Routing #2

Spring 2024
cs168.io

Rob Shakir

Last Time

- Talked about what a router *is* and why we need/want them.
- Defined routing and forwarding.
- Thought about what makes routing *valid*
- Demonstrated human-based routing and forwarding.

Plan for today

- Types of routing protocols.
- More about *Distance-Vector* routing protocols.

Inter-domain and Intra-domain routing

- The Internet does not have a single giant routing protocol.

Inter-domain and Intra-domain routing

- The Internet does not have a single giant routing protocol.

- The Internet is a network of networks.
    - How we route traffic on one may not be the best way on another (why?)

Inter-domain and Intra-domain routing

- The Internet does not have a single giant routing protocol.

- The Internet is a network of networks.
  - How we route traffic on one may not be the best way on another (why?)
  - Networks differ!
    - Physical size, number of hosts, number of routers, bandwidth, latency, failure rate, topology, support staff size, when they were built, $ available…

- So…
  - Let individual networks choose how to route *inside* their network (intradomain)
  - …have all networks agree on how to route *between* each other (inter-domain)

Intra-Domain Routing

- ~Within a single network.
    - Technically an “autonomous system”.
    - Run by one operator.
    - Some different protocol requirements – reachability to all different nodes, and to use all capacity efficiently.
    - Base protocols are often called *Interior Gateway Protocols* or IGPs.
        - A number are used actively today – OSPF, IS-IS are the most common.

Inter-domain Routing

- Routing between networks.
  - Between autonomous systems really.
  - Used to make many networks into the Internet.
  - Protocols are called Exterior Gateway Protocols (EGPs).
  - There is only one – all ASes must agree.

- The Internet has used BGP since the 1990s.

Choosing Routing Protocols

- Interior and Exterior (intra- and inter-domain) is a convenient shorthand.

- In practice, the lines are more blurred.
  - BGP is used inside some networks as well as at the edges.

- Comes down to what information needs to be propagated and what type of routing decision is needed.
  - We’ll cover BGP in more depth later.

- We’ll understand the general difference between *Distance-Vector* and *Link-State* protocols.

Least-Cost Routing

Least-Cost Routing

- We said we wanted “good” routes.

Least-Cost Routing

- We said we wanted “good” routes.

- Goal #1: Routes that work.
  - No loops, no dead-ends.

Least-Cost Routing

- We said we wanted “good” routes.

- Goal #1: Routes that work.
  - No loops, no dead-ends.

- Goal #2: Routes that are in some way “good”.
  - Commonly done by *minimising* some metric, which we might call cost.
  - Hence *least-cost* routing.

Least-Cost Routing

- We said we wanted “good” routes.

- Goal #1: Routes that work.
  - No loops, no dead-ends.

- Goal #2: Routes that are in some way “good”.
  - Commonly done by *minimising* some metric, which we might call cost.
  - Hence *least-cost* routing.

- What did we minimise in the activity last time?

Least-Cost Routing

- We said we wanted “good” routes.

- Goal #1: Routes that work.
    - No loops, no dead-ends.

- Goal #2: Routes that are in some way “good”.
    - Commonly done by *minimising* some metric, which we might call cost.
    - Hence *least-cost* routing.

- What did we minimise in the activity last time?
    - Number of people who handled the envelope – the *hop count*

Least-Cost Routing

- What else might we minimise?

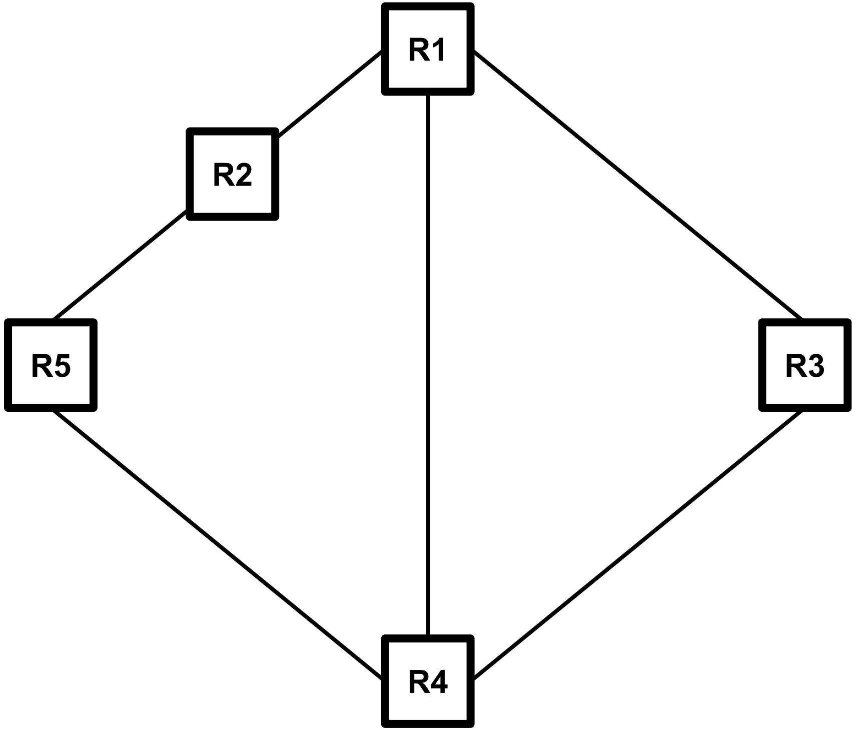
Least-Cost Routing

- What else might we minimise?
    - Price
    - Propagation delay
    - Distance
    - Unreliability
    - Bandwidth constraints

- Metrics can be arbitrarily chosen.
    - We can generically refer to this as “cost”.

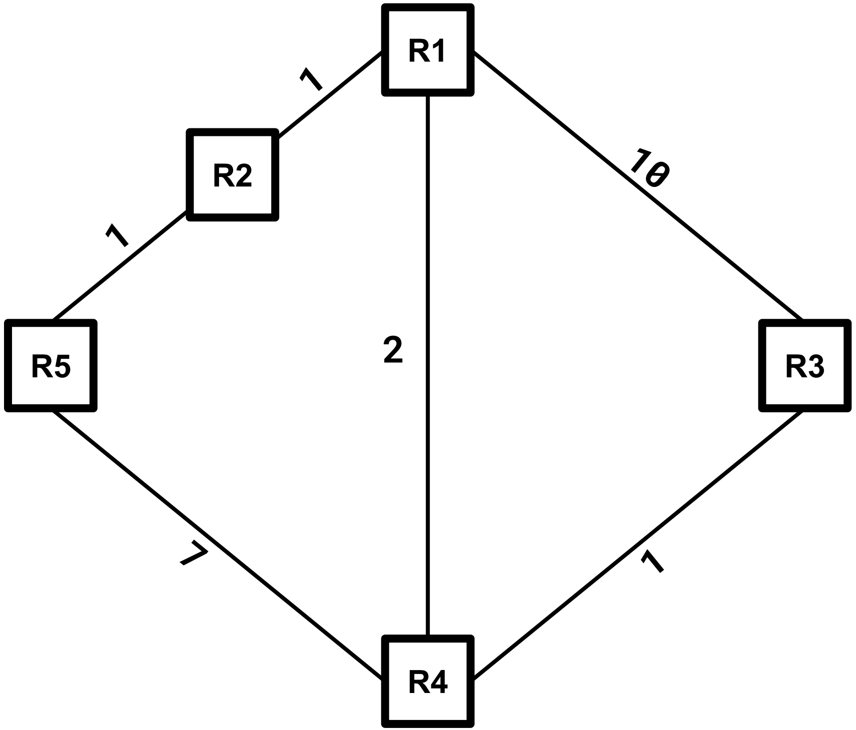
Least-Cost Routing

- For a specific network topology, say…



Least-Cost Routing

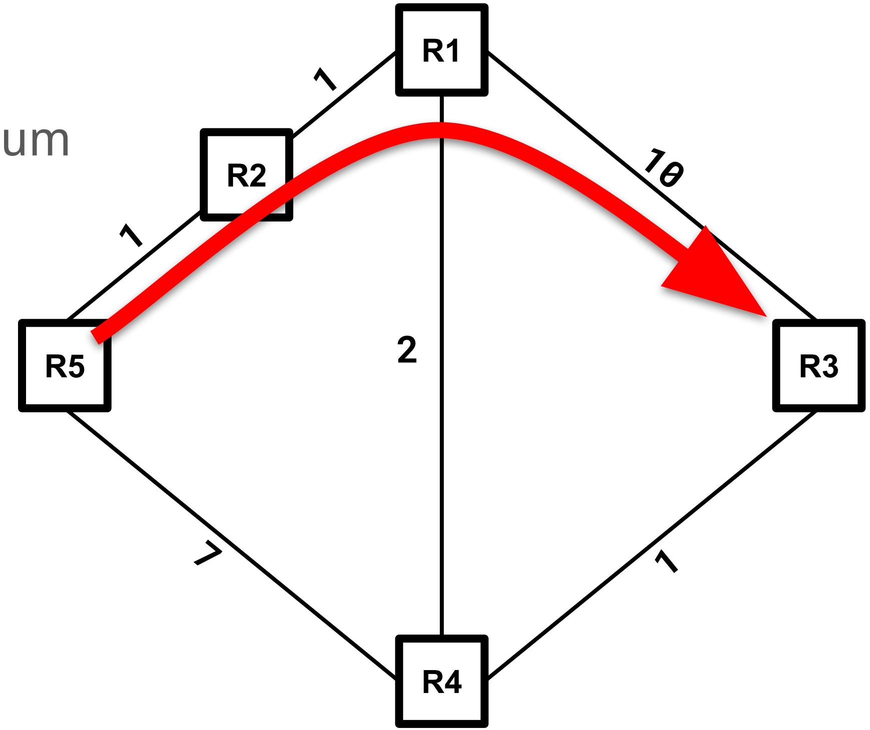
- For a specific network topology, say…
- A cost is associated with each edge.



Least-Cost Routing

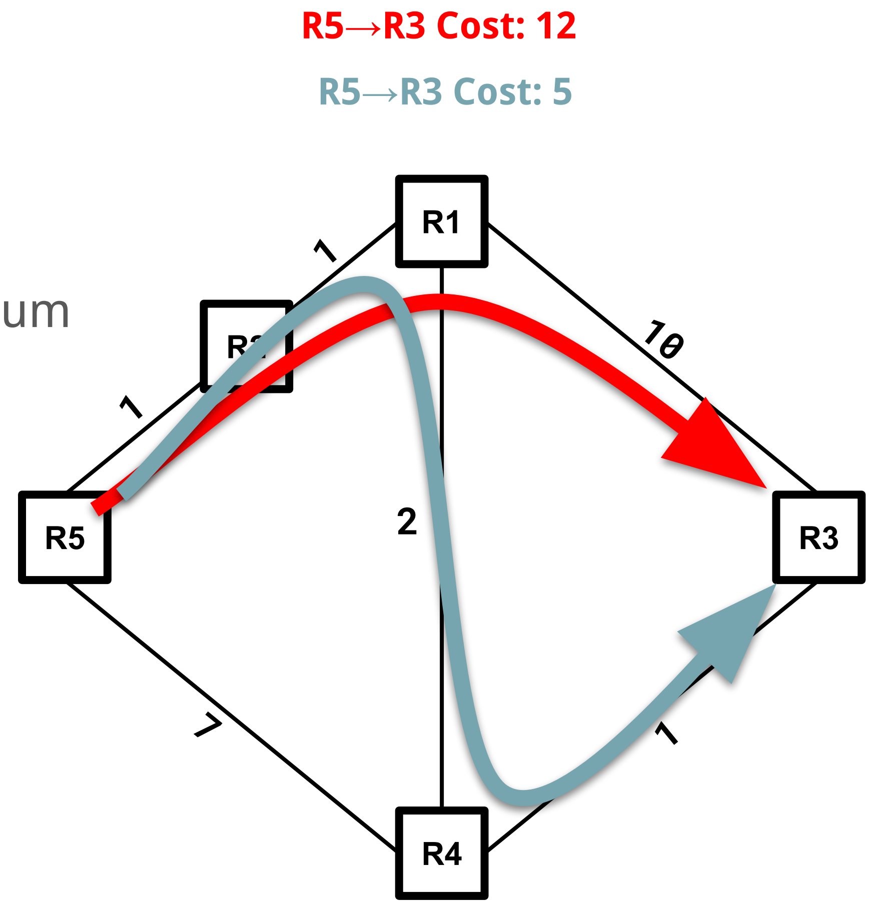
R5→R3 Cost: 12

- For a specific network topology, say…
- A cost is associated with each edge.
- And we find the path with the smallest sum



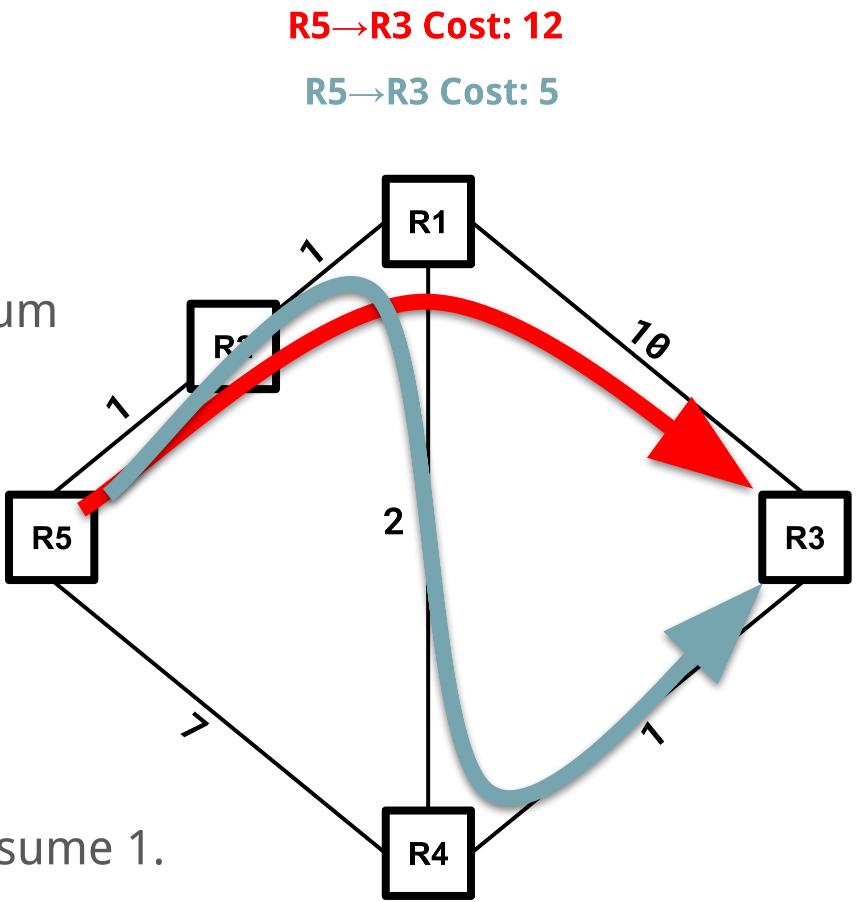
Least-Cost Routing

- For a specific network topology, say…
- A cost is associated with each edge.
- And we find the path with the smallest sum



Least-Cost Routing

- For a specific network topology, say…
- A cost is associated with each edge.
- And we find the path with the smallest sum



- In our activity:
  - Every edge had a cost of 1
  - Hence we minimised for the fewest edges
  - ⇒ fewest number of hops.

- Generally, if an edge cost is not given, assume 1.

Where do the costs come from?

- Local to a router.
  - Each router knows the cost of its own links.

- Costs are always positive integers.
  - Can’t traverse an edge and make a path cheaper!

- Costs are almost always symmetrical.
  - A→B generally costs the same as B→A.
  - Some rare exceptions.

- In practice, generally configured by an operator.
- Some protocols allow for autoconfiguration.

Are least cost routes good routes?

- Least-cost routes are an easy way to avoid loops.
  - No (sensible) metric is minimised by traversing a loop.
- Least-cost routes are destination based.
- They form a spanning tree.

“Simple” Route Types

“Connected”/”Direct” Routes

- Sometimes we need to be able to route to things that we’re actually connected to directly.

- Host A is directly connected to router 1.
    - These routes are created simply because we tell a router something about its configuration.

- Often created manually by operators.

“Static” Routes

- Routes that we aren’t necessarily directly connected to – but we always want to be there.

- “Static” because they don’t change and there’s no routing protocol used to discover them.

- Again, often manually created by an operator.

Distance-Vector Routing

Distance-Vector Routing Protocols

- Long history on the Internet and ARPANET.

- The prototypical D-V protocol is RIP.

- Strong relationship to the Bellman-Ford shortest path algorithm.
  - Our exercise was a version of Bellman-Ford.
  - With some tweaks to make it a useful routing protocol.

- We'll talk about how such a protocol actually works today.

Bellman-Ford and our In-Class Routing

```
def bellman_ford (dst, routers, links):
  distance = {}; nexthop = {}

  for each r in routers:
    distance[r] = INFINITY
    nexthop[r] = None
  distance[dst] = 0

  for _ in range(len(routers)-1):
    for (r1,r2,dist) in links:
      if distance[r1] + dist < distance[r2]:
        distance[r2] = distance[r1] + dist
        nexthop[r2] = r1

  return distance, nexthop
```

From our exercise - **magic number**

From our exercise - **best friend**

Bellman-Ford and our In-Class Routing

From our exercise - **magic number**

Start with infinity as the cost, except for our destination

```
def bellman_ford (dst, routers, links):
  distance = {}; nexthop = {}

  for each r in routers:
    distance[r] = INFINITY
    nexthop[r] = None
  distance[dst] = 0

  for _ in range(len(routers)-1):
    for (r1,r2,dist) in links:
      if distance[r1] + dist < distance[r2]:
        distance[r2] = distance[r1] + dist
        nexthop[r2] = r1

  return distance, nexthop
```

From our exercise - **best friend**

Bellman-Ford and our In-Class Routing

```
def bellman_ford (dst, routers, links):
  distance = {}; nexthop = {}

  for each r in routers:
    distance[r] = INFINITY
    nexthop[r] = None
  distance[dst] = 0

  for _ in range(len(routers)-1):
    for (r1,r2,dist) in links:
      if distance[r1] + dist < distance[r2]:
        distance[r2] = distance[r1] + dist
        nexthop[r2] = r1

  return distance, nexthop
```

From our exercise - **magic number**

From our exercise - **best friend**

Start with infinity as the cost, except for our destination

As we get new offers, compare them to our current cost.

As we get new offers, compare them to our current cost

Bellman-Ford and our In-Class Routing

```
def bellman_ford (dst, routers, links):
  distance = {}; nexthop = {}

  for each r in routers:
    distance[r] = INFINITY
    nexthop[r] = None
  distance[dst] = 0

  for _ in range(len(routers)-1):
    for (r1,r2,dist) in links:
      if distance[r1] + dist < distance[r2]:
        distance[r2] = distance[r1] + dist
        nexthop[r2] = r1

  return distance, nexthop
```

From our exercise - **magic number**

From our exercise - **best friend**

Start with infinity as the cost, except for our destination

As we get new offers, compare them to our current cost.

Accept the offer and update our best friend

Bellman-Ford and our In-Class Routing

```
def bellman_ford (dst, routers, links):
  distance = {}; nexthop = {}

  for each r in routers:
    distance[r] = INFINITY
    nexthop[r] = None
  distance[dst] = 0

  for _ in range(len(routers)-1):
    for (r1,r2,dist) in links:
      if distance[r1] + dist < distance[r2]:
        distance[r2] = distance[r1] + dist
```

From our exercise - **magic number**

From our exercise - **best friend**

Start with infinity as the cost, except for our destination

As we get new offers, compare them to our current cost.

Accept the offer and update our best friend

But we didn't do this in a loop…
We did it in parallel.

Bellman-Ford and our In-Class Routing

```
def bellman_ford (dst, routers, links):
  distance = {}; nexthop = {}

  for each r in routers:
    distance[r] = INFINITY
    nexthop[r] = None
  distance[dst] = 0

  for _ in range(len(routers)-1):
    for (r1,r2,dist) in links:
      if distance[r1] + dist < distance[r2]:
        distance[r2] = distance[r1] + dist
        nexthop[r2] = r1

  retu
```

From our exercise - **magic number**

From our exercise - **best friend**

Start with infinity as the cost, except for our destination

As we get new offers, compare them to our current cost.

Accept the offer and update our best friend

And we did this asynchronously - there was no strict order amongst you.

