Interior Gateway Protocols

Damien Saucez
Inria Sophia Antipolis

UBINET/CSSR 01/25/2013
Evolving Internet II

Contact information

- Damien Saucez
- Office: Inria, Lagrange L.145
- Email: damien.saucez@inria.fr
- Phone: +33 4 89.73.24.18

Outline of the course

- 12/14/2012: naming and addressing
- 12/21/2012: routing and forwarding
- 01/25/2013: interior gateway protocols
- 02/01/2013: exterior gateway protocols
- 02/08/2013: security in networks
- 02/15/2013: final examination

Table of Content

- Reminder
- Convergence issues
- Multicast
- Operational corner
- Research corner

Reminder

Link state routing

- Each router knows the entire topology and computes, locally, shortest paths
- Topology dissemination using Link State Packets (LSP) flooding
- Route computation using Dijkstra (or similar)

Distance vector routing

- No view of the entire topology
- At startup, a router only knows about itself (and the cost of each of its directly connected links)
- Each router sends to all its neighbors a distance vector with the best idea of the distance from itself to every node that it knows of
- When a router receives a distance vector, it updates its estimation of the distance to the destinations in the distance vector

Dijkstra algorithm

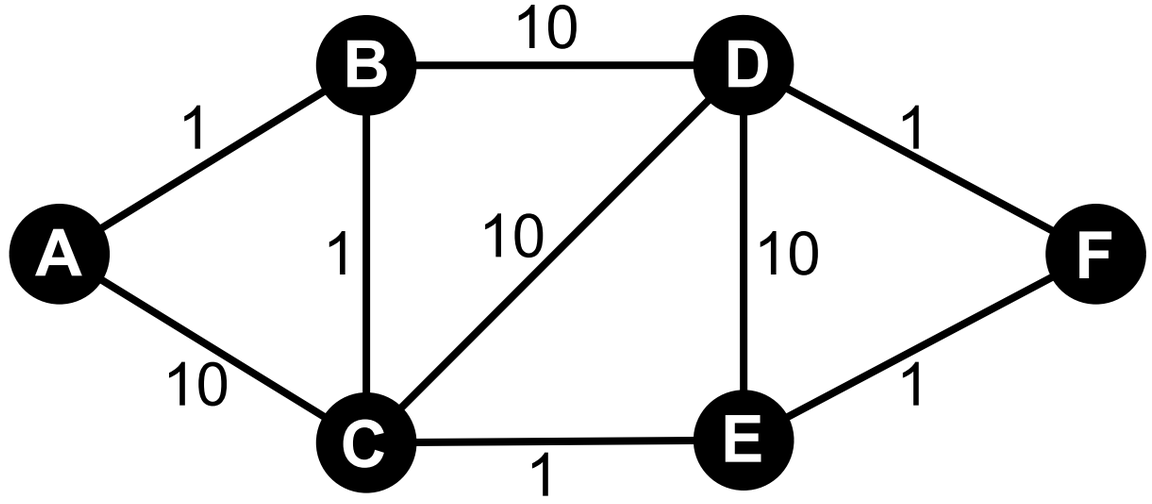
- Dijkstra algorithm computes the shortest path between a source vertex *s* and a destination vertex *d* in a positive, non-zero, weighted graph G(**V**, **E**)

```
foreach v in V do
    v.distance := ∞
    v.previous := undefined
s.distance := 0
U := V   # priority queue of Unvisited vertices, ordered by ascending distance
while U ≠ {} do
    v := pop_minimum(U)
    foreach n in v.neighbors do
        distance := v.distance + V[v, n].weight
        if distance < n.distance then
            n.distance := distance
            n.previous := v
            sort(U)
```

- Complexity: *O*(|**E**| + |**V**| log |**V**|)

Dijkstra example

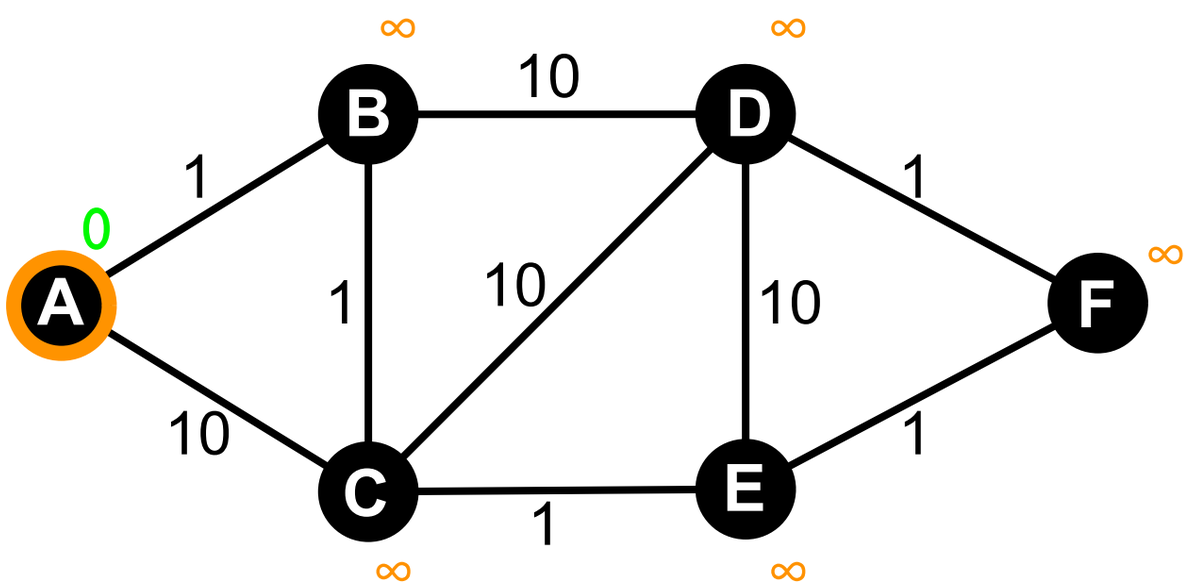
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

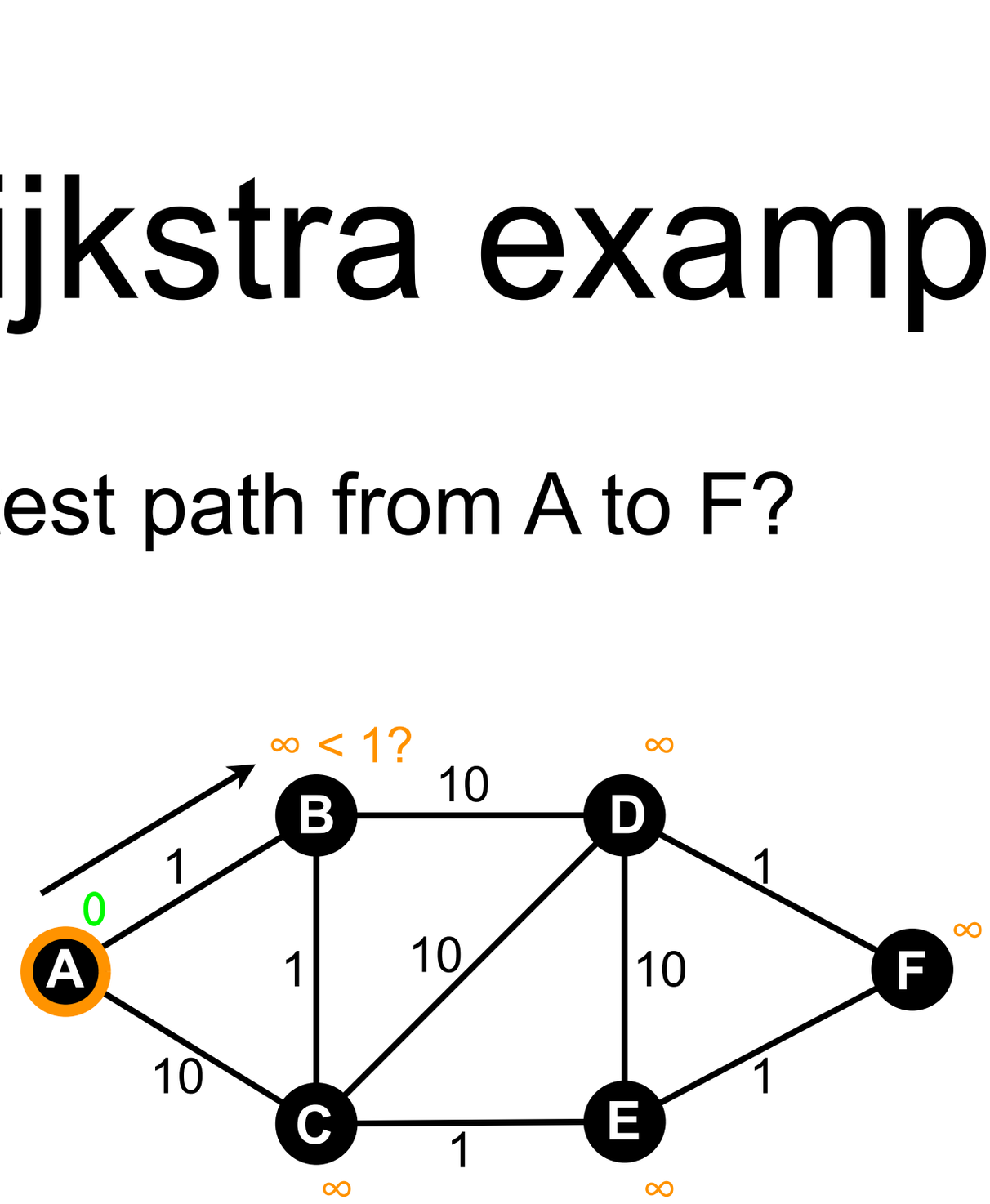
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

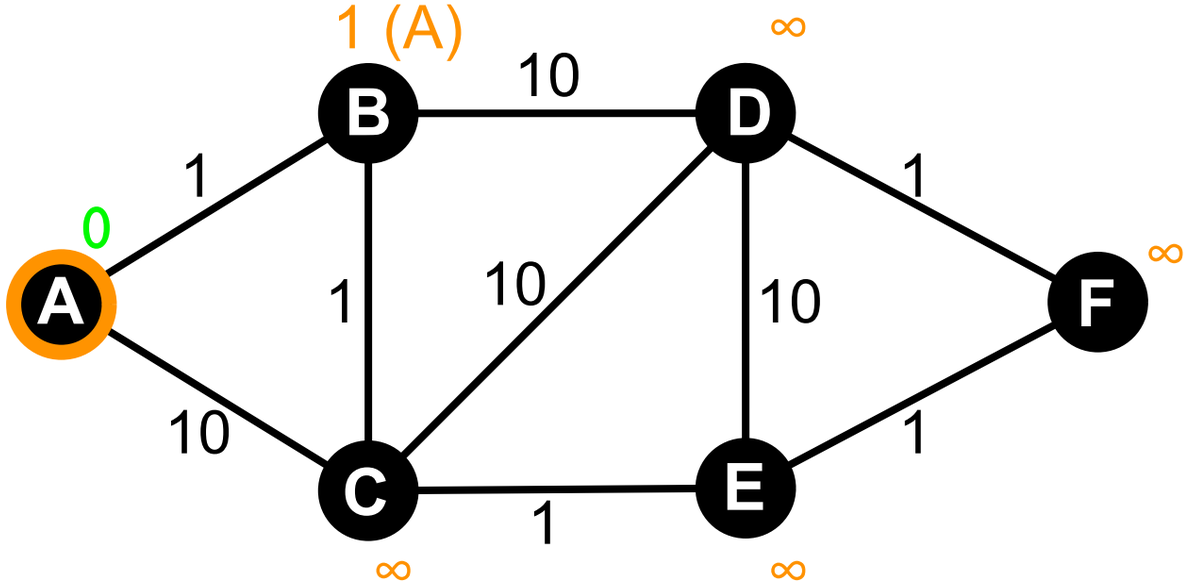
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

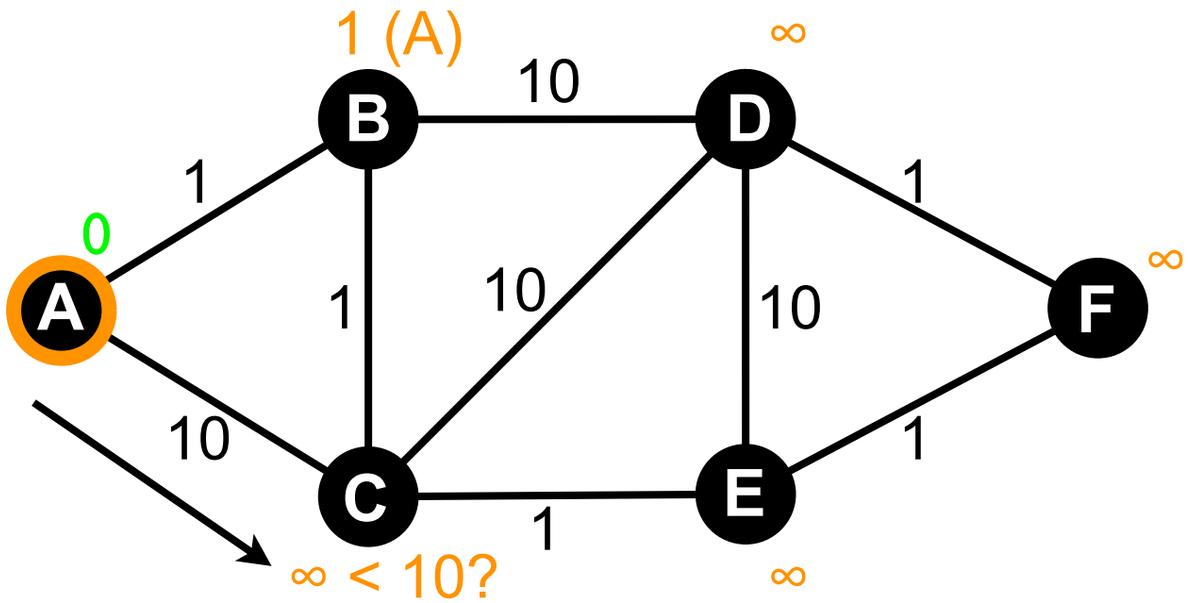
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

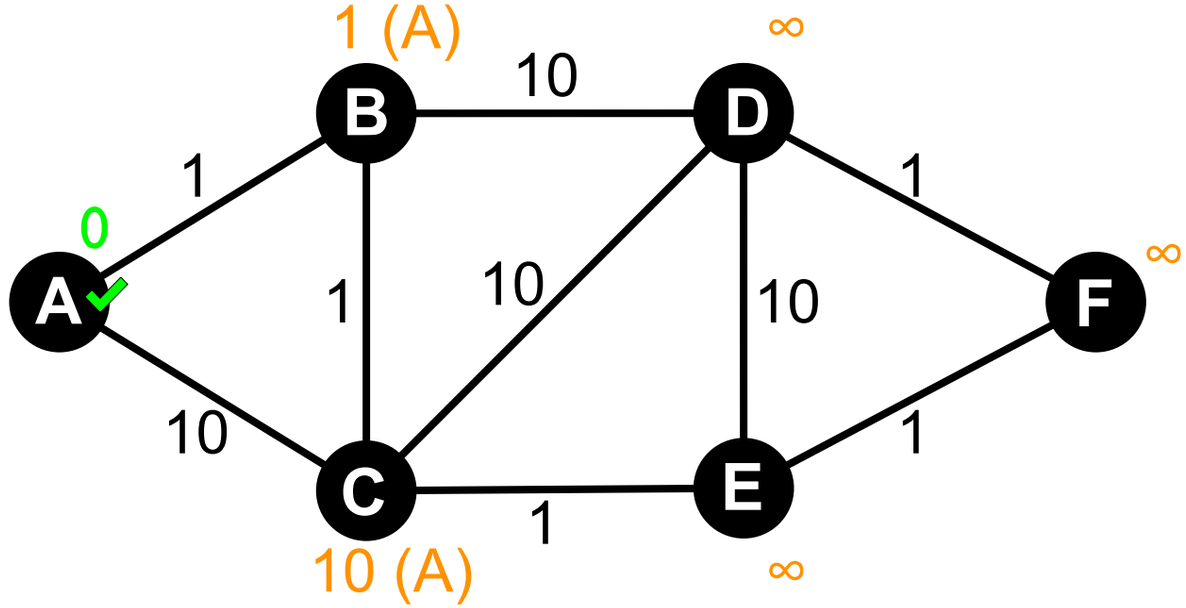
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

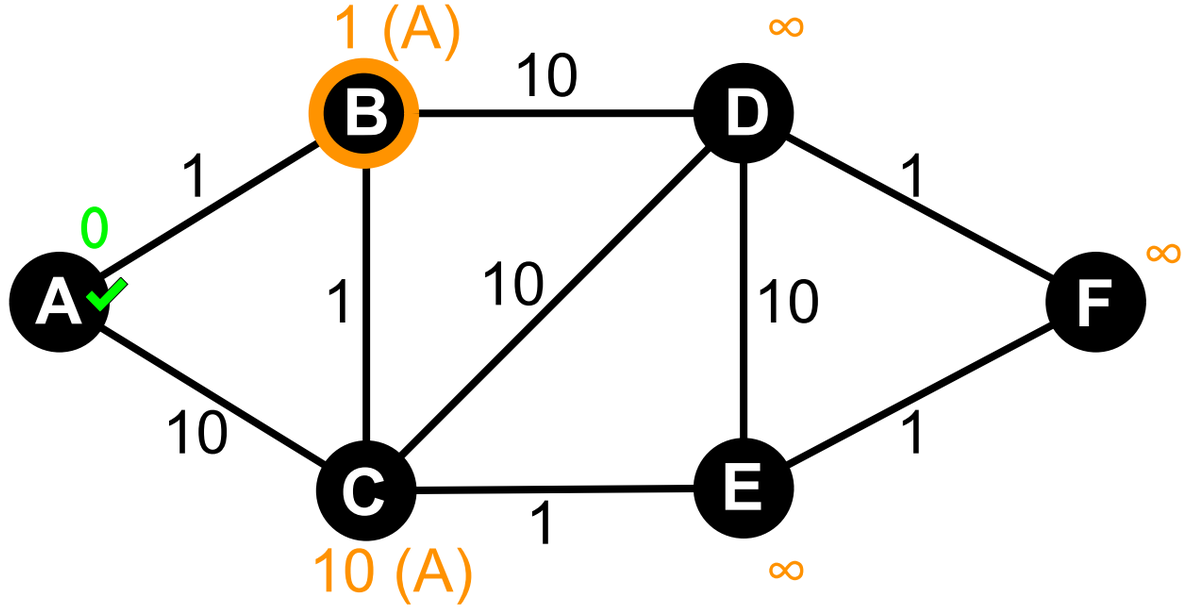
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

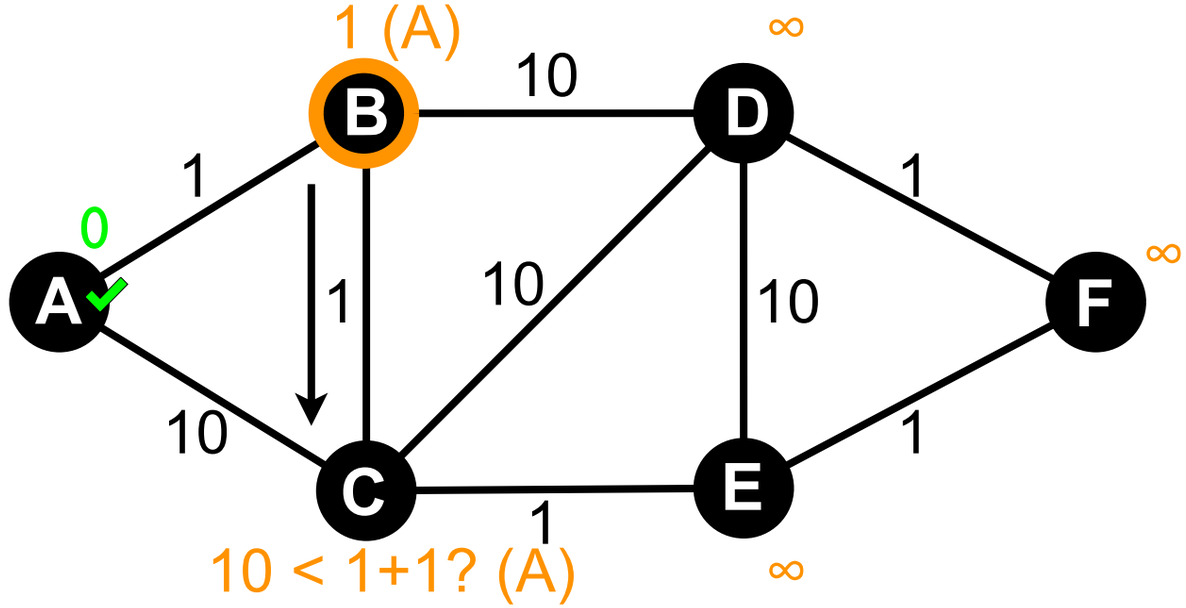
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

