Boosting existing networks with SDN

A bird in the hand is worth two in the bush

Laurent Vanbever

ETH Zürich (D-ITET)

Swisscom Innovation

May, 28 2015

Software-Defined Network

In few years, SDN has attracted **tremendous** industry interest (and money)

VMware Acquires Once-Secretive Start-Up Nicira for $1.26 Billion

JULY 23, 2012 AT 1:25 PM PT | Tweet | Share | 8+1 | in Share | Share | Print

VMware, the software company best known for its virtualization technology that forms the backbones of so-called cloud computing today, said it will pay $1.26 billion for Nicira, a networking start-up that has sought to do to networks what VMware has done to computers.

nicira

The news comes on the same day that VMware was to report quarterly earnings. And while I don't usually cover VMware's earnings, I may as well mention the results: The company reported revenue for the quarter ended June rose to $1.12 billion, while earnings on a per-share basis were 68 cents. Analysts had been expecting sales of $1.12 billion and earnings of 66 cents.

Nicira had been running in stealth mode for quite awhile; I got to reveal its plans to the world last February.

The deal amounts to a nice payoff for Nicira's investors including Andreessen Horowitz, Lightspeed Venture Partners and NEA, as well as VMware founder Diane Greene and venture capitalist Andy Rachleff.

With $600M Invested in SDN Startups, the Ecosystem Builds

Scott Raynovich, June 10, 2014

More than $600 million has been invested in at least two dozen software-defined networking (SDN) startups so far, according to Rayno Report research. You can expect that to continue to climb. With the SDN ecosystem starting to take hold with a broad range of alliances and distribution partnerships, we're just getting started.

The Arista IPO will help build visibility for next-generation, software-driven networking. But Arista is selling its own hardware and is not an SDN pure-play. A new line of SDN startups, with a more radical approach to software-based networking, is building momentum. These newer SDN startups are just getting their gear into customers' hands and starting to build sales channels, so you can expect a long revenue ramp.

This excitement is boosting startup valuations, according to Rayno Report research. There are now at least ten SDN startups with valuations over $100 million. As I reported in April, a recent investment in Cumulus Networks pushed up the valuation of the private company north of $300 million, according to industry sources. Big Switch, which did a deal in 2012 valuing it near $170 million, took money from Intel in 2013, most likely boosting its valuation to over $200 million, according to several sources.

Related Articles

- How to Effectively Embed SDN in the Enterprise
- NFV and SDN: What's the Difference Two Years Later?
- sFlow Creator Peter Phaal On Taming The Wilds Of SDN & Virtual Networking
- Featured Article: Bringing Data-Driven SDN to the Network Edge
- NFV Delivers Pervasive Intelligence for MNOs

Open Networking Foundation

03/2011 founded

148 members

34 startup-members

… and growing!

The SDN momentum also grows in academia

The SDN momentum also grows in academia

OpenFlow: enabling innovation in campus networks
N McKeown, T Anderson, H Balakrishnan… - ACM SIGCOMM …, 2008 - dl.acm.org
Abstract This whitepaper proposes **OpenFlow**: a way for researchers to run experimental protocols in the networks they use every day. **OpenFlow** is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. Our **...**
Cited by 2829 Related articles All 106 versions Web of Science: 456 Cite Save

The SDN momentum also grows in academia

OpenFlow: enabling innovation in campus networks
N McKeown, T Anderson, H Balakrishnan… - ACM SIGCOMM …, 2008 - dl.acm.org
Abstract This whitepaper proposes **OpenFlow**: a way for researchers to run experimental protocols in the networks they use every day. **OpenFlow** is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. Our **...**
Cited by 2829 Related articles All 106 versions Web of Science: 456 Cite Save

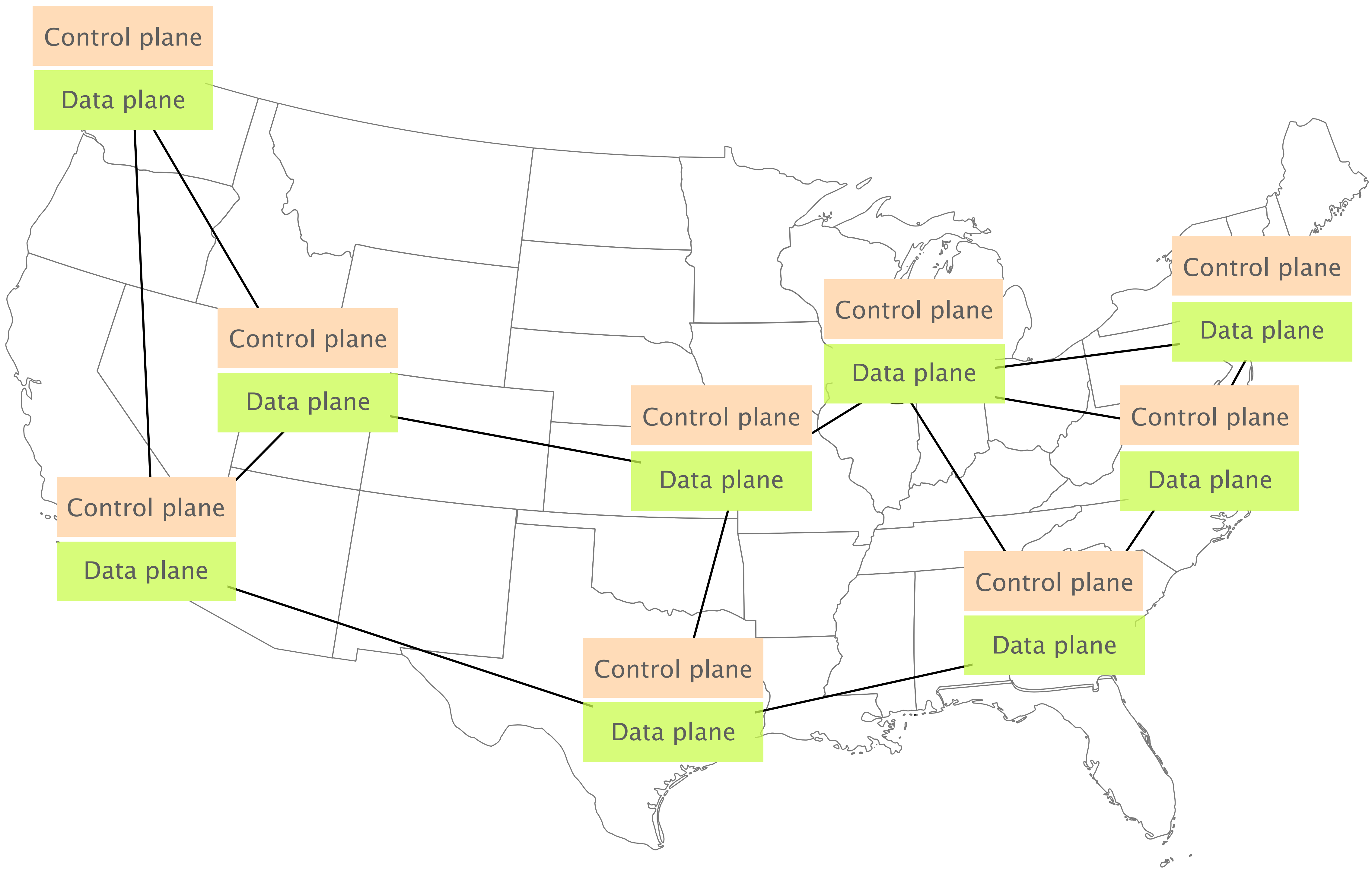
in ~6 years

SDN is reaching into many CS communities

Networking	Systems	Distributed Algorithms	Security	PL
SIGCOMM	OSDI	PODC	CCS	PLDI
NSDI	SOSP	DISC	NDSS	POPL
HotNets	SOCC		Usenix Security	OOPSLA
CoNEXT				

Why?!

A network is a distributed system whose behavior depends on each element configuration



Configuring each element is often done manually, using arcane low-level, vendor-specific “languages”

Configuring each element is often done manually, using arcane low-level, vendor-specific “languages”

Cisco IOS

```
!
ip multicast-routing
!
interface Loopback0
 ip address 120.1.7.7 255.255.255.255
 ip ospf 1 area 0
!
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.17
 encapsulation dot1Q 17
 ip address 125.1.17.7 255.255.255.0
 ip pim bsr-border
 ip pim sparse-mode
!
!
router ospf 1
 router-id 120.1.7.7
 redistribute bgp 700 subnets
!
router bgp 700
 neighbor 125.1.17.1 remote-as 100
 !
 address-family ipv4
  redistribute ospf 1 match internal external 1 external 2
  neighbor 125.1.17.1 activate
 !
 address-family ipv4 multicast
  network 125.1.79.0 mask 255.255.255.0
  redistribute ospf 1 match internal external 1 external 2
```

Juniper JunOS

```
interfaces {
   so-0/0/0 {
        unit 0 {
            family inet {
                address 10.12.1.2/24;
            }
            family mpls;
        }
    }
   ge-0/1/0 {
        vlan-tagging;
        unit 0 {
            vlan-id 100;
            family inet {
                address 10.108.1.1/24;
            }
            family mpls;
        }
        unit 1 {
            vlan-id 200;
            family inet {
                address 10.208.1.1/24;
            }
        }
    }
…
}
protocols {
    mpls {
        interface all;
    }
    bgp {
```

“Human factors are responsible

for 50% to 80% of network outages”

Juniper Networks, *What’s Behind Network Downtime?*, 2008

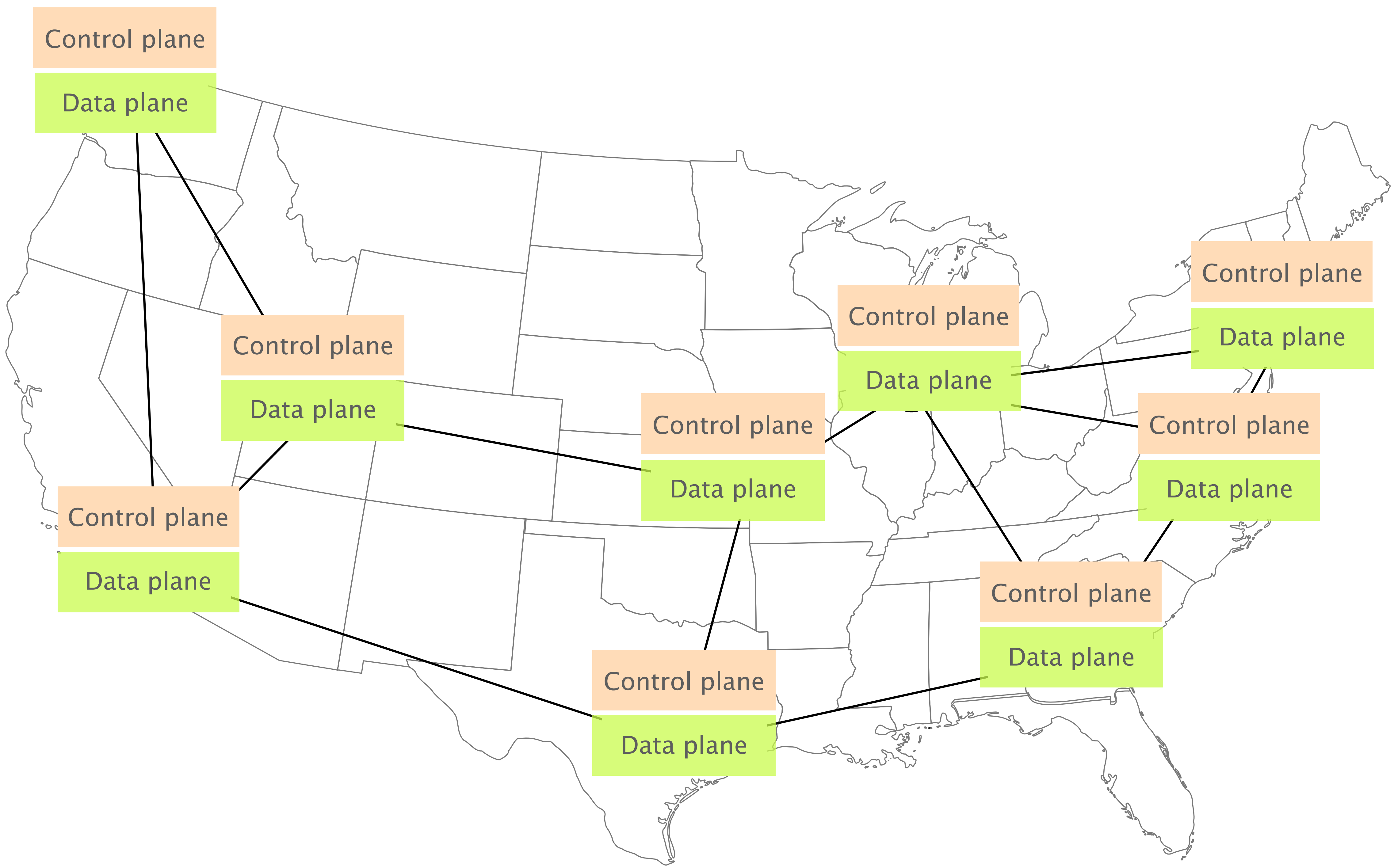
"Cost per network outage

can be as high as 750 000$"

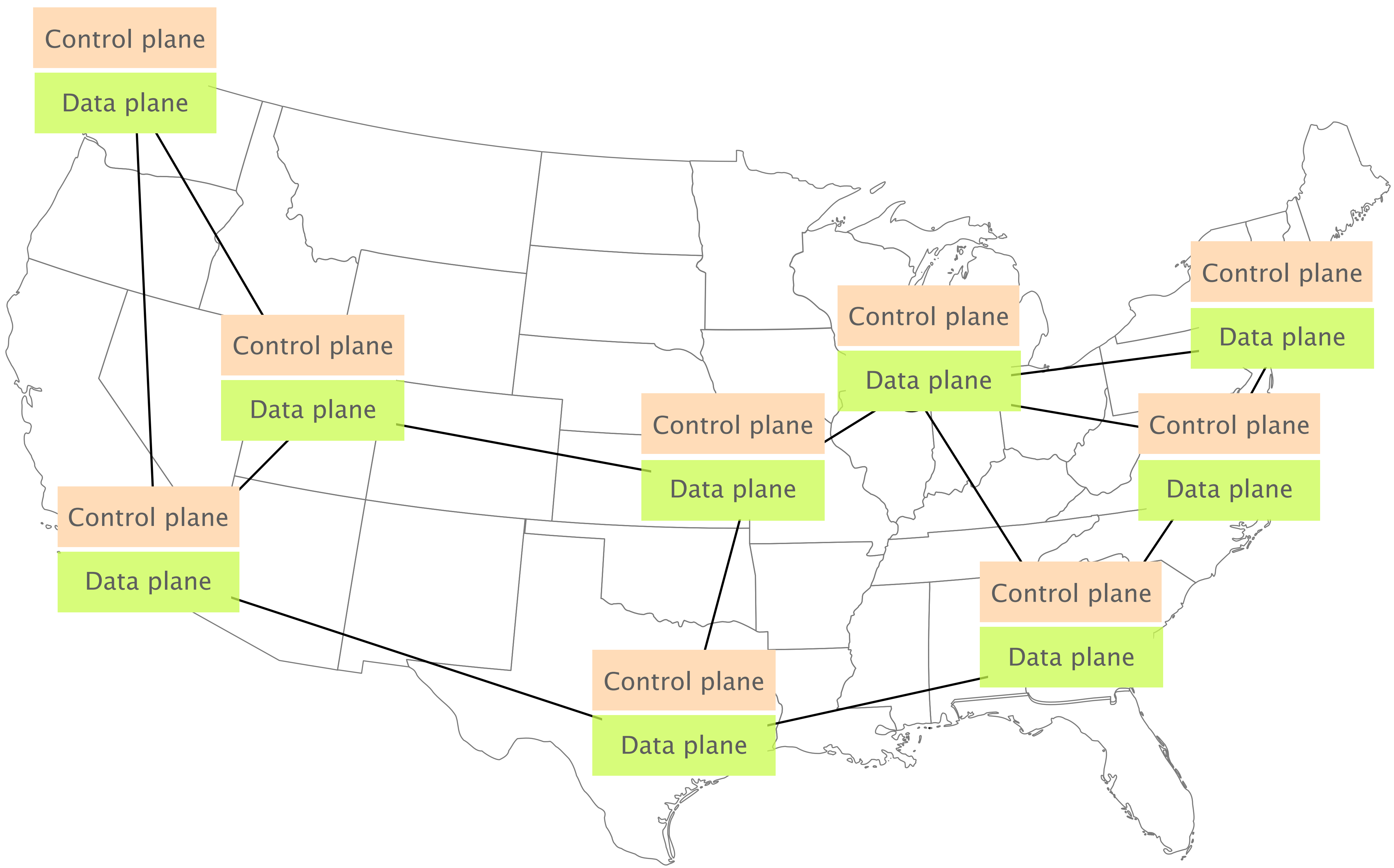
Smart Management for Robust Carrier Network Health

and Reduced TCO!, NANOG54, 2012

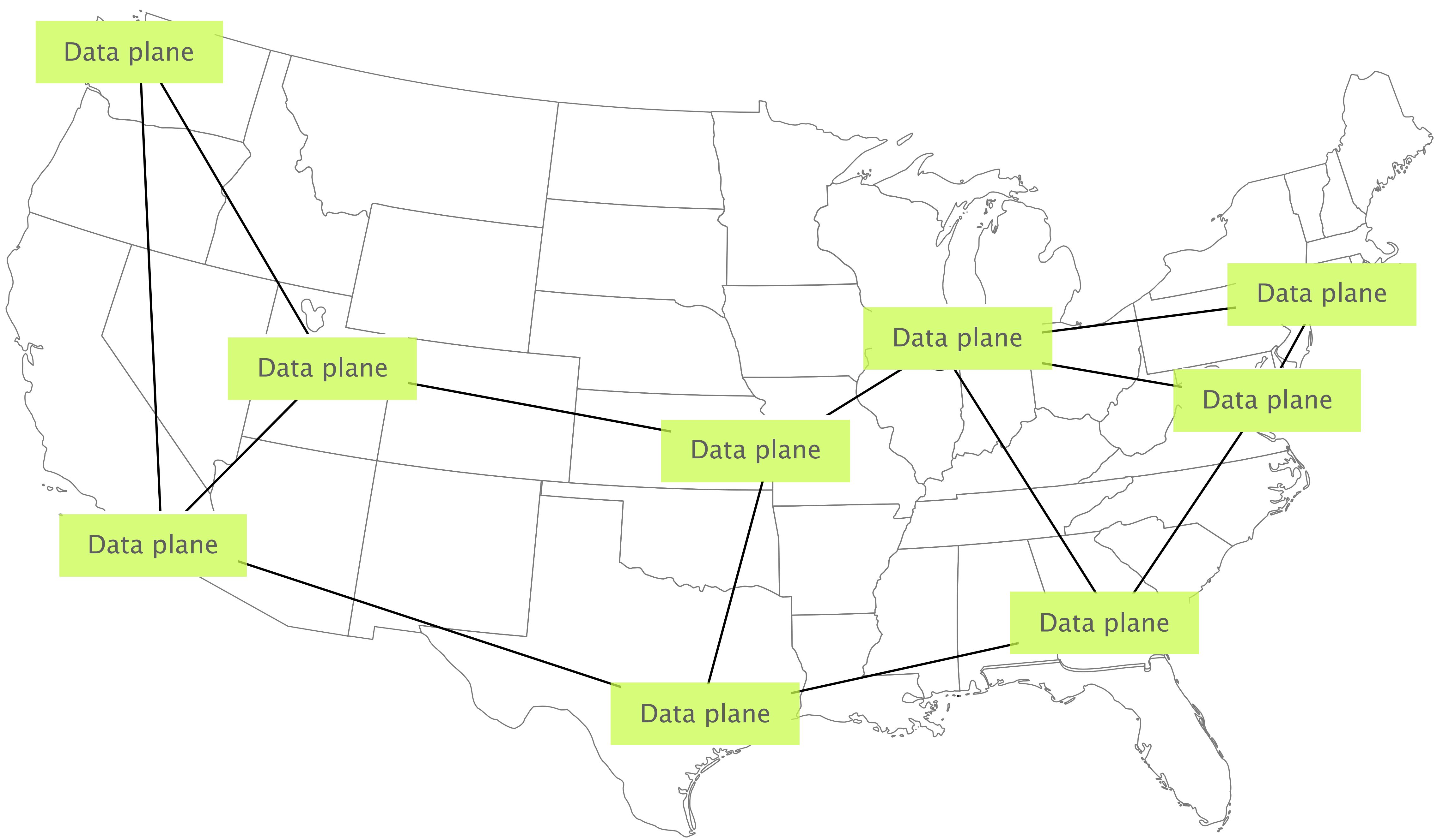
In contrast, SDN simplifies networks…



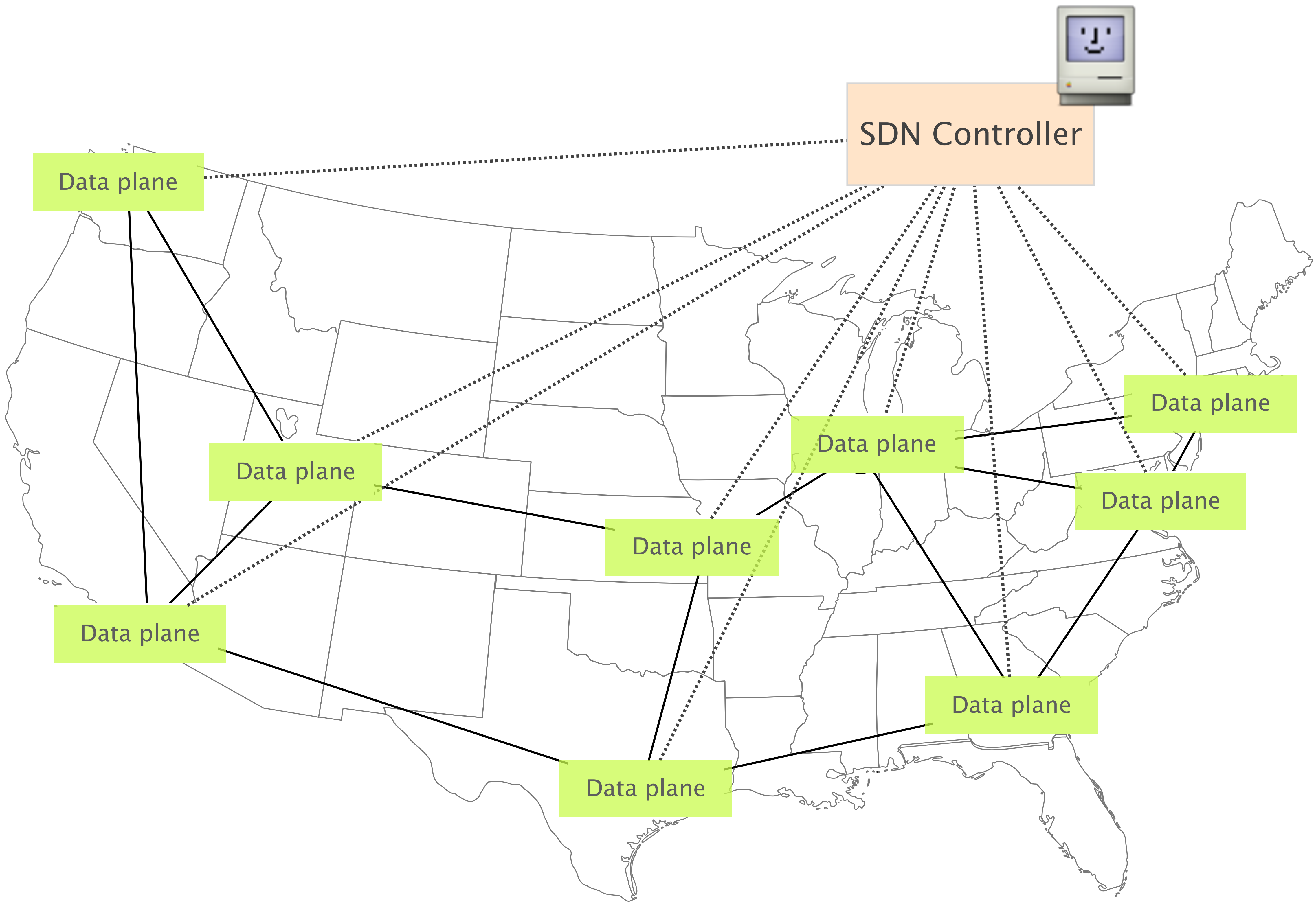
… by removing the intelligence from the equipments



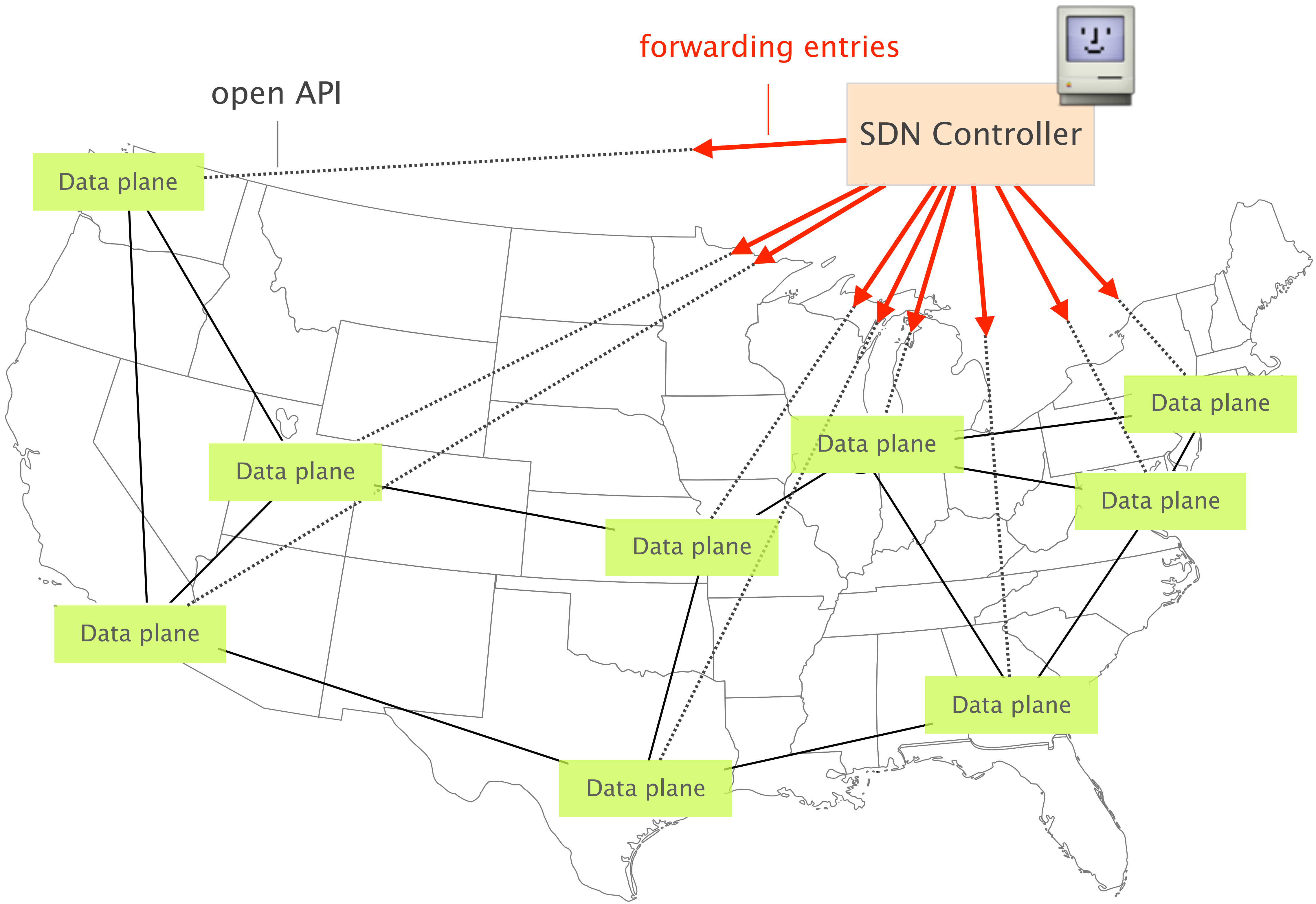
… by removing the intelligence from the equipments



… and **centralizing it** in a SDN controller that can run arbitrary programs



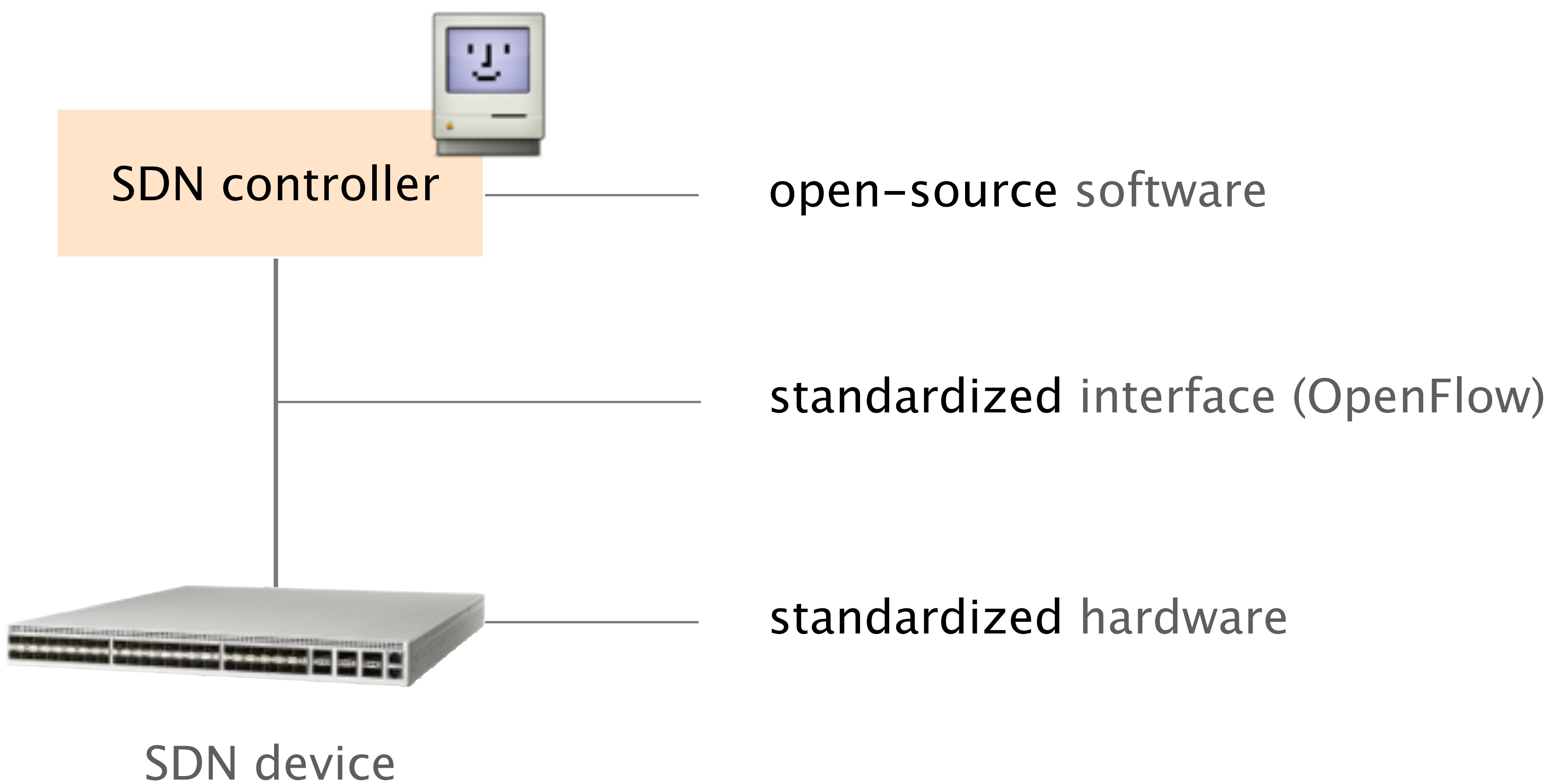
The SDN controller programs forwarding state in the devices using an open API (e.g., OpenFlow)



SDN also enables us, researchers, to innovate, at a much faster pace

Cisco™ device

SDN also enables us, researchers, to innovate, at a much faster pace



Sounds great

Sounds great, but…

How do you go from a traditional network to a SDN-enabled one?

Well… not easily

Deploying SDN requires to upgrade network …

- devices
- management systems
- operators

challenging, time-consuming and therefore **costly**

To succeed, SDN-based technologies should possess at least 3 characteristics

Small investment

Low risk

High return

To succeed, SDN-based technologies should possess at least 3 characteristics

Small investment

provide benefits
under partial deployment
(ideally, with a single switch)

Low risk

High return

To succeed, SDN-based technologies should possess at least 3 characteristics

Small investment

Low risk

require minimum changes to operational practices

be compatible with existing technologies

High return

To succeed, SDN-based technologies should possess at least 3 characteristics

Small investment

Low risk

High return — solve a timely problem

This talk is about two such SDN-based technologies

Fibbing

improved flexibility

Supercharged

performance boost

Fibbing

improved flexibility

central control over distributed system

Supercharged

performance boost

Fibbing

improved flexibility

Supercharged

performance boost

reduce convergence time by 1000x

Fibbing
improved flexibility

central control over distributed system

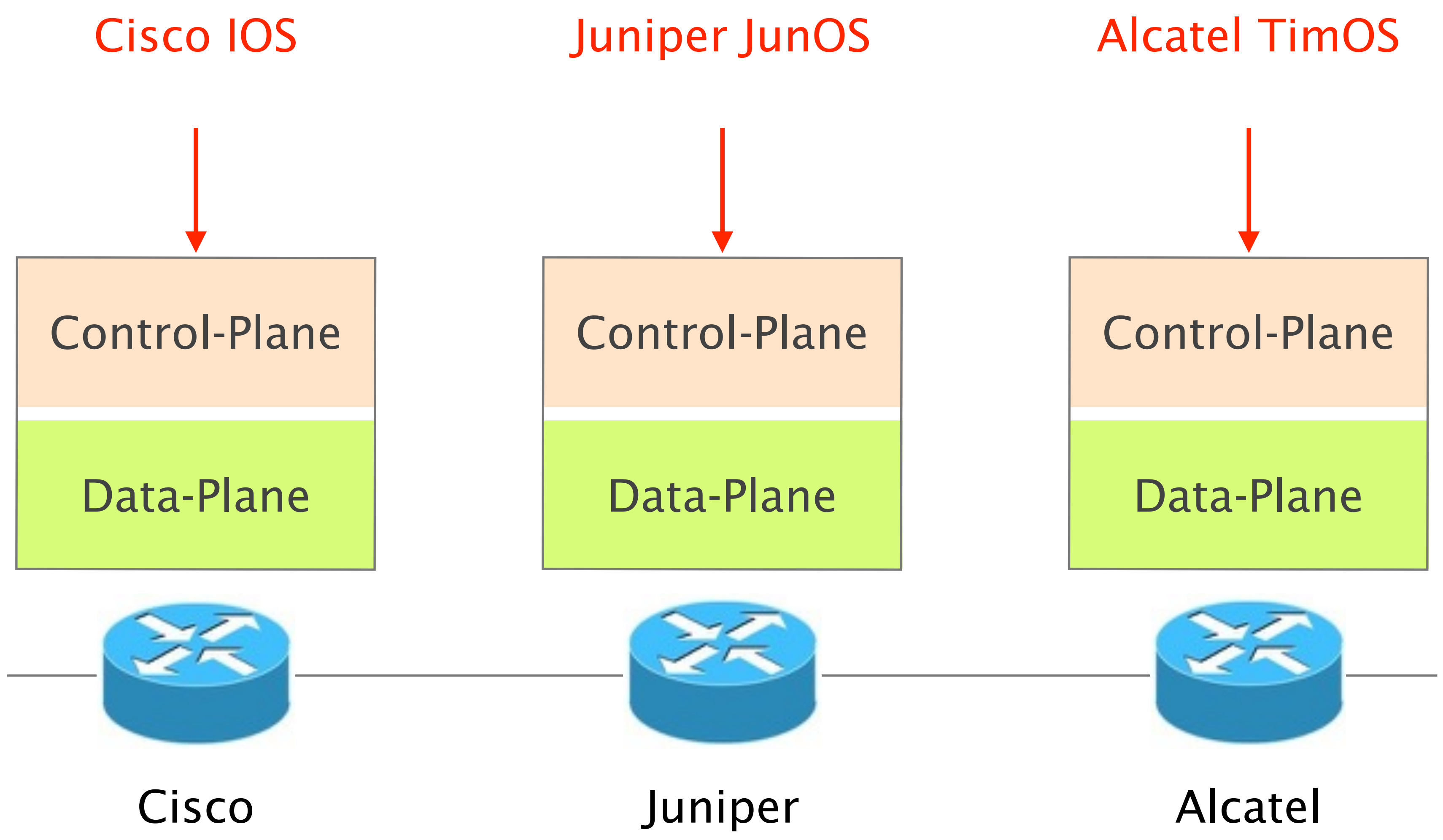
Supercharged
performance boost

Wouldn't it be great to manage
an existing network "à la SDN"?

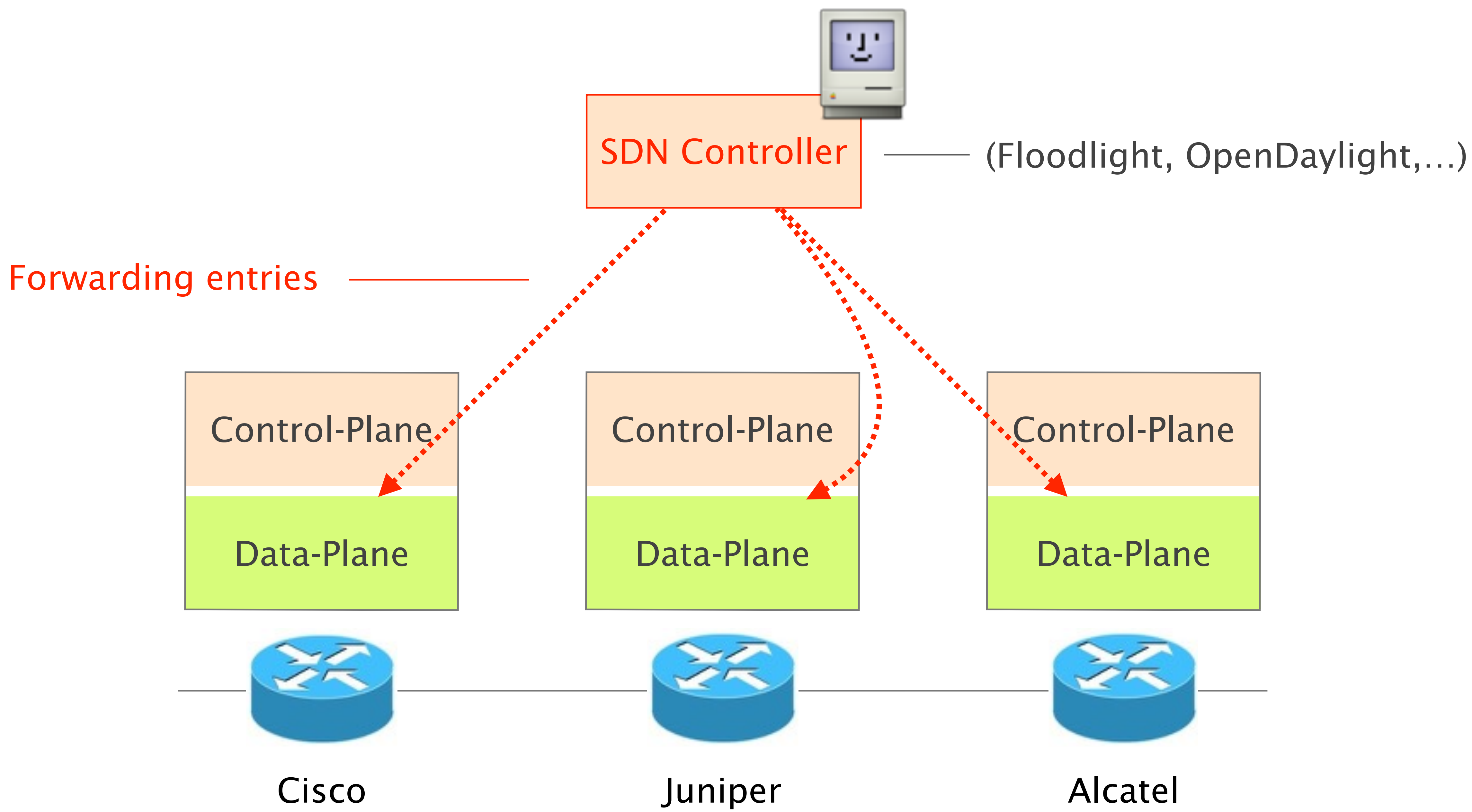
Wouldn’t it be great to manage
an existing network “à la SDN”?

what does it mean?