Bellman-Ford and our In-Class Routing

```
def bellman_ford (dst, routers, links):
  dist

  for
    di
    ne
  dist

  for _ in range(len(routers)-1):
    for (r1,r2,dist) in links:
      if distance[r1] + dist < distance[r2]:
        distance[r2] = distance[r1] + dist
        nexthop[r2] = r1

  return distance, nexthop
```

And no-one iterated through all the people in the room…

we self-terminated when we *converged*.

From our exercise - **magic number**

m our exercise - **best friend**

Start with infinity as the cost, except for our destination

As we get new offers, compare them to our current cost.

Accept the offer and update our best friend

Building a Distance-Vector Protocol

- The same core approach as Bellman-Ford.
- Thinking about your table…

Your Table	
Dst	***NextHop, Distance***
Sarah	Person in front of me, 14

Building a Distance-Vector Protocol

Your Table	
Dst	***NextHop, Distance***
Sarah	Person in front of me, 14

- Person to your left tells you “I can reach Sarah in 7”.
  - We call this communication **advertising a route** with distance/cost = 7.
- You updated your table…
  - With the cost + 1 (distance to Sarah, plus the distance to your neighbour)
  - If the cost was less than the one in your table.

Building a Distance-Vector Protocol

Your Table	
Dst	***NextHop, Distance***
Sarah	~~Person in front of me, 14~~ Person to my left, 8

- Person to your left tells you "I can reach Ian in 7".
  - We call this communication **advertising a route** with distance/cost = 7.
- You updated your table…
  - With the cost + 1 (distance to Ian, plus the distance to your neighbour)
  - If the cost was less than the one in your table.

Building a Distance-Vector Protocol

Your Table	
Dst	***NextHop, Distance***
Sarah	~~Person in front of me, 14~~ Person to my left, 8

- Person in front tells you, “I can reach Rachel in 3”.
    - Rachel?

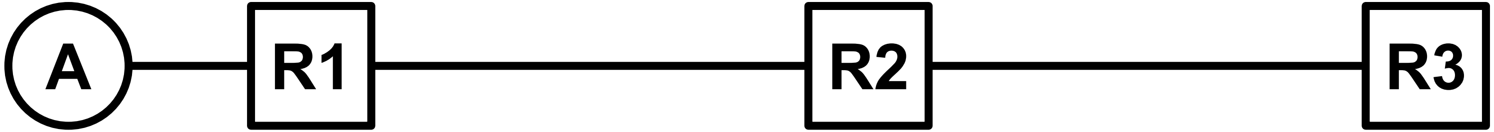
Building a Distance-Vector Protocol

Your Table	
Dst	***NextHop, Distance***
Sarah	~~Person in front of me, 14~~ Person to my left, 8
Rachel	Person in front of me, 4

- Add a new row (for a new destination) Rachel.
- Using the same cost logic.
  - Cost to Rachel plus the distance to your neighbour = 3+1 = 4
- We can keep doing this for all destinations we hear about.

Distance-Vector

Dst	*Nxt,Cost*

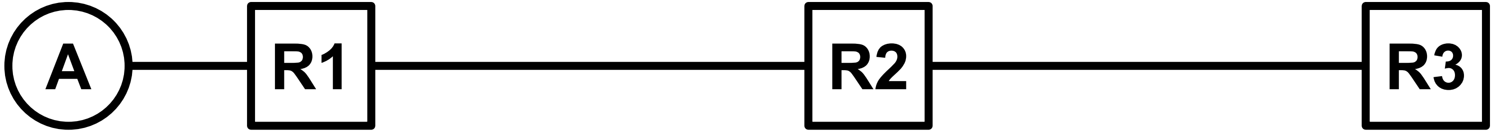


Dst	*Nxt,Cost*

Dst	*Nxt,Cost*

Distance-Vector

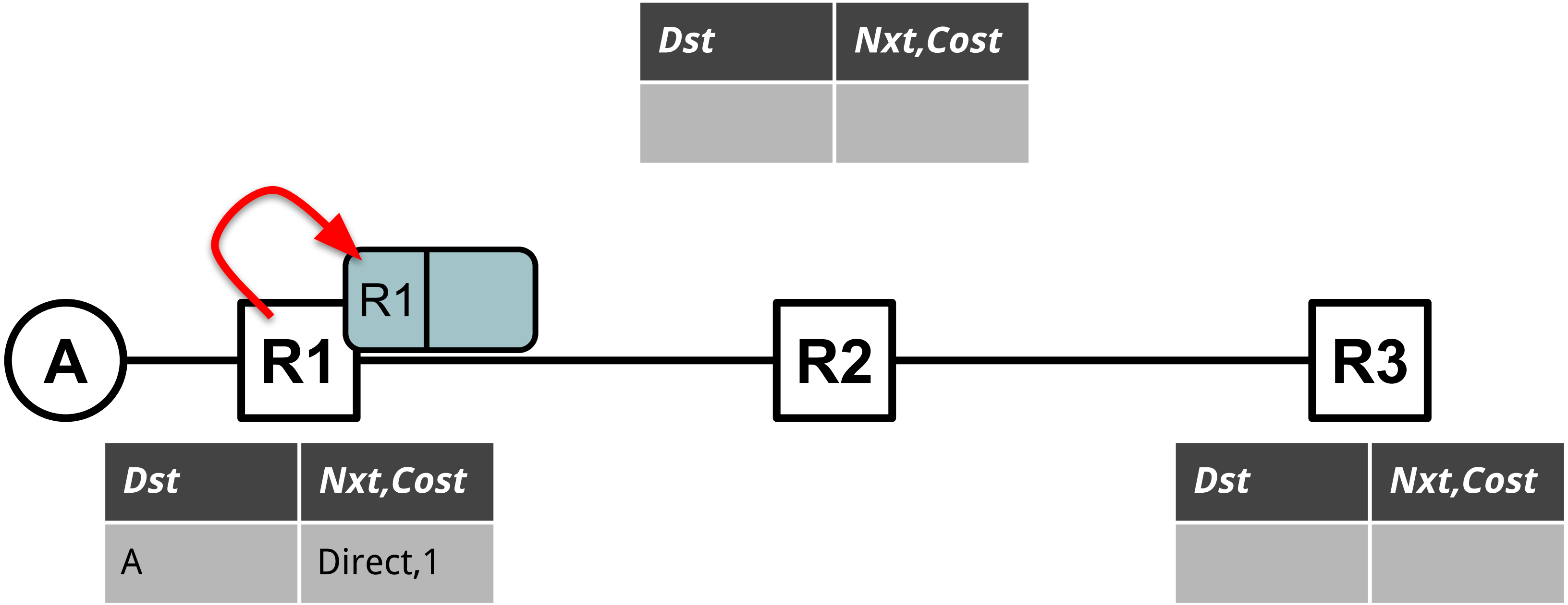
Dst	*Nxt,Cost*



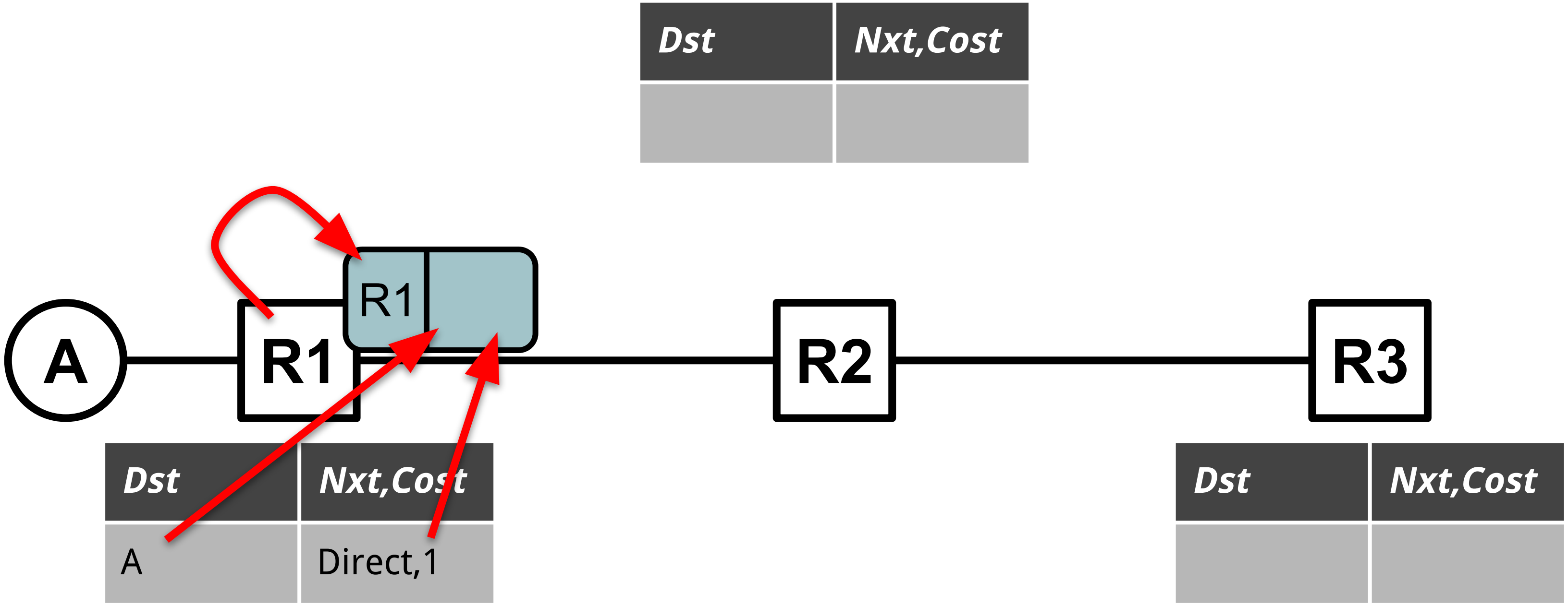
Dst	*Nxt,Cost*
A	Direct,1

Dst	*Nxt,Cost*

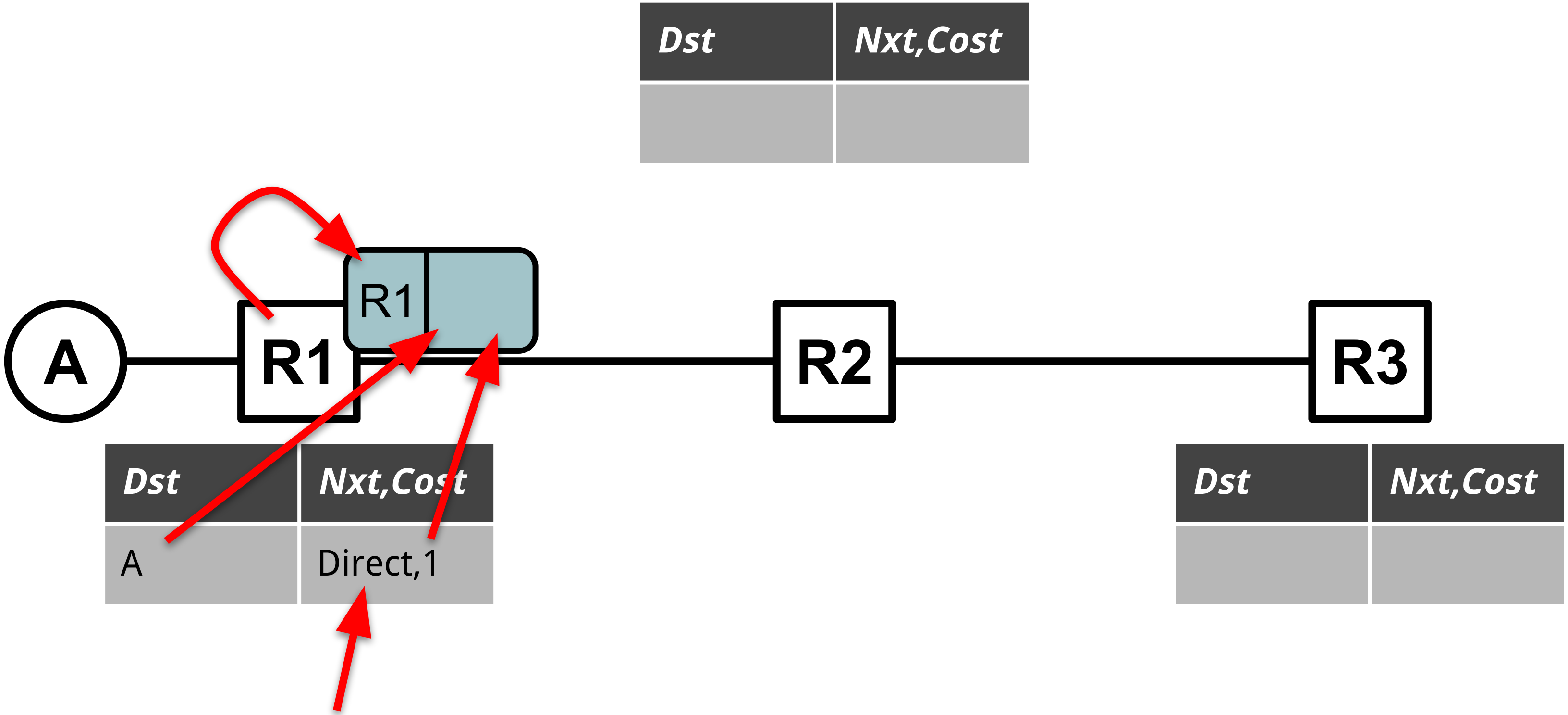
Distance-Vector



Distance-Vector

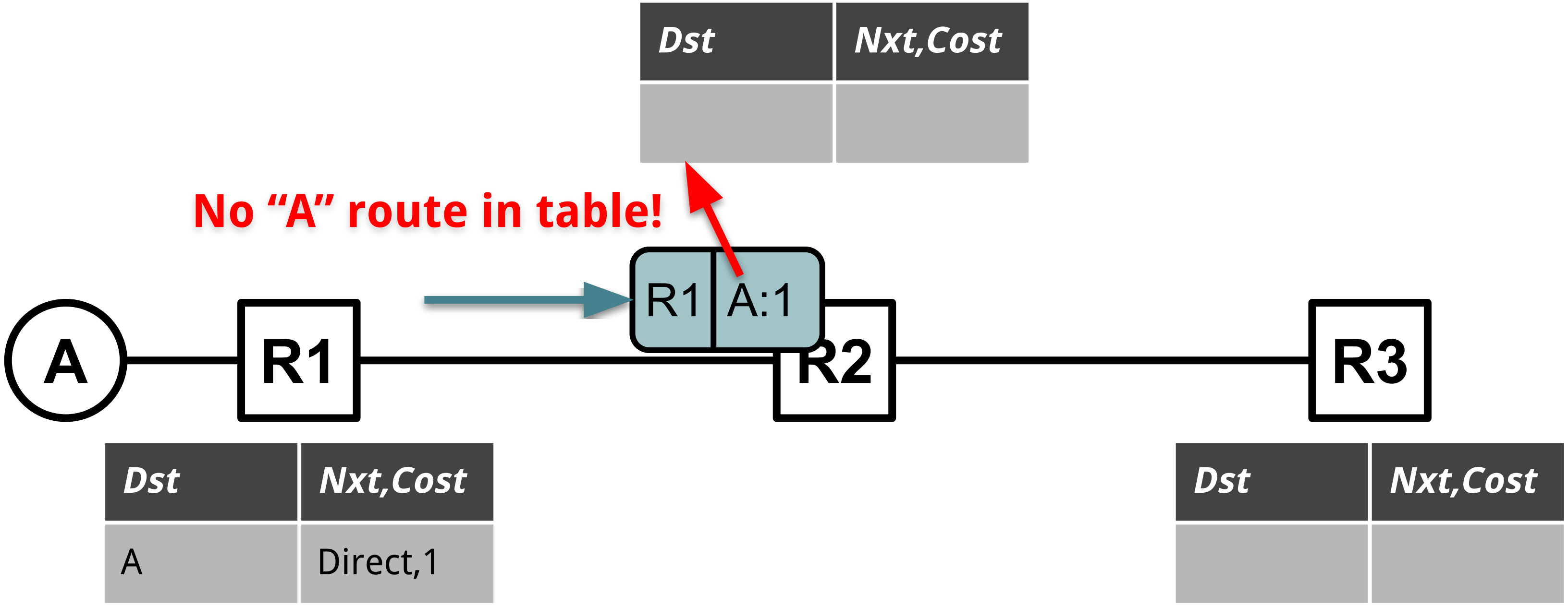


Distance-Vector



Only R1 needs to know its own next hop!

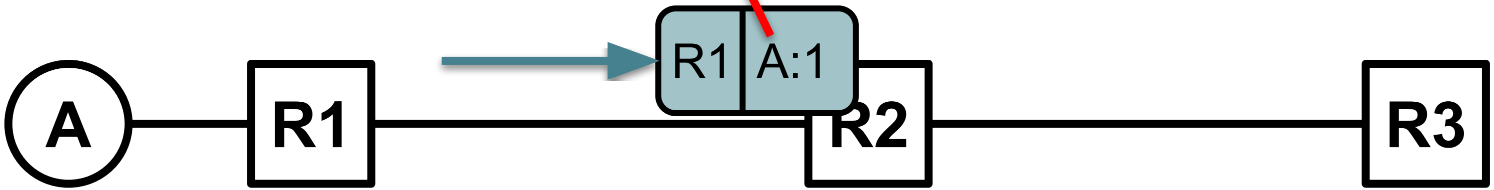
Distance-Vector



Distance-Vector

Dst	Nxt,Cost
A	

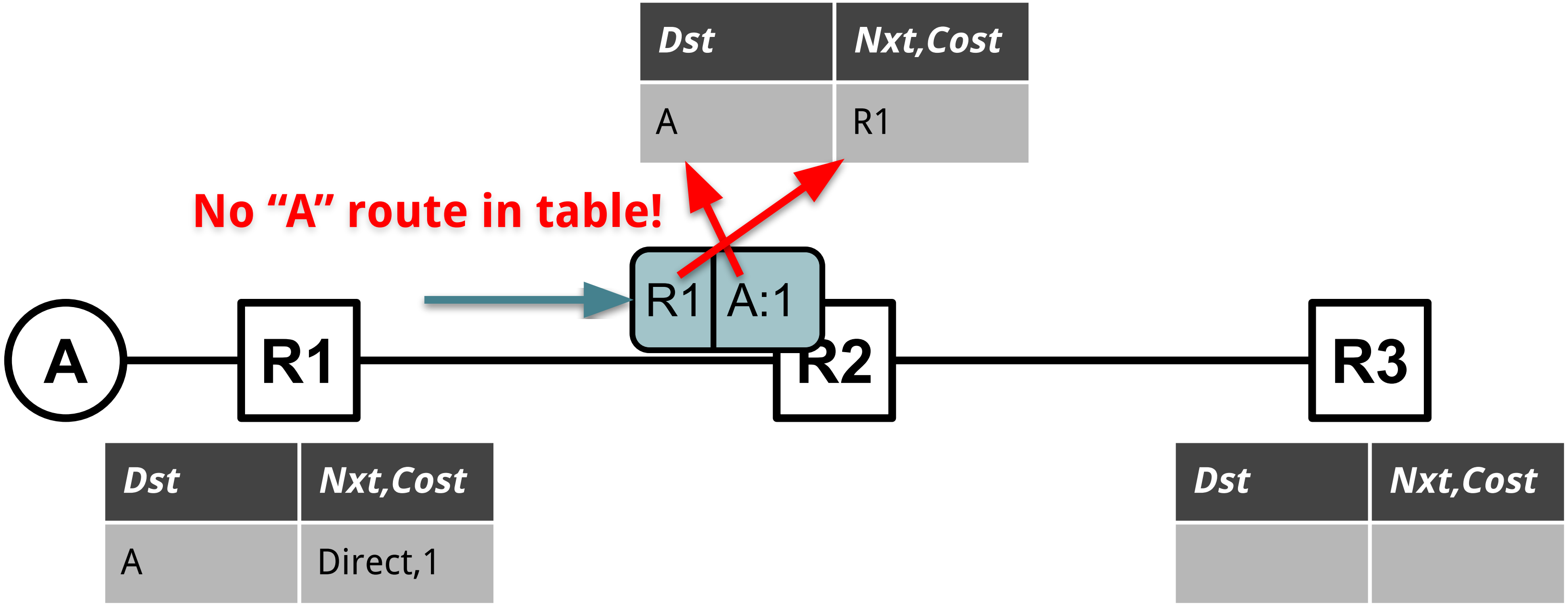
No “A” route in table!



