Enrutamiento

Distance-Vector...

Area de Ingeniería Telemática
http://www.tlm.unavarra.es

Redes
4º Ingeniería Informática

Hoy...

1. Introducción a las redes
2. Tecnologías para redes de área local
3. Conmutación de circuitos
4. Tecnologías para redes de área extensa y última milla
5. **Encaminamiento (sesión 2)**

- Arquitectura de conmutadores de paquetes
- Control de acceso al medio
- Transporte extremo a extremo

Algoritmos para encontrar caminos

- Algoritmo de **Dijkstra** para la distancia mínima
- Conocemos: Nodos i Nodo raiz r pesos w(i,j)
- Mantenemos: d(i) mejor distancia de r al nodo i conocida hasta ahora

  T conjunto de nodos a los que ya conocemos la distancia mas corta

  d(i) es la distancia menor de r hasta el nodo i pasando solo por nodos que están en T
- Algoritmo:

Inicializar

d(i)=infinito d(r)=0 si el nodo i es vecino de r [w(i,r)<infinito] d(i)=w(i,r)

T={r}

Mientras haya nodos que no pertenezcan a T

elegir el nodo i que no este en T con menor d(i)

añadir el nodo i a T

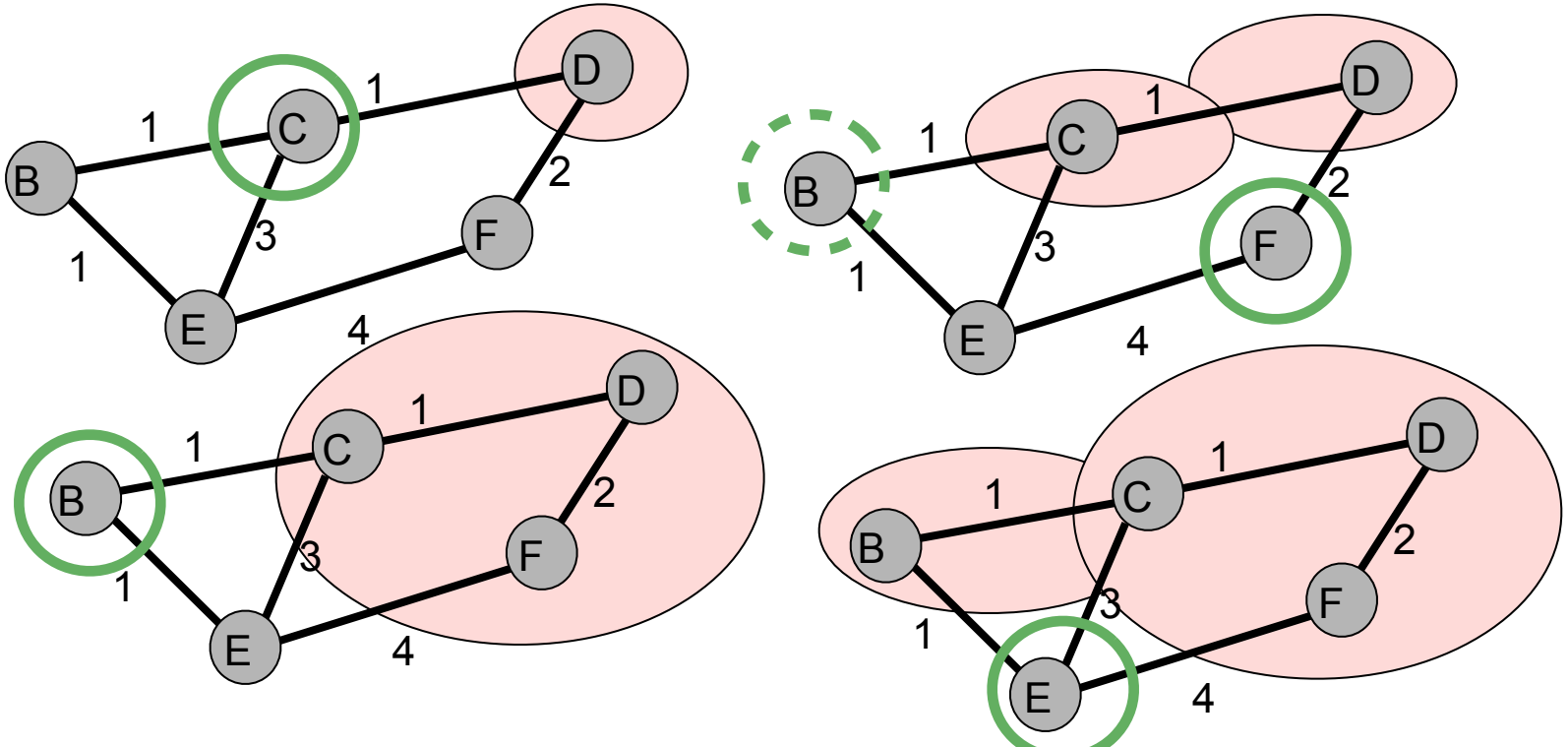
actualizar d(k) de los nodos vecinos al nodo i que no están en T.