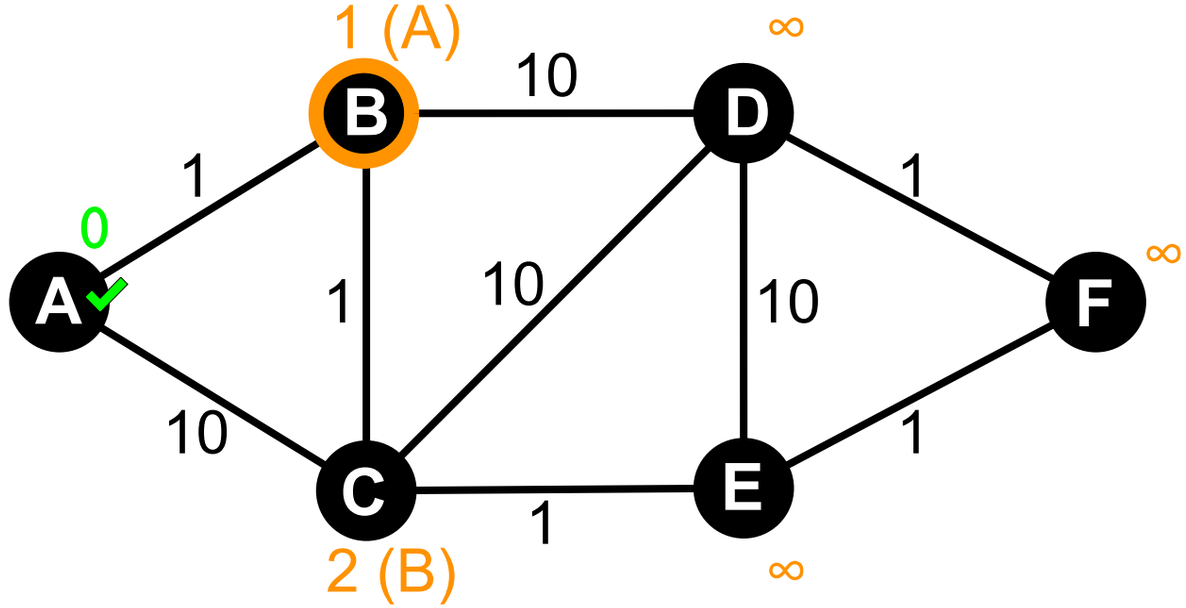
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

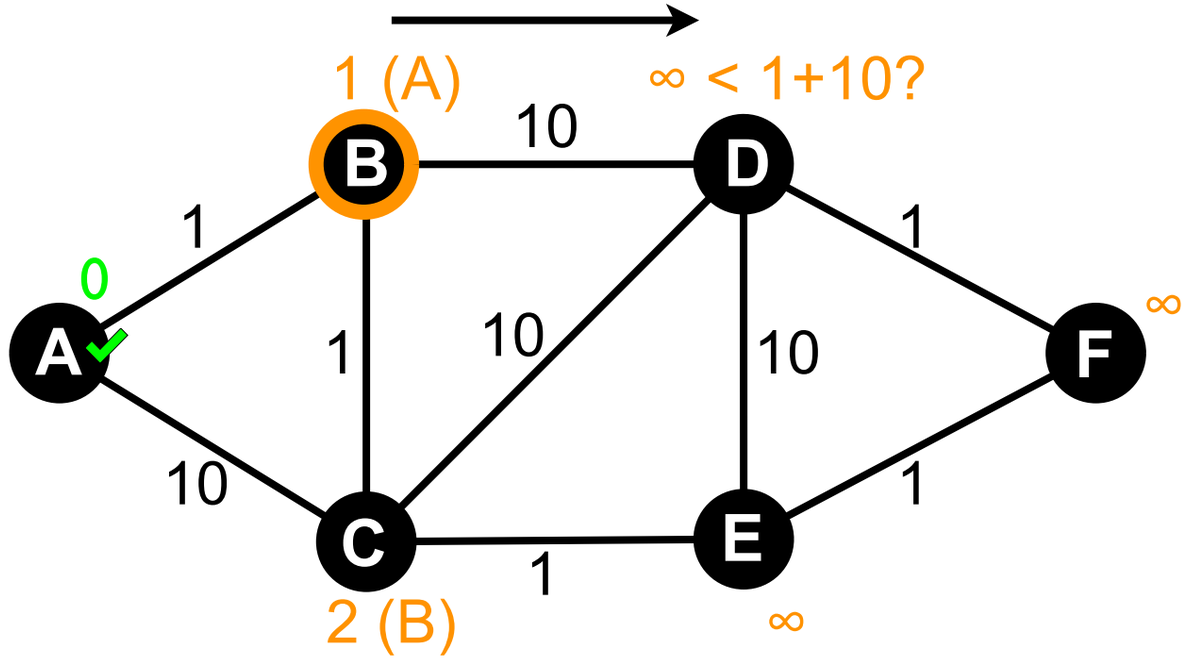
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

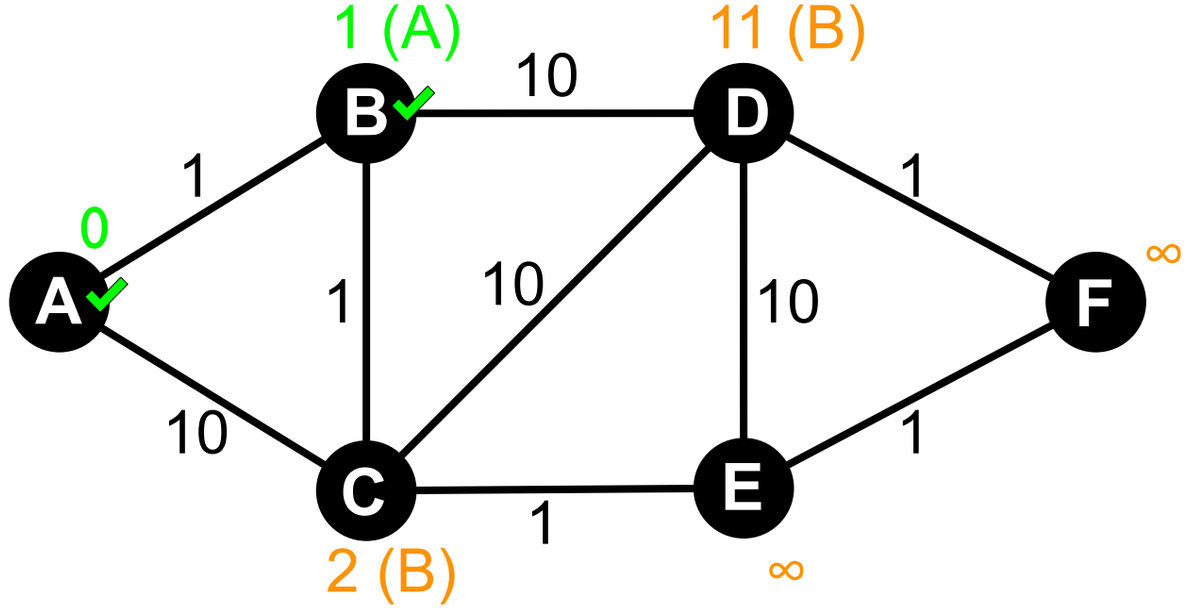
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

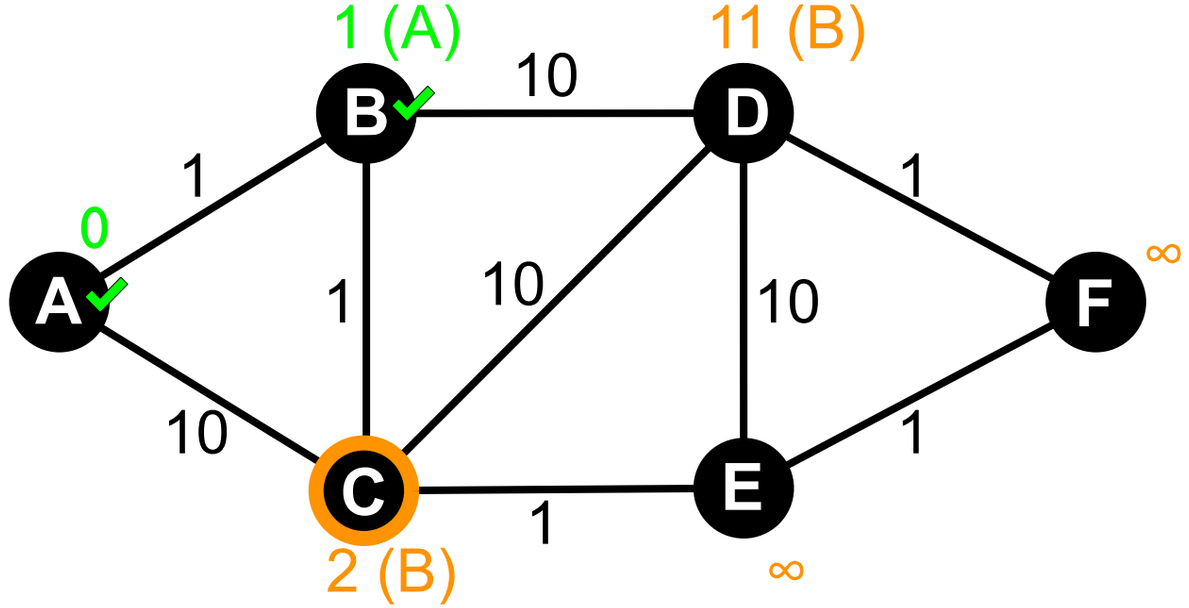
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

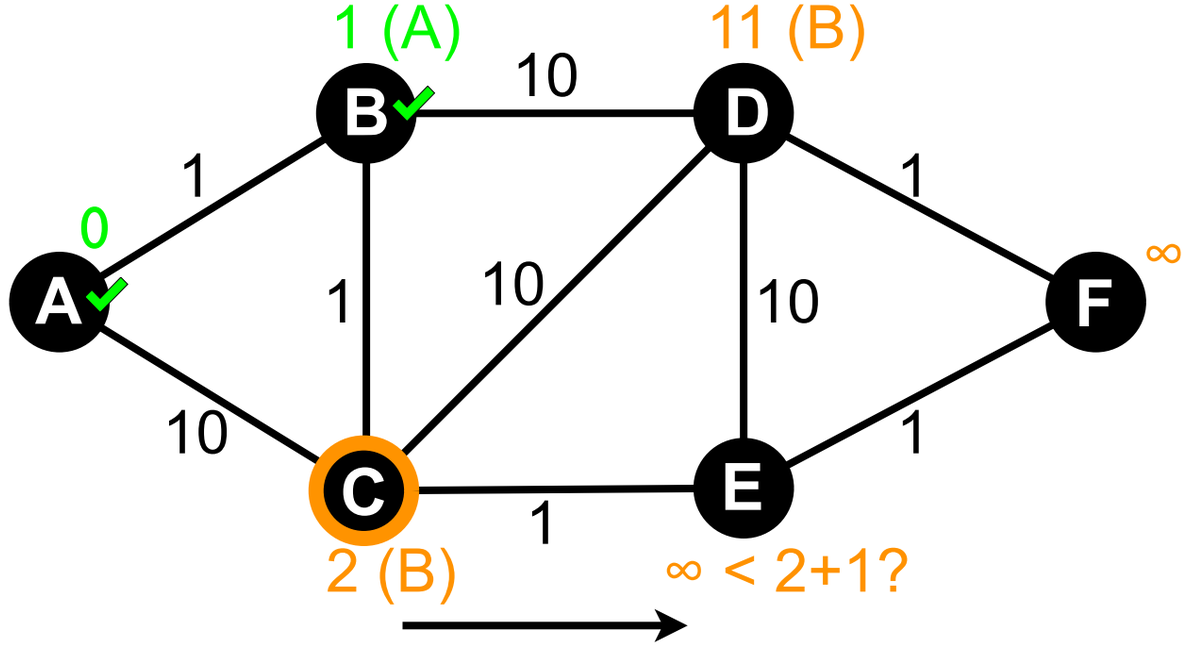
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

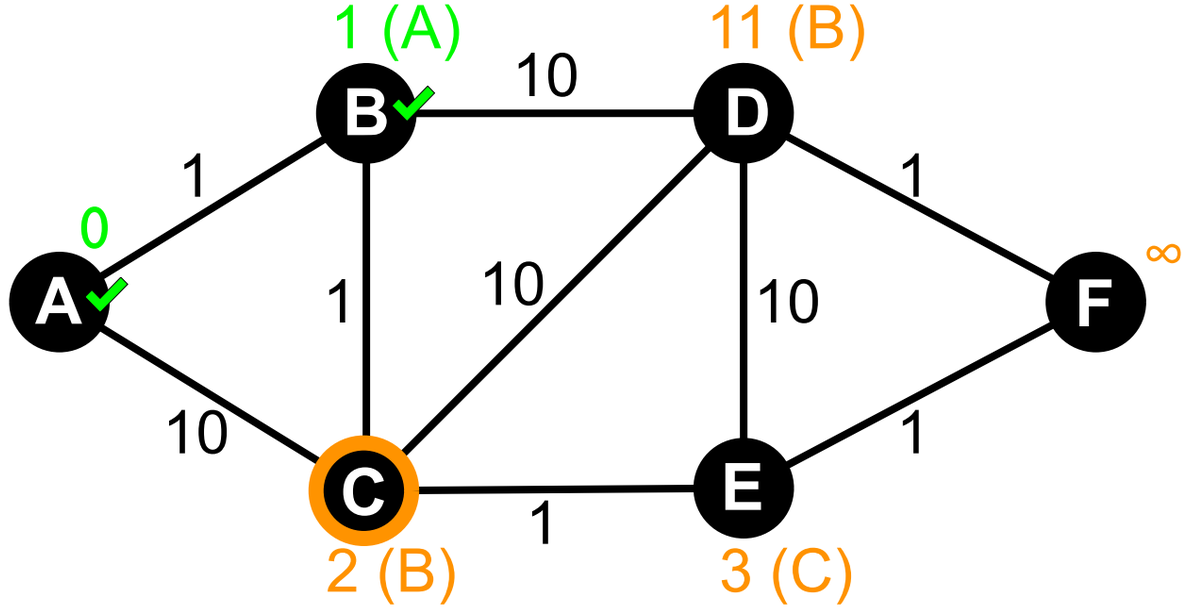
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

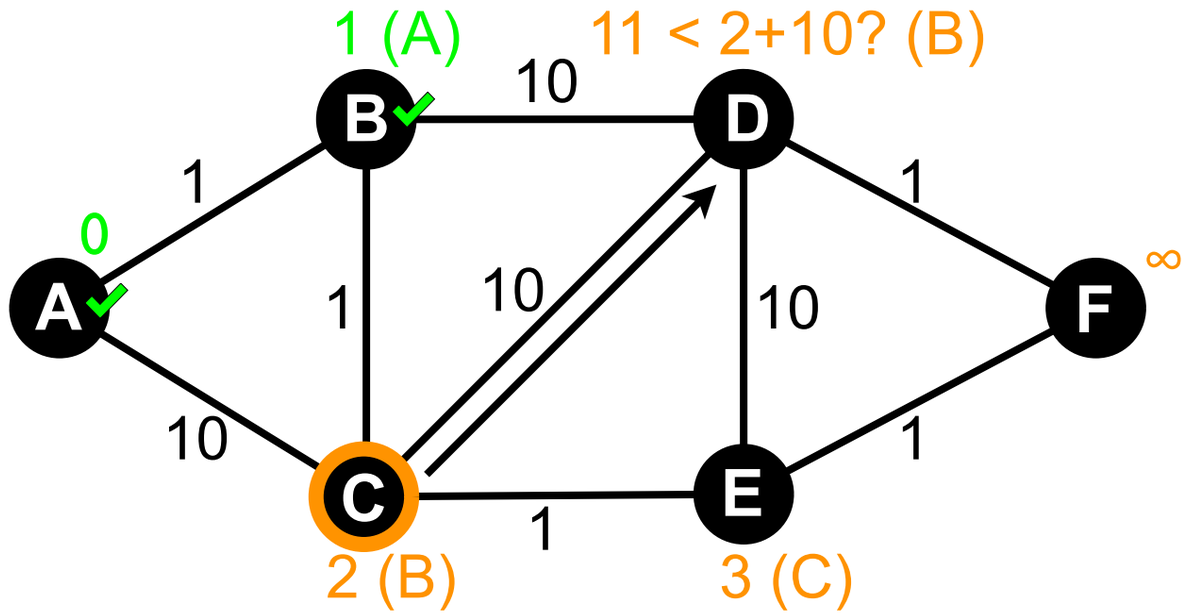
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

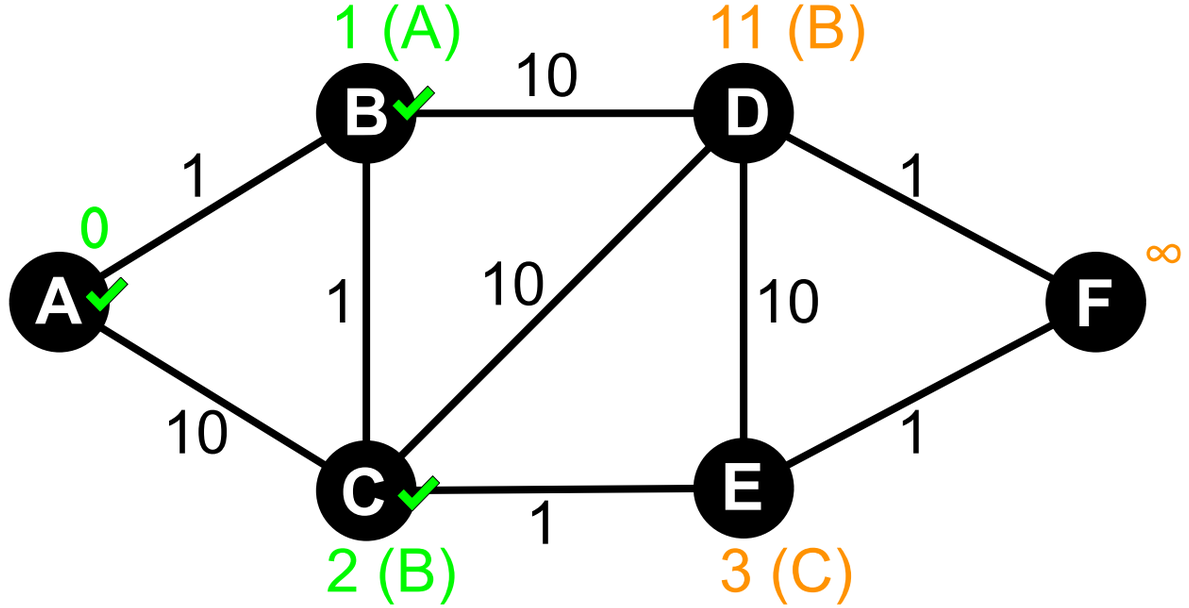
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

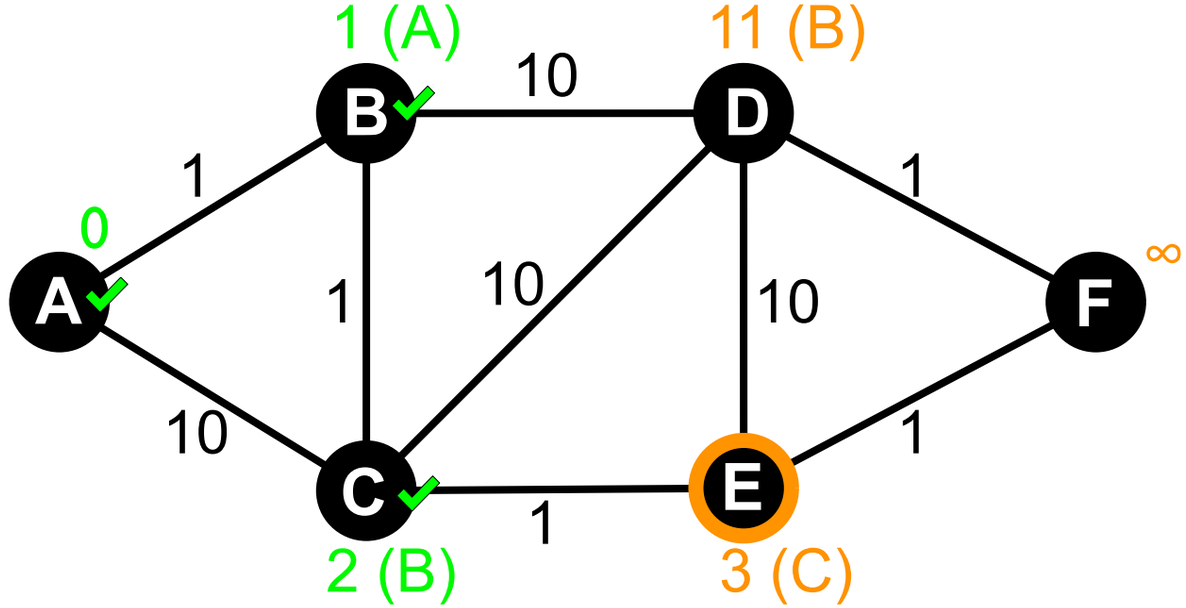
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

