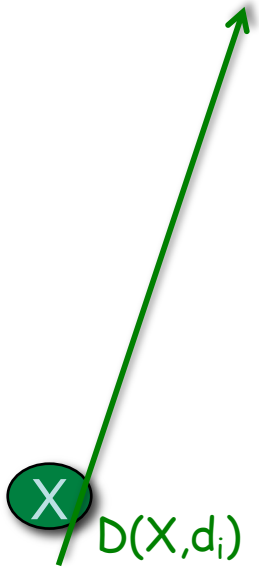
upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Tecnologías Avanzadas de Red
Área de Ingeniería Telemática

Algoritmos de enrutamiento

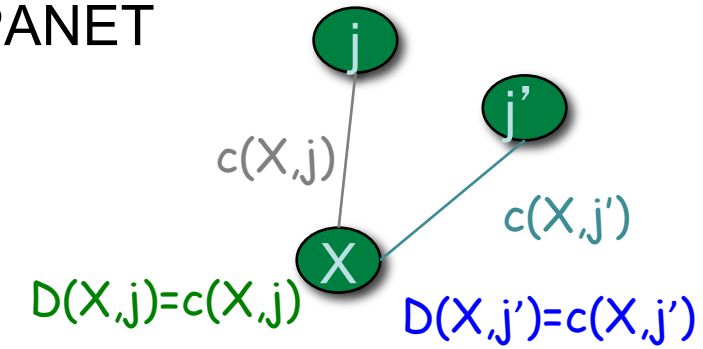
Distance Vector

- Cada nodo X llega a conocer la distancia desde él a todos los destinos
  - $D(X,d_i)$
- Inicialmente cada nodo solo conoce distancia a sus vecinos (…)
  - $D(X,d) = c(X,d)$
- Periódicamente comunica D(X,d) a todos sus vecinos
  - Informan con un ***vector*** con las ***distancias*** a los destinos
    ($D(X,d_1)$, $D(X,d_2)$, $D(X,d_3)$, $D(X,d_4)$…)
  - Asíncrono
- Al recibir información actualiza (… …):
  - $D(X,d) \leftarrow \text{mín}_{j/c(X,j)<\infty}\{c(X,j)+D(j,d)\}$
- Algoritmo de ***Bellman-Ford*** distribuido
- Empleado desde los comienzos de la ARPANET



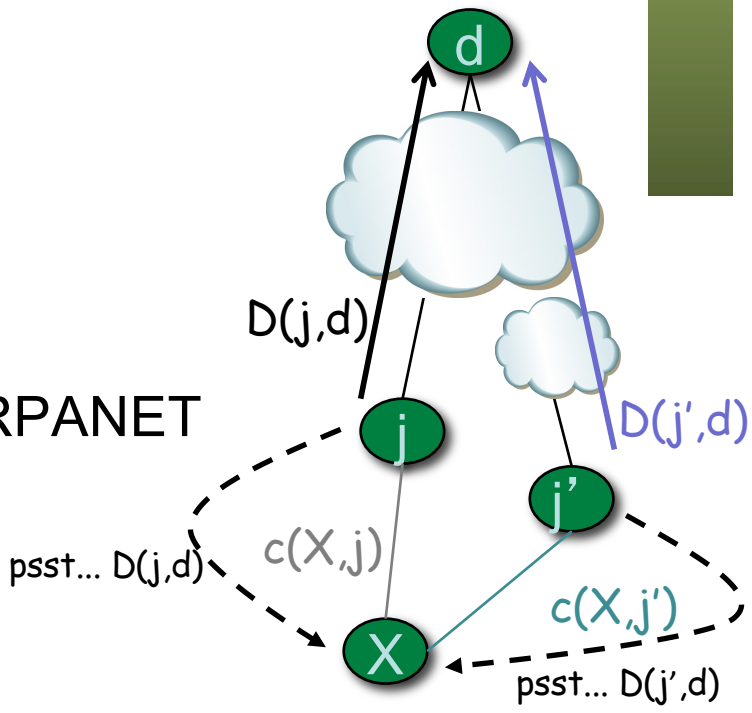
Distance Vector

- Cada nodo X llega a conocer la distancia desde él a todos los destinos
  - $D(X,d_i)$
- Inicialmente cada nodo solo conoce distancia a sus vecinos (…)
  - $D(X,d) = c(X,d)$
- Periódicamente comunica D(X,d) a todos sus vecinos
  - Informan con un ***vector*** con las ***distancias*** a los destinos
    ($D(X,d_1)$, $D(X,d_2)$, $D(X,d_3)$, $D(X,d_4)$…)
  - Asíncrono
- Al recibir información actualiza (… …):
  - $D(X,d) \leftarrow \text{mín}_{j/c(X,j)<\infty}\{c(X,j)+D(j,d)\}$
- Algoritmo de ***Bellman-Ford*** distribuido
- Empleado desde los comienzos de la ARPANET



