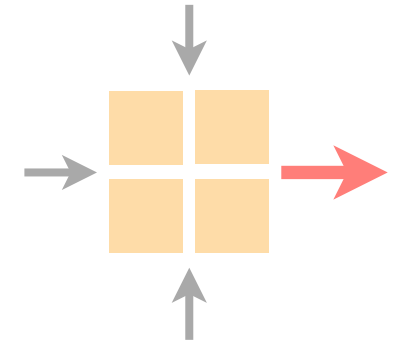


Communication Networks

Spring 2019



Laurent Vanbever

nsg.ee.ethz.ch

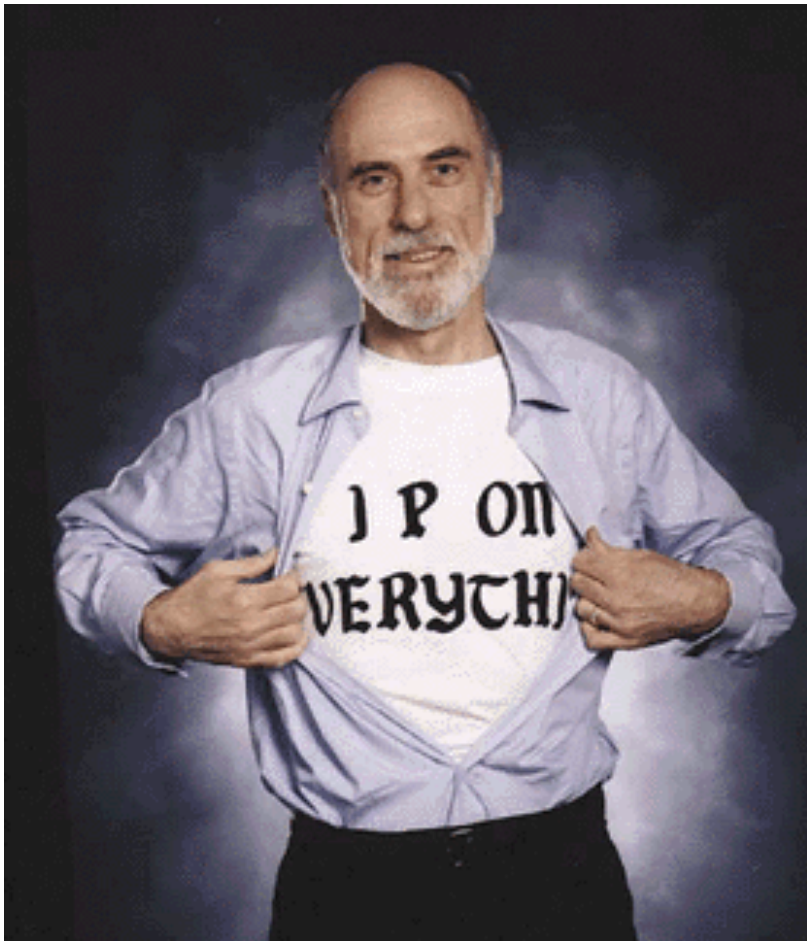
ETH Zürich (D-ITET)

March 25 2019

Materials inspired from Scott Shenker & Jennifer Rexford

Last week on
Communication Networks

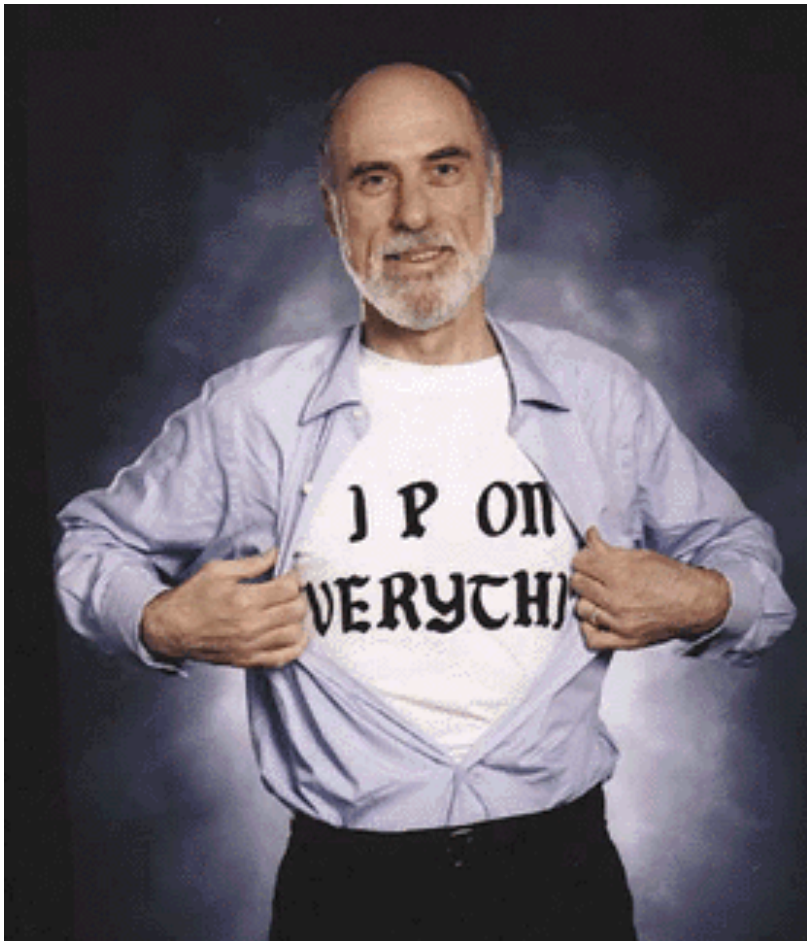
Internet Protocol and Forwarding



source: Boardwatch Magazine

- 1 **IP addresses**
use, structure, allocation
- 2 **IP forwarding**
longest prefix match rule
- 3 **IP header**
IPv4 and IPv6, wire format

Internet Protocol and Forwarding



1 IP addresses use, structure, allocation

IP forwarding

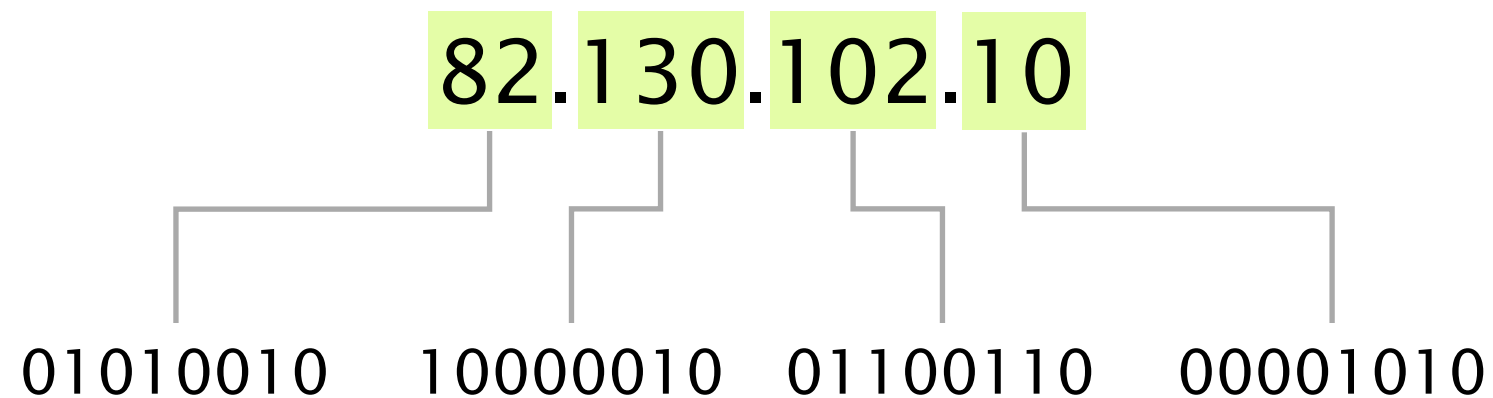
longest prefix match rule

IP header

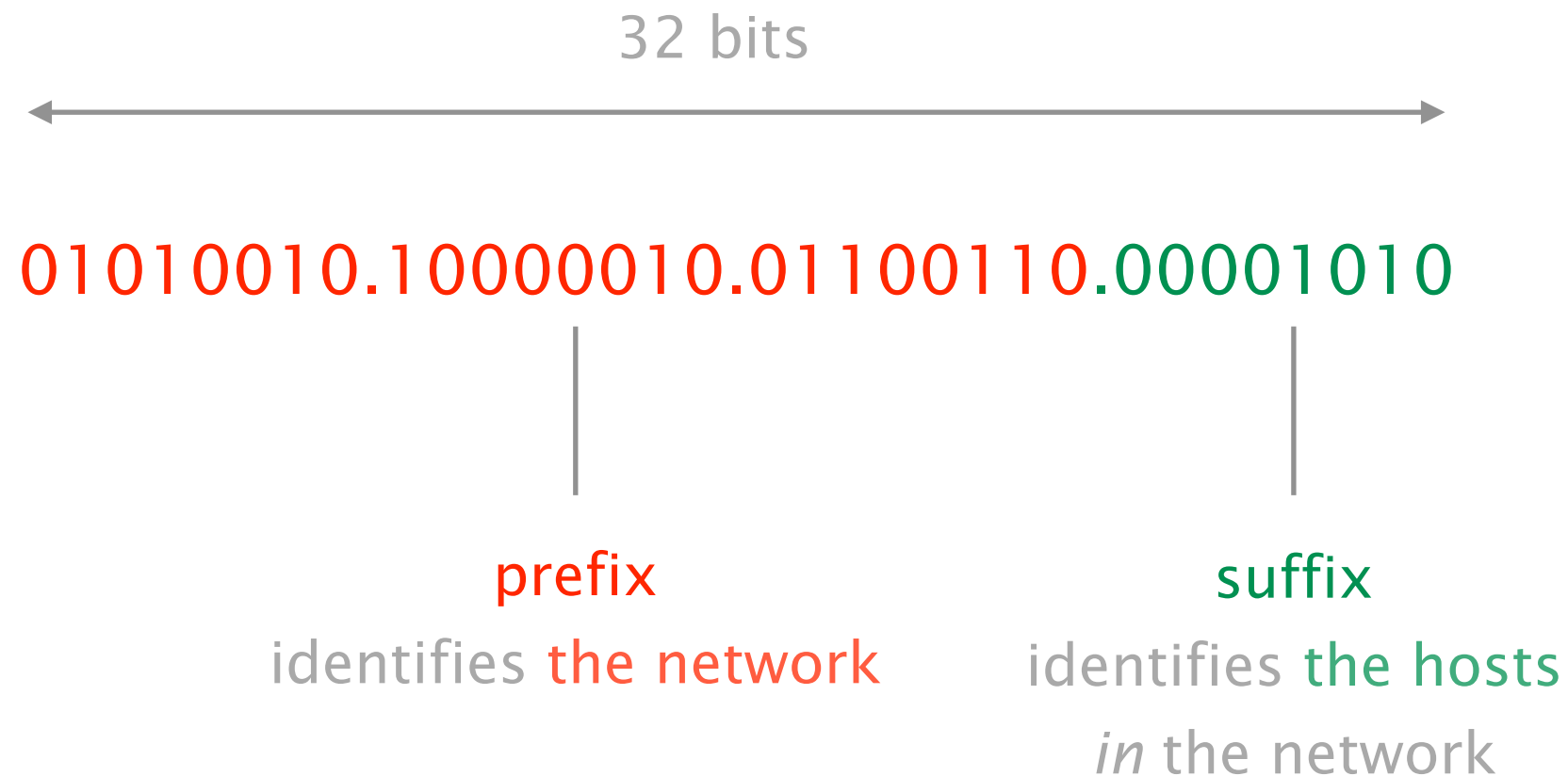
IPv4 and IPv6, wire format

IPv4 addresses are unique 32-bits number
associated to a network interface (on a host, a router, ...)

IP addresses are usually written
using dotted-quad notation



IP addressing is hierarchical, composed of a prefix (network address) and a suffix (host address)



Each prefix has a given length,
usually written using a “slash notation”

IP prefix 82.130.102.0 /24

 |

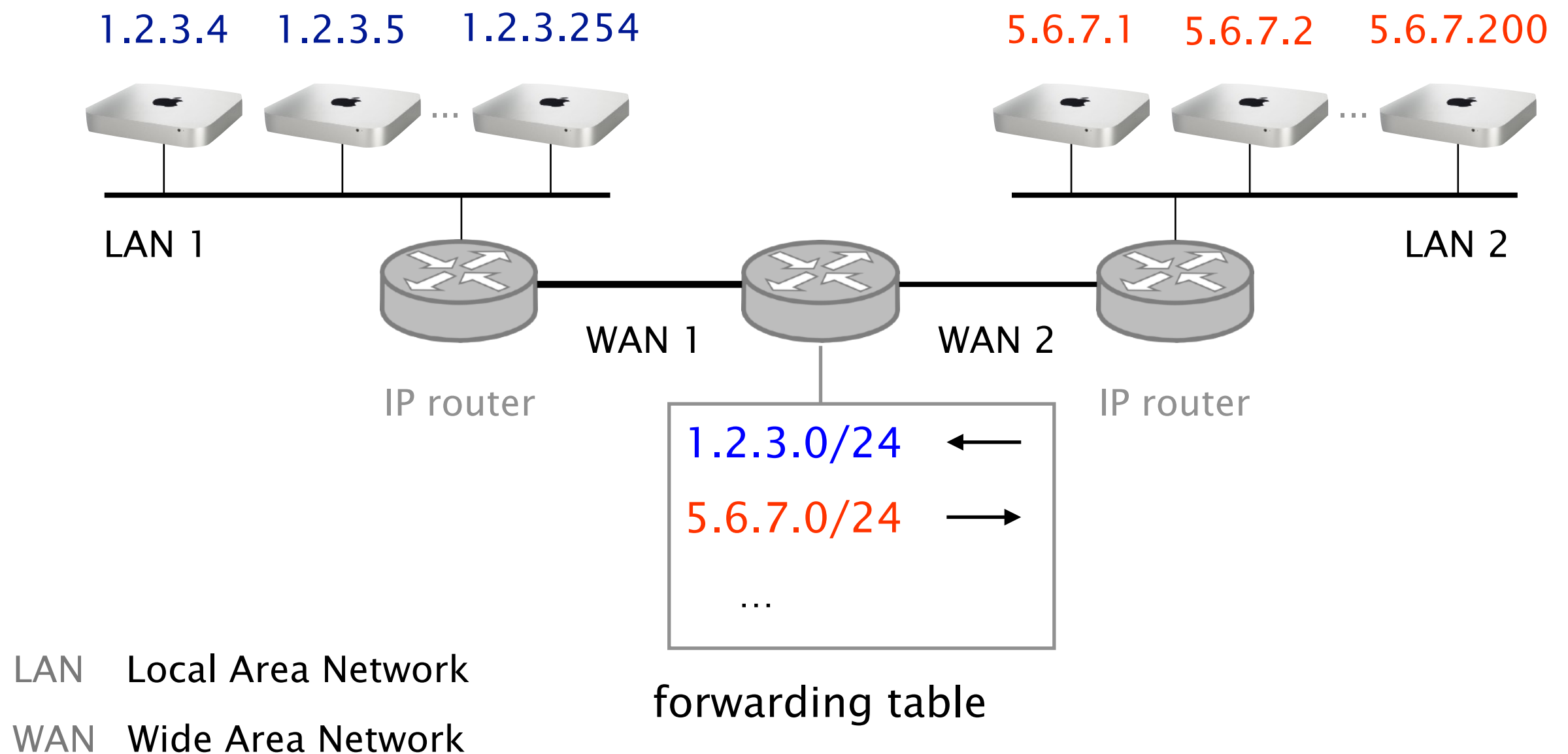
 prefix length (in bits)

Prefixes are also sometimes specified
using an address and a mask

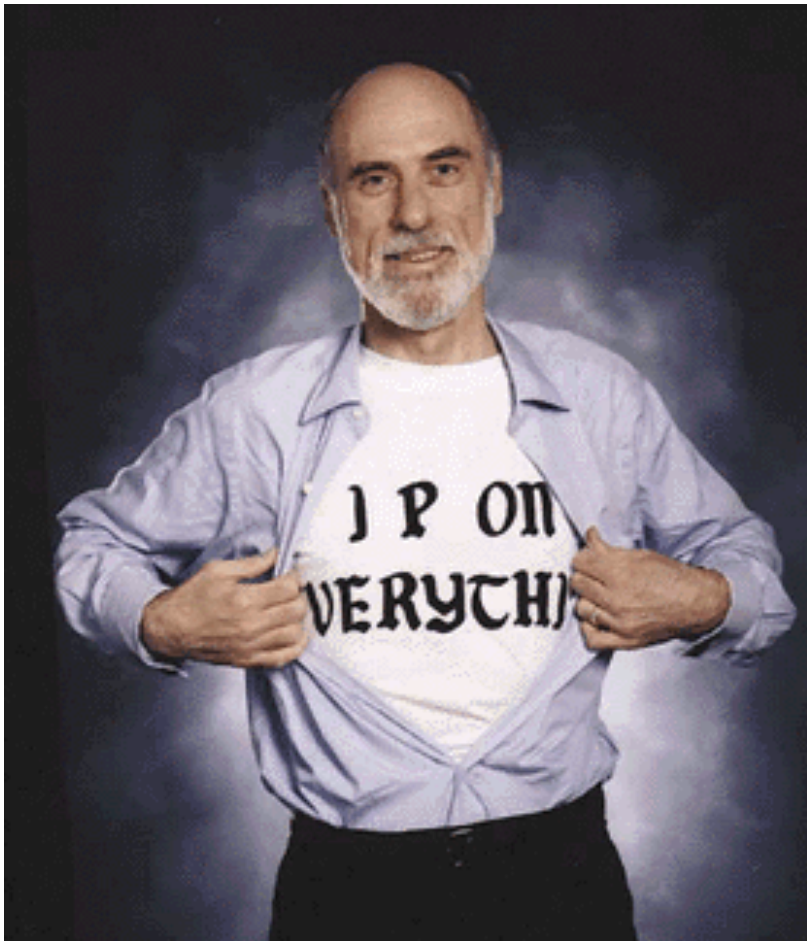
Address	82.130.102.0
	01010010.10000010.01100110. 00000000
	11111111.11111111.11111111. 00000000
Mask	255.255.255.0

Routers forward packet to their destination
according to the network part, *not* the host part

Doing so enables to scale the forwarding tables



Internet Protocol and Forwarding



IP addresses

use, structure, allocation

2

IP forwarding

longest prefix match rule

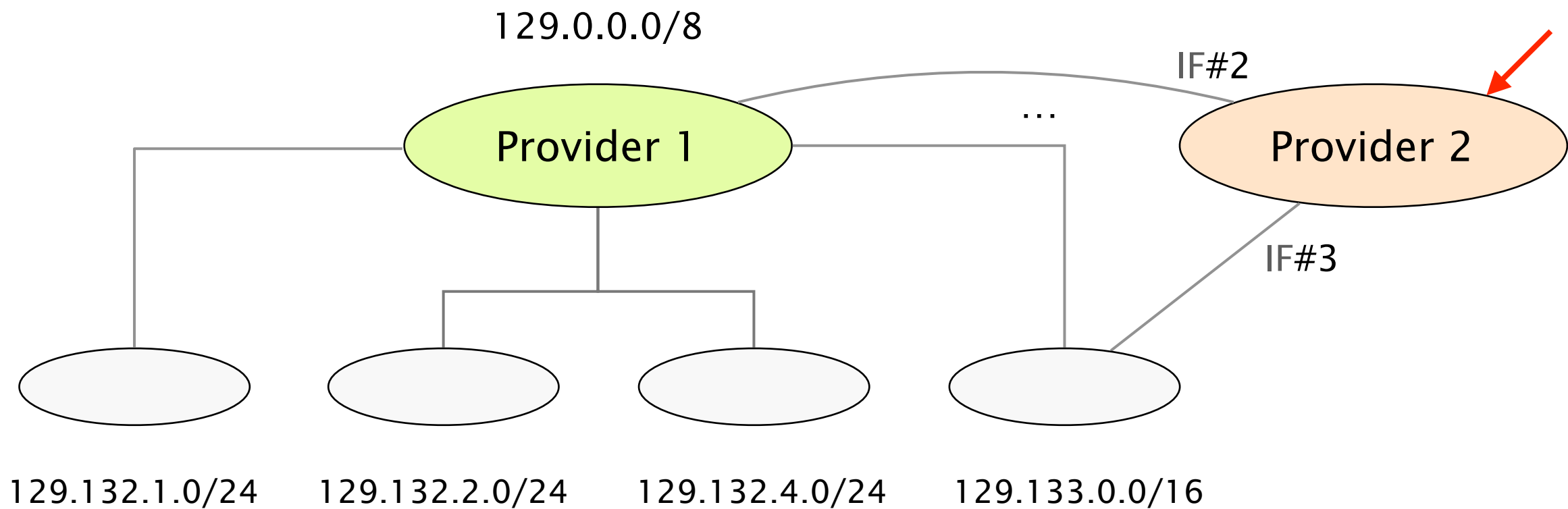
IP header

IPv4 and IPv6, wire format

Routers maintain forwarding entries
for each Internet prefix

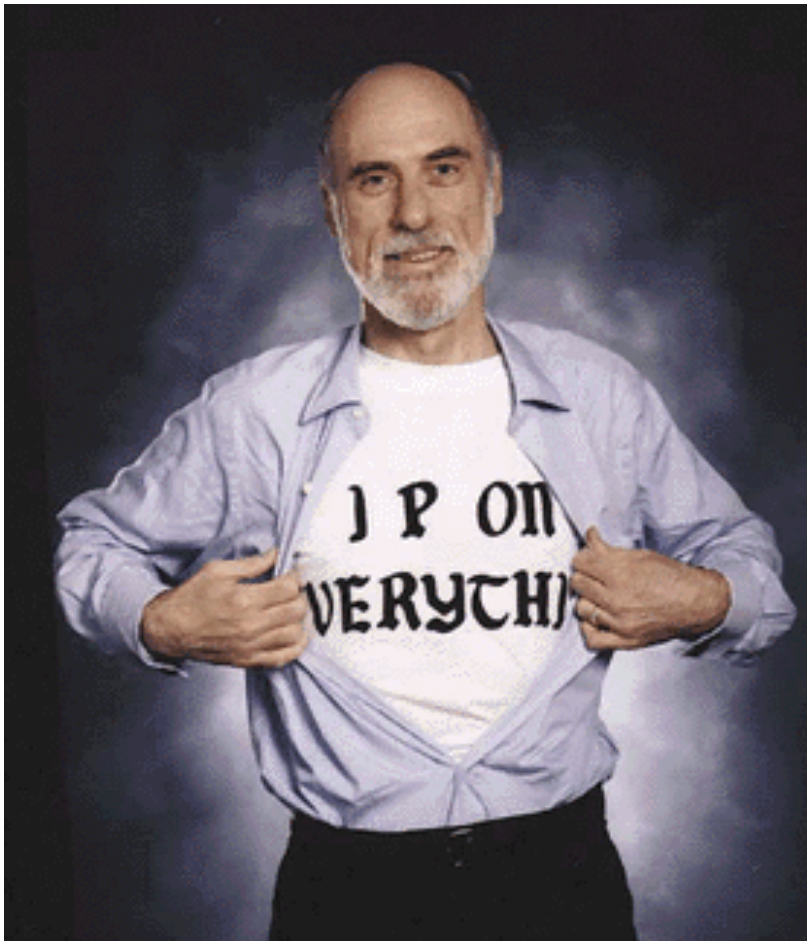
Provider 2's Forwarding table

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



To resolve ambiguity, forwarding is done along the *most specific* prefix (*i.e.*, the longer one)