Si d(i)+w(k,i) < d(k) entonces d(k)=d(i)+w(k,i)

[es menor la distancia pasando por i que la que ya tenia]

Dijkstra ejemplo

T	d(B)	d(C)	d(D)	d(E)	d(F)
{D}	infinito	1	0	infinito	2
{D,C}	**2**	1	0	**4**	2
{D,C,F}	2	1	0	4	2
{D,C,F,B}	2	1	0	**3**	2
{D,C,F,B,E}	2	1	0	3	2



Manteniendo el camino

- Usando el algoritmo de **Dijkstra** para el camino mínimo
- Conocemos: Nodos i Nodo raiz r pesos w(i,j)
- Mantenemos: d(i) mejor distancia de r al nodo i conocida hasta ahora
  s(i) siguiente nodo a i en el camino hacia r
  T conjunto de nodos a los que ya conocemos la distancia mas corta

Algoritmo:

Inicializar

d(i)=infinito d(r)=0 si el nodo i es vecino de r [w(i,r)<infinito] d(i)=w(i,r)

T={r}

s(i)=desconocido si i es vecino de r [w(i,r)<infinito] s(i)=r

Mientras haya nodos que no pertenezcan a T

elegir el nodo i que no este en T con menor d(i)

añadir el nodo i a T

actualizar d(k) de los nodos vecinos al nodo i que no están en T.

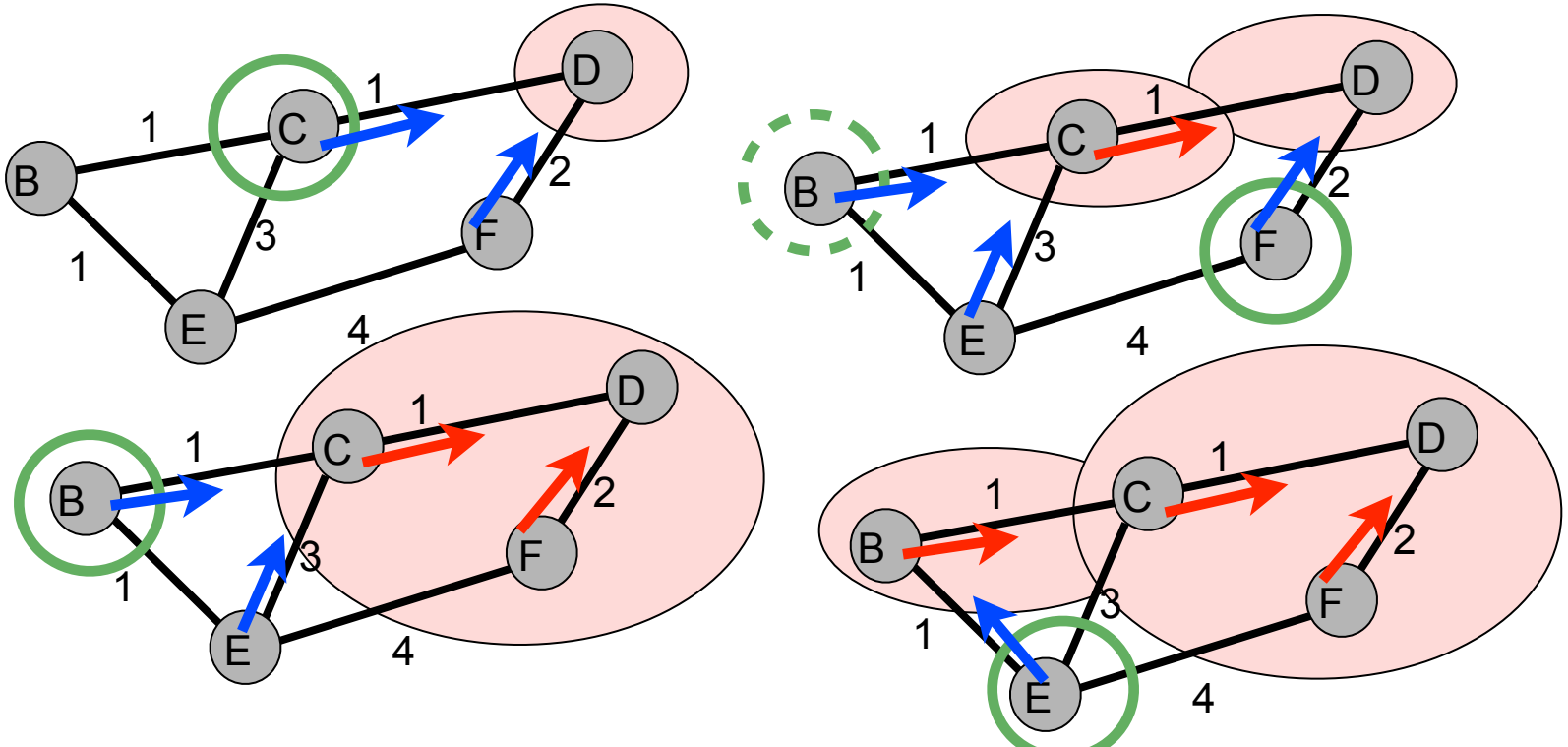
Si d(i)+w(k,i) < d(k) entonces d(k)=d(i)+w(k,i)

