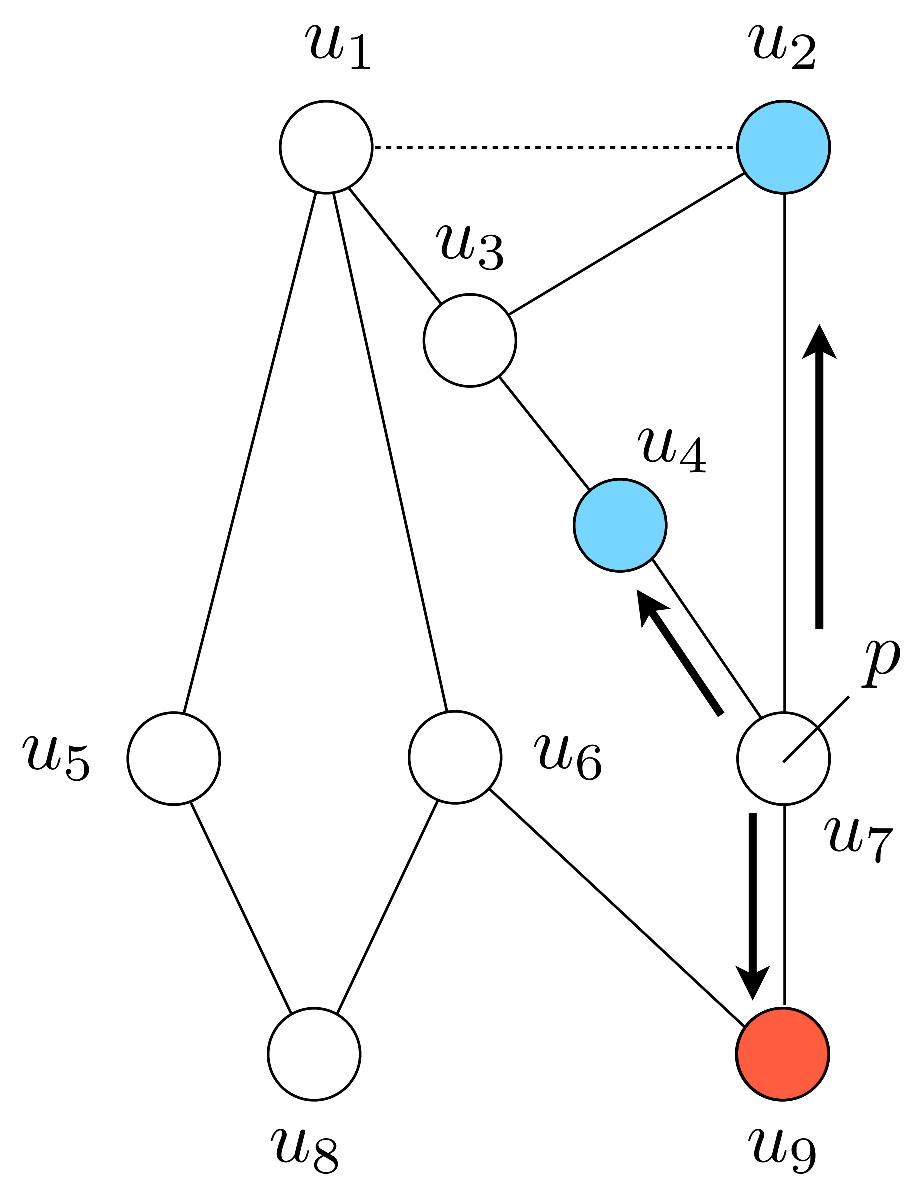
Current routing state for *p*



Current routing state for *p*

2 route attributes

- learned from customer
- learned from provider



2 exportation rules

- customer routes to every neighbor
- provider routes to customers

2 route attributes

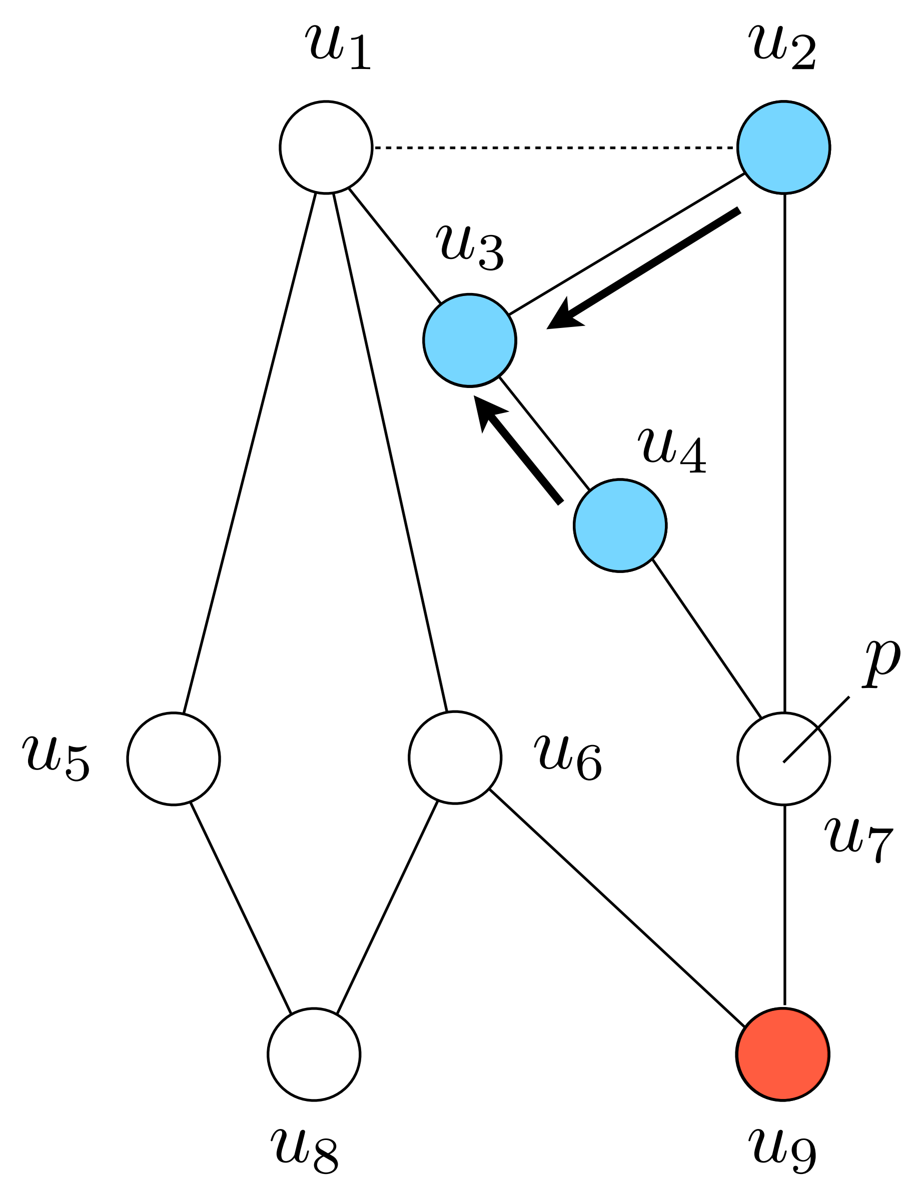
learned from **customer**

learned from **provider**

2 exportation rules

- customer routes to every neighbor
- provider routes to customers

Current routing state for p



2 route attributes

