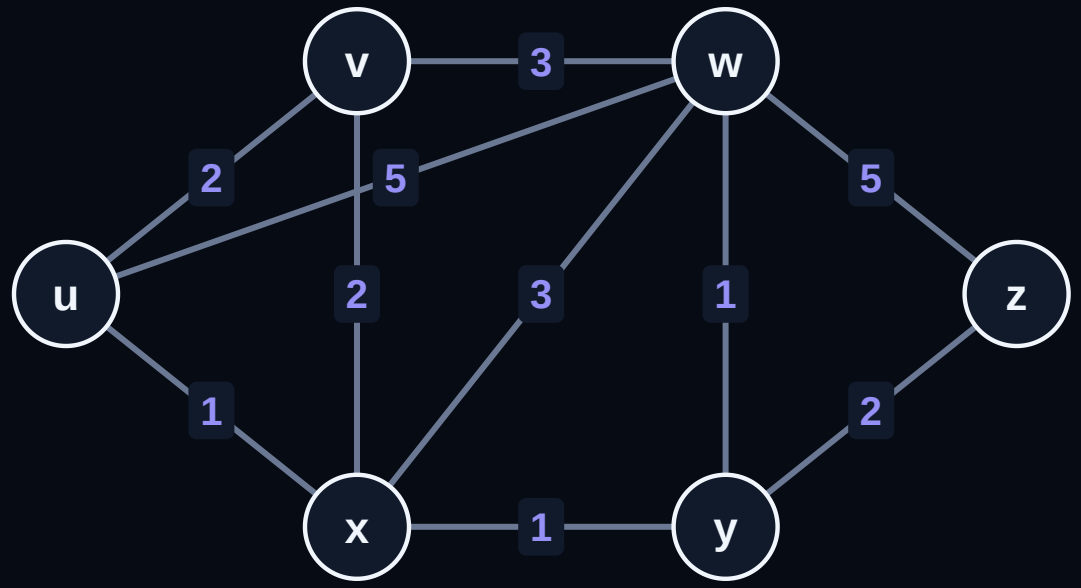
Distance-vector routing The table, built from what the neighbours say

Link state gave every router the whole map. Here a router hears only its neighbours, and still finds the least-cost path to every destination. We build the equation, run the algorithm, watch it fail when a link gets worse, and compare the two ways of filling the table.

Yaxiong Xie

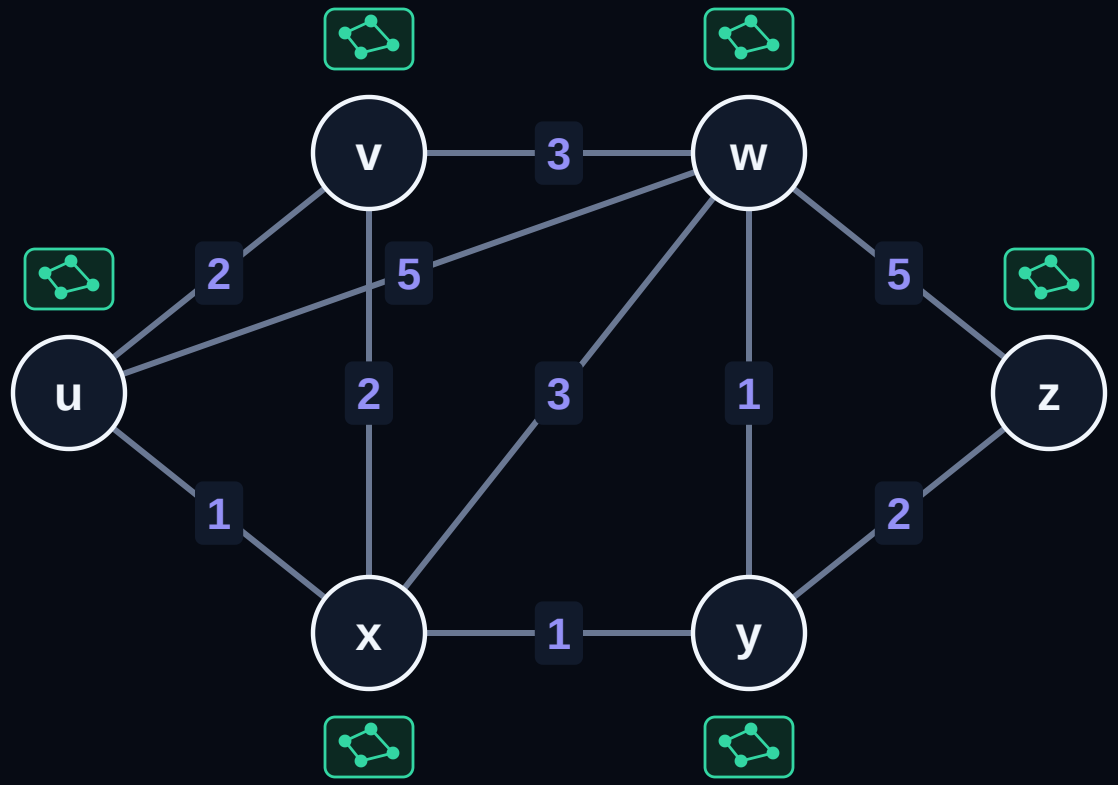


each router talks only to its neighbours

and keeps one number per destination

what goes wrong when a link gets worse

Link state gave every router the whole map



every router floods its links
to every other router

every router holds the same map
of the whole network

one link changes: a flood to all,
and every router reruns Dijkstra

Every router heard every link. Now: the same table, from the neighbours only.

Six topics, each from its own problem

01



The equation

one number from each neighbour

02



The algorithm

trade vectors until nothing changes

03



A cost changes

good news fast, bad news loops

04



Counting

how long the loop lasts

05



Poisoned reverse

one lie that breaks the loop

06



LS vs DV

what each way pays

Each one: why it is needed, how it works, what it costs.

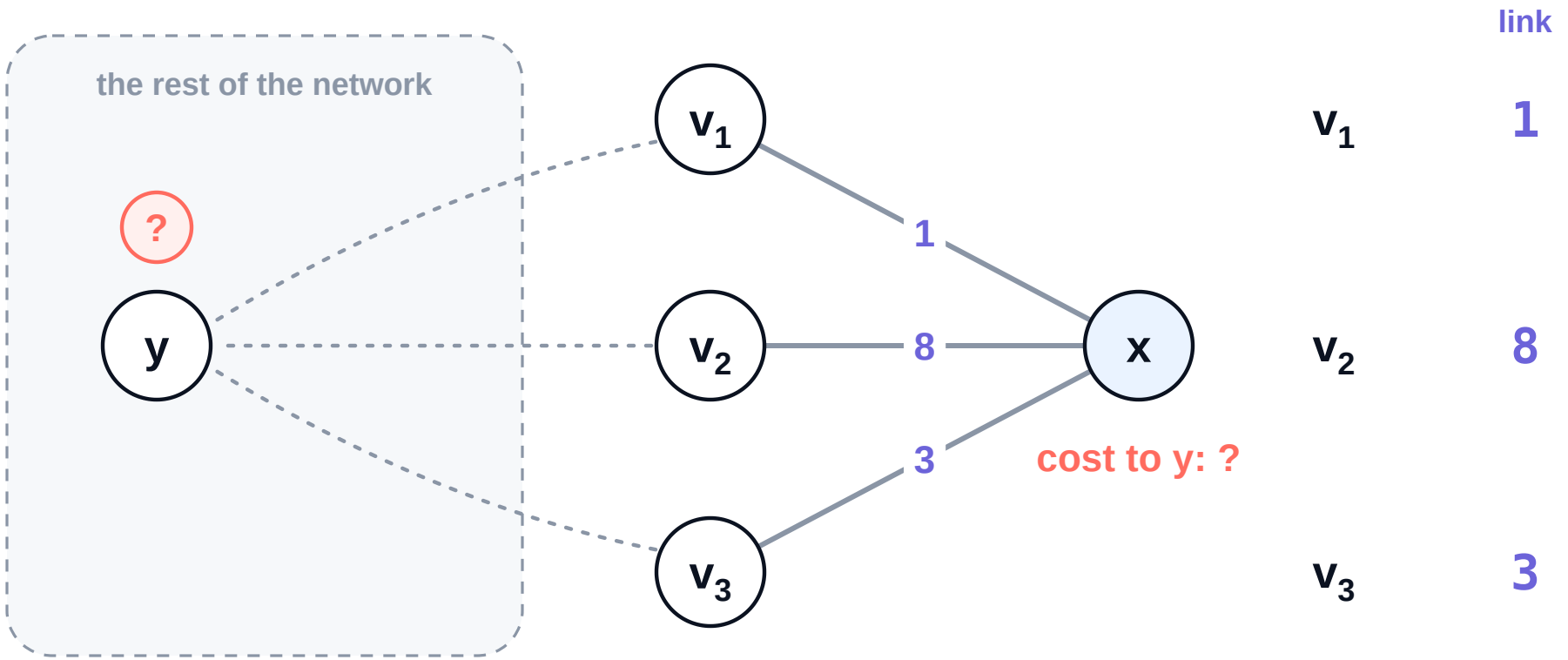
x sees only its own links. How does it reach y?

01

The Bellman-Ford equation

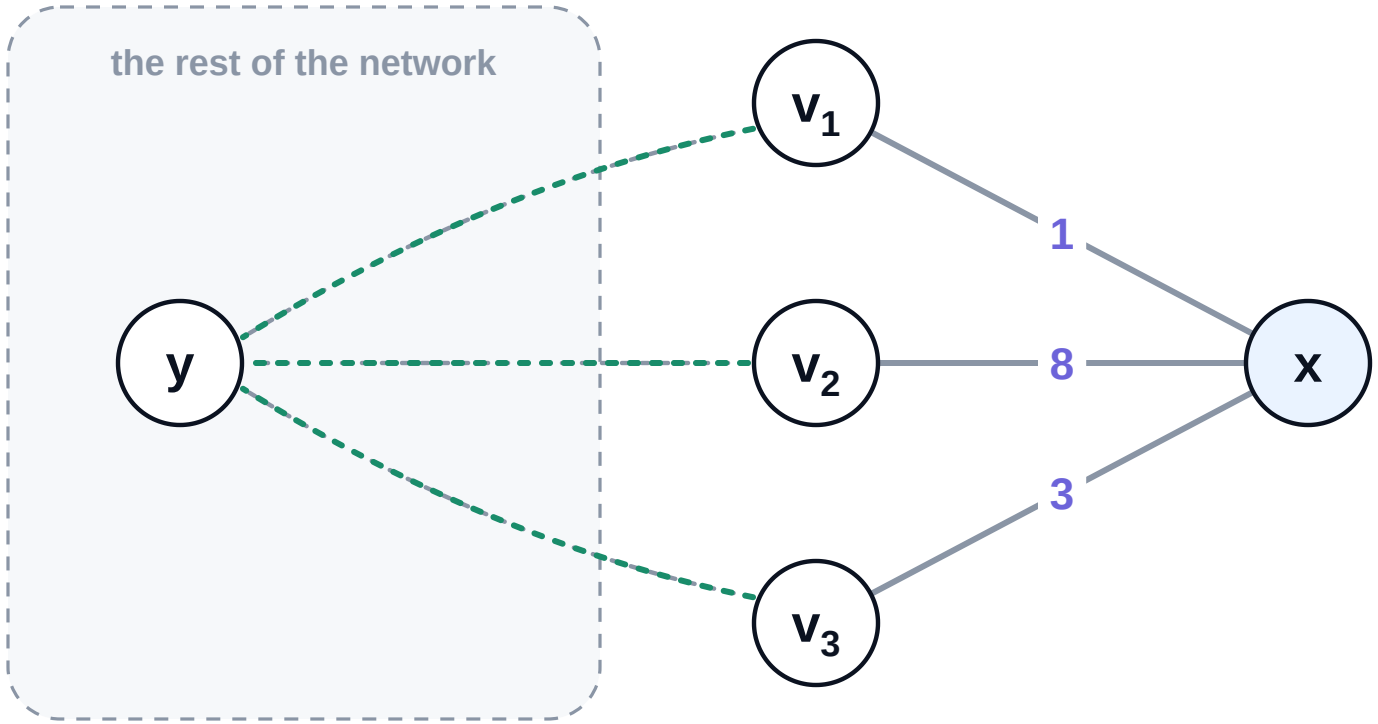
x knows what each of its links costs. Everything past its neighbours is out of sight, and x still needs a next hop for y.

What does x know on its own?



The cost of each of its own links, and nothing beyond its neighbours.

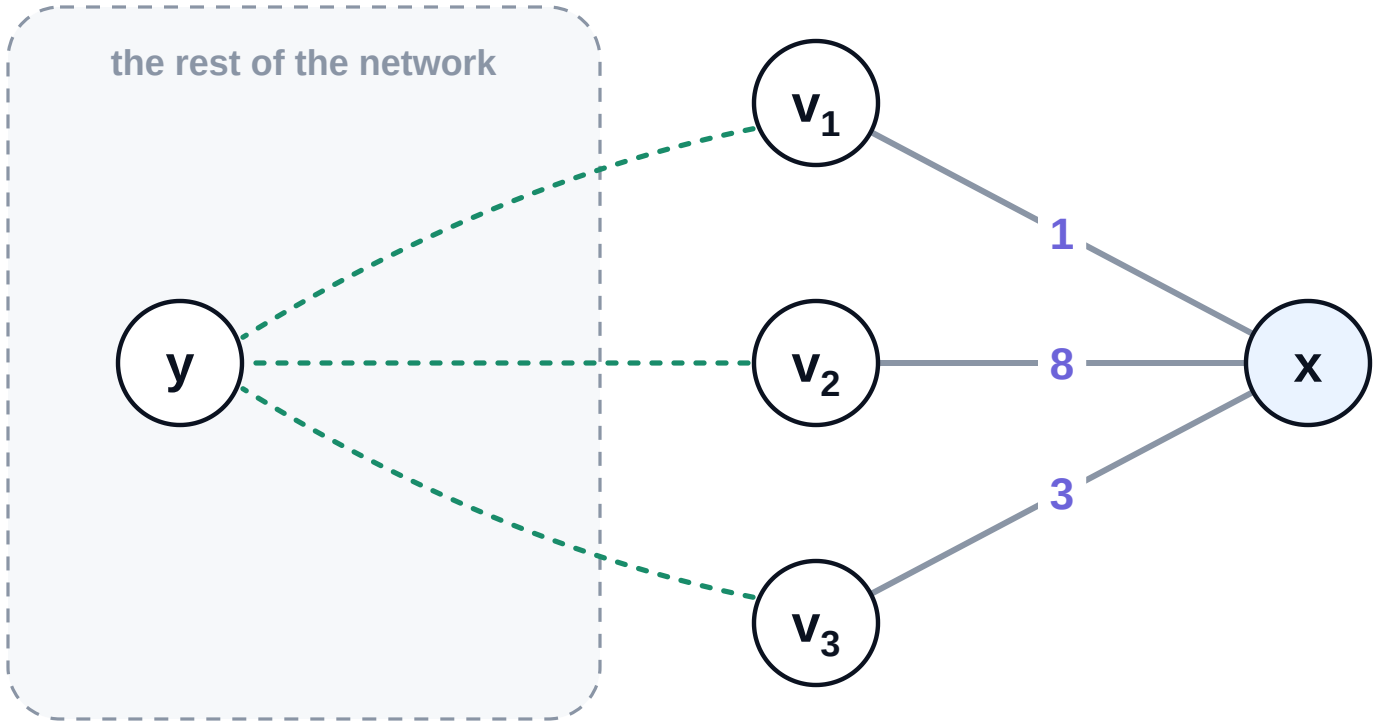
What can a neighbour tell x?



	link	report
v_1	1	7
v_2	8	2
v_3	3	4

Its own least cost to y, measured from the neighbour, not from x.