Internet Protocol and Forwarding



IP addresses

use, structure, allocation

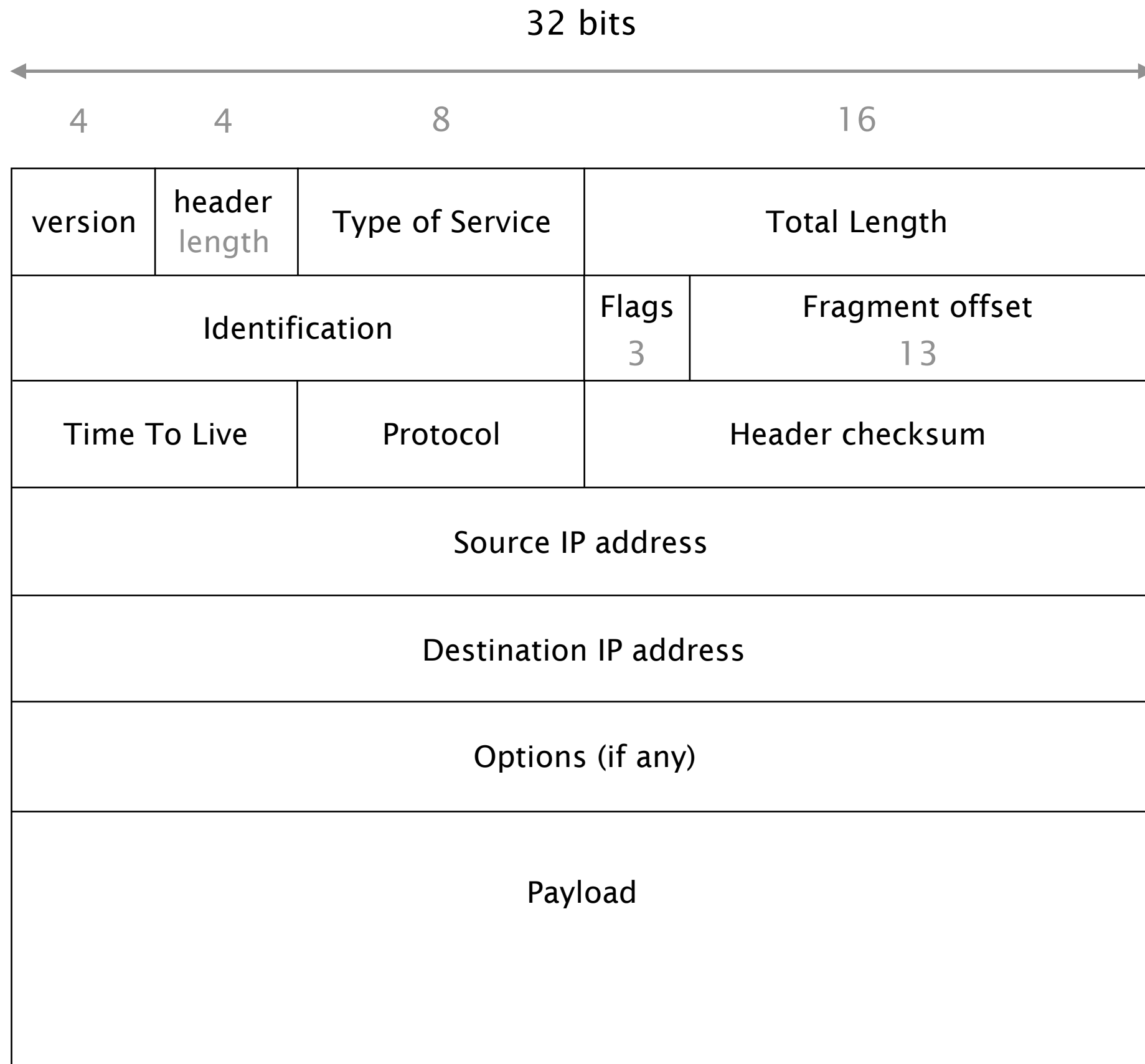
IP forwarding

longest prefix match rule

3

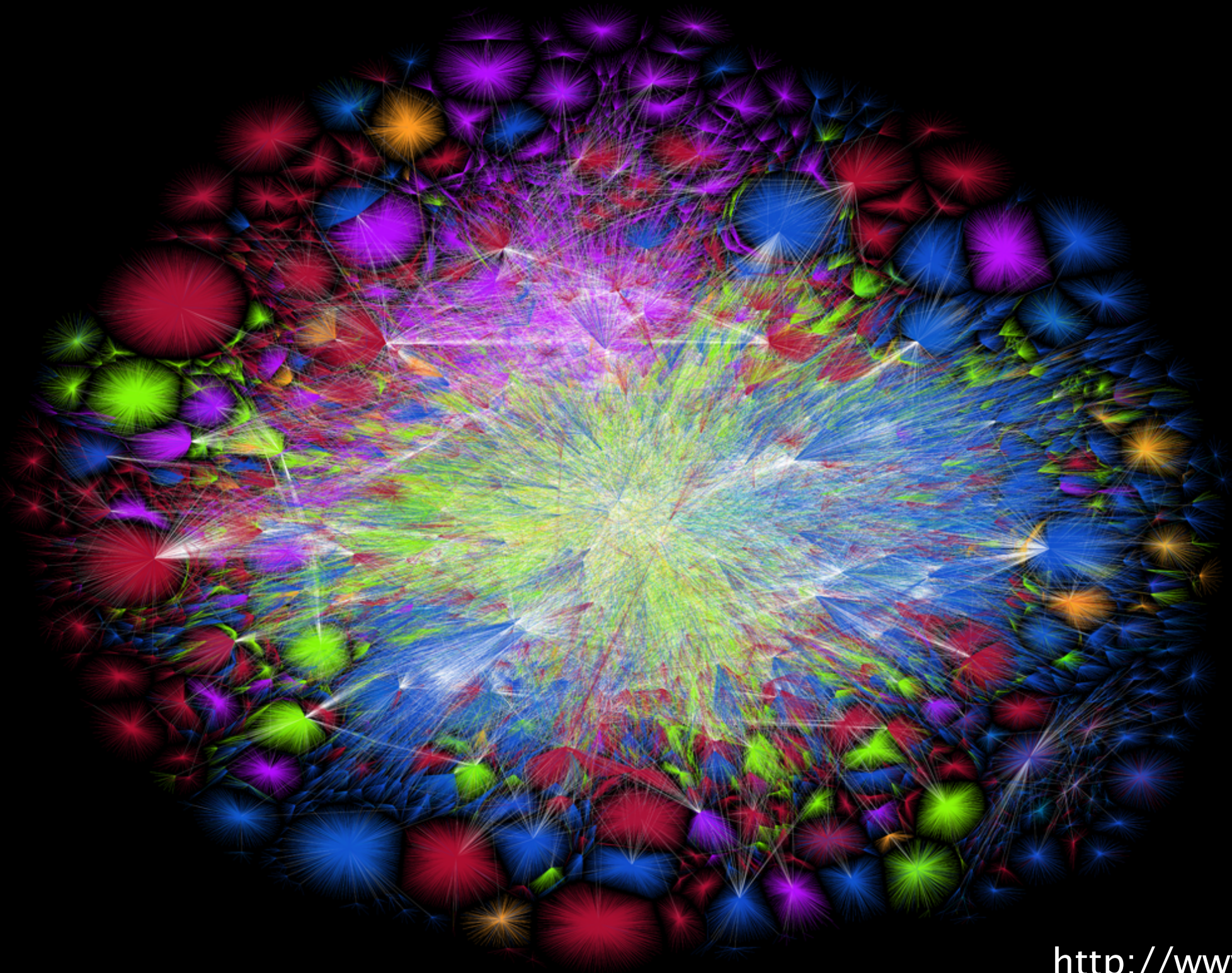
IP header

IPv4 and IPv6, wire format



This week on
Communication Networks

Internet routing




› traceroute www.google.ch

› traceroute www.google.ch

```
1  rou-etx-1-ee-tik-etx-dock-1 (82.130.102.1)
2  rou-ref-rz-bb-ref-rz-etx (10.10.0.41)
3  rou-fw-rz-ee-tik (10.1.11.129)
4  rou-fw-rz-gw-rz (192.33.92.170)
5  swiix1-10ge-1-4.switch.ch (130.59.36.41)
6  swiez2 (192.33.92.11)
7  swiix2-p1.switch.ch (130.59.36.250)
8  equinix-zurich.net.google.com (194.42.48.58)
9  66.249.94.157 (66.249.94.157)
10 zrh04s06-in-f24.1e100.net (173.194.40.88)
```


Internet routing comes into two flavors:
intra- and *inter-domain* routing



inter-domain
routing

Find paths between networks

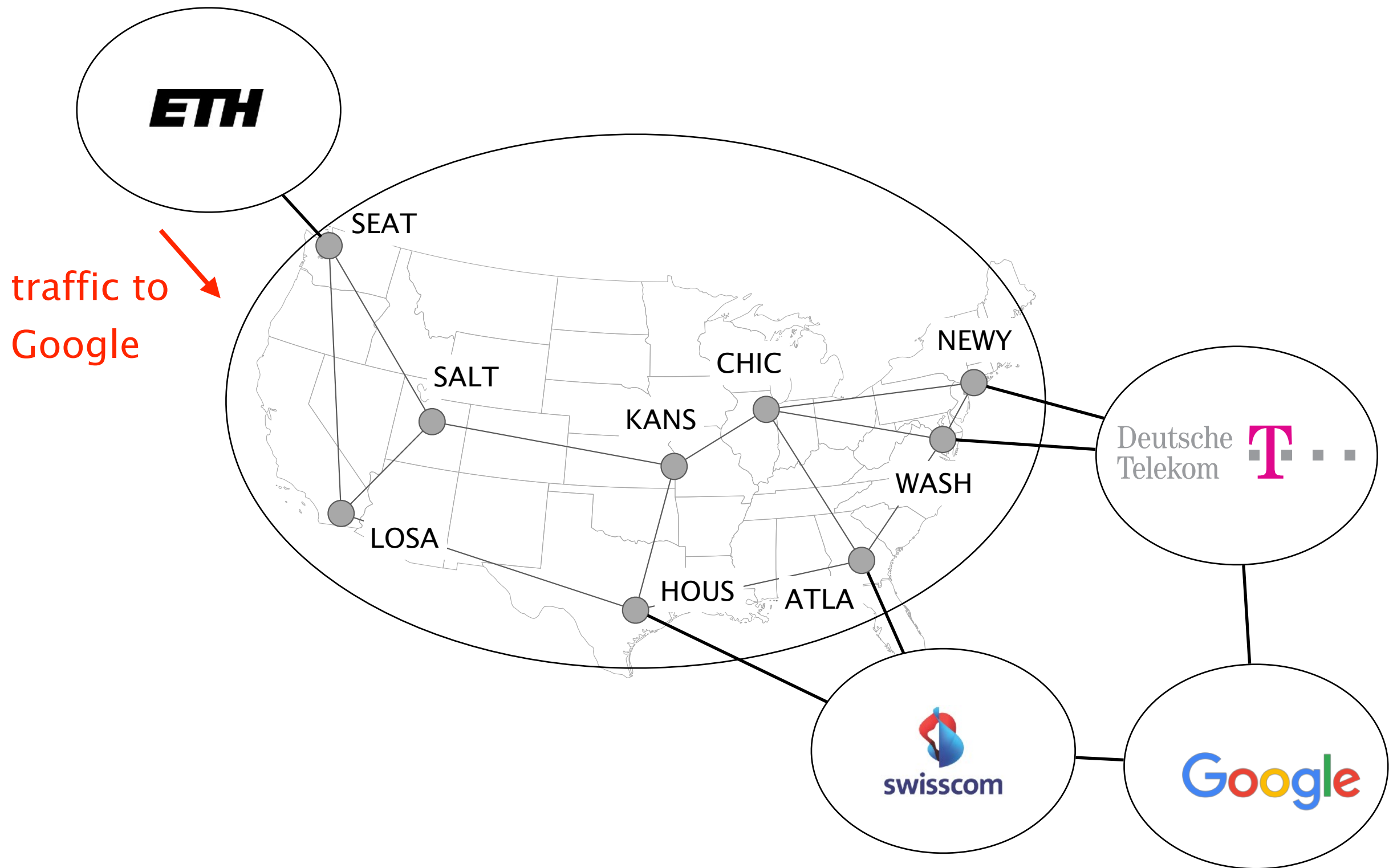
intra-domain
routing

Find paths within a network

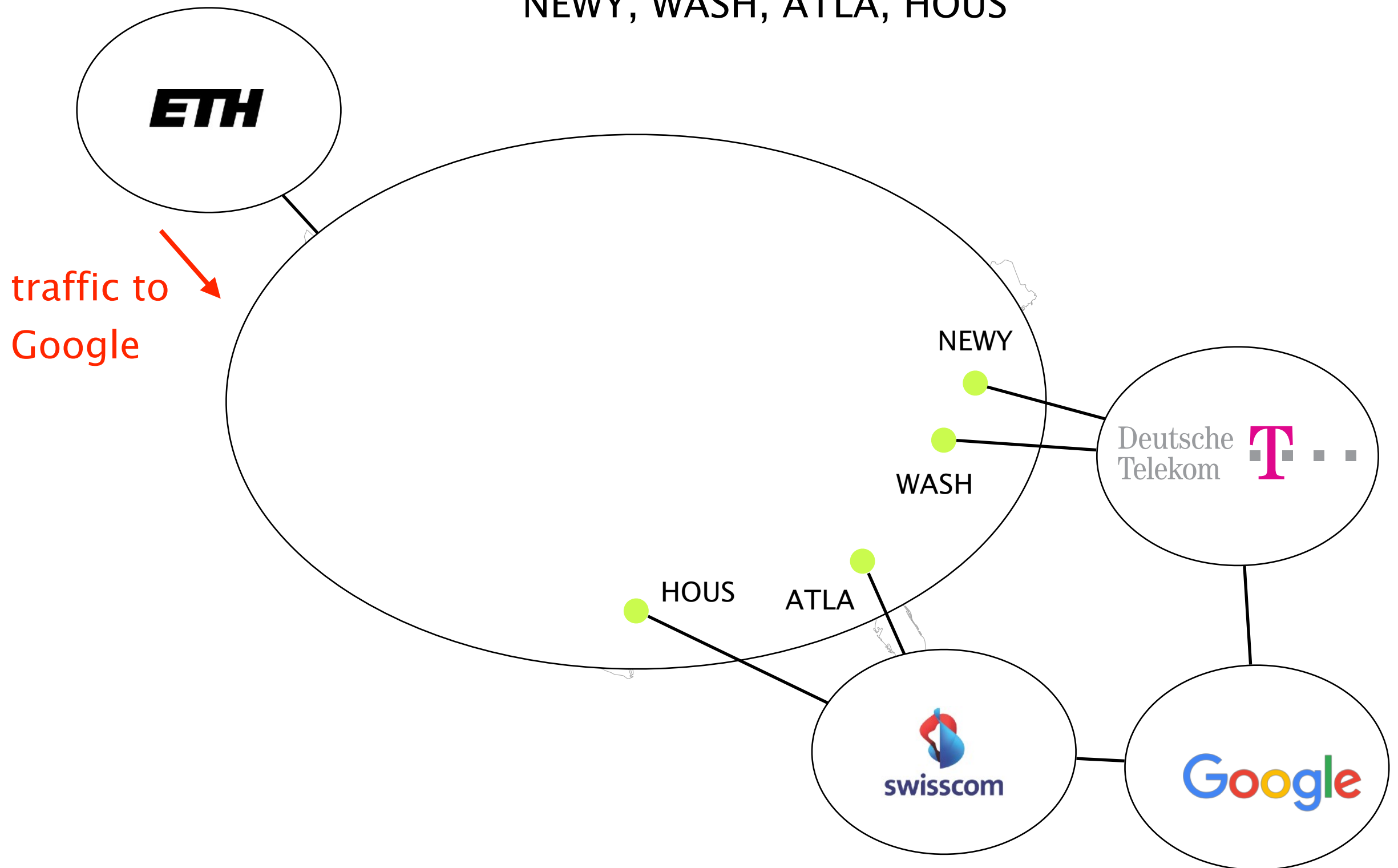
inter-domain
routing

intra-domain
routing

Find paths **between** networks



Google can be reached via
NEWY, WASH, ATLA, HOUS



Google can be reached via
NEWY, WASH, ATLA, HOUS

best exit point

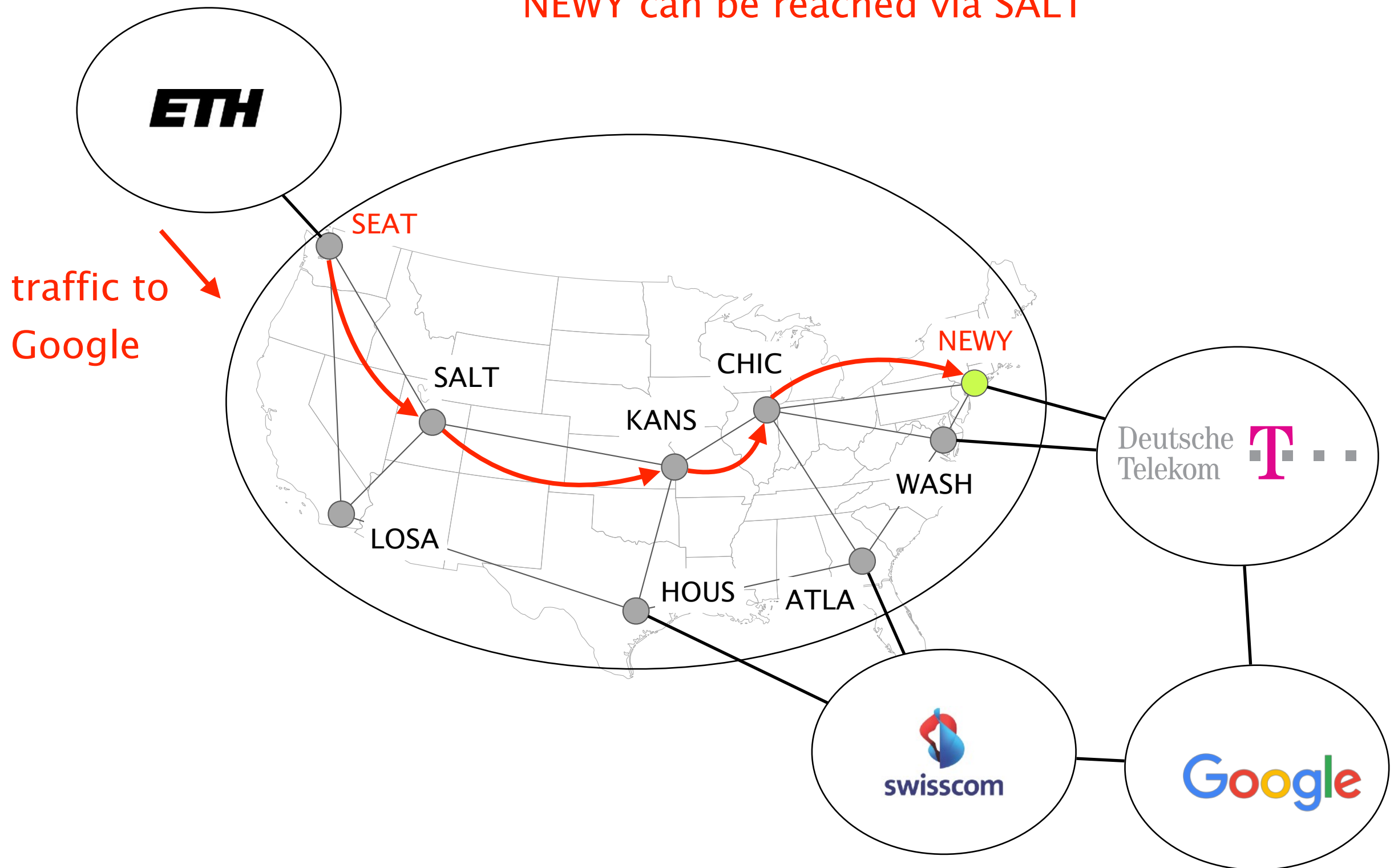
based on money, performance, ...

inter-domain
routing

intra-domain
routing

Find paths **within** a network

NEWY can be reached via SALT



› traceroute **www.google.ch**

rou-etx-1-ee-tik-etx-dock-1

rou-ref-rz-bb-ref-rz-etx

rou-fw-rz-ee-tik

rou-fw-rz-gw-rz

swiix1-10ge-1-4.switch.ch

swiez2

swiix2-p1.switch.ch

equinix-zurich.net.google.com

66.249.94.157

zrh04s06-in-f24.1e100.net

intra-domain routing

intra-domain routing

intra-domain routing

› traceroute **www.google.ch**

rou-etx-1-ee-tik-etx-dock-1

rou-ref-rz-bb-ref-rz-etx

rou-fw-rz-ee-tik

rou-fw-rz-gw-rz

swiix1-10ge-1-4.switch.ch

swiez2

swiix2-p1.switch.ch

equinix-zurich.net.google.com

66.249.94.157

zrh04s06-in-f24.1e100.net

inter-domain routing

inter-domain routing

Internet routing

from here to there, and back



- 1 **Intra-domain routing**

Link-state protocols
Distance-vector protocols

- 2 **Inter-domain routing**

Path-vector protocols

Internet routing

from here to there, and back



1

Intra-domain routing

Link-state protocols

Distance-vector protocols

Inter-domain routing

Path-vector protocols

Intra-domain routing enables routers to compute forwarding paths to any internal subnet

what kind of paths?

Network operators don't want arbitrary paths,
they want **good paths**

definition

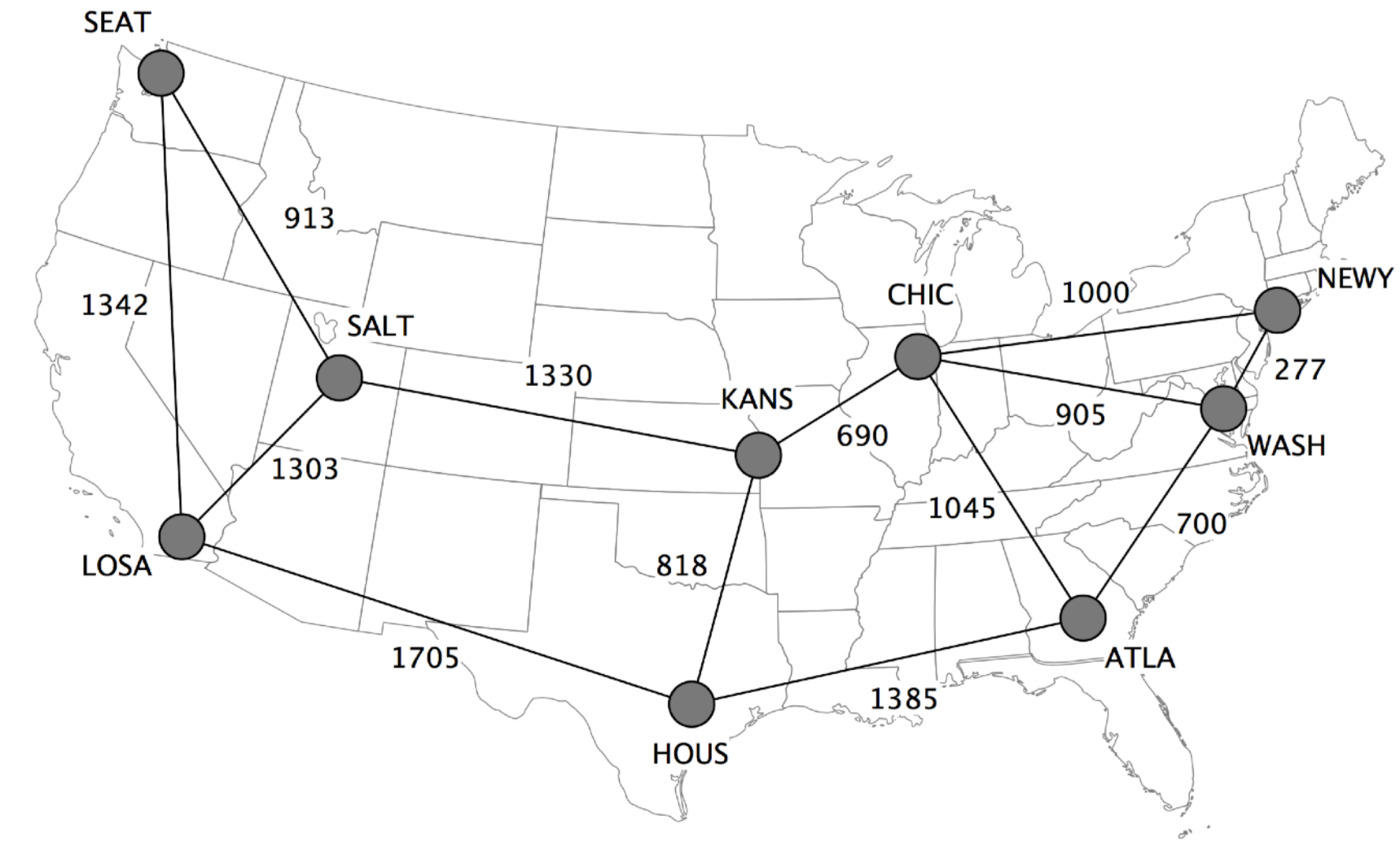
A good path is a path that
minimizes some network-wide metric

typically delay, load, loss, cost

approach

Assign to each link a weight (usually static),
compute the *shortest-path* to each destination

When weights are assigned **proportionally** to the distance, shortest-paths will minimize the end-to-end delay



Internet2, the US based research network

When weights are assigned proportionally to the distance, shortest-paths will minimize the end-to-end delay

if traffic is such that
there is no congestion

When weights are assigned **inversely proportionally** to each link capacity, **throughput is maximized**

if traffic is such that
there is no congestion

Internet routing

from here to there, and back



1

Intra-domain routing

Link-state protocols

Distance-vector protocols

Inter-domain routing

Path-vector protocols

In Link-State routing, routers build a precise map of the network by flooding local views to everyone

Each router keeps track of its incident links and cost as well as whether it is up or down

Each router broadcast its own links state to give every router a complete view of the graph

Routers run Dijkstra on the corresponding graph to compute their shortest-paths and forwarding tables

