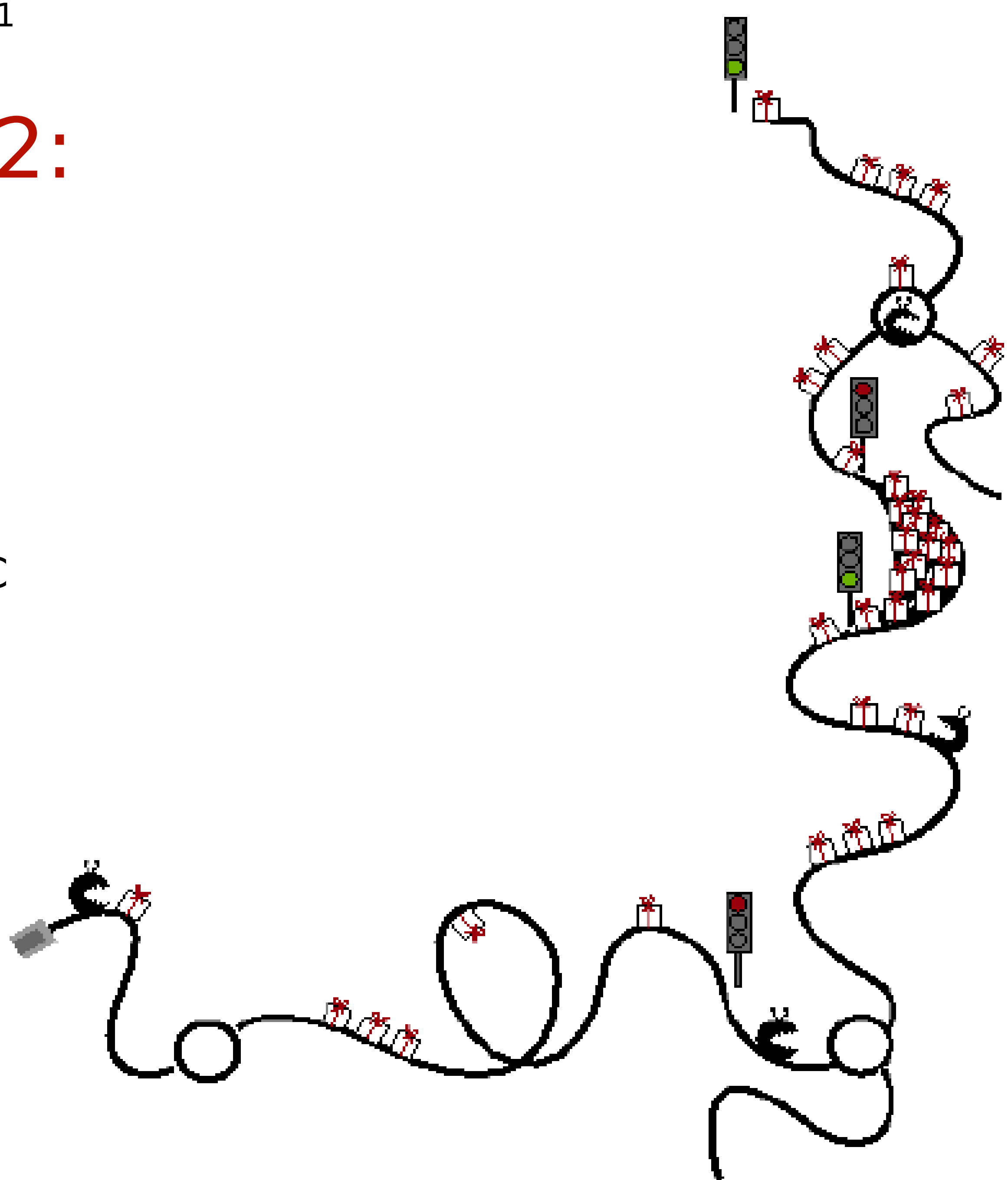
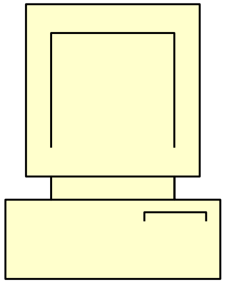
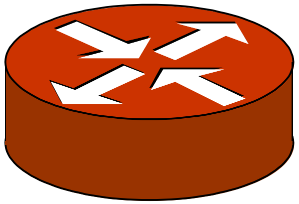
Lecture 11-12: Routing

Olof Hagsand KTH CSC



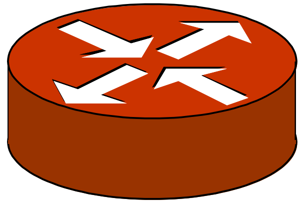
What is a router?





- Host (end-system)
  - One or many network interfaces
  - Cannot forward packets between them
- Router
  - Can forward packets between multiple interfaces
  - Forwarding on Layer 3

What does a router do?



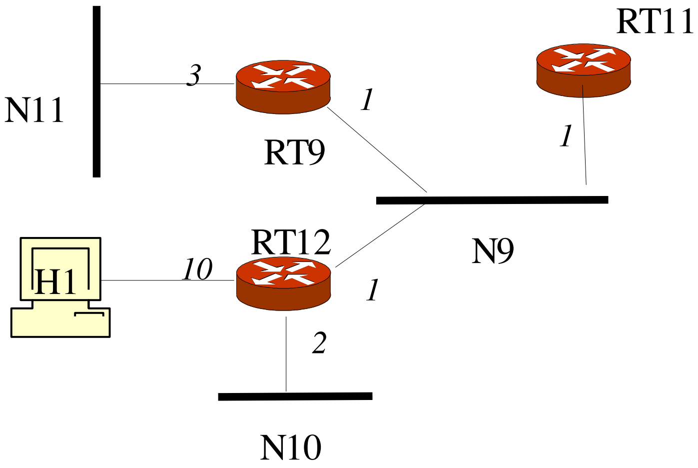
- Packet forwarding
- Not only IPv4:
- IPv6, MPLS, Tunneling,...
  - (But never naming,..)
- Filter traffic
  - Access lists based on src/dst, etc.
- Metering/Shaping/Policing
  - Measuring, forming and dropping traffic
- Compute routes: build forwarding table
- In the ”background”: routing
- In ”real-time”: forwarding

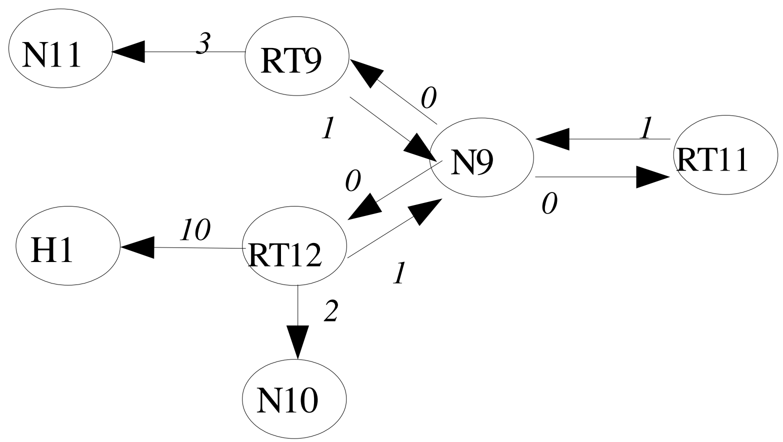
Routing algorithms

- How does a router find a best path?
- Most solutions based on SPF (Shortest Path First) algorithms that are well known in graph theory.
  - Bellman-Ford
  - Dijkstra
- Link-State protocols (OSPF, IS-IS) use Dijkstra
- Distance-Vector protocols (RIP, IGRP, BGP) use Bellman-Ford
- Apart from that, there may also be other algorithms in
  - Multicast routing
  - Ad-hoc routing
  - Sensor networks
  - Delay-tolerant networks