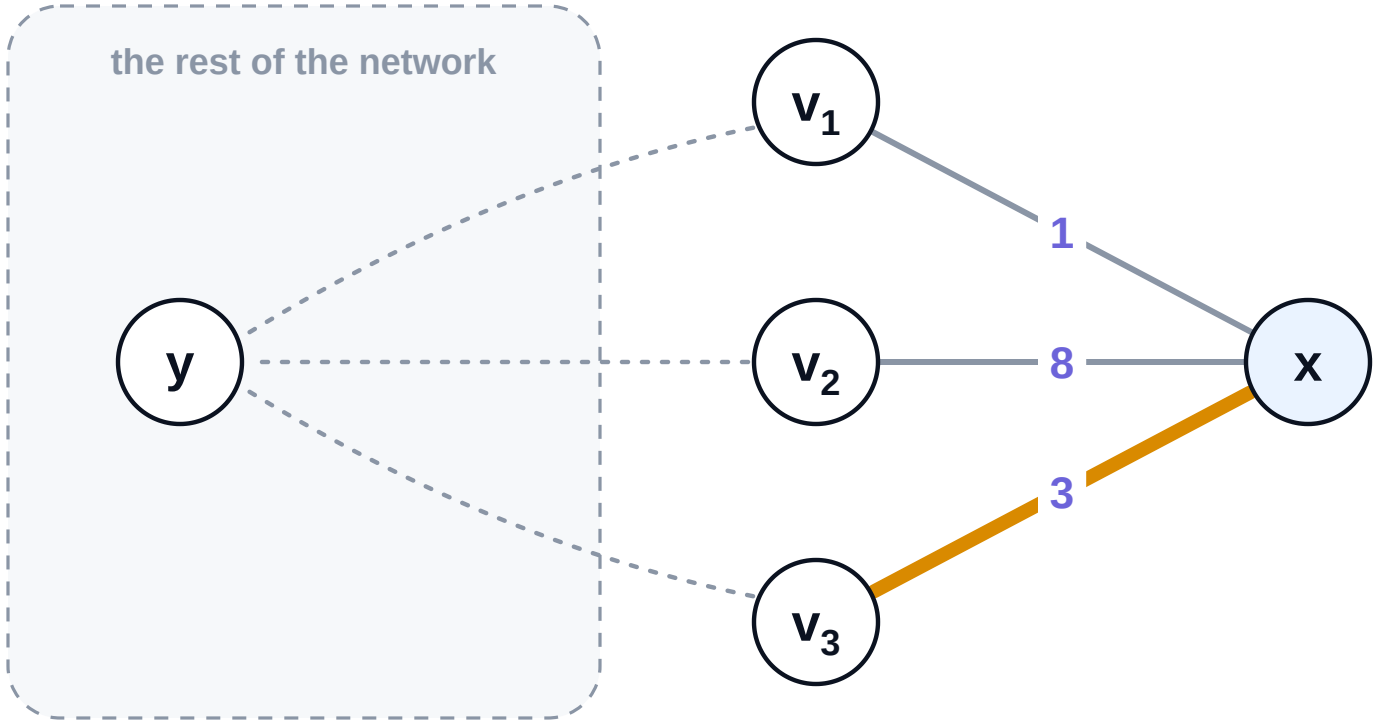
What does each report cost x?



	link		report		total
v_1	1	+	7	=	8
v_2	8	+	2	=	10
v_3	3	+	4	=	7

One candidate per neighbour: the link to it plus what it reported.

Which candidate wins?

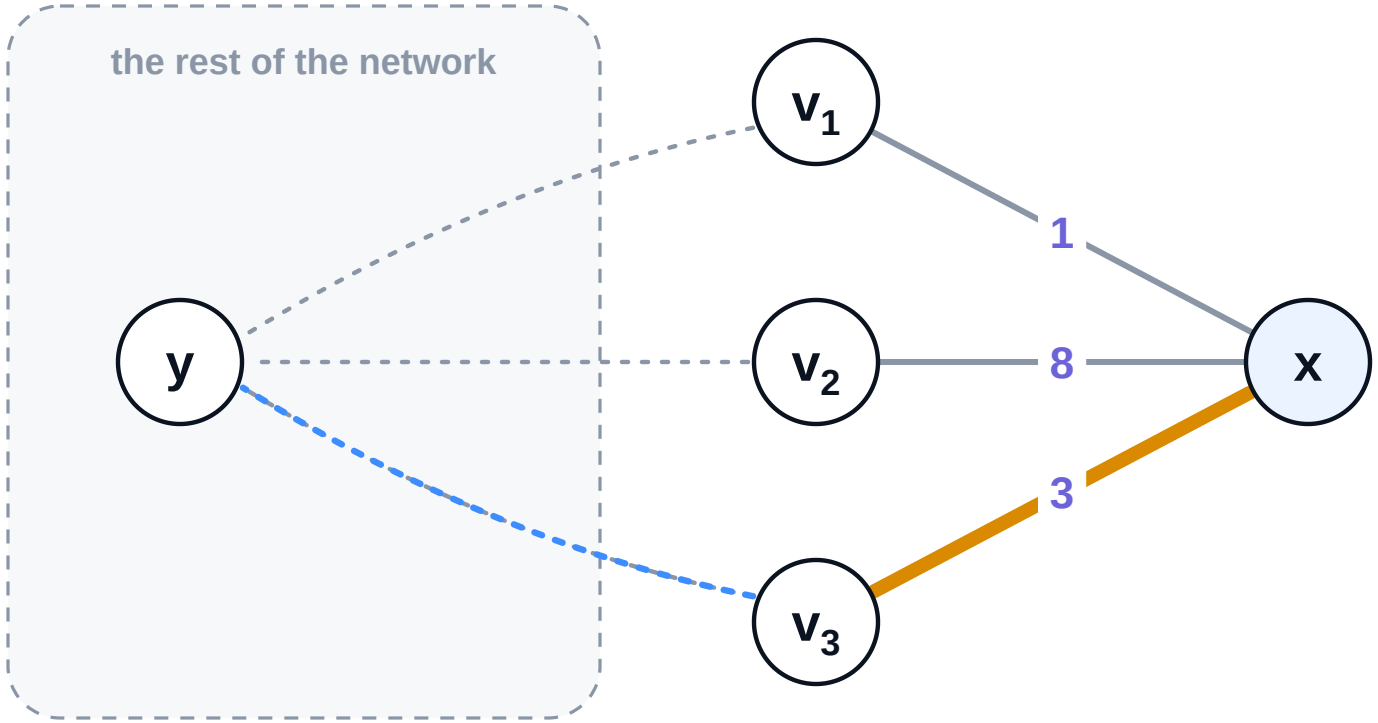


	link		report		total
v_1	1	+	7	=	~~8~~
v_2	8	+	2	=	~~10~~
v_3	3	+	4	=	**7**

the smallest total: 7, through v_3

7 through v_3: neither the cheapest link nor the smallest report.

The winning neighbour is the next hop



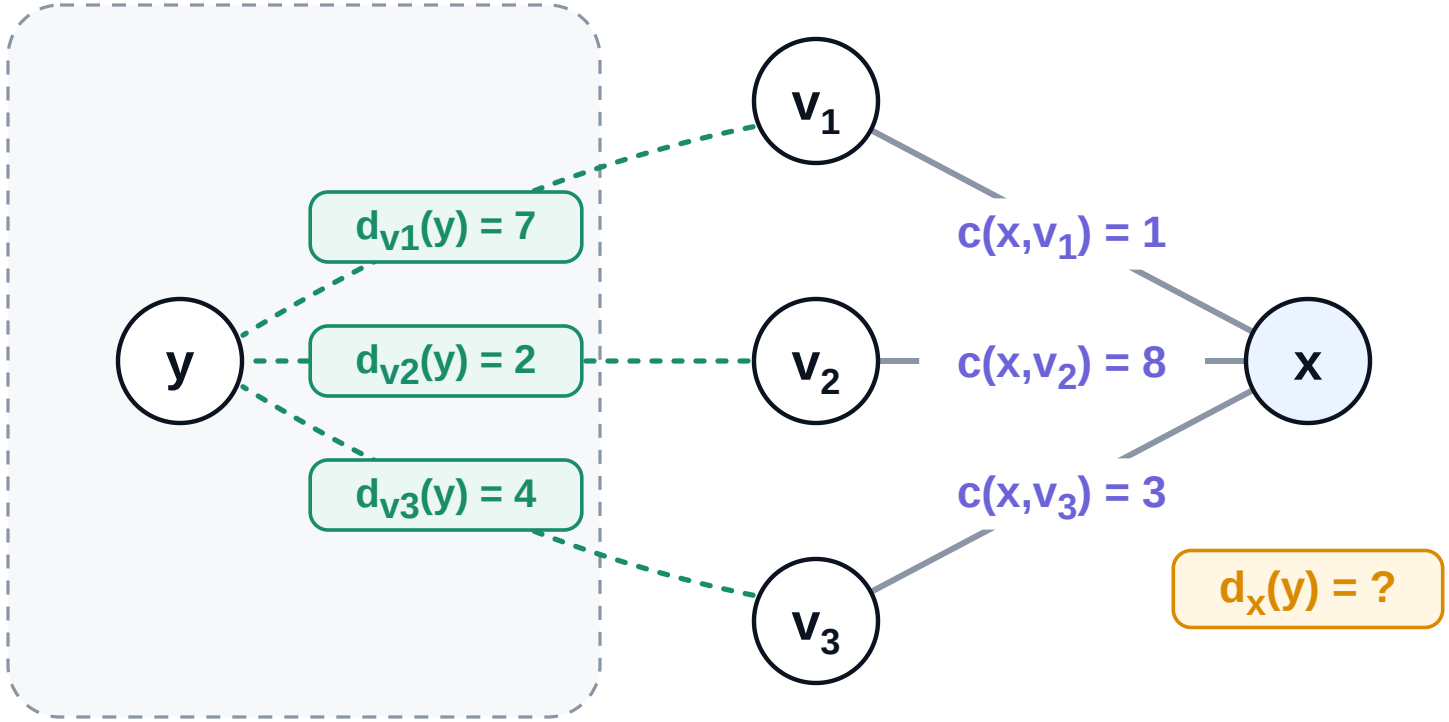
x's forwarding table

destination	next hop	cost
y	v_3	7

packets for y leave on the link to v_3

The minimum is x's cost. The neighbour that gave it is x's next hop.

Each number on the picture gets a name



$c(x,v)$

the cost of the link from x to v

v is any neighbour of x: v_1, v_2 or v_3

$d_v(y)$

v's least cost to y

the number v reports to x

$d_x(y)$

x's least cost to y

the number x wants to find

c: x's own links. $d_v(y)$: the reports. $d_x(y)$: the answer x wants.

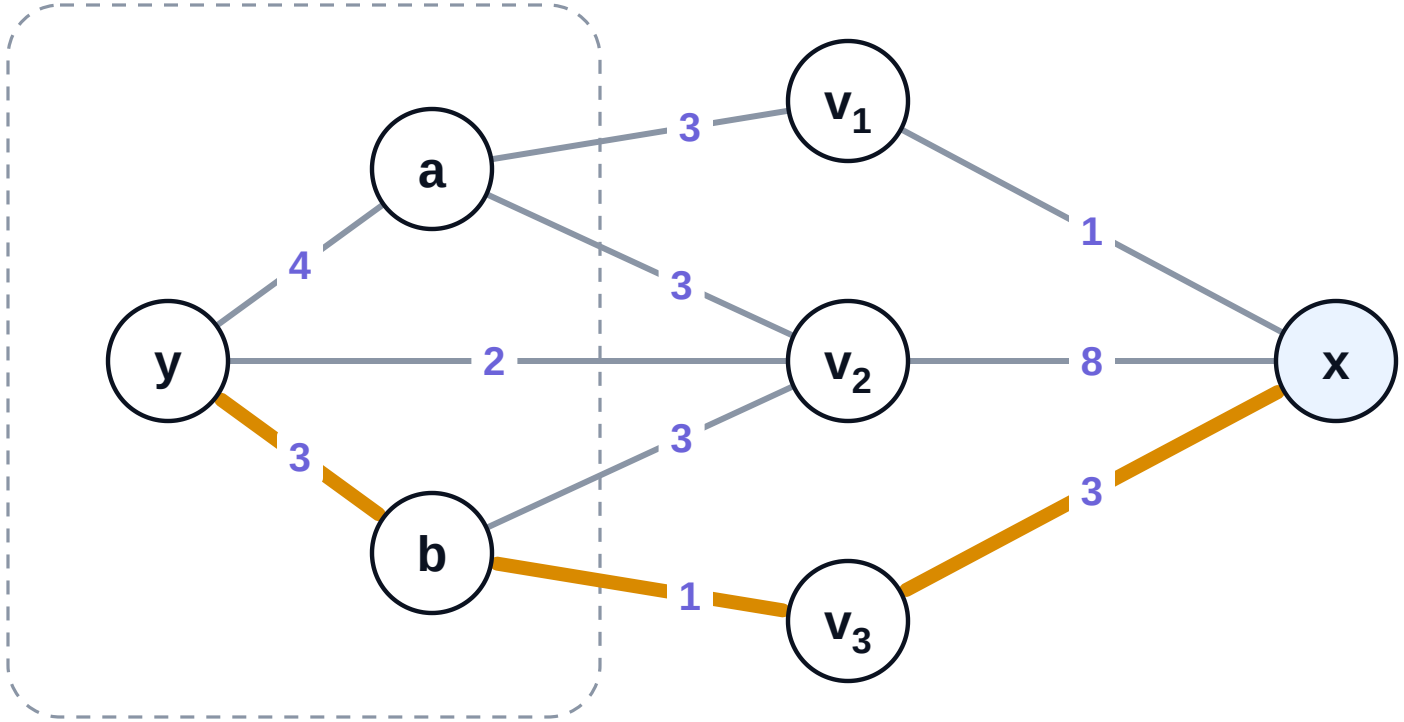
Put together: the Bellman-Ford equation

$$d_x(y) = \min_v \{ c(x,v) + d_v(y) \}$$

$c(x,v) + d_v(y)$	reach y through neighbour v: the link, then v's report 1 + 7 8 + 2 3 + 4 on the picture
$\min_v$	try every neighbour v, keep the smallest min{ 8, 10, 7 }
$d_x(y)$	that smallest is x's least cost to y $d_x(y) = 7$ next hop v_3

Every neighbour offers link + report; the smallest offer is x's cost to y.

Why is the minimum the right answer?



every path from x to y, by its first link

v_1 8 9 13

best 8 = 1 + 7

v_2 10 14 15

best 10 = 8 + 2

v_3 7 9 14

best 7 = 3 + 4

best of the three bests: 7, through v_3

Group the paths by their first link: each group's best is link plus report.

Why is asking the neighbours enough?

on its own: list every path

x v_1 a y	8
x v_1 a v_2 y	9
x v_1 a v_2 b y	13
x v_2 y	10
x v_2 b y	14
x v_2 a y	15
x v_3 b y	7
x v_3 b v_2 y	9
x v_3 b v_2 a y	14

a real network: far too many

with its neighbours: three numbers

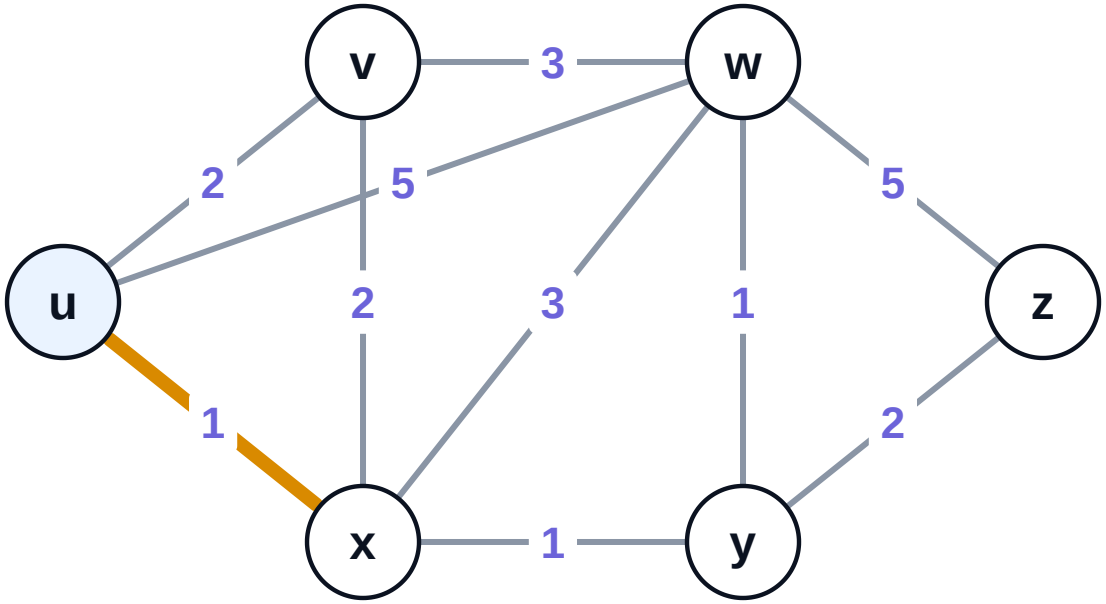
v_1 to y: 7

v_2 to y: 2

v_3 to y: 4

Each report already stands for every path behind that neighbour.