y tambien s(k)=i

[si el camino por i es mejor i es el nuevo siguiente salto de k]

Dijkstra ejemplo con camino

T	d(B) / s(B)	d(C) / s(C)	d(D) / s(D)	d(E) / s(E)	d(F) / s(F)
{D}	infinito / desc	1 / D	0 / soy yo	infinito / desc	2 / D
{D,C}	**2 / C**	1 / D	0 / soy yo	**4 / C**	2 / D
{D,C,F}	2 / C	1 / D	0 / soy yo	4 / C	2 / D
{D,C,F,B}	2 / C	1 / D	0 / soy yo	**3 / B**	2 / D
{D,C,F,B,E}	2 / C	1 / D	0 / soy yo	3 / B	2 / D



Otro algoritmo

- Algoritmo de Bellman-Ford
- Conocemos: Nodos i Nodo raiz r pesos w(i,j)
- Mantenemos: $d_h(i)$ mejor distancia de r al nodo i conocida hasta ahora
  $s_h(i)$ siguiente salto del nodo i hacia r (mejor conocido hasta ahora)
  h numero de saltos que hemos considerado
  $d_h(i)$ es la mejor distancia de i al nodo r dando como mucho h saltos

Algoritmo:

Inicializar

$d_0(i)$=infinito $d_0(r)$=0 [en 0 saltos solo desde r se puede llegar a r]

$s_0(i)$=desconocido $s_0(r)$= no hace falta siguiente salto yo soy la raiz

Para h=1,2,3,4 .... hasta cuando?

Para cada uno de los nodos i

Para cada uno de los nodos k [vecinos de i]

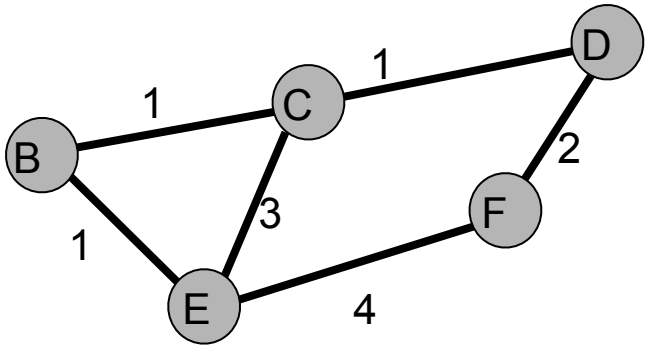
$d_h(i)=\min\{ d_{h-1}(i)+w(i,k) \}$

$s_h(i)$= el k con que se obtiene el maximo [nada si son todos infinito]

[igual que en el anterior solo que ahora consideramos que puede haber mejorado el camino de cualquiera de nuestros vecinos]

Bellman-Ford ejemplo

h	d(B) / s(B)	d(C) / s(C)	d(D) / s(D)	d(E) / s(E)	d(F) / s(F)
0	infinito / desc	infinito / desc	0 / soy yo	infinito / desc	infinito / desc
1	infinito / desc				
2					
3					
4					
5 ...					



salen todos infinitos

La minima distancia de B a D dando como mucho un salto es infinito
no se puede llegar en un salto

- Para cada nodo por ejemplo para el B

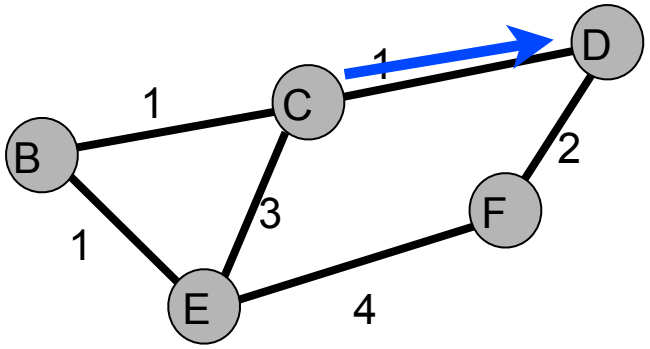
  $d_1(B)$ en la siguiente fila depende de la fila actual y las distancias

  $d_0(B)+w(B,B)$ $d_0(C)+w(B,C)$ $d_0(D)+w(B,D)$ $d_0(E)+w(B,E)$ $d_0(F)+w(B,F)$

  inf + 0 inf + 1 0 + inf inf + 1 inf + inf

Bellman-Ford ejemplo

h	d(B) / s(B)	d(C) / s(C)	d(D) / s(D)	d(E) / s(E)	d(F) / s(F)
0	infinito / desc	infinito / desc	0 / soy yo	infinito / desc	infinito / desc
1	infinito / desc	1 / D			
2					
3					
4					
5 ...					



