Network programmability

A primer on routing synthesis

Laurent Vanbever

ETH Zürich (D-ITET)

Schloss Dagstuhl

January 18 2017

Human factors are responsible

for 50% to 80% of network outages

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

JUL 8, 2015 @ 09:36 PM 11,261 VIEWS

United Airlines Blames Router for Grounded Flights

Alexandra Talty, CONTRIBUTOR

I cover personal finance and travel.

FOLLOW ON FORBES (110)

Opinions expressed by Forbes Contributors are their own.

FULL BIO

After a computer problem caused nearly two hours of grounded flights for United Airlines this morning and ongoing delays throughout the day, the airline announced the culprit: a faulty router.

Spokeswoman Jennifer Dohm said that the router problem caused "degraded network connectivity," which affected various applications.

A computer glitch in the airline's reservations system caused the Federal Aviation Administration to impose a groundstop at 8:26 a.m. E.T. Planes that were in the air continued to operate, but all planes on the ground were held. There were reports of agents writing tickets by hand. The ground stop was lifted around 9:47 a.m. ET.

The outage was due to

one faulty Internet device

Facebook, Tinder, Instagram suffer widespread issues

IMAGE: GETTY IMAGES

BY JENNI RYALL
AUSTRALIA

JAN 27, 2015

***UPDATED**: Tuesday, Jan. 27 / 4:32 a.m. EST* — A Facebook spokeswoman told *Mashable* that the outage was due to a change to the site's configuration systems, and not a hacker attack. "Earlier this evening many people had trouble accessing Facebook and Instagram. This was not the result of a third party attack but instead occurred after we introduced a change that affected our configuration systems. We moved quickly to fix the problem, and both services are back to 100% for everyone.", she said.

***UPDATED**: Tuesday, Jan. 27 / 2:14 a.m. EST* — Facebook, Tinder and Twitter appear to be back to normal after a 40 minute outage and mass freak out.

The outage was due to a change to the site's configuration systems

Traders work on the floor of the New York Stock Exchange (NYSE) in July 2015. (Photo by Spencer Platt/Getty Images)

DOWNTIME

UPDATED: "Configuration Issue" Halts Trading on NYSE

The article has been updated with the time trading resumed.

A second update identified the cause of the outage as a "configuration issue."

A third update added information about a software update that created the configuration issue.

NYSE network operators identified the culprit of the 3.5 hour outage, blaming the incident on a "network configuration issue"

The Internet Under Crisis Conditions

Learning from September 11

Committee on the Internet Under Crisis Conditions:
Learning from September 11

Computer Science and Telecommunications Board
Division on Engineering and Physical Sciences

NATIONAL RESEARCH COUNCIL
OF THE NATIONAL ACADEMIES

National Research Council. The Internet Under Crisis Conditions: Learning from September 11

The Internet Under Crisis Conditions

Learning from September 11

Committee on the Internet Under Crisis Conditions:
Learning from September 11

Computer Science and Telecommunications Board
Division on Engineering and Physical Sciences

NATIONAL RESEARCH COUNCIL
OF THE NATIONAL ACADEMIES

Internet advertisements rates

suggest that

The Internet was more stable

than normal on Sept 11

Internet advertisements rates

suggest that

The Internet was more stable

than normal on Sept 11

Information suggests that

operators were watching the news

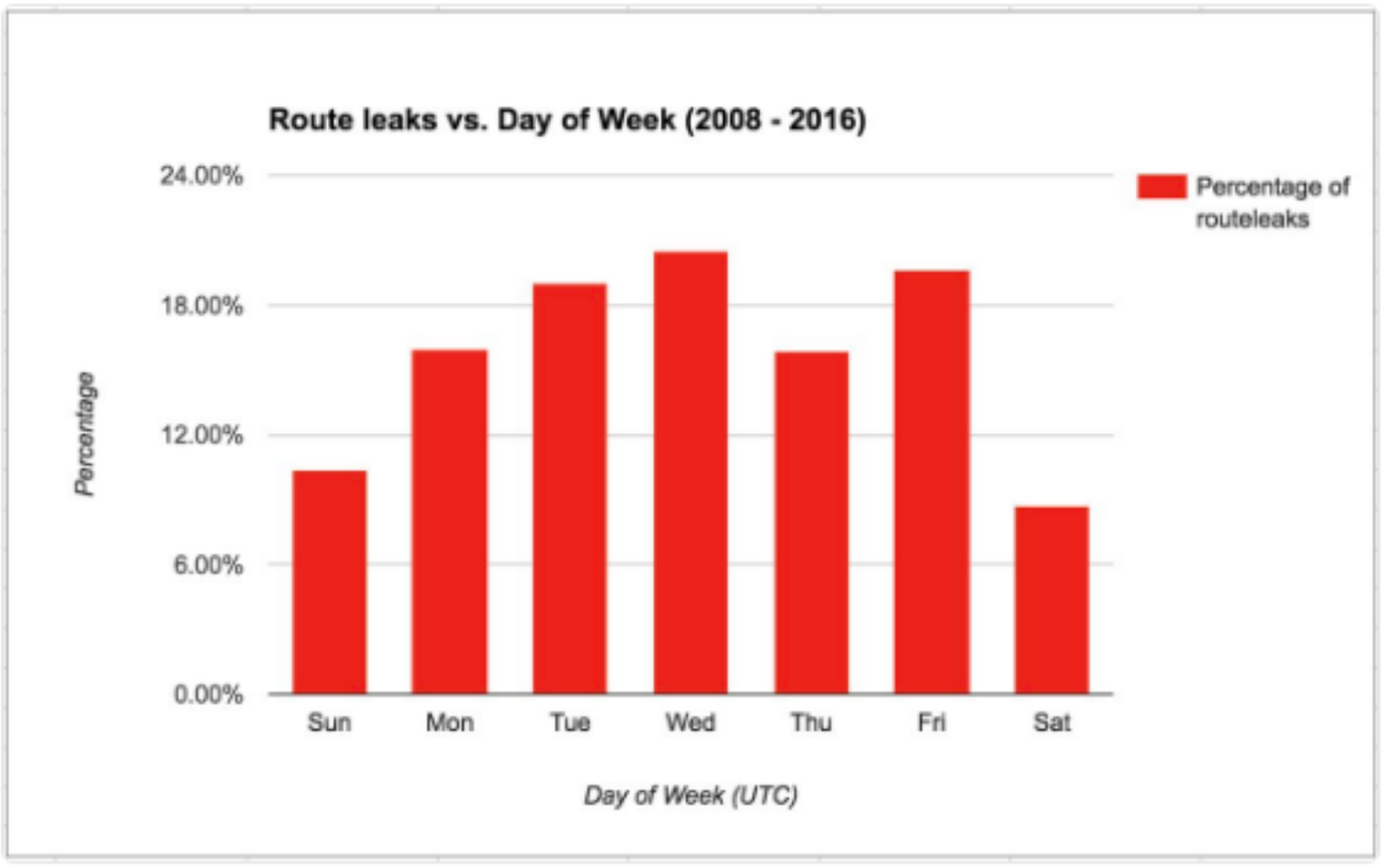
instead of making changes

to their infrastucture

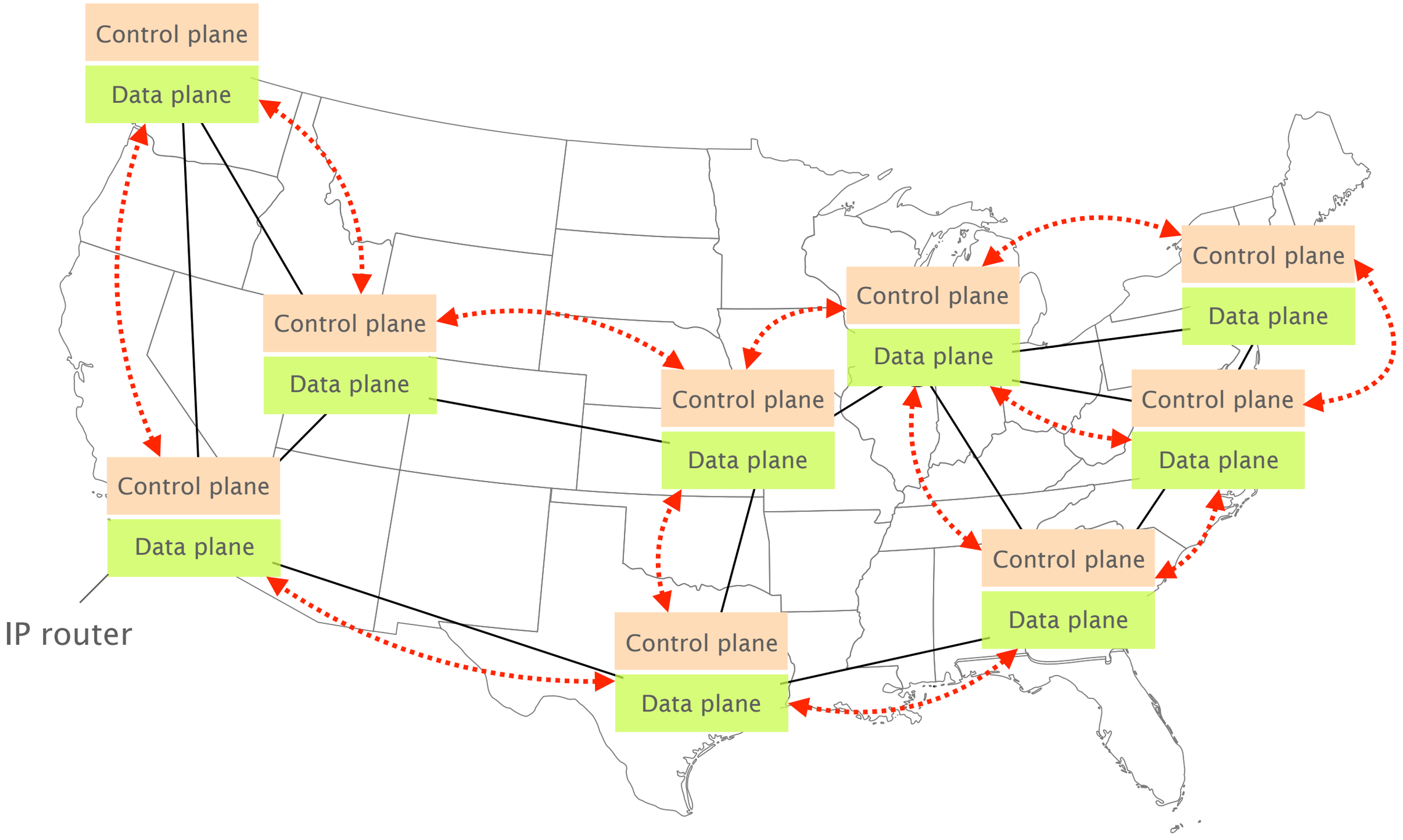
Job Snijders
@JobSnijders

Follow

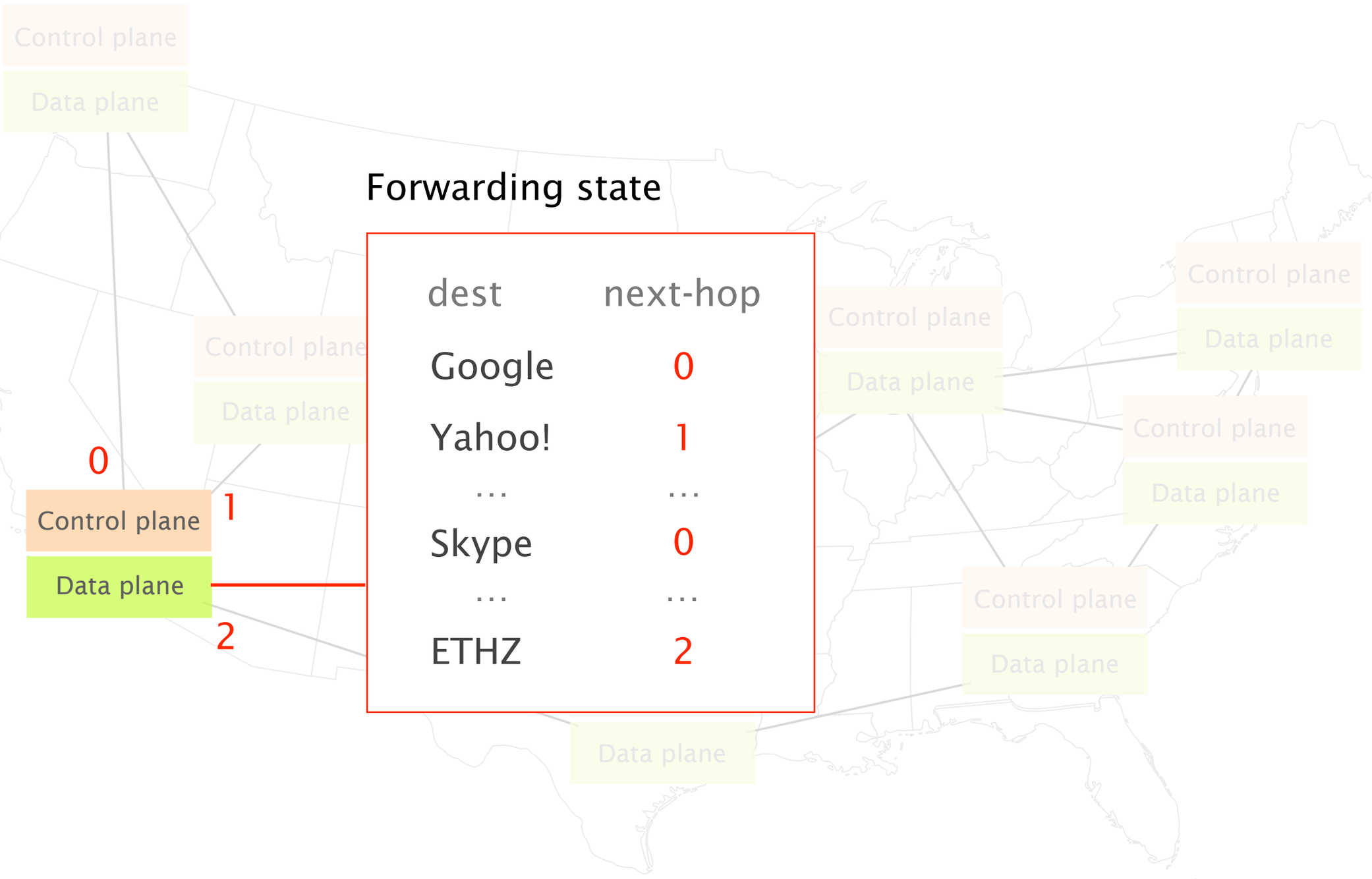
Fun fact: most BGP route leaks happen on Wednesdays, but in the weekend us humans collectively take a break! :-)



Think of the network as a distributed system running a distributed algorithm



This algorithm produces the forwarding state which drives Internet traffic to its destination



Operators adapt their network forwarding behavior by configuring each network device individually

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 {
```

A single mistyped line is enough to bring down the entire network

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
```