On the link-state network, the same answer



the link-state lecture's network

u: cost to z, from three reports

via v: 2 + 5 = 7

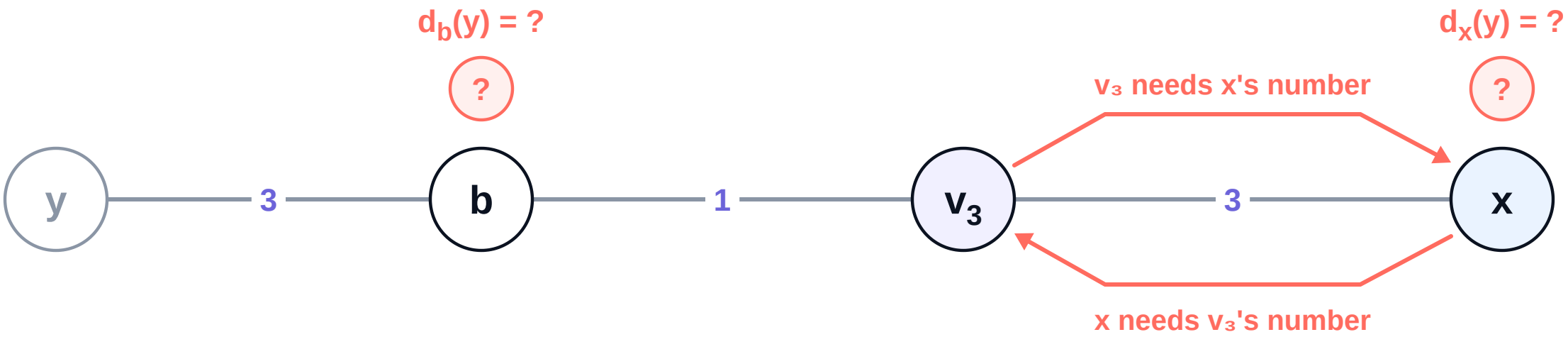
via x: 1 + 3 = 4

via w: 5 + 3 = 8

✓ Dijkstra found the same:
D(z) = 4, first link (u, x)

The cost and next hop Dijkstra found, from three numbers and no map.

Where does v_3's 4 come from?



$$d_{v3}(y) = \min\{ 1 + d_b(y) , 3 + d_x(y) \}$$

v_3 runs the same equation, and needs its neighbours' numbers first, x's too.

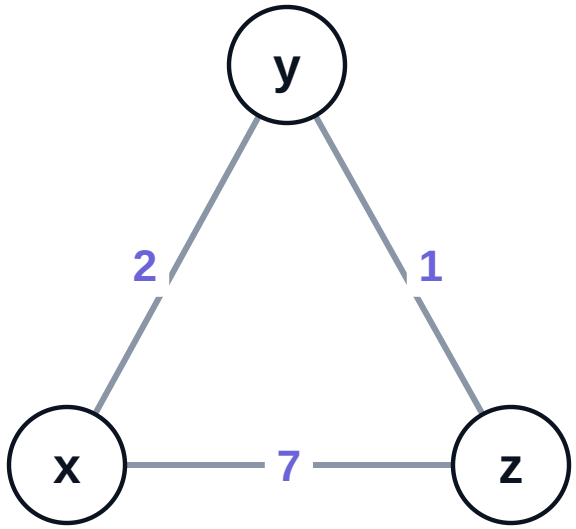
Every router needs its neighbours' answers first

02

The distance-vector algorithm

x waits for v_3, v_3 waits for its neighbours, and x is one of them. Nobody starts with an answer.

What can each router write down at the start?



x's table

	to x	to y	to z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

y's table

	to x	to y	to z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

z's table

	to x	to y	to z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

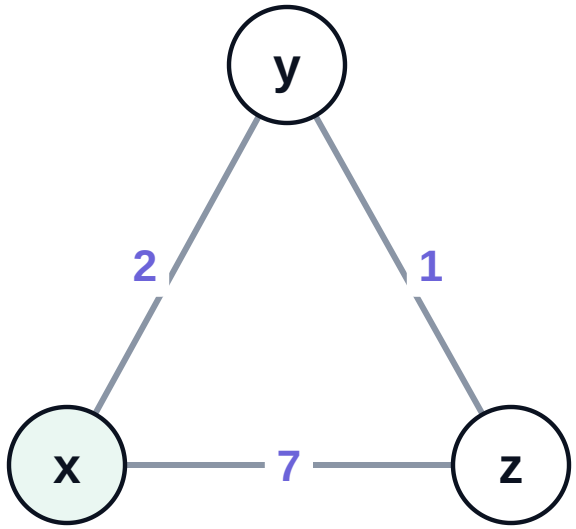
a first guess

Where a router has no link, its first guess is infinity (∞).

Its own link costs, as first guesses. x's 7 to z is a guess, not an answer.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

What does a router send?



x's table

	to x	to y	to z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

y's table

	to x	to y	to z
x	0	2	7
y	2	0	1
z	∞	∞	∞

z's table

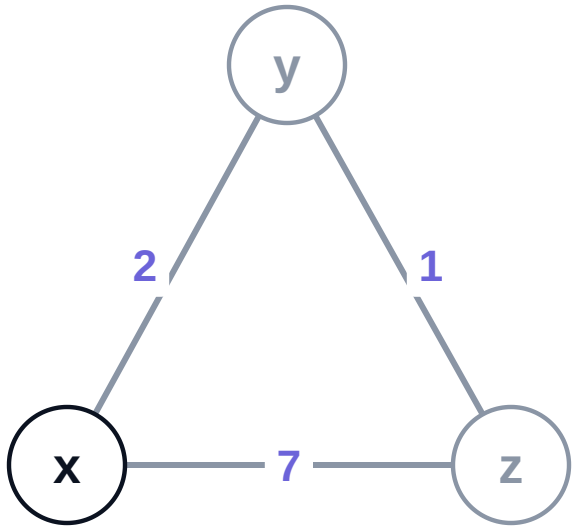
	to x	to y	to z
x	0	2	7
y	∞	∞	∞
z	7	1	0

its row: $D_x = [0, 2, 7]$

Here every router is a neighbour of the others. In general, only the routers at the other end of its links.

Its own row, to its neighbours only. That row is its distance vector.

What does x keep?



x's table

	to x	to y	to z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

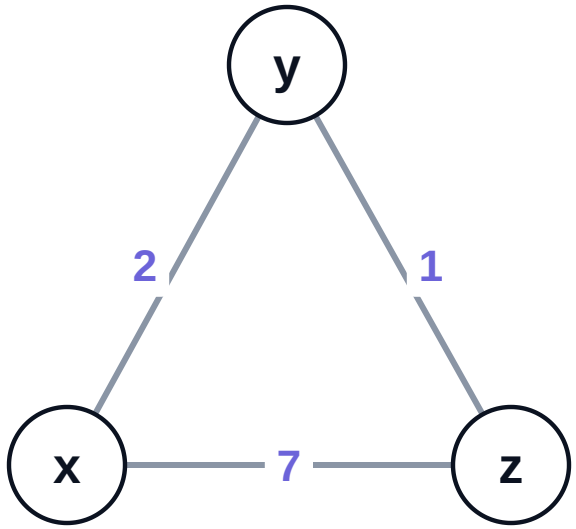
its own vector
the row it sends to its neighbours

the last vector from each neighbour
∞ for now; each new vector replaces the last

nothing else: no map, no paths

Its own vector and the latest vector from each neighbour. That is all its state.

Every router sends its first vector



x's table

	to x	to y	to z
x	0	2	7
y	2	0	1
z	7	1	0

y's table

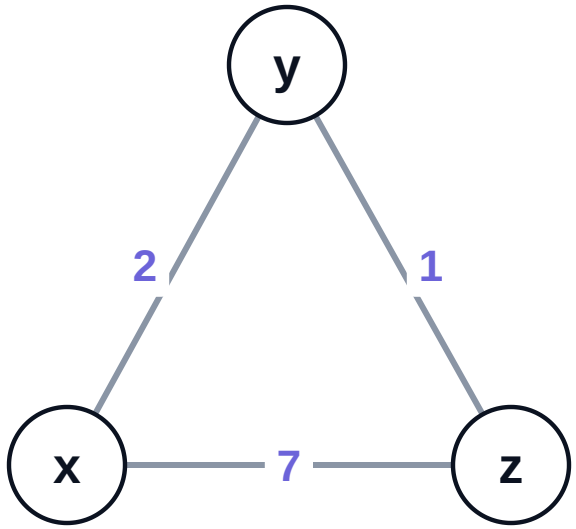
	to x	to y	to z
x	0	2	7
y	2	0	1
z	7	1	0

z's table

	to x	to y	to z
x	0	2	7
y	2	0	1
z	7	1	0

All three send; each receives one vector per neighbour.

x recomputes its cost to y



x's table

	to x	to y	to z
x	0	2	7
y	2	0	1
z	7	1	0

y's table

	to x	to y	to z
x	0	2	7
y	2	0	1
z	7	1	0

z's table

	to x	to y	to z
x	0	2	7
y	2	0	1
z	7	1	0

unchanged

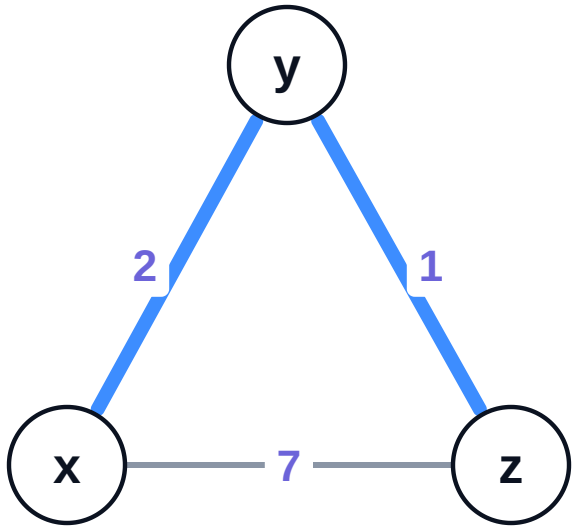
Bellman-Ford: $d_x(y) = \min\{ c(x,y) + d_y(y) , c(x,z) + d_z(y) \}$

$$= \min\{ 2 + 0 , 7 + 1 \} = 2$$

Still 2. The direct link is still the best way to y.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

x recomputes its cost to z



x's table

	to x	to y	to z
x	0	2	3
y	2	0	1
z	7	1	0

y's table

	to x	to y	to z
x	0	2	7
y	2	0	1
z	7	1	0

z's table

	to x	to y	to z
x	0	2	7
y	2	0	1
z	7	1	0

cheaper, through y

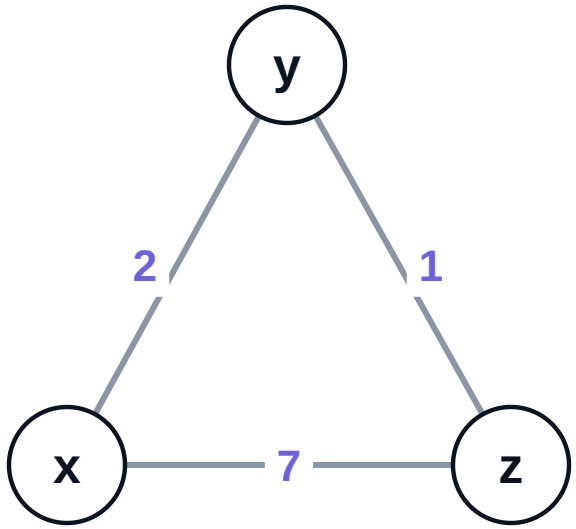
Bellman-Ford: $d_x(z) = \min\{ c(x,y) + d_y(z) , c(x,z) + d_z(z) \}$

$$= \min\{ 2 + 1 , 7 + 0 \} = 3$$

3 through y, cheaper than x's own link to z.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

y and z ran the same step at the same time



x's table

	to x	to y	to z
x	0	2	3
y	2	0	1
z	3	1	0

y's table

	to x	to y	to z
x	0	2	3
y	2	0	1
z	3	1	0

nothing cheaper

z's table

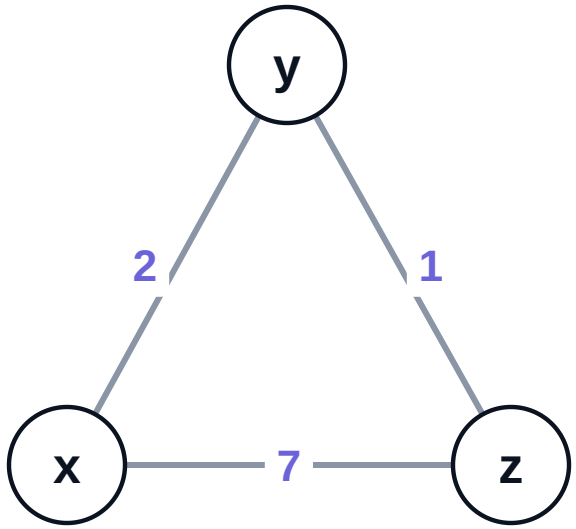
	to x	to y	to z
x	0	2	3
y	2	0	1
z	3	1	0

`min{ 7 + 0 , 1 + 2 } = 3`

z found the same shortcut; y found nothing new. x and z send again.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

What happens in the second exchange?



x's table

	to x	to y	to z
x	0	2	3
y	2	0	1
z	3	1	0

unchanged

y's table

	to x	to y	to z
x	0	2	3
y	2	0	1
z	3	1	0

unchanged

z's table

	to x	to y	to z
x	0	2	3
y	2	0	1
z	3	1	0

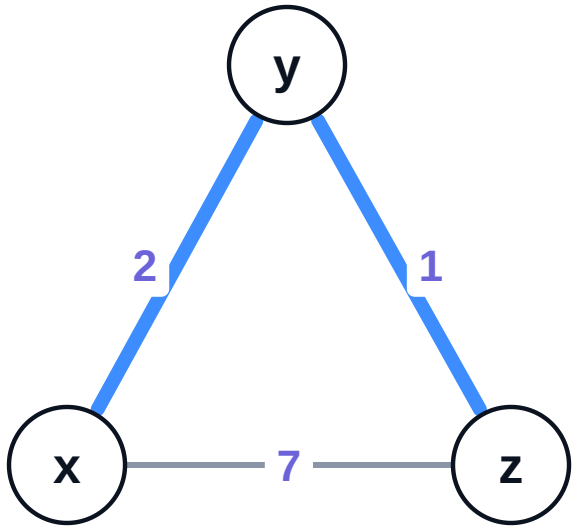
unchanged



nothing to send

No vector changes, so nothing is sent. The algorithm stops by itself.

What do the tables hold now?



the link x–z (7) is never used

x's table

	to x	to y	to z
x	0	2	3
y	2	0	1
z	3	1	0
via	–	y	y

y's table

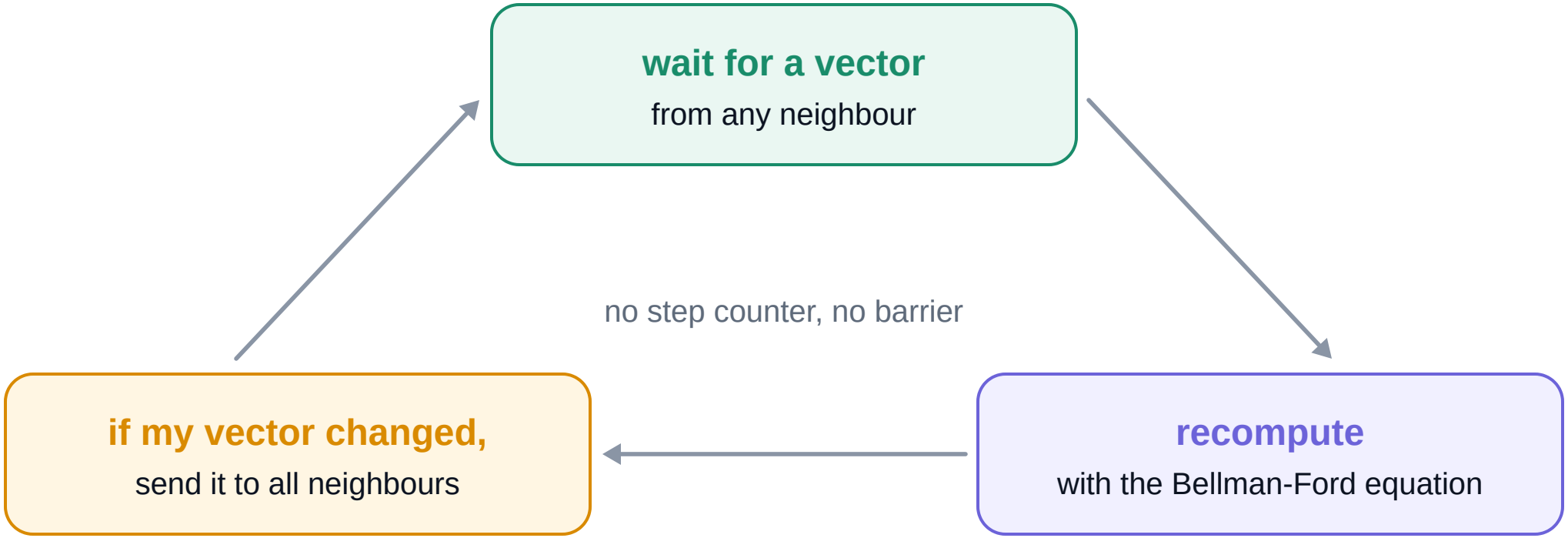
	to x	to y	to z
x	0	2	3
y	2	0	1
z	3	1	0
via	x	–	z

z's table

	to x	to y	to z
x	0	2	3
y	2	0	1
z	3	1	0
via	y	y	–

The true least costs and next hops, and no router ever saw the graph.

