Computer Networks and Internet Technology

2021W703033 VO Rechnernetze und Internettechnik
Winter Semester 2021/22

Jan Beutel

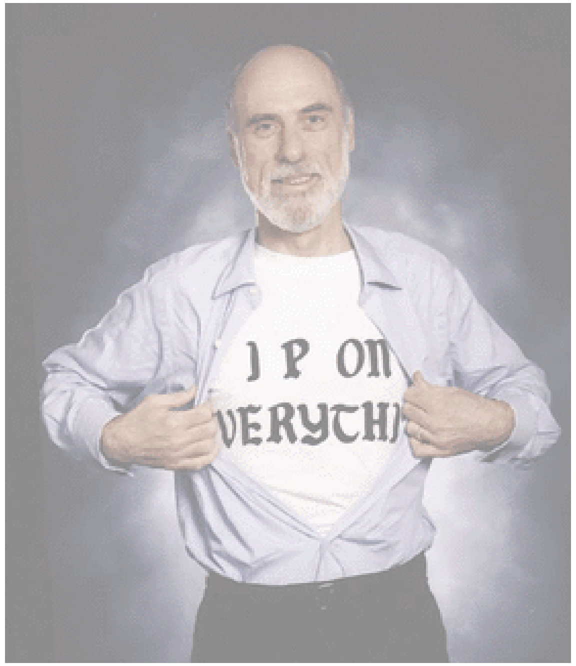
universität innsbruck

With materials and inspiration from Scott Shenker, Jennifer Rexford, Phil Levis, Adrian Perrig and Laurent Vanbever

Communication Networks and Internet Technology

Recap of last weeks lecture

Internet Protocol and Forwarding



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

82.130.102.10

01010010 10000010 01100110 00001010

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

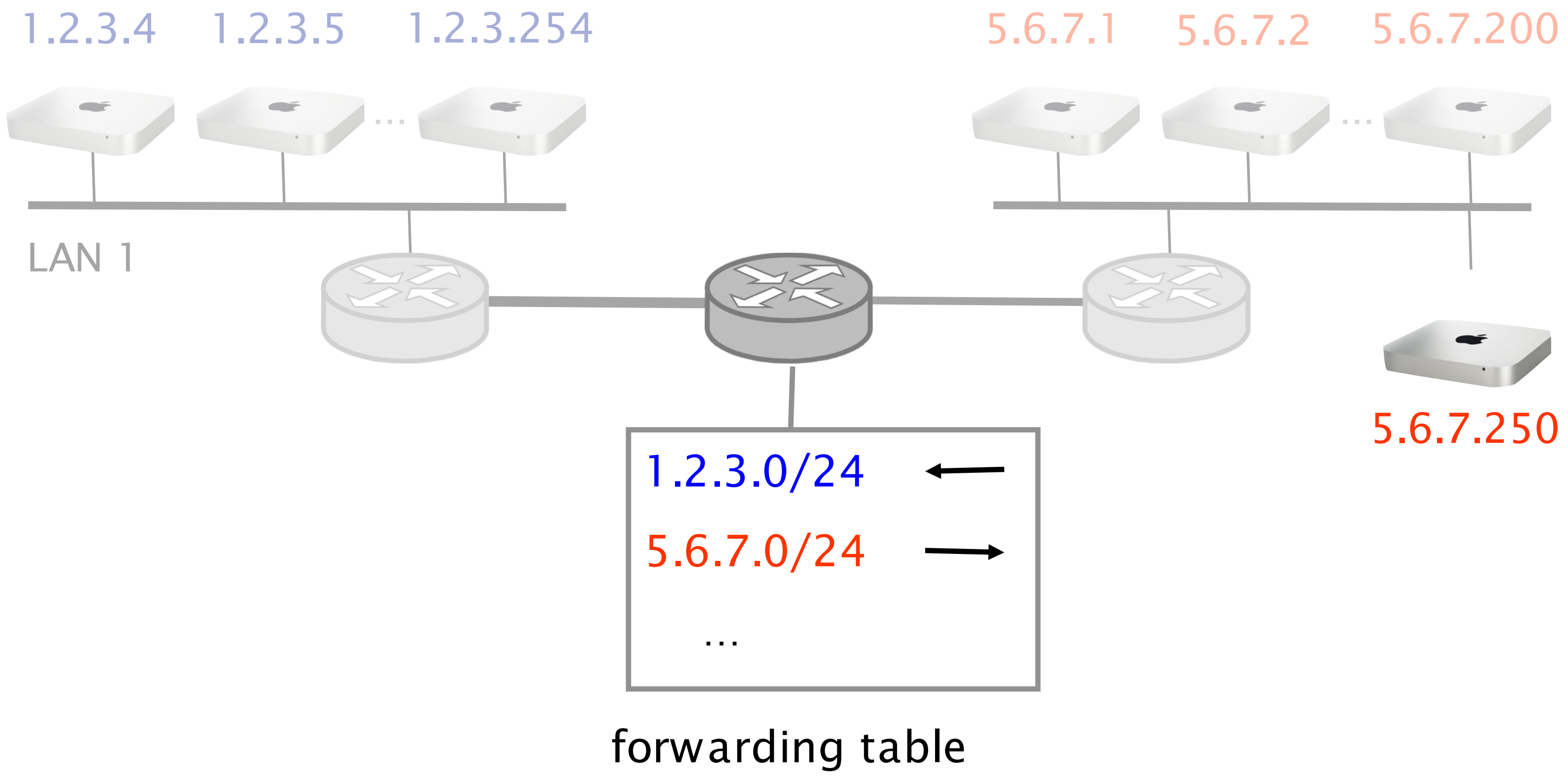
32 bits

01010010.10000010.01100110.00001010

prefix
identifies the network

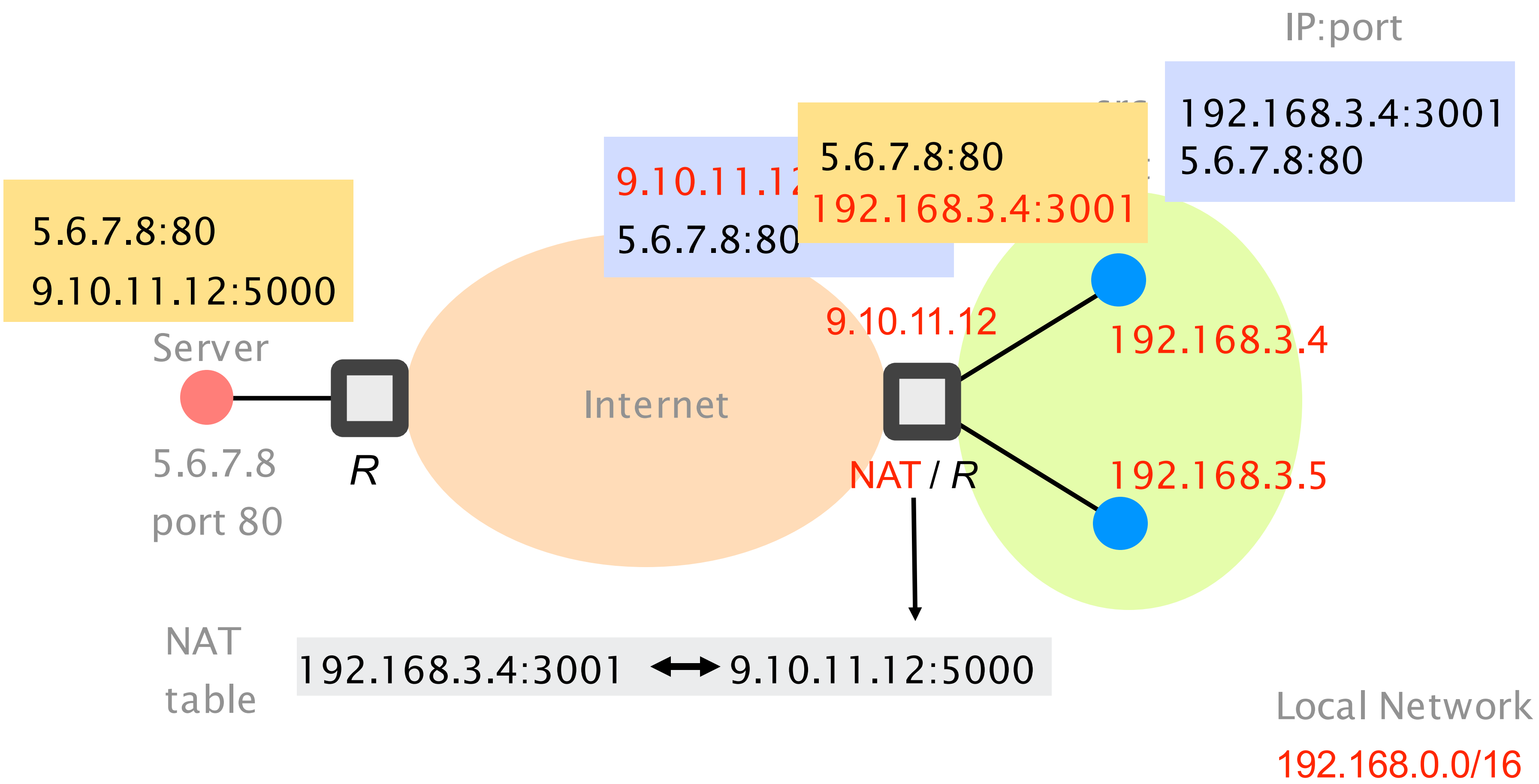
suffix
identifies the hosts
in the network

Hierarchical addressing enables to add new hosts without changing or adding forwarding rules