Anything else than 700 creates blackholes

Juniper JunOS

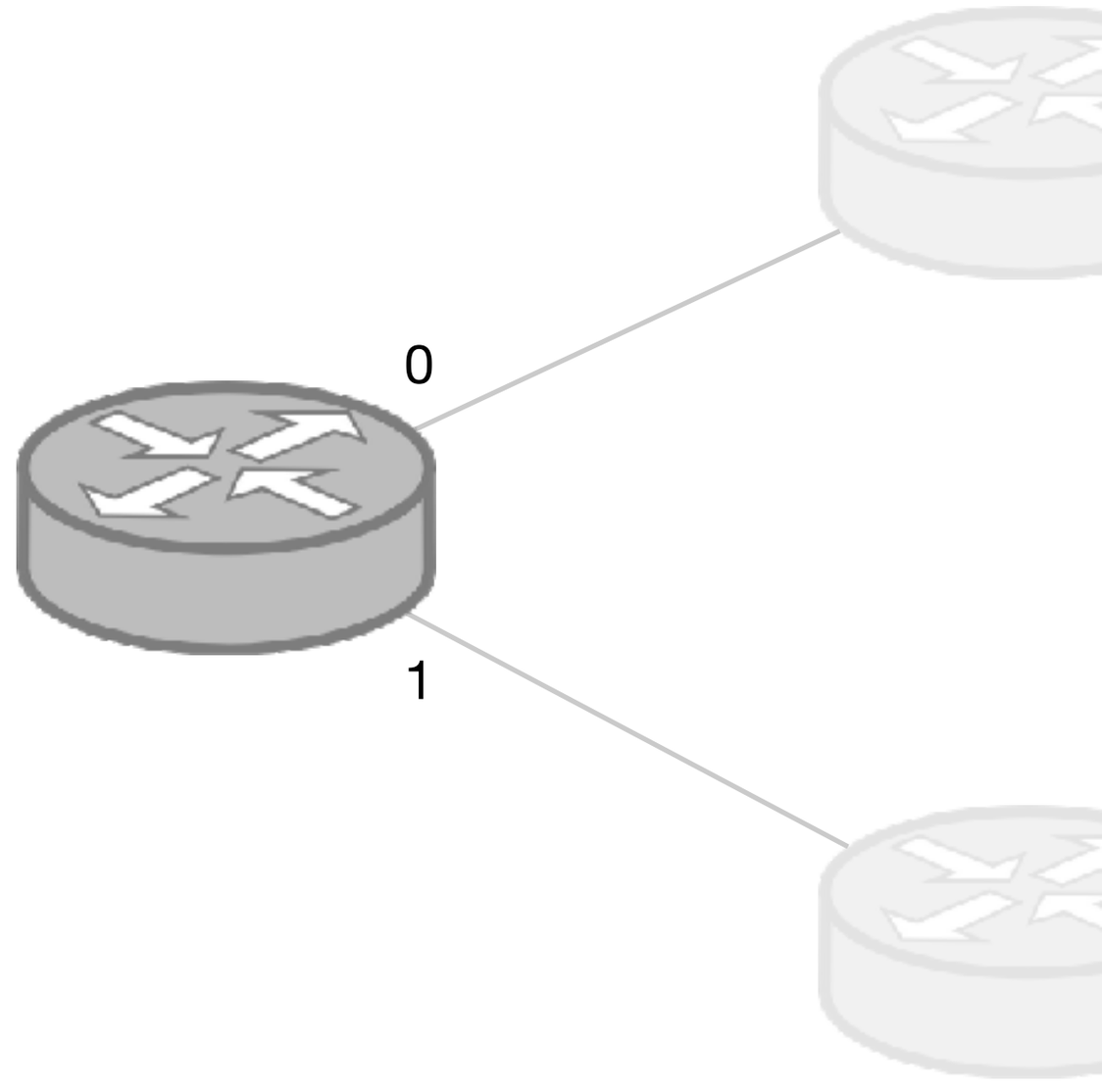
```
interfaces {
   so-0/0/0 {
        unit 0 {
            family inet {
                address 10.12.1.2/24;
            }
            family mpls;
        }
    }
   ge-0/1/0 {
        vlan-tagging;
        unit 0 {
            vlan-id 100;
            family inet {
                address 10.108.1.1/24;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

How can we program network-wide forwarding state in existing networks?

The forwarding state computed by a router depends on two inputs

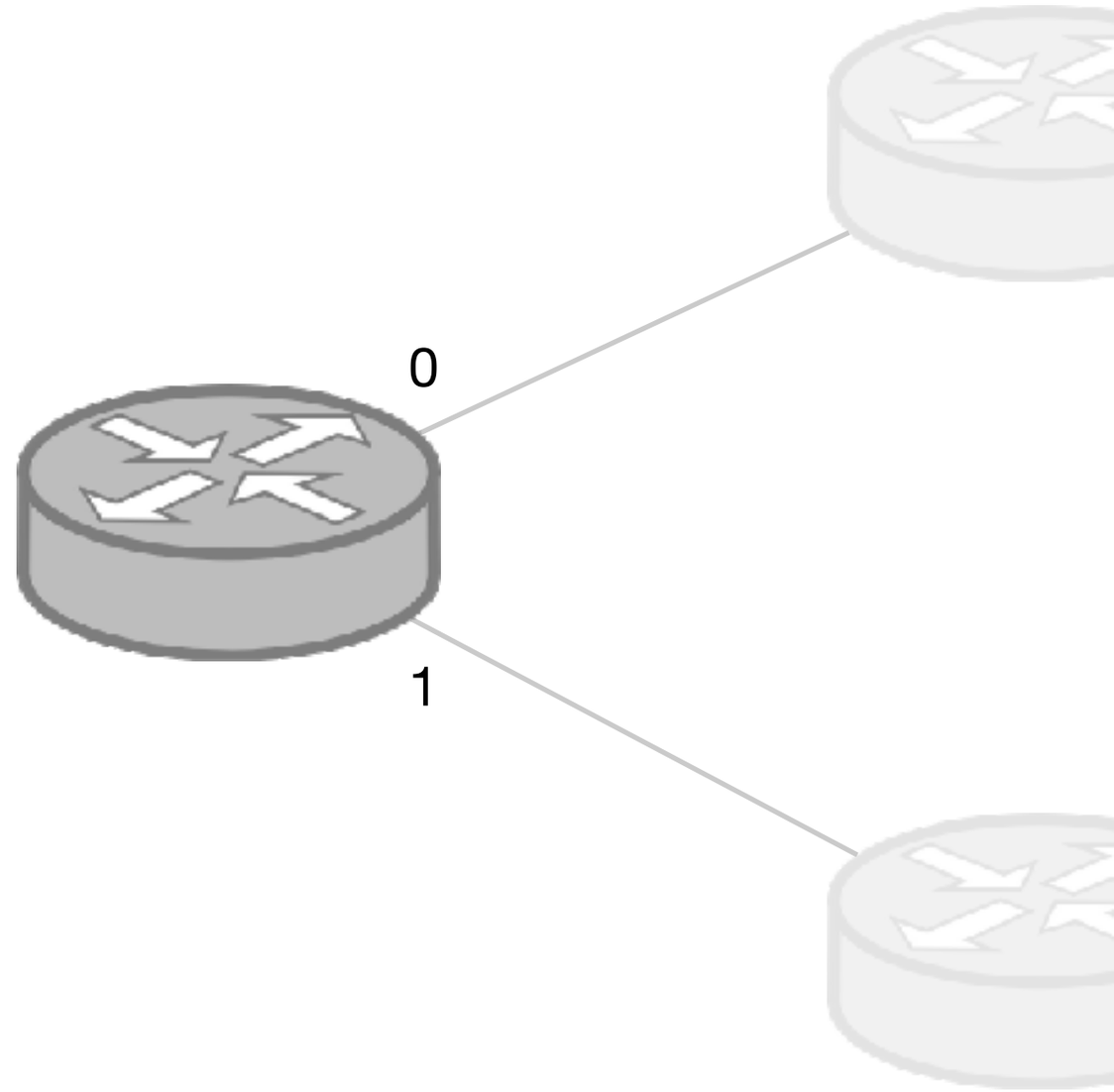
Forwarding state

	prefix	next-hop
1	1.0.0.0/24	0
2	1.0.1.0/16	1
…	…	…
300k	100.0.0.0/8	0
…	…	…
600k	200.99.0.0/24	1



The router configuration specifies how the router compute its state

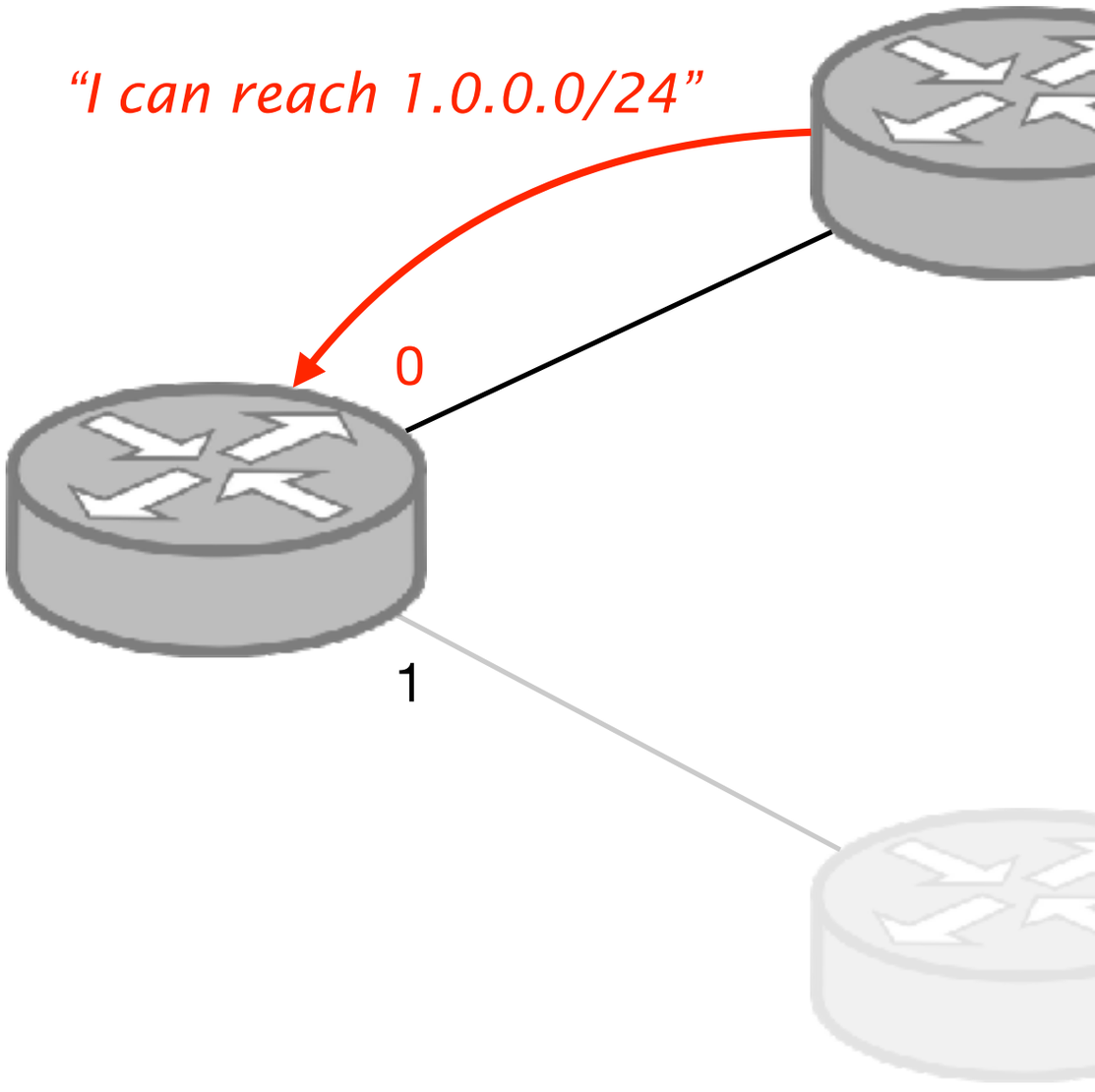
```
!
ip multicast-routing
!
interface Loopback0
 ip address 120.1.7.7 255.255.255.255
 ip ospf 1 area 0
!
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.17
 encapsulation dot1Q 17
 ip address 125.1.17.7 255.255.255.0
 ip pim bsr-border
 ip pim sparse-mode
!
!
router ospf 1
 router-id 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
```



The routing messages sent by neighboring devices

Forwarding state

	prefix	next-hop
1	1.0.0.0/24	0
2	1.0.1.0/16	1
…	…	…
300k	100.0.0.0/8	0
…	…	…
600k	200.99.0.0/24	1



Given a forwarding state we want to program,
we therefore have two ways to provision it

Given a network-wide forwarding state
to provision, one can synthesize

way 1 the routing messages shown to the routers

way 2 the configurations run by the routers

Given a network-wide forwarding state

output to provision, one can **synthesize**

inputs the routing messages shown to the routers

functions the configurations run by the routers

Network programmability

through synthesis

Fibbing

“the inputs”

SyNET

“the functions”

Network programmability

through synthesis

Fibbing

"the inputs"

SyNET

"the functions"

[SIGCOMM'15]

Joint work with:

Stefano Vissicchio, Olivier Tilmans and Jennifer Rexford

Fibbing

Fibbing

= lying

Fibbing

to **control** router's forwarding table

Central Control Over Distributed Routing

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

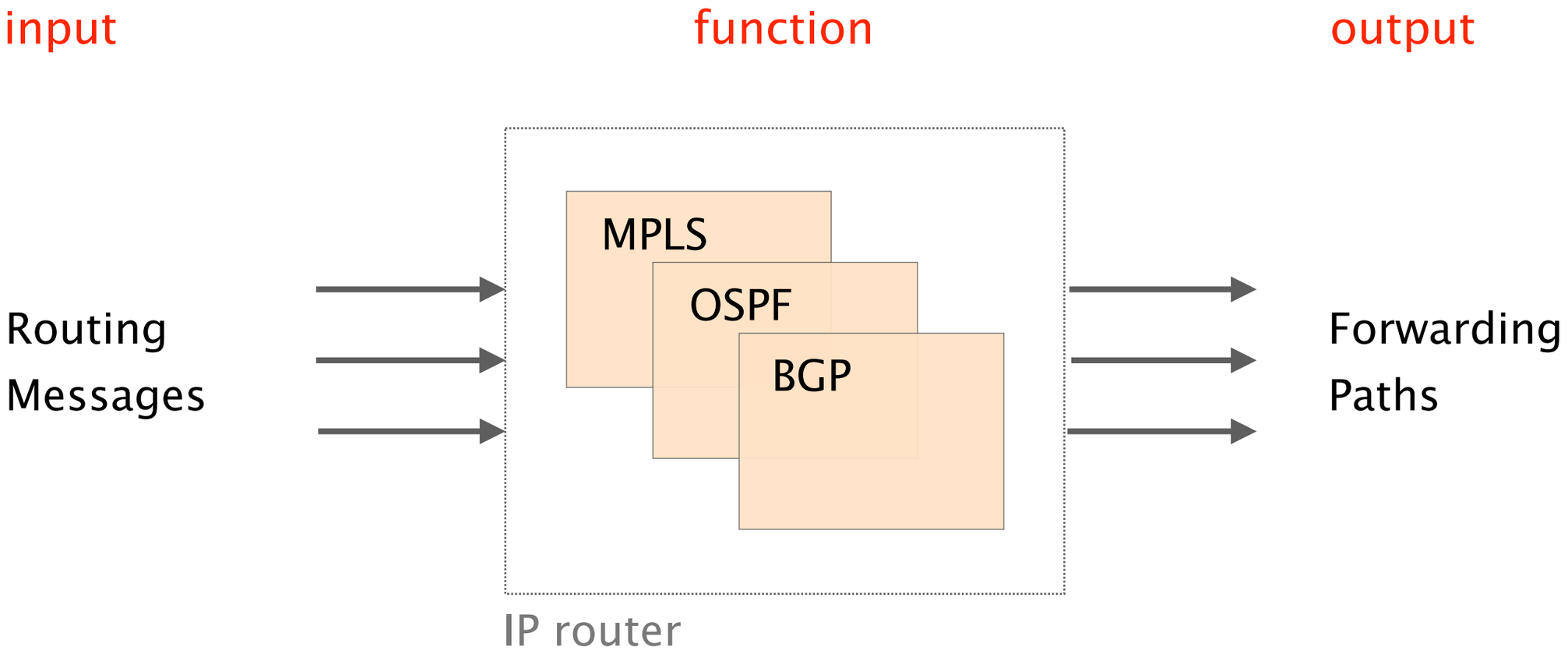
Expressivity