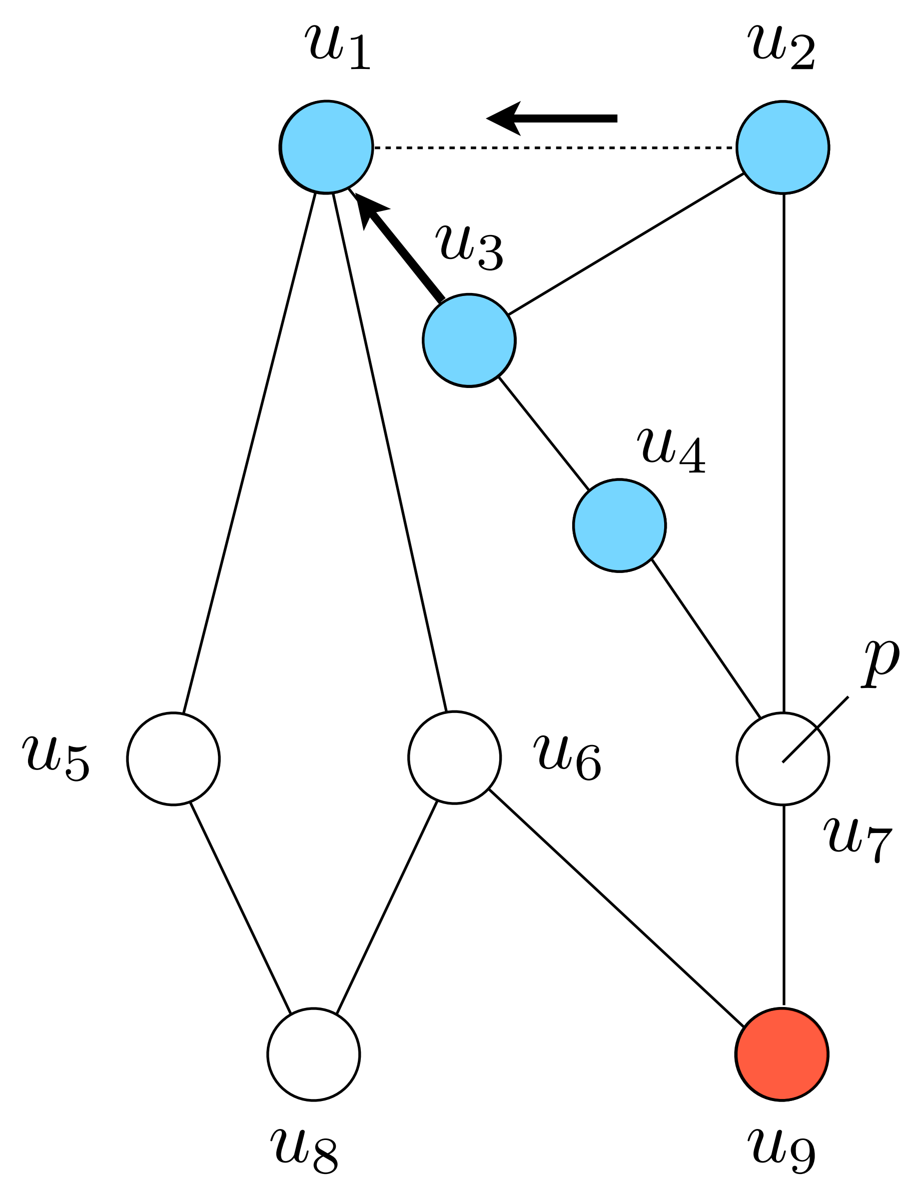
learned from customer

learned from provider

2 exportation rules

- customer routes to every neighbor
- provider routes to customers

Current routing state for p



2 route attributes

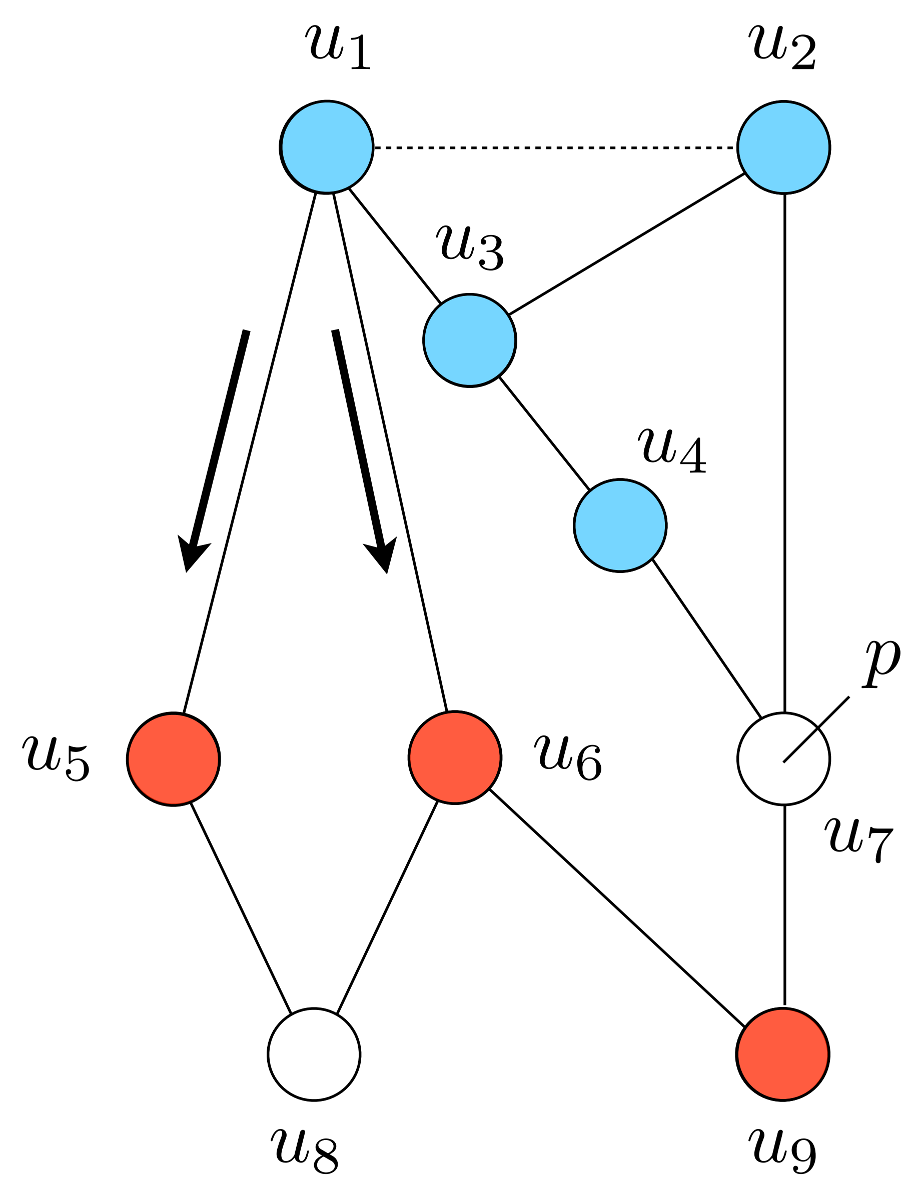
learned from **customer**

learned from **provider**

2 exportation rules

- customer routes to every neighbor
- provider routes to customers

Current routing state for p



2 route attributes

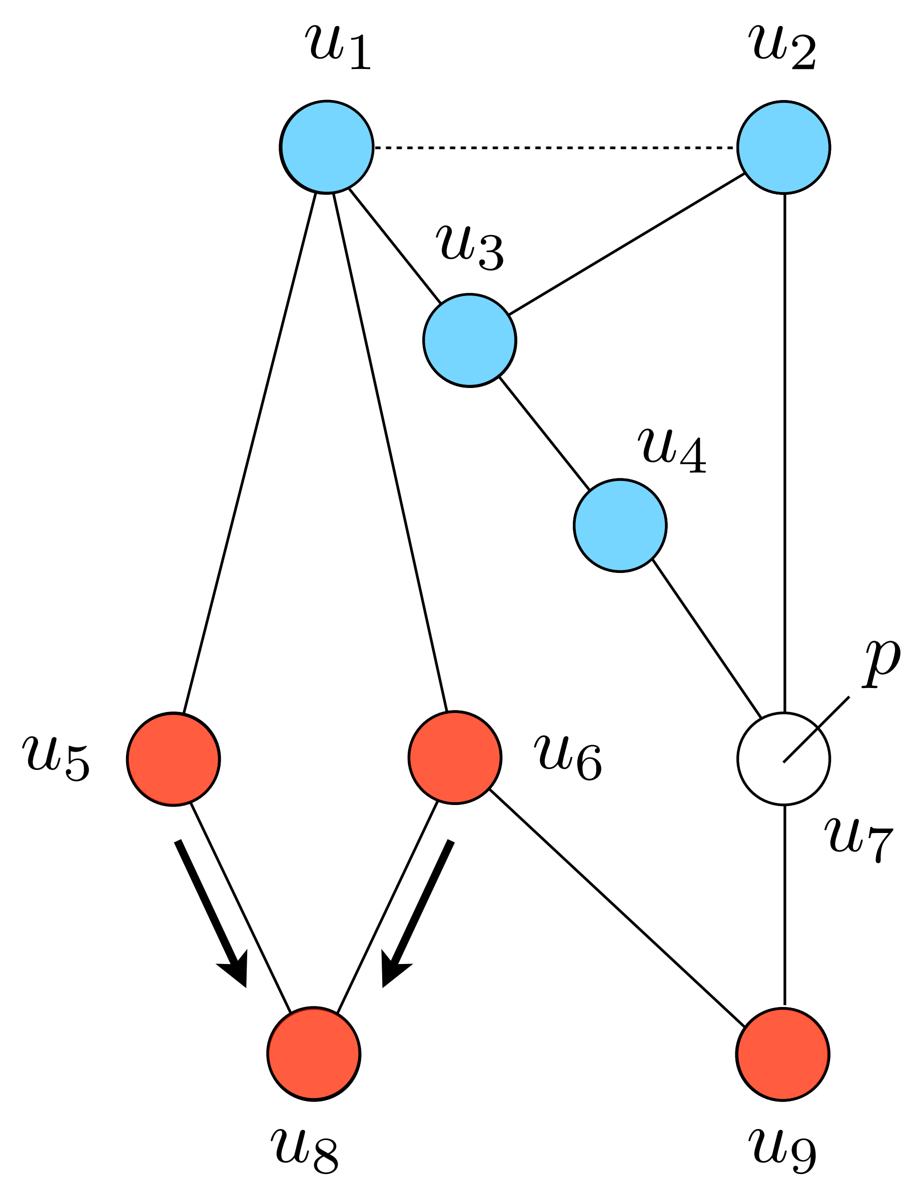