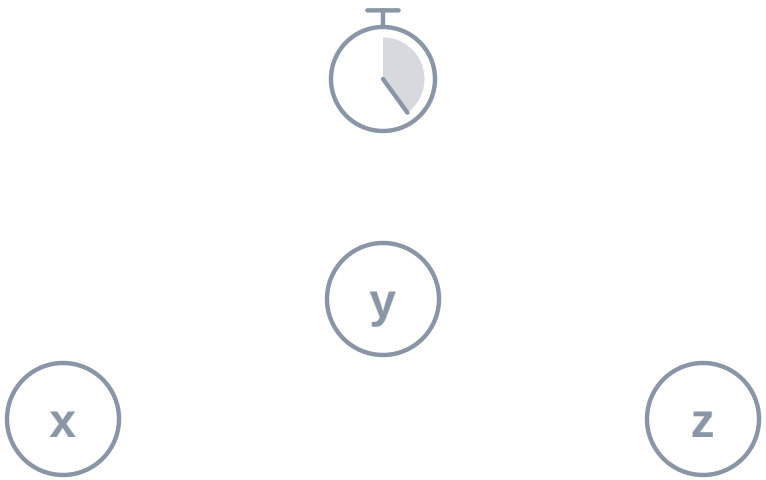
The loop every router runs



Wait, recompute, send if changed. The same three steps at every router.

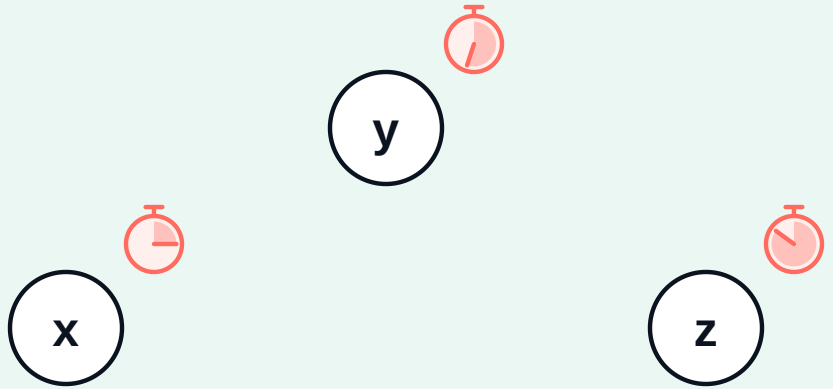
Do the routers take turns?

one clock, everyone in step



not how it works

each router on its own time

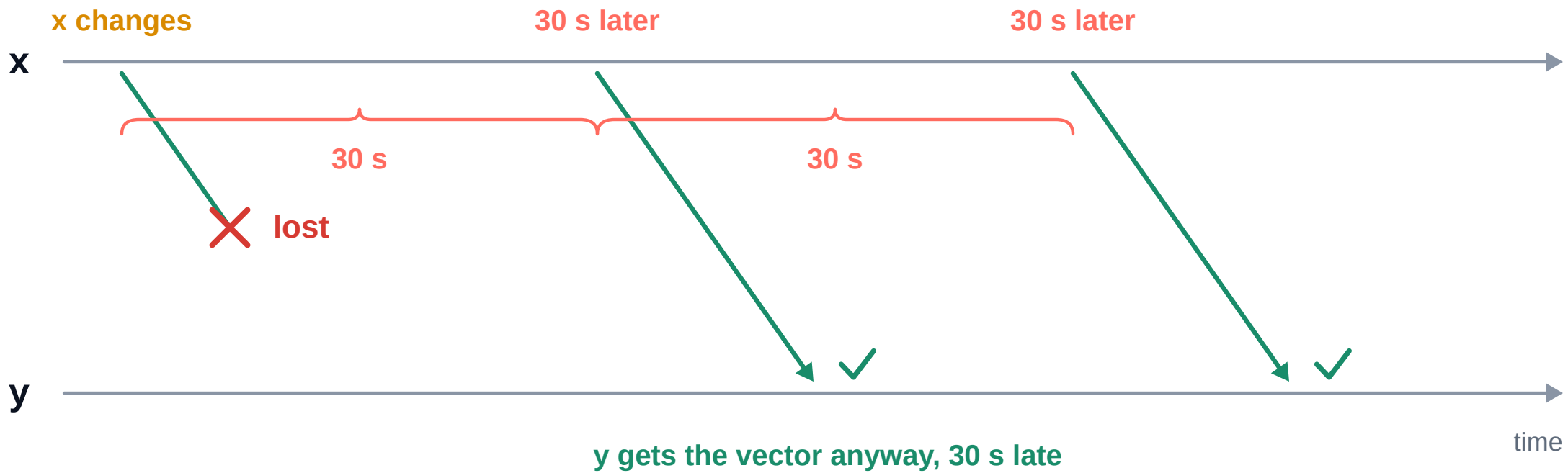


a vector arrives whenever it arrives

No rounds and no shared clock. We drew rounds only to follow along.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

What if a vector is lost?



RIP, a distance-vector protocol, resends its vector every 30 seconds.

Routers resend periodically: a lost vector is replaced, a silent neighbour noticed.

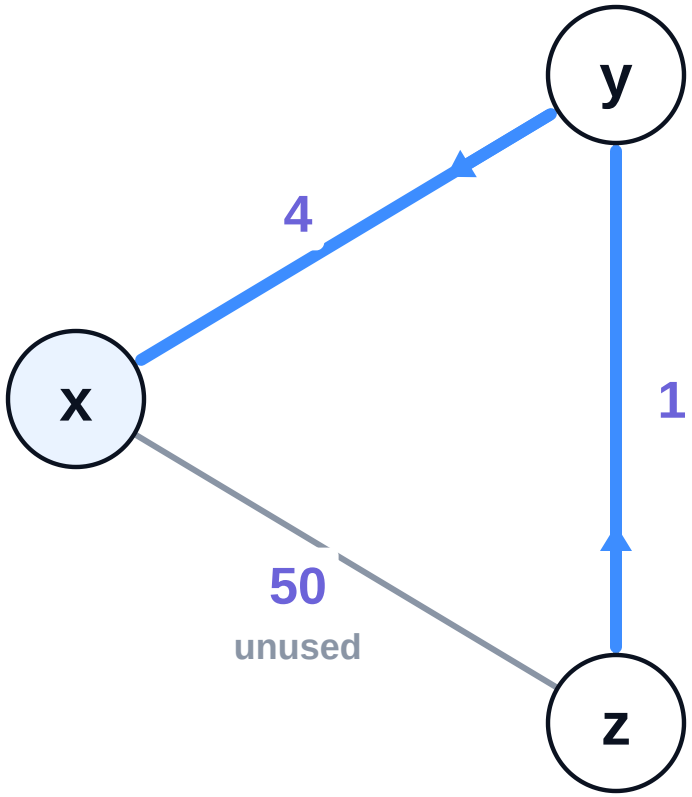
A link cost changes. Only its two ends see it

03

A link cost changes

Every other router still holds numbers that were computed for the old cost.

A network that has settled

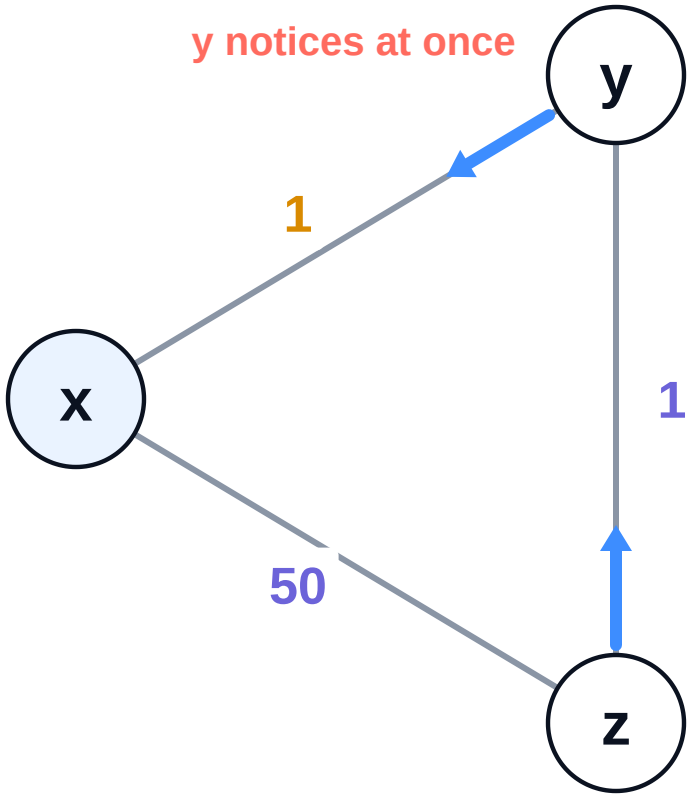


router	own link to x	neighbour said	cost to x	next hop
y	4	z: 5	4	x
z	50	y: 4	5	y

z reaches x through y: `1 + 4 = 5`

Every table is correct. We follow only the costs to x: y has 4, z has 5.

What happens when a link gets cheaper?



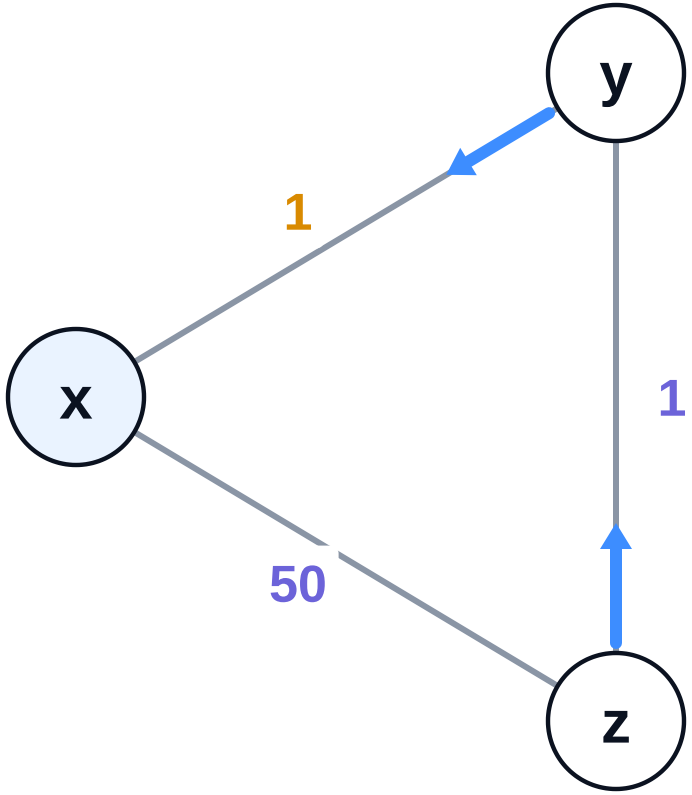
router	own link to x	neighbour said	cost to x	next hop
y	1	z: 5	4	x
z	50	y: 4	5	y

a change in its own link wakes a router, like a neighbour's vector

It is y's own link, so y recomputes at once. z has heard nothing yet.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

y recomputes and tells z



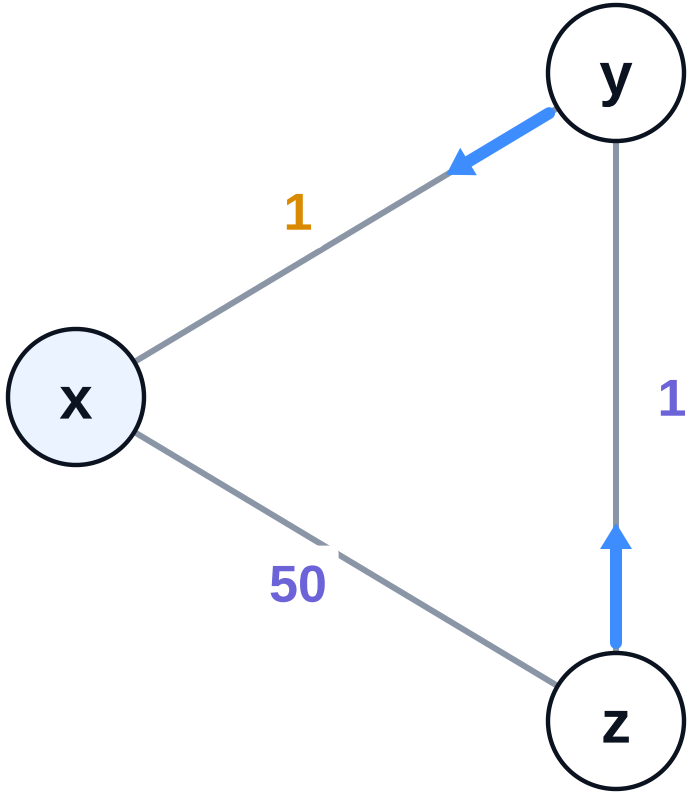
z hears 1

router	own link to x	neighbour said	cost to x	next hop
y	1	z: 5	1	x
z	50	y: 1	5	y

```
y:  min{ 1 + 0 , 1 + 5 } = 1
```

y's cost to x falls from 4 to 1, and y sends its new vector.

z recomputes, and it is over



router	own link to x	neighbour said	cost to x	next hop
y	1	z: 2	1	x
z	50	y: 1	2	y

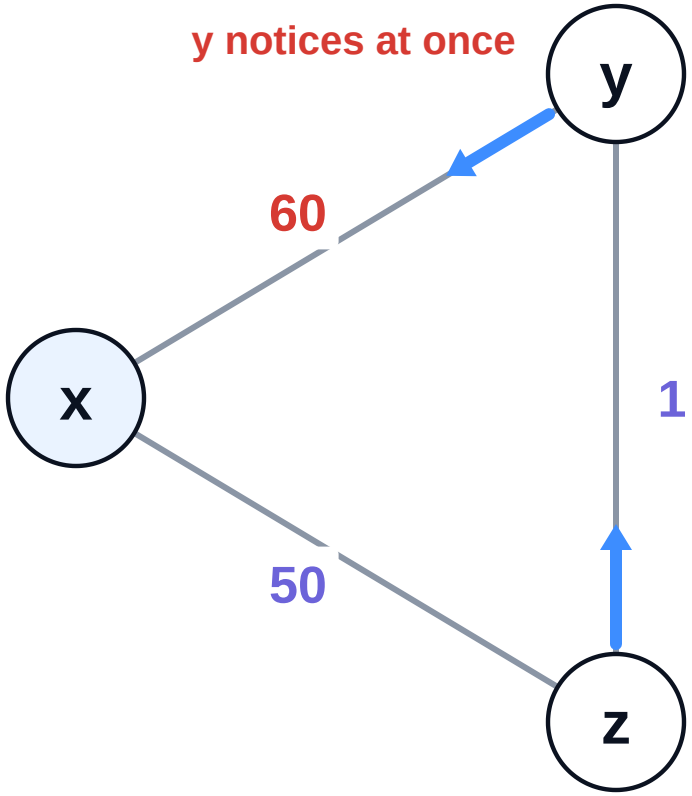
y: min{ 1 + 0 , 1 + 2 } = 1

2 exchanges

unchanged: nothing more to send

Two exchanges and every table is right. A cost that falls spreads fast.