any path, anywhere

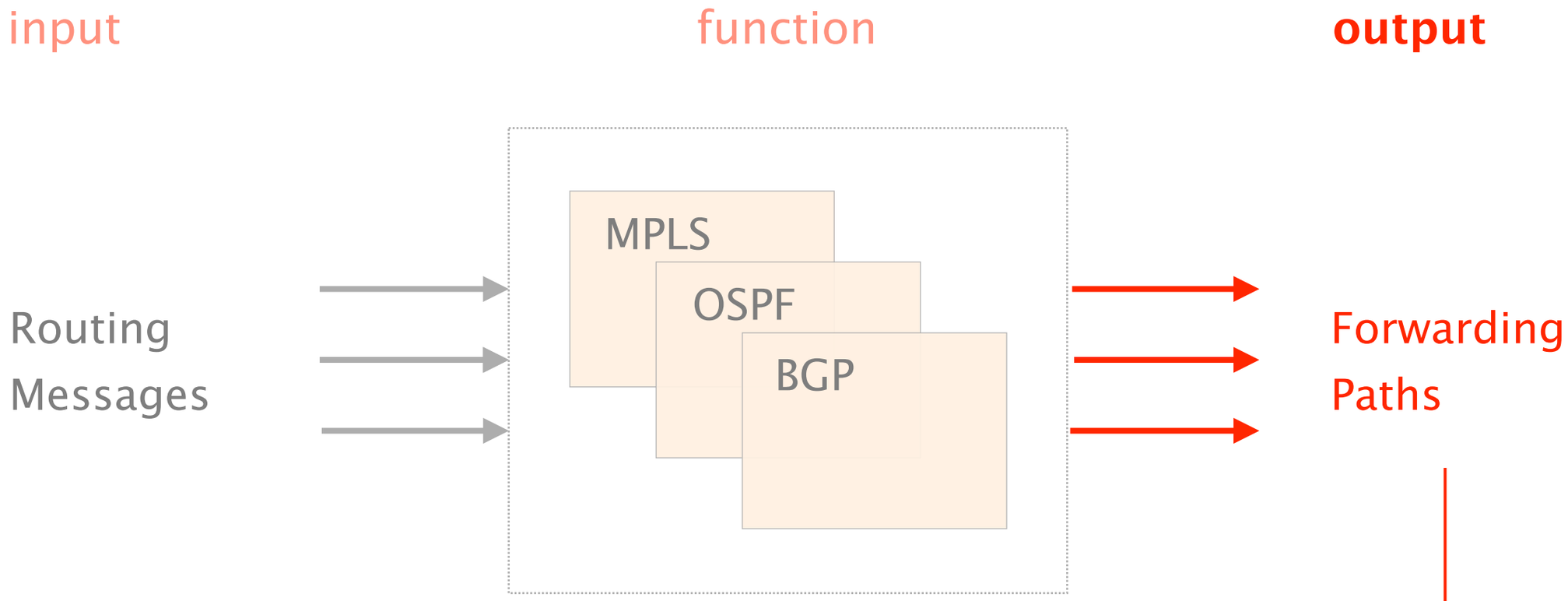
Scalability

1 lie is better than 2

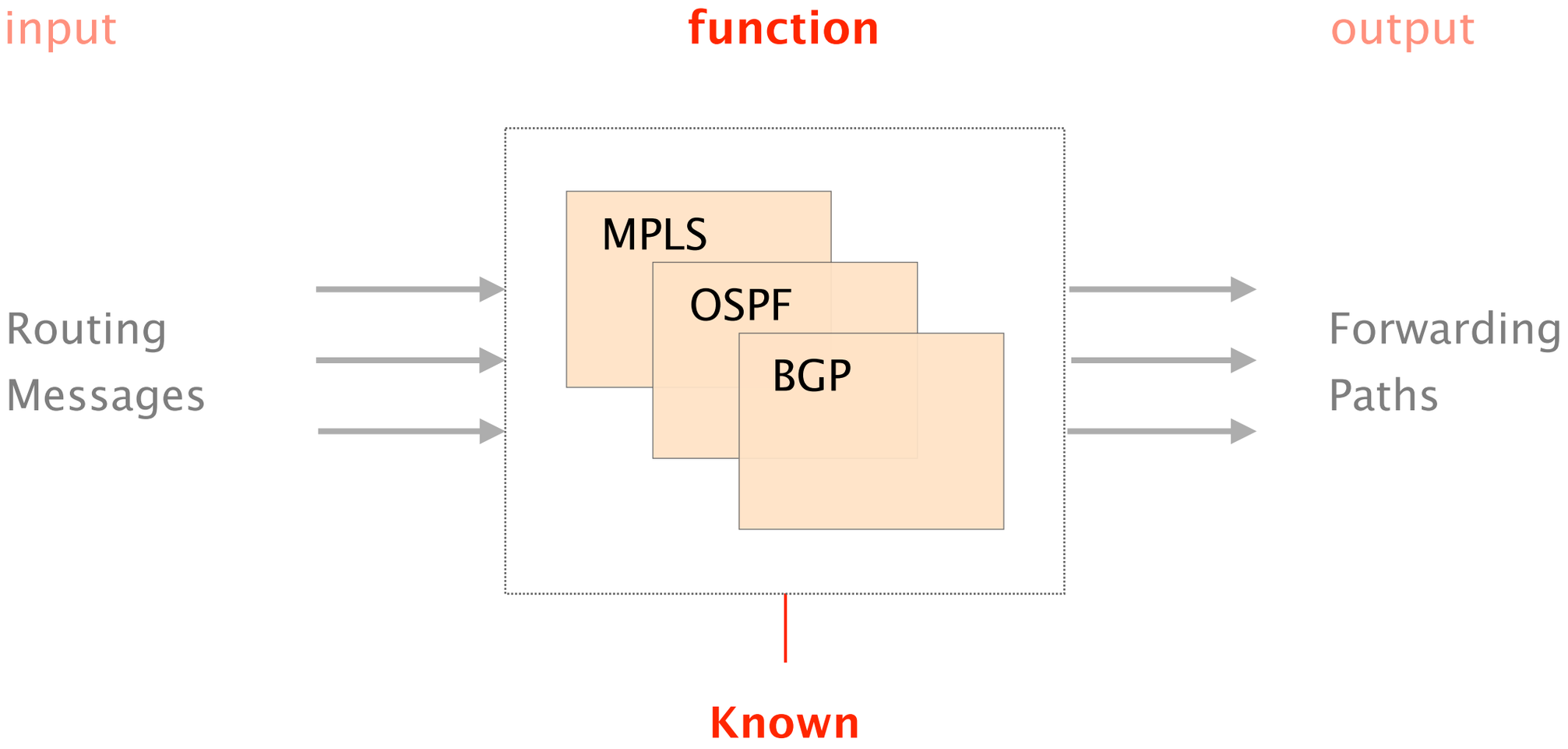
A router implements a function
from routing messages to forwarding paths



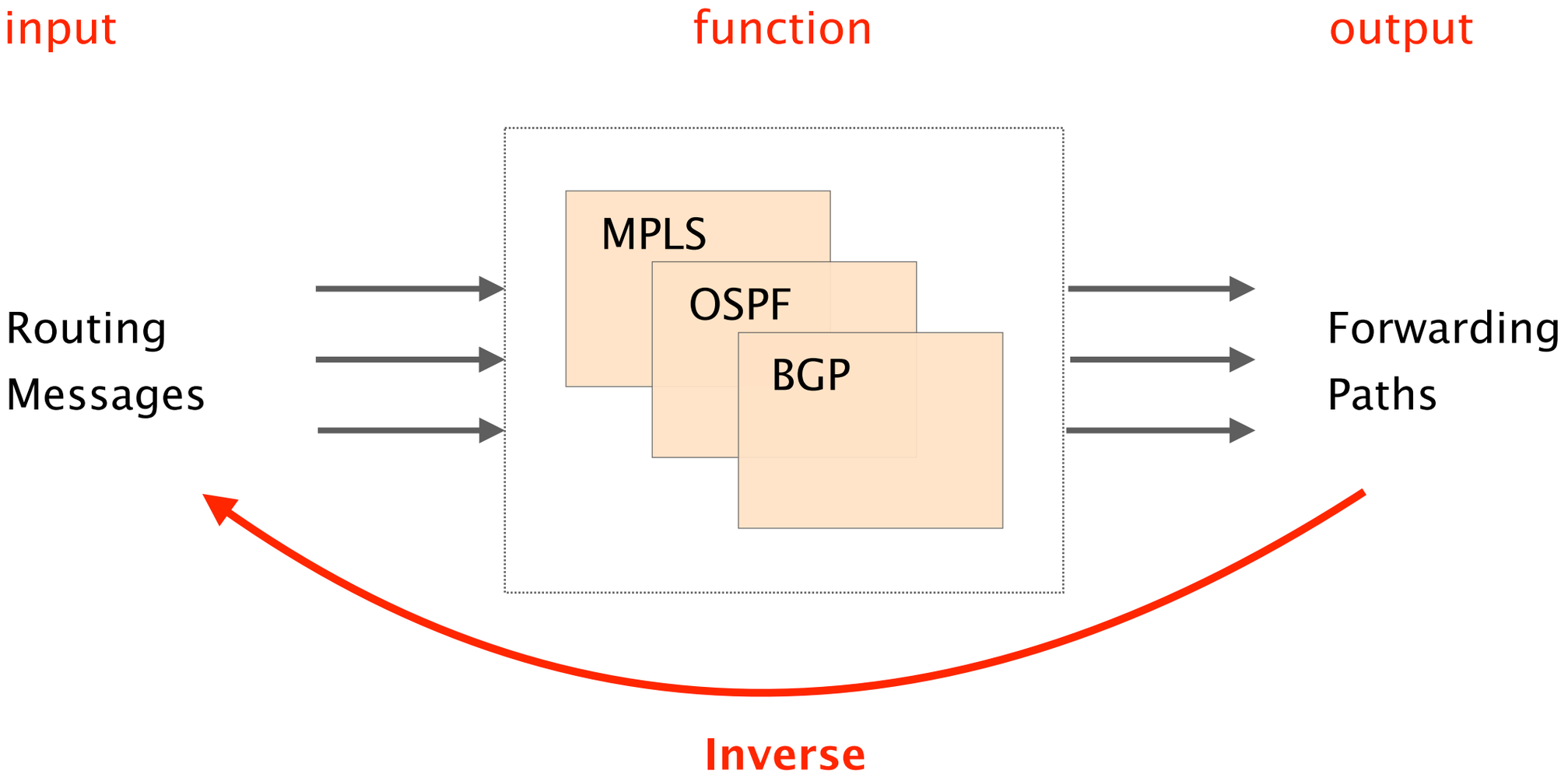
The forwarding paths are known,
provided by the operators or by the controller



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



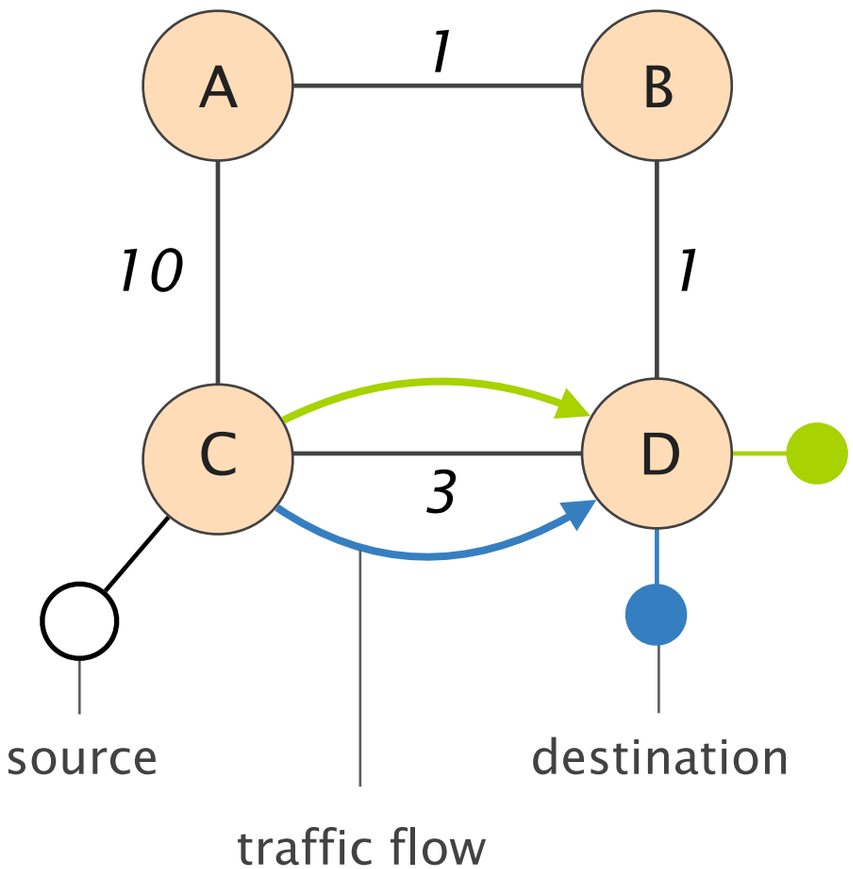
The type of input to be computed depends on the routing protocol

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

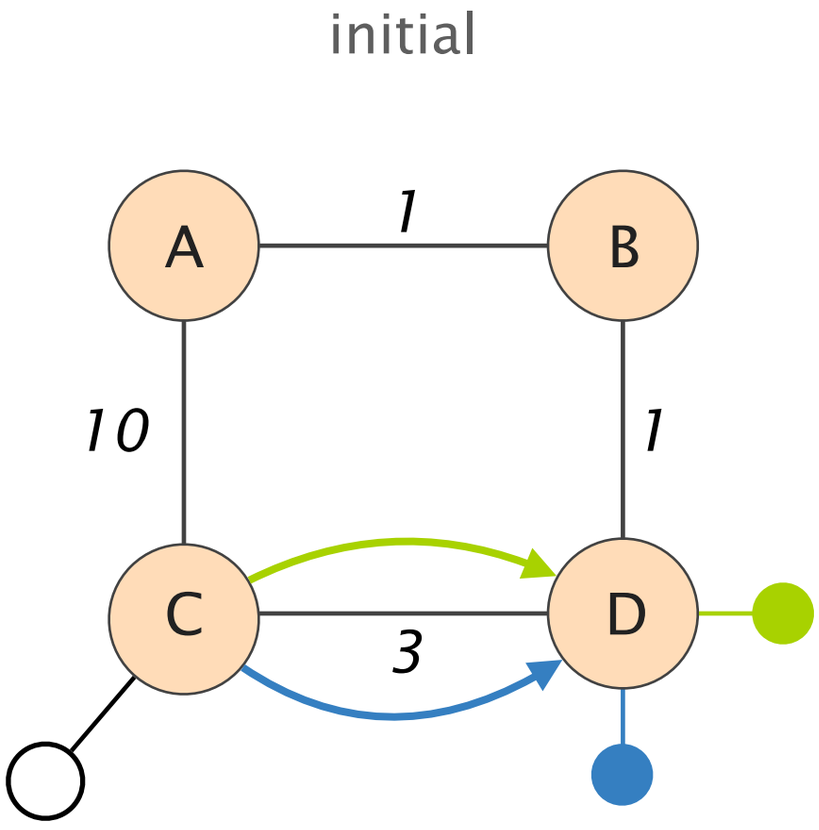
We focus on routers running link-state protocols that take the network graph as input and run Dijkstra

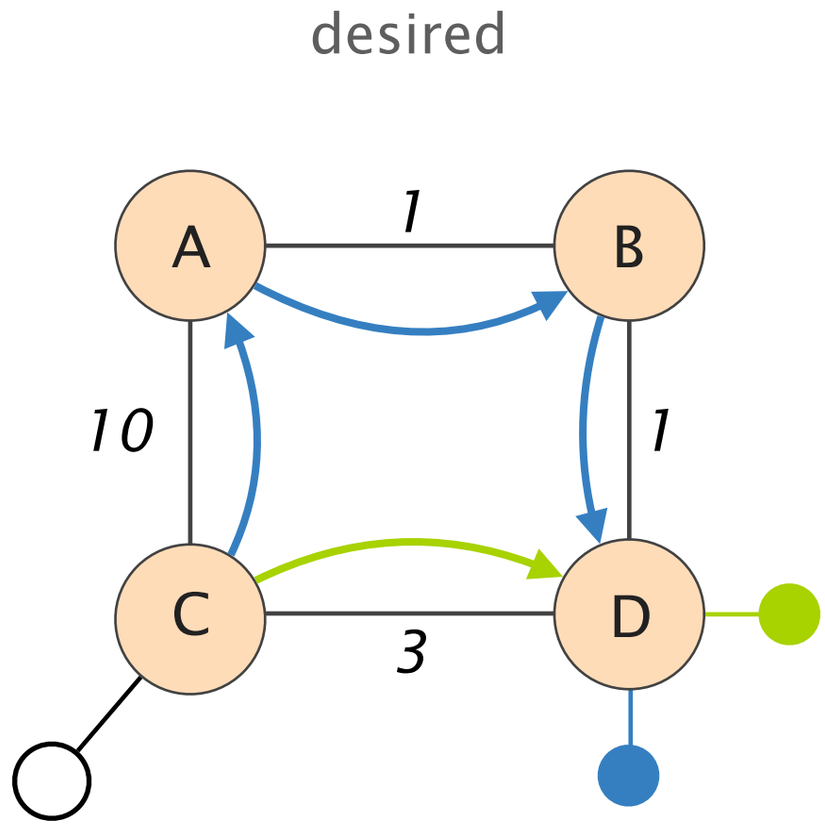
Protocol	Family	Algorithm/ Function	Router Input
IGP	Link-State	Dijkstra	Network graph
BGP	Path-Vector	Decision process	Routing paths

Consider this network where a source sends traffic to 2 destinations



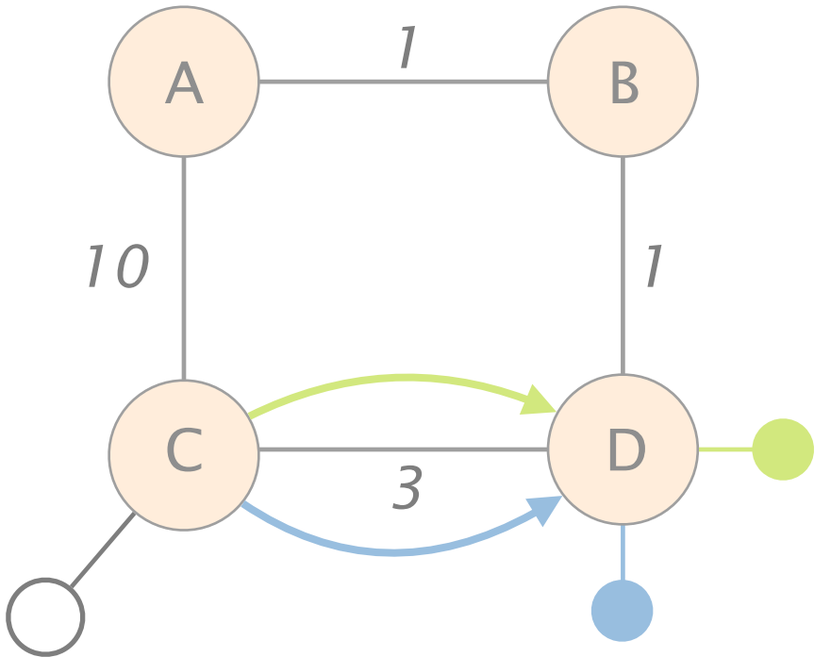
As congestion appears, the operator wants to shift away one flow from (C,D)



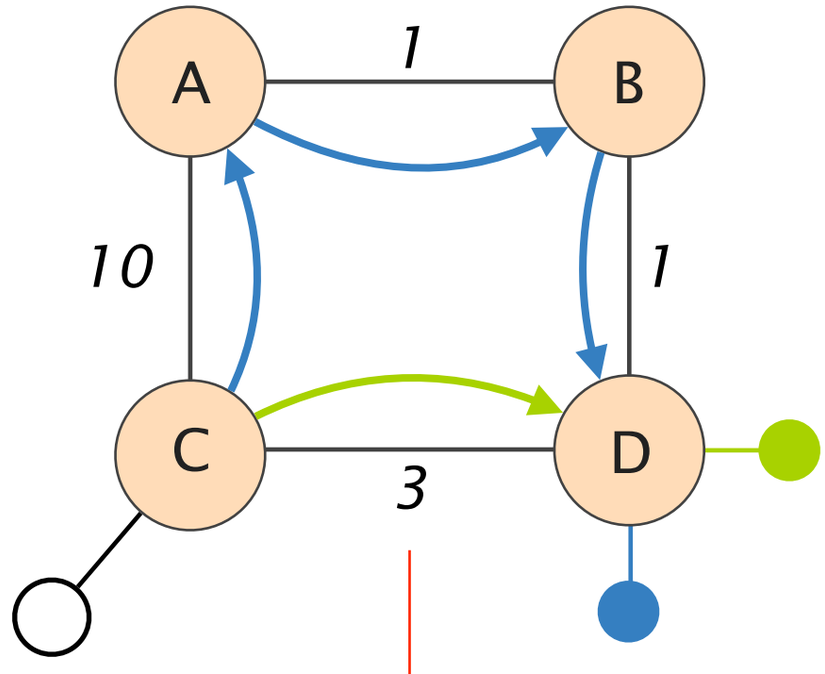


Moving only one flow is impossible though as both destinations are connected to D

initial

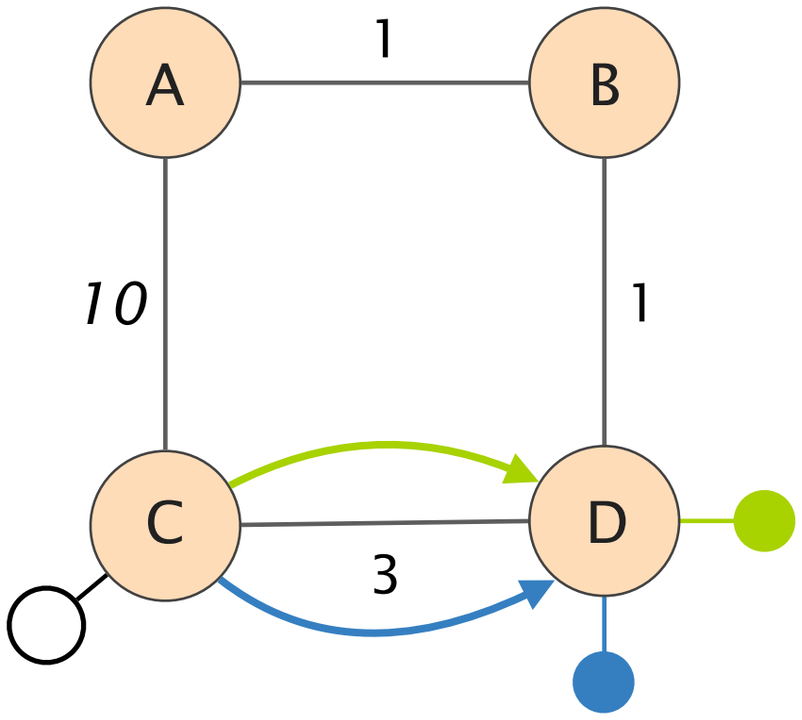


desired

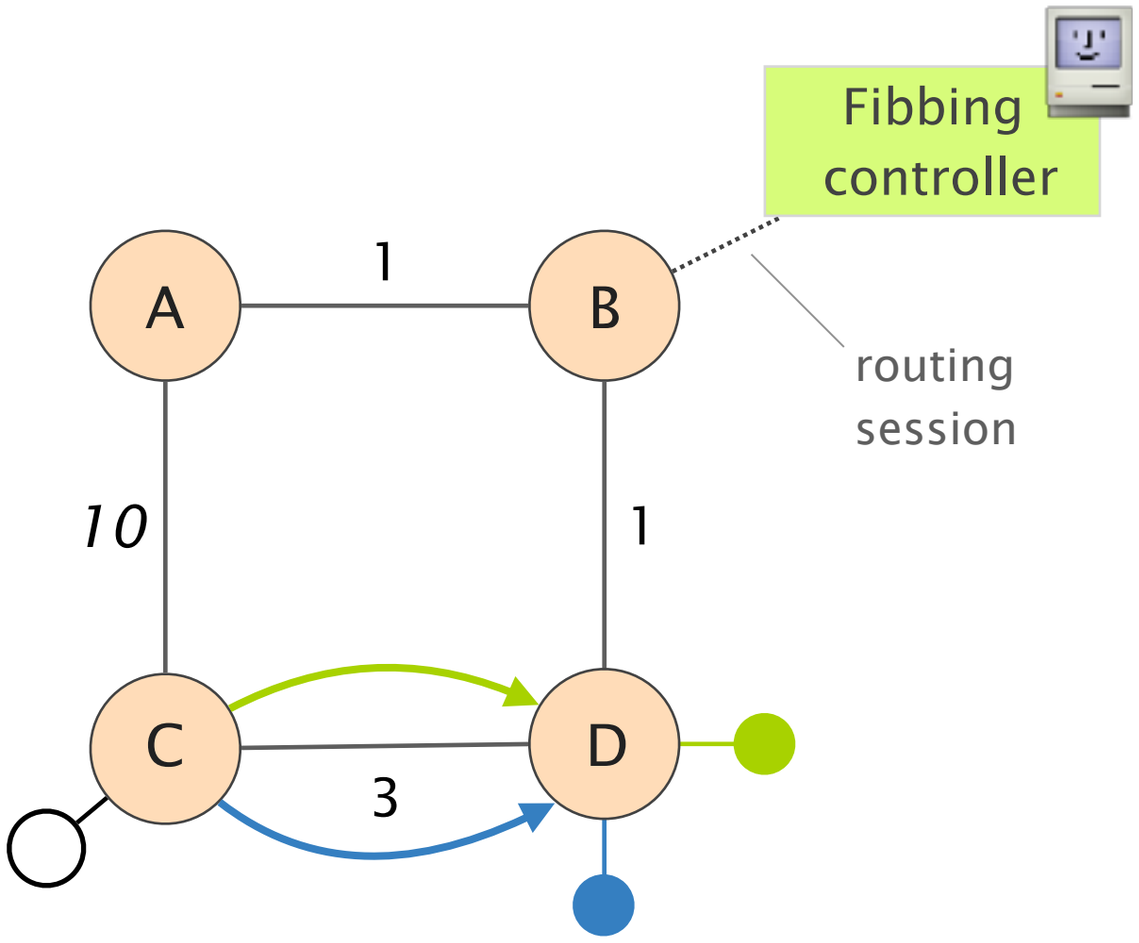


impossible to achieve by reweighing the links

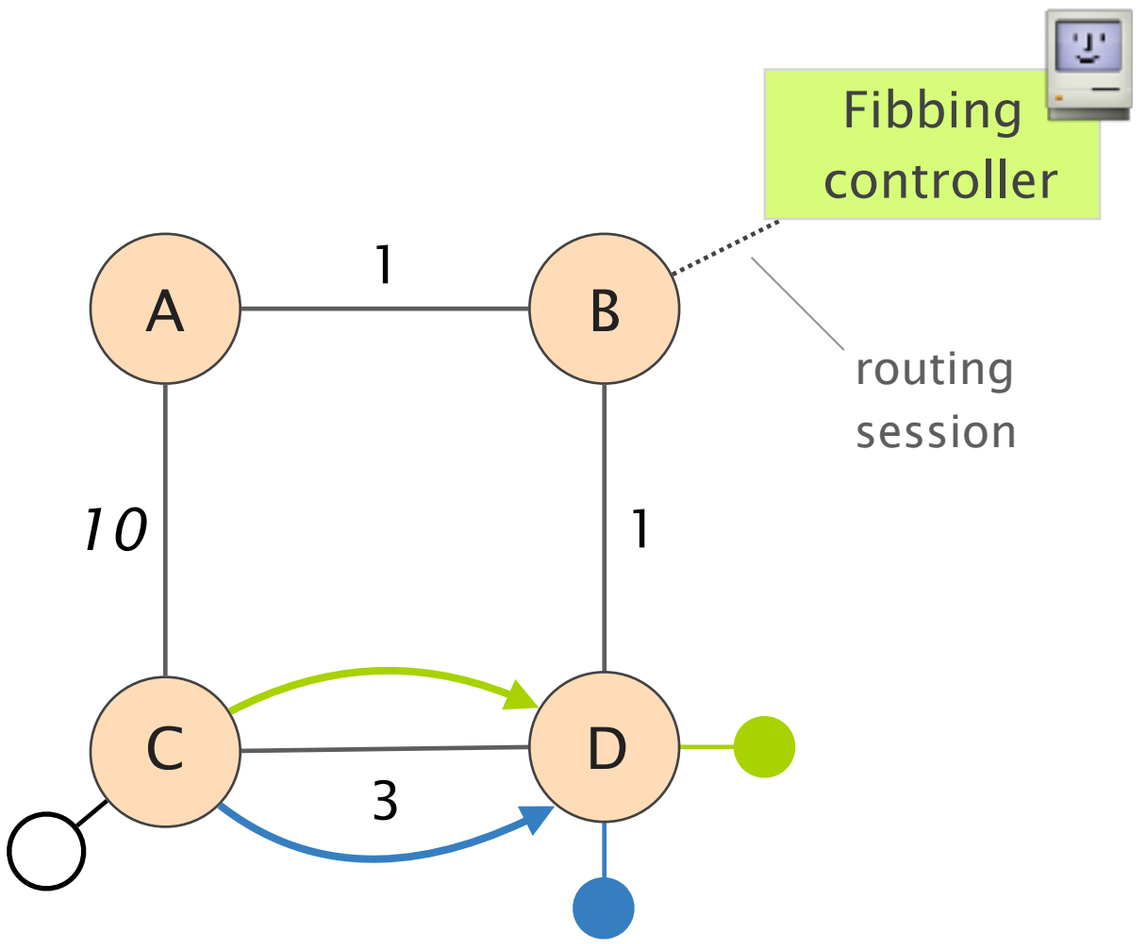
Let’s lie to the router



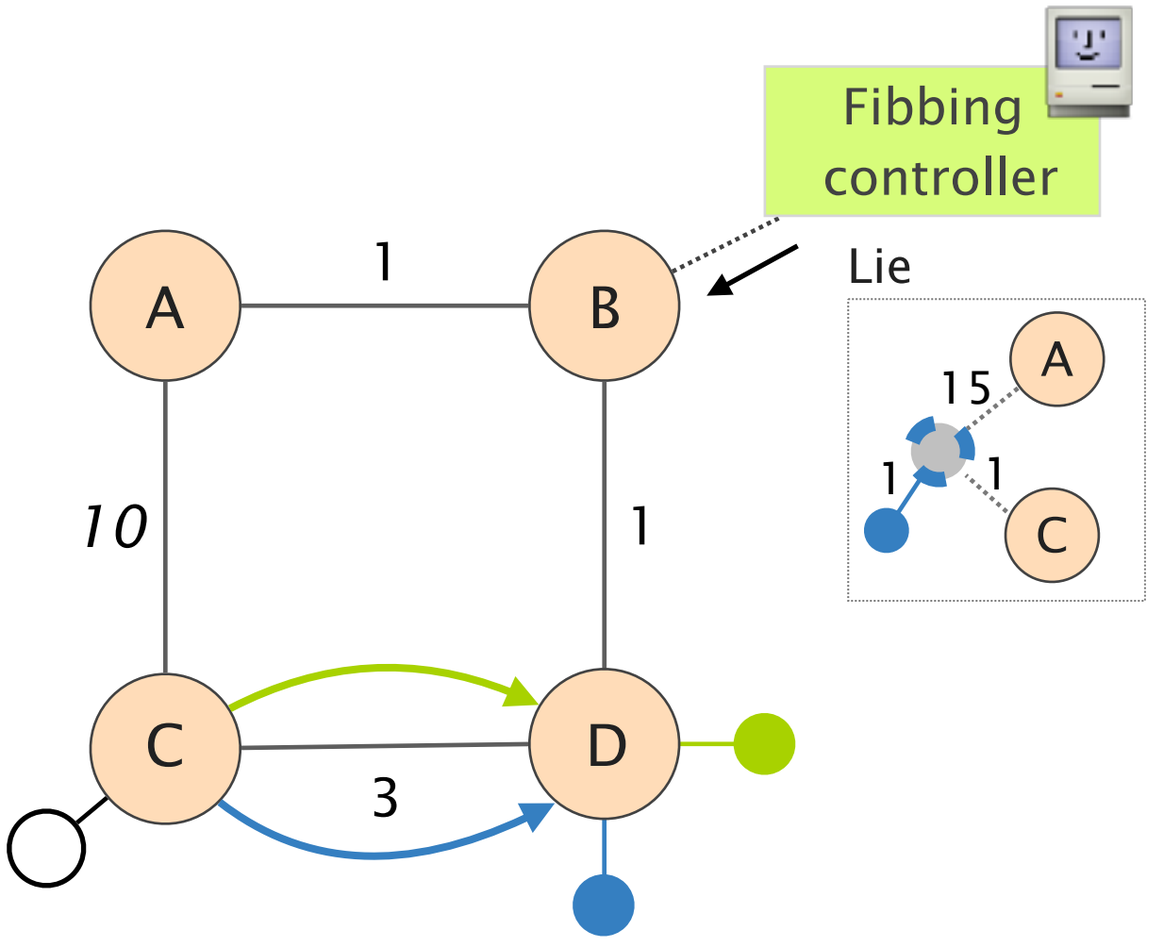
Let's lie to the router



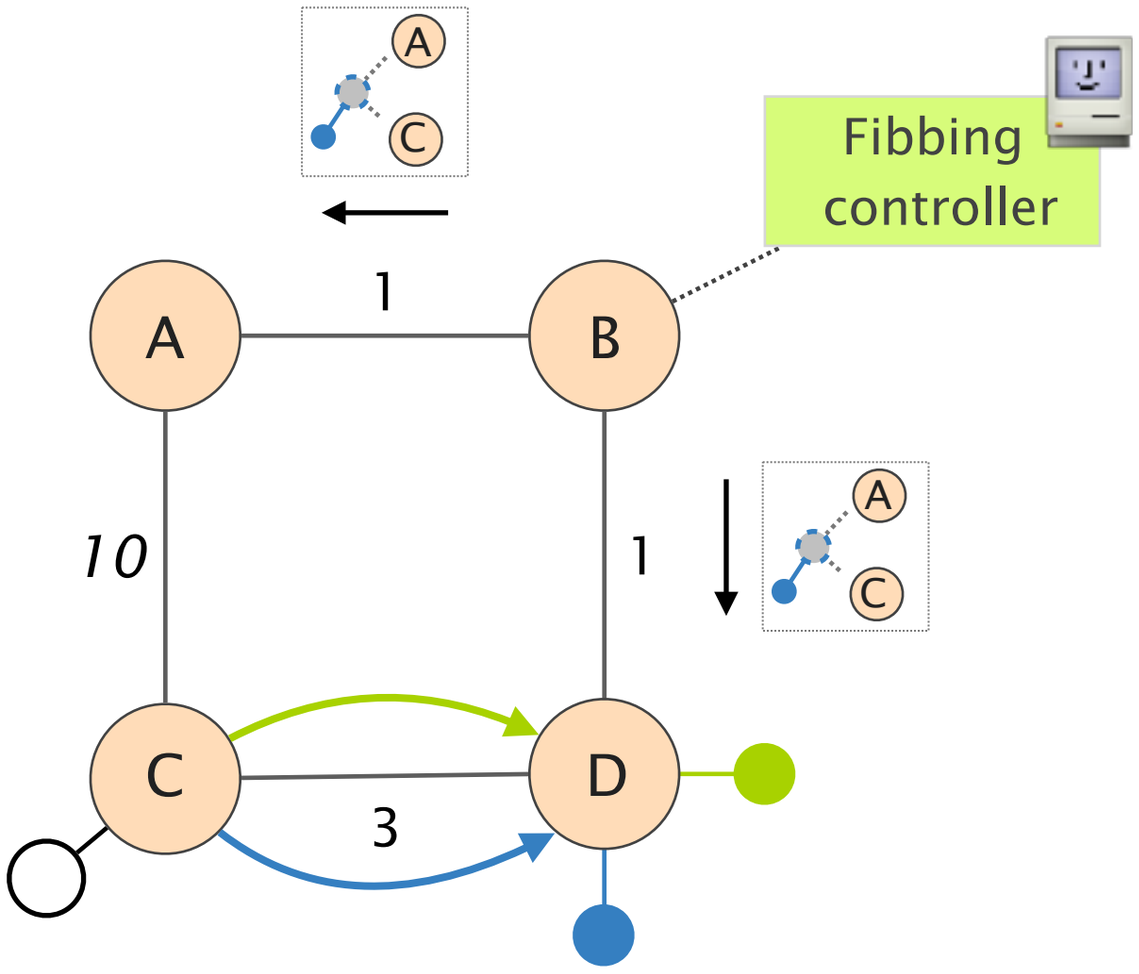
Let's **lie to the router,** by injecting fake nodes, links and destinations



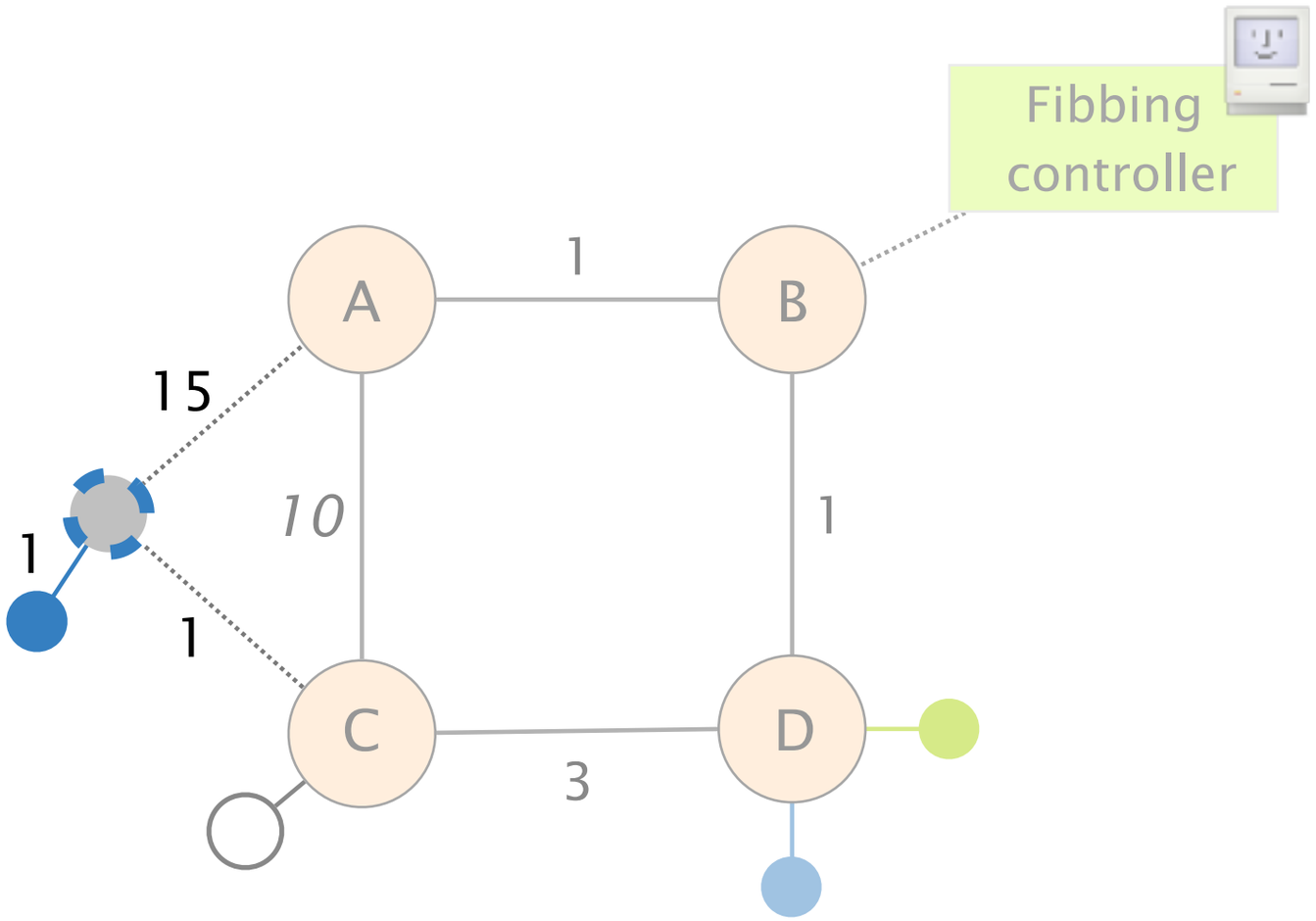
Let's **lie to the router,** by injecting fake nodes, links and destinations



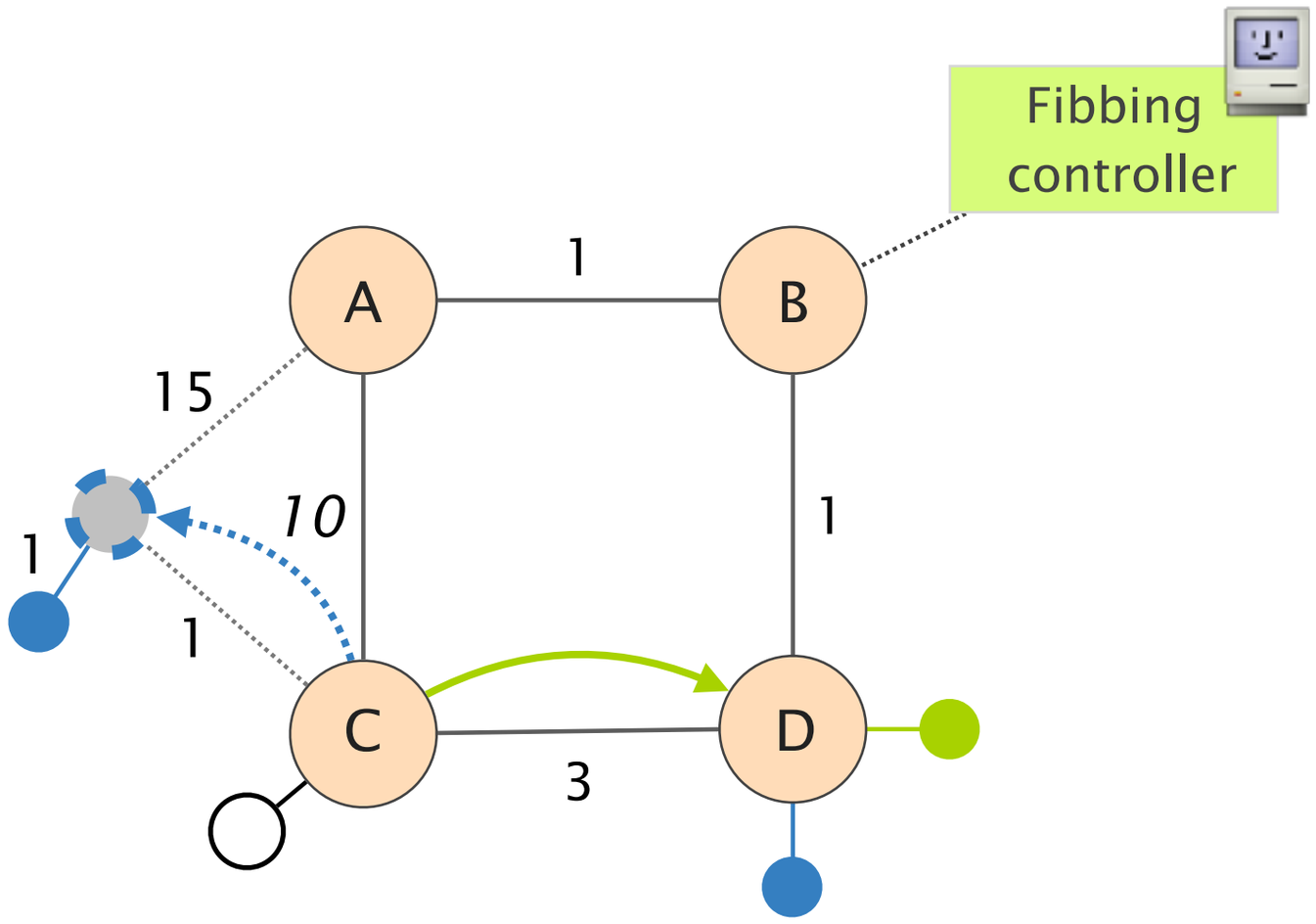
Lies are propagated network-wide by the protocol



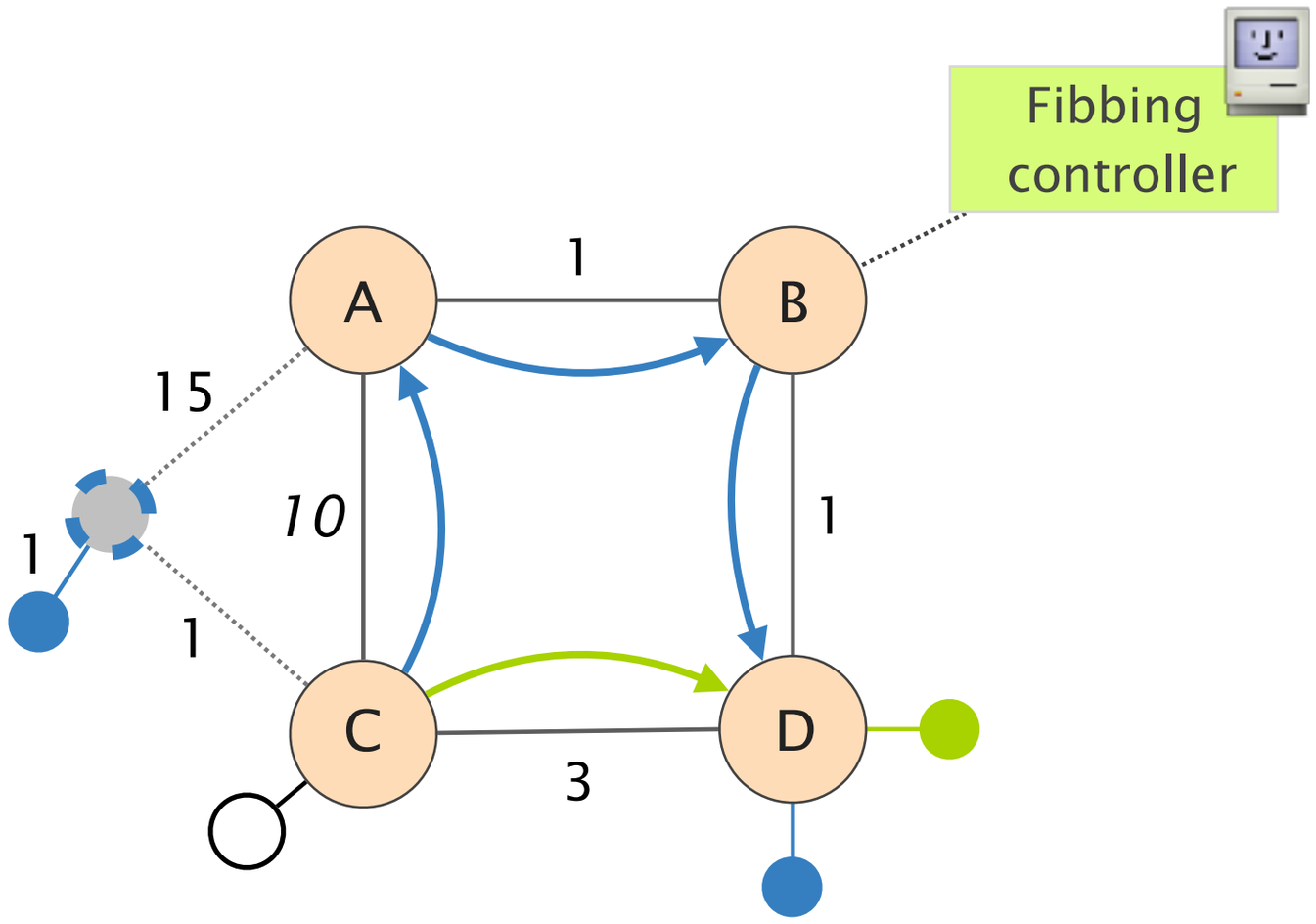
After the injection, this is the topology seen by all routers, on which they compute Dijkstra



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

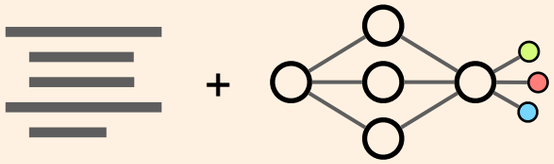


As the virtual node does not really exist,
actual traffic is *physically* sent to A



Fibbing workflow

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



path reqs.

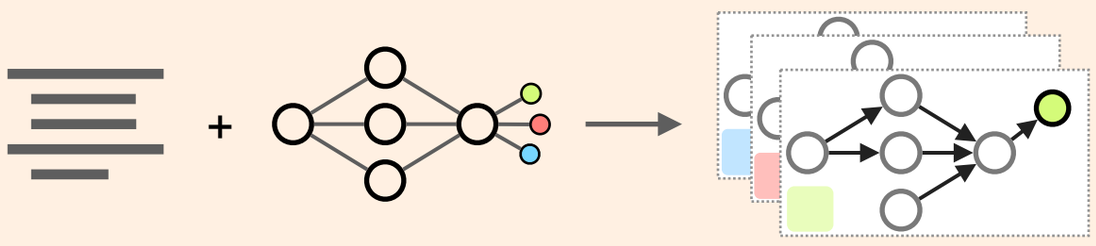
network graph

Operators requirements are expressed in a high-level language