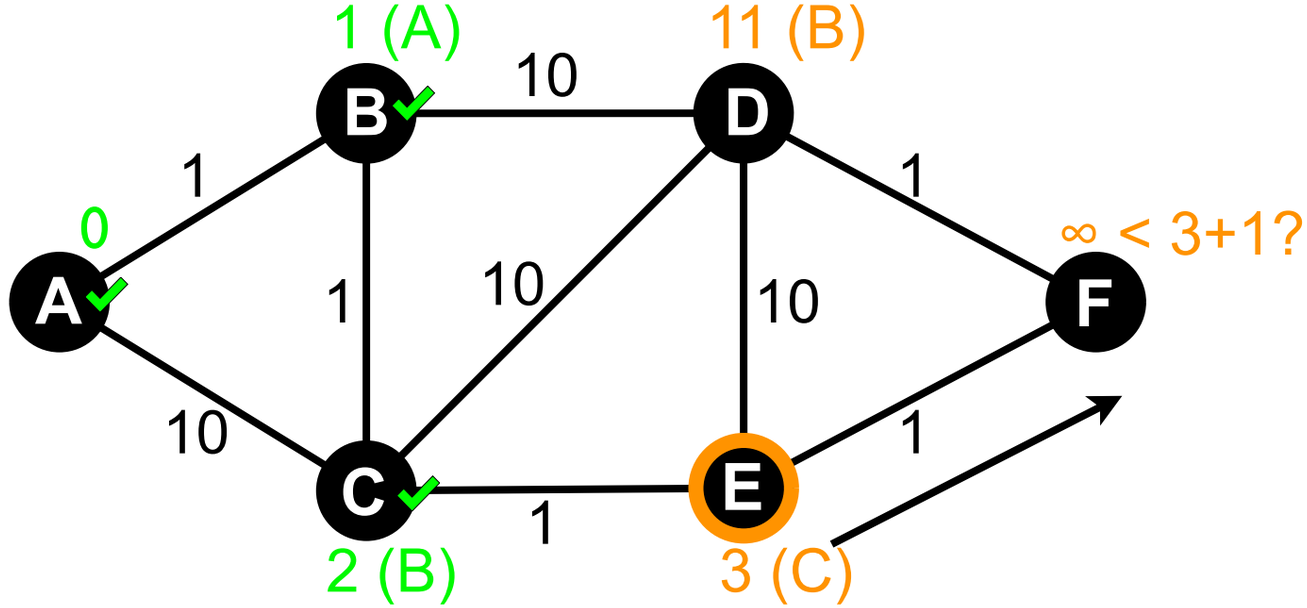
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

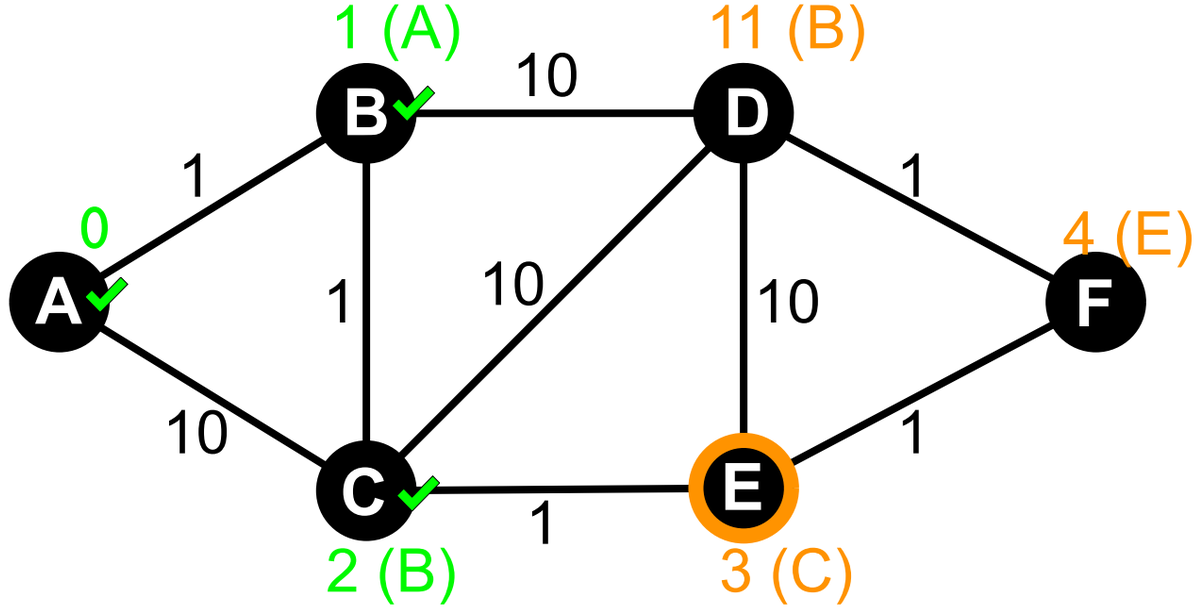
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

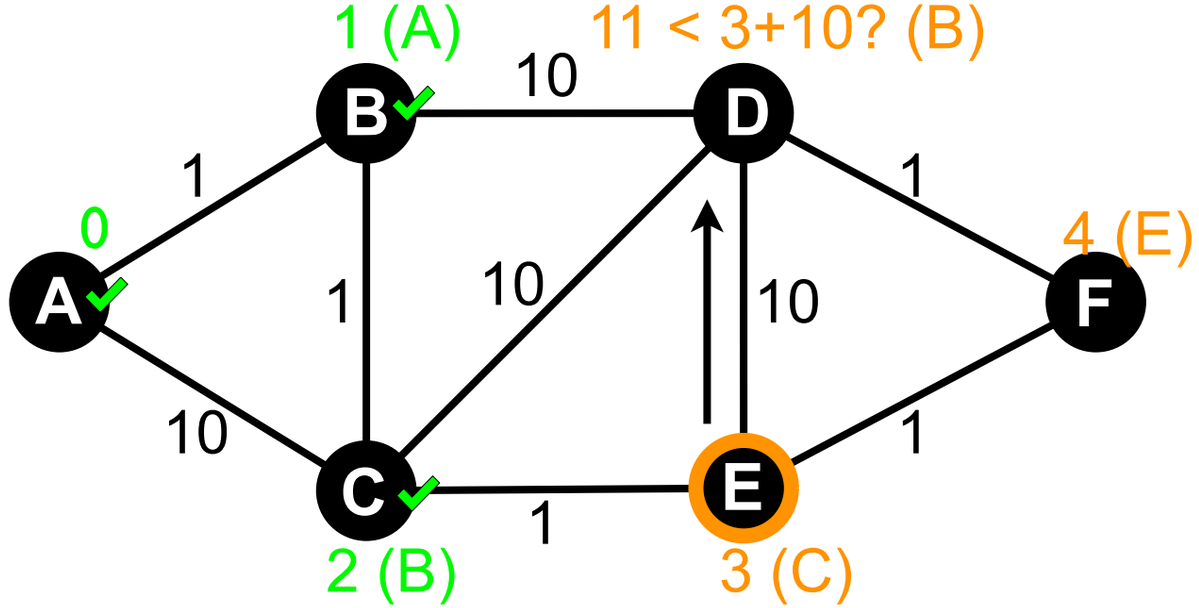
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

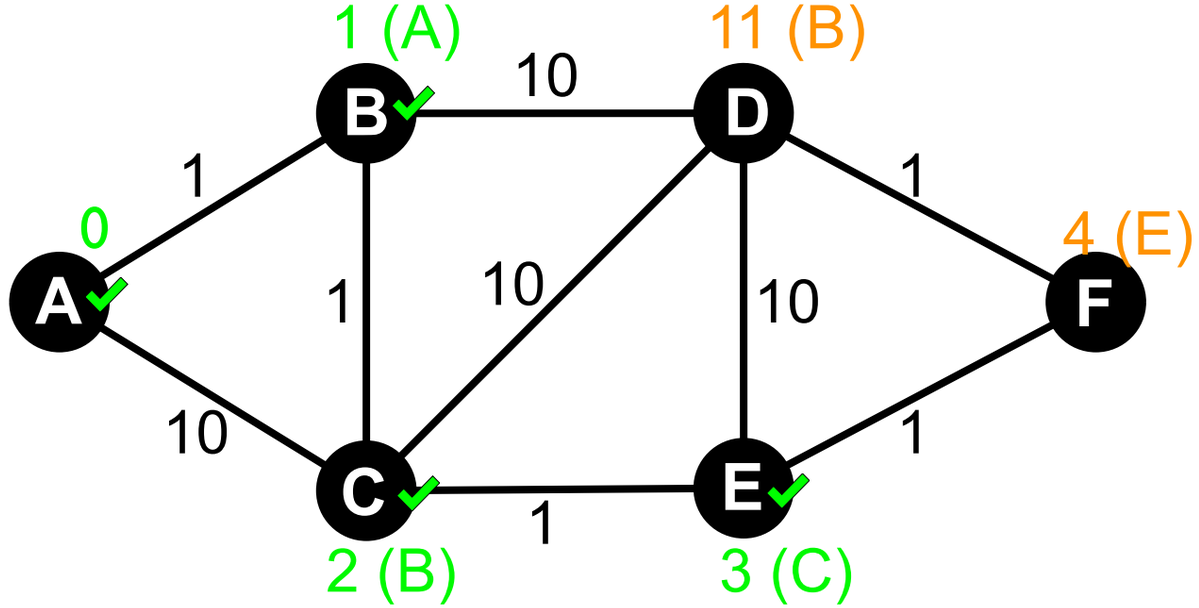
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

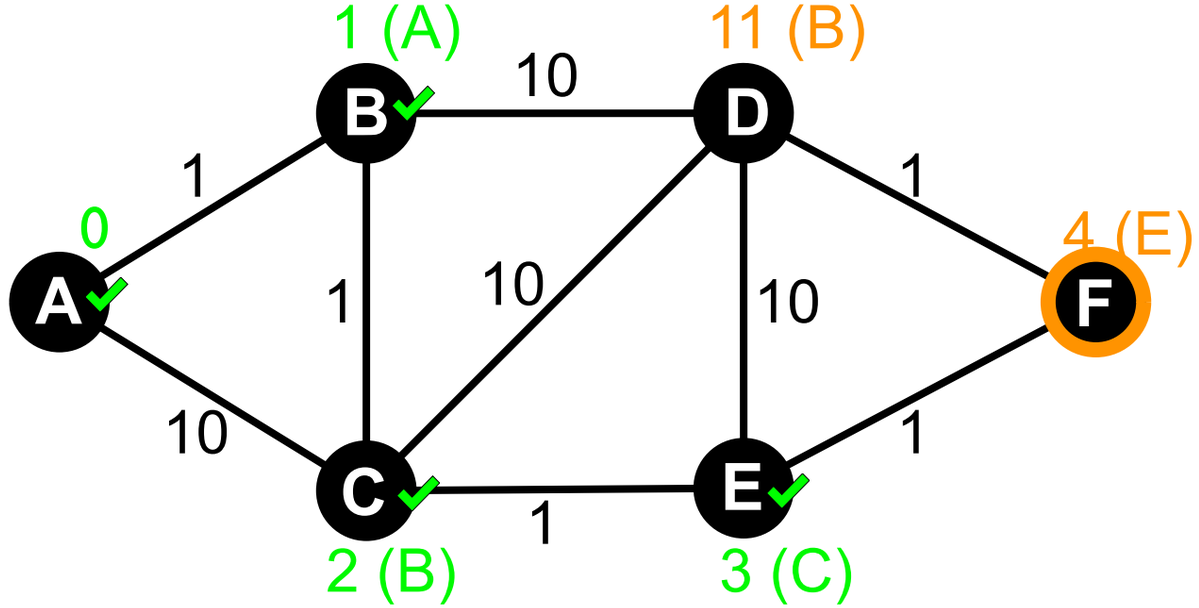
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

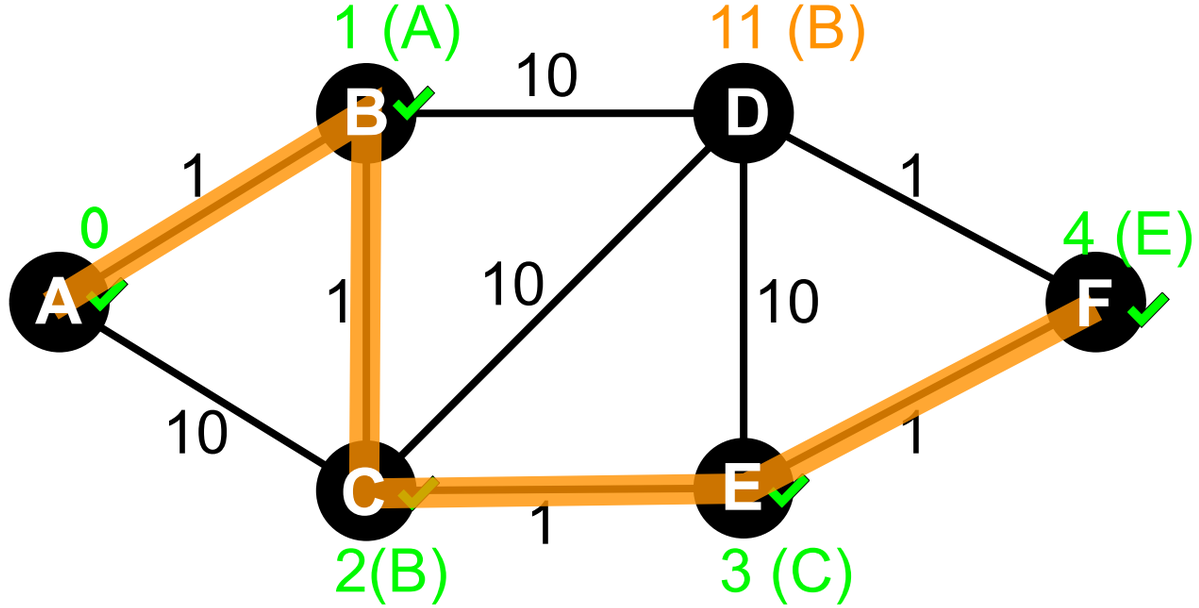
- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

Dijkstra example

- Shortest path from A to F?



inspired from http://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

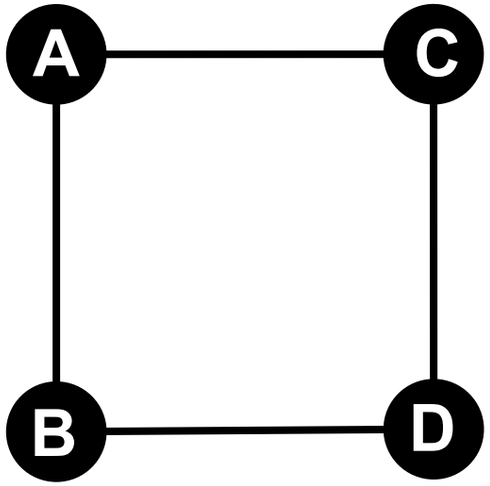
Convergence issues

Problem with DV

- In distance vector routing protocol, a router has no way to know if it is part of the routes that it receives
  - routers may think they can reach a destination that they can’t

Count-to-infinity problem

Destination	Distance
C	1, 3

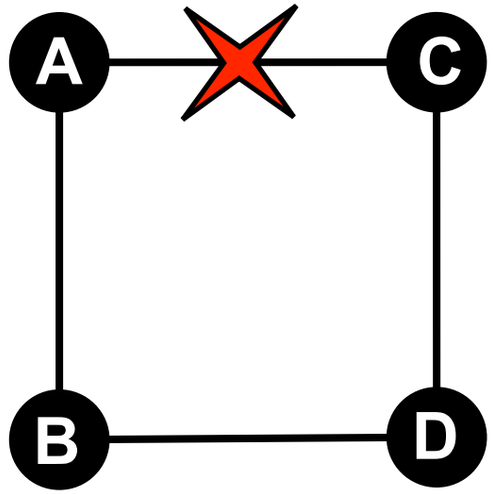


Destination	Distance
C	2, 2

Count-to-infinity problem

- First failure: A can reach C via B

Destination	Distance
C	3

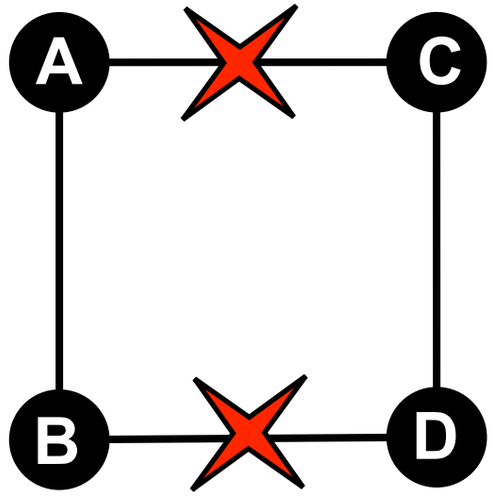


Destination	Distance
C	2, 4

Count-to-infinity problem

- Second failure: C is not reachable

Destination	Distance
C	3

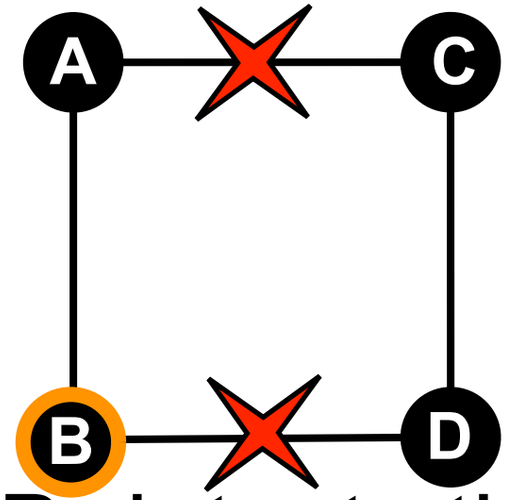


Destination	Distance
C	2, 4

Count-to-infinity problem

- Second failure: C is not reachable

Destination	Distance
C	3

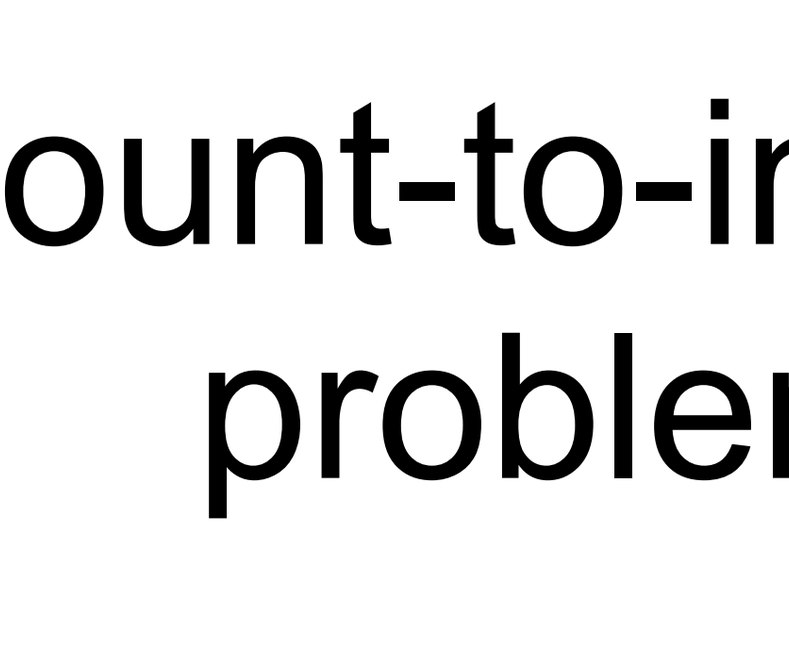


Destination	Distance
C	4

- B detects the failure but thinks it can reach C via A

Count-to-infinity problem

Destination	Distance
C	3

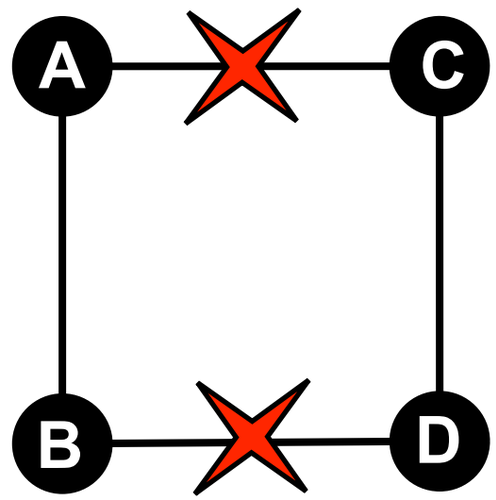


Destination	Distance
C	**4**

Count-to-infinity problem

- A now thinks it can reach C via B... we have a problem!