Instead of configuring a network using configuration "languages" …

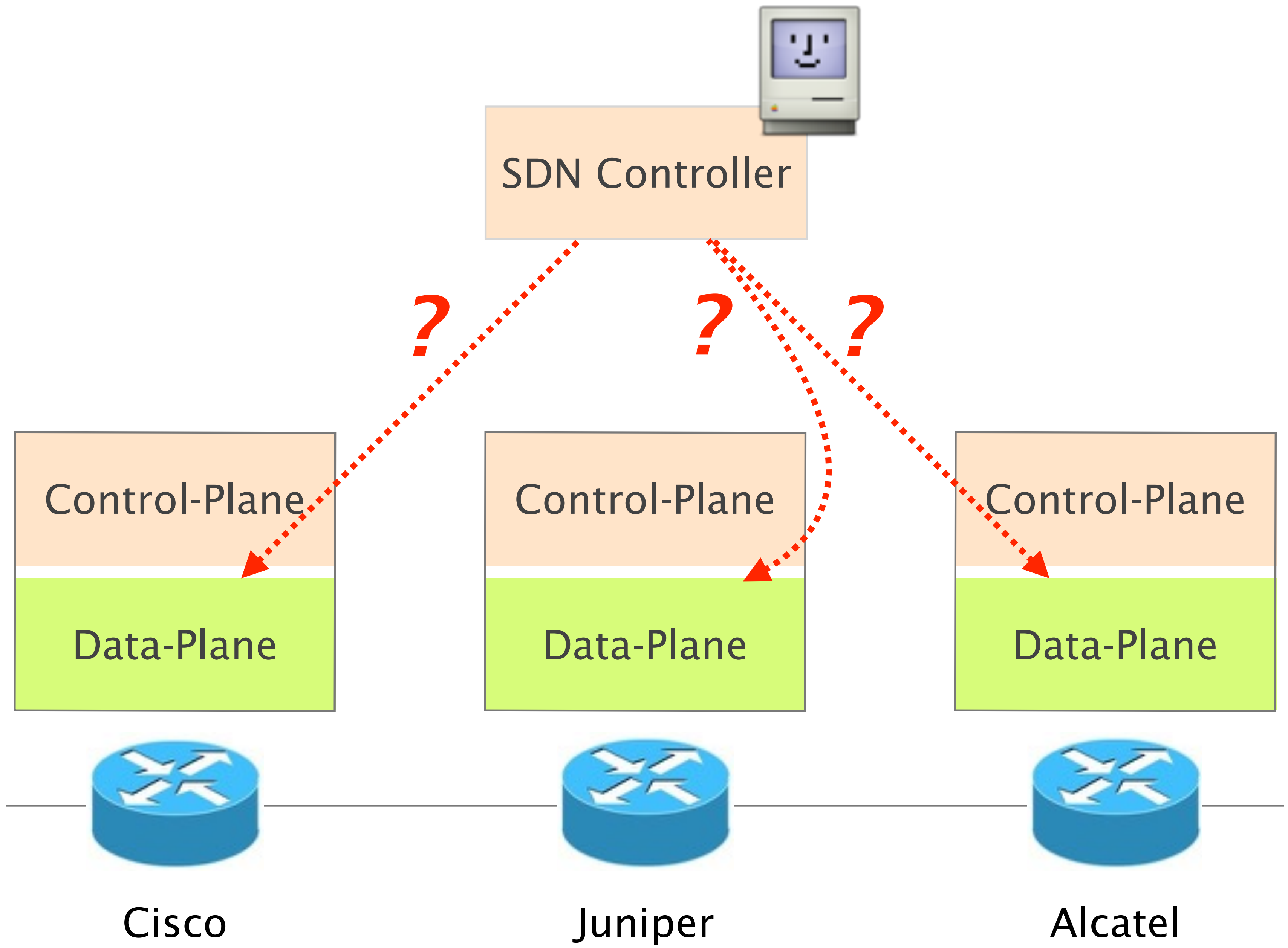


… program it from a central SDN controller



For that, we need an API

that *any* router can understand



Routing protocols are perfect candidates to act as such API

- messages are standardized

  routers must speak the same language

- behaviors are well-defined

  e.g., shortest-path routing

- implementations are widely available

  nearly all routers support OSPF

Fibbing

Fibbing

= lying

Fibbing

to **control** router's forwarding table

Central Control Over Distributed Routing

Joint work with: Stefano Vissicchio, Olivier Tilmans and Jennifer Rexford

1 **Fibbing**

lying made useful

2 **Expressivity**

any path, anywhere

3 **Scalability**

1 lie is better than 2

Central Control Over Distributed Routing

1 Fibbing

lying made useful

Expressivity

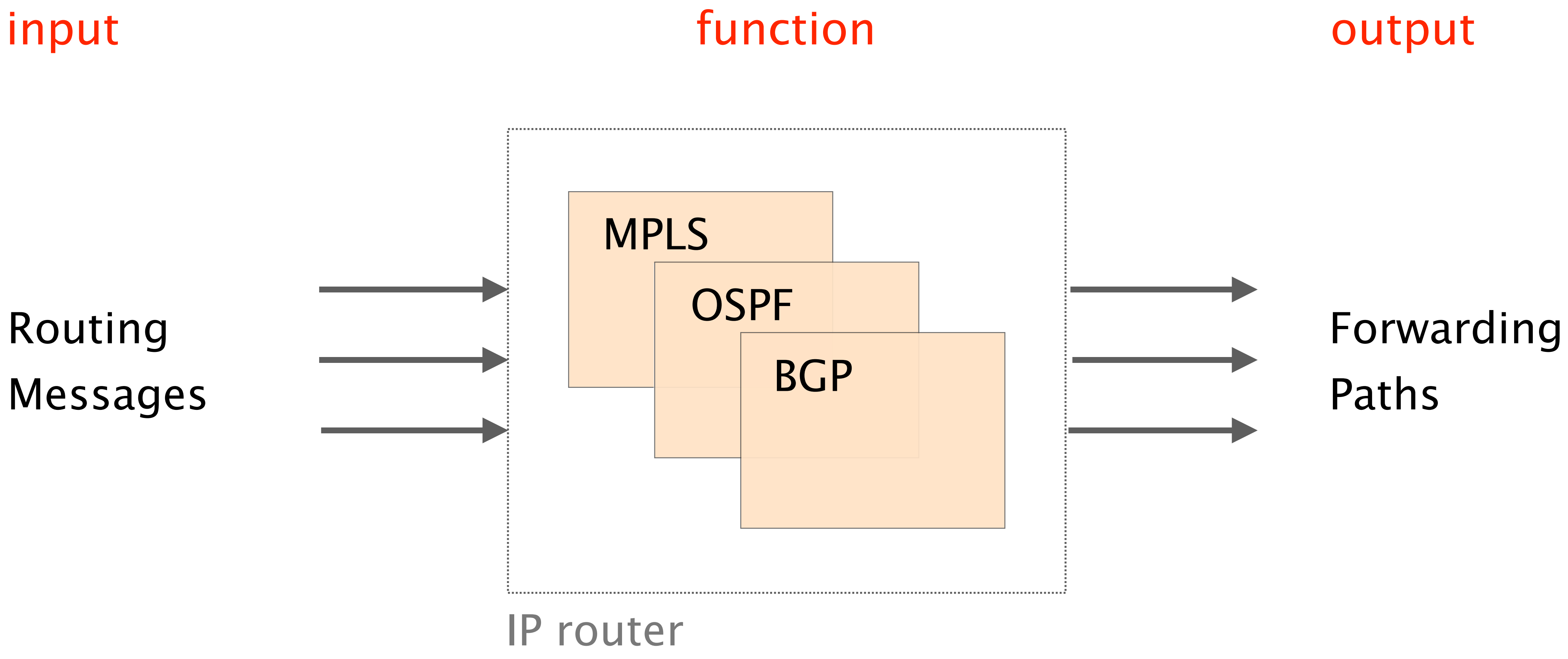
any path, anywhere

Scalability

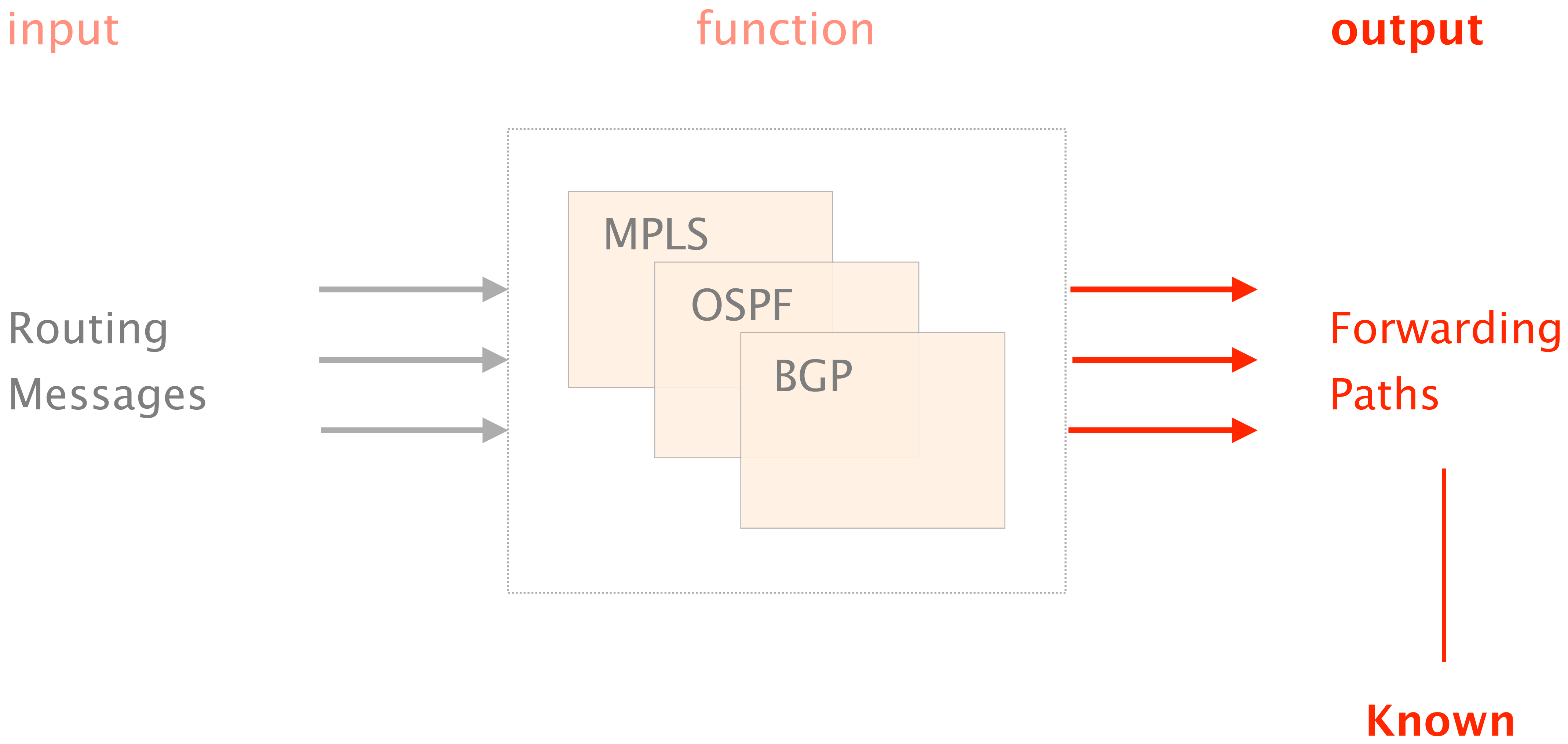
1 lie is better than 2

A router implements a function

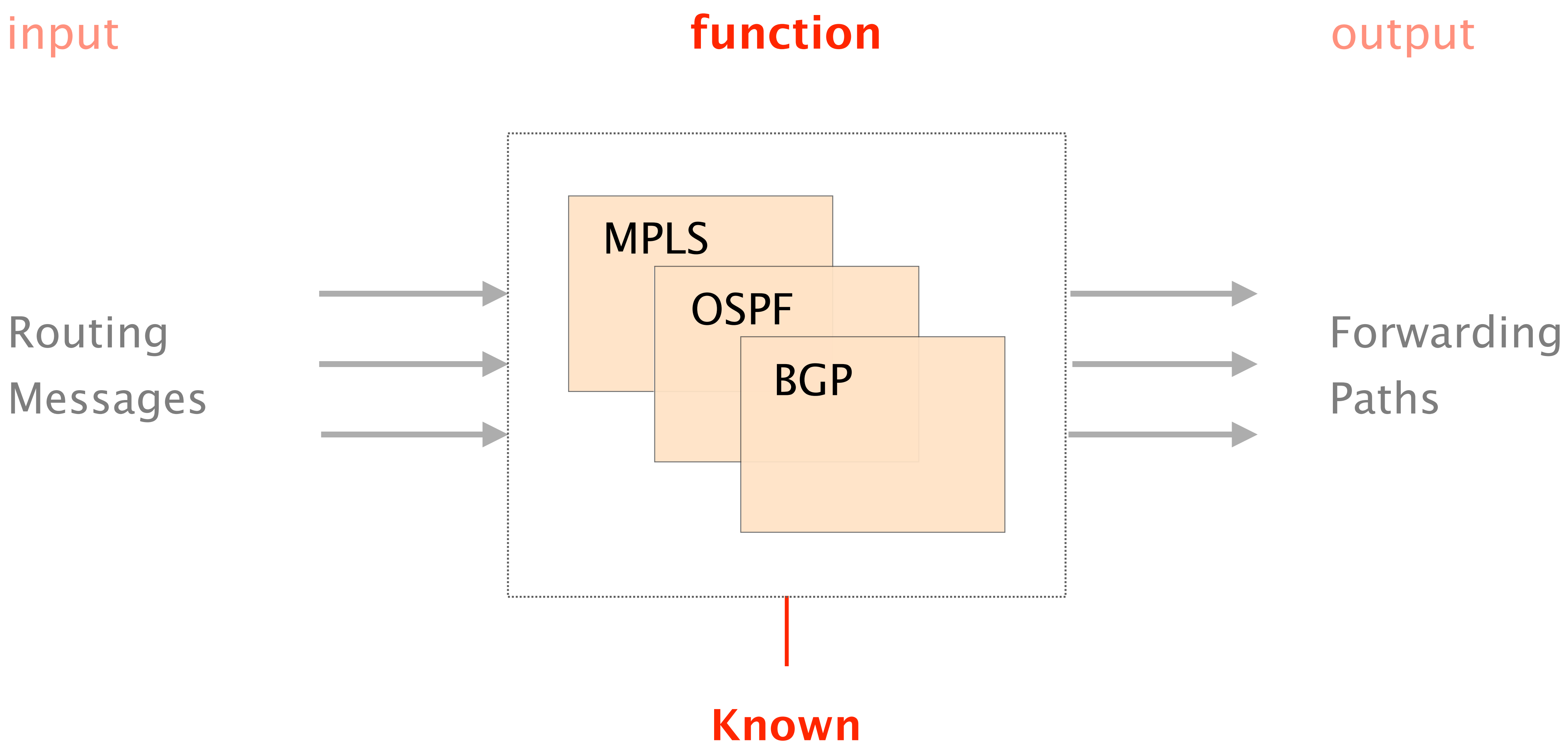
from **routing messages** to **forwarding paths**



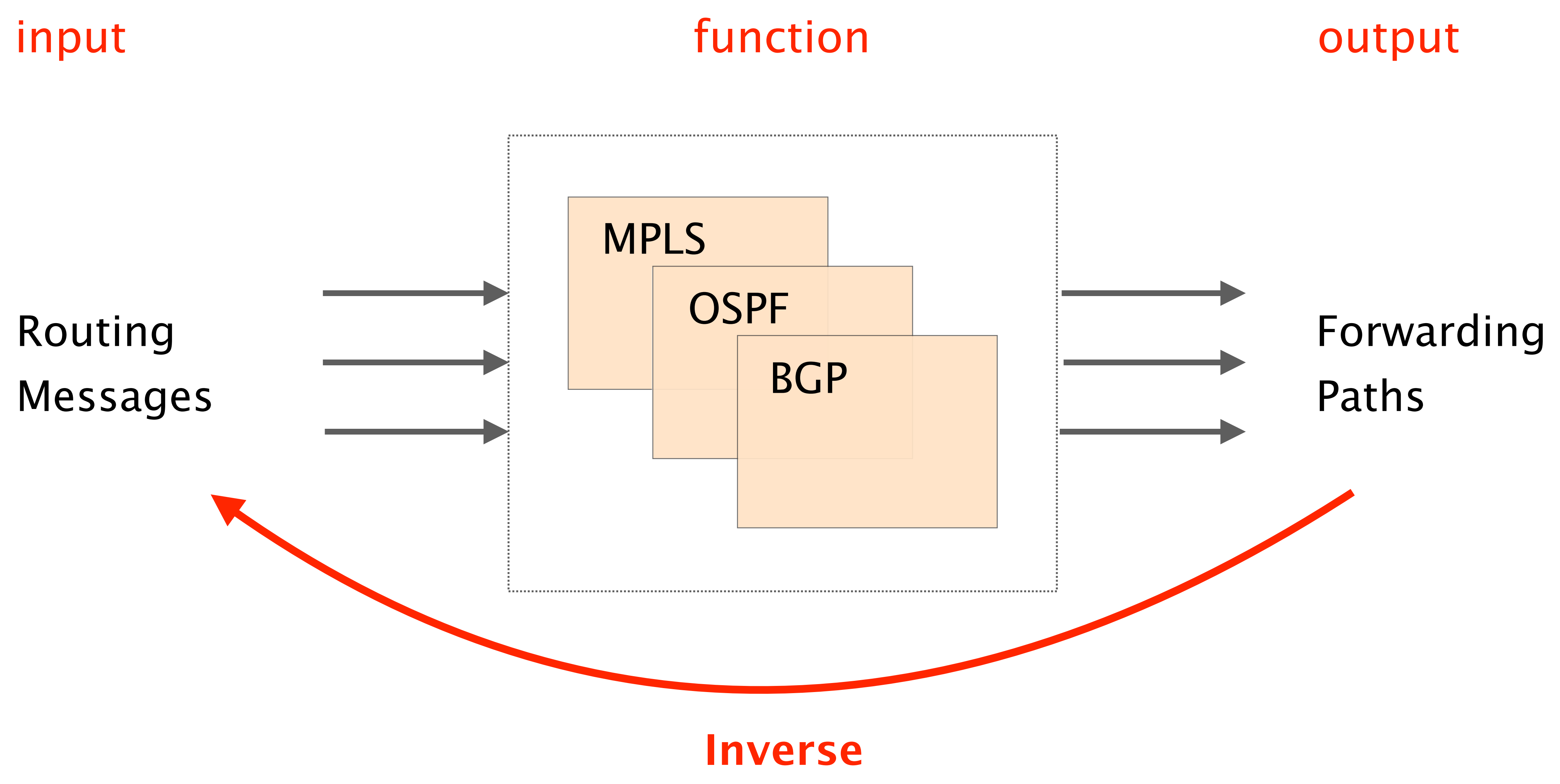
The forwarding paths are known,
provided by the operators or by the controller



The function is known, from the protocols' specification & the configuration



Given a path and a function, our framework computes corresponding routing messages by inverting the function



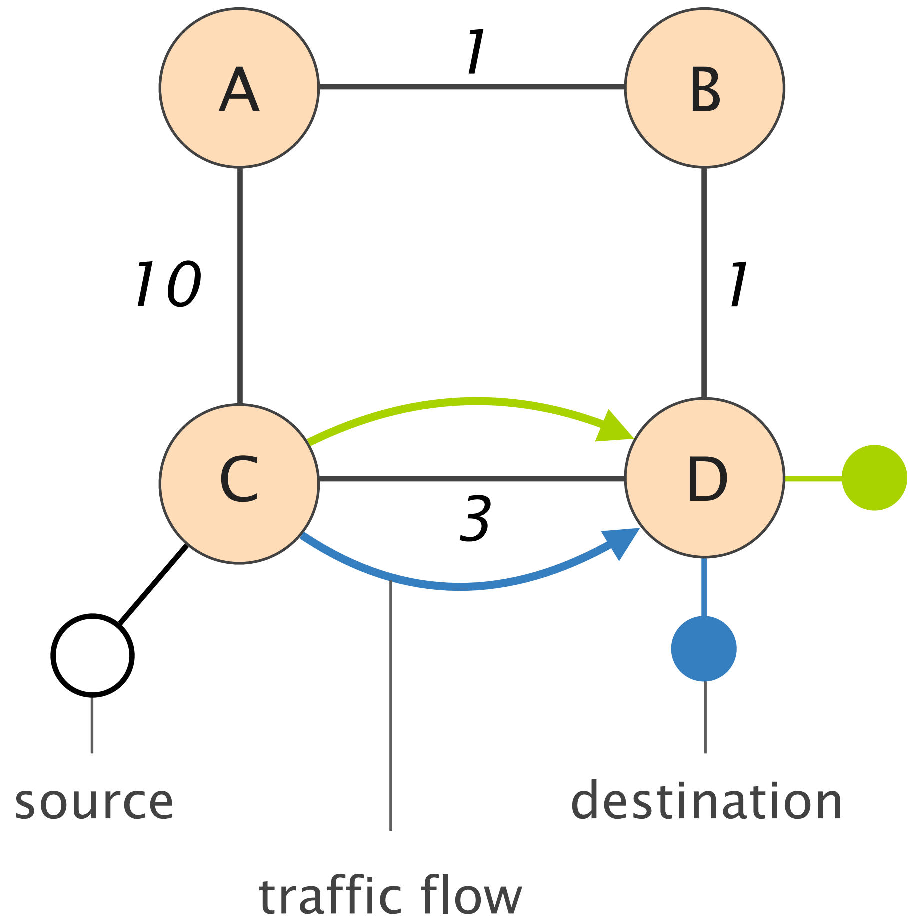
The type of input to be computed depends on the routing protocol

Protocol	Family	Algorithm/ Function	Router Input
IGP	Link-State	Dijkstra	Network graph
BGP	Path-Vector	Decision process	Routing paths

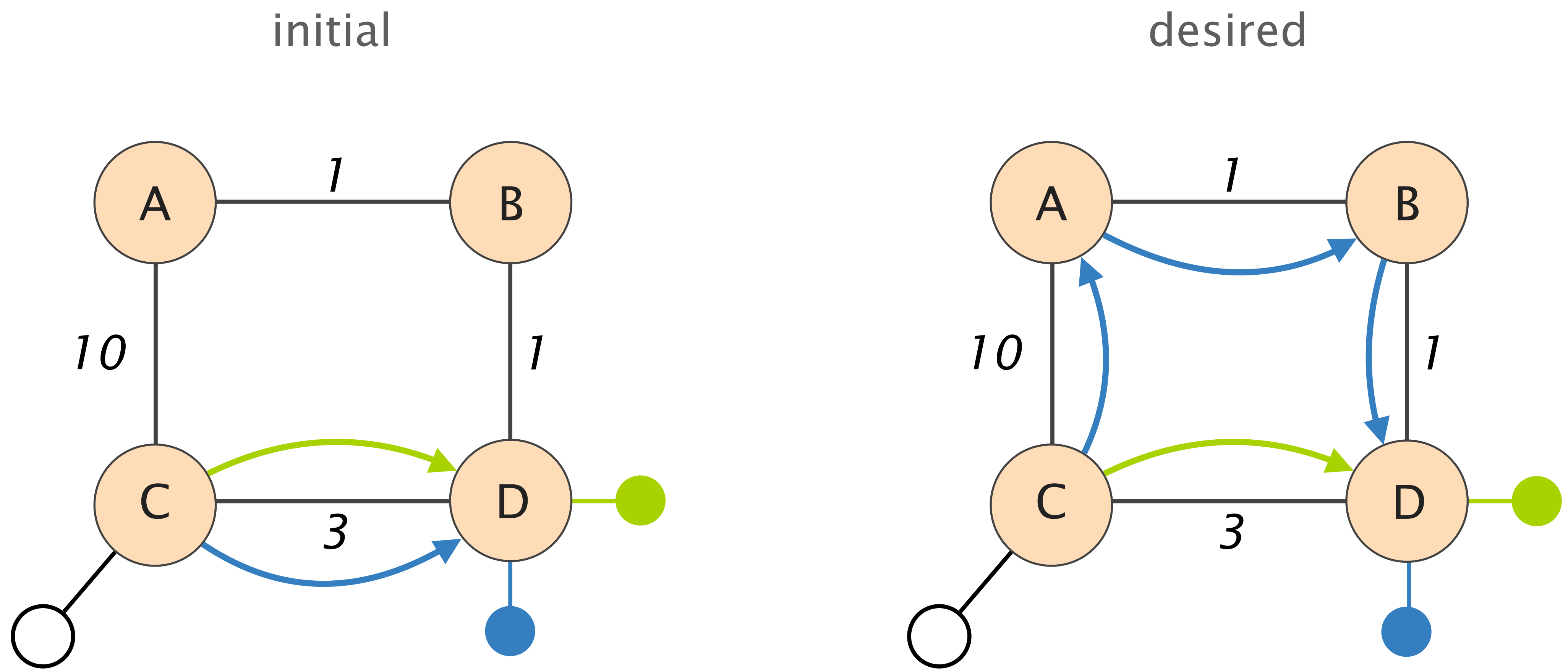
We focus on routers running link-state protocols that take the network graph as input and run Dijkstra

Protocol	Family	Algorithm/ Function	Router Input
IGP	Link-State	Dijkstra	Network graph
BGP	Path-Vector	Decision process	Routing paths

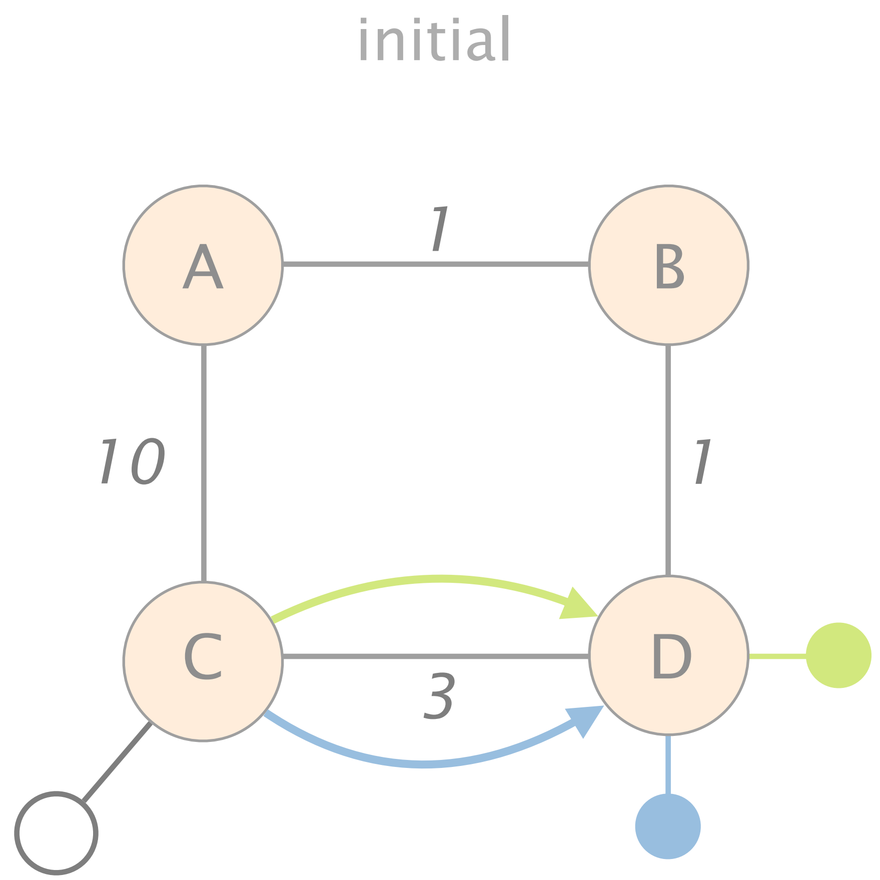
Consider this network where a source sends traffic to 2 destinations

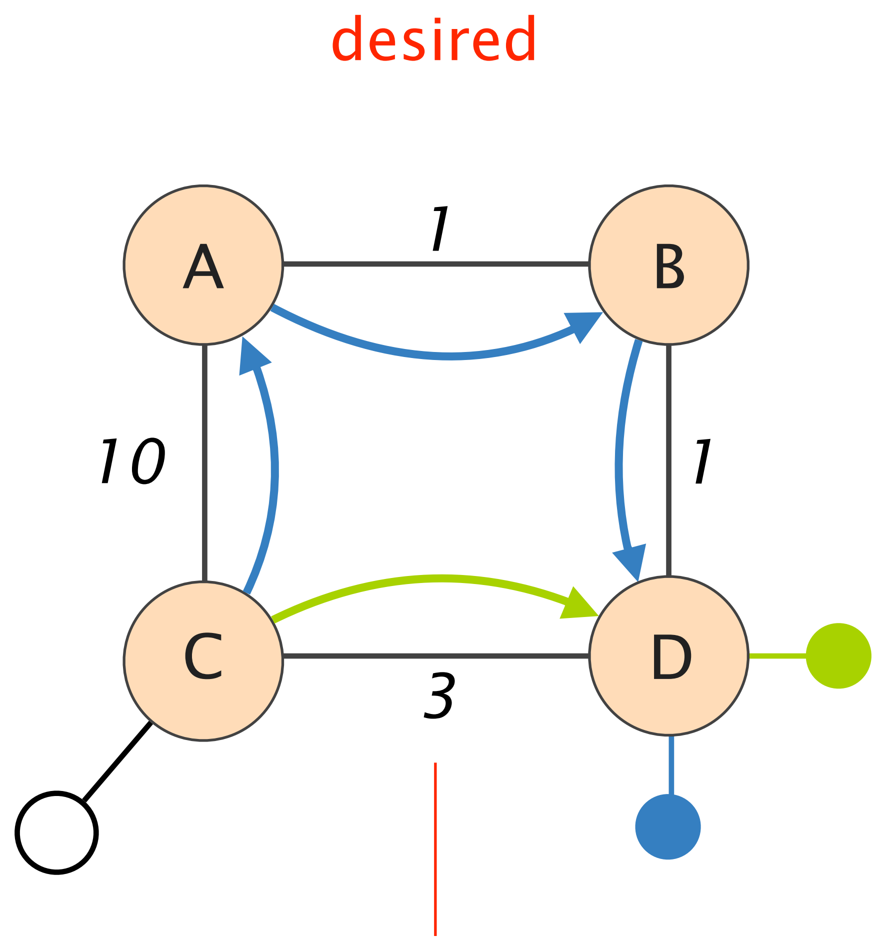


As congestion appears, the operator wants to shift away one flow from (C,D)



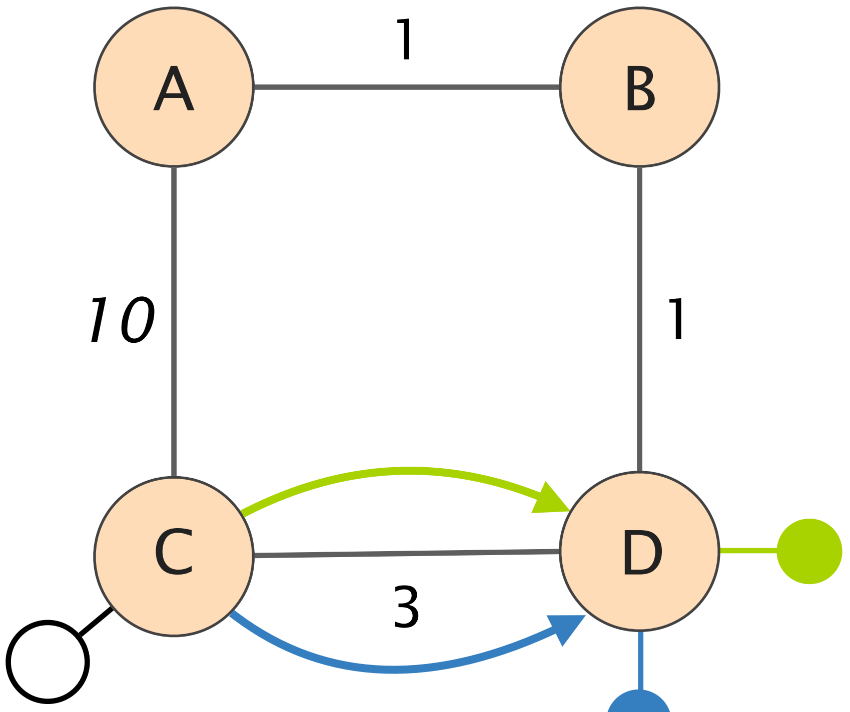
Moving only one flow is impossible though as both destinations are connected to D