Syntax of Fibbing's path requirements language

pol	::=	$(s_1; \ldots; s_n)$	Fibbing Policy
s	::=	$p \mid b$	Requirement
r	::=	p_1 and $p_2 \mid p_1$ or $p_2 \mid p$	Path Req.
p	::=	Path(n^+)	Path Expr.
n	::=	$id \mid * \mid n_1$ and $n_2 \mid n_1$ or n_2	Node Expr.
n	::=	$id \mid * \mid n_1$ and $n_2 \mid n_1$ or n_2	Node Expr.
b	::=	r as backupof $((id_1, id_2)^+)$	Backup Req.

Out of these,
the compilation stage produces DAGs

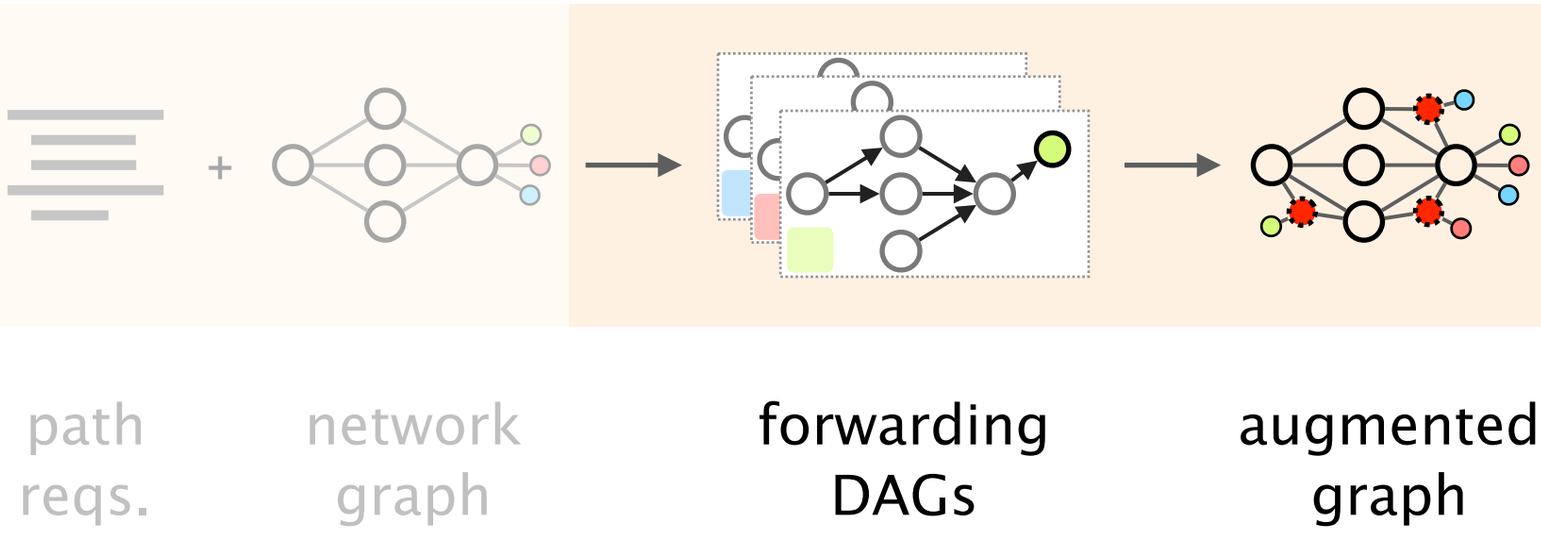


path reqs.

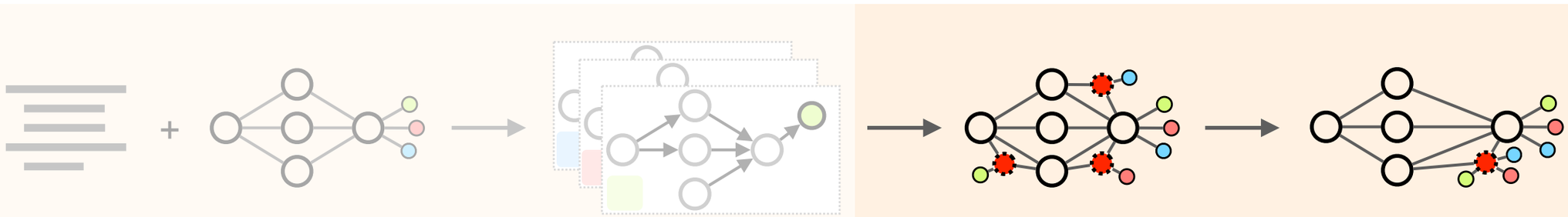
network graph

forwarding DAGs

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

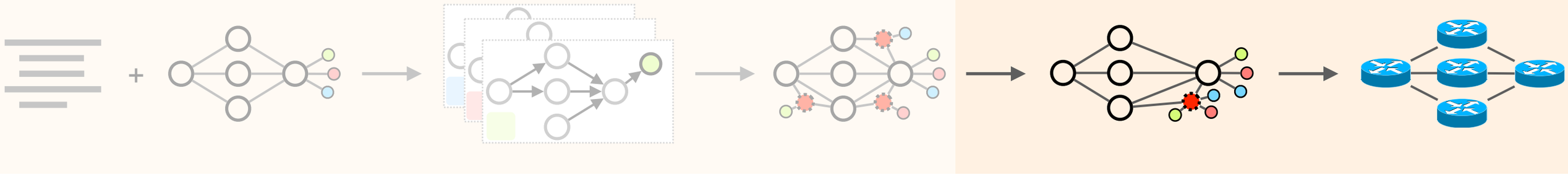


The optimization stage reduces the amount of lies necessary



The injection stage injects
the lies in the production network

Injection



path reqs.

network graph

forwarding DAGs

augmented graph

reduced graph

running 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

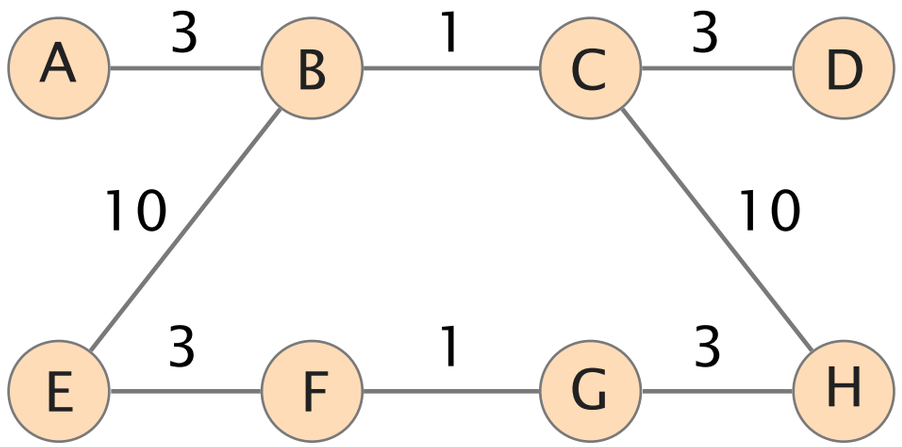
- any path is loop-free

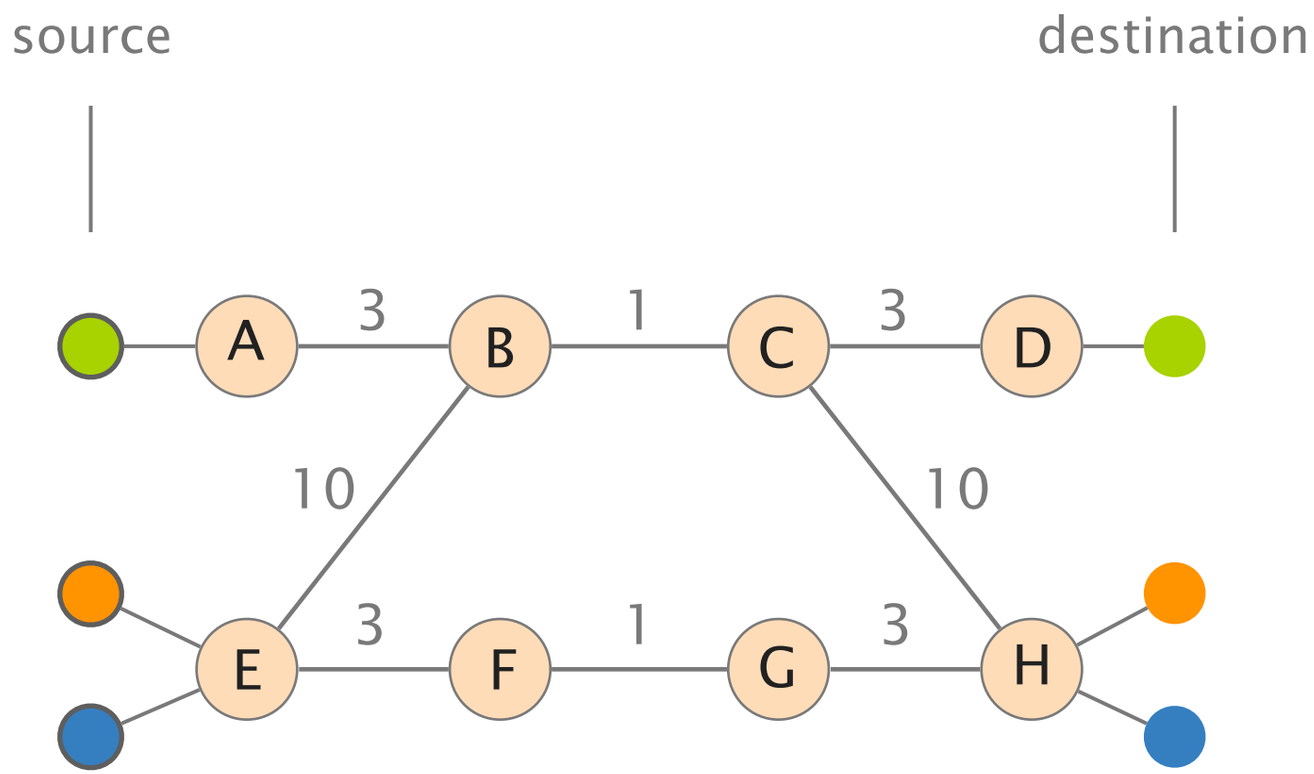
  (*e.g.*, [s1, a, b, a, d] is not possible)

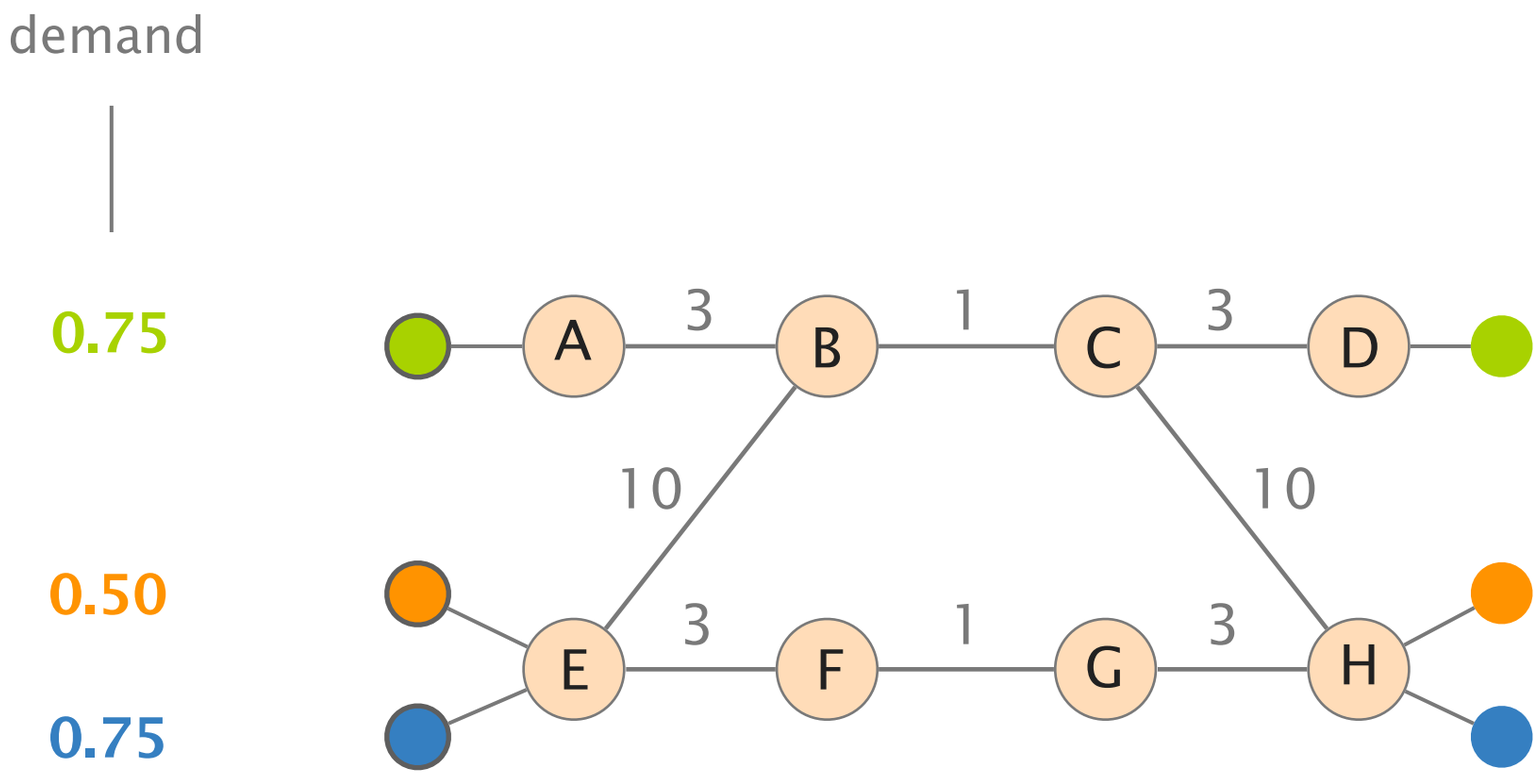
- paths are consistent

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

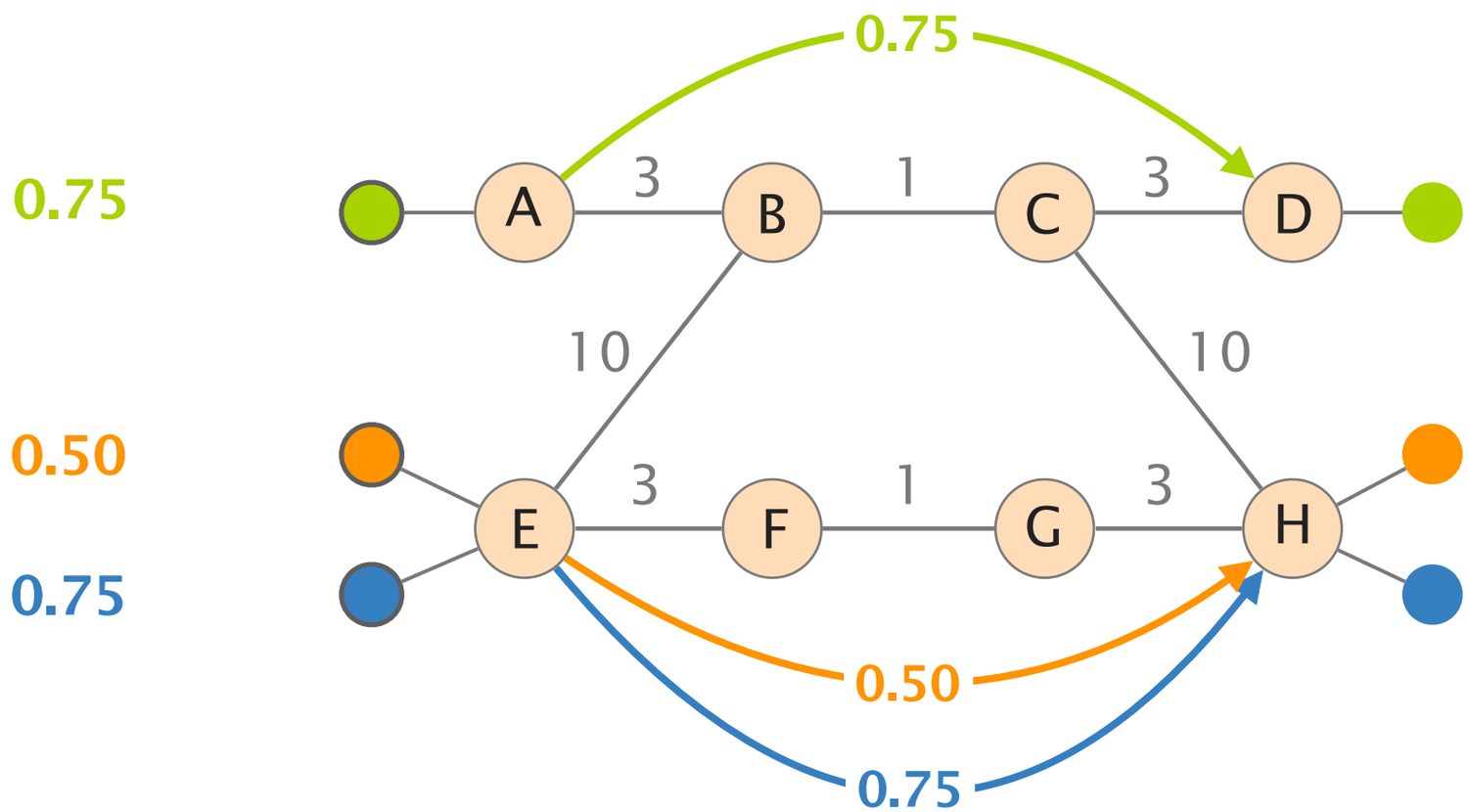
Fibbing can load-balance traffic on multiple paths




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

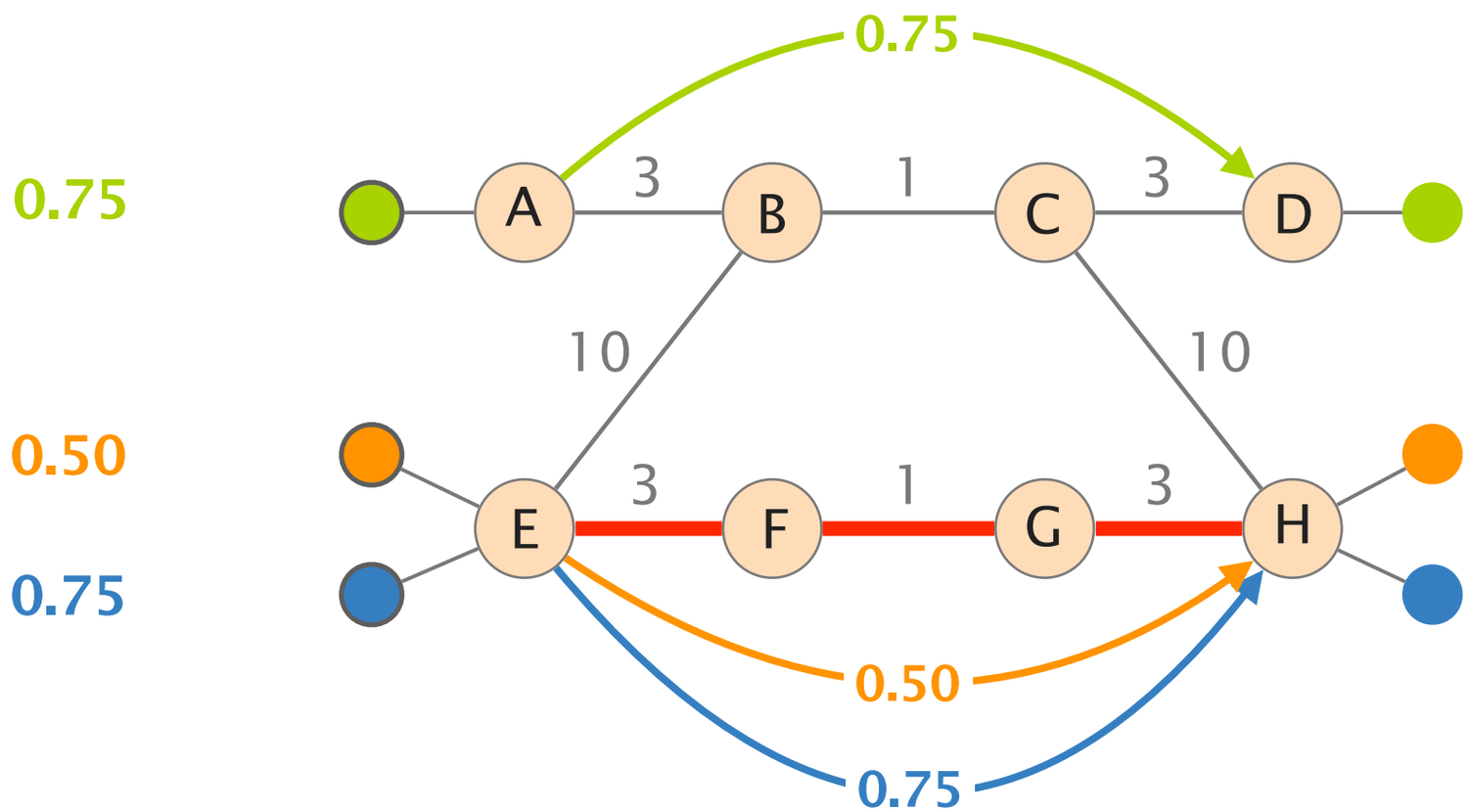


Links have a capacity of 1

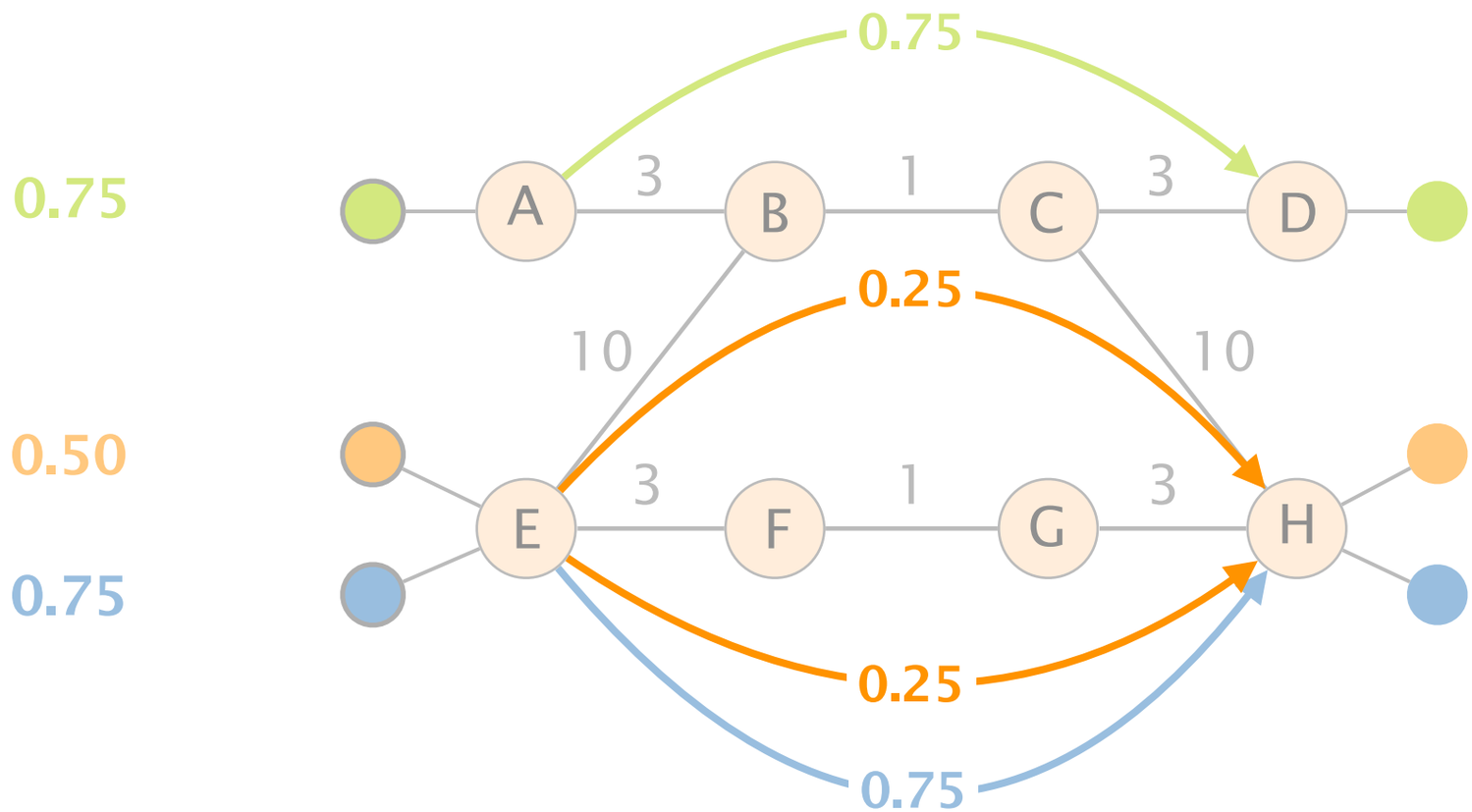


Links have a capacity of 1

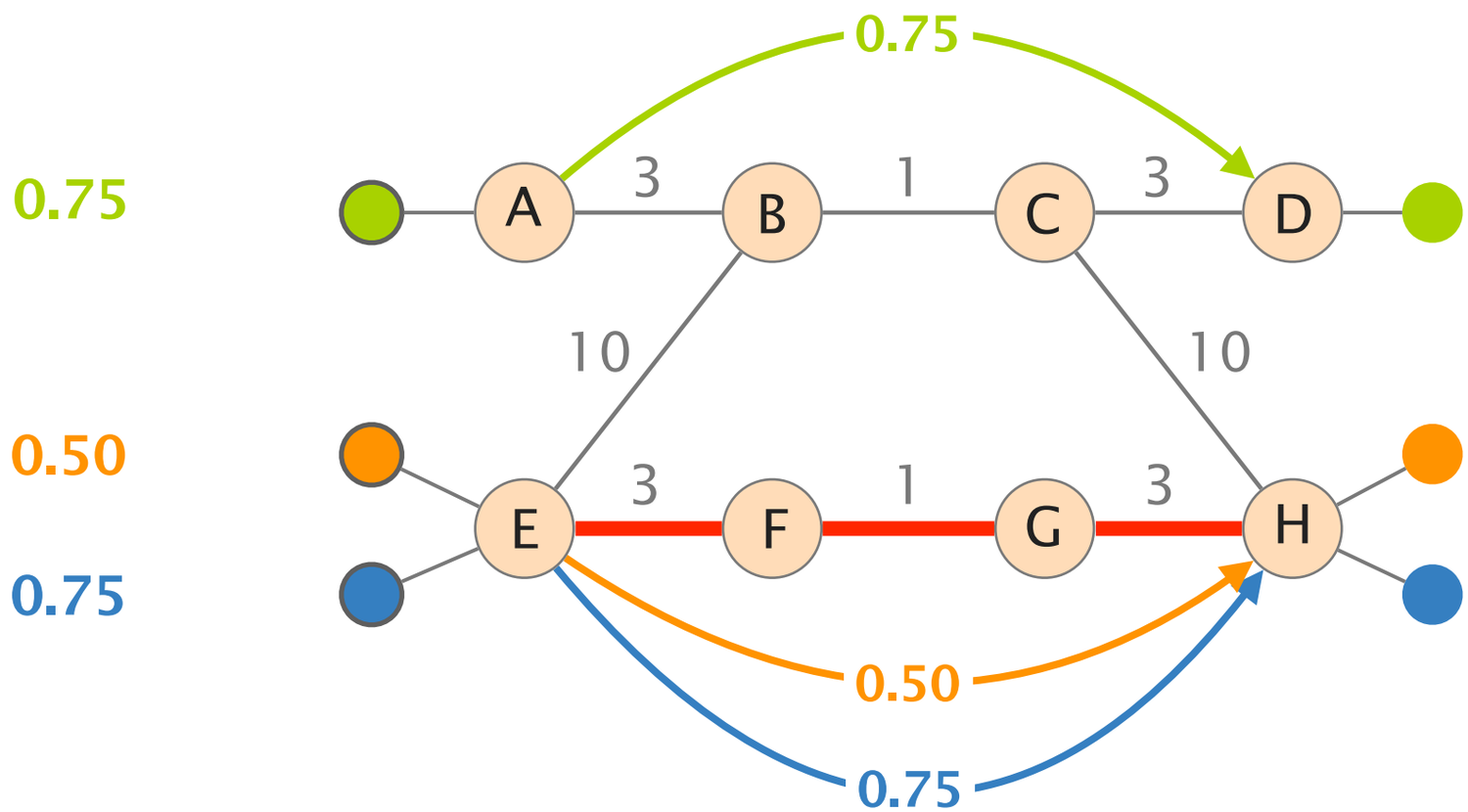
With such demands and forwarding,
the lower path is congested (1.25)



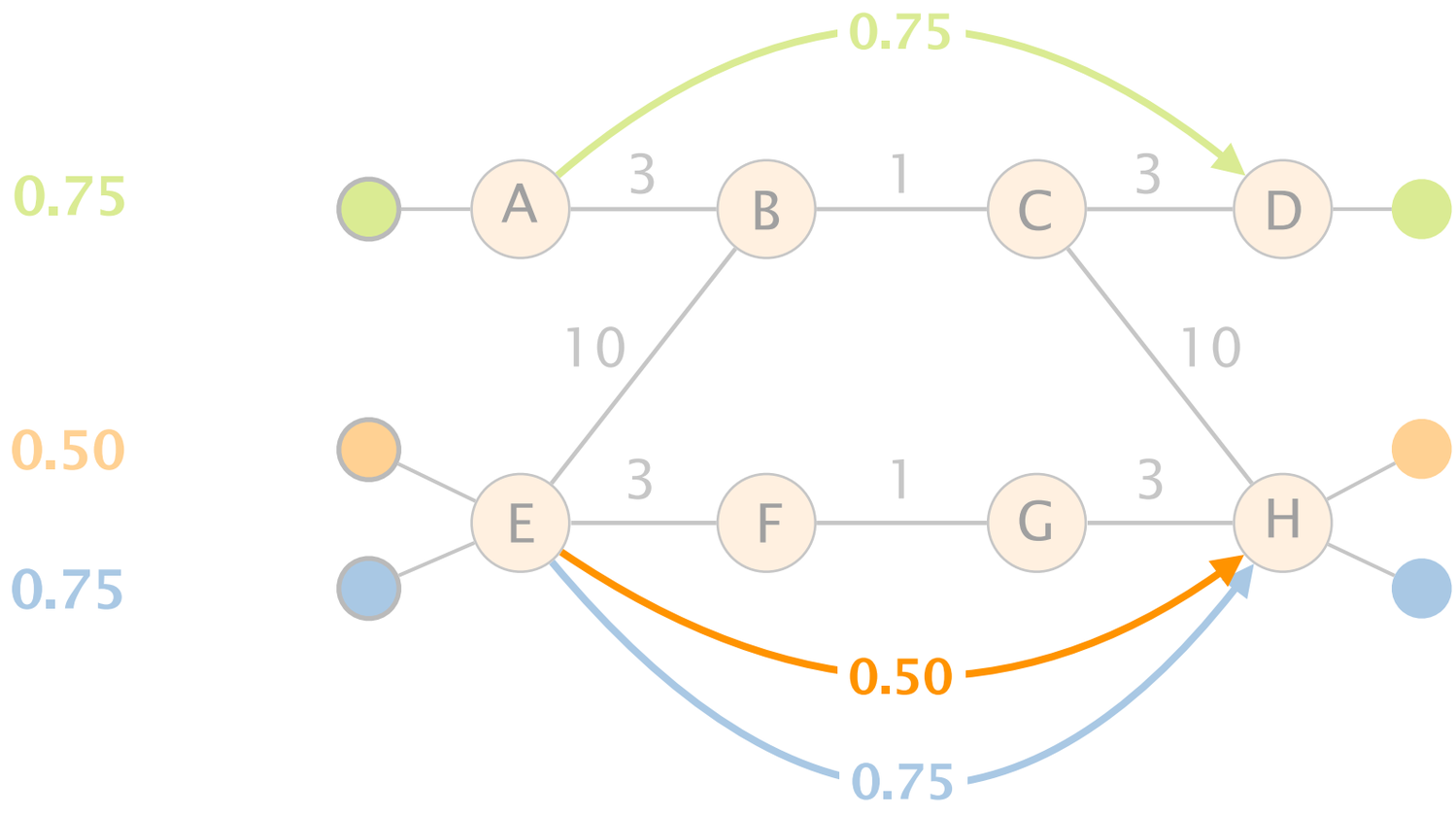
Congestion can be alleviated by splitting the orange flow into two equal parts (.25)



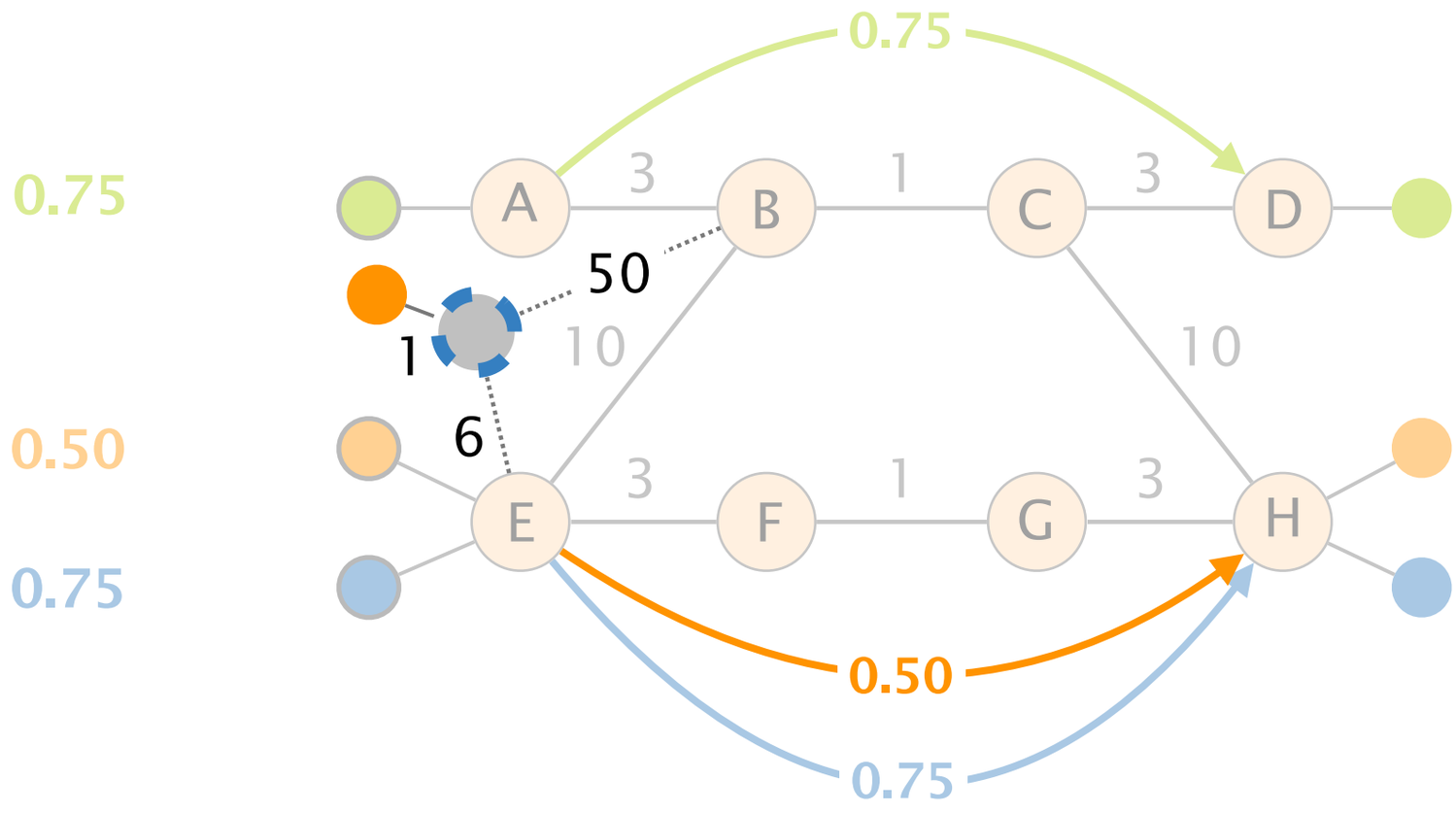
This is impossible to achieve
using a link-state protocol



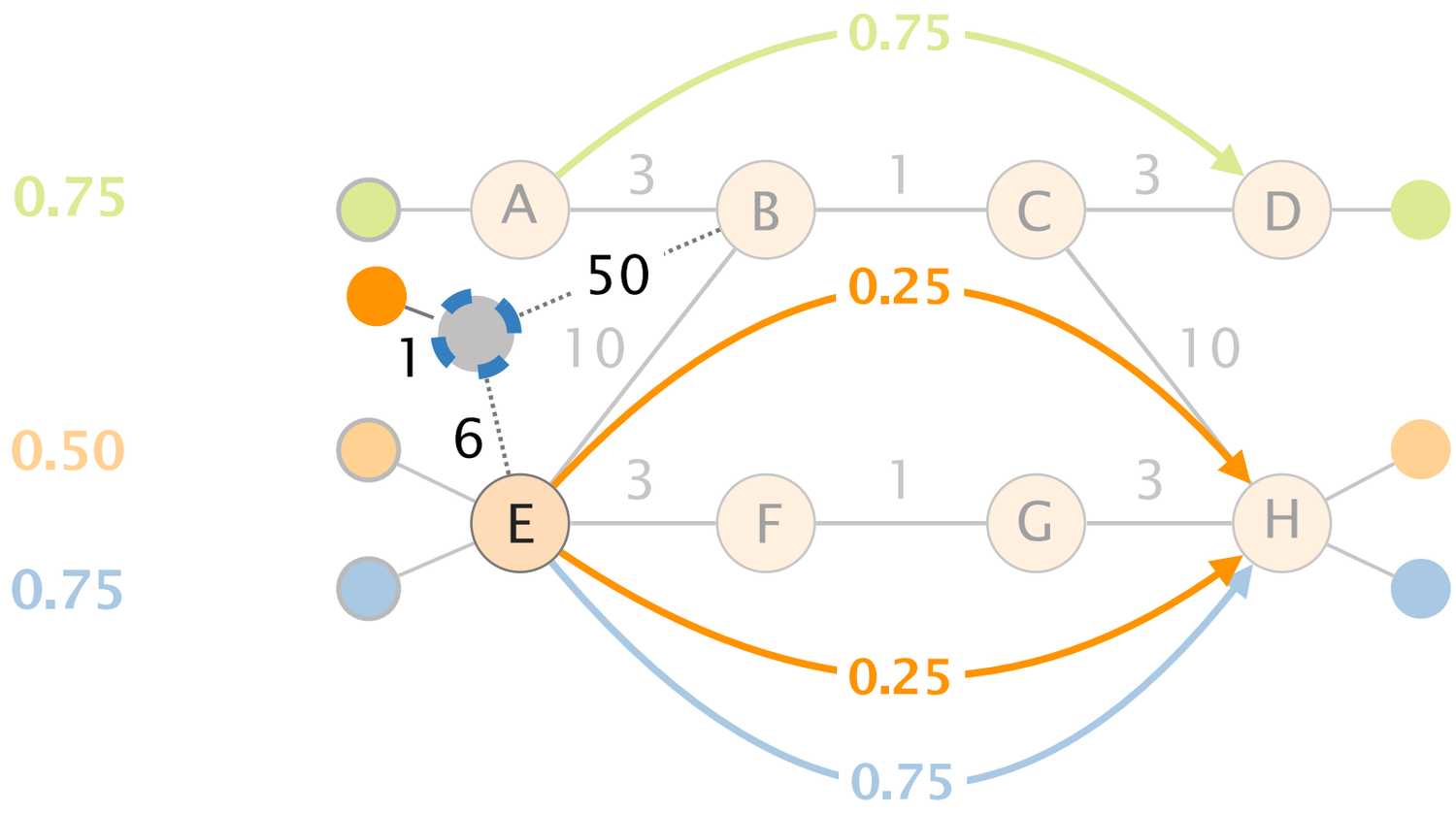
This is easily achievable with Fibbing



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

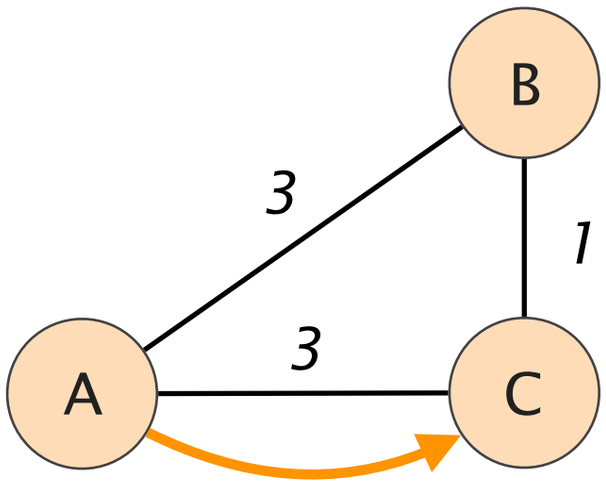
to compute lies

space

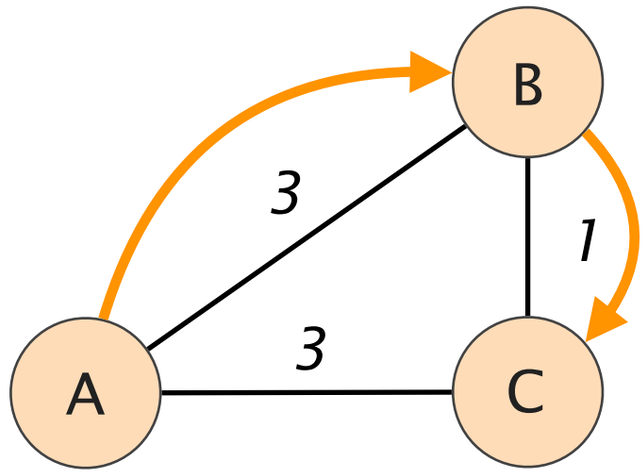
# of lies

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

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

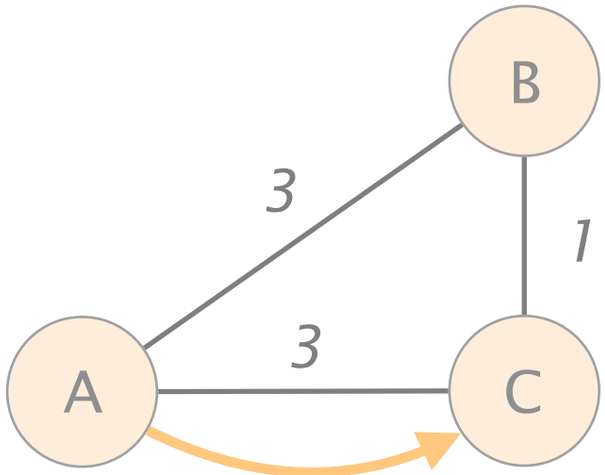


initial

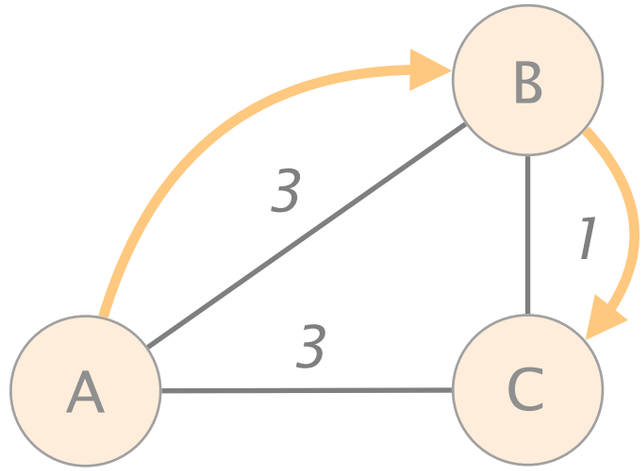


desired