Dst	Nxt,Cost
A	Direct,1

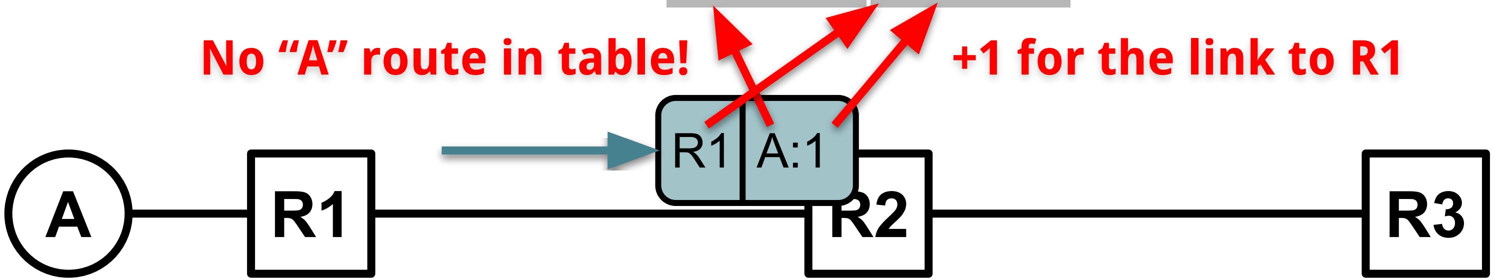
Dst	Nxt,Cost

Distance-Vector



Distance-Vector

Dst	Nxt,Cost
A	R1,2

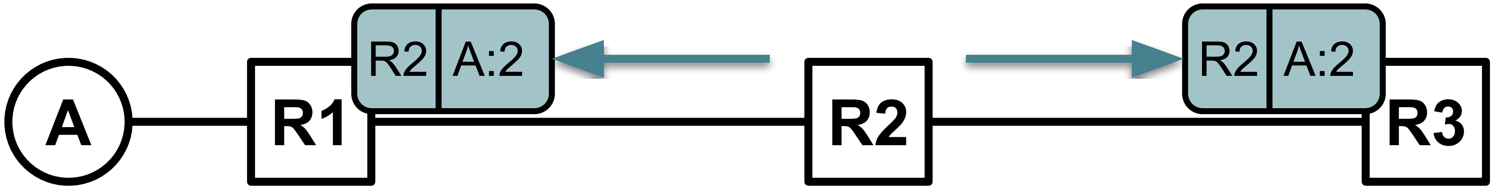


Dst	Nxt,Cost
A	Direct,1

Dst	Nxt,Cost

Distance-Vector

Dst	*Nxt,Cost*
A	R1,2

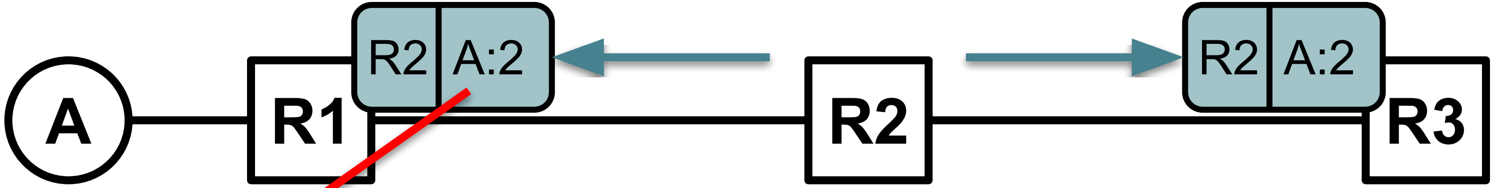


Dst	*Nxt,Cost*
A	Direct,1

Dst	*Nxt,Cost*

Distance-Vector

Dst	Nxt,Cost
A	R1,2



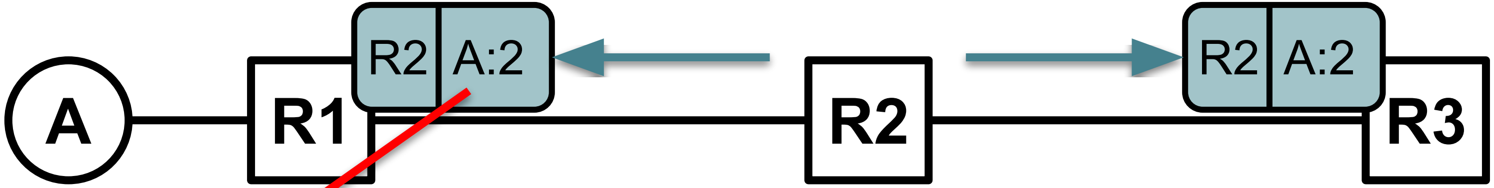
Dst	Nxt,Cost
A	Direct,1

2+1 = 3

Dst	Nxt,Cost

Distance-Vector

Dst	Nxt,Cost
A	R1,2



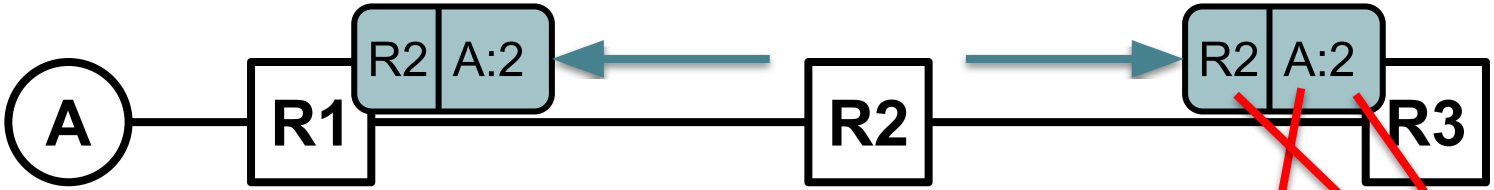
Dst	Nxt,Cost
A	Direct,1

2+1 = 3
Worse than current route
⇒ do not adjust.

Dst	Nxt,Cost

Distance-Vector

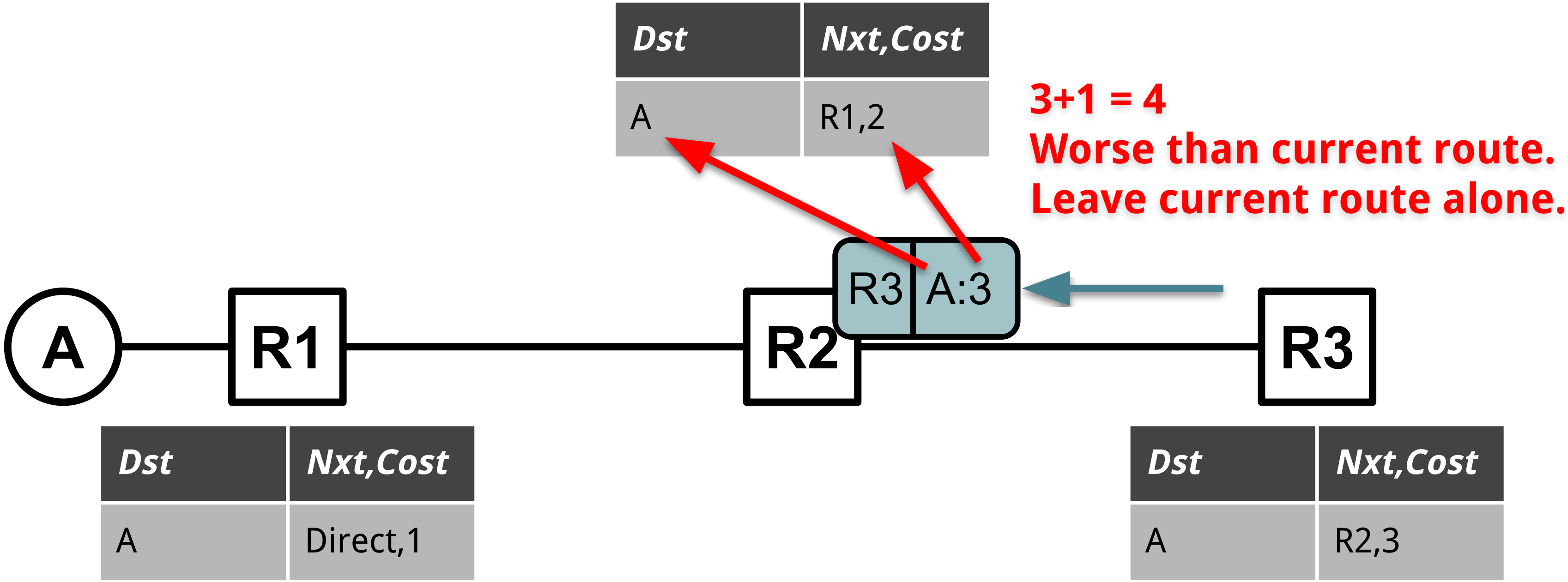
Dst	Nxt,Cost
A	R1,2



Dst	Nxt,Cost
A	Direct,1

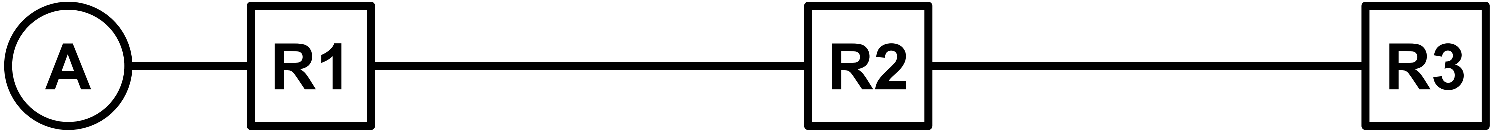
Dst	Nxt,Cost
A	R2,3

Distance-Vector



Distance-Vector

Dst	*Nxt,Cost*
A	R1,2



Dst	*Nxt,Cost*
A	Direct,1

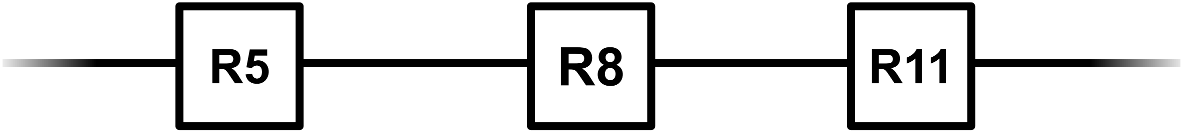
Dst	*Nxt,Cost*
A	R2,3

We've converged! 🥳

Questions?

D-V: An exception to the update rule

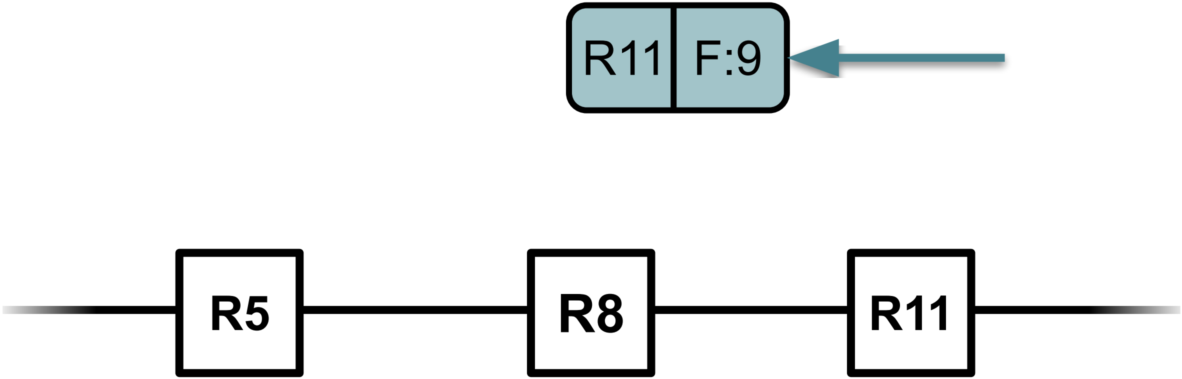
- Our logic for when to update a route:
  - If destination not in table -- add to table



Dst	*Nxt,Cost*
F	R5,6

D-V: An exception to the update rule

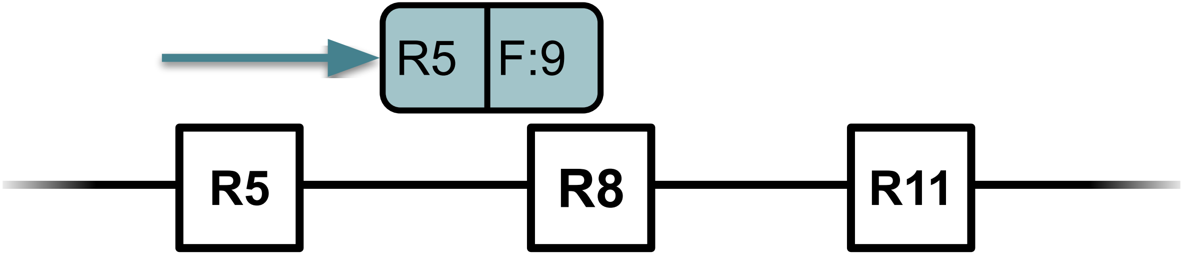
- Our logic for when to update a route:
  - If destination not in table -- add to table
  - If current_route_distance > advertised_distance + distance_to_neighbor -- replace current



Dst	*Nxt,Cost*
F	R5,6

D-V: An exception to the update rule

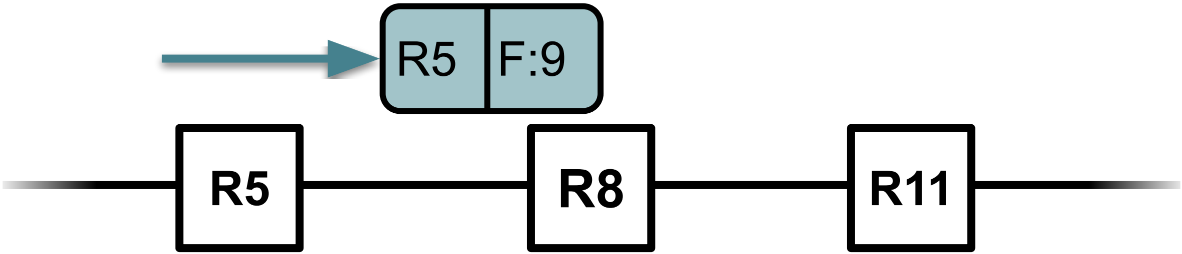
- Our logic for when to update a route:
  - If destination not in table -- add to table
  - If current_route_distance > advertised_distance + distance_to_neighbor -- replace current



Dst	***Nxt,Cost***
F	R5,6

D-V: An exception to the update rule

- Our logic for when to update a route:
  - If destination not in table -- add to table
  - If current_route_distance > advertised_distance + distance_to_neighbor -- replace current
  - **If advertiser is current_next_hop -- replace current**



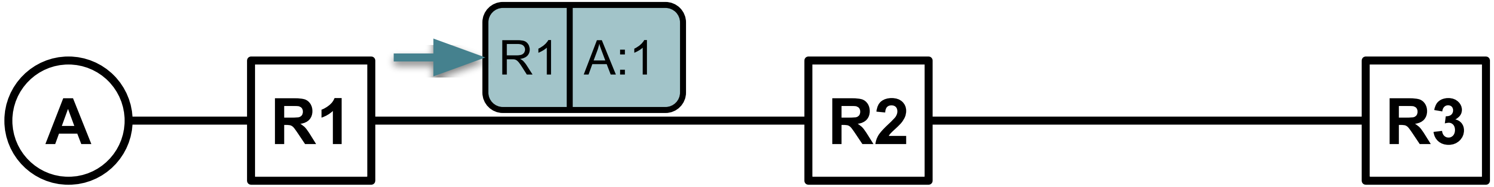
Dst	*Nxt,Cost*
F	R5,**10**

D-V:

Is our D-V protocol reliable?

D-V: Reliability

Dst	*Nxt,Cost*



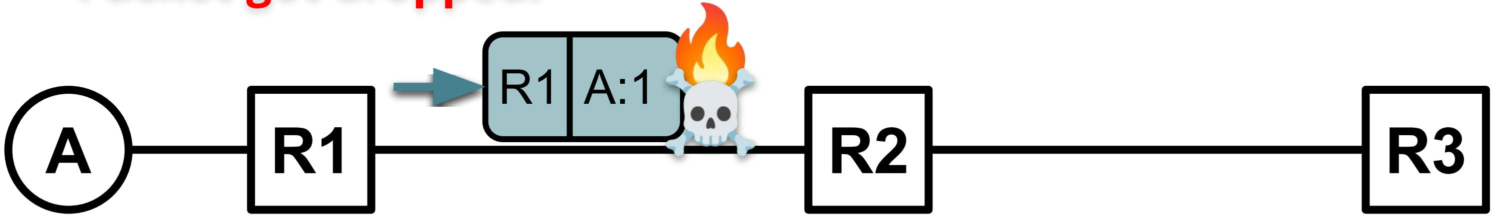
Dst	*Nxt,Cost*
A	Direct,1

Dst	*Nxt,Cost*

D-V: Reliability

Dst	Nxt,Cost

Something bad happened!
Packet got dropped!



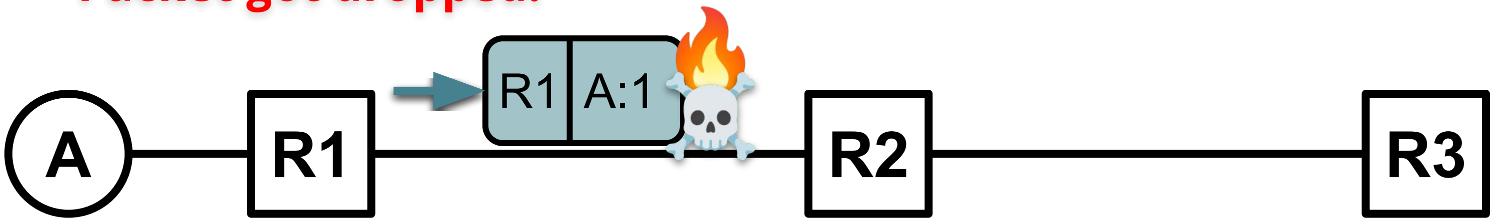
Dst	Nxt,Cost
A	Direct,1

Dst	Nxt,Cost

D-V: Reliability

Dst	Nxt,Cost

Something bad happened!
Packet got dropped!



Dst	Nxt,Cost
A	Direct,1

<u>Super simple reliability</u>

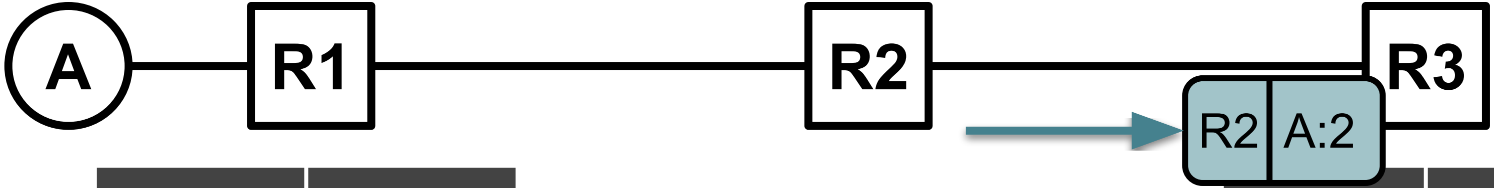
Resend advertisements every X seconds. (X=*advertisement interval*)

This should always work *eventually* (assuming link works at all).
Sending on change (*triggered updates*) acts as an *optimisation*.

Questions?

