SWIFT

Predictive Fast Reroute upon Remote BGP Disruptions

Laurent Vanbever

ETH Zürich (D-ITET)

Munich Internet Research Retreat

November 25 2016

Human factors are responsible

for 50% to 80% of network outages

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

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"

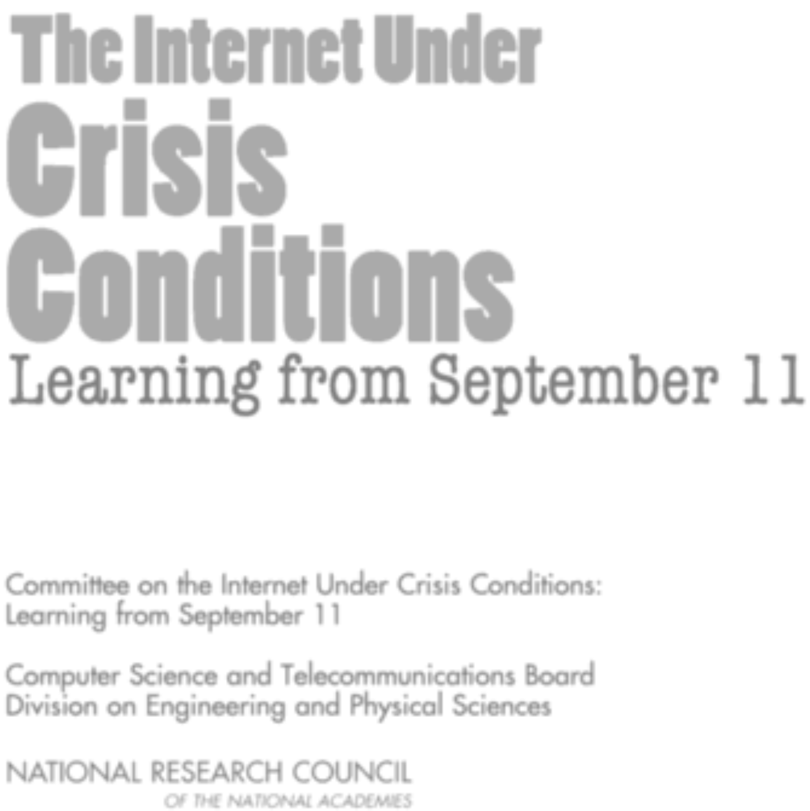
The Internet Under Crisis Conditions

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


The Internet Under
Crisis
Conditions
Learning from September 11

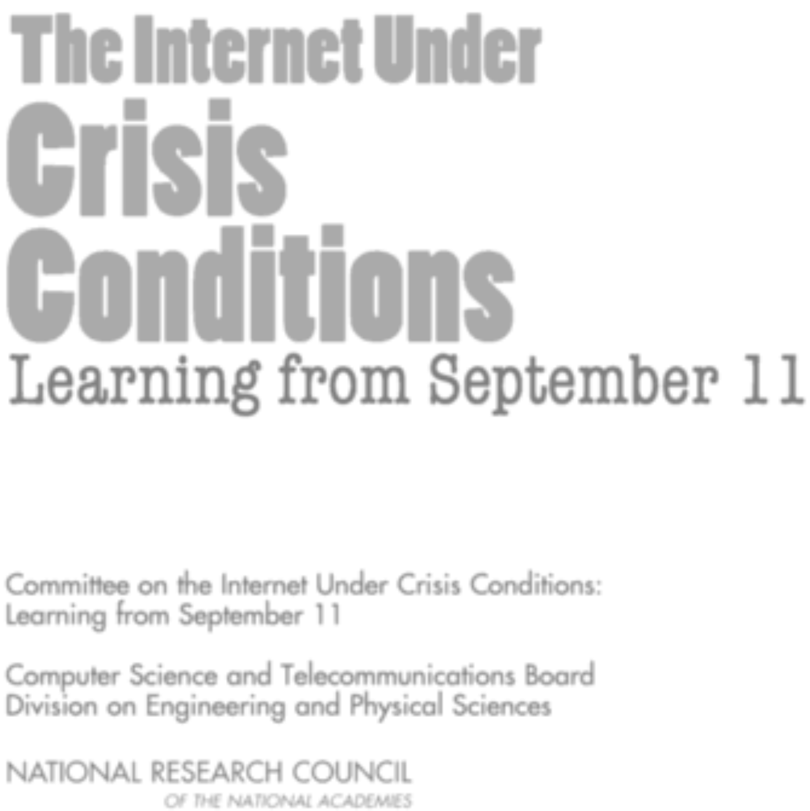
Committee on the Internet Under Crisis Conditions:
Learning from September 11

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



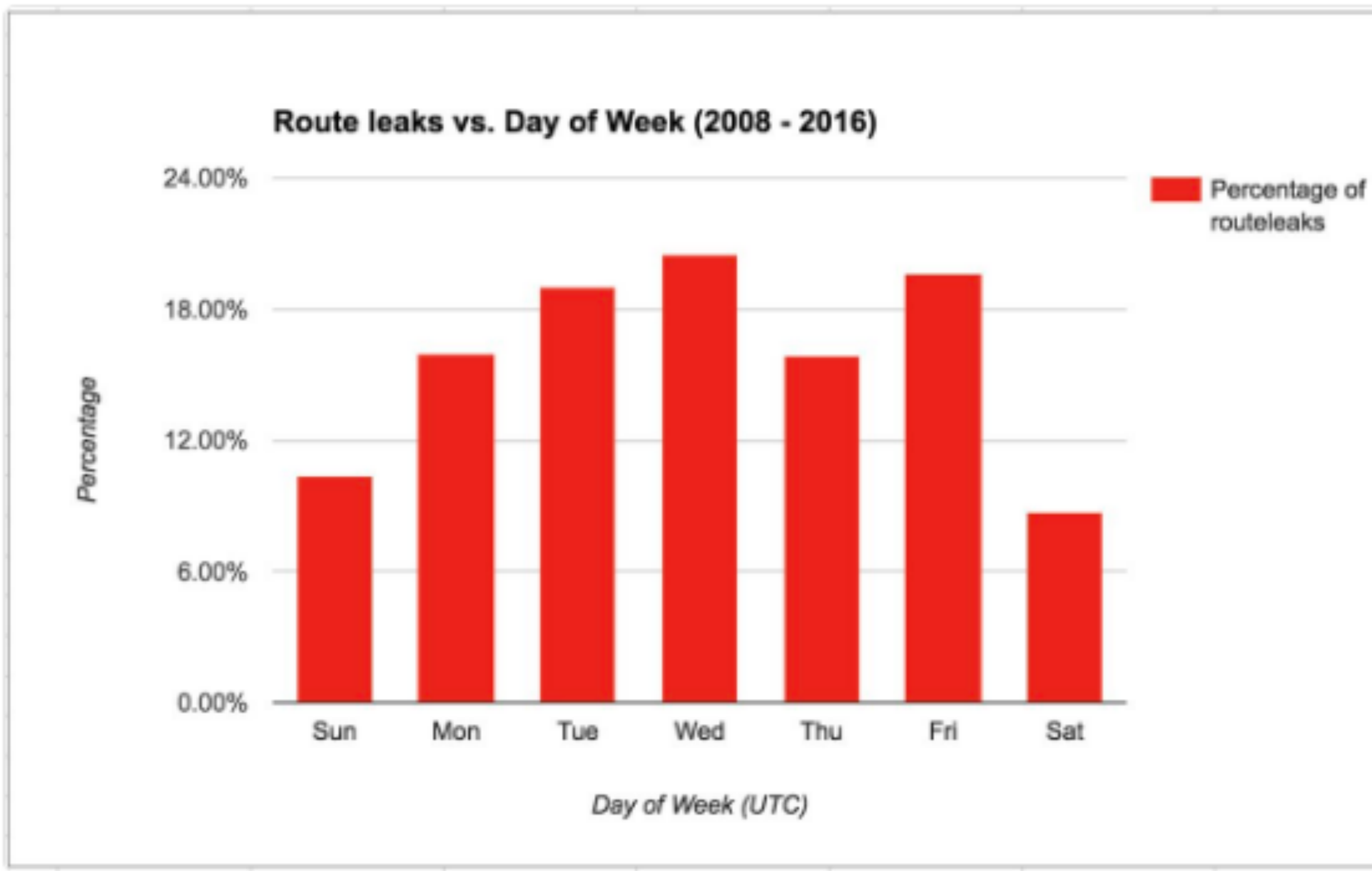
Internet advertisements rates suggest that The Internet was more stable than normal on Sept 11

Information suggests that operators were watching the news instead of making changes to their infrastucture

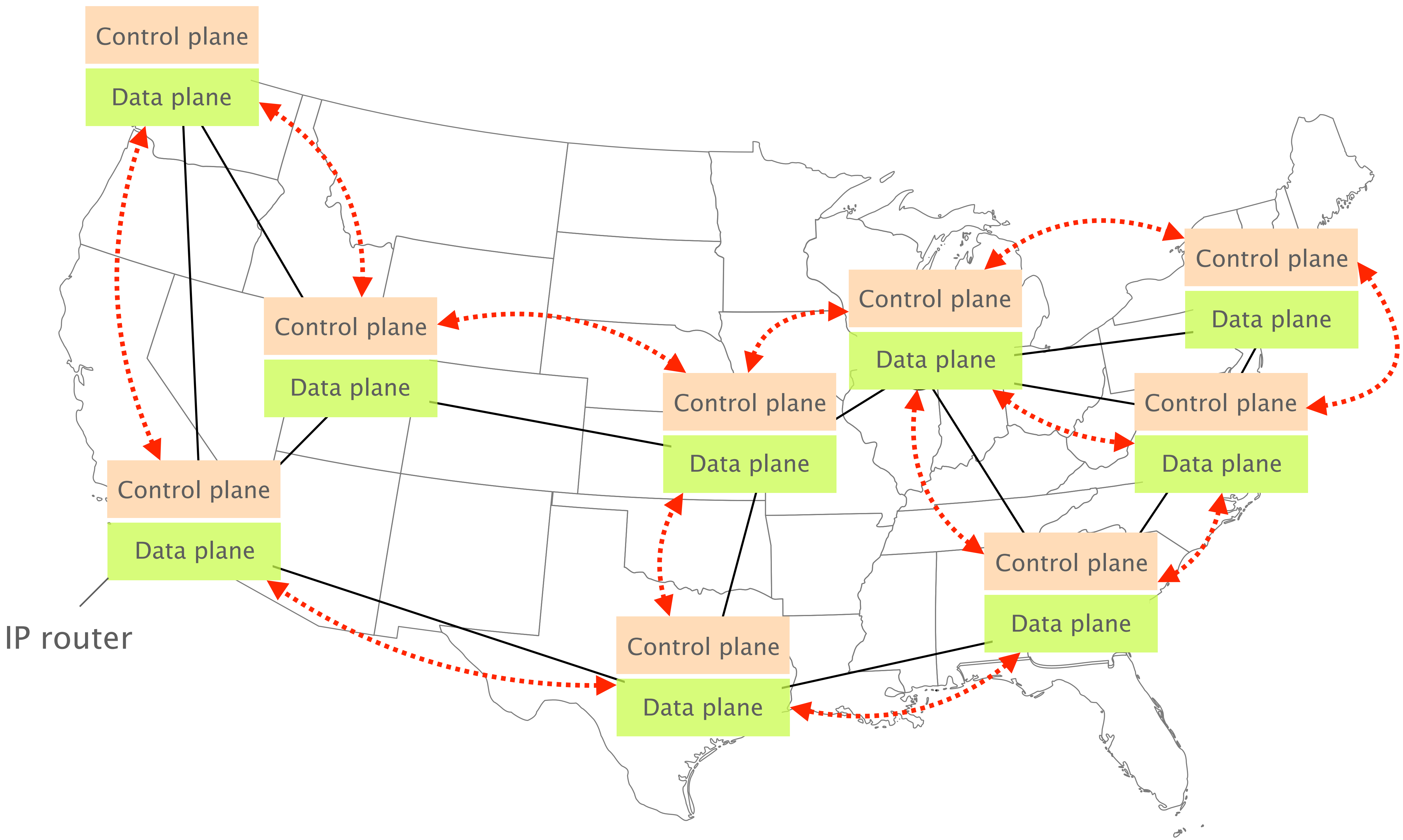
Job Snijders
@JobSnijders

Follow

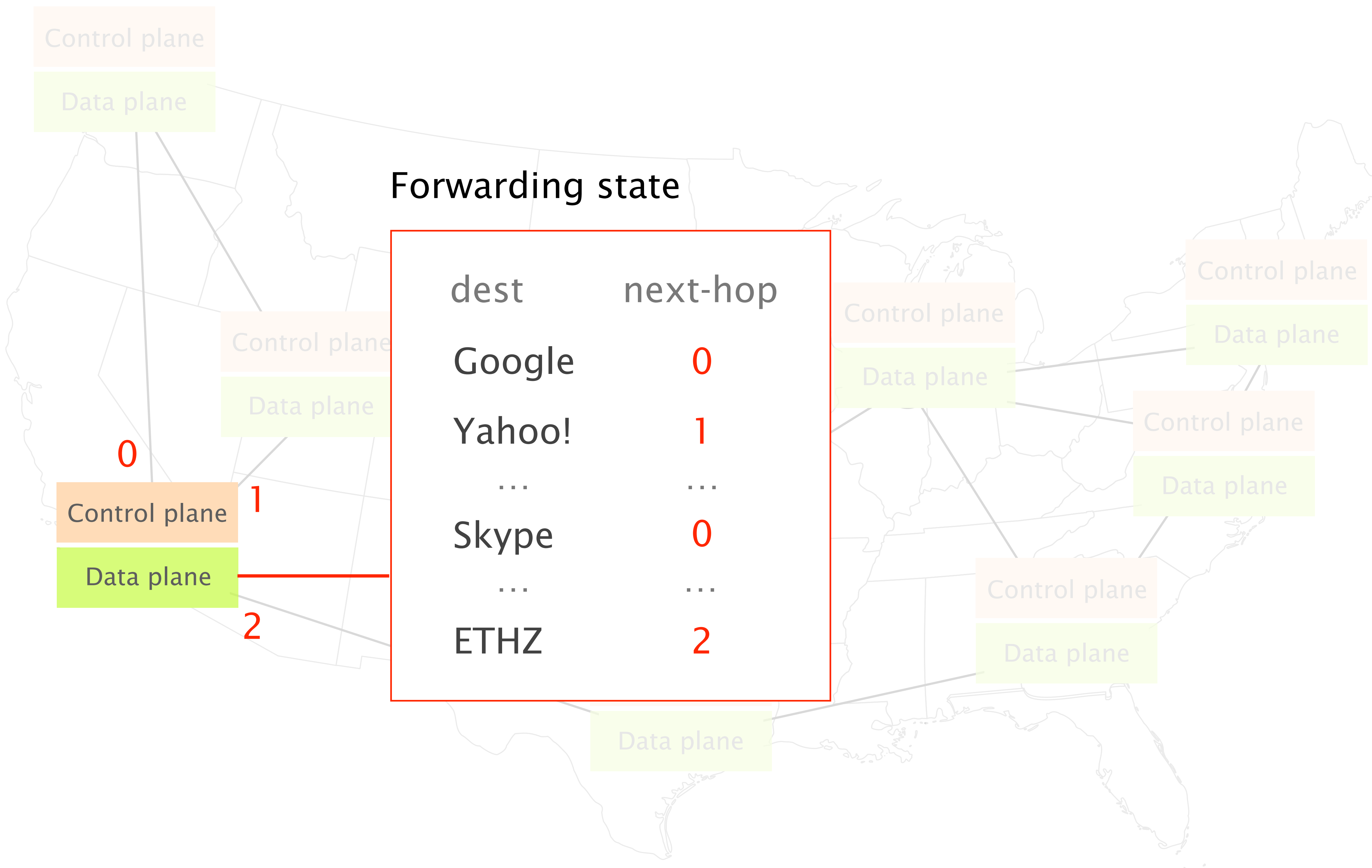
Fun fact: most BGP route leaks happen on Wednesdays, but in the weekend us humans collectively take a break! :-)



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



dest	next-hop
Google	0
Yahoo!	1
…	…
Skype	0
…	…
ETHZ	2

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
```

redistribute bgp 700 subnets — Anything else than 700 creates blackholes

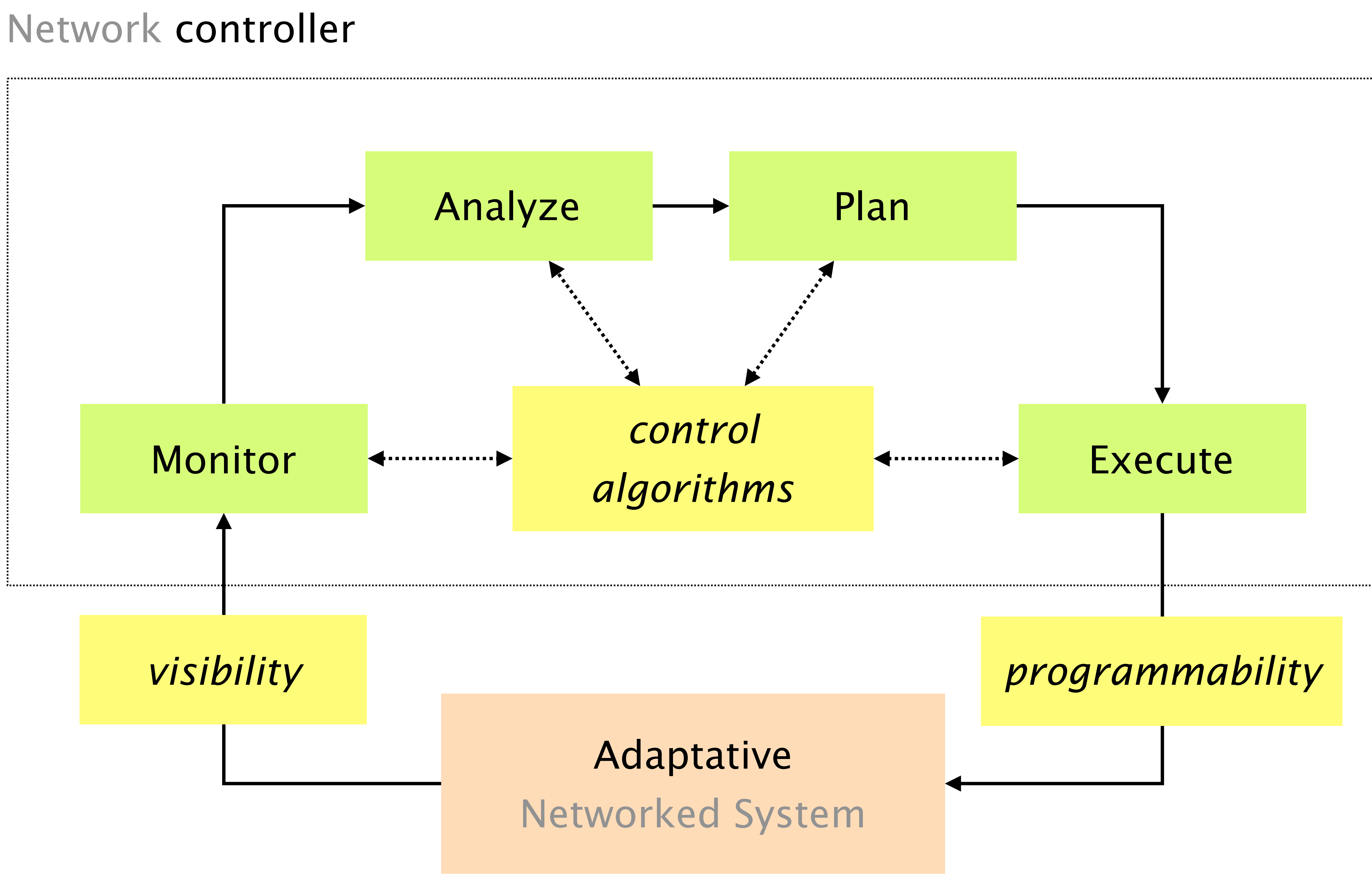
Juniper JunOS