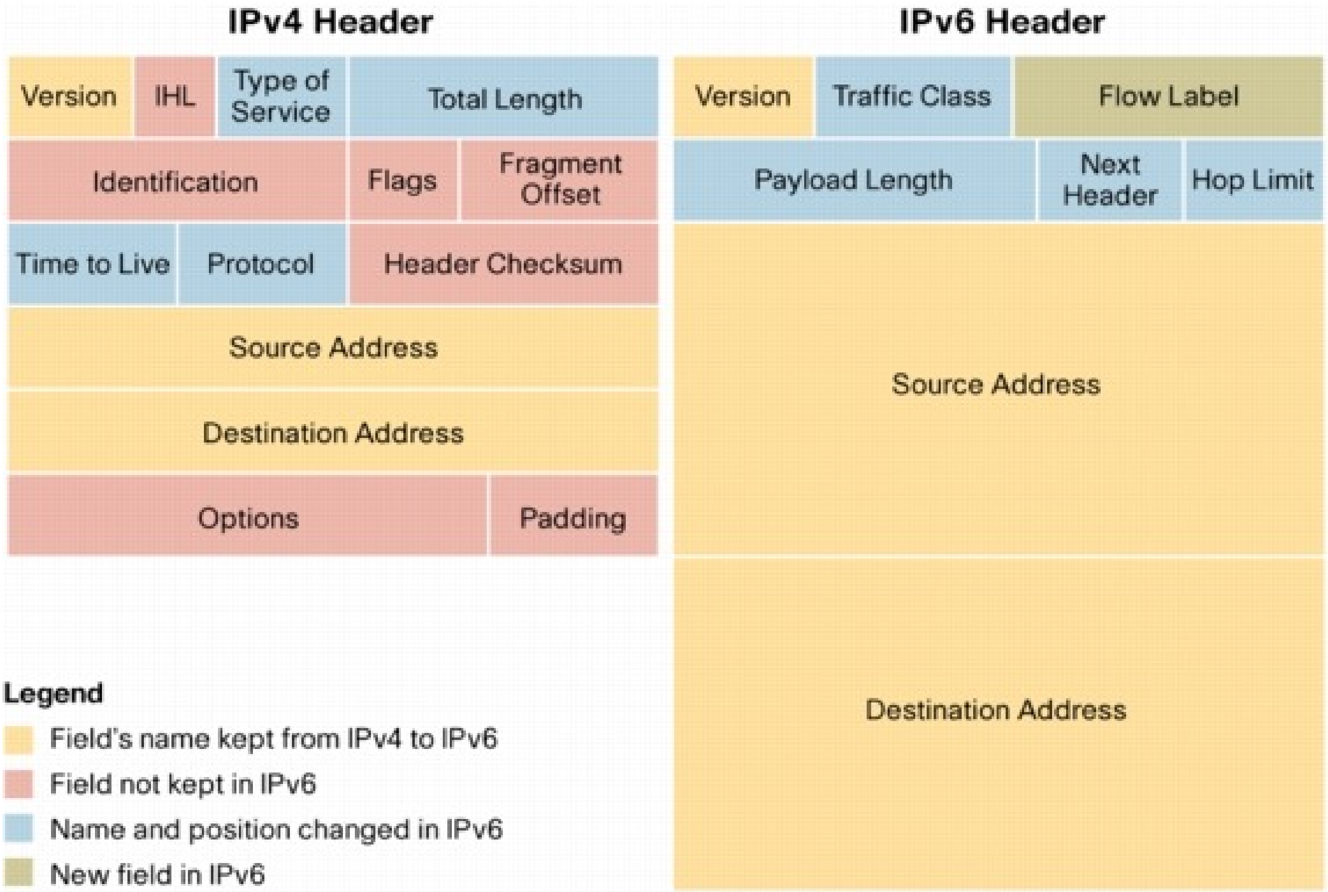


The Internet with NAT

Hosts behind NAT get a private address



IPv4 vs IPv6

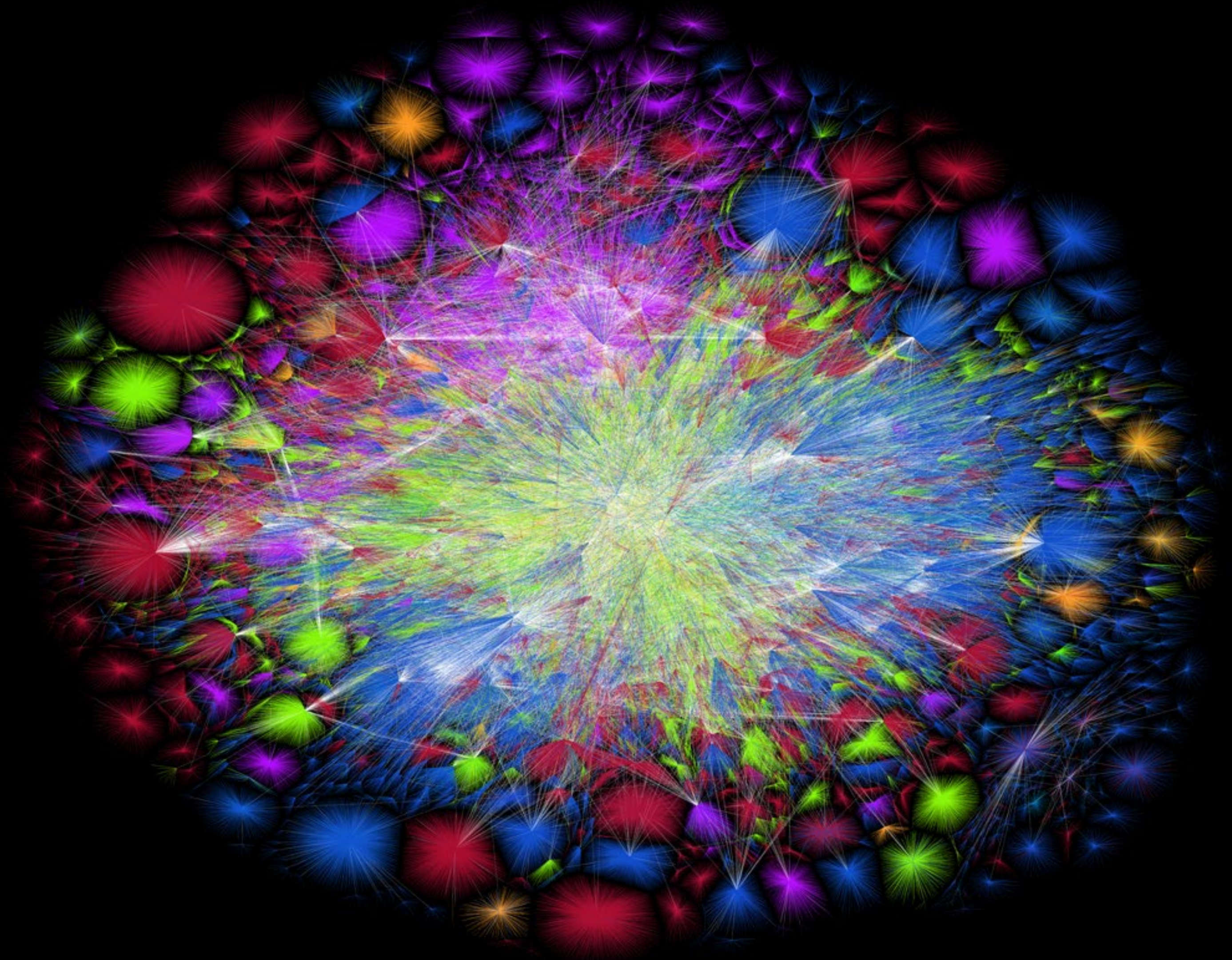


Source: http://bit.ly/1HXc2BS

Communication Networks and Internet Technology

This weeks lecture

Internet routing



http://www.opte.org

› traceroute www.google.at

› traceroute www.google.at

1 sr1t3-vl10.uibk.ac.at (138.232.0.25)

2 bfw-tech-bond0.uibk.ac.at (193.171.74.209)

3 br-tech-t3.uibk.ac.at (193.171.74.197)

4 ibk1.aco.net (193.171.19.41)

5 195.113.179.150 (195.113.179.150)

6 r98-bm.cesnet.cz (195.113.179.149)

7 195.113.235.109 (195.113.235.109)

8 r2-r93.cesnet.cz (195.113.157.70)

9 * * *

10 prg03s12-in-f3.1e100.net (142.251.36.131)

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

inter-domain
routing

Find paths between networks

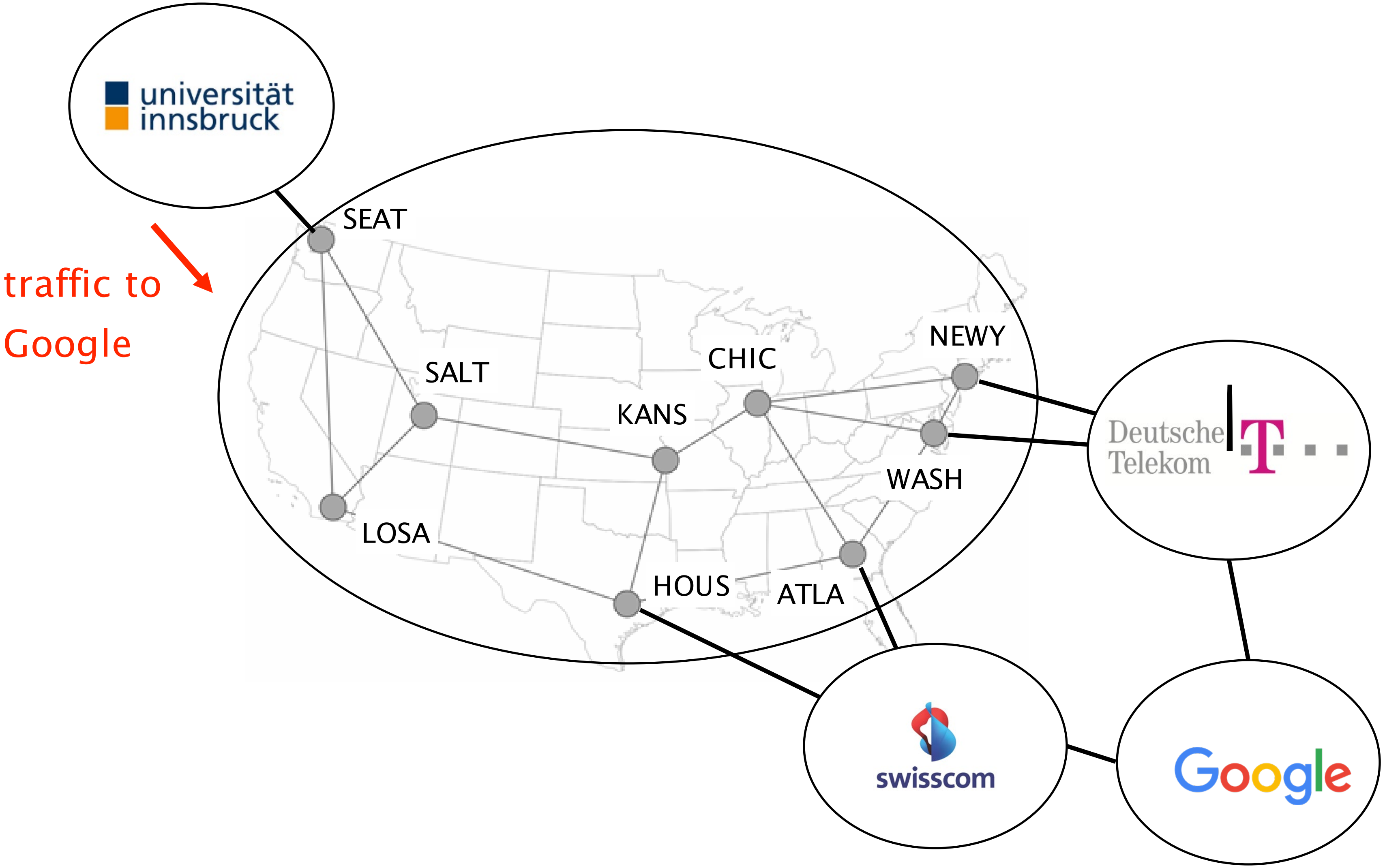
intra-domain
routing

Find paths within a network

inter-domain
routing

intra-domain
routing

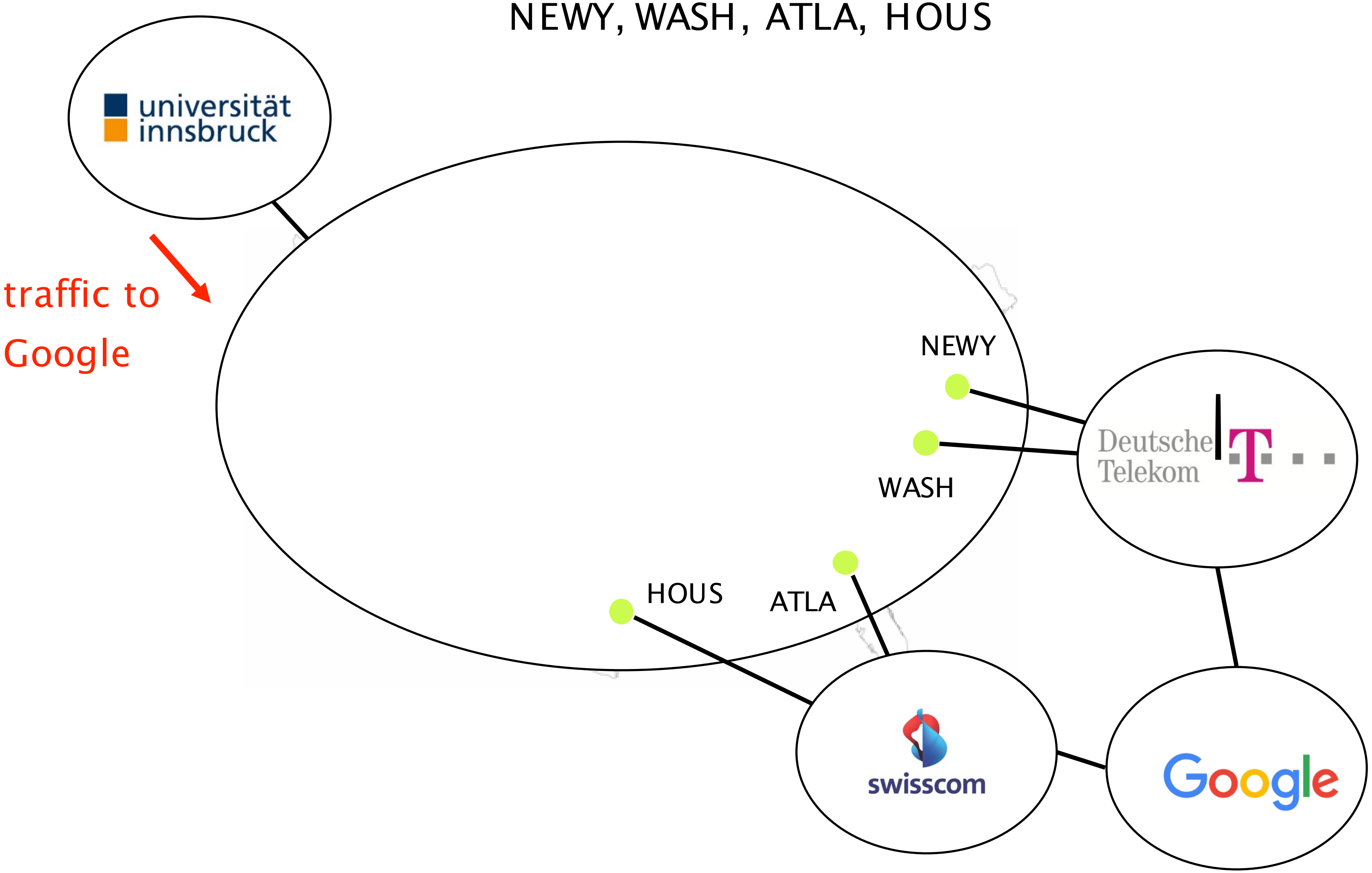
Find paths between networks



universität innsbruck
traffic to Google
SEAT
SALT
LOSA
KANS
CHIC
NEWY
WASH
HOUS
ATLA
Deutsche Telekom
swisscom
Google

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, ...