Flooding is performed as in L2 learning

Node sends its link-state
on all its links

Next node does the same,
except on the one where
the information arrived

Flooding is performed as in L2 learning,
except that it is reliable

Node sends its link-state
on all its links

Next node does the same,
except on the one where
the information arrived

All nodes are **ensured** to
receive the *latest version*
of all link-states

challenges

packet loss

out of order arrival

Flooding is performed as in L2 learning,
except that it is reliable

Node sends its link-state
on all its links

Next node does the same,
except on the one where
the information arrived

All nodes are **ensured** to
receive the *latest version*
of all link-states

solutions

ACK & retransmissions

sequence number

time-to-live for each link-state

A link-state node initiate flooding in 3 conditions

Topology change

link or node failure/recovery

Configuration change

link cost change

Periodically

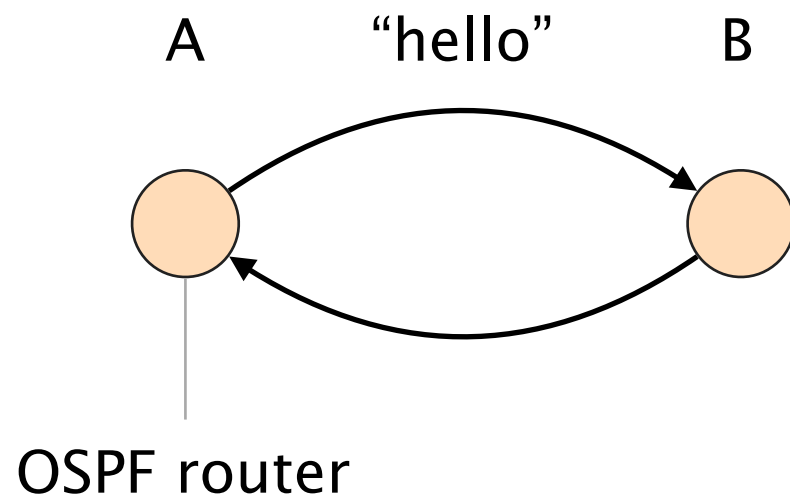
refresh the link-state information

every (say) 30 minutes

account for possible data corruption

Once a node knows the entire topology,
it can compute shortest-paths using Dijkstra's algorithm

By default, Link-State protocols detect topology changes using software-based beaconing



Routers periodically exchange “Hello” in both directions (*e.g.* every 30s)

Trigger a failure after few missed “Hellos” (*e.g.*, after 3 missed ones)

Tradeoffs between:

- detection speed
- bandwidth and CPU overhead
- false positive/negatives

During network changes,
the link-state database of each node might differ

control-plane
consistency

all nodes have the
same link-state database

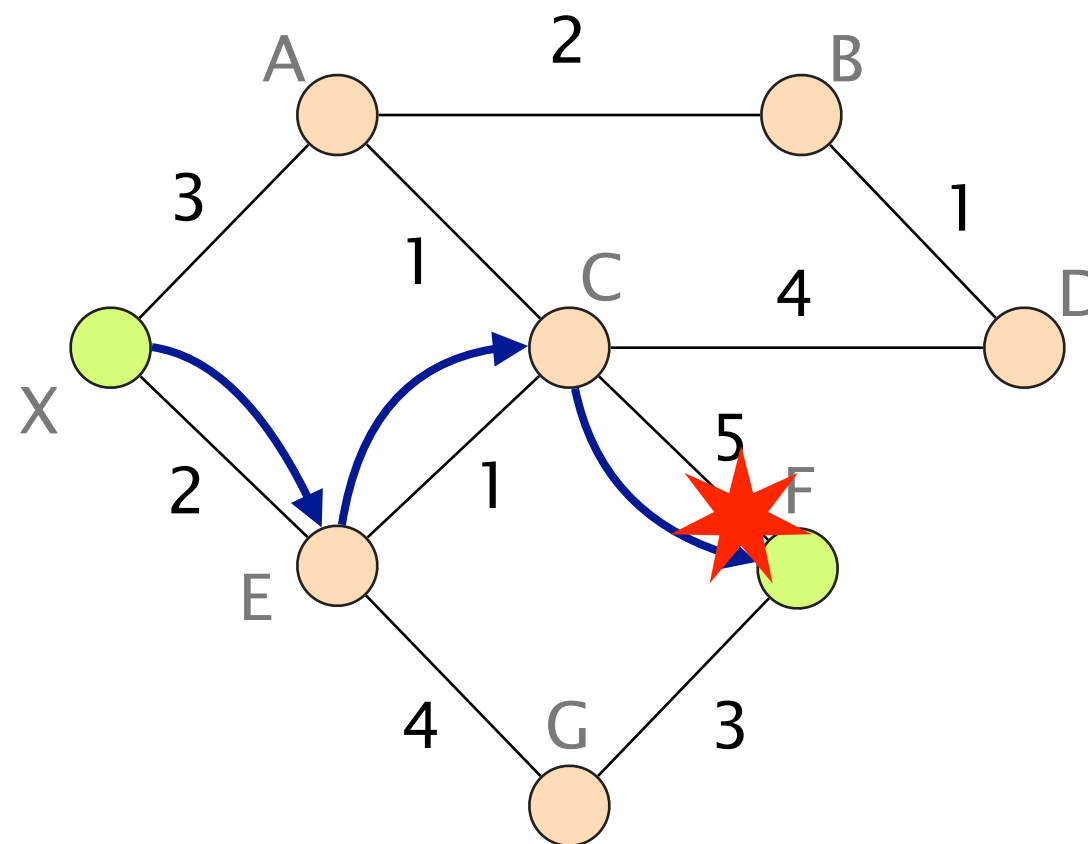
necessary

forwarding
validity

the global forwarding state
directs packet to its destination

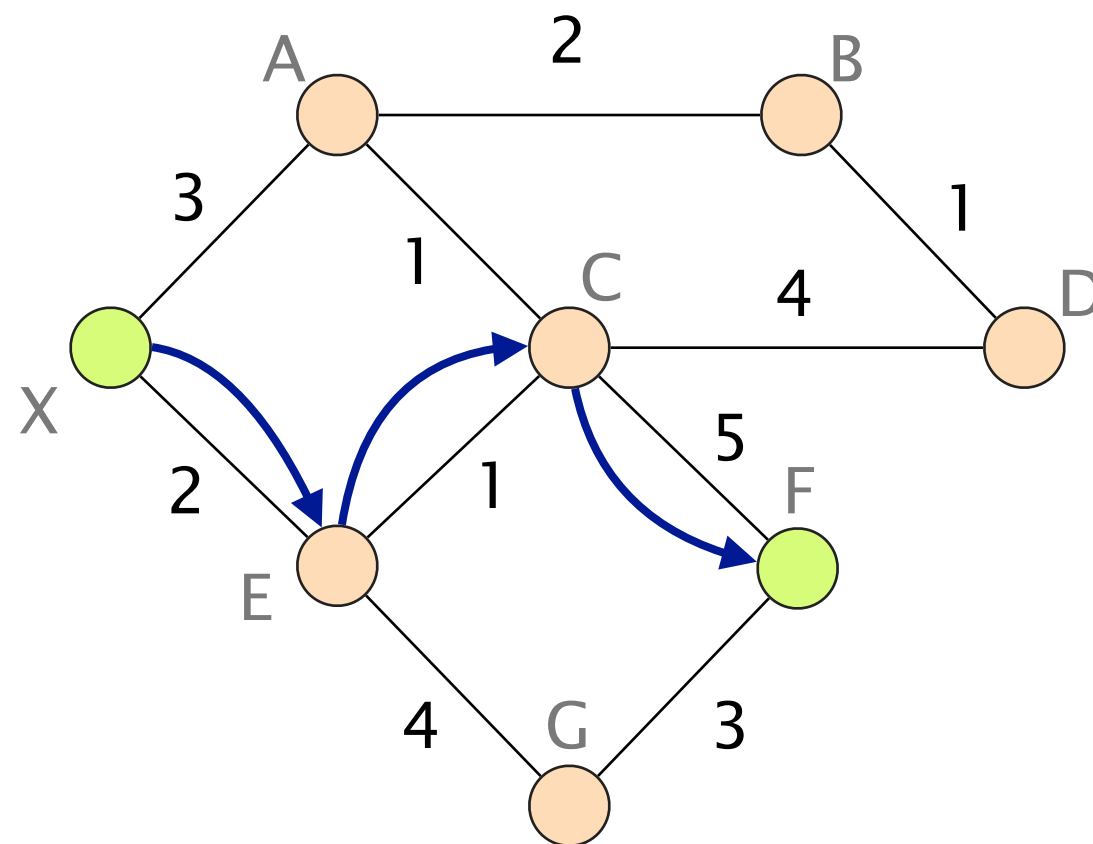
Inconsistencies lead to transient disruptions
in the form of blackholes or forwarding loops

Blackholes appear due to detection delay,
as nodes do not immediately detect failure

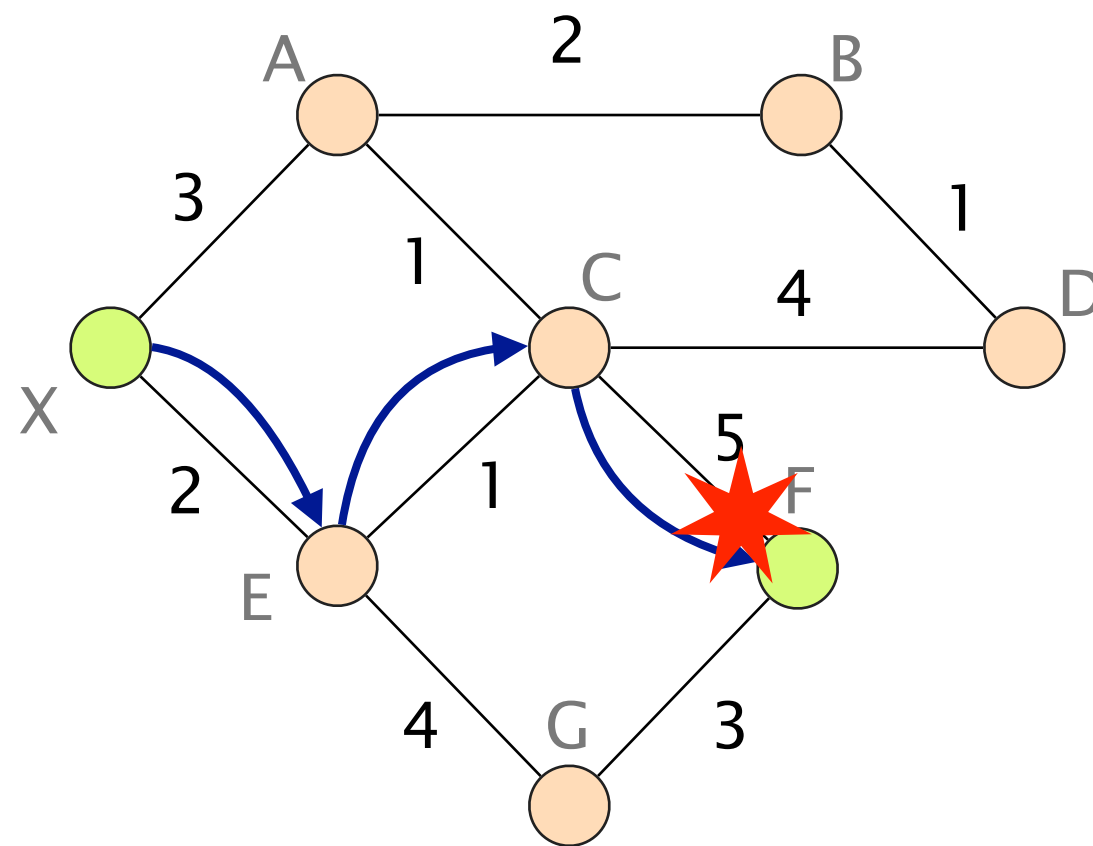


depends on the timeout for detecting lost hellos

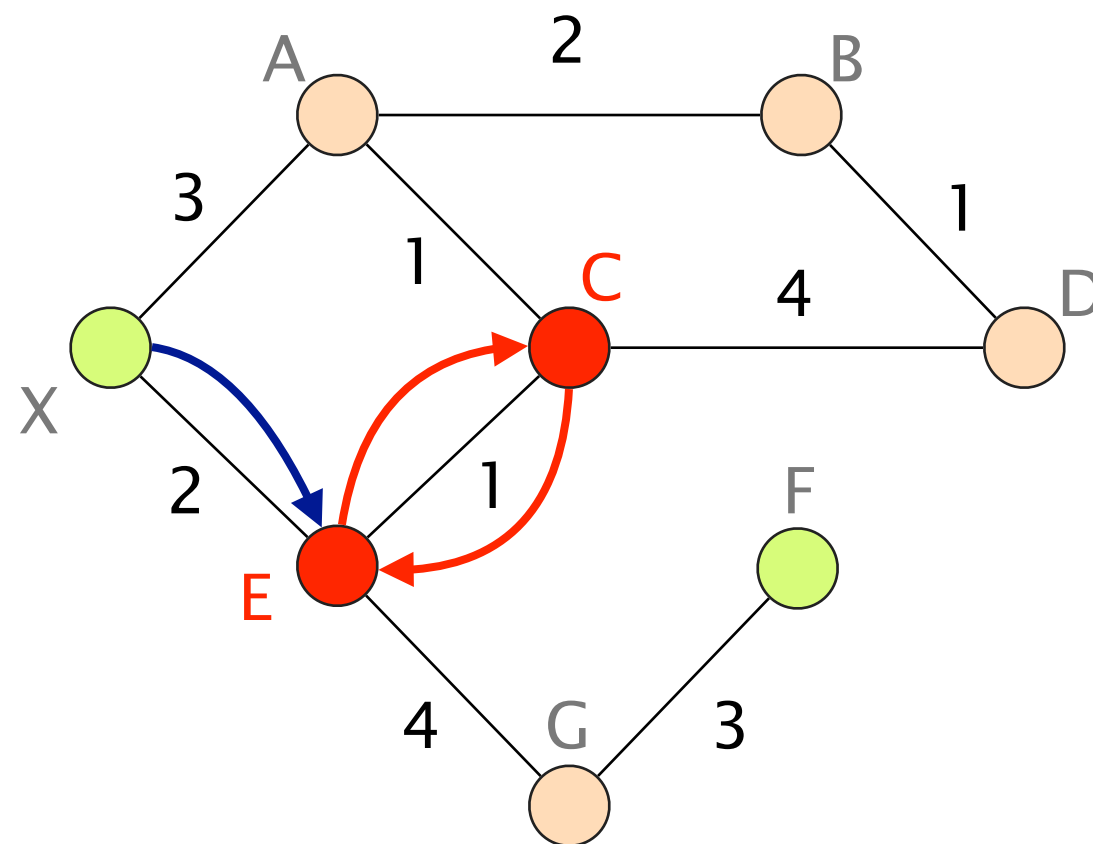
Transient loops appear due to
inconsistent link-state databases



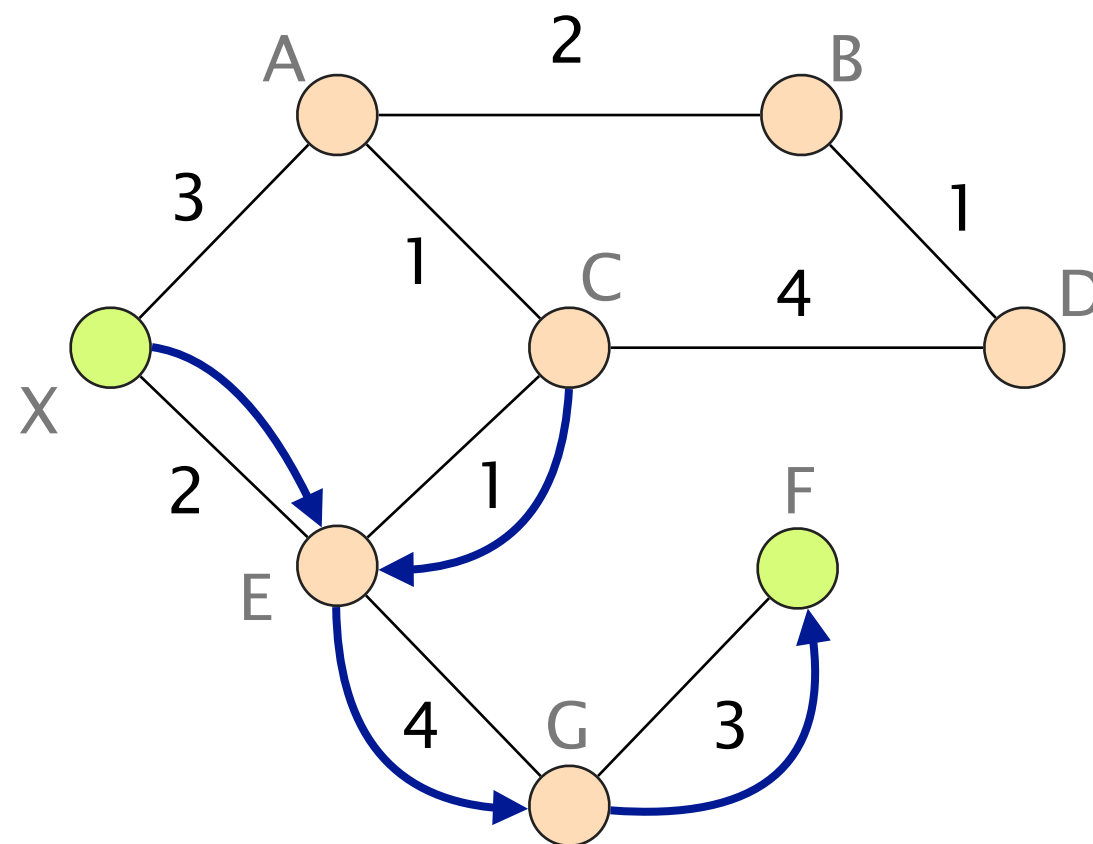
Initial forwarding state



C learns about the failure
and immediately reroute to E



A loop appears as E
isn't yet aware of the failure



The loop disappears as soon as
E updates its forwarding table

Convergence is the process during which the routers seek to actively regain a consistent view of the network

Network convergence time depends on 4 main factors

factors

time the routers take for...

detection

realizing that a link or a neighbor is down

flooding

flooding the news to the entire network

computation

recomputing shortest-paths using Dijkstra

table update

updating their forwarding table

In practice, network convergence time is mostly driven by table updates

	time	improvements
detection	few ms	smaller timers
flooding	few ms	high-priority flooding
computation	few ms	incremental algorithms
table update	potentially, <i>minutes!</i>	better table design

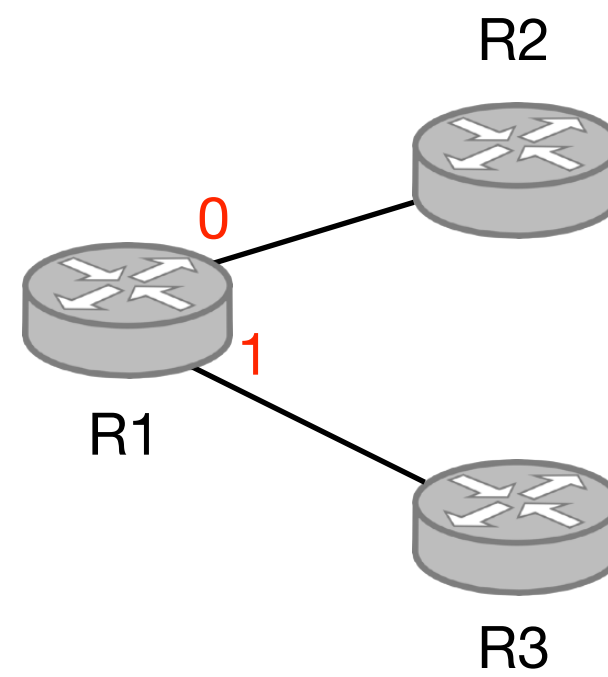
table update

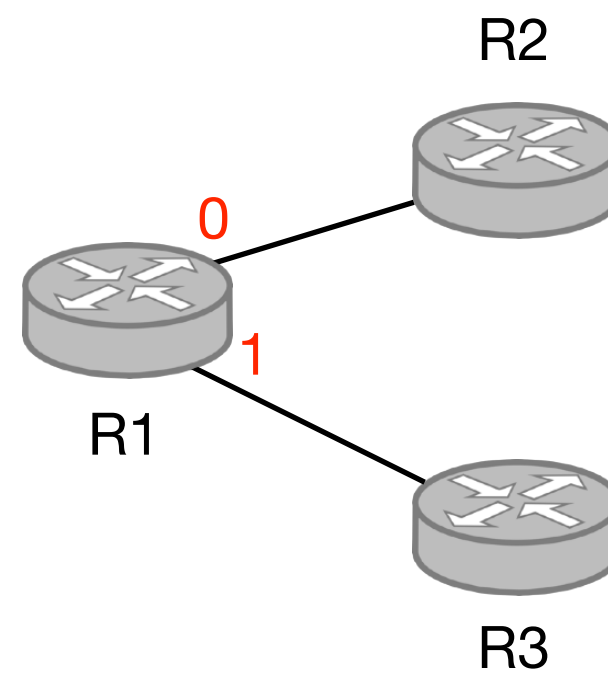
potentially, *minutes!*

better table design



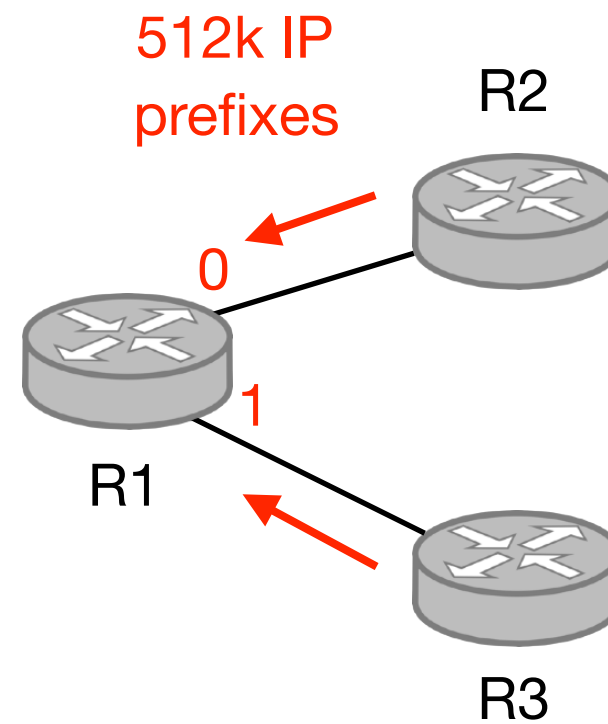
R1





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

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



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

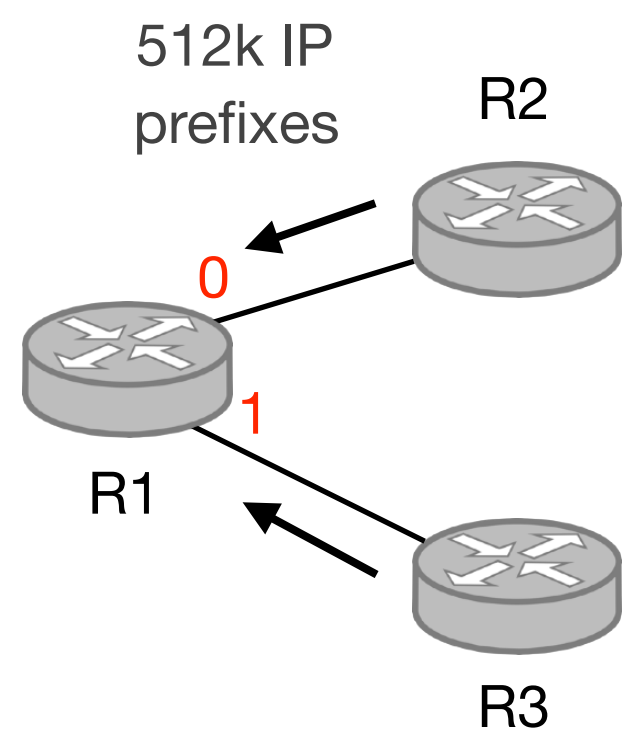
Provider #2 (\$\$)
IP: 198.51.100.2
MAC: 02:bb



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)



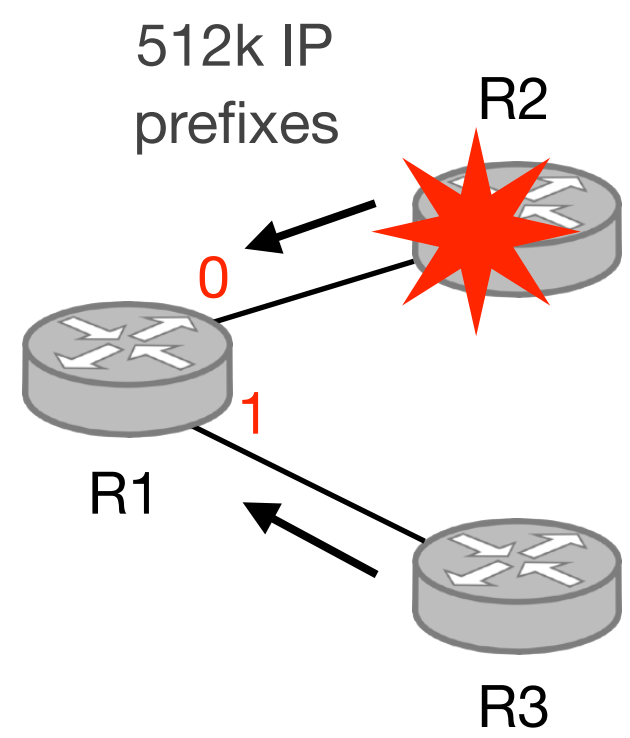
Provider #1 (\$)
IP: 203.0.113.1
MAC: 01:aa