Graphs vs networks

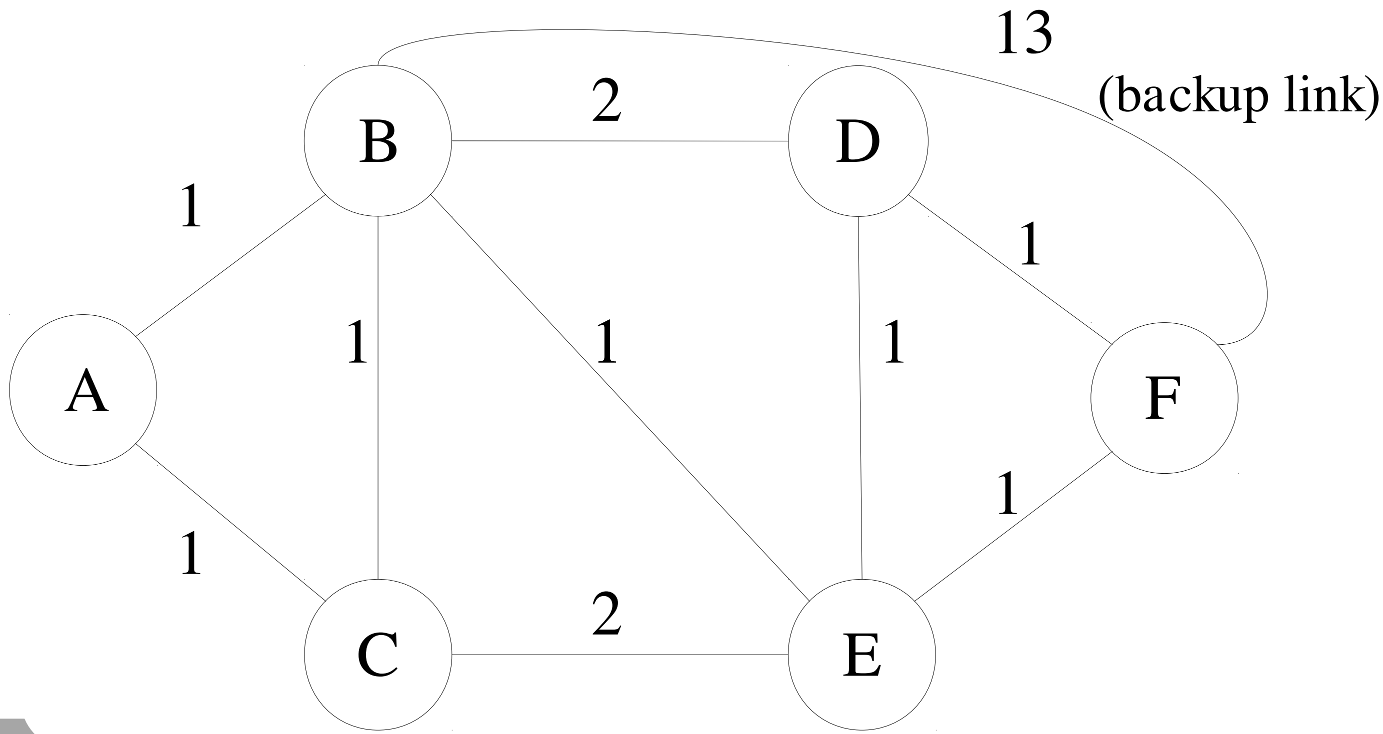
- Algorithms are usually defined on graphs whereas protocols work on networks
- Graphs have nodes and edges whereas networks have interfaces, broadcast links, addresses, hierarchical layering, etc.
- Note the modelling of the broadcast link N9





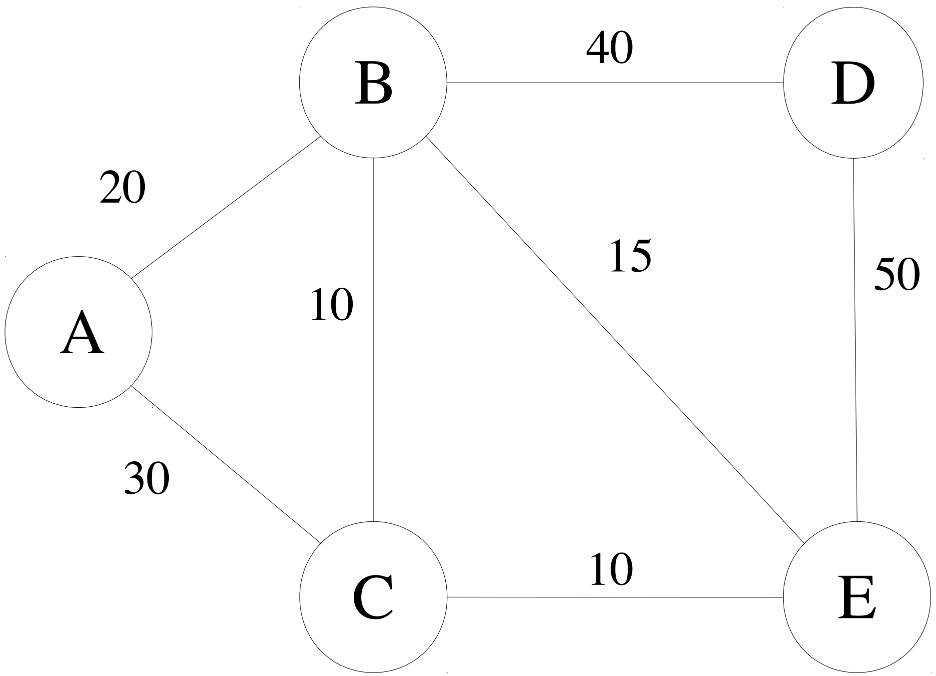
Shortest Path First (SPF)

- Given link metrics (weights) on each individual link
- Find the path (sequence of links) where the sum of the metrics of all links (cumulative cost) is lowest
- Equal cost multipath (ECMP): A *set* of paths with the least (same) cost
- What is the SPF from A to F?



Alternative: Widest path first

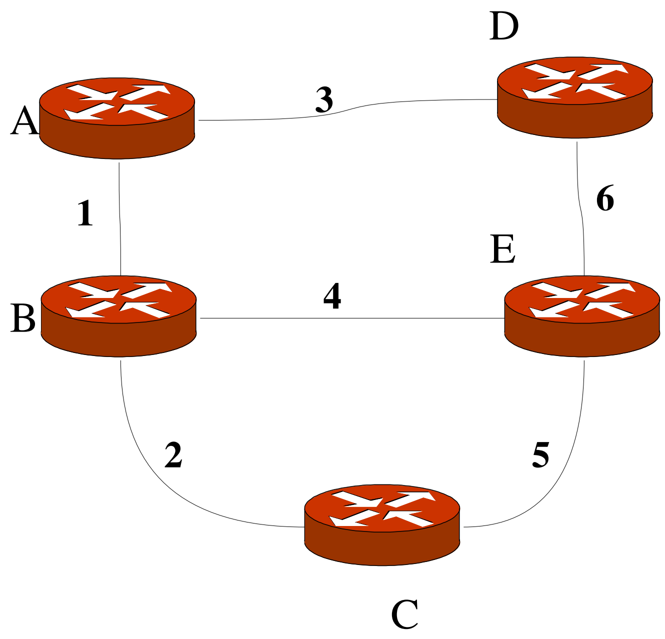
- Numbers denote width: load or bandwidth
  - –*Available* bandwidth
- It is easy to extend SPF algoritms with a widest-path computation rather than shortest path.
- What is the widest path from A -> E?