learned from customer

learned from provider

2 exportation rules

- customer routes to every neighbor
- provider routes to customers

Current routing state for *p*



Final routing state for *p*

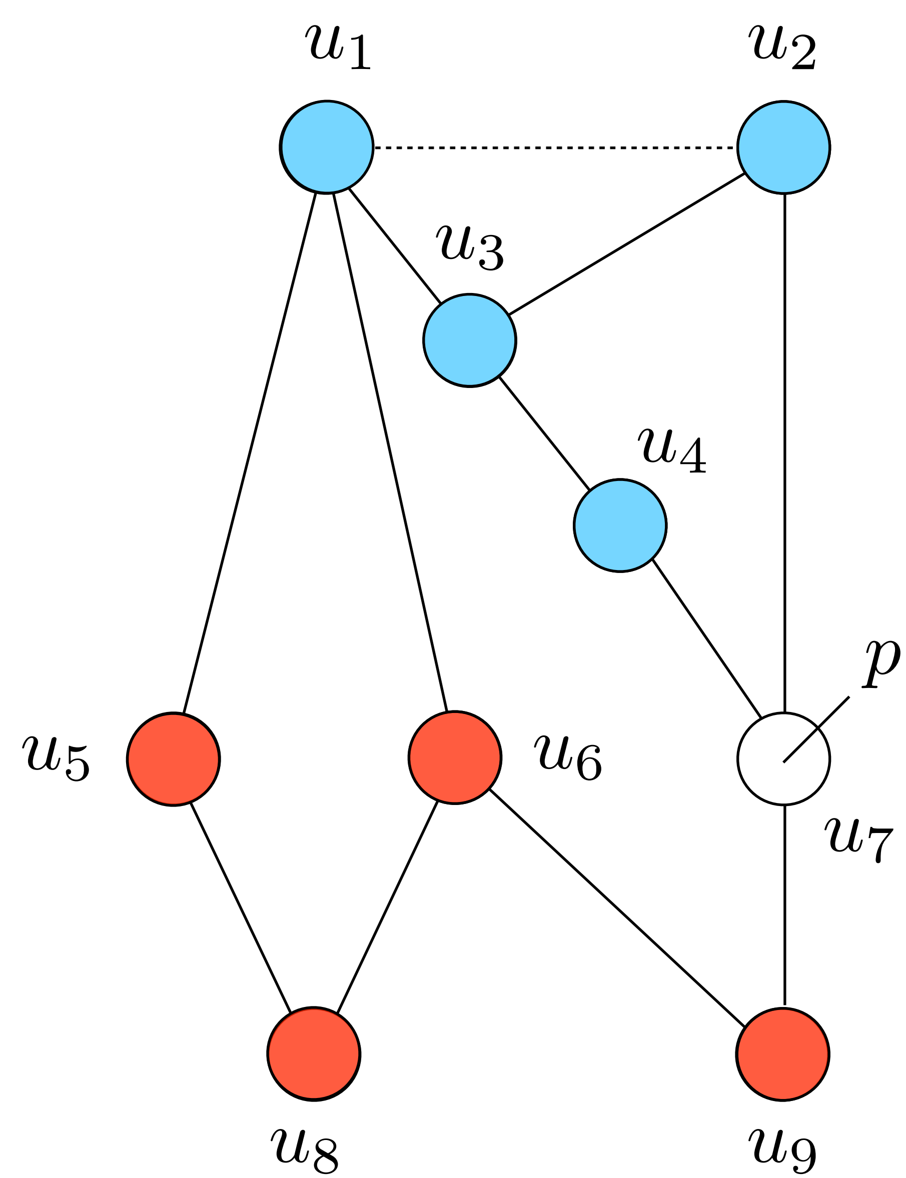
2 route attributes

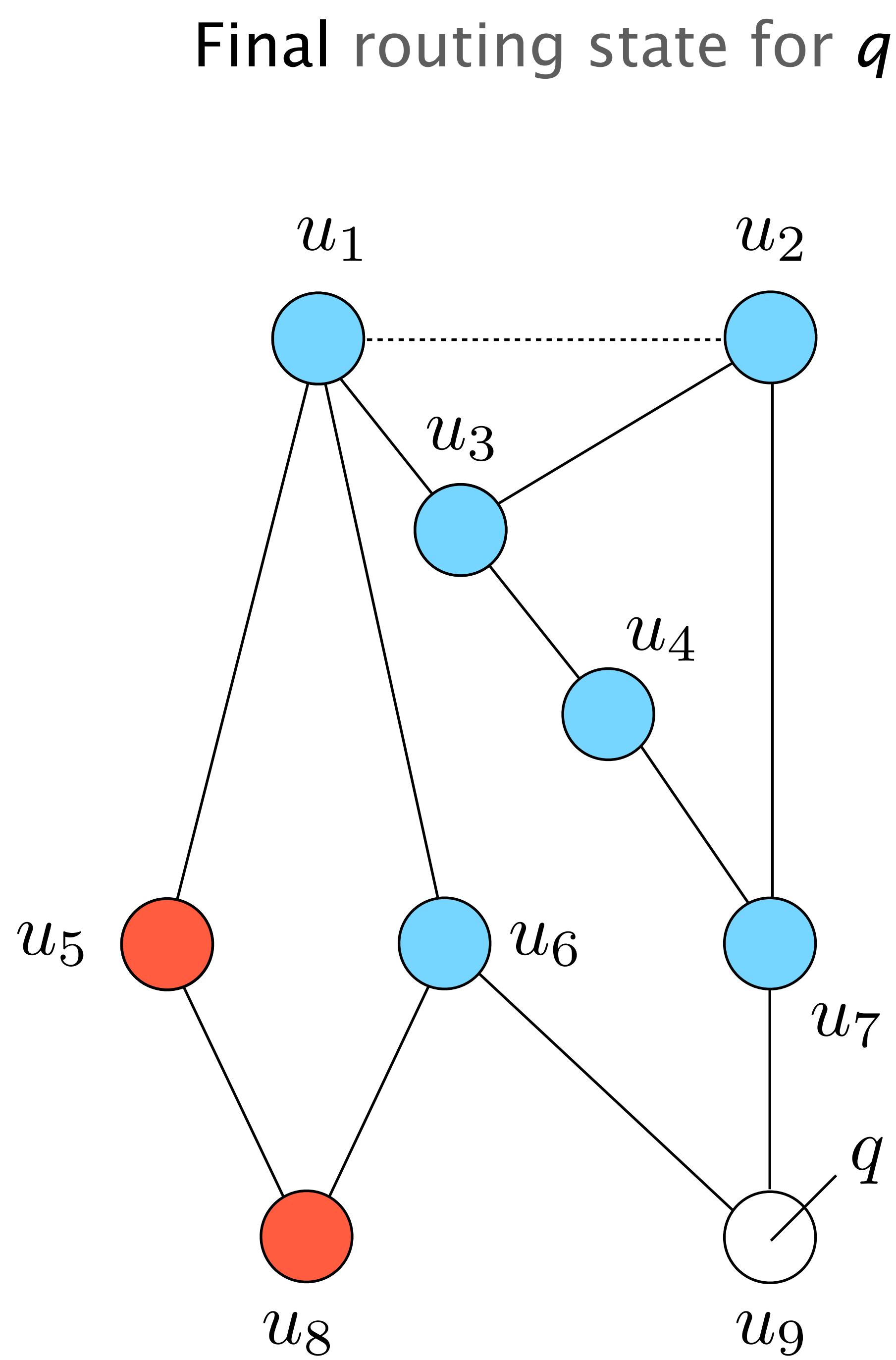
learned from **customer**

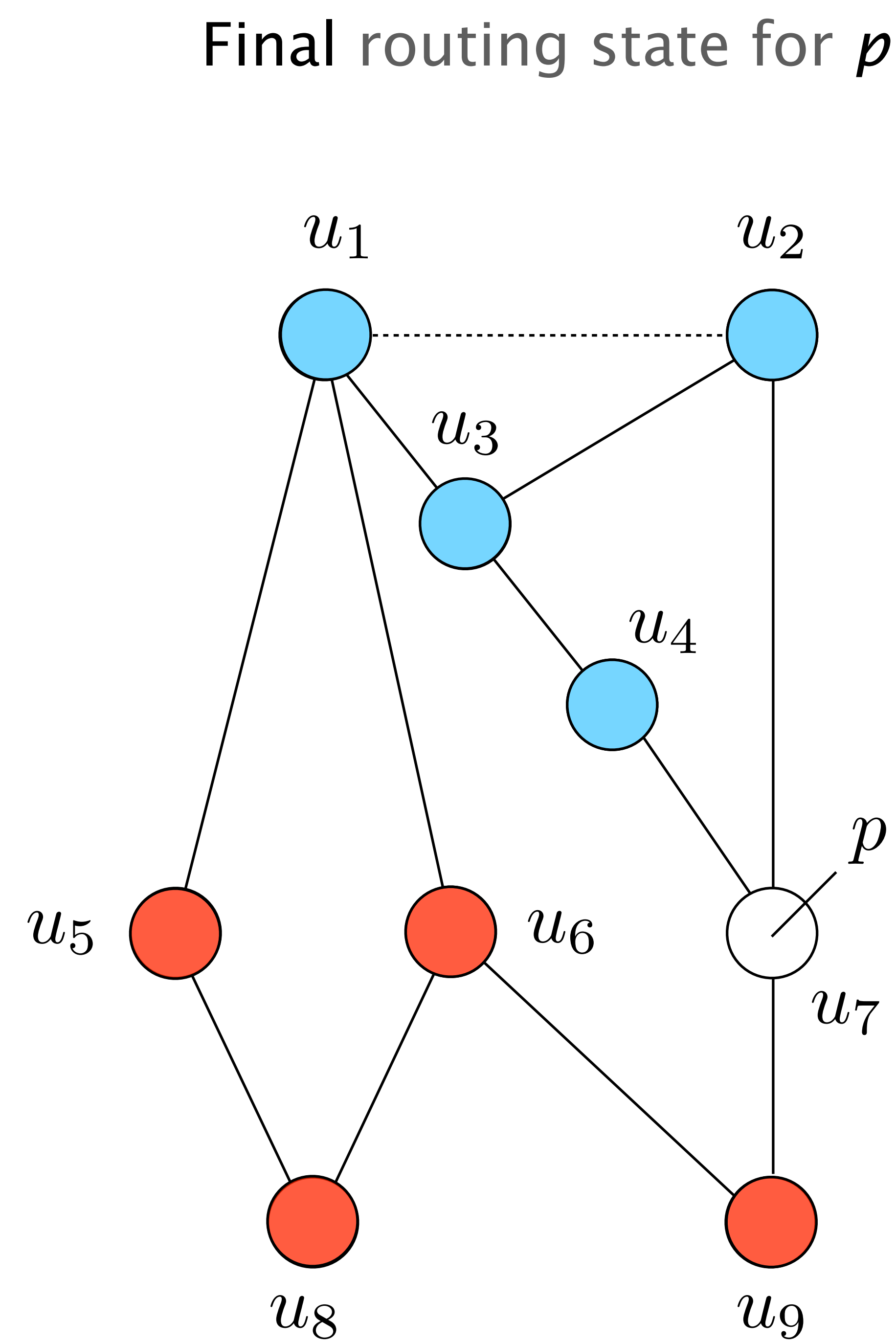
learned from **provider**

2 exportation rules

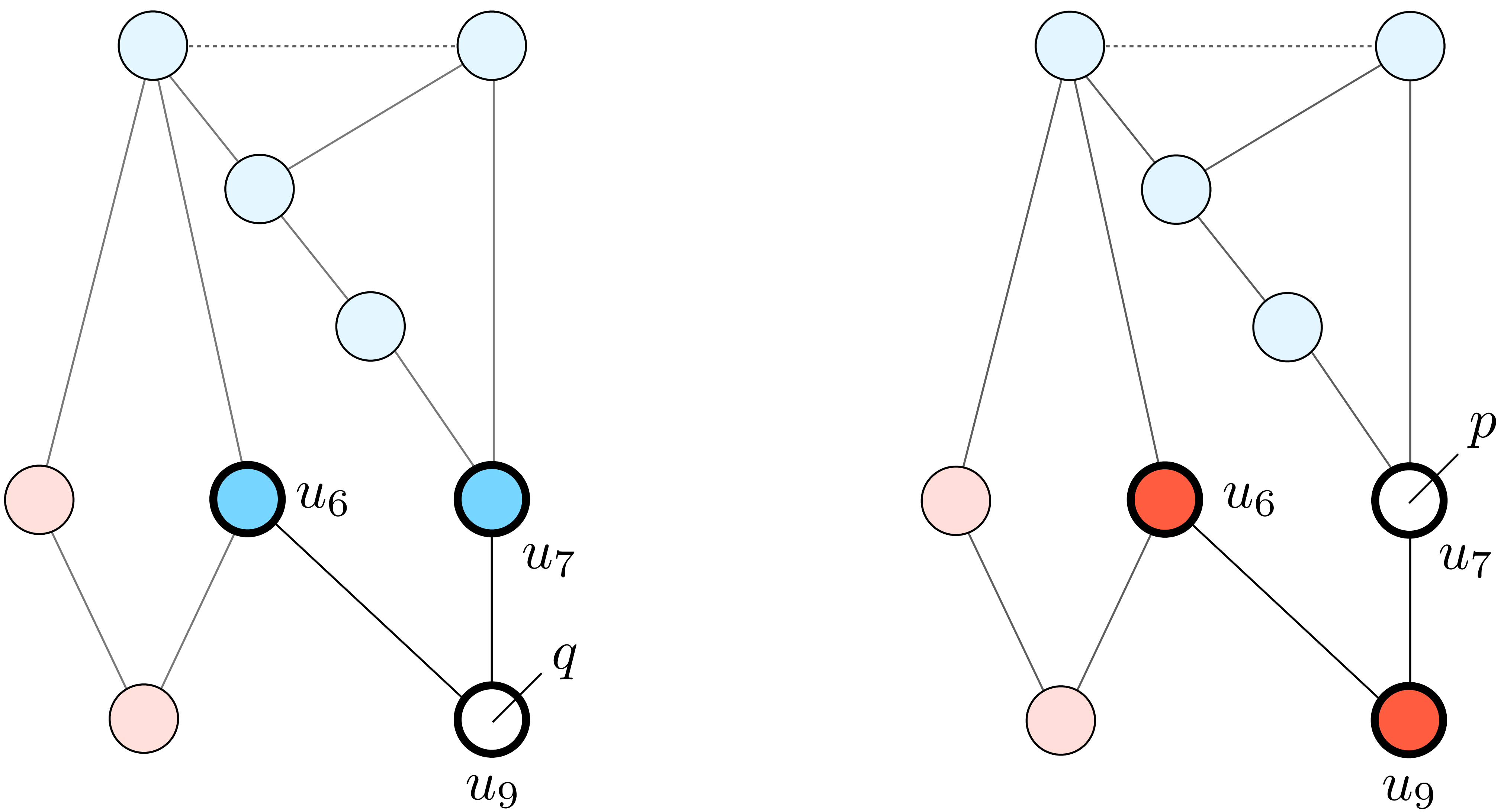
- customer routes to every neighbor
- provider routes to customers