D-V: Split Horizon

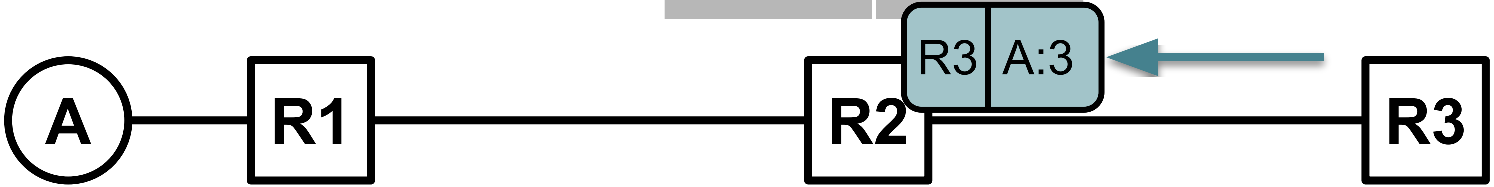
Dst	Nxt,Cost
A	R1,2



Dst	Nxt,Cost
A	Direct,1

Dst	Nxt,Cost
A	R2,3

D-V: Split Horizon

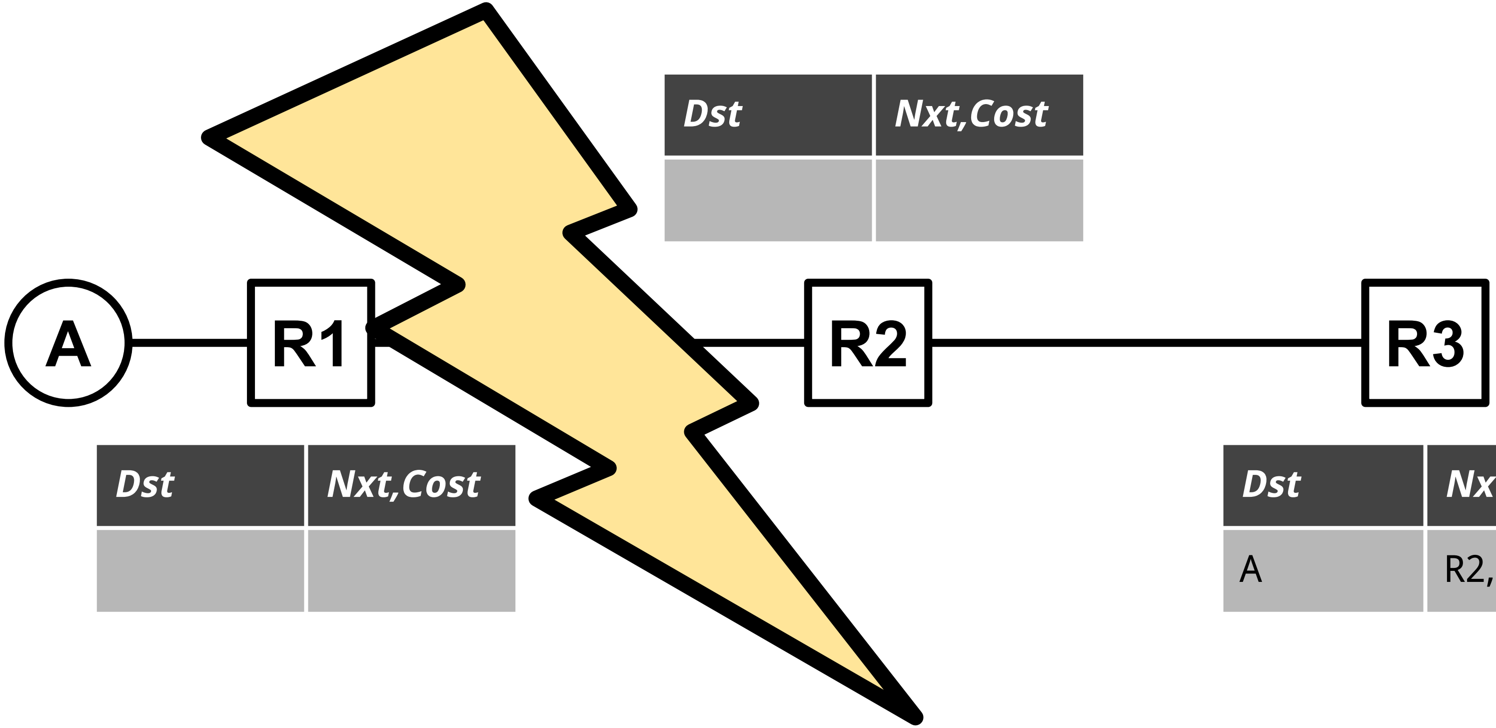


Dst	Nxt,Cost
A	R1,2

Dst	Nxt,Cost
A	Direct,1

Dst	Nxt,Cost
A	R2,3

D-V: Split Horizon



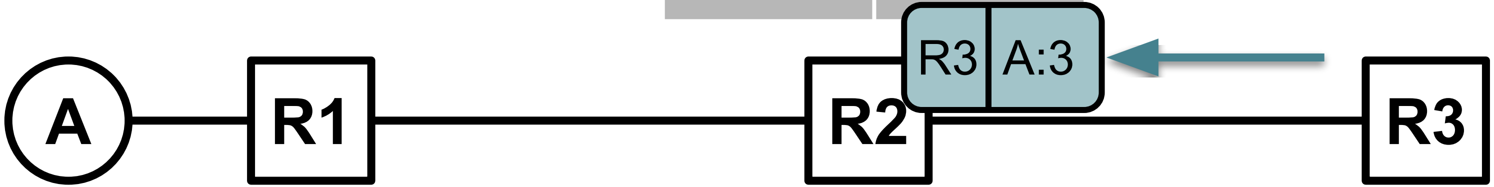
Dst	Nxt,Cost

Dst	Nxt,Cost

Dst	Nxt,Cost
A	R2,3

D-V: Split Horizon

Dst	Nxt,Cost
A	R3, 4



Dst	Nxt,Cost

Dst	Nxt,Cost
A	R2,3

D-V: Split Horizon

Dst	Nxt,Cost
A	R3,4

Dst	Nxt,Cost
A	R2,5

Dst	Nxt,Cost
A	R2,3

Huh?! A is local to R1?!

D-V: Split Horizon

- What is the advantage in advertising a path back to the person who sent it you?

- Telling them about your entry *via them*:
  - Doesn’t tell them anything new.
  - Misleads them into thinking you have an independent path.

D-V: Split Horizon

- What is the advantage in advertising a path back to the person who sent it you?

- Telling them about your entry *via them*:
  - Doesn't tell them anything new.
  - Misleads them into thinking you have an independent path.

- Solution:
  - If you are using a next-hop's path for some destination – don't advertise it to them.
  - Referred to as **Split Horizon**

Questions?

D-V: Counting to Infinity

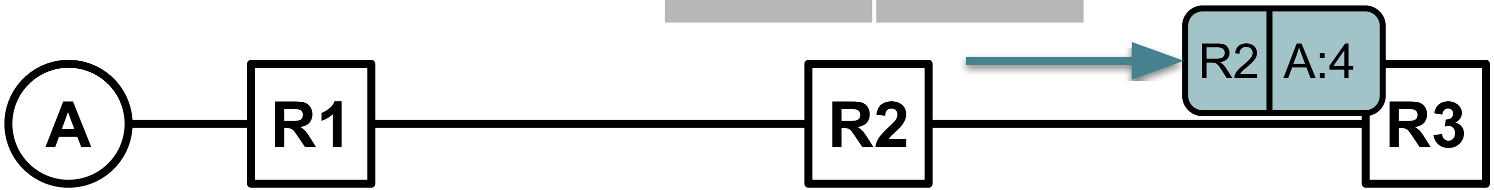
Dst	Nxt,Cost
A	R3,4

Dst	Nxt,Cost
A	R2,5

Dst	Nxt,Cost
A	R2,3

D-V: Counting to Infinity

Dst	Nxt,Cost
A	R3,4



Dst	Nxt,Cost
A	R2,5

Dst	Nxt,Cost
A	R2,~~3~~ 5

D-V: Counting to Infinity

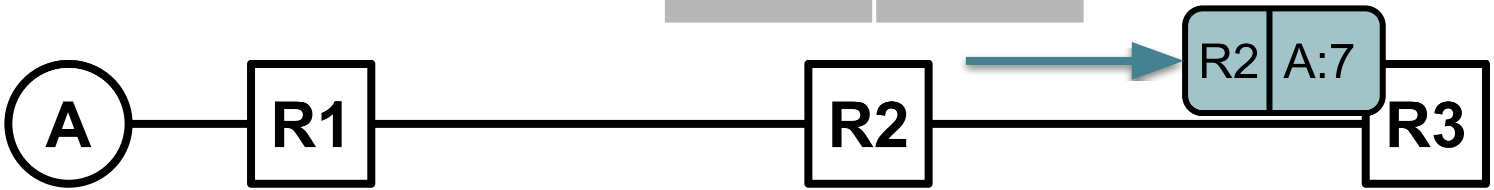
Dst	Nxt,Cost
A	R3,~~4~~ 6

Dst	Nxt,Cost
A	R2,5

Dst	Nxt,Cost
A	R2,5

D-V: Counting to Infinity

Dst	Nxt,Cost
A	R3,6



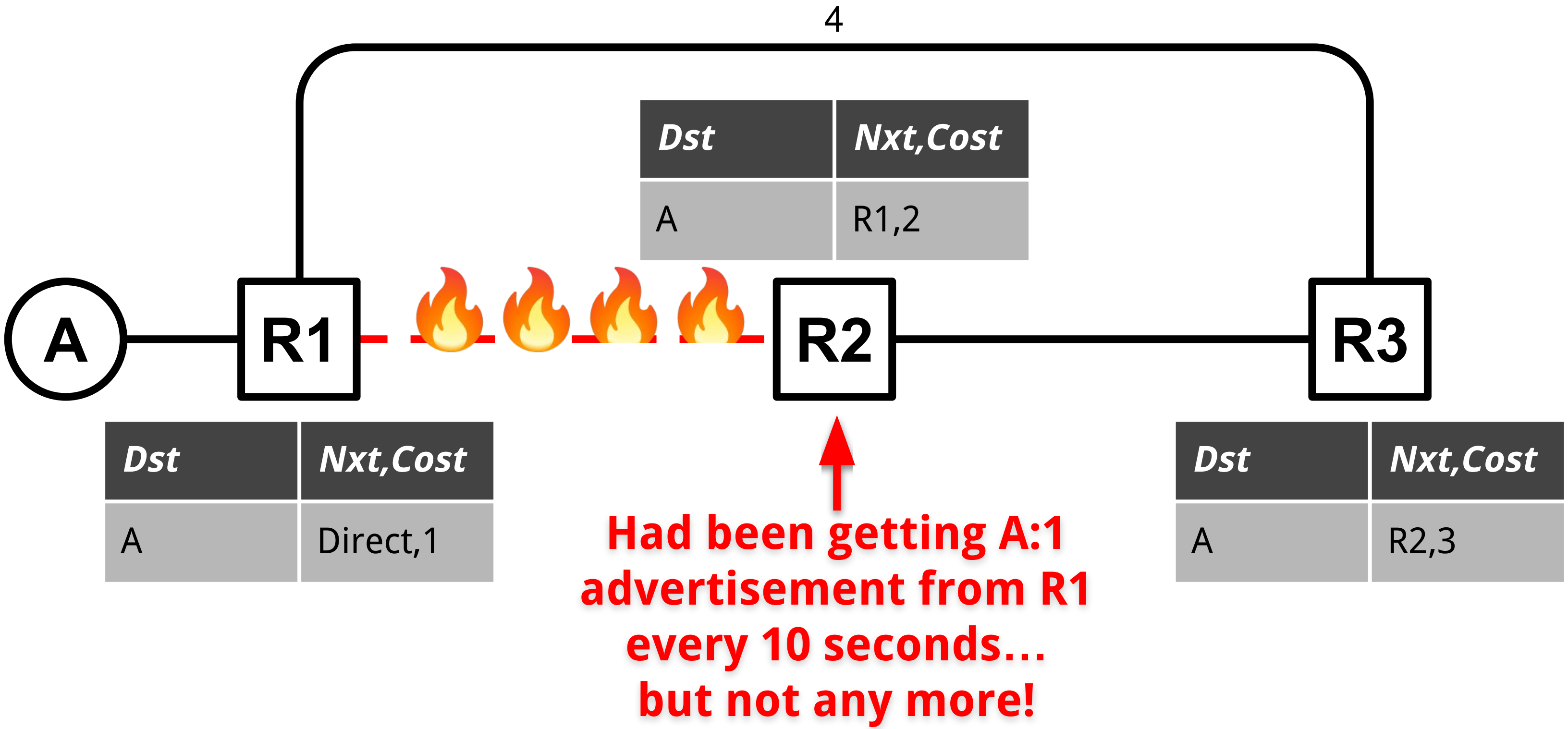
Dst	Nxt,Cost
A	R2,5

Dst	Nxt,Cost
A	R2,~~5~~ 7

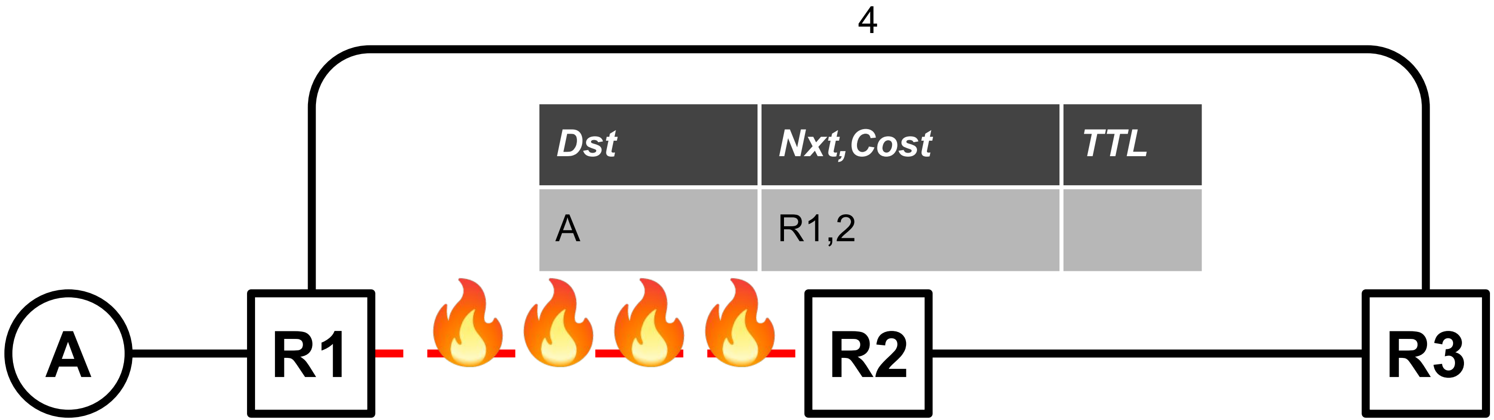
Route costs on R2/R3 *count to infinity*!

Solution: Pick a maximum value (e.g., 16) and stop there.

D-V: Failures


4
Dst
Nxt,Cost
A
R1,2
A
R1
R2
R3
Dst
Nxt,Cost
A
Direct,1
Dst
Nxt,Cost
A
R2,3
Had been getting A:1
advertisement from R1
every 10 seconds…
but not any more!

D-V: Failures



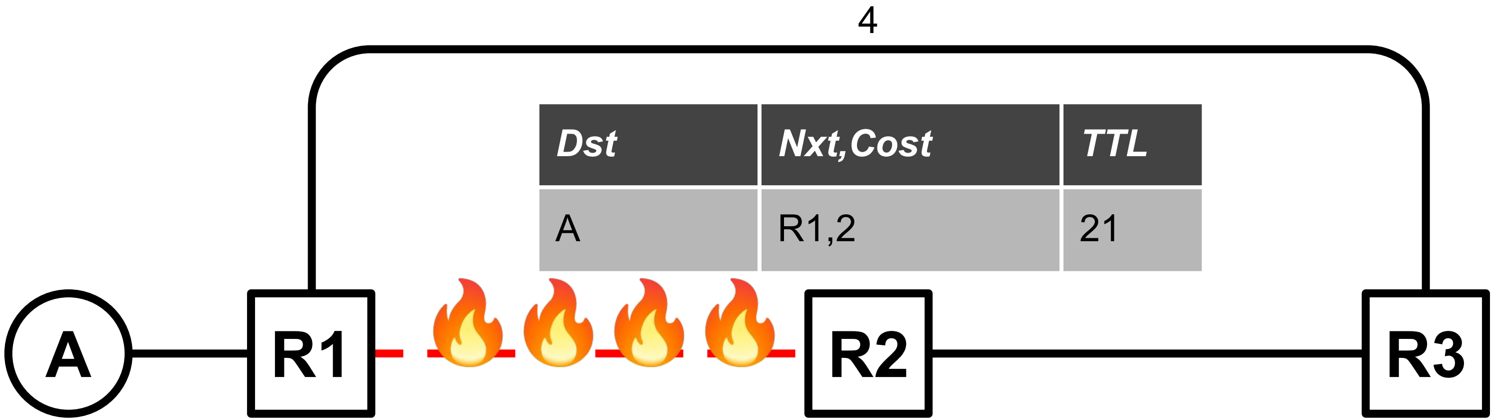
Dst	Nxt,Cost	TTL
A	R1,2	

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	

Each route only has a finite *Time To Live* (e.g., 21 seconds).
Gets "recharged" by the periodic advertisements.
If you don't get a periodic update (e.g., 10 seconds)… expire & remove route.

D-V: Failures



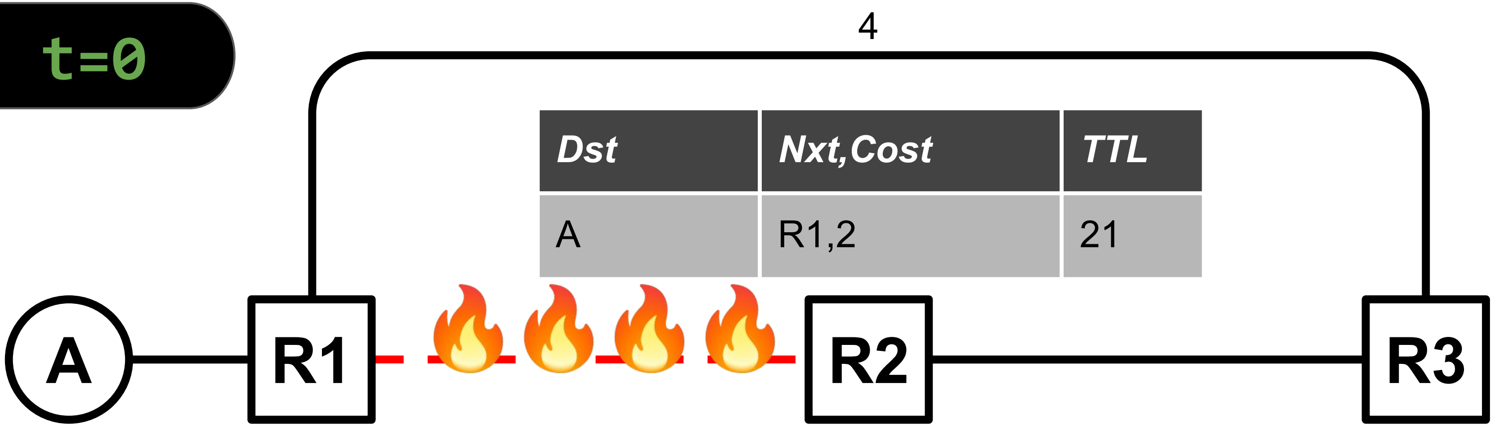
Dst	Nxt,Cost	TTL
A	R1,2	21

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	21

D-V: Failures

t=0



Dst	Nxt,Cost	TTL
A	R1,2	21

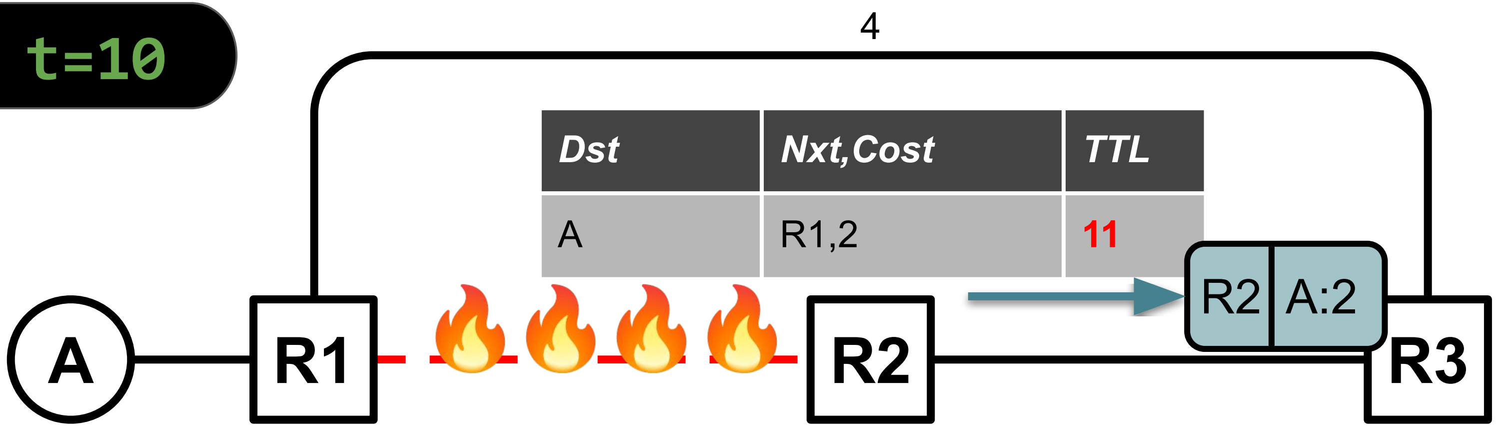
Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	21