```
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 {
```

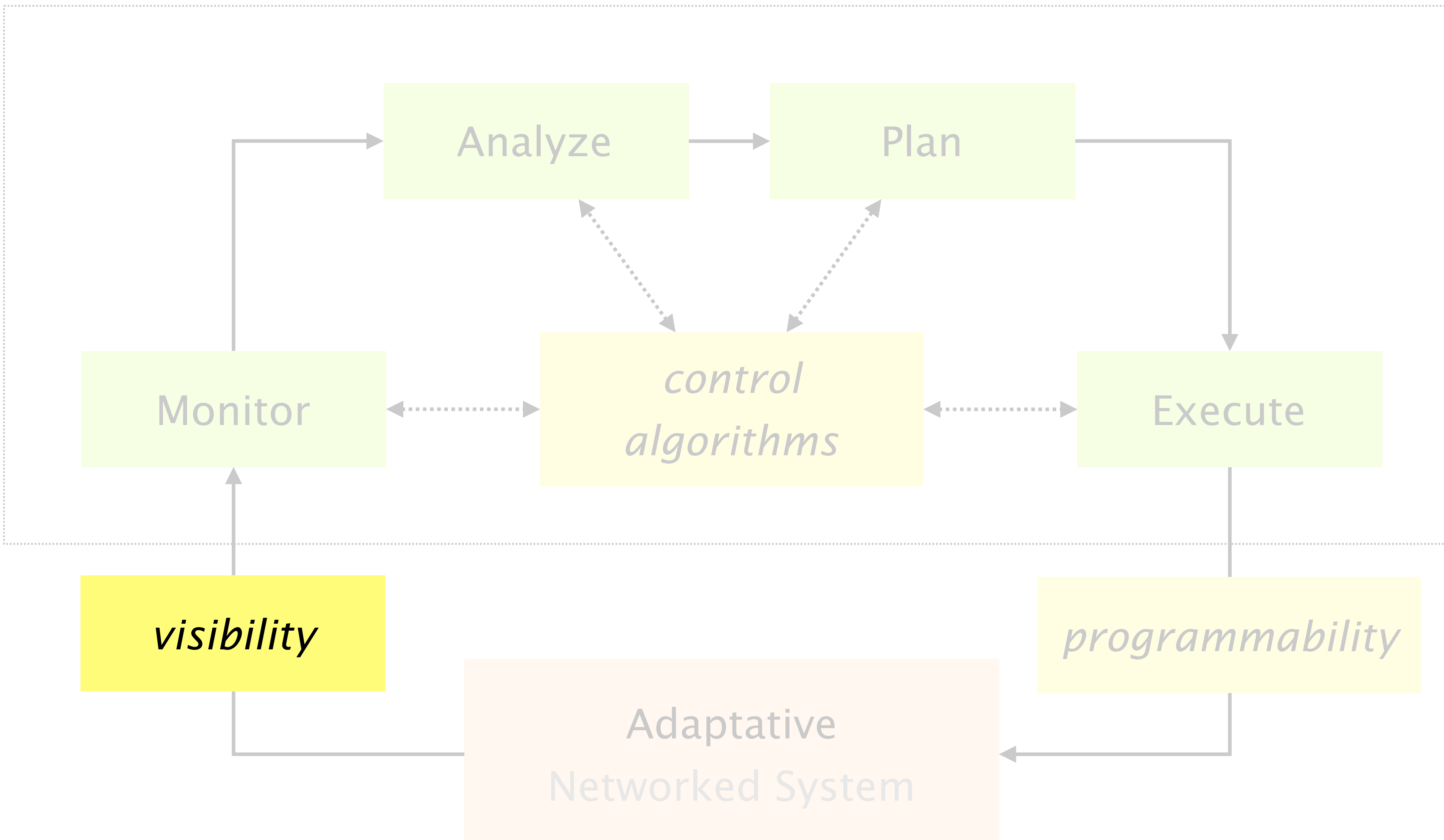
My research goal? **Automate!**

Remove the need to rely on humans

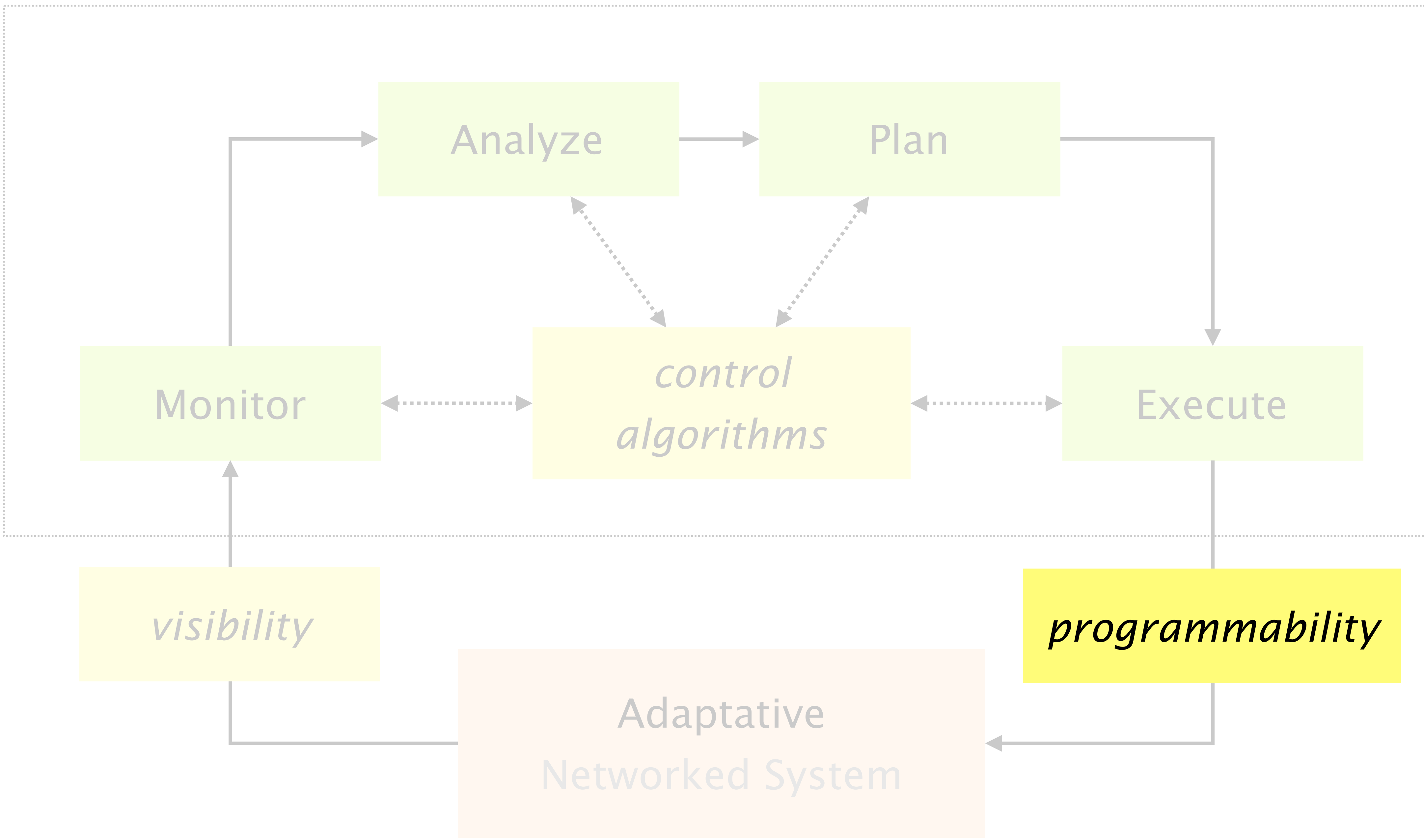
Develop a complete & sound network controller which can automatically enforces high-level requirements


Network controller
Analyze
Plan
Monitor
control algorithms
Execute
visibility
programmability
Adaptative
Networked System

Develop efficient and fine-grained measurement techniques, *i.e.* sensors

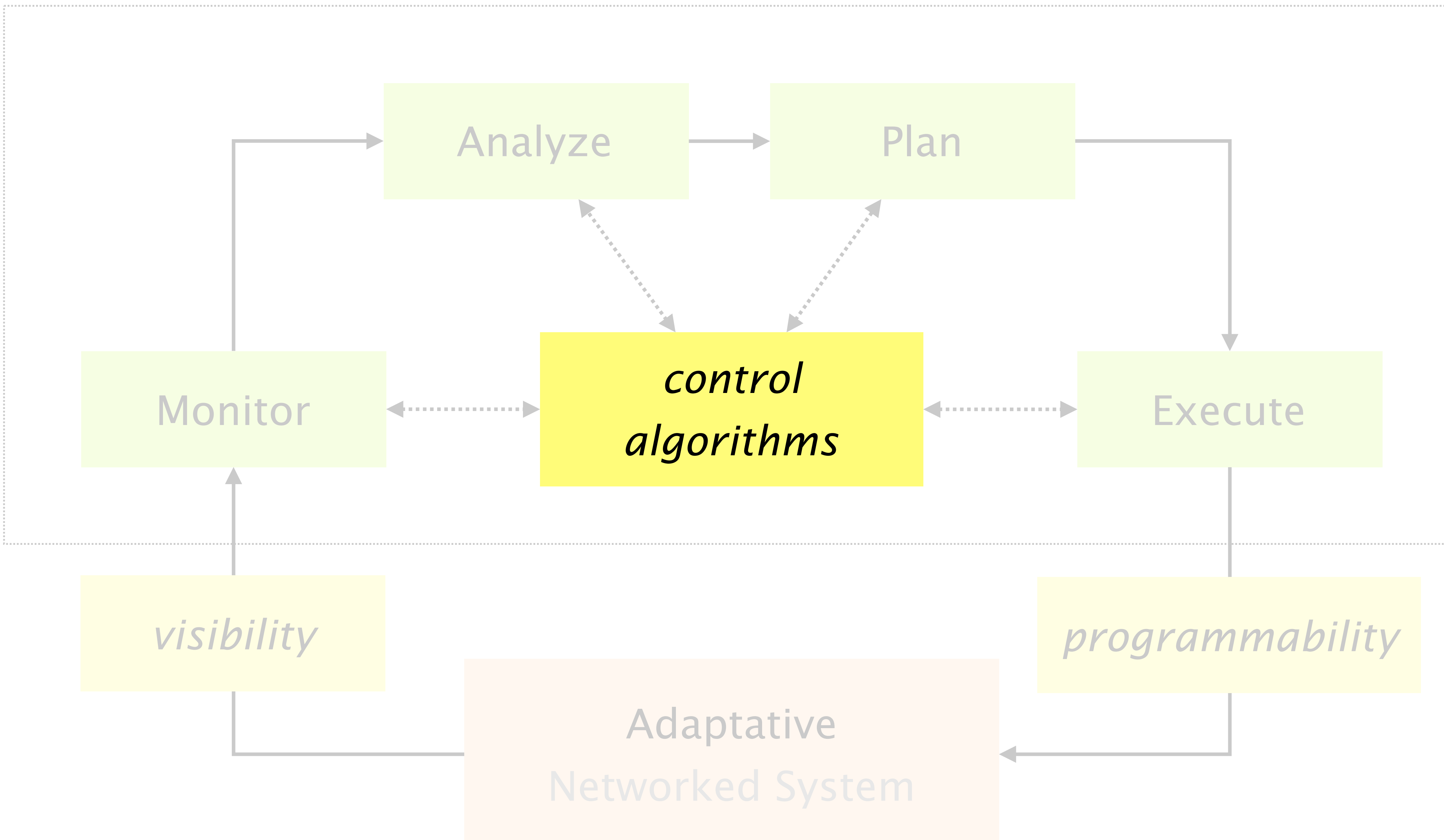


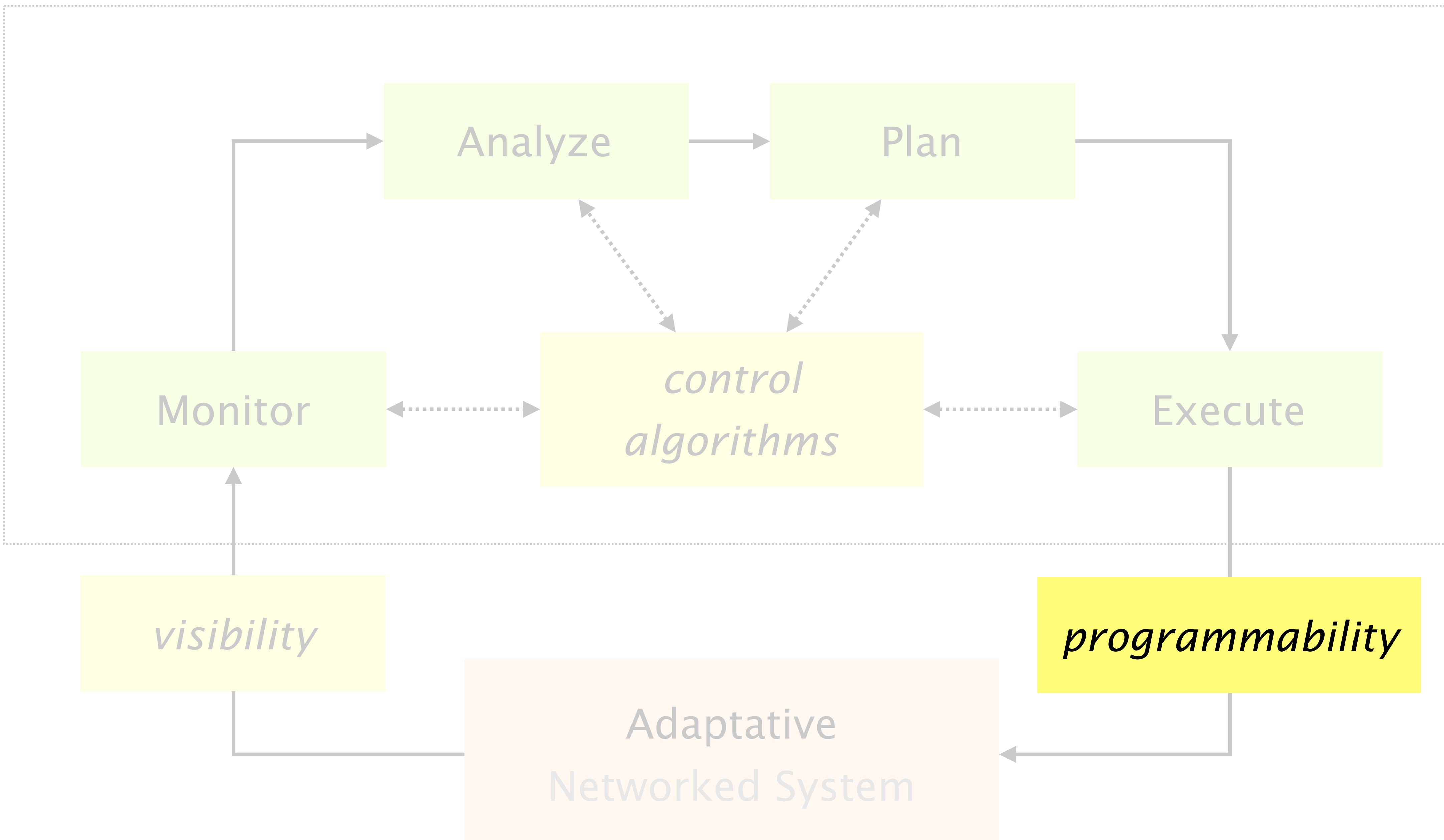
Develop fine-grained declarative control interfaces with a clear semantic, *i.e.* **actuators**



Develop efficient control algorithms

leveraging this new generation of sensors/actuators



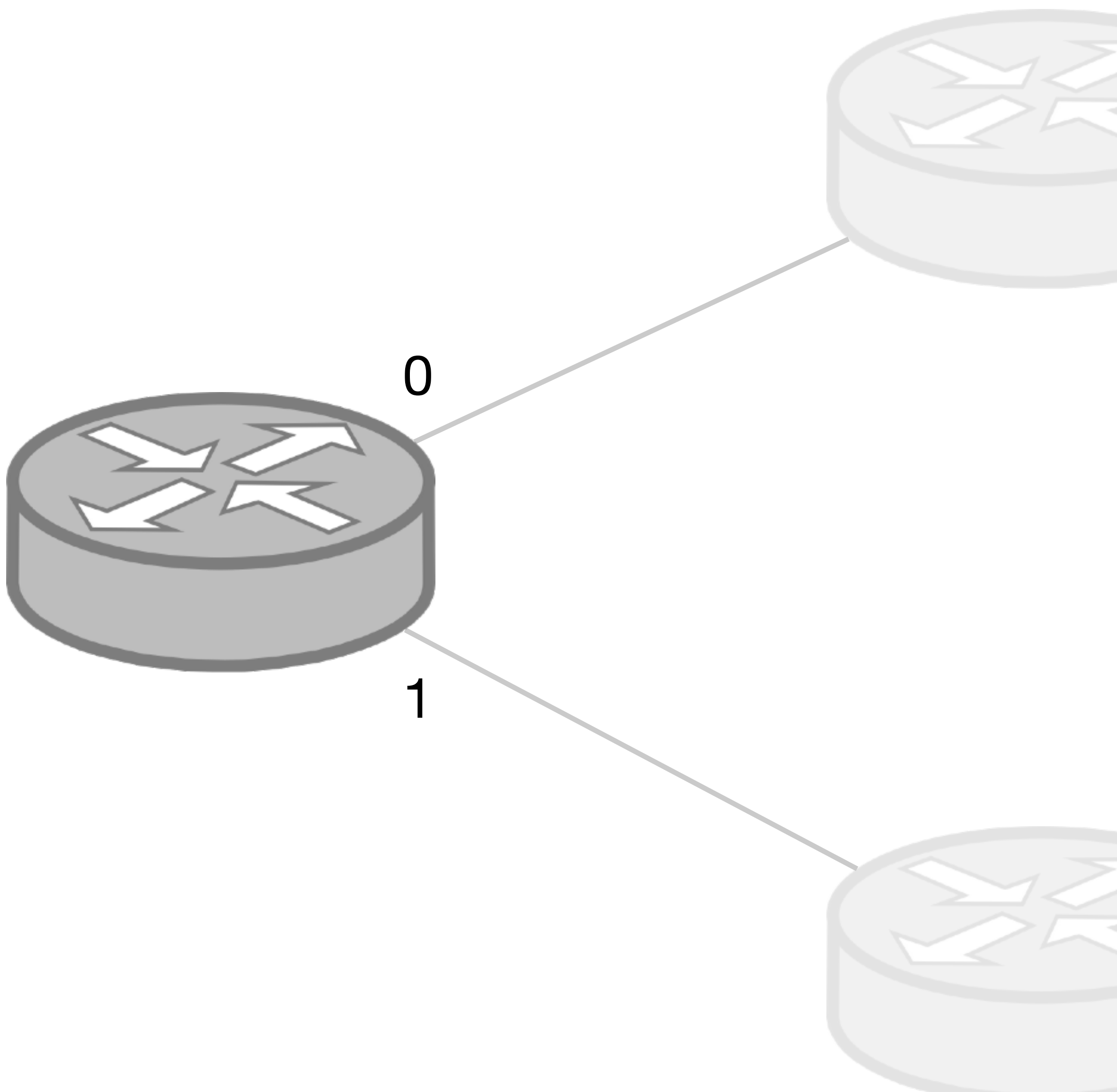

Analyze
Plan
Monitor
control algorithms
Execute
visibility
programmability
Adaptative Networked System

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

The **forwarding state** computed by a router depends on **two inputs**

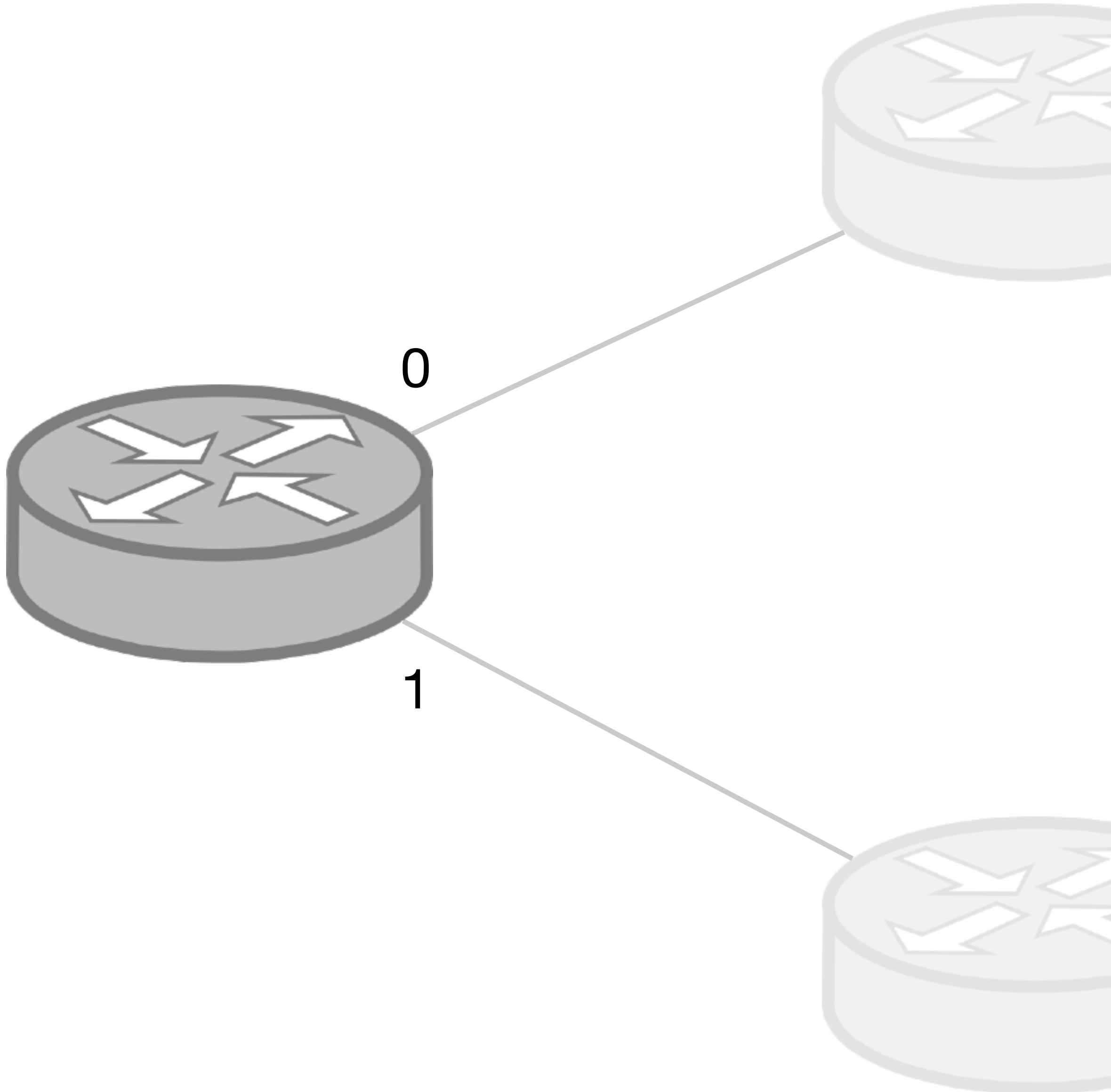
Forwarding state

	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

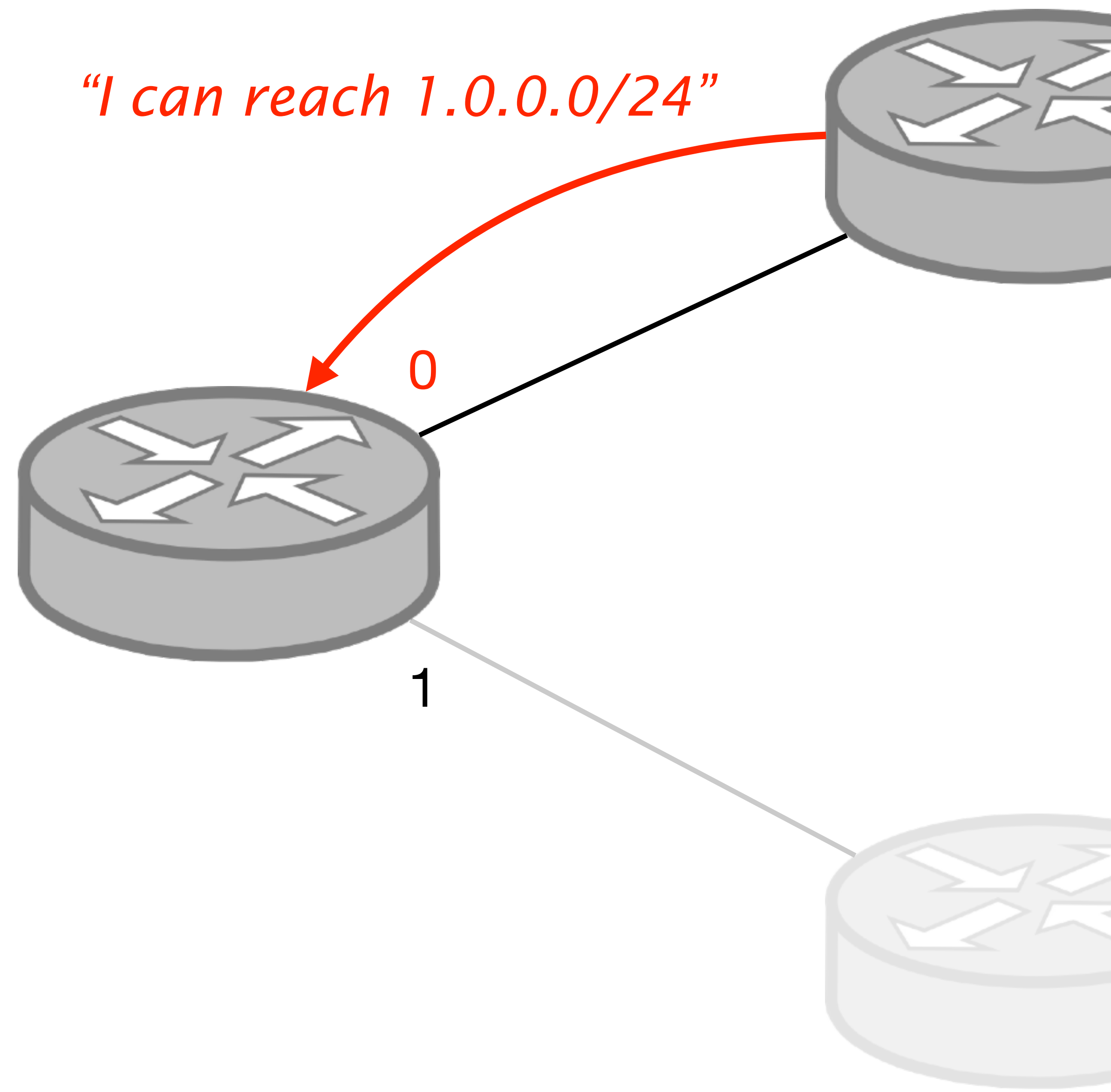
```
!
ip multicast-routing
!
interface Loopback0
 ip address 120.1.7.7 255.255.255.255
 ip ospf 1 area 0
!
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.17
 encapsulation dot1Q 17
 ip address 125.1.17.7 255.255.255.0
 ip pim bsr-border
 ip pim sparse-mode
!
!
router ospf 1
 router-id 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
  network 125.1.79.0 mask 255.255.255.0
```



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 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

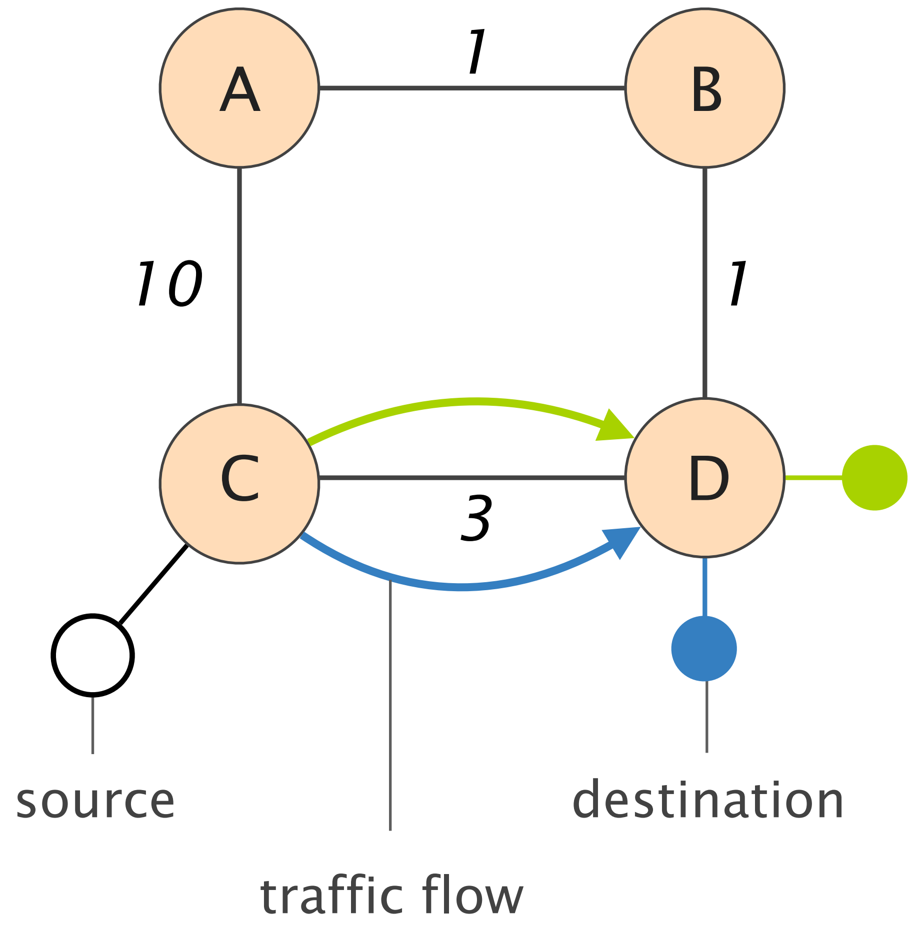
"the inputs"

SyNET

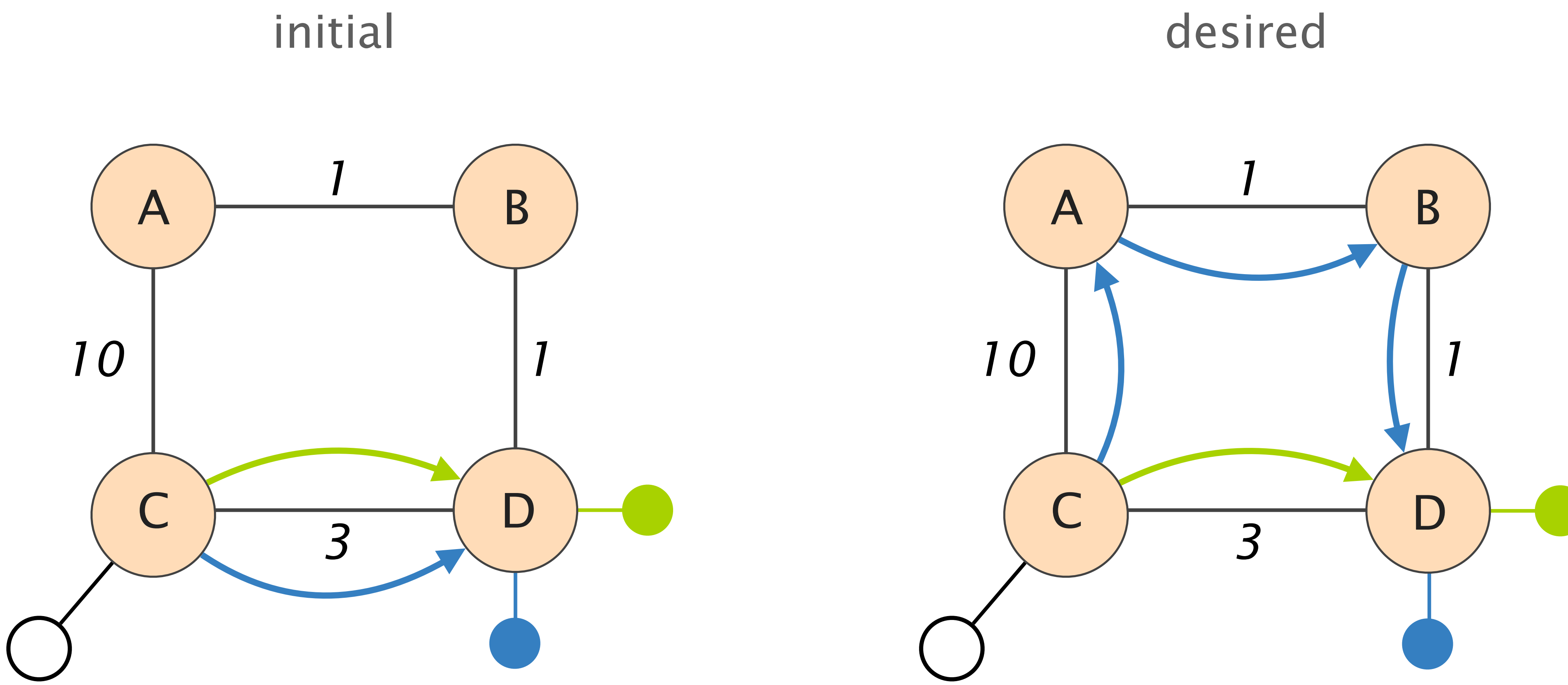
"the functions"

[SIGCOMM'15]

Consider this network where a source sends traffic to 2 destinations

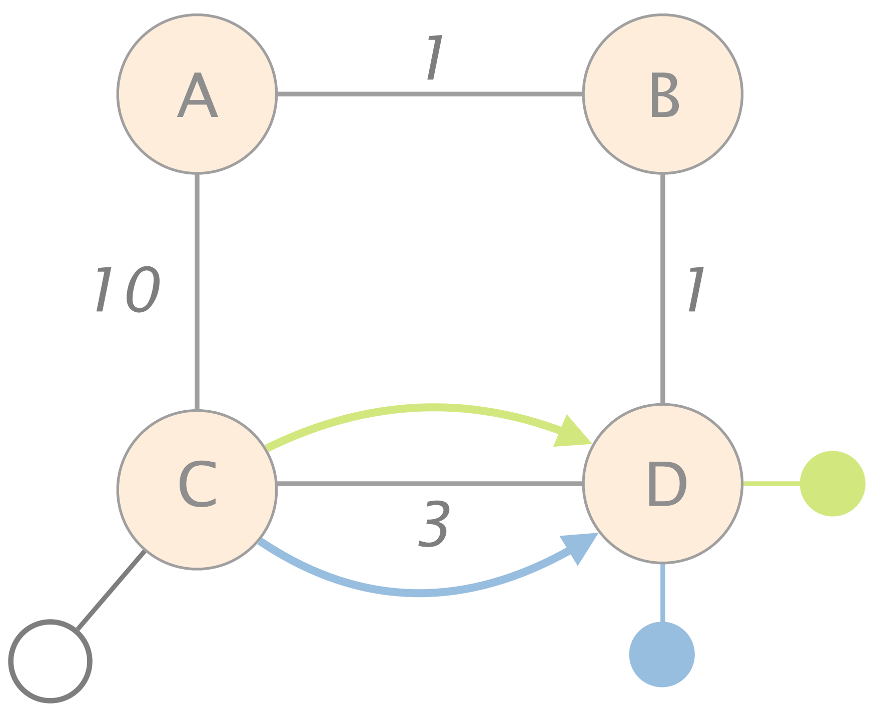


As congestion appears, the operator wants to shift away one flow from (C,D)

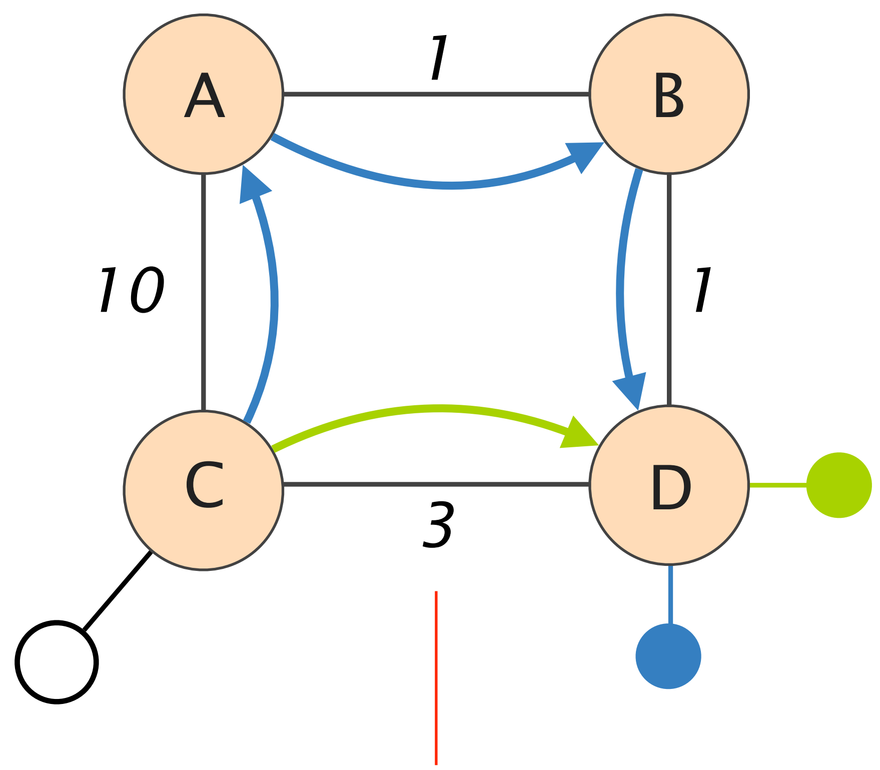


Moving only one flow is impossible though as both destinations are connected to D

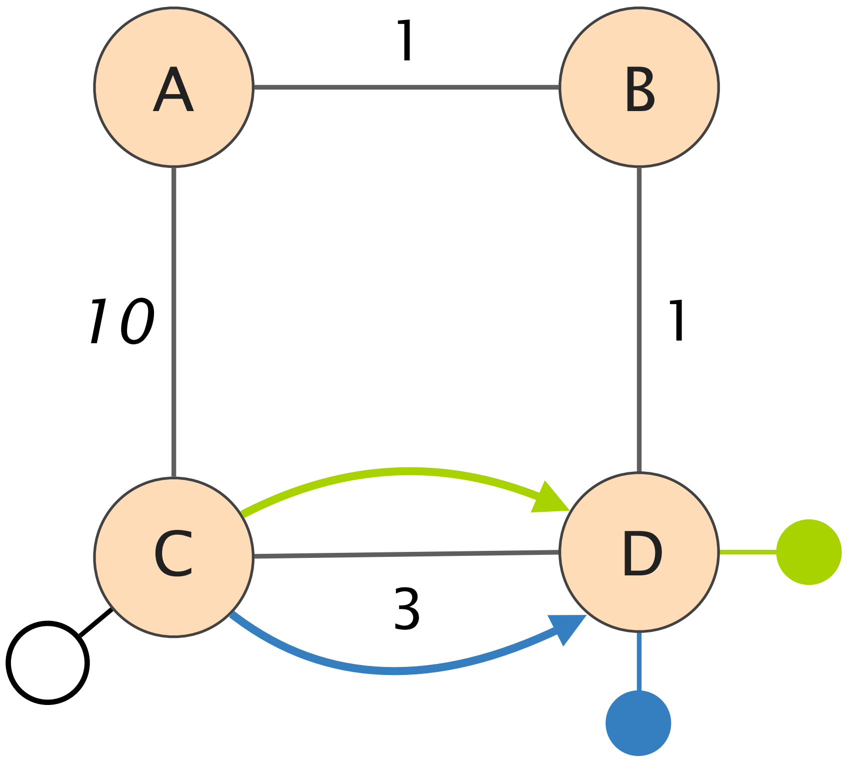
initial