Distance-Vector/Bellman-Ford

- Each router sends a list of distance-vectors (route with costs) to each neighbour periodically
- Every router selects the route with smallest metric (positive integer)
- The underlying algorithm is called Bellman-Ford.
- Protocols that use Bellman-Ford are called Distance-vector protocols

Example: Distance-vector



A:s initial state: (directly connected networks)

Dest	Cost	NextHop
B	1	-
D	3	-

A distributes this DV to its neighbours (B and D)

A receives B:s (initial) distance vector

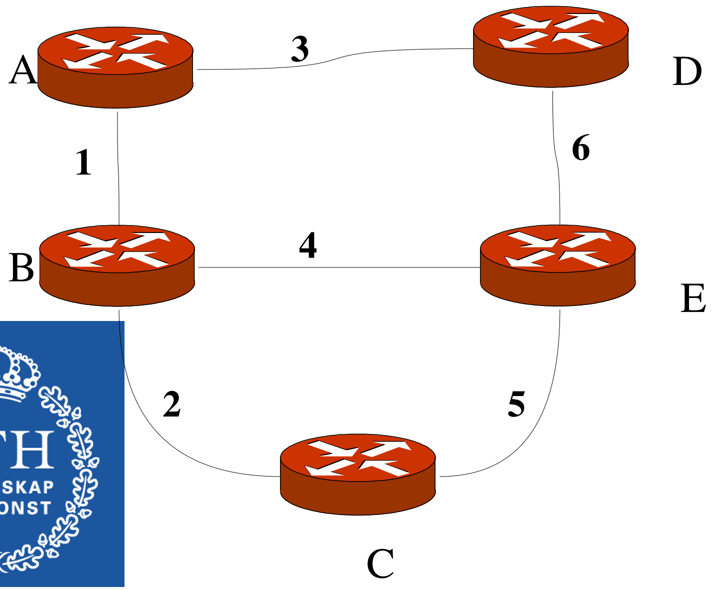
Dest	Cost
A	1
C	2
E	4

A:s state after merging B:s DV:

Dest	Cost	NextHop
B	1	-
C	3	B
D	3	-
E	5	B

A distributes this DV to its neighbours (B and D)

Example: Complete and final state



	A	B	C	D	E
A		1		3	
B	1		2		4
C		2			5
D	3				6
E		4	5	6	

Link metric matrix

A's Distance-Vector

	A	B	C	D	E
A	0				
B		0			
C			0		
D				0	
E					0

Initial state

	A	B	C	D	E
A	0	1	3	3	5
B	1	0	2	4	4
C	3	2	0	6	5
D	3	4	6	0	6
E	5	4	5	6	0

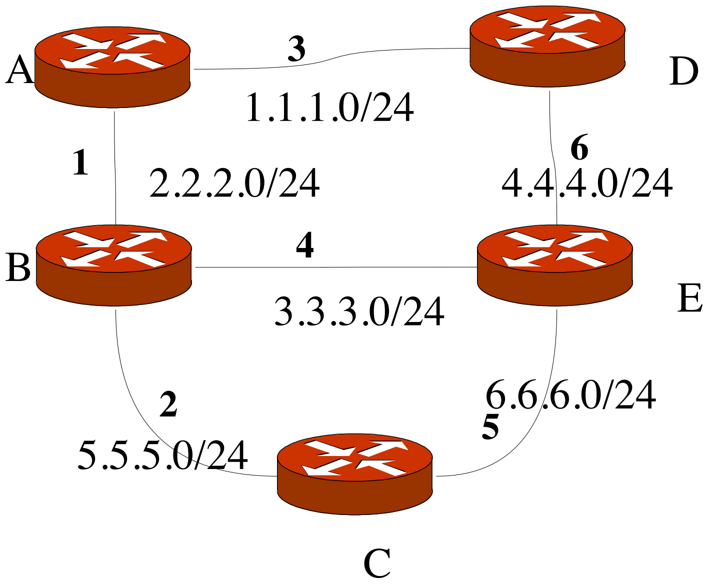
Final state

The algorithm

- Keep a table with an entry for each destination D in the network.
- Store the metric M (distance) and next-hop N for each D in the table.
- Periodically, send the table to all neighbors (the distance-vector).
- For each update that comes in from neighbor N' (to D with a new metric):
  - Add the cost of the link to N' to the new metric to get M'.
  - Replace the route if M' < M.
  - If N = N', always replace the route.
- In most protocols, M is bounded, typically to 16. This upper bound is defined as unreachable(infinity).

Going to real networks

- IP networks require destinations and nexthops (not just nodes)
  - Destinations are networks eg 192.16.32.0/24
  - Next-hops are IP addresses, eg 192.16.32.1
- Suppose the topology changes , eg routers, links crash?
  - Use timers (counters) and age the entries
  - Send updates every (e.g.) 30s
  - If you do not hear from a router in (e.g.) 180s, mark it as invalid



Dest	Cost	NextHop
1.1.1.0/24	3	-
2.2.2.0/24	1	-
3.3.3.0/24	5	2.2.2.2
4.4.4.0/24	9	1.1.1.2
5.5.5.0/24	3	2.2.2.2
6.6.6.0/24	8	2.2.2.2

Converged routing state of A

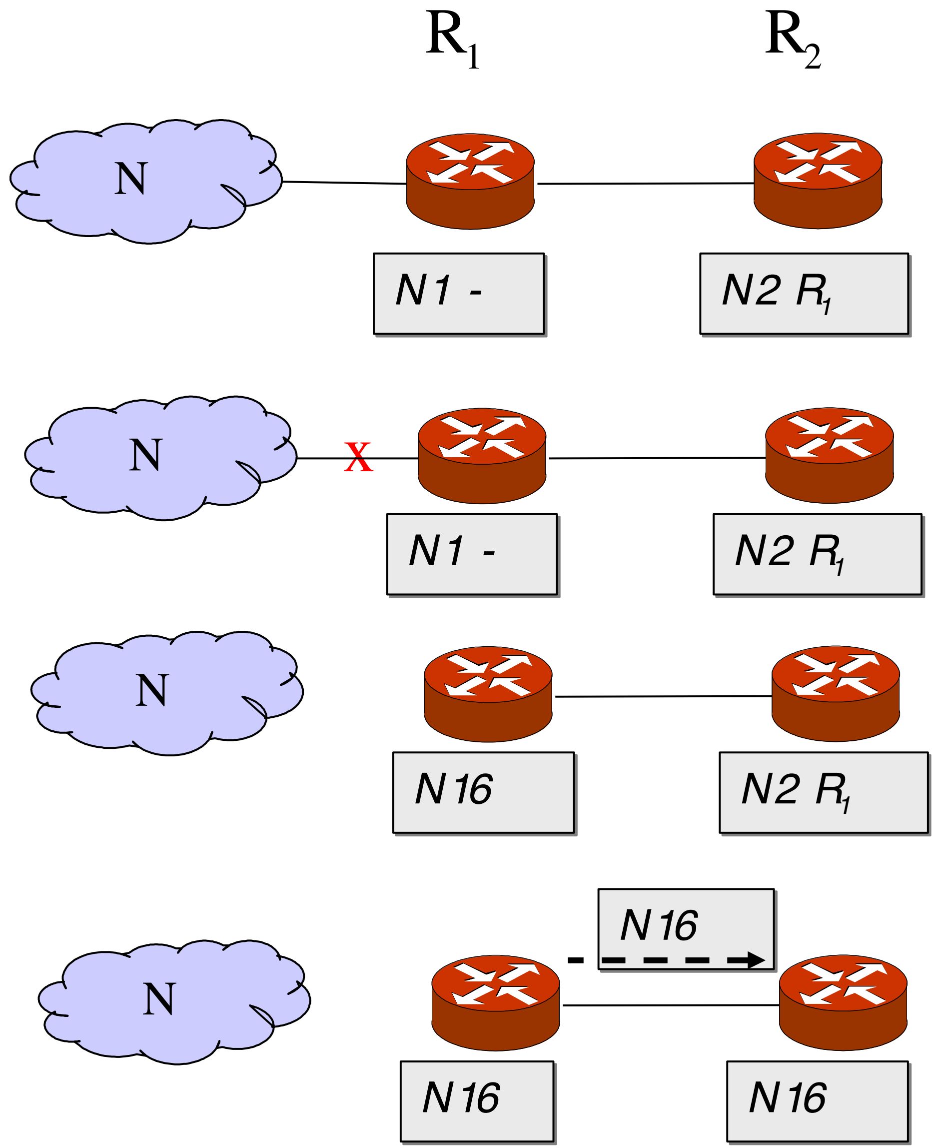
D.V. Problem: Count to Infinity (Two-node instability)

Initially, R_1 and R_2 both have a route to N with metric 1 and 2, respectively.