D-V: Failures

t=10

Dst	Nxt,Cost	TTL
A	R1,2	11

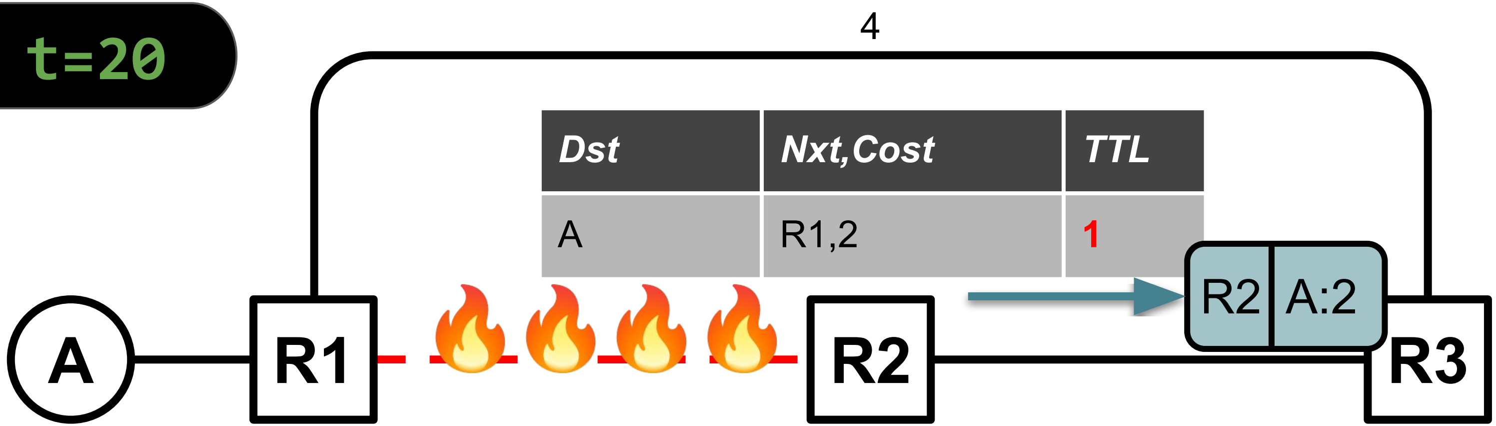


Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	21

D-V: Failures

t=20



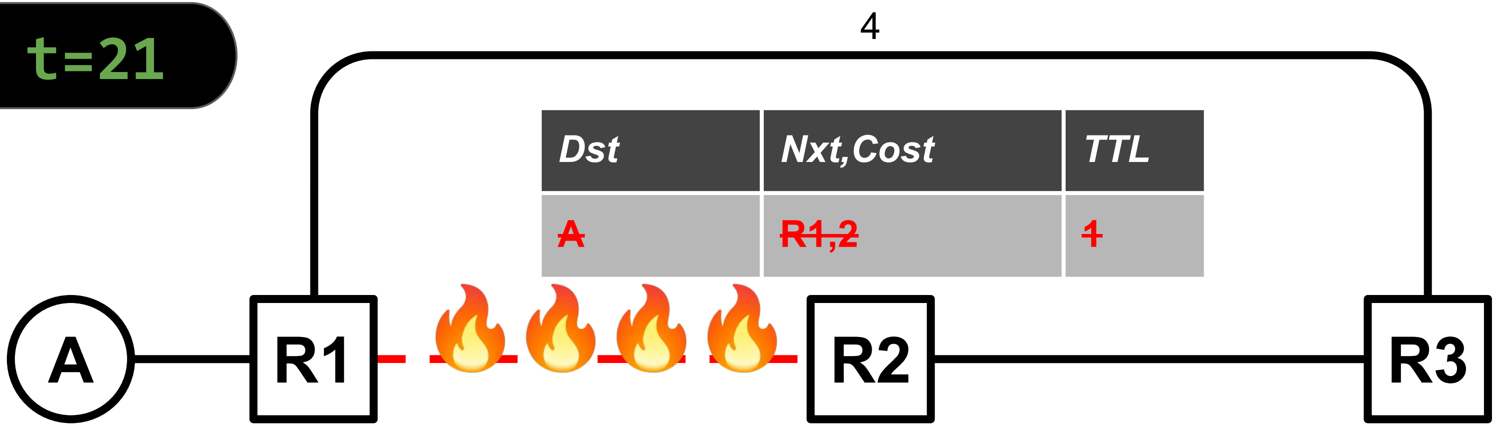
Dst	Nxt,Cost	TTL
A	R1,2	1

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	21

D-V: Failures

t=21



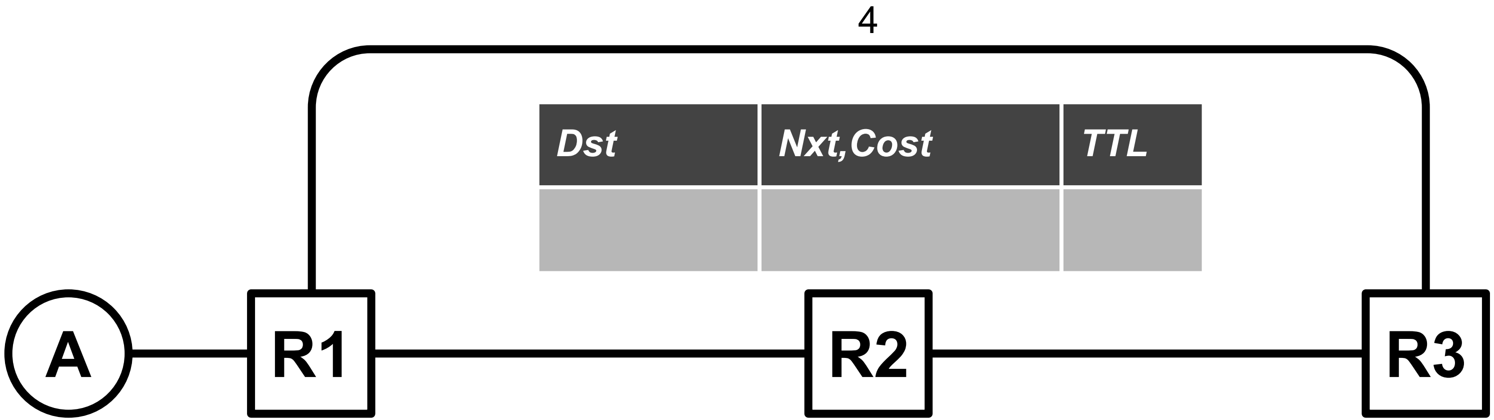
Dst	Nxt,Cost	TTL
~~A~~	~~R1,2~~	~~1~~

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	21

How do we deal with changing topology?
Link failures.

D-V: Failures



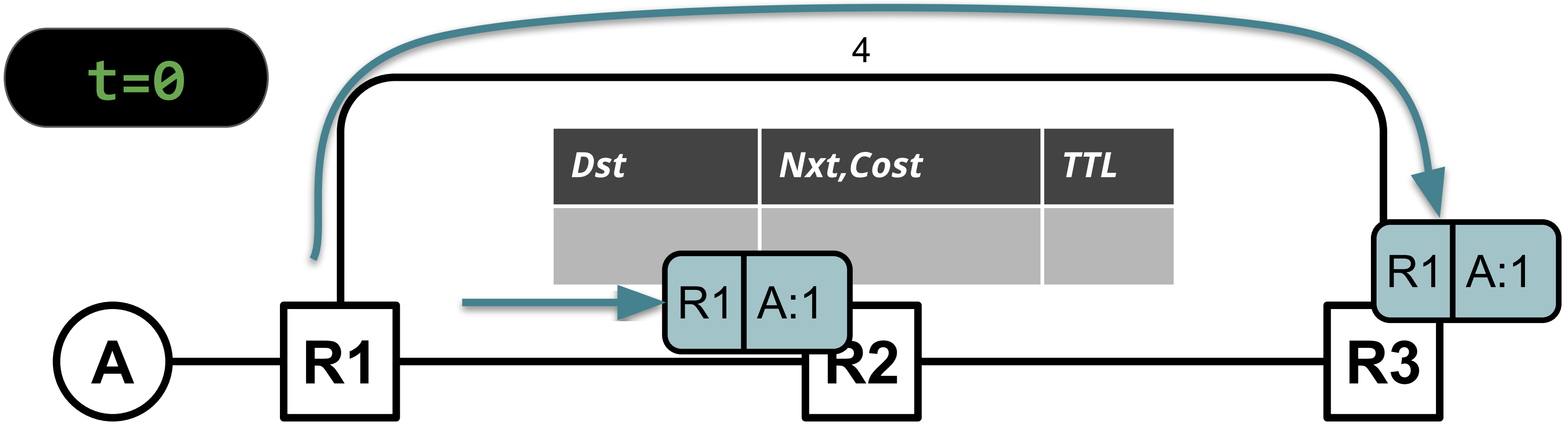
Dst	Nxt,Cost	TTL

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL

Static and connected routes don't expire

D-V: Failures



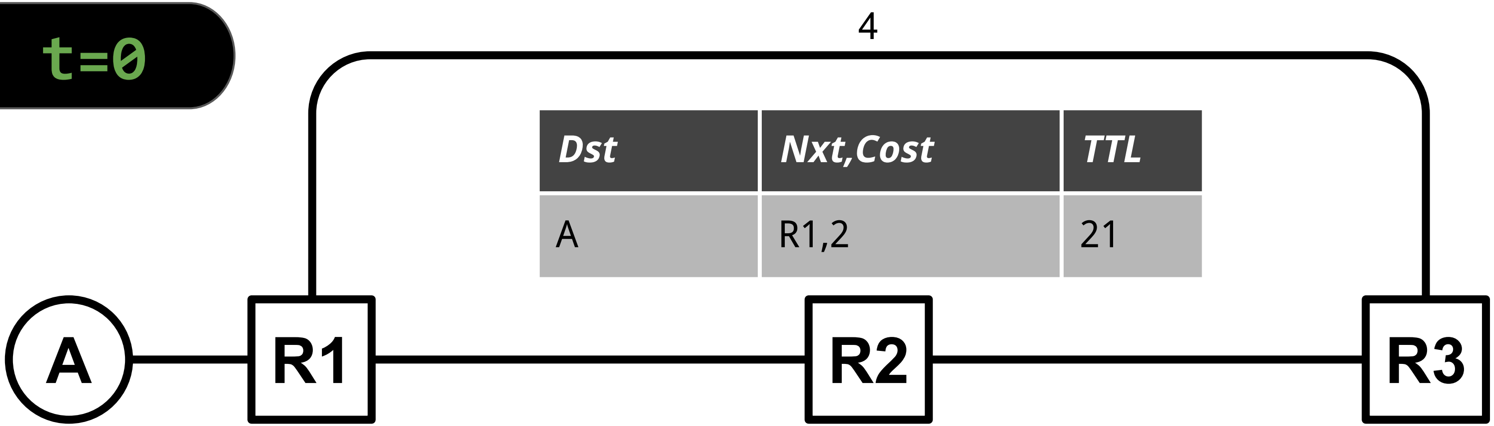
Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL