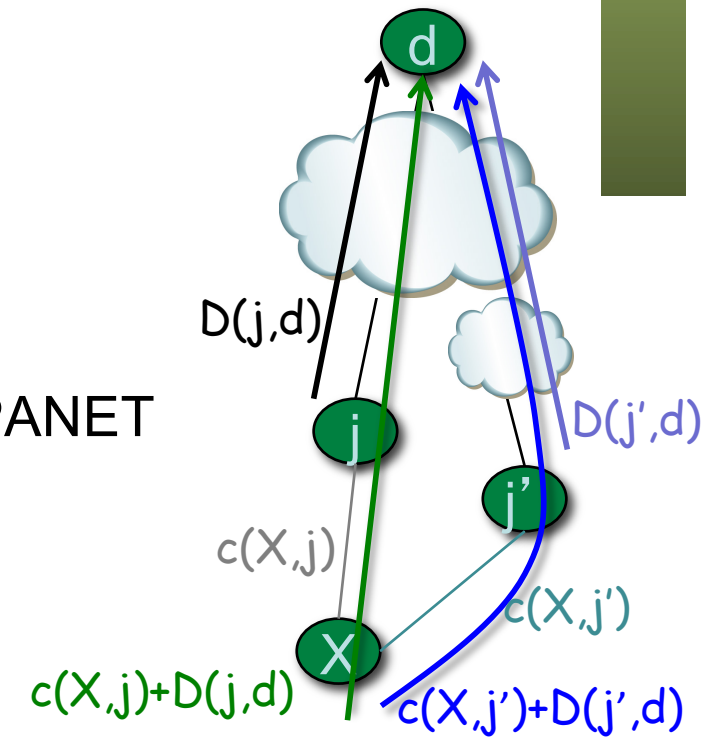
Distance Vector

- Cada nodo X llega a conocer la distancia desde él a todos los destinos
  - $D(X,d_i)$
- Inicialmente cada nodo solo conoce distancia a sus vecinos (…)
  - $D(X,d) = c(X,d)$
- Periódicamente comunica D(X,d) a todos sus vecinos
  - Informan con un ***vector*** con las ***distancias*** a los destinos
    ($D(X,d_1)$, $D(X,d_2)$, $D(X,d_3)$, $D(X,d_4)$…)
  - Asíncrono
- Al recibir información actualiza (… …):
  - $D(X,d) \leftarrow \min_{j/c(X,j)<\infty}\{c(X,j)+D(j,d)\}$
- Algoritmo de ***Bellman-Ford*** distribuido
- Empleado desde los comienzos de la ARPANET



Distance Vector

- Cada nodo X llega a conocer la distancia desde él a todos los destinos
  - $D(X,d_i)$
- Inicialmente cada nodo solo conoce distancia a sus vecinos (…)
  - $D(X,d) = c(X,d)$
- Periódicamente comunica D(X,d) a todos sus vecinos
  - Informan con un ***vector*** con las ***distancias*** a los destinos
    ($D(X,d_1)$, $D(X,d_2)$, $D(X,d_3)$, $D(X,d_4)$…)
  - Asíncrono
- Al recibir información actualiza (… …):
  - $D(X,d) \leftarrow \text{mín}_{j/c(X,j)<\infty}\{c(X,j)+D(j,d)\}$
- Algoritmo de ***Bellman-Ford*** distribuido
- Empleado desde los comienzos de la ARPANET



upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Tecnologías Avanzadas de Red
Área de Ingeniería Telemática