desired

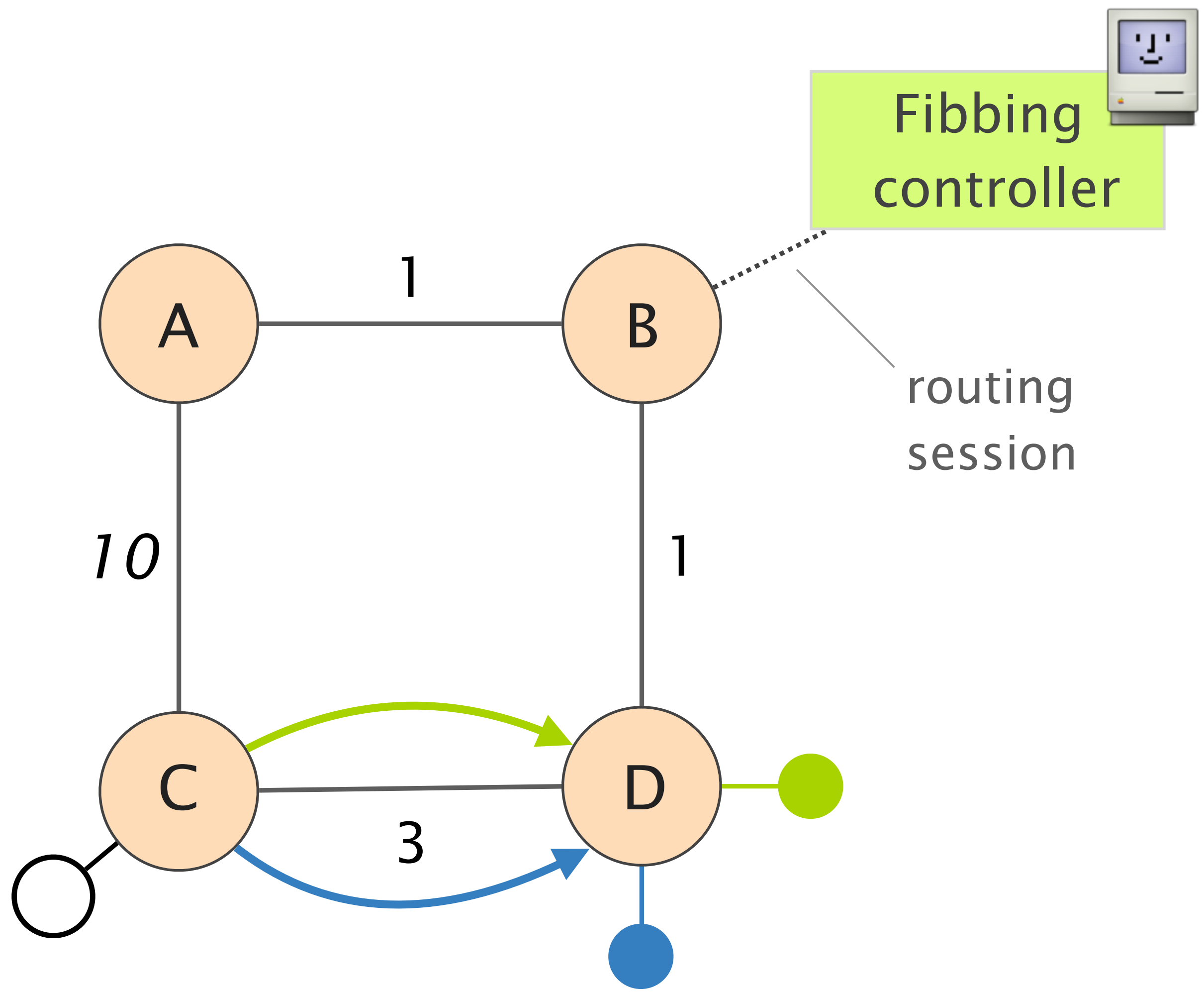


impossible to achieve by reweighing the links

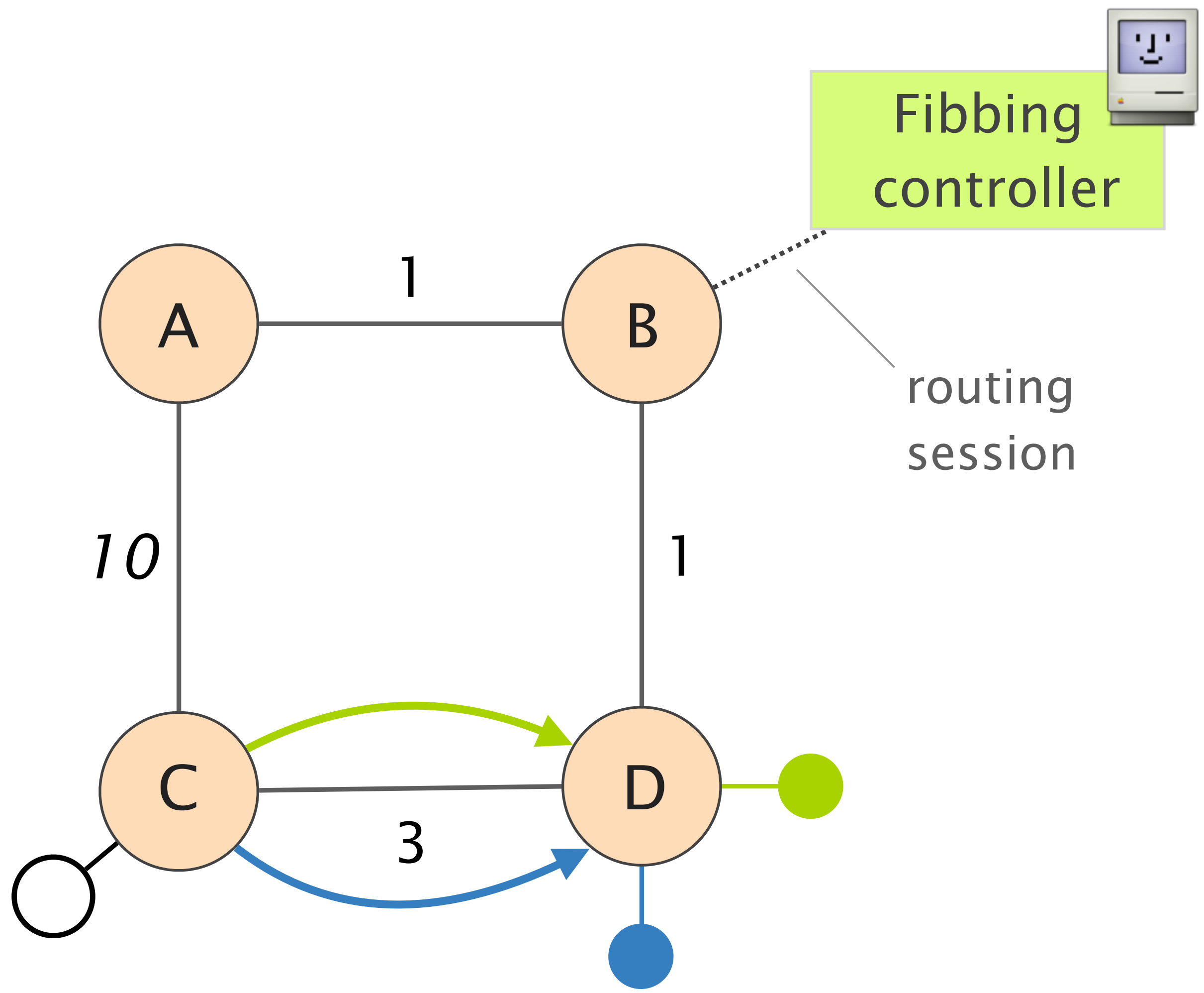


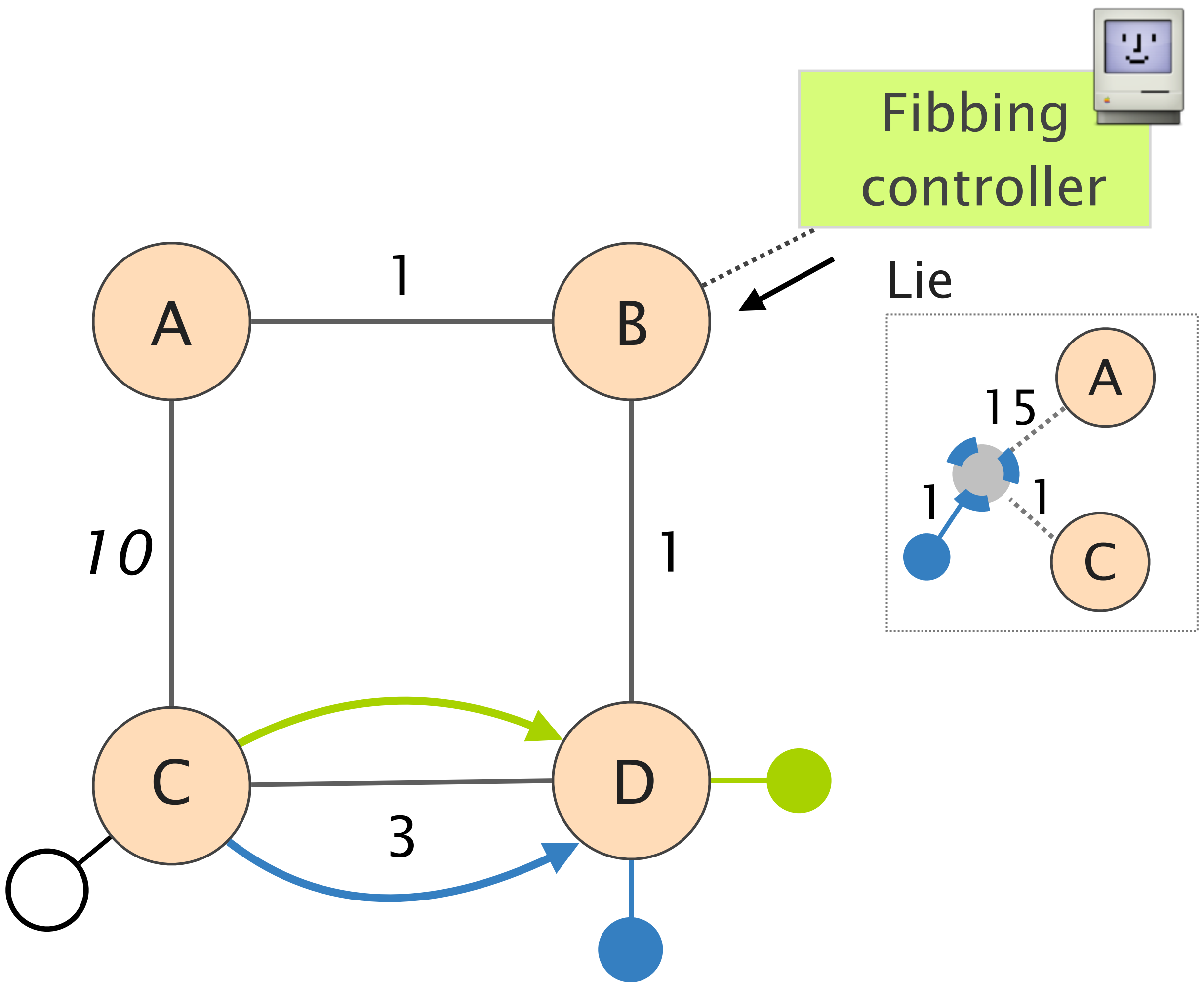
A
B
C
D
1
10
1
3

Let’s **lie** to the routers

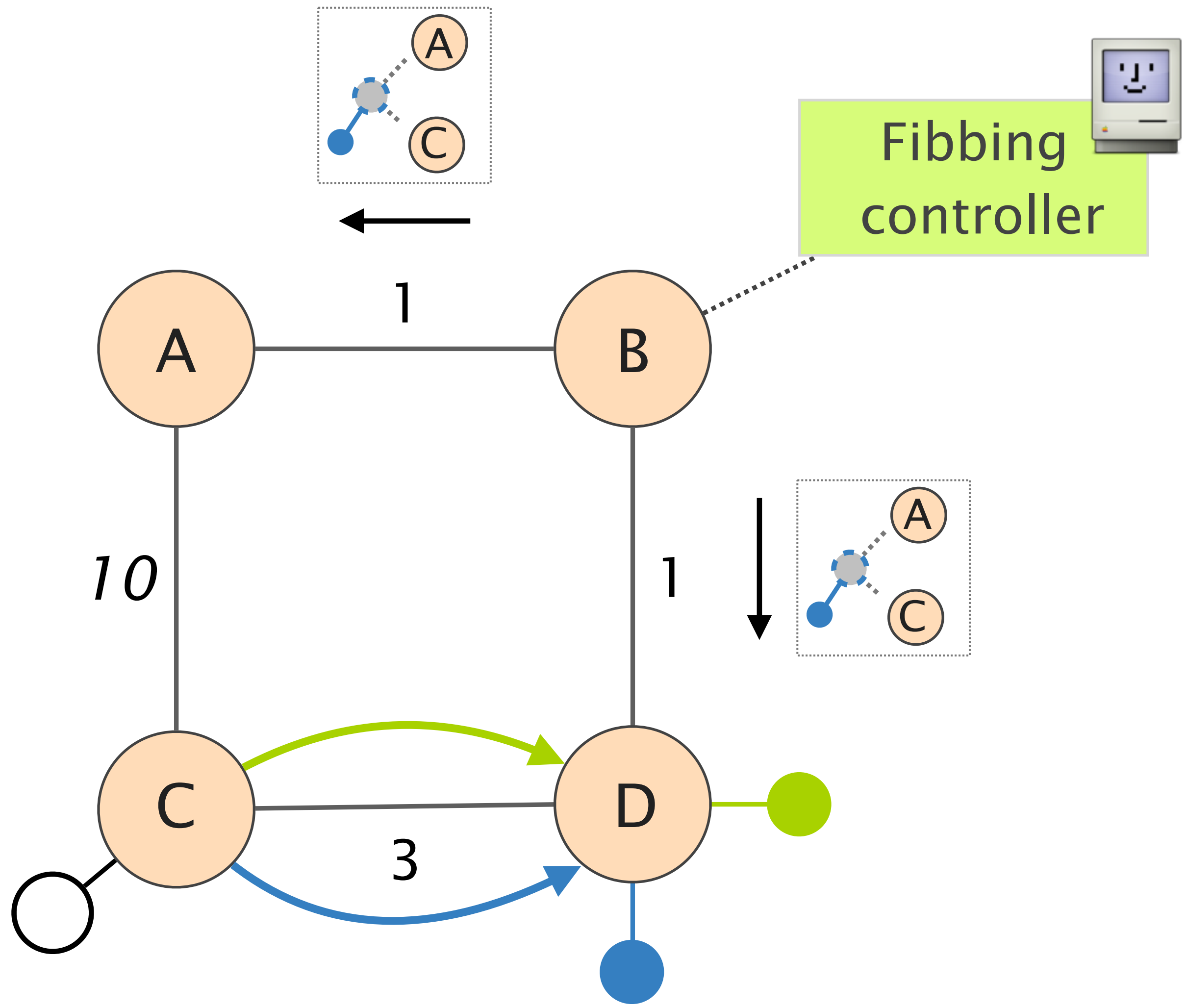


Let's **lie** to the routers, by **injecting fake nodes, links and destinations**

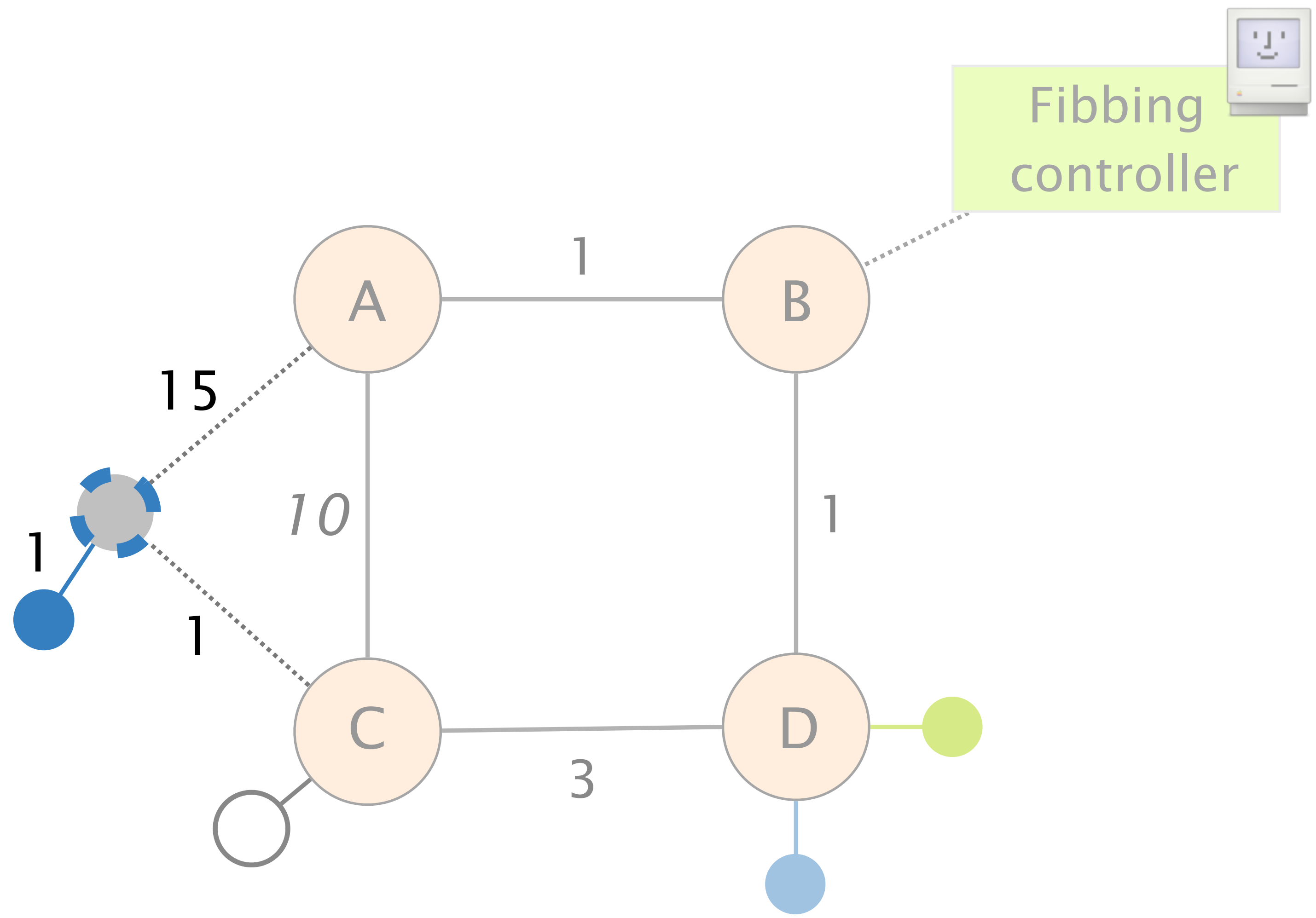



Fibbing
controller
Lie
A
B
C
D
1
10
1
3
15
1
1
A
C

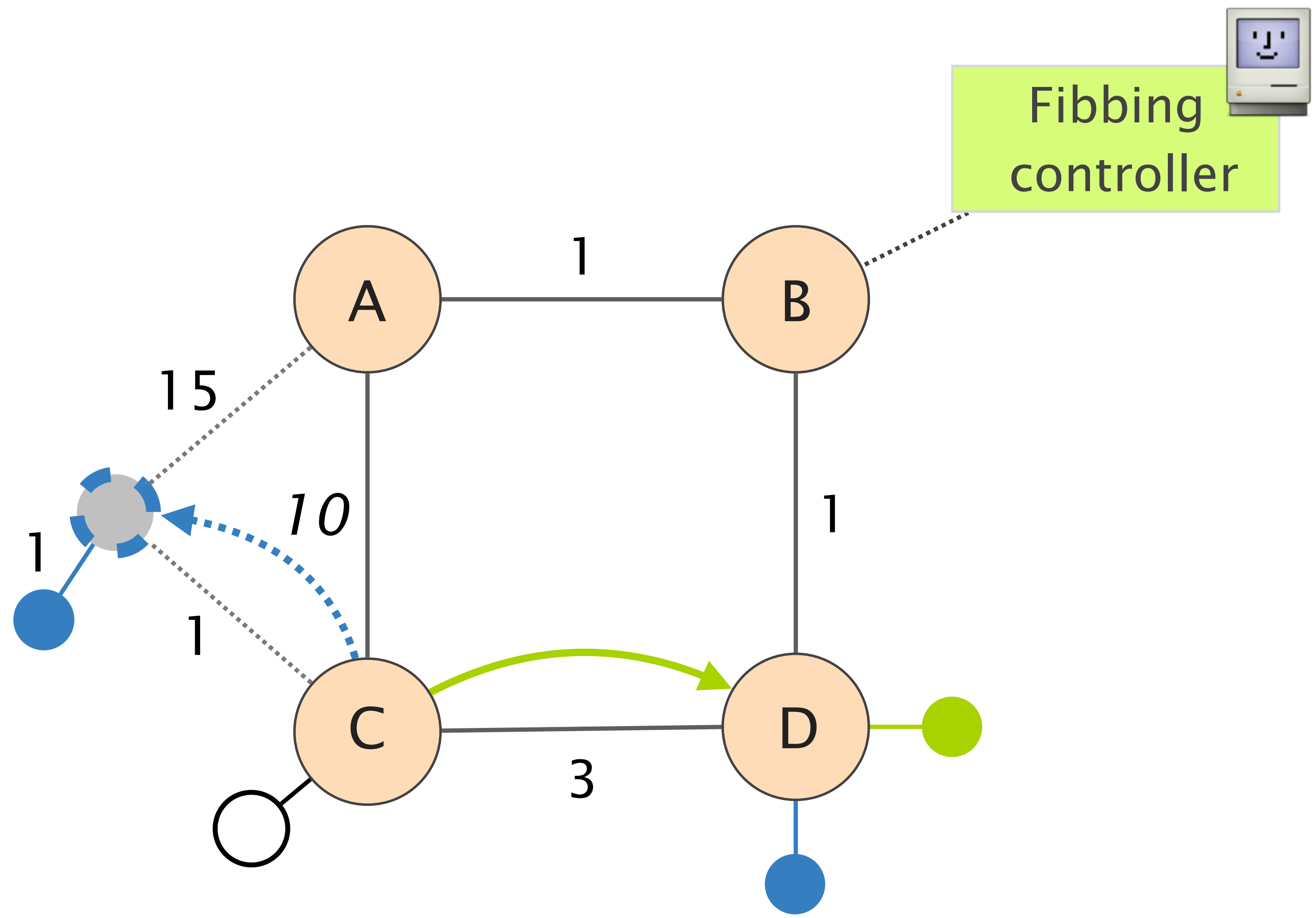
Lies are propagated network-wide
by the routing protocol



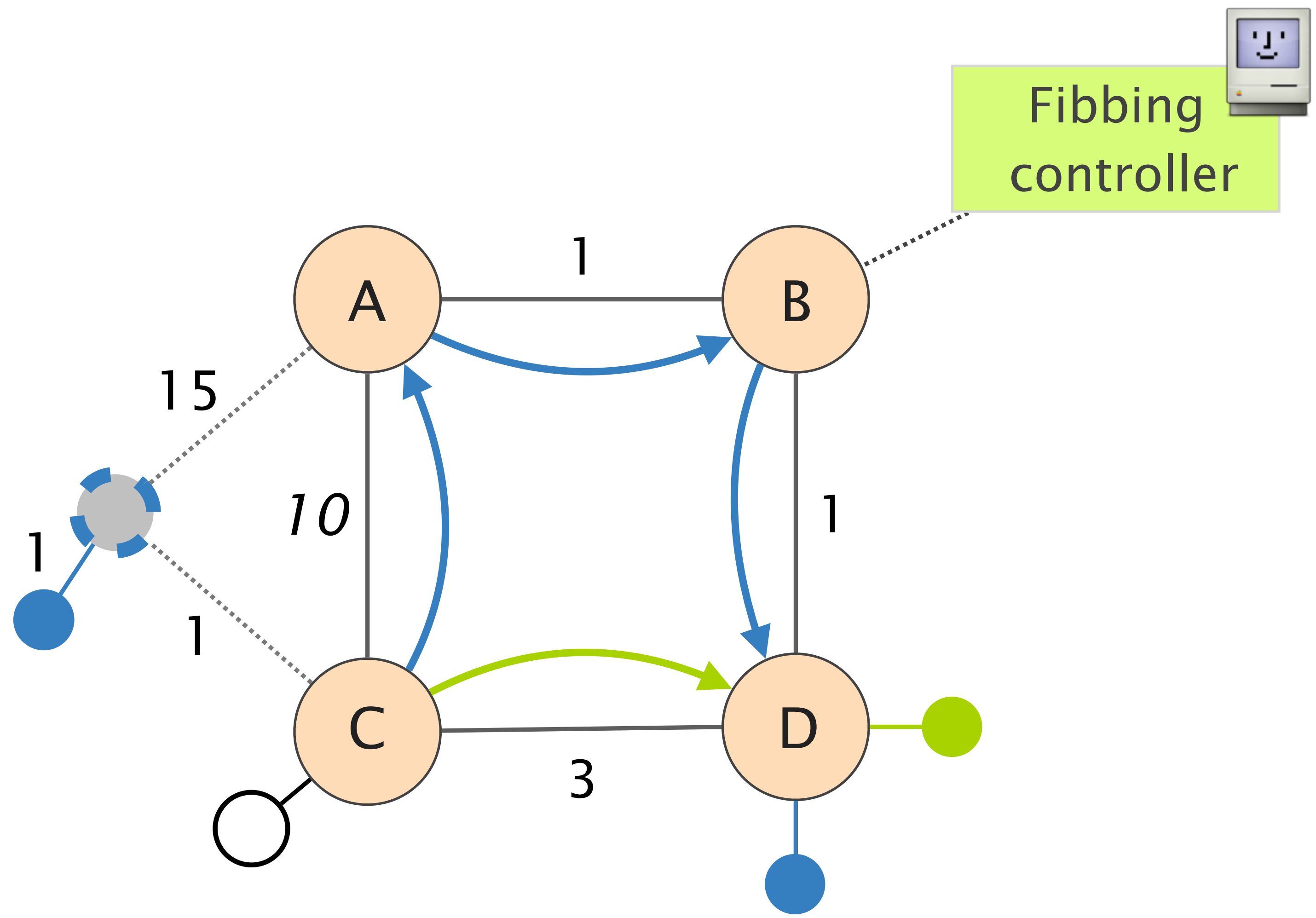
All routers compute their shortest-paths on the augmented topology



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**



Synthesizing routing messages is powerful

Theorem

Fibbing can program any set of non-contradictory paths

Theorem

Fibbing can program any set of non-contradictory paths

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)

Synthesizing routing messages is fast and works in practice

We developed efficient algorithms

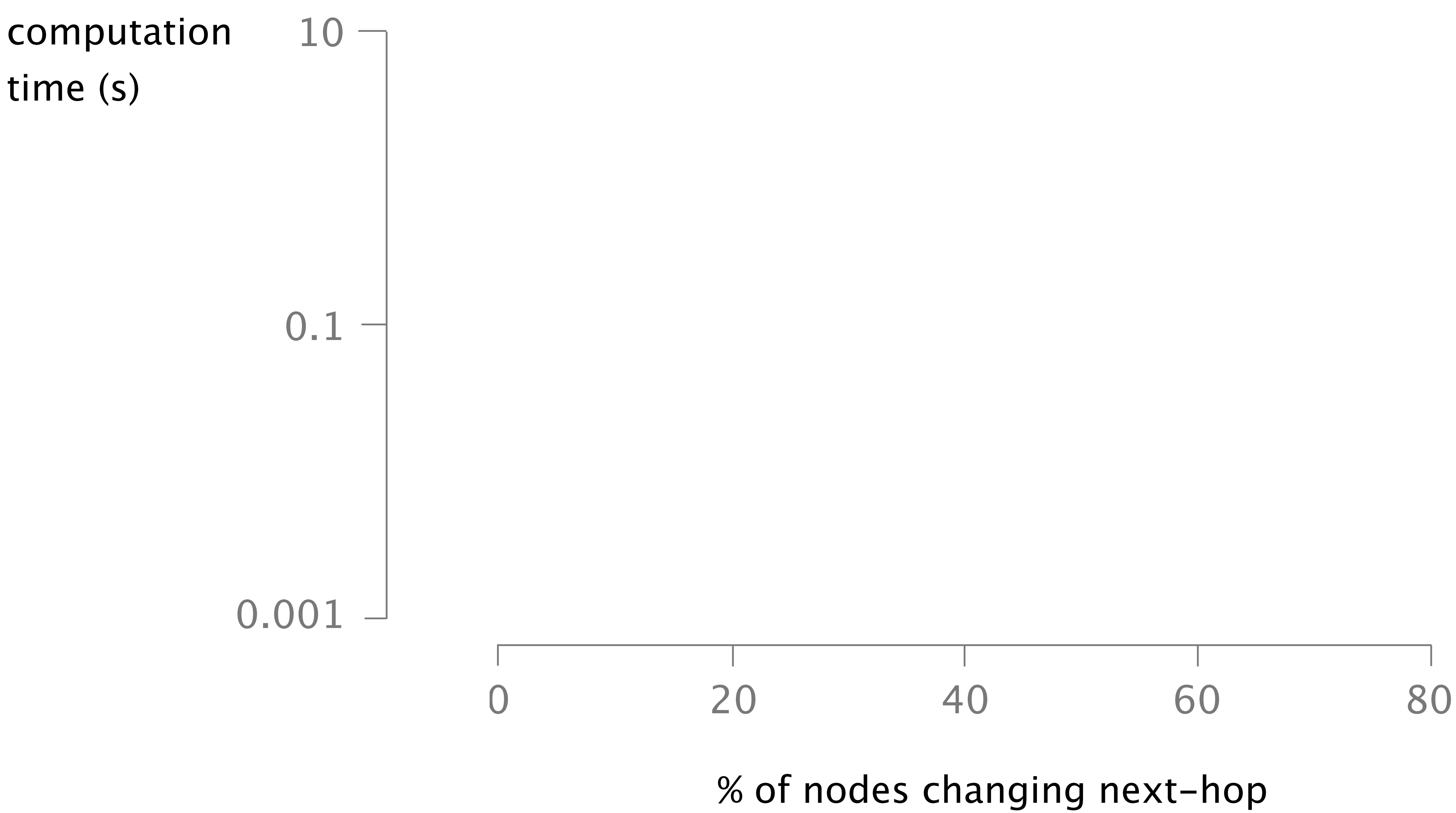
polynomial in the # of requirements

Compute and minimize topologies in ms

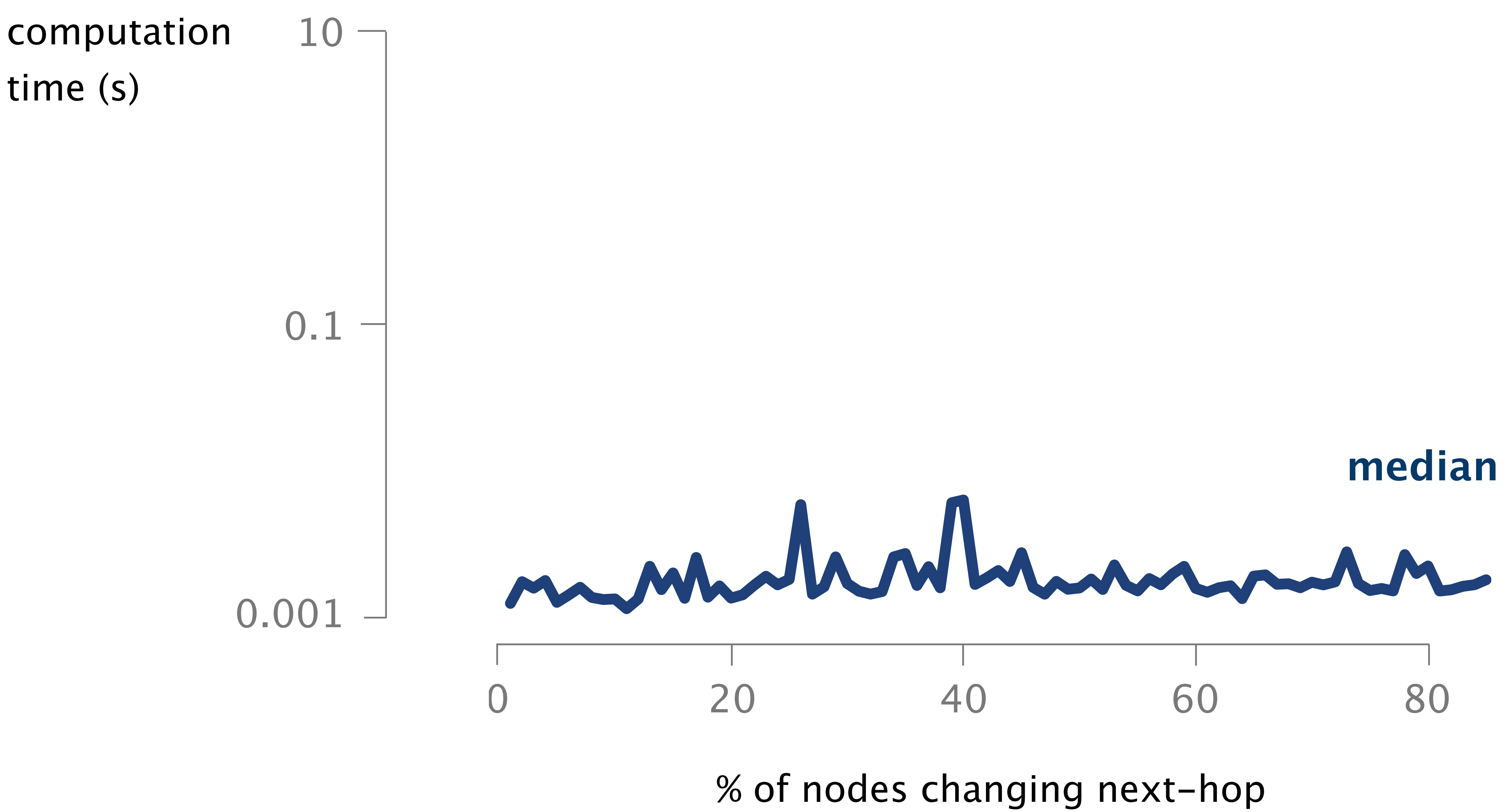
independently of the size of the network

We tested them against real routers

works on both Cisco and Juniper


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

Fibbing computes routing messages to inject in ~1ms



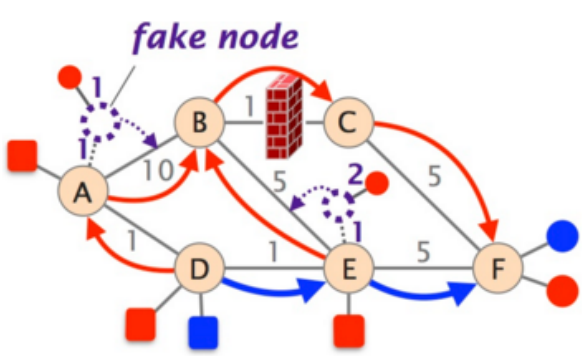
Check out our webpage

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

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

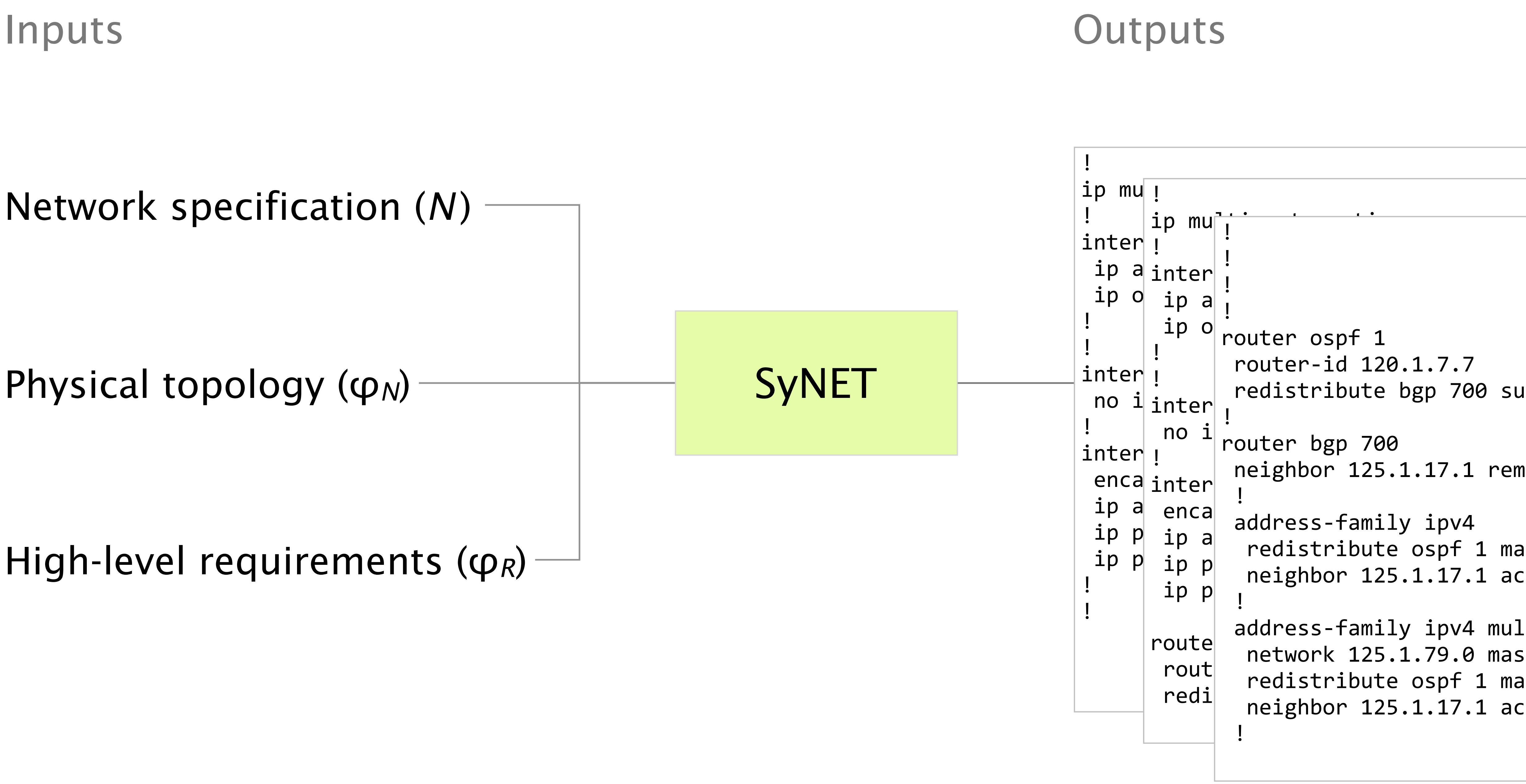
Dijkstra-based shortest-path routing

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 (φN)
High-level requirements (φR)
SyNET
router ospf 1
router-id 120.1.7.7
router bgp 700
address-family ipv4
address-family ipv4 mul

SyNET can generate configurations for (small) networks

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

SyNET can generate configurations for (small) networks

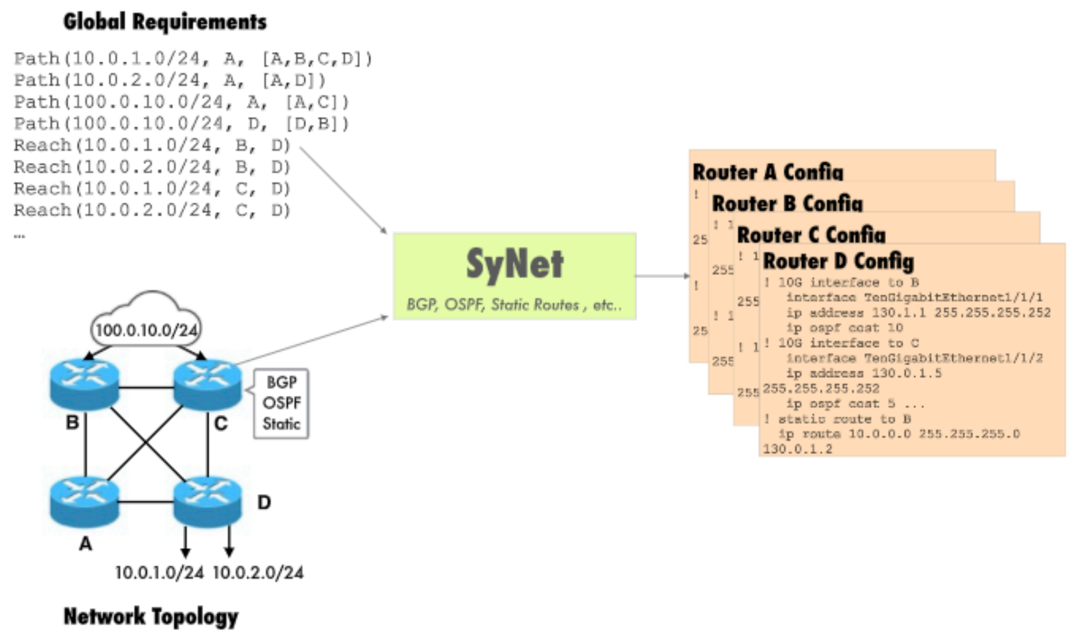
		# routers		
		4	9	16
	static	1.8s	18.2s	116.1s
# protocols	static, OSPF	4.2s	37.0s	197.0s
	static, OSPF, BGP	13.8s	189.4s	577.4s