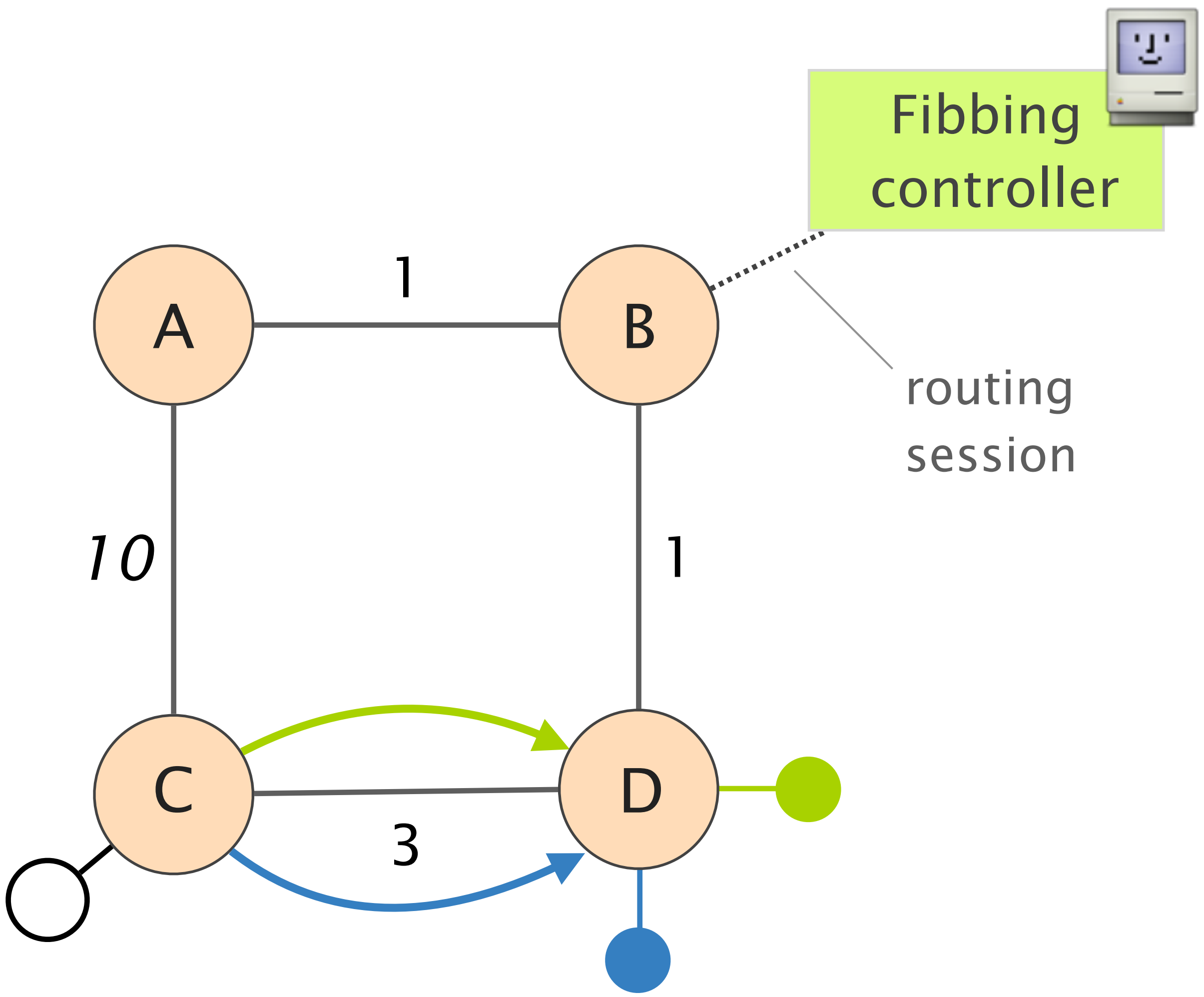


impossible to achieve by reweighing the links

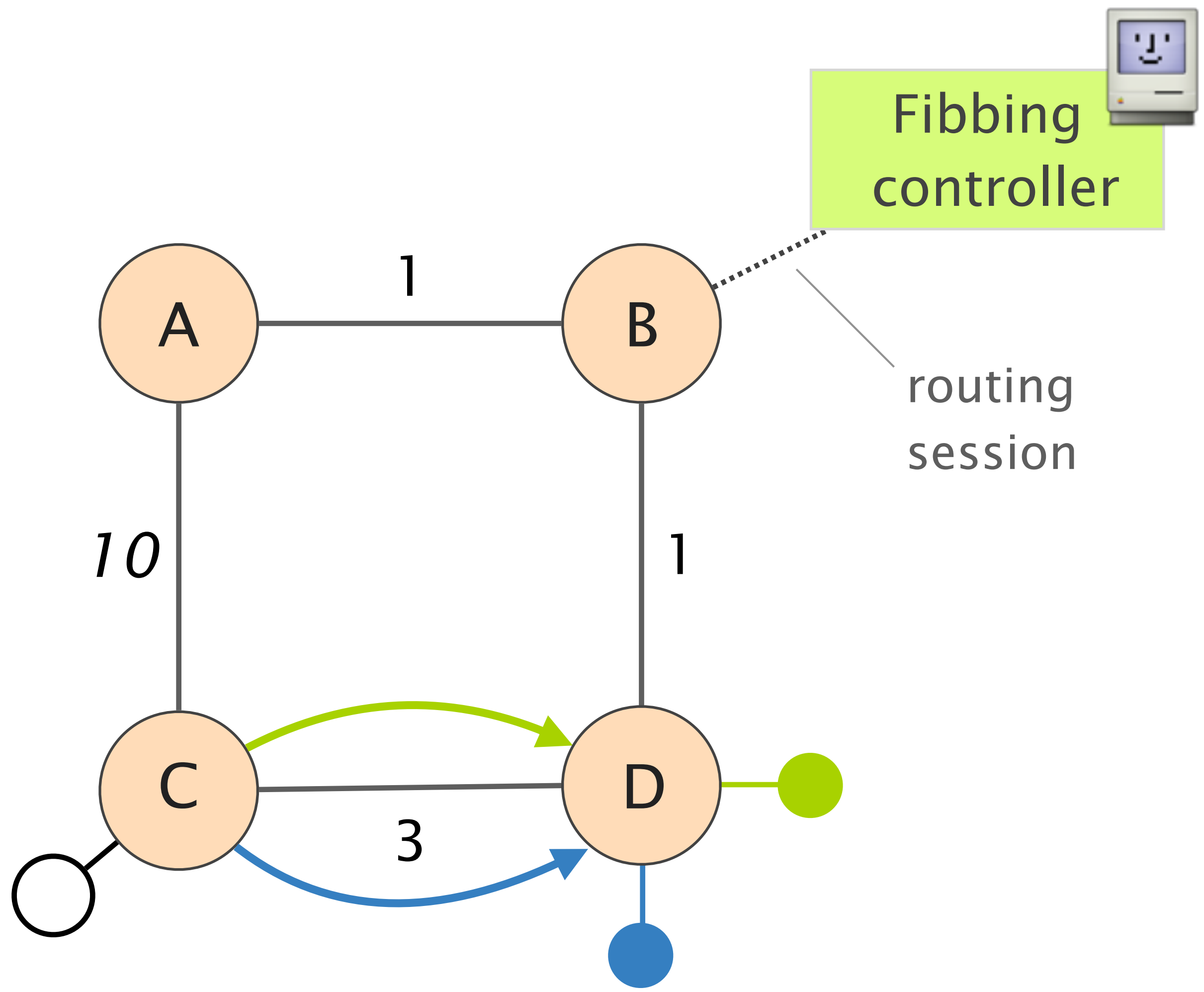
Let’s lie to the router



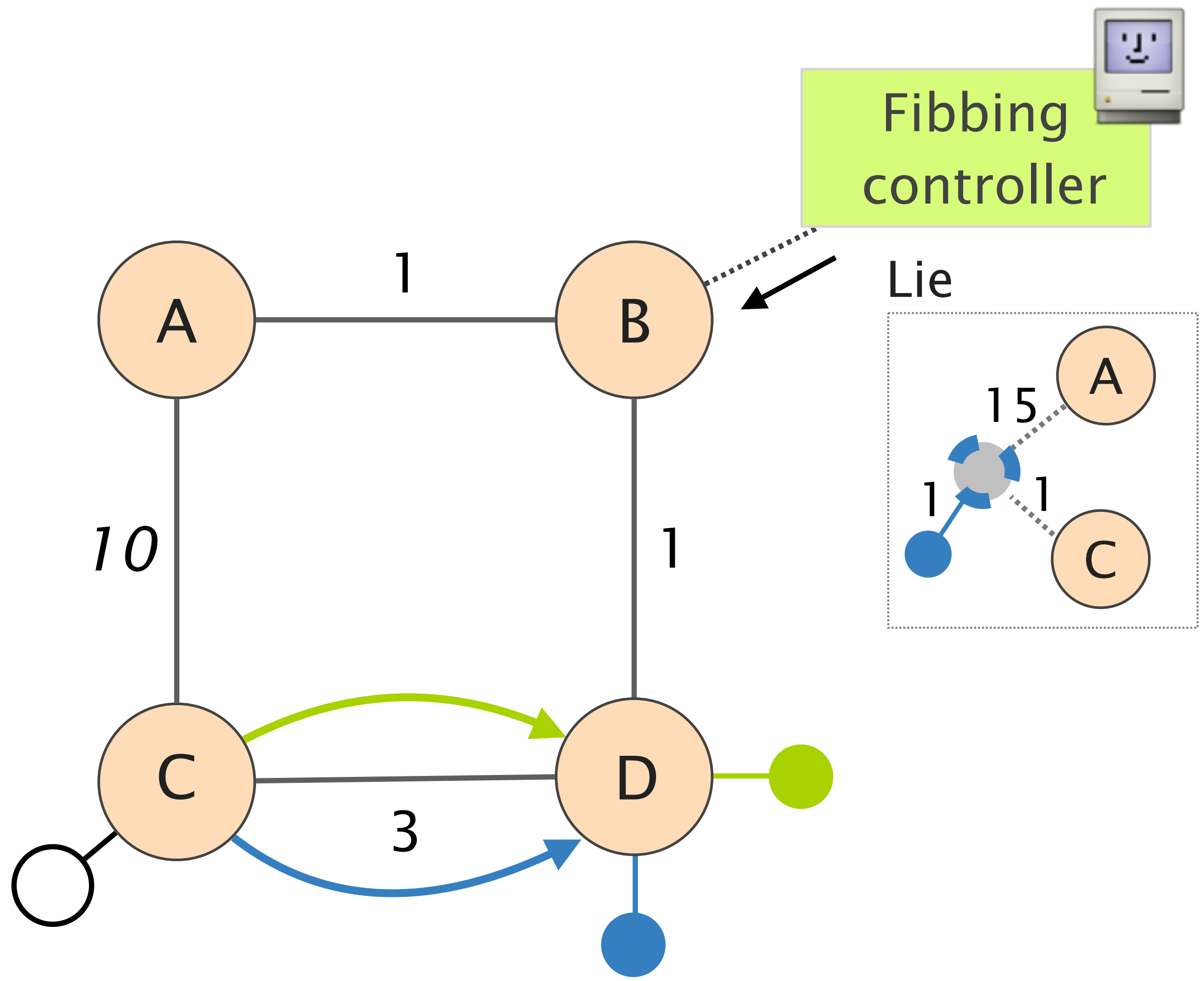
Let's lie to the router



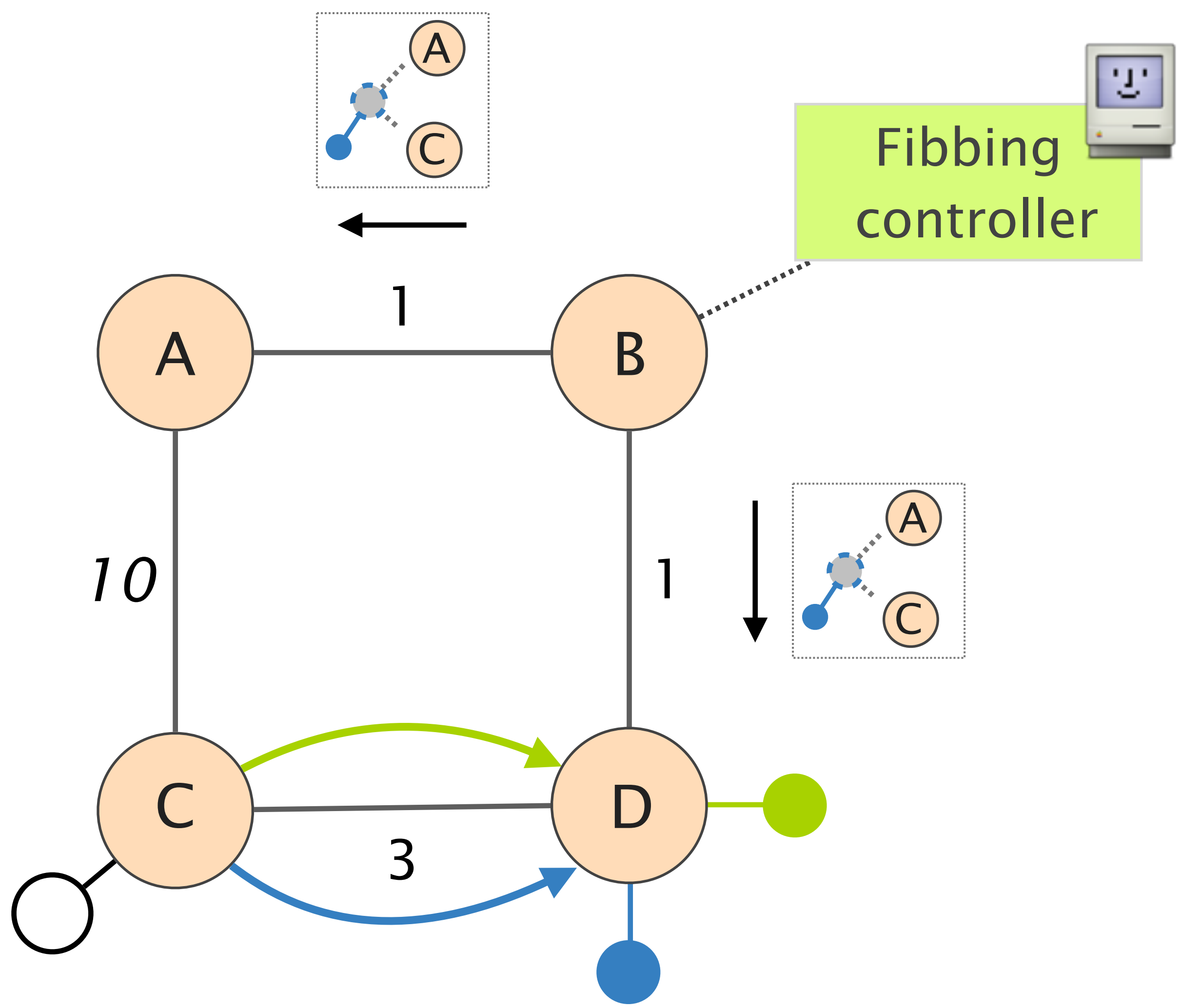
Let's **lie to the router,** by injecting fake nodes, links and destinations



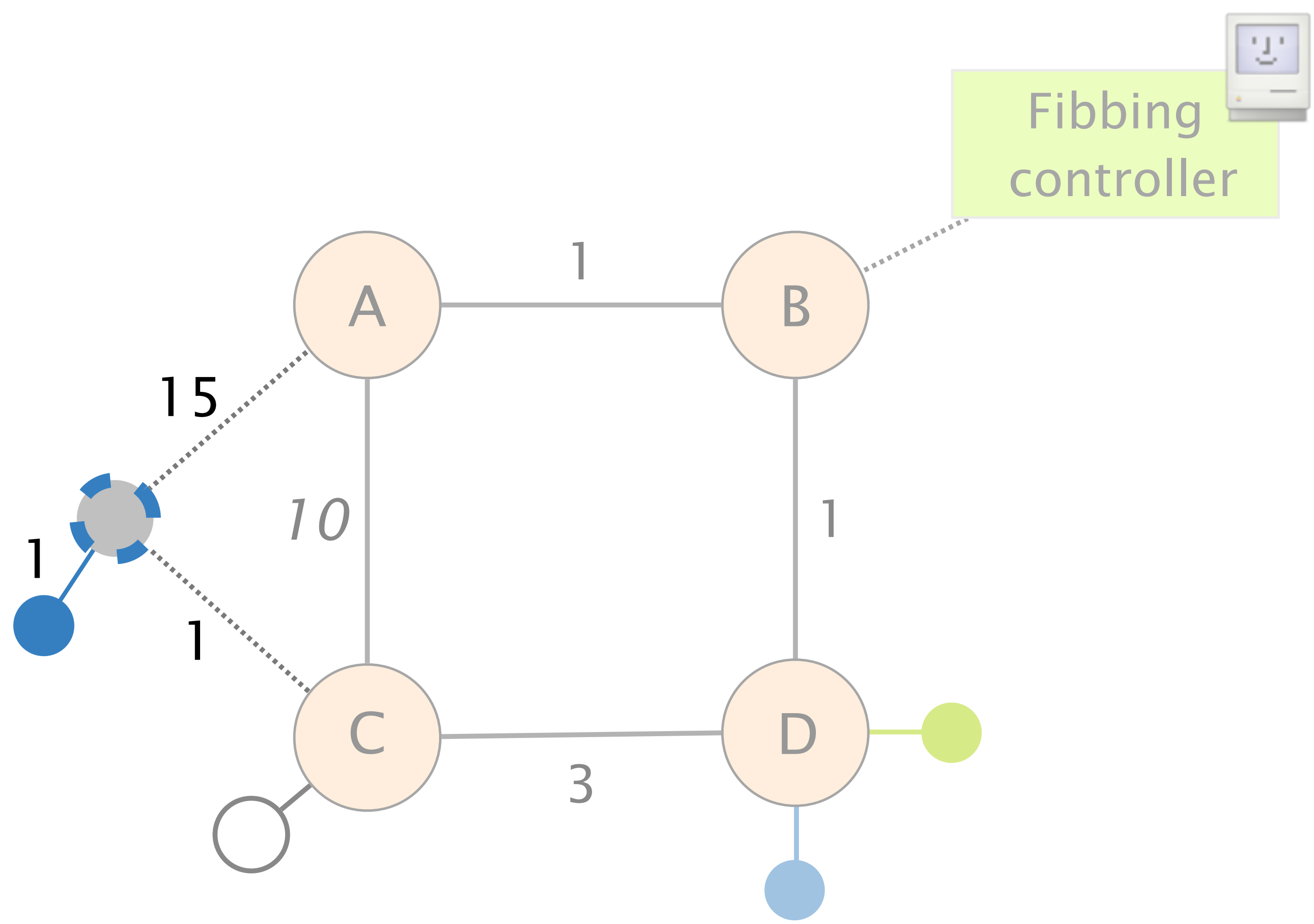
Let's **lie to the router,** by injecting fake nodes, links and destinations



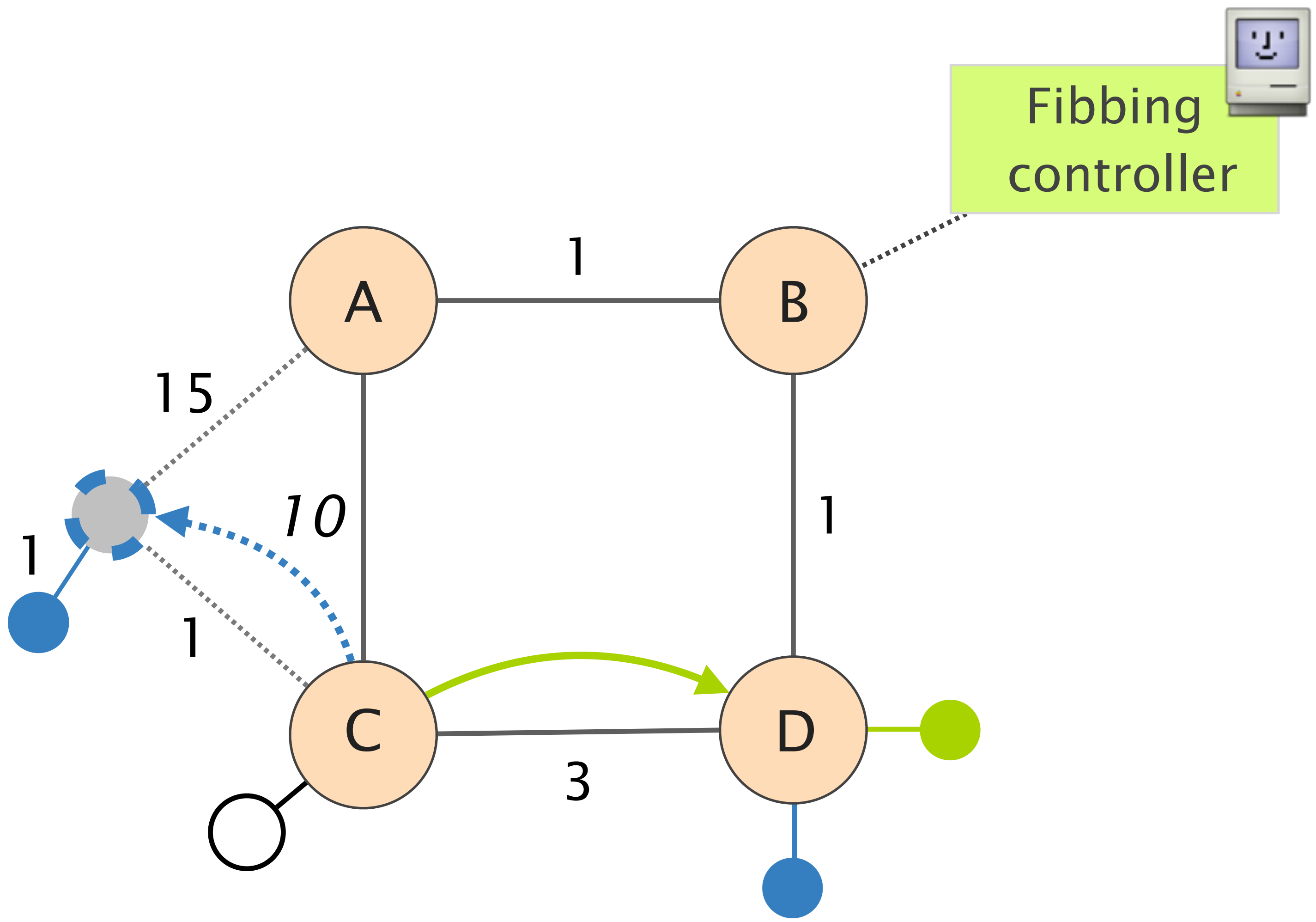
Lies are propagated network-wide by the protocol



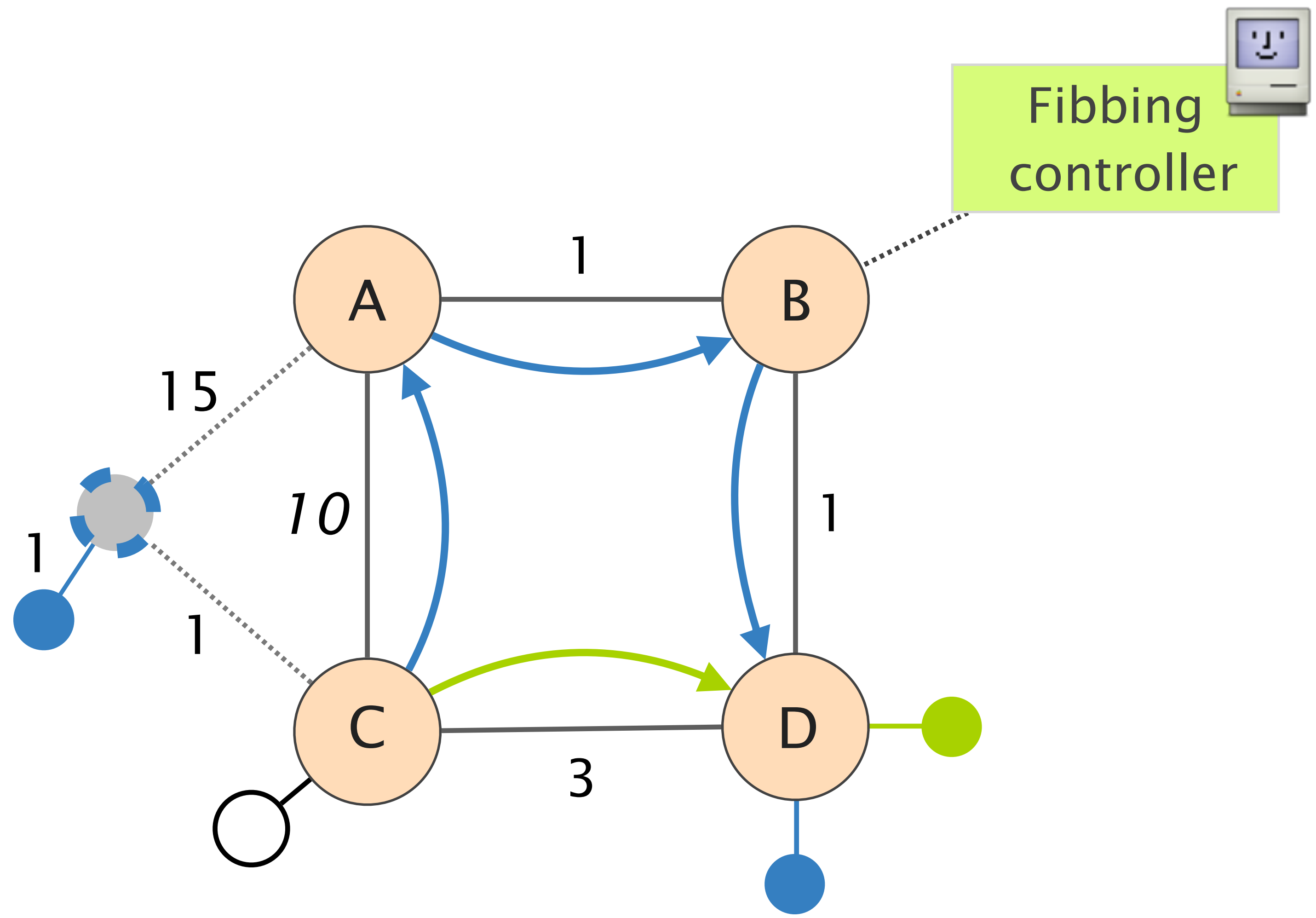
After the injection, this is the topology seen by all routers, on which they compute Dijkstra



Now, C prefers the virtual node (cost 2) to reach the blue destination…



As the virtual node does not really exist,
actual traffic is *physically* sent to A

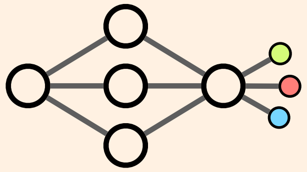


Fibbing

workflow

Fibbing starts from the operators requirements and a up-to-date representation of the network

path reqs.

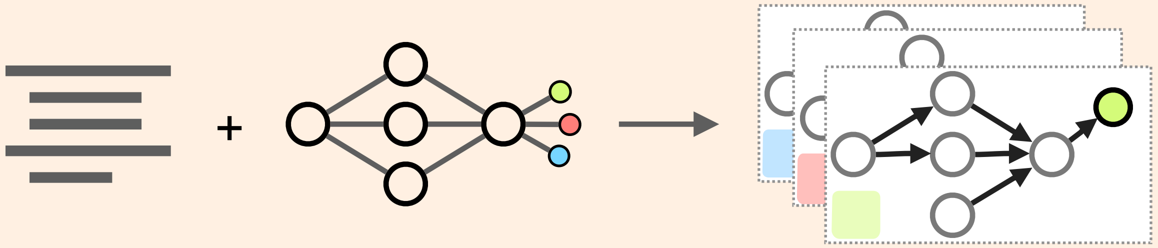


network graph

Out of these,

the **compilation** stage produces DAGs

Compilation

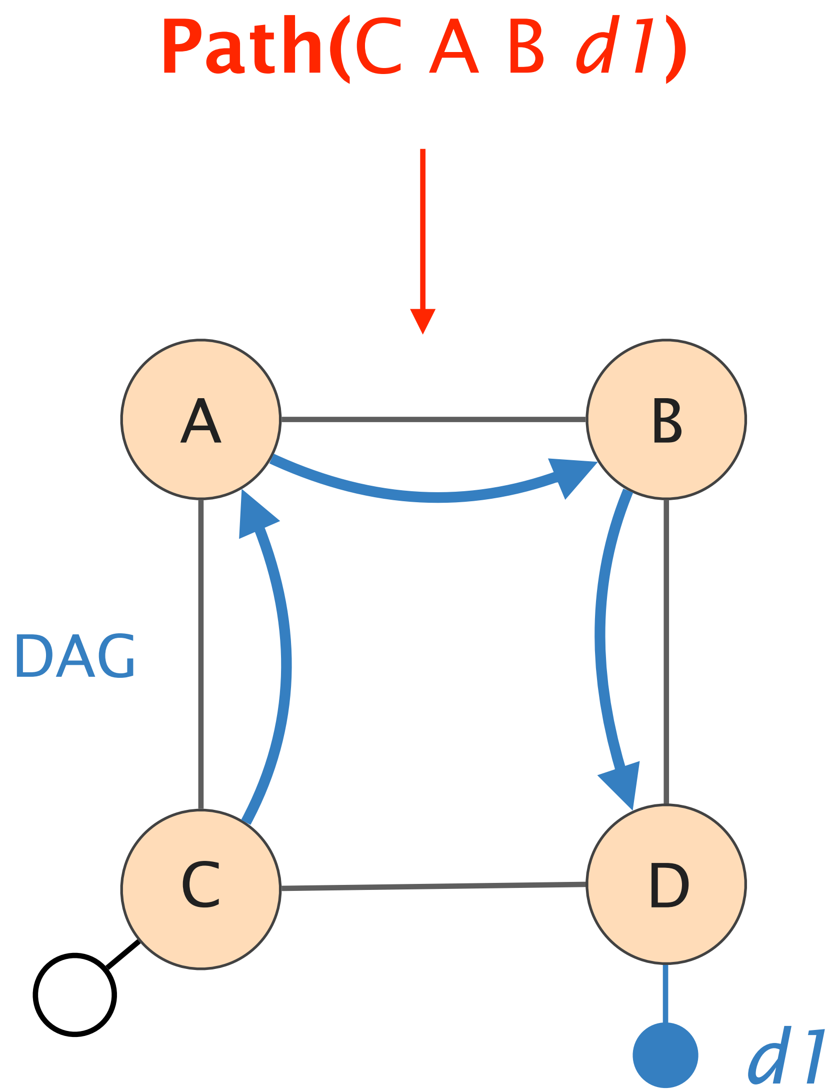


path
reqs.

network
graph

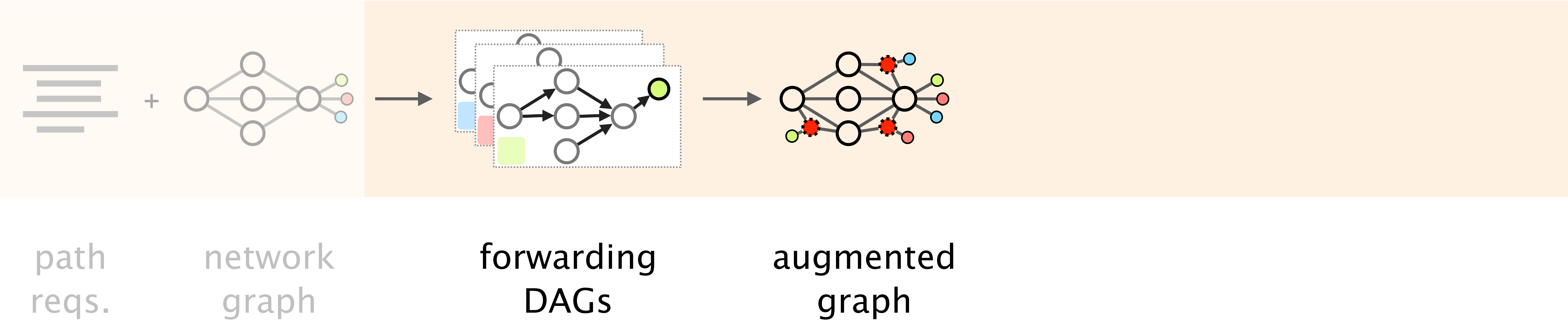
forwarding
DAGs

Forwarding graphs (DAGs) are compiled from high-level requirements

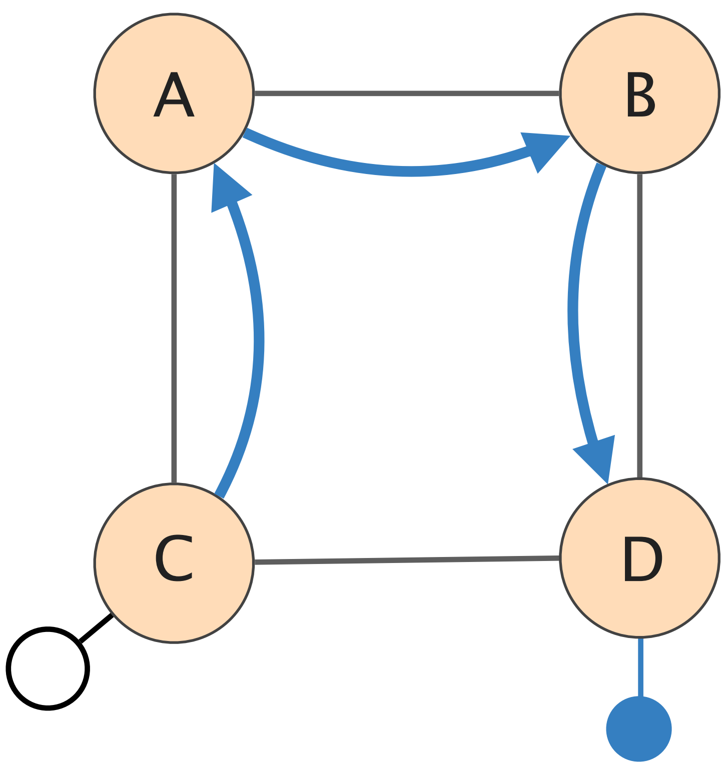


The **augmentation** stage augments the network graph with lies to implement each DAG

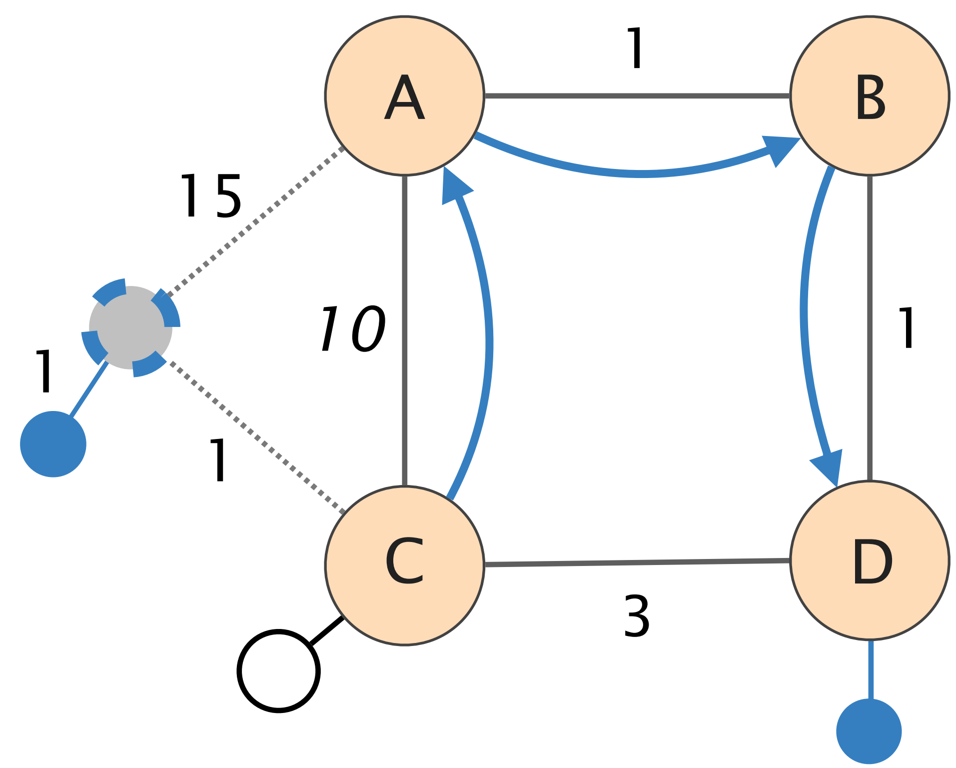
Augmentation



The **augmentation** stage augments the network graph with lies to implement each DAG

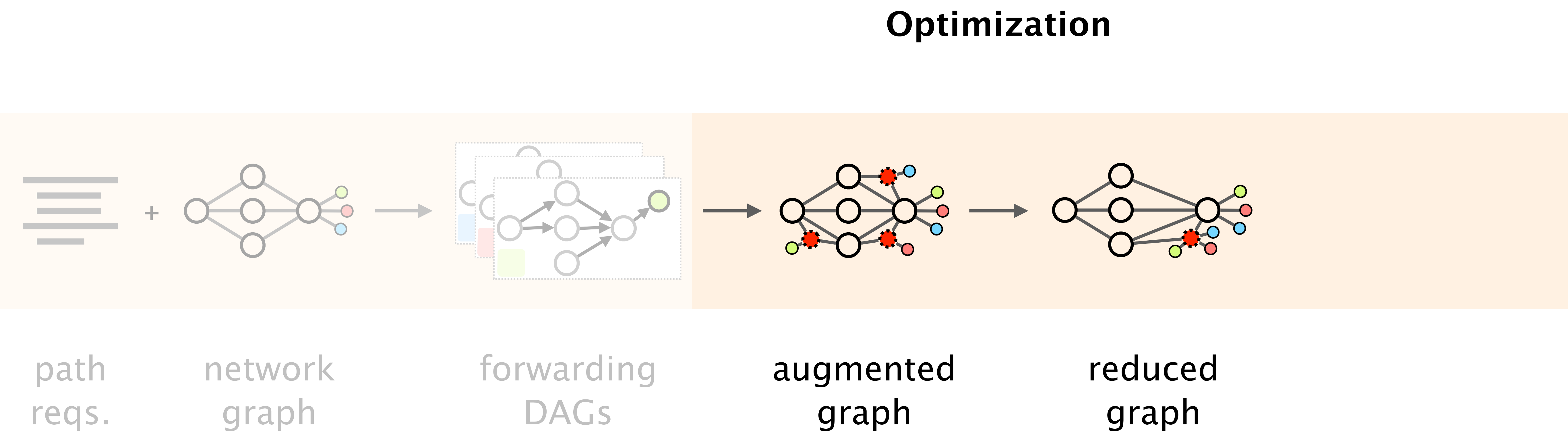


Compilation output

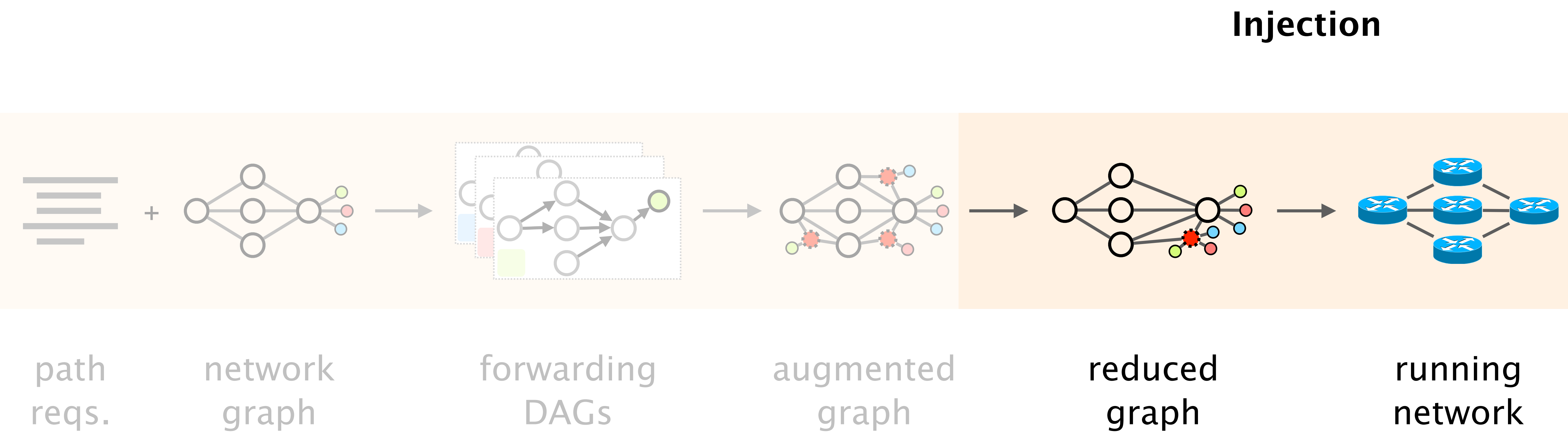


Augmentation output

The **optimization** stage reduces the amount of lies necessary



The **injection** stage injects
the lies in the production network



Central Control Over Distributed Routing

Fibbing

lying made useful

2 Expressivity

any path, anywhere

Scalability

1 lie is better than 2

Fibbing is powerful

Fibbing is powerful

Theorem | Fibbing can program any set of non-contradictory paths

Fibbing is powerful

Theorem

Fibbing can program
any set of non-contradictory paths

Fibbing is powerful

Theorem

Fibbing can program any set of non-contradictory paths

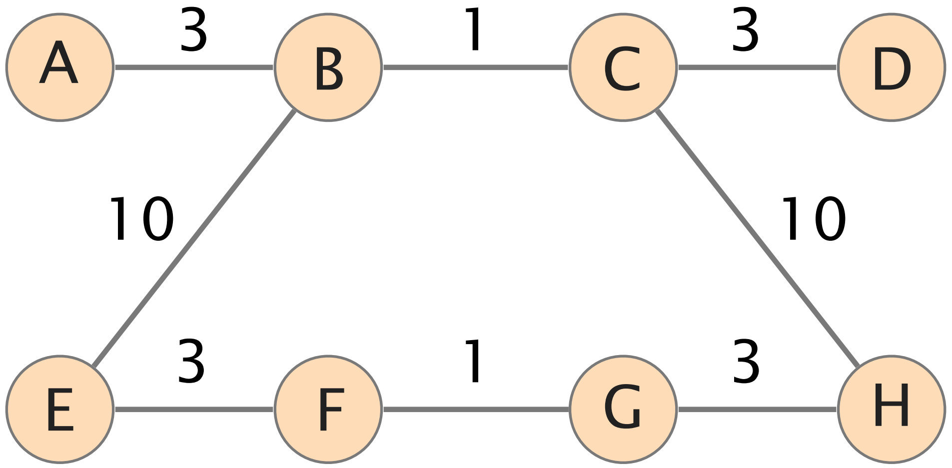
- any path is loop-free

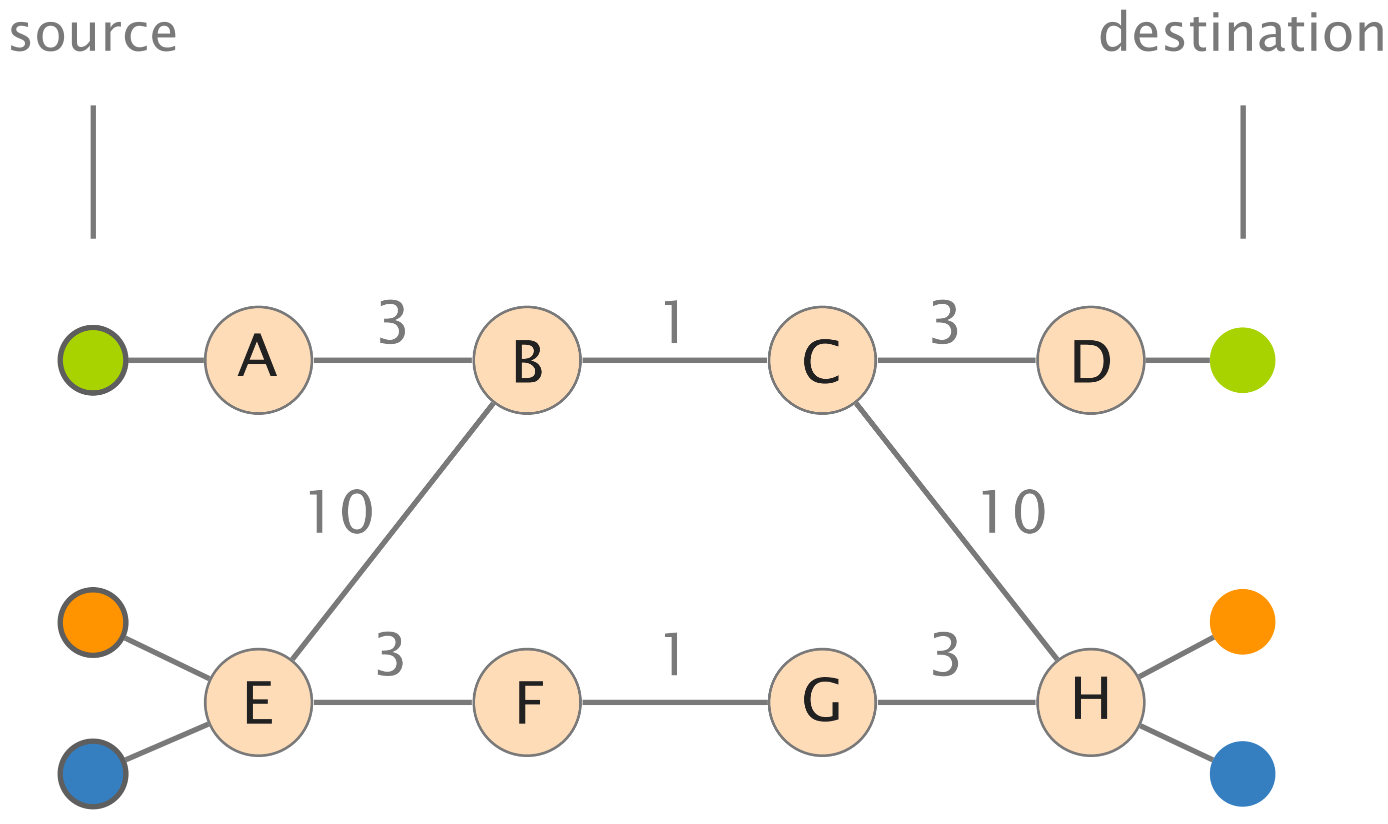
  (*e.g.*, [s1, a, b, a, d] is not possible)