RIP: Características y funcionamiento básico

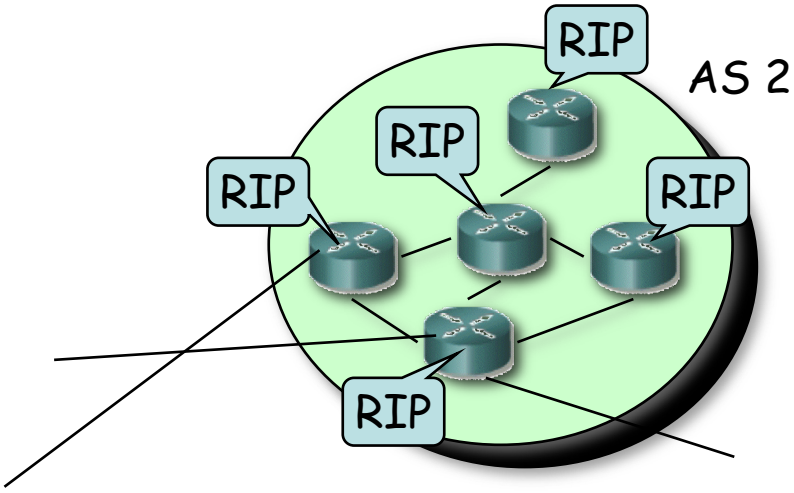
Distance Vector

- Cada nodo tiene unas distancias estimadas a cada destino (vector de distancias)
- Se las envía a todos sus vecinos periódicamente
- Generalmente algoritmo de Bellman-Ford distribuido
- No necesitan conocer la topología completa de la red
- Usado en la ARPANET hasta 1979
- Ejemplos: RIP, Xerox XNS RIP, IPX RIP, Cisco IGRP, DEC's DNA Phase IV, Apple's RTMP

RIP: Características

- *Routing Information Protocol*
- Distance Vector
- IGP
- RFC 1058 (v1), STD 56 (v2)
- routed en Unix BSD
- Emplea UDP
- Métrica:
  - Número de saltos
  - 16 = ∞
- Se envía el vector de distancias cada 30 segs (+/- 0 a 5s al azar)
- Cambios en la topología:
  - Ruta a red N por router G
  - Si no recibimos vector de G en 180segs marcar como inválida (∞)

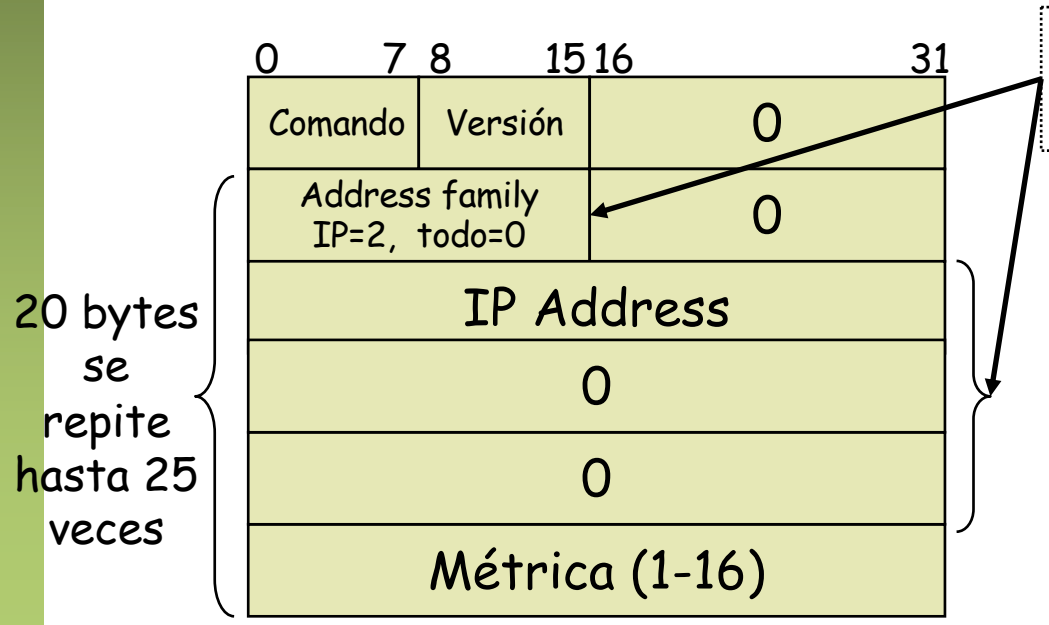
- No escala para redes grandes
- Mejor para redes con enlaces homogéneos
- Simple
- Malos tiempos de convergencia