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

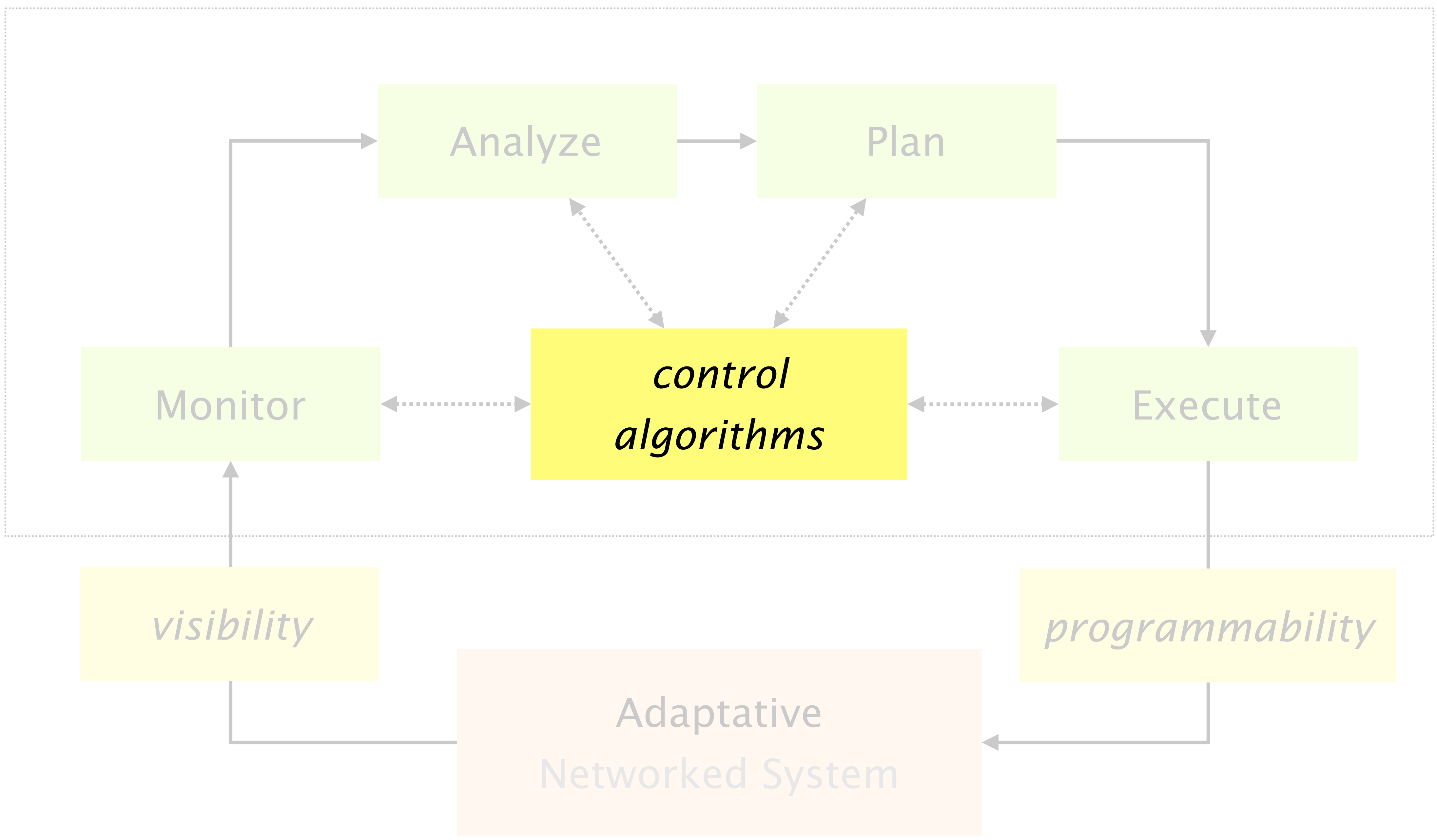
Fibbing

“the inputs”

SyNET

“the functions”

Now that we’ve programmability,
What can we do with it?



Analyze
Plan
Monitor
control algorithms
Execute
visibility
programmability
Adaptative
Networked System

SWIFT

Predictive Fast Reroute upon Remote BGP Disruptions

Laurent Vanbever

ETH Zürich (D-ITET)

Munich Internet Research Retreat

November 25 2016

25.9 seconds

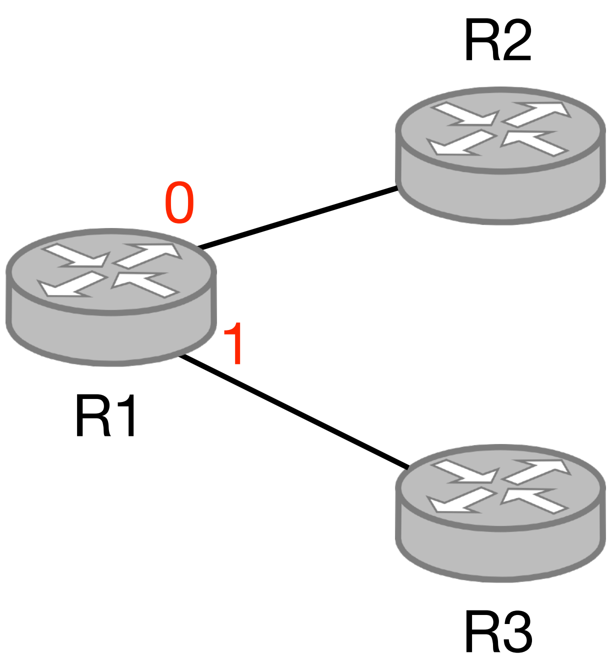
25.9 seconds

max. monthly downtime

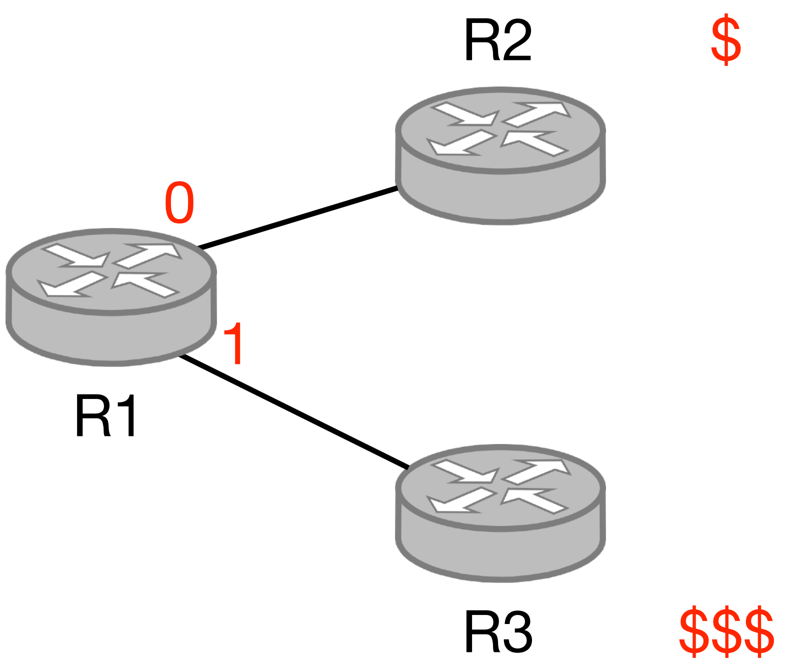
under a 99.999% SLA

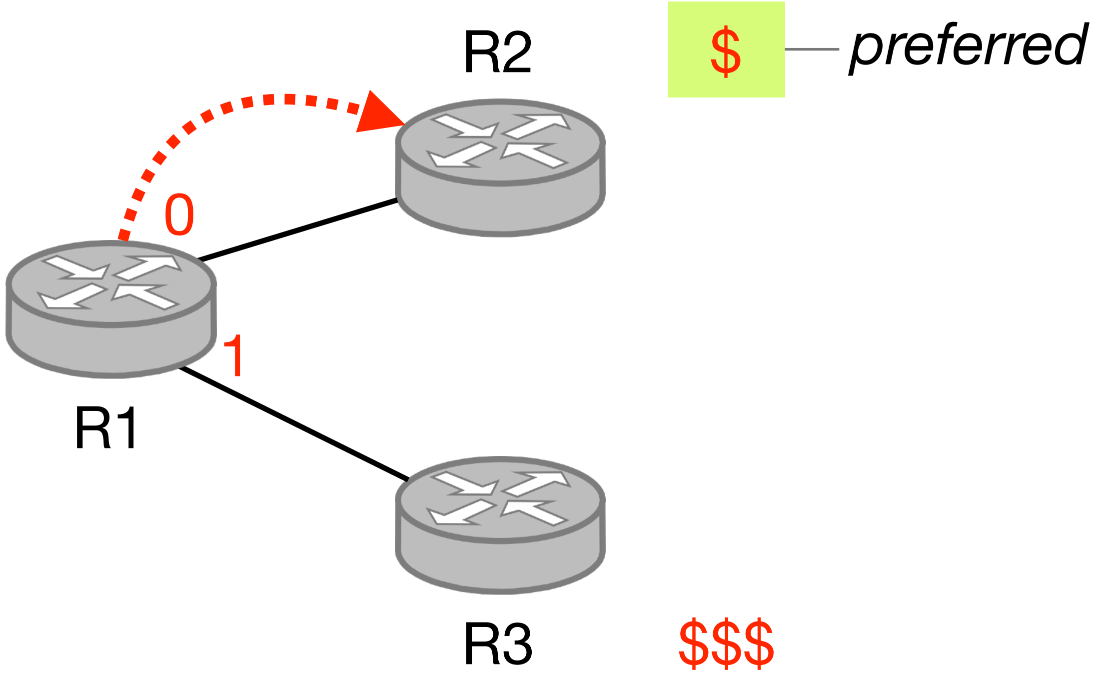
IP routers are slow to converge upon remote link and node failures

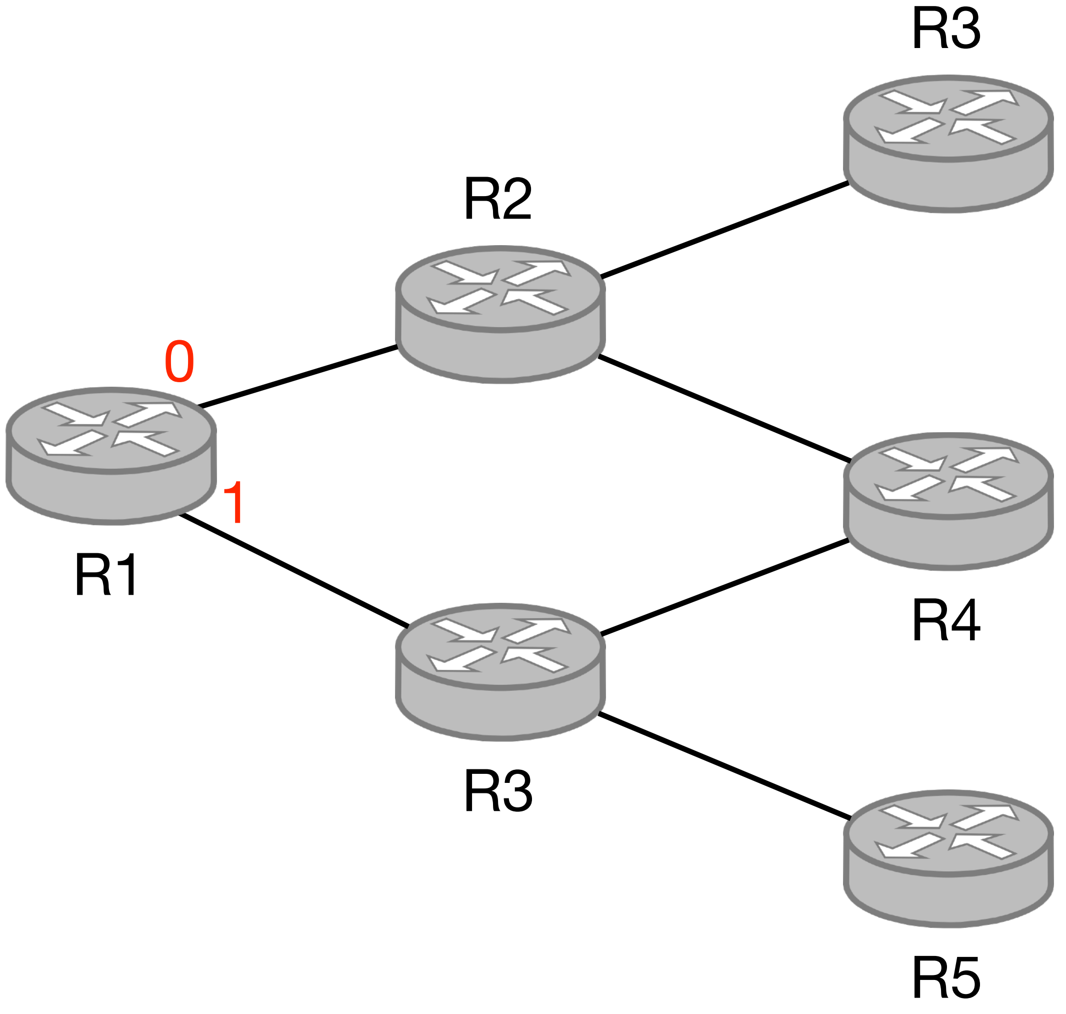
R1

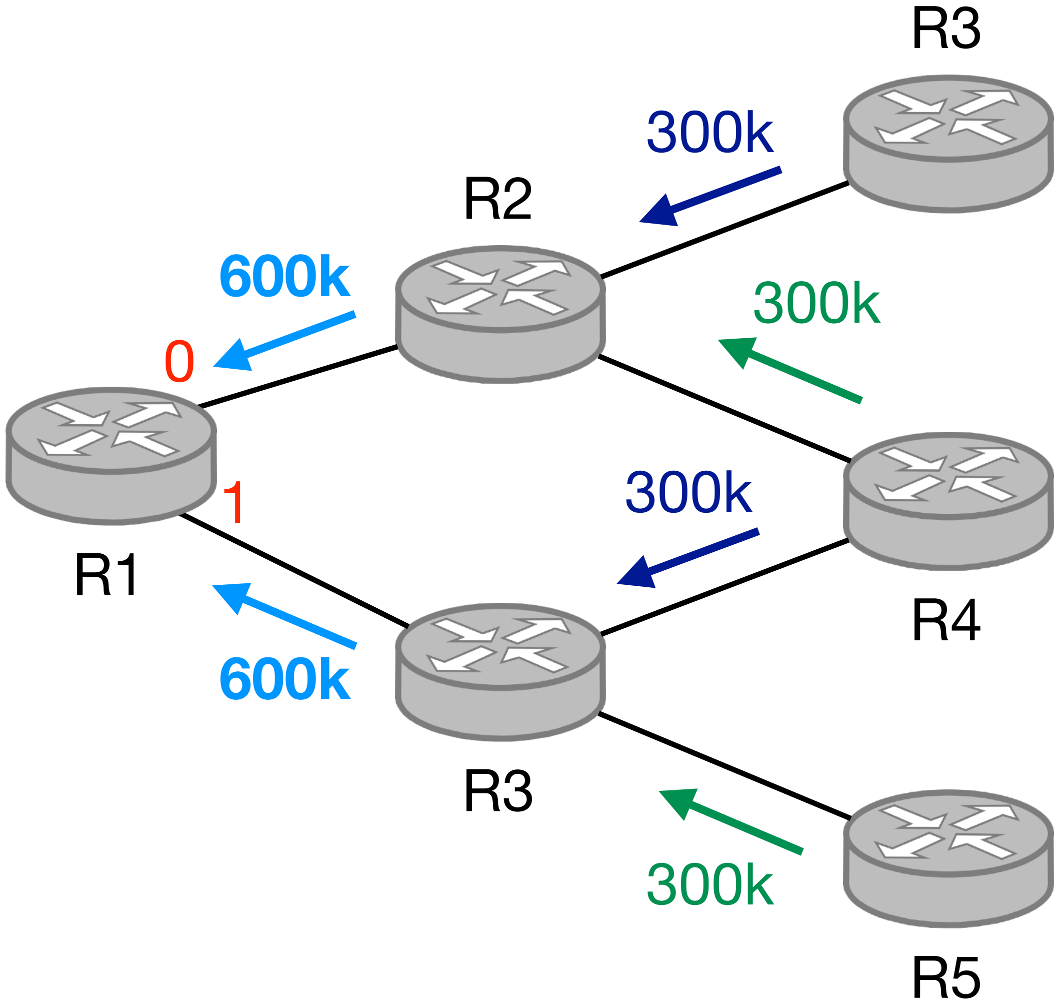

R2
0
1
R1
R3

R1 prefers to send traffic via R2 when possible, as it is much cheaper than via R3



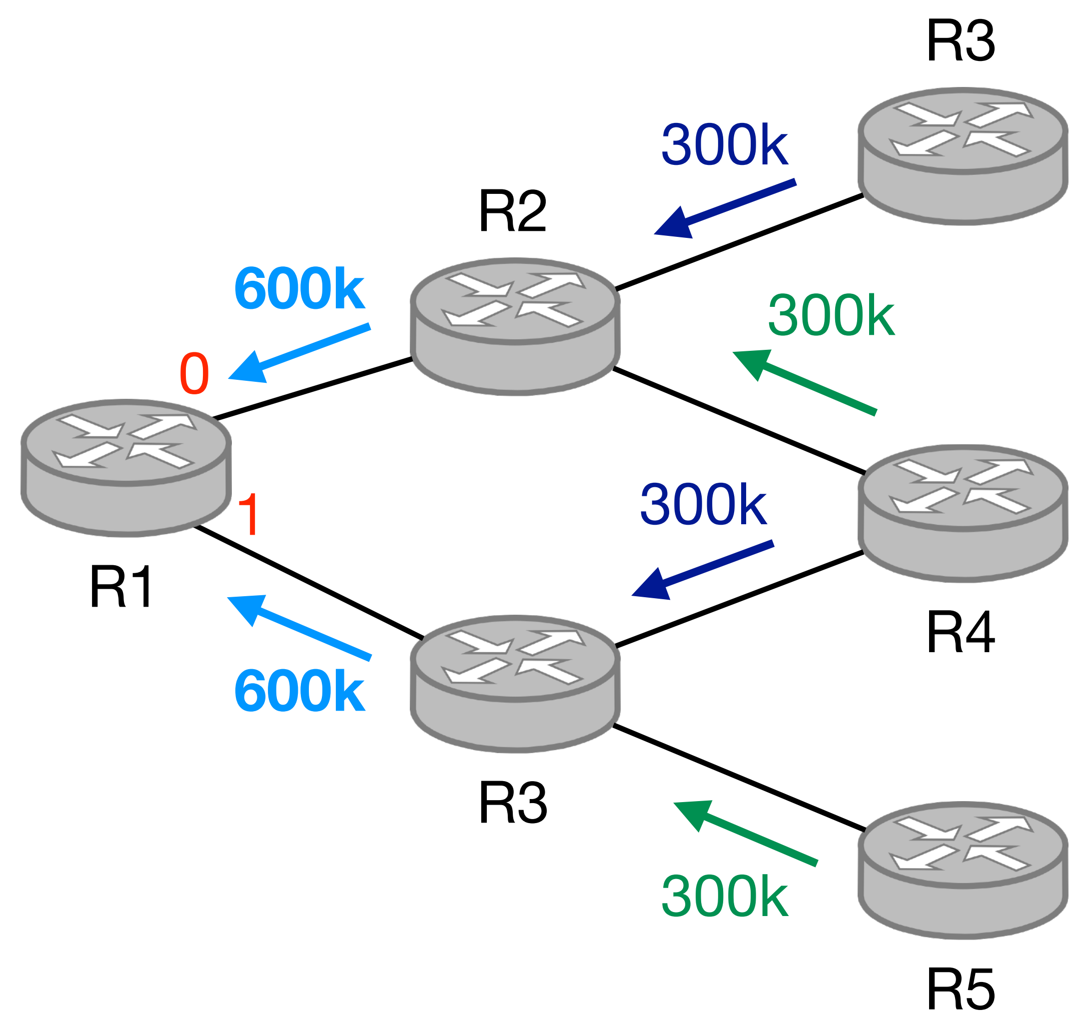

R2
$
preferred
0
1
R1
R3
$$$


R3
R2
0
1
R1
R4
R3
R5


R3
300k
R2
600k
300k
0
R1
1
300k
R4
600k
R3
300k
R5

R1's Forwarding Table

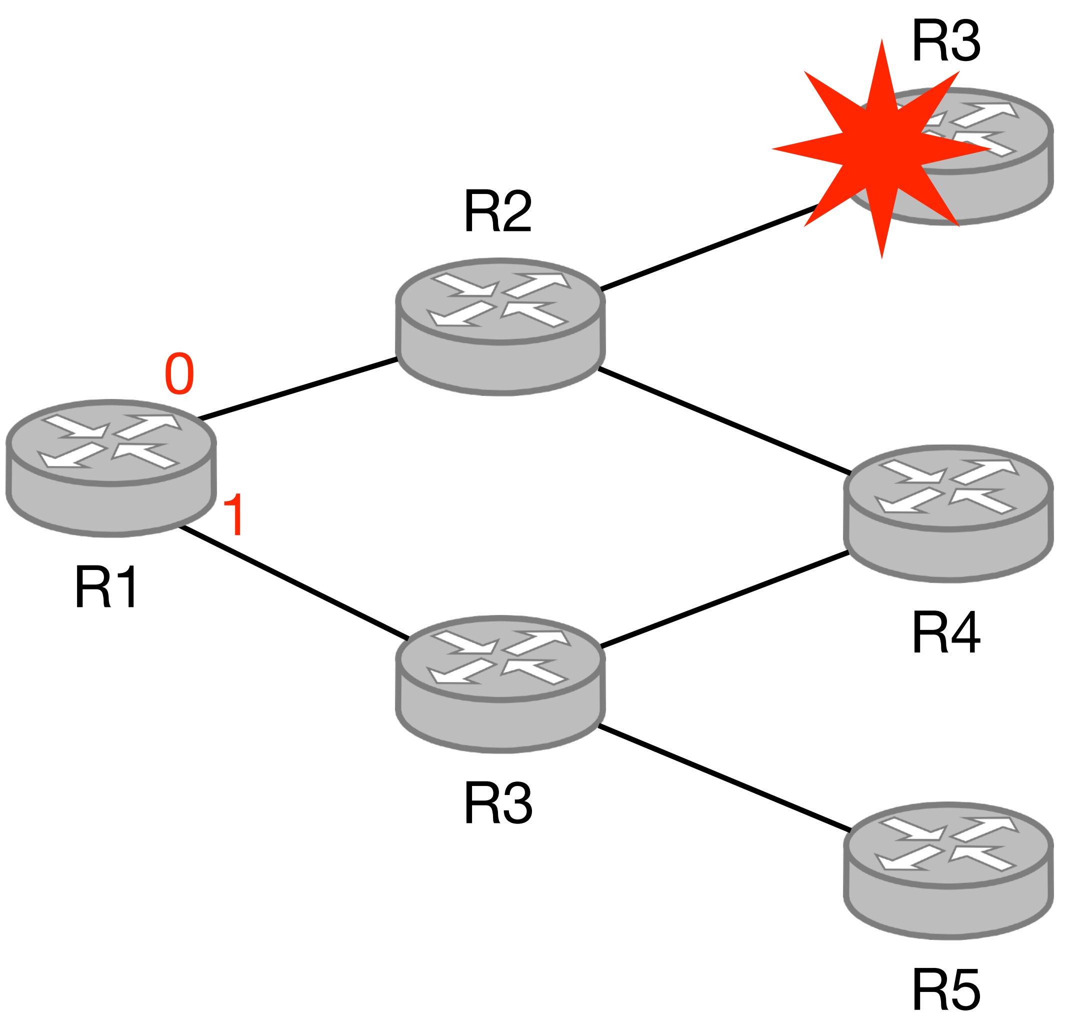
	prefix	Next-Hop
1	1.0.0.0/24	0
2	1.0.1.0/16	0
…	…	…
300k	100.0.0.0/8	0
…	…	…
600k	200.99.0.0/24	0



What if R3 fails?

R1's Forwarding Table

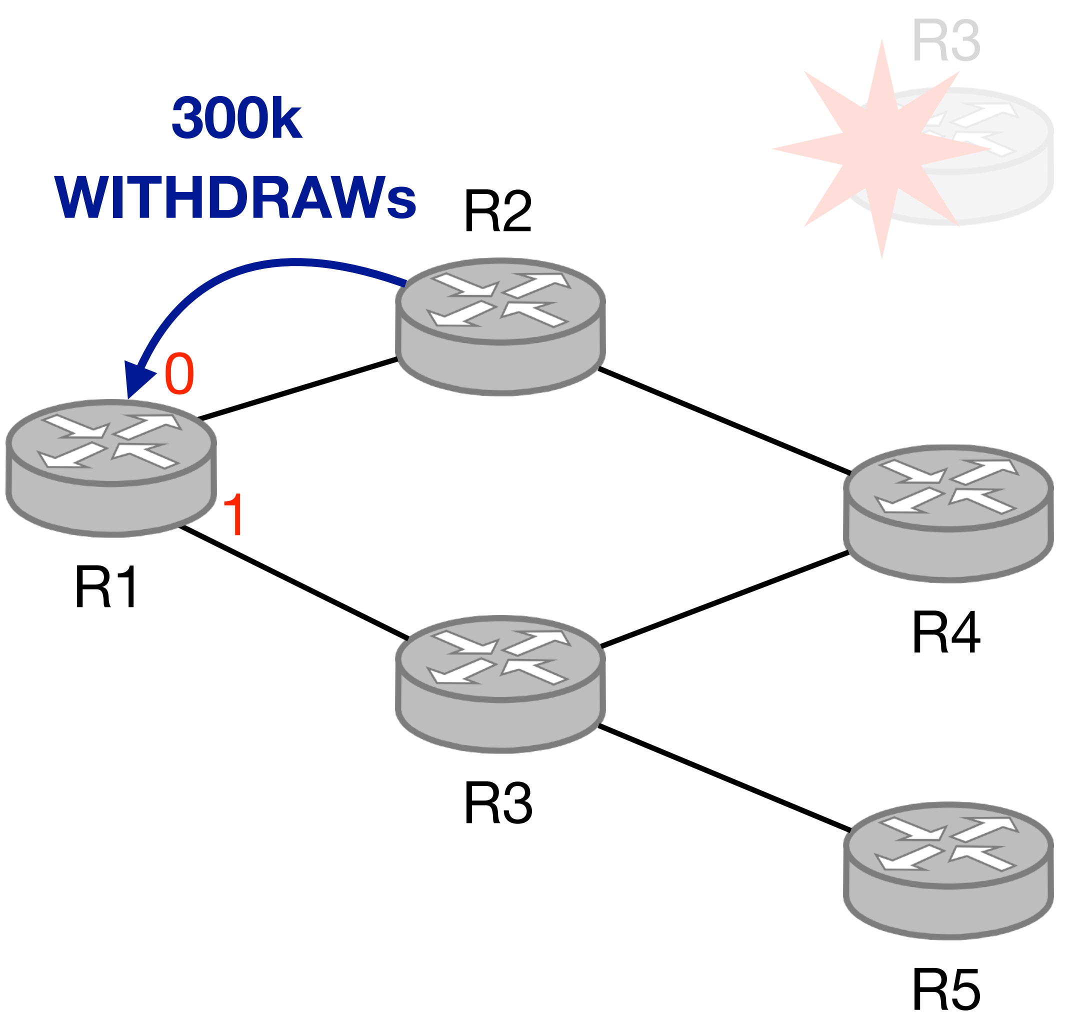
	prefix	Next-Hop
1	1.0.0.0/24	0
2	1.0.1.0/16	0
…	…	…
300k	100.0.0.0/8	0
…	…	…
600k	200.99.0.0/24	0



R2 sends 300k routing messages withdrawing the routes from R3

R1's Forwarding Table

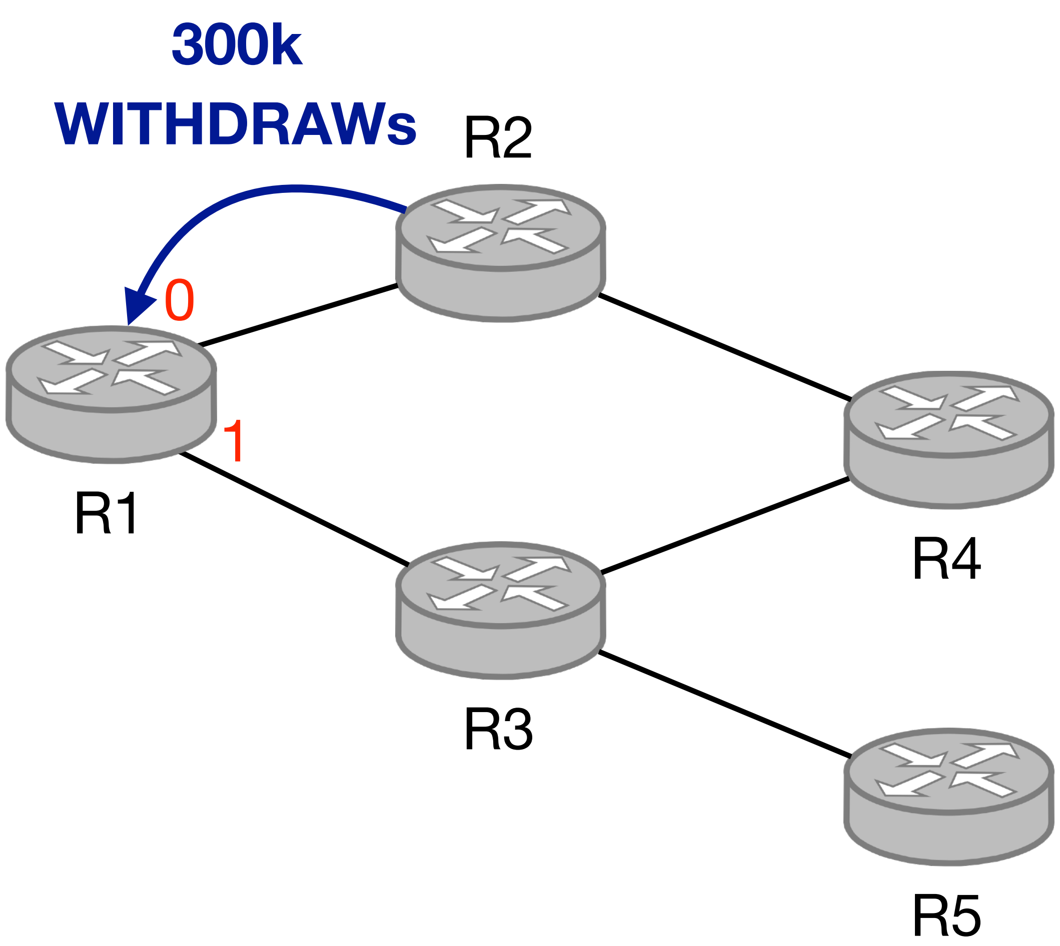
	prefix	Next-Hop
1	1.0.0.0/24	0
2	1.0.1.0/16	0
…	…	…
300k	100.0.0.0/8	0
…	…	…
600k	200.99.0.0/24	0



R1 receives the messages one-by-one and updates its forwarding table entry-by-entry

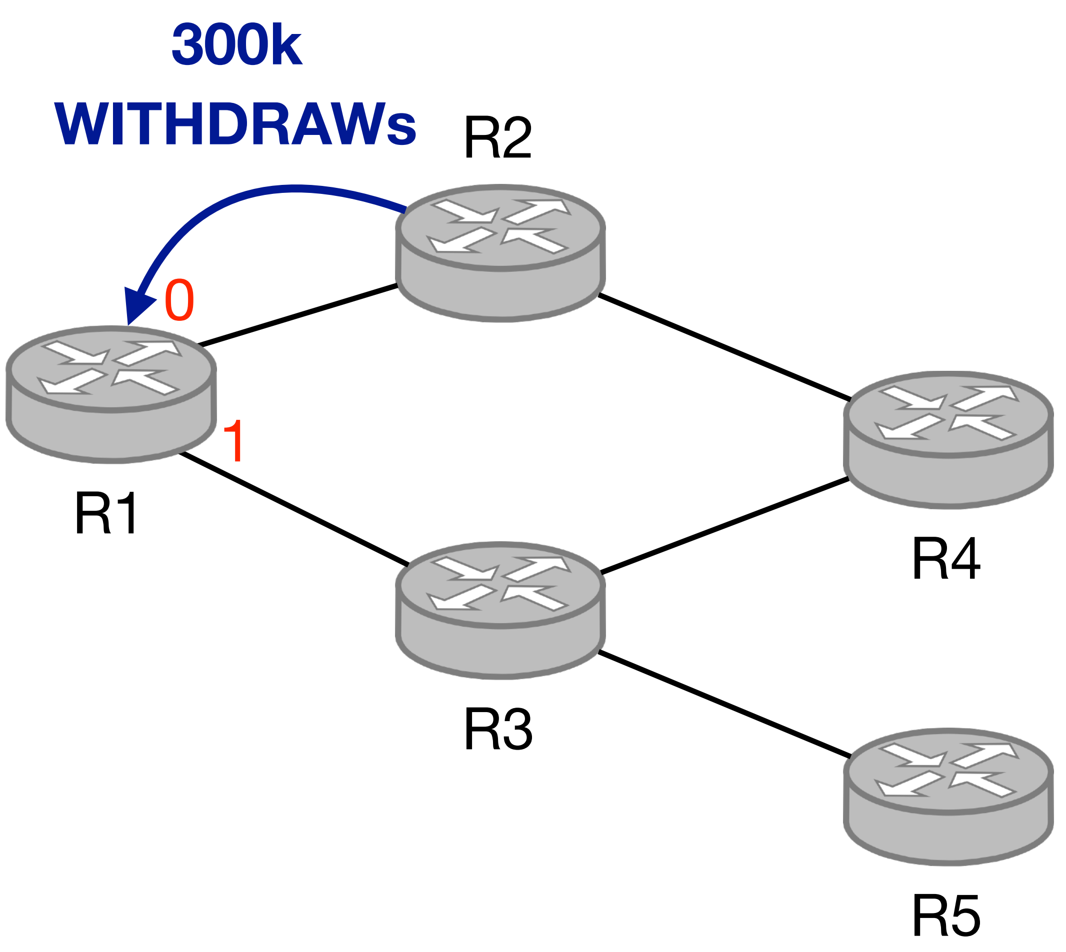
R1's Forwarding Table

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



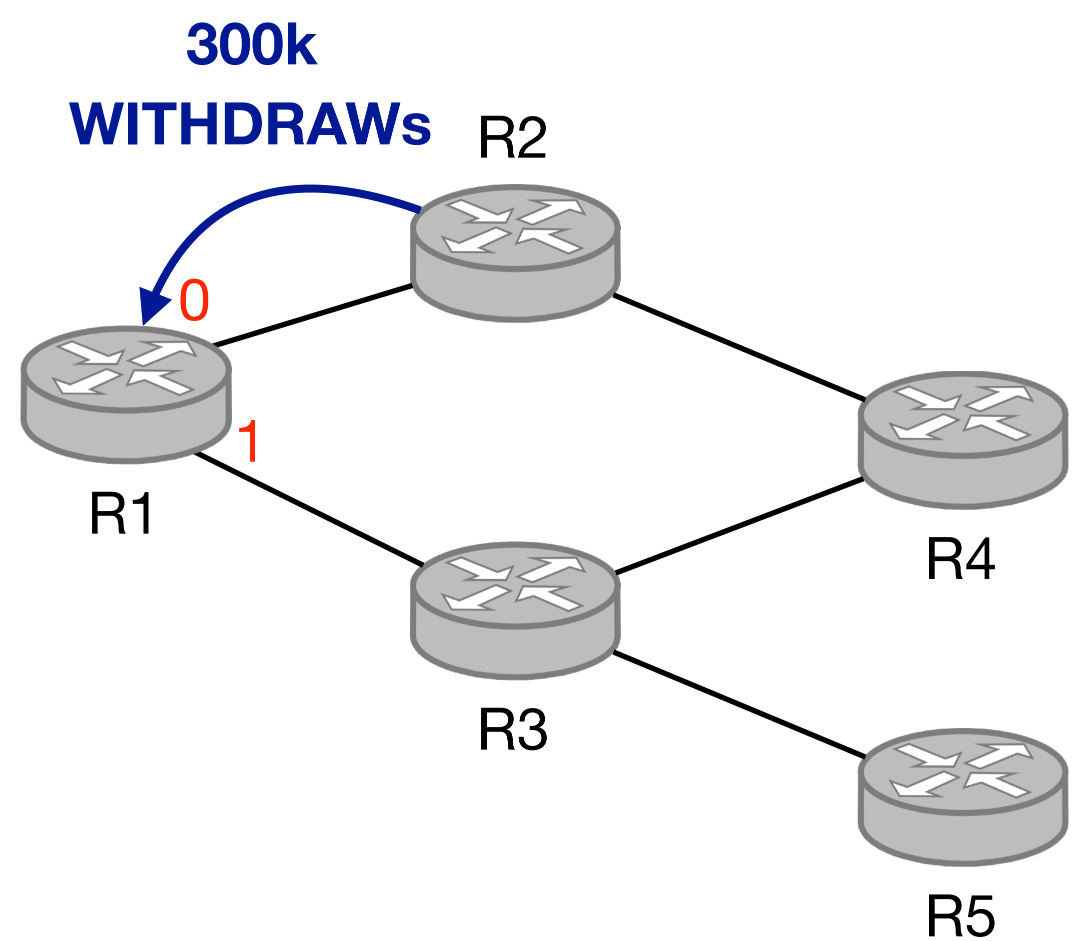
R1’s Forwarding Table

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



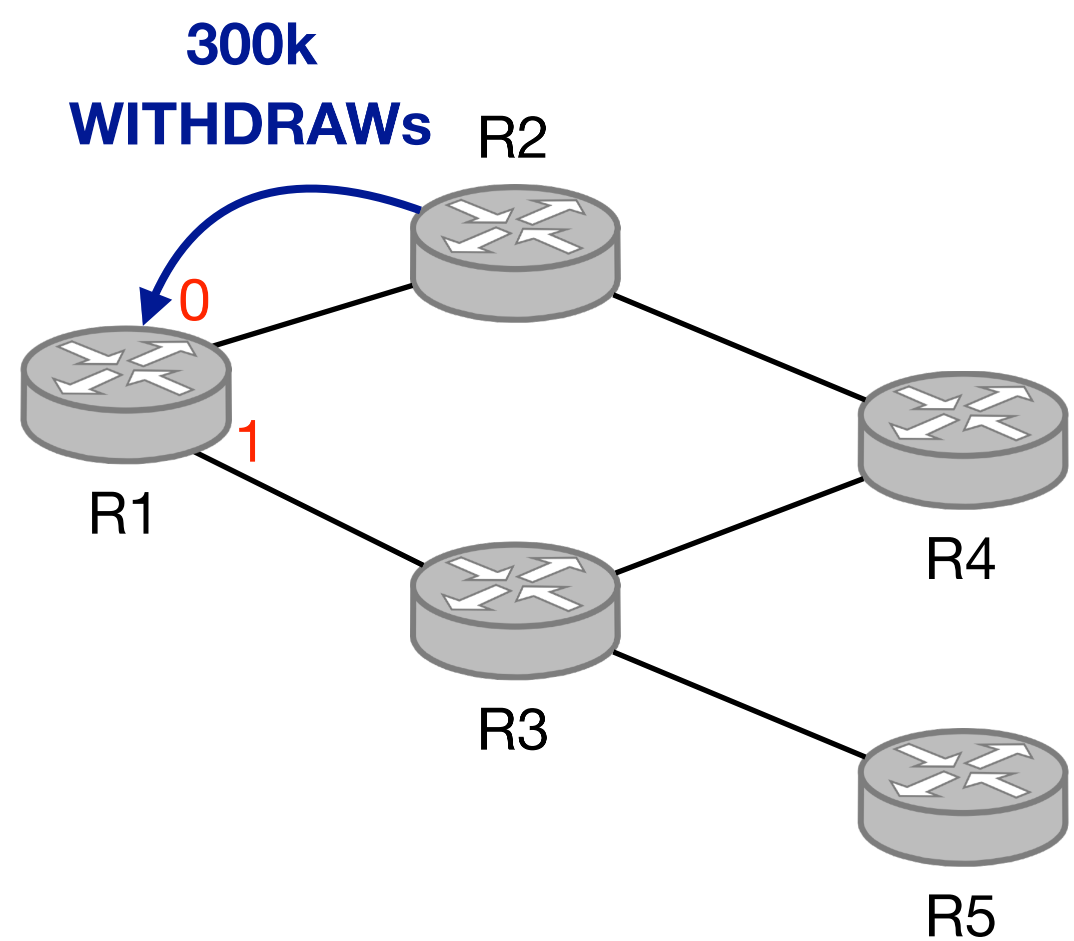
R1's Forwarding Table

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



R1's Forwarding Table

	prefix	Next-Hop
1	1.0.0.0/24	1
2	1.0.1.0/16	1
...	...	...
300k	100.0.0.0/8	1
...	...	...
600k	200.99.0.0/24	0



Internet convergence

a two-phase process