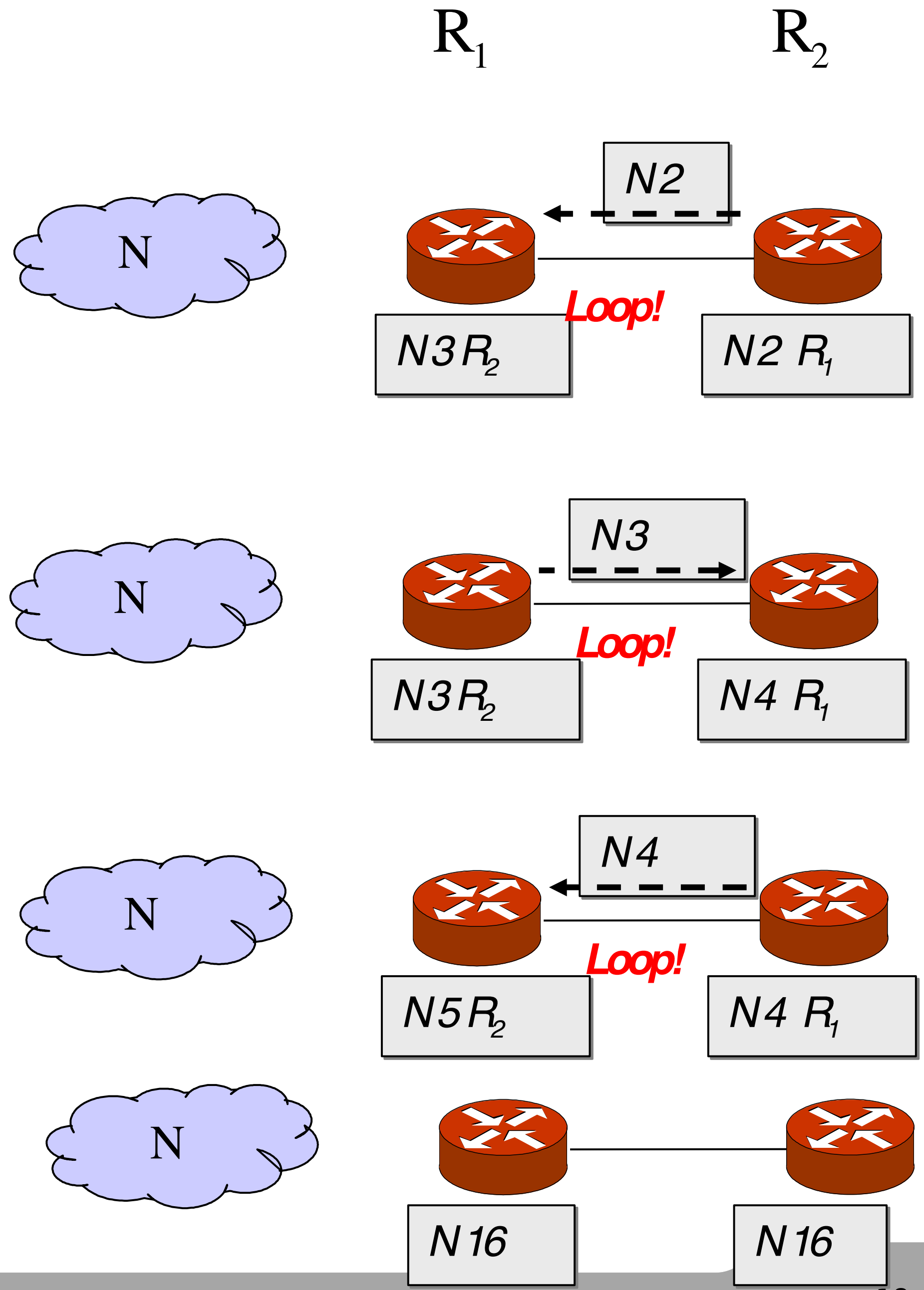
The link between R_1 and N fails.

Now R_1 removes its route to N, by setting its metric to 16 (infinity).

Now two things can happen: Either R_1 reports its route to R_2. Everything is fine.



D.V. Problem: Count to Infinity



The other alternative is that R_2, which still has a route to N, advertises it to R_1. Now things start to go wrong: packets to N are looped until their TTL expires!

Eventually (~10-20s), R_1 sends an update to R_2. The cost to N increases, but the loop remains.

Yet some time later, R_2 sends an update to R_1.

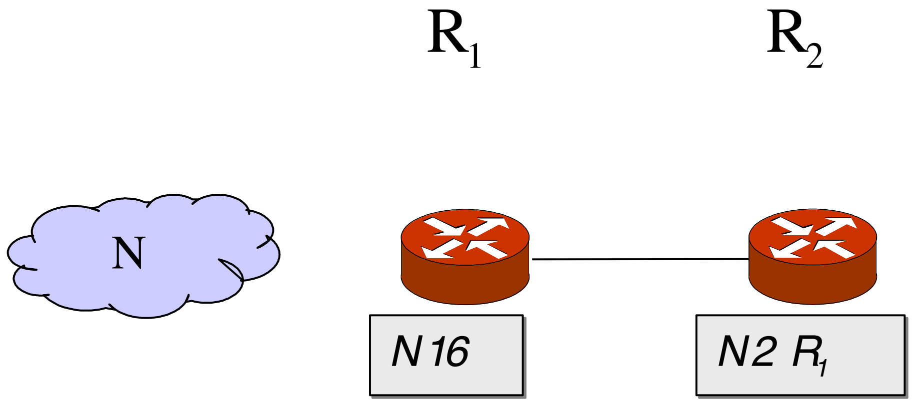
...

Finally, the cost reaches infinity at 16, and N is unreachable. The loop is broken!

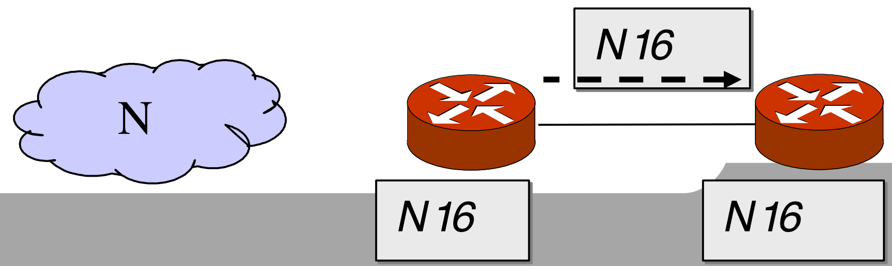
Solution: Split Horizon

- *Do not send routes back over the same interface from where the route 'arrived'.*
- This helps in avoiding “mutual deception”: two routers tell each other they can reach a destination via each other.

R_2, does not announce the route to N to R_1 since that is where it was learnt.



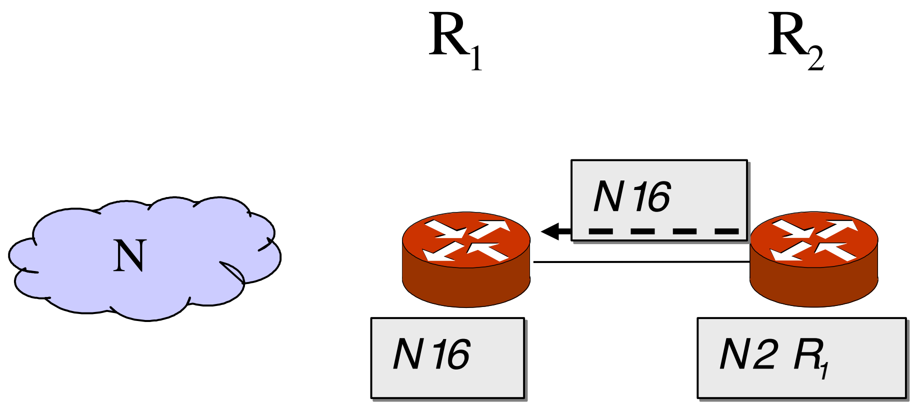
Eventually, R_1 reports its route to R_2 and everything is fine.