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)



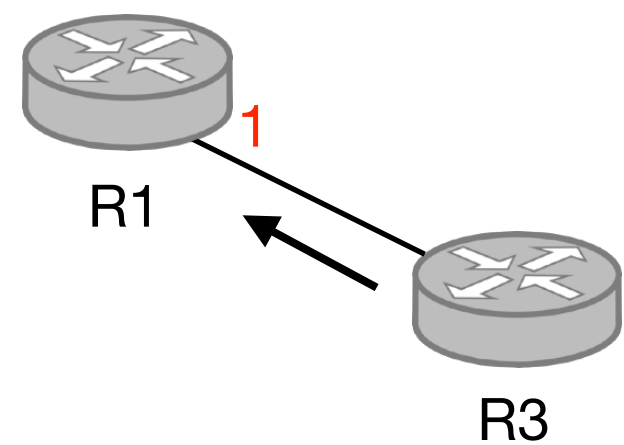
Provider #1 (\$)
IP: 203.0.113.1
MAC: 01:aa

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

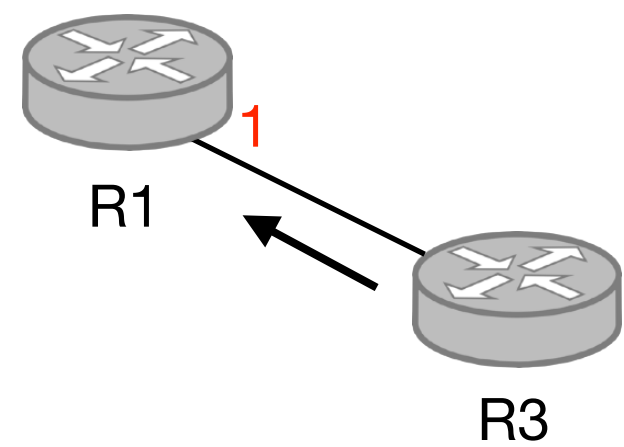
	prefix	Next-Hop
1	1.0.0.0/24	(01:aa, 0)
2	1.0.1.0/16	(01:aa, 0)
...
256k	100.0.0.0/8	(01:aa, 0)
...
512k	200.99.0.0/24	(01:aa, 0)



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

R1's Forwarding Table

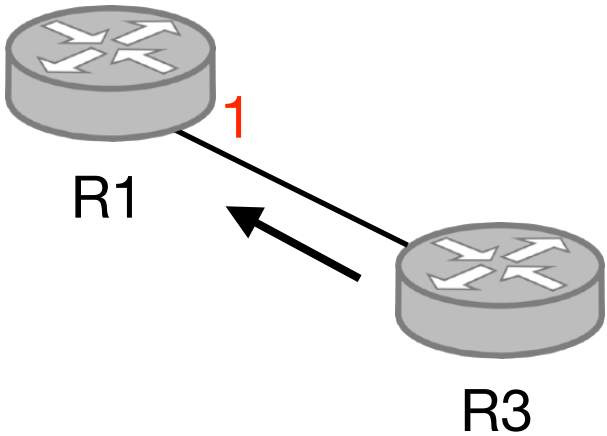
	prefix	Next-Hop
1	1.0.0.0/24	(02:bb, 1)
2	1.0.1.0/16	(01:aa, 0)
...
256k	100.0.0.0/8	(01:aa, 0)
...
512k	200.99.0.0/24	(01:aa, 0)



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

R1's Forwarding Table

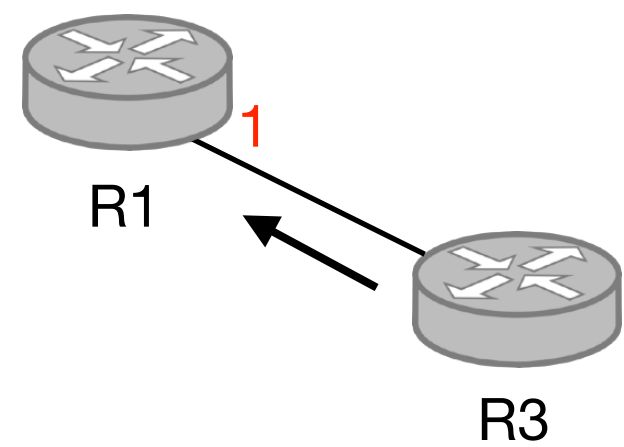
	prefix	Next-Hop
1	1.0.0.0/24	(02:bb, 1)
2	1.0.1.0/16	(02:bb, 1)
...
256k	100.0.0.0/8	(01:aa, 0)
...
512k	200.99.0.0/24	(01:aa, 0)



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

R1's Forwarding Table

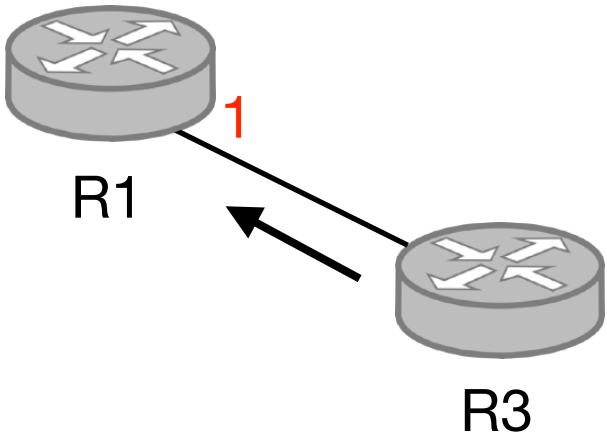
	prefix	Next-Hop
1	1.0.0.0/24	(02:bb, 1)
2	1.0.1.0/16	(02:bb, 1)
...
256k	100.0.0.0/8	(02:bb, 1)
...
512k	200.99.0.0/24	(01:aa, 0)



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

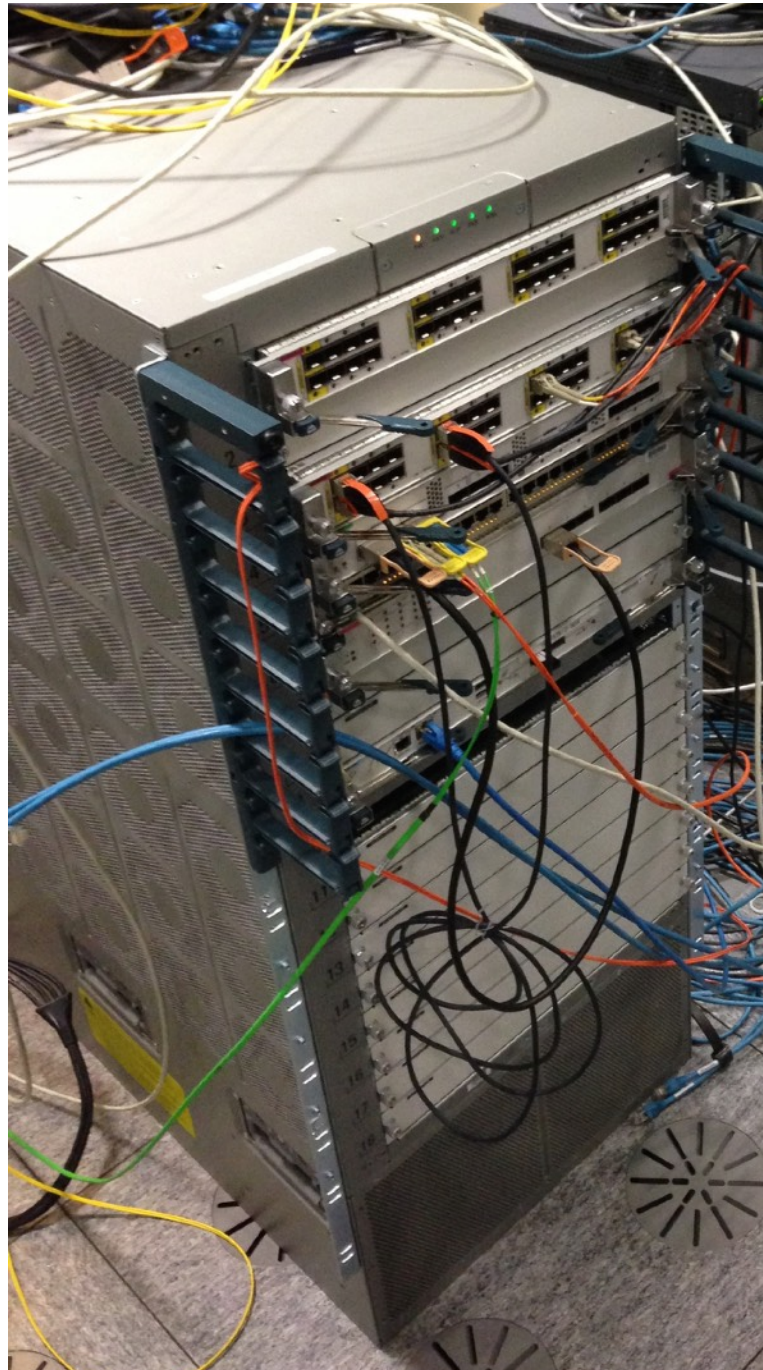
R1's Forwarding Table

	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	(02:bb, 1)



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

How long does it take for ETH routers to converge?



Cisco Nexus 9k

ETH **recent routers**

25 deployed

convergence
time (s)

150

10

1

0.1

1K

5K

10K

50K

100K

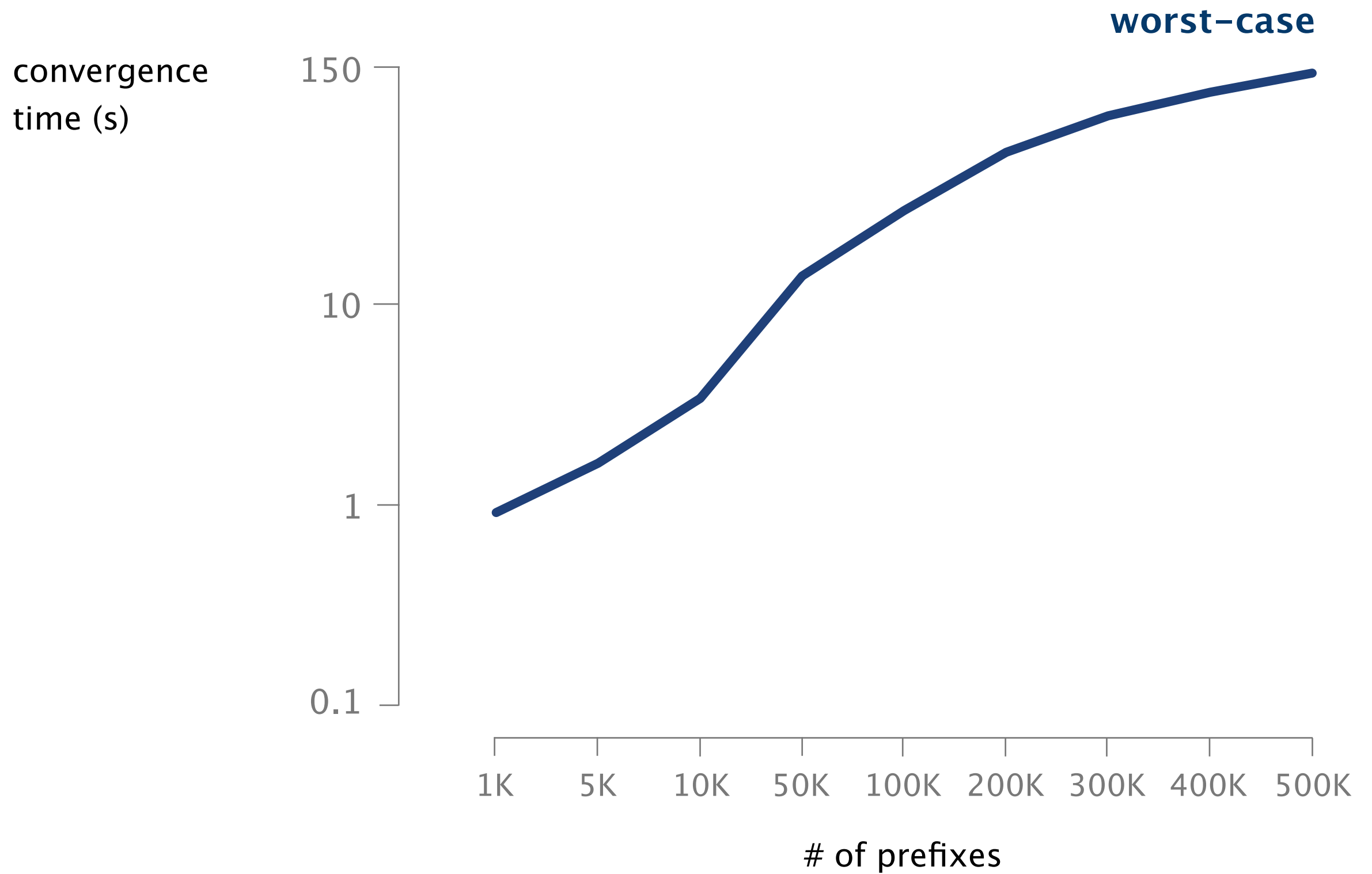
200K

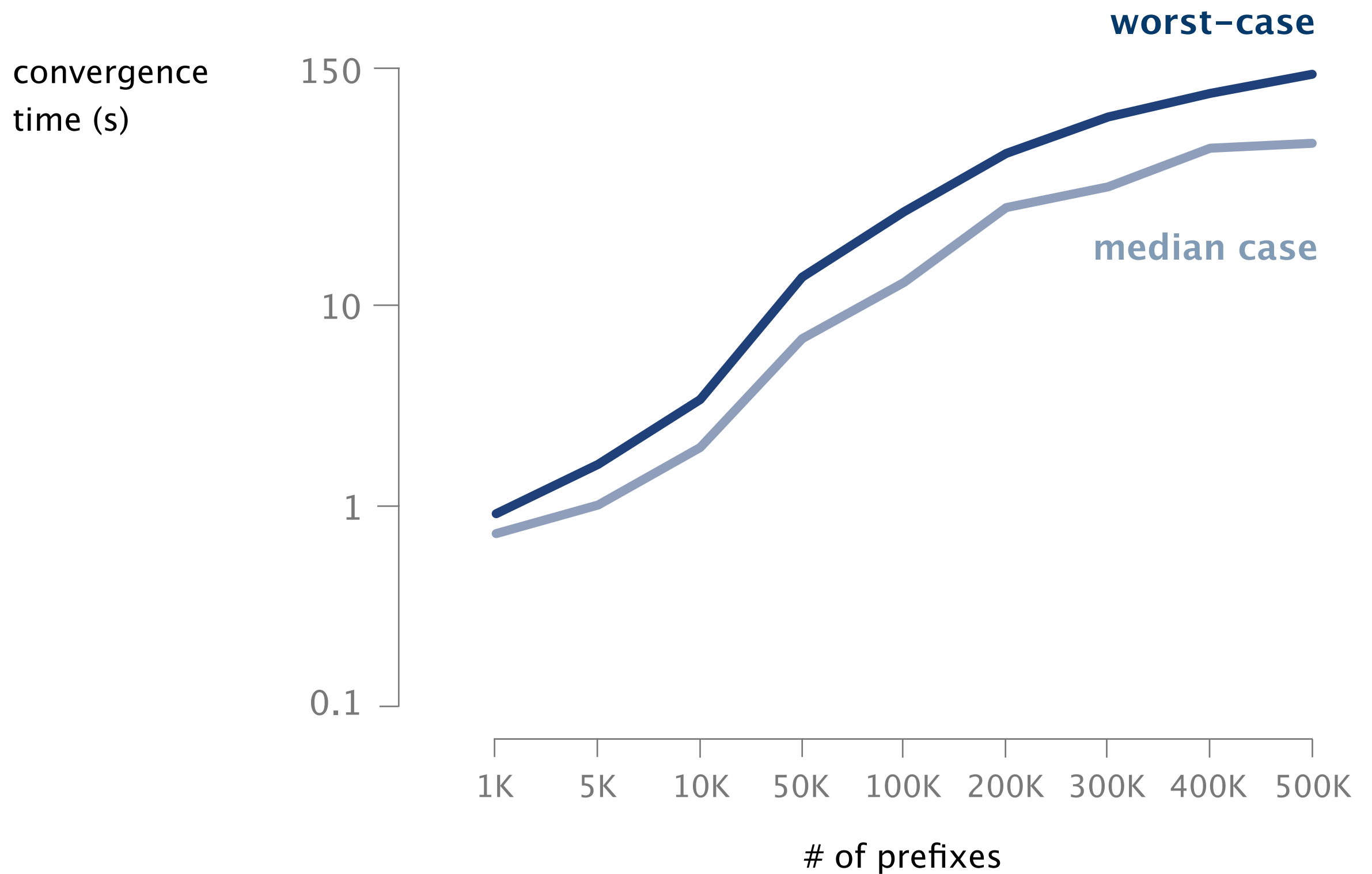
300K

400K

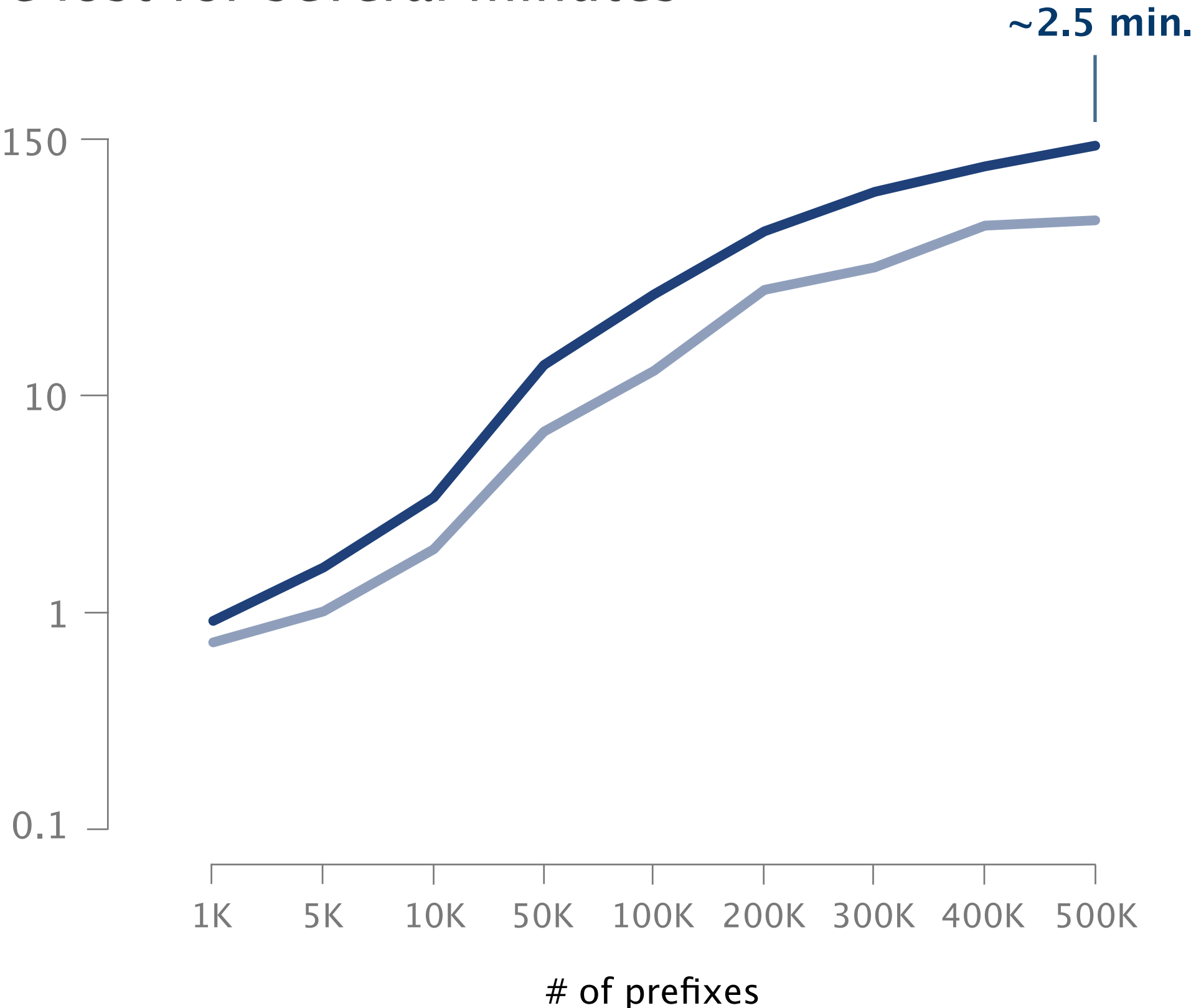
500K

of prefixes





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

Two universal tricks you can apply to any computer sciences problem

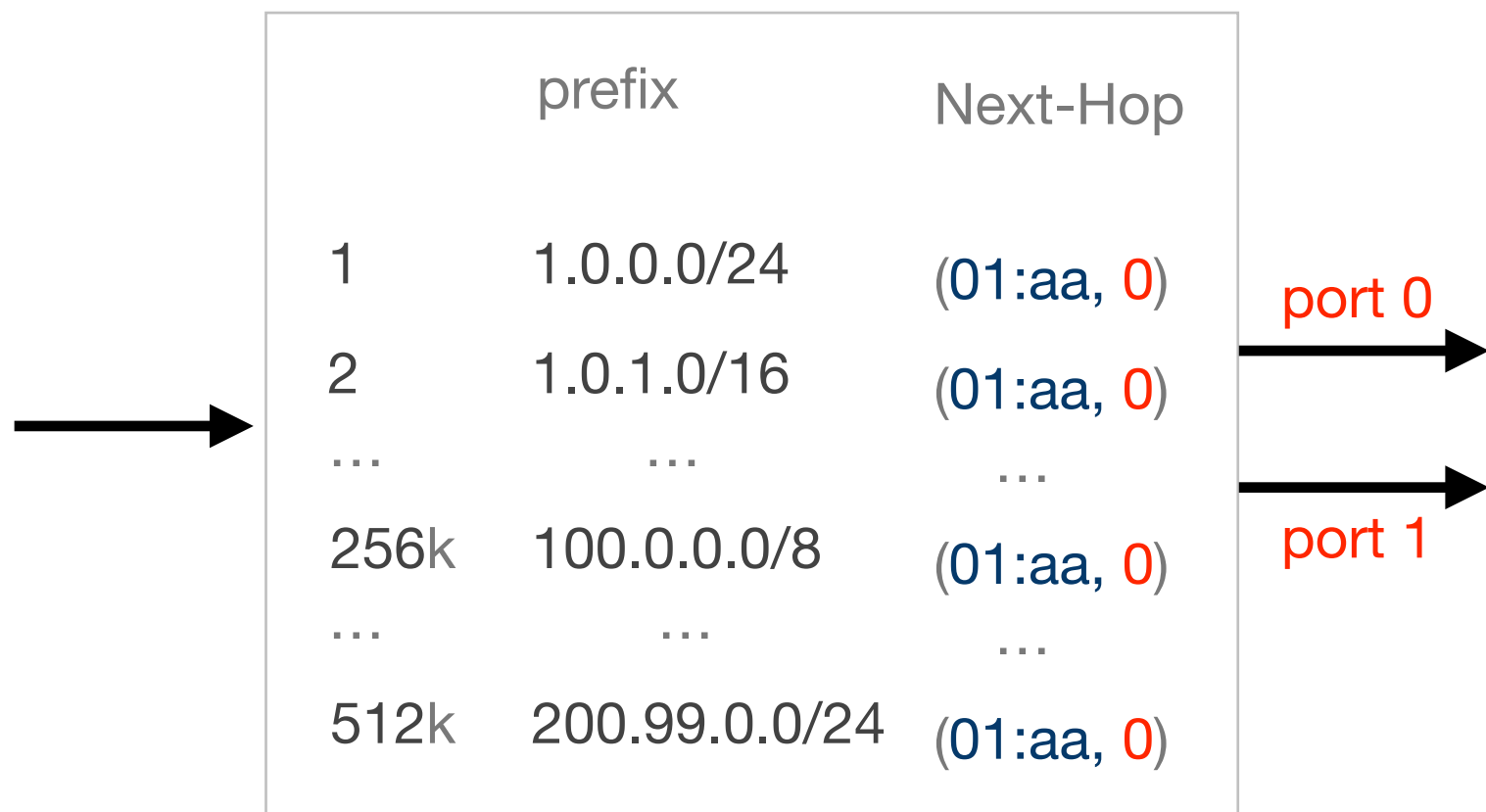
When you need... more flexibility,
you add... a layer of indirection

When you need... more scalability,
you add... a hierarchical structure

When you need... more flexibility,
you add... a layer of indirection

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)