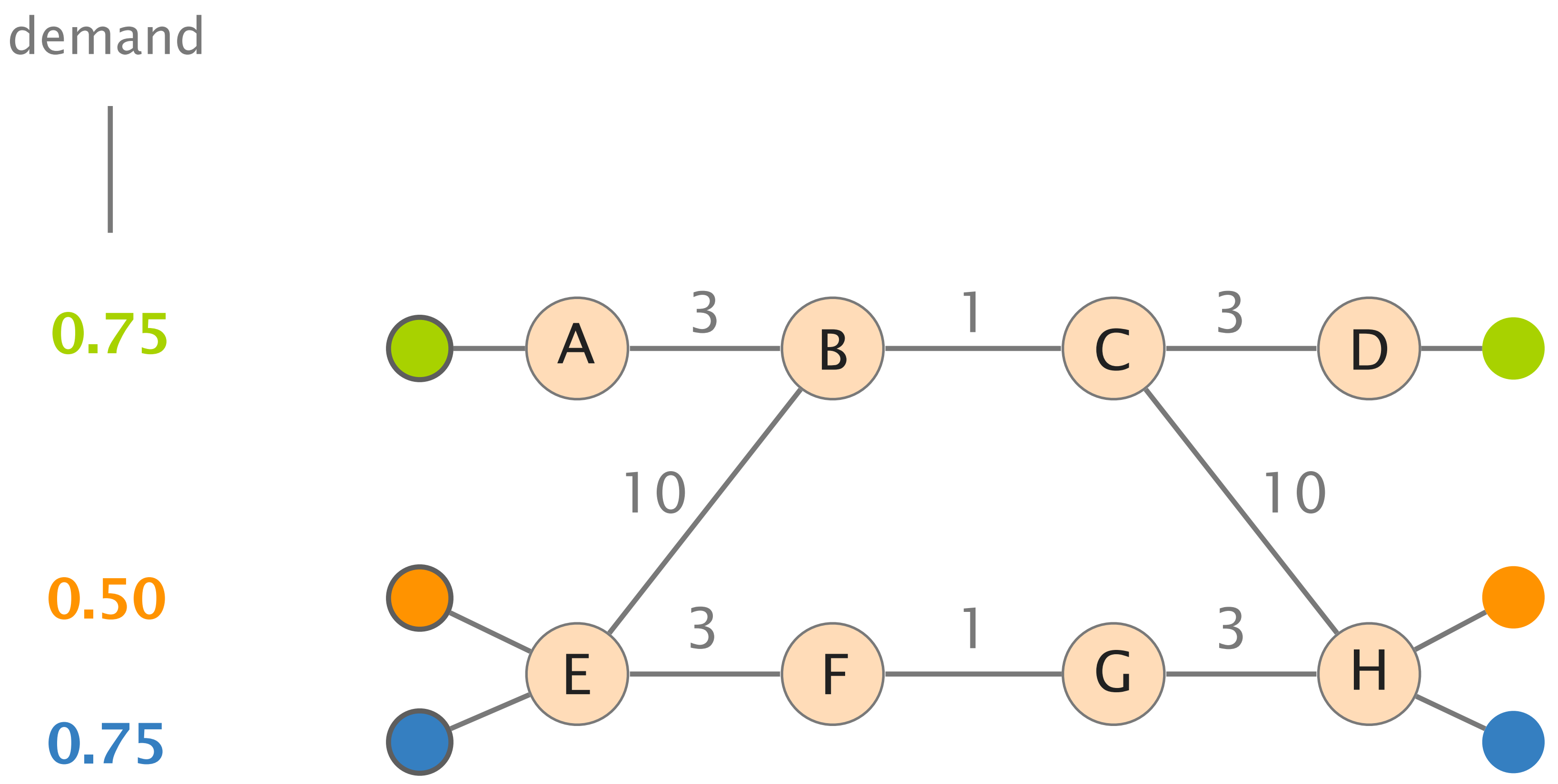
- paths are consistent

  (*e.g.* [s1, a, b, d] and [s2, b, a, d] are inconsistent)

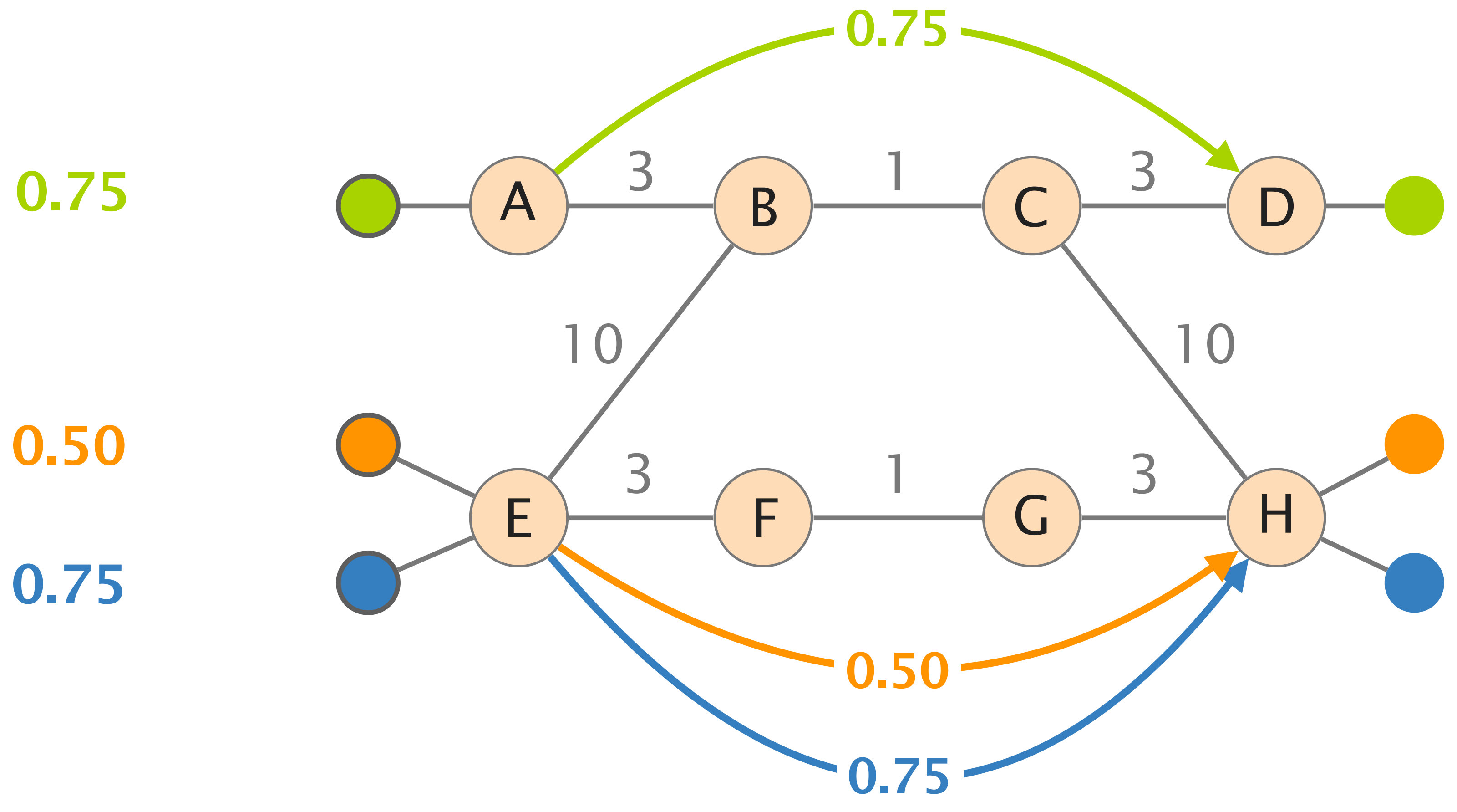
Fibbing can load-balance traffic on multiple paths




source
destination
A
B
C
D
E
F
G
H
3
1
3
10
10
3
1
3

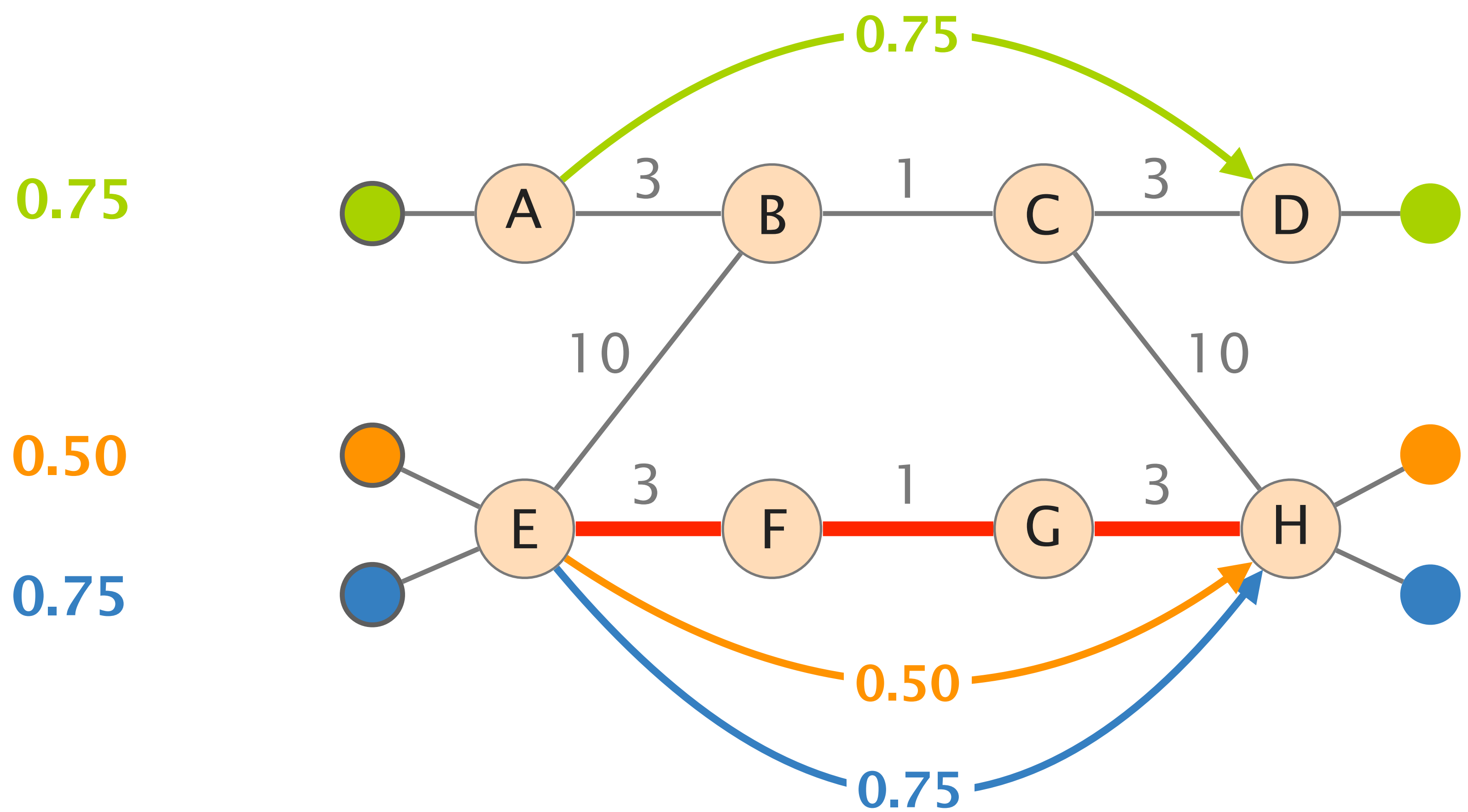


Links have a capacity of 1

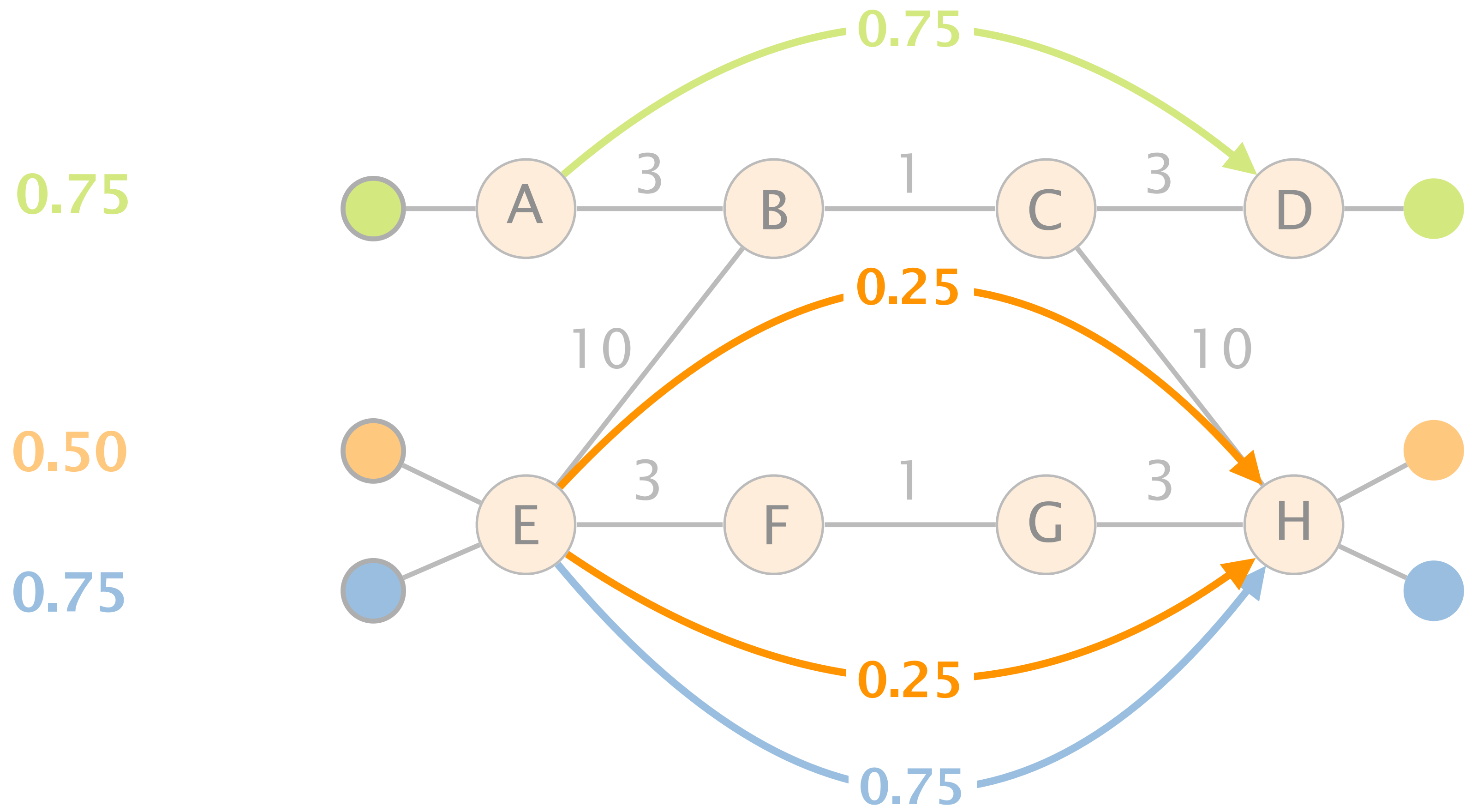


Links have a capacity of 1

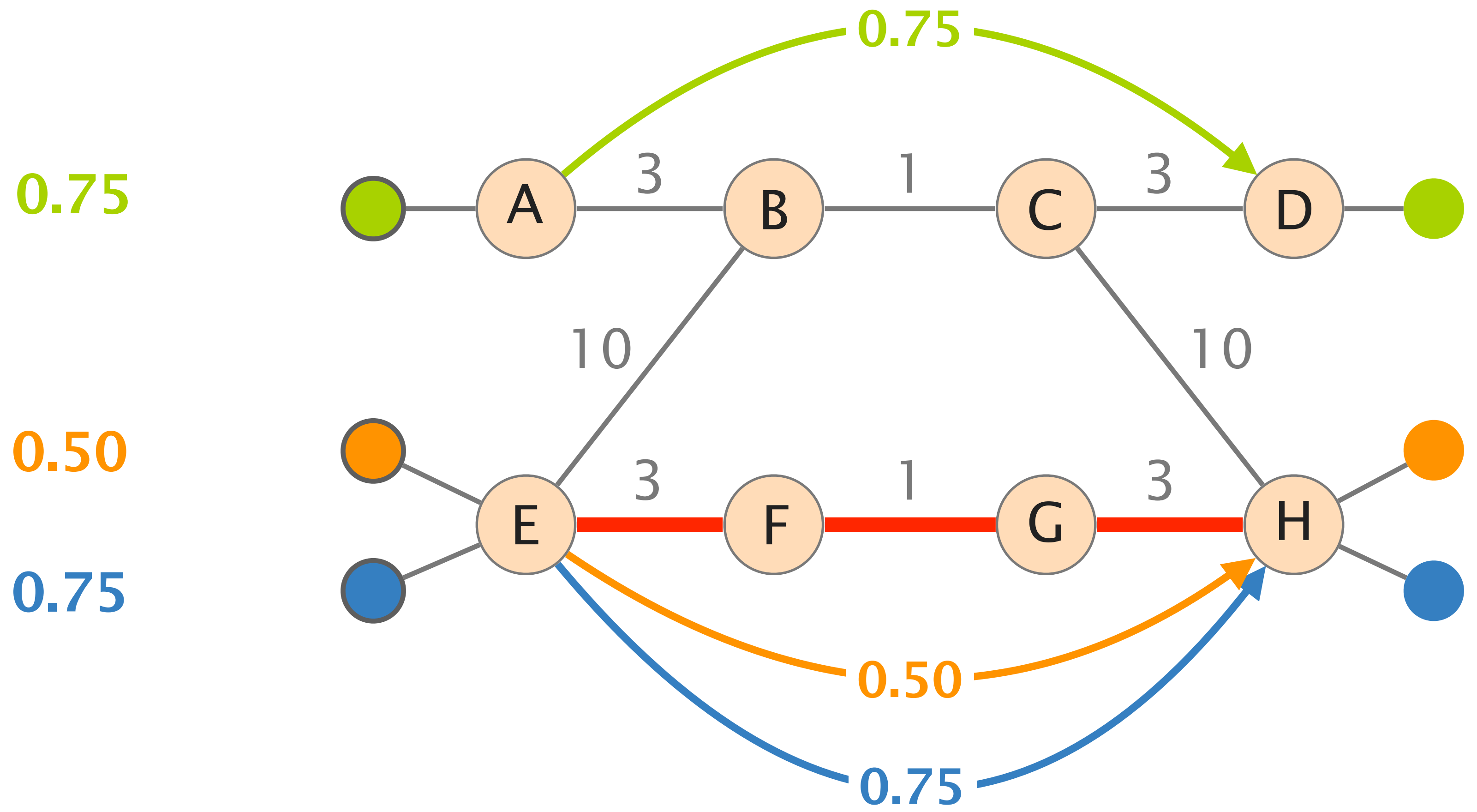
With such demands and forwarding, the lower path is congested (1.25)



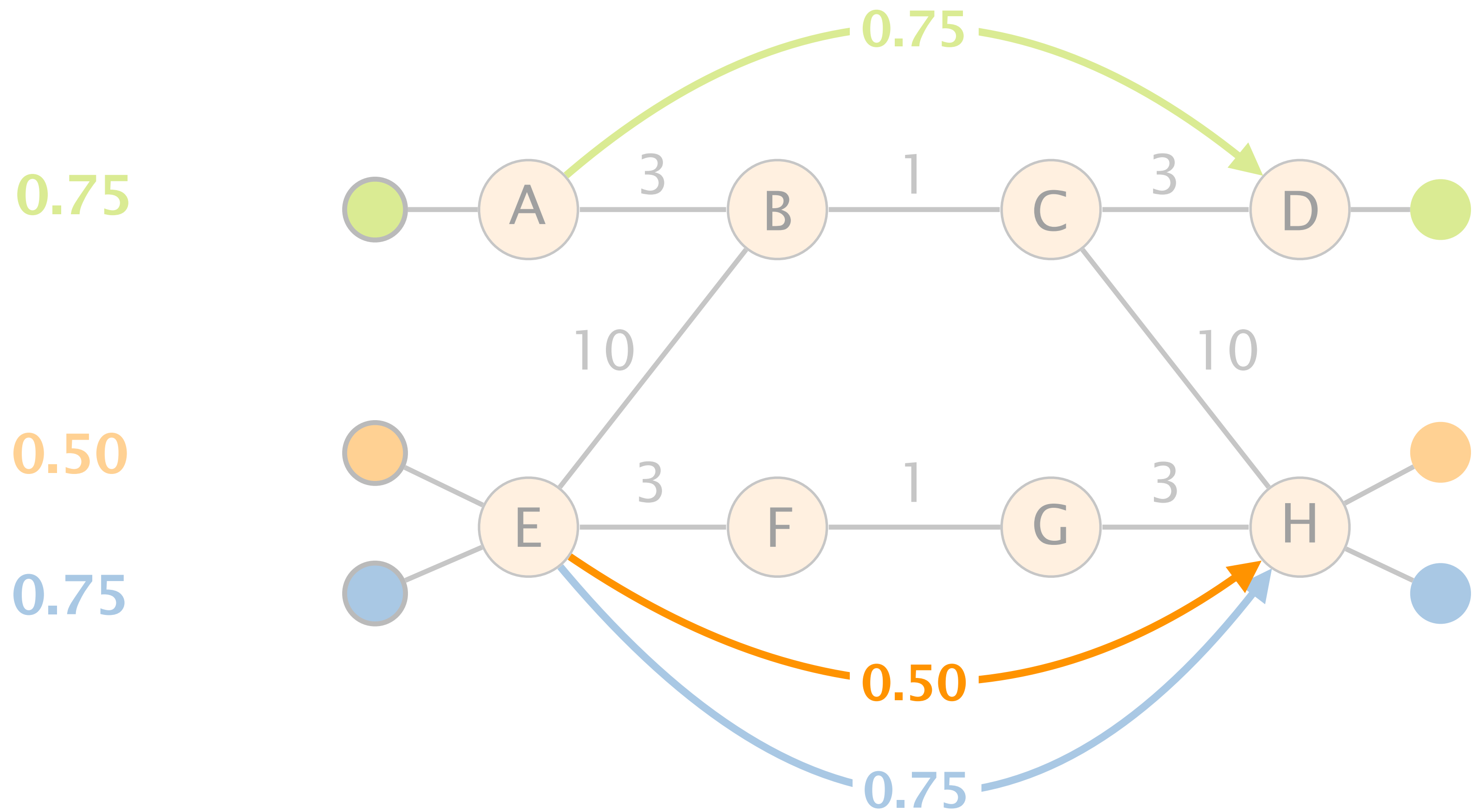
Congestion can be alleviated by splitting the orange flow into two equal parts (.25)



This is impossible to achieve using a link-state protocol

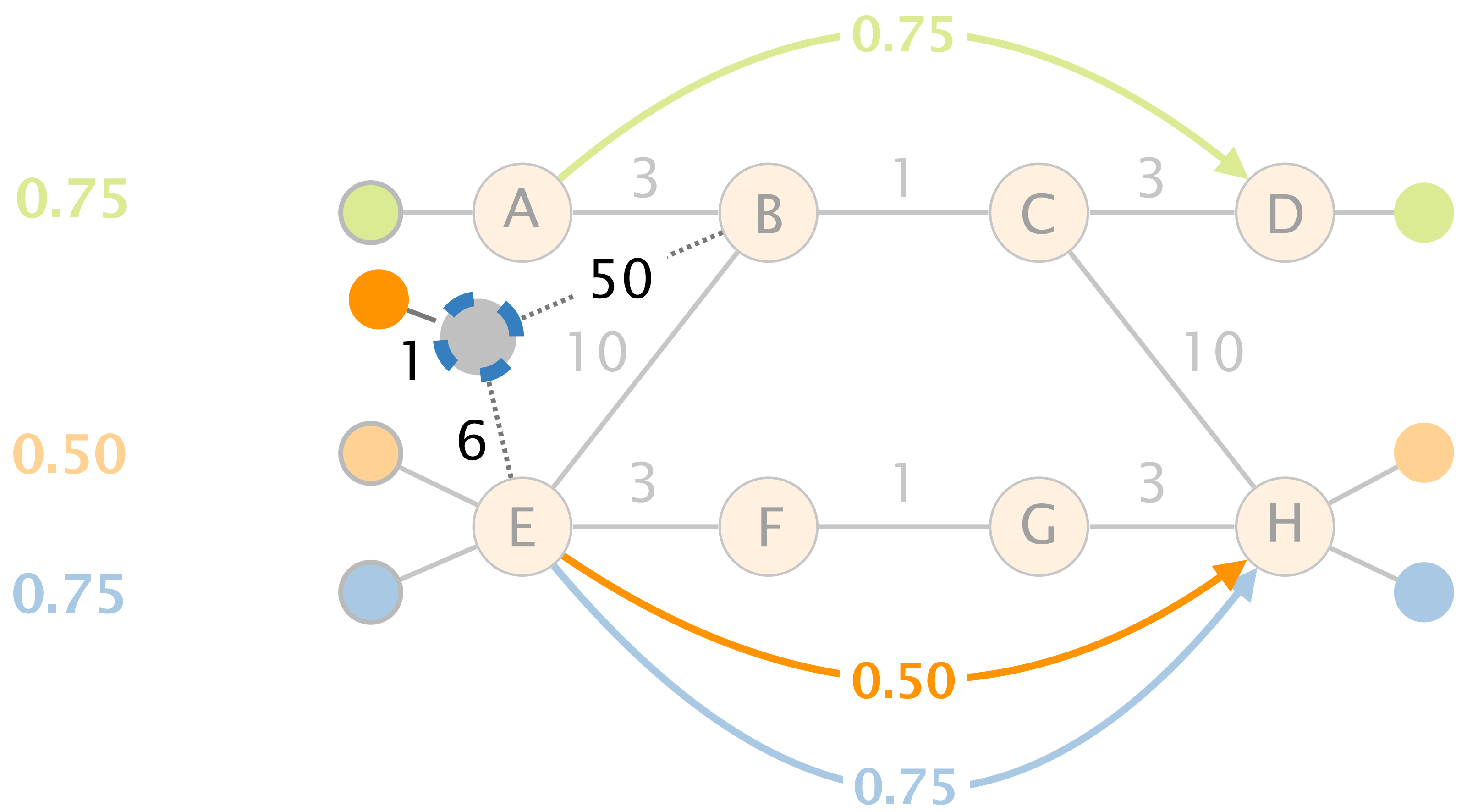


This is easily achievable with Fibbing

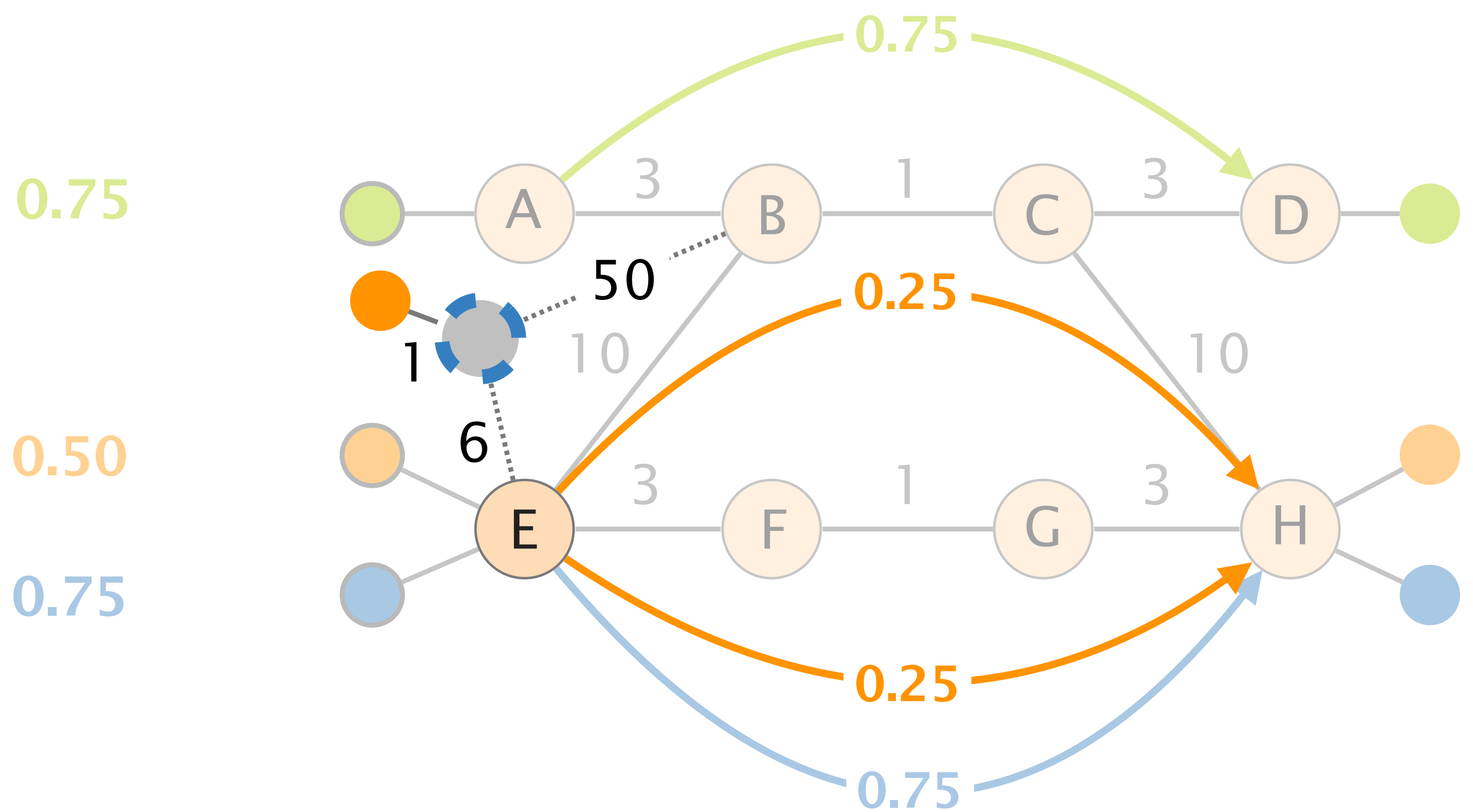


One lie is introduced,

announcing the orange destination



Now E has two equal cost paths (7) to reach only the orange destination and use them both



Central Control Over Distributed Routing

Fibbing

lying made useful

Expressivity

any path, anywhere

3 **Scalability**

1 lie is better than 2

Scalability

Scalability

time

to compute lies

space

# of lies

Scalability

time

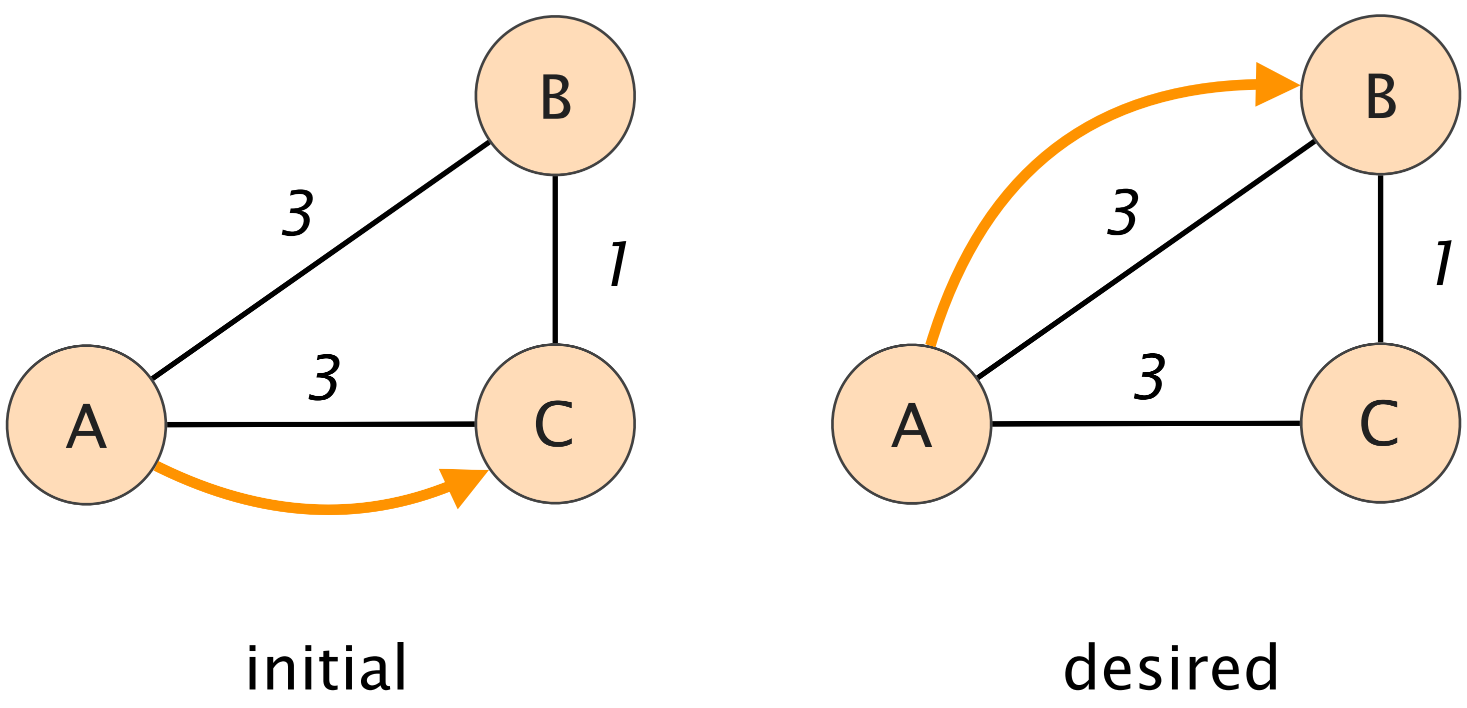
to compute lies

space

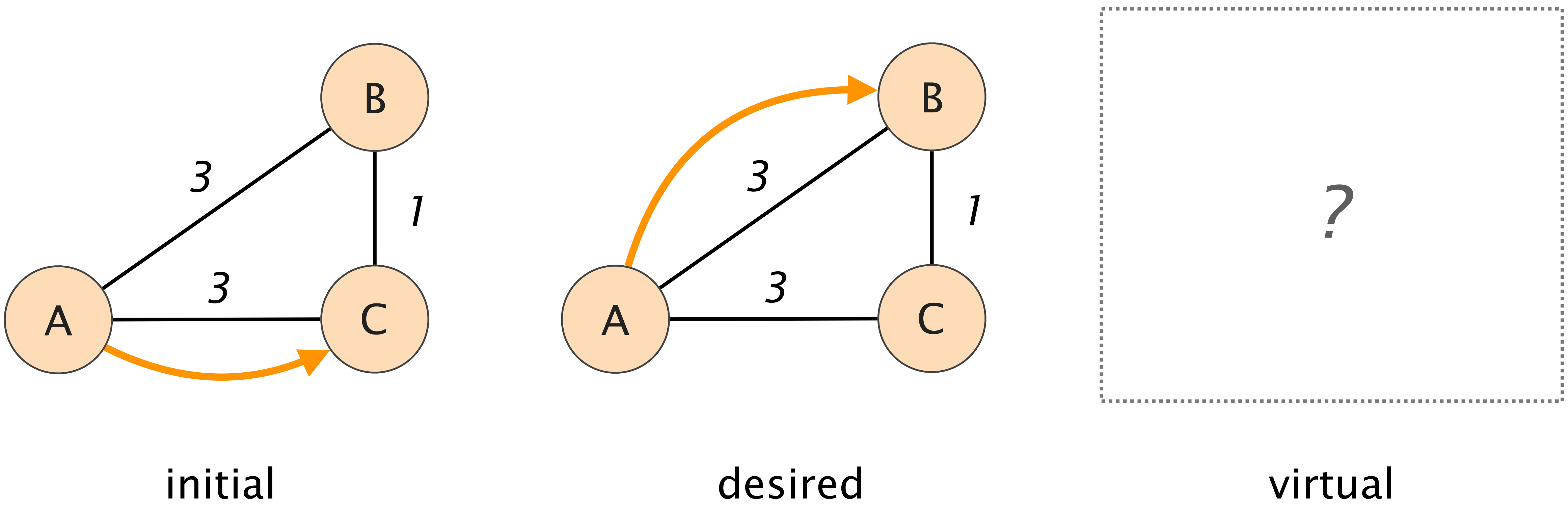
# of lies

Computing virtual topologies is easy:
polynomial in the number of requirements

Computing virtual topologies is easy: polynomial in the number of requirements



Computing virtual topologies is easy: polynomial in the number of requirements



For each router *r* whose next-hop

for a destination *d* changes to *j:*

For each router *r* whose next-hop
for a destination *d* changes to *j:*

- Let *w* be the current path weight between *r* and *d*
- Create one virtual node *v* advertising *d* with a weight $x < w$
- Connects it to *r* and *j*