Back to the start. Now the link costs 60

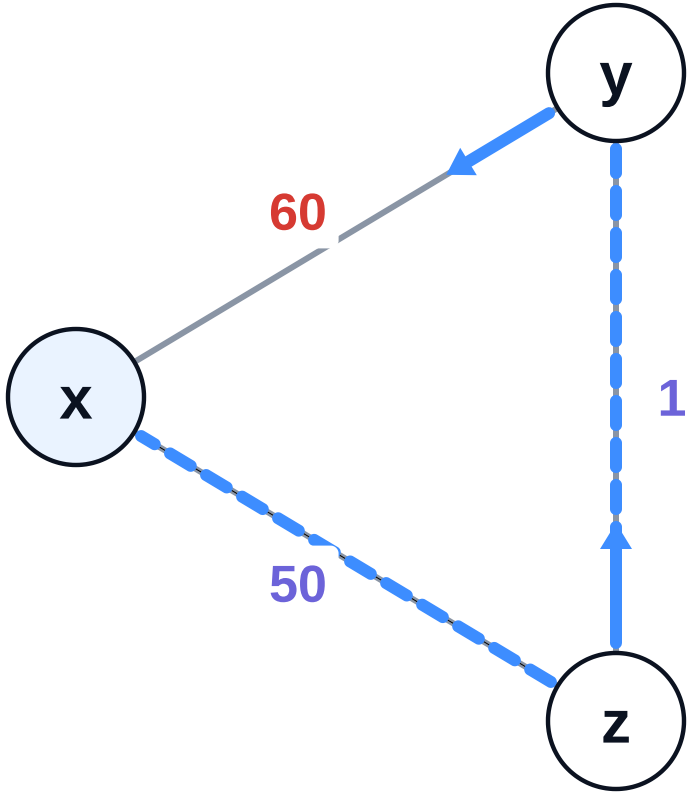


router	own link to x	neighbour said	cost to x	next hop
y	60	z: 5	4	x
z	50	y: 4	5	y

the same network and tables as before the fall

Same network, same tables, and one link now costs 60 instead of 4.

What should the tables say now?



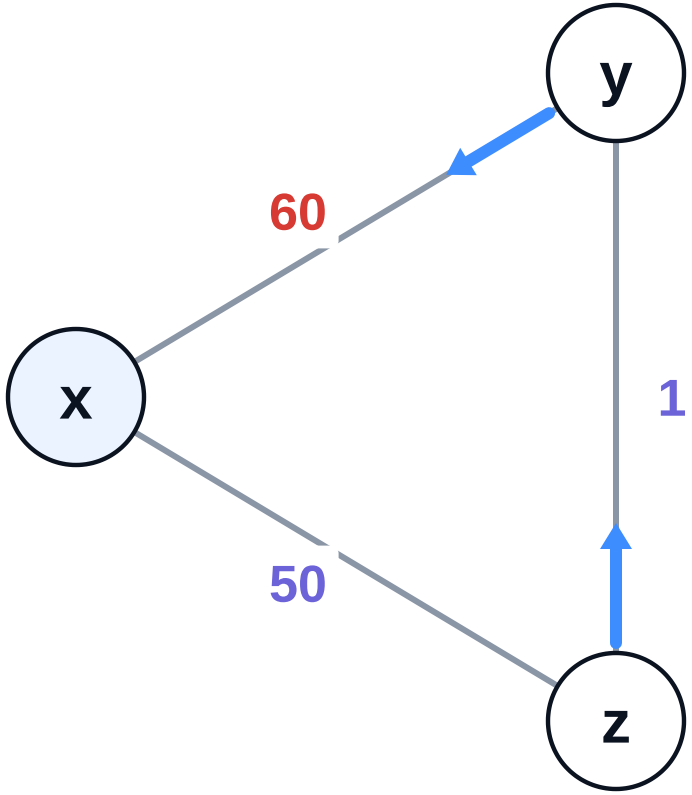
router	own link to x	neighbour said	cost to x	next hop
y	60	z: 5	4	x
z	50	y: 4	5	y

y should have 51, next hop z

z should have 50, next hop x

y should reach x for 51 through z, and z for 50 on its own link.

What does y have to work with?



router	own link to x	neighbour said	cost to x	next hop
y	60	z: 5	4	x
z	50	y: 4	5	y

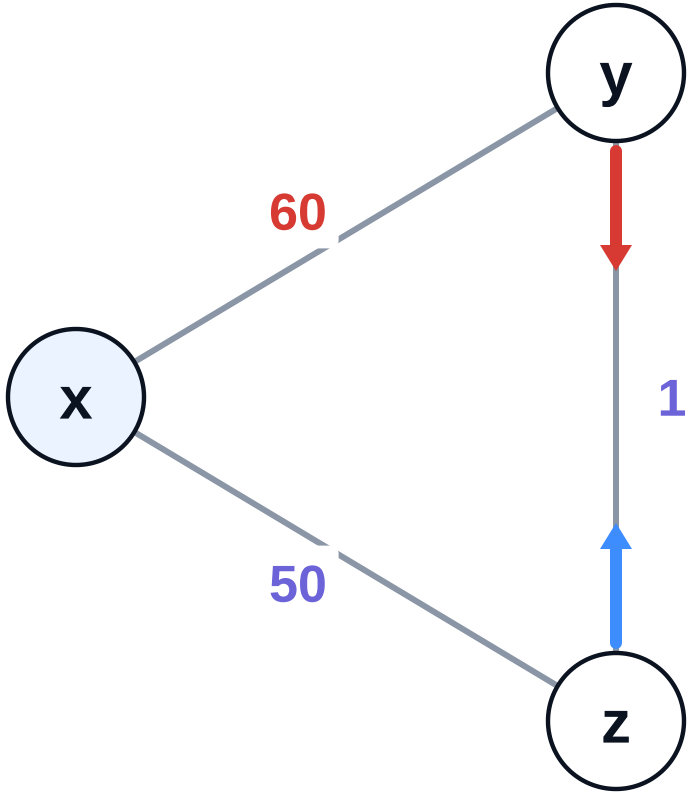
own link: 60, new

z said 5, before the change

z has heard nothing, so it has sent nothing new

A new link cost and an old report. y has nothing else.

y picks 6, through z



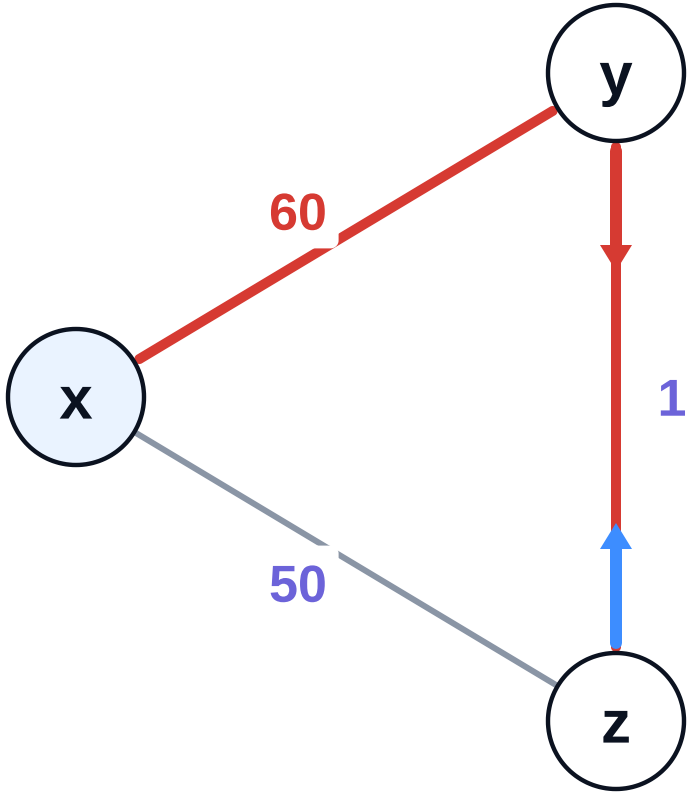
router	own link to x	neighbour said	cost to x	next hop
y	60	z: 5	6	z
z	50	y: 4	5	y

y: min{ ~~60 + 0~~ , 1 + 5 } = 6

6 looks cheaper than its own link at 60

6 looks cheaper than 60, so y now sends packets for x to z.

Where did z's 5 come from?



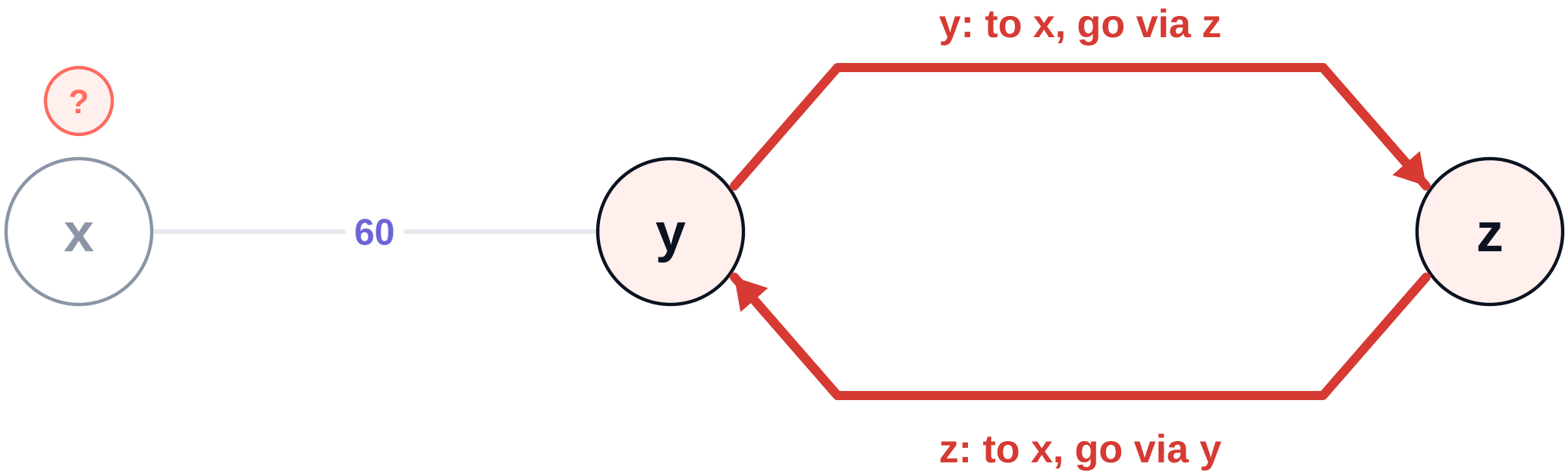
router	own link to x	neighbour said	cost to x	next hop
y	60	z: 5	6	z
z	50	y: 4	5	y

z's 5 = 1 + 4: through y, over the old 4

that link now costs 60

z's 5 was a route through y, over the link that now costs 60.

But z's route to x goes through y



Each points at the other for x, and neither can tell.

Why can't y see the loop?

what z sent



one number, no path

what z did not send

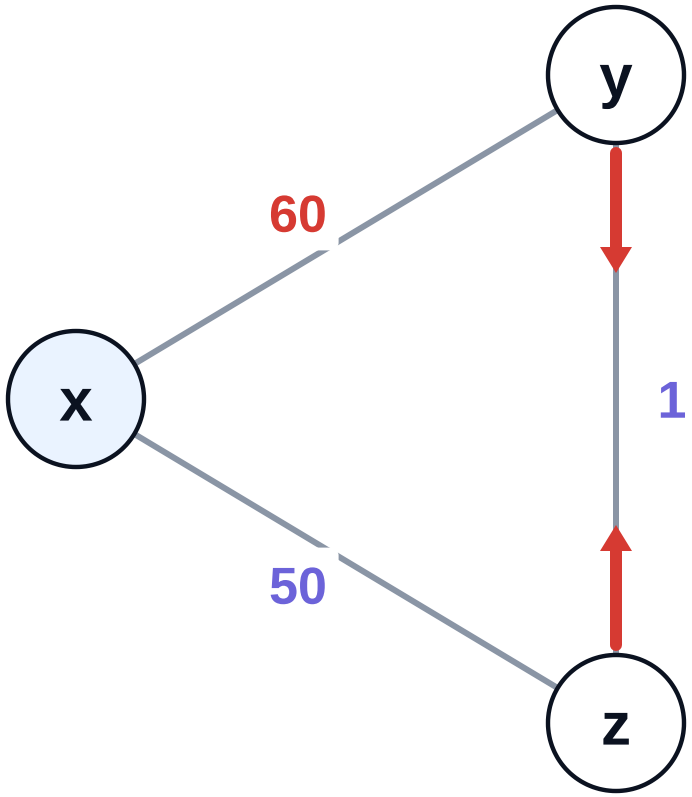


path: z → y → x

y would see its own name

A distance vector carries a cost and no path: y has nothing to check.

Where does a packet for x go now?



router	own link to x	neighbour said	cost to x	next hop
y	60	z: 5	6	z
z	50	y: 4	5	y

TTL 58 … every bounce, one less

at 0 the packet is dropped: x is unreachable meanwhile

Packets for x bounce between y and z until their TTL runs out.

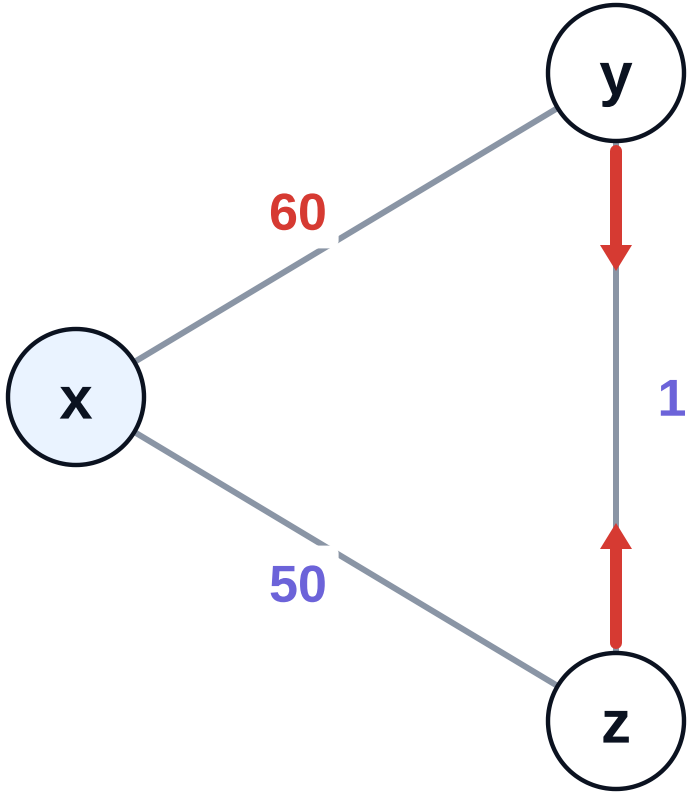
Packets for x are looping. Does it end by itself?