... 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

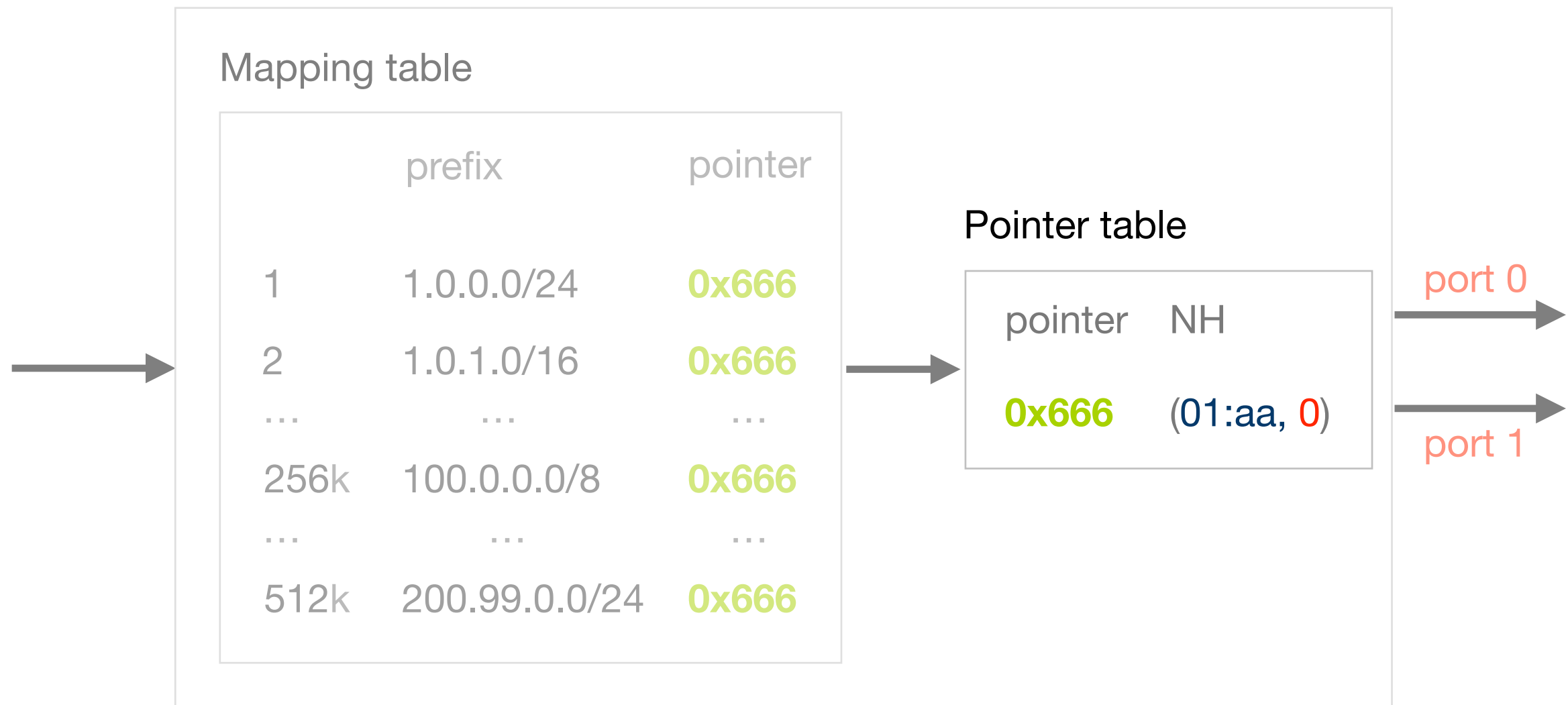
pointer	NH
0x666	(01:aa, 0)

port 0

port 1

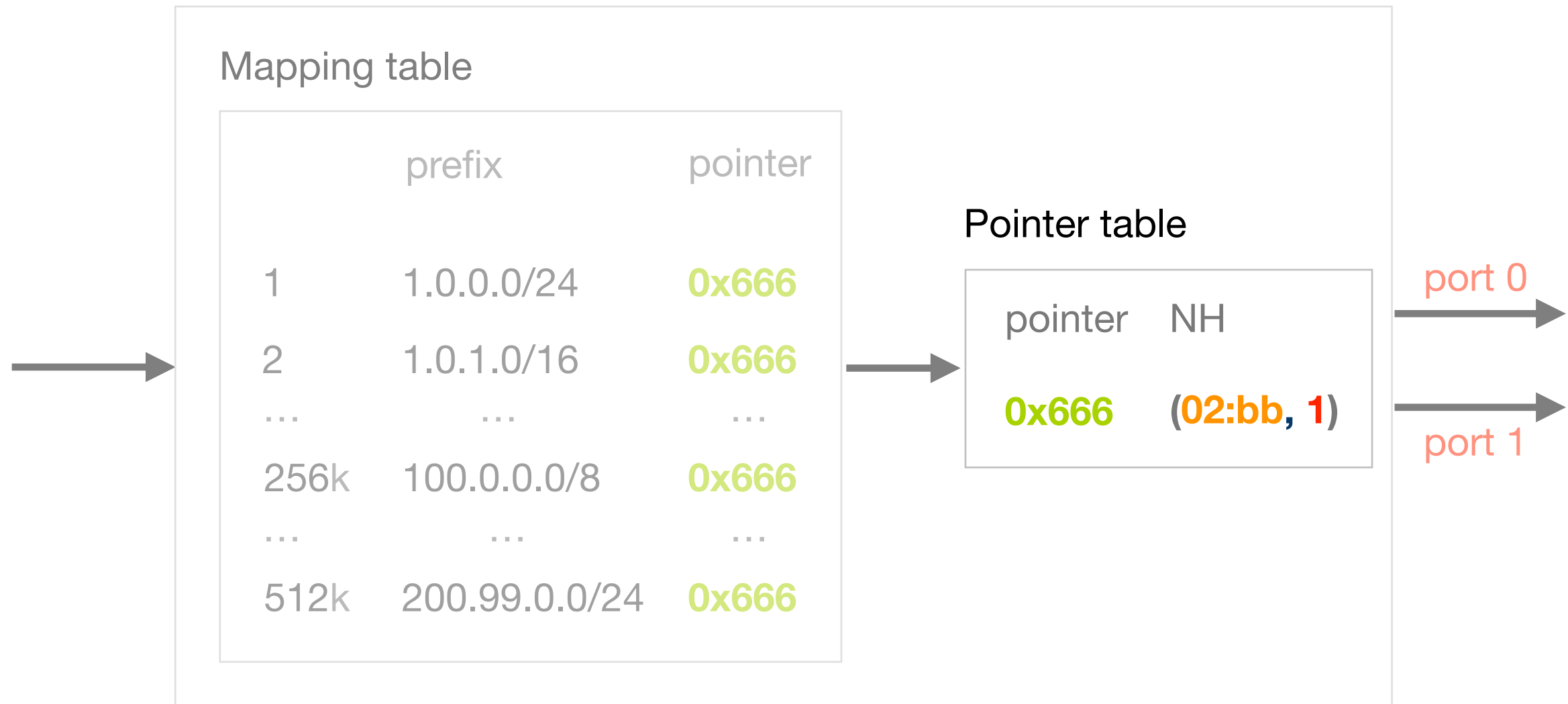
Upon failures, we update the pointer table

Router Forwarding Table

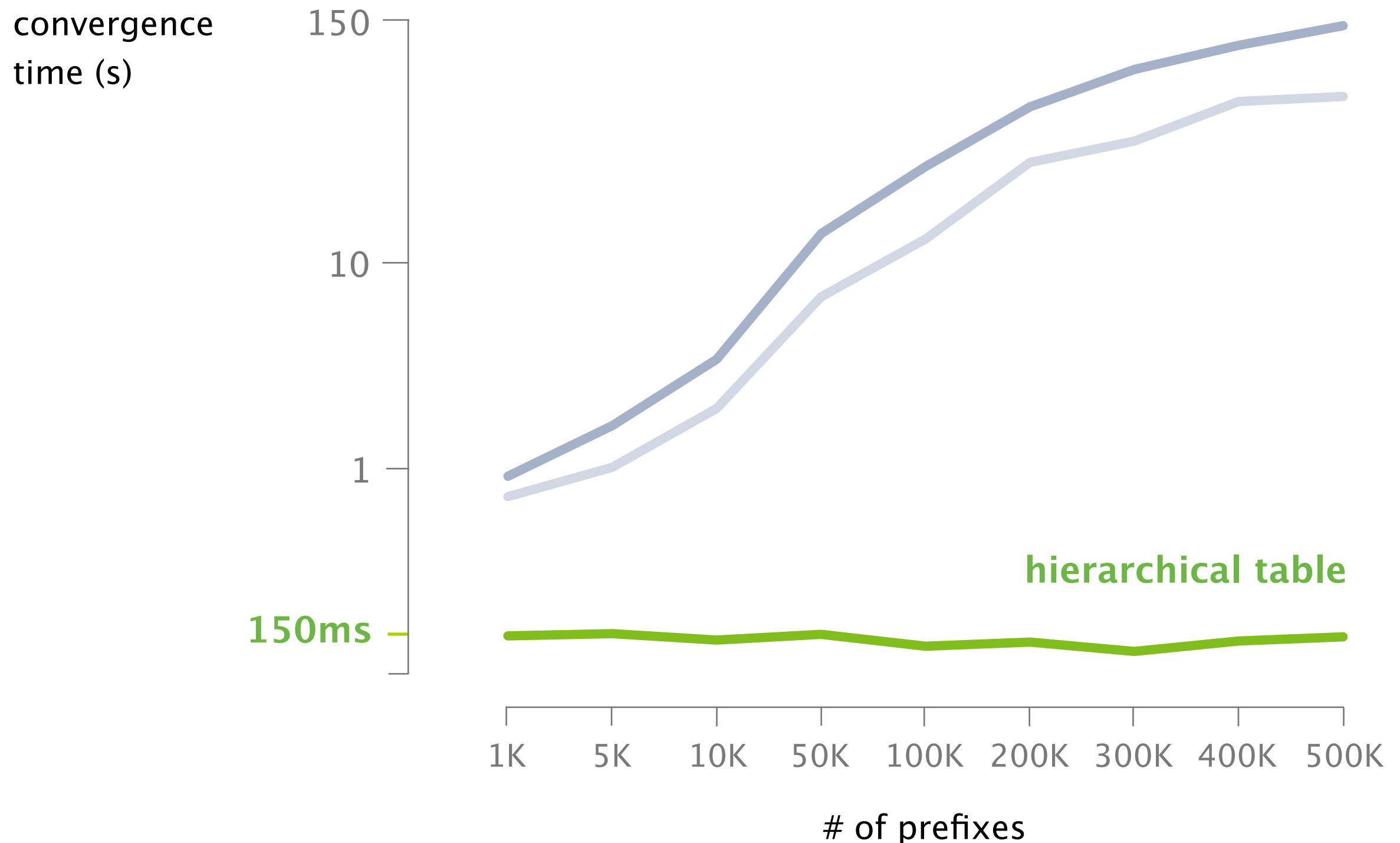


Here, we only need to do one update

Router Forwarding Table



Hierarchical table enables to converge within 150ms,
independently on the number of prefixes



Today, two Link-State protocols are widely used:
OSPF and IS-IS



OSPF

Open Shortest Path First



IS-IS

Intermediate Systems²



OSPF

Open Shortest Path First

used in many enterprise & ISPs

work on top of IP

only route IPv4 by default



IS-IS

Intermediate Systems²



OSPF

Open Shortest Path First



IS-IS

Intermediate Systems²

used mostly in large ISPs
work on top of link-layer
network protocol agnostic

Internet routing

from here to there, and back



1

Intra-domain routing

Link-state protocols

Distance-vector protocols

Inter-domain routing

Path-vector protocols

Distance-vector protocols are based on
Bellman-Ford algorithm

Let $d_x(y)$ be the cost of the least-cost path
known by x to reach y

Let $d_x(y)$ be the cost of the least-cost path
known by x to reach y

Each node bundles these distances
into one message (called a vector)
that it **repeatedly** sends to all its neighbors

until convergence

Let $d_x(y)$ be the cost of the least-cost path known by x to reach y

Each node bundles these distances into one message (called a vector) that it repeatedly sends to all its neighbors

until convergence

Each node updates its distances based on neighbors' vectors:

$$d_x(y) = \min\{ c(x,v) + d_v(y) \} \quad \text{over all neighbors } v$$

Similarly to Link-State,
3 situations cause nodes to send new DVs

Topology change

link or node failure/recovery

Configuration change

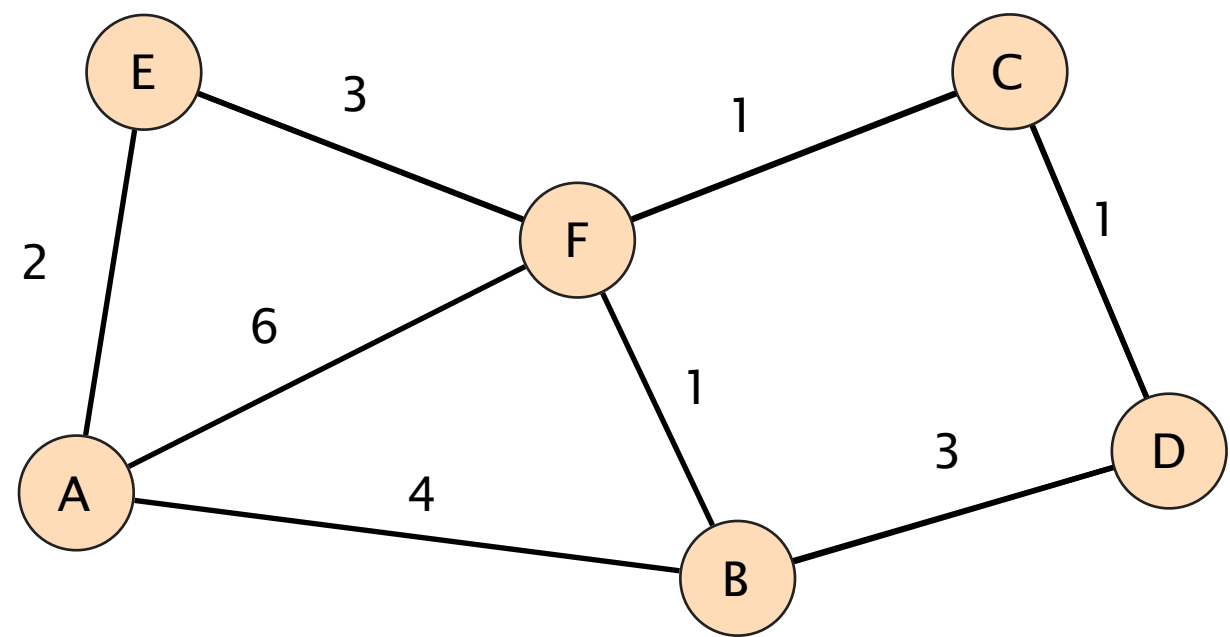
link cost change

Periodically

refresh the link-state information

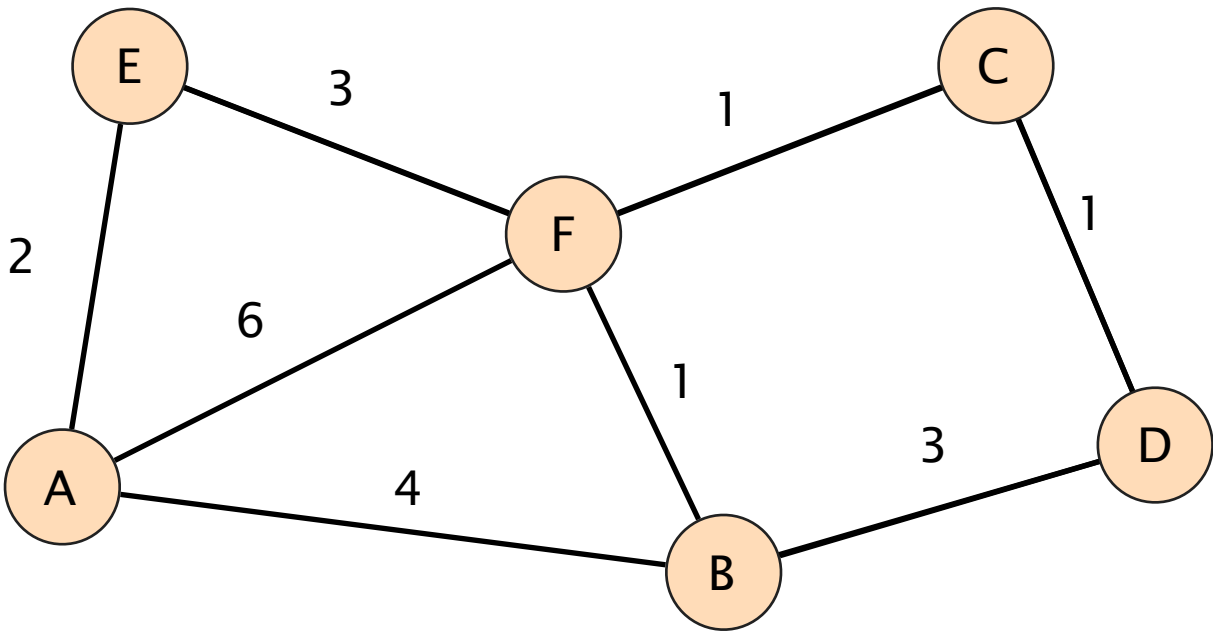
every (say) 30 minutes

account for possible data corruption



Optimum 1-hop path

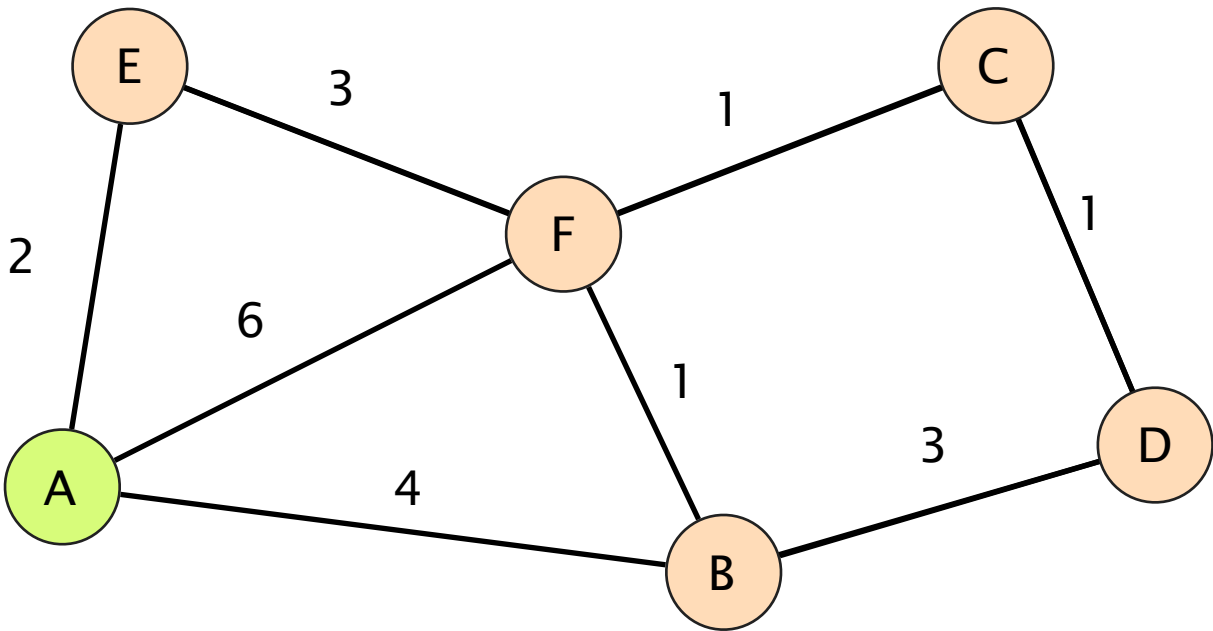
A			B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	∞	-	C	∞	-
D	∞	-	D	3	D
E	2	E	E	∞	-
F	6	F	F	1	F



C			D			E			F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	∞	-	A	∞	-	A	2	A	A	6	A
B	∞	-	B	3	B	B	∞	-	B	1	B
C	0	C	C	1	C	C	∞	-	C	1	C
D	1	D	D	0	D	D	∞	-	D	∞	-
E	∞	-	E	∞	-	E	0	E	E	3	E
F	1	F	F	∞	-	F	3	F	F	0	F

Optimum 1-hop path

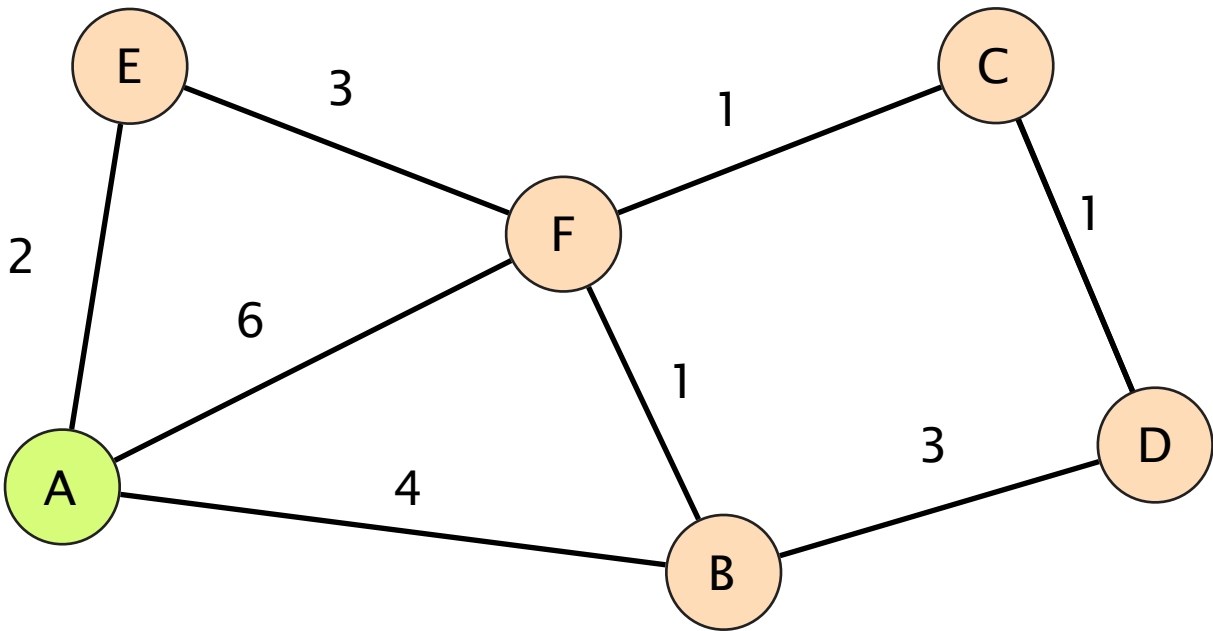
A			B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	∞	-	C	∞	-
D	∞	-	D	3	D
E	2	E	E	∞	-
F	6	F	F	1	F



C			D			E			F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	∞	-	A	∞	-	A	2	A	A	6	A
B	∞	-	B	3	B	B	∞	-	B	1	B
C	0	C	C	1	C	C	∞	-	C	1	C
D	1	D	D	0	D	D	∞	-	D	∞	-
E	∞	-	E	∞	-	E	0	E	E	3	E
F	1	F	F	∞	-	F	3	F	F	0	F

Optimum 2-hops path

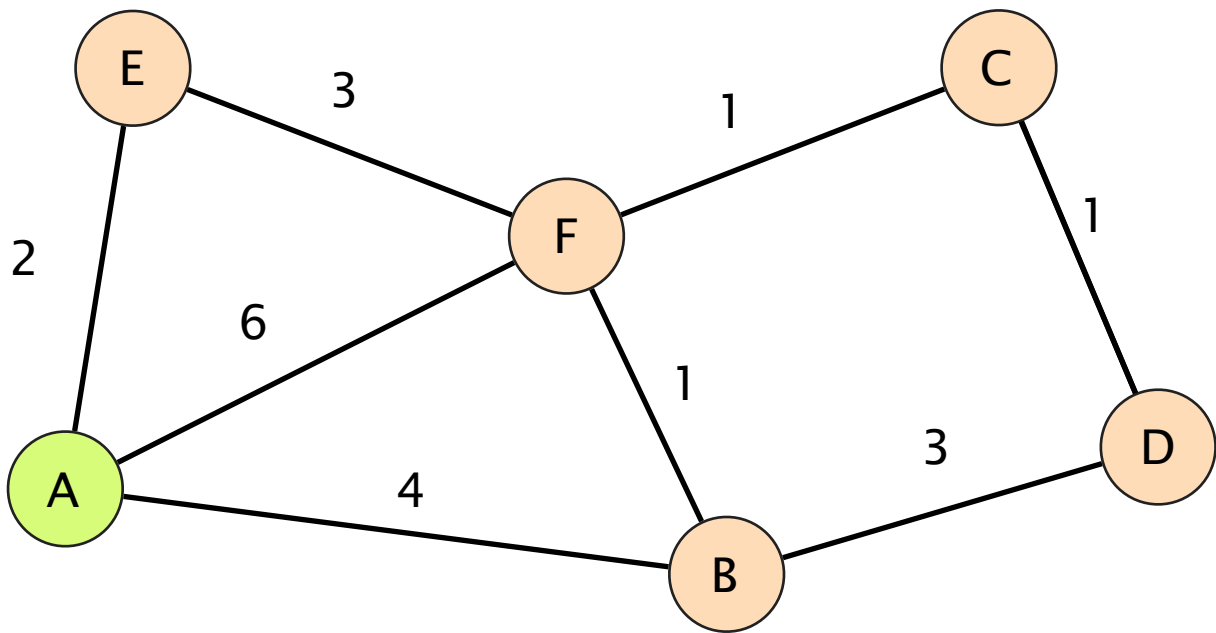
A			B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	7	F	C	2	F
D	7	B	D	3	D
E	2	E	E	4	F
F	5	E	F	1	F