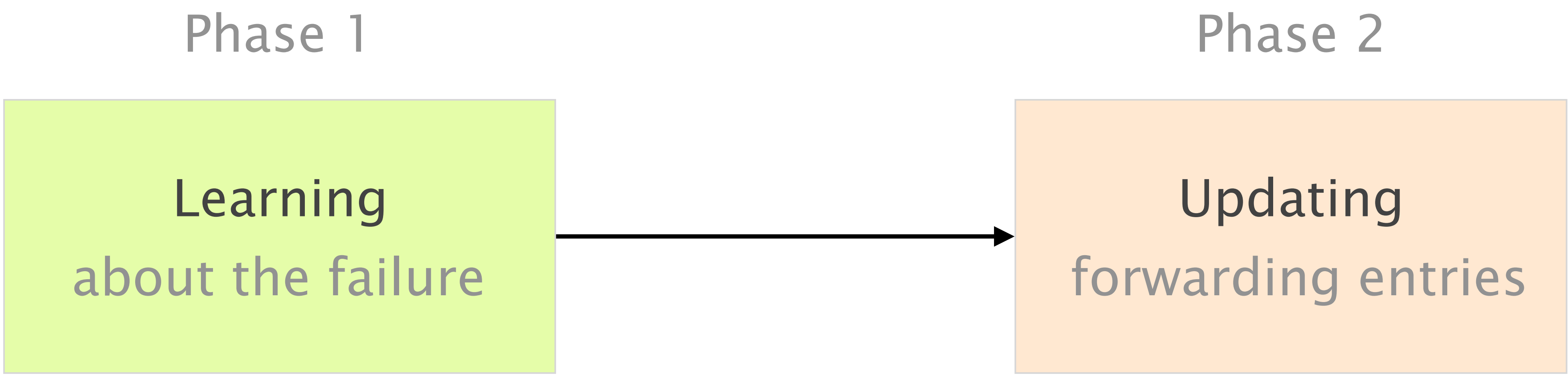
Internet convergence

a two-phase process

Both of which are *terribly* slow…

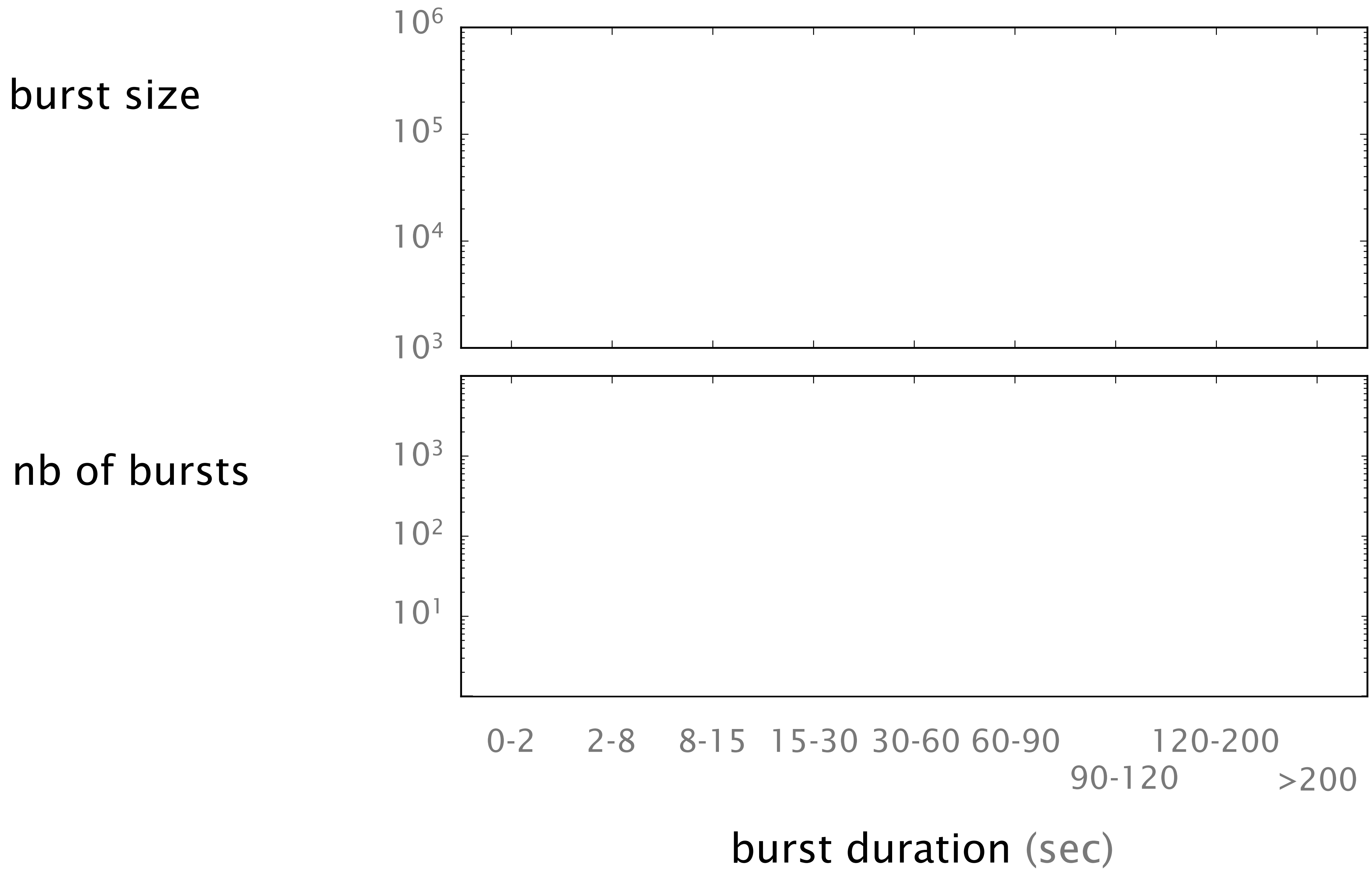
Internet convergence

a two-phase process



We measured how long it takes for large bursts of BGP updates to propagate in the Internet

dataset	a month (July’16) worth of Internet updates from ~200 routers scattered around the globe
methodology	detect the beginning and end of a burst using a 10 sec sliding window

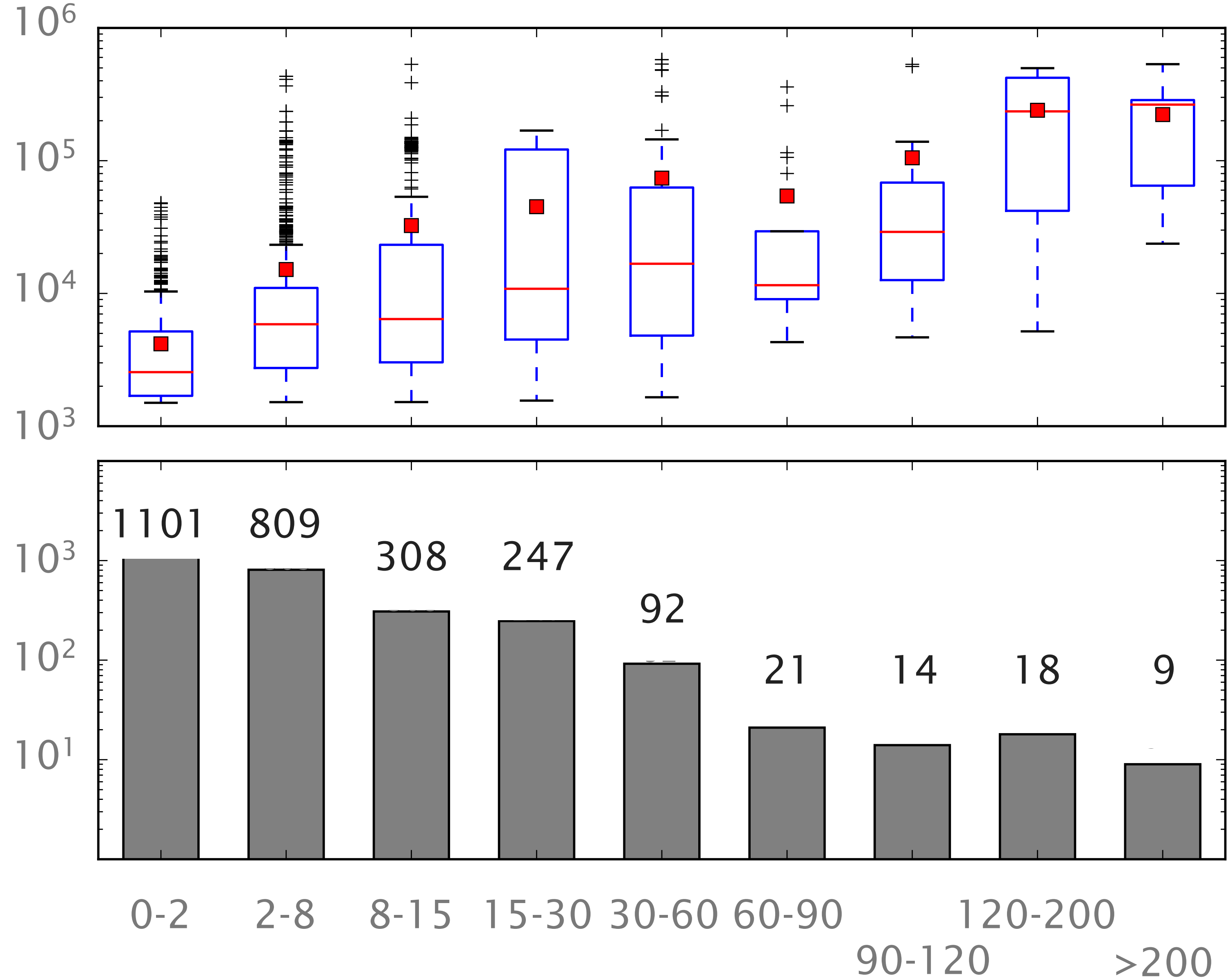


burst size
nb of bursts
10^6
10^5
10^4
10^3
10^3
10^2
10^1
0-2
2-8
8-15
15-30
30-60
60-90
90-120
120-200
>200
burst duration (sec)

We found a total of 2619 bursts over the month

burst size

nb of bursts

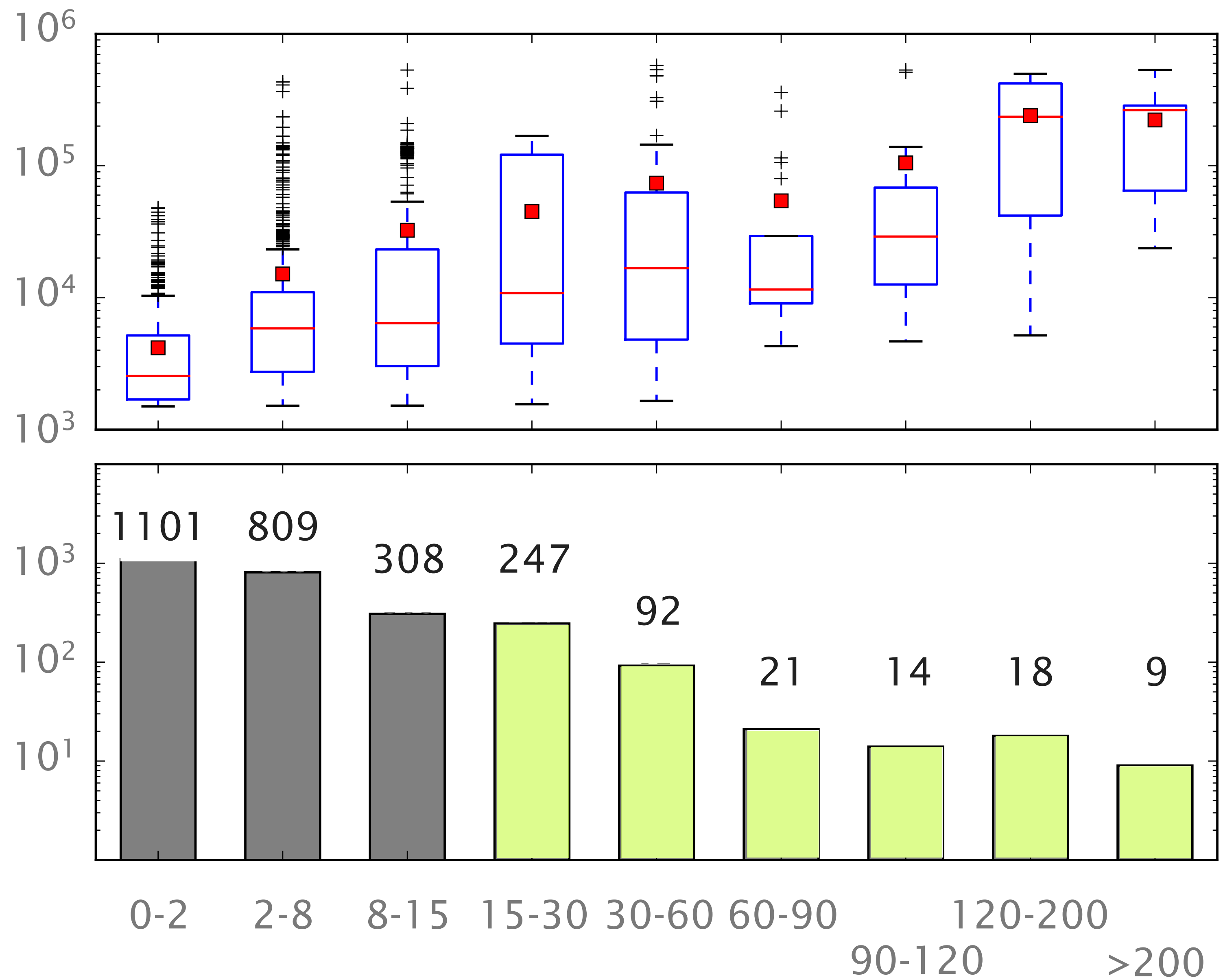


burst duration (sec)

~15% of the bursts takes more than 15s to be learned

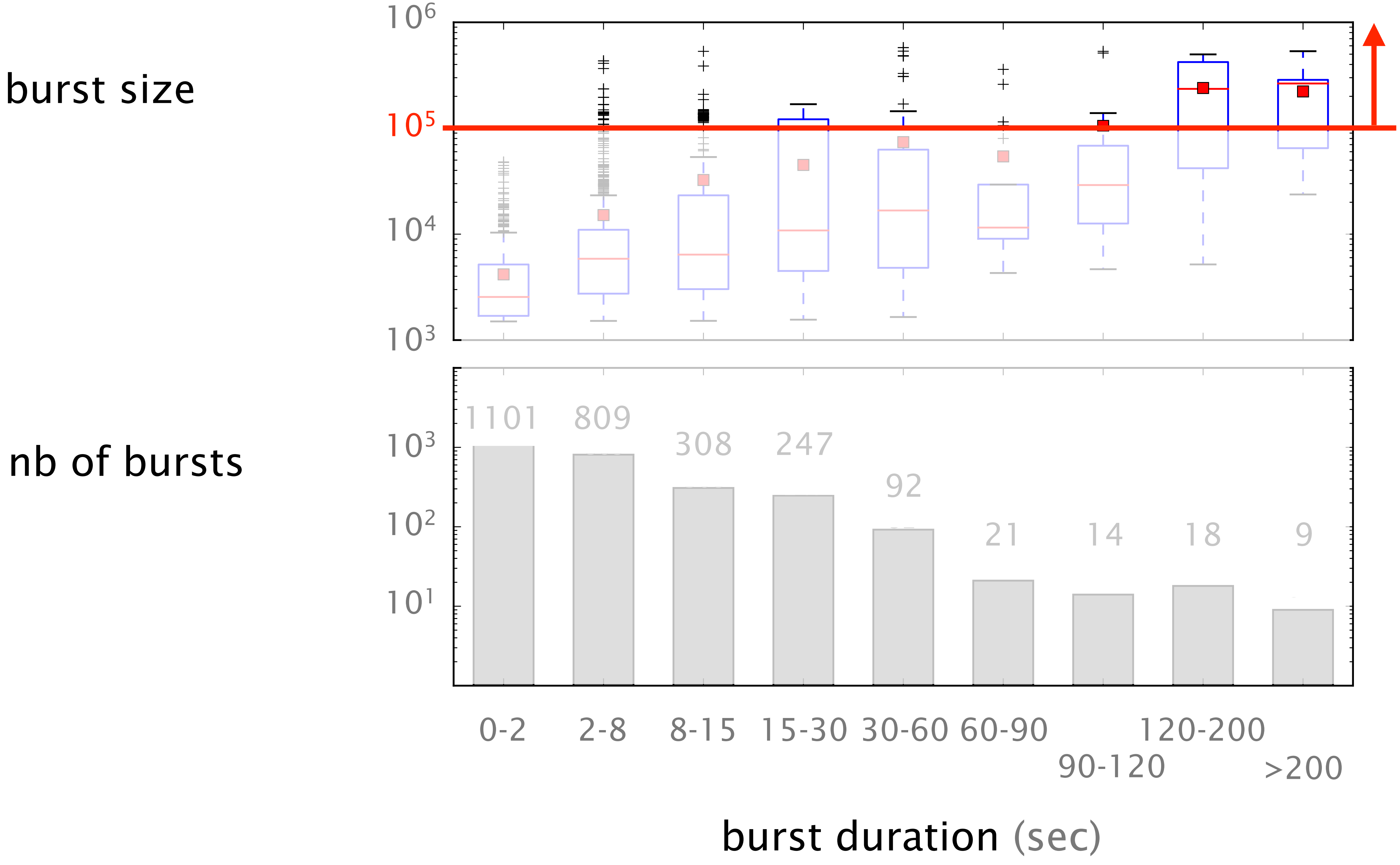
burst size

nb of bursts



burst duration (sec)

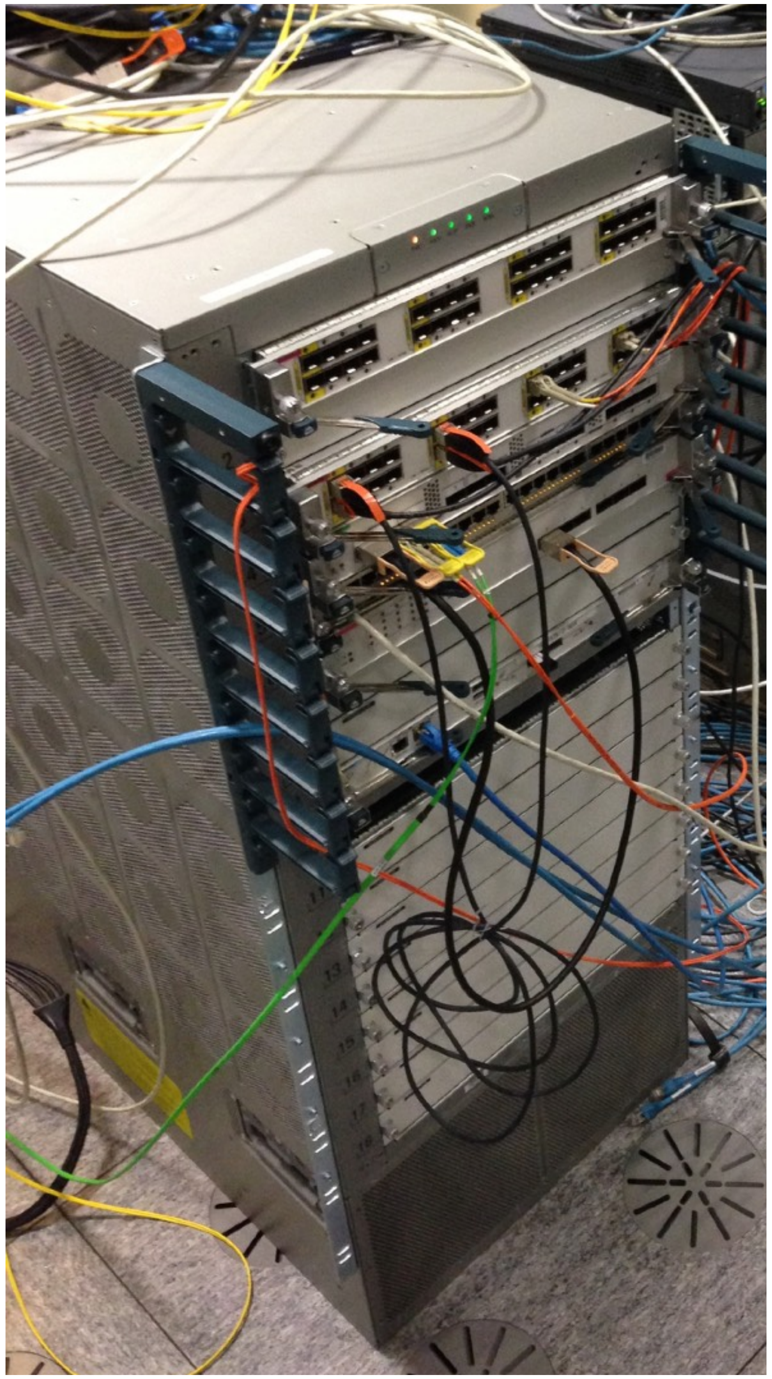
~10% of the bursts contained more than 100k prefixes



Internet convergence

a two-phase process

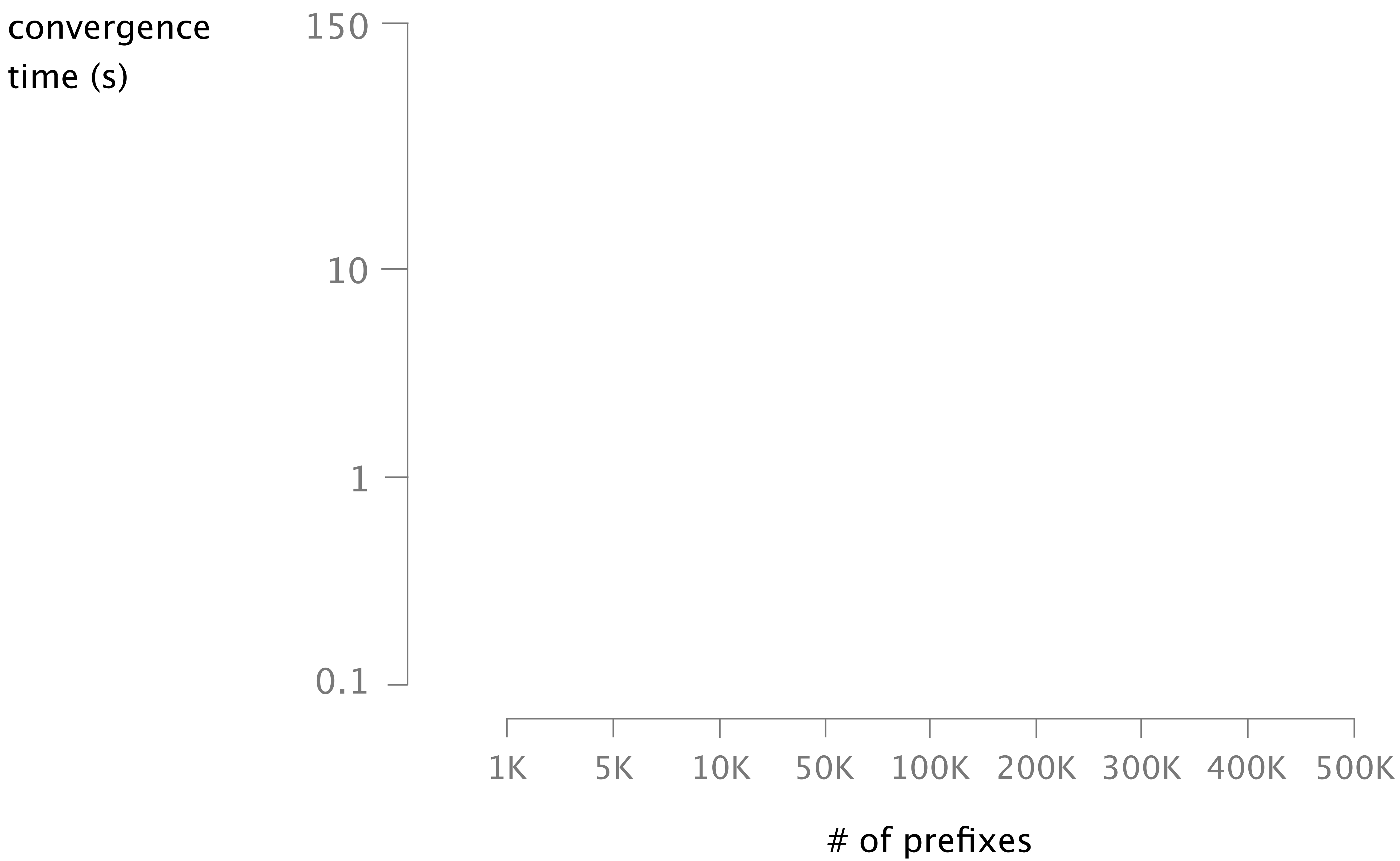
We **measured** how long it takes recent routers
to **update a growing number of forwarding entries**



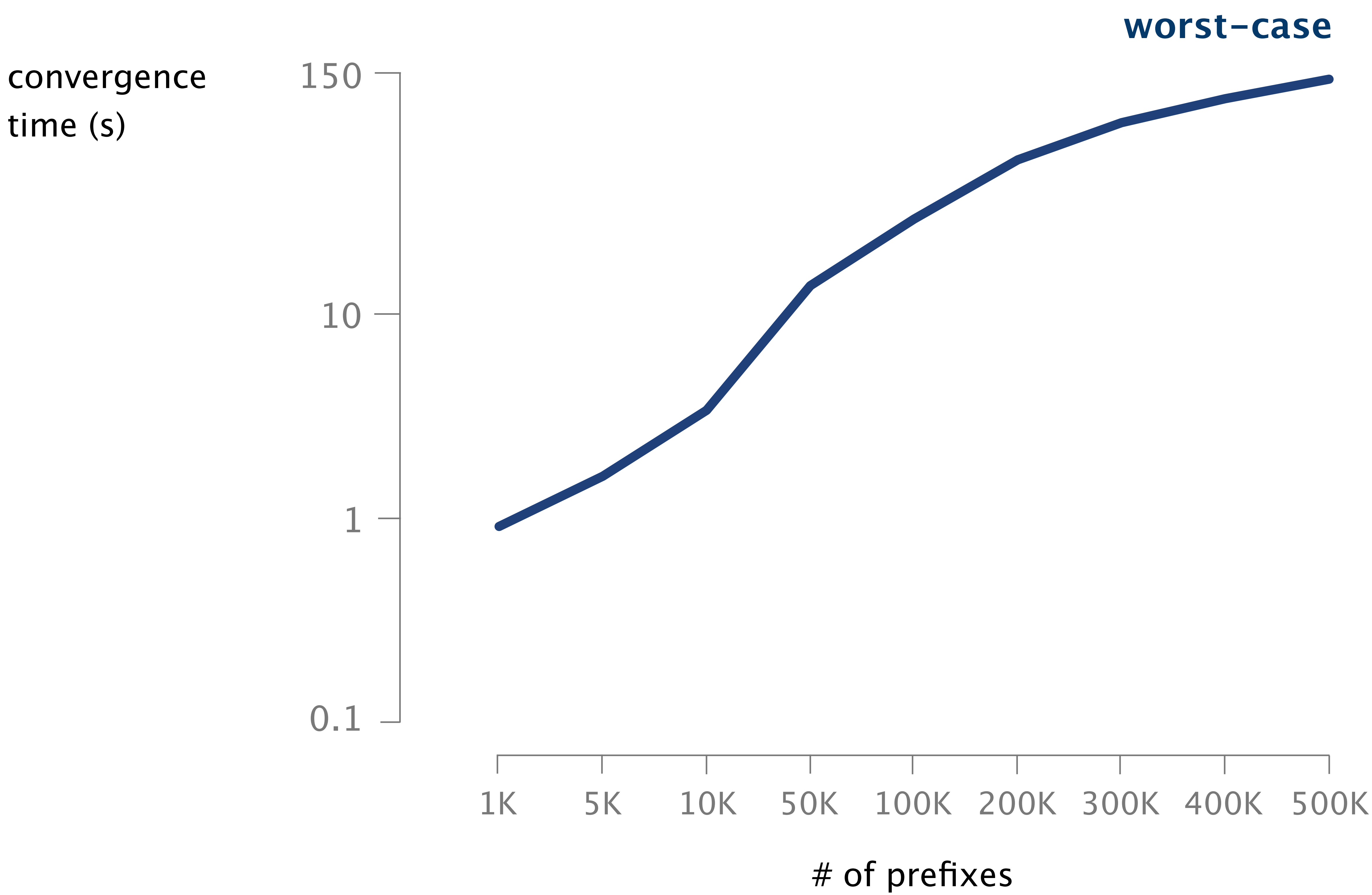
Cisco Nexus 7k

ETH recent routers

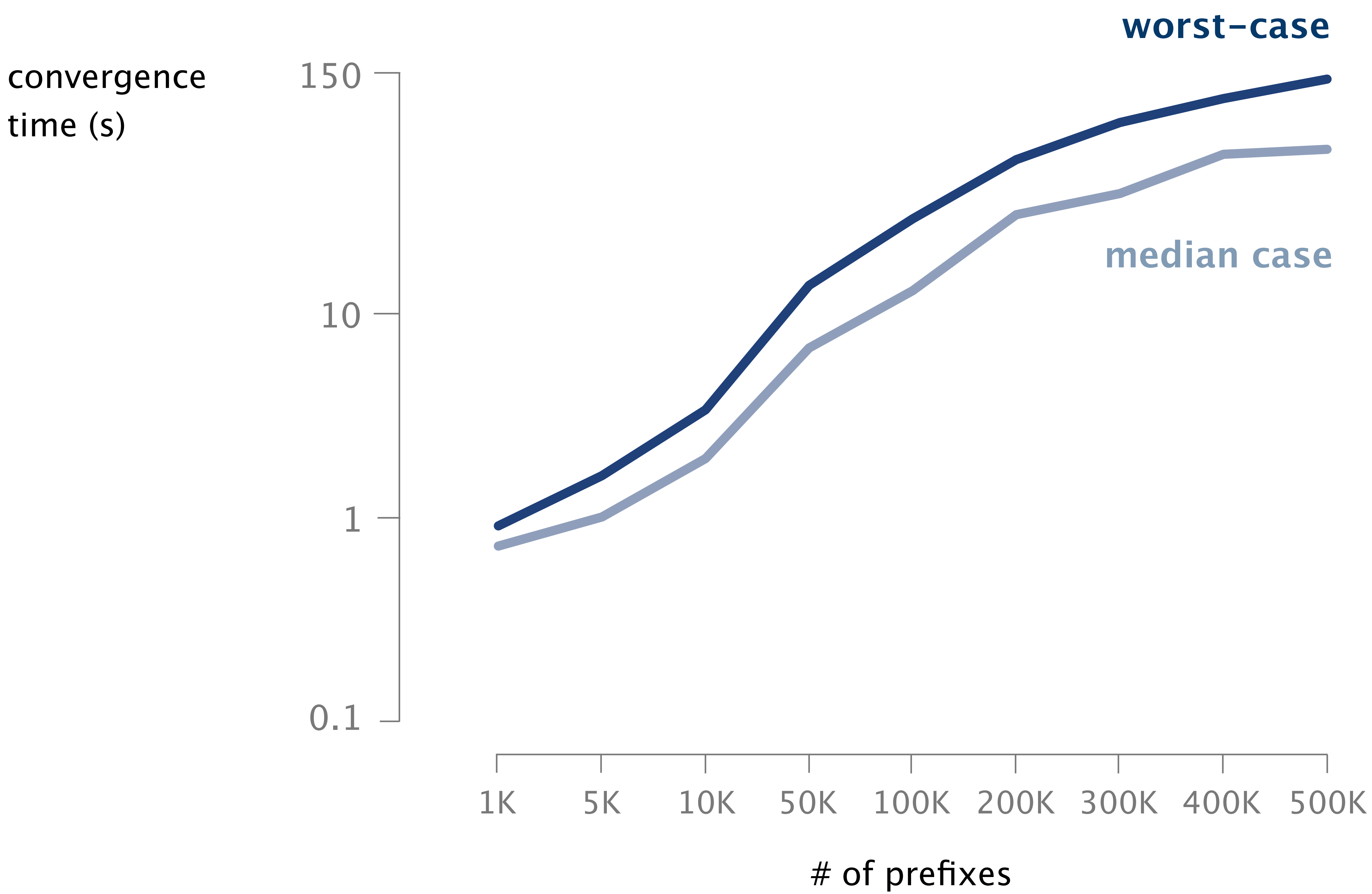
25 deployed



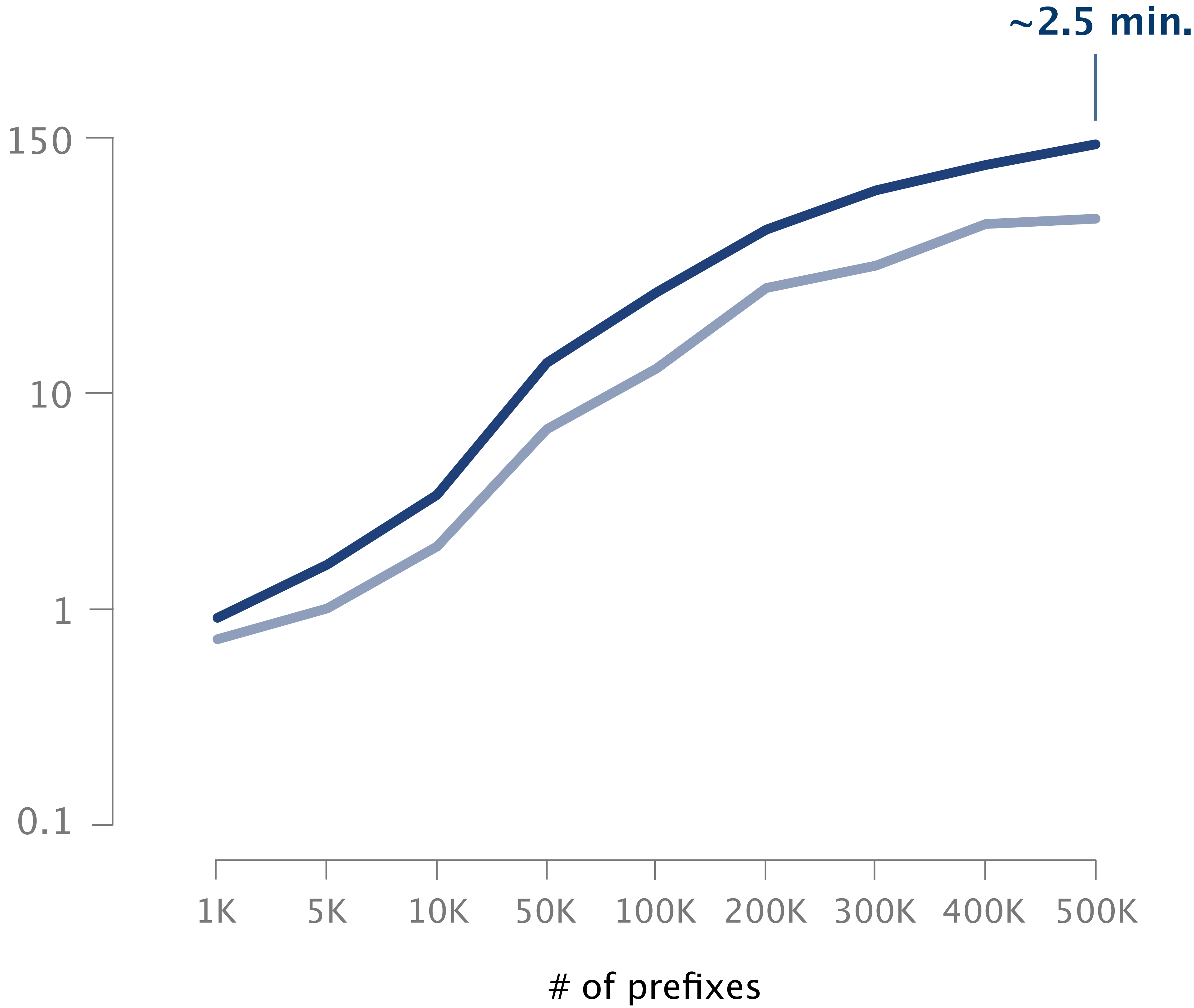
convergence
time (s)
150
10
1
0.1
1K
5K
10K
50K
100K
200K
300K
400K
500K
# of prefixes



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

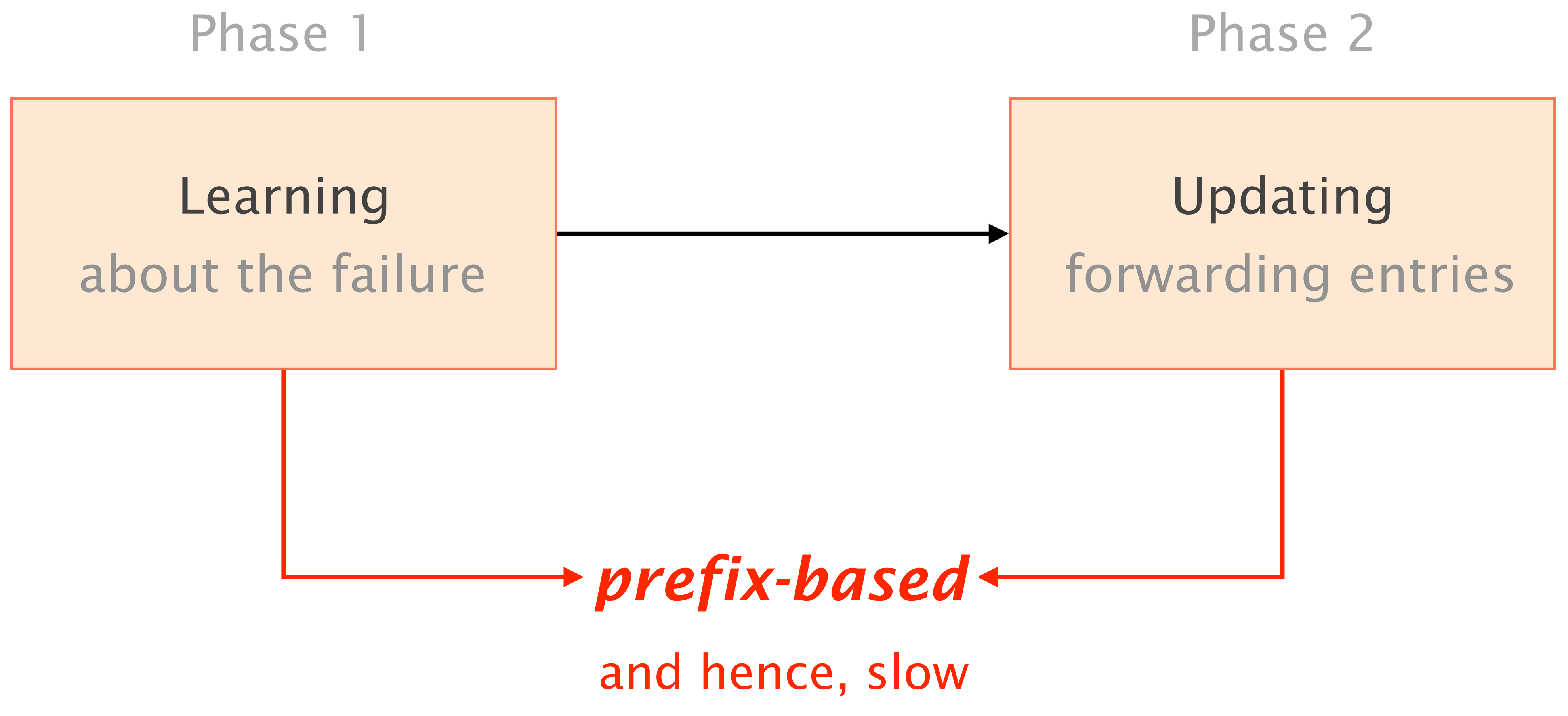

worst-case
median case
convergence
time (s)
150
10
1
0.1
1K
5K
10K
50K
100K
200K
300K
400K
500K
# of prefixes

Traffic can be lost for several minutes



Internet convergence

a two-phase process



SWIFT: Predictive Fast Rerouting

Joint work with: Thomas Holterbach, Alberto Dainotti, Stefano Vissicchio

SWIFT: Predictive Fast Rerouting

speed up... learning

about the failure

SWIFT: Predictive Fast Rerouting

speed up… learning
about the failure

solution *predict* the extent
of a failure from
few messages

SWIFT: Predictive Fast Rerouting

speed up… learning
about the failure

solution *predict* the extent
of a failure from