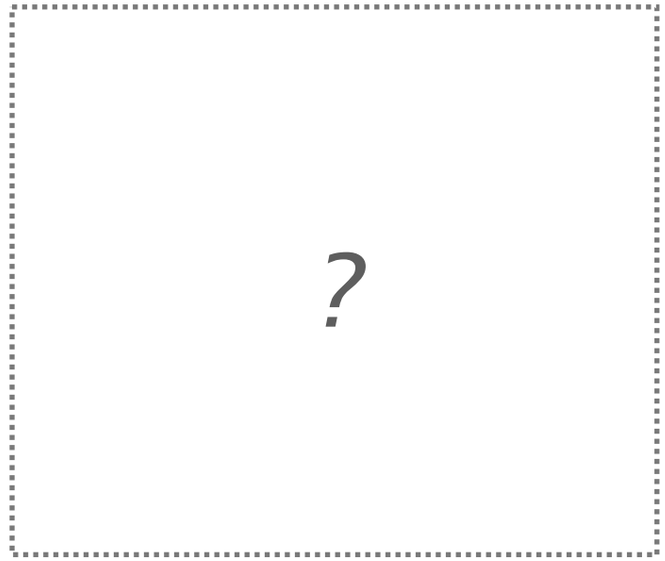
Computing virtual topologies is easy:
polynomial in the number of requirements



initial



desired



virtual

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*

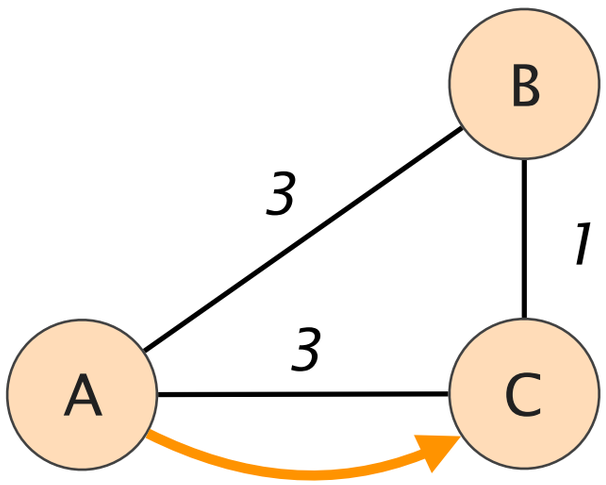
- Create one virtual node v advertising d with a weight $x < w$

always possible

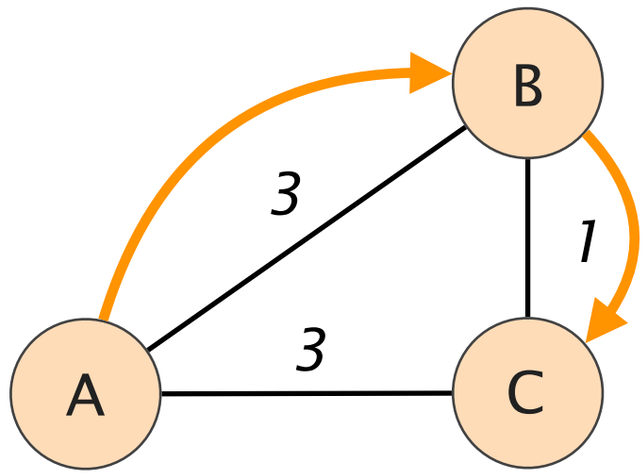
by reweighting the initial graph

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

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



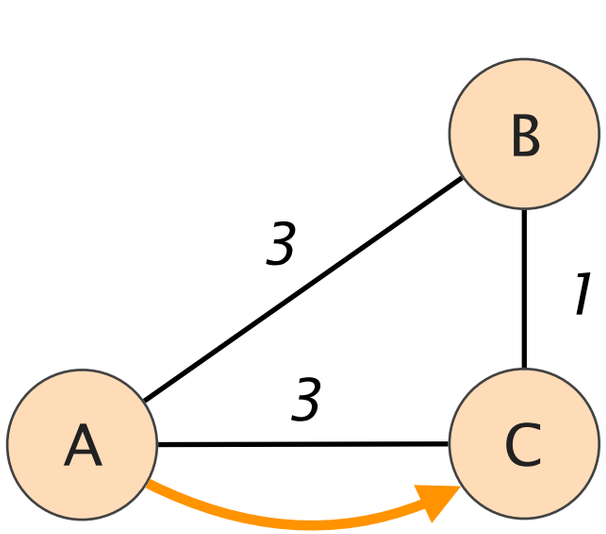
initial



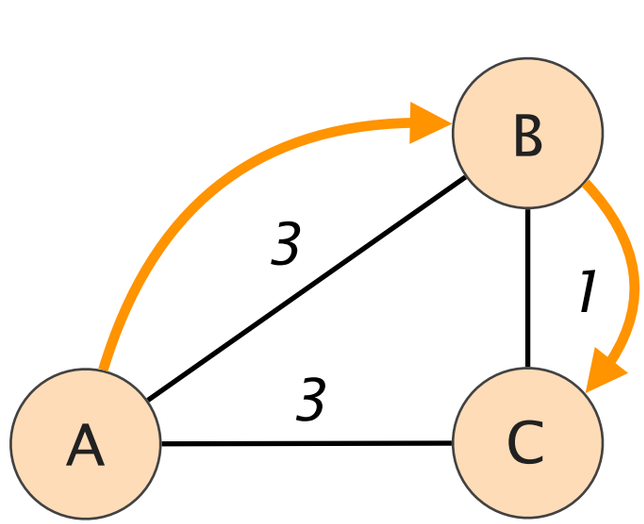
desired

virtual

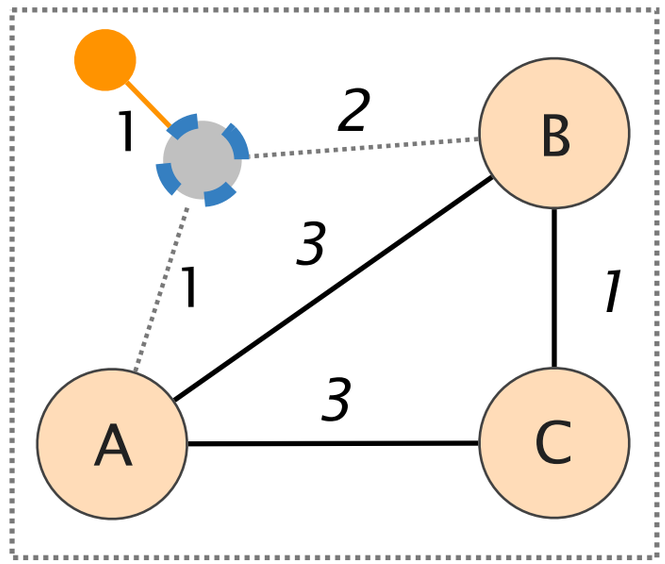
Computing virtual topologies is easy:
polynomial in the number of requirements



initial



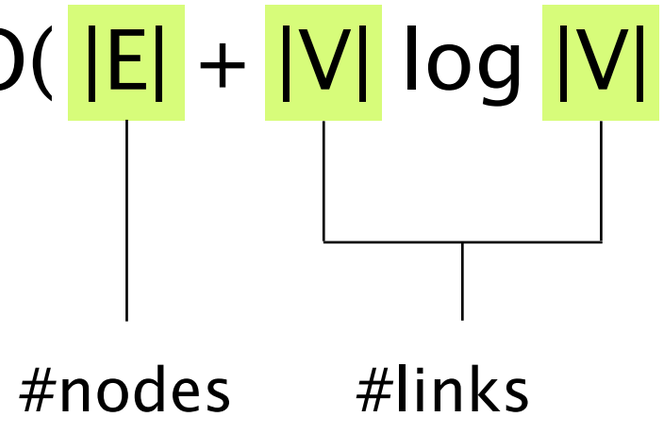
desired



virtual

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

Dijkstra's algorithm complexity



Scalability

time

to compute lies

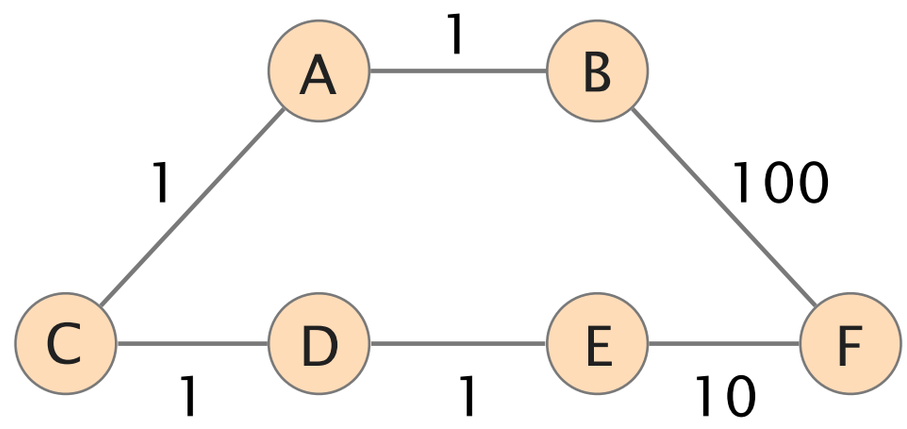
space

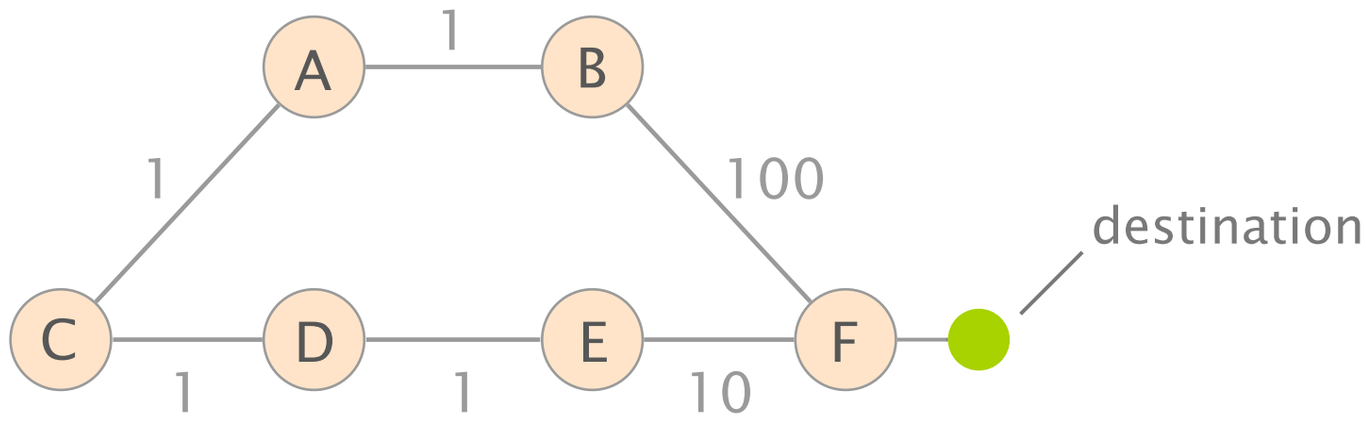
# of lies

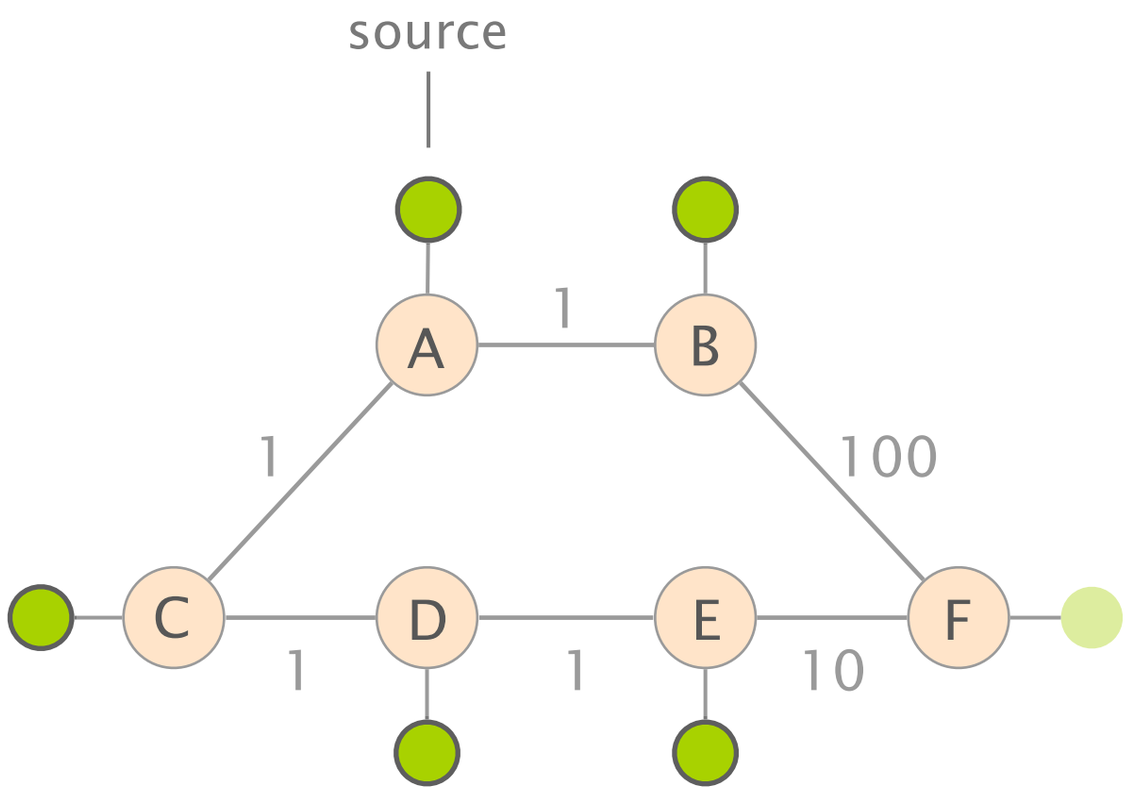
Good news

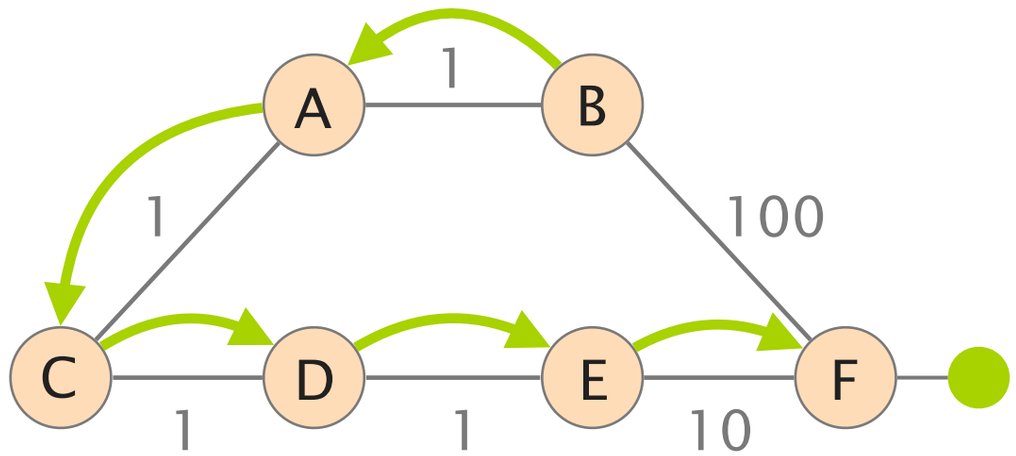
Lots of lies are not required,
some of them are redundant

Let’s us consider
a simple example


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

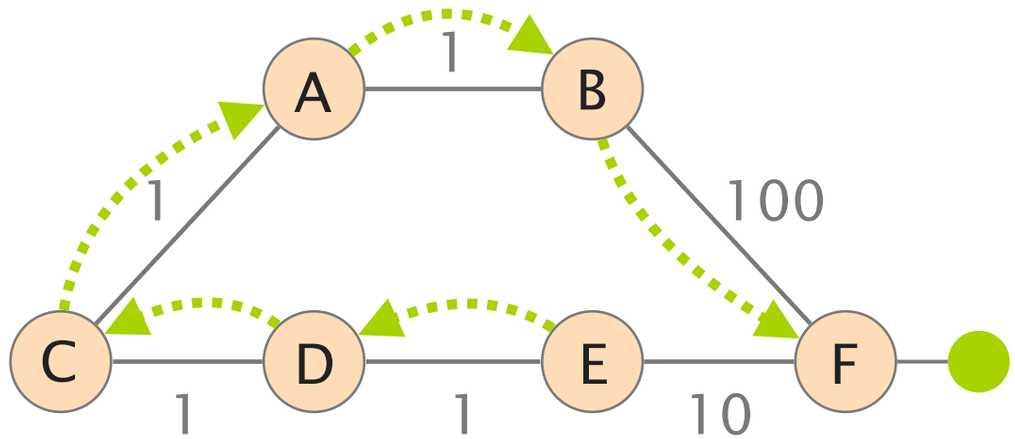

A
B
C
D
E
F
destination
1
1
100
1
1
10


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



original shortest-path

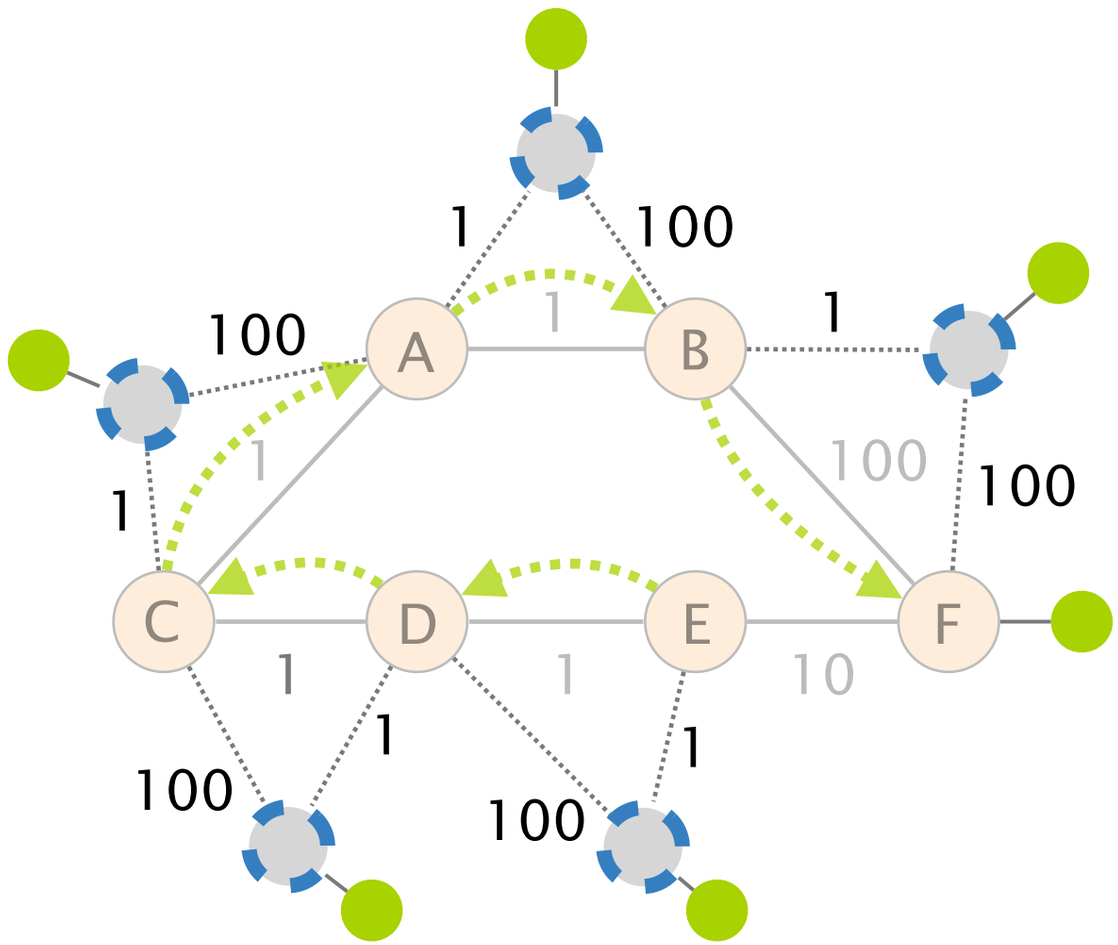
"down and to the right"



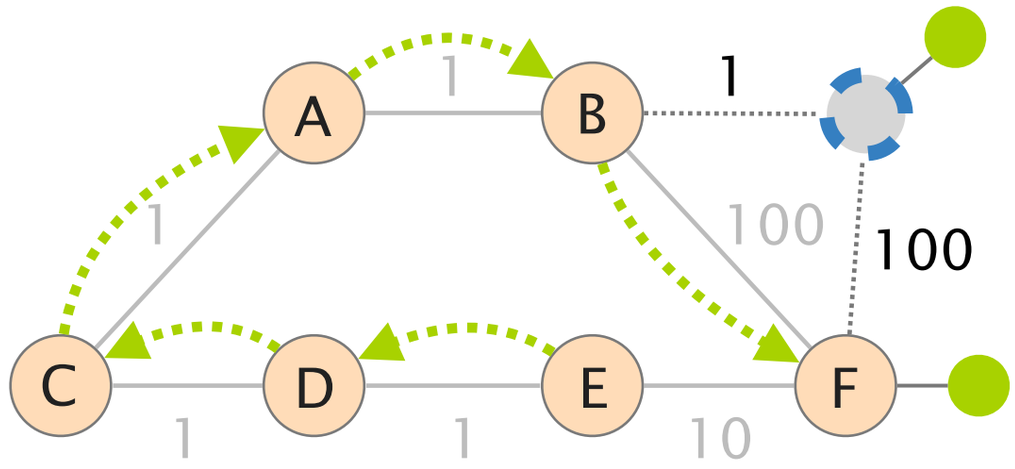
desired shortest-path

“up and to the right”

Our naive algorithm would create 5 lies—one per router



A single lie is sufficient (and necessary)

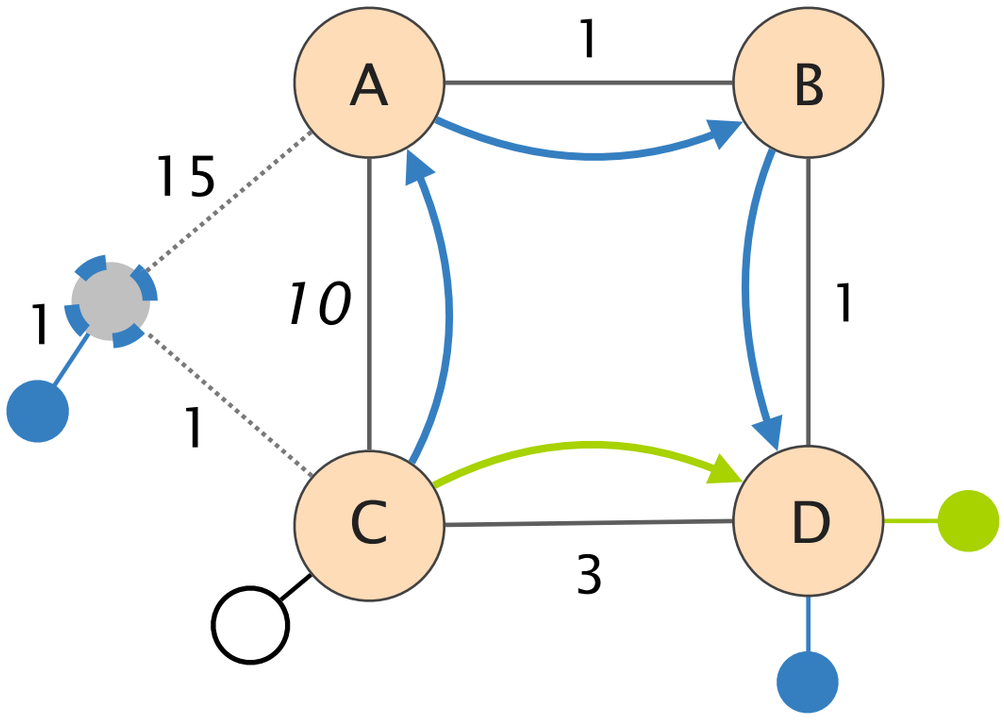


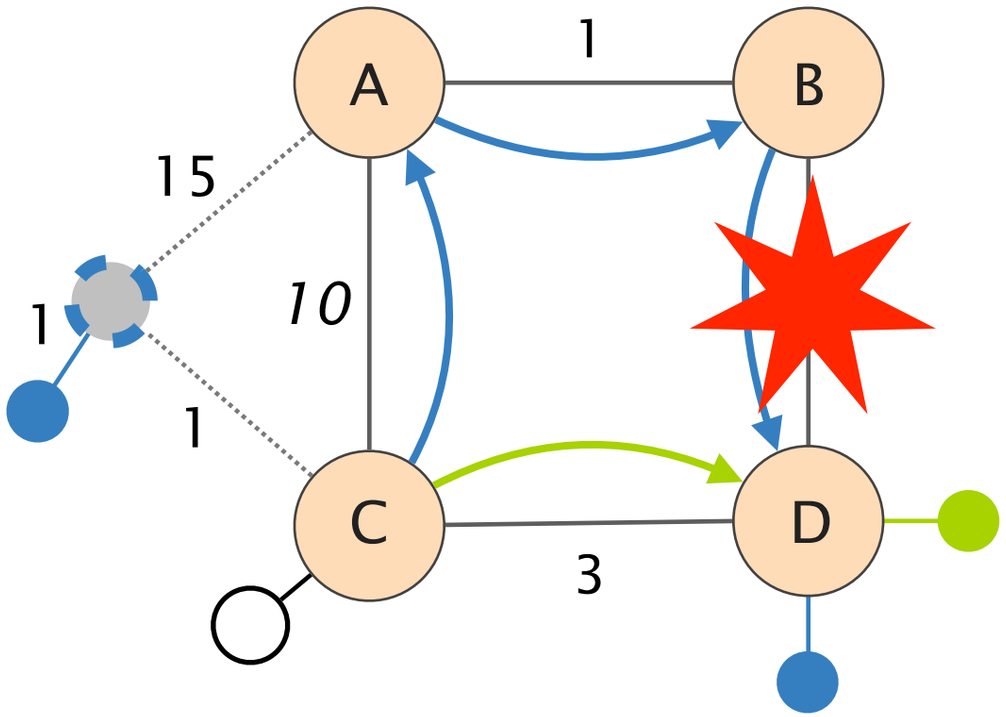
We can minimize the topology size using an Integer Linear Program

While efficient,
an ILP is inherently slow

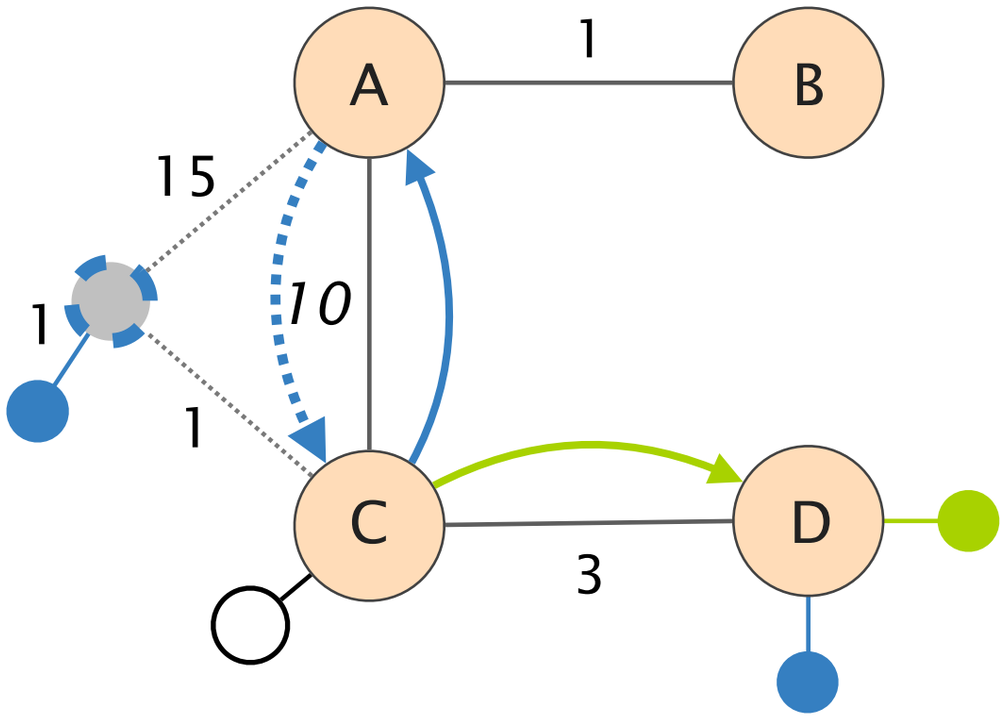
	Naive	Integer Linear Program
time	optimal	slow
space (topology size)	large	optimal

Computation time matters
in case of network failures

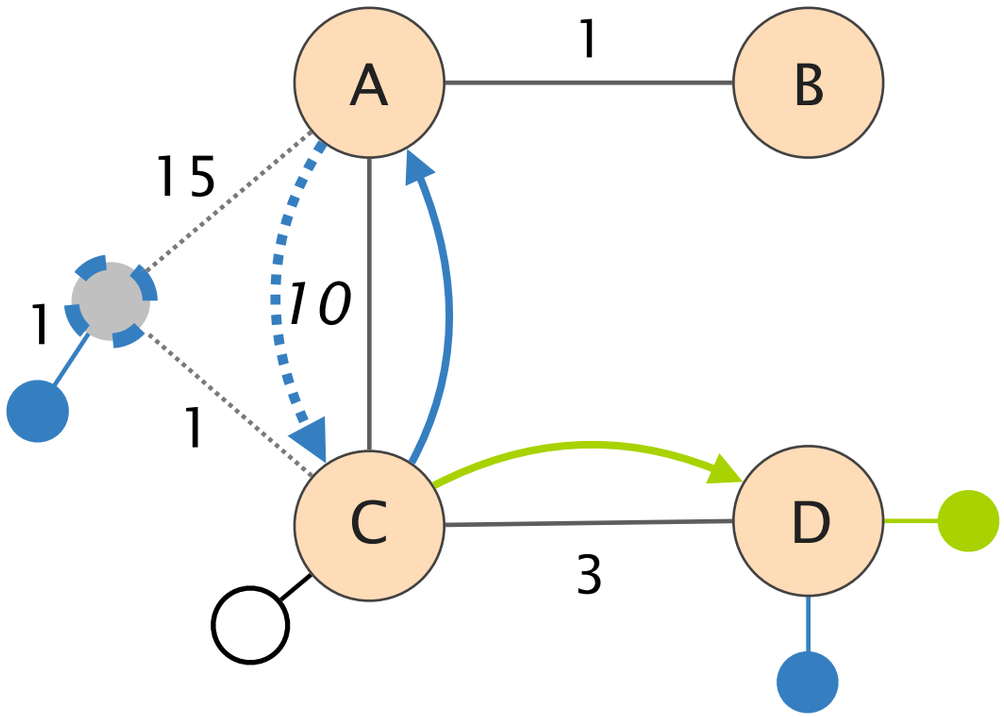

A
B
C
D
1
15
10
1
1
1
3


A
B
C
D
1
15
10
1
1
3

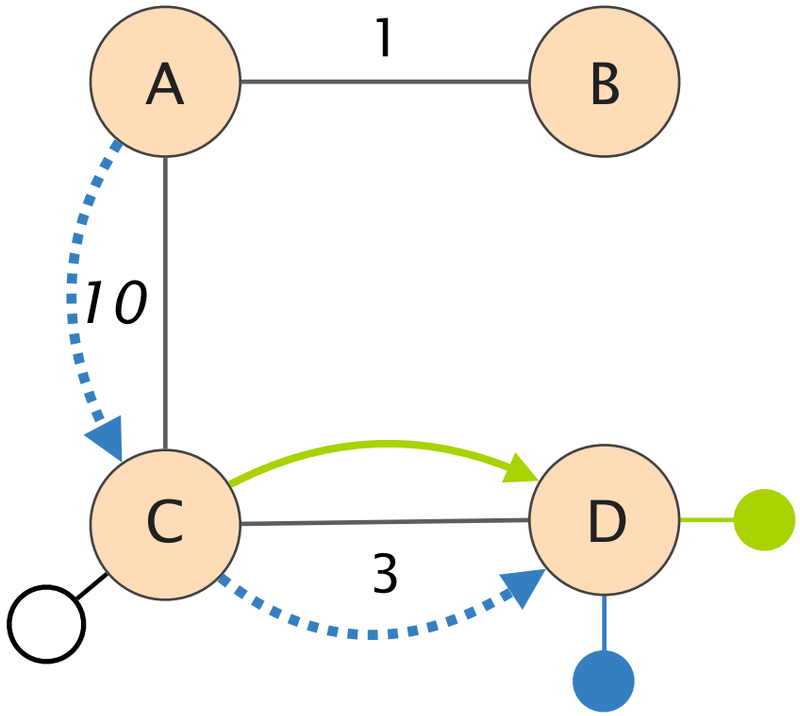
A loop is created as C starts to use A which still forwards according to the lie



The solution is to remove the lie