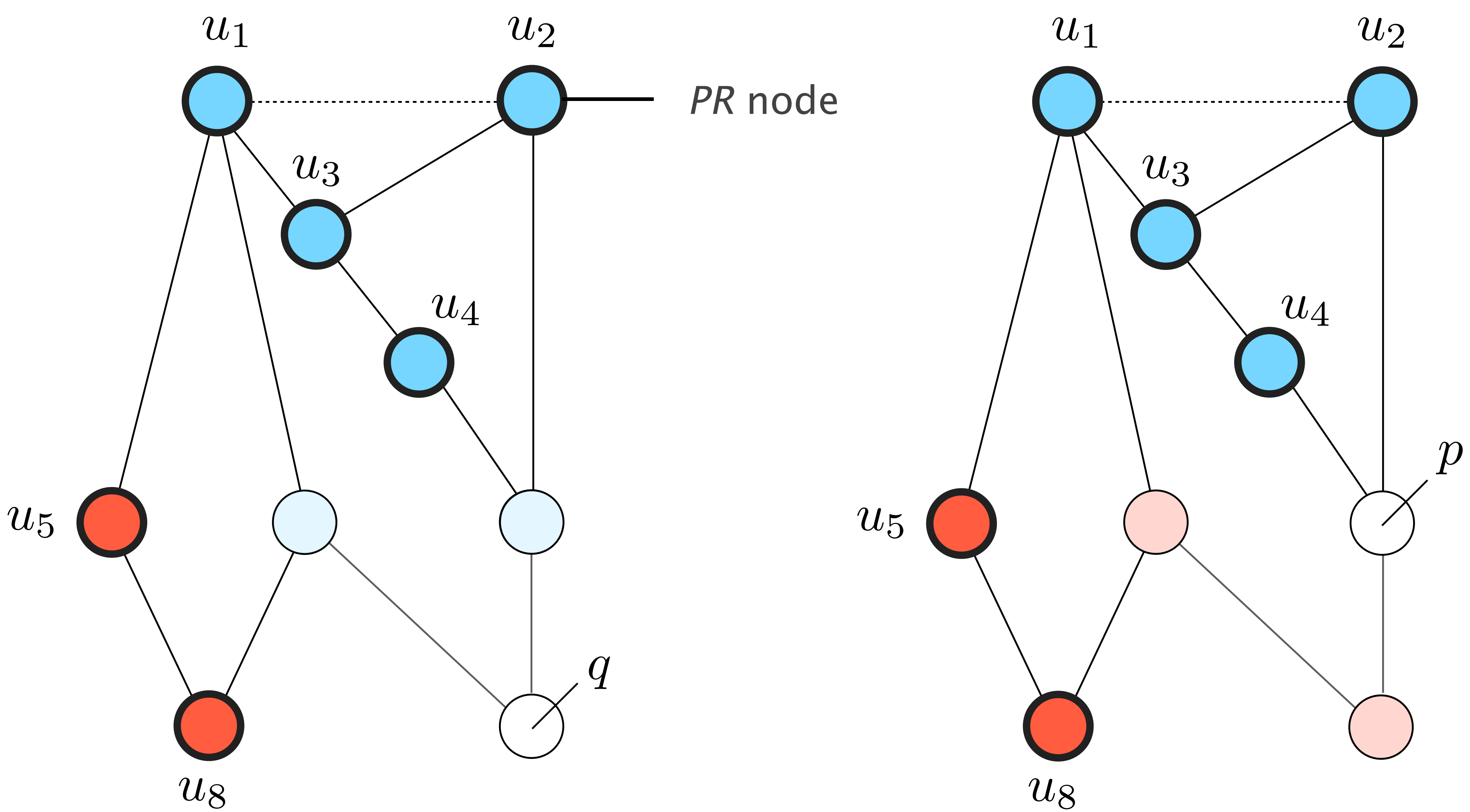

Final routing state for q
u_1
u_2
u_3
u_4
u_5
u_6
u_7
q
u_8
u_9


Final routing state for p
u_1
u_2
u_3
u_4
p
u_5
u_6
u_7
u_8
u_9

These three node elect different attribute for both q and p. They cannot filter.

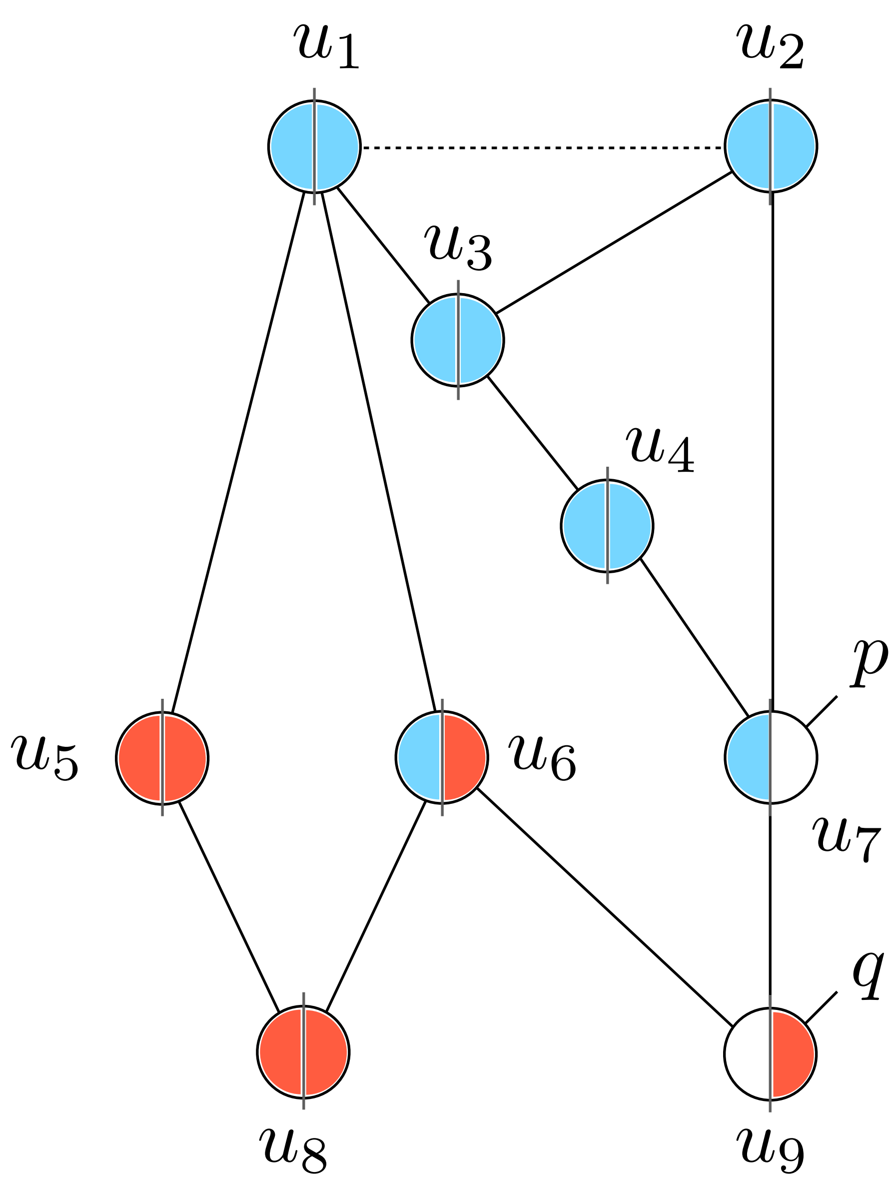


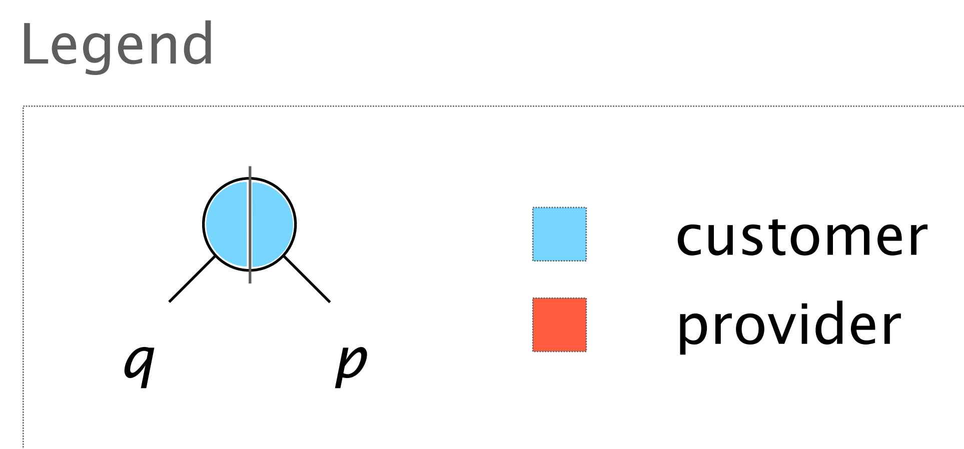
These node elect the same attribute for q and p. They are of type PR.



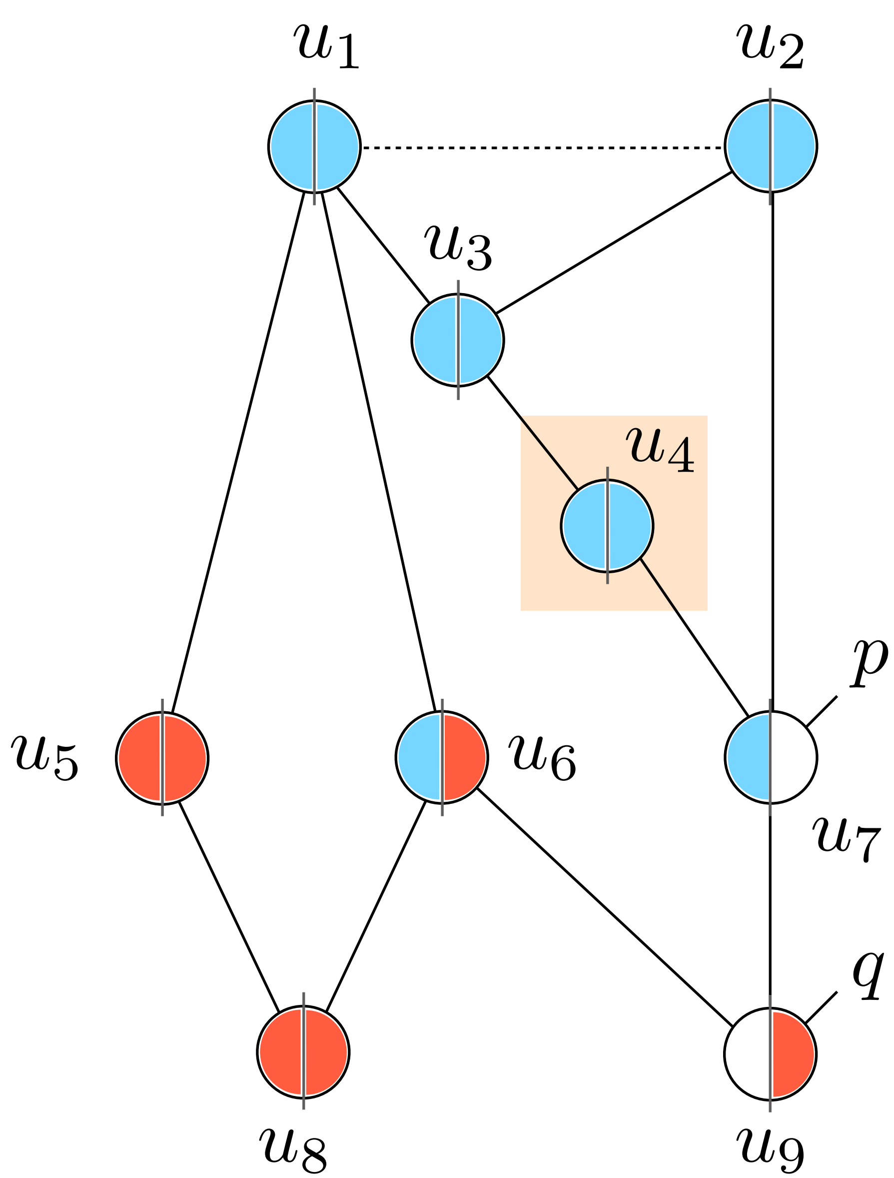
What if PR nodes filter?