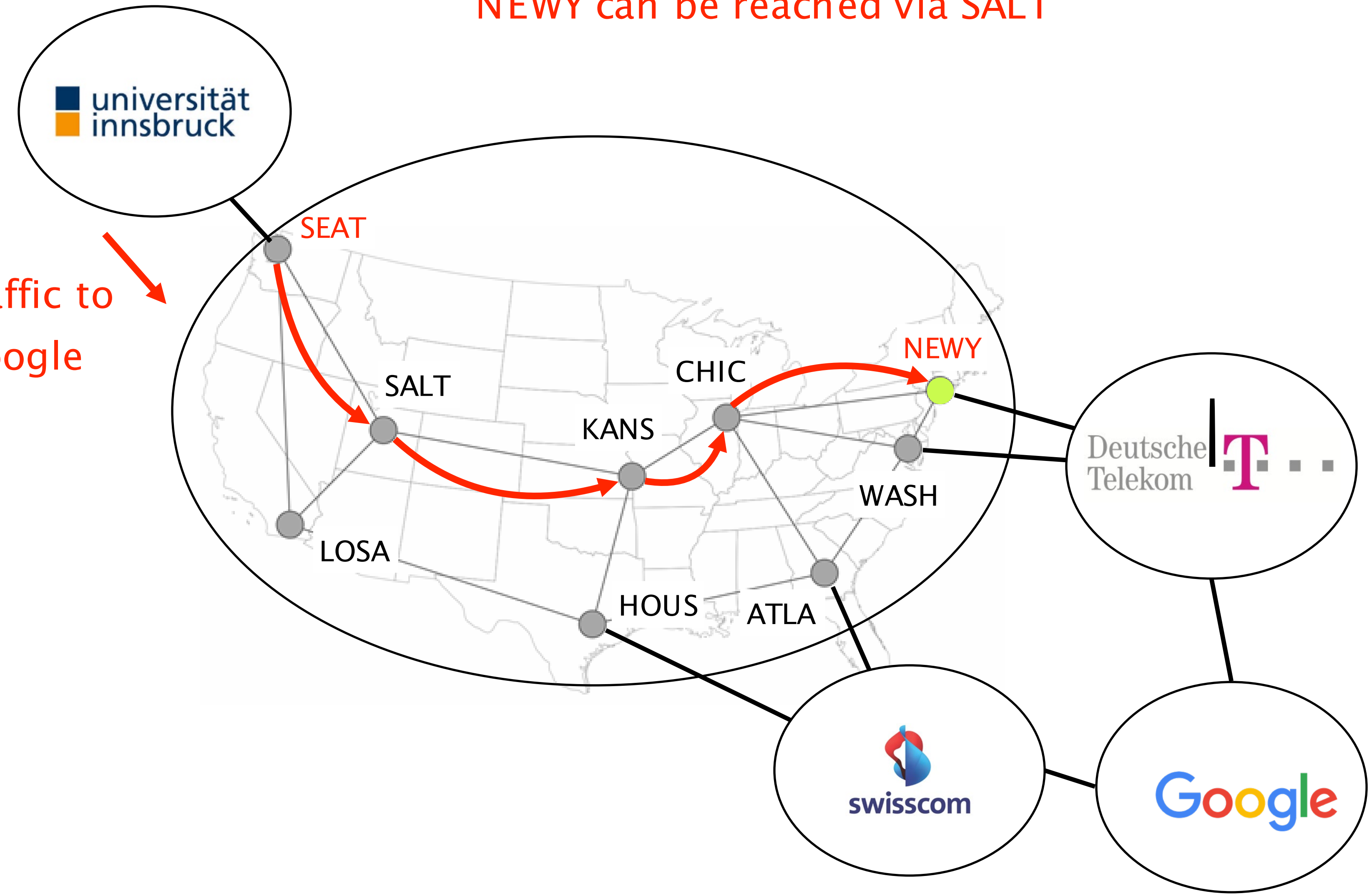
inter-domain
routing

intra-domain
routing

Find paths within a network

NEWY can be reached via SALT

traffic to Google



› traceroute **www.google.at**

```
sr1t3-vl10.uibk.ac.at
bfw-tech-bond0.uibk.ac.at
br-tech-t3.uibk.ac.at
ibk1.aco.net
195.113.179.150
r98-bm.cesnet.cz
195.113.235.109
r2-r93.cesnet.cz
* * *
prg03s12-in-f3.1e100.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

intra-domain routing

intra-domain routing

intra-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

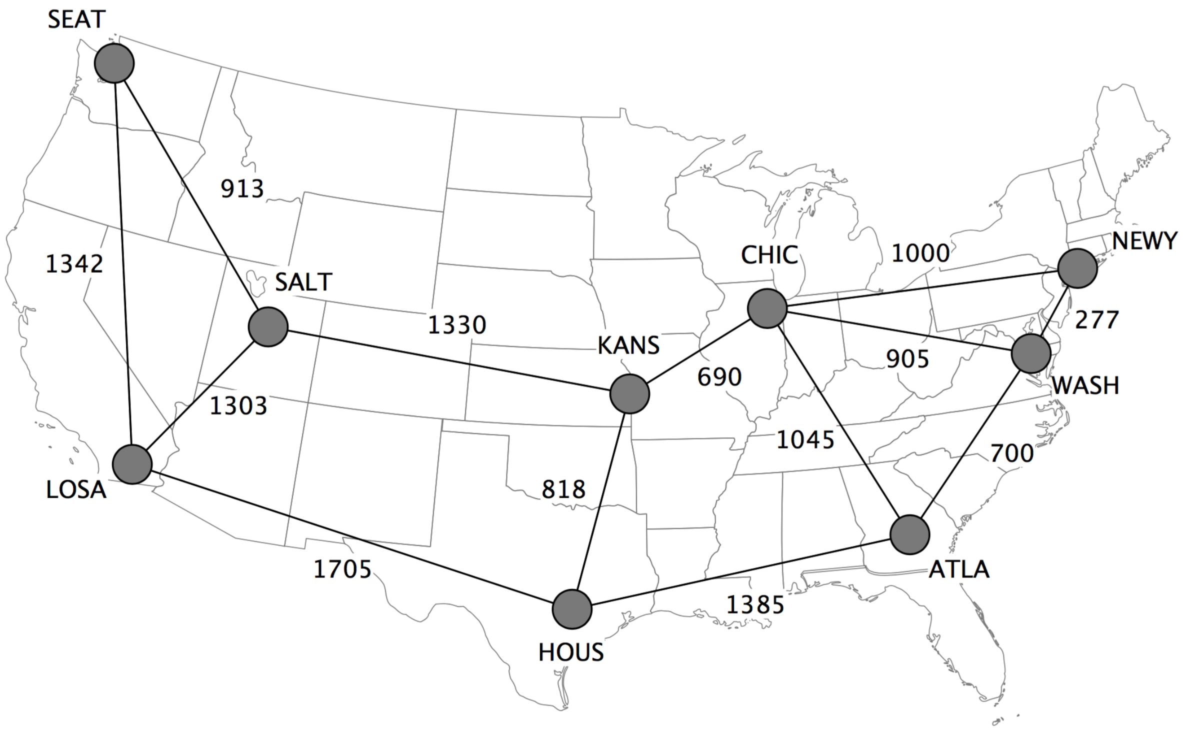
A good path is a path that minimizes some network-wide metric

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

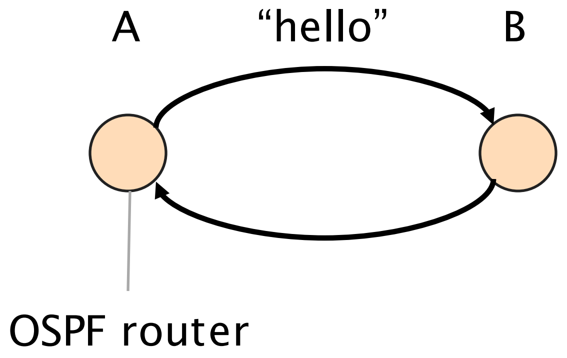
time-to-live for each link-state

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

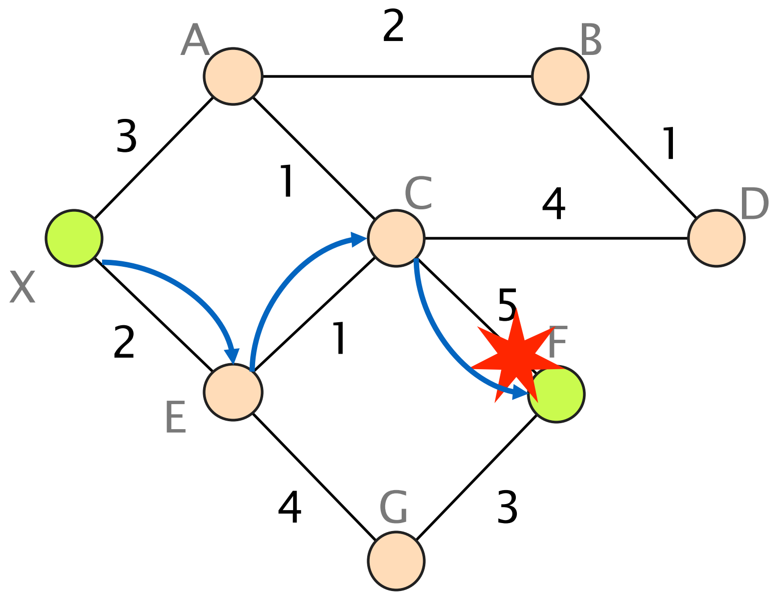
control-plane consistency — all nodes have the same link-state database

necessary

forwarding validity — the global forwarding state directs packet to its destination

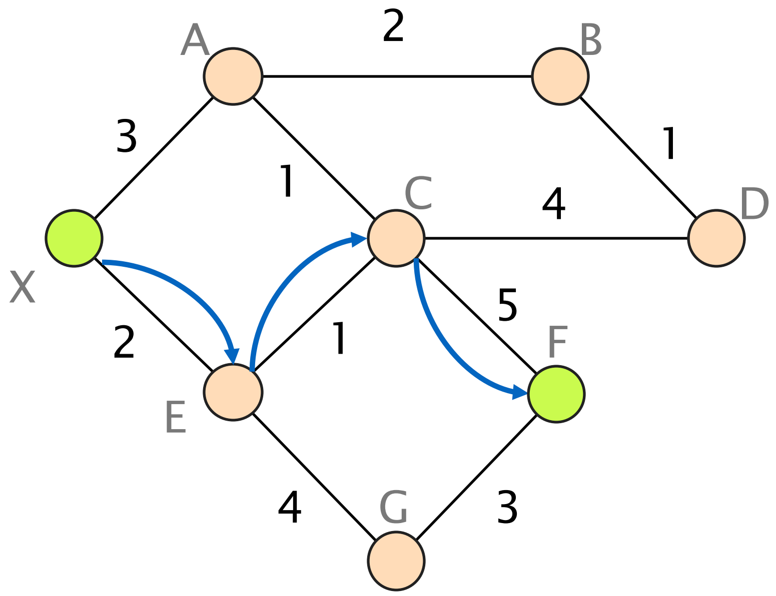
Inconsistencies lead to transient disruptions
in the form of blackholes or forwarding loops

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

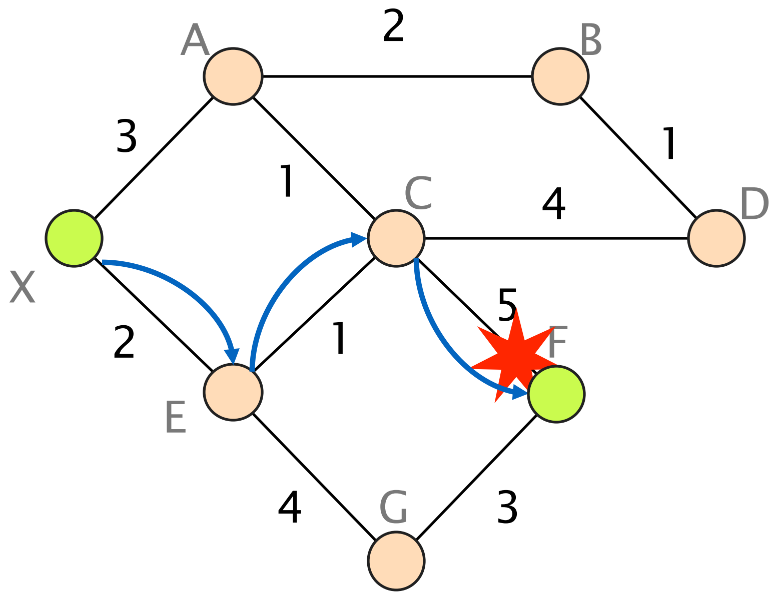


depends on the timeout for detecting lost hellos

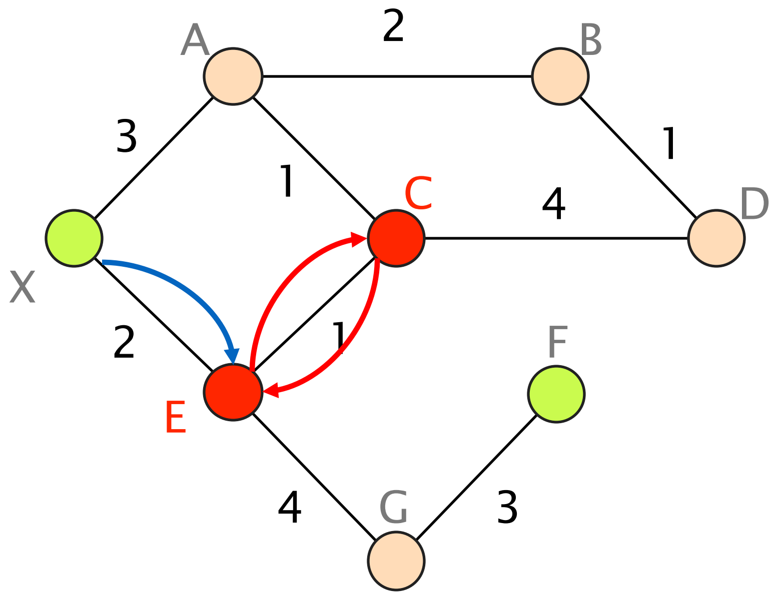
Transient loops appear due to inconsistent link-state databases



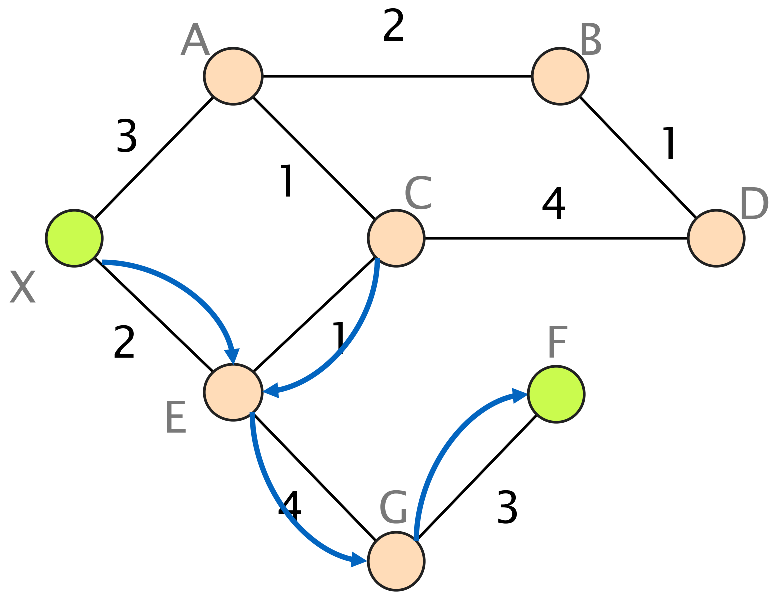
Initial forwarding state



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

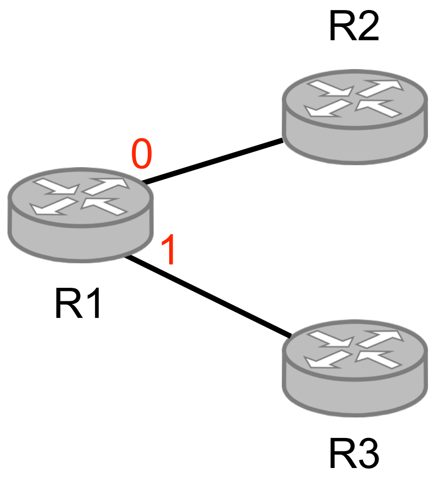
factors	time the routers take for…
detection	realizing that a link or a neighbor is down
flooding	flooding the news to the entire network
computation	recomputing shortest-paths using Dijkstra
table update	updating their forwarding table

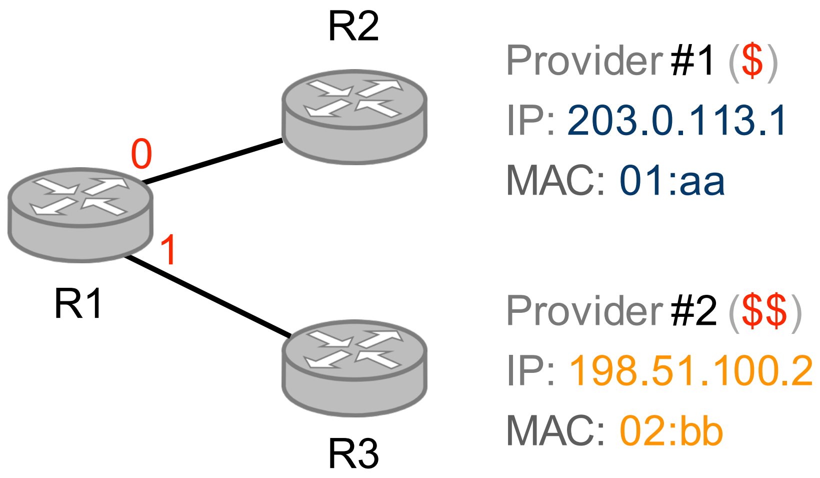
In practice, network convergence time is mostly driven by table updates

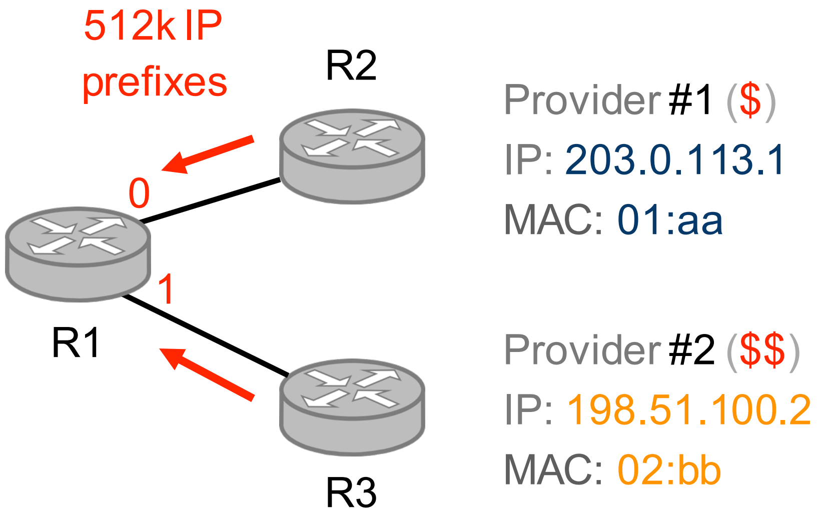
	time	improvements
detection	few ms	smaller timers
flooding	few ms	high-priority flooding
computation	few ms	incremental algorithms
table update	potentially, *minutes!*	better table design

table update potentially, *minutes!* **better table design**