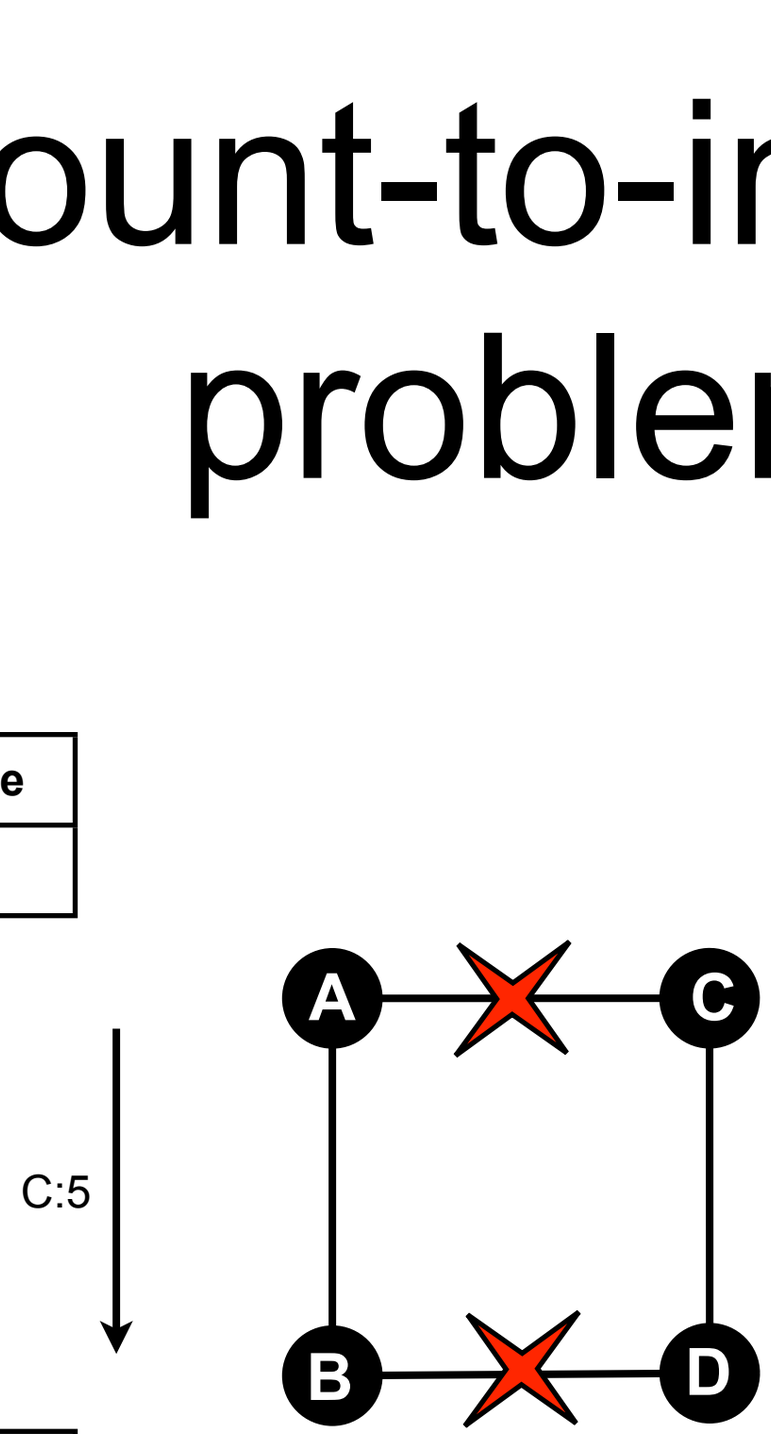
Destination	Distance
C	5



Destination	Distance
C	4

Count-to-infinity problem

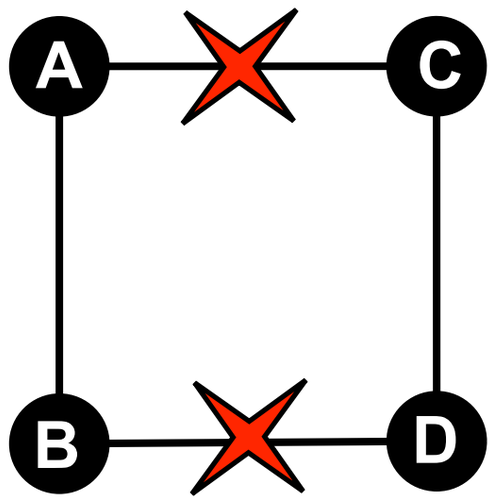
Destination	Distance
C	5



Destination	Distance
C	4

Count-to-infinity problem

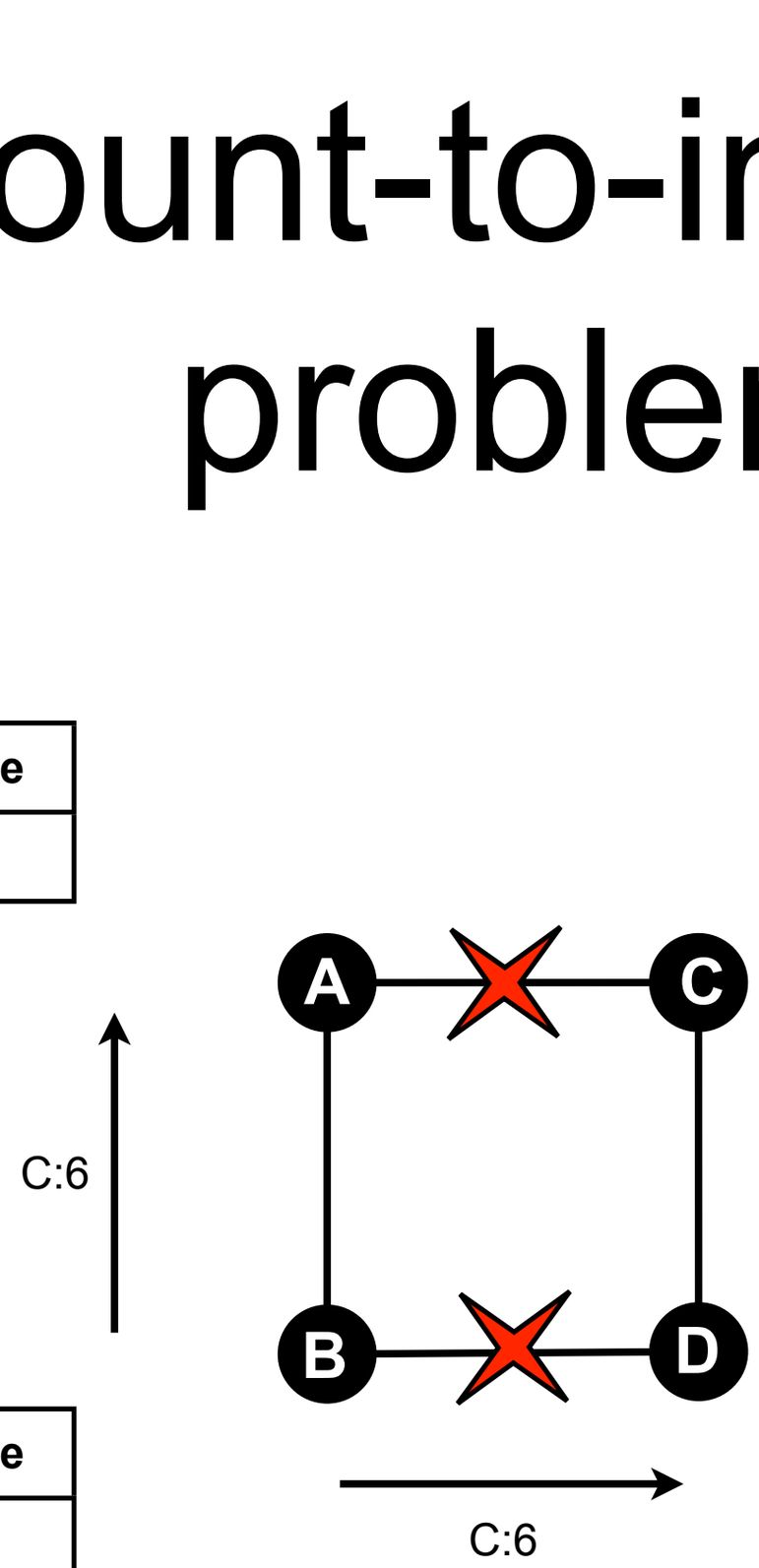
Destination	Distance
C	5



Destination	Distance
C	6

Count-to-infinity problem

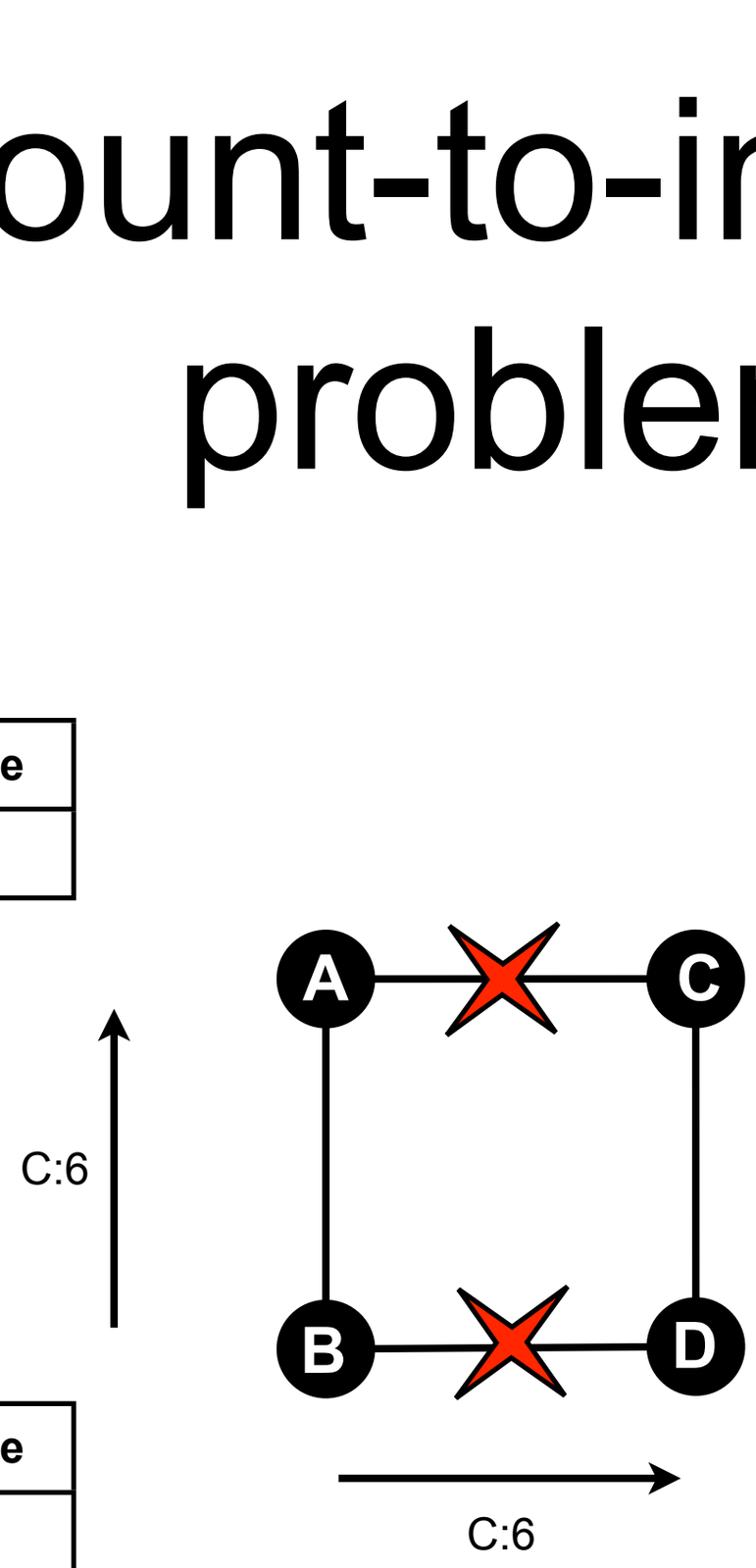
Destination	Distance
C	5



Destination	Distance
C	**6**

Count-to-infinity problem

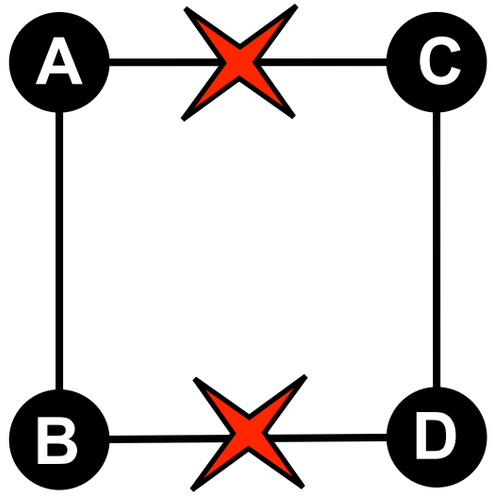
Destination	Distance
C	7



Destination	Distance
C	6

Count-to-infinity problem

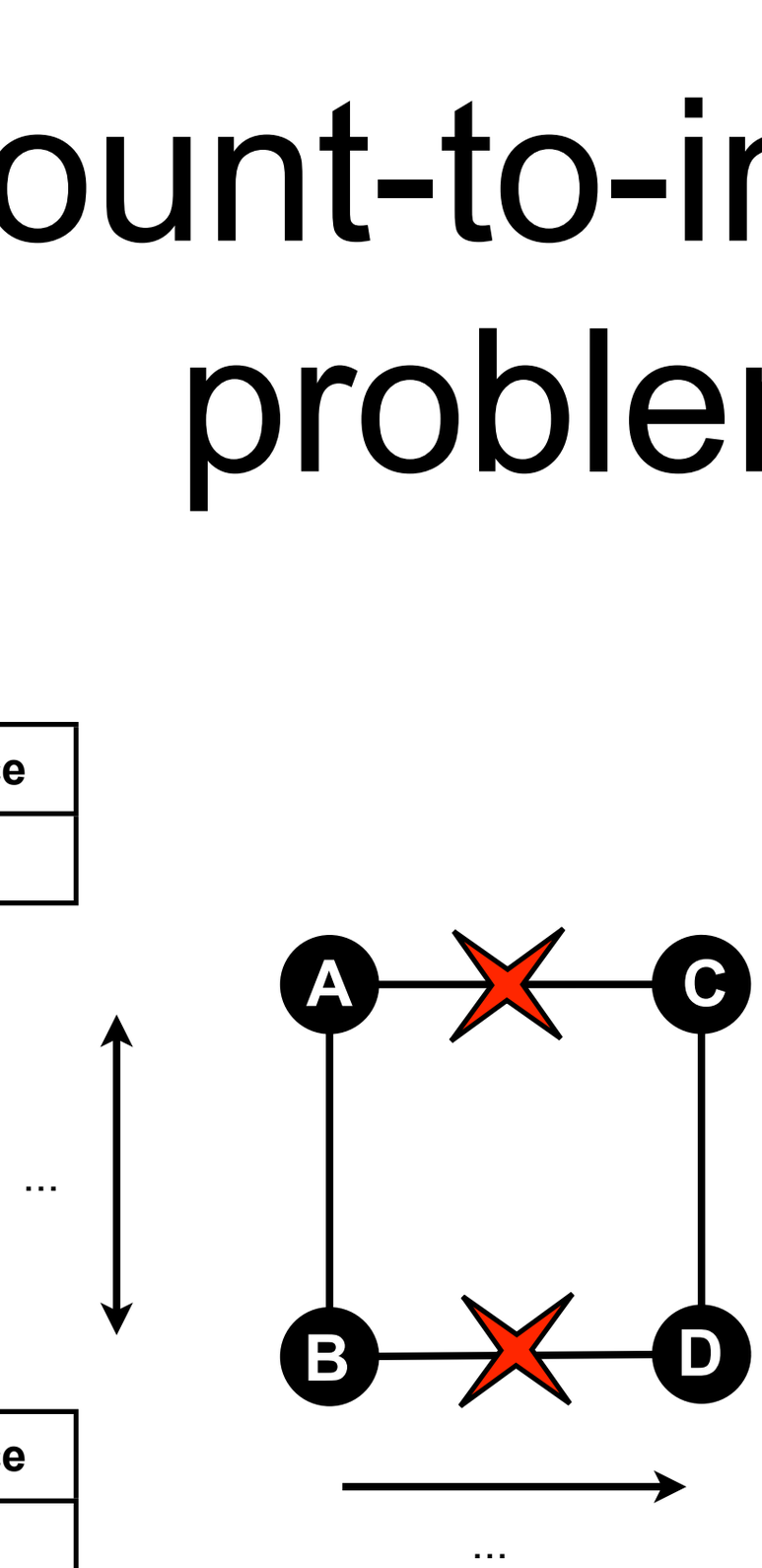
Destination	Distance
C	7



Destination	Distance
C	6

Count-to-infinity problem

Destination	Distance
C	∞

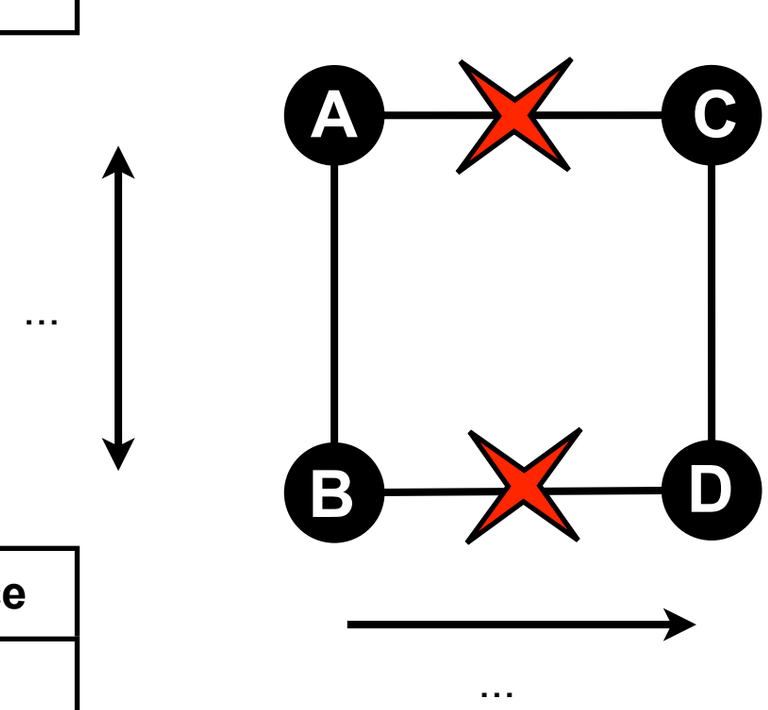


Destination	Distance
C	∞

Count-to-infinity problem

- ∞ = 16 in RIP

Destination	Distance
C	∞



Destination	Distance
C	∞

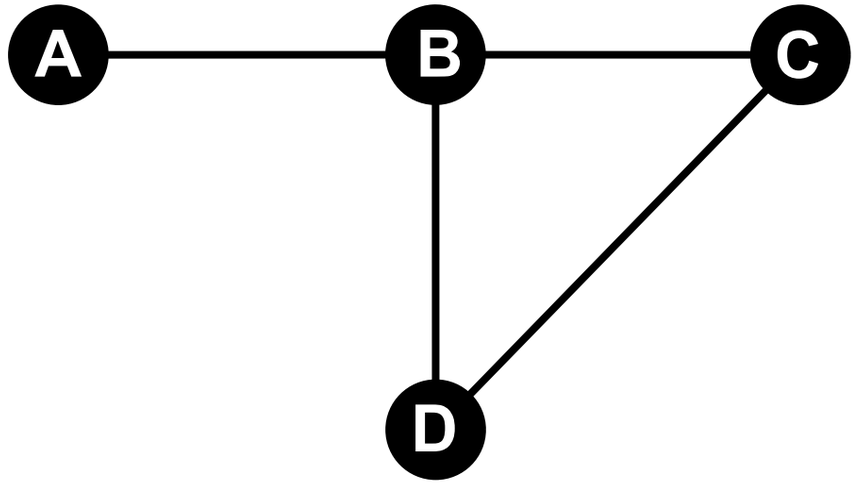
Solving count-to-infinity with split-horizon

- Where does counting to infinity comes from?
  - a router announces on a link a route that it learned via this link
- **Split horizon** to avoid count-to-infinity
  - each router creates a distance vector **for each link**
    - does not announce routes learned on link *i* in the vector sent on link *i*