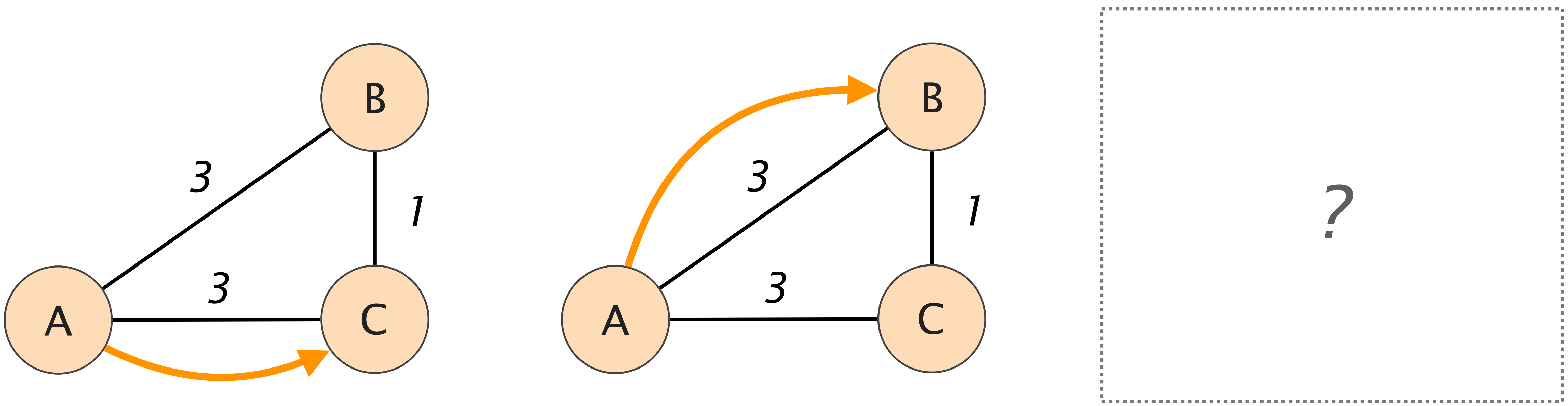
- Create one virtual node v advertising d with a weight $x < w$

always possible

by reweighting the initial graph

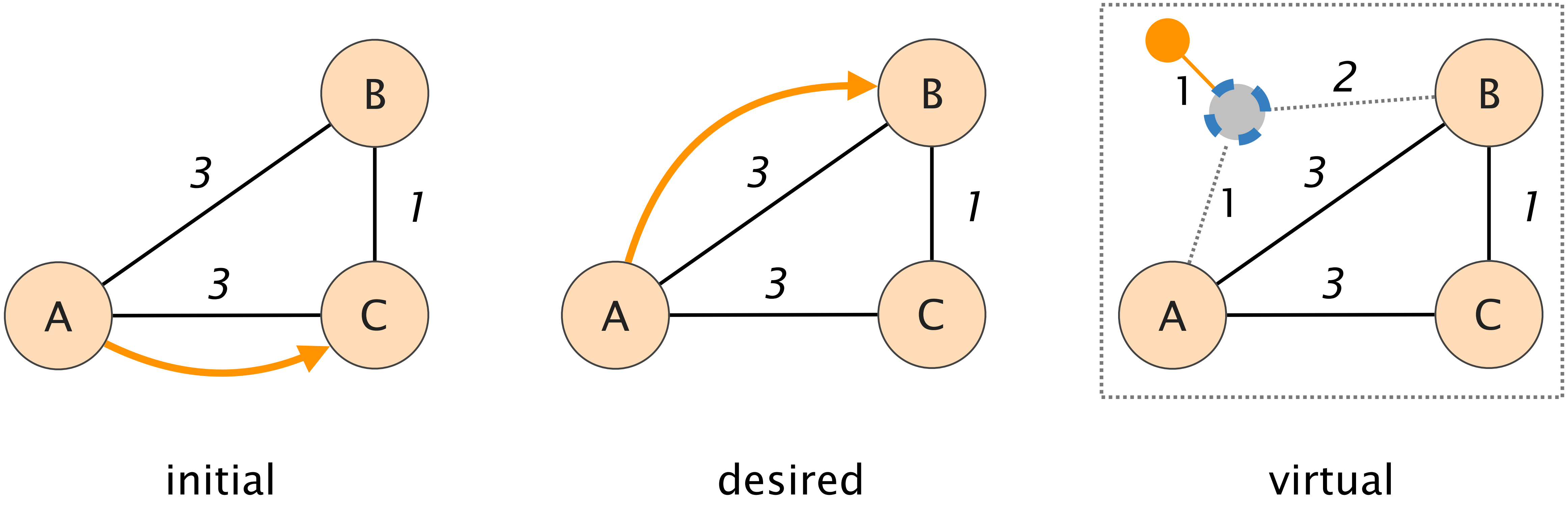
- Create one virtual node v advertising d with a weight $x < w$

Computing virtual topologies is easy:
polynomial in the number of requirements



initial desired virtual

Computing virtual topologies is easy: polynomial in the number of requirements



The resulting topology can be huge
and each router needs to run Dijkstra on it

Dijkstra's algorithm complexity

$$O(\,|E| + |V| \log |V|\,)$$

#nodes #links

Scalability

time

to compute lies

space

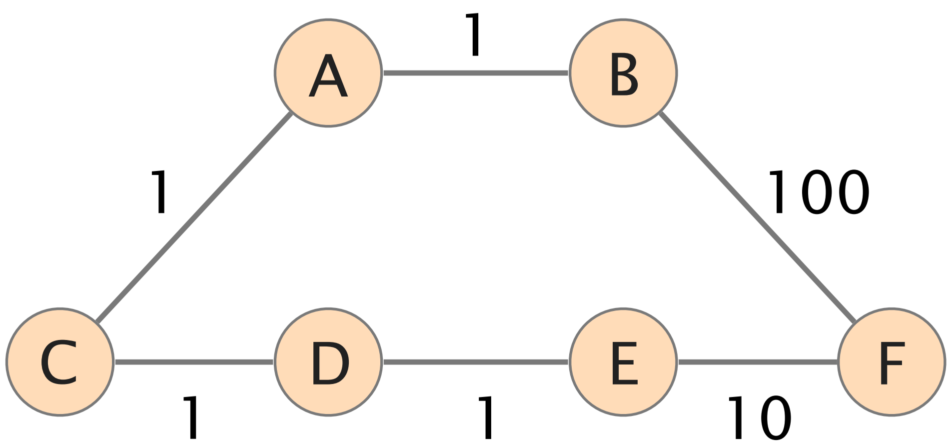
# of lies

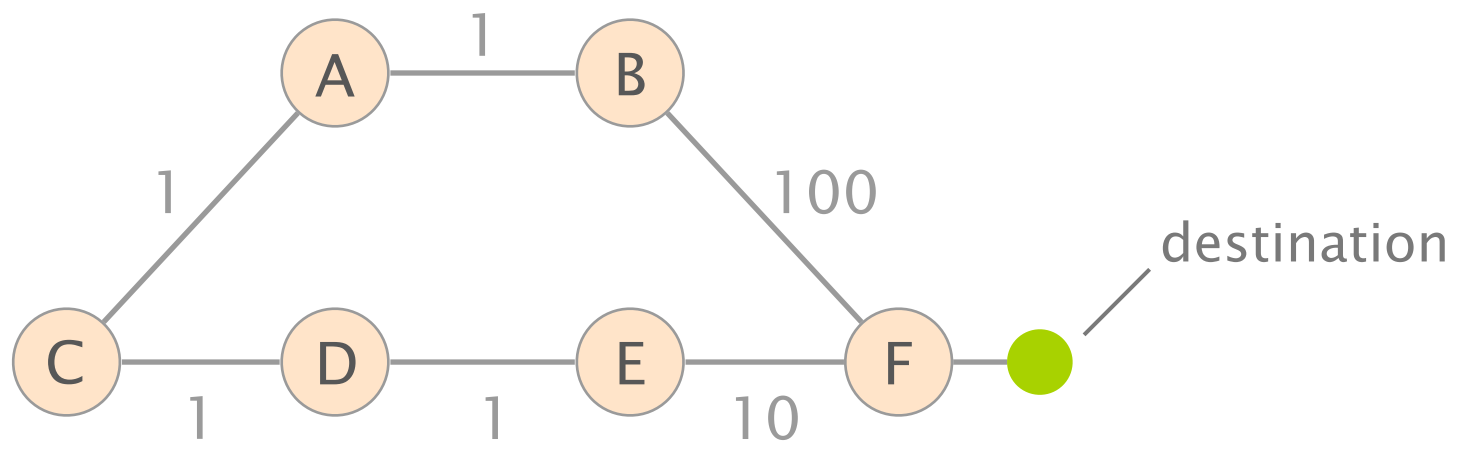
Good news

Lots of lies are not required,
some of them are redundant

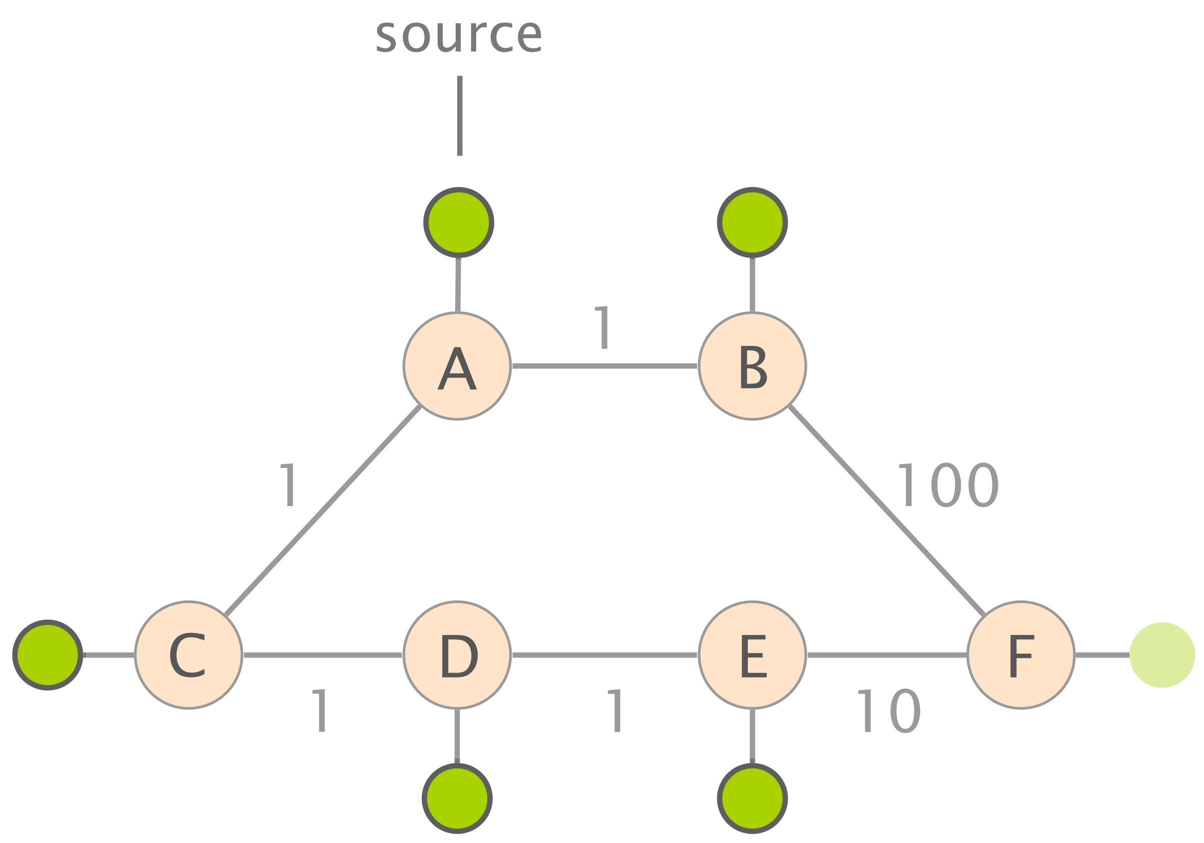
Let’s us consider

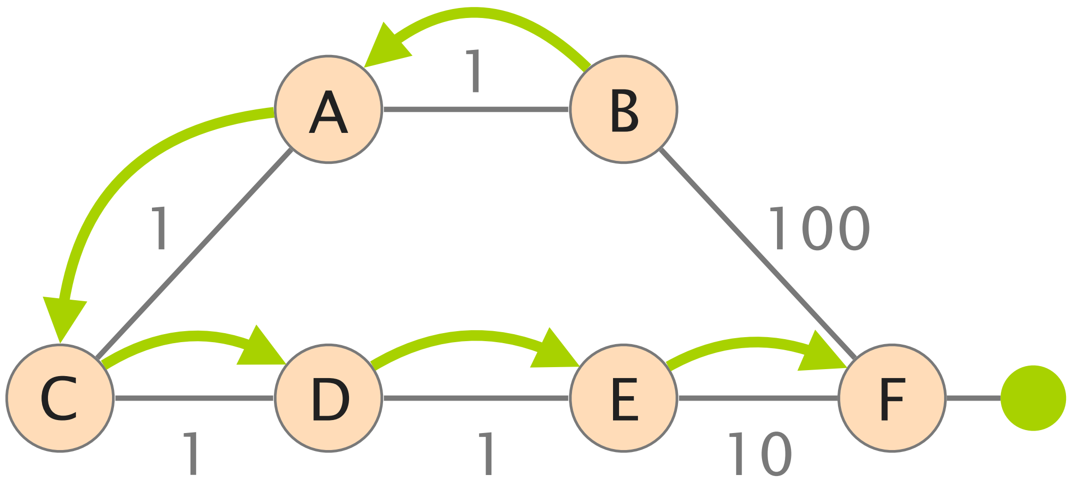
a simple example


1
A
B
1
100
C
D
E
F
1
1
10



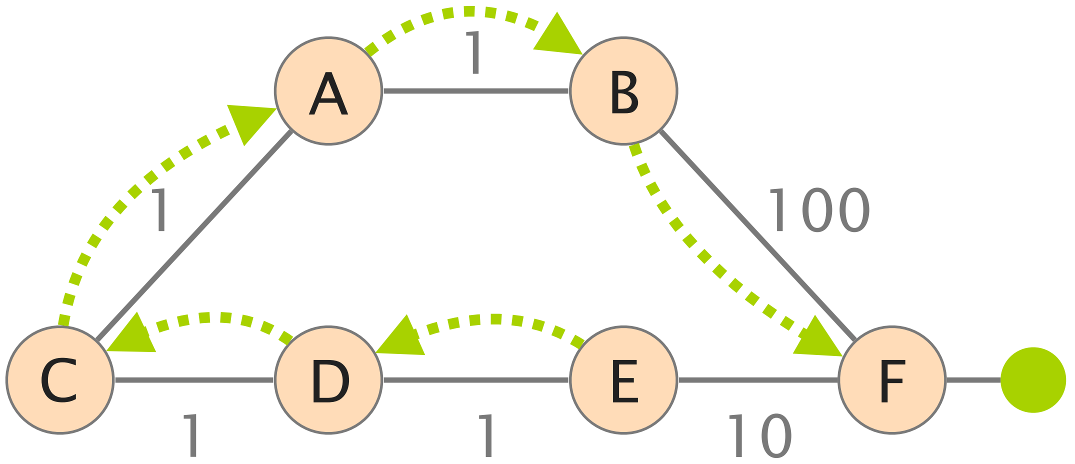
1
A
B
1
100
destination
C
D
E
F
1
1
10


source
A
B
C
D
E
F
1
1
100
1
1
10



original shortest-path

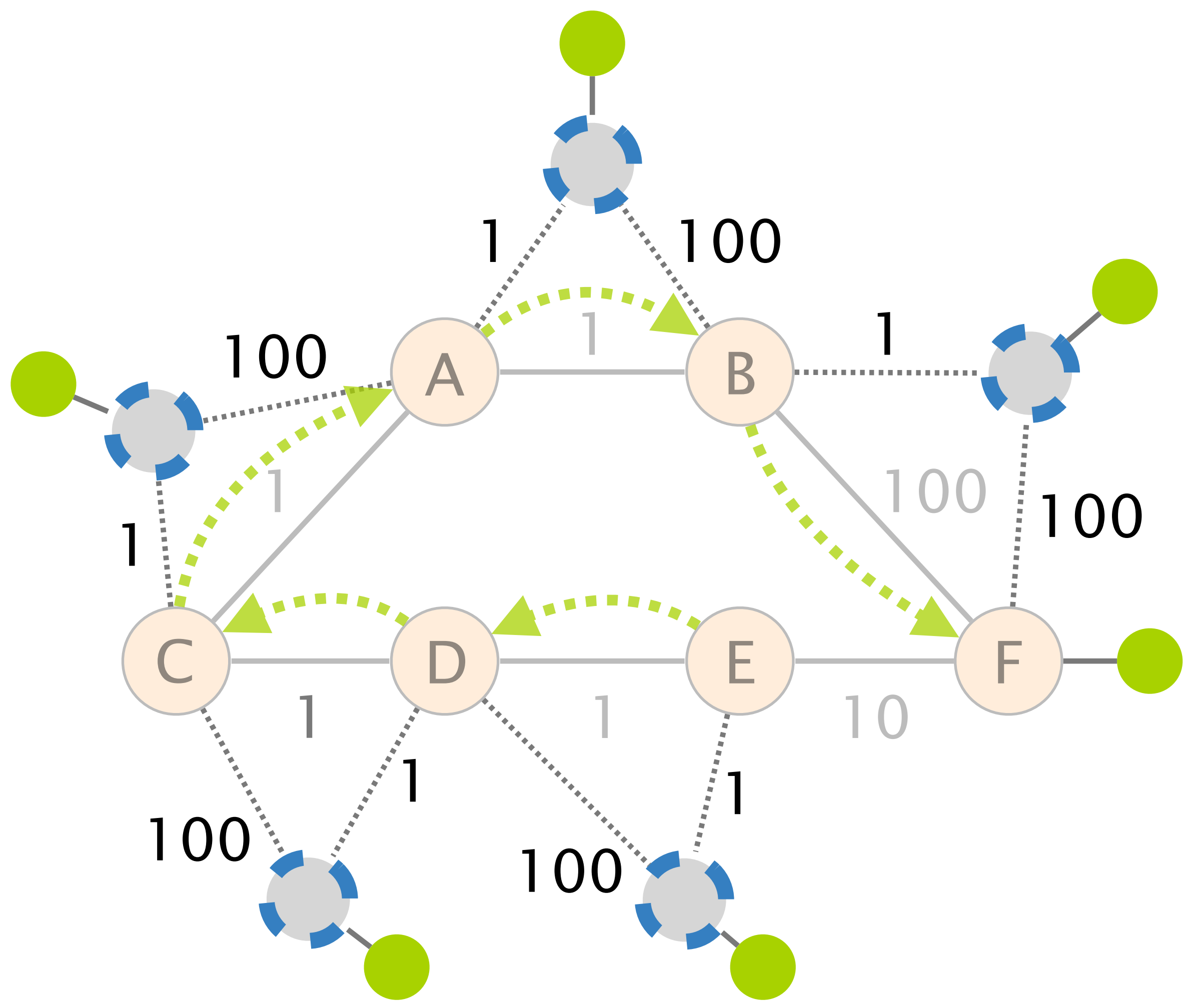
"down and to the right"



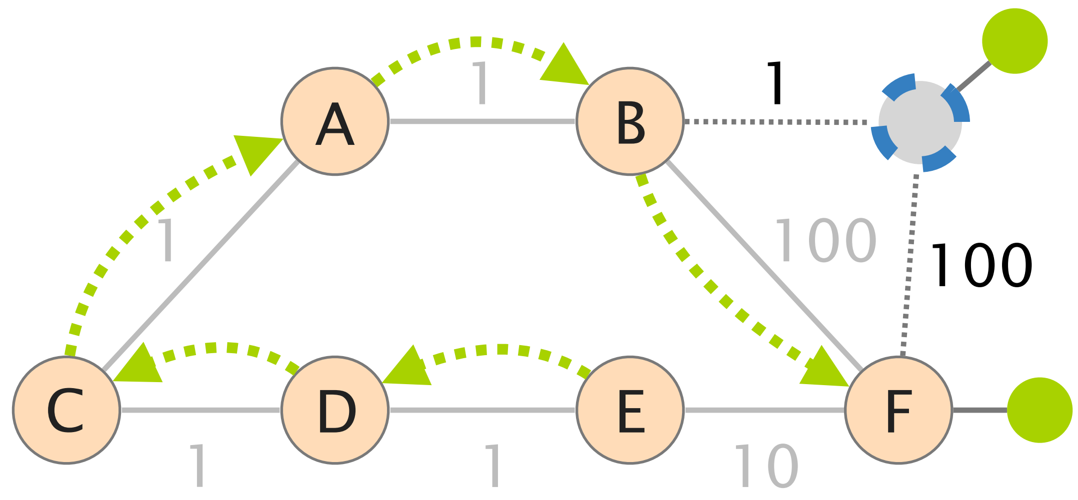
desired shortest-path

“up and to the right”

Our naive algorithm would create 5 lies—one per router



A single lie is sufficient (and necessary)

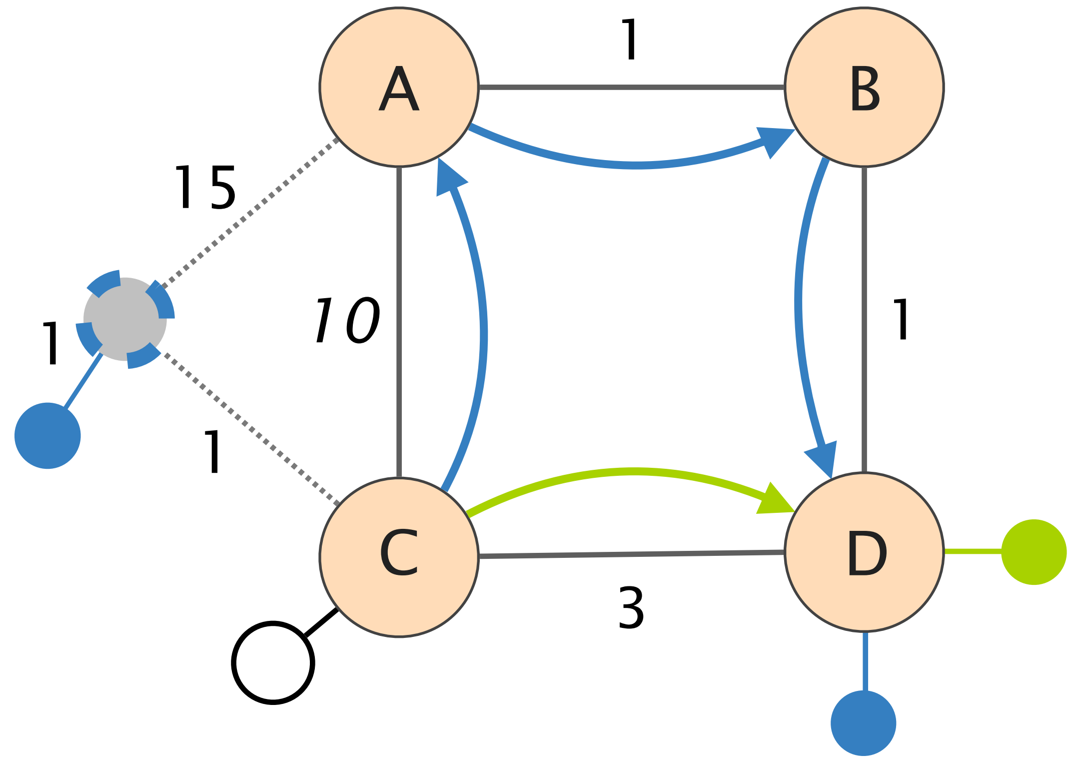


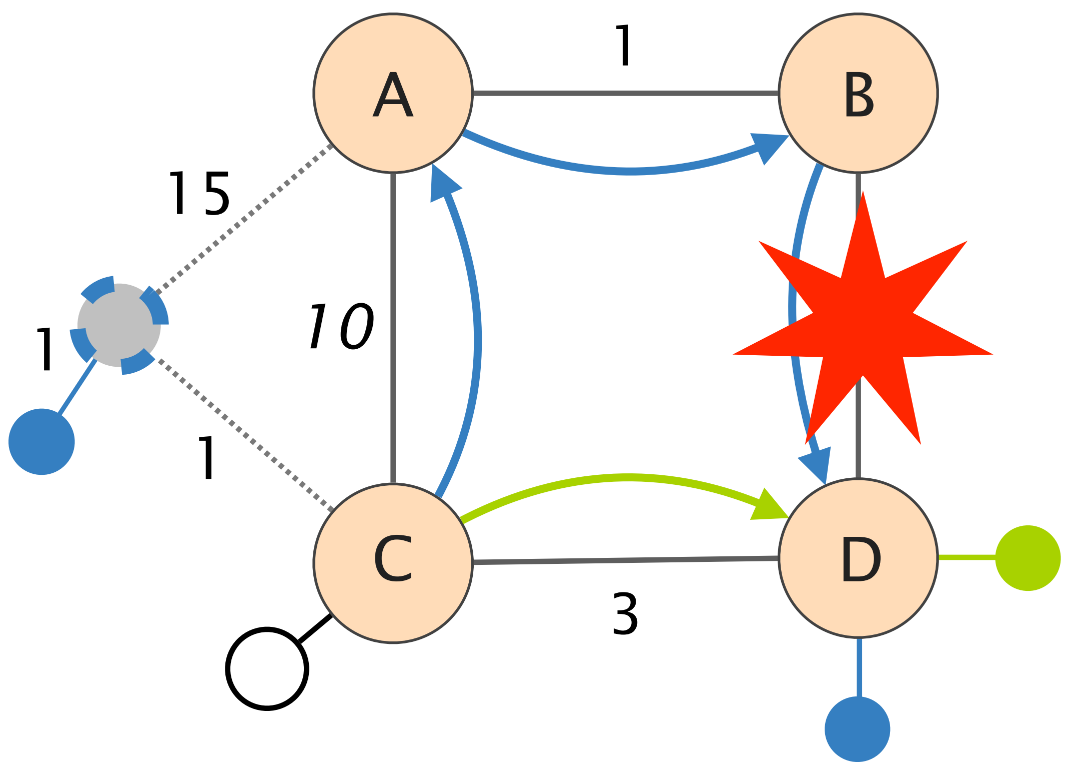
We can minimize the topology size
using an Integer Linear Program

While efficient, an ILP is inherently slow

	Naive	Integer Linear Program
time	optimal	slow
space (topology size)	large	optimal

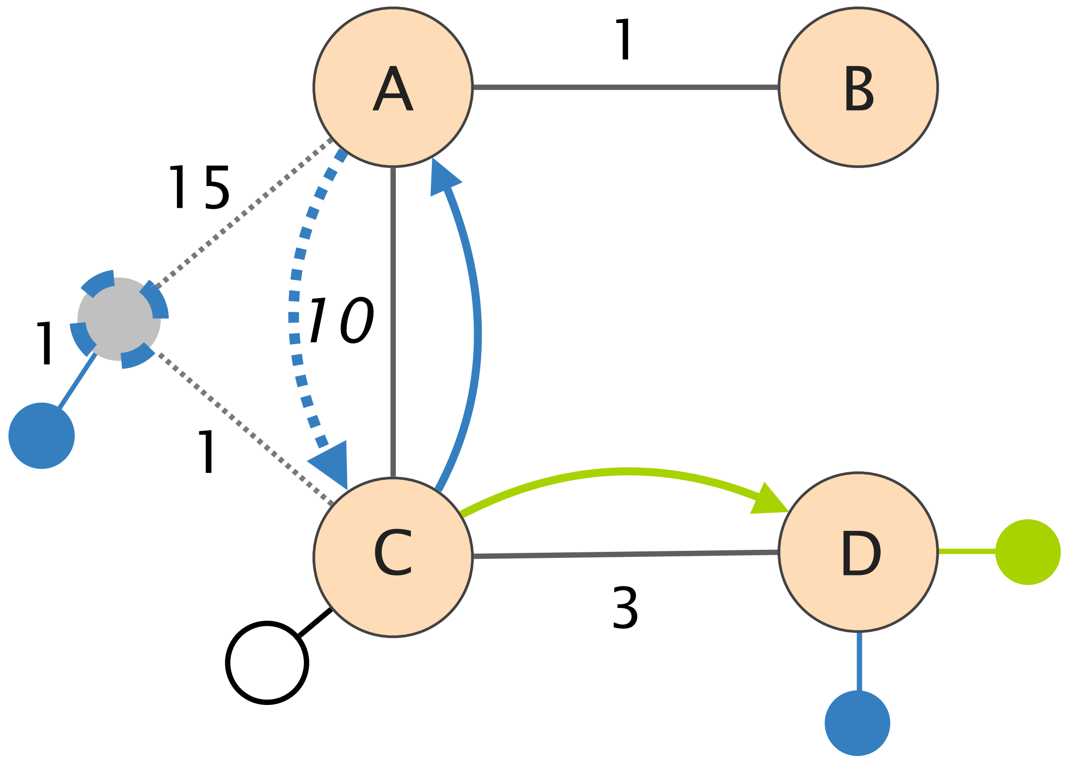
Computation time matters
in case of network failures


A
B
C
D
1
15
10
1
1
1
3

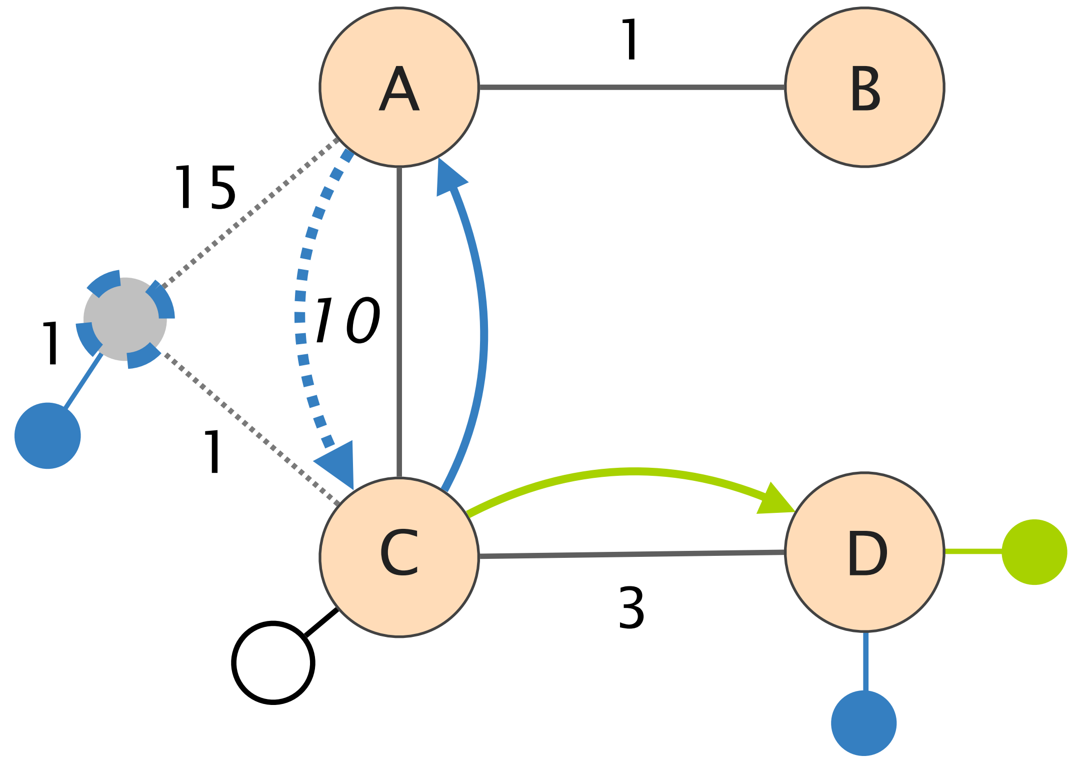


A
B
C
D
1
15
10
1
1
3

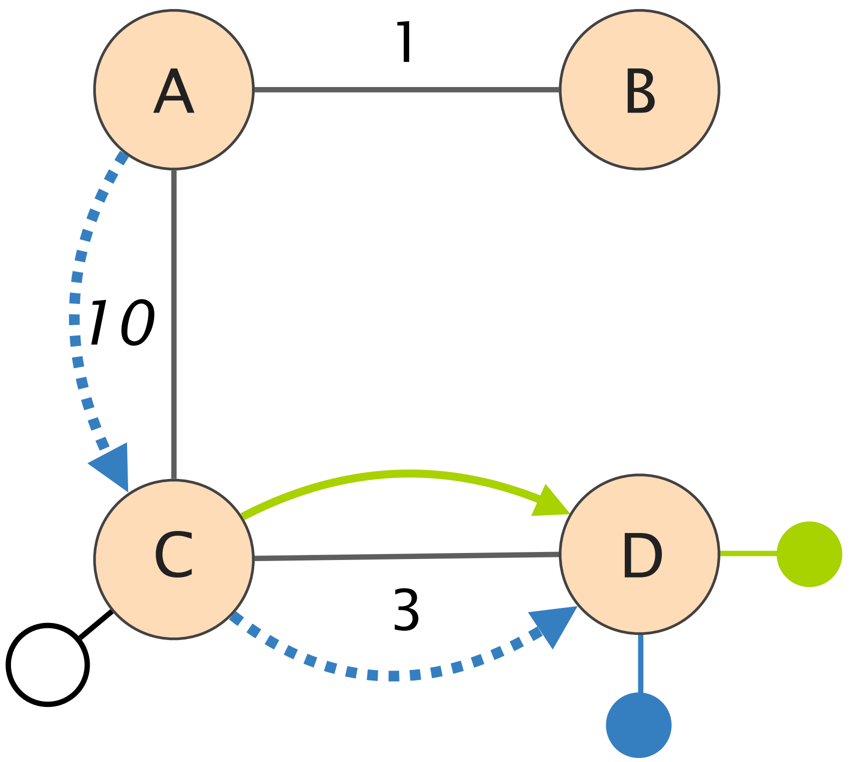
A loop is created as C starts to use A which still forwards according to the lie



The solution is to remove the lie



The solution is to remove the lie



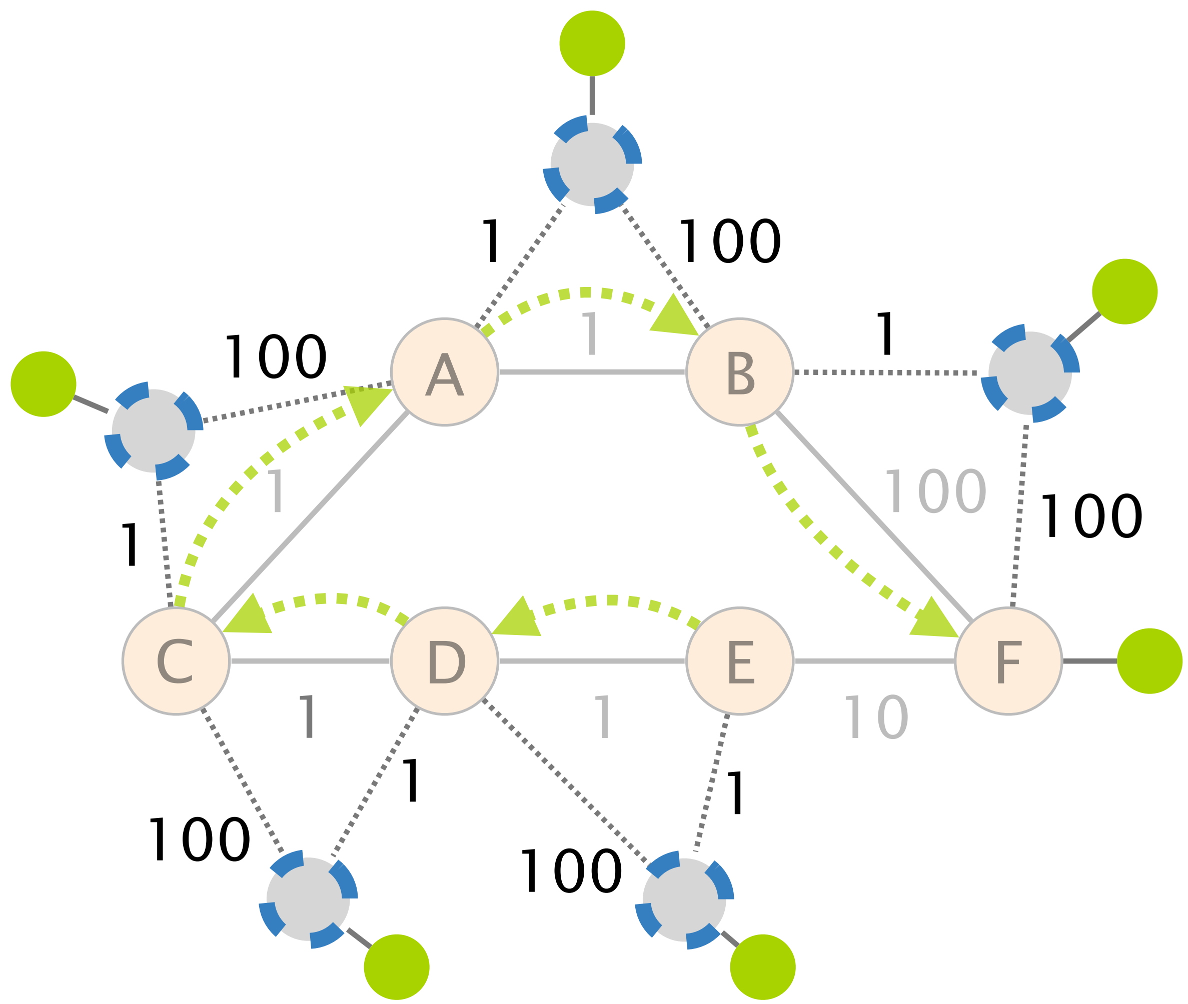
Upon failures, the network topology

has to be recomputed**, fast**

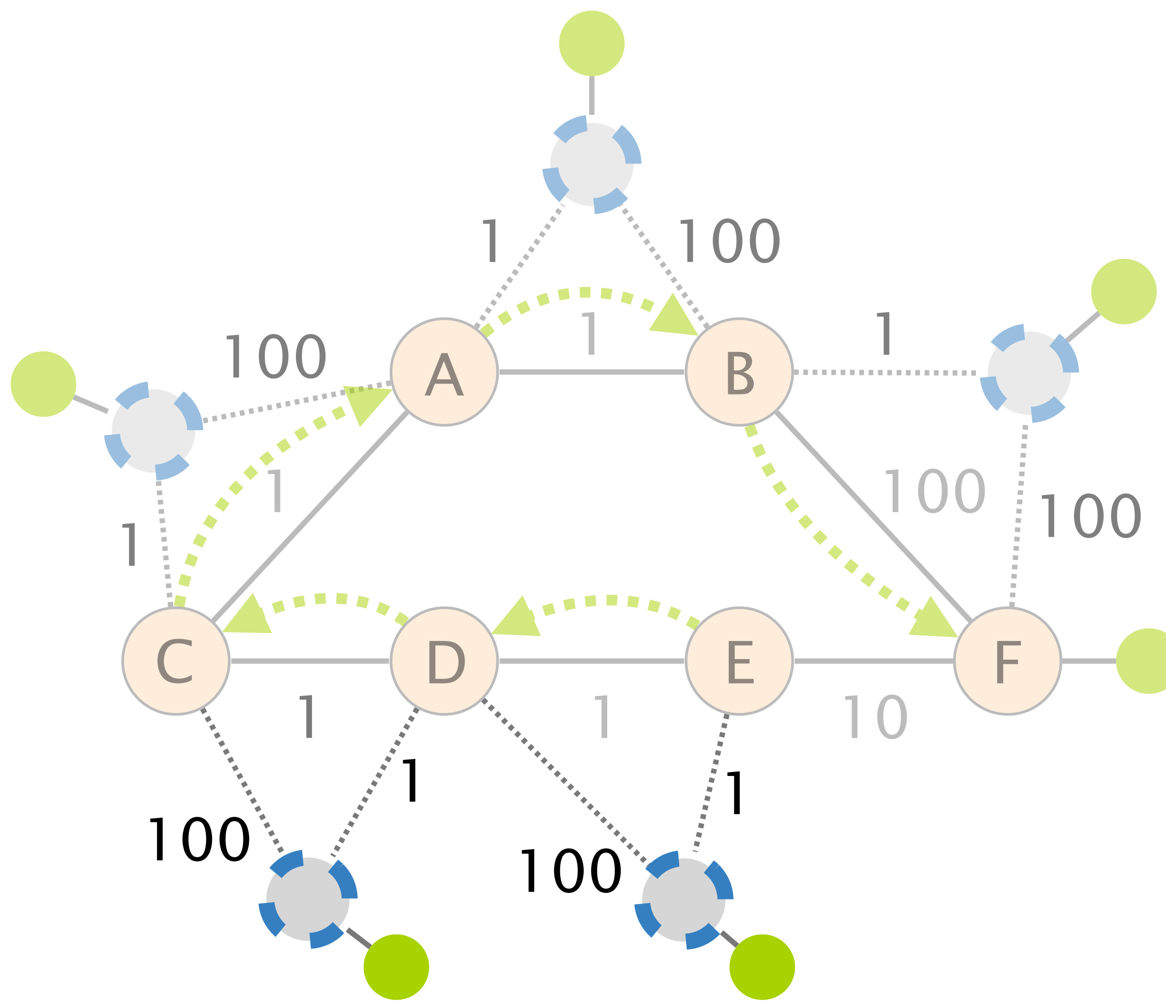
	Naive	Integer Linear Program
time	optimal	slow
space (topology size)	large	optimal