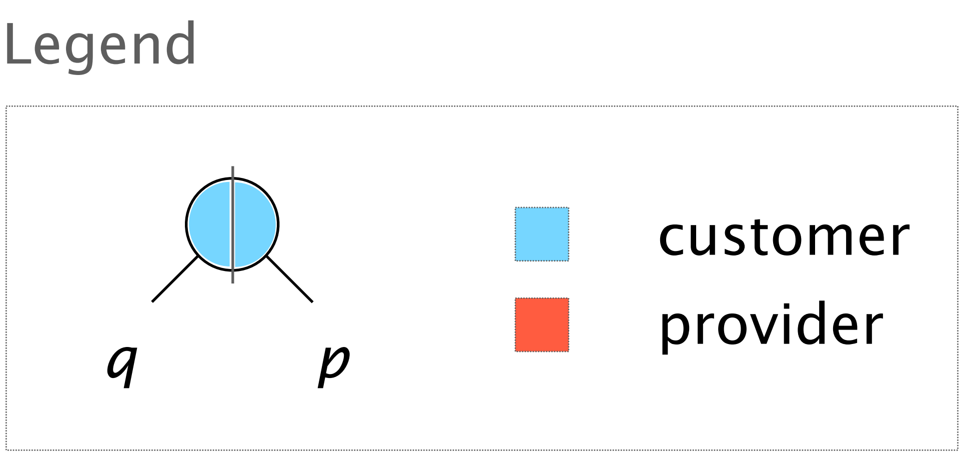
Combined routing state



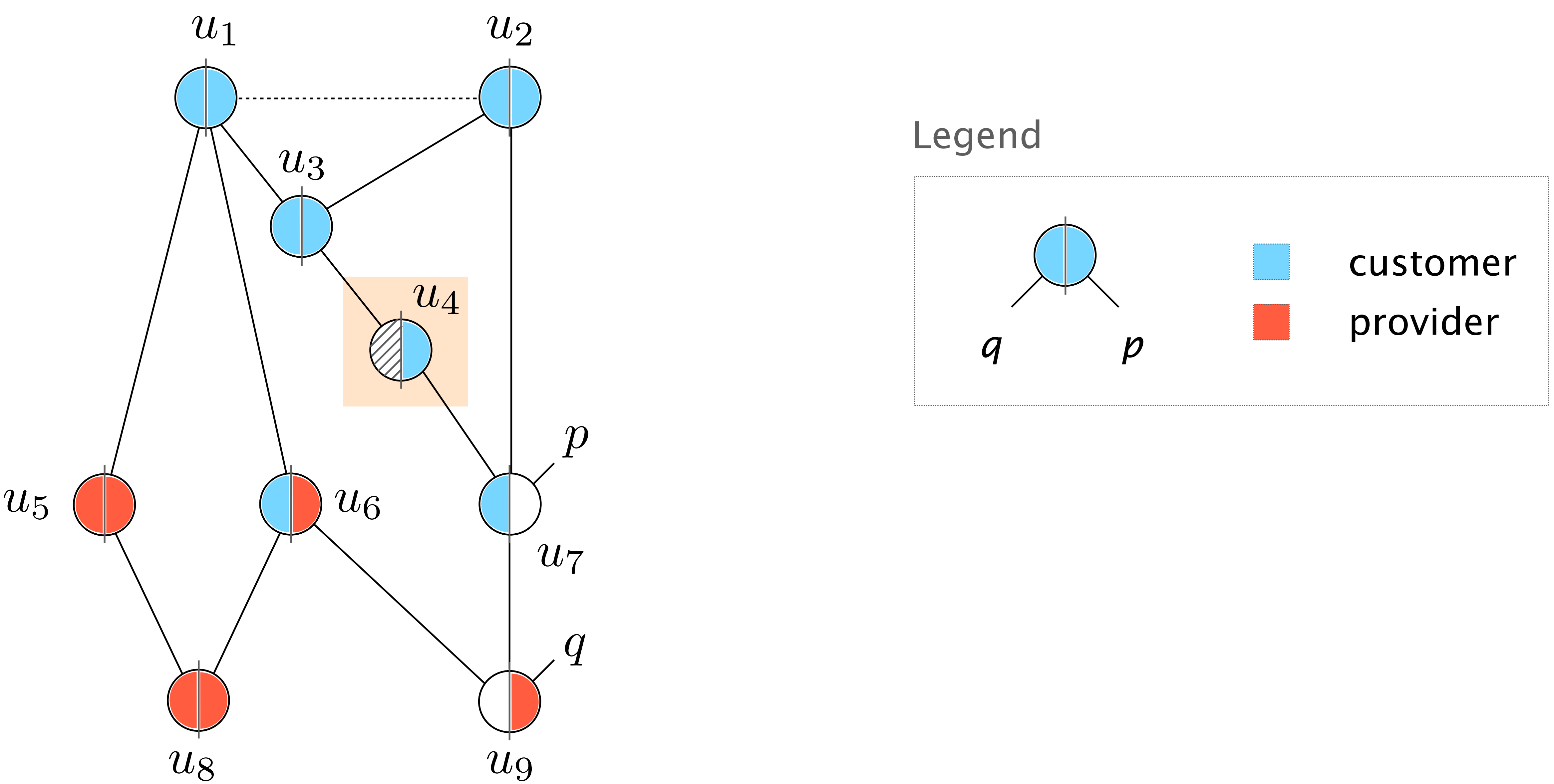


u$_4$ filters *q* and stops propagating it to u$_3$

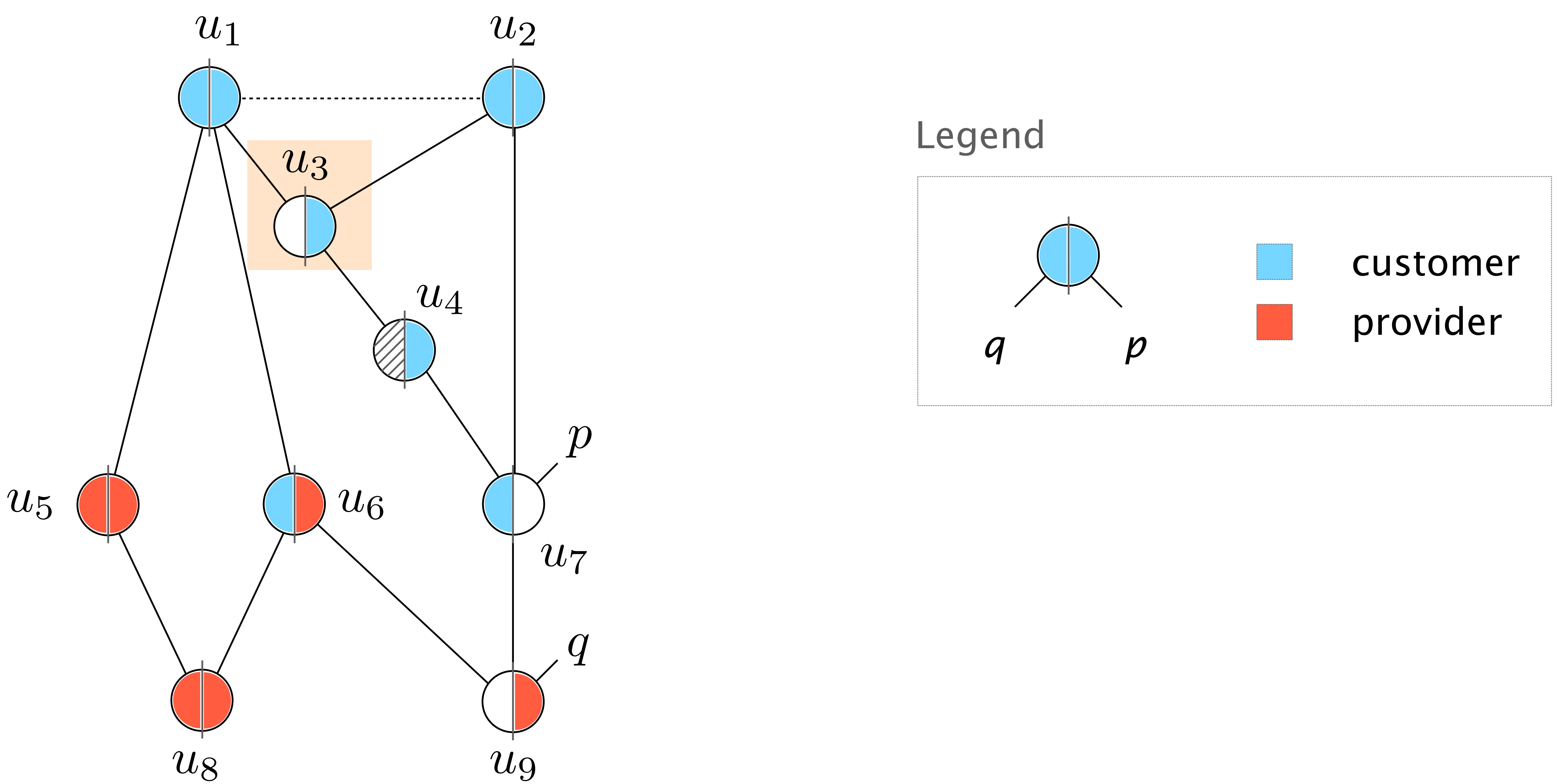




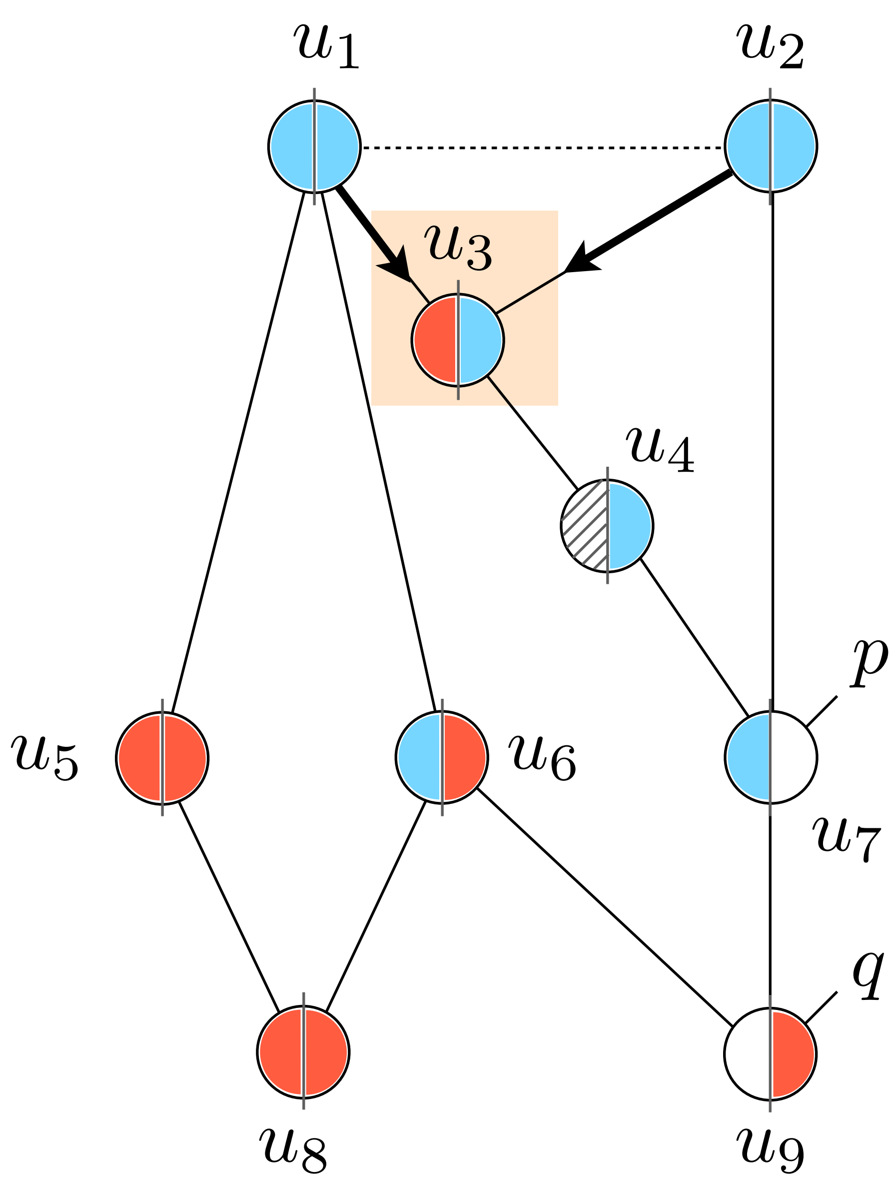
u_4 filters q and stops propagating it to u_3