few messages

challenge speed and precision

SWIFT: Predictive Fast Rerouting

speed up...	learning about the failure	updating the data plane
solution	*predict* the extent of a failure from few messages	
challenge	speed and precision	

SWIFT: Predictive Fast Rerouting

speed up…	learning about the failure	updating the data plane
solution	*predict* the extent of a failure from few messages	update *groups* of entries instead of individual ones
challenge	speed and precision	

SWIFT: Predictive Fast Rerouting

speed up…	learning about the failure	updating the data plane
solution	*predict* the extent of a failure from few messages	update *groups* of entries instead of individual ones
challenge	speed and precision	failure model

SWIFT: Predictive Fast Rerouting

1 **Predicting**
out of few messages

2 **Updating**
groups of entries

3 **Supercharging**
existing systems

SWIFT: Predictive Fast Rerouting

1 Predicting

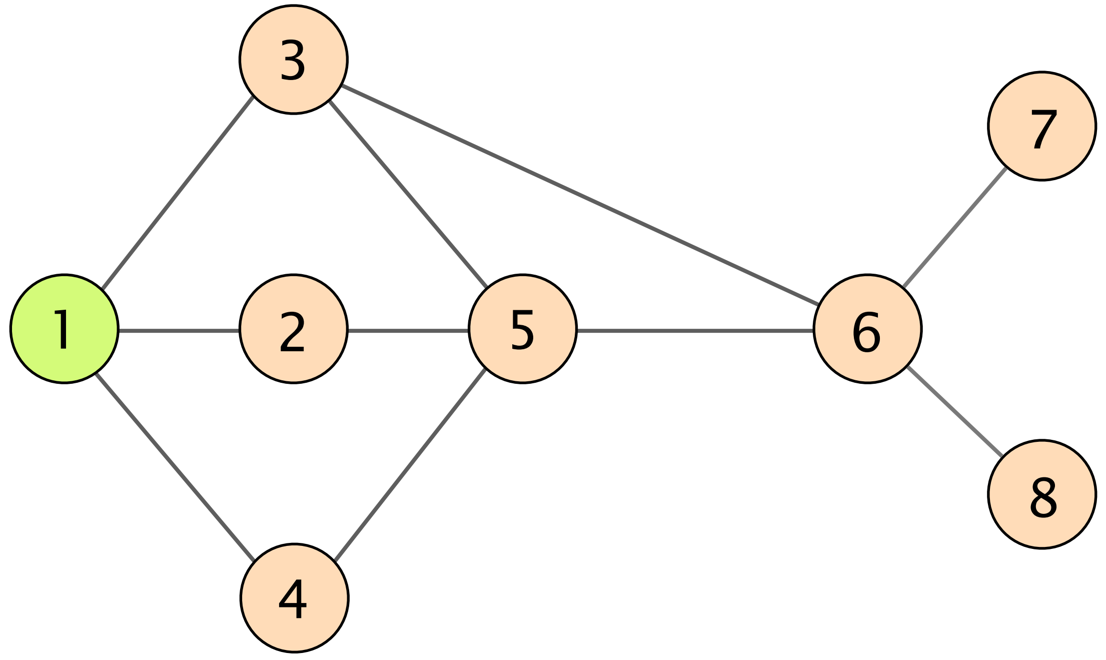
out of few messages

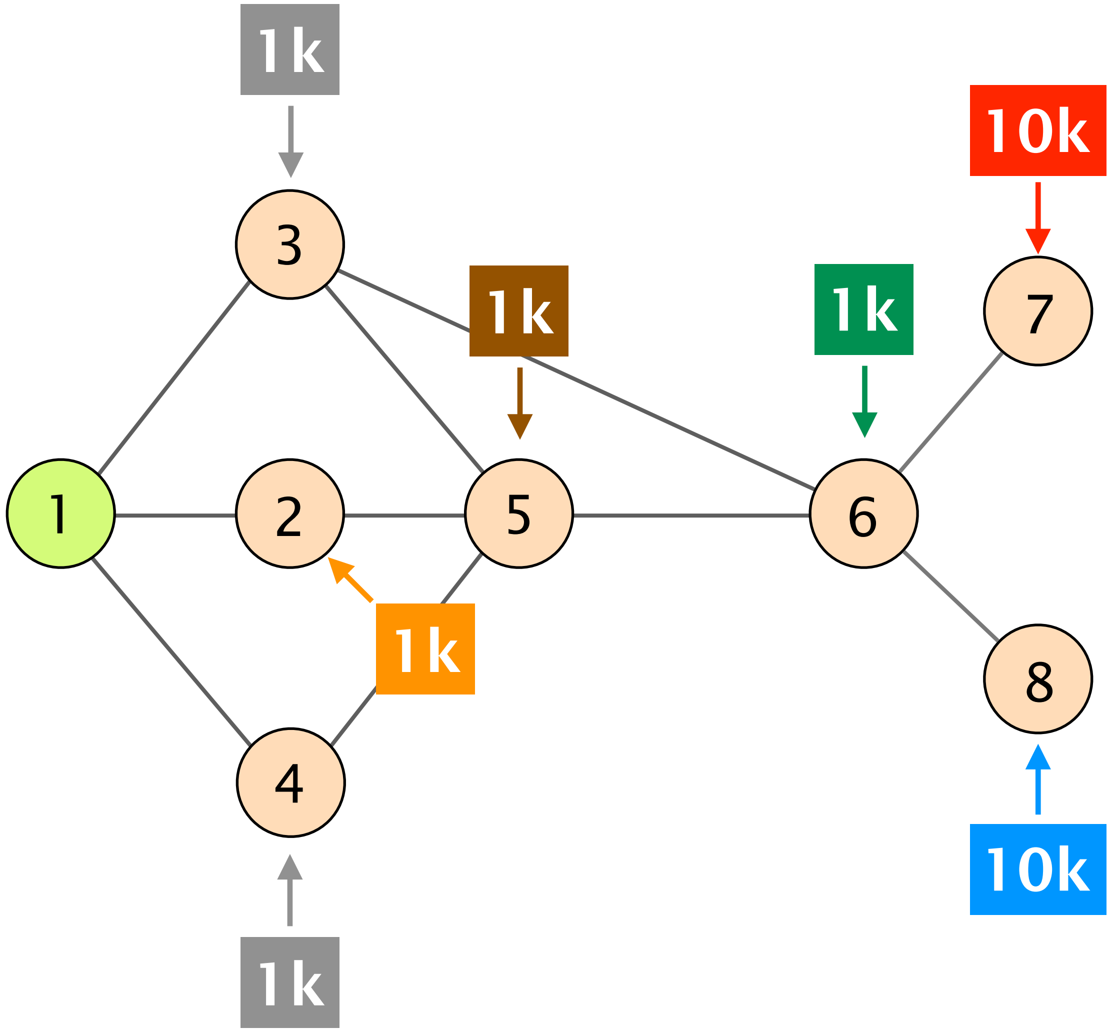
Updating

groups of entries

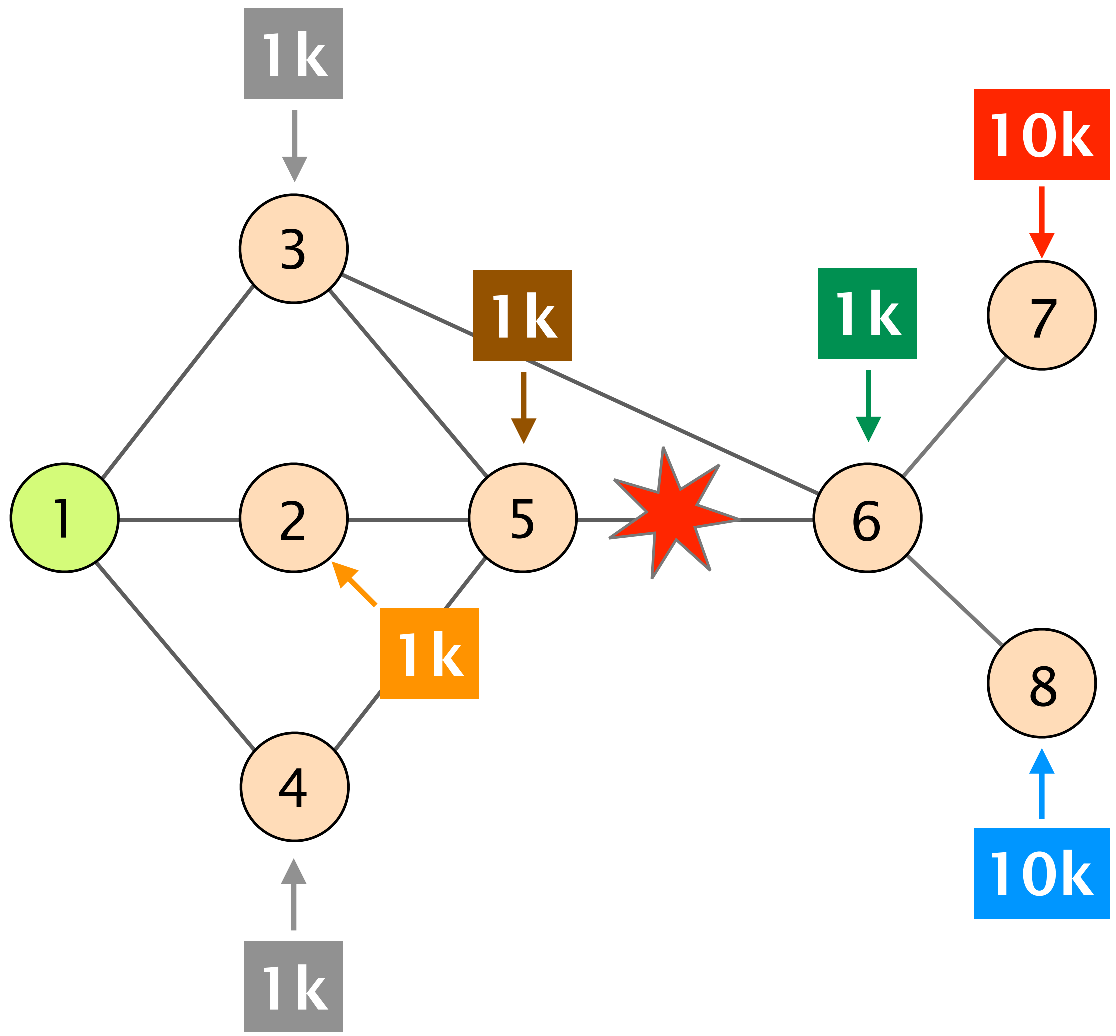
Supercharging

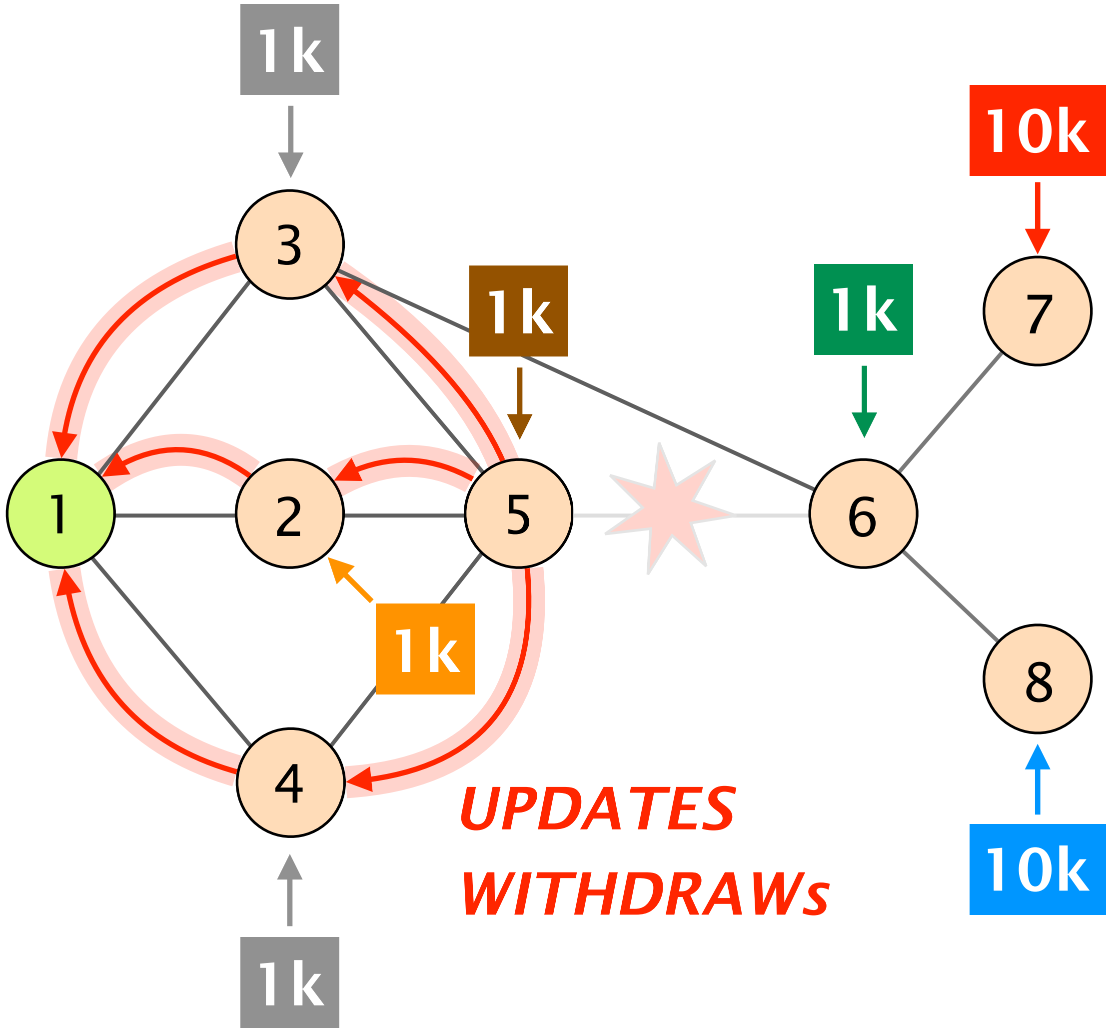
existing systems


1
2
3
4
5
6
7
8



1k
10k
3
1k
1k
7
1
2
5
6
1k
8
4
10k
1k


1k
10k
3
1k
1k
7
1
2
5
6
1k
8
4
10k
1k


1k
10k
3
1k
1k
7
1
2
5
6
1k
8
4
UPDATES
WITHDRAWs
10k
1k

The stream of messages following a disruption contain **redundant information** about the failed resource

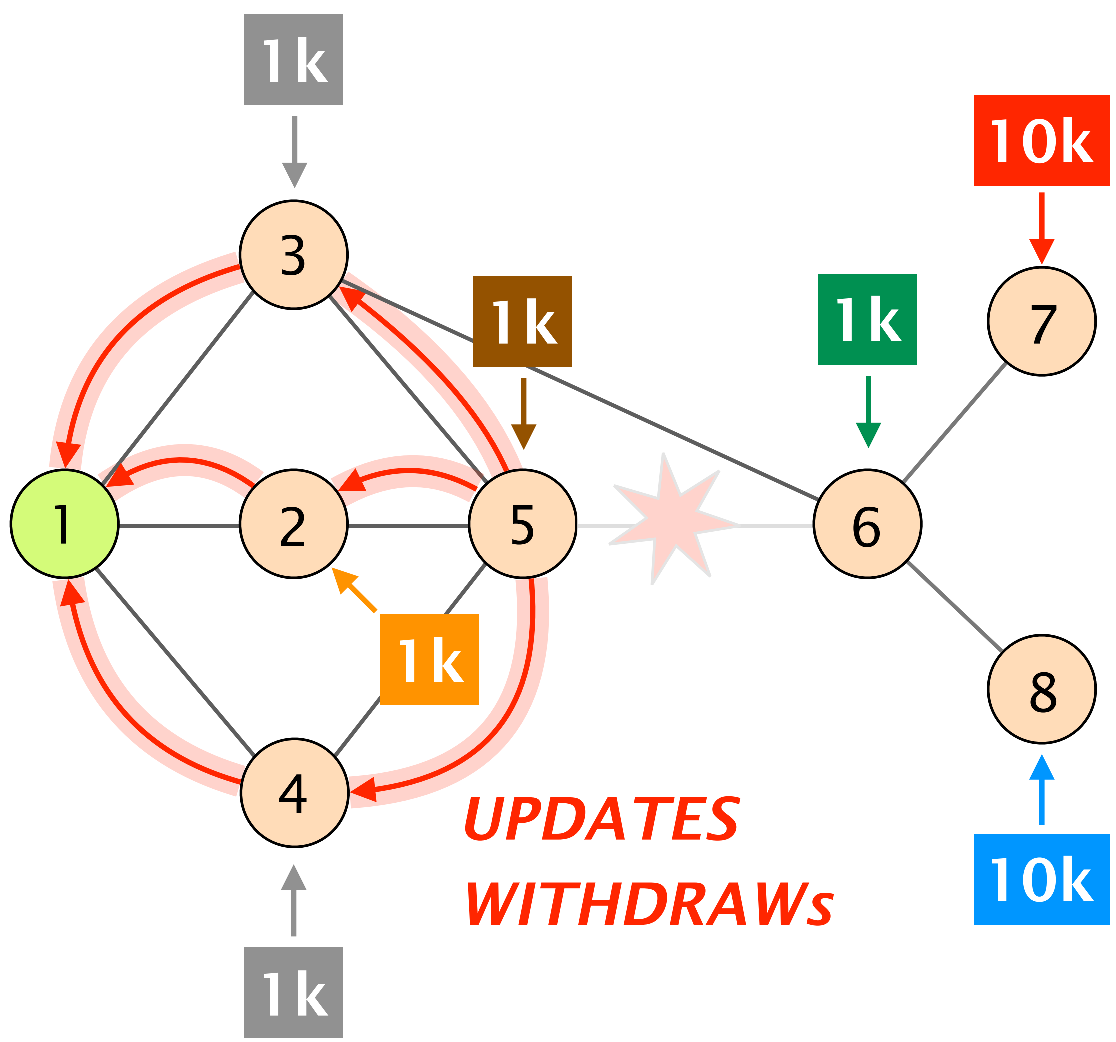
The stream of messages following a disruption contain ==redundant information== about the failed resource

enables prediction

Redundancy comes in two forms:

positive or *negative*

positive	unaffected prefixes are routed on paths which do *not contain* the failed link
negative	affected prefixes must have been routed on a path which *does contain* the failed link



affected prefixes:

(1 2 5 6 7) 10k

(1 2 5 6 8) 10k

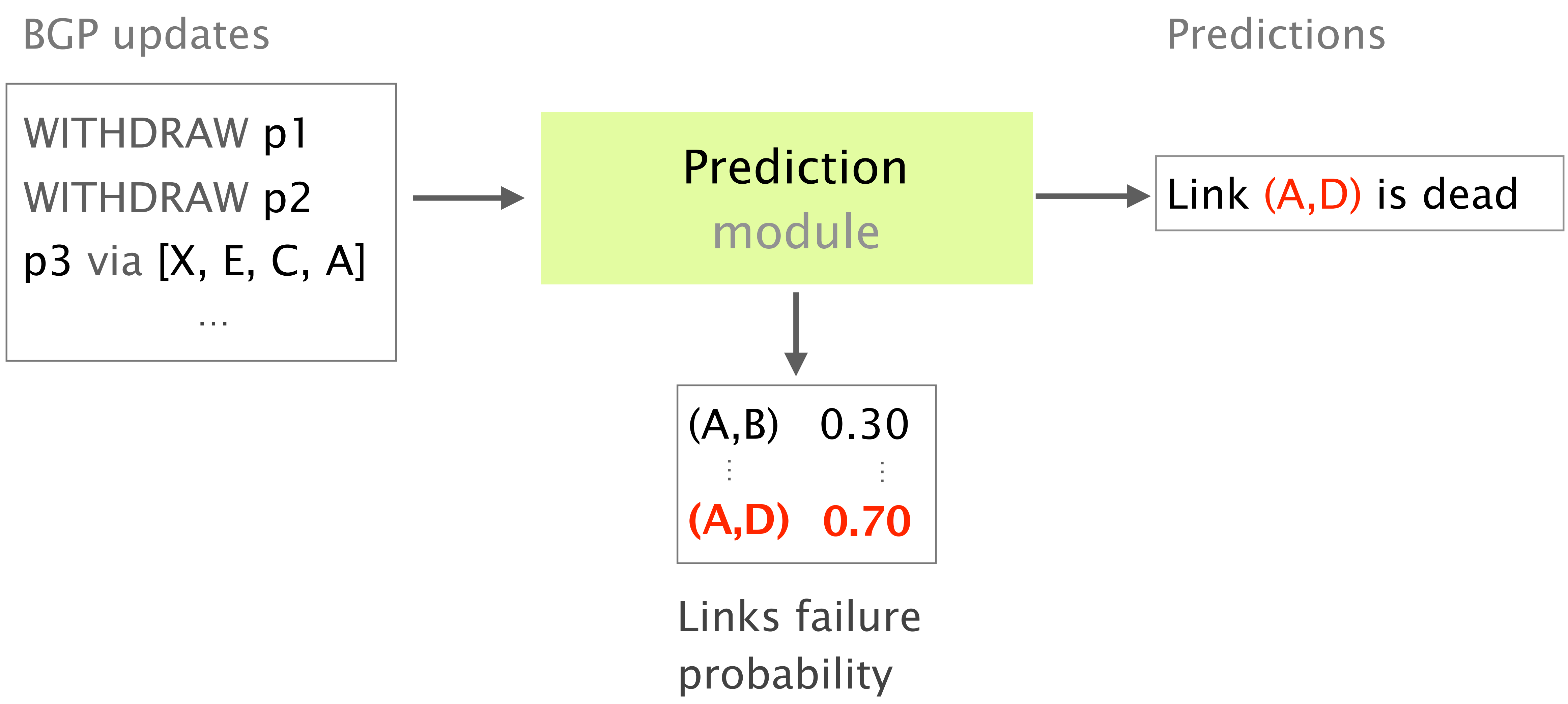
(1 2 5 6) 1k

unaffected prefixes:

(1 2) 1k

(1 2 5) 1k

SWIFT leverages redundancy to predict which link(s) has failed early on into the burst of updates



Step 1

burst detection

Step 1
burst detection

Whenever the frequency of WITHDRAWALs is higher than a threshold (e.g., >99th percentile)