Split-horizon does not avoid all loops

Destination	Distance
A	1

Destination	Distance
A	2



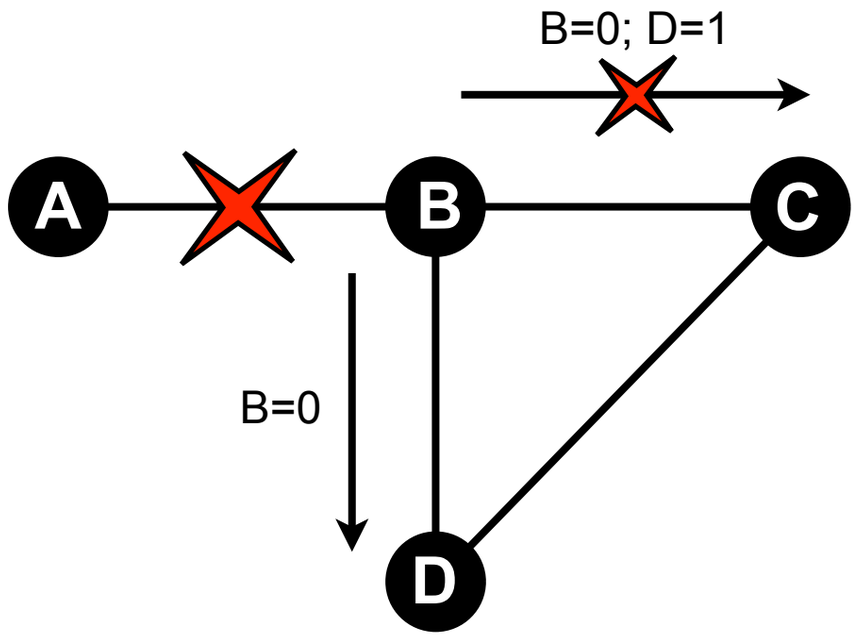
Destination	Distance
A	2

Split-horizon does not avoid all loops

Destination	Distance
A	∞

B=0; D=1

Destination	Distance
A	2

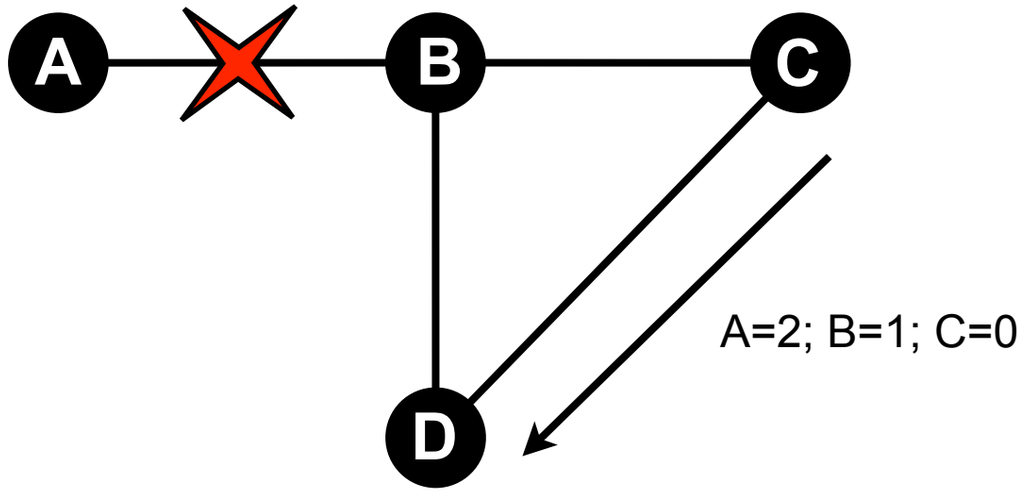


Destination	Distance
A	∞

Split-horizon does not avoid all loops

Destination	Distance
A	∞

Destination	Distance
A	2

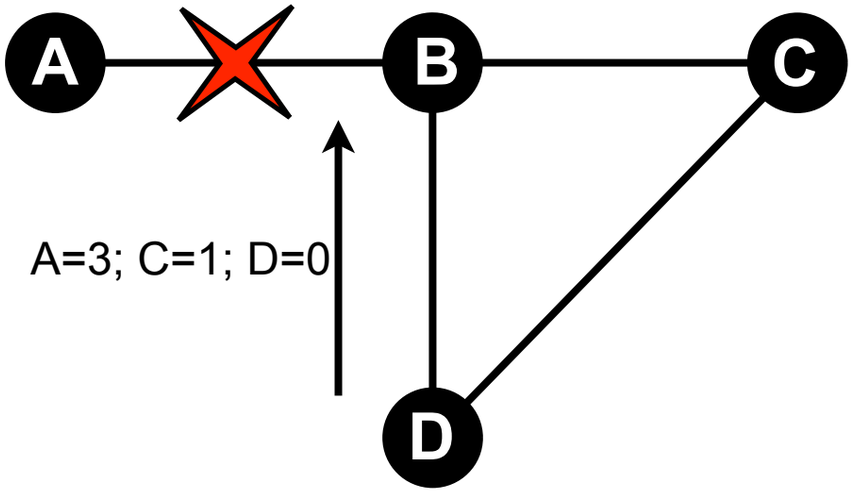


Destination	Distance
A	3

Split-horizon does not avoid all loops

Destination	Distance
A	∞

Destination	Distance
A	2



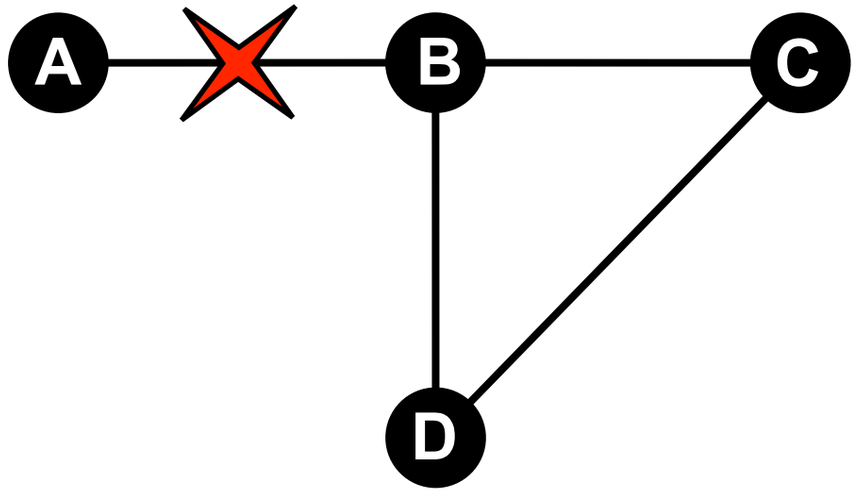
Destination	Distance
A	3

Split-horizon does not avoid all loops

Destination	Distance
A	4

A=3; B=0; D=1

Destination	Distance
A	2



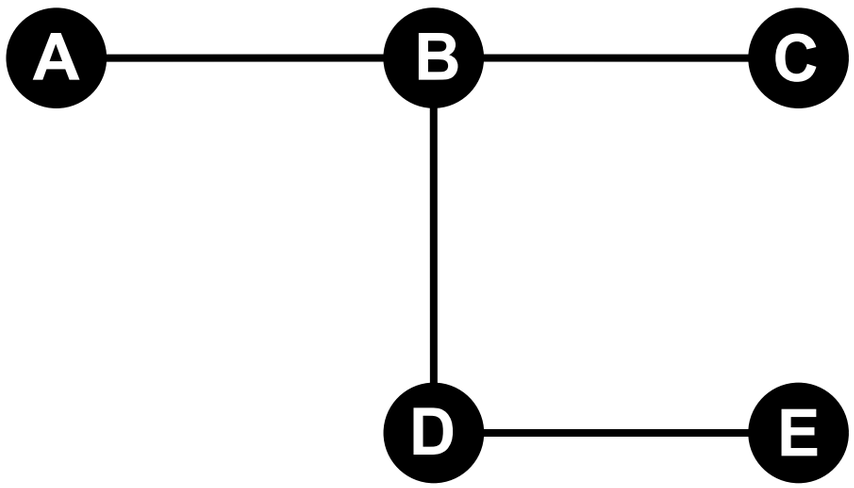
Destination	Distance
A	3

Dealing with the problem

- Triggered updates
  - advertise vectors only upon failure, not periodically
- Split-horizon with poisoning
  - instead of not advertising a route on the link on which it has been learned, advertise it with an infinite cost to count faster to infinity
    - faster convergence
- Source tracing
  - routes are annotated with the router that precedes the originator router
- Hold time
  - withdrawals routes are not propagated immediately
    - works most of the time but increases convergence time
- Path vector
  - the path to reach the destination is included in the distance vector
  - loops are detected at the cost of larger routing table and extra control plane overhead
- Distributed Update Algorithm (DUAL)
  - blindly update a route only if cost decreases
  - if cost increases, freeze the routing table and ensure that every router is aware of the change before updating to the new route

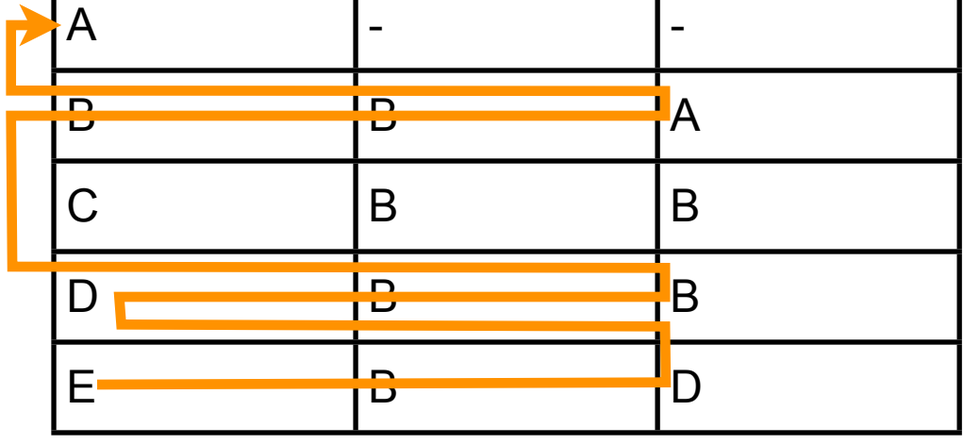
Source tracing example

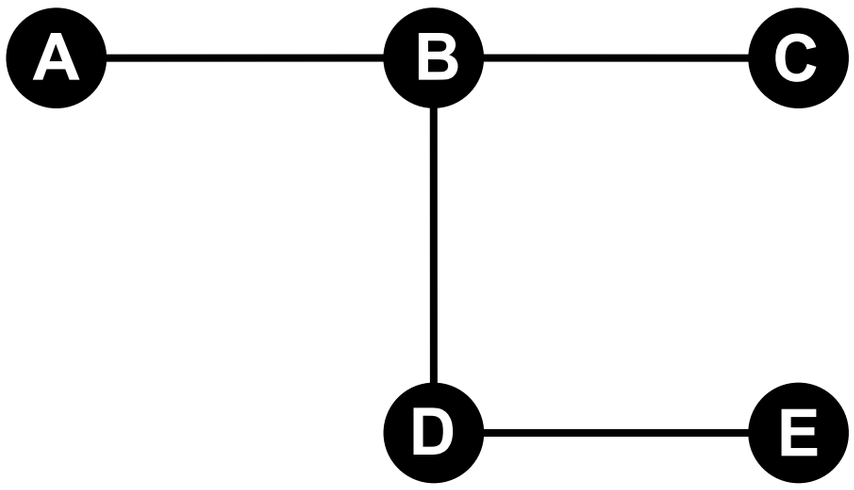
Destination	Next	Predecessor
A	-	-
B	B	A
C	B	B
D	B	B
E	B	D



Source tracing example

Destination	Next	Predecessor
A	-	-
B	B	A
C	B	B
D	B	B
E	B	D





Path to E: A,B,D,E

Distributed Update Algorithm (DUAL) [G89]

- Objective
  - remove transient loops in DV and LS routing protocols
  - always use routes of minimum cost
- Relies on the principle that it is impossible to create a loop if a shortest path is selected

[G89] J. J. Garcia-Luna-Aceves. 1989. A unified approach to loop-free routing using distance vectors or link states. In *Symposium proceedings on Communications architectures & protocols* (SIGCOMM '89). 212-223.

slides inspired by http://kfall.net/ucbpage/EE122/lec17/sld002.htm

DUAL (contd.)

- Each router maintains two tables: L and D
  - for each neighbor *n*, L[*n*] is the cost of the link to *n*
    - local configuration
  - for each neighbor *n* and each destination *d*, D[*n*, *d*] is the distance for *n* to reach *d*
    - constructed with the distance vector received from the neighbors

DUAL (contd.)

- For each destination *d*, a router selects a neighbor *n* that minimizes the objective function
  - cost(*d*) = L[*n*] + D[*n*, *d*]
- Two cases are possible upon update
  - cost(*d*)' < cost(*d*)
  - cost(*d*)' > cost(*d*)

DUAL (contd.)

- cost'(*d*) = L[*m*] + D[*m*, *d*]
- cost(*d*) = L[*n*] + D[*n*, *d*]
- if cost'(*d*) < cost(*d*) then use *m* instead of *n* as next-hop to reach *d* as no loop can be created by reducing the cost

DUAL (contd.)

- if cost'(*d*) > cost(*d*) then start special action by looking for acceptable neighbors *a*
  - *a* is acceptable if D[*a*, *d*] < cost(*d*) (i.e., cost before the update)
  - if the set of acceptable neighbors is not empty then
    - select a neighbors *a* that minimizes
      - L[*a*] + D[*a*, *d*]
  - otherwise, start a diffusion process
    - freeze update of route for *d* during the process
      - it may cause black holes, but prevent loops

Diffusion process

- Send a query message to every neighbor *n* except the neighbor *u* that caused the update
  - the query contains D and the frozen route (L[*u*] + D[*u*, *d*])
  - the query is for getting D[*n*, *d*] of every neighbor *n*

Diffusion process (contd.)

- Upon query reception, routers in passive state simply reply with their cost to *d*
  - a router is in passive mode its route to *d* does not change because of the query (i.e., not using the query sender)

Diffusion process (contd.)

- If the route to *d* changes because of a query, the router passes in active mode,
  - propagates the query to all its neighbors (except the one it received the query from)
  - once the routers received a reply from all its neighbors,
    - it computes the new best route (i.e., min(L[*m*] + D[*m*, *d*])),
    - returns in passive mode,
    - and replies to the query it received
- The reply eventually arrives the router that initiated the diffusion process, ending so the diffusion process
  - every router has a route to *d*

What about link sate?

- Forwarding loops are also possible in link state routing protocols
- for example when the view of the network is not the same on every router

Conditions for a forwarding loop to happen

- A forwarding loop may happen during convergence when *A* reaches *B* via *C* before the change but *C* reaches *B* via *A* after the change
- this might happen because of the delay of updating the FIB

Loop free convergence

- In a stable network, incrementing the metric of a link A-B by one leads to a loop-free convergence process [FSB07]
- Potentially many steps from initial metric m_i to target metric m_t
  - can be improved by computing the key metric m in [m_i, m_t] that ensures the use of a path not via A-B
  - see details in [FSB07]

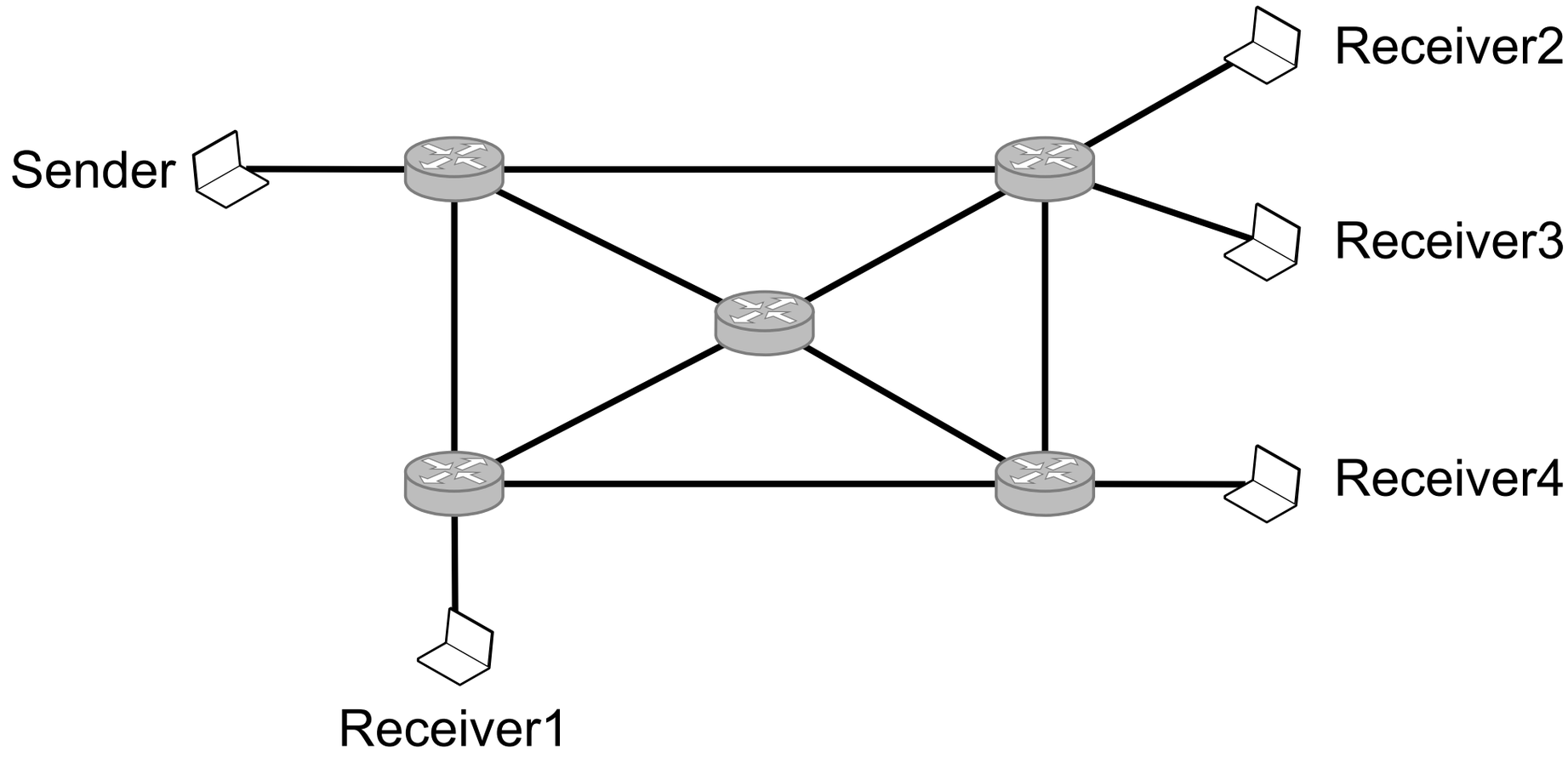
[FSB07] P. François, M. Shand and O. Bonaventure. Disruption-free topology reconfiguration in OSPF Networks. IEEE INFOCOM, 2007

Multicast

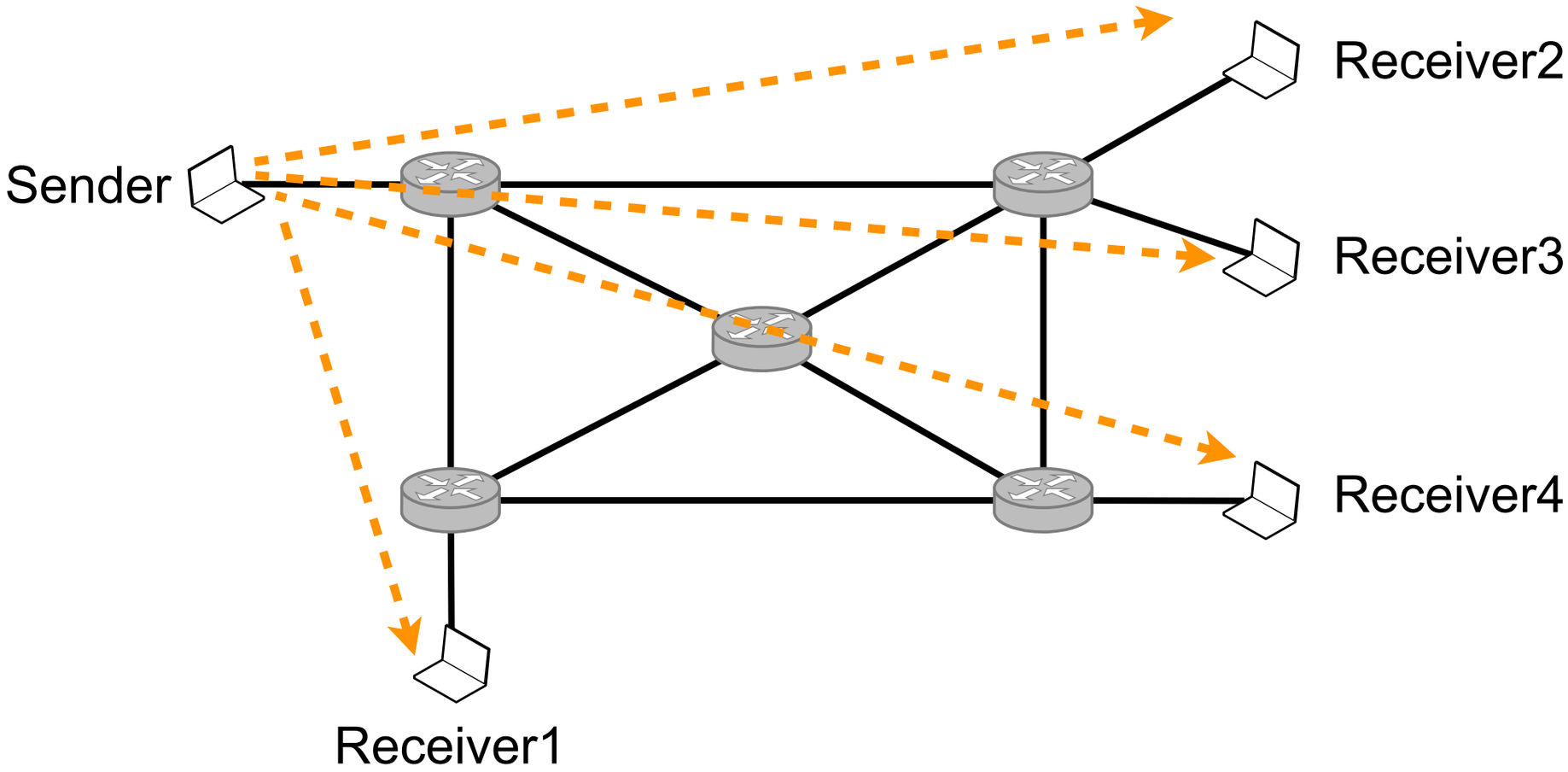
Motivation

- Unicast
  - packets are sent from one source to one destination
- How to transmit the same information to a set of *N* receivers?
  - source-based: the source sends *N* copies, one per receives
    - simple to implement but not efficient in term of bandwidth usage
  - network-based: the source sends one copy of the information and the network efficiently distribute the information to the receivers
    - more efficient, but requires the network to cooperate