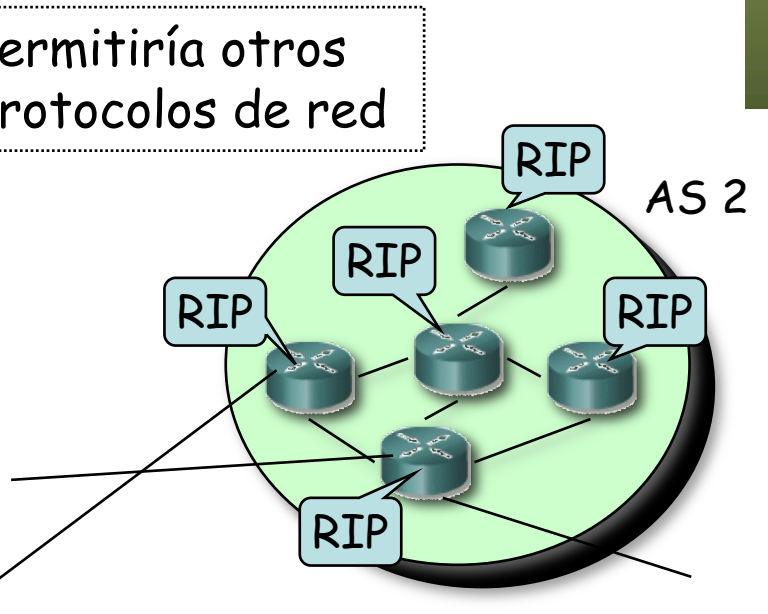


RIP: Características

Tipos de PDUs:

- *Request*
  - Comando=1
  - Se puede pedir el coste a unos destinos o a todos
- *Response*
  - Comando=2
  - El *next-hop* es la IP que envía la PDU
  - Periódico o en respuesta a un *request*





RIP: Funcionamiento

Inicialización

- Manda un *request* especial por cada interfaz
- IP destino *broadcast*

Recibe un *request*

- Si es de inicialización manda todo el vector
- Si no, responde con los valores solicitados

Periódicamente

- Timer 30seg (de 25 a 35)
- Manda un *response* con todo el vector por cada interfaz
- IP destino broadcast