R1

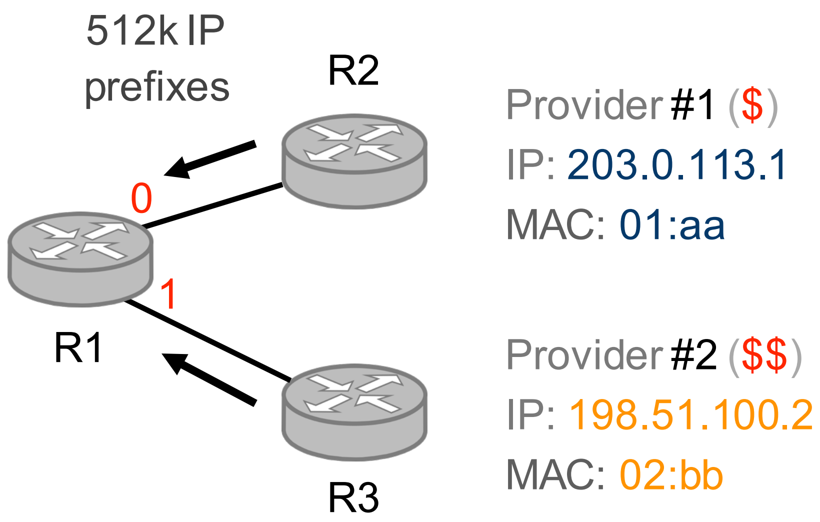

R2
0
1
R1
R3


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


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

R1's Forwarding Table

prefix	Next-Hop

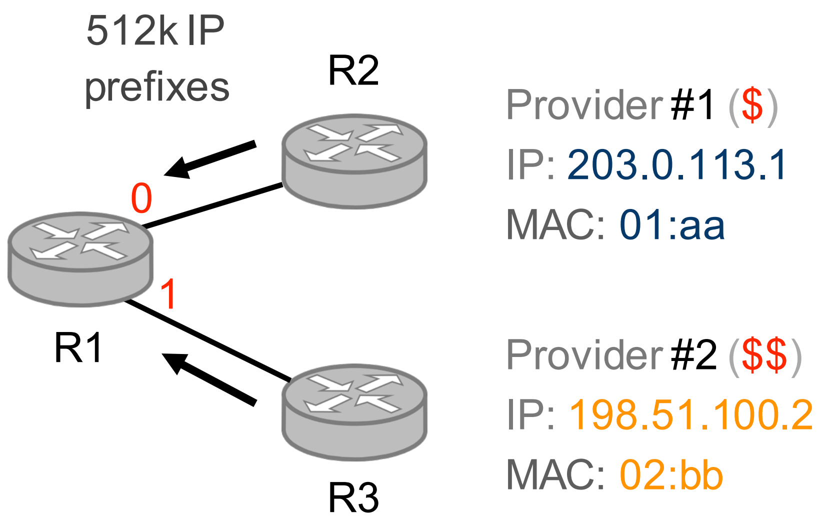


All 512k entries point to R2

because it is cheaper

R1's Forwarding Table

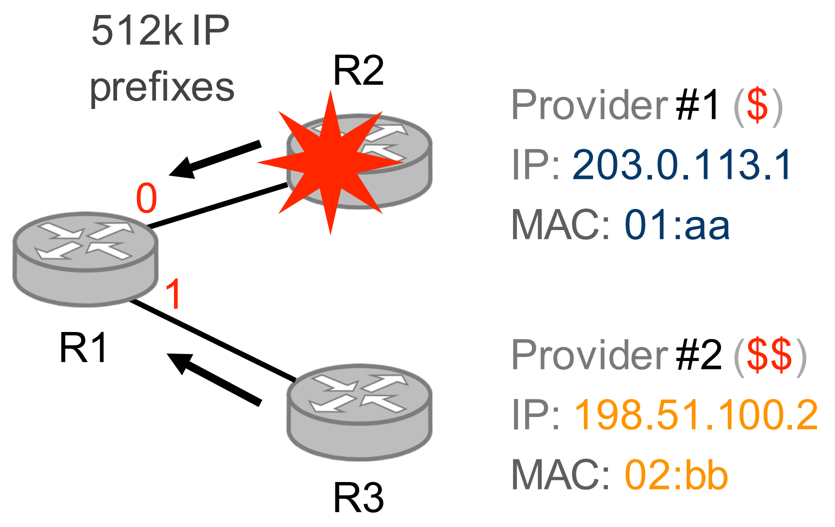
	prefix	Next-Hop
1	1.0.0.0/24	(01:aa, 0)
2	1.0.1.0/16	(01:aa, 0)
…	…	…
256k	100.0.0.0/8	(01:aa, 0)
…	…	…
512k	200.99.0.0/24	(01:aa, 0)



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

R1's Forwarding Table

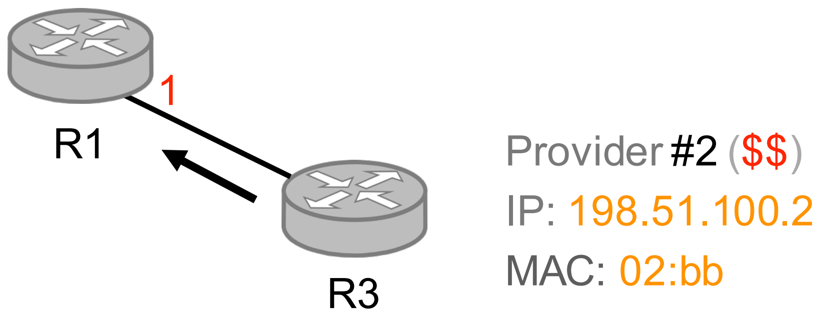
	prefix	Next-Hop
1	1.0.0.0/24	(01:aa, 0)
2	1.0.1.0/16	(01:aa, 0)
…	…	…
256k	100.0.0.0/8	(01:aa, 0)
…	…	…
512k	200.99.0.0/24	(01:aa, 0)



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

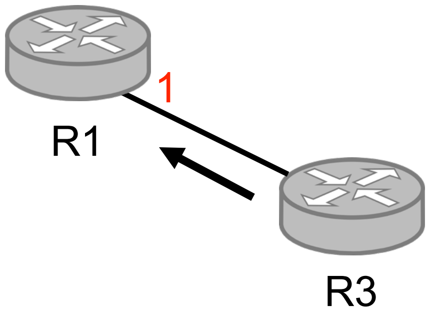
R1's Forwarding Table

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



R1's Forwarding Table

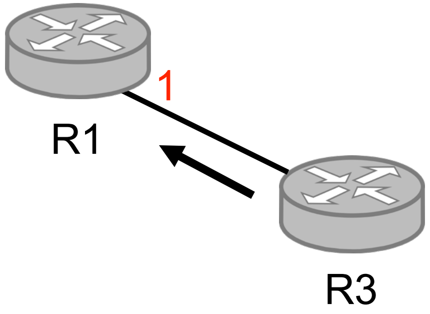
	prefix	Next-Hop
1	1.0.0.0/24	(02:bb, 1)
2	1.0.1.0/16	(01:aa, 0)
…	…	…
256k	100.0.0.0/8	(01:aa, 0)
…	…	…
512k	200.99.0.0/24	(01:aa, 0)



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

R1's Forwarding Table

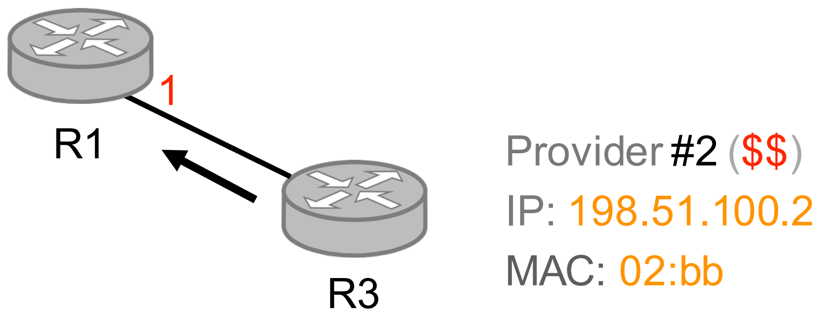
	prefix	Next-Hop
1	1.0.0.0/24	(02:bb, 1)
2	1.0.1.0/16	(02:bb, 1)
…	…	…
256k	100.0.0.0/8	(01:aa, 0)
…	…	…
512k	200.99.0.0/24	(01:aa, 0)



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

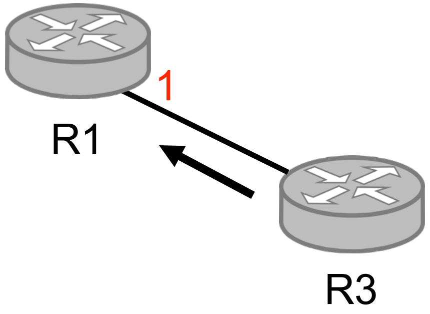
R1’s Forwarding Table

	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)



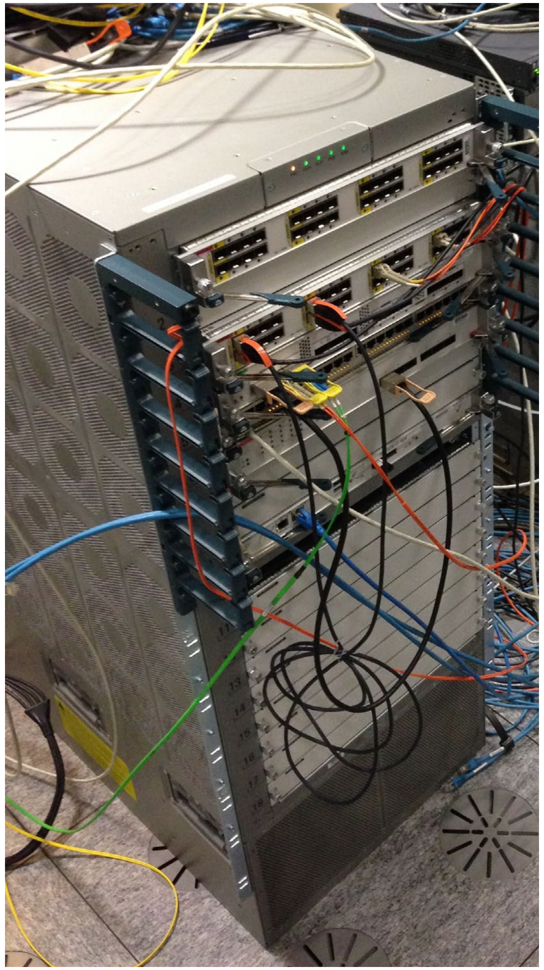
R1's Forwarding Table

	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

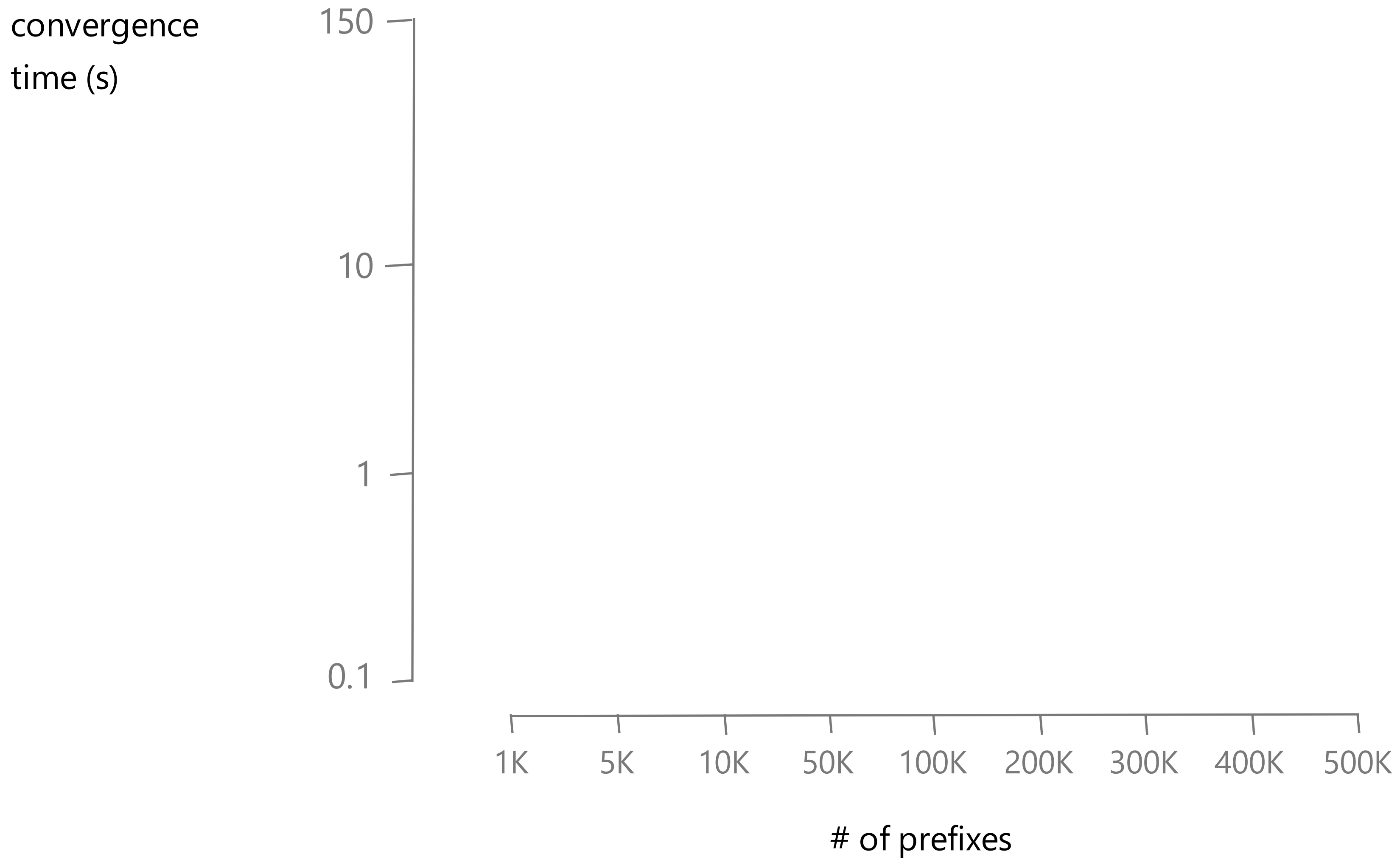
How long does it take for routers to converge?

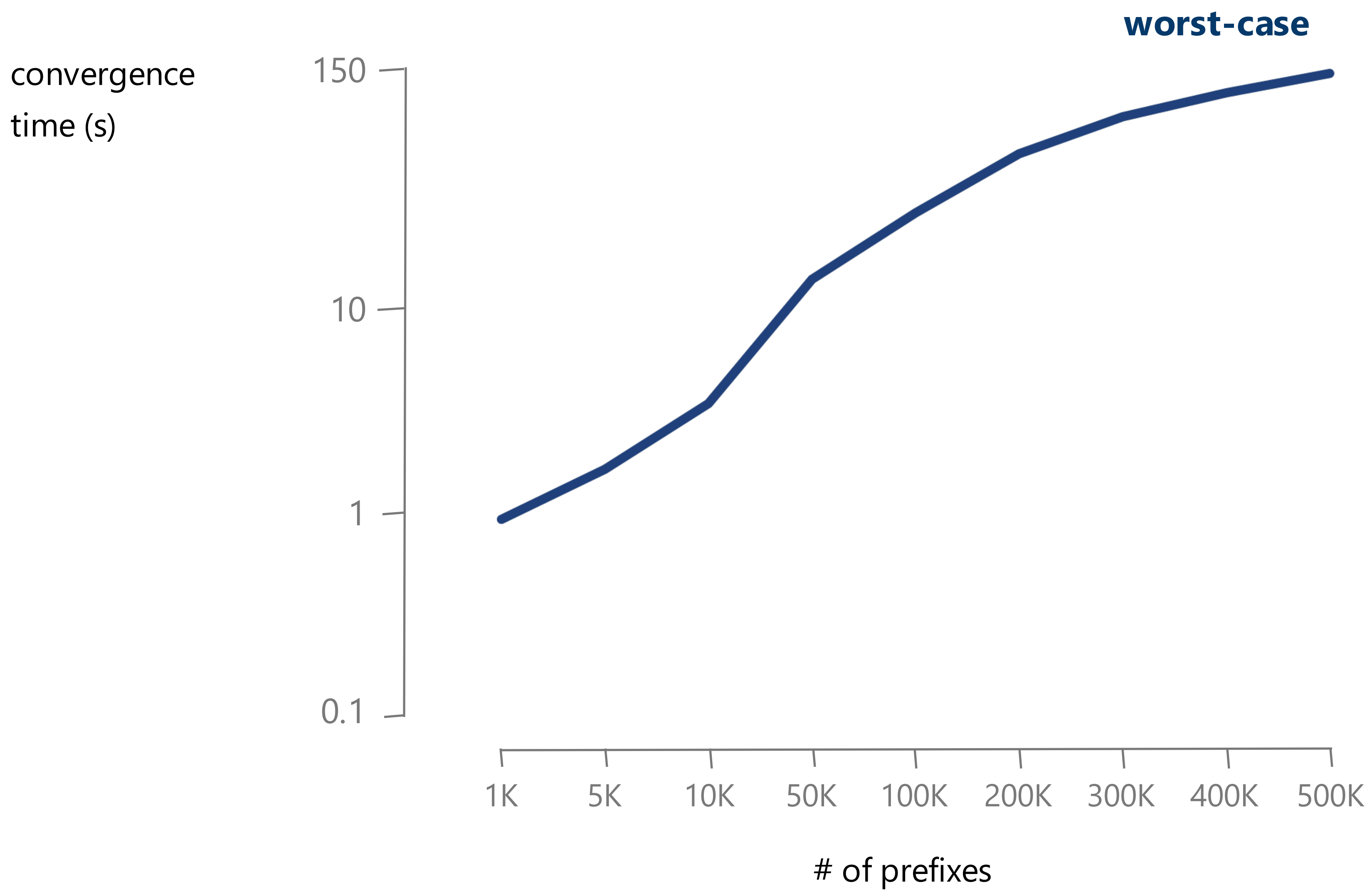


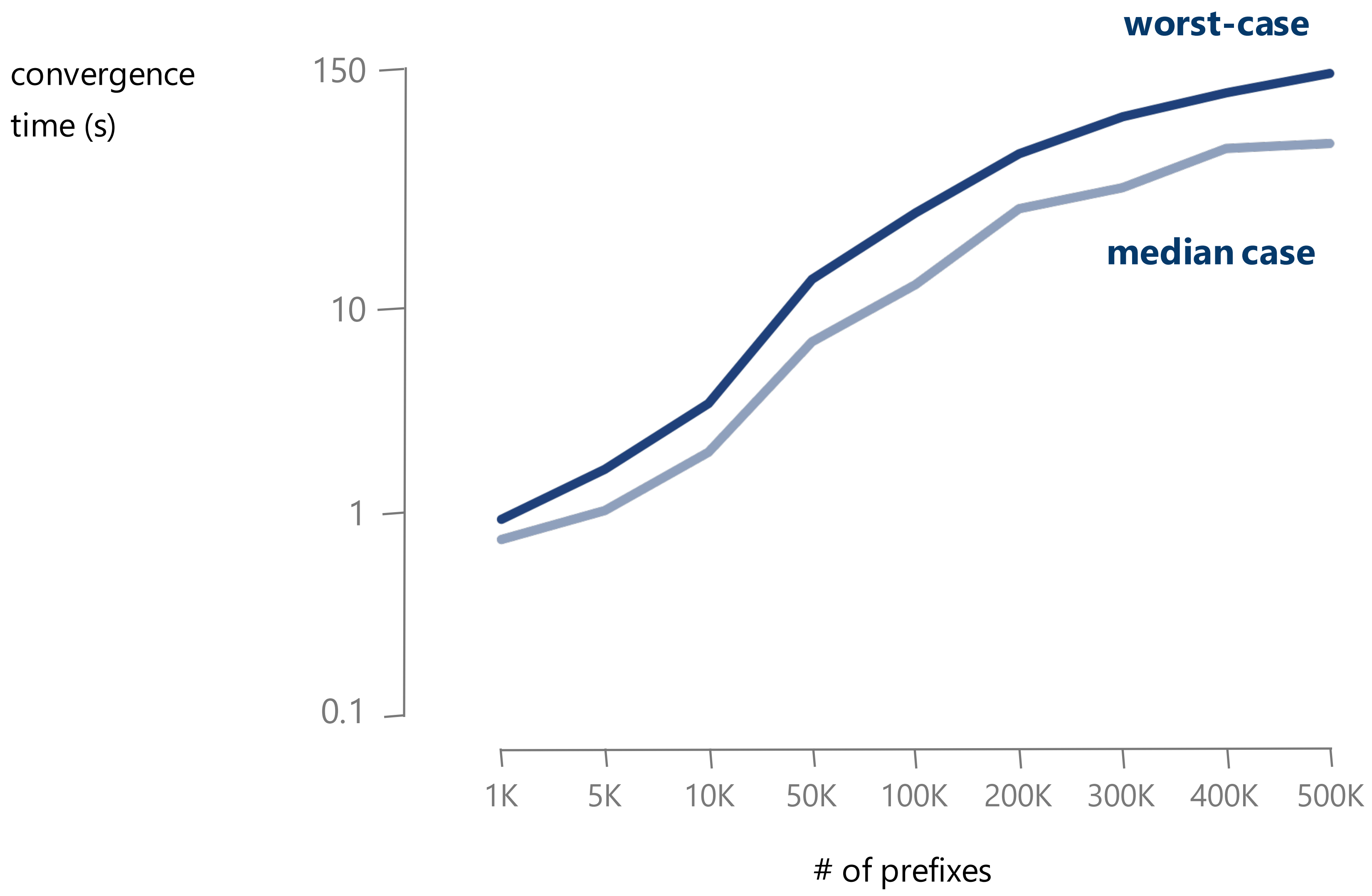
Cisco Nexus 9k

recent routers

 Deployed at ETH Zurich

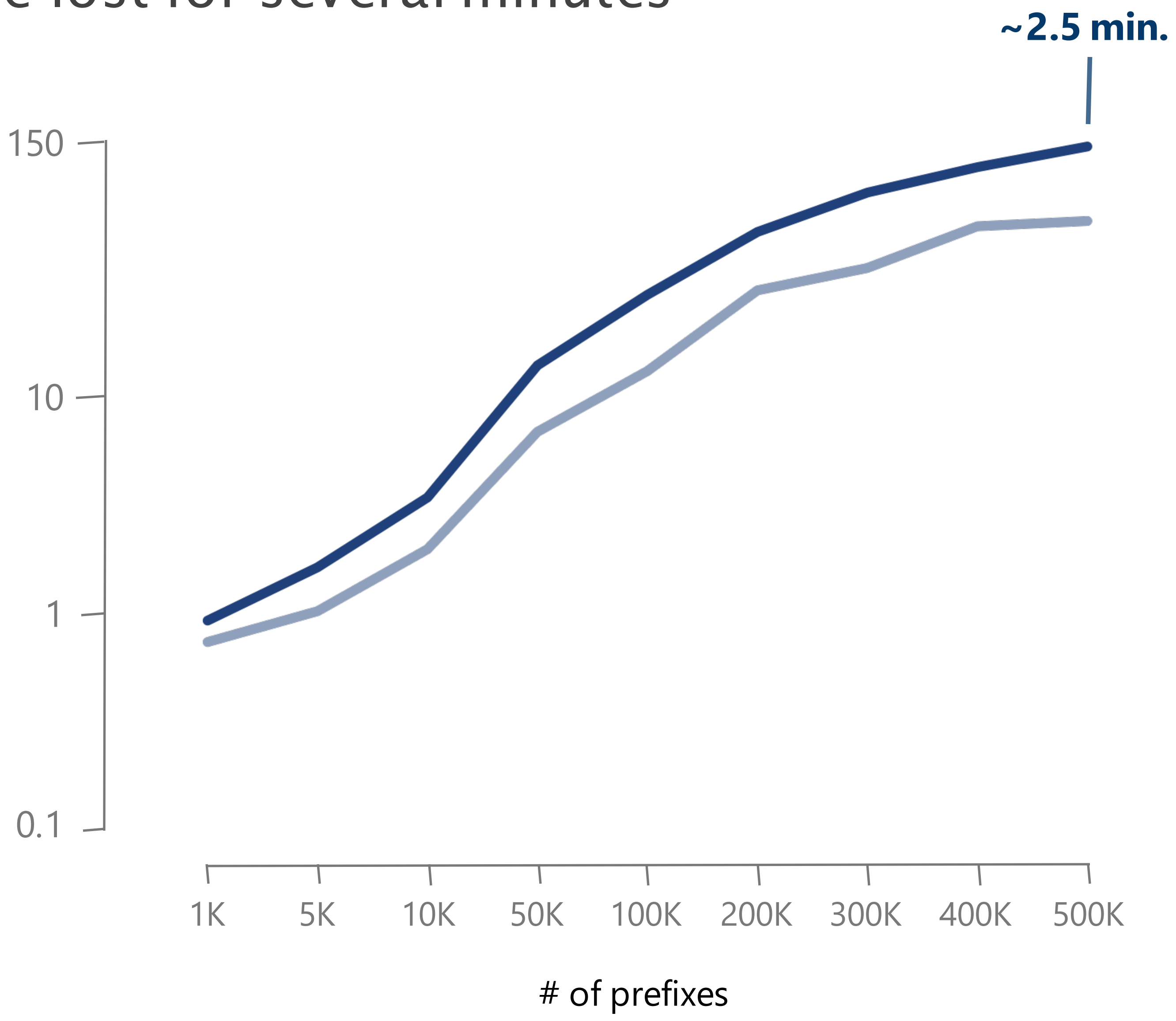

convergence
time (s)
150
10
1
0.1
1K
5K
10K
50K
100K
200K
300K
400K
500K
# of prefixes


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



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