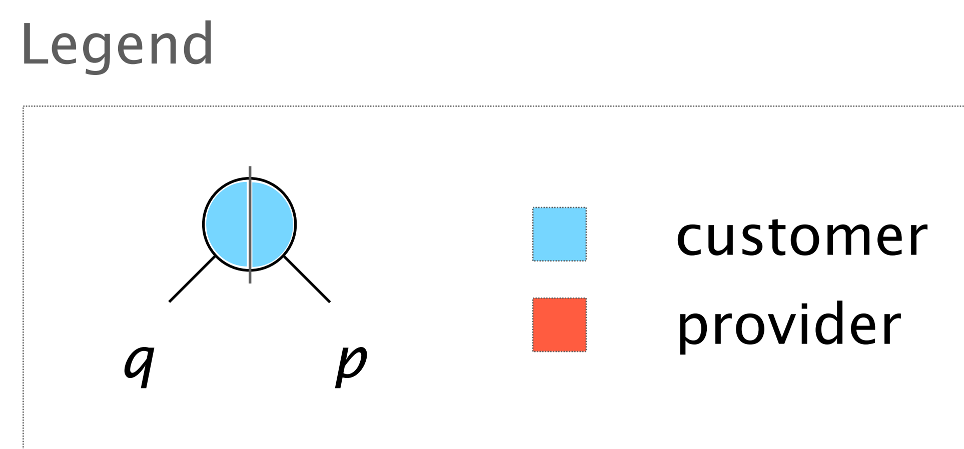


u₃ looses its only customer route to *q*

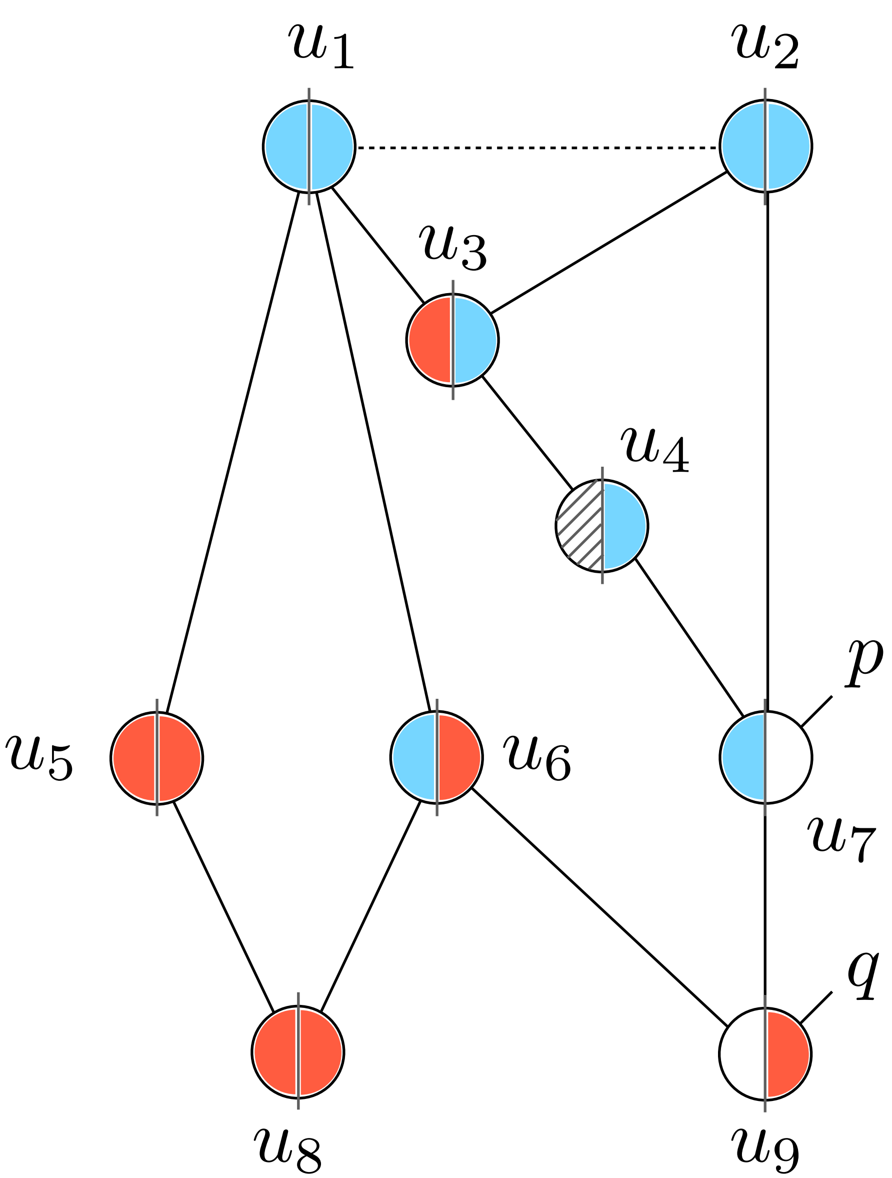


u$_3$ starts using a provider route for *q*

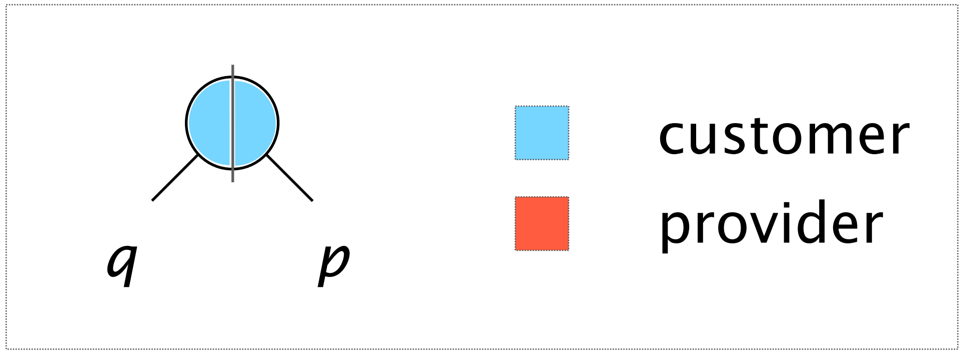




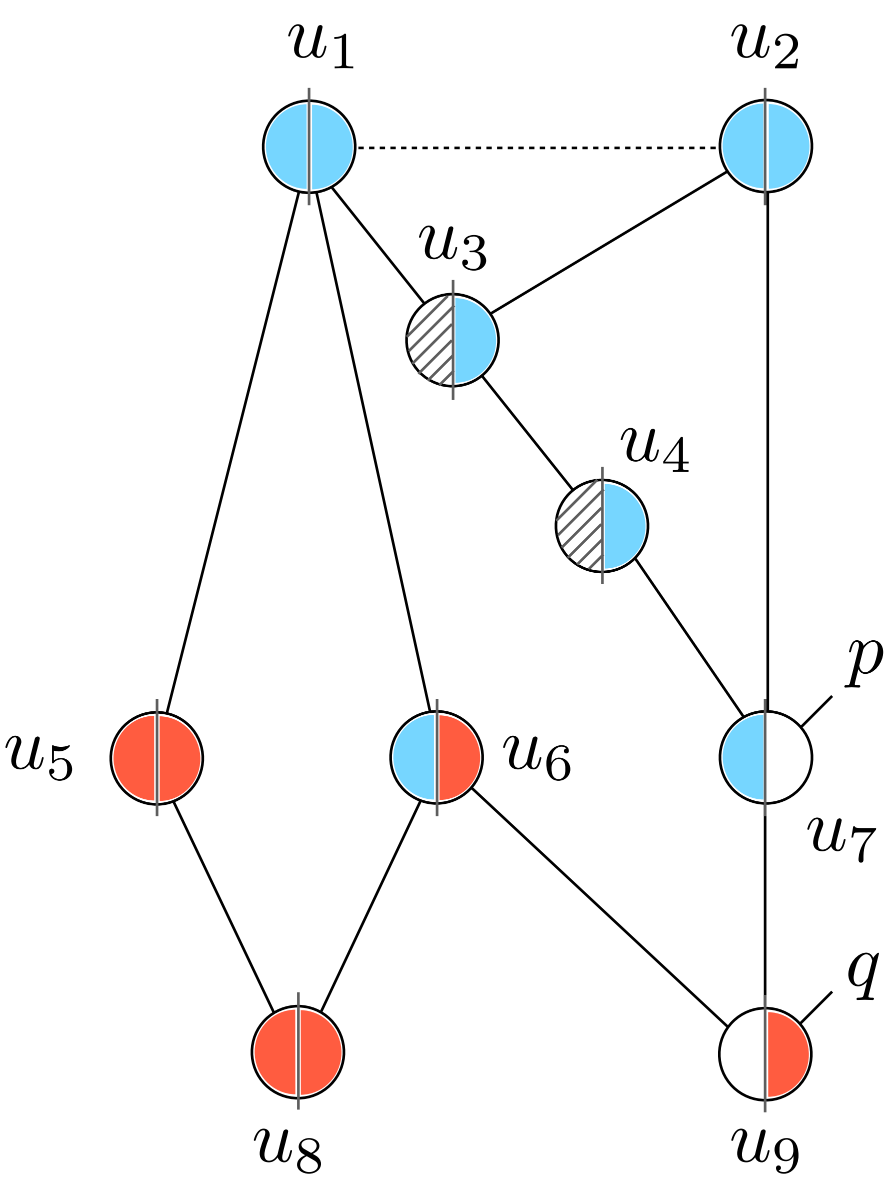
But what if u₃ filters?



Legend



if u_3 filters, it uses a customer route again for forwarding *q*



… *and* it saves space!

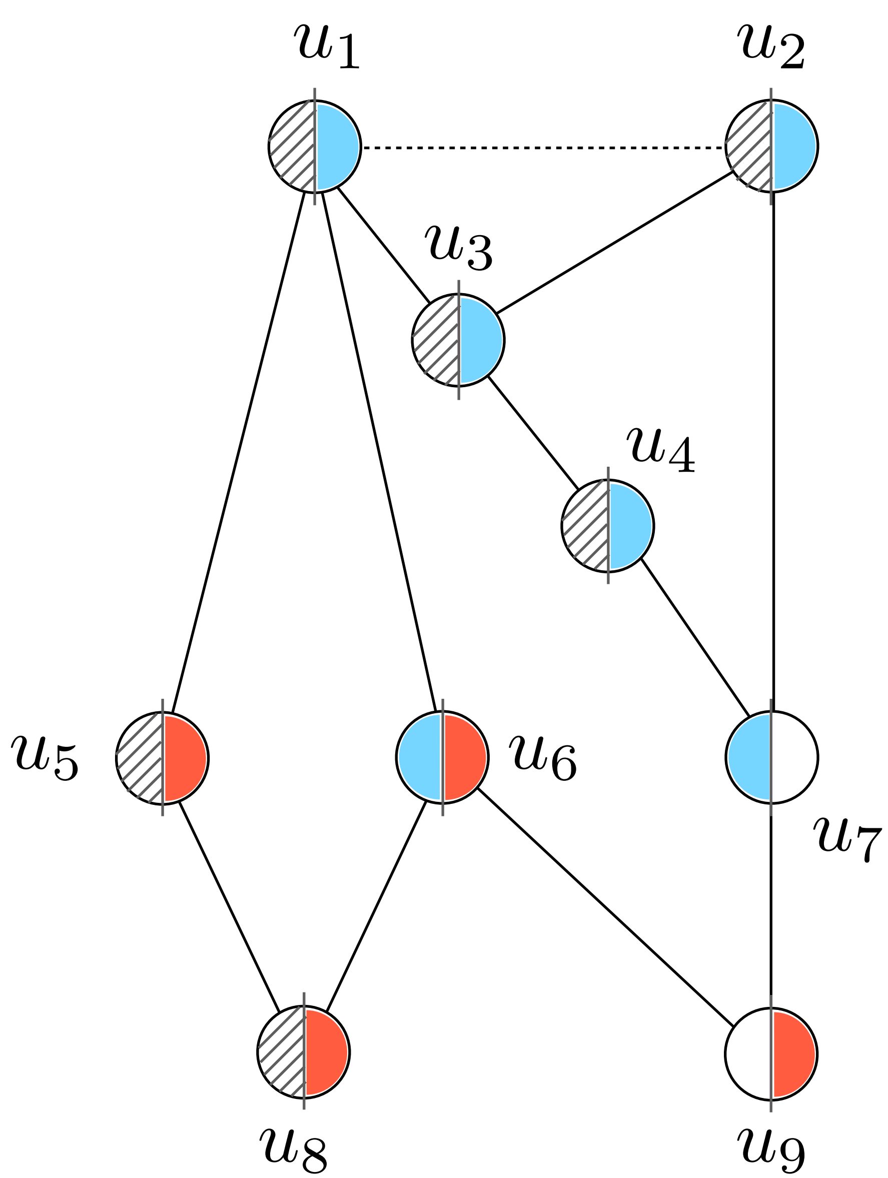
All PR nodes filtering is a Nash Equilibrium