C			D			E			F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	7	F	A	7	B	A	2	A	A	5	B
B	2	F	B	3	B	B	4	F	B	1	B
C	0	C	C	1	C	C	4	F	C	1	C
D	1	D	D	0	D	D	∞	-	D	2	C
E	4	F	E	∞	-	E	0	E	E	3	E
F	1	F	F	2	C	F	3	F	F	0	F

Optimum 3-hops path

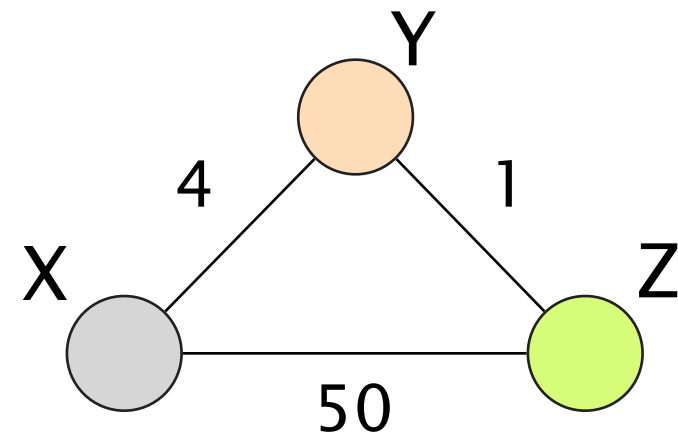
A			B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	6	E	C	2	F
D	7	F	D	3	D
E	2	E	E	4	F
F	5	E	F	1	F



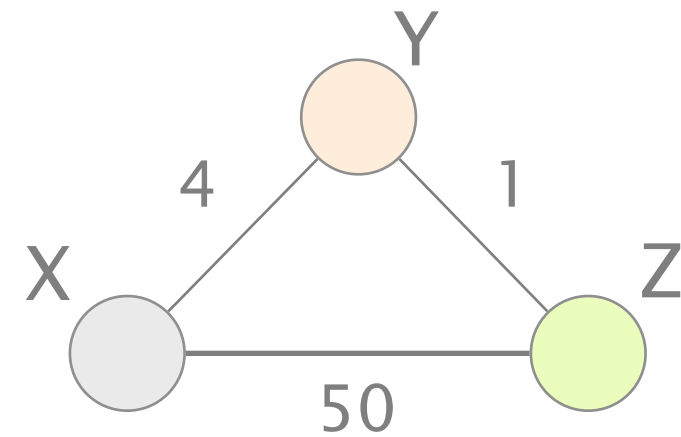
C			D			E			F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	6	F	A	7	B	A	2	A	A	5	B
B	2	F	B	3	B	B	4	F	B	1	B
C	0	C	C	1	C	C	4	F	C	1	C
D	1	D	D	0	D	D	5	F	D	2	C
E	4	F	E	5	C	E	0	E	E	3	E
F	1	F	F	2	C	F	3	F	F	0	F

Let's consider the convergence process
after a link cost change

Consider the following network



Consider the following network
leading to the following vectors



Y
vector

dest.	via
X	Z

Y reaches X directly

X	4	6
---	---	---

Z
vector

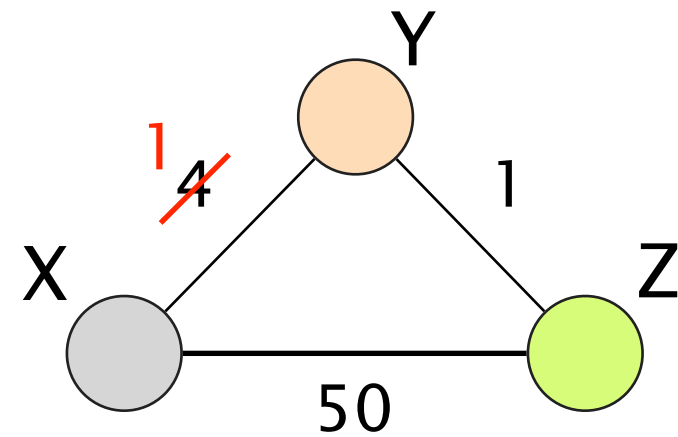
dest.	via
X	Y

Z reaches X via Y

X	50	5
---	----	---

$t = 0$

(X,Y) weight changes
from 4 to 1



time $t=0$

Y
vector

dest.	via
X	Z

X	4	6
---	---	---

Z
vector

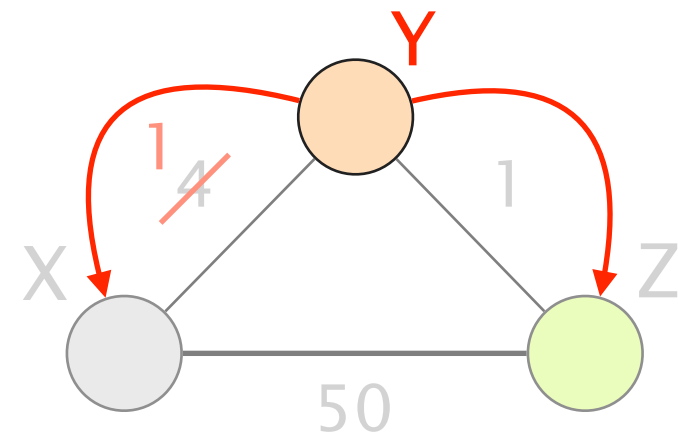
dest.	via
X	Y

X	50	5
---	----	---

Node detects local cost change, update their vectors,
and notify their neighbors if it has changed

t = 1

Y updates its vector,
sends it to X and Z



t=0

Y
vector

dest.	via	
	X	Z
X	4	6

t=1

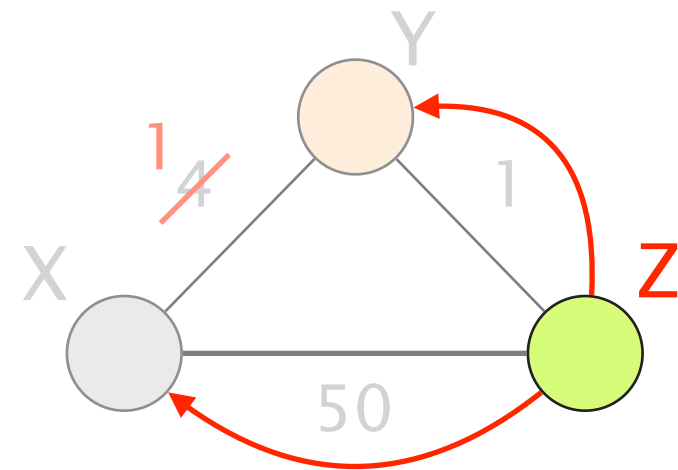
dest.	via	
	X	Z
X	1	6

Z
vector

dest.	via	
	X	Y
X	50	5

t = 2

Z updates its vector,
sends it to X and Y



t=0

Y
vector

dest.	via
X	4
Z	6

X 4 6

t=1

dest.	via
X	1
Z	6

X 1 6

t=2

Z
vector

dest.	via
X	50
Y	2

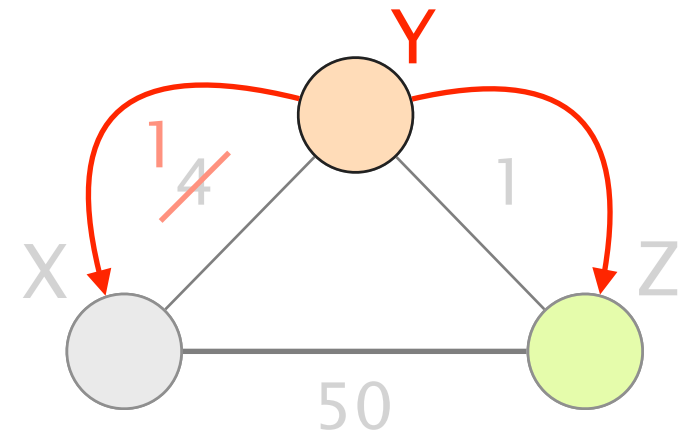
X 50 5

dest.	via
X	50
Y	2

X 50 2

t = 3

Y updates its vector,
sends it to X and Z



t=0

Y
vector

dest.	via	
	X	Z
X	4	6

t=1

dest.	via	
	X	Z
X	1	6

t=2

t=3

Z
vector

dest.	via	
	X	Y
X	50	5

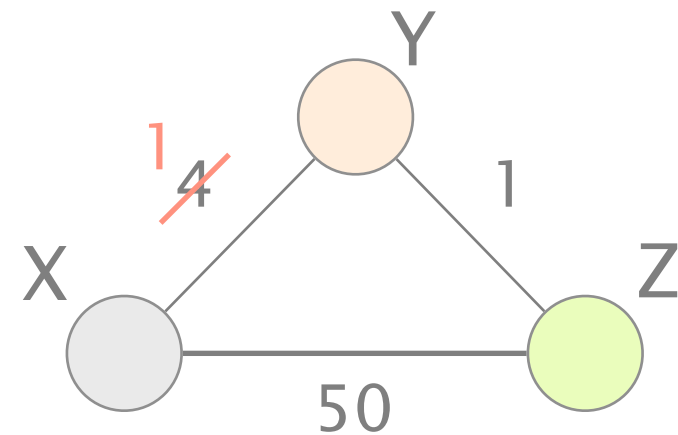
dest.	via	
	X	Y
X	50	2

dest.	via	
	X	Z
X	1	3

$t > 3$

no one moves anymore

network has converged!



$t=0$

$t=1$

$t=2$

$t > 3$

Y
vector

dest.	via
X	Z

X 4 6

dest.	via
X	Z

X 1 6

dest.	via
X	Z

X 1 3

Z
vector

dest.	via
X	Y

X 50 5

dest.	via
X	Y

X 50 2

dest.	via
X	Y

X 50 2

The algorithm terminates
after 3 iterations

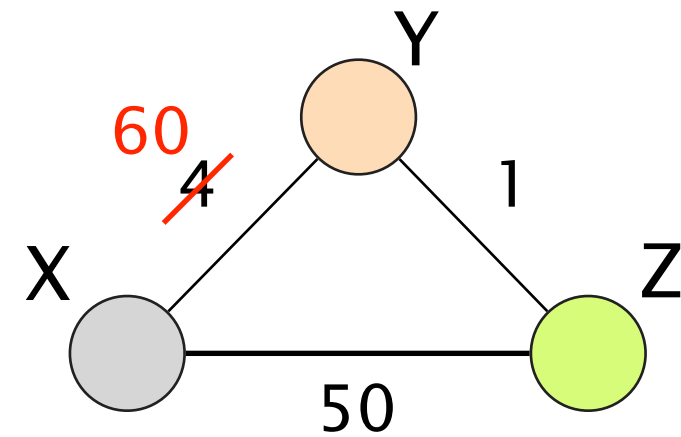
Good news travel fast!

Good news travel fast!

What about bad ones?

$t = 0$

(X,Y) weight changes
from 4 to 60



time $t=0$

Y
vector

dest.	via
X	Z

X	4	6
---	---	---

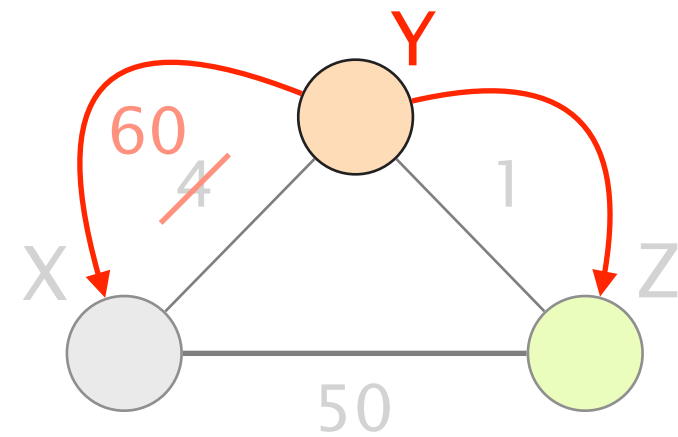
Z
vector

dest.	via
X	Y

X	50	5
---	----	---

t = 1

Y updates its vector,
sends it to X and Z



t=0

Y
vector

dest.	via
X	4
Z	6

X 4 6

t=1

dest.	via
X	60
Z	6

X 60 6

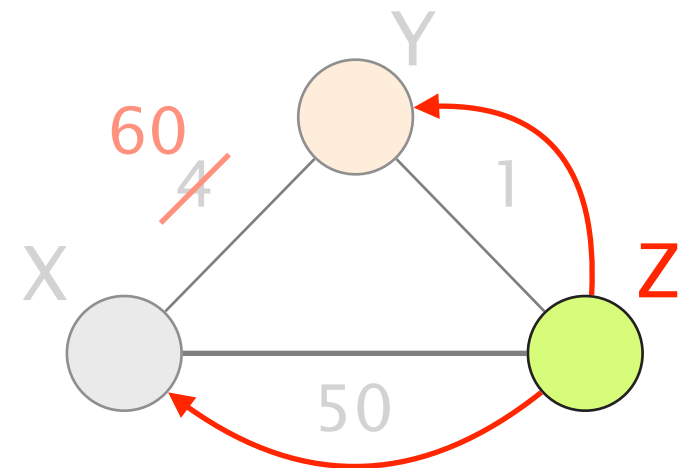
Z
vector

dest.	via
X	50
Y	5

X 50 5

t = 2

Z updates its vector,
sends it to X and Y



t=0

Y
vector

dest.	via
X	4
Z	6

X 4 6

t=1

dest.	via
X	60
Z	6

X 60 6

t=2

Z
vector

dest.	via
X	50
Y	7

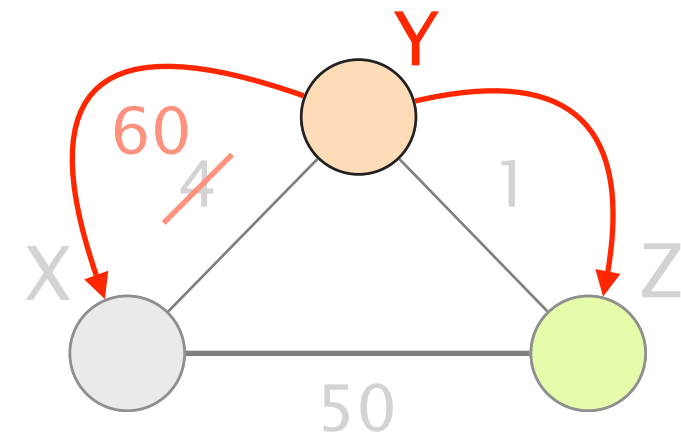
X 50 5

dest.	via
X	50
Y	7

X 50 7

t = 3

Y updates its vector,
sends it to X and Z



t=0

Y
vector

dest.	via	
	X	Z
X	4	6

t=1

dest.	via	
	X	Z
X	60	6

t=2

t=3

Z
vector

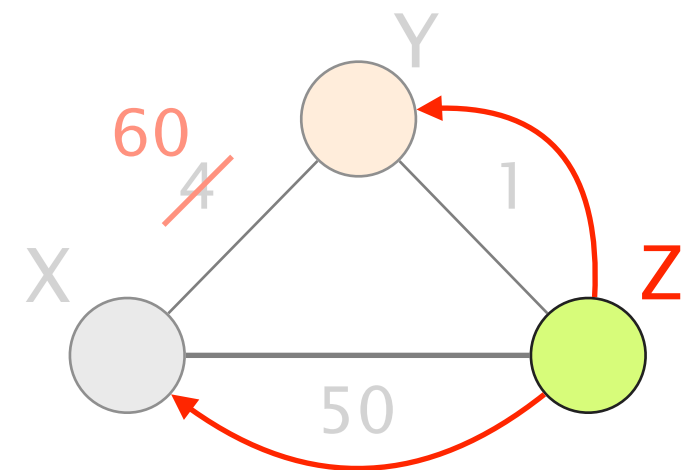
dest.	via	
	X	Y
X	50	5

dest.	via	
	X	Y
X	50	7

dest.	via	
	X	Z
X	60	8

$t = 4$

Z updates its vector,
sends it to X and Y...



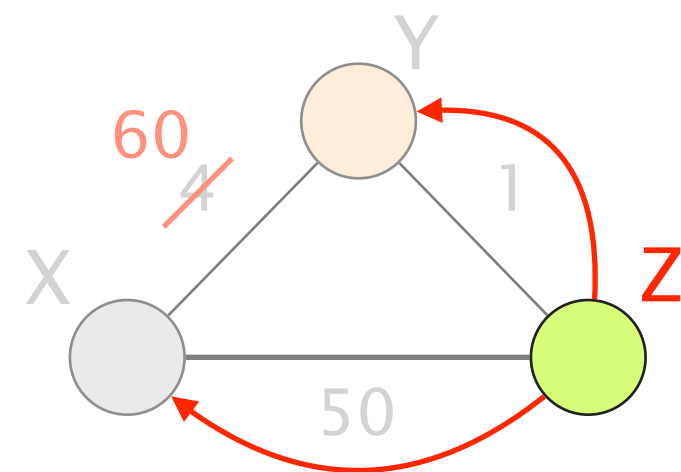
$t=4$

Y
vector

Z
vector

dest.	via
X	Y

X 50 9



t=4

t=44

... many iterations later ...

Y
vector

dest.	via	
	X	Z

X 60 51

Z
vector

dest.	via	
	X	Y

X 50 9

dest.	via	
	X	Y

X 50 52

The algorithm terminates
after 44 iterations!

Bad news travel slow!

This problem is known as
count-to-infinity, a type of routing loop

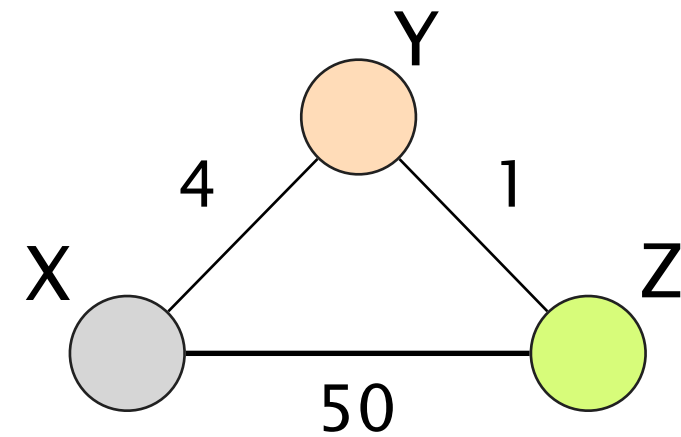
Count-to-infinity leads to very slow convergence
what if the cost had changed from 4 to 9999?

Routers don't know when neighbors use them
Z does not know that Y has switched to use it

Let's try to fix that

Whenever a router uses another one,
it will announce it an infinite cost

The technique is known as poisoned reverse



Y
vector

dest.	via
X	Z
X	4 ∞

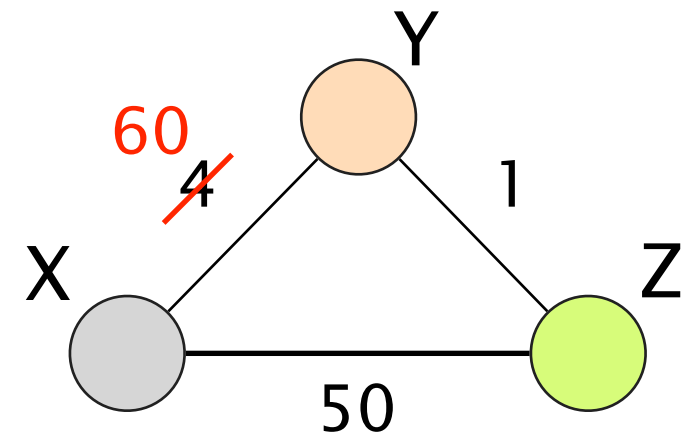
As Z uses Y to reach X,
it announces to Y an infinite cost

Z
vector

dest.	via
X	Y
X	50 5

$t = 0$

(X,Y) weight changes
from 4 to 60



time $t=0$

Y
vector

dest.	via
X	Z

X	4	∞
---	---	----------

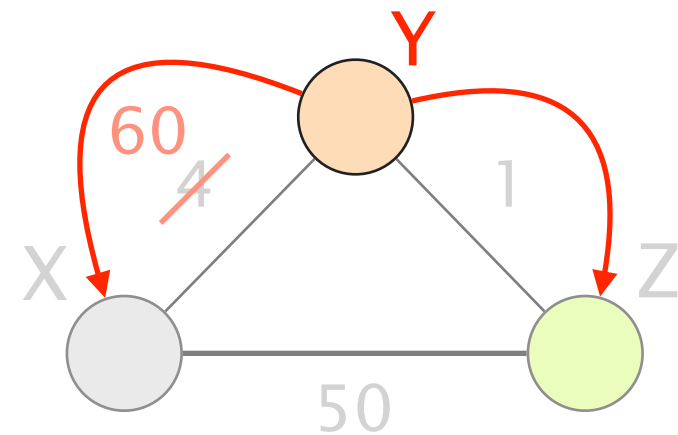
Z
vector

dest.	via
X	Y

X	50	5
---	----	---

t = 1

Y updates its vector,
sends it to X and Z



t=0

Y
vector

dest.	via	
	X	Z
X	4	∞

t=1

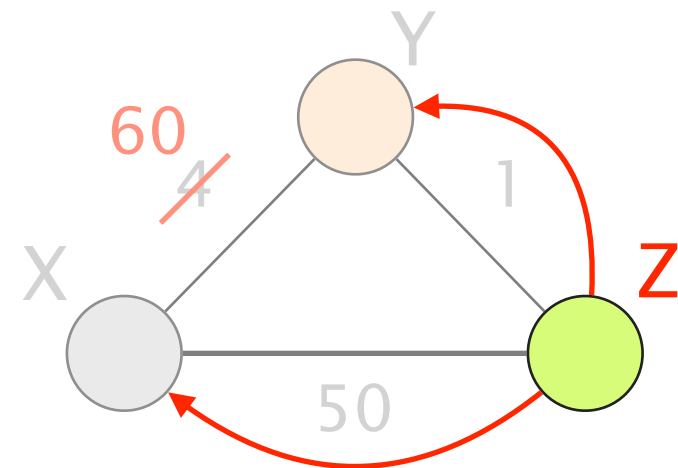
dest.	via	
	X	Z
X	60	∞

Z
vector

dest.	via	
	X	Y
X	50	5

t = 2

Z updates its vector,
sends it to X and Y



t=0

Y
vector

dest.	via
X	Z

X 4 ∞

t=1

dest.	via
X	Z

X 60 ∞

t=2

Z
vector

dest.	via
X	Y

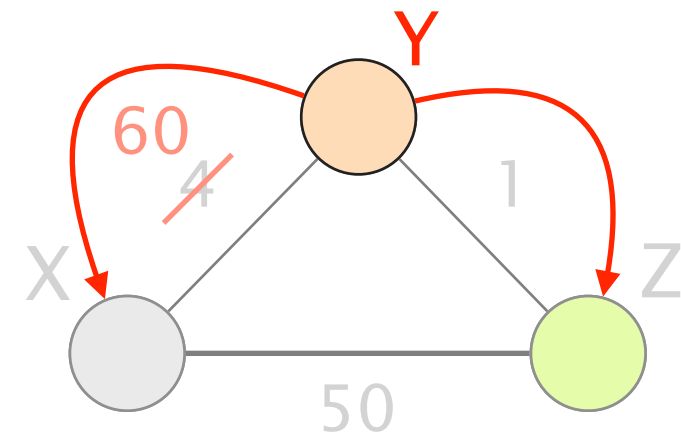
X 50 5

dest.	via
X	Y

X 50 61

t = 3

Y updates its vector,
sends it to X and Z



t=0

Y
vector

dest.	via	
	X	Z
X	4	∞

t=1

dest.	via	
	X	Z
X	60	∞

t=2

t=3

Z
vector

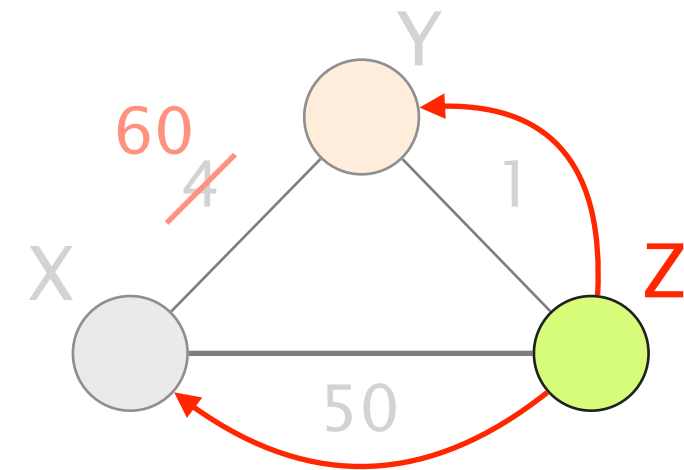
dest.	via	
	X	Y
X	50	5

dest.	via	
	X	Y
X	50	61

dest.	via	
	X	Z
X	60	51

$t = 4$

Z updates its vector,
sends it to X and Y



$t=4$

Y
vector

Z
vector

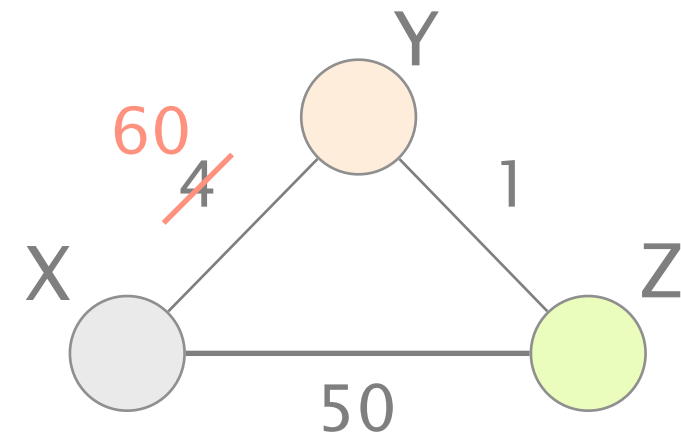
dest.	via
X	Y

X 50 ∞

$t > 4$

no one moves

network has converged!