04

Counting to infinity

y routes to x through z, and z through y. Both keep running the same equation.

y sends its 6 to z



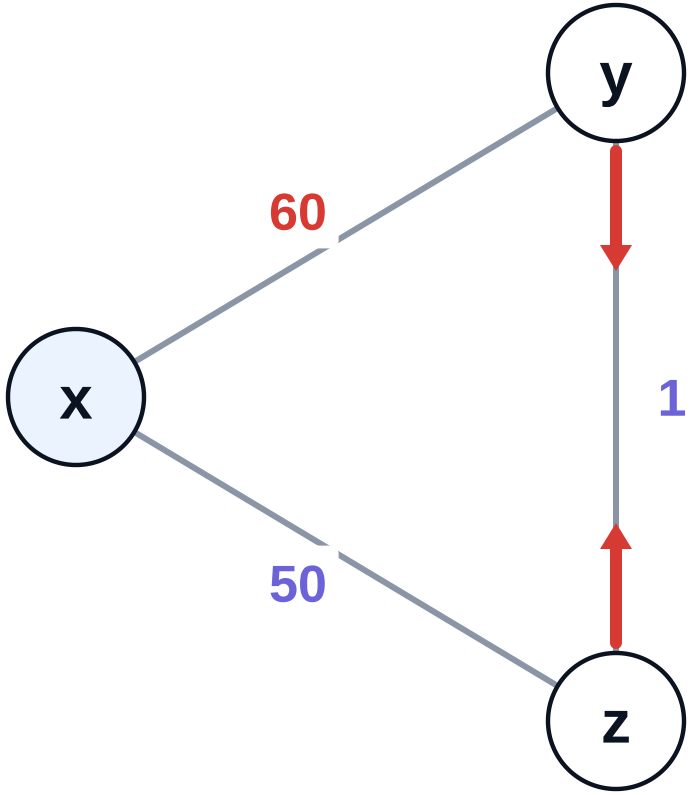
router	own link to x	neighbour said	cost to x	next hop
y	60	z: 5	6	z
z	50	y: 6	5	y

z replaces y's old 4 with 6

nothing in the message says it is wrong

z replaces y's old 4 with 6. It has no reason to doubt it.

z recomputes and gets 7



router	own link to x	neighbour said	cost to x	next hop
y	60	z: 5	6	z
z	50	y: 6	7	y

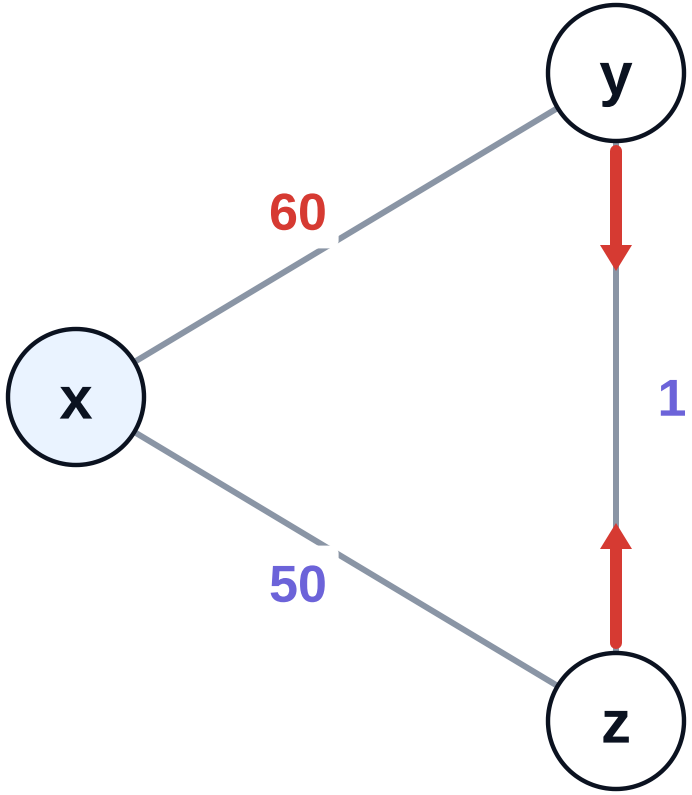
z: min{ 50 + 0 , 1 + 6 } = 7

the count:

y	z
6	7

7 still looks cheaper than z's own link at 50. z keeps routing through y.

Then y gets 8, z gets 9, y gets 10



router	own link to x	neighbour said	cost to x	next hop
y	60	z: 9	10	z
z	50	y: 8	9	y

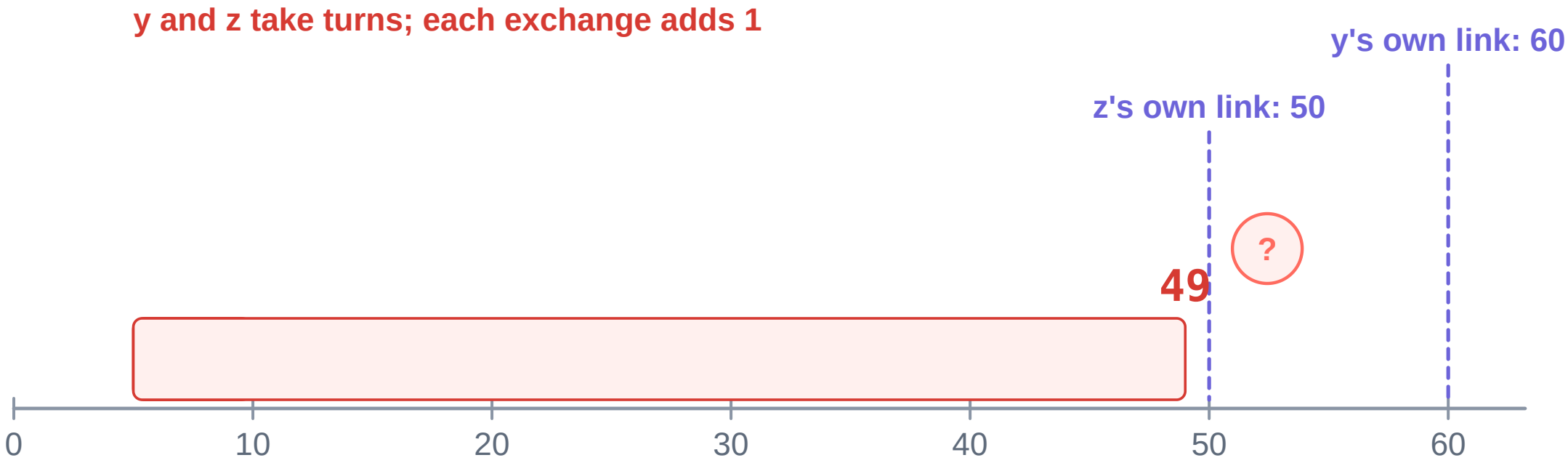
y: min{ 60 + 0 , 1 + 9 } = 10

the count:

y	z	y	z	y
6	7	8	9	10

Each exchange adds 1, and neither router changes its next hop.

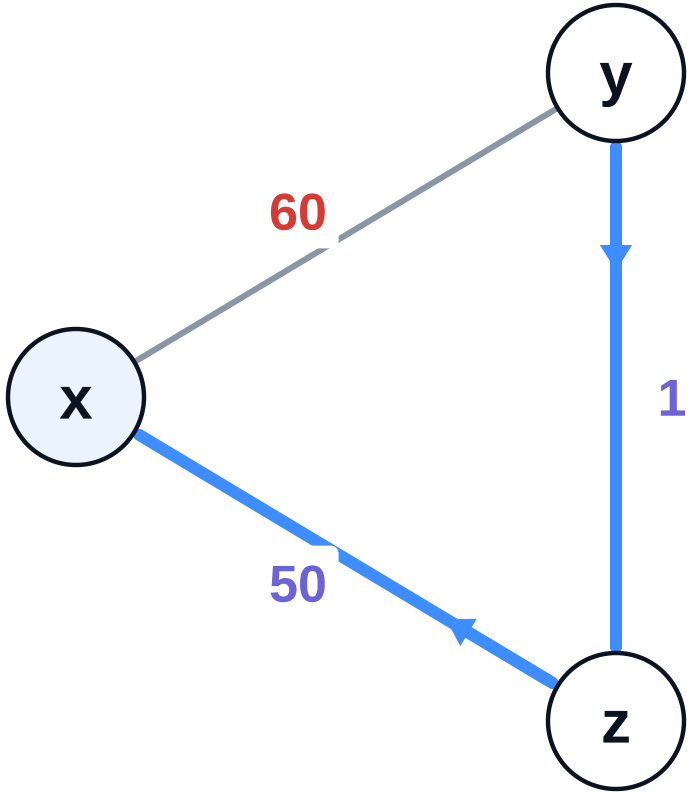
When does the counting stop?



Only when going through the other router costs more than its own link.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

z's own link finally wins



router	own link to x	neighbour said	cost to x	next hop
y	60	z: 50	51	z
z	50	y: 50	50	x

y: min{ 60 + 0 , 1 + 50 } = 51

44 exchanges after the change

After 44 exchanges y has 51 and z has 50. The loop lasted the whole time.

What set the number 44?

this network: z's own link costs 50



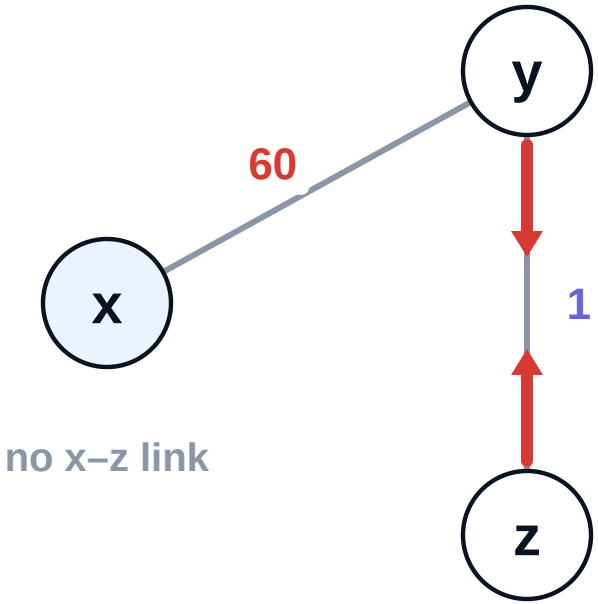
z's link at 500, and the link that changed at 1000

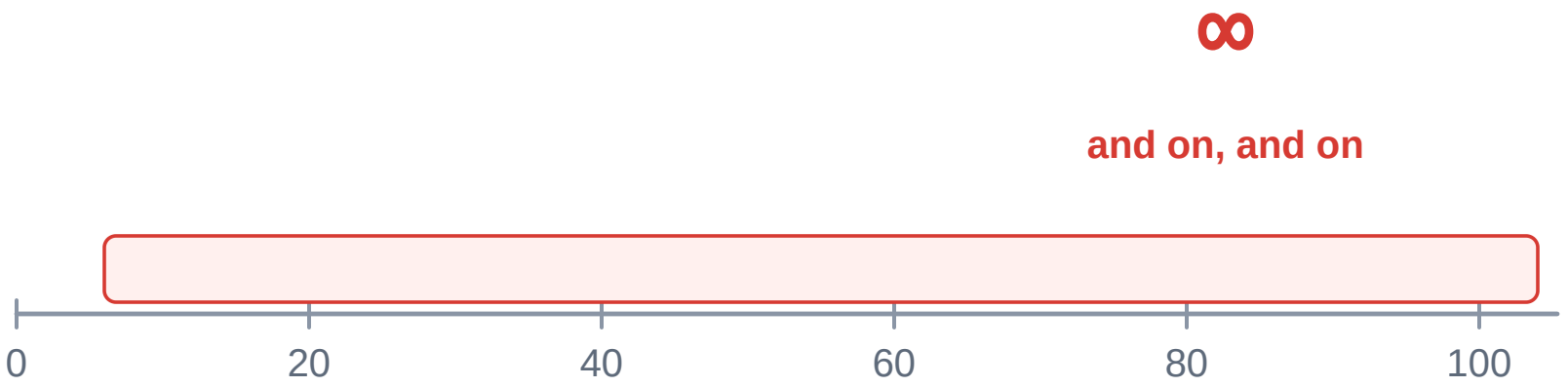


the same two routers, the same rule; only the costs changed

It climbs 1 per exchange up to the cheapest real route. Costs set its length.

What if z had no link to x at all?

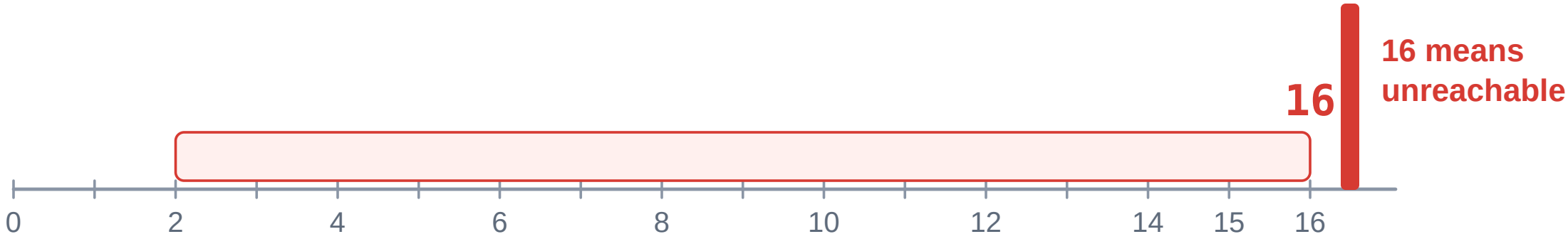




counting to infinity

With no other route the two would count up for ever: counting to infinity.

How do real protocols stop the count?



every link costs 1 (one hop), so the count reaches 16 in a few exchanges

the price: a destination 16 hops away cannot be reached at all

RIP counts hops and treats 16 as infinity (RFC 2453).

Pick a small number to mean infinity. The count ends fast; routes stay short.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

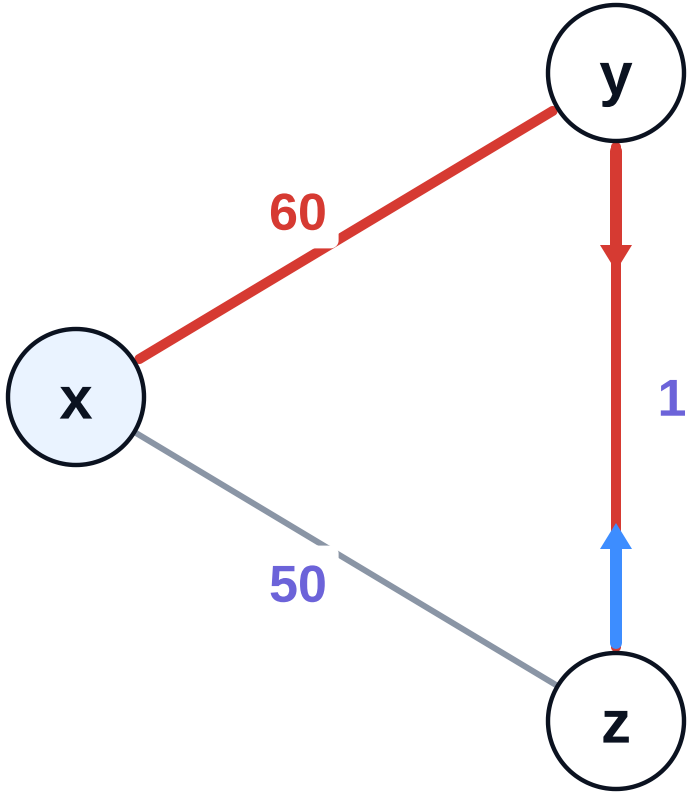
y trusted a cost that z had computed through y

05

Poisoned reverse

The loop grew from one number that z sent to y. Can z avoid sending it?

Which number started the loop?



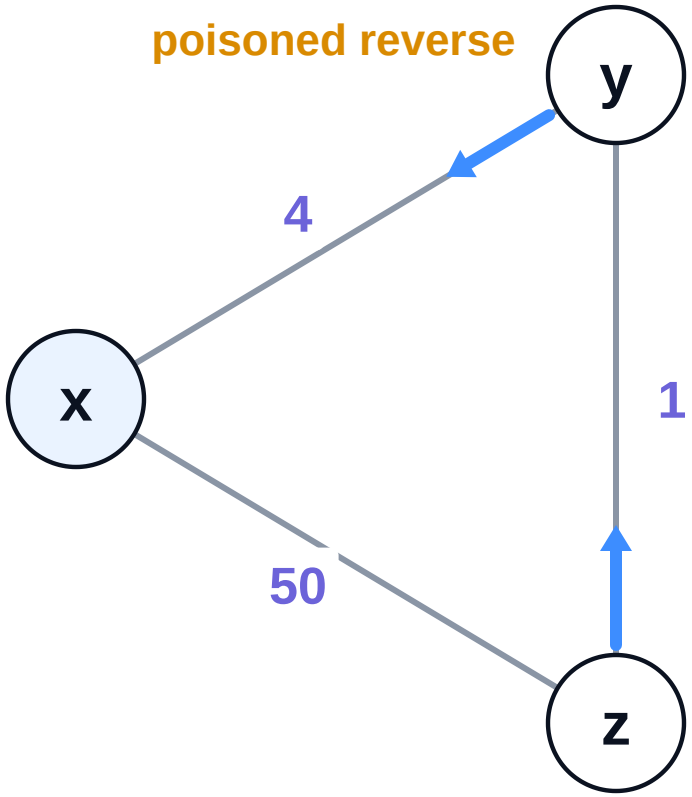
router	own link to x	neighbour said	cost to x	next hop
y	60	z: 5	6	z
z	50	y: 4	5	y

y: min{ 60 + 0 , 1 + 5 } = 6

that 5 was 1 + 4: z's route through y itself

z told y a cost that depended on y's own link. That one number started it.

z tells y its cost to x is infinity



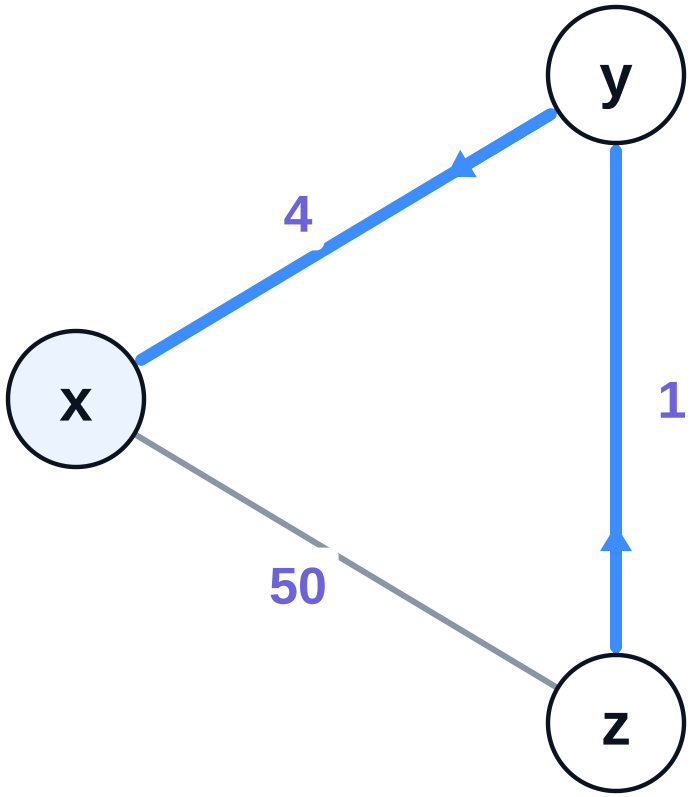
router	own link to x	neighbour said	cost to x	next hop
y	4	z: ∞	4	x
z	50	y: 4	5	y

if z reaches x through y, z tells y: to x = ∞

to its other neighbours z still sends the real cost

A router reports ∞ to the neighbour it routes through, the truth to the rest.