	Naive	Merger	Integer Linear Program
time	optimal	fast	slow
space (topology size)	large	small	optimal

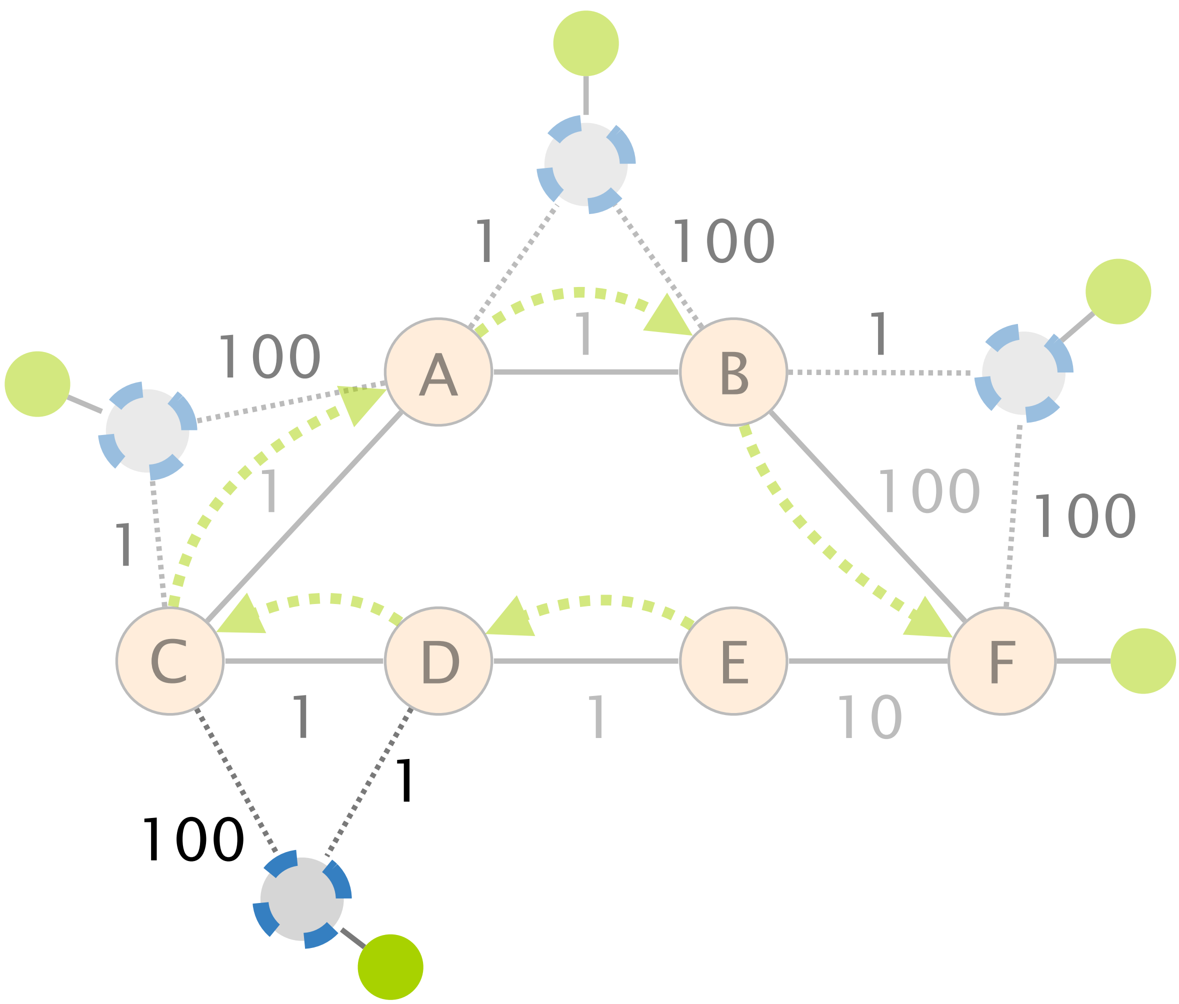
Merger iteratively tries to merge lies produced by the Naive algorithm



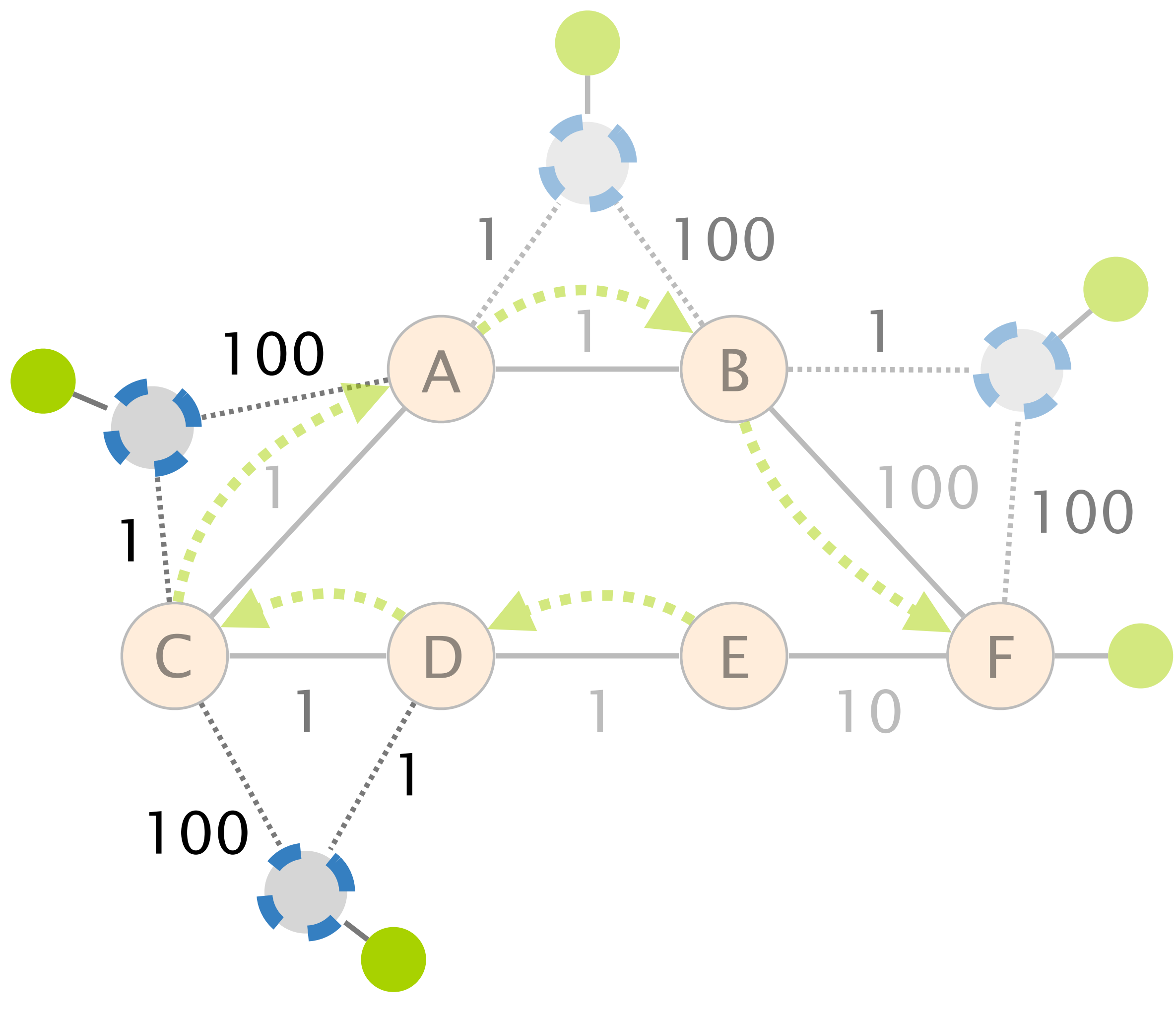
Merger iteratively tries to merge lies produced by the Naive algorithm



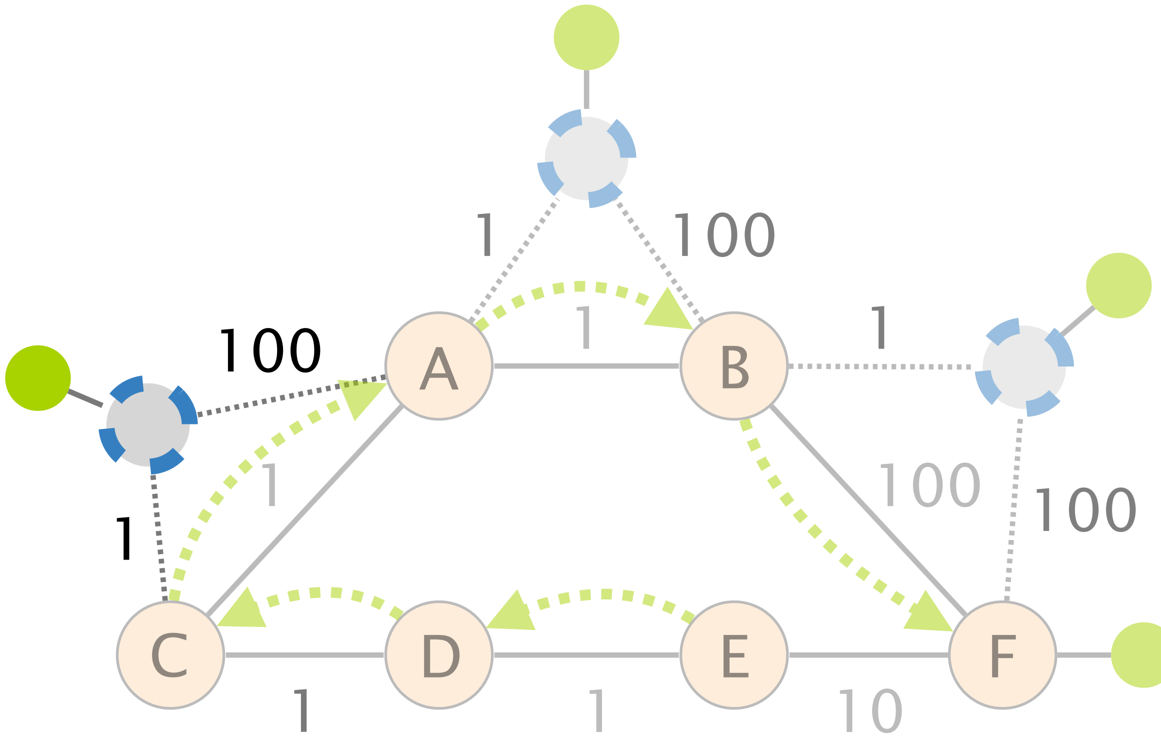
Merger iteratively tries to merge lies produced by the Naive algorithm



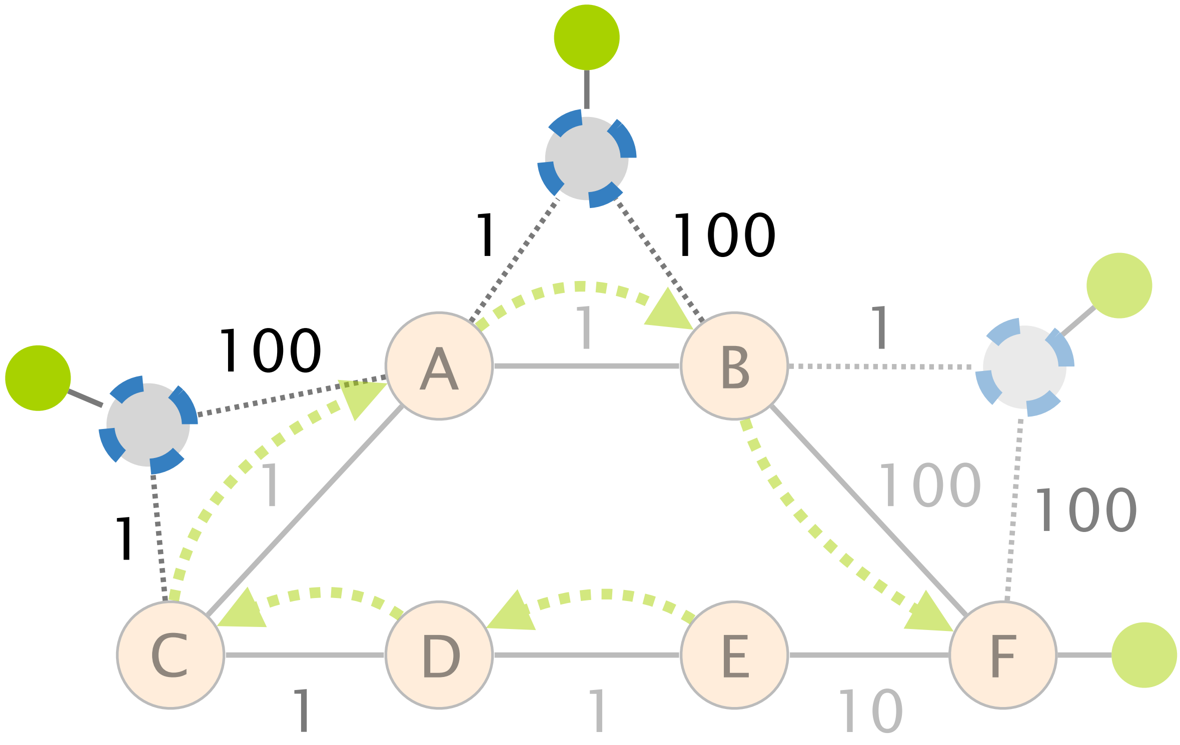
Merger iteratively tries to merge lies produced by the Naive algorithm



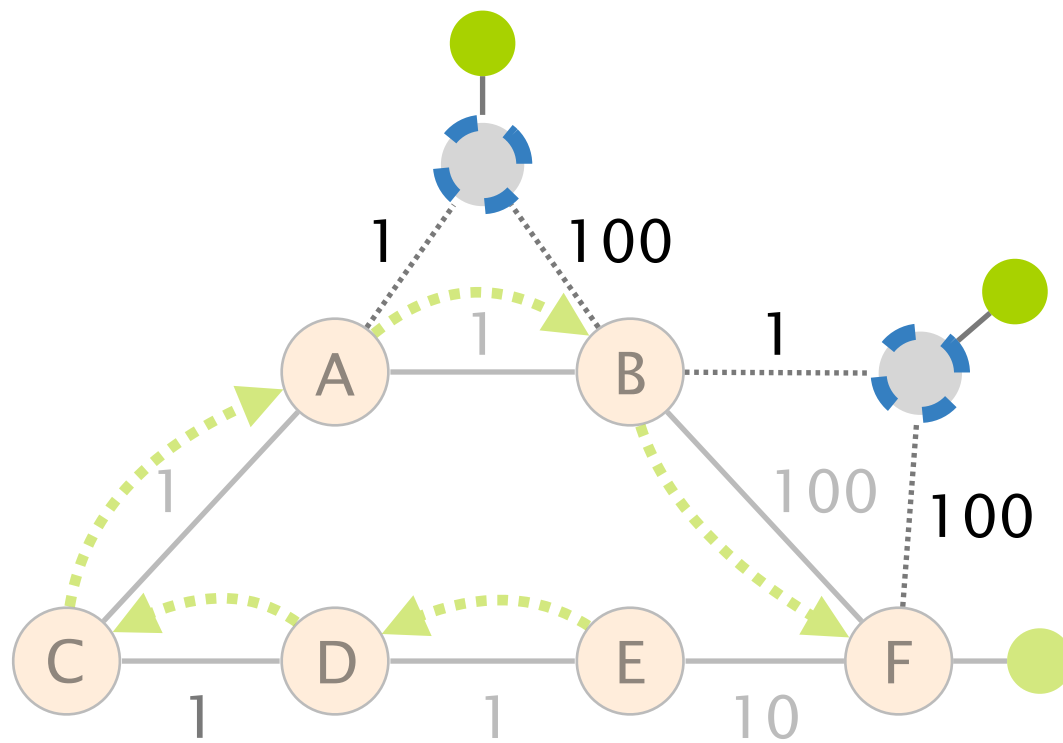
Merger iteratively tries to merge lies produced by the Naive algorithm



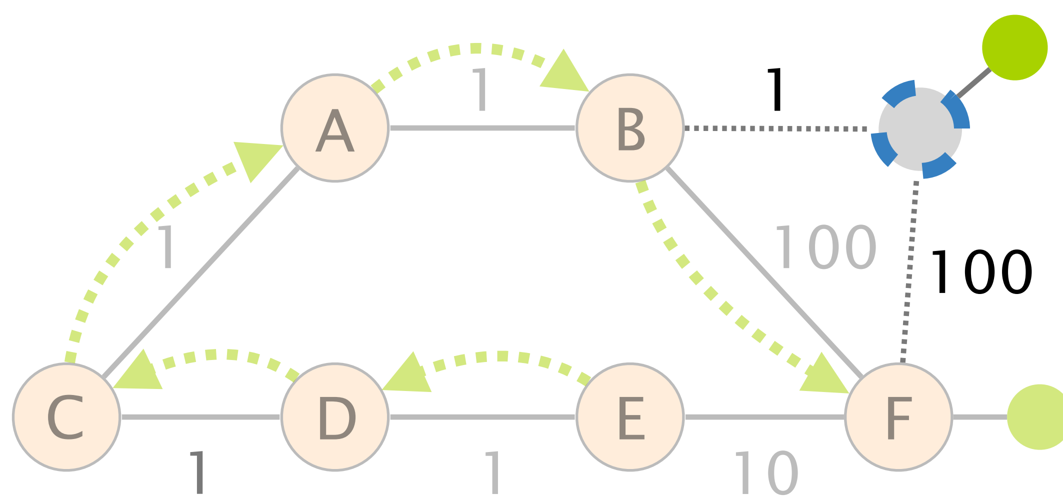
Merger iteratively tries to merge lies produced by the Naive algorithm



Merger iteratively tries to merge lies produced by the Naive algorithm



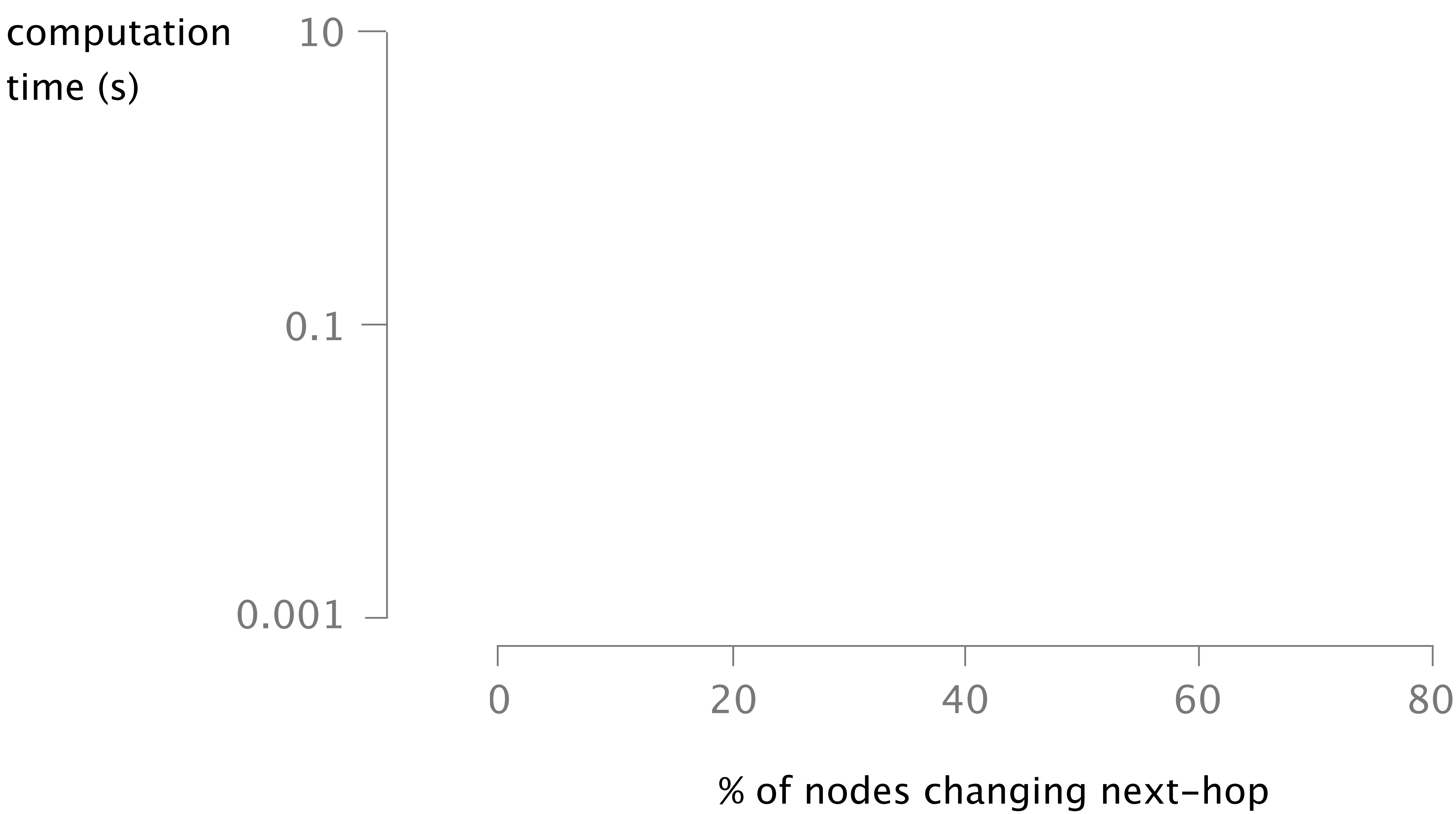
Merger iteratively tries to merge lies produced by the Naive algorithm



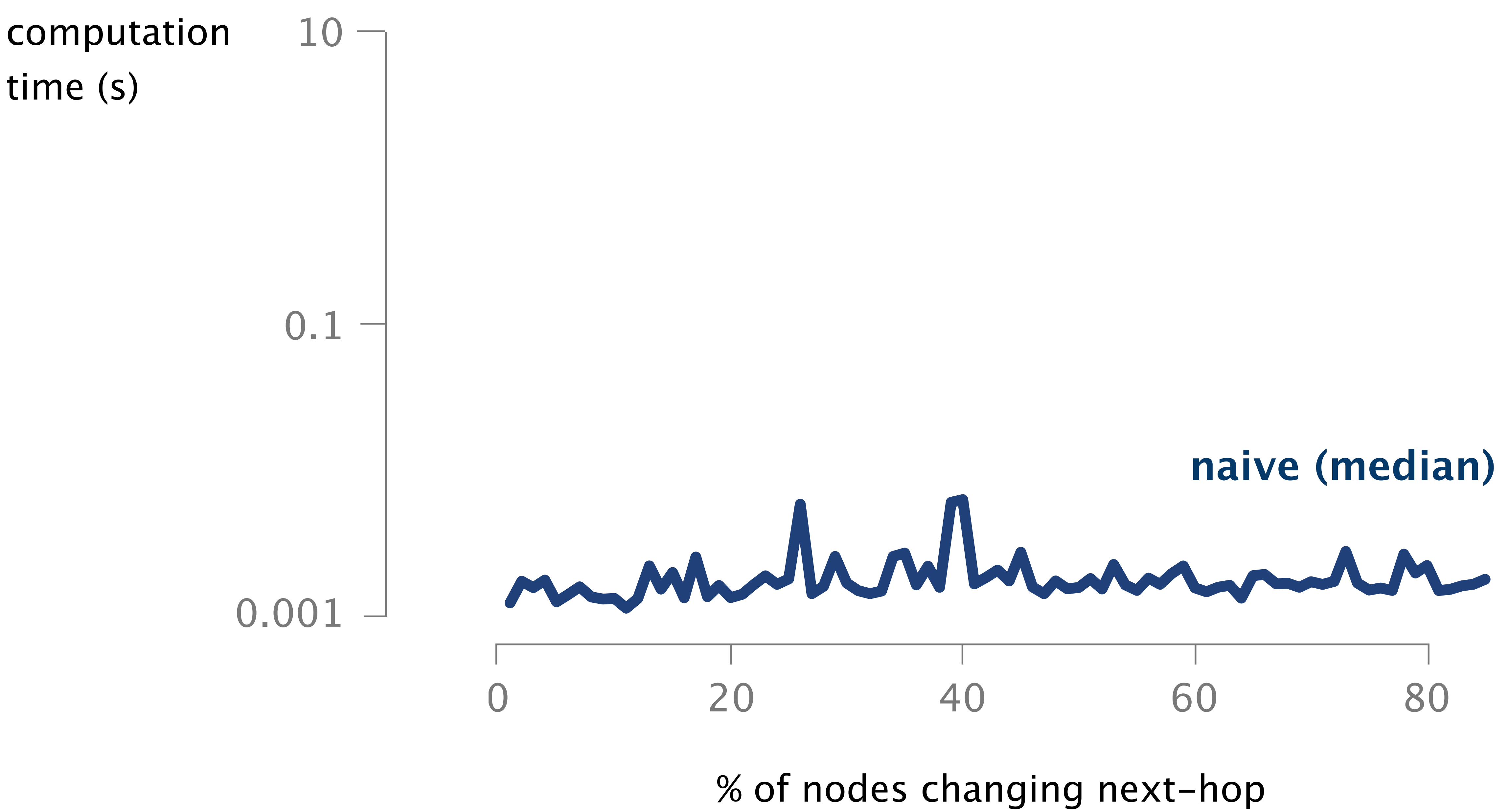
	Naive	Merger	Integer Linear Program
time	optimal	fast	slow
space (topology size)	large	small	optimal

Let's compare the performance of Naive and Merger

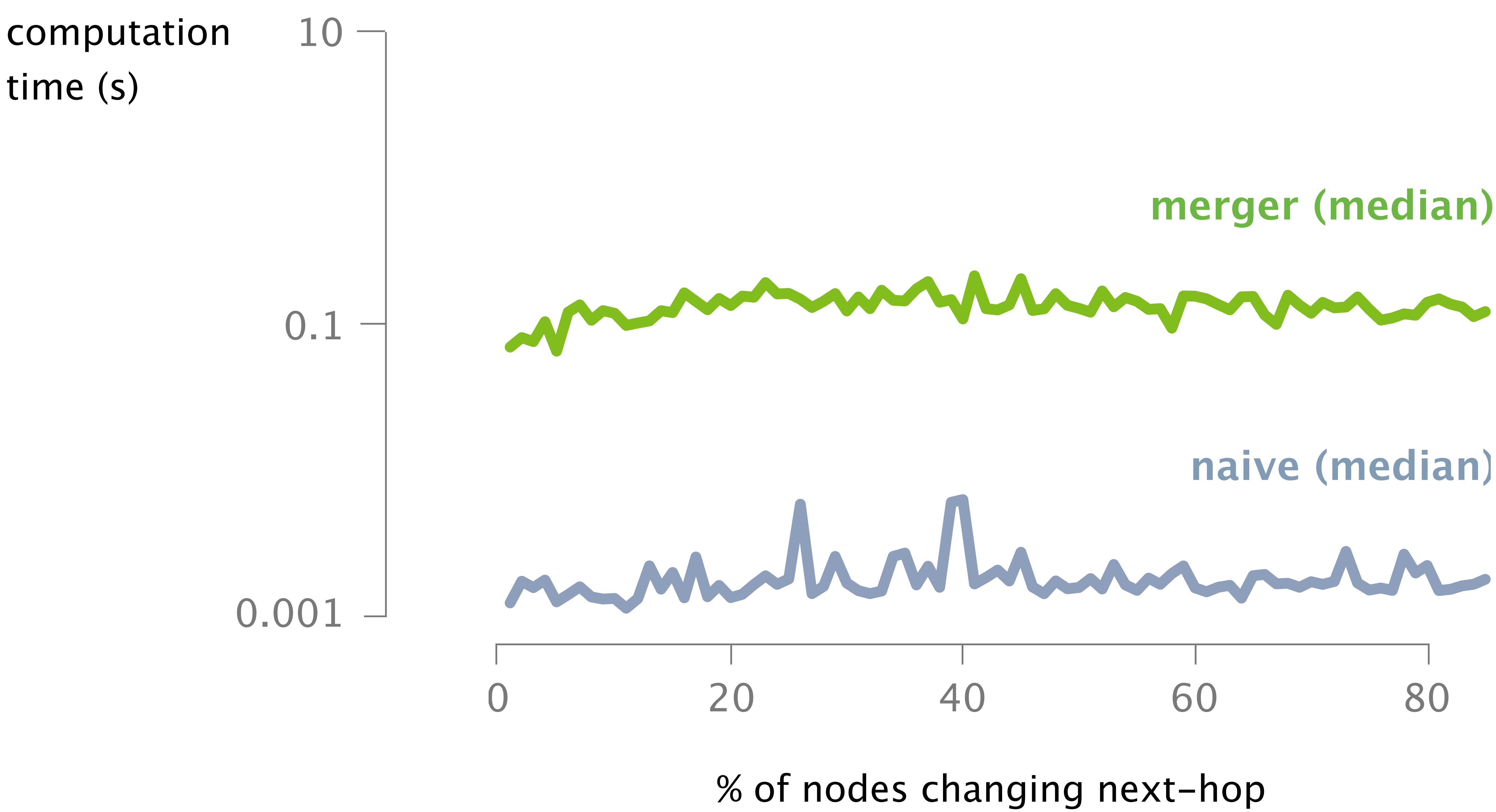
	Naive	Merger	Integer Linear Program
time	optimal	fast	slow
space (topology size)	large	small	optimal

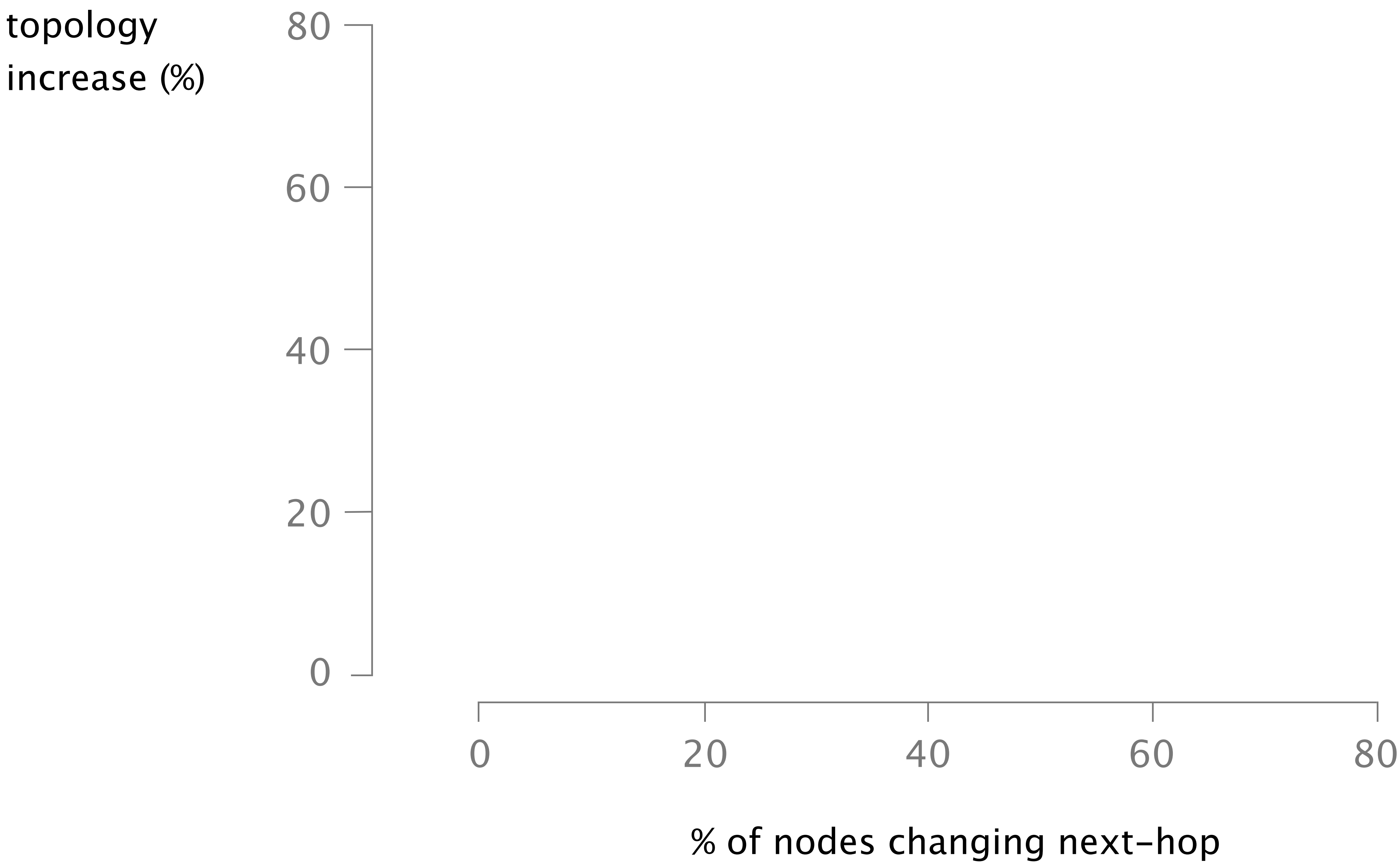

computation
time (s)
10
0.1
0.001
0
20
40
60
80
% of nodes changing next-hop

Naive computes entire virtual topologies in ms

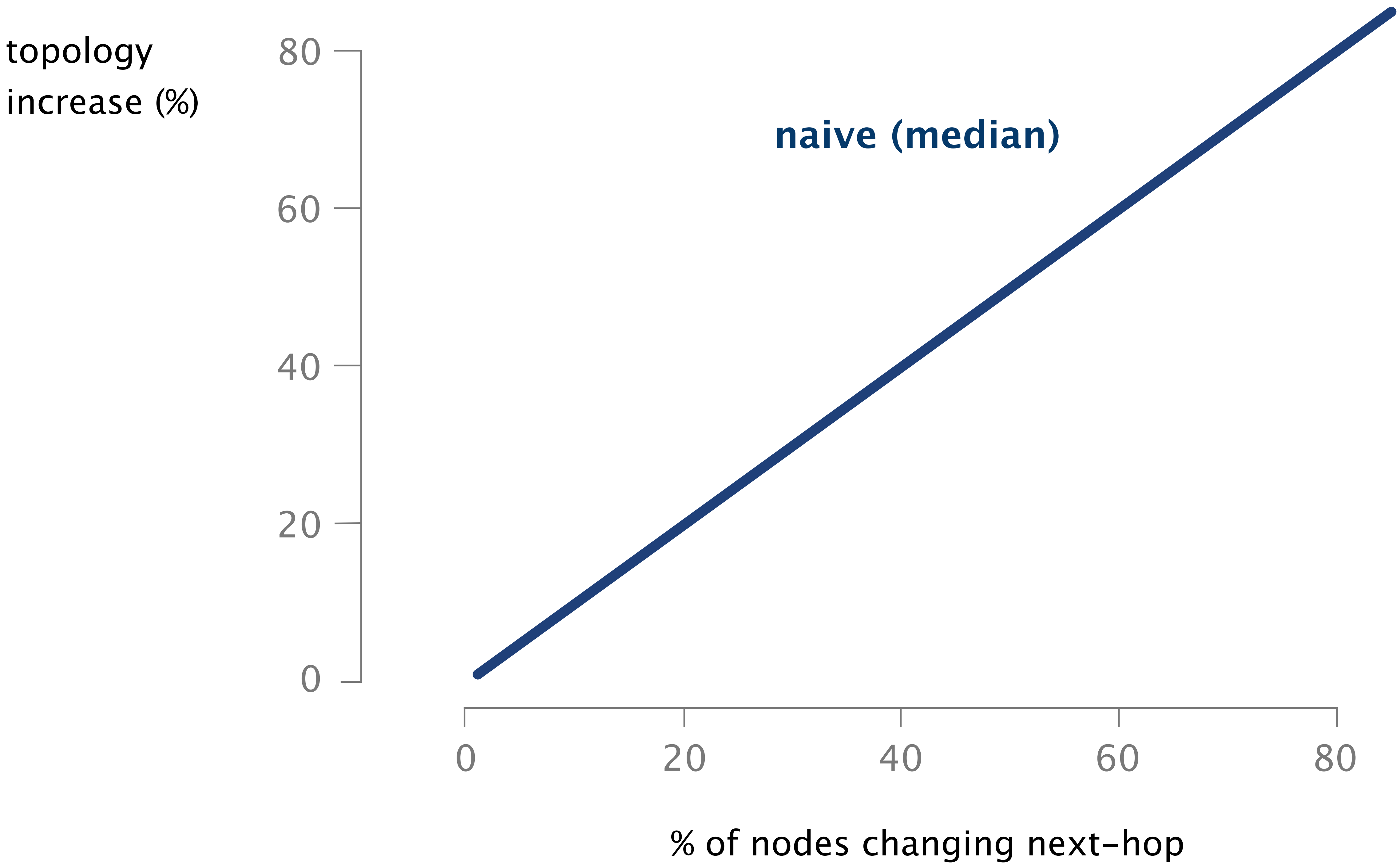


Merger is relatively slower, but still, sub-second

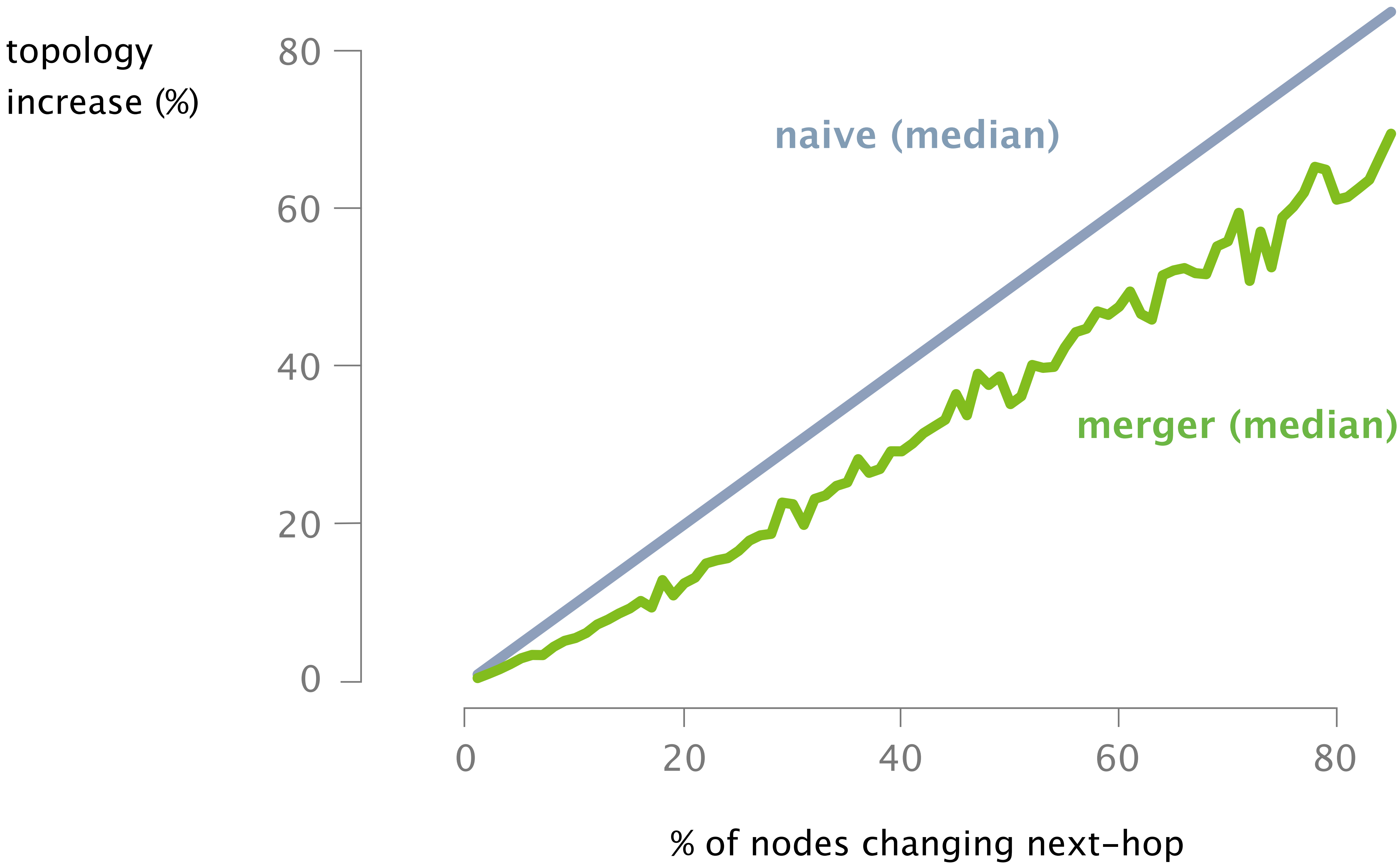



topology
increase (%)
80
60
40
20
0
0
20
40
60
80
% of nodes changing next-hop

Naive introduces one lie per changing next-hop



Merger reduces the size of the topology by 25% on average (50% in the best case)



We implemented a fully-fledged Fibbing prototype and tested it against real routers

We implemented a fully-fledged Fibbing prototype and tested it against real routers

2 measurements

How many lies can a router sustain?