Recibe *response*

- **Actualiza** su vector y tabla de rutas
- Si la tiene reinicializa timer

Caduca timer de una ruta

- Timer de 180s para cada una
- Pasa a coste ∞
- Inicia timer para borrarla

Timer de borrado

- Timer de 120s para una ruta invalidada

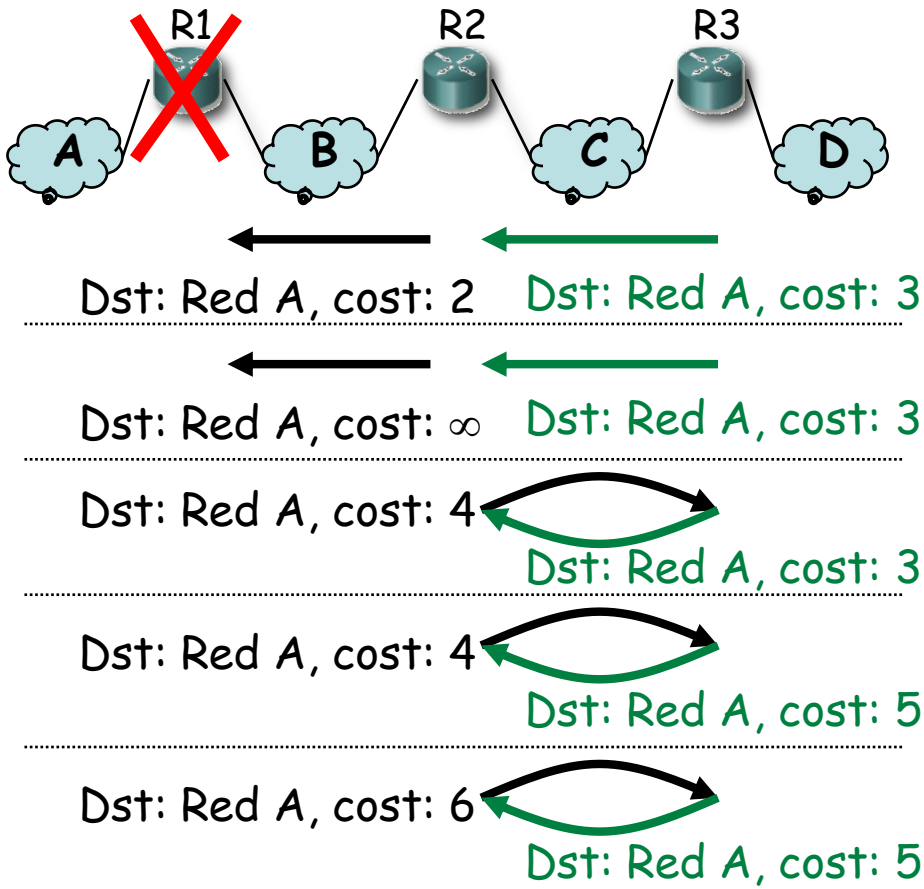
RIP: Actualización

1. Añadir 1 a la métrica de cada destino anunciado en el paquete de RIP recibido
2. Para cada entrada en el paquete
   - Si el destino no está en la tabla de rutas
     1. Añadirlo
   - Si no (sí está en la tabla)
     1. Si el siguiente salto en la tabla es el mismo que quien ha mandado el paquete de IP
        - Sustituir el coste por el nuevo
     2. Si no (diferente *next-hop*)
        - Si el coste es menor que el de la tabla
          - Sustituir el coste y el *next-hop*

RIP: Cuentas a infinito

Bad news travel slowly

- Supongamos que R1 falla (…)
- Aprox. 3min después R2 marca la ruta como inválida (…)
- Si antes de que envíe el vector a R3 se lo envía él (…)
- ¡ Ahora piensa que se va por R3 !
- Pero cuando informa a R3 del nuevo camino éste verá un aumento en el coste (…)
- Y así *ad infinitum* (…)
- Proceso de cuenta a infinito
- Infinito = 16 !



Cuentas a infinito

Split horizon

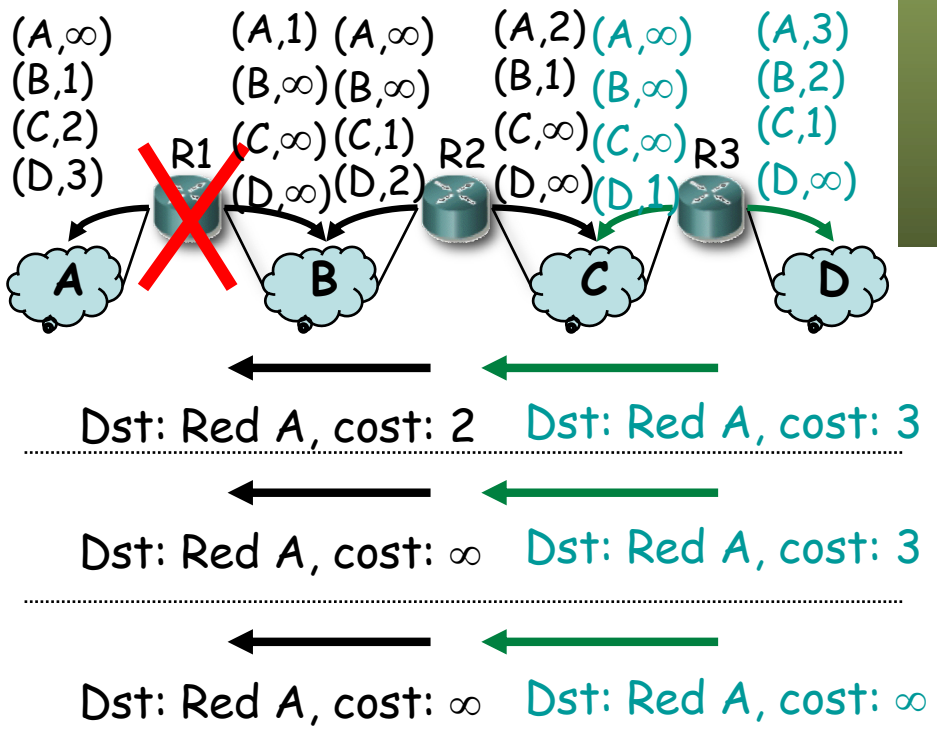
- Al enviar vector por un interfaz **no incluir** los destinos a los que se llega por él
- Mensajes más pequeños
- Evita el bucle anterior

Ejemplo (… …):