Does z still forward packets correctly?



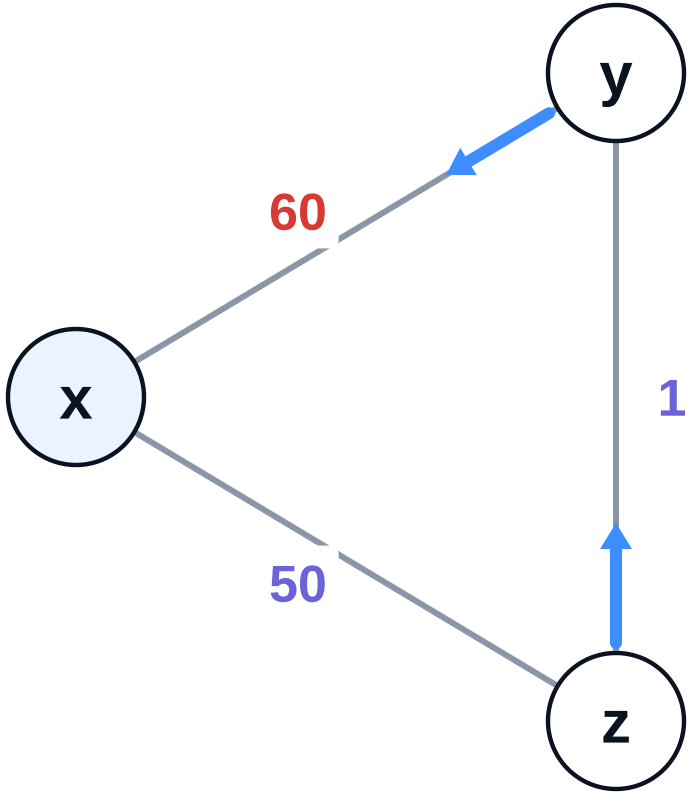
router	own link to x	neighbour said	cost to x	next hop
y	4	z: ∞	4	x
z	50	y: 4	5	y

z's table: 5 through y, unchanged

z's message to y: to x = ∞

z's own table does not change. Only the message to y does.

The link rises to 60 again. What does y compute?



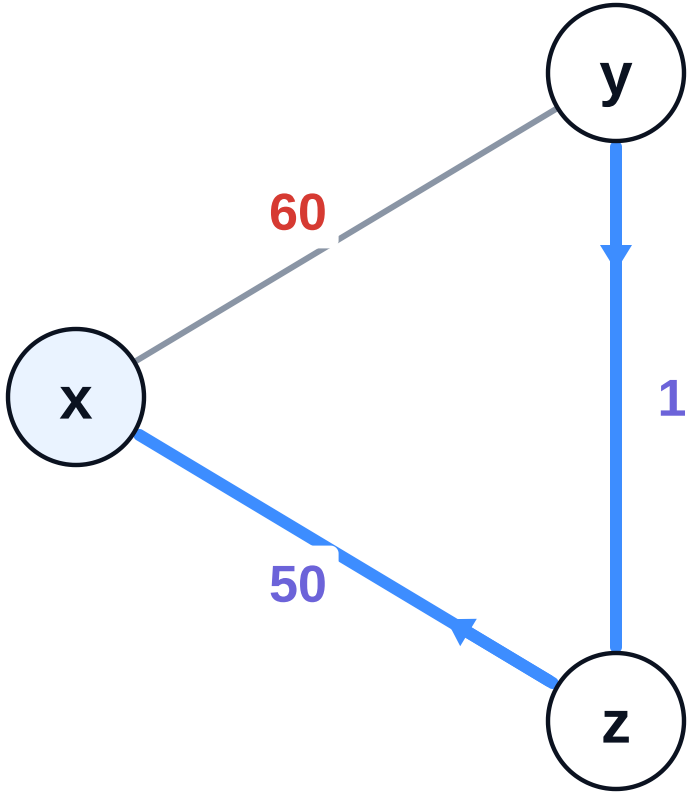
router	own link to x	neighbour said	cost to x	next hop
y	60	z: ∞	60	x
z	50	y: 60	5	y

y: min{ 60 + 0 , 1 + ∞ } = 60

y has no false number to fall back on, so it keeps its own link at 60.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

z switches to its own link, and y follows



router	own link to x	neighbour said	cost to x	next hop
y	60	z: 50	51	z
z	50	y: ∞	50	x

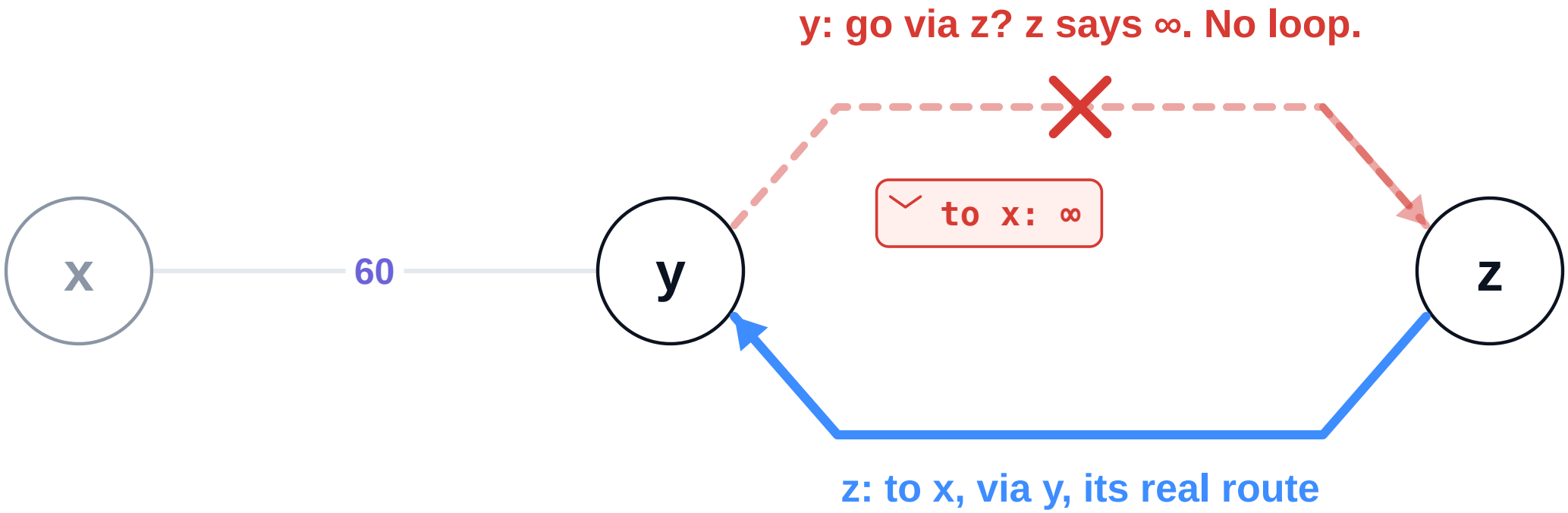
z: min{ 50 + 0 , 1 + ∞ } = 50

3 exchanges instead of 44

Three exchanges, and at no moment do two routers point at each other.

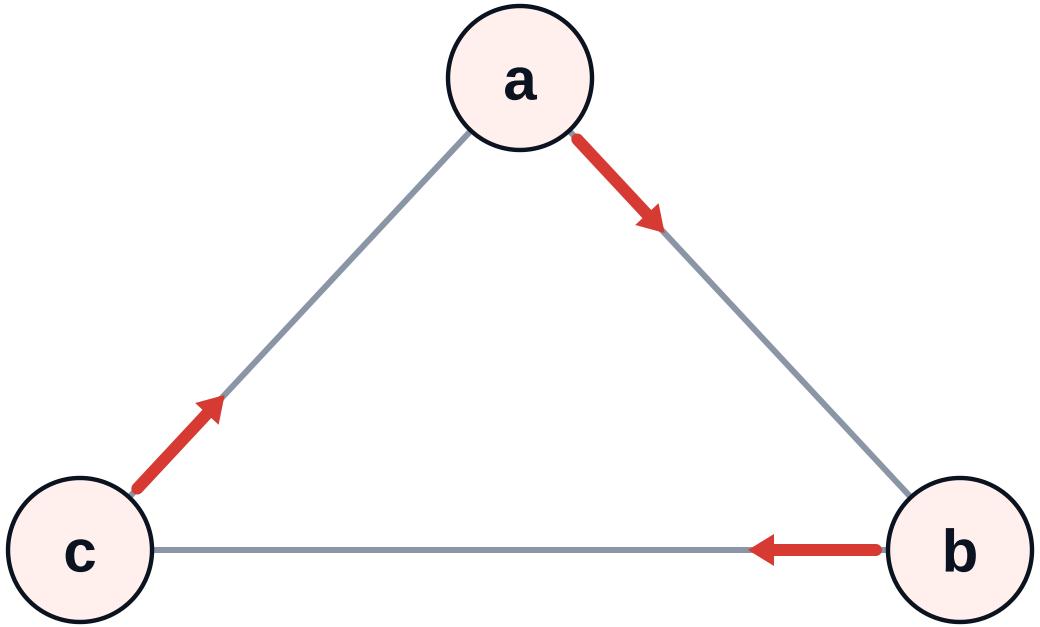
1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

Why does it work here?



In a loop of two, the router that would be fooled is exactly the one told ∞.

Does it work for a loop of three?



next hops: a → b → c → a

each router poisons only its next hop

a routes through b: a tells b ∞

b routes through c: b tells c ∞

c routes through a: c tells a ∞

but a uses the number **b sends to a**,

and b sends a its **real (wrong) cost**

the loop is never told

RFC 2453 §3.4.4: with poisoned reverse, three routers can still be “engaged in mutual deception”.

No. a relies on b, and b poisons only c: the number a uses is never poisoned.

Two algorithms fill the same table. What does each pay?

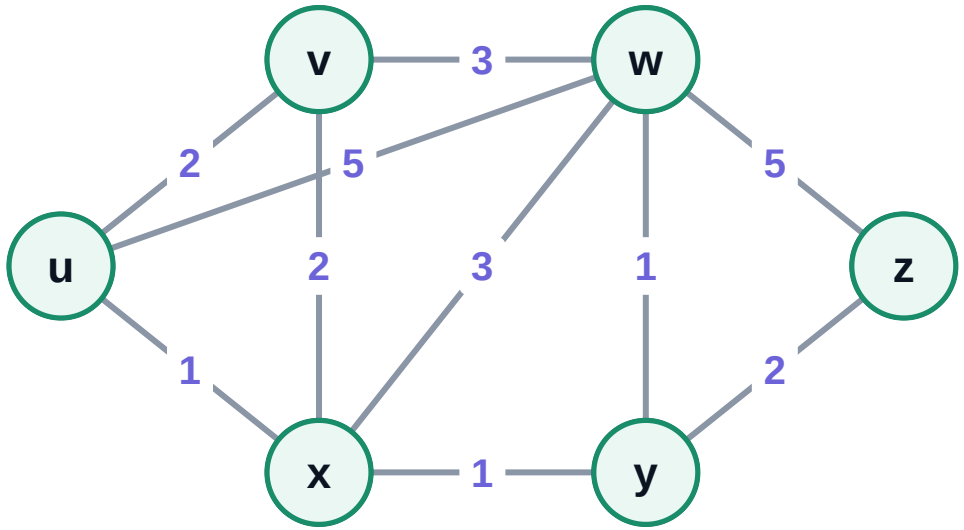
06

Link state vs distance vector

Link state sends each router's links to everyone. Distance vector sends each router's costs to its neighbours.

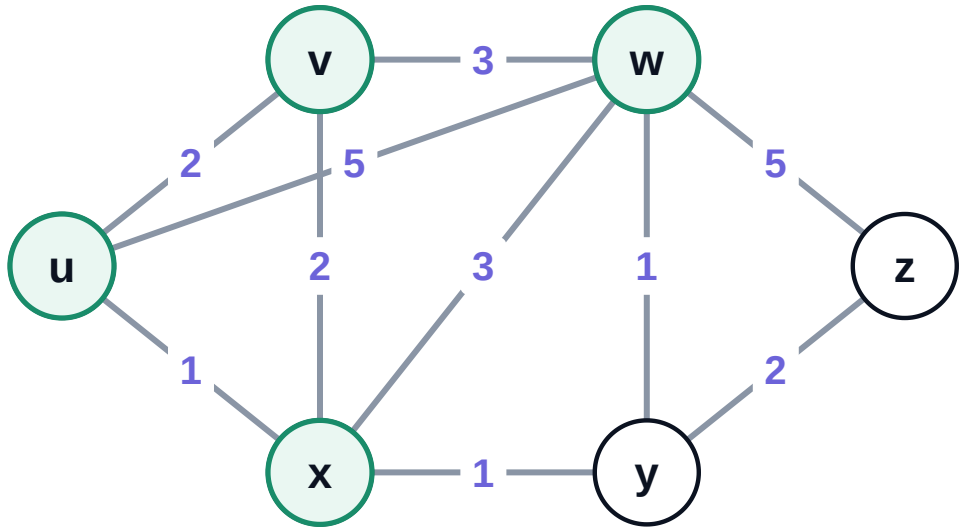
Who hears one router's message?

link state



u's message reaches all five routers

distance vector



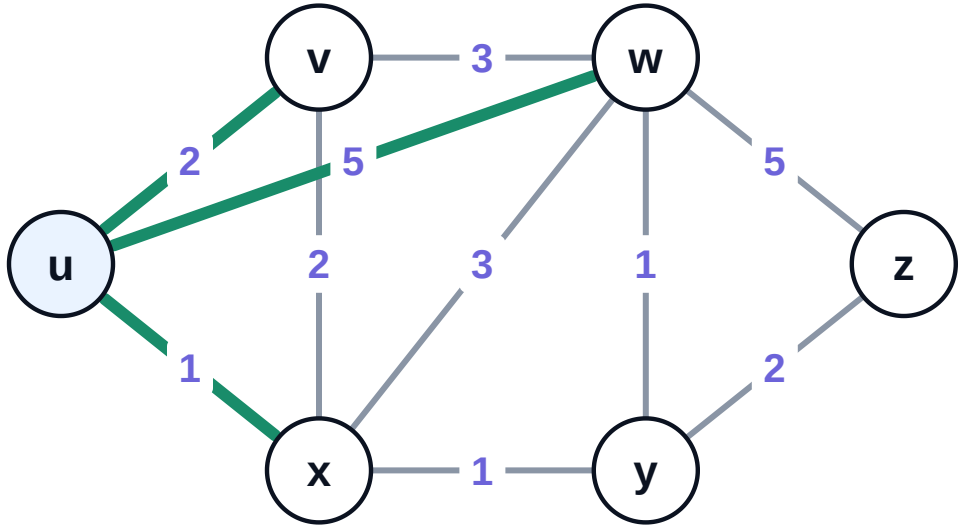
only u's three neighbours hear u

Link state: every router. Distance vector: the neighbours only.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

What is in each message?