Traffic can be lost for several minutes



The problem is that forwarding tables are flat

Entries do not share any information

even if they are identical

Upon failure, all of them have to be updated

inefficient, but also unnecessary

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)

port 0

port 1

… with that

Router Forwarding Table

Mapping table

	prefix	pointer
1	1.0.0.0/24	0x666
2	1.0.1.0/16	0x666
…	…	…
256k	100.0.0.0/8	0x666
…	…	…
512k	200.99.0.0/24	0x666

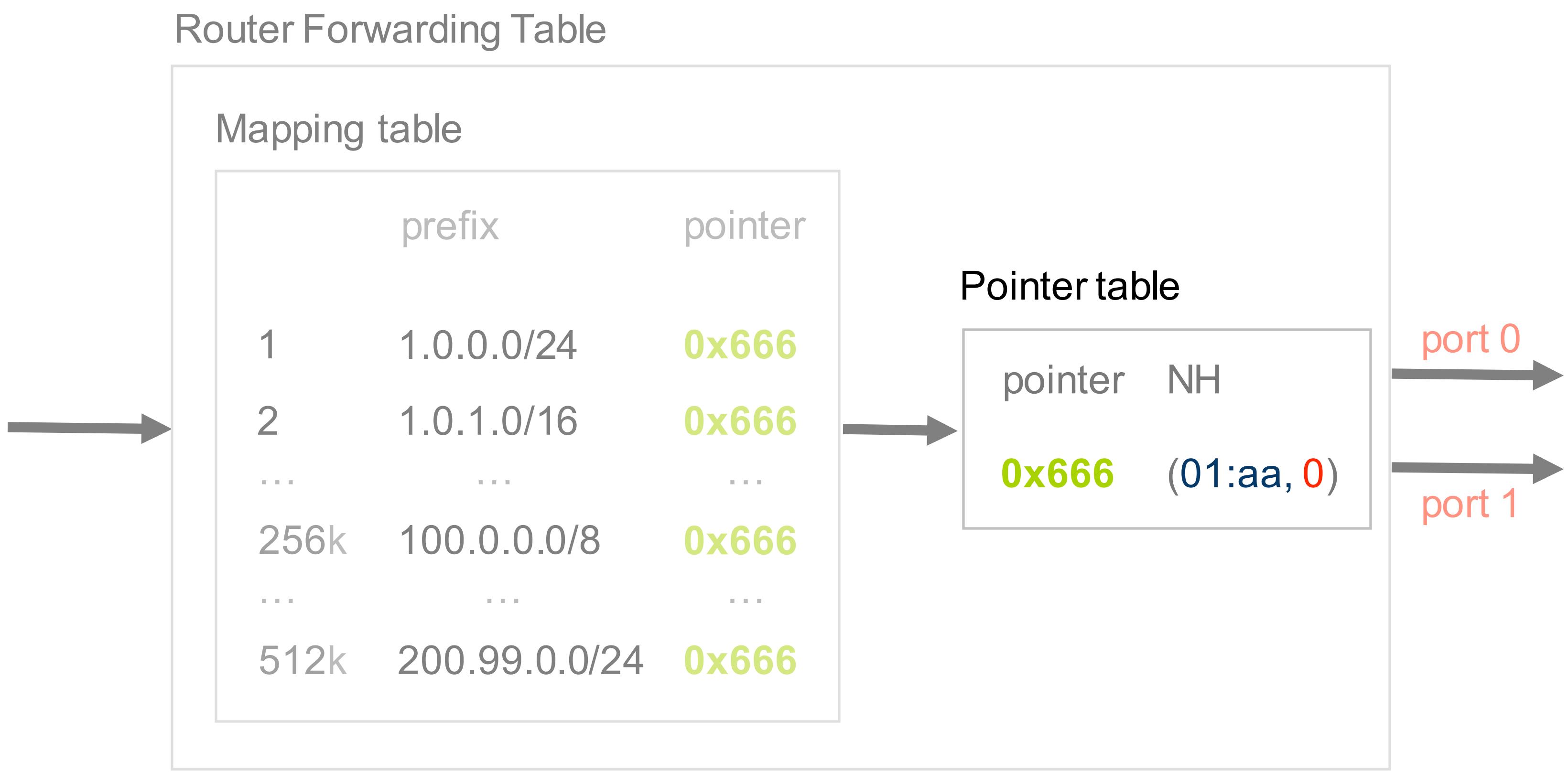
Pointer table

pointer	NH
0x666	(01:aa, 0)

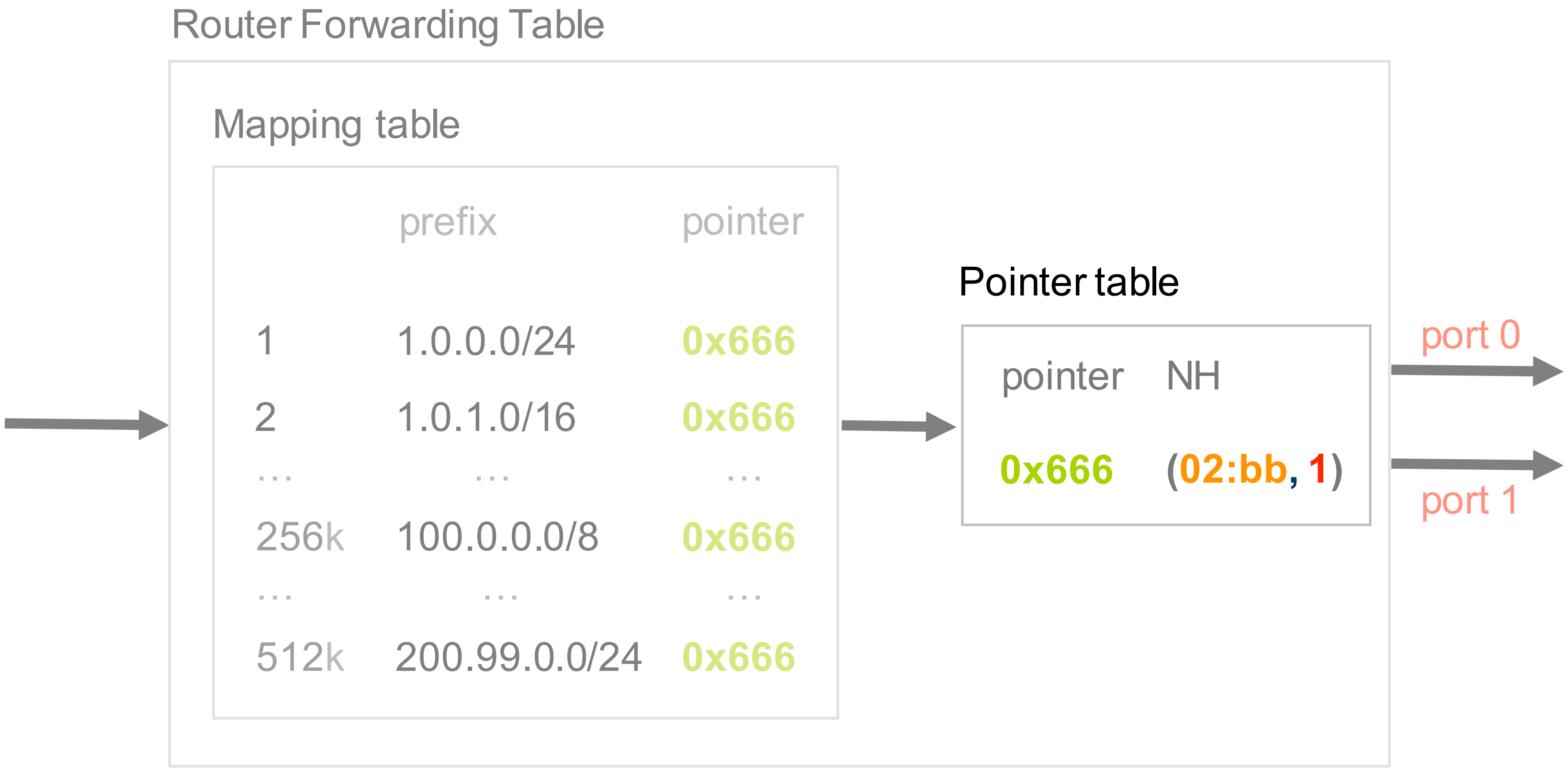
port 0

port 1

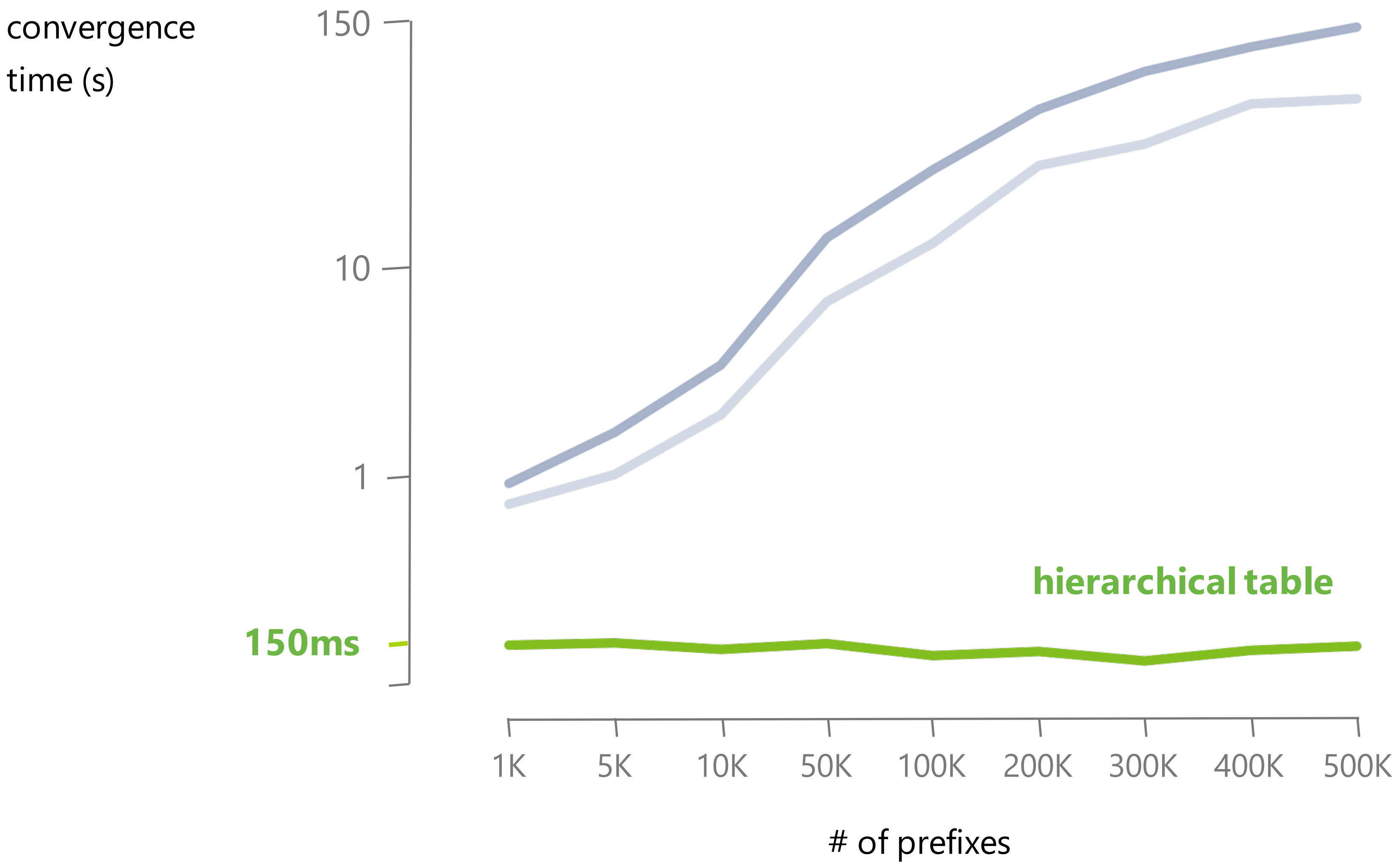
Upon failures, we update the pointer table



Here, we only need to do one update



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

IS-IS

Open Shortest Path First

Intermediate Systems^2

OSPF

IS-IS

Open Shortest Path First

Intermediate Systems²

used in many enterprise & ISPs

work on top of IP

only route IPv4 by default

OSPF

IS-IS

Open Shortest Path First

Intermediate Systems^2

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

until convergence

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

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

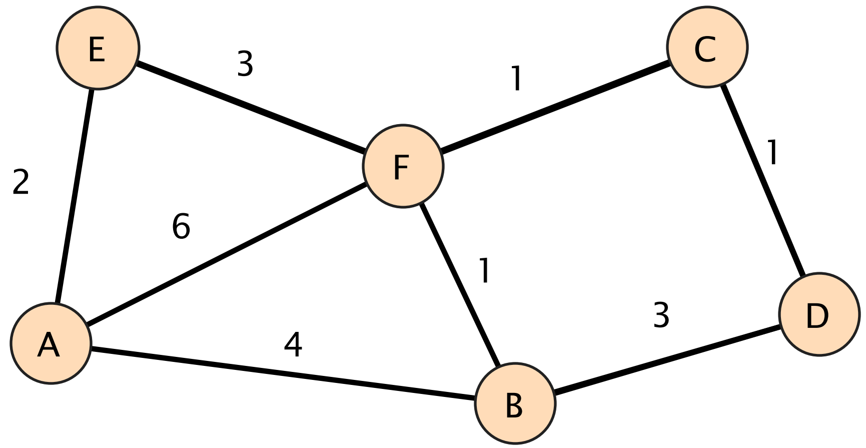
until convergence

Each node updates its distances based on neighbors' vectors:

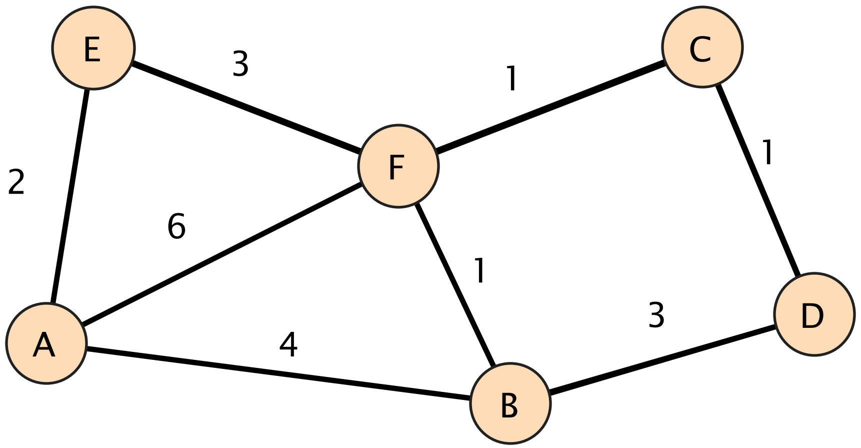
$d_x(y) = \min\{ c(x,v) + d_v(y) \}$ over all neighbors v

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


E
3
C
1
1
2
F
6
1
3
D
A
4
B

Optimum 1-hop path



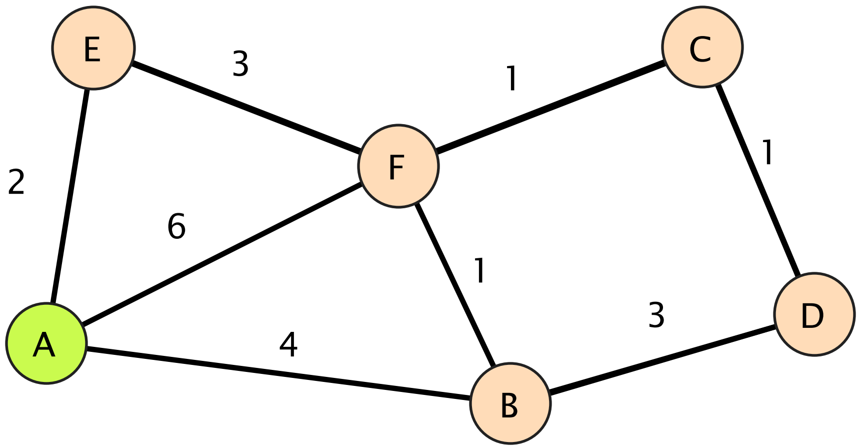
A			B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	∞	–	C	∞	–
D	∞	–	D	3	D
E	2	E	E	∞	–
F	6	F	F	1	F

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		
Dst	Cst	Hop
A	0	A
B	**4**	**B**
C	∞	–
D	∞	–
E	**2**	**E**
F	**6**	**F**

B		
Dst	Cst	Hop
A	4	A
B	0	B
C	∞	–
D	3	D
E	∞	–
F	1	F



C		
Dst	Cst	Hop
A	∞	–
B	∞	–
C	0	C
D	1	D
E	∞	–
F	1	F

D		
Dst	Cst	Hop
A	∞	–
B	3	B
C	1	C
D	0	D
E	∞	–
F	∞	–

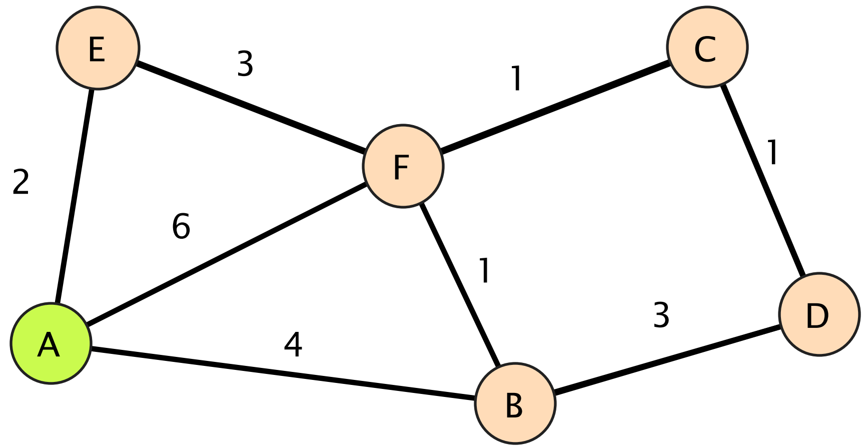
E		
Dst	Cst	Hop
A	2	A
B	∞	–
C	∞	–
D	∞	–
E	0	E
F	3	F

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

Optimum 2-hops path

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