Step 1
burst detection

Whenever the frequency of WITHDRAWALs is higher than a threshold (e.g., >99th percentile)

Step 2
link prediction

Step 1
burst detection

Whenever the frequency of WITHDRAWALs is higher than a threshold (e.g., >99th percentile)

Step 2
link prediction

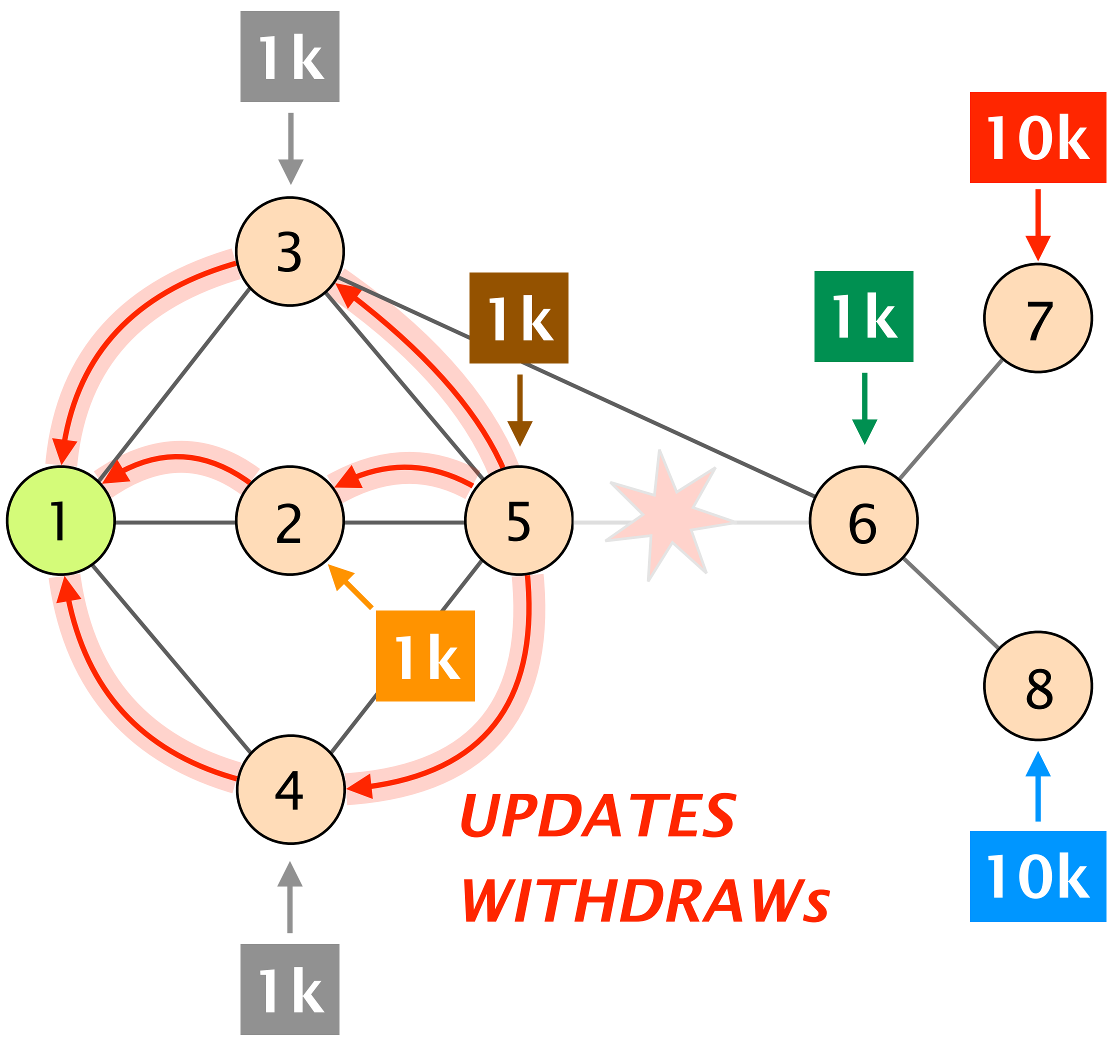
Return the link(s) that maximizes the weighted geometric mean between:

Withdrawal share $WS(l, t)$	Path share $PS(l, t)$
fraction of withdraws crossing link *l*	proportion of prefixes withdrawn on link *l*

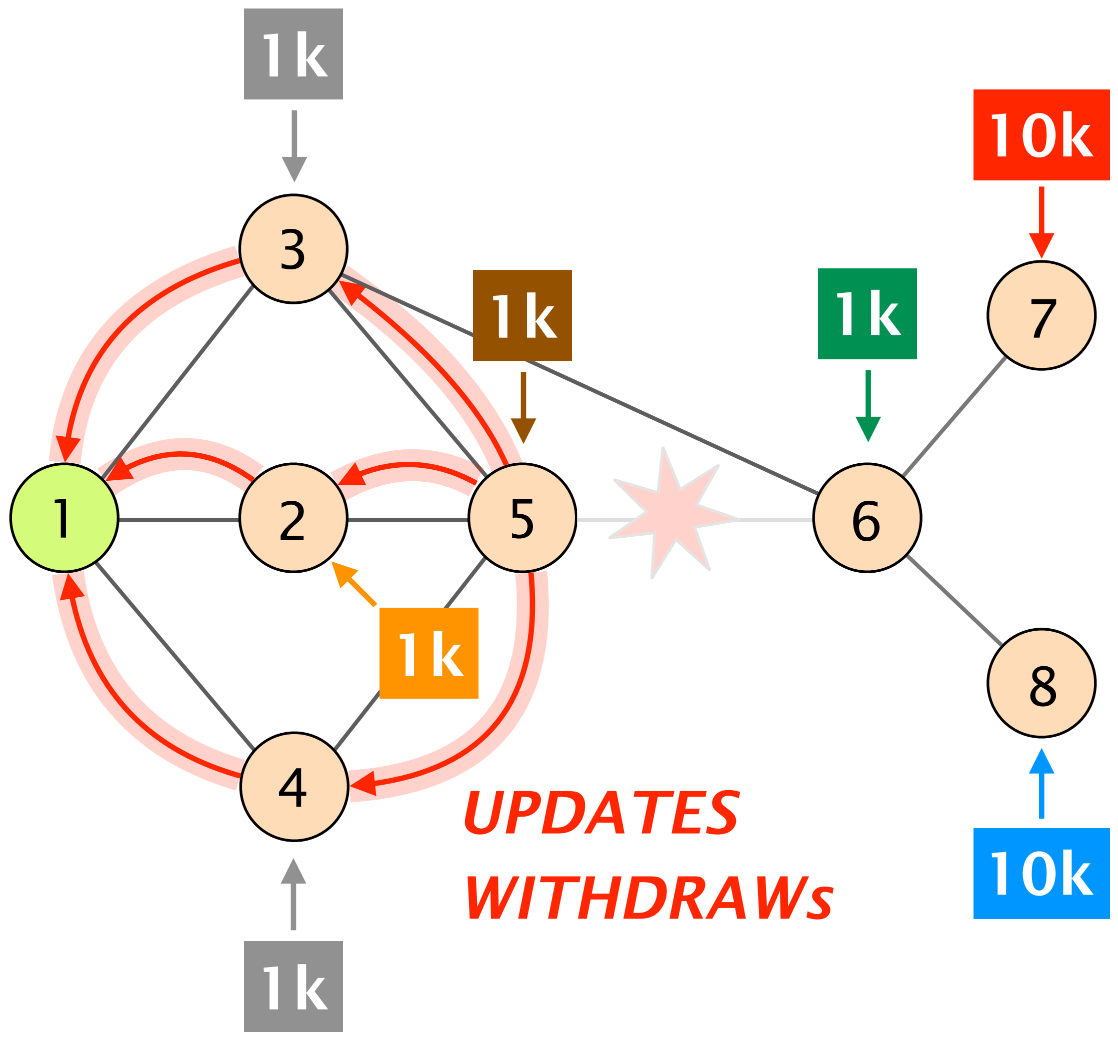
When run on the full burst,
SWIFT is **guaranteed** to find the right link

Theorem

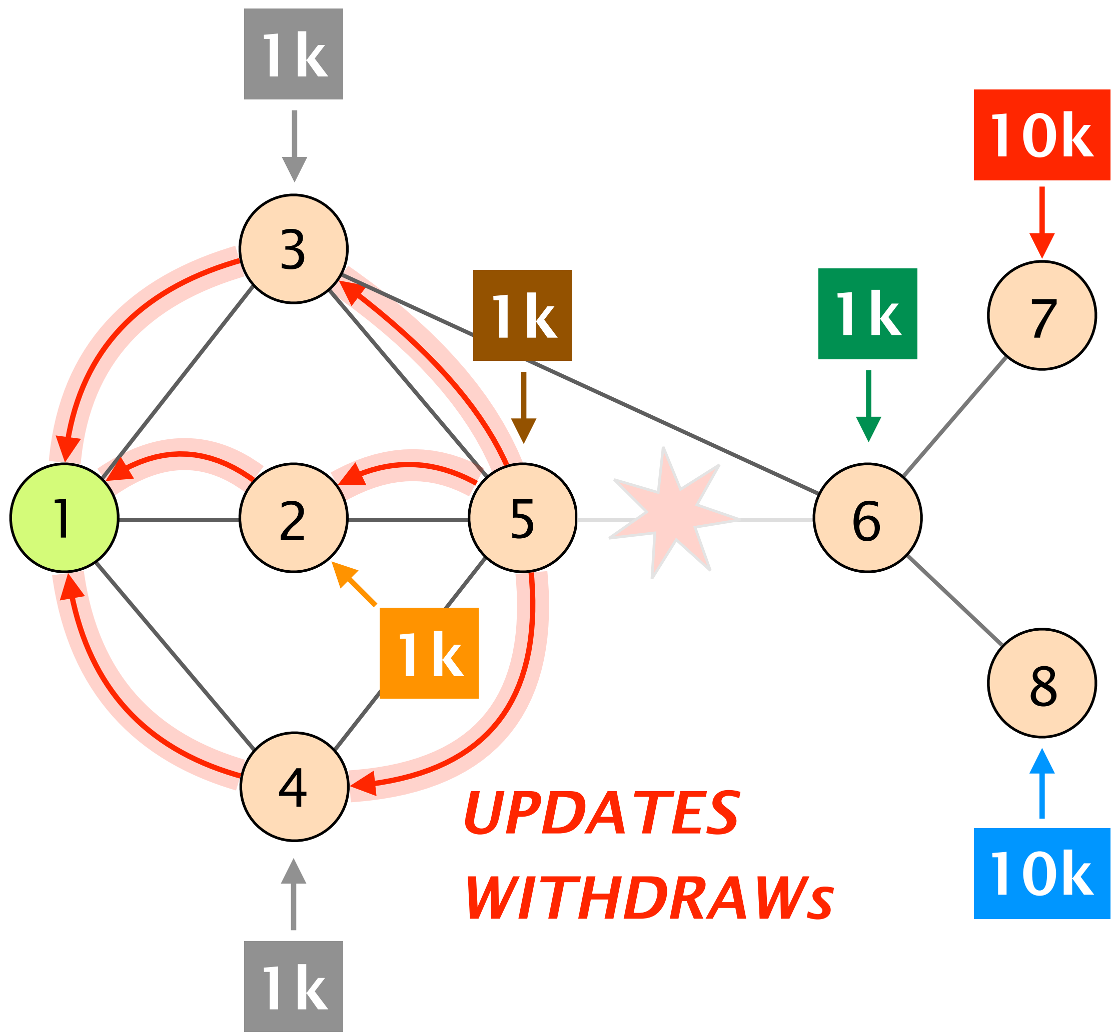
If all ASes inject at least one prefix,
BPA will always correctly pinpoint
the failed link



link	WS	PS	FS
(1,2)			
(2,5)			
(5,6)			
(6,7)			
(6,8)			
other			



link	WS	PS	FS
(1,2)	1	.91	.95
(2,5)	1	.95	.97
(5,6)	1	1	1
(6,7)	.5	1	.7
(6,8)	.5	1	.7
other	0	0	0



link	WS	PS	FS
(1,2)	1	.91	.95
(2,5)	1	.95	.97
(5,6)	1	1	1
(6,7)	.5	1	.7
(6,8)	.5	1	.7
other	0	0	0

When run on the full burst,
SWIFT is guaranteed to find the right link

Theorem

If all ASes inject at least one prefix,
SWIFT will always correctly pinpoints
the failed link

When run on the full burst,

SWIFT is guaranteed to find the right link

not that helpful…

Yet, SWIFT predictions work well in realistic scenarios

Intuition

Messages tend to be interleaved

providing diverse path information early on

Also, SWIFT can compensate for lack of information, by being *overly* cautious (rerouting more)

Returns set of links failures

all links with high fit score

Runs multiple times sequentially

after 2.5k, 5k, 7.5k, 10k,…

Returns set of links failures

all links with high fit score

Runs multiple times sequentially

after 2.5k, 5k, 7.5k, 10k,...

Increase the number of **false positives**

the # of prefixes wrongly predicted as dead

Good news

False positives are **not** an issue!

26 seconds	*vs*	129 600 seconds
allowed downtime for 99.999%		allowed free-riding on a peering link

SWIFT predicts ~90% of the withdrawn prefixes based on only 2.5k messages

	50th	75th	90th
2.5K	**87.50%**	99.10%	99.99%
5.0K	89.70%	98.80%	98.99%