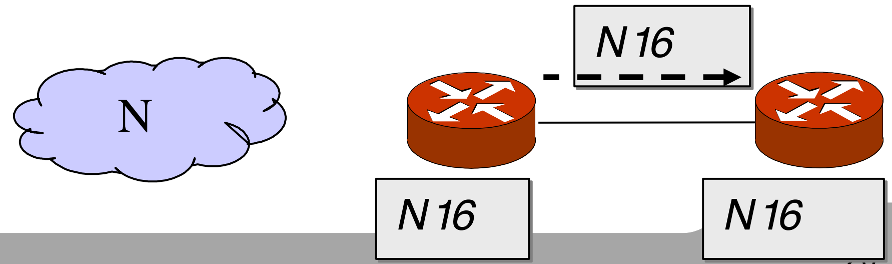
Solution: Split Horizon + Poison Reverse

- *Advertise reverse routes with a metric of 16 (i.e., unreachable).*
- Does not add information but breaks loops faster
- Adds protocol overhead

R_2 always announces an unreachable route to N to R_1.

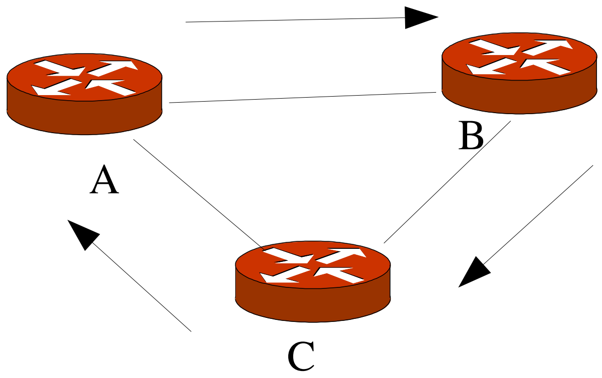


Eventually, R_1 reports its route to R_2 and everything is fine.



Remaining problems

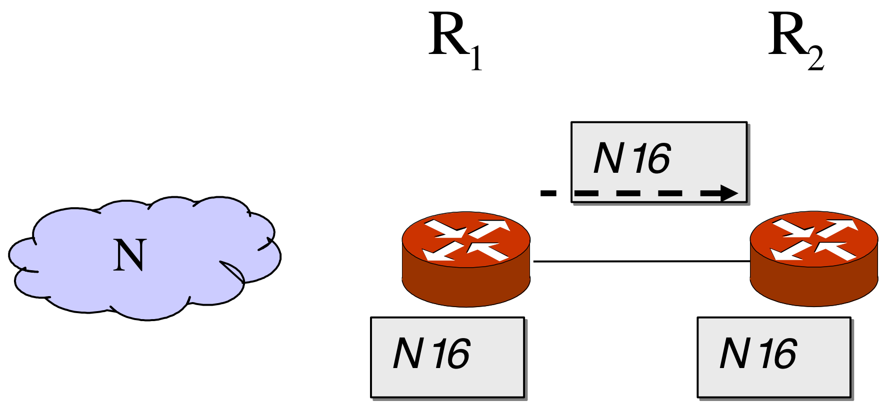
- More than two routers involved in mutual deception
  - –A may believe it has a route through B, B through C, and C through A
- In this case, split horizon with poison reverse does not help



Solution: Triggered Update

- *Send out update immediately when metrics change*
- But only the changed route, not the complete table
- This may lead to a cascade of updates
  - –Apply the rule above recursively!
  - –Therefore, triggered updates are not allowed more often than, for example 1-5 seconds.
- A router may use triggered update only when deleting routes (16).

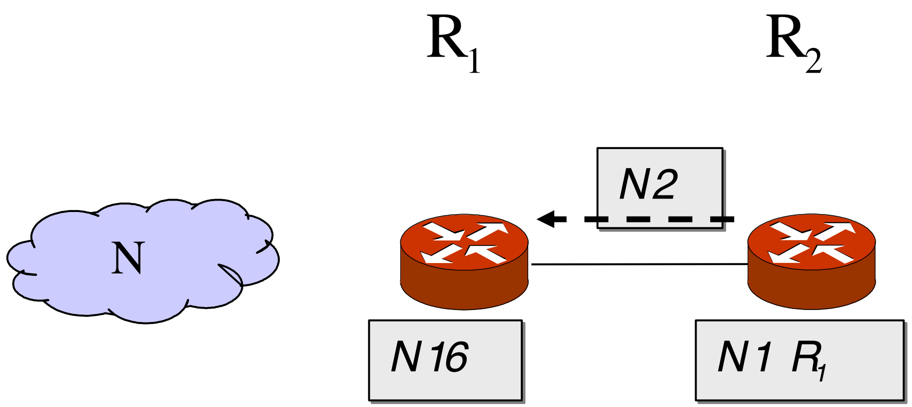
R_1 immediately announces the broken link when it happens.



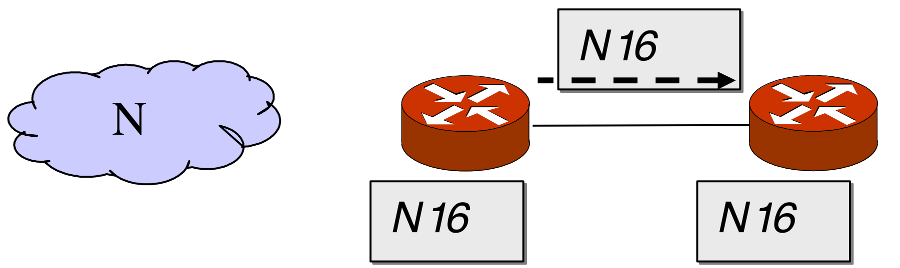
Solution: Hold Down

- *When a route is removed, no update of this route is accepted* for some period of time (hold-down time)- to give everyone a chance to remove the route.

R_1 ignores updates to N from R_2 for some period of time.



Eventually, R_1 sends the update to R_2.