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



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

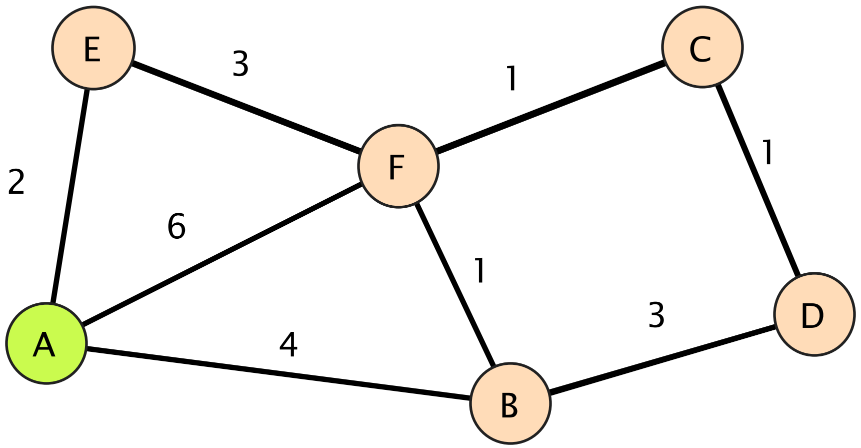
D		
Dst	Cst	Hop
A	**7**	**B**
B	3	B
C	1	C
D	0	D
E	∞	–
F	**2**	**C**

E		
Dst	Cst	Hop
A	2	A
B	**4**	**F**
C	**4**	**F**
D	∞	–
E	0	E
F	3	F

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

Optimum 3-hops path

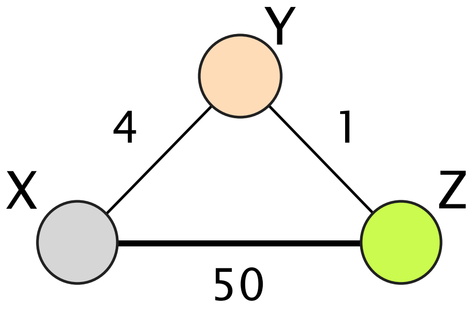
A			B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	6	E	C	2	F
D	7	F	D	3	D
E	2	E	E	4	F
F	5	E	F	1	F



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

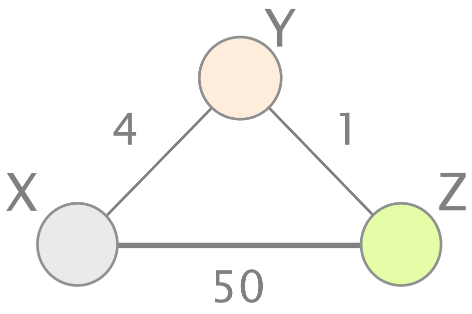
Let’s consider the convergence process after a link cost change

Consider the following network



Consider the following network

leading to the following vectors



Y vector

dest.	via X	via Z
X	4	6

Y reaches X directly

Z vector

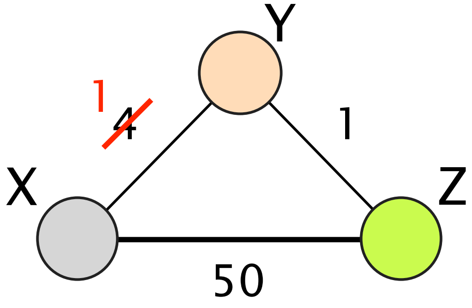
dest.	via X	via Y
X	50	5

Z reaches X via Y

t = 0

(X,Y) weight changes

from 4 to 1



time t=0

Y vector

dest.	via X	via Z
X	4	6

Z vector

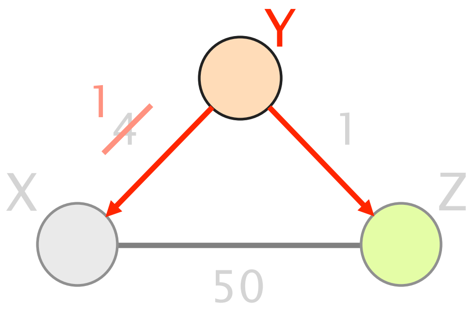
dest.	via X	via Y
X	50	5

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	via Z
X	4	6

t=1

dest.	via X	via Z
X	1	6

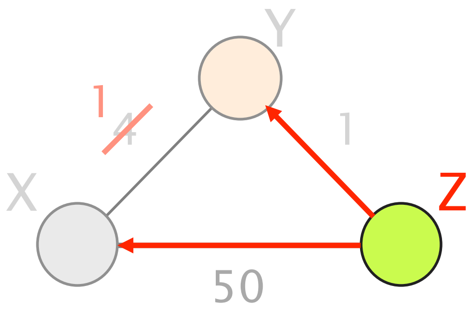
Z vector

dest.	via X	via Y
X	50	5

t = 2

Z updates its vector,

sends it to X and Y

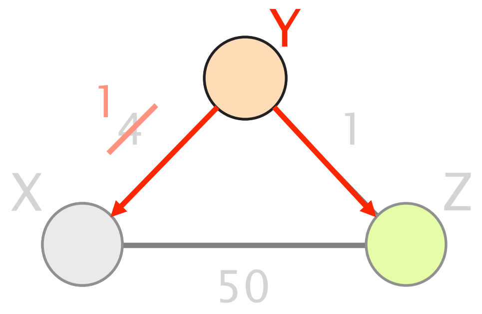


	t=0			t=1			t=2		
	dest.	via		dest.	via		dest.	via	
		X	*Z*		*X*	*Z*			
Y vector	*X*	4	6	*X*	1	6			
	dest.	via					dest.	via	
		X	*Y*					*X*	*Y*
Z vector	*X*	50	5				*X*	50	2

t = 3

Y updates its vector,

sends it to X and Z



Y vector

t=0

dest.	via X	via Z
X	4	6

t=1

dest.	via X	via Z
X	1	6

t=3

dest.	via X	via Z
X	1	3

Z vector

t=0

dest.	via X	via Y
X	50	5

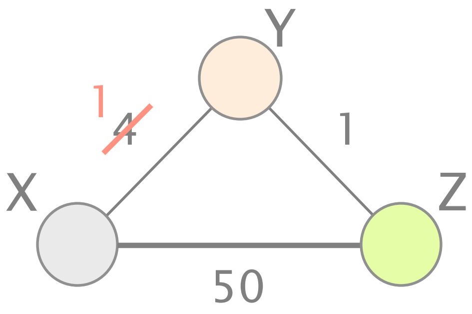
t=2

dest.	via X	via Y
X	50	2

t > 3

no one moves anymore

network has converged!



	t=0			t=1			t=2			t>3		
Y vector	dest.	via X	via Z	dest.	via X	via Z				dest.	via X	via Z
	X	4	6	X	1	6				X	1	3
Z vector	dest.	via X	via Y				dest.	via X	via Y	dest.	via X	via Y
	X	50	5				X	50	2	X	50	2

The algorithm terminates
after 3 iterations

Good news travel fast!

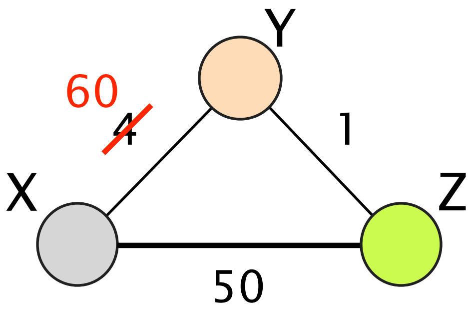
Good news travel fast!