Any node has two incentives to filter *q*-routes:

- retrieve a better route to forward traffic
- gain space in its routing and forwarding tables

with no node having an unilateral incentive to move away

Routing state post filtering is route consistent



Simple route consistent algorithm

Considering a node u,

a child prefix q,

its parent prefix p,

Simple route consistent algorithm

Considering a node u,

a child prefix q,

its parent prefix p,

Algorithm

If u is not the destination for q and

If elected q-route $\geq$ elected p-route

then u filters q-routes

The algorithm is *provably* correct

Theorem 3 — No matter the order in which node runs the algorithm, a route consistent state is eventually reached

The algorithm is *provably* correct

Theorem 1

For every node u, the elected q-route can only worsen when an arbitrary set of nodes filter q-routes

Theorem 3

No matter the order in which node runs the algorithm, a route consistent state is eventually reached

The algorithm is *provably* correct

Theorem 1 — For every node u, the elected q-route can only worsen when an arbitrary set of nodes filter q-routes

Theorem 2 — The elected q-route at a node u for which the elected q-route < elected p-route is not affected if an arbitrary set of nodes filters

Theorem 3 — No matter the order in which node runs the algorithm, a route consistent state is eventually reached

The algorithm is *provably* correct

Theorem 1

For every node u, the elected q-route can only worsen when an arbitrary set of nodes filter q-routes

Theorem 2

The elected q-route at a node u for which the elected q-route $<$ elected p-route is not affected if an arbitrary set of nodes filters

Theorem 3

No matter the order in which node runs the algorithm, a route consistent state is eventually reached

DRAGON relies on *isotonicity*, a property which characterizes the combined policies of two neighbors

Isotonicity

If an AS *u* prefers one route over another, a neighboring AS does not have the opposite preference