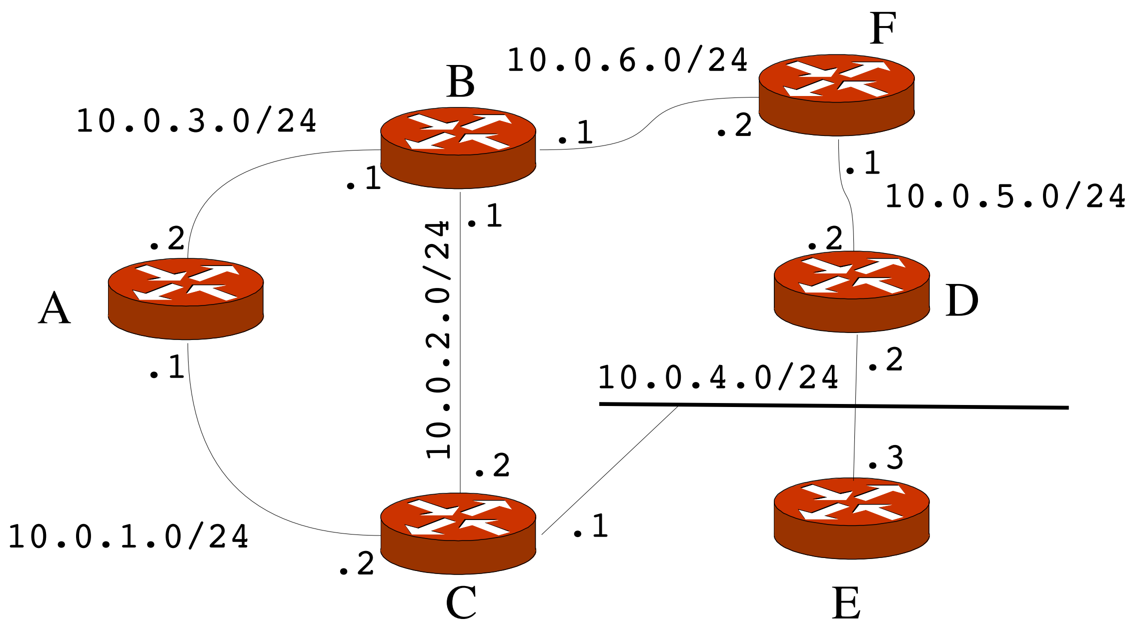


Dijkstra's shortest path first

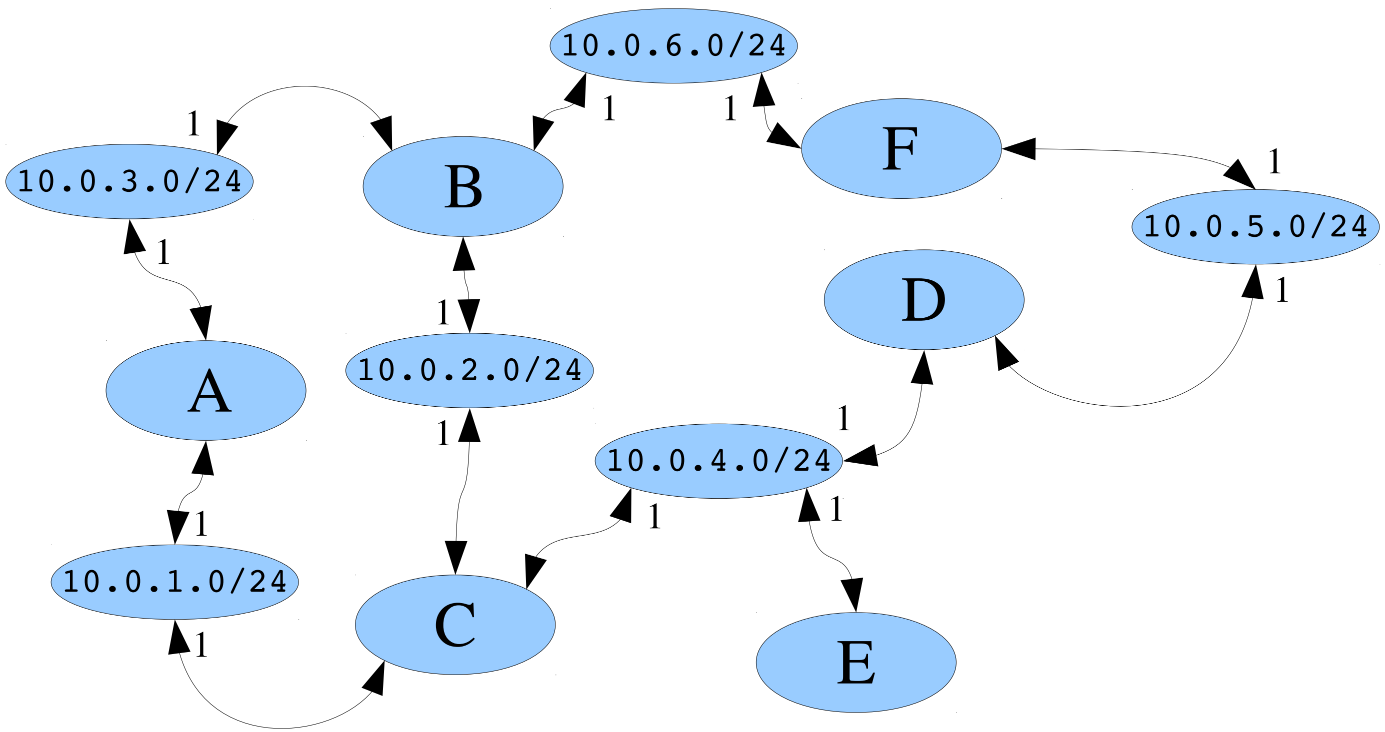
From the link-state database, compute a shortest path delivery tree using a permanent set S and a tentative set Q:

1. Define the root of the tree: the router
2. Assign a cost of 0 to this node and make it the first permanent node.
3. Examine each neighbor node of the last permanent node.
4. Assign a cumulative cost to each node and make it tentative.
5. Among the list of tentative nodes:
   - Find the node with the smallest cumulative cost and make it permanent.
   - If a node can be reached from more than one direction, select the direction with the smallest cumulative cost.
6. Repeat steps 3 to 5 until every node is permanent.

Example network



Example graph



Exercise: Dijkstra from A

Permanent set	Tentative set
`A 0 -`	`10.0.3.0/24 1 -` `10.0.1.0/24 1 -`

Exercise: Dijkstra

Permanent set	Tentative set
A 0 -	~~10.0.3.0/24 1 -~~
10.0.3.0/24 1 -	10.0.1.0/24 1 -
	B 1 -

Exercise: Dijkstra

Permanent set	Tentative set
A 0 -	
10.0.3.0/24 1 -	10.0.1.0/24 1 -
B 1 -	~~B 1 -~~
	10.0.2.0/24 2 B
	10.0.6.0/24 2 B

Exercise: Dijkstra

Permanent set	Tentative set
A 0 -	
10.0.3.0/24 1 -	~~10.0.1.0/24 1 -~~
B 1 -	
10.0.1.0/24 1 -	10.0.2.0/24 2 B
	10.0.6.0/24 2 B
	C 1 -

Exercise: Dijkstra

Permanent set	Tentative set
A 0 -	
10.0.3.0/24 1 -	
B 1 -	
10.0.1.0/24 1 -	10.0.2.0/24 2 B
C 1 -	10.0.6.0/24 2 B
	~~C 1 -~~
	10.0.2.0/24 2 C
	10.0.4.0/24 2 C

Exercise: Dijkstra

Permanent set	Tentative set
A 0 -	
10.0.3.0/24 1 -	
B 1 -	
10.0.1.0/24 1 -	~~10.0.2.0/24 2 B~~
C 1 -	10.0.6.0/24 2 B
10.0.2.0/24 2 B	
10.0.2.0/24 2 C	~~10.0.2.0/24 2 C~~
	10.0.4.0/24 2 C

Note: ECMP

ECMP: Equal Cost MultiPath. More than

Exercise: Dijkstra

Permanent set	Tentative set
A 0 -	
10.0.3.0/24 1 -	
B 1 -	
10.0.1.0/24 1 -	
C 1 -	10.0.6.0/24 2 B
10.0.2.0/24 2 B	
10.0.2.0/24 2 C	
10.0.4.0/24 2 C	~~10.0.4.0/24 2 C~~
	D 2 C
	E 2 C

Exercise: Dijkstra

Permanent set	Tentative set
A 0 -	
10.0.3.0/24 1 -	
B 1 -	
10.0.1.0/24 1 -	
C 1 -	10.0.6.0/24 2 B
10.0.2.0/24 2 B	
10.0.2.0/24 2 C	
10.0.4.0/24 2 C	
E 2 C	~~D 2 C~~
D 2 C	~~E 2 C~~
	10.0.5.0/24 3 C

Exercise: Dijkstra

Permanent set	Tentative set
A 0 -	
10.0.3.0/24 1 -	
B 1 -	
10.0.1.0/24 1 -	
C 1 -	~~10.0.6.0/24 2 B~~
10.0.2.0/24 2 B	
10.0.2.0/24 2 C	
10.0.4.0/24 2 C	
E 2 C	
D 2 C	
10.0.6.0/24 2 B	10.0.5.0/24 3 C
	F 2 B

Exercise: Dijkstra

Permanent set	Tentative set
A 0 -	
10.0.3.0/24 1 -	
B 1 -	
10.0.1.0/24 1 -	
C 1 -	
10.0.2.0/24 2 B	
10.0.2.0/24 2 C	
10.0.4.0/24 2 C	
E 2 C	
D 2 C	
10.0.6.0/24 2 B	10.0.5.0/24 3 C
F 2 B	~~F 2 B~~
	10.0.5.0/24 3 B