The solution is to remove the lie

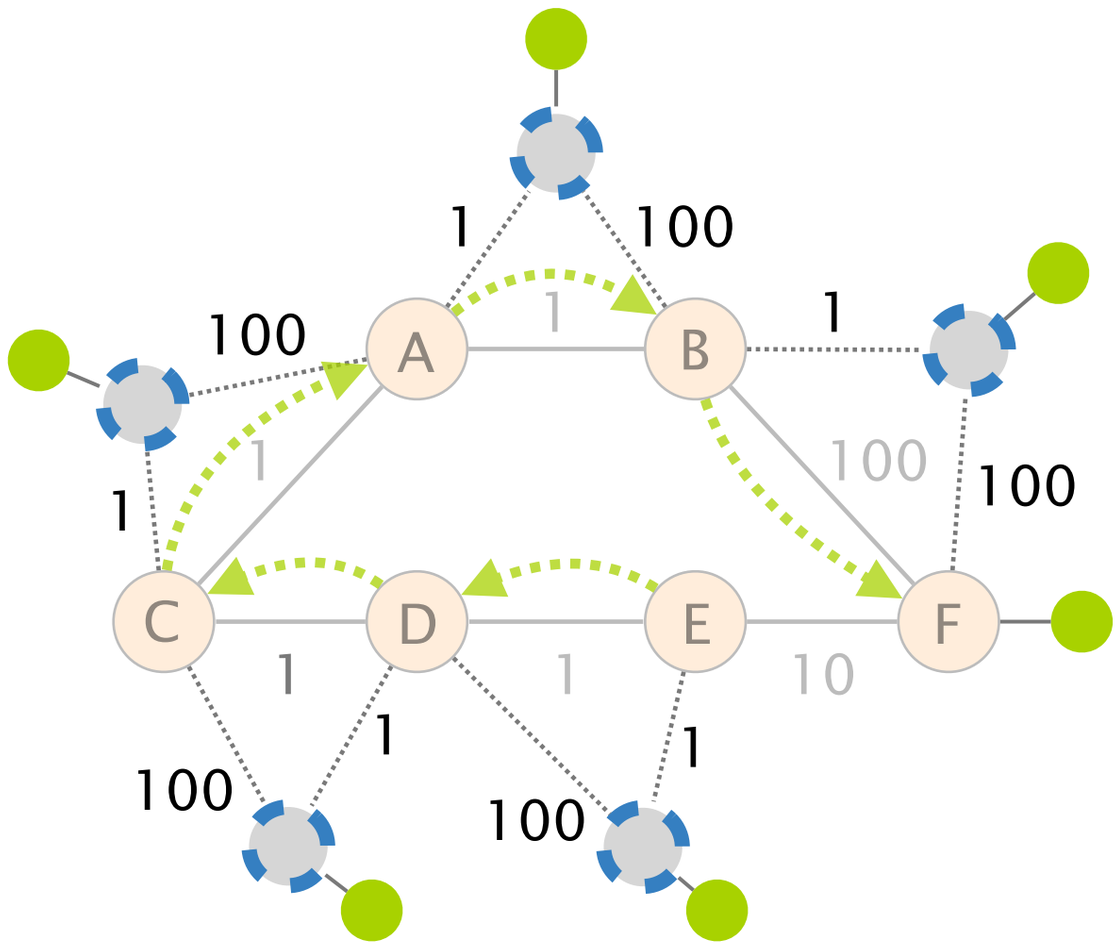


Upon failures, the network topology has to be recomputed, **fast**

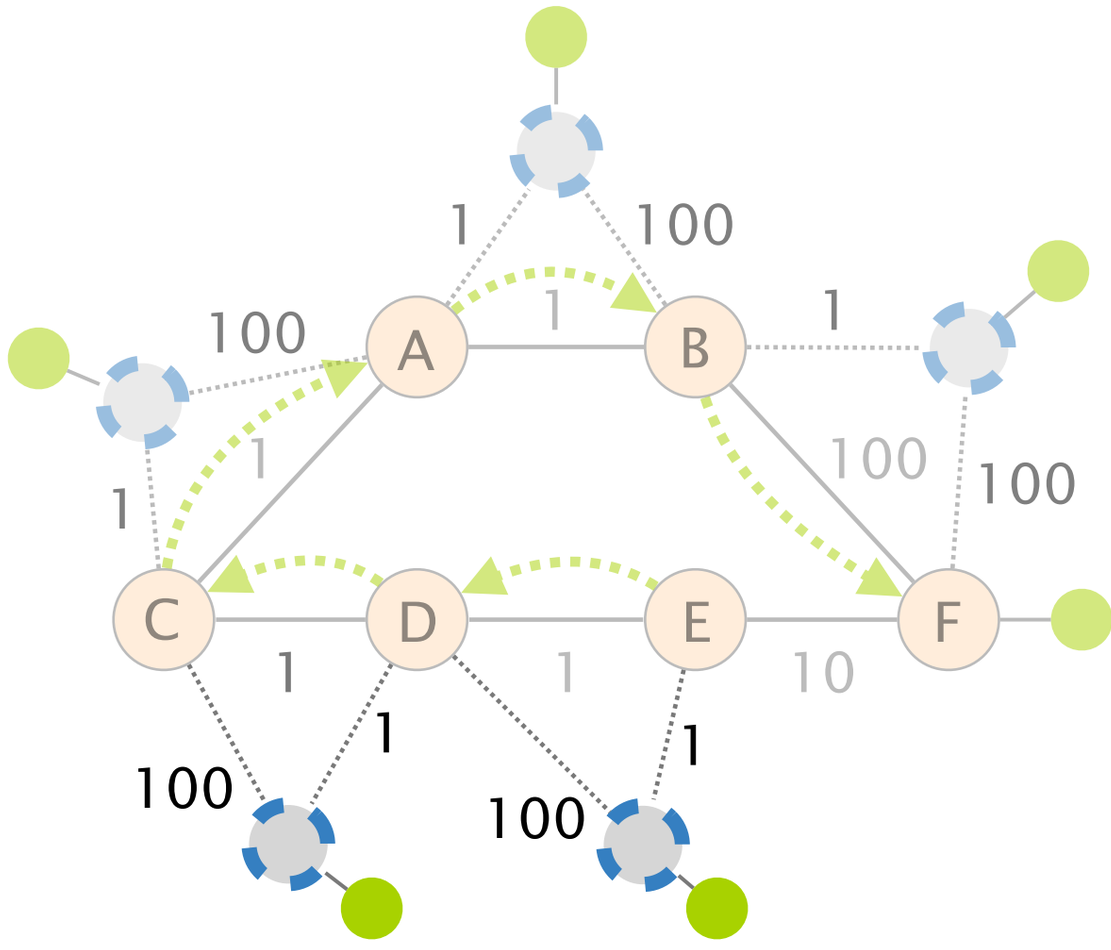
	Naive	Integer Linear Program
time	optimal	slow
space (topology size)	large	optimal

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

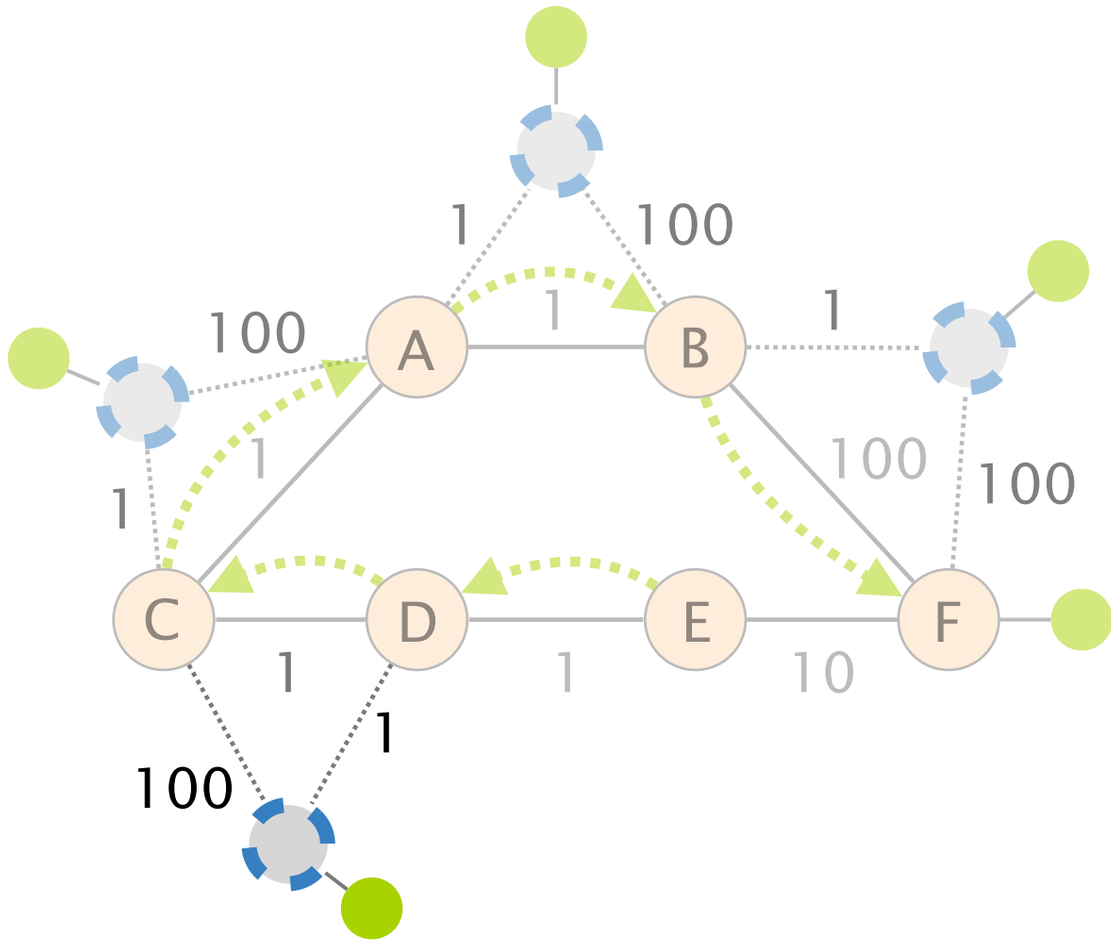
Merger iteratively tries to merge lies produced by the Naive algorithm



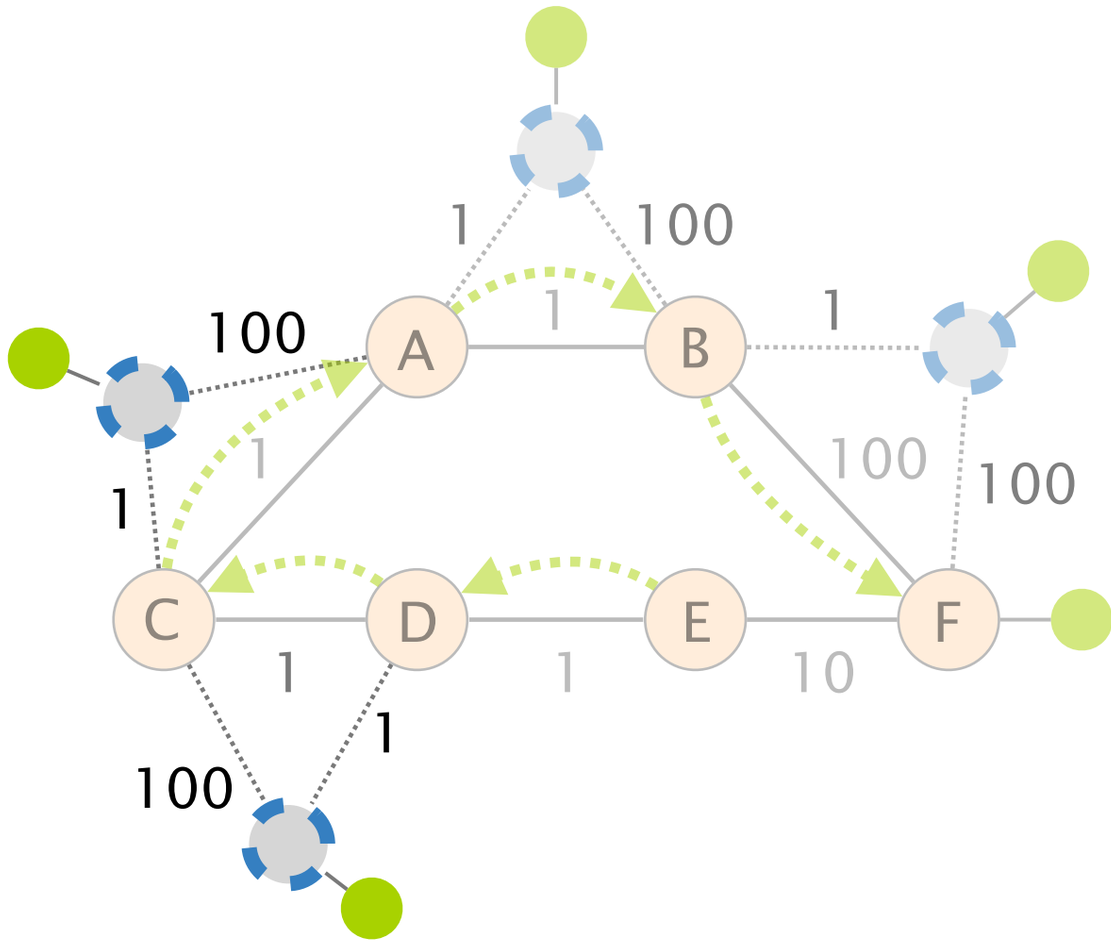
Merger iteratively tries to merge lies produced by the Naive algorithm



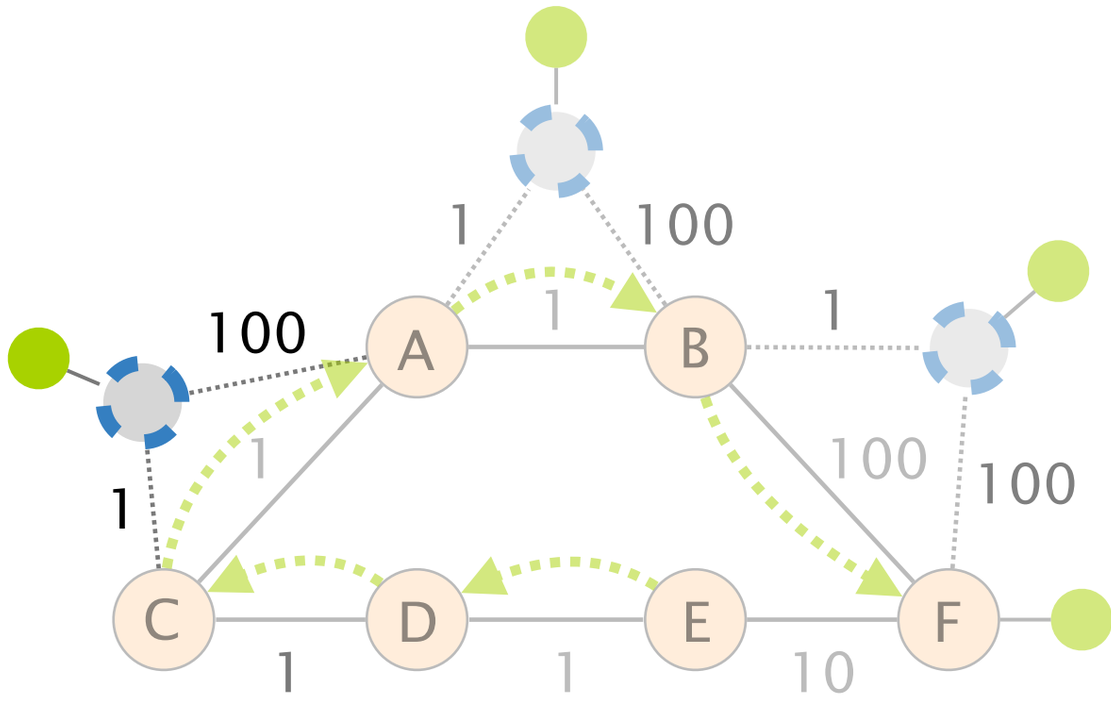
Merger iteratively tries to merge lies produced by the Naive algorithm



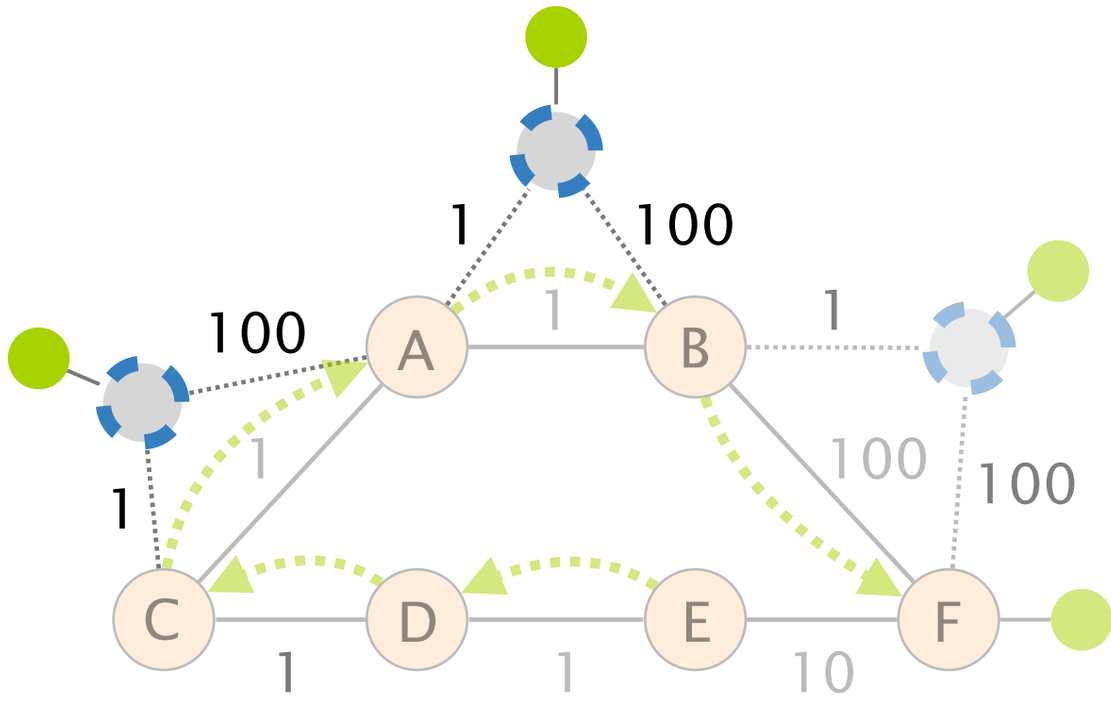
Merger iteratively tries to merge lies produced by the Naive algorithm



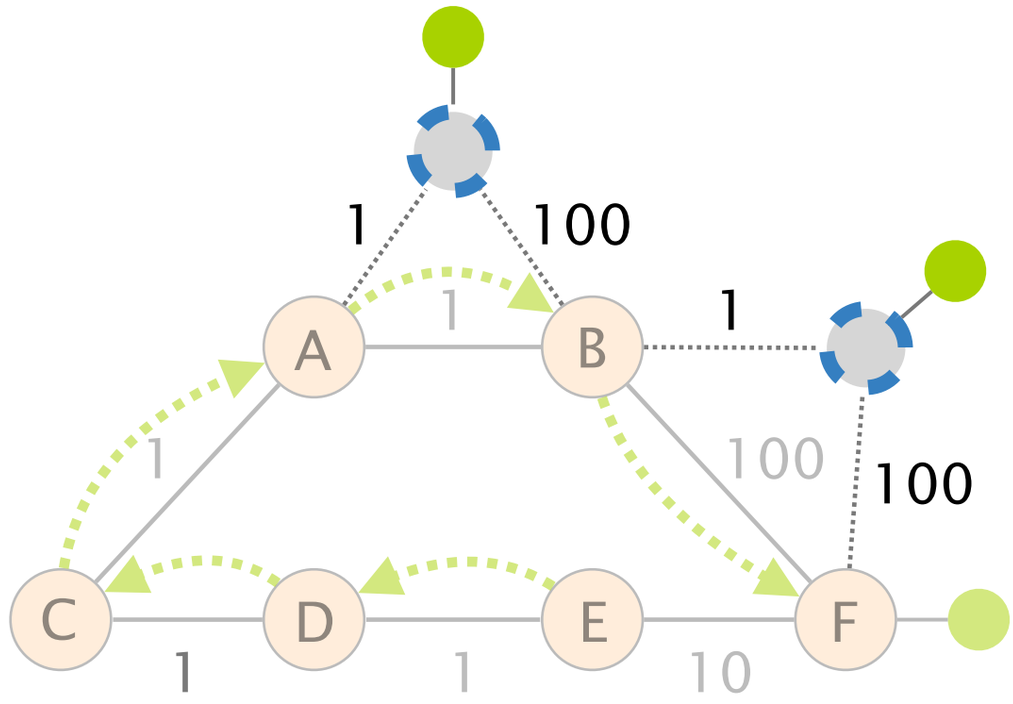
Merger iteratively tries to merge lies produced by the Naive algorithm



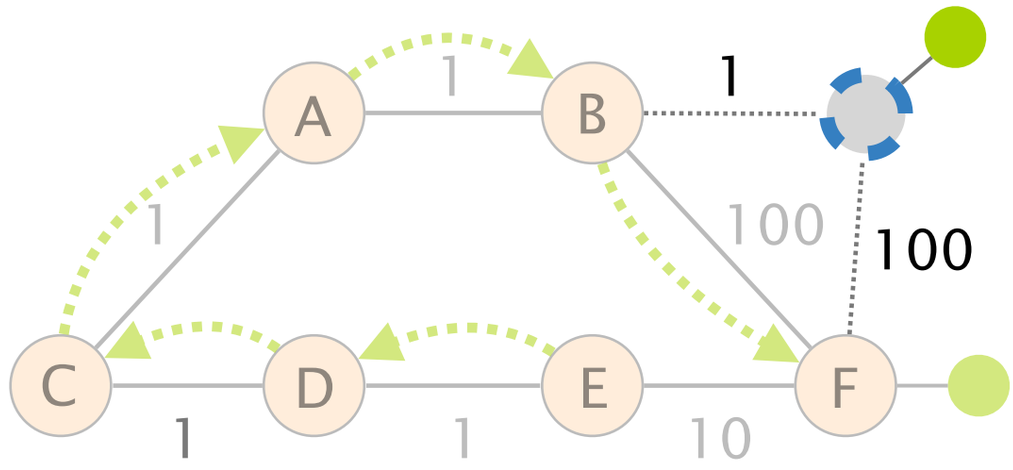
Merger iteratively tries to merge lies produced by the Naive algorithm



Merger iteratively tries to merge lies produced by the Naive algorithm



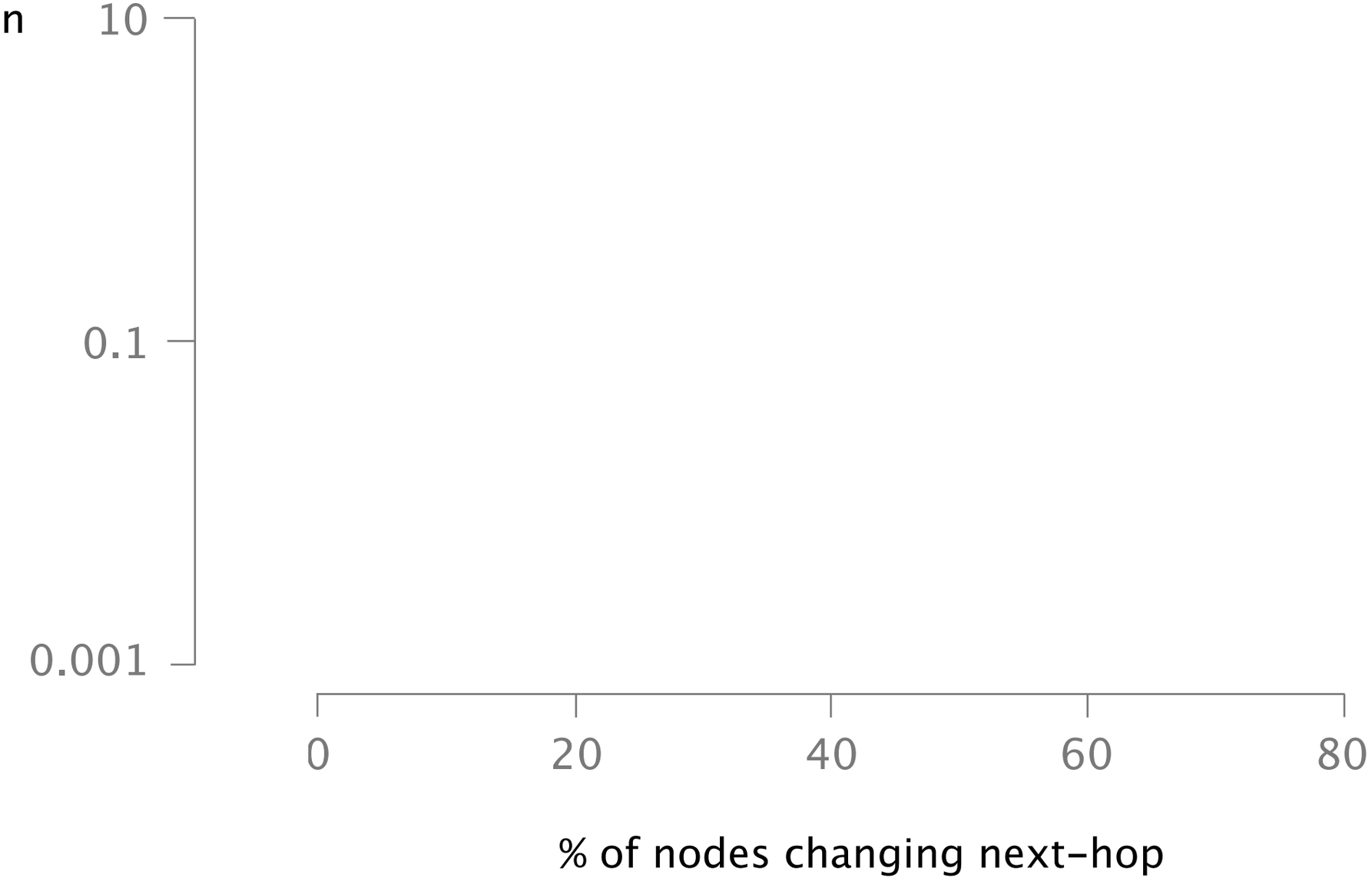
Merger iteratively tries to merge lies produced by the Naive algorithm



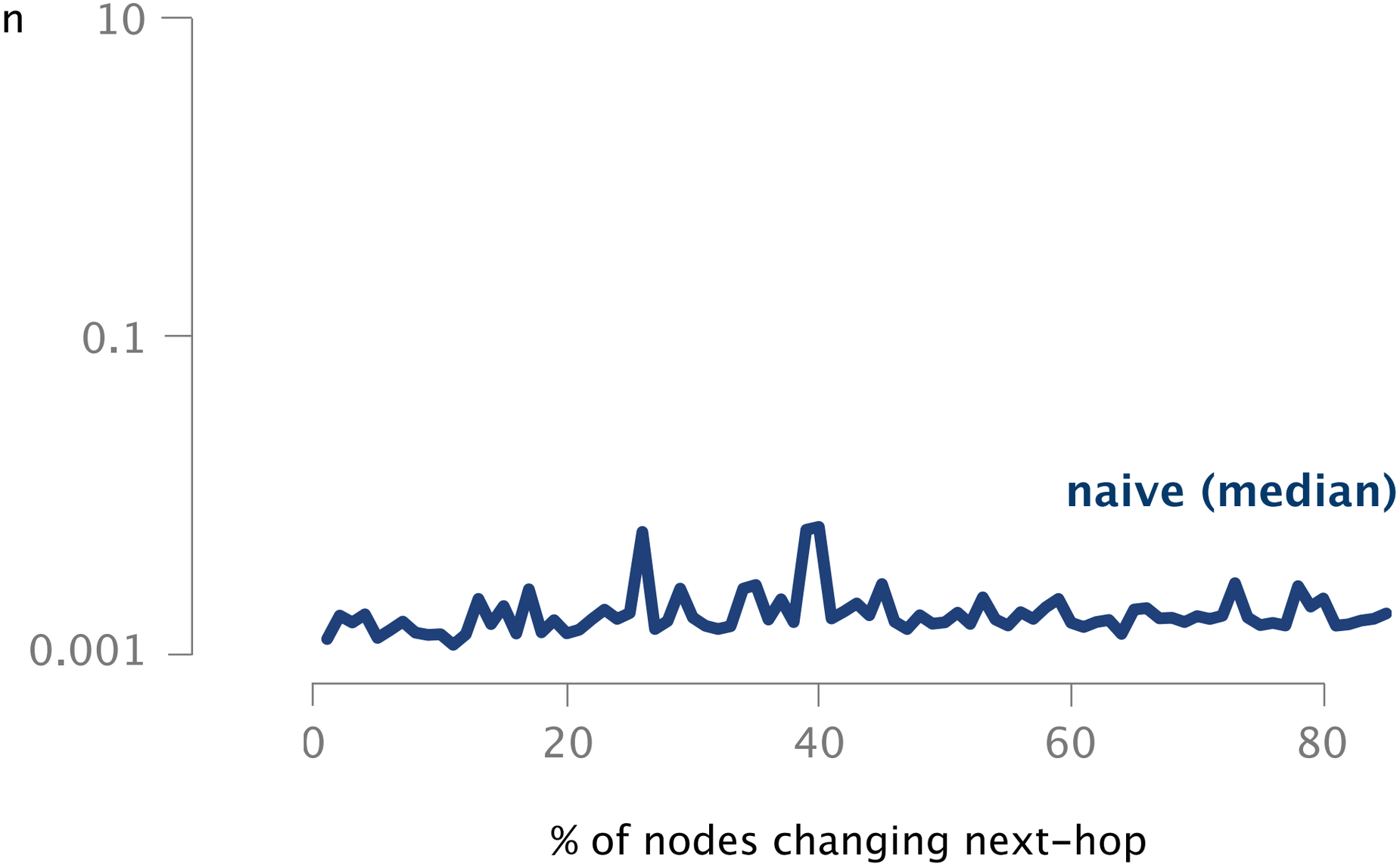
	Naive	Merger	Integer Linear Program
time	optimal	fast	slow
space (topology size)	large	small	optimal

Let’s compare the performance
of Naive and Merger

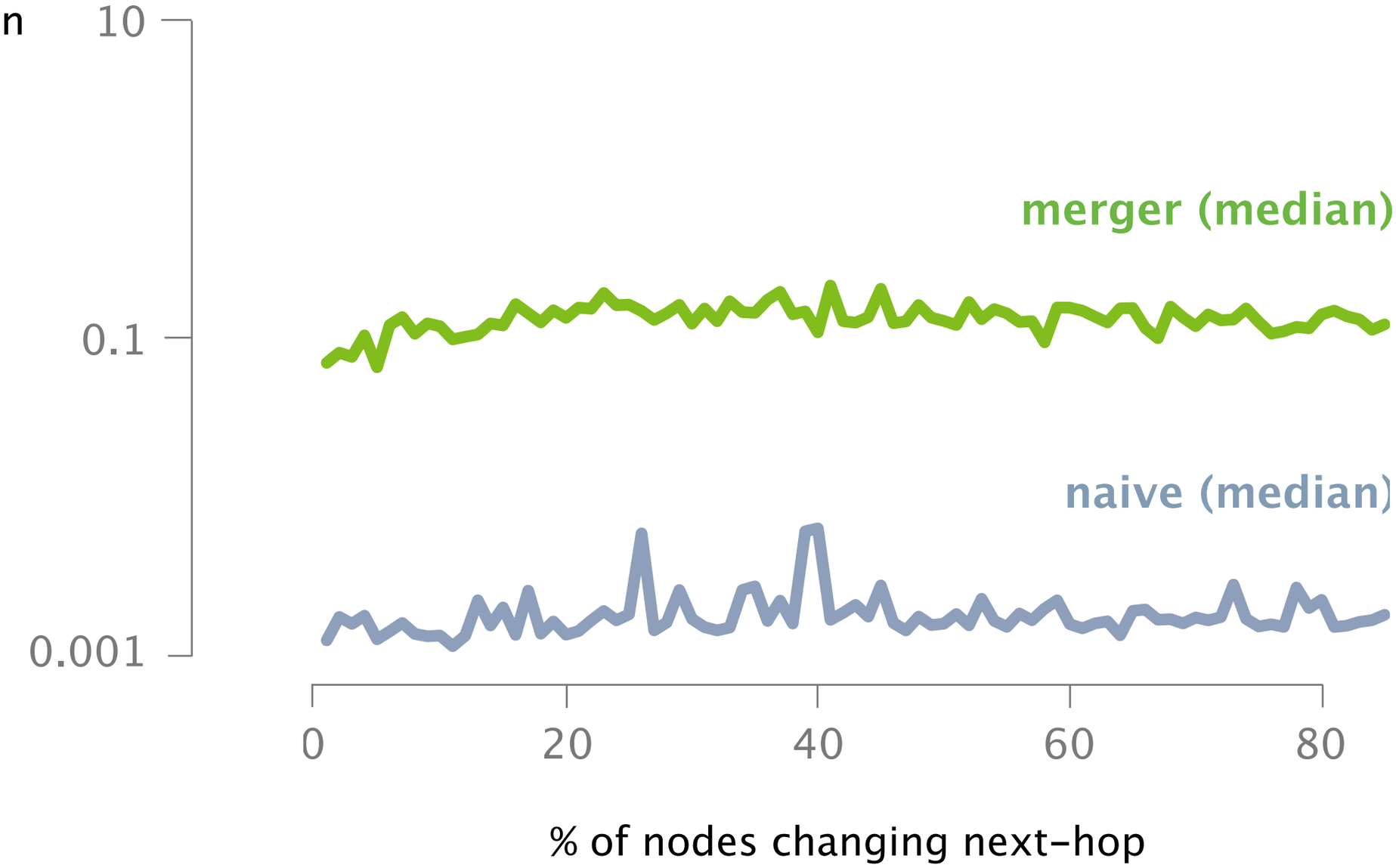
	Naive	Merger	Integer Linear Program
time	optimal	fast	slow
space (topology size)	large	small	optimal

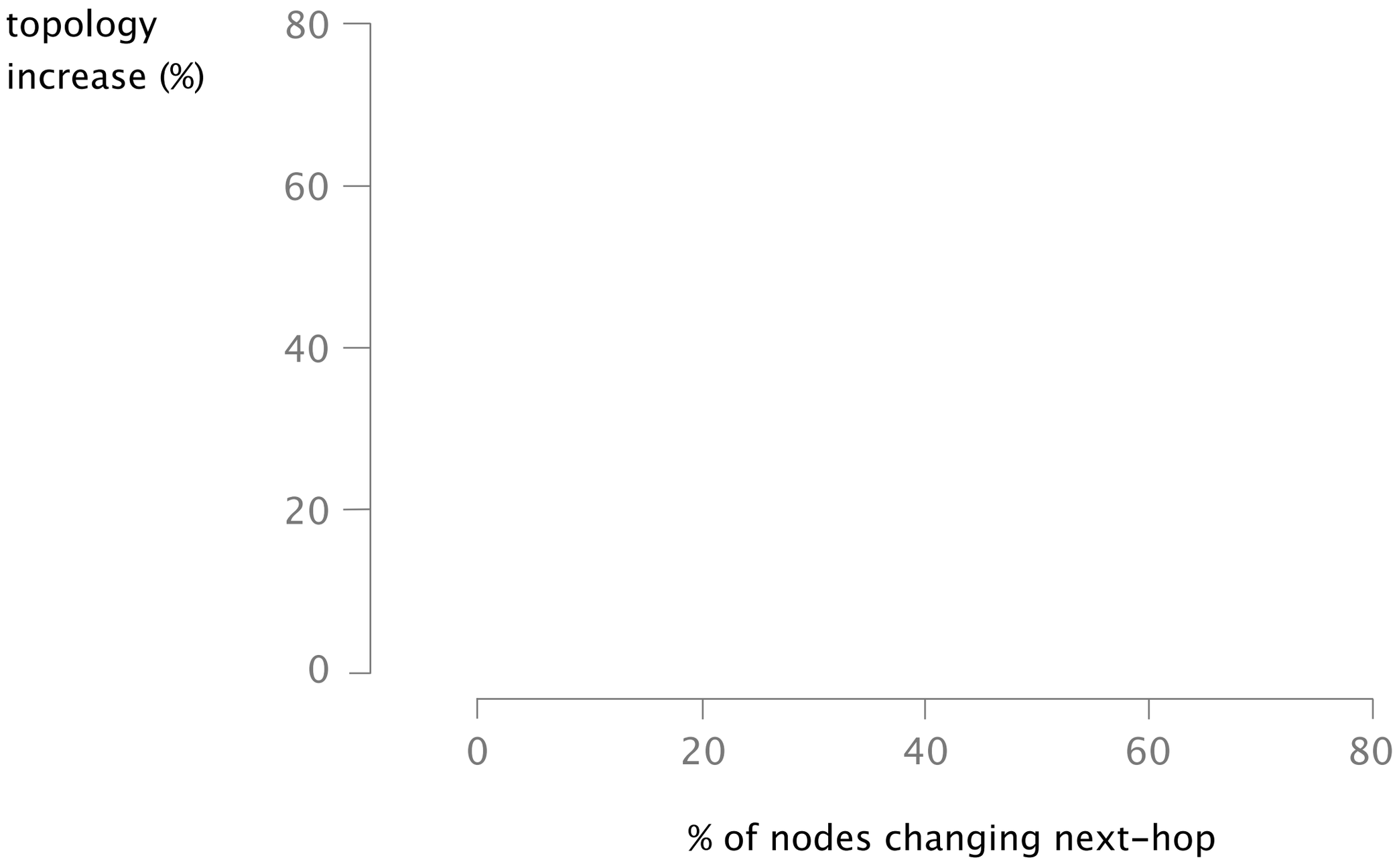

computation time (s)
10
0.1
0.001
0
20
40
60
80
% of nodes changing next-hop

Naive computes entire virtual topologies in ms

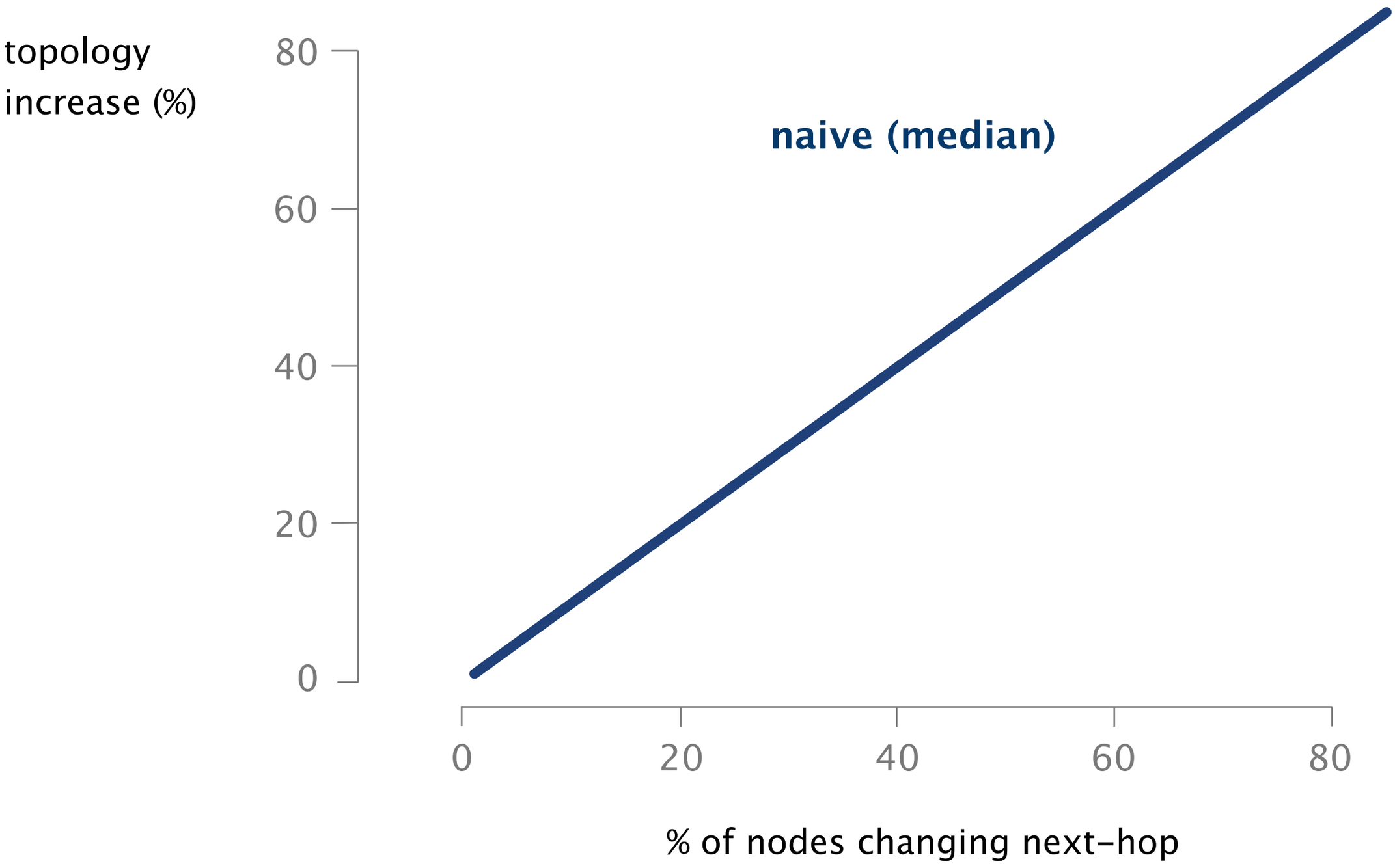


Merger is relatively slower,
but still, sub-second

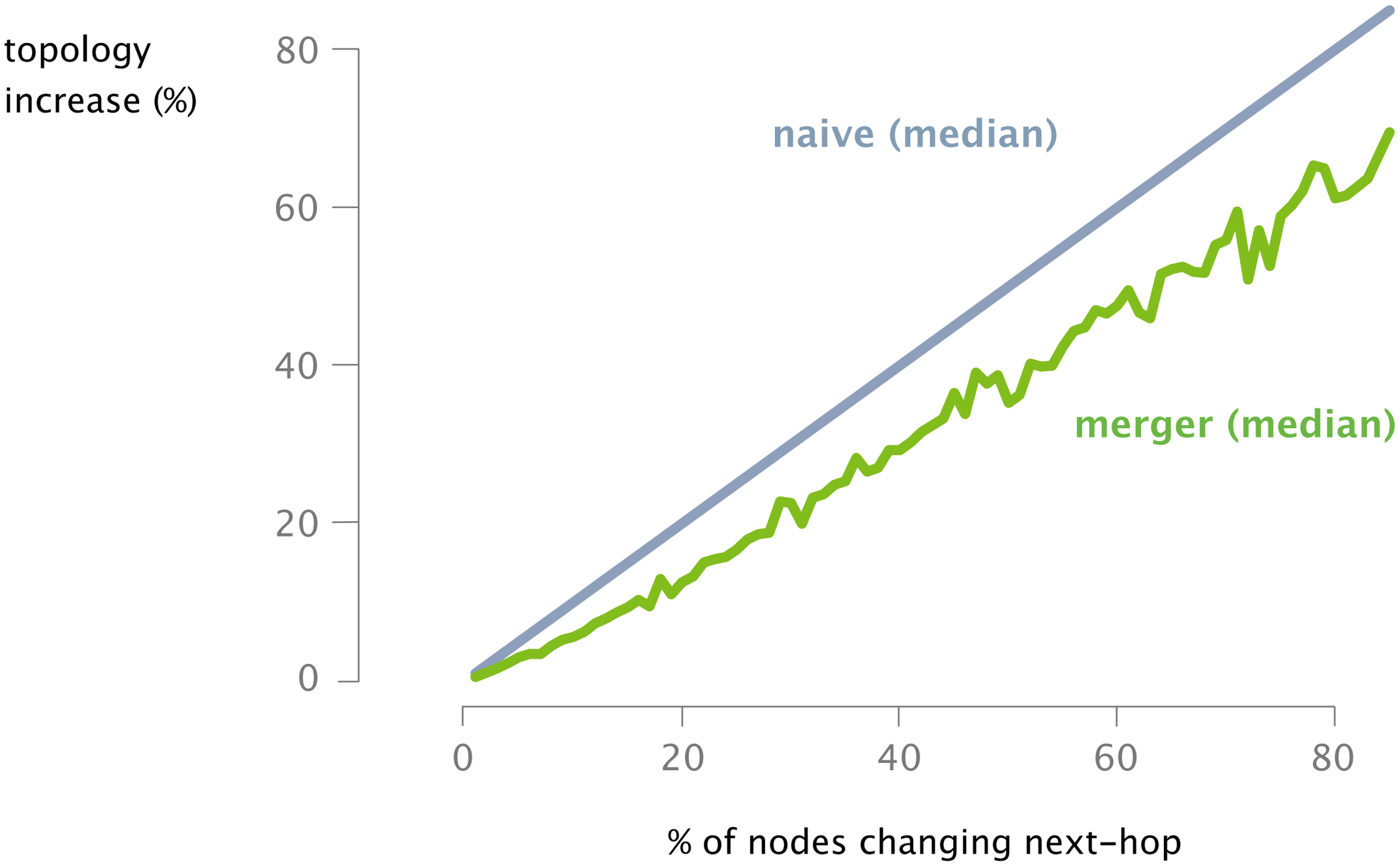



topology
increase (%)
80
60
40
20
0
0
20
40
60
80
% of nodes changing next-hop

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

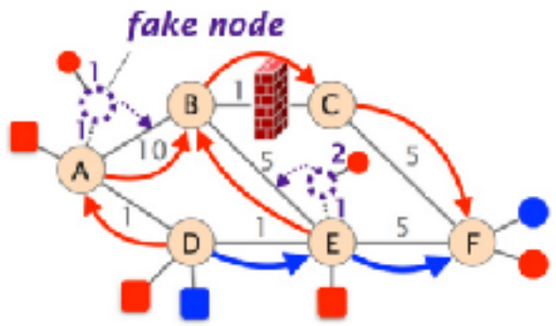
Check out our webpage

http://fibbing.net

Fibbing: Small Lies for Better Networks

Fibbing is an architecture that enables central control over distributed routing. This way, it combines the advantages of SDN (flexibility, expressivity, and manageability) and traditional (robustness, and scalability) approaches.