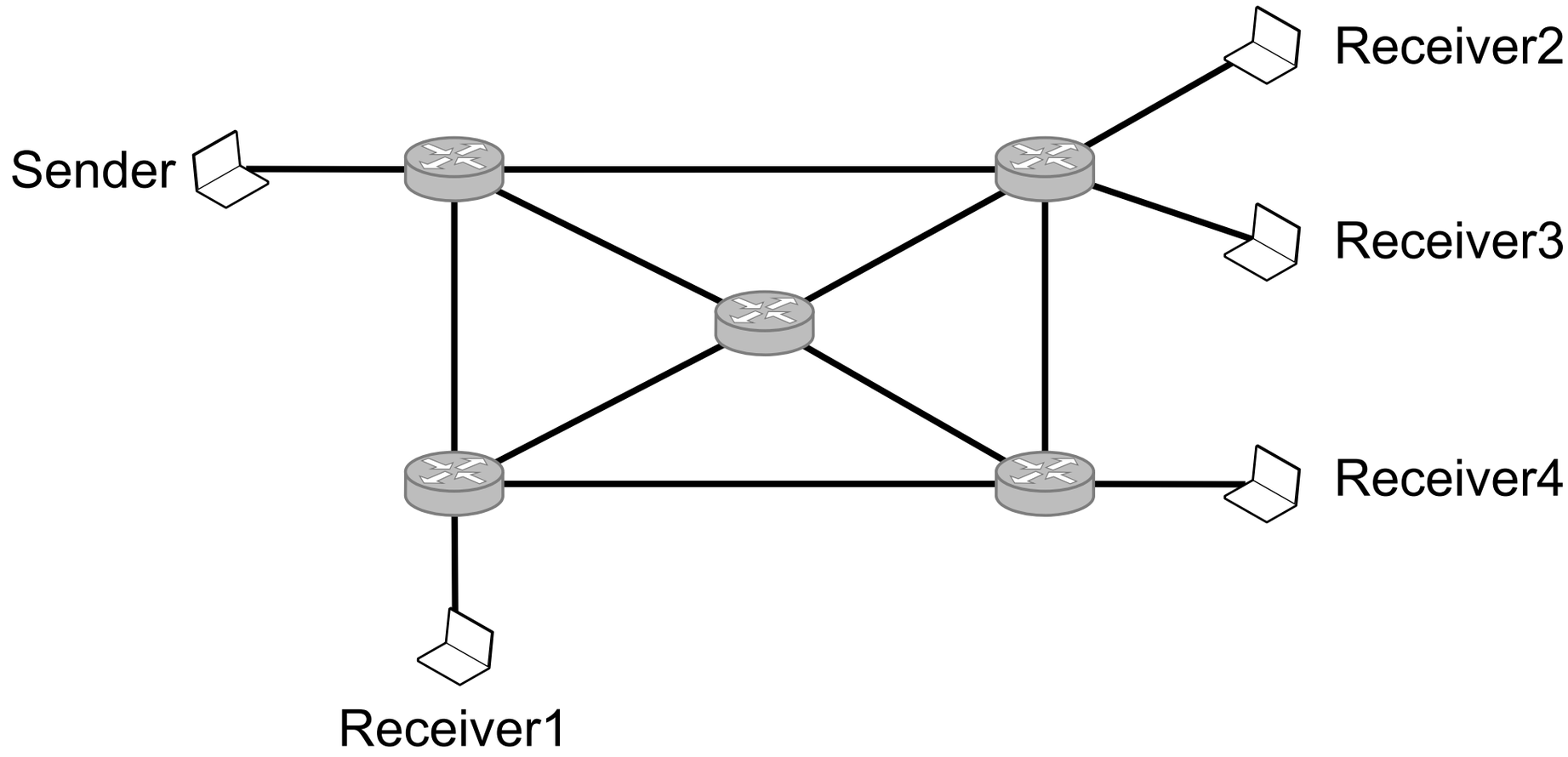
Principle of the solution



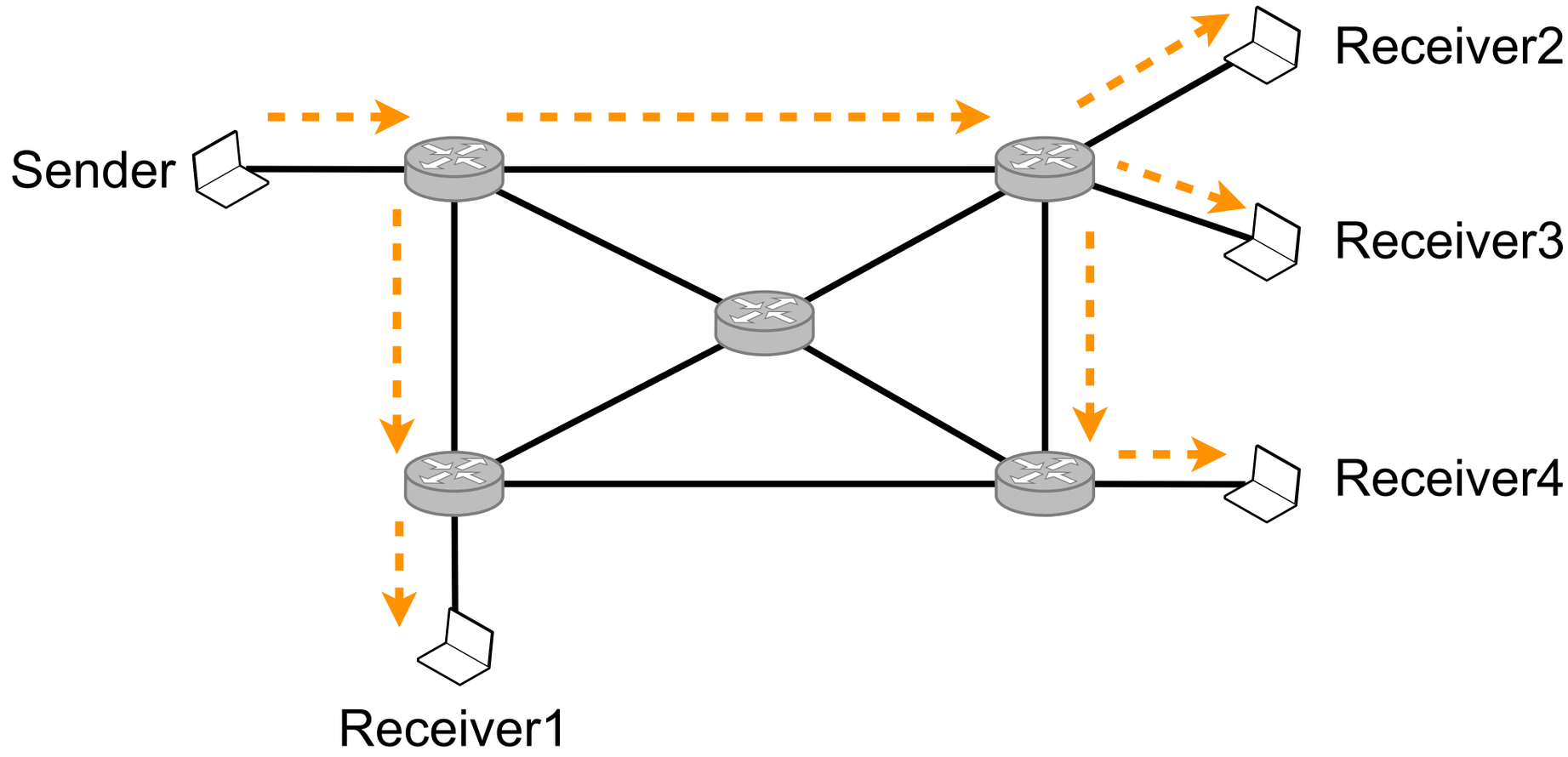
Principle of the solution



Principle of the solution



Principle of the solution



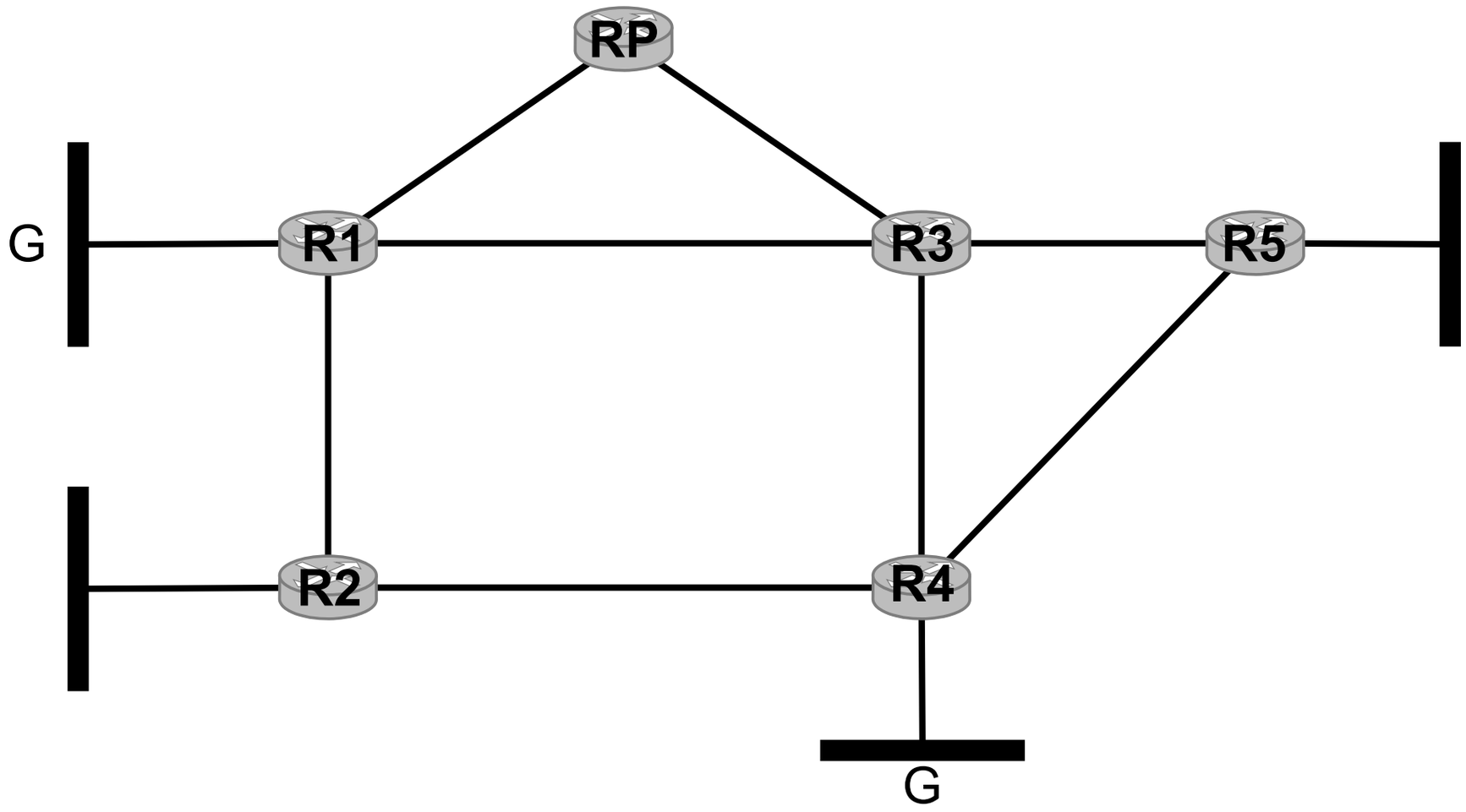
Network-based multicast

- The sender sends packets to a group of receivers identified by a multicast group address
- Receivers indicate their interest in receiving packets for a multicast group
- Routers know the existence of local receivers for every group
  - distribute this information to other routers
  - decide on which links transmit multicast packets
    - constraint 1: every receiver of a group must receive all the packets for that group (for the time they are in the group)
    - constraint 2: packets should not be sent more than once on a link
- Multicast packets are disseminated with a multicast tree

Unidirectional shared tree (e.g., PIM-SM)

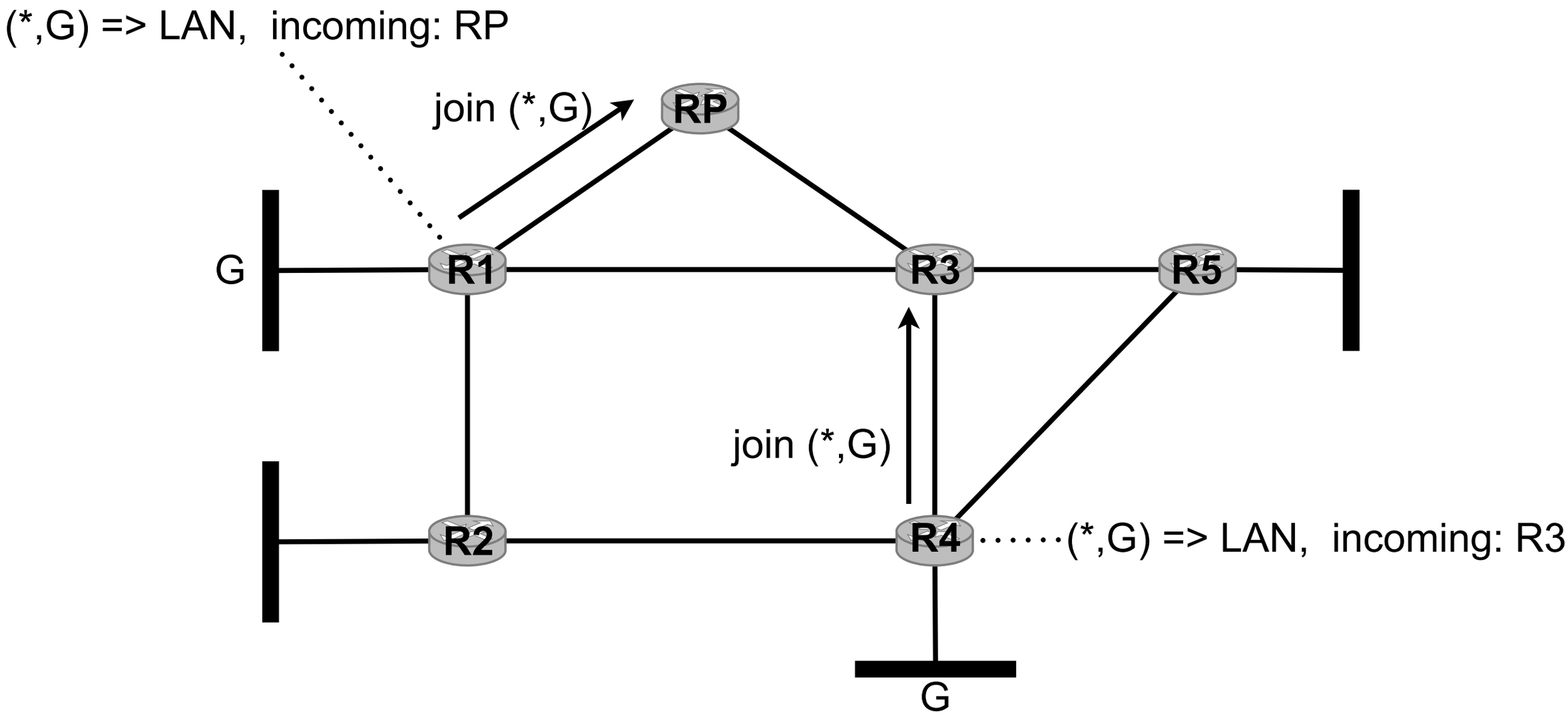
- Principles of the unidirectional shared tree
  - one router plays the role of **R**endez-vous **P**oint (RP)
  - every router knows the RP address
  - RP is the root of a shared (*,G) tree
  - multicast packets sent by hosts are redirected to the RP
    - the RP distributes the packets over the (*,G) tree
  - receivers dynamically join the shared (*,G) tree
- Group membership learned with the Internet Group Management Protocol (IGMP)

A (over) simplified shared tree



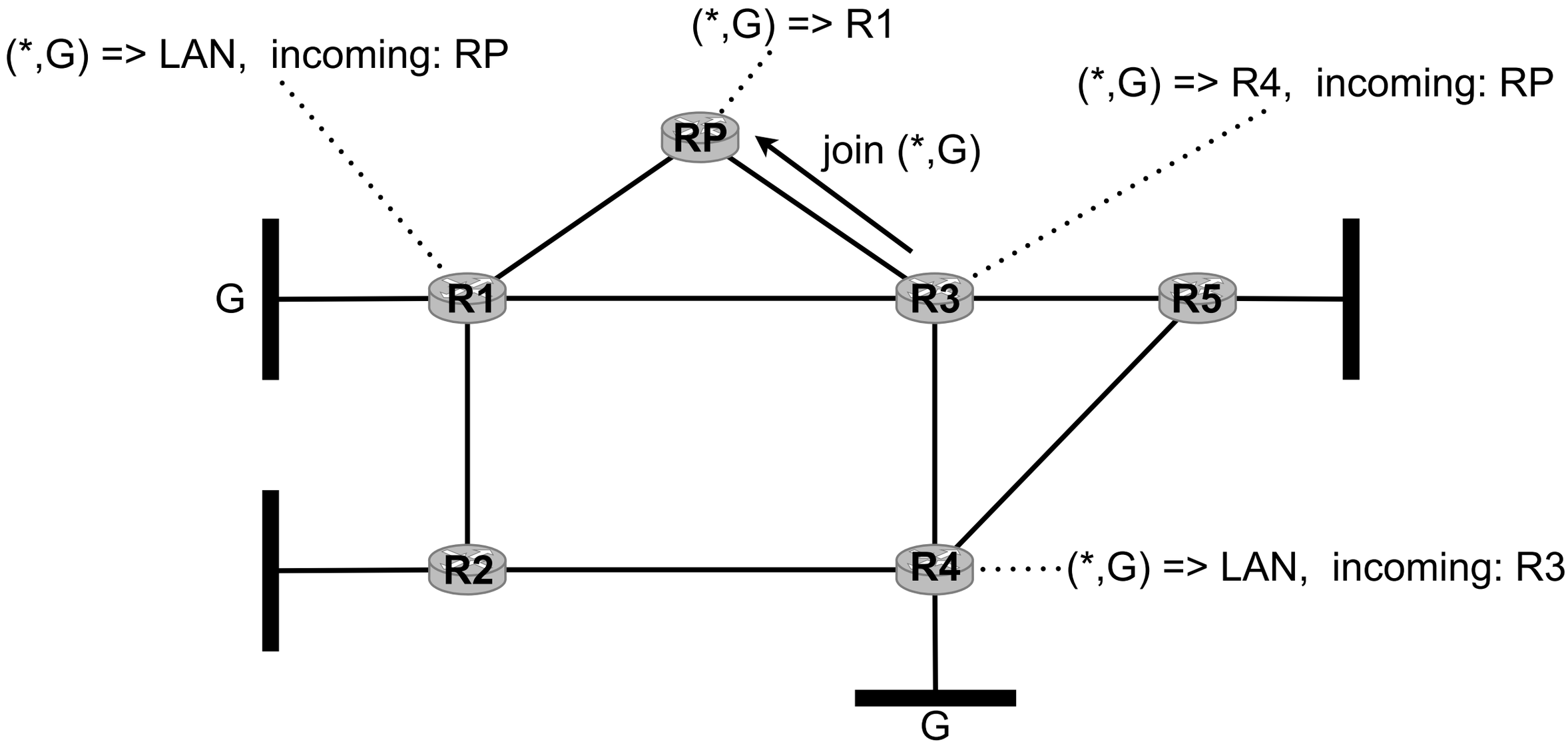
A router attached to a group sends a join message to the Rendez-vous Point to build the shared tree