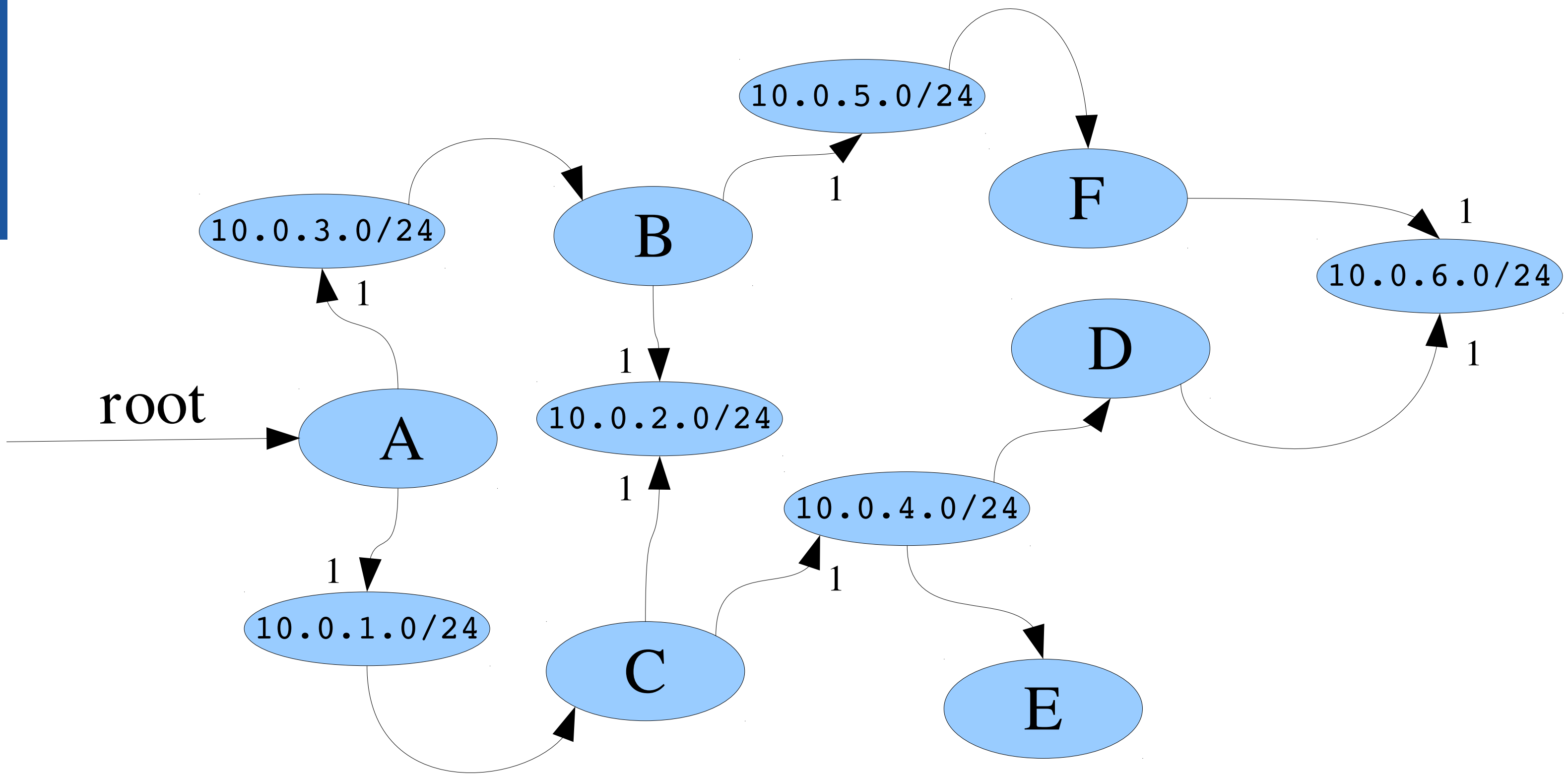
Exercise: Dijkstra (complete)

Permanent set	Tentative set
A 0 -	
10.0.3.0/24 1 -	
B 1 -	
10.0.1.0/24 1 -	
C 1 -	
10.0.2.0/24 2 B	
10.0.2.0/24 2 C	
10.0.4.0/24 2 C	
E 2 C	
D 2 C	
10.0.6.0/24 2 B	~~10.0.5.0/24 3 C~~
F 2 B	
10.0.5.0/24 3 B	~~10.0.5.0/24 3 B~~
10.0.5.0/24 3 C	

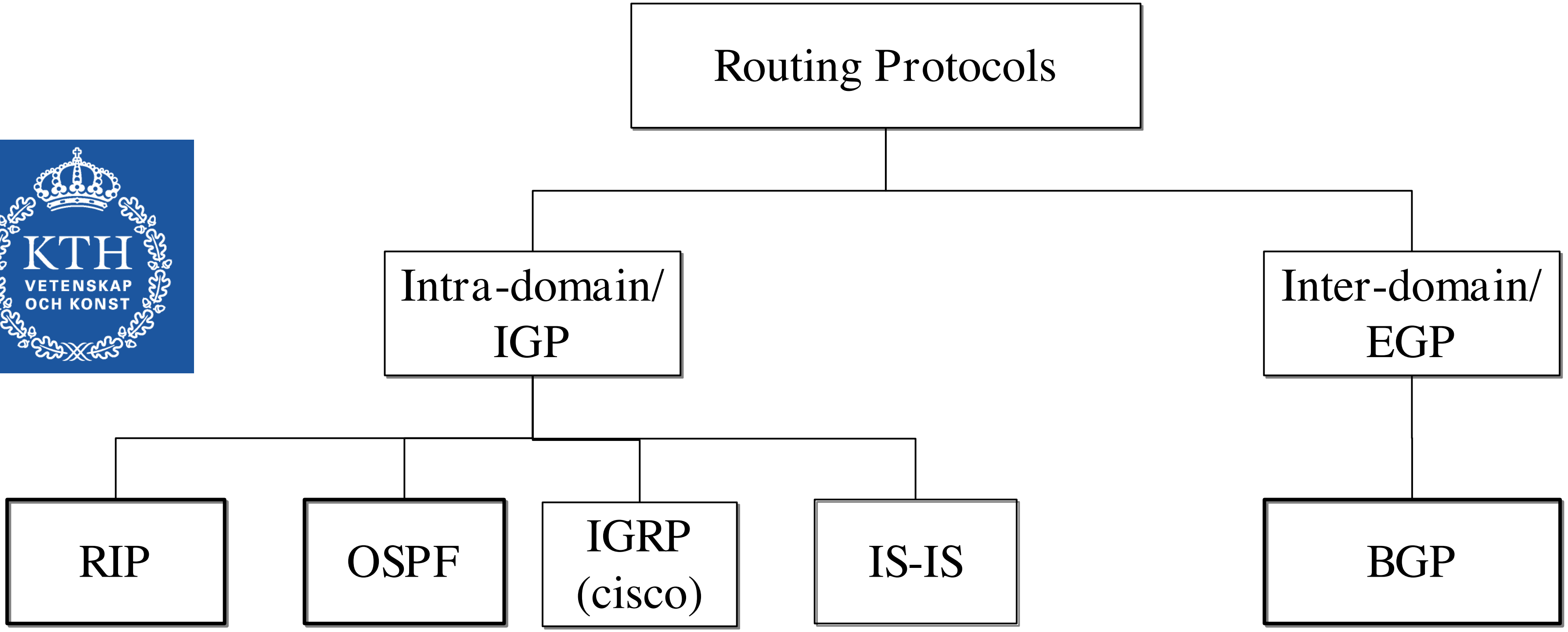
Note: ECMP

Exercise: Dijkstra tree graph view

- Compare with table view in the previous slide
- Note the ECMP routes to 10.0.2.0/24 and 10.0.6.0/24



Popular Unicast Routing Protocols



Routing Information Protocol - RIP

- RIP-1 (RFC 1058), RIP-2 (RFC 2453)
- Metric is hop counts
  - 1: directly connected
  - 16: infinity
  - RIP cannot support networks with diameter > 14.
- RIP uses distance vector
- RIP messages are carried via UDP datagrams.
  - IP Multicast (RIP-2): 224.0.0.9
  - Broadcast (RIP-1)

Disadvantages with RIP

- Slow convergence
  - –Changes propagate slowly
  - –Each neighbor only speaks ~every 30 seconds; information propagation time over several hops is long
- Instability
  - –After a router or link failure RIP takes minutes to stabilize.
- Hops count may not be the best indication for which is the best route.
- The maximum useful metric value is 15
  - –Network diameter must be less than or equal to 15.
- RIP uses lots of bandwidth
  - –It sends the whole routing table in updates.