Observation

required for *optimality*, not *correctness*

verified in a lot of actual routing policies

DRAGON: Distributed Route AGgregatiON

Background

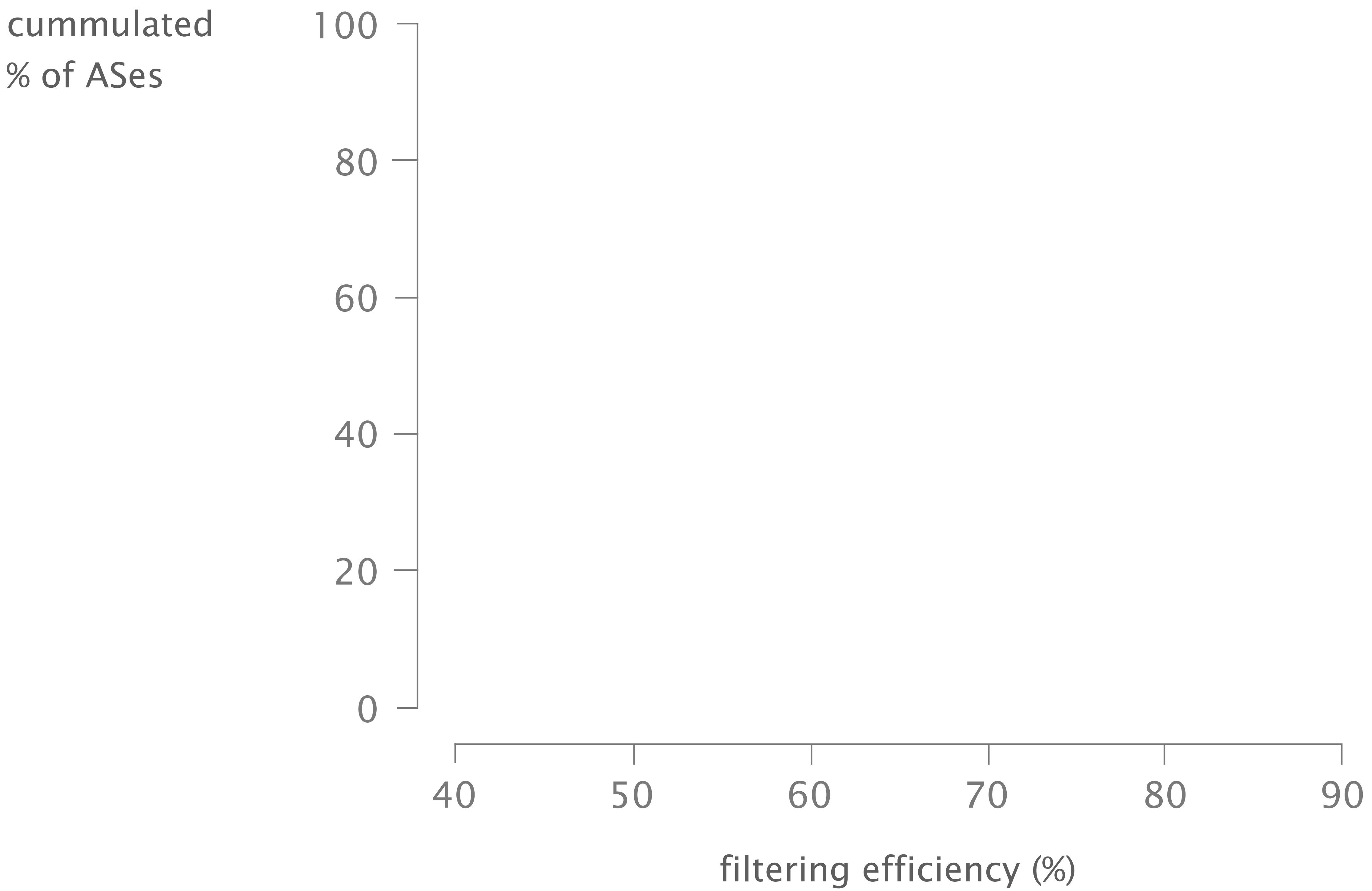
Route aggregation 101

Distributed filtering

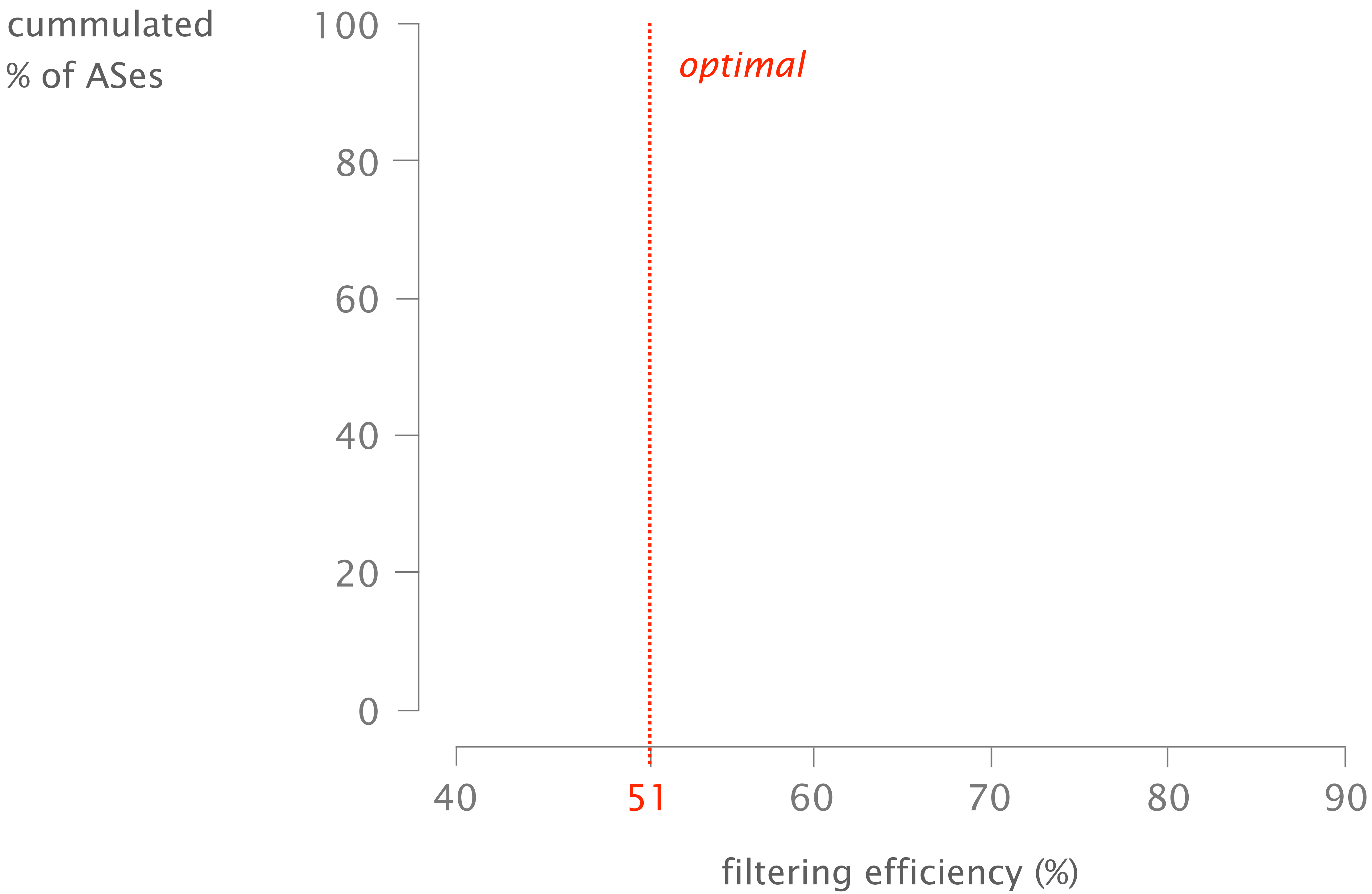
preserving consistency

3 Performance

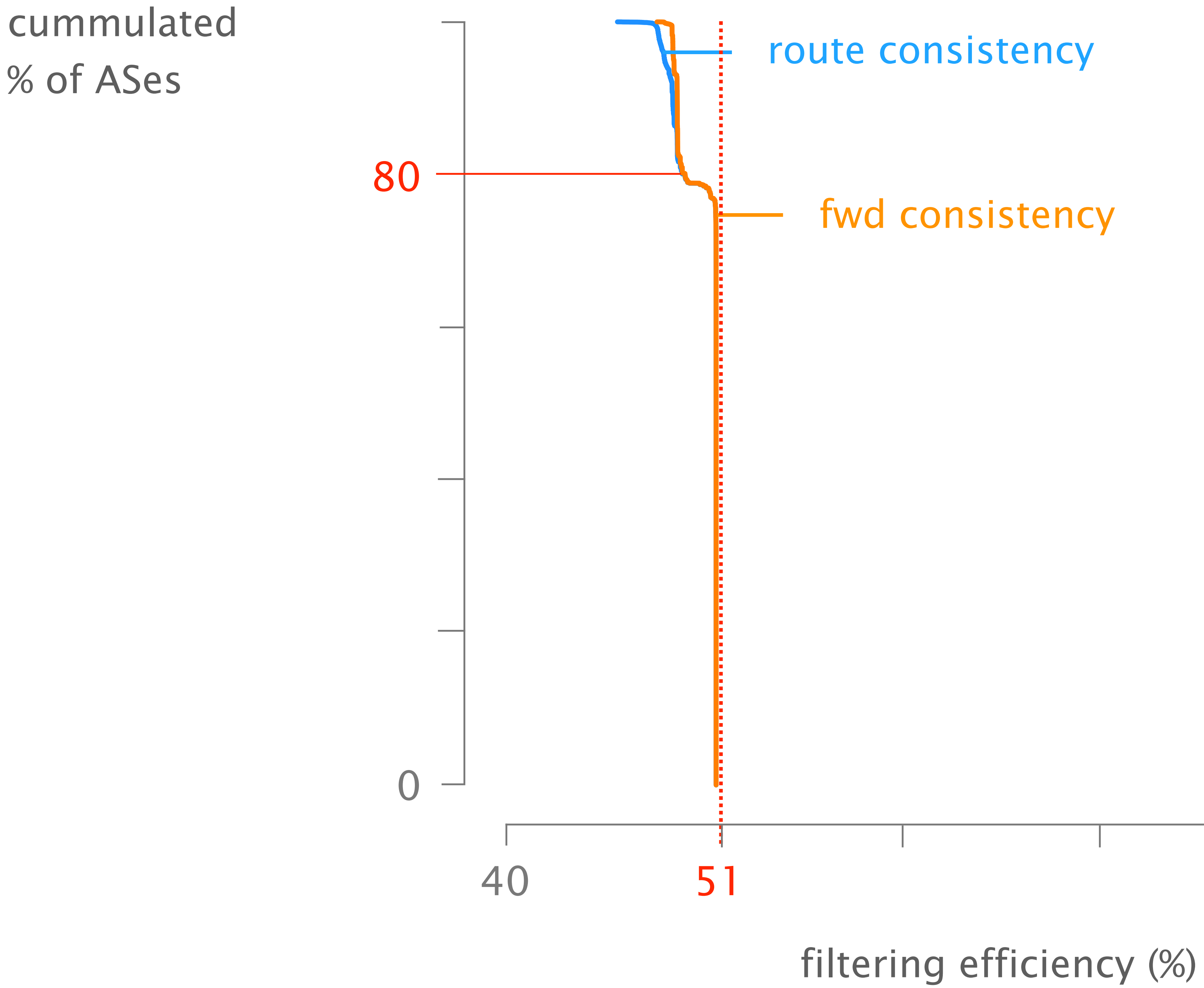
up to 80% of filtering efficiency


cummulated
% of ASes
100
80
60
40
20
0
40
50
60
70
80
90
filtering efficiency (%)

In today's Internet, optimal filtering is ~50% as half of the Internet prefixes are parentless



~80% of the ASes reaches optimal filtering efficiency



DRAGON node can automatically introduce aggregation prefix to filter prefixes without parent

Node can *autonomously* announce aggregation prefixes

based on local computation and preserving consistency

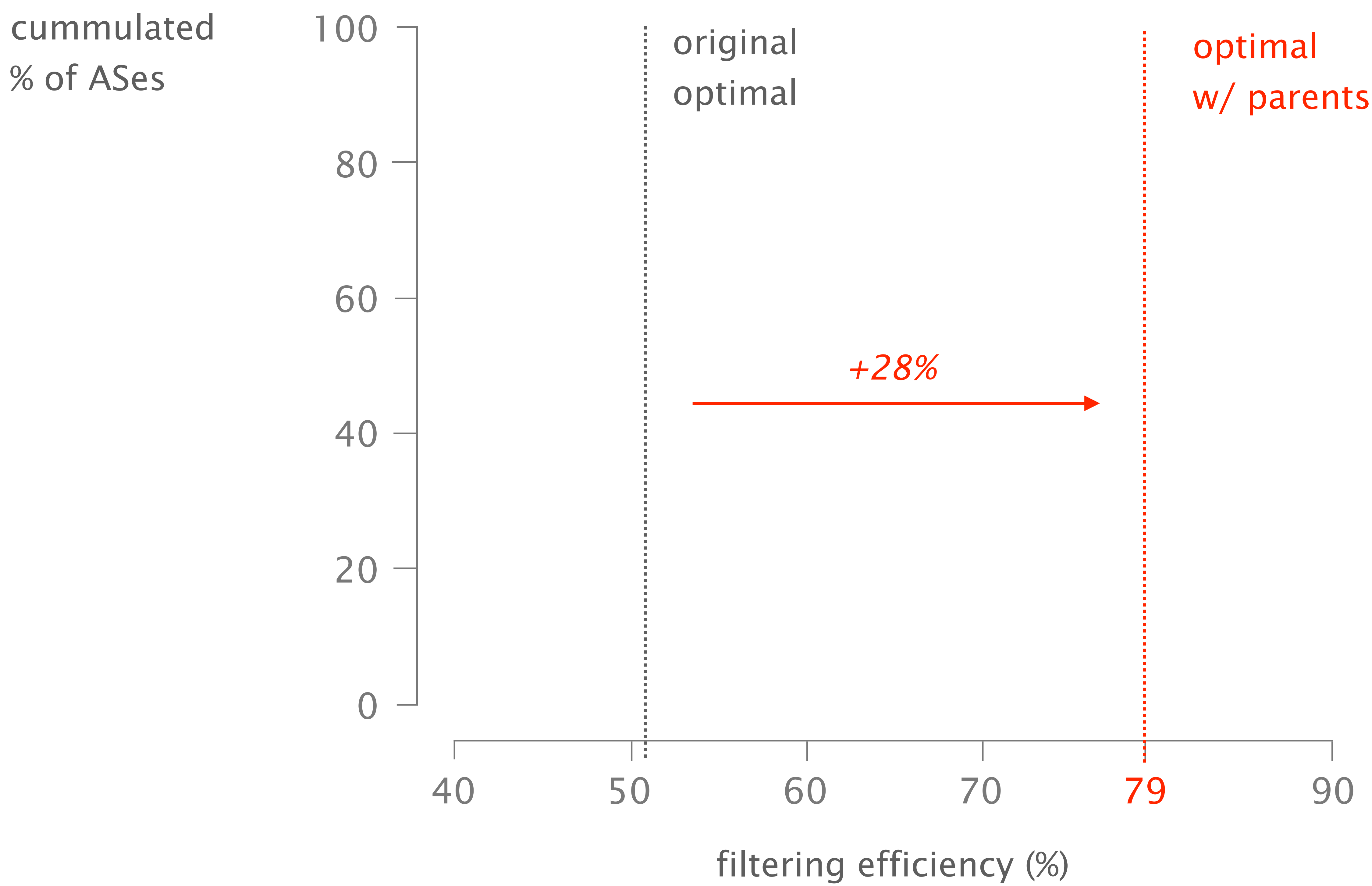
Routing system self-organizes itself in case of conflict

when more than one node announce the same parent prefix

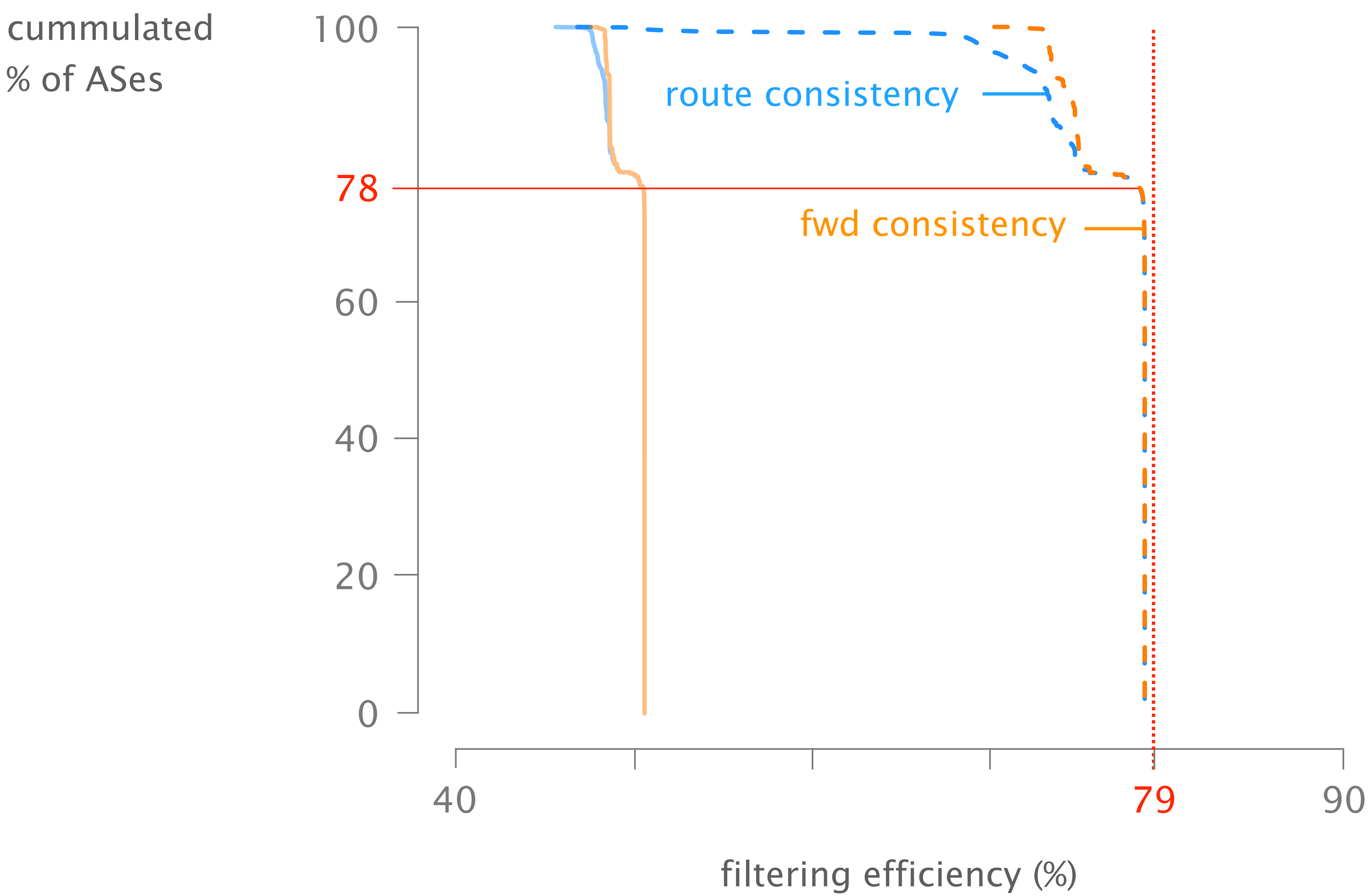
Number of aggregation prefixes introduced can be tuned

e.g., maximum prefix length or minimum # covered children

Introducing <10% of parent prefixes boosts the optimal efficiency to 79%



Again, ~80% of the ASes reaches optimal filtering efficiency



DRAGON: Distributed Route AGgregatiON

Background

Route aggregation 101

Distributed filtering

preserving consistency

Performance

up to 80% of filtering efficiency

DRAGON is a distributed route-aggregation algorithm which automatically harnesses any aggregation potential

DRAGON works on today's routers

only require a software update and offers incentives to do it

DRAGON preserves *routing* and *forwarding* decision

leveraging the isotonicity properties of Internet policies

DRAGON is more general than BGP

shortest-path, ad-hoc networks, etc.