What about bad ones?

t = 0

(X,Y) weight changes

from 4 to 60



time t=0

Y vector

dest.	via X	via Z
X	4	6

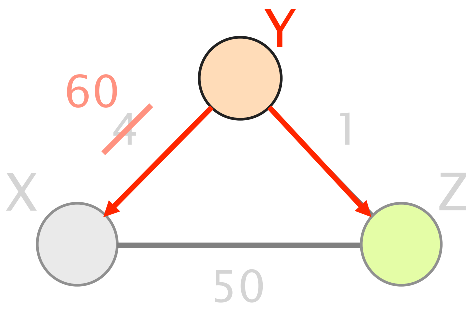
Z vector

dest.	via X	via Y
X	50	5

t = 1

Y updates its vector,

sends it to X and Z



t=0

Y vector

dest.	via X	via Z
X	4	6

Z vector

dest.	via X	via Y
X	50	5

t=1

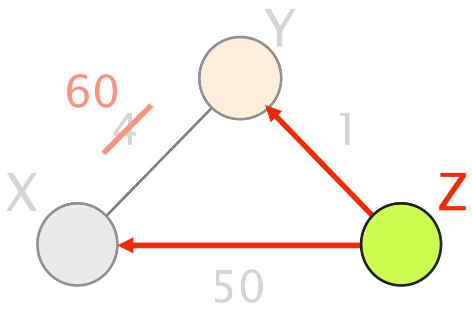
Y vector

dest.	via X	via Z
X	60	6

t = 2

Z updates its vector,

sends it to X and Y

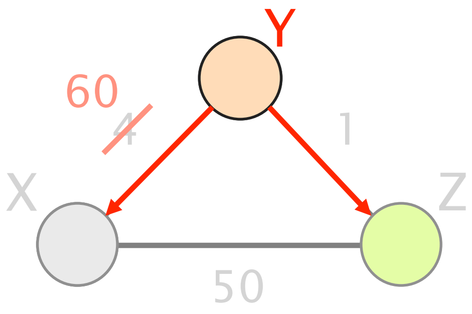


	t=0			t=1			t=2		
Y vector	dest.	via X	via Z	dest.	via X	via Z			
	X	4	6	X	60	6			
Z vector	dest.	via X	via Y				dest.	via X	via Y
	X	50	5				X	50	7

t = 3

Y updates its vector,

sends it to X and Z



t=0

Y vector

dest.	via X	via Z
X	4	6

Z vector

dest.	via X	via Y
X	50	5

t=1

Y vector

dest.	via X	via Z
X	60	6

t=2

Z vector

dest.	via X	via Y
X	50	7

t=3

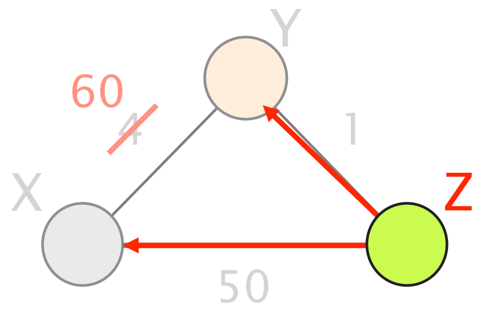
Y vector

dest.	via X	via Z
X	60	8

t = 4

Z updates its vector,

sends it to X and Y…

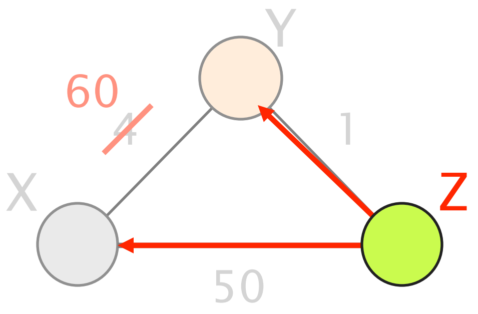


t=4

Y
vector

Z
vector

dest.	via X	via Y
X	50	9



t=4

Y vector

Z vector

dest.	via X	via Y
X	50	9

… many iterations later …

t=44

Y vector

dest.	via X	via Z
X	60	51

Z vector

dest.	via X	via 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

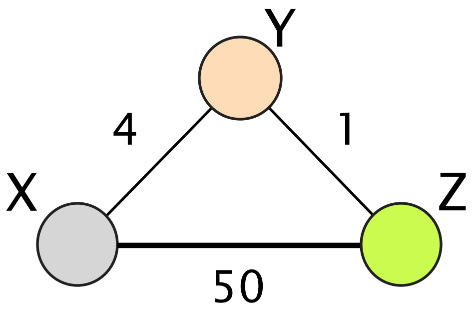
Z does not know that Y has switched to use it

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	via Z
X	4	∞

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

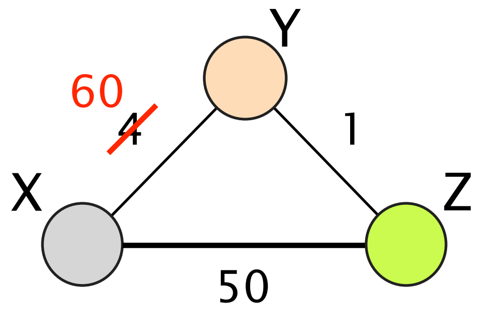
Z
vector

dest.	via X	via Y
X	50	5

t = 0

(X,Y) weight changes

from 4 to 60



time t=0

Y vector

dest.	via X	via Z
X	4	∞

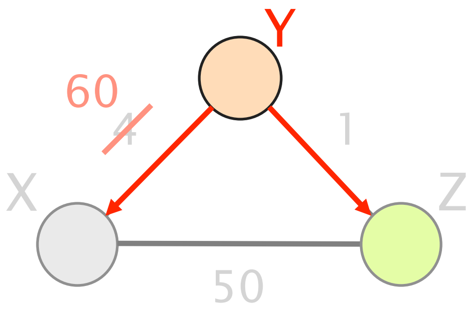
Z vector

dest.	via X	via Y
X	50	5

t = 1

Y updates its vector,

sends it to X and Z



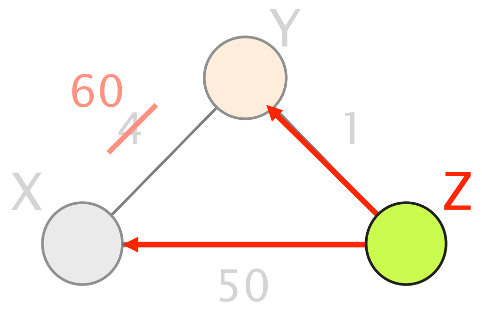
	t=0			t=1		
Y vector	dest.	via X	via Z	dest.	via X	via Z
	X	4	∞	X	60	∞

	t=0		
Z vector	dest.	via X	via Y
	X	50	5

t = 2

Z updates its vector,

sends it to X and Y



t=0

Y vector

dest.	via X	via Z
X	4	∞

Z vector

dest.	via X	via Y
X	50	5

t=1

Y vector

dest.	via X	via Z
X	60	∞

t=2

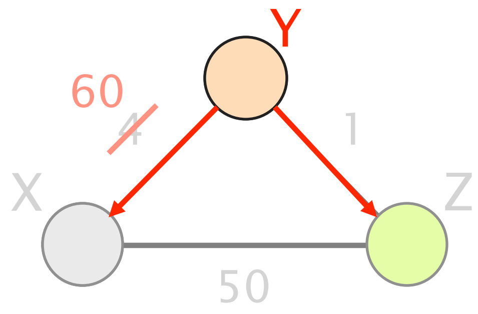
Z vector

dest.	via X	via Y
X	50	61