Why would anyone use RIP?

–It is easy to implement

–It is generally available

–Implementations have been rigorously tested

–It is simple to configure.

–It has little overhead (for small networks)

Link-state routing

- Each router spreads information about its links to its neighbours.
- This information is flooded to every router in the routing domain so that every router has knowledge of the entire network topology.
- Using Dijkstra's algorithm, the shortest path to each prefix in the network is calculated
- OSPF and IS-IS are two well-known link-state routing protocols
- OSPF is popular among organizations (KTH uses OSPF)
- IS-IS is popular among operators (SUNET uses IS-IS)

Comparison with distance-vector

- Link-state uses a distributed database model
- Distance-vector uses a distributed processing model
- Link-state pros:
  - –More functionality due to distribution of original data, no dependency on intermediate routers
    - Easier to troubleshoot
  - –Fast convergence: when the network changes, new routes are computed quickly
  - –Less bandwidth consuming
- Distance-vector pros:
  - –Less complex – easier to implement and administrate
  - –Needs less memory

The OSPF protocol

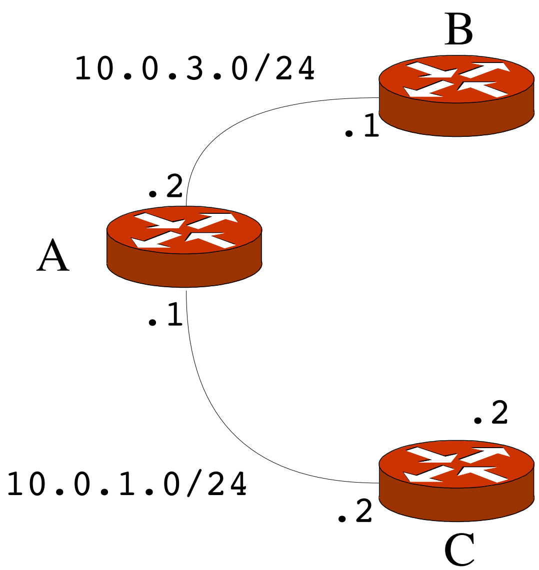
1) The *hello* protocol
   - Is there anybody out there?
   - Detection of neighboring routers
   - Election of designated routers
2) The *exchange* protocol
   - Exchange database between neighbours
3) Reliable *flooding*
   - When links change/age send: update to neighbours and flood *recursively*.
4) *Shortest path* calculation
   - Dijkstra's algorithm
   - Compute shortest path tree to all destinations

The link-state

- Each router describes its environment in the form of networks (links) that it is connected to
- These link-states are the elements of the distributed database
- Fundamental task in OSPF is to distribute the link-states to all nodes in a reliable way
- Then, each node can compute Dijkstra on the *same* database
  - The result (shortest path) is consistent everywhere

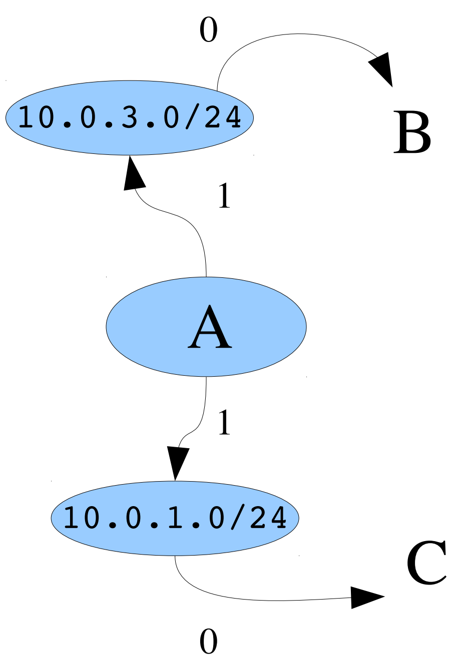
Example: OSPF link state

- Translate the network below to link states (from As point of view)



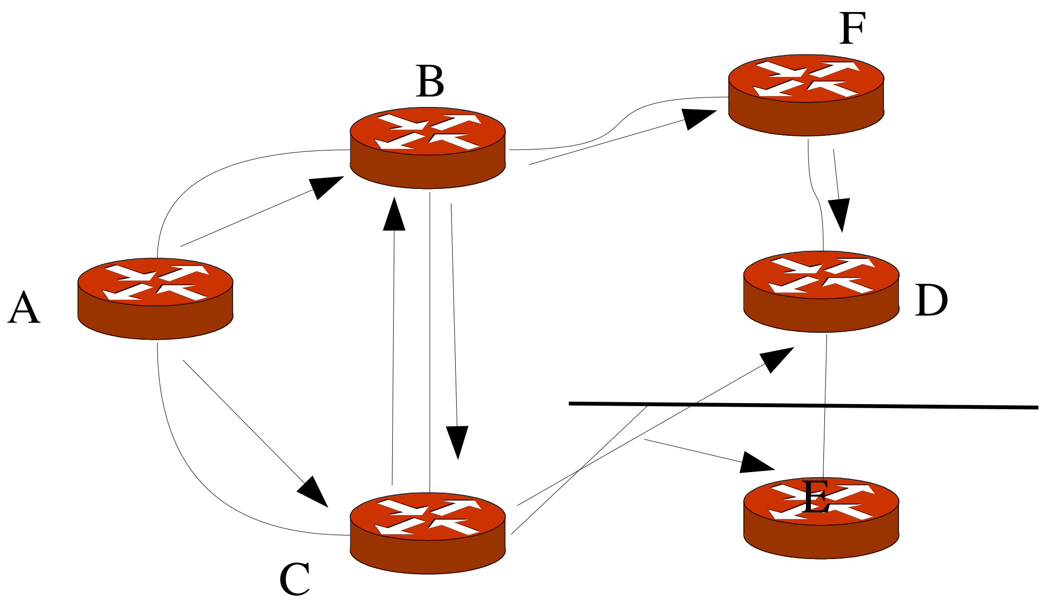
Example: OSPF link state

- Every node creates the link-state of its connected links
- Example: A is connected to two 'transit' links, which are in-turn connected to B and C respectively
- The transit links in this case 'belongs' to A, since A is *designated router* of these sub-networks.
- A therefore distributes the three link-states in the figure:
  - One for A itself (it is connected to two transit networks)
  - Two transit links which are connected to A and B, and A and C respectively.



Flooding link-state

- Every node distributes its link-state to all others
- Initially and after link changes (also every ~30 mins)
- Example: 'A' floods its link-state by sending it to its neighbors, who in turn distributes it to their neighbors, etc
- Flooding is made reliably and to all routers
  - –No need for periodic retransmit – no waste of bandwidth
- The flooding protocol is the most complex part of OSPF (not Dijkstra!)



Autonomous Systems (AS)

- A set of routers that has a single routing policy, that run under a single technical administration
  - A single network or group of networks
  - University, business, organization, operator
- This is viewed by the outside world as an Autonomous System
  - All interior policies, protocols, etc are hidden within the AS
- Represented in the Internet by an Autonomous System Number (ASN). 0-65535
  - Example: ASN 1653 for SUNET
- Currently, operators are switching to four-byte ASNs
  - RFC 4893: BGP Support for Four-octet AS Number Space