7.5K	92.99%	99.10%	99.99%
10K	95.40%	99.60%	99.99%

Despite not being optimized for it,
SWIFT reroutes few number of non-disrupted prefixes

	50th	75th	90th
2.5K	**0.2x**	1.4x	8.9x
5.0K	0.2x	1.6x	7.2x
7.5K	0.2x	1.8x	7.8x
10K	0.4x	2.8x	9.6x

SWIFT: Predictive Fast Rerouting

Predicting

out of few messages

2 Updating

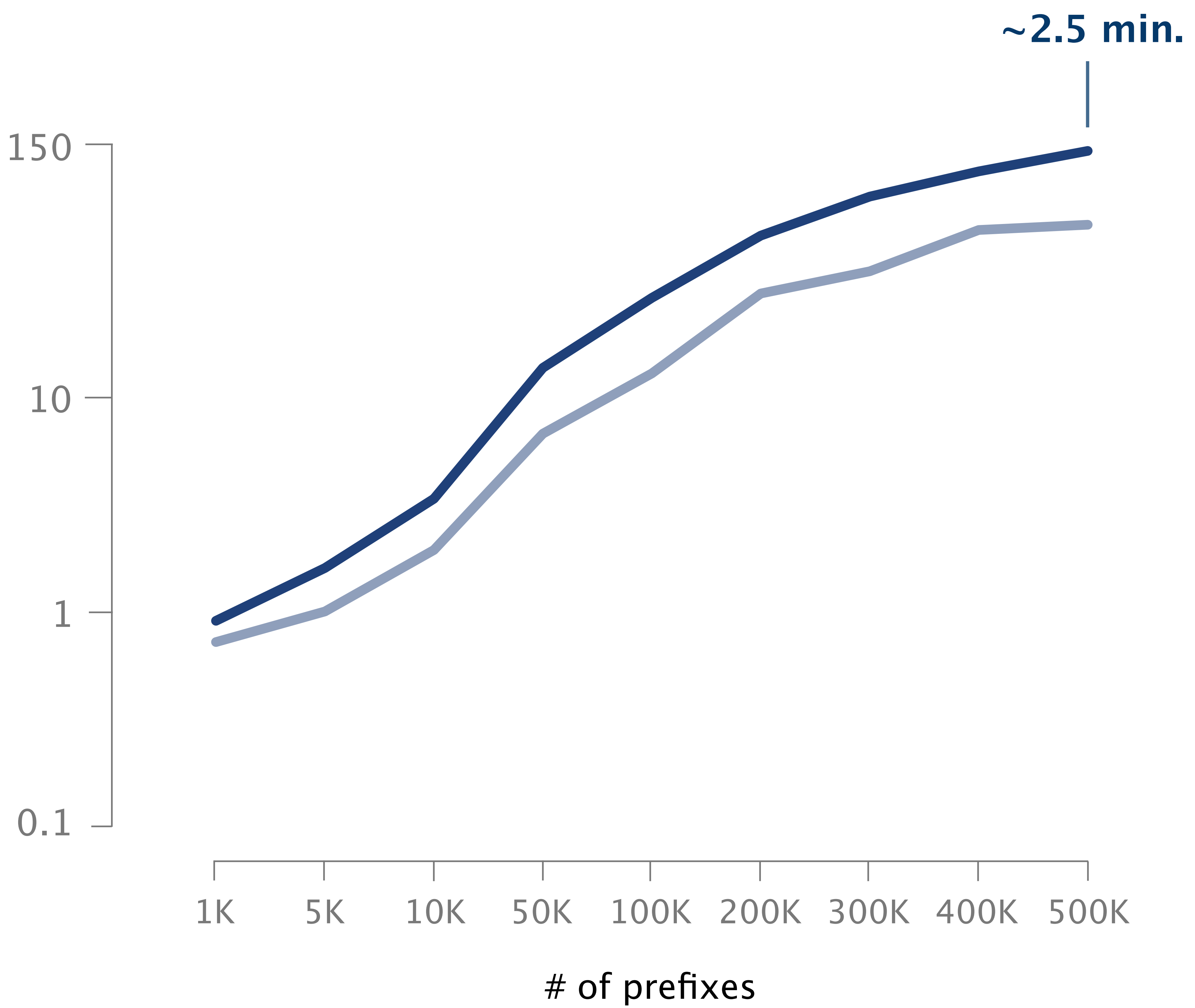
groups of entries

Supercharging

existing systems

Upon a prediction,

SWIFT needs to update the data-plane


~2.5 min.
150
10
1
0.1
1K
5K
10K
50K
100K
200K
300K
400K
500K
# of prefixes

In the Internet though,

any subset of prefixes can fail, in theory

$$\sim 2^{700,000}$$

number of possibilities...

In the Internet though,

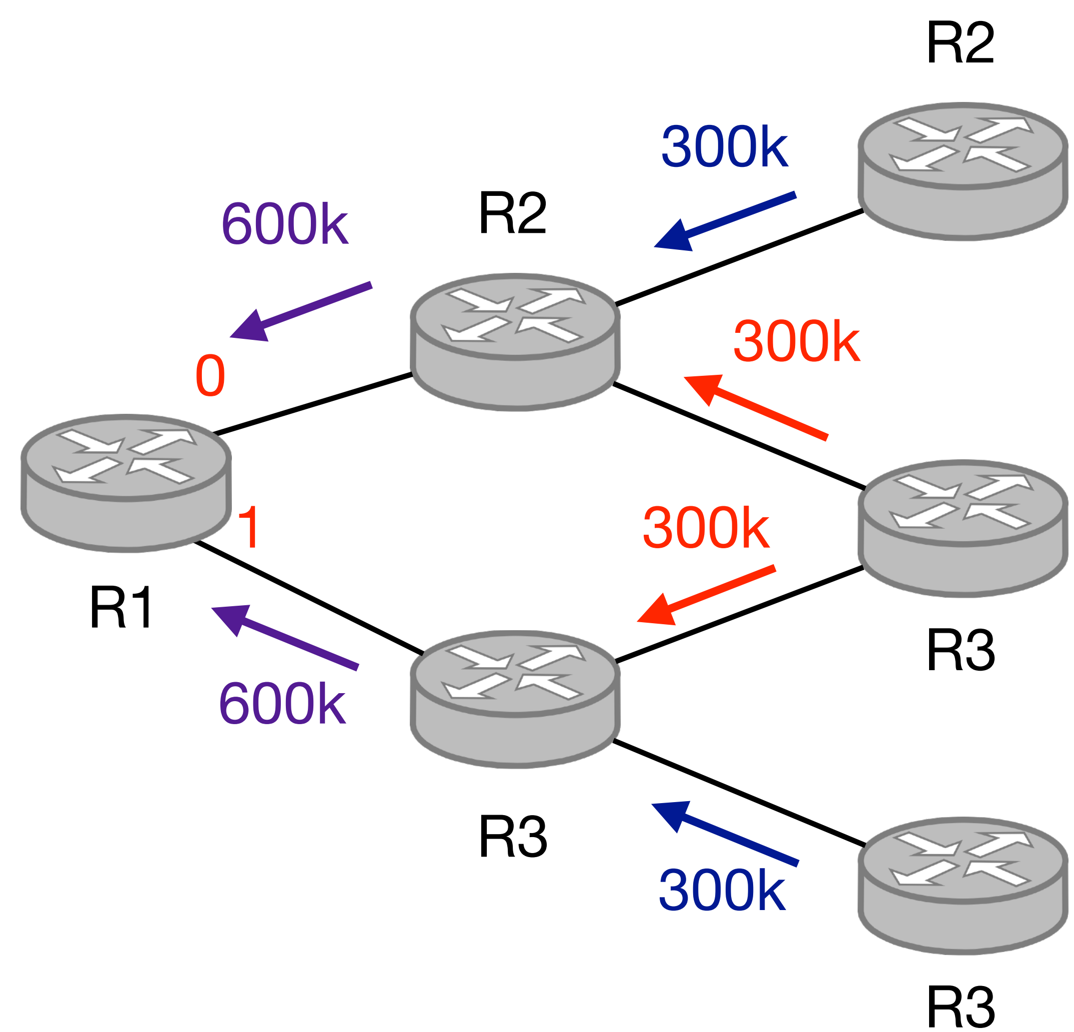
any subset of prefixes can fail, in theory, **not in practice**

To speed-up update time, SWIFT groups prefixes according to the paths they take

All prefixes going via (R1,R2) starts with 10

R1's Forwarding Table

	prefix	tag	NH
1	1.0.0.0/24	10 01 …	0
2	1.0.1.0/16	10 01 …	0
…	…		…
300k	100.0.0.0/8	10 11 …	0
…	…		…
600k	200.99.0.0/24	10 11 …	0



If (R1,R2) fails (or is predicted to have failed)

updating one rule is enough to reroute all traffic

m(10.*) >> fwd(1)

Since the AS graph is too large to be encoded, SWIFT reduces it first using two techniques

Ignore any link seeing less than 1.5k pfxes

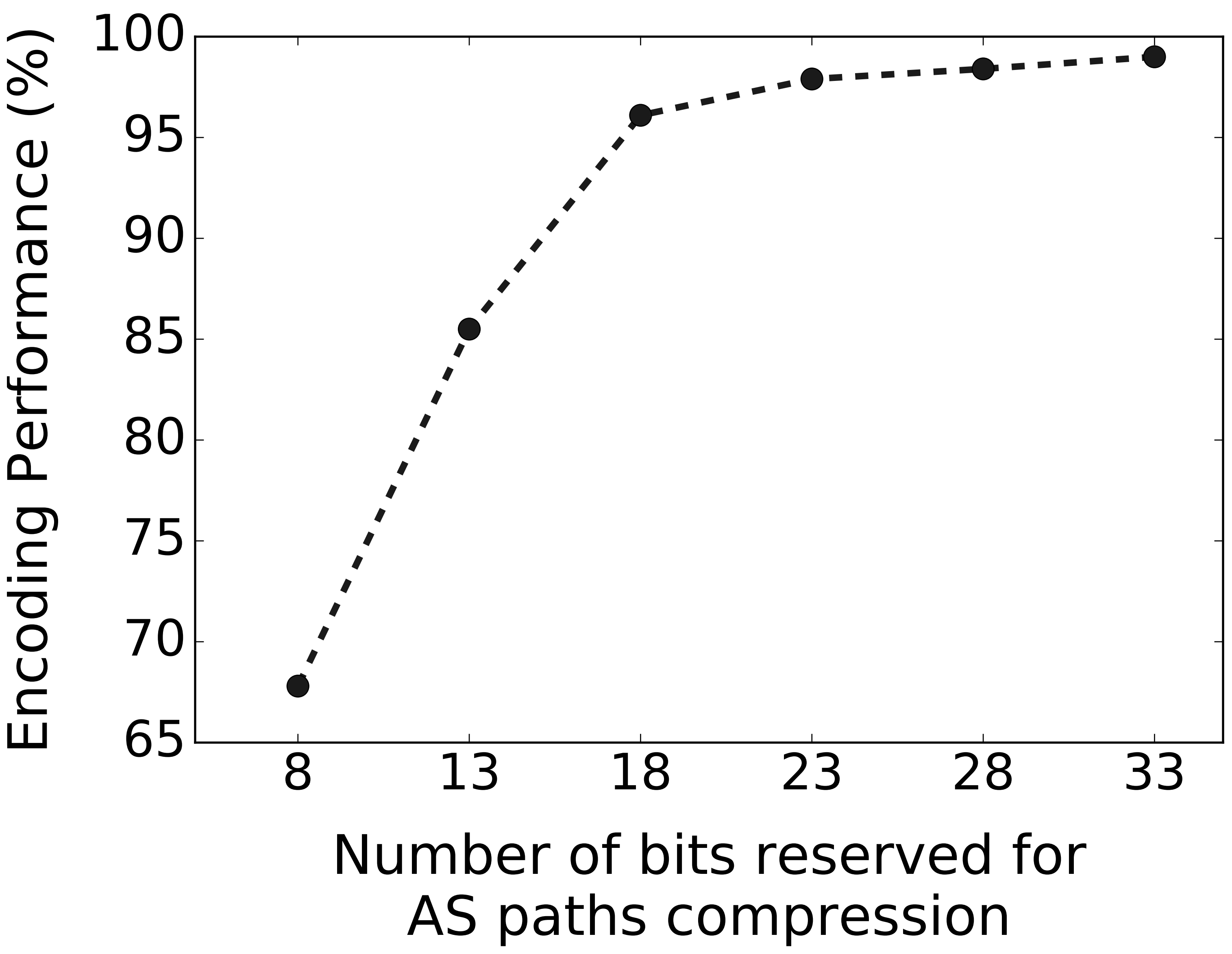
anything less converges fast enough already

Ignore link far away from the SWIFTed node

less likely to create large bursts of UPDATEs

These two optimizations enable to reroute

96% of the predicted prefixes using only 18 bits



SWIFT: Predictive Fast Rerouting

Predicting

out of few messages

Updating

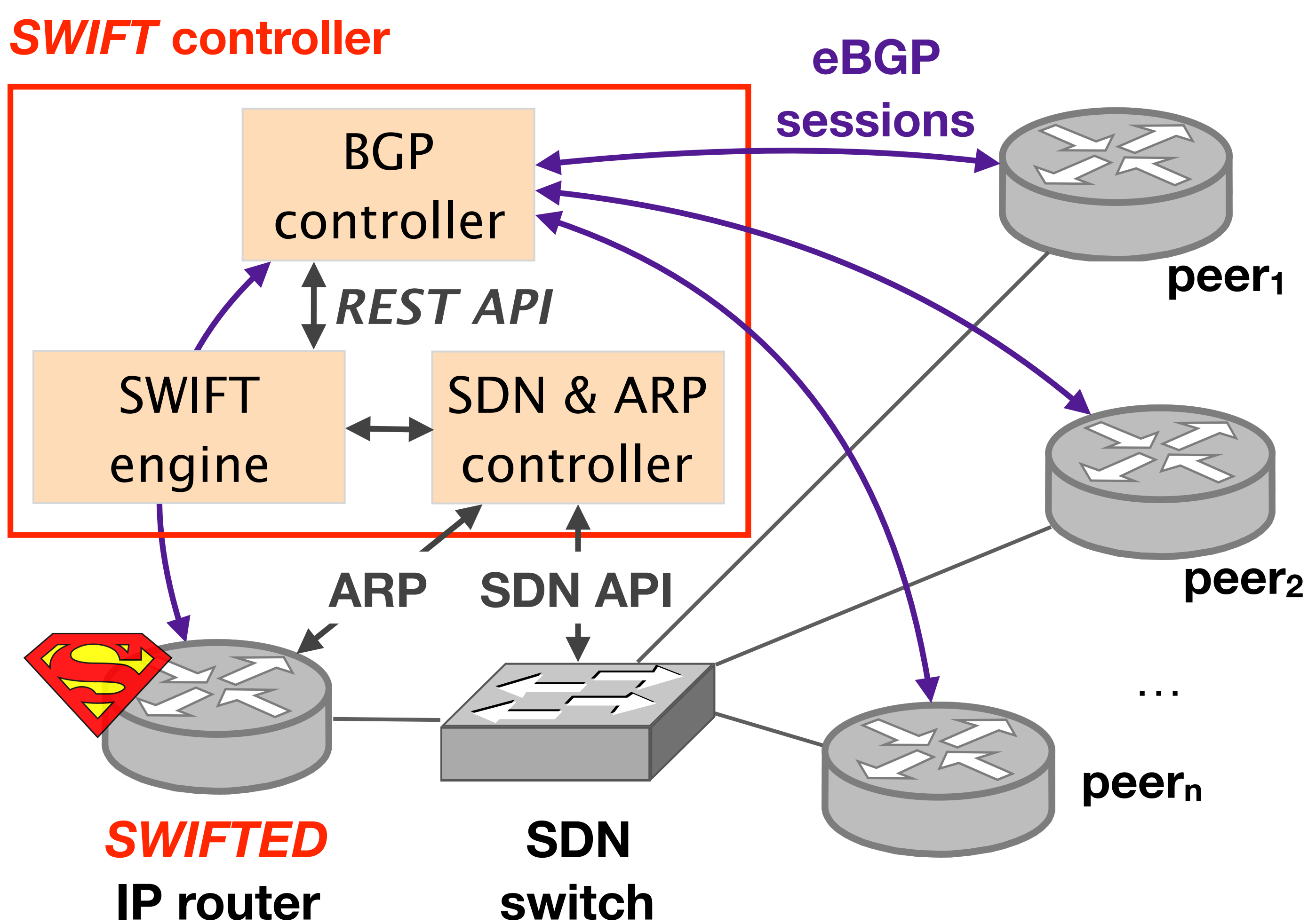
groups of entries

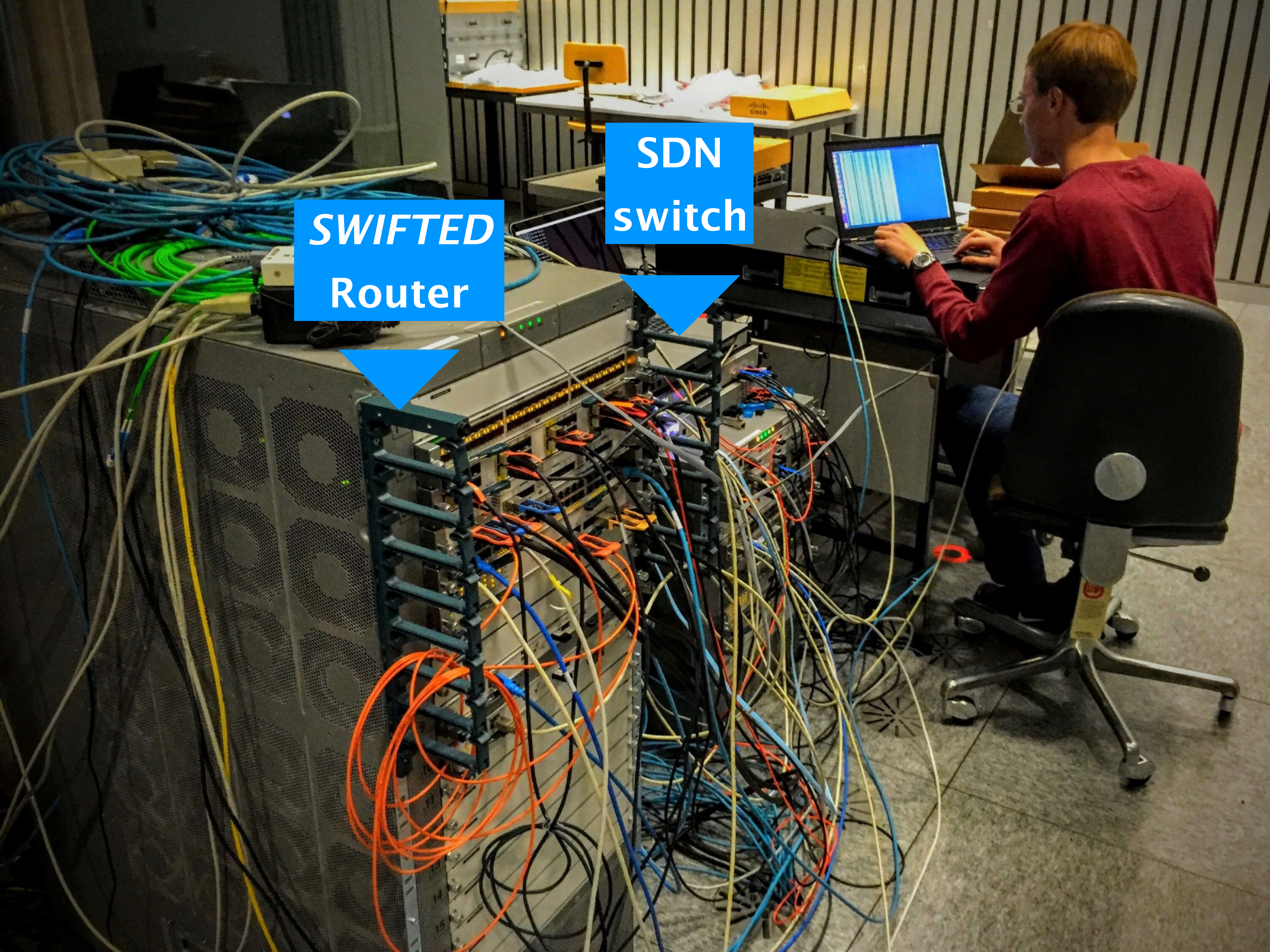
3 Supercharging

existing systems

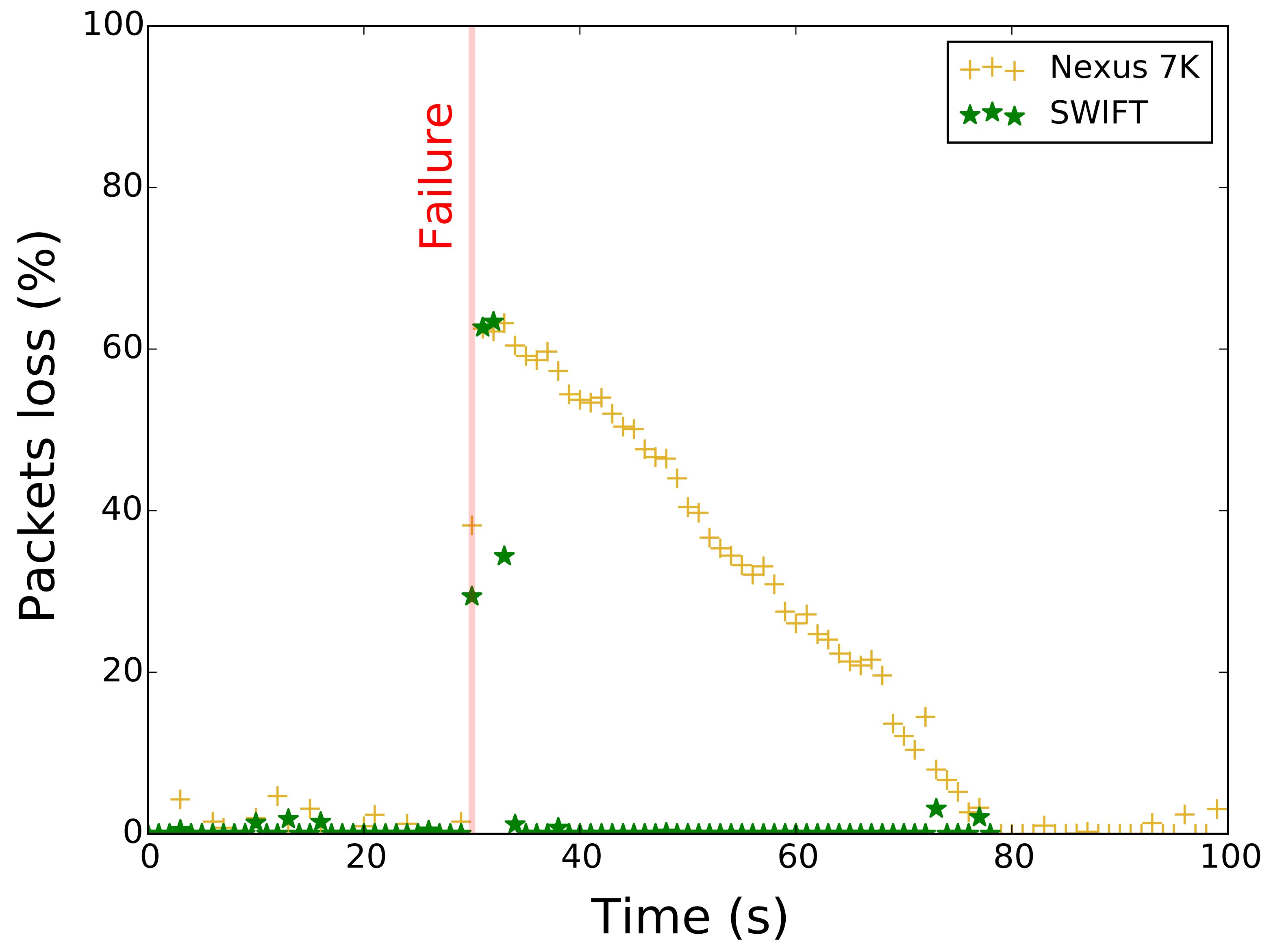
We implemented a full SWIFT prototype which

can boost existing routers convergence performance




SDN
switch
SWIFTED
Router

SWIFT reduces the convergence time of a Cisco Nexus 7k from 55s to maximum 3s (i.e., 95% decrease)



SWIFT

Predictive Fast Reroute upon Remote BGP Disruptions

Laurent Vanbever

www.vanbever.eu

Munich Internet Research Retreat

November 25 2016