A (over) simplified shared tree



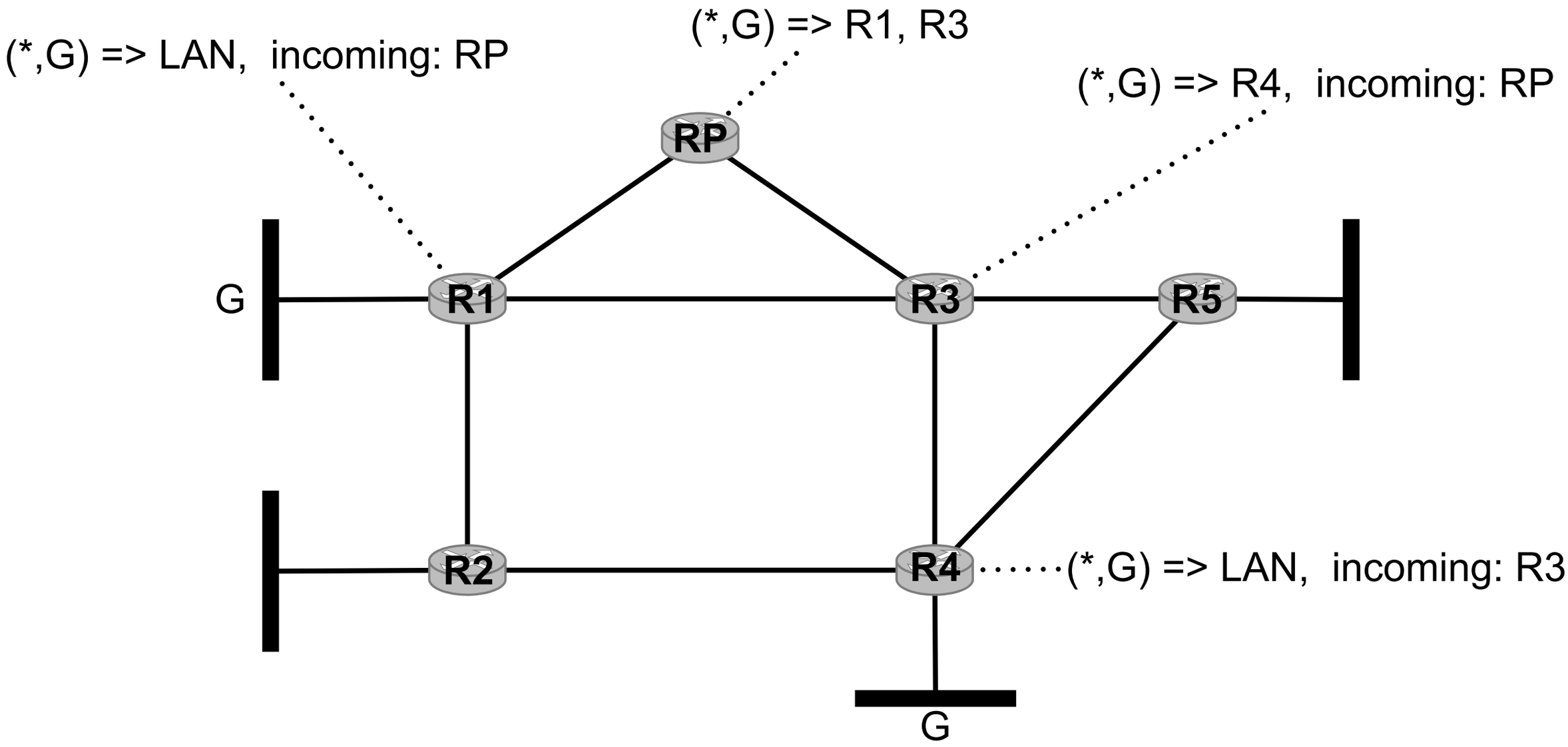
A router attached to a group sends a join message to the Rendez-vous Point to build the shared tree

A (over) simplified shared tree



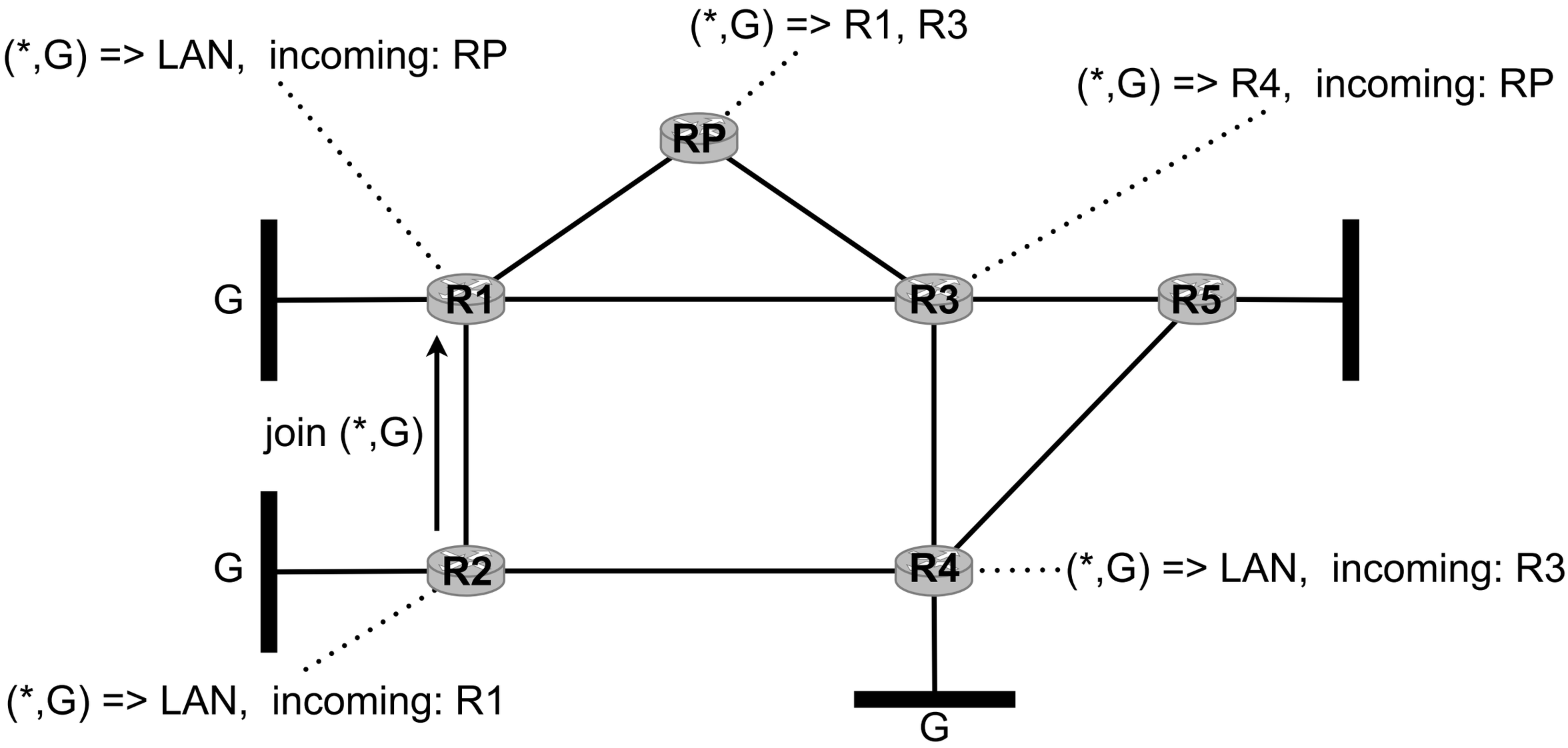
A router attached to a group sends a join message to the Rendez-vous Point to build the shared tree

A (over) simplified shared tree



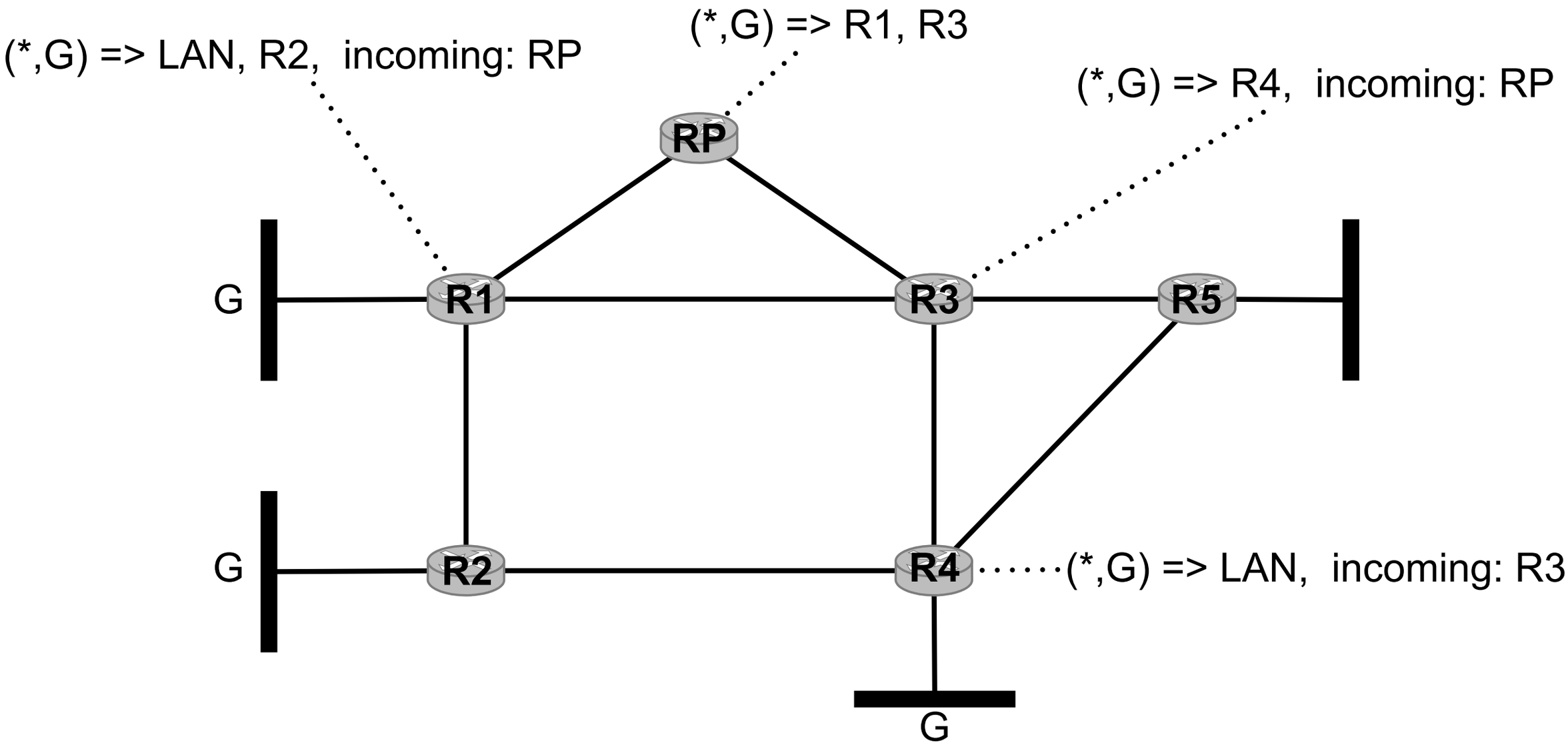
A router attached to a group sends a join message to the Rendez-vous Point to build the shared tree

A (over) simplified shared tree



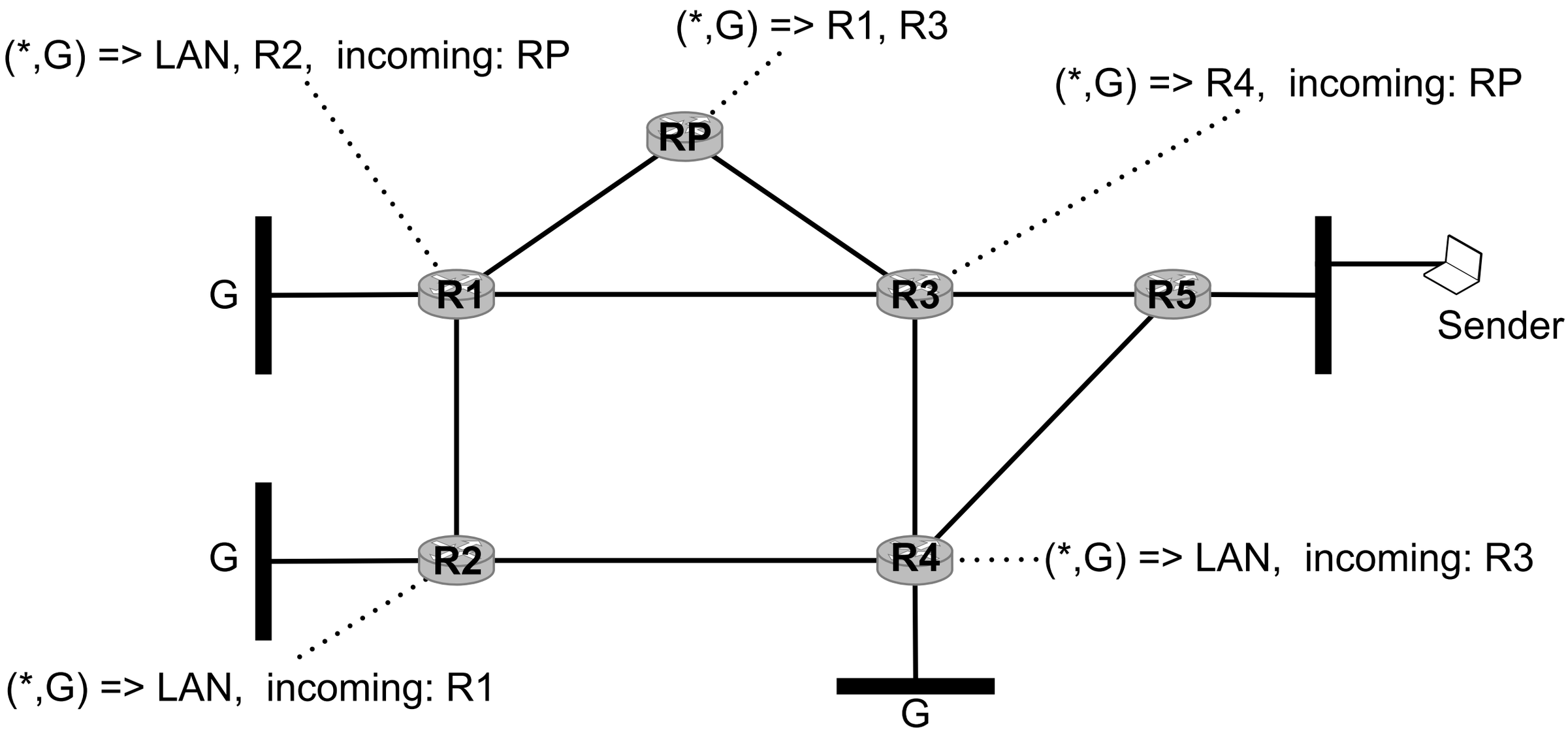
A router attached to a group sends a join message to the Rendez-vous Point to build the shared tree

A (over) simplified shared tree



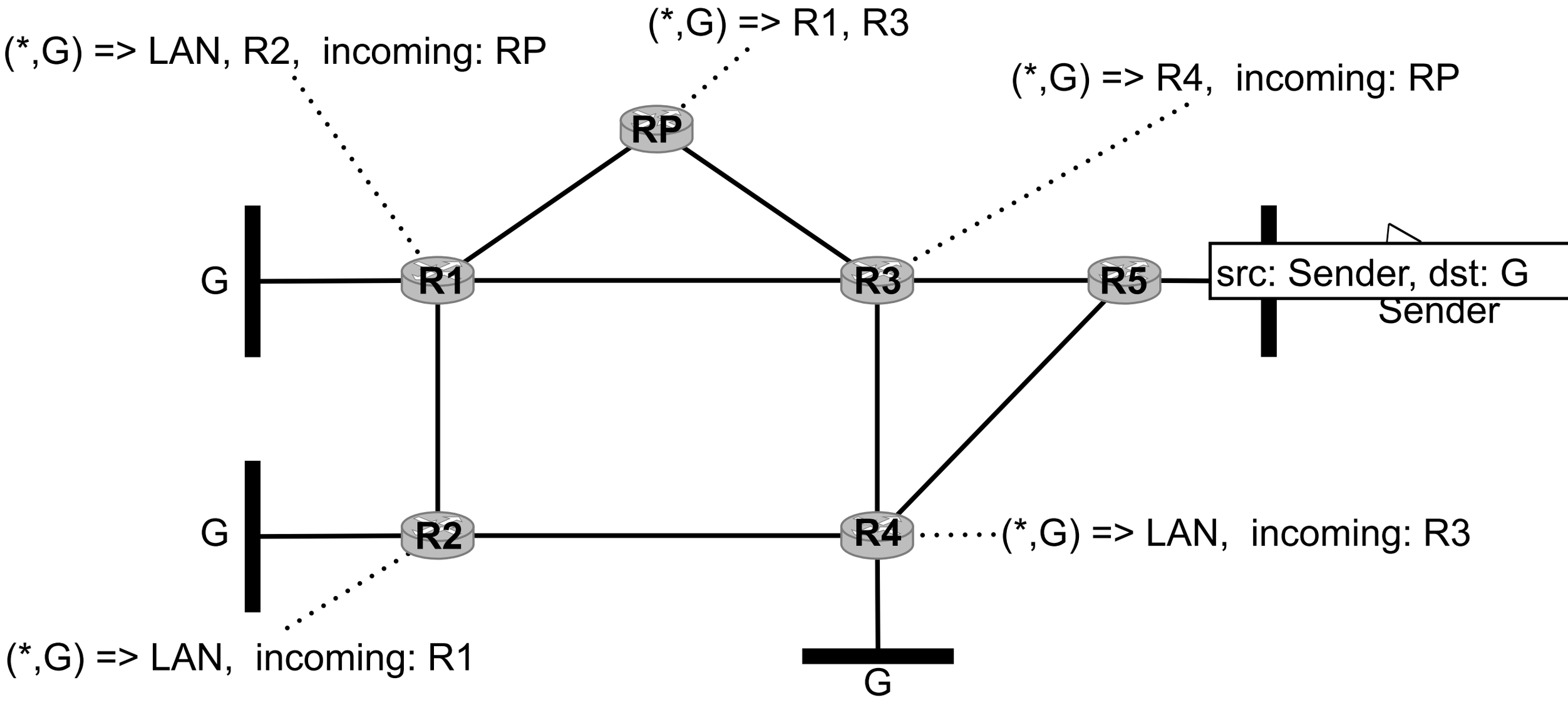
A router attached to a group sends a join message to the Rendez-vous Point to build the shared tree

A (over) simplified shared tree



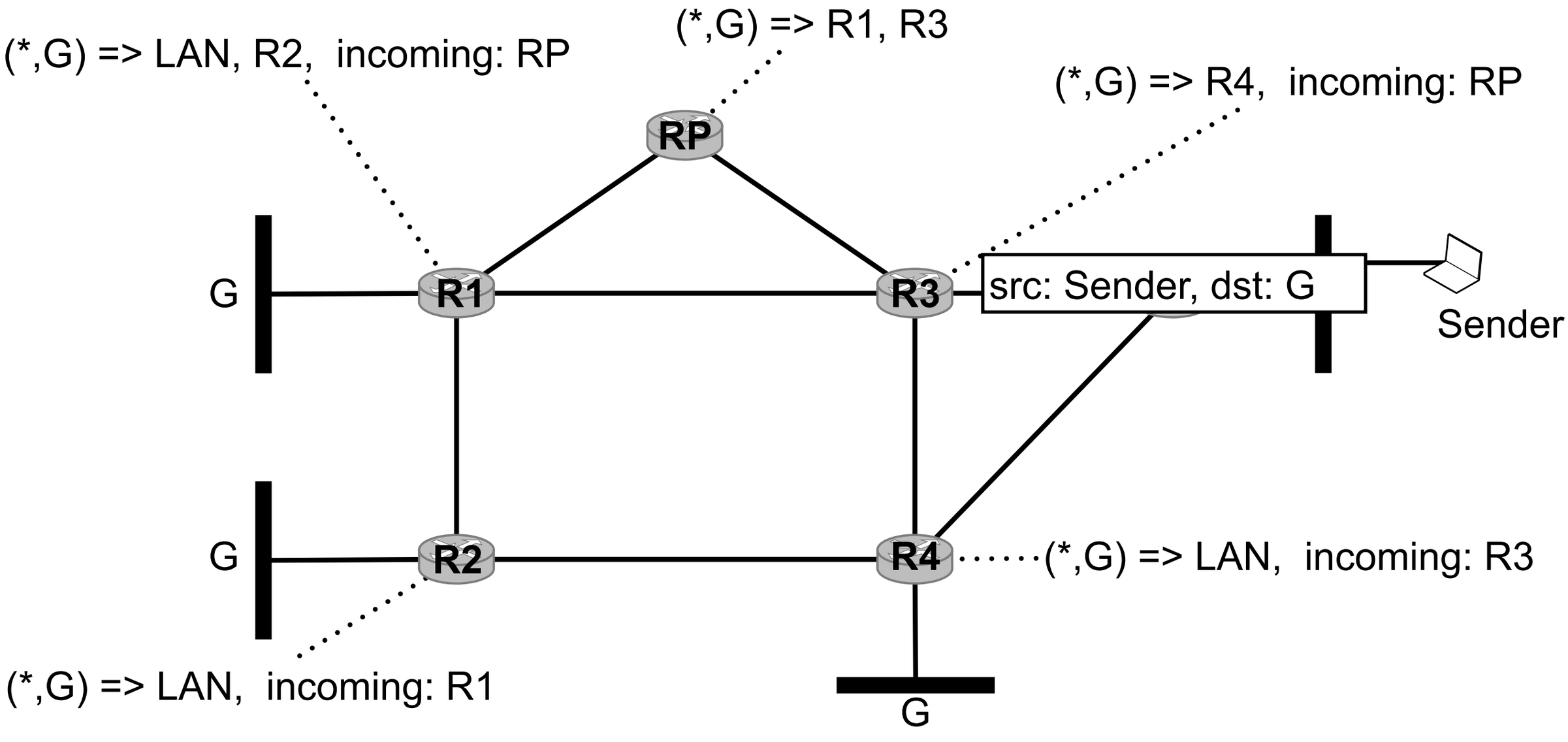
- R5 has no state for G, packet toward G are intercepted and sent to RP in a *register* message
- Packet is decapsulated and sent on (*,G)
- Optionally, RP can create a source tree to receive G packets natively