Fibbing introduces fake nodes and links into an underlying link-state routing protocol, so that routers compute their own forwarding tables based on the augmented topology. Fibbing is expressive, and readily supports flexible load balancing, traffic engineering, and backup routes. Fibbing works with any unmodified routers speaking OSPF.



Fibbing won the Best Paper Award at SIGCOMM 2015!

Read the papers | Look at the presentations | Watch the demo | Get the code

Carlo Daffara @cdaffara · Nov 13

Oh boy. A totally new way of thinking about the basis of SDN. Looks simple and practical. Will test. fibbing.net

2 1

OpenDaylight Project and 2 others follow

Jan-Erik Mångs @jemangs · Nov 17

Fibbing: **Central Control Over Distributed Routing** < Brilliant :-) fibbing.net

AMS-IX and 1 other follow

Michiel Appelman @michielappelman · Aug 21

Interesting concept and cool webpage: fibbing.net – **Central Control Over Distributed Routing**

Olivier Bonaventure Retweeted

ACM SIGCOMM @ACMSIGCOMM · Aug 20

SIGCOMM 2015 best paper award: "**Central Control Over Distributed Routing**" by Vissicchio et. Al., conferences.sigcomm.org/sigcomm/2015/p… #sigcomm2015

Brian Krent @BrianKrent · Nov 13

"Central Control Over Distributed Routing"

fibbing.net/files/sig15.pdf

"Fibbing: Small Lies for Better Networks"

fibbing.net

CSAIL at MIT follows

John Evdemon @jevdemon · Nov 13

Fibbing is an architecture that enables **central control over distributed routing**. Interesting idea. fibbing.net

Network programmability

through synthesis

Fibbing

"the inputs"

SyNET

"the functions"

current focus

under submission

Fibbing is limited by the configurations running on the routers

Works with a single protocol family

Dijkstra-based shortest-path routing

Can lead to loads of messages

if the configuration is not adapted

Suffers from reliability issues

need to remove the lies upon failures

Inputs

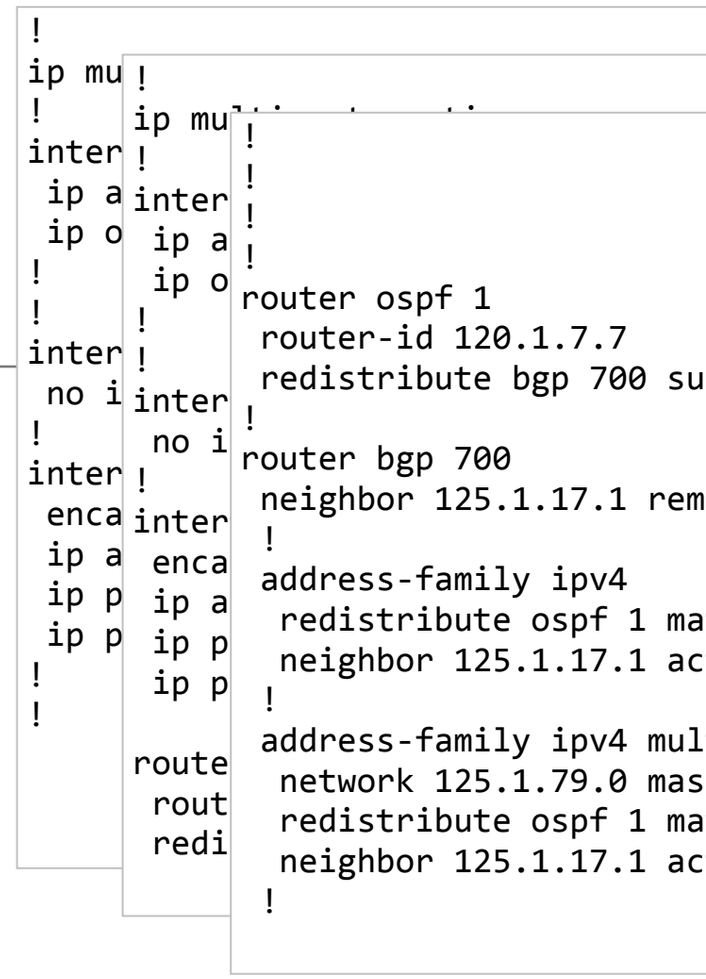
Outputs

Network specification (*N*)

Physical topology (φ_N)

High-level requirements (φ_R)

SyNET



Inputs

Network specification (N)

Physical topology (φ_N)

High-level requirements (φ_R)

Inputs

Network specification (*N*)

Physical topology (φ_N)

High-level requirements (φ_R)

A set of Datalog *rules* that formalize how routers build their forwarding state