How long does it take to process a lie?

Existing routers can easily sustain Fibbing-induced load, even with huge topologies

# fake nodes	router memory (MB)	
1000	0.7	
5 000	6.8	
10 000	14.5	
50 000	76.0	
100 000	153	DRAM is cheap

Because it is entirely distributed, programming forwarding entries is fast

# fake nodes	installation time (s)	
1000	0.9	
5 000	4.5	
10 000	8.9	
50 000	44.7	
100 000	89.50	894.50 µs/entry

Central Control Over Distributed Routing

Fibbing

lying made useful

Expressivity

any path, anywhere

Scalability

1 lie is better than 2

Fibbing realizes some of the SDN promises today, on an existing network

Facilitate SDN deployment

SDN controller can program routers and SDN switches

Simplify controller implementation

most of the heavy work is still done by the routers

Maintain operators' mental model

good old protocols running, easier troubleshooting

Fibbing

improved flexibility

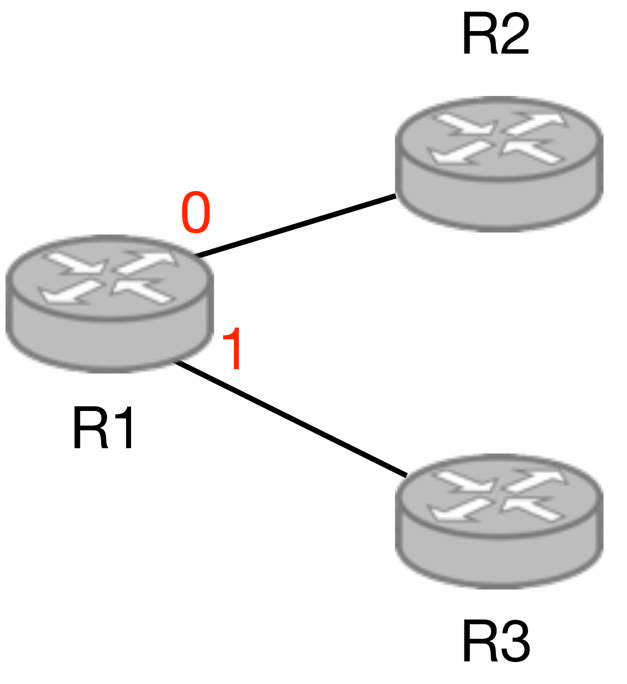
Supercharged

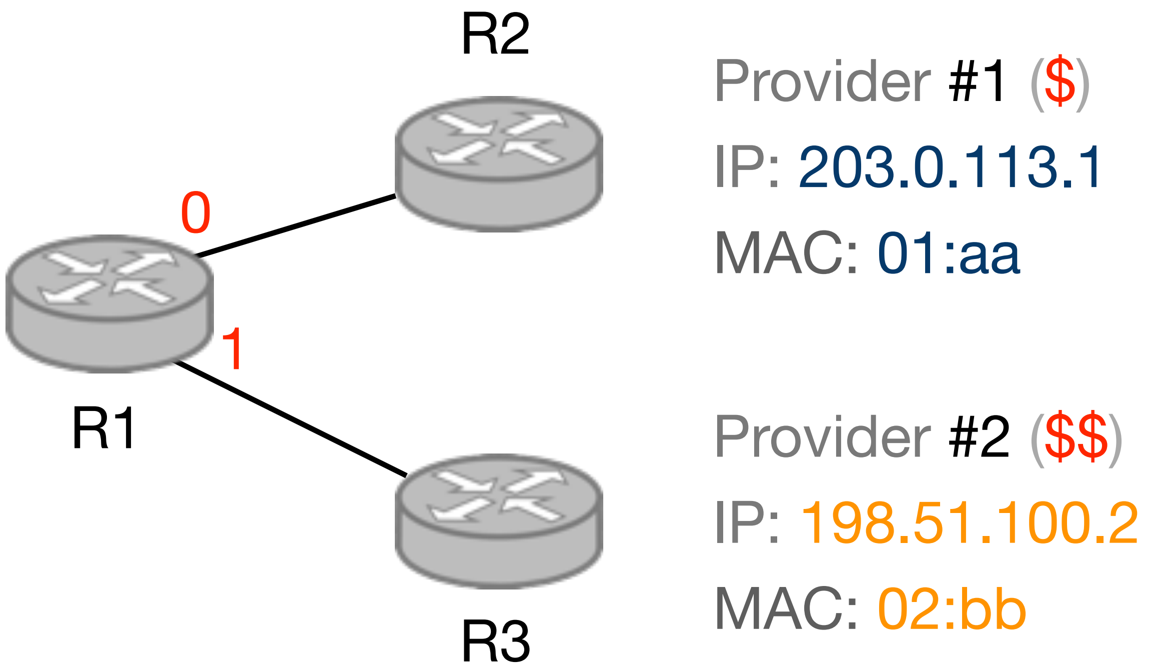
performance boost

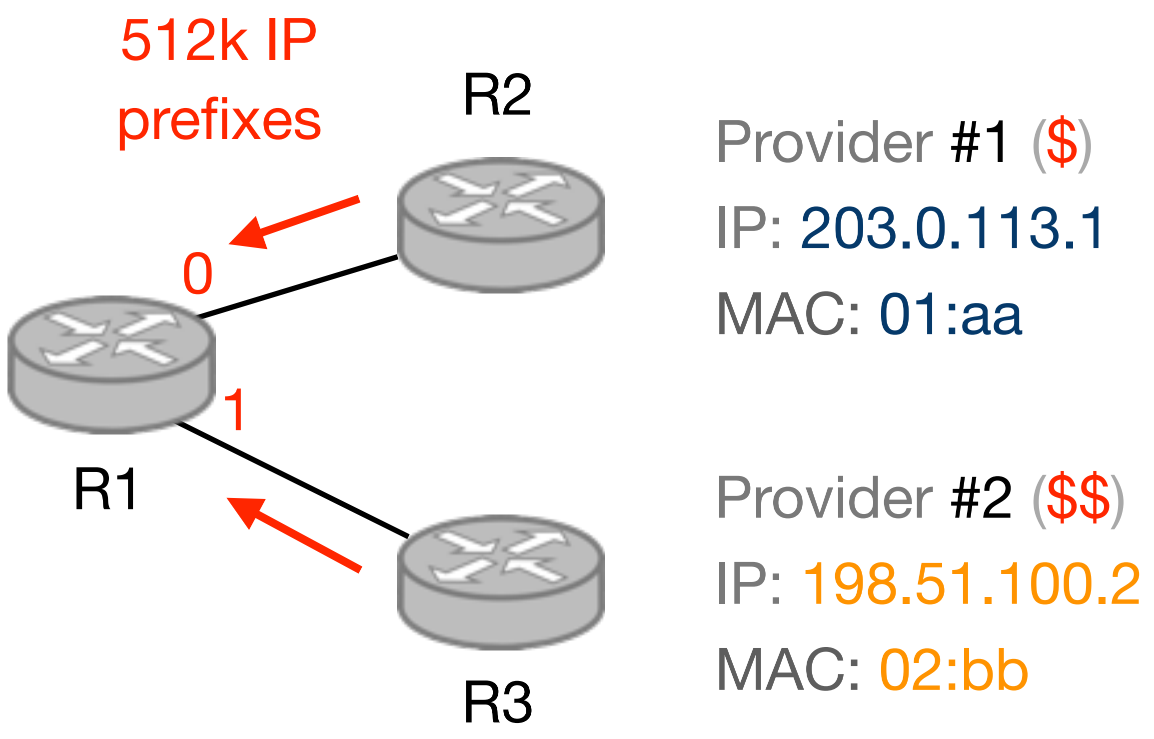
reduce convergence time
by **1000x**

IP routers are pretty slow to converge upon link and node failures

R1

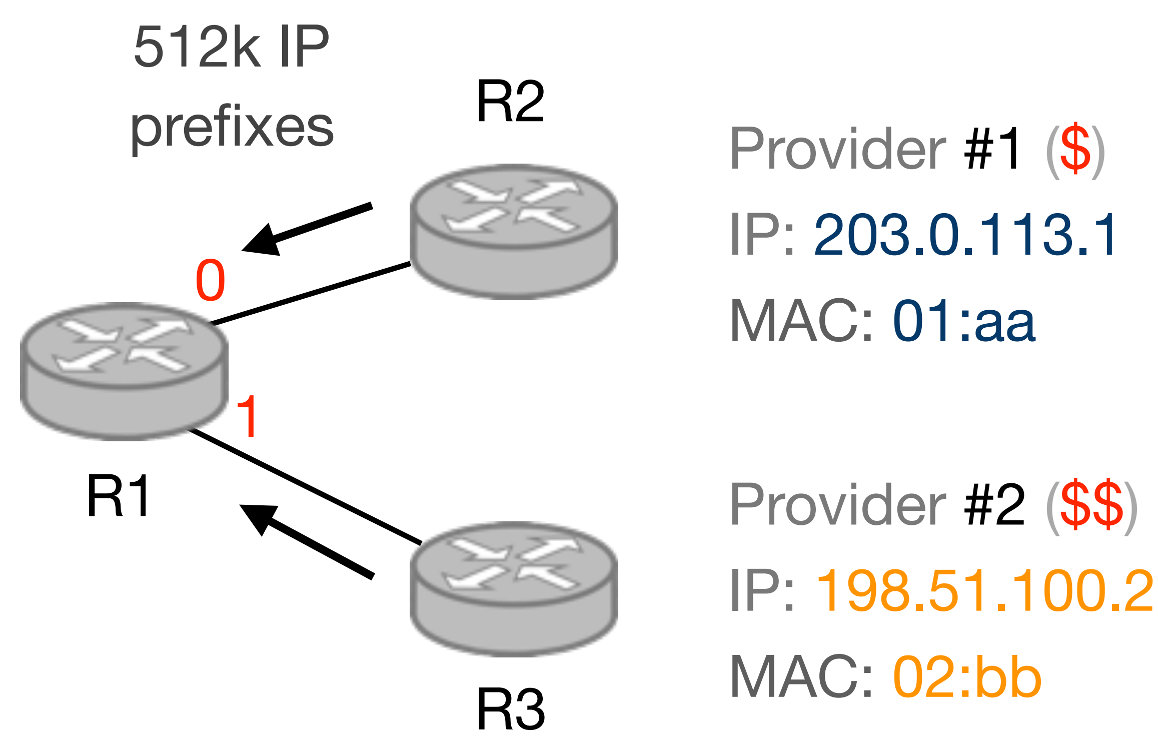

R2
0
1
R1
R3


R2
Provider #1 ($)
IP: 203.0.113.1
MAC: 01:aa
0
1
R1
Provider #2 ($$)
IP: 198.51.100.2
MAC: 02:bb
R3


512k IP
prefixes
R2
0
1
R1
R3
Provider #1 ($)
IP: 203.0.113.1
MAC: 01:aa
Provider #2 ($$)
IP: 198.51.100.2
MAC: 02:bb

R1's Forwarding Table

prefix	Next-Hop

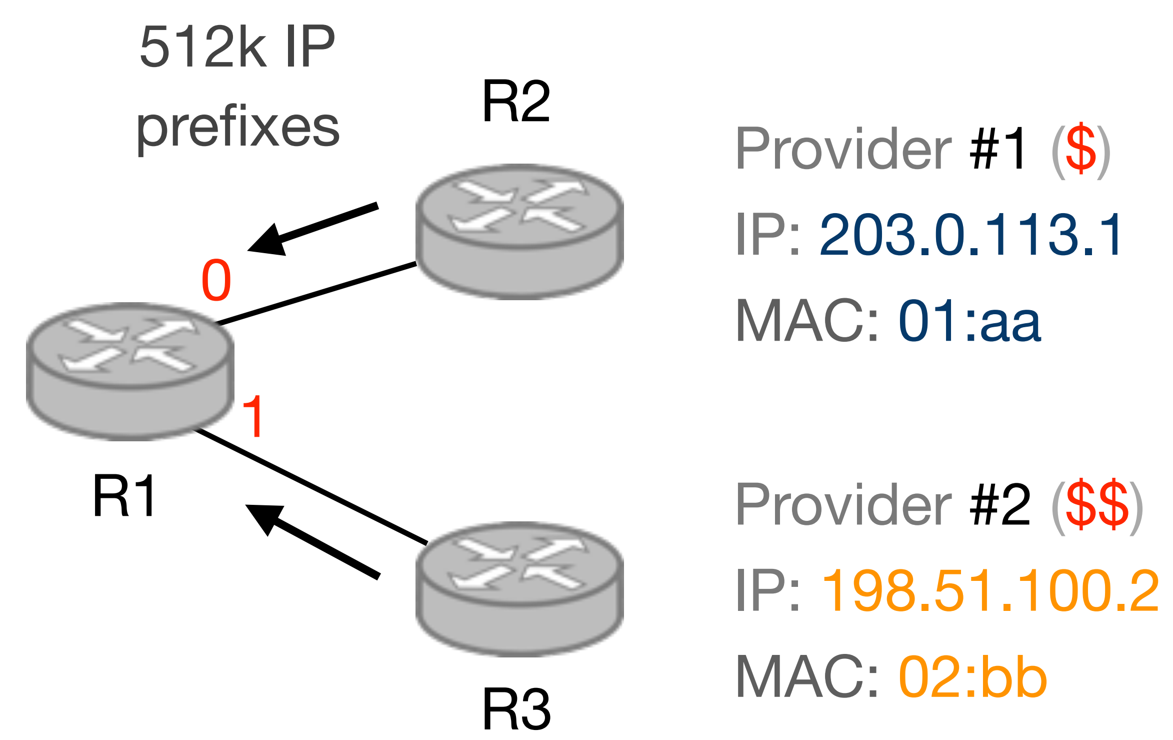


All 512k entries point to R2

because it is cheaper

R1's Forwarding Table

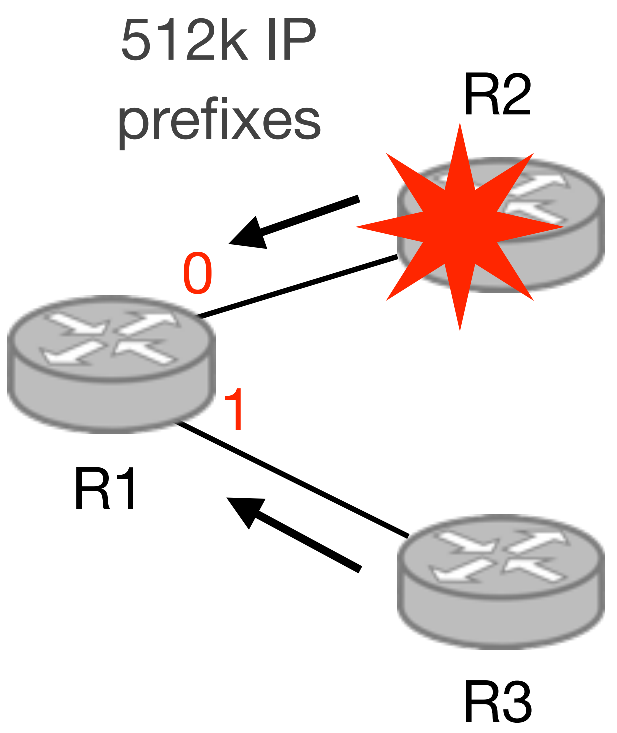
	prefix	Next-Hop
1	1.0.0.0/24	(01:aa, 0)
2	1.0.1.0/16	(01:aa, 0)
…	…	…
256k	100.0.0.0/8	(01:aa, 0)
…	…	…
512k	200.99.0.0/24	(01:aa, 0)



Upon failure of R2,
all 512k entries have to be updated

R1's Forwarding Table

	prefix	Next-Hop
1	1.0.0.0/24	(01:aa, 0)
2	1.0.1.0/16	(01:aa, 0)
…	…	…
256k	100.0.0.0/8	(01:aa, 0)
…	…	…
512k	200.99.0.0/24	(01:aa, 0)



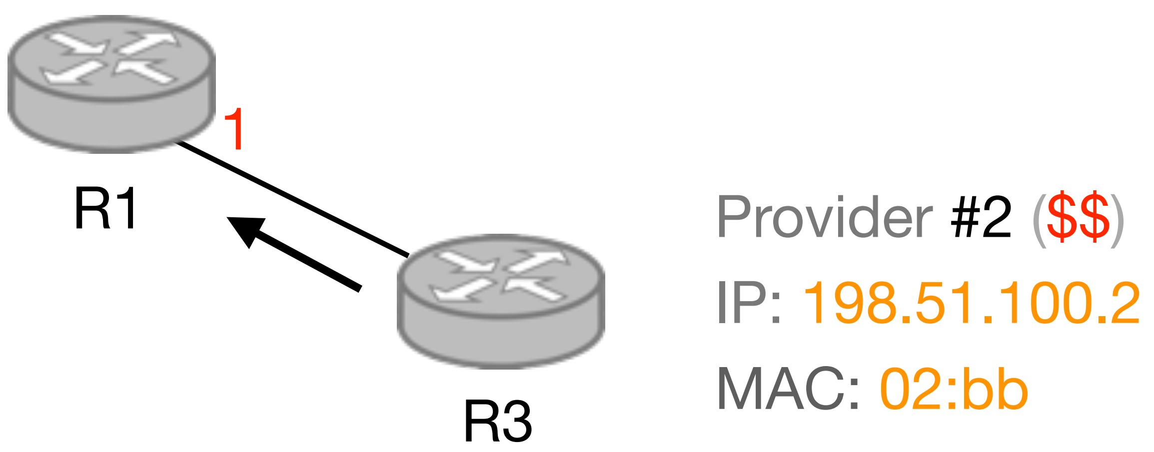
Provider #1 ($)
IP: 203.0.113.1
MAC: 01:aa

Provider #2 ($$)
IP: 198.51.100.2
MAC: 02:bb

Upon failure of R2, all 512k entries have to be updated

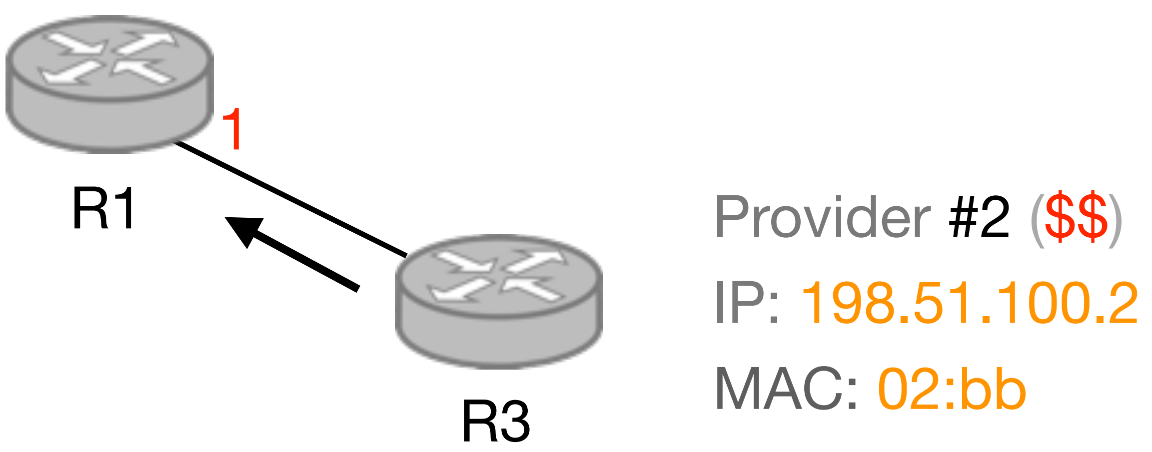
R1's Forwarding Table

	prefix	Next-Hop
1	1.0.0.0/24	(01:aa, 0)
2	1.0.1.0/16	(01:aa, 0)
…	…	…
256k	100.0.0.0/8	(01:aa, 0)
…	…	…
512k	200.99.0.0/24	(01:aa, 0)



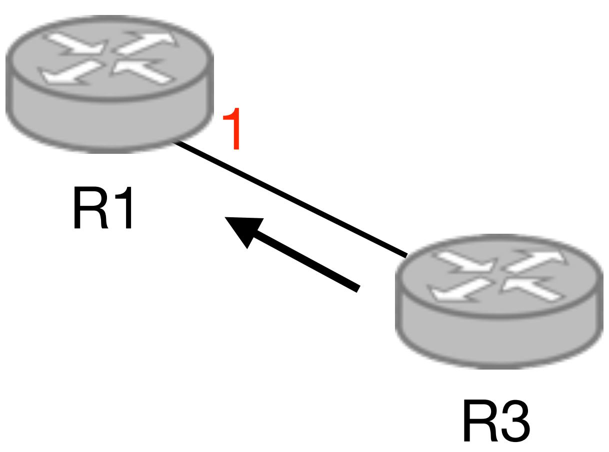
R1’s Forwarding Table

	prefix	Next-Hop
1	1.0.0.0/24	(02:bb, 1)
2	1.0.1.0/16	(01:aa, 0)
…	…	…
256k	100.0.0.0/8	(01:aa, 0)
…	…	…
512k	200.99.0.0/24	(01:aa, 0)



R1's Forwarding Table

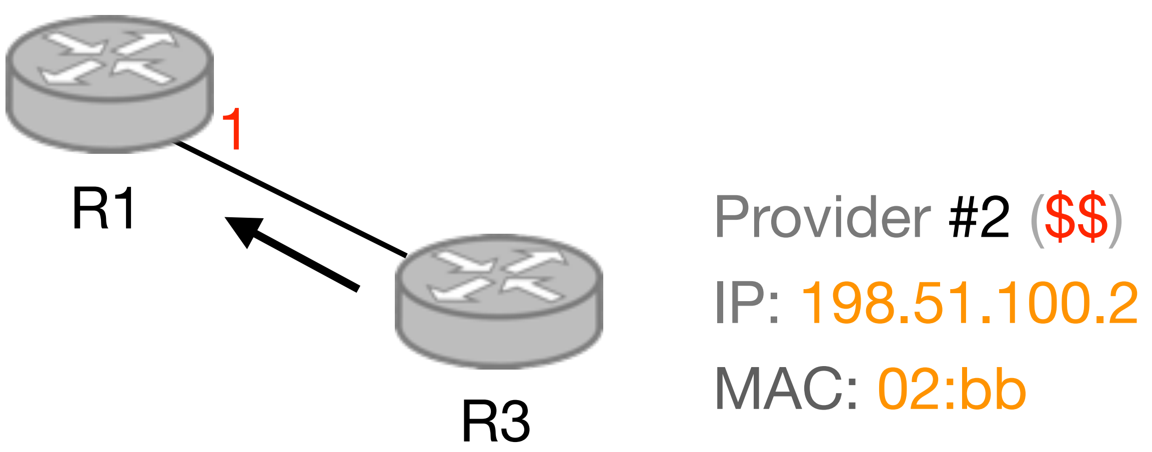
	prefix	Next-Hop
1	1.0.0.0/24	(02:bb, 1)
2	1.0.1.0/16	(02:bb, 1)
…	…	…
256k	100.0.0.0/8	(01:aa, 0)
…	…	…
512k	200.99.0.0/24	(01:aa, 0)



Provider #2 ($$)
IP: 198.51.100.2
MAC: 02:bb

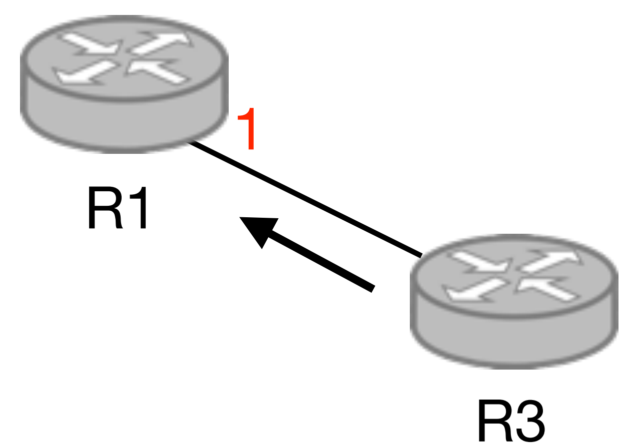
R1's Forwarding Table

	prefix	Next-Hop
1	1.0.0.0/24	(02:bb, 1)
2	1.0.1.0/16	(02:bb, 1)
…	…	…
256k	100.0.0.0/8	(02:bb, 1)
…	…	…
512k	200.99.0.0/24	(01:aa, 0)



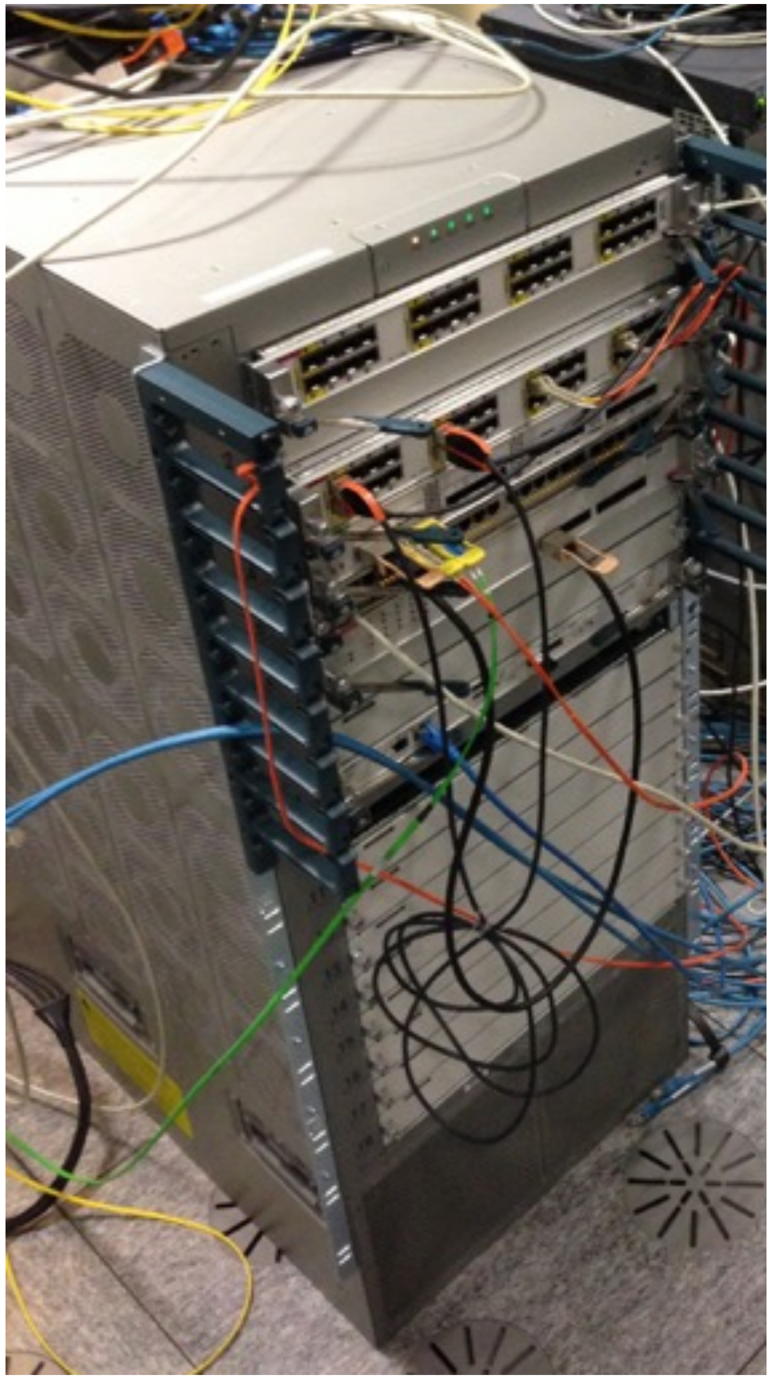
R1's Forwarding Table

	prefix	Next-Hop
1	1.0.0.0/24	(02:bb, 1)
2	1.0.1.0/16	(02:bb, 1)
…	…	…
256k	100.0.0.0/8	(02:bb, 1)
…	…	…
512k	200.99.0.0/24	(02:bb, 1)



Provider #2 ($$)
IP: 198.51.100.2
MAC: 02:bb

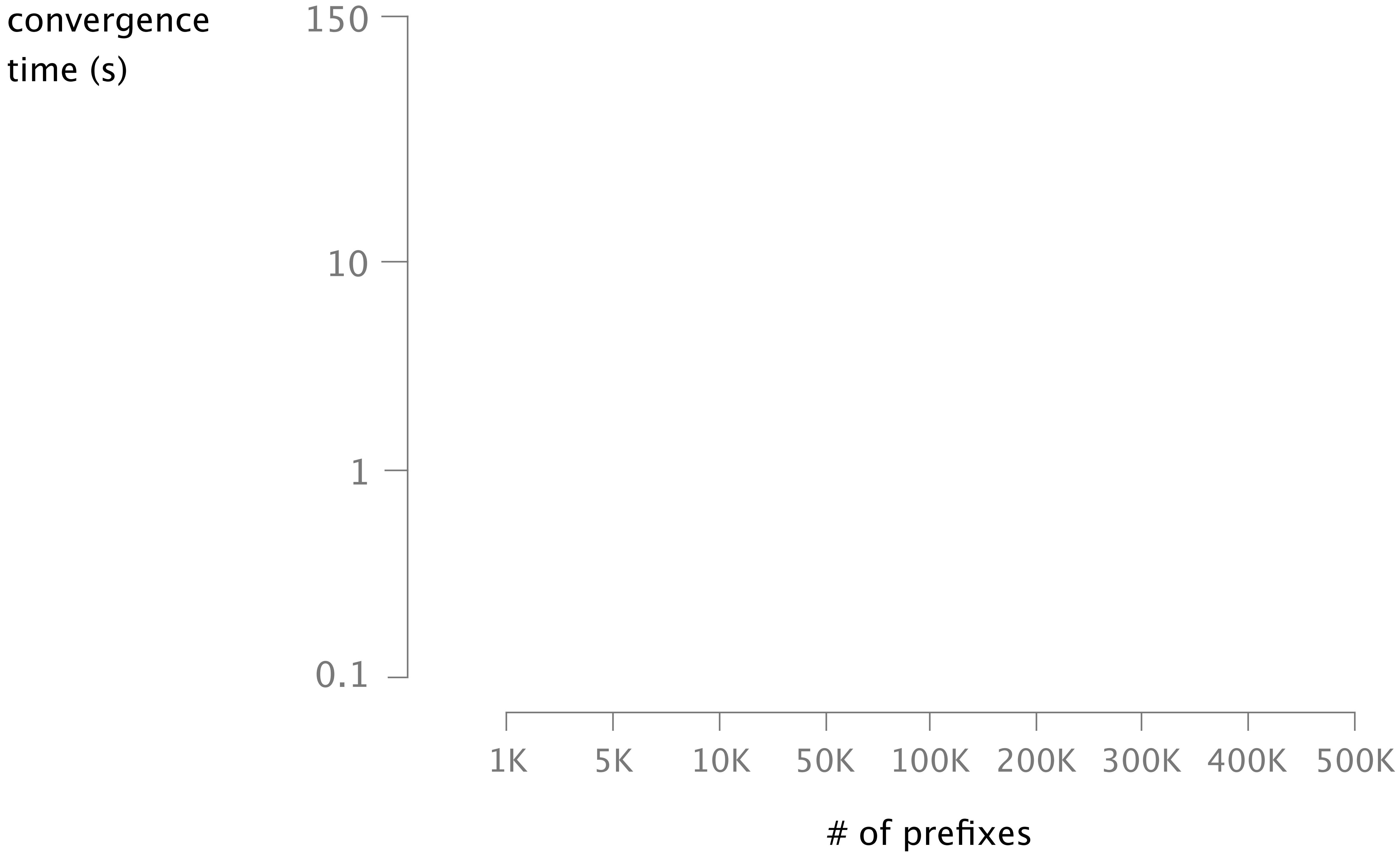
We measured how long it takes
in our home network