$t=4$

$t > 4$

Y
vector

dest.	via
X	Z

X 60 51

Z
vector

dest.	via
X	Y

X 50 ∞

dest.	via
X	Y

X 50 ∞

While poisoned reverse solved this case,
it does **not** solve loops involving 3 or more nodes...

see exercise session

Actual distance-vector protocols mitigate this issue by using small “infinity”, *e.g.* 16

Link-State vs Distance-Vector routing

	Message complexity	Convergence speed	Robustness
Link-State	$O(nE)$ message sent n: #nodes E: #links	relatively fast	node can advertise incorrect link cost nodes compute their own table
Distance-Vector	between neighbors only	slow	node can advertise incorrect path cost errors propagate

Internet routing

from here to there, and back



Intra-domain routing

Link-state protocols

Distance-vector protocols

2

Inter-domain routing

Path-vector protocols

Internet

Internet

Internet



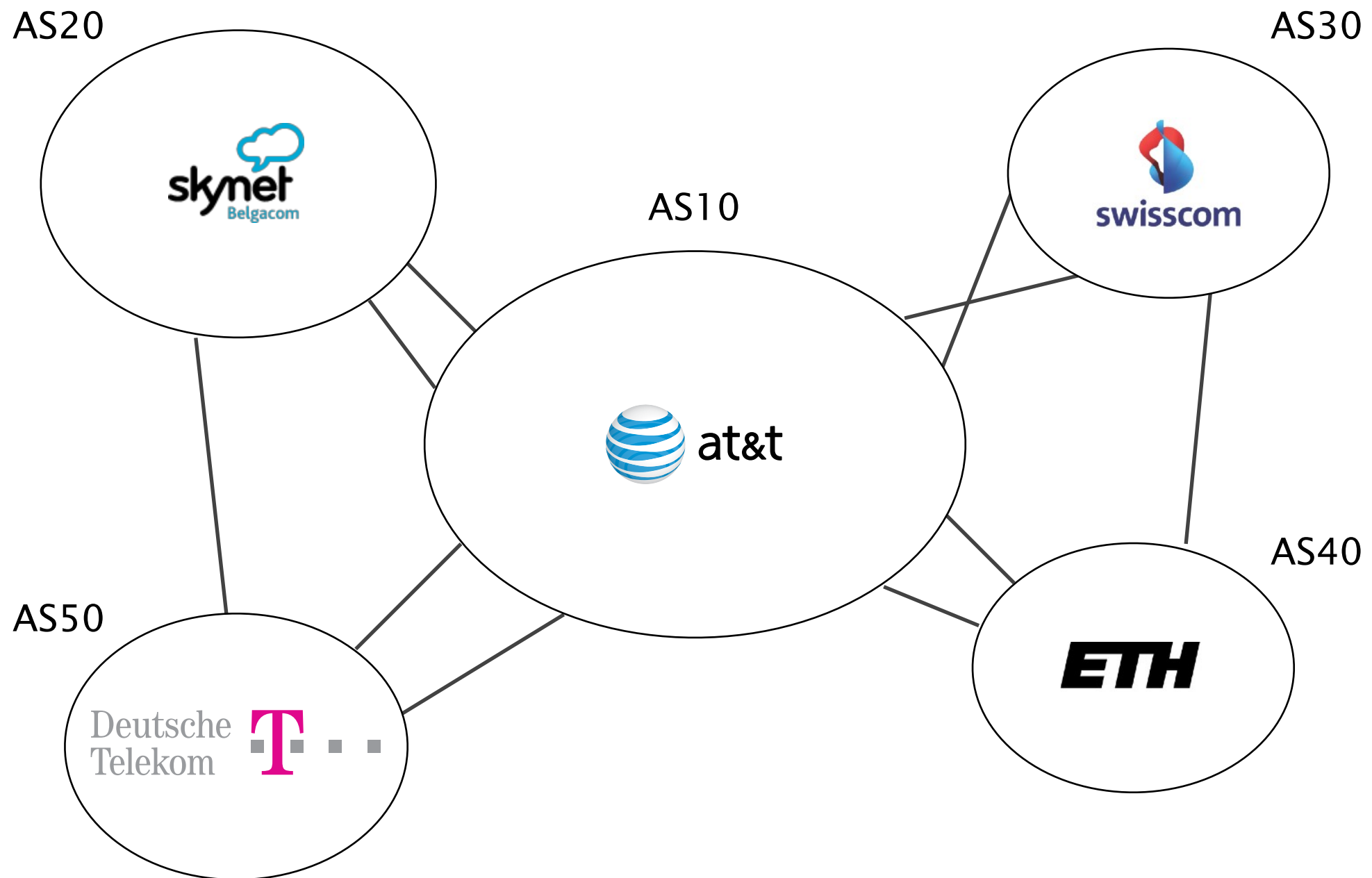
A network of *networks*

Internet

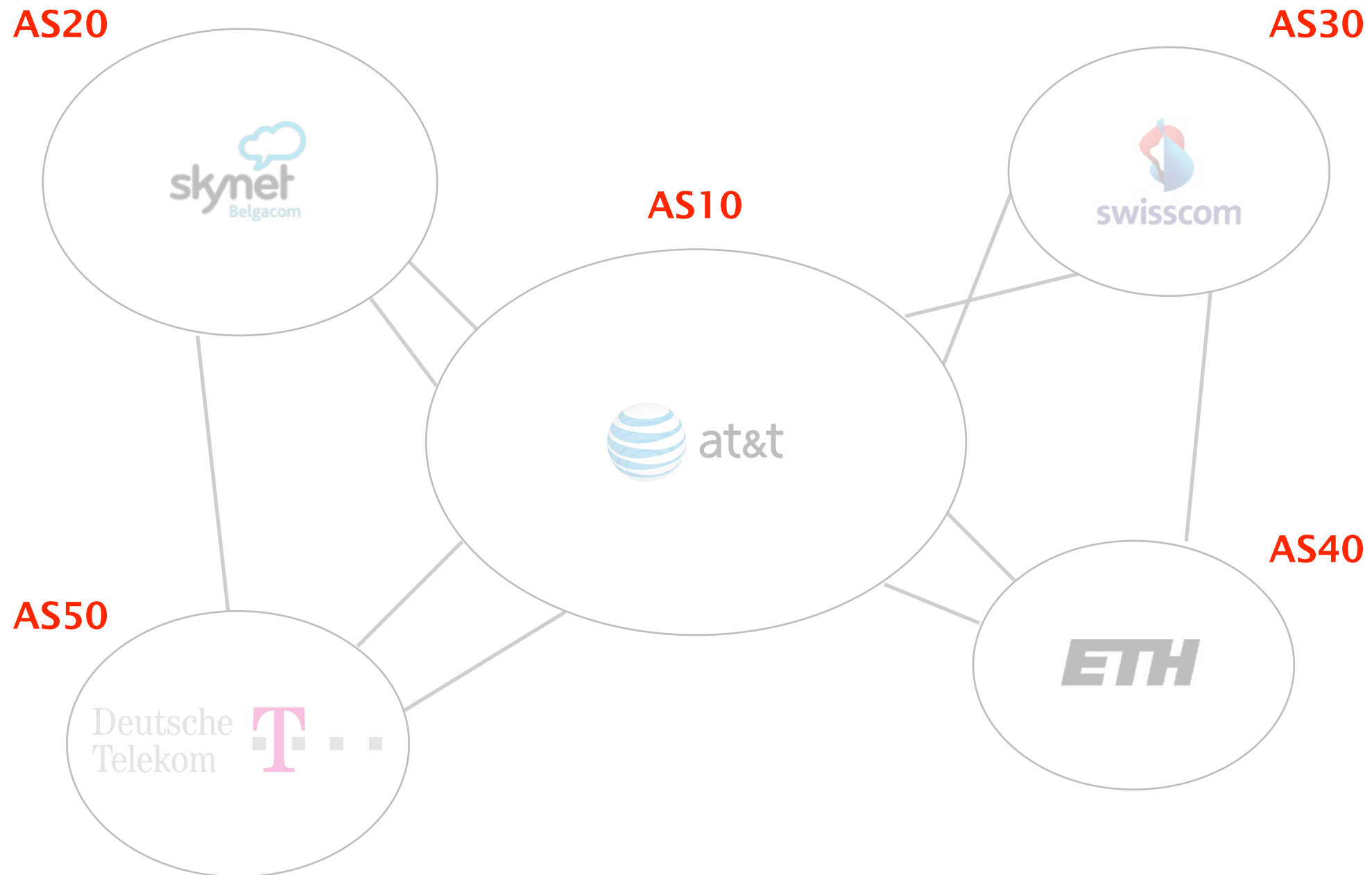


Border Gateway Protocol (BGP)

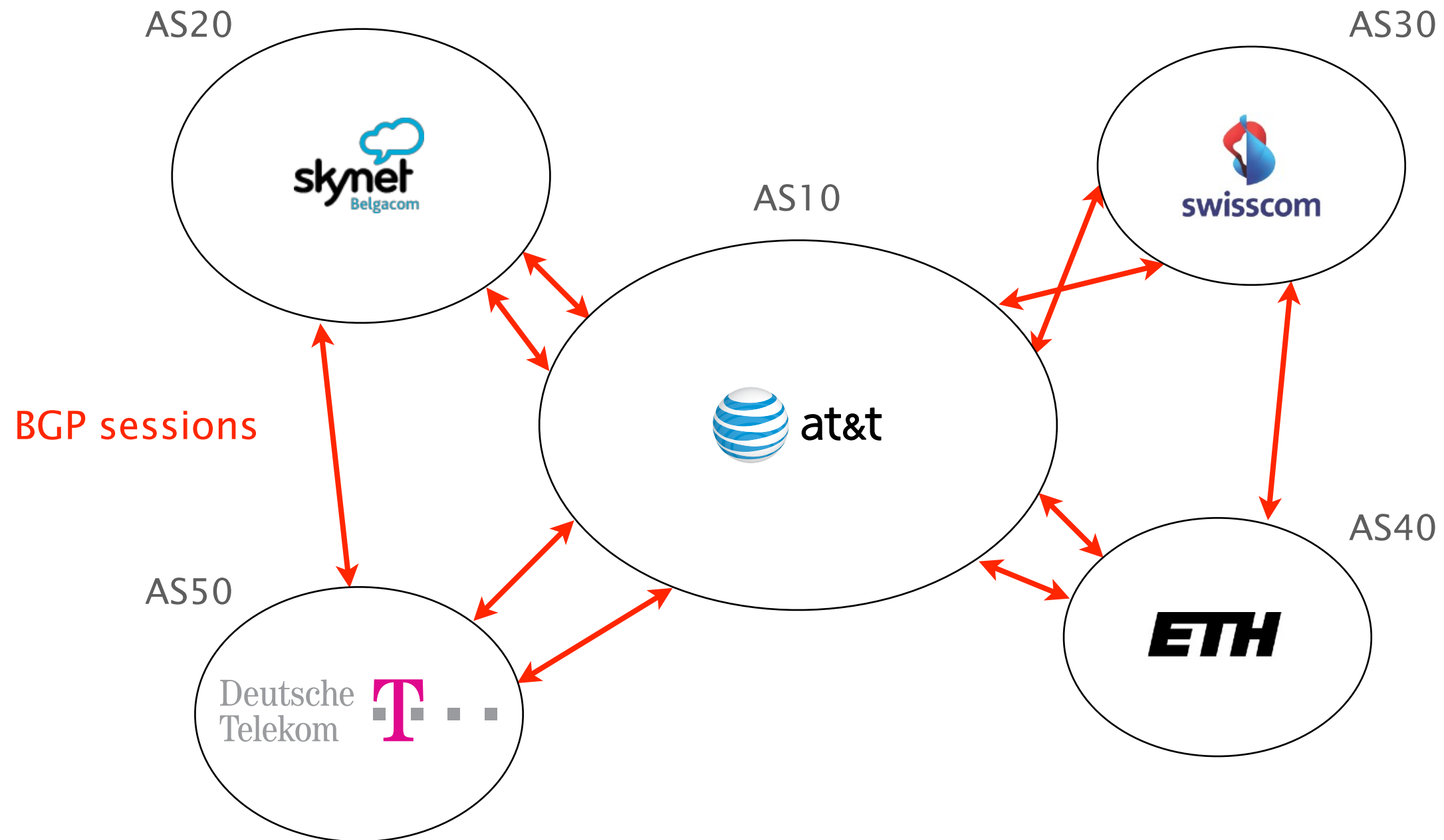
The Internet is a network of networks,
referred to as Autonomous Systems (AS)



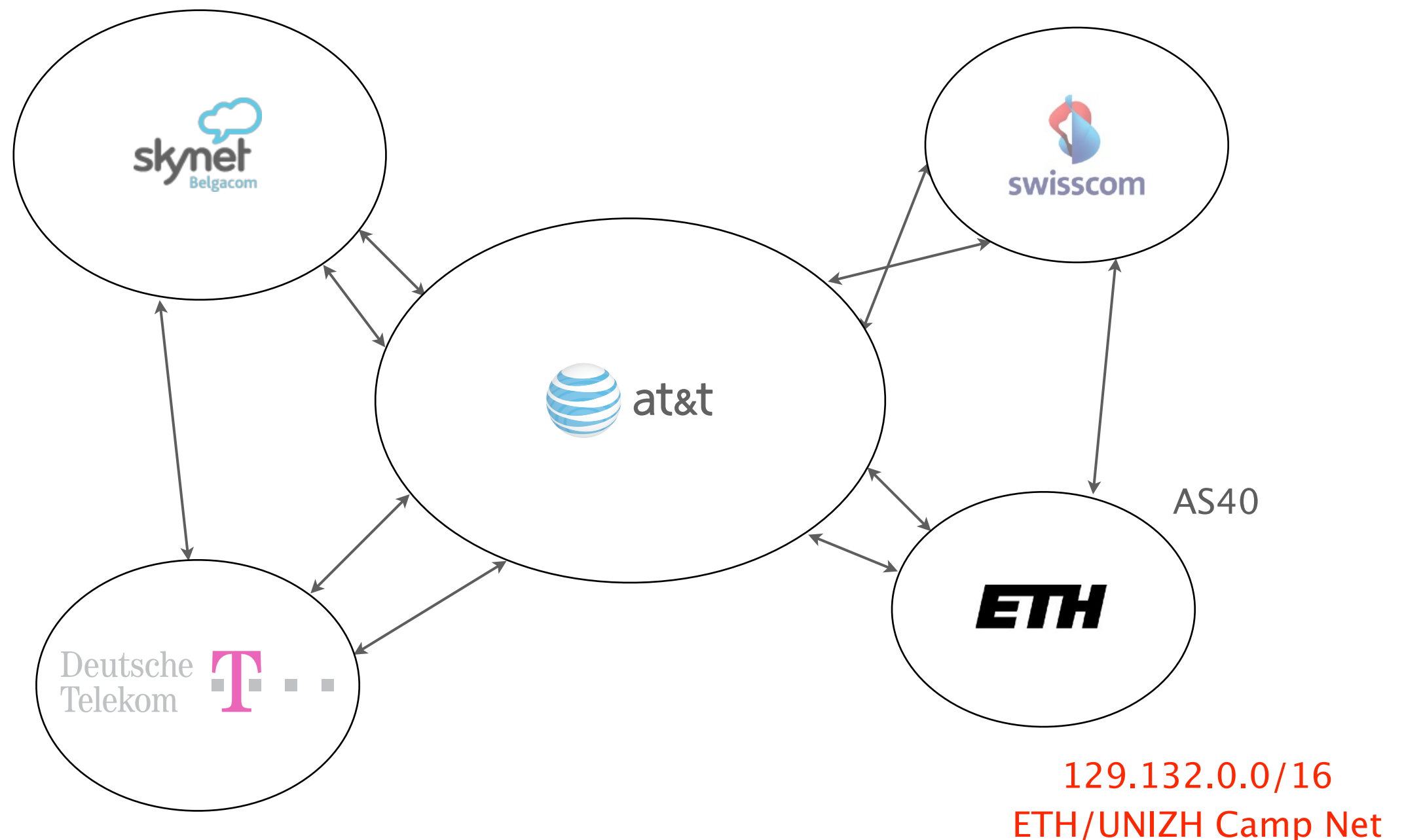
Each AS has a number (encoded on 16 bits)
which identifies it



BGP is the routing protocol “glueing”
the entire Internet together



Using BGP, ASes exchange information about the IP prefixes they can reach, directly or indirectly



BGP needs to solve three key challenges: scalability, privacy and policy enforcement

There is a huge # of networks and prefixes

700k prefixes, >50,000 networks, millions (!) of routers

Networks don't want to divulge internal topologies

or their business relationships

Networks need to control where to send and receive traffic

without an Internet-wide notion of a link cost metric

Link-State routing **does not** solve these challenges

Floods topology information

high processing overhead

Requires each node to compute the entire path

high processing overhead

Minimizes some notion of total distance

works only if the policy is shared and uniform

Distance-Vector routing is on the right track

pros

Hide details of the network topology

nodes determine only “next-hop” for each destination

Distance-Vector routing is on the right track, but not really there yet...

pros

Hide details of the network topology

nodes determine only “next-hop” for each destination

cons

It still minimizes some common distance

impossible to achieve in an inter domain setting

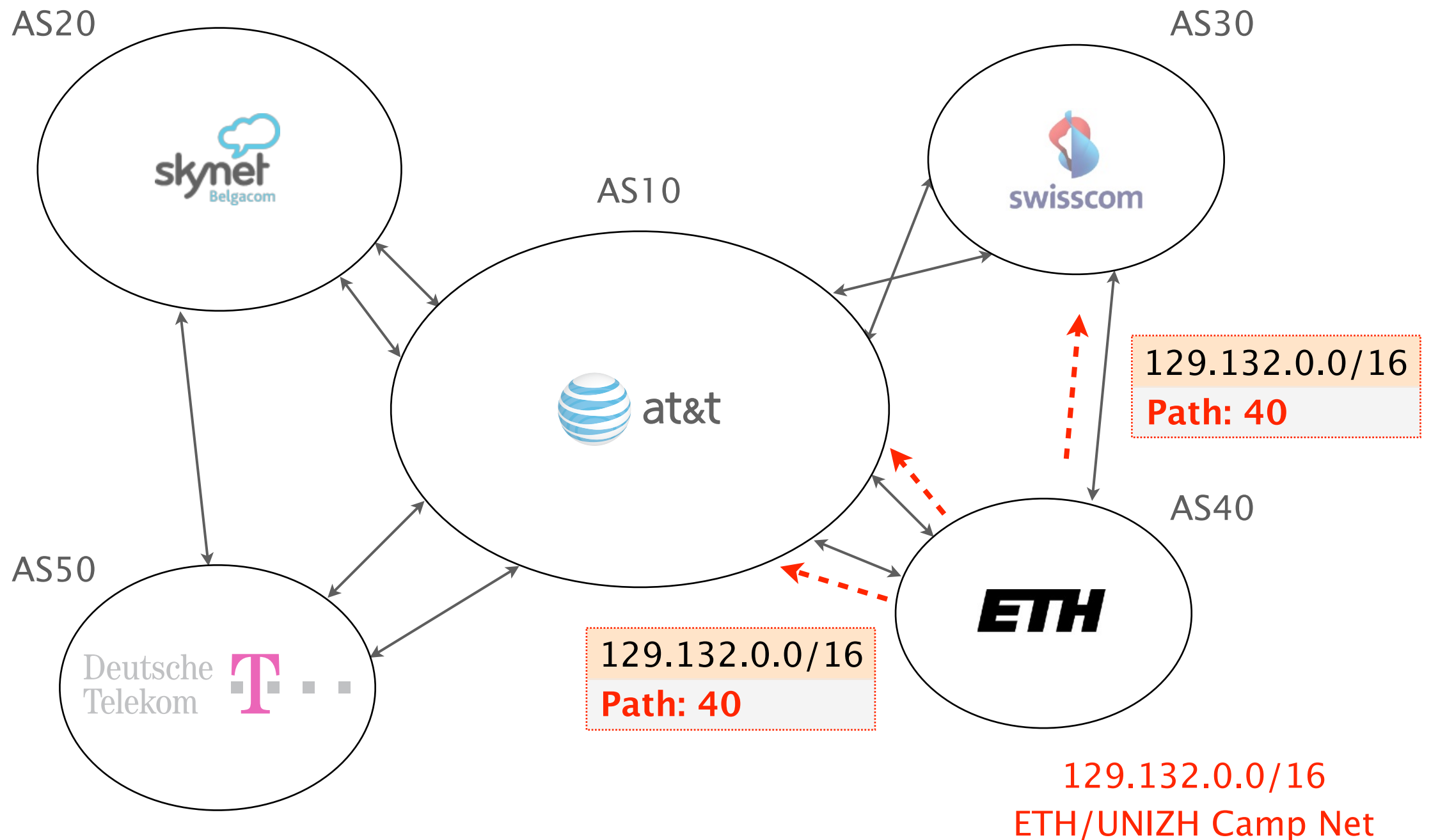
It converges slowly

counting-to-infinity problem

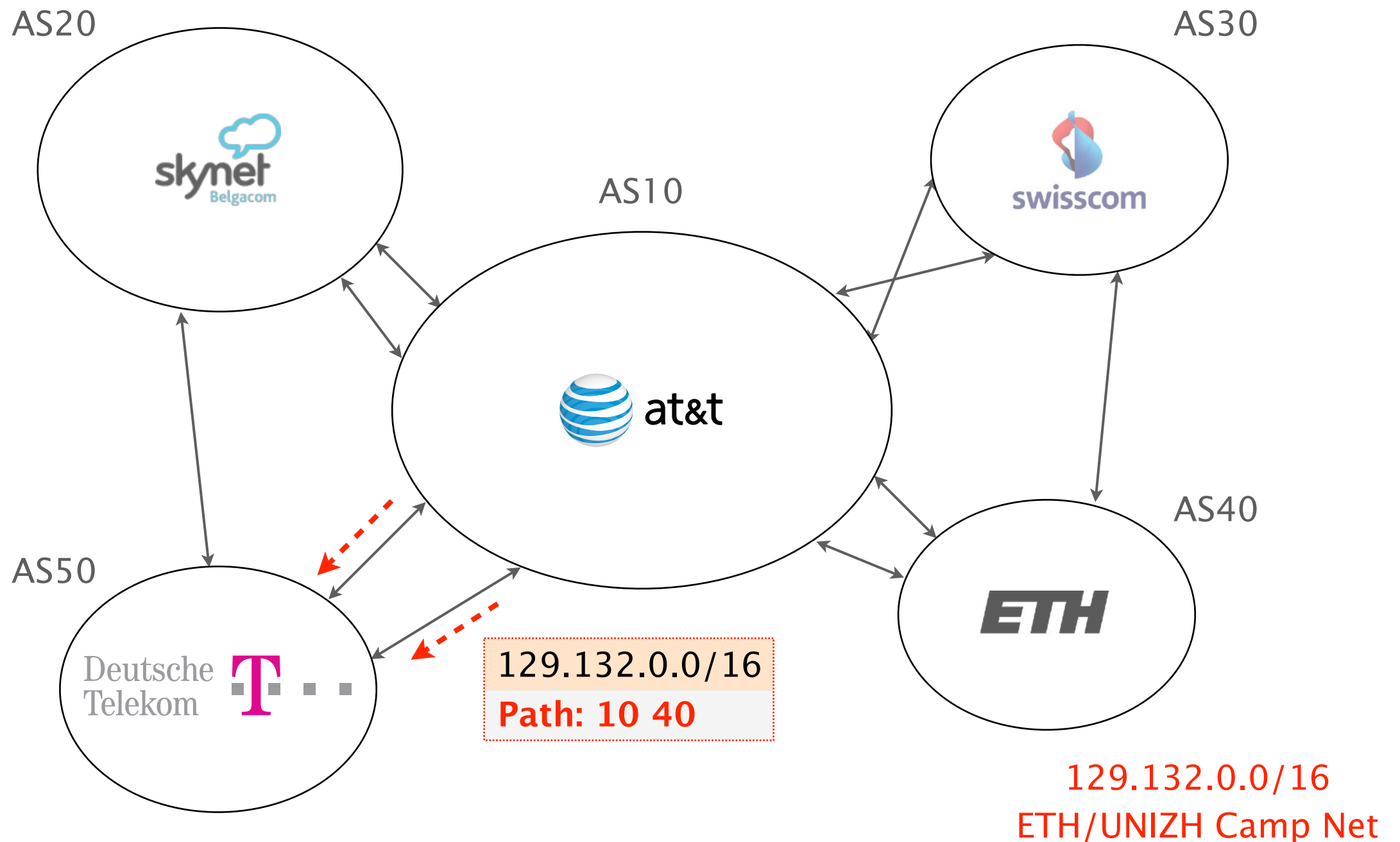
BGP relies on **path-vector routing** to support flexible routing policies and avoid count-to-infinity