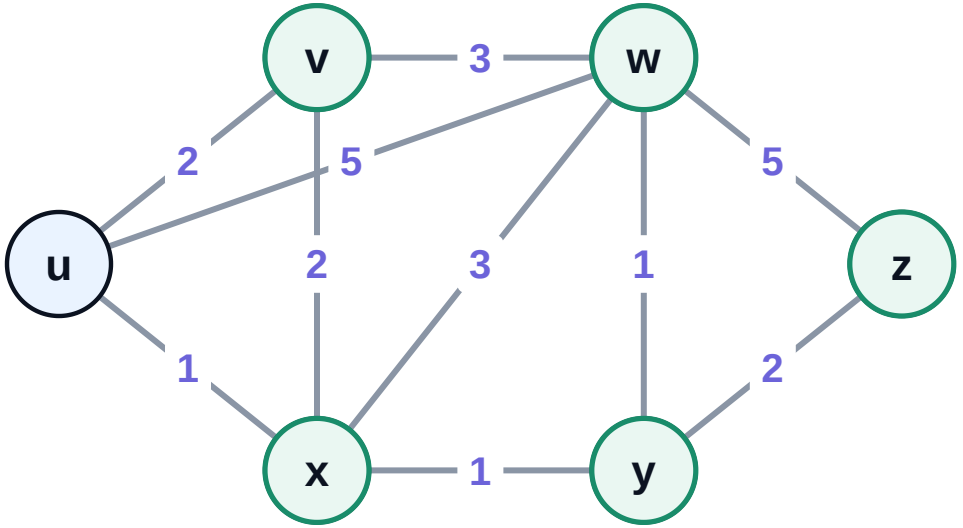
link state



`my links: v 2, x 1, w 5`

a fact about me, sent to everyone

distance vector



`my costs: v 2, w 3, x 1, y 2, z 4`

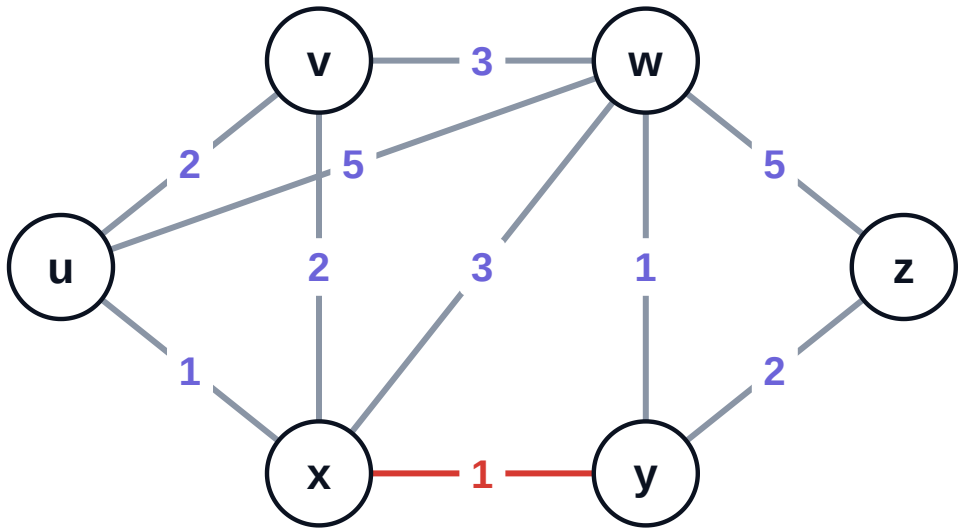
a claim about everywhere, to my neighbours

Link state reports what a router sees; distance vector, what it has concluded.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

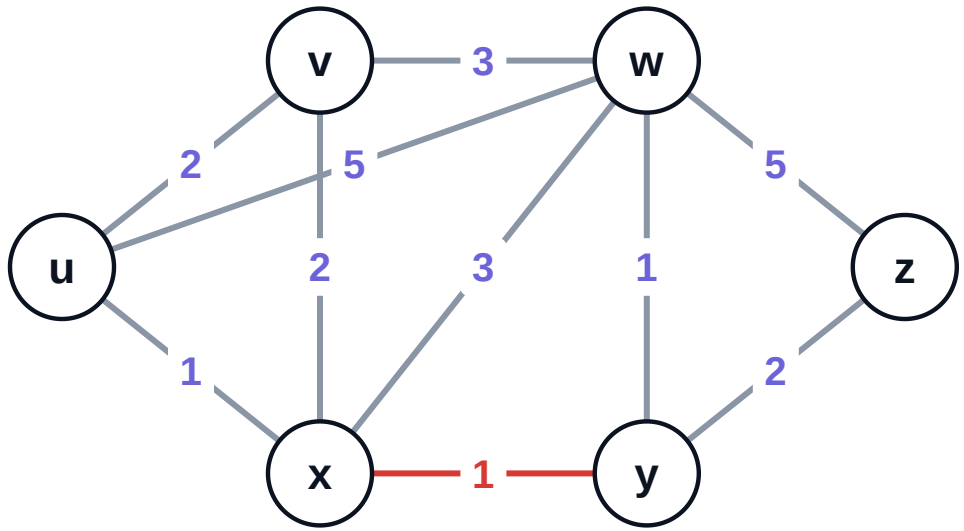
What does one link change cost in messages?

link state



one flood, then silence

distance vector

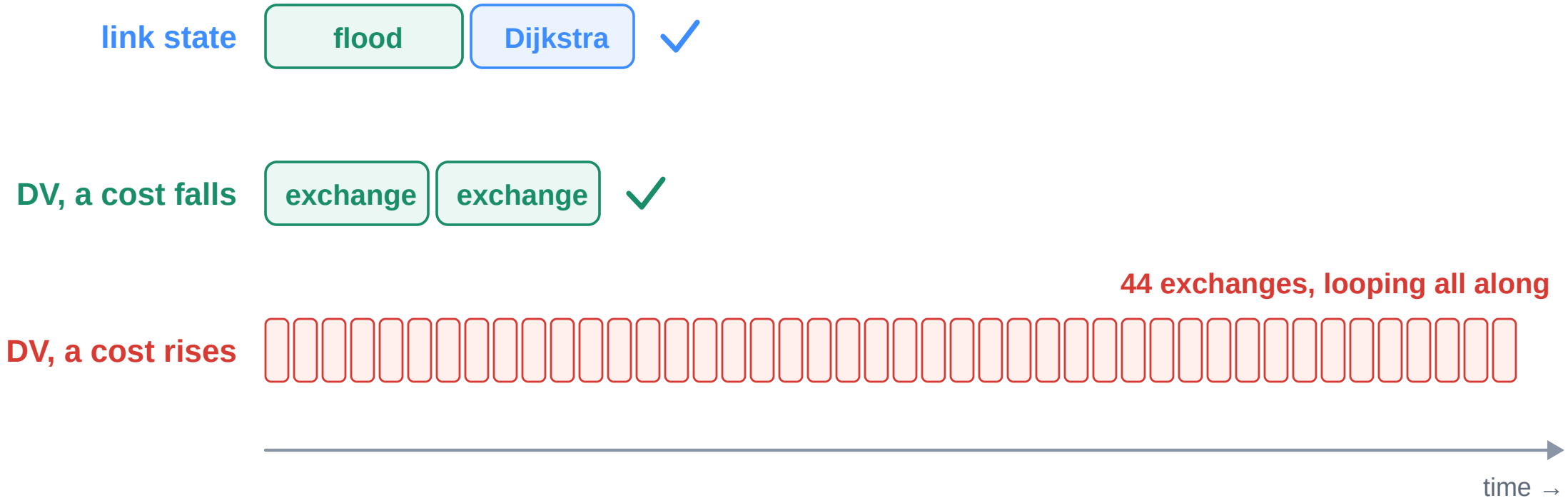


exchanges until nothing changes

after a rise, 44 of them (point 04)

Link state pays one flood every time. Distance vector pays per exchange.

How long until every table is right?

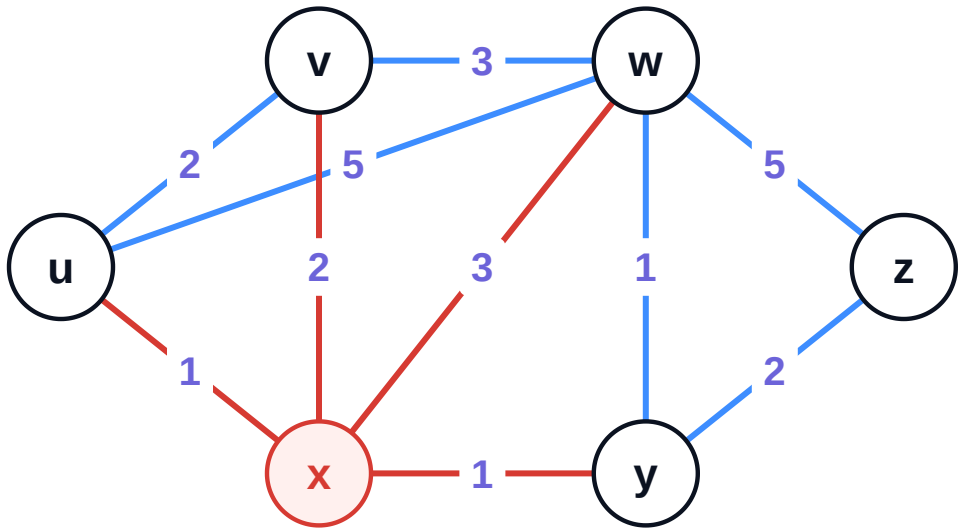


Link state: one flood and one computation. DV: as long as the costs make it.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

What happens when one router gets it wrong?

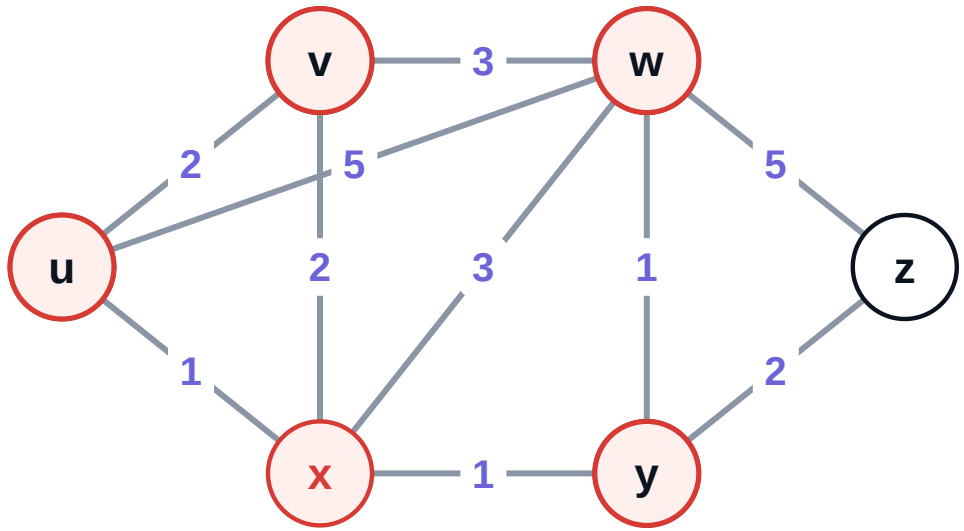
link state



x misreports its own links: only those

every other link is reported by its own routers

distance vector



x claims “to z: `0`”; the others believe it

every router now sends z's traffic to x

Link state: the damage stays on its own links. Distance vector: it spreads.

Why does nobody catch the wrong claim?

what u can see



its links, and what
its neighbours say

from x

`to z: 0`

accepted
nothing to check it against

added to
u: link 1 + 0 = 1

passed on
u tells its neighbours “1”

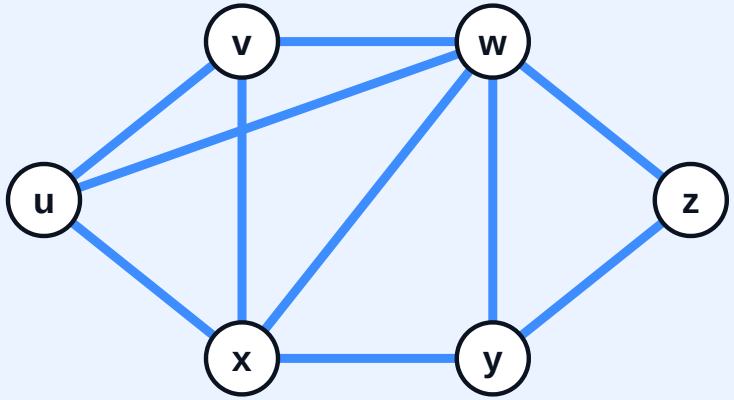
z is out of u's sight: nothing tells u the 0 is false

In 1997 one faulty router at a small ISP sent such claims to backbone routers; large parts of the internet were disconnected for up to several hours (K&R 5.2).

A router cannot check a claim about a destination it cannot see.

What does each router store?

a link-state router



every link in the network

and Dijkstra over all of it

a distance-vector router

	to v	to w	to x	to y	to z
u	**2**	**3**	**1**	**2**	**4**
v	0	3	2	3	5
x	2	2	0	1	3
w	3	0	2	1	3

one vector per neighbour

and a minimum per destination

Distance vector keeps less and computes less. It gives up predictability.

1 Equation · 2 Algorithm · 3 Link change · 4 Counting · 5 Poisoning · 6 LS vs DV

Side by side

	link state	distance vector
who hears a message	every router	the neighbours
what it says	my own links	my cost to everywhere
one link change	one flood	exchanges until quiet
time to converge	one flood + Dijkstra	set by the costs; can loop
one faulty router	its own links only	spreads everywhere
what a router keeps	the whole map	its neighbours' vectors

Neither wins every row. Both are in use, in different places.

1 Equation | 2 Algorithm | 3 Link change | 4 Counting | 5 Poisoning | 6 LS vs DV

Six ideas, one per point

01 The equation

cost = min over neighbours of
link + report; winner = next hop

02 The algorithm

send your vector to the neighbours,
recompute on arrival, stop when quiet

03 A cost changes

a fall spreads in two exchanges;
a rise meets old reports: a loop

04 Counting

+1 per exchange until a real route
is cheaper; with none, for ever

05 Poisoned reverse

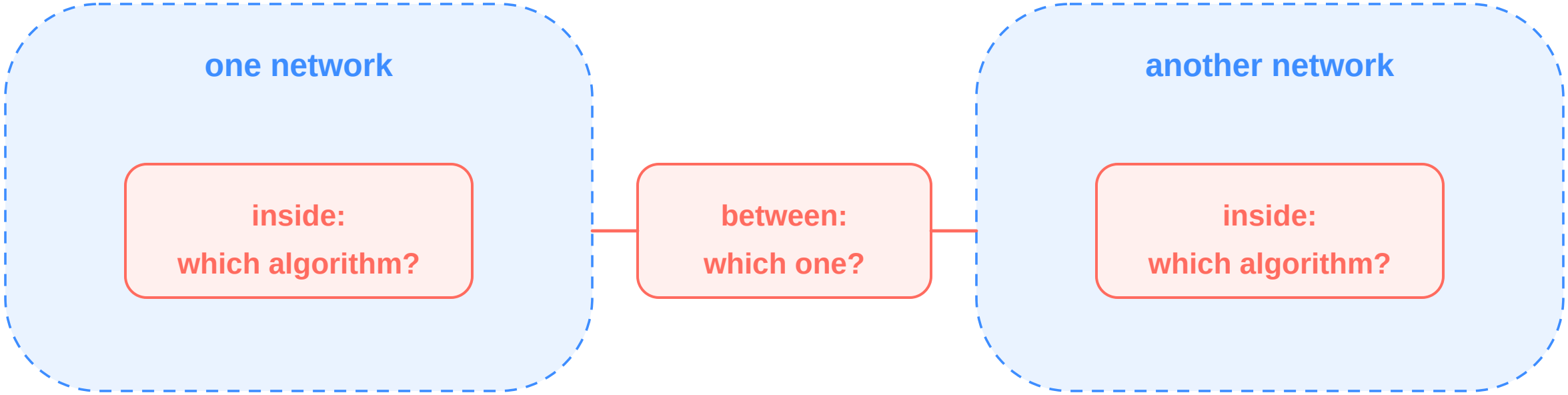
tell your next hop ∞: it breaks
loops of two, not loops of three

06 LS vs DV

LS: facts to everyone, bounded.
DV: claims to neighbours, trusted

A router can find least-cost paths from its neighbours' numbers alone, at a price.

Which one does the internet use?



next lecture: inside one network

Reading: Kurose & Ross, Section 5.2.2.

Both, in different places. The next lecture starts inside one network.