Dst	Nxt,Cost	TTL

D-V: Failures

t=0



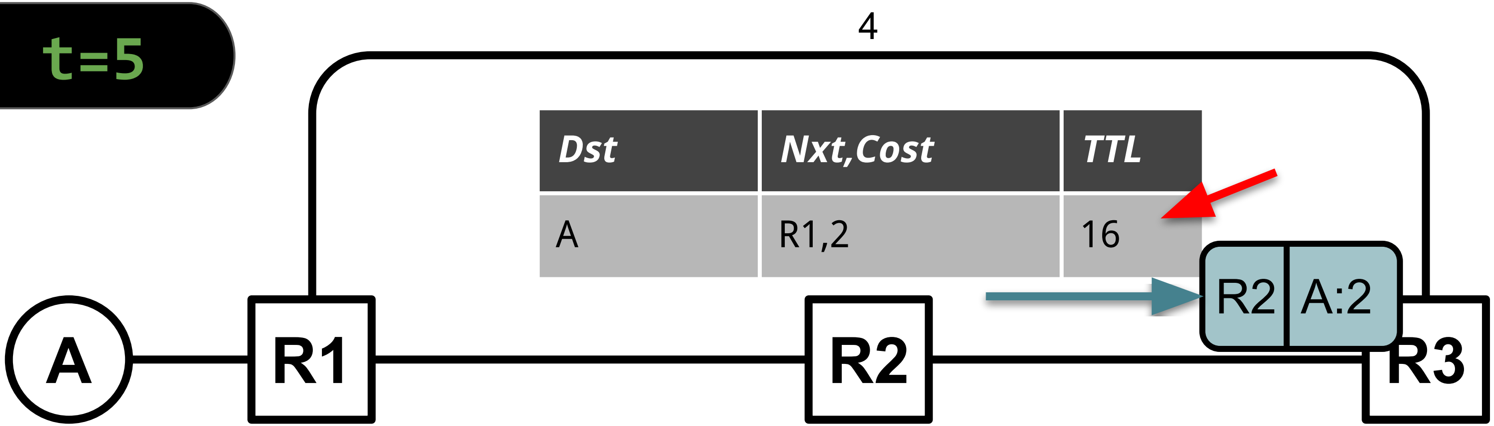
Dst	*Nxt,Cost*	*TTL*
A	R1,2	21

Dst	*Nxt,Cost*	*TTL*
A	Direct,1	---

Dst	*Nxt,Cost*	*TTL*
A	R1,5	21

D-V: Failures

t=5



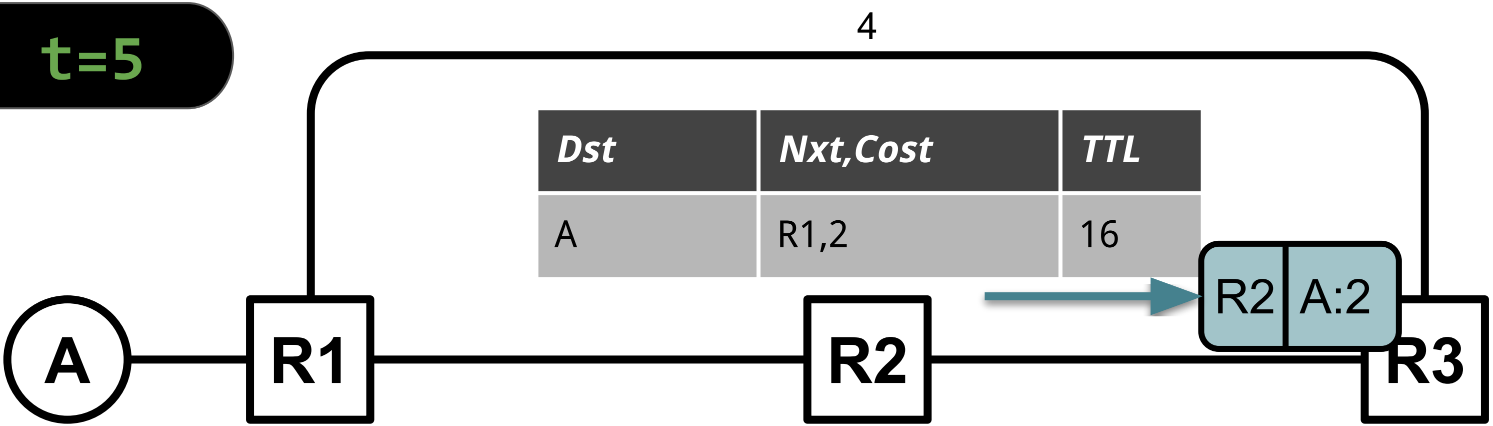
Dst	Nxt,Cost	TTL
A	R1,2	16

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R1,5	16

D-V: Failures

t=5

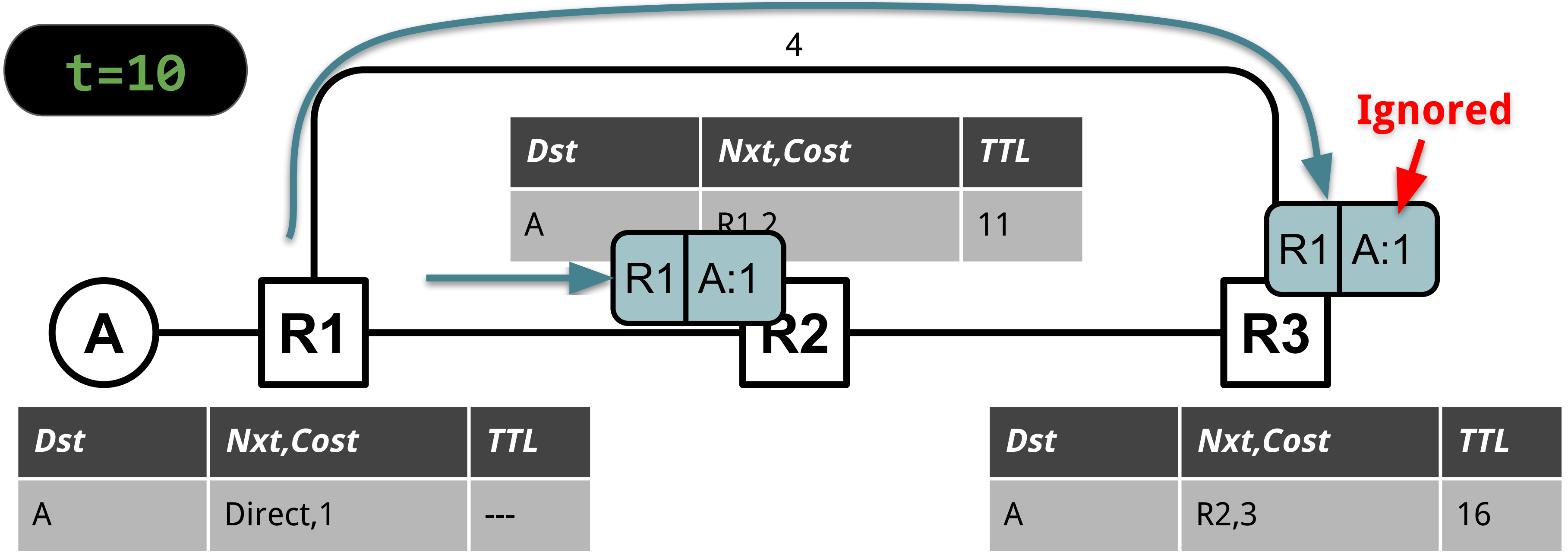


Dst	Nxt,Cost	TTL
A	R1,2	16

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	~~R1,5~~ R2,3	~~16~~ 21

D-V: Failures

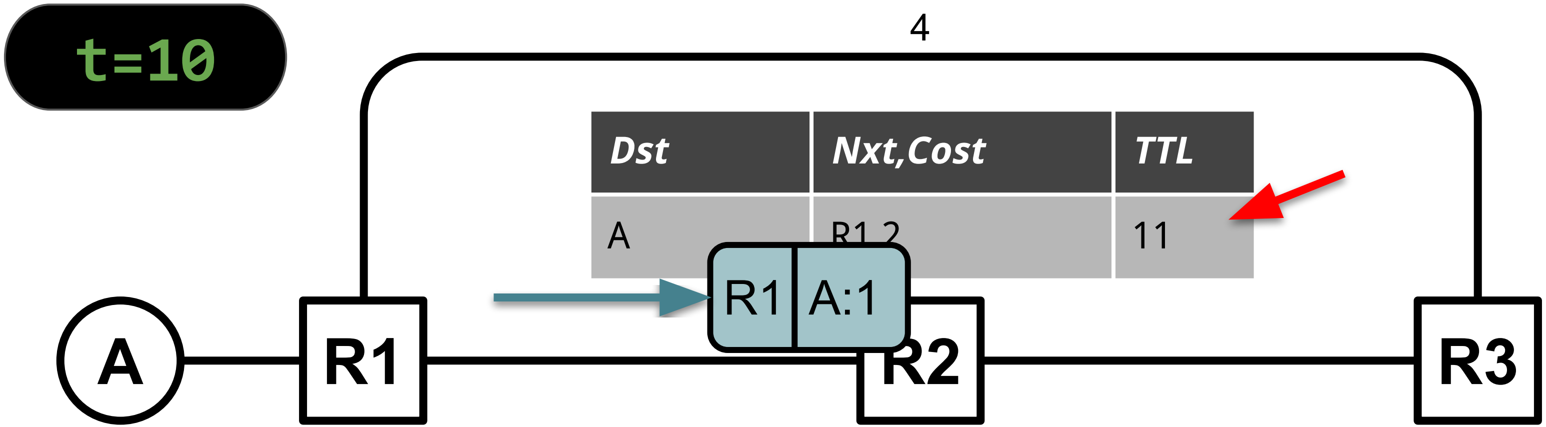


Dst	Nxt,Cost	TTL
A	R1,2	11

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	16

D-V: Failures



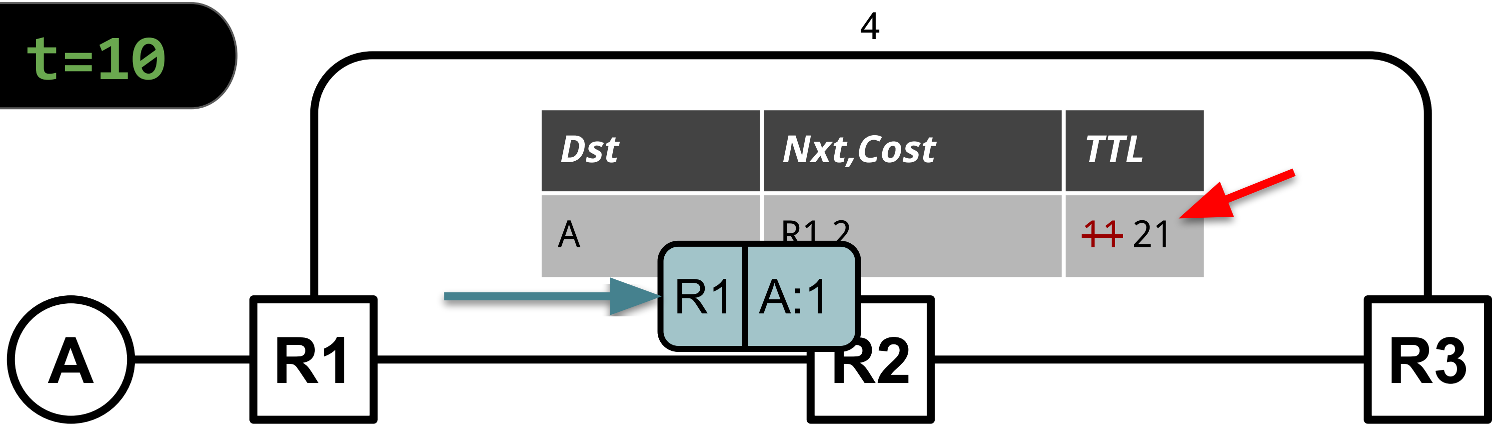
Dst	Nxt,Cost	TTL
A	R1,2	11

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	16

D-V: Failures

t=10



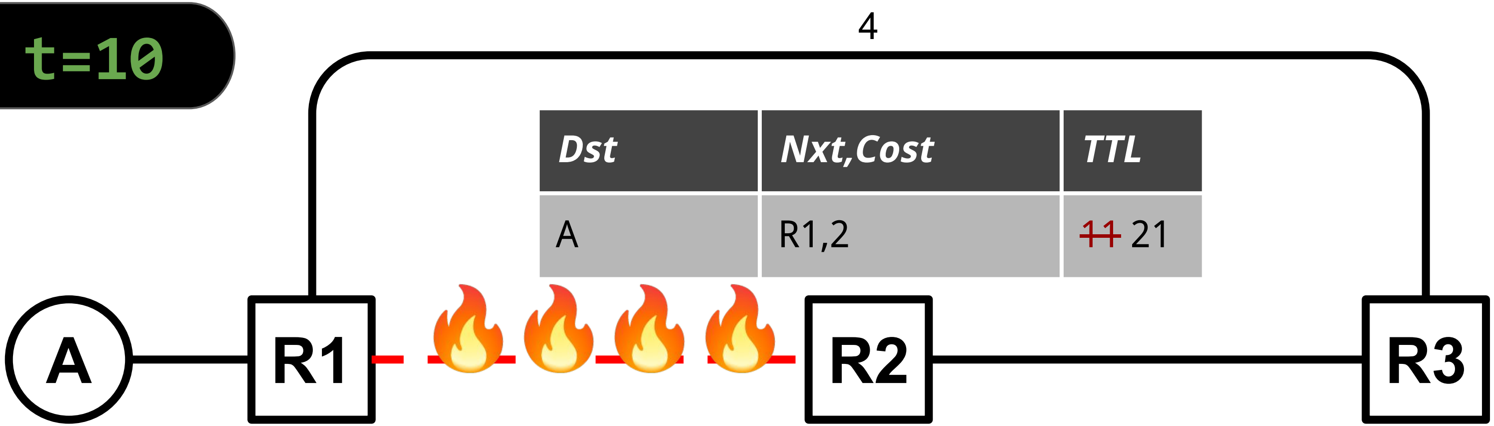
Dst	Nxt,Cost	TTL
A	R1,2	~~11~~ 21

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	16

D-V: Failures

t=10



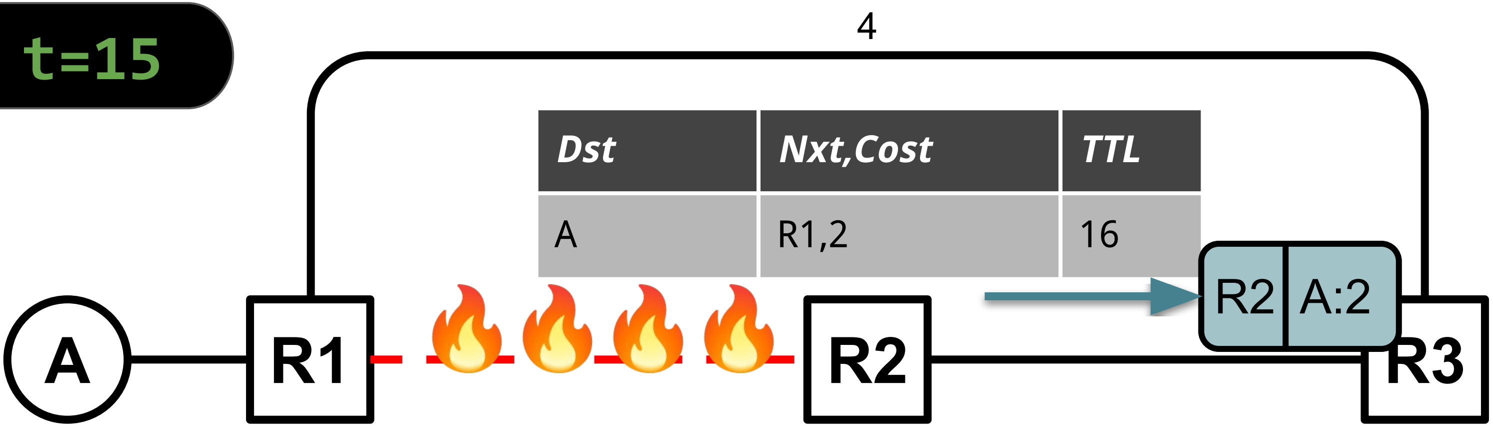
Dst	Nxt,Cost	TTL
A	R1,2	~~11~~ 21

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	16

D-V: Failures

t=15

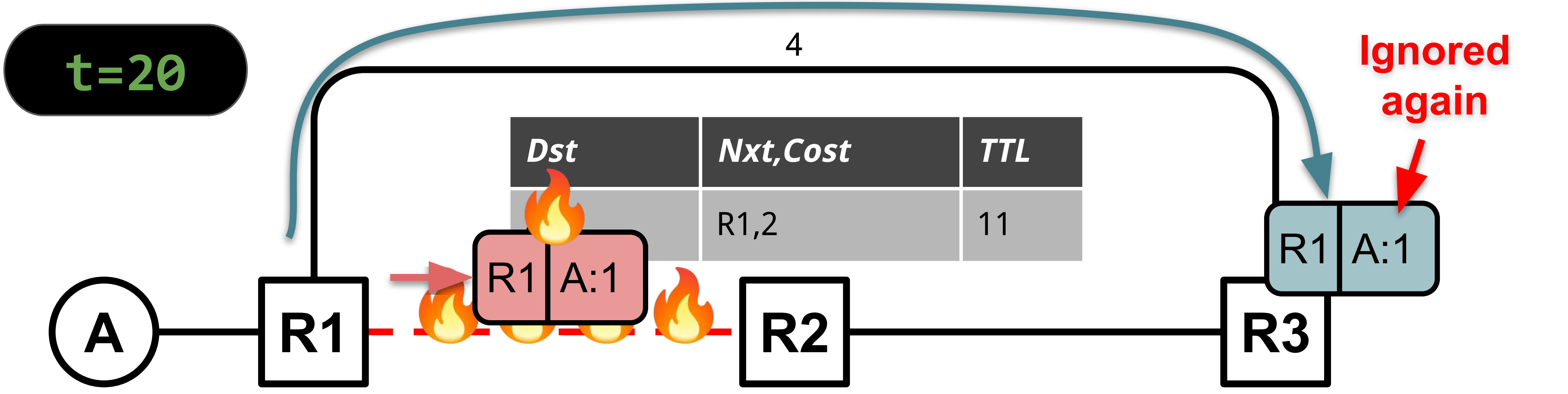


Dst	Nxt,Cost	TTL
A	R1,2	16

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	~~11~~ 21

D-V: Failures



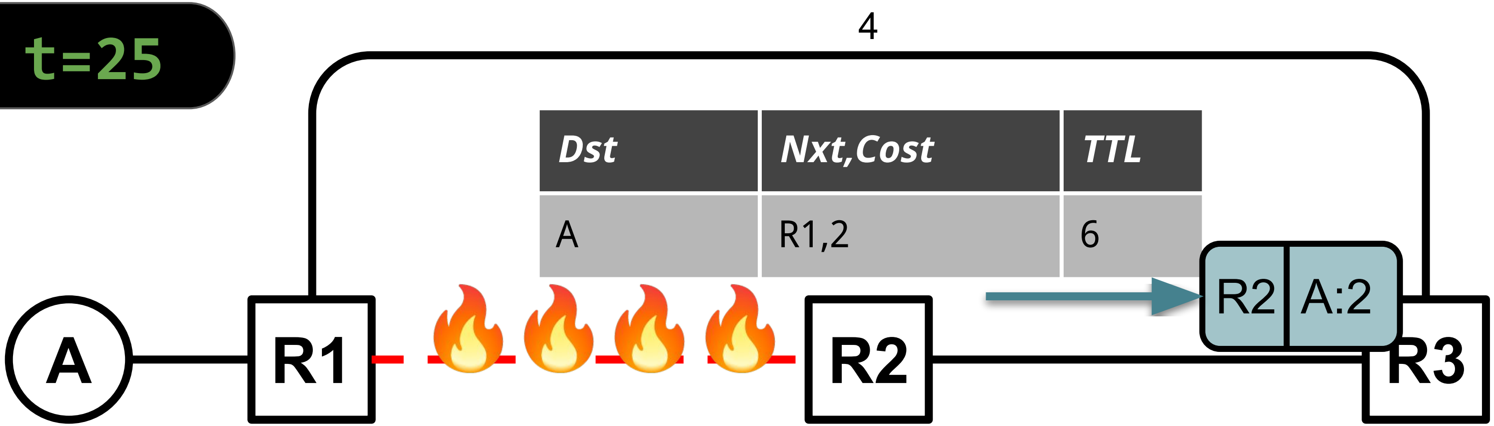
Dst	Nxt,Cost	TTL
	R1,2	11

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	16

D-V: Failures

t=25

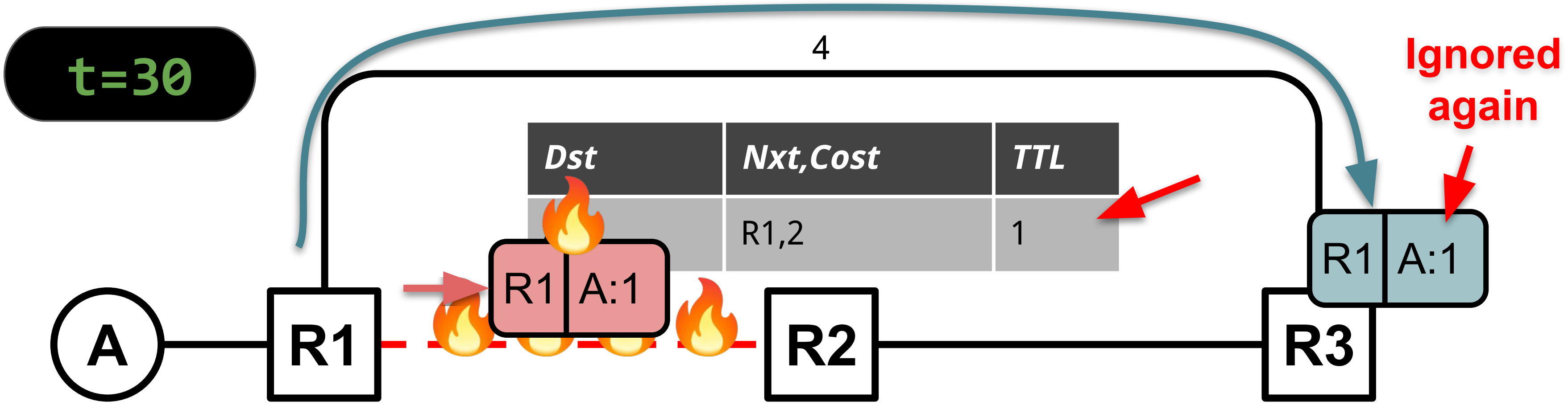


Dst	Nxt,Cost	TTL
A	R1,2	6

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	~~16~~ 21

D-V: Failures



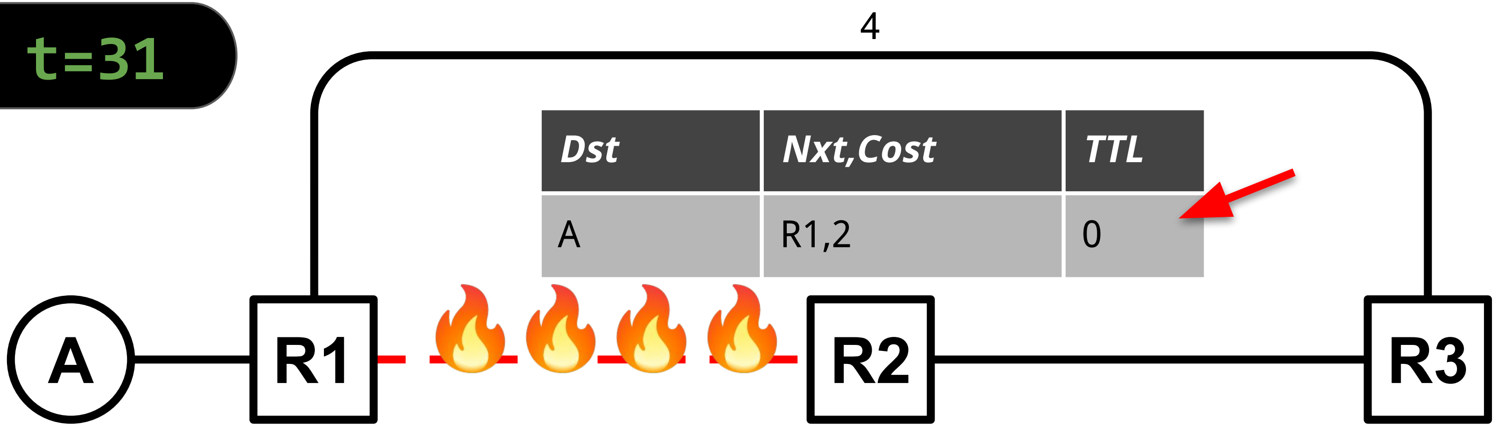
Dst	Nxt,Cost	TTL
	R1,2	1

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	16