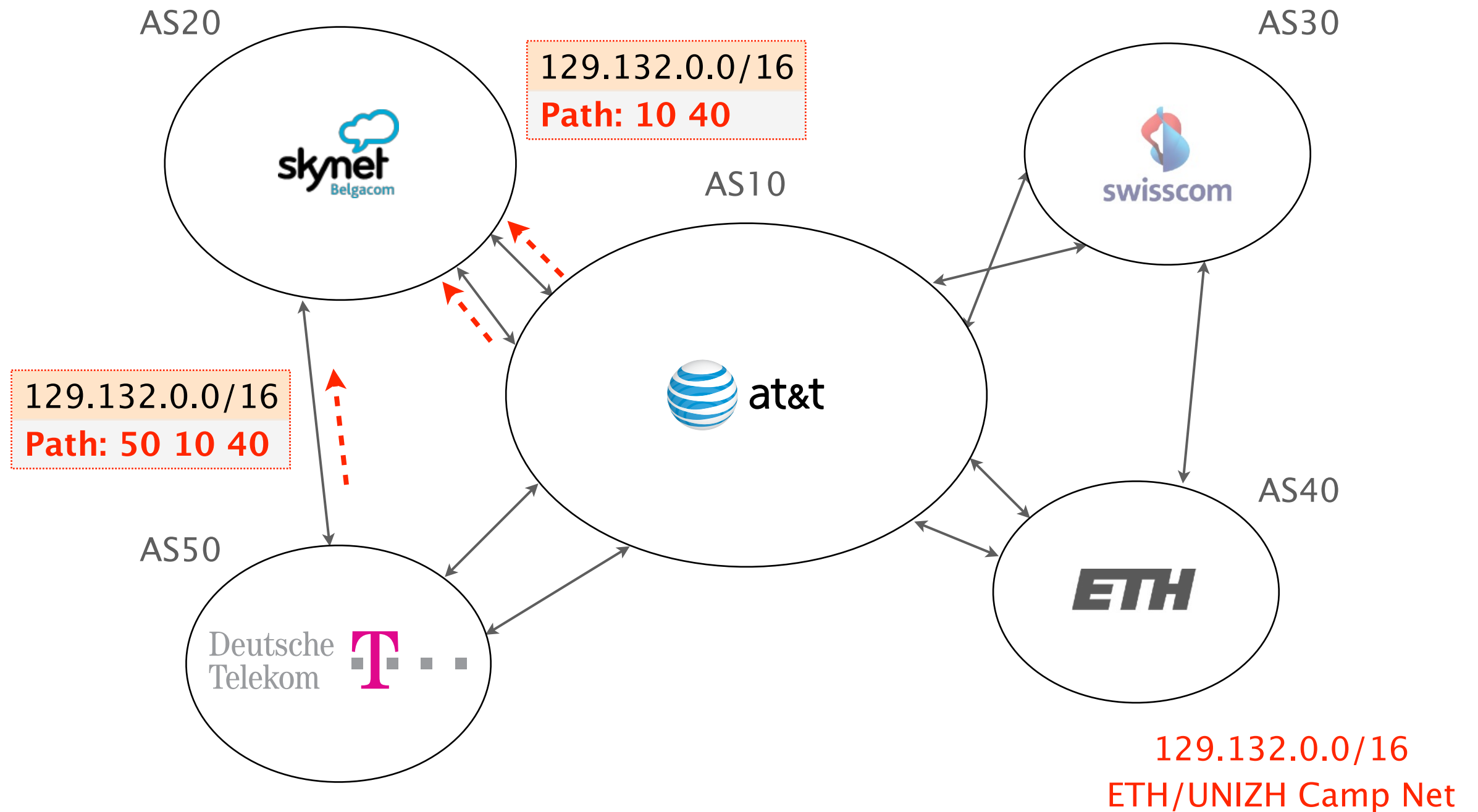
key idea advertise the **entire path** instead of distances

BGP announcements carry complete path information instead of distances



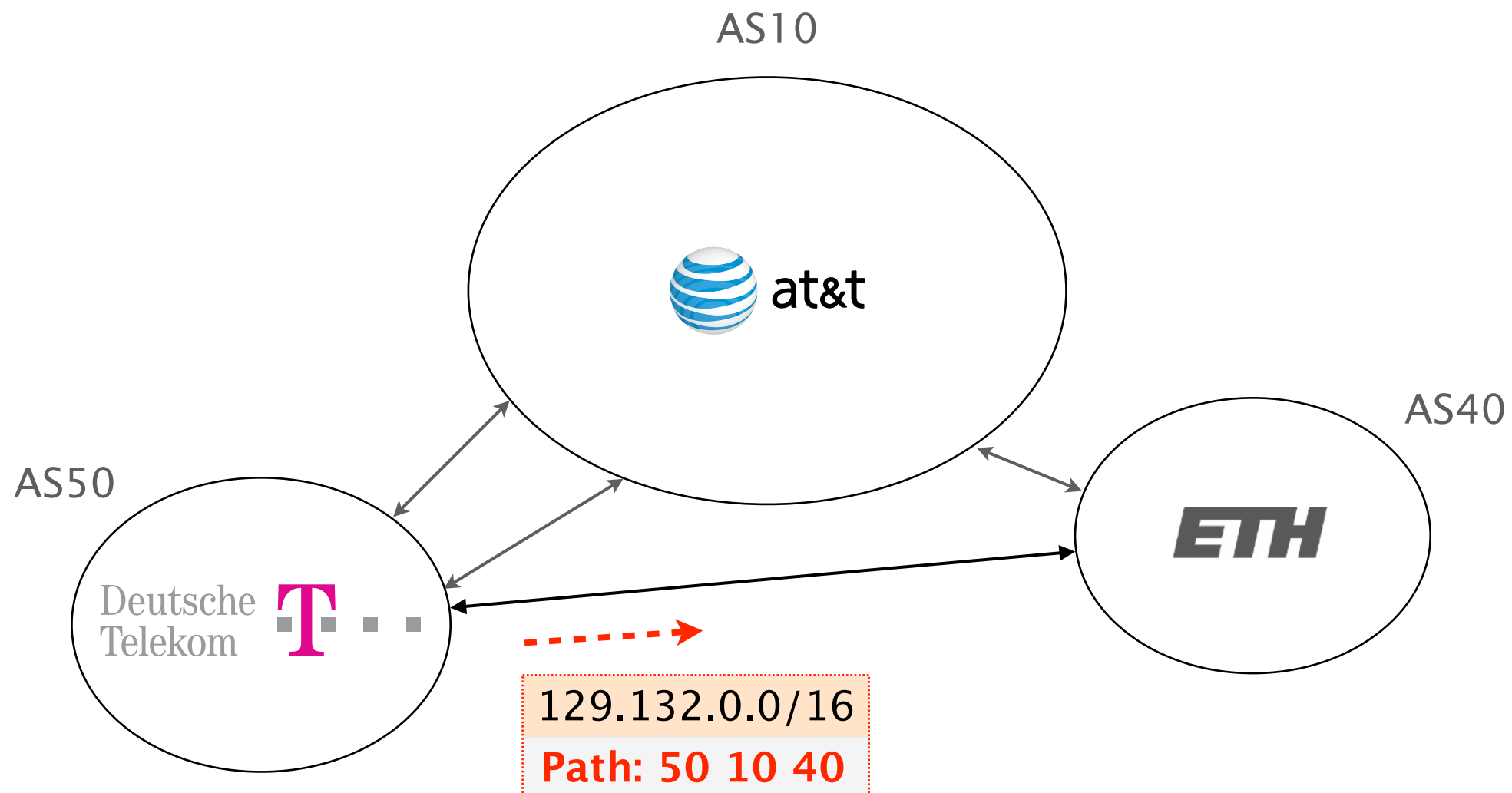
Each AS appends itself to the path
when it propagates announcements





Complete path information enables ASes to easily detect a loop

ETH sees itself in the path and discard the route



Life of a BGP router is made of three consecutive steps

while true:

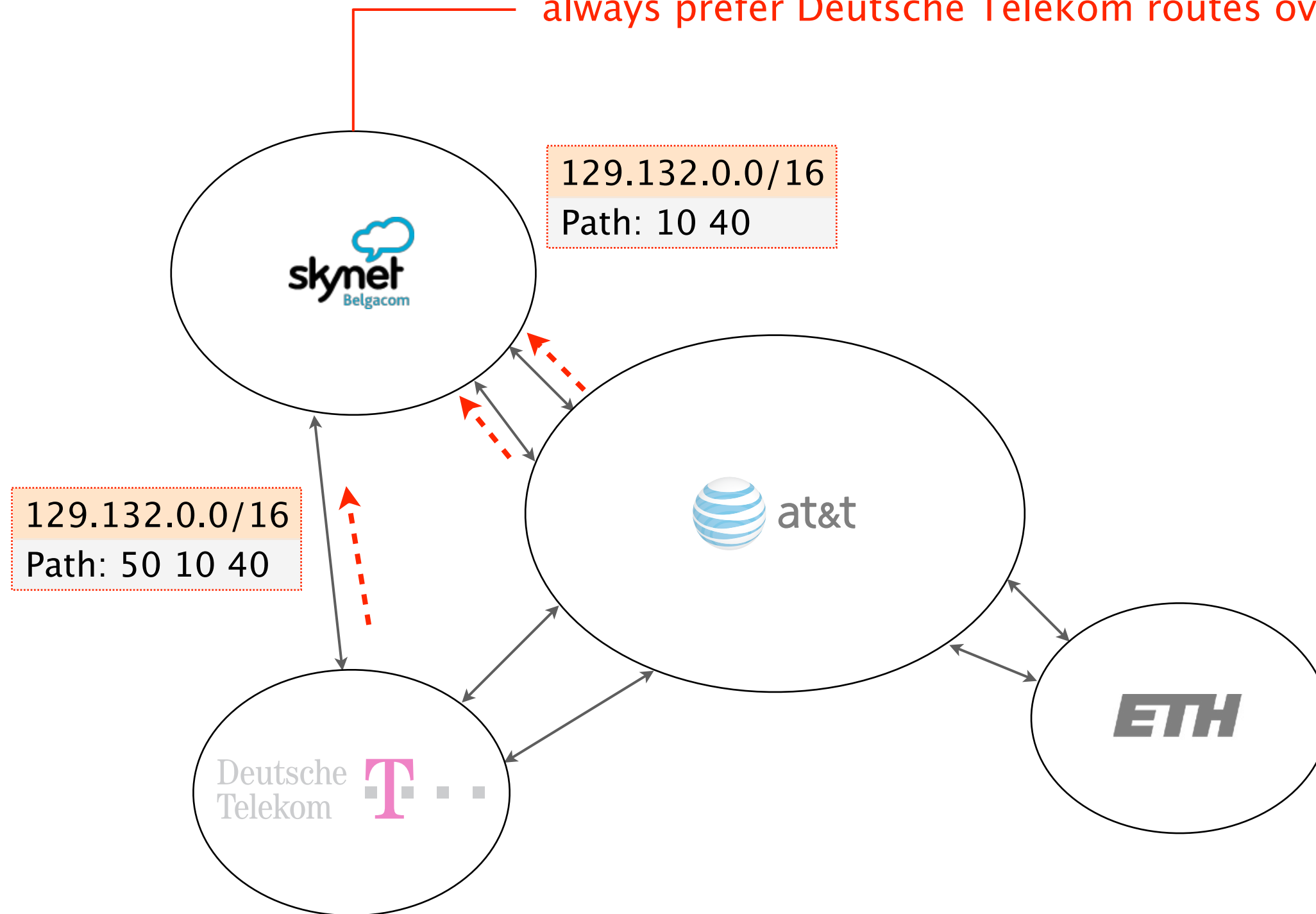
- receives routes from my neighbors
- select one best route for each prefix
- export the best route to my neighbors

Each AS can apply local routing policies

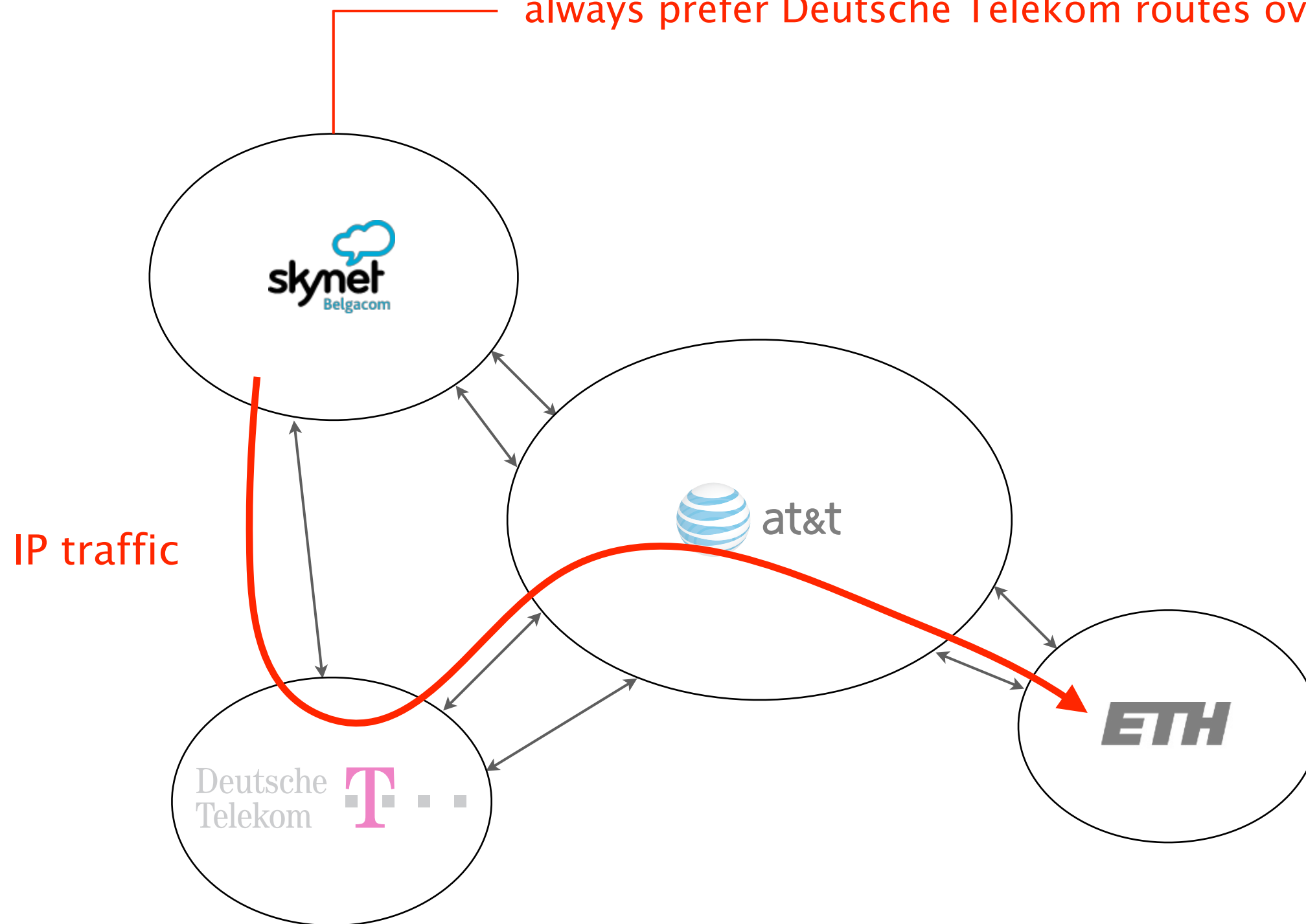
Each AS is free to

- select and use any path
preferably, the cheapest one

always prefer Deutsche Telekom routes over AT&T



always prefer Deutsche Telekom routes over AT&T

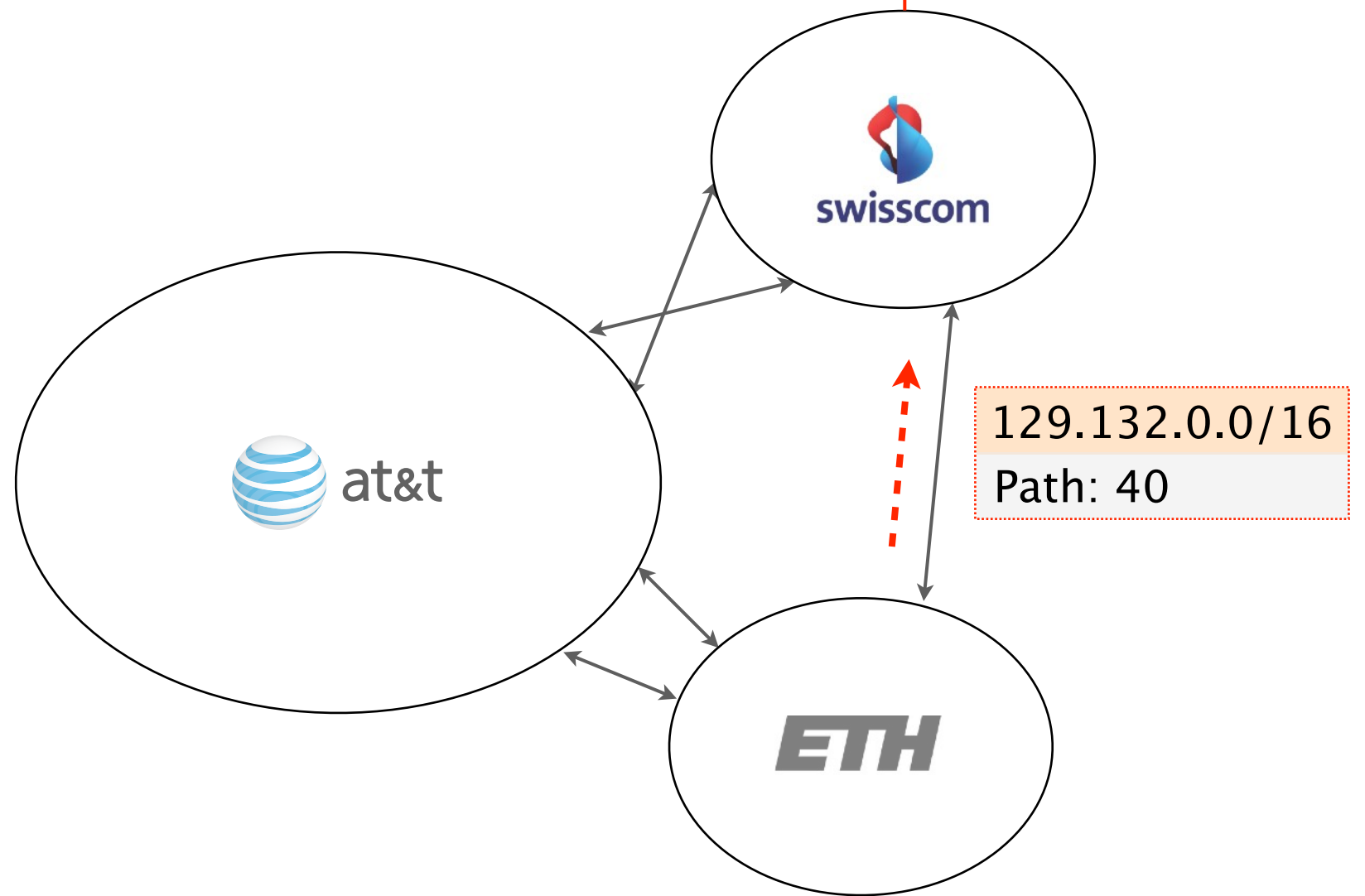


Each AS can apply local routing policies

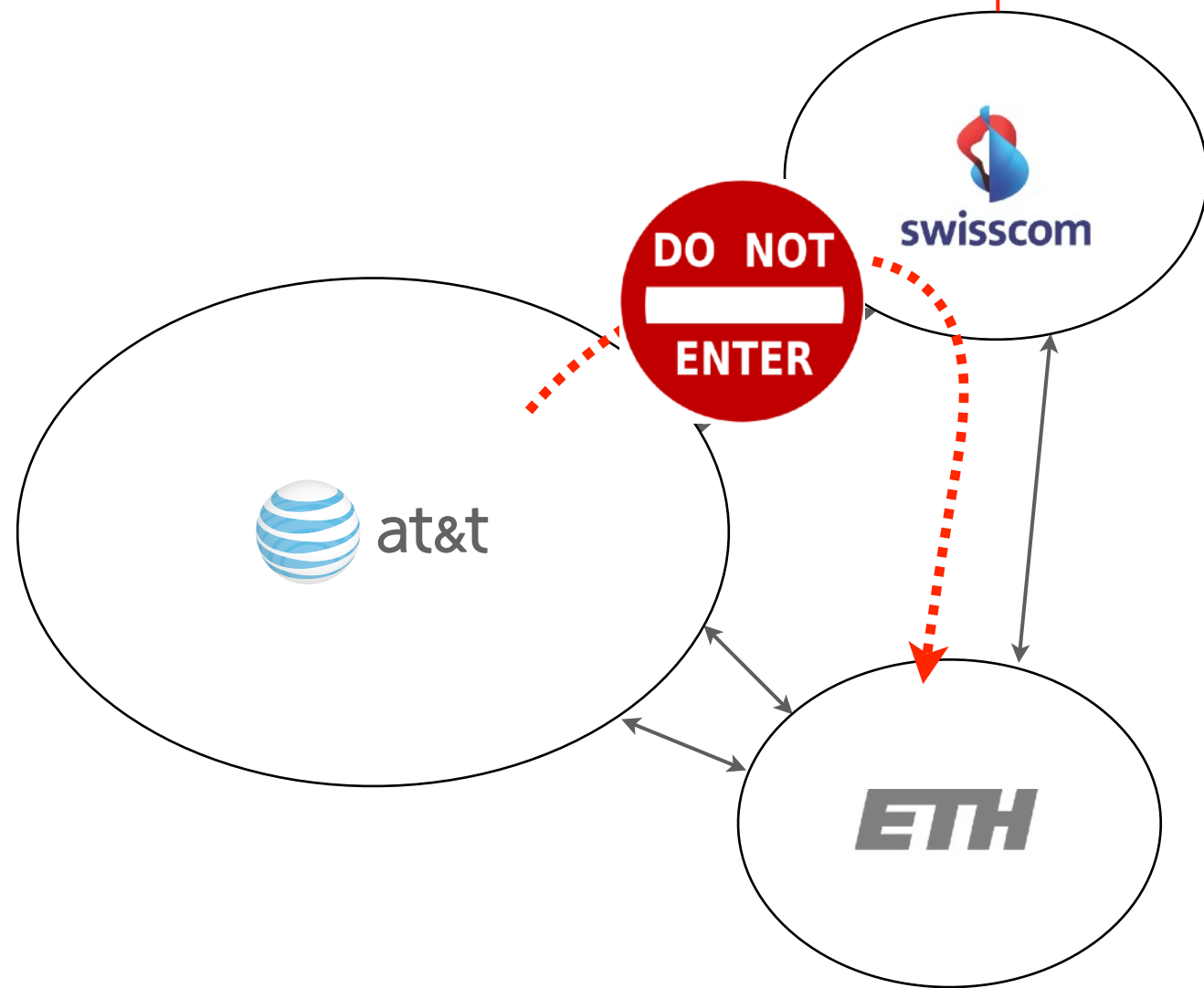
Each AS is free to

- select and use any path
preferably, the cheapest one
- decide which path to export (if any) to which neighbor
preferably, none to minimize carried traffic

do not export ETH routes to AT&T



do not export ETH routes to AT&T

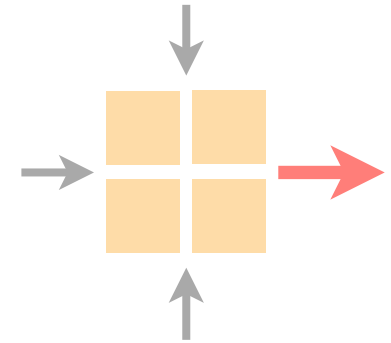


Next week on
Communication Networks

Internet routing policies

Communication Networks

Spring 2019



Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich (D-ITET)

March 25 2019