- Caduca timer (180s) en R2 (…)
- Caduca timer (30s) en R2, envía vector (…)

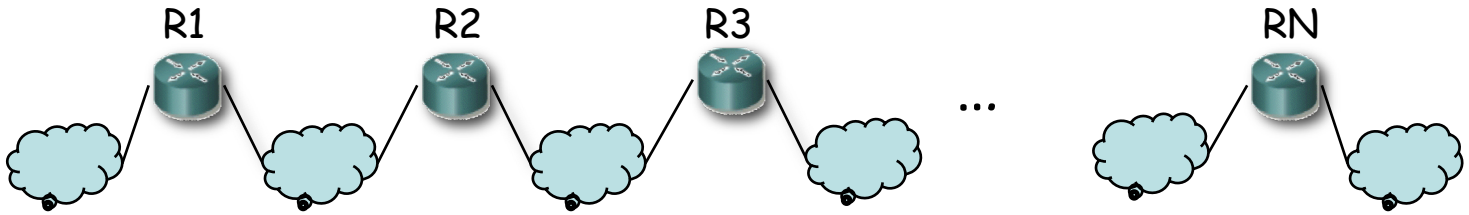
Split horizon with poisoned reverse

- Al enviar vector por un interfaz anunciar los destinos a los que se llega por él con métrica ∞
- No hay que esperar al timeout de la ruta
- Mensajes vuelven a ser grandes



Bad news travel slowly

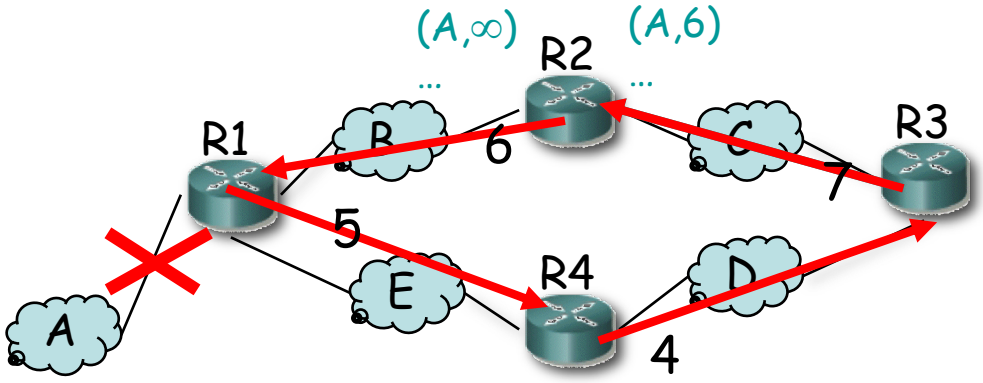
- Convergencia lenta
- Ejemplos:
  - Actualización de información
    - Caso peor N x 30seg para llegar al otro extremo
  - Pérdida de ruta
    - Caso peor N x 180seg hasta el otro extremo
- ¿ Mejorar estos tiempos ?
  - ***Triggered updates***: Enviar el vector en cuanto se produzca un cambio en el mismo



Cuentas a infinito

- Supongamos la topología de la figura
- Usan *split horizon with poisoned reverse*
- Las flechas son las rutas hacia la Red A (…)
- Supongamos que falla el interfaz de R1 en la Red A (…)
- R1 anuncia coste ∞ a R2 y R4 (…)
- Puede que antes de que avisen a R3 él envíe su actualización periódica (…)
- R4 introduce una entrada hacia la Red A por R3 (…)
- R4 anunciará esa ruta a R1 (…)
- R1 creerá que se llega por R4 con coste 5 (…)
- R1 lo anunciará a R2 (…)
- R2 creerá que se llega por R1 (…)
- Y luego R2 hasta llegar a R3 (…)

¡ Cuenta a infinito !



Cuentas a infinito

Solución

- *Hold down period*
- Al marcar una ruta como inválida
- Esperar un tiempo antes de aceptar nuevas rutas a ese destino
- Ejemplo:
  - R4 entra en *hold down*
  - Ignora ruta anunciada por R3

¿Cuánto esperar?

- Depende del tamaño de la red
- Se sobredimensiona (120s)
- Si hay una ruta alternativa tardará en descubrirla (…)

Split horizon + posioned reverse + Triggered updates + hold down interval
¡ Ya no es tan simple !