```
Fwd(Net, Node, Next) :-
  Route(Net, Node, Next, Proto),
  SetAD(Protocol, Node, Cost)
  minAD(Net, Node, Cost)
```

Inputs

Network specification (N)

Physical topology (φ_N)

High-level requirements (φ_R)

A set of constraints over the *input* predicates of the Datalog program

Inputs

Network specification (N)

Physical topology (φ_N)

High-level requirements (φ_R)

A set of constraints over the *output* predicates of the Datalog program

"Traffic from R1 to R5
should flow via R2 and R3"

Inputs

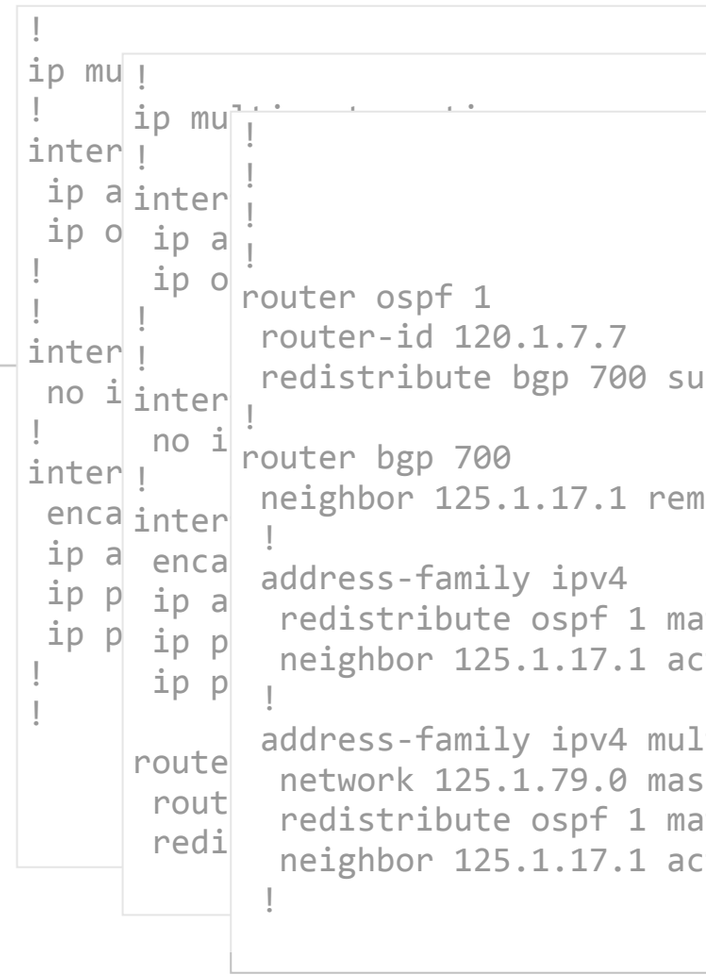
Outputs

Network specification (*N*)

Physical topology (φ_N)

High-level requirements (φ_R)

SyNET



problem

Given N, φ_N, φ_R

SyNet generates a Datalog input I such that the topology φ_N and routing φ_R constraints are satisfied for the given network specification N

problem

Given N, φ_N, φ_R

SyNet generates a Datalog input I such that the topology φ_N and routing φ_R constraints are satisfied for the given network specification N

challenge

this is undecidable (in general)

problem

Given N, φ_N, φ_R

SyNet generates a Datalog input I such that the topology φ_N and routing φ_R constraints are satisfied for the given network specification N

challenge

this is undecidable (in general)

key ideas

make the problem finite, use divide-and-conquer

convert into a satisfiability question (on SMT constraints)

“scale” using domain-specific heuristics

SyNET can generate configurations for (small) networks

	# routers		
	4	9	16
	static		
# protocols	static, OSPF		
	static, OSPF, BGP		

SyNET can generate configurations
for (small) networks

		# routers		
		4	9	16
# protocols	static	1.8s	18.2s	116.1s
	static, OSPF	4.2s	37.0s	197.0s
	static, OSPF, BGP	13.8s	189.4s	577.4s

SyNET can generate configurations
for (small) networks (new version goes to 81 routers)

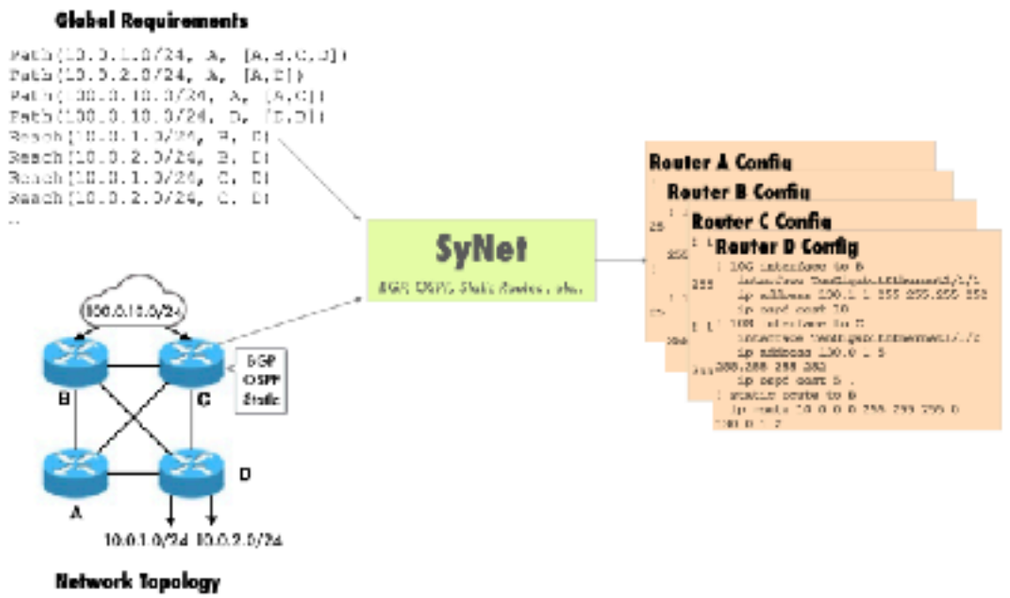
		# routers		
		4	9	16
# protocols	static	1.8s	18.2s	116.1s
	static, OSPF	4.2s	37.0s	197.0s
	static, OSPF, BGP	13.8s	189.4s	577.4s

Check out our webpage

synet.ethz.ch

SyNet: Network-wide Configuration Synthesis

SyNet automatically synthesizes configurations for routers running **multiple interacting** protocols, including policy-based protocols (**BGP**) and shortest-path protocols (**OSPF**), and it also supports static routes. **SyNet** guarantees that the network's routers converge to a forwarding state that conforms with all high-level requirements provided by the network operator.



Why Multiple Protocols?

Routing protocols have different expressiveness. Configuring multiple protocols is therefore often required to produce a forwarding state compliant with the operator's requirements.

Automatic vs. Manual Configuration

Routing protocols are complex. Moreover, protocols often have complex interdependencies. For example, BGP uses interdomain routing costs as input for selecting the best route. Not surprisingly, the majority of network downtimes are caused by incorrect

Network programmability

through synthesis

Fibbing

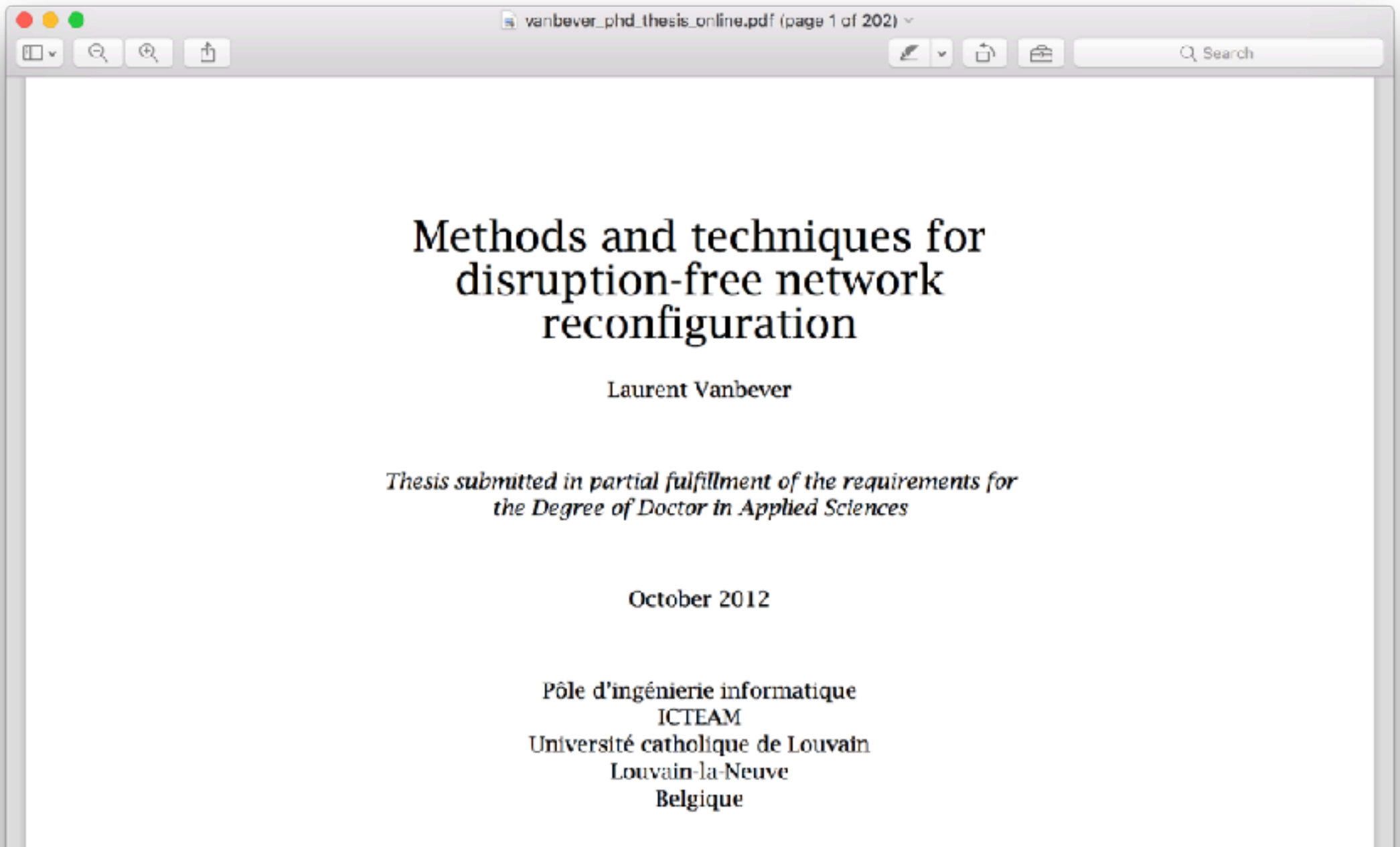
“the inputs”

SyNET

“the functions”

Raw network programmability is only the beginning, what about configuration updates?

[SIGCOMM'11, INFOCOM'12, TON'12, TON'13, INFOCOM'13, TON'17]


vanbever_phd_thesis_online.pdf (page 1 of 202)

Search

Methods and techniques for disruption-free network reconfiguration

Laurent Vanbever

Thesis submitted in partial fulfillment of the requirements for the Degree of Doctor in Applied Sciences

October 2012

Pôle d'ingénierie informatique
ICTEAM
Université catholique de Louvain
Louvain-la-Neuve
Belgique

Raw network programmability is only the beginning,
what about network visibility?

[Hotnets'16]

vanbever_millefeuille_hotnets_2016.pdf (page 1 of 7)

Search

Mille-Feuille: Putting ISP traffic under the scalpel

Olivier Tilmans [‡*], Tobias Bühler [§], Stefano Vissicchio [†], Laurent Vanbever [§]
[‡] Université catholique de Louvain, [§] ETH Zürich, [†] University College London
[‡]olivier.tilmans@uclouvain.be, [§]{buehlert, lvanbever}@ethz.ch,
[†]s.vissicchio@cs.ucl.ac.uk

ABSTRACT

For Internet Service Provider (ISP) operators, getting an accurate picture of how their network behaves is challenging. Given the traffic volumes that their networks carry and the impossibility to control end-hosts, ISP operators are typically forced to randomly sample traffic, and rely on aggregated statistics. This provides coarse-grained visibility, at a time resolution that is far from ideal (seconds or minutes).

In this paper, we present *Mille-Feuille*, a novel monitoring architecture that provides fine-grained visibility over ISP traffic. Mille-Feuille schedules activation and deactivation

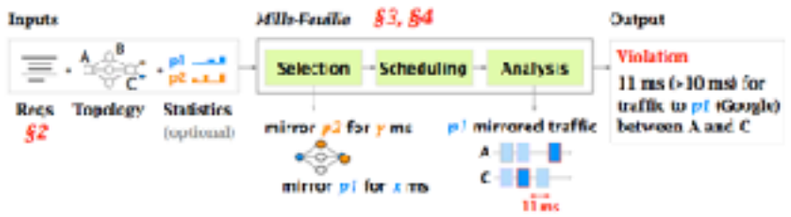
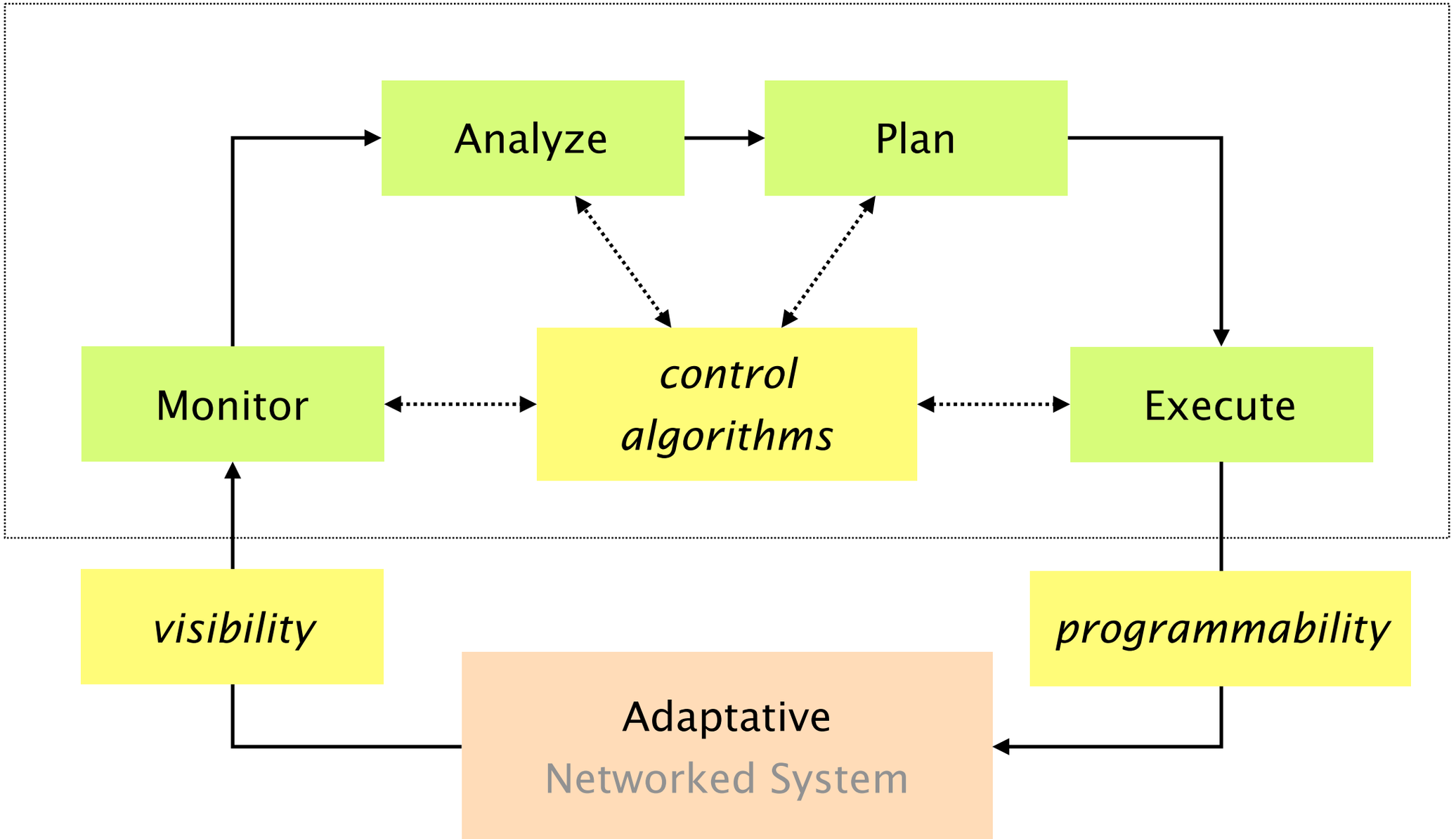


Figure 1: From high-level requirements, Mille-Feuille activates and deactivates fine-grained mirroring rules network-wide, to capture thin slices of traffic, at scale, in $\mathcal{O}(ms)$.

operators lack control of which aggregated statistics are re-

Network controller



Network programmability

A primer on routing synthesis

Laurent Vanbever

www.vanbever.eu

Schloss Dagstuhl

January 18 2017