t = 3

Y updates its vector,

sends it to X and Z



Y vector

t=0

dest.	via X	via Z
X	4	∞

t=1

dest.	via X	via Z
X	60	∞

t=3

dest.	via X	via Z
X	60	51

Z vector

t=0

dest.	via X	via Y
X	50	5

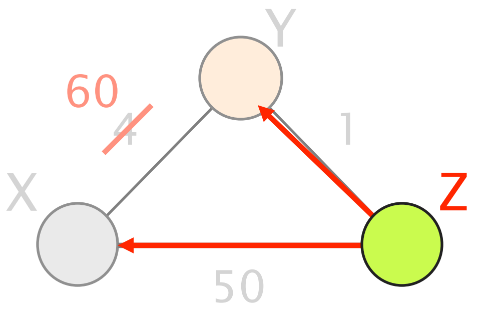
t=2

dest.	via X	via Y
X	50	61

t = 4

Z updates its vector,

sends it to X and Y



t=4

Y
vector

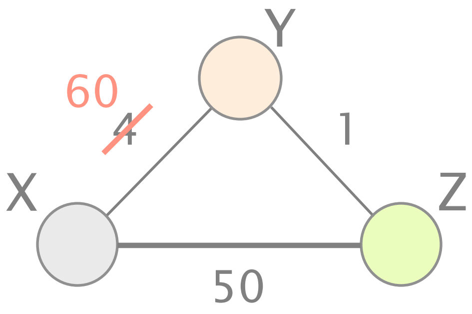
Z
vector

dest.	via X	via Y
X	50	∞

t > 4

no one moves

network has converged!



t=4

t>4

Y vector (t>4)

dest.	via X	via Z
X	60	51

Z vector (t=4)

dest.	via X	via Y
X	50	∞

Z vector (t>4)

dest.	via X	via 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

Communication Networks and Internet Technology

Short Recap on this weeks lecture

Internet routing

from here to there, and back

1 **Intra-domain routing**

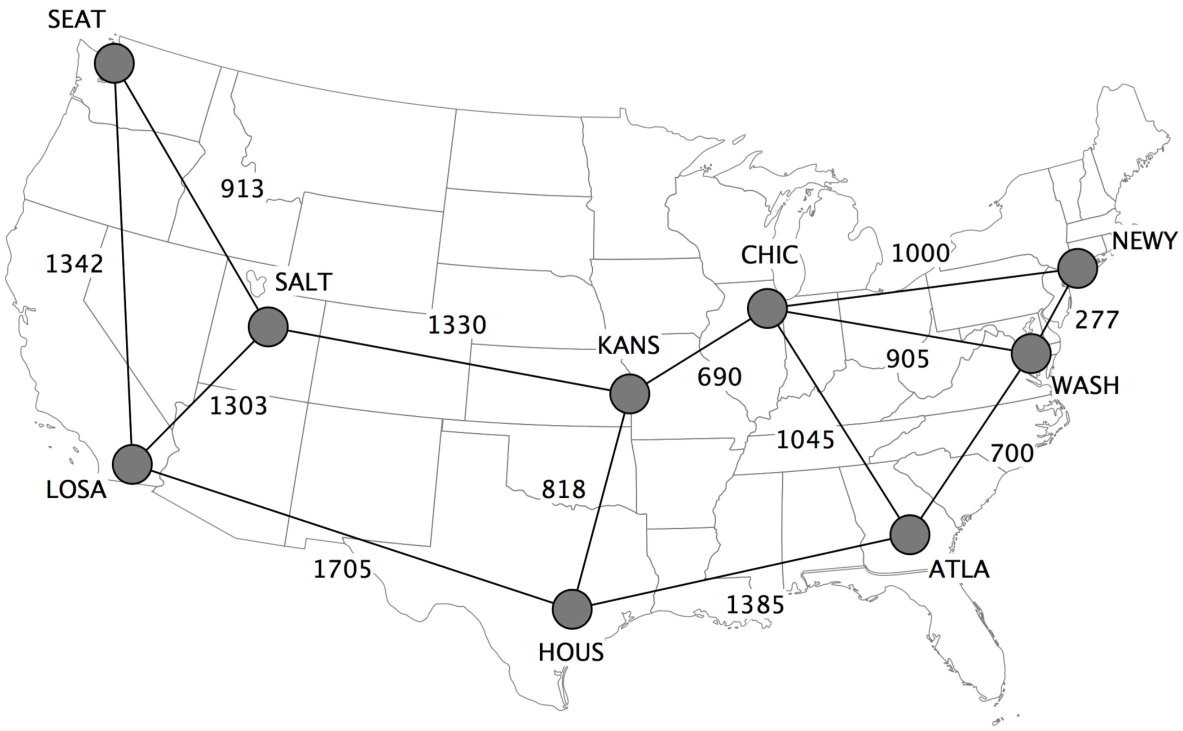
Link-state protocols

Distance-vector protocols

2 **Inter-domain routing**

Path-vector protocols

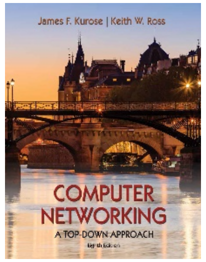
When weights are assigned proportionally to the distance, shortest-paths will minimize the end-to-end delay



Internet2, the US based research network

Reading: Book Kurose & Ross

Class textbook:
Computer Networking: A Top-Down Approach (8^{th} *ed.*)
J.F. Kurose, K.W. Ross
Pearson, 2020
http://gaia.cs.umass.edu/kurose_ross



- Week 05
  - 4.1 (Introduction to the Network Layer), 4.3 (What's Inside a Router), 4.5 (The Internet Protocol)

- Week 06 + 07
  - 4.6 (Routing the Internet)

Check Your Knowledge

PROBLEM SOLVING HOME | TRY A RANDOM PROBLEM

INTERACTIVE END-OF-CHAPTER EXERCISES

CHAPTER 4: NETWORK LAYER: DATA PLANE

- Network Address Translation
- Longest Prefix Matching (similar to Chapter 4, P9, P10)
- Subnet Addressing
- IPv6 Tunneling and Encapsulation
- Packet Scheduling

CHAPTER 5: NETWORK LAYER: CONTROL PLANE

- Dijkstra's Link State Algorithm (similar to Chapter 5, P3)
- Dijkstra's Link State Algorithm - Advanced
- Bellman Ford Distance Vector algorithm (similar to Chapter 5, P5)
- Openflow Flow Tables

n can then be displayed (hopefully
e text. Most importantly, you can
l.

rk labs) for our book, available

ding new problems here in the

http://gaia.cs.umass.edu/kurose_ross/interactive/