D-V: Failures

t=31



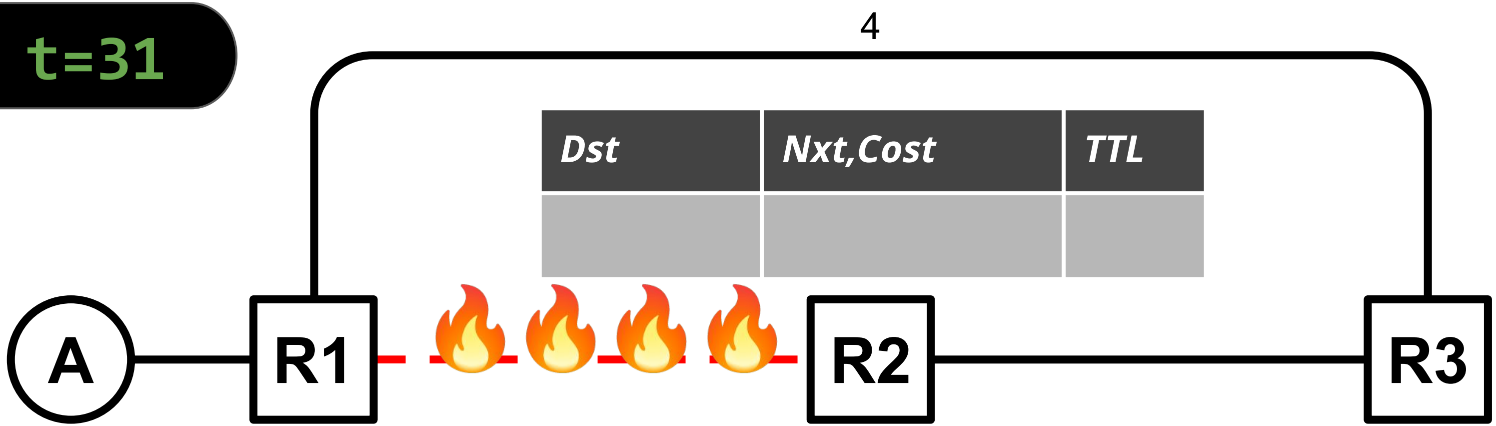
Dst	Nxt,Cost	TTL
A	R1,2	0

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	15

D-V: Failures

t=31



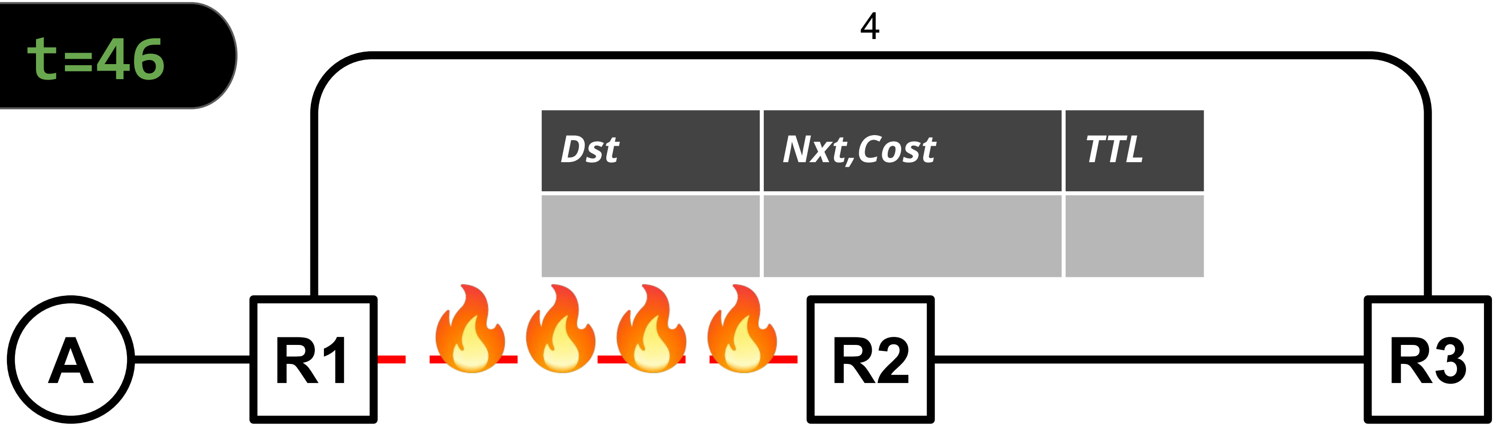
Dst	Nxt,Cost	TTL

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	15

D-V: Failures

t=46

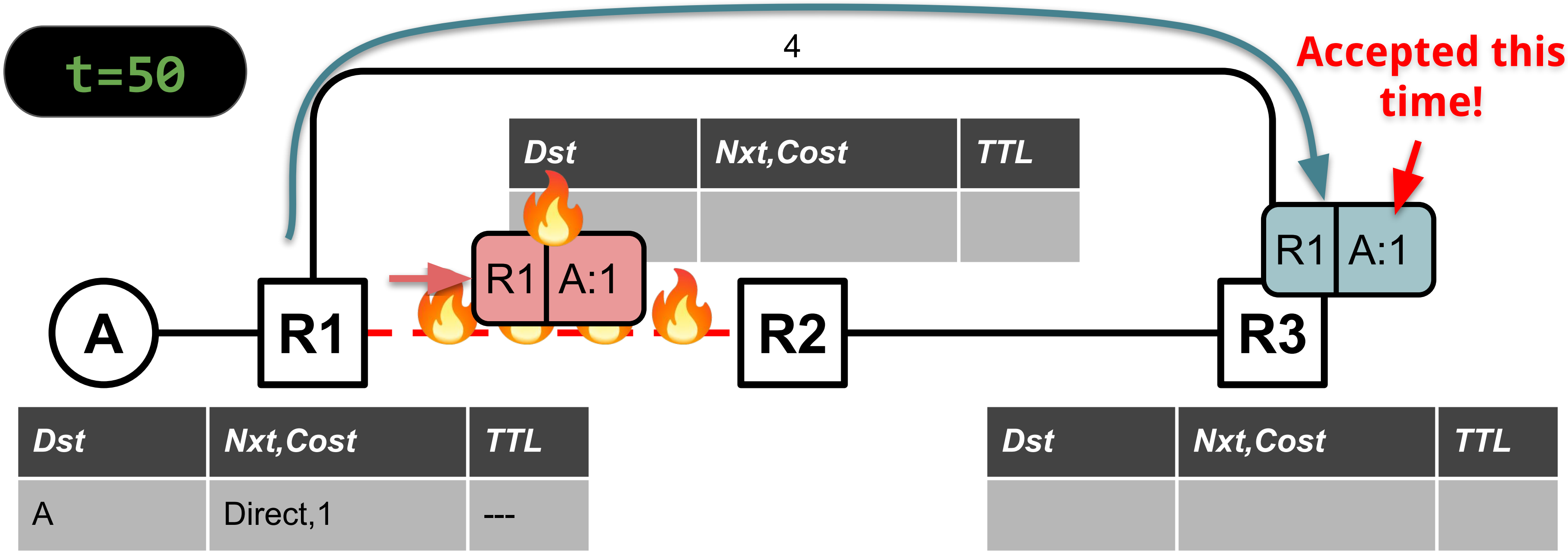


Dst	Nxt,Cost	TTL

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL

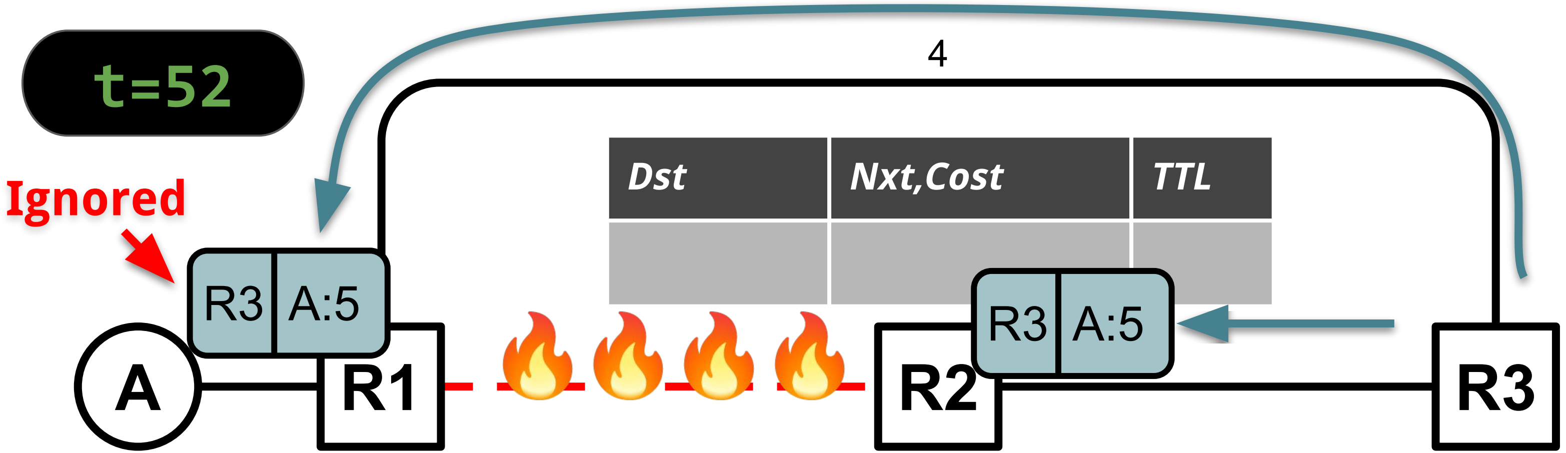
D-V: Failures


t=50
4
Accepted this time!
Dst
Nxt,Cost
TTL
R1
A:1
R1
A:1
A
R1
R2
R3
Dst
Nxt,Cost
TTL
A
Direct,1

Dst
Nxt,Cost
TTL

D-V: Failures

t=52



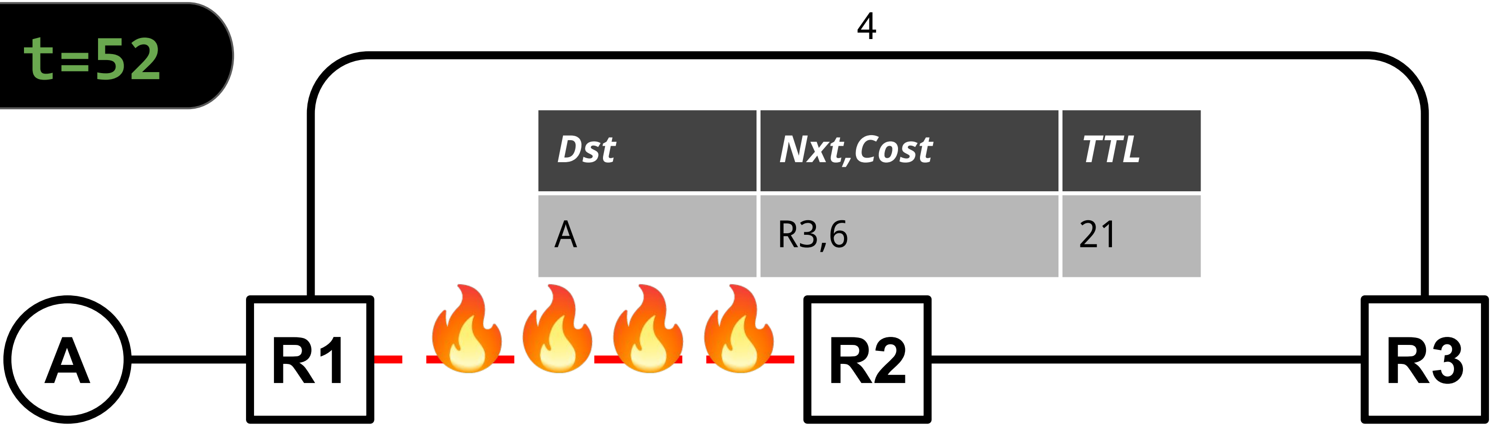
Dst	Nxt,Cost	TTL

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R1,5	19

D-V: Failures

t=52



Dst	Nxt,Cost	TTL
A	R3,6	21

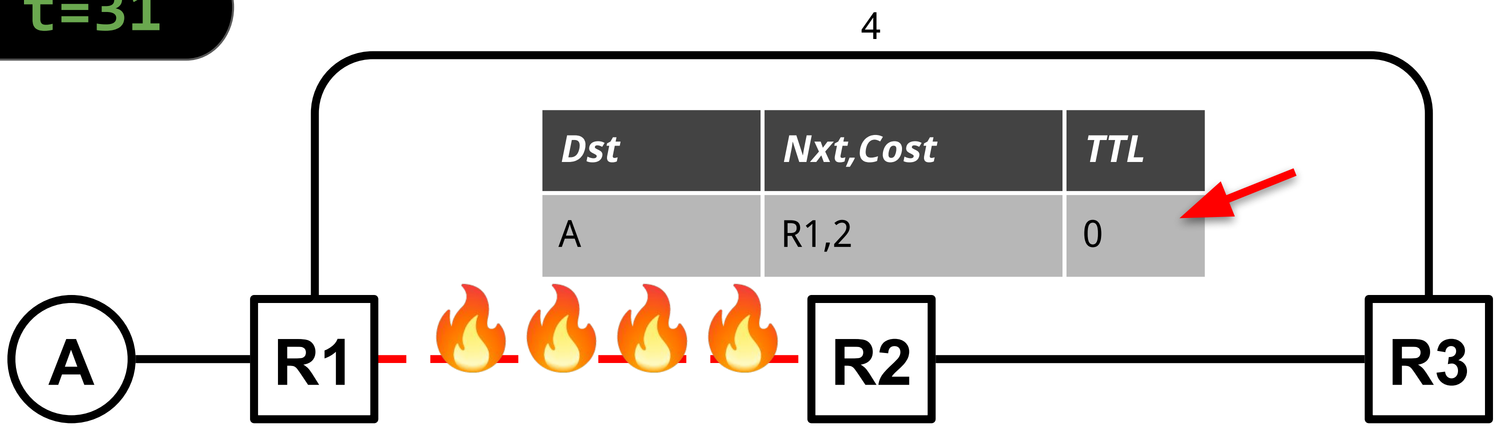
Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R1,5	19

Showing the absence of a route - poisoning.

D-V: Poisoning

t=31



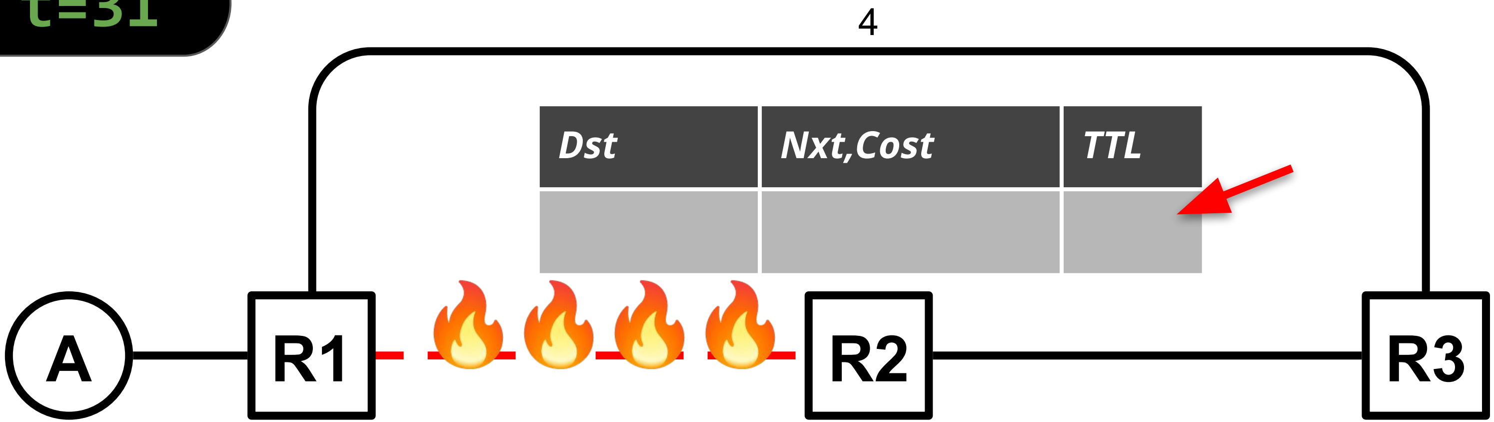
Dst	Nxt,Cost	TTL
A	R1,2	0

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	15

D-V: Poisoning

t=31

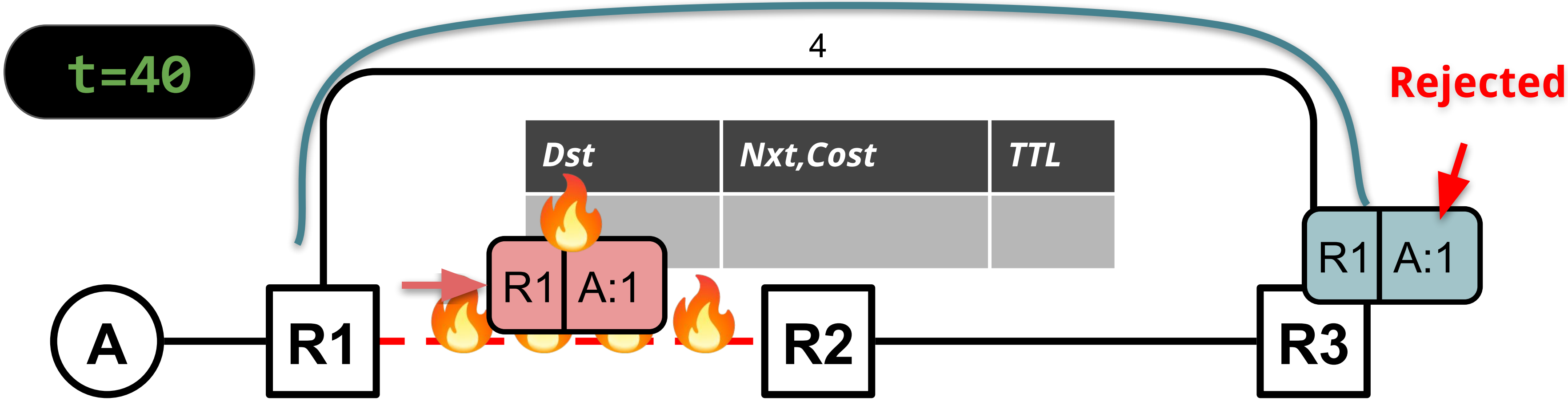


Dst	Nxt,Cost	TTL

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	15

D-V: Poisoning

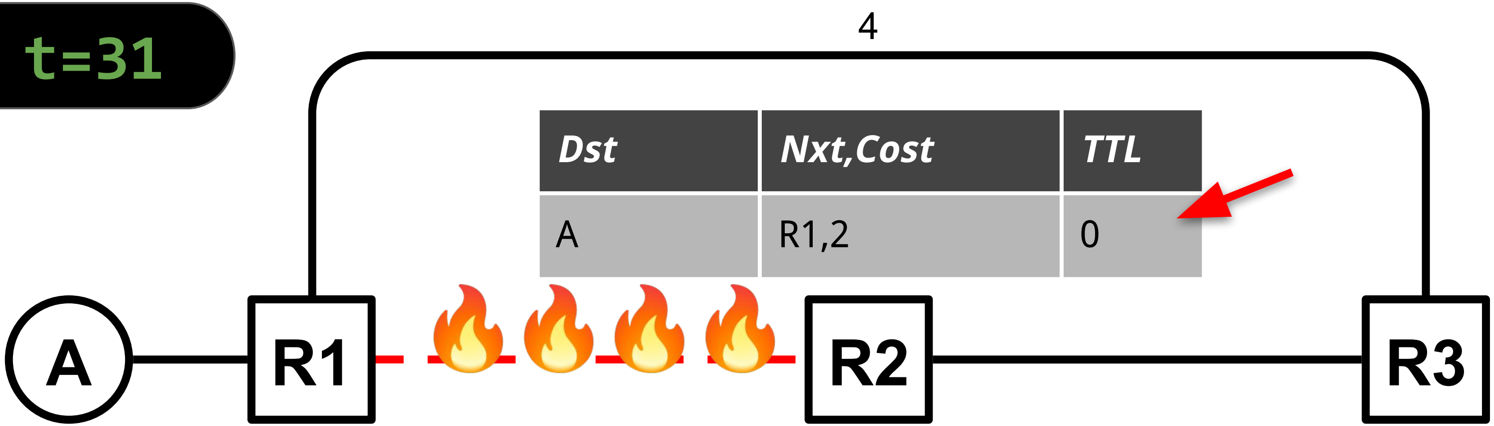


Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	6

D-V: Poisoning

t=31



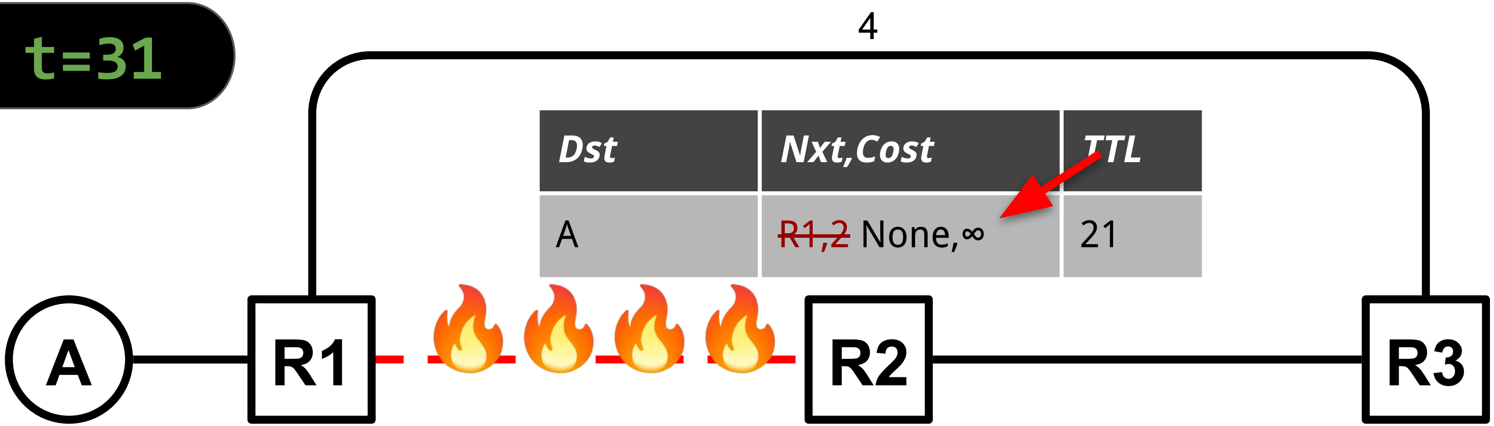
Dst	Nxt,Cost	TTL
A	R1,2	0

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	15

D-V: Poisoning

t=31



R2:

Dst	Nxt,Cost	TTL
A	~~R1,2~~ None,∞	21

R1:

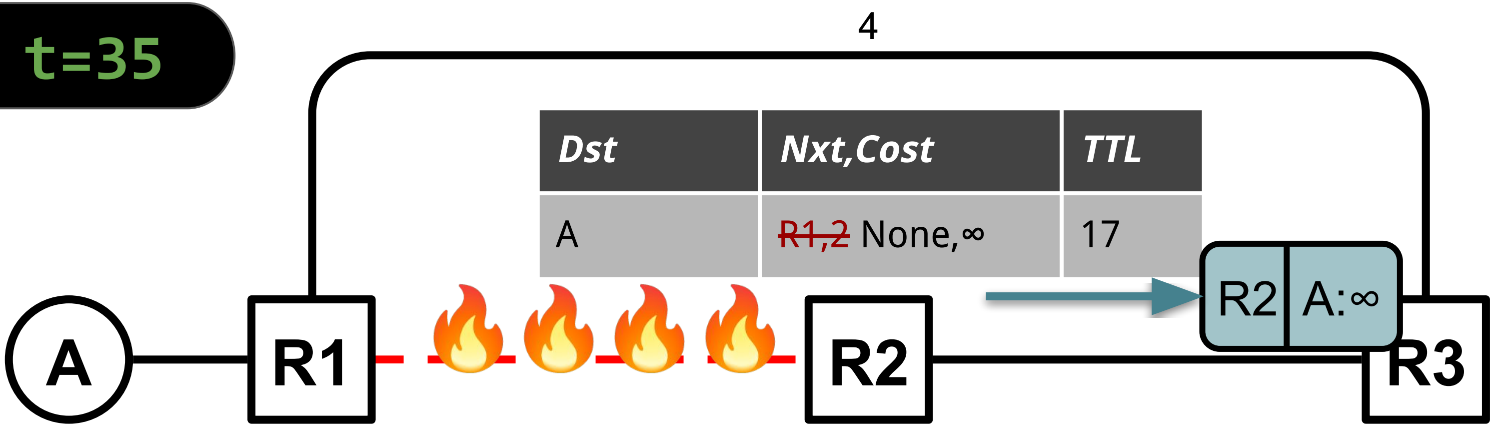
Dst	Nxt,Cost	TTL
A	Direct,1	---

R3:

Dst	Nxt,Cost	TTL
A	R2,3	15

D-V: Poisoning

t=35



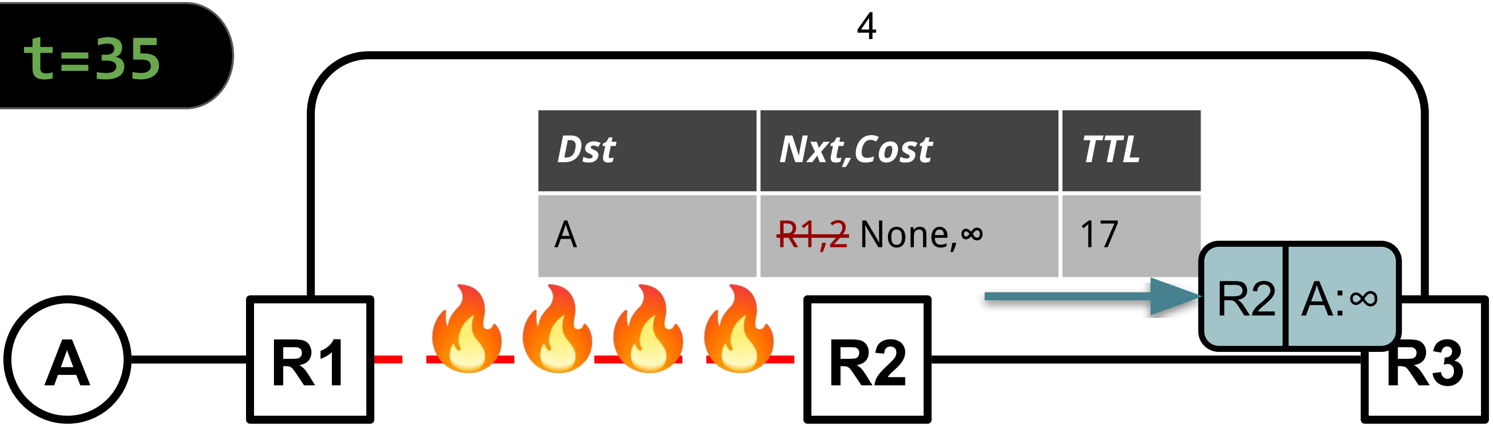
Dst	*Nxt,Cost*	*TTL*
A	~~R1,2~~ None,∞	17

Dst	*Nxt,Cost*	*TTL*
A	Direct,1	---

Dst	*Nxt,Cost*	*TTL*
A	R2,3	11

D-V: Poisoning

t=35

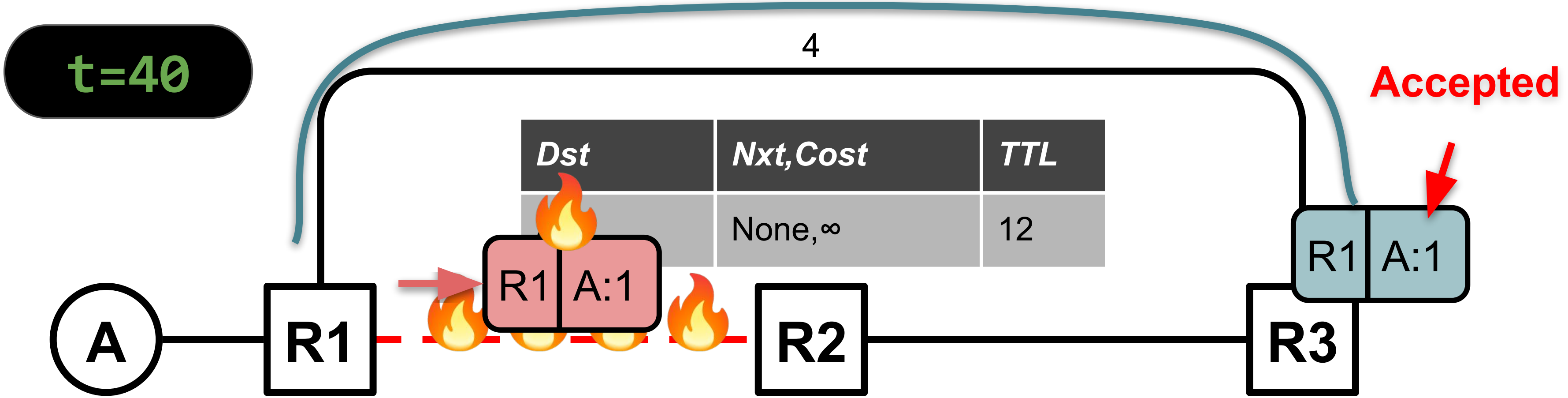


Dst	Nxt,Cost	TTL
A	~~R1,2~~ None,∞	17

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,~~3~~ ∞	21

D-V: Poisoning



Dst	Nxt,Cost	TTL
	None,∞	12

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	~~R2,∞~~ R1,5	~~16~~ 21

D-V: Poison

- Key idea:
  - Instead of just *not* advertising a route
  - .. actively advertise that you *don't* have a route

- Do this by advertising an impossibly high cost
  - A "poison" route

- This route should propagate like other routes, poisoning the entry on any other router that was using it

- Can be much faster than waiting for timeouts!

D-V: Poison

- And this doesn't just work for timed advertisements…

- If you get a poison advertisement and it changes your table…
    - Will trigger you to send poison
    - Propagates dead routes as fast as they can reach and be processed by neighbor!

- .. can be much, *much* faster than waiting for timeouts!

D-V: Poison

- Besides expired routes, where else did we *not* advertise something?

D-V: Poison

- Besides expired routes, where else did we *not* advertise something?
  - Split horizon!

- In split horizon, we had a route but chose not to advertise
  - Don’t want to advertise a route back to router that advertised it to us!
  - Can lead to sending things backwards (or even looping)

D-V: Poison

- Besides expired routes, where else did we *not* advertise something?
  - Split horizon!

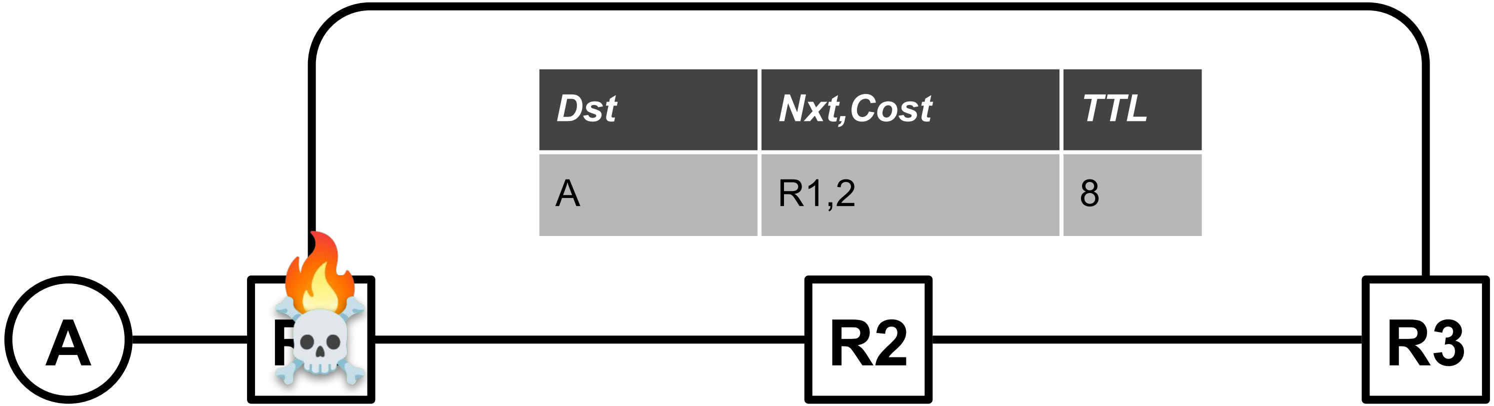
- In split horizon, we had a route but chose not to advertise
  - Don't want to advertise a route back to router that advertised it to us!
  - Can lead to sending things backwards (or even looping)

- Instead of *not* advertising in this case… *advertise infinite cost*
  - We call this *poison reverse*

  - Same exact idea as split horizon, but more aggressive

D-V: Poison Reverse

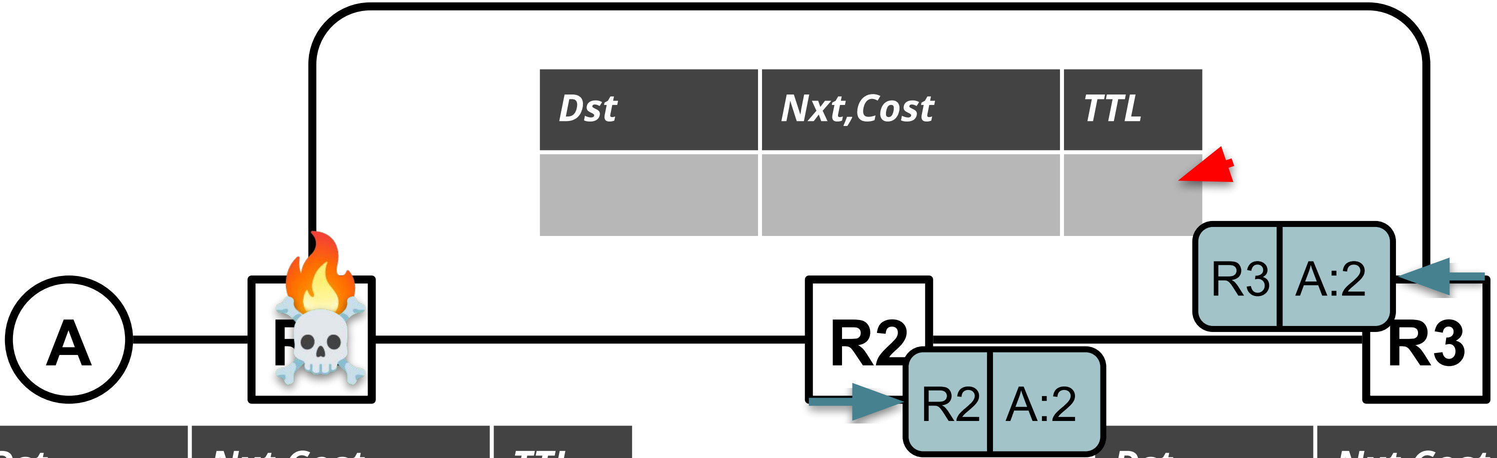
Dst	Nxt,Cost	TTL
A	R1,2	8



Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R1,2	8

D-V: Poison Reverse

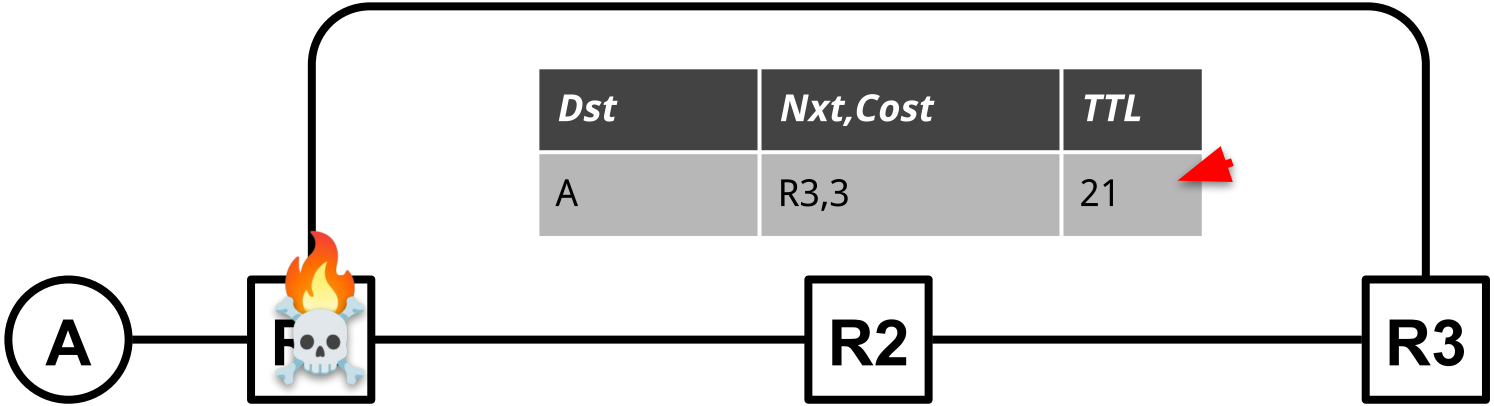


Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL

Dst	Nxt,Cost	TTL

D-V: Poison Reverse



Dst	Nxt,Cost	TTL
A	R3,3	21

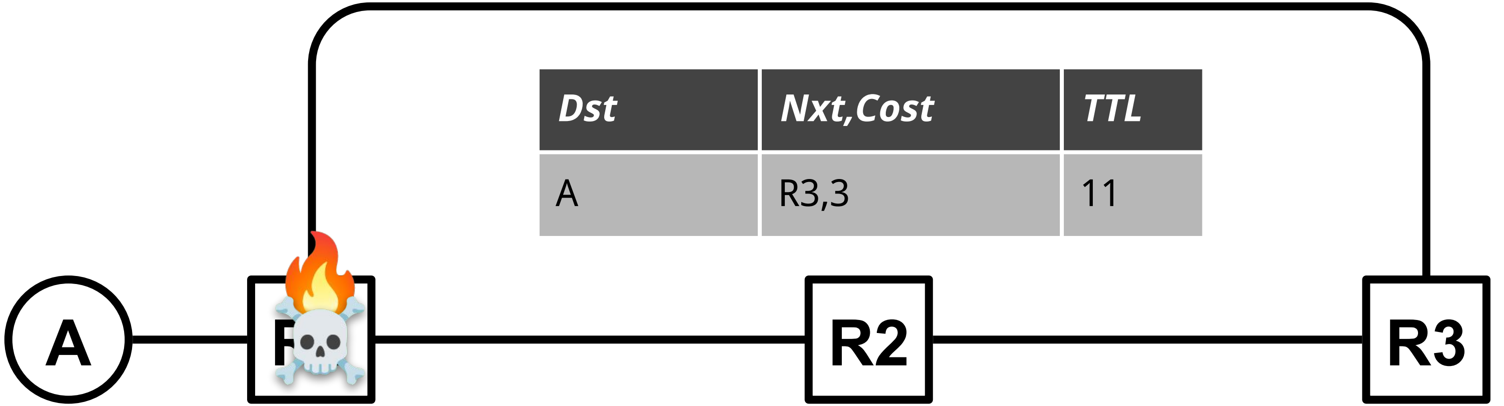
Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	21

With split horizon, loopy state exists until expiration

D-V: Poison Reverse

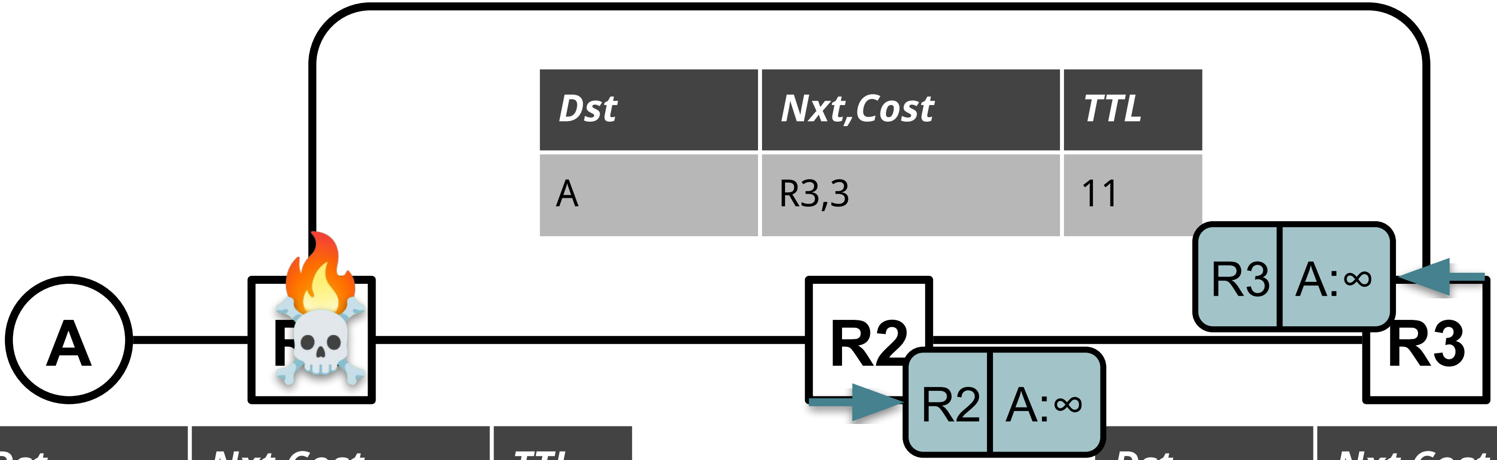
Dst	Nxt,Cost	TTL
A	R3,3	11



Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	11

D-V: Poison Reverse

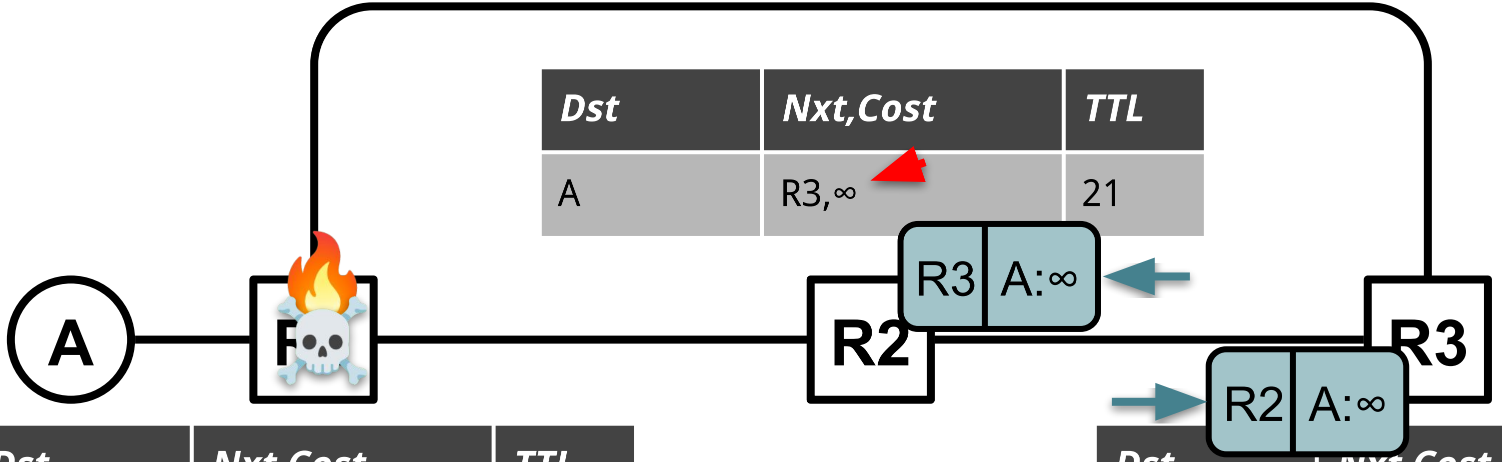


Dst	Nxt,Cost	TTL
A	R3,3	11

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,3	11

D-V: Poison Reverse



Dst	Nxt,Cost	TTL
A	R3,∞	21

Dst	Nxt,Cost	TTL
A	Direct,1	---

Dst	Nxt,Cost	TTL
A	R2,∞	21

With poison reverse, loopy state exists until next advertisement

D-V: Poison

- Poisoning and poison reverse…

- In both cases, without poisoning, you would have *not* sent a route
- Instead, *send an explicitly terrible route* (any other route will be better)
  - (And never forward using these terrible infinite-length routes.)

D-V: More triggers

- We know that our table changing should trigger us to send an update
- Can be useful to handle other events too...

D-V: More triggers

- We know that our table changing should trigger us to send an update

- Can be useful to handle other events too…

- Sometimes we can detect when a link becomes available
  - Immediately send new neighbor advertisements
  - No need to wait for timer

D-V: More triggers

- We know that our table changing should trigger us to send an update
- Can be useful to handle other events too…
- Sometimes we can detect when a link becomes available
  - Immediately send new neighbor advertisements
  - No need to wait for timer
- Sometimes we can detect when a link fails
  - Immediately poison all table entries using that link
  - .. if there are any, advertise the newly poisoned ones!

From B-F to D-V

- We refined our update rule
- We resolved some loopy problems with split horizon
- We ensured that we eventually converge instead of counting to infinity
- We made it robust to packet drops by advertising periodically
- We saw that we can adapt to new links easily
- We can identify failed links and dead routes by missing advertisements
- We can converge faster by explicitly signaling the absence of a route
- We can adapt more quickly by advertising when “triggered” by events

- This is now a pretty good routing protocol!

Next Time

- Other types of routing protocols - *Link State.*
- Thus far - addressing has been an abstract concept.
- How do we address hosts on the Internet?
  - IPv4, IPv6.
- How do we avoid the need to advertise every single host in routing protocols?