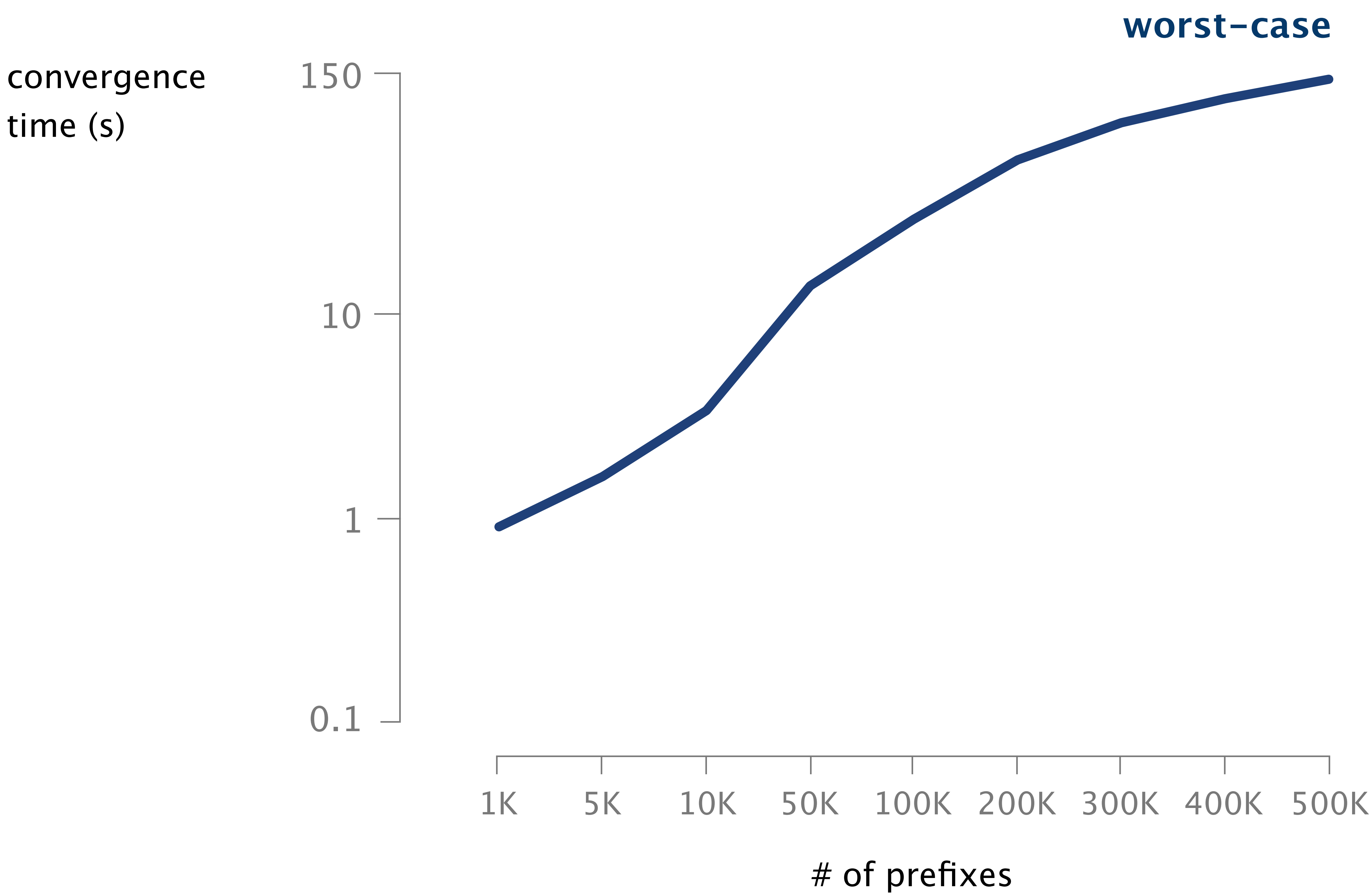


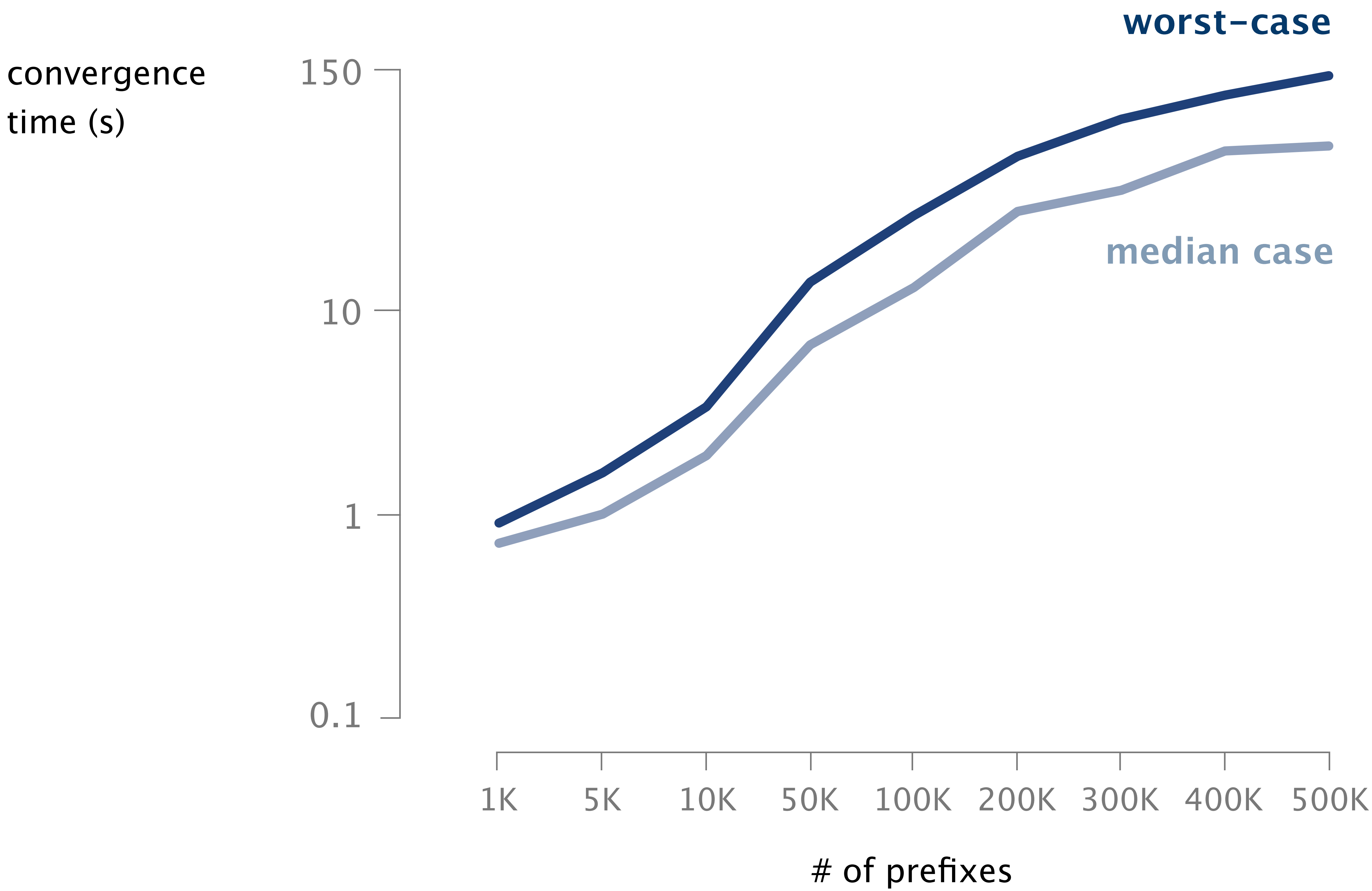
Cisco Nexus 9k

ETH recent routers

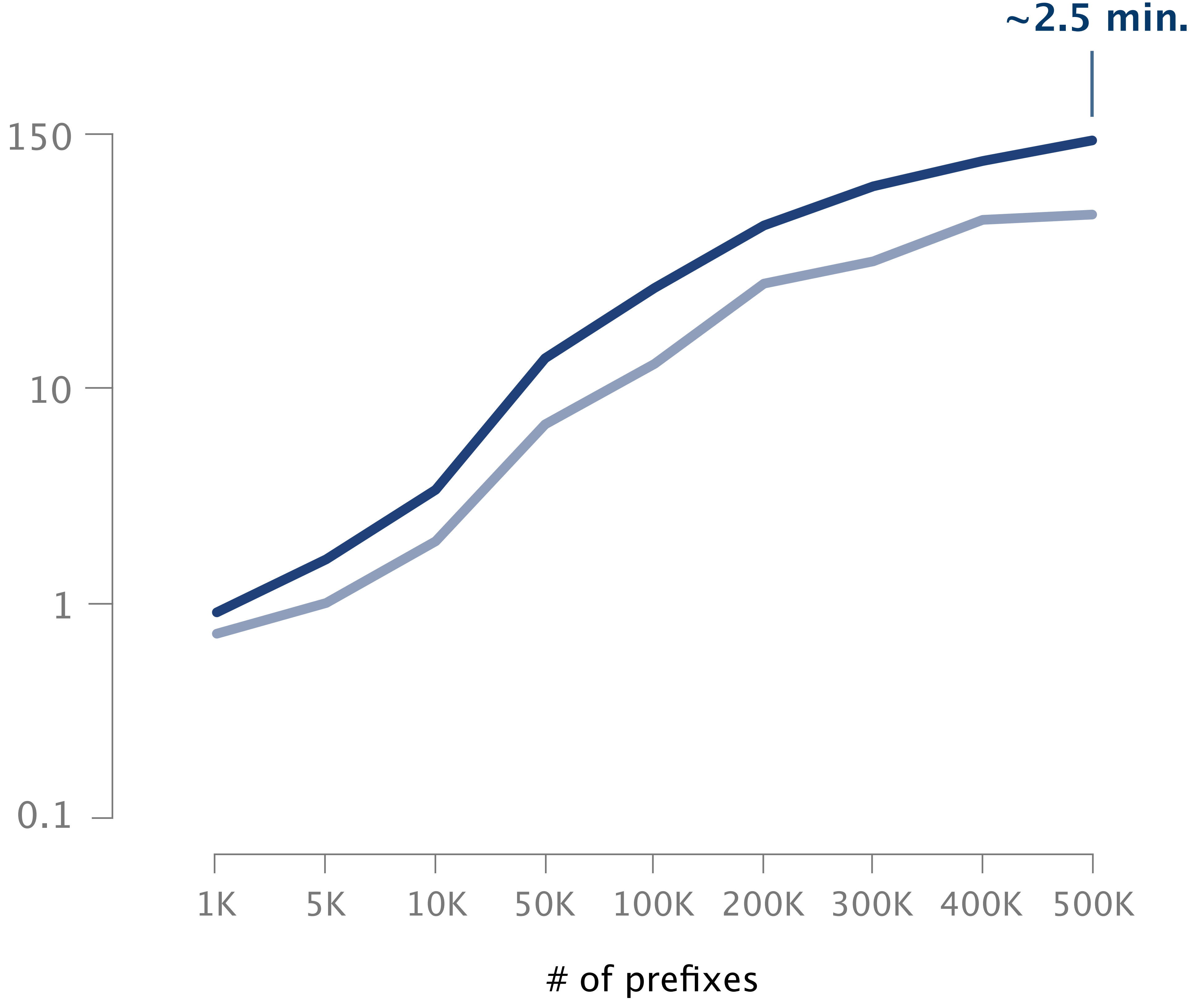
25 deployed


convergence
time (s)
150
10
1
0.1
1K
5K
10K
50K
100K
200K
300K
400K
500K
# of prefixes


worst-case
convergence
time (s)
150
10
1
0.1
1K
5K
10K
50K
100K
200K
300K
400K
500K
# of prefixes


worst-case
convergence
time (s)
150
10
1
0.1
median case
1K
5K
10K
50K
100K
200K
300K
400K
500K
# of prefixes

Traffic can be lost for several minutes



The problem is that forwarding tables are flat

Entries do not share any information

even if they are identical

Upon failure, all of them have to be updated

inefficient, but also unnecessary

The problem is that forwarding tables are flat

Entries do not share any information

even if they are identical

Upon failure, all of them have to be updated

inefficient, but also unnecessary

Solution: introduce a hierarchy

as with any problem in CS…

replace this…

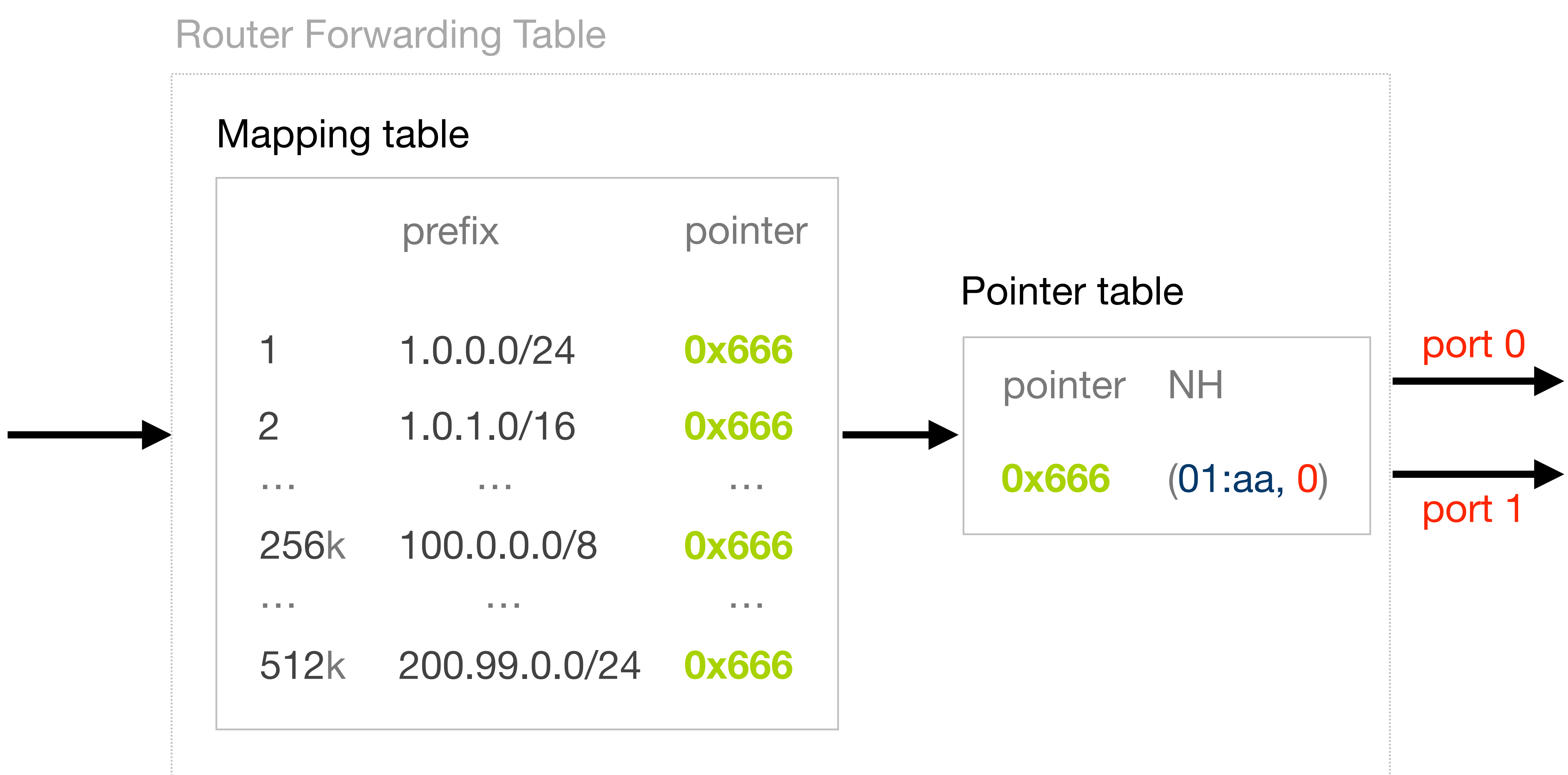
Router Forwarding Table

	prefix	Next-Hop
1	1.0.0.0/24	(01:aa, 0)
2	1.0.1.0/16	(01:aa, 0)
…	…	…
256k	100.0.0.0/8	(01:aa, 0)
…	…	…
512k	200.99.0.0/24	(01:aa, 0)

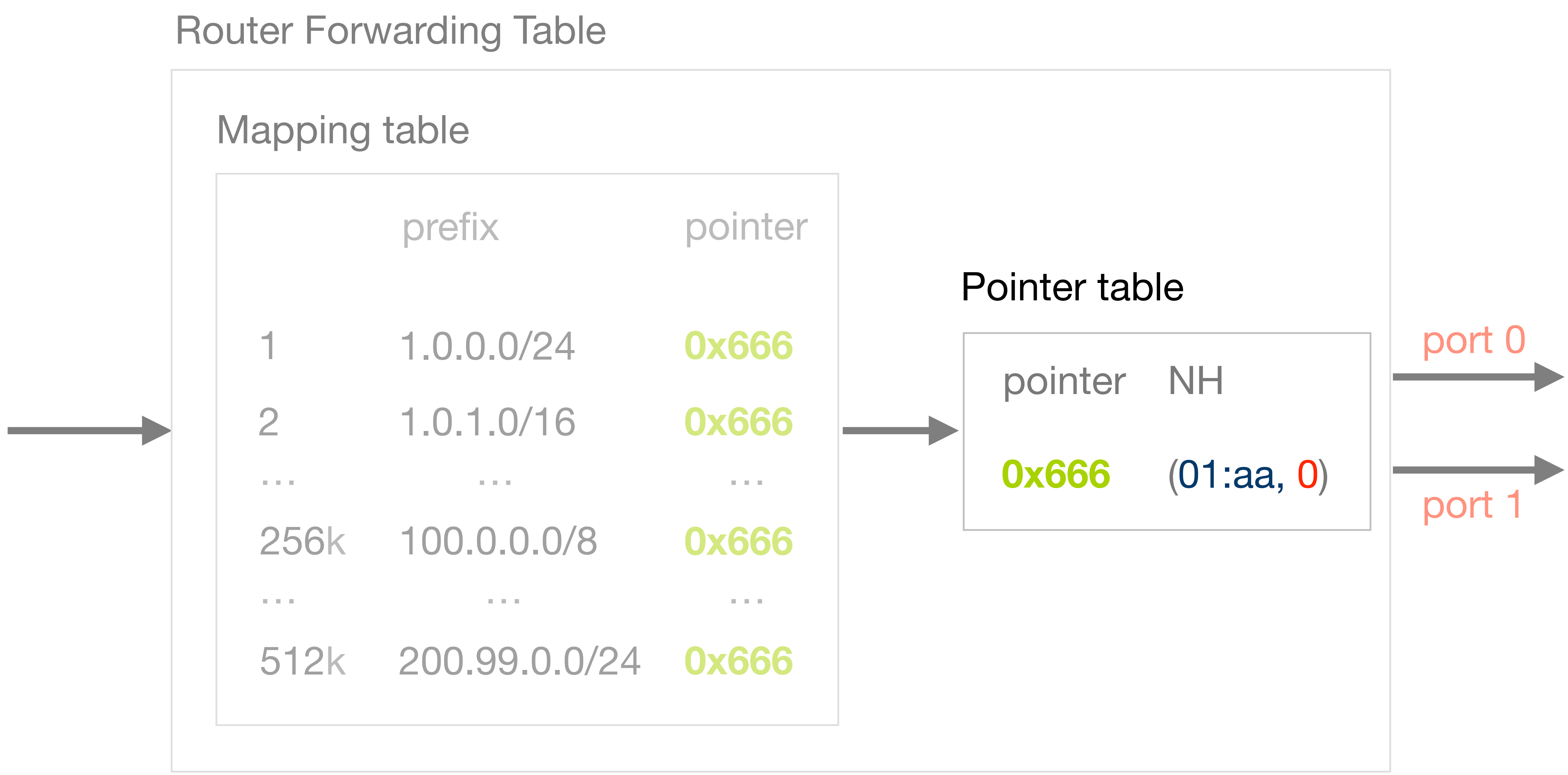
port 0

port 1

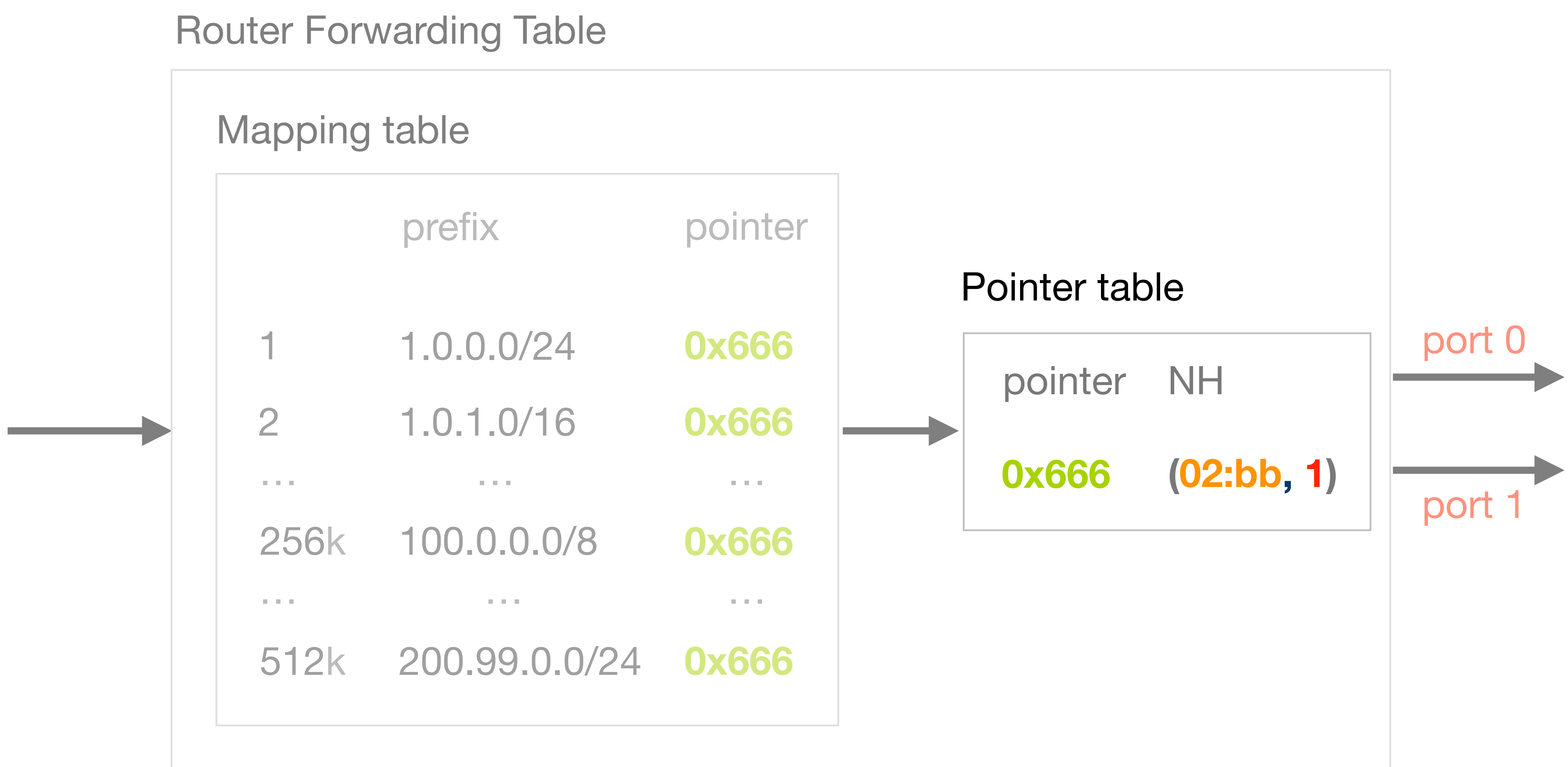
… with that


Router Forwarding Table
Mapping table
prefix pointer
1 1.0.0.0/24 0x666
2 1.0.1.0/16 0x666
… … …
256k 100.0.0.0/8 0x666
… … …
512k 200.99.0.0/24 0x666
Pointer table
pointer NH
0x666 (01:aa, 0)
port 0
port 1

Upon failures, we update the pointer table



Here, we only need to do one update



Nowadays, only high-end routers have hierarchical forwarding table

Expensive

by orders of magnitude

Limited availability

only a few vendors, on few models

Limited benefits

of fast convergence, if not used network-wide

We can build a hierarchical table

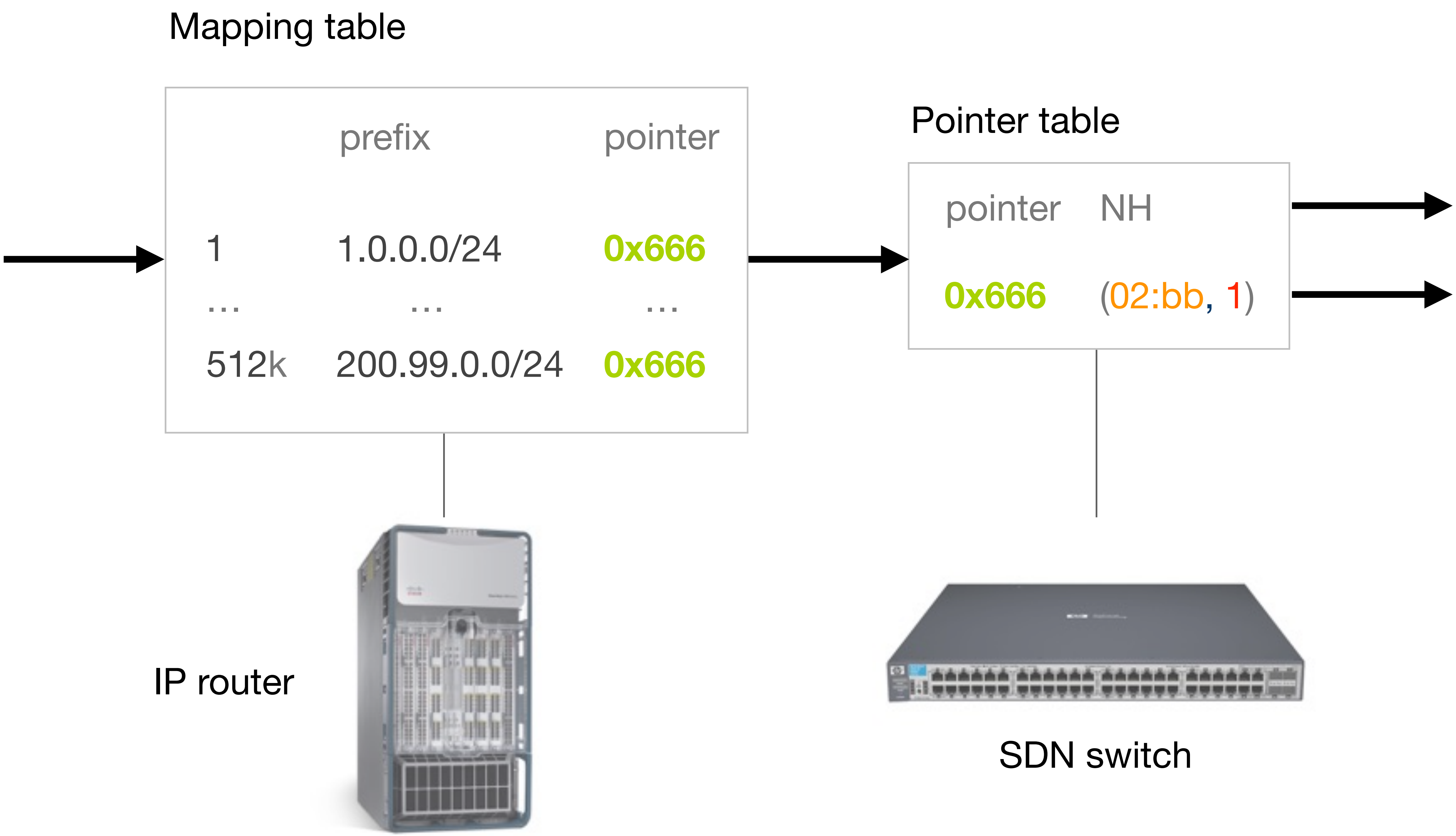
Mapping table

	prefix	pointer
1	1.0.0.0/24	**0x666**
…	…	…
512k	200.99.0.0/24	**0x666**

Pointer table

pointer	NH
0x666	(02:bb, 1)

We can build a hierarchical table using two adjacent devices



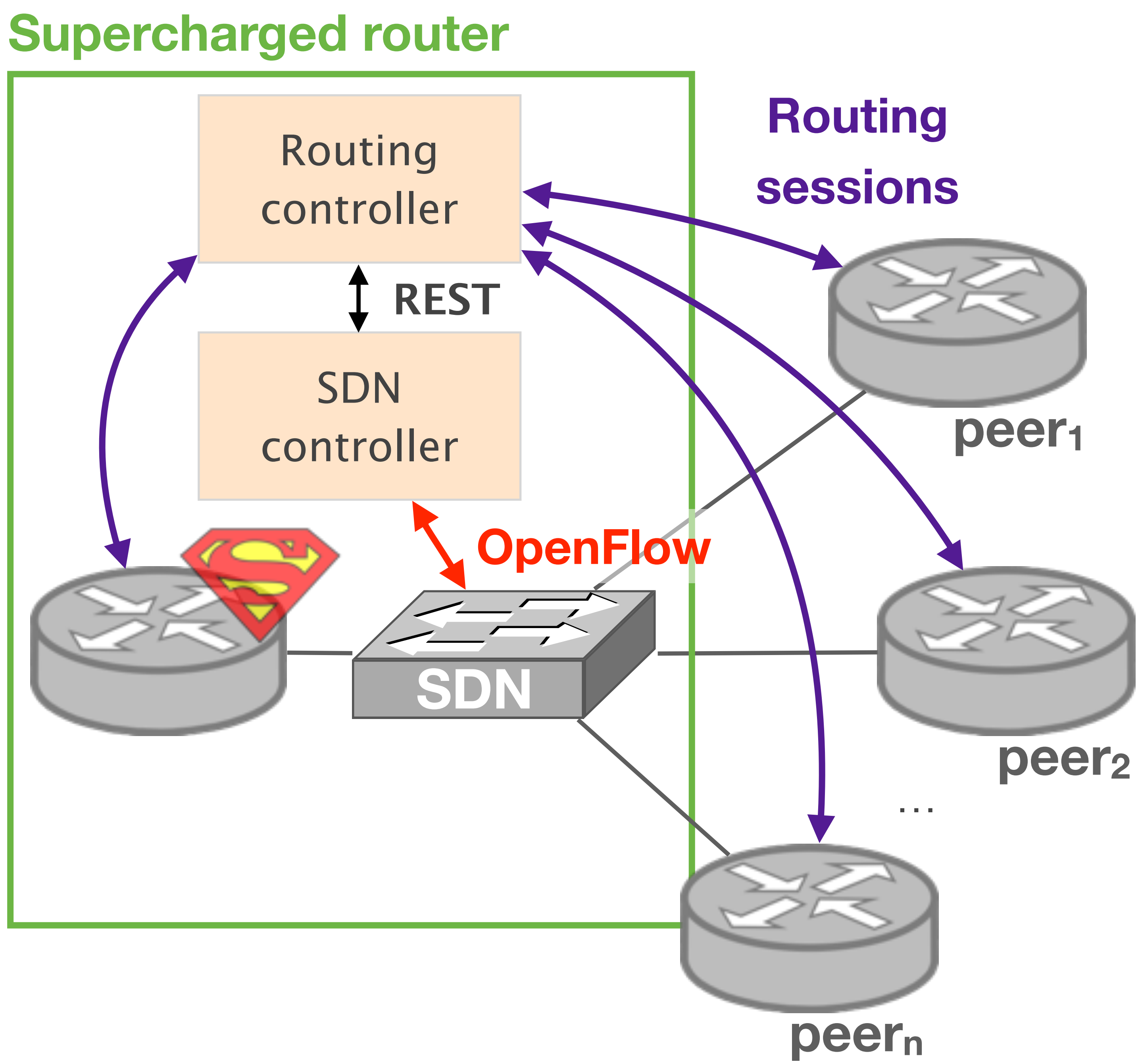
Supercharged

Supercharged

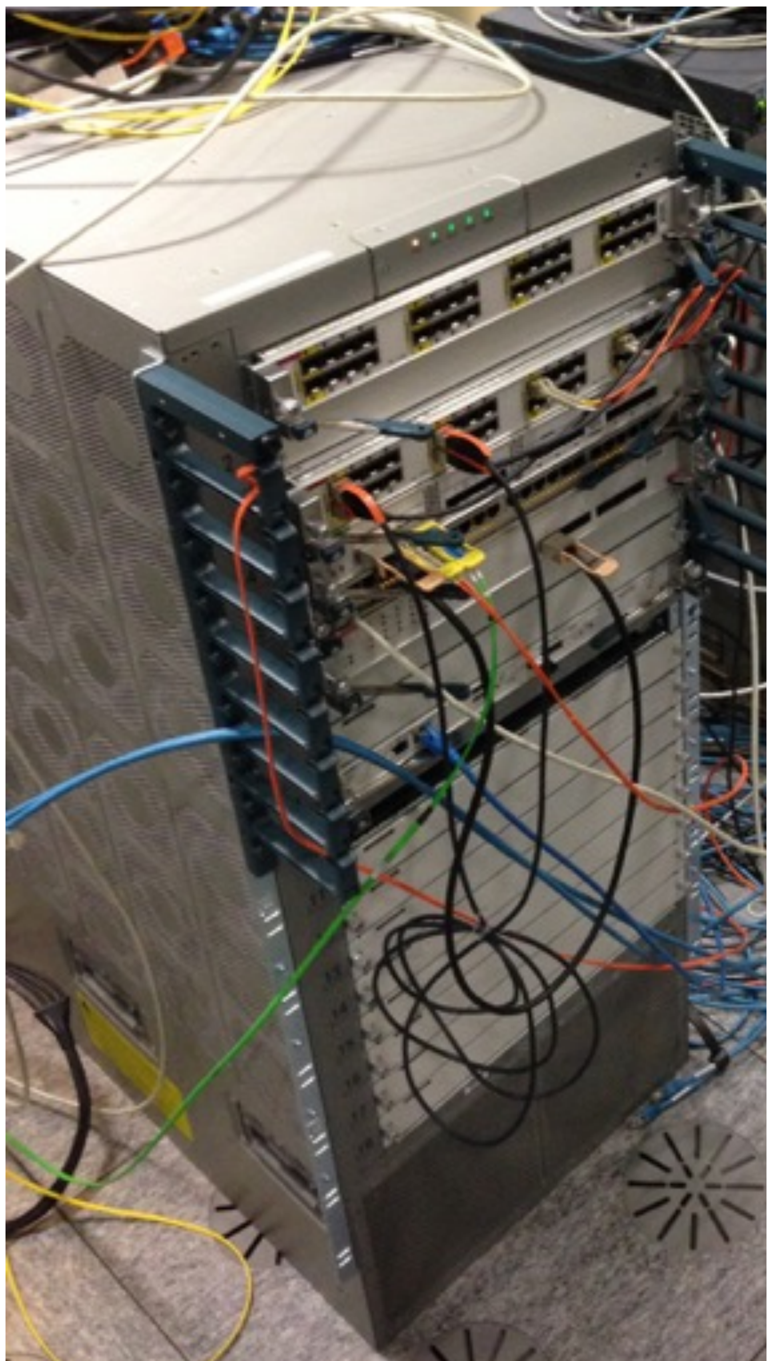
boost routers performance

by **combining** them with **SDN** devices

We have implemented a fully-functional "router supercharger"



We used it to supercharge the same router as before



Cisco Nexus 9k

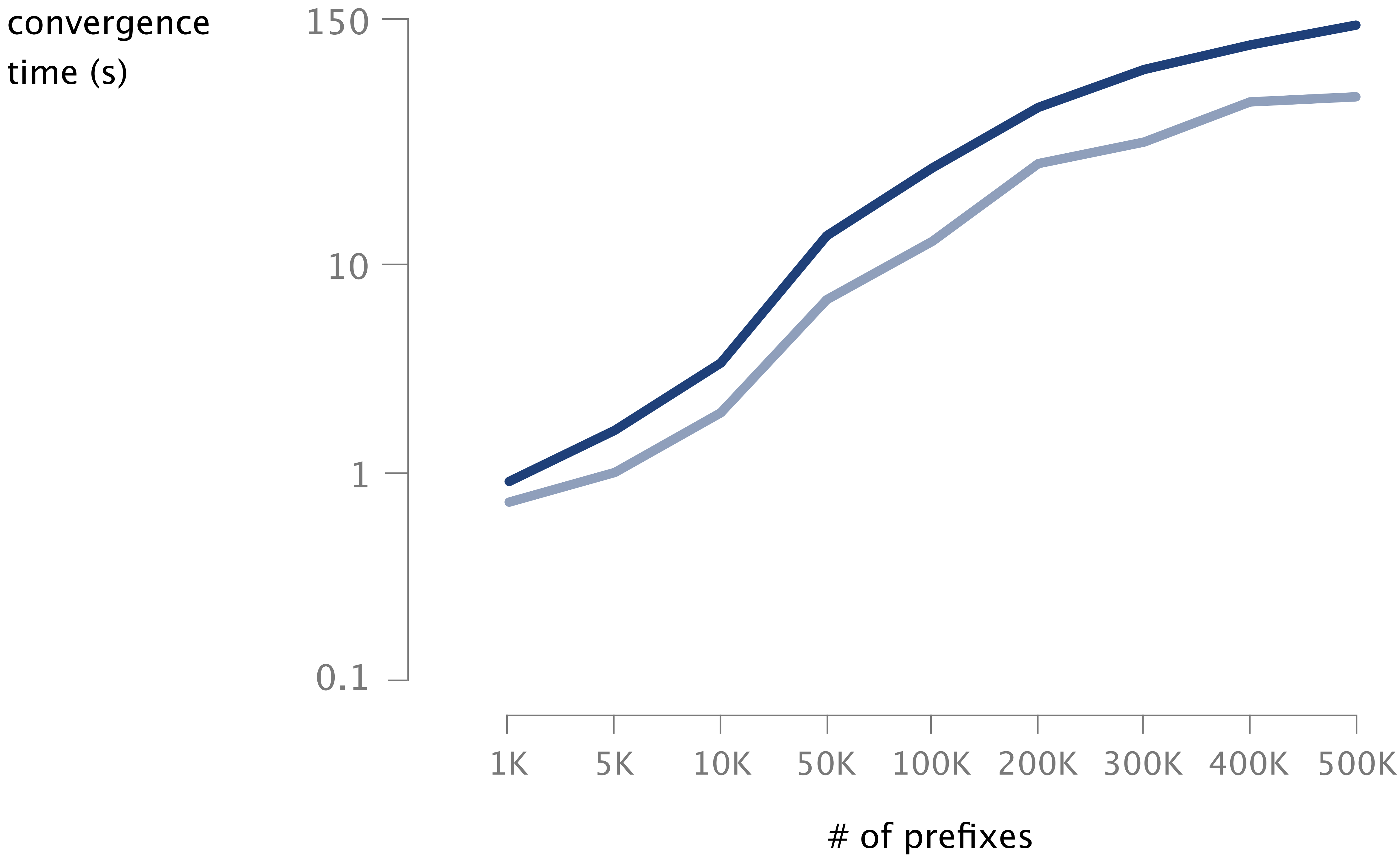
ETH recent routers

25 deployed

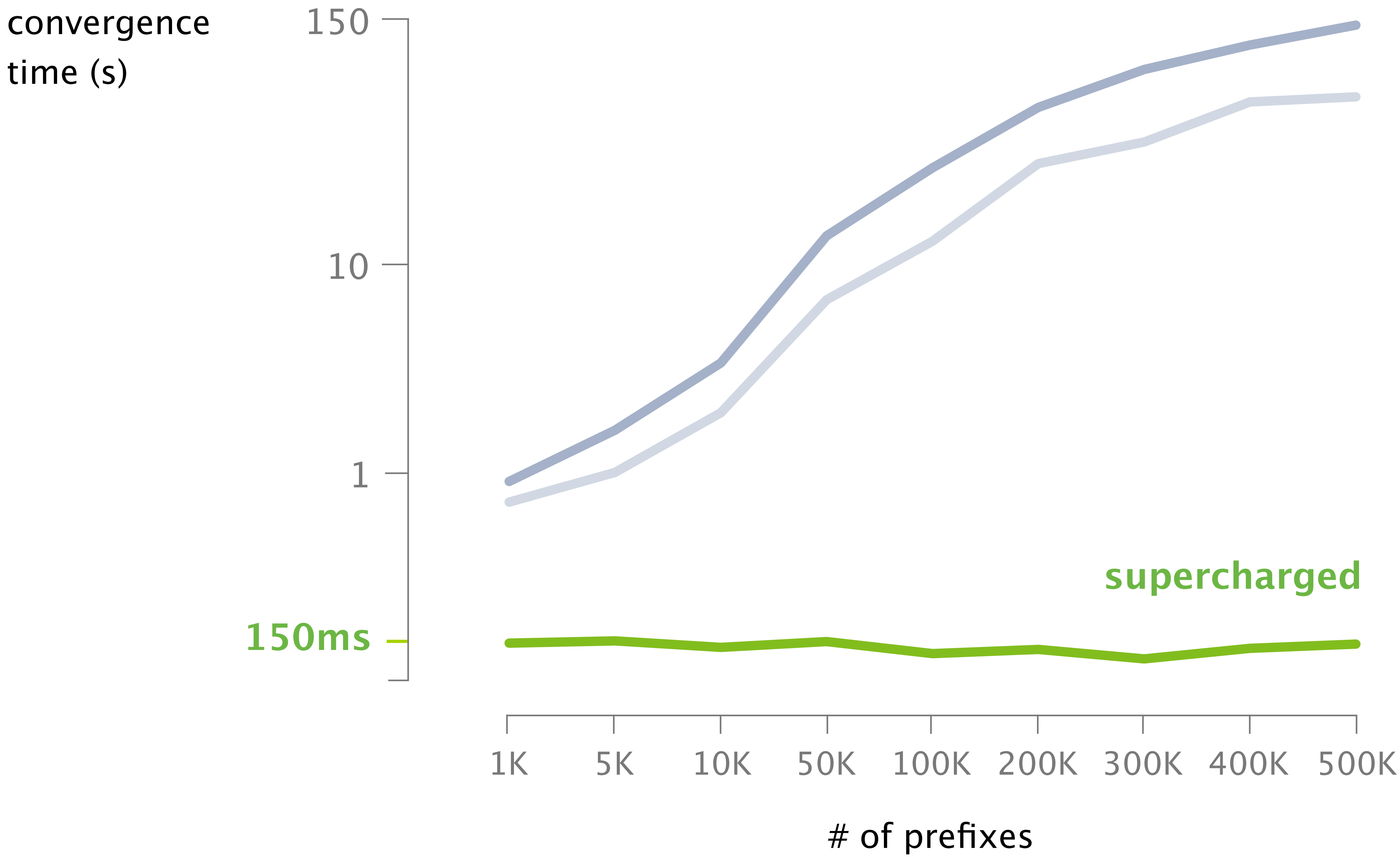
+ (old) SDN HP switch

~2k$ cost

While the router took more than 2 min to converge in the worst-case



The supercharged router systematically converged within 150ms



Other aspects of a router performance can be supercharged

- convergence time

  systematic sub-second convergence

- memory size

  offload to SDN if no local forwarding entry

- bandwidth management

  overwrite poor routers decisions

This talk was about two SDN-based technologies

that improve today's networks

Fibbing

improved flexibility

central control over distributed system

Supercharged

performance boost

reduce convergence time by 1000x

Boosting existing networks with SDN

A bird in the hand is worth two in the bush

Laurent Vanbever

www.vanbever.eu

Swisscom Innovation

May, 28 2015