A (over) simplified shared tree



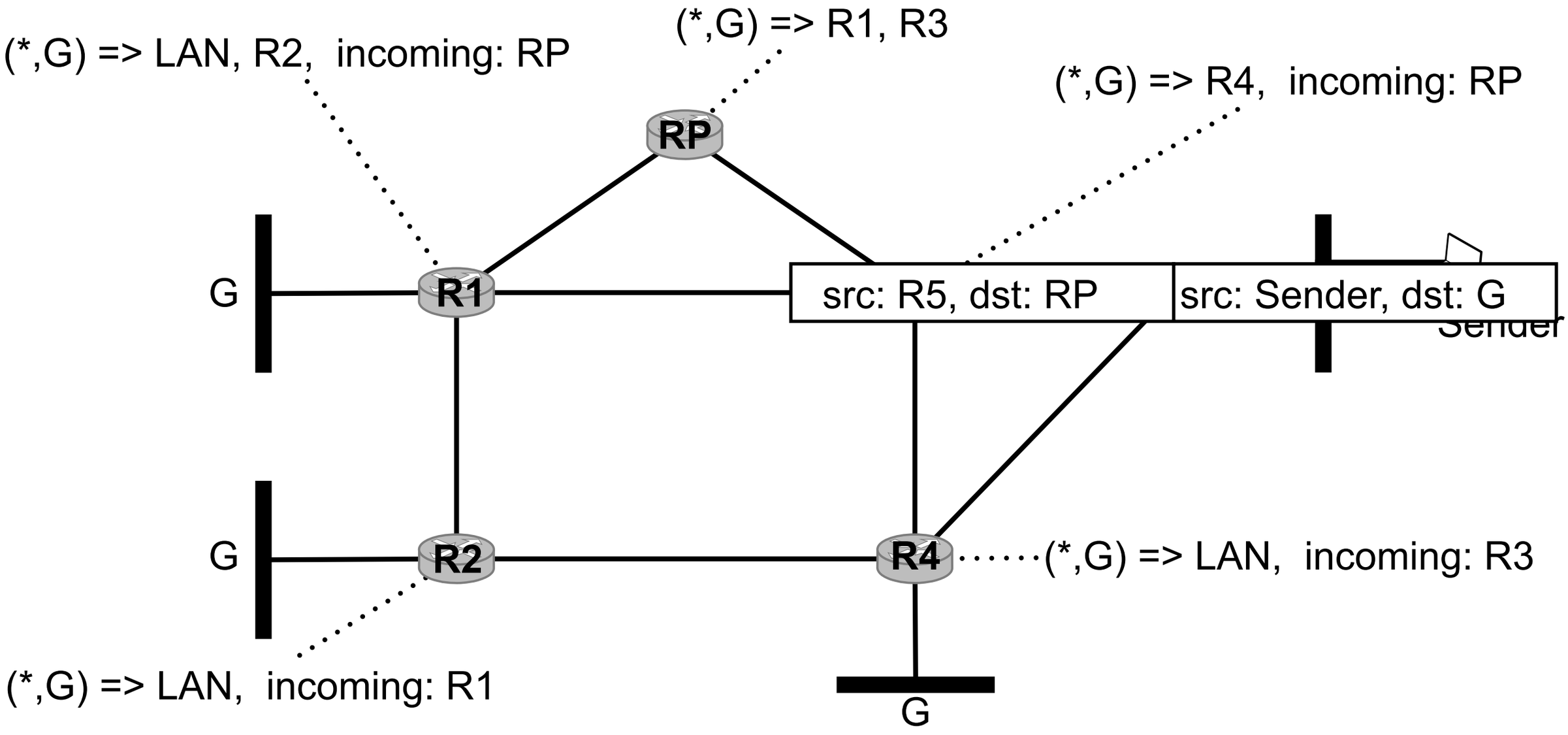
- R5 has no state for G, packet toward G are intercepted and sent to RP in a *register* message
- Packet is decapsulated and sent on (*,G)
- Optionally, RP can create a source tree to receive G packets natively

A (over) simplified shared tree



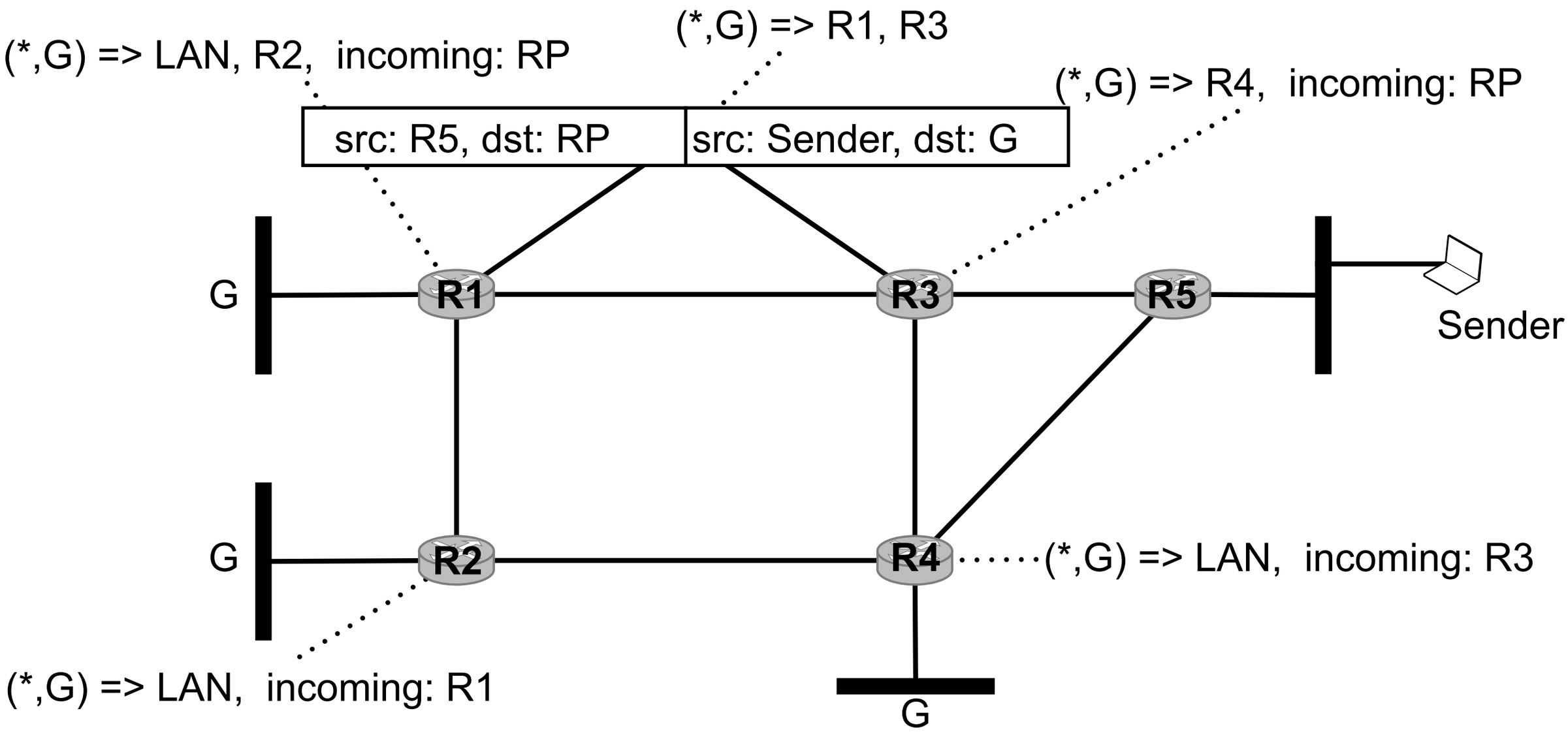
- R5 has no state for G, packet toward G are intercepted and sent to RP in a *register* message
- Packet is decapsulated and sent on (*,G)
- Optionally, RP can create a source tree to receive G packets natively

A (over) simplified shared tree



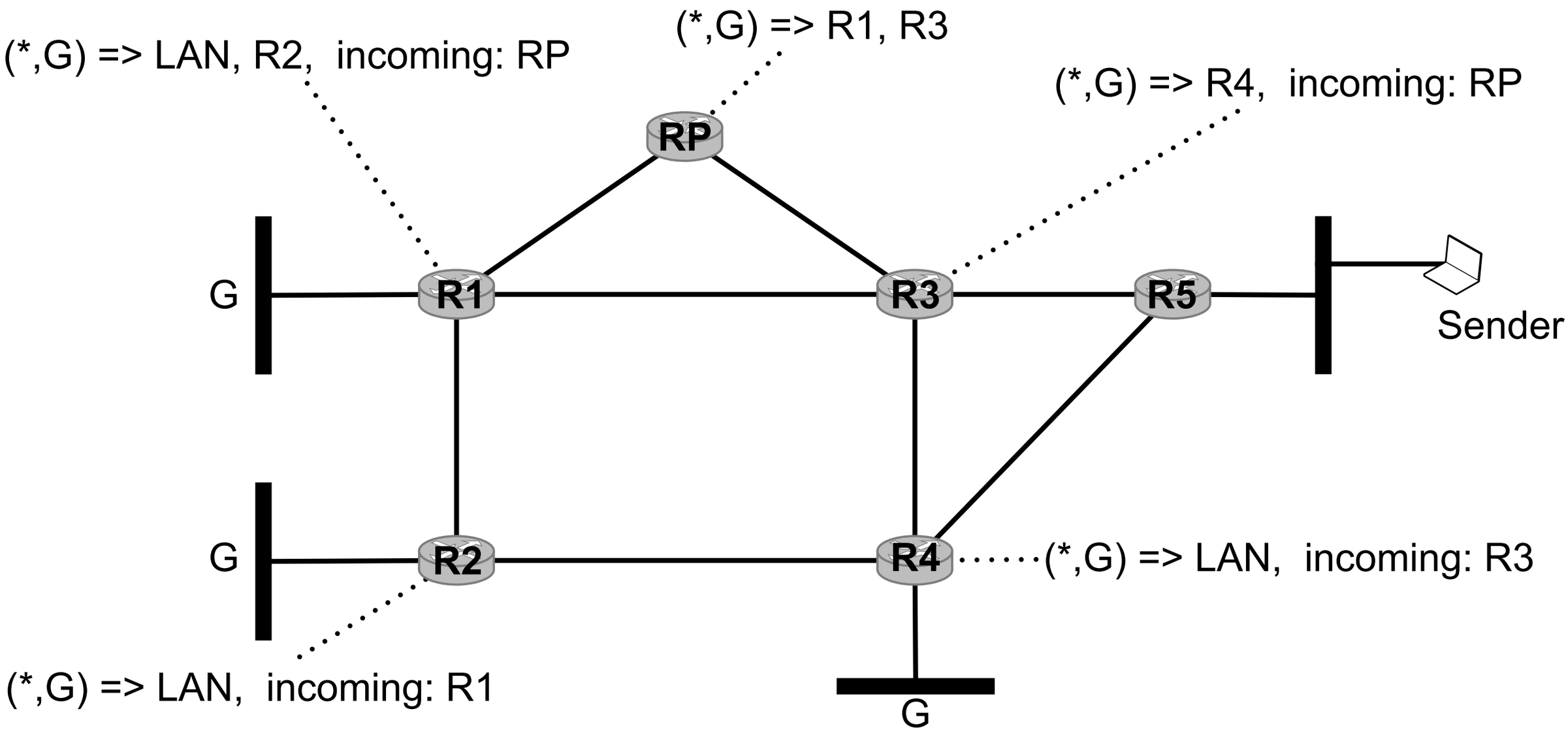
- R5 has no state for G, packet toward G are intercepted and sent to RP in a *register* message
- Packet is decapsulated and sent on (*,G)
- Optionally, RP can create a source tree to receive G packets natively

A (over) simplified shared tree



- R5 has no state for G, packet toward G are intercepted and sent to RP in a *register* message
- Packet is decapsulated and sent on (*,G)
- Optionally, RP can create a source tree to receive G packets natively

A (over) simplified shared tree



- R5 has no state for G, packet toward G are intercepted and sent to RP in a *register* message
- Packet is decapsulated and sent on (*,G)
- Optionally, RP can create a source tree to receive G packets natively

Network-based multicast

- Pros
  - more efficient utilization of the bandwidth
  - no unnecessary packet duplication
- Cons
  - routers must implement a multicast protocol
    - implementation and configuration cost
  - routers must maintain state
    - memory cost, risk upon failure
  - multicast packet require special treatments
    - processing cost
- Bandwidth gain should always be compared to router costs

Operational corner

inspired from: http://bgp.nu/~dward/IPFRR/IPFRR_overview_NANOG.pdf

Recovery to failure must be fast

- How fast?
  - sub-second: typical requirement in most IP networks
  - sub-200 ms: normal application are not sensitive to connectivity loss smaller than 200ms
  - sub-50 ms: some specialized IP network have such requirement!

50ms is very small

50ms is very small

- Impossible?

50ms is very small

- Impossible?

1. detect the failure (~milliseconds)

50ms is very small

- Impossible?

1. detect the failure (~milliseconds)
2. report failure (~milliseconds)

50ms is very small

- Impossible?

1. detect the failure (~milliseconds)
2. report failure (~milliseconds)
3. generate and flood LSP (~10s of milliseconds)

50ms is very small

- Impossible?

1. detect the failure (~milliseconds)
2. report failure (~milliseconds)
3. generate and flood LSP (~10s of milliseconds)
4. recompute shortest path tree (~10s of milliseconds)

50ms is very small

- Impossible?

1. detect the failure (~milliseconds)
2. report failure (~milliseconds)
3. generate and flood LSP (~10s of milliseconds)
4. recompute shortest path tree (~10s of milliseconds)
5. installing the new routes (~100s of milliseconds)

50ms is very small

- Impossible?
  1. detect the failure (~milliseconds)
  2. report failure (~milliseconds)
  3. generate and flood LSP (~10s of milliseconds)
  4. recompute shortest path tree (~10s of milliseconds)
  5. installing the new routes (~100s of milliseconds)
- real example ~280ms for 1500 nodes and 2500 prefixes

50ms is very small

- Impossible?
  1. detect the failure (~milliseconds)
  2. report failure (~milliseconds)
  3. generate and flood LSP (~10s of milliseconds)
  4. recompute shortest path tree (~10s of milliseconds)
  5. installing the new routes (~100s of milliseconds)
- real example ~280ms for 1500 nodes and 2500 prefixes

No, computing the SPF is not what takes time!

IP fast reroute to achieve sub-50 ms recovery

- Concept
  - it is too late to compute a new route when a failure is detected
  - react locally upon failure
  - precompute alternative routes before any failure
    - alternative routes must be consistent across the network

Upon failure

- Upon failure, a node:
  1. detects the failure (e.g., no more signal on the link)
  2. uses the pre-computed alternative route

     forwarding is already restored!
  3. generates and floods LSP describing the failure
  4. every node recomputes shortest path tree
  5. installs the new routes
- Disruption does not exceed 50ms = time to detect failure + a few ms to switch to the alternate

Research corner

Imagine that routing is not decentralized anymore, what would it change?

- fill me
- fill me
- fill me

Homework

due date 02/01/2013

For next time

- Form 11 groups
  - each group must have between 2 and 4 members
    - every student must be in a group
- Assign a unique number to each group, from 1 to 11
- Be ready for next time :-D

Multicast

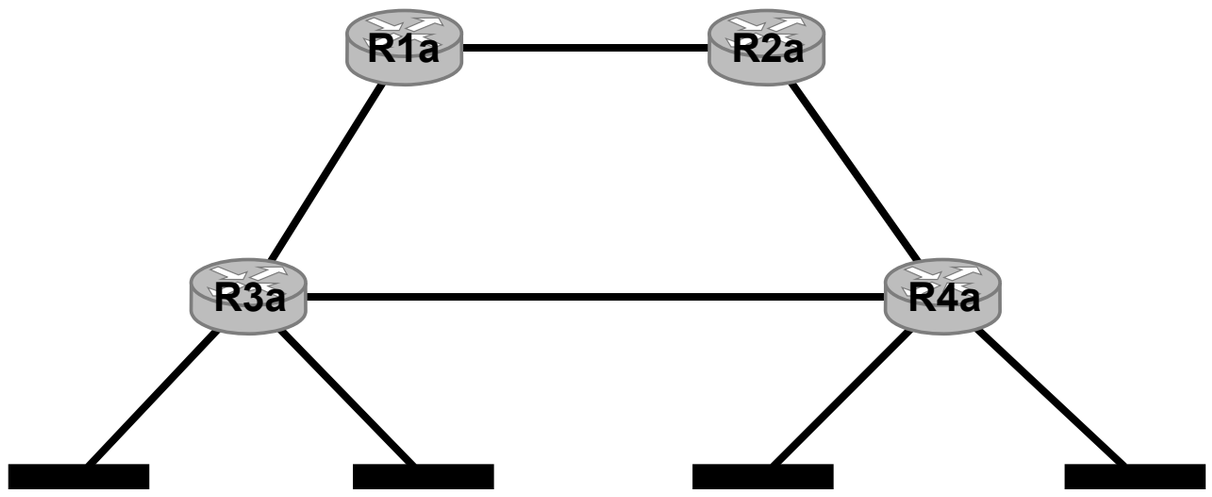
- Skim over Protocol Independent Multicast-Sparse Mode (PIM-SM) specifications [RFC2362]

First lab: OSPF

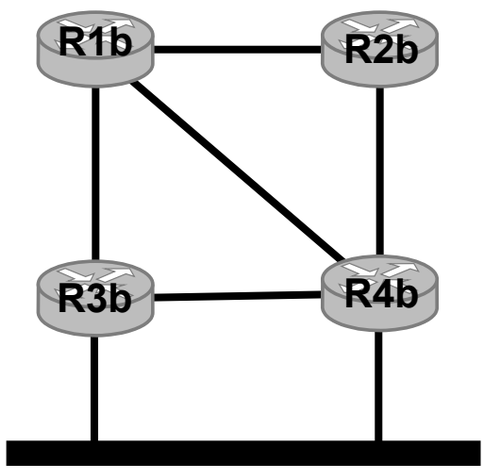
- Configure and test your topology in Dynagen
  - use IOS image c7200-advipservicesk9_li-mz.151-4.XB5 that has been provided during the course (idlepc = 0x6358fb08)
- Design the router' addressing plan by yourself
  - objective: minimize address wastage, addresses and prefixes must belong to 10.0.0.0/8
- Assign link weights by yourself such that the network is optimally used
- Interconnect nodes in the topology with OSPF
- Your assigned topology: md5(first name | last name) % 4
- Send a 2 pages report with your configuration files and tests

Topologies

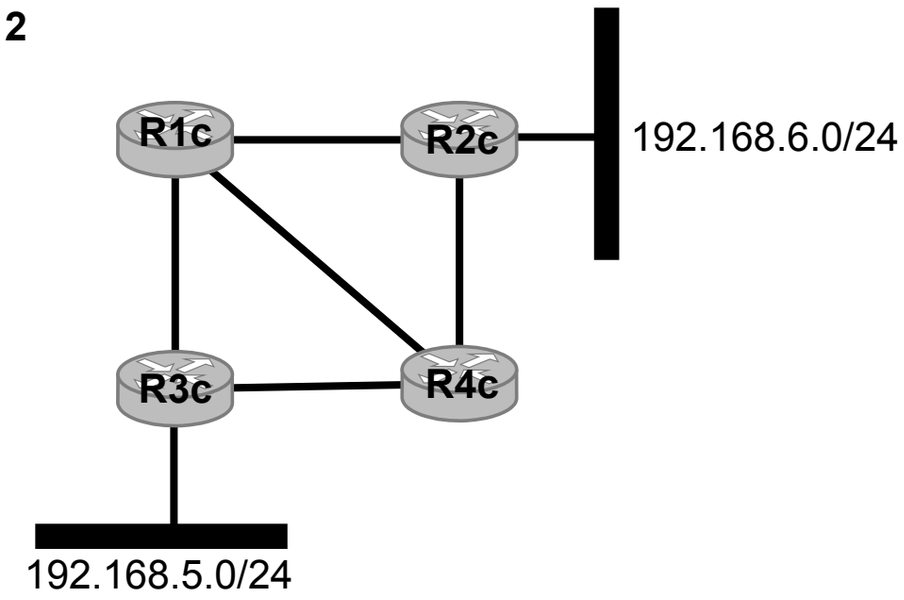
Topo 0



Topo 1



Topo 2



Topo 3