El unico que no sale infinito es el D con distancia 1

La minima distancia de C a D dando como mucho un salto es 1

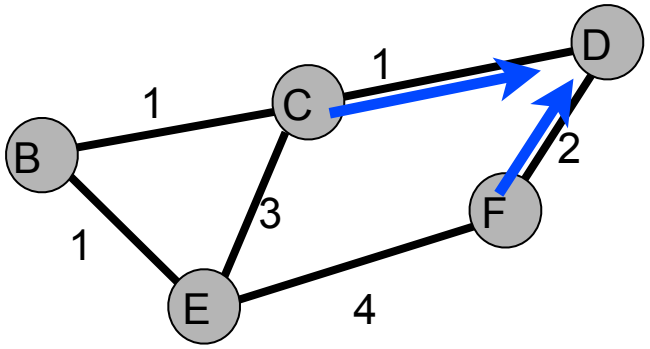
- Para el C

  $d_1(C)$ en la siguiente fila depende de la fila actual

$d_0(B)+w(C,B)$	$d_0(C)+w(C,C)$	$d_0(D)+w(C,D)$	$d_0(E)+w(C,E)$	$d_0(F)+w(C,F)$
inf + 1	inf + 0	0 + 1	inf + 3	inf + inf

Bellman-Ford ejemplo

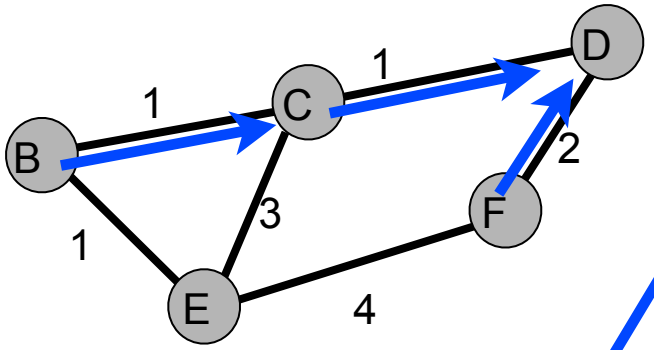
h	d(B) / s(B)	d(C) / s(C)	d(D) / s(D)	d(E) / s(E)	d(F) / s(F)
0	infinito / desc	infinito / desc	0 / soy yo	infinito / desc	infinito / desc
1	infinito / desc	1 / D	0 / soy yo	infinito / desc	2 / D
2					
3					
4					
5 ...					



- En una iteracion tenemos el arbol de los mejores caminos a D con un salto
- En este caso es ya el camino correcto pero no tiene por que ser (si la raiz hubiera sido C tendriamos que el mejor camino de E a C con un salto es de distancia 3)

Bellman-Ford ejemplo

h	d(B) / s(B)	d(C) / s(C)	d(D) / s(D)	d(E) / s(E)	d(F) / s(F)
0	infinito / desc	infinito / desc	0 / soy yo	infinito / desc	infinito / desc
1	infinito / desc	1 / D	0 / soy yo	infinito / desc	2 / D
2	2 / C				
3					
4					
5 ...					



Ahora a traves de C tenemos distancia 2

En realidad no hace falta probar mas que con los vecinos
En los que no son vecinos de B siempre dara infinito

- Segunda iteración para el B

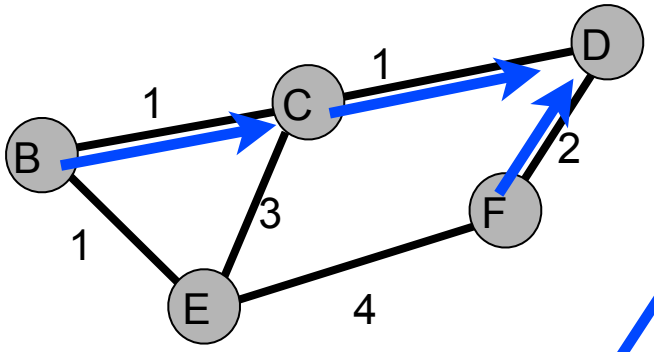
$d_2(B)$

$d_1(B)+w(B,B)$	$d_1(C)+w(B,C)$	$d_1(D)+w(B,D)$	$d_1(E)+w(B,E)$	$d_1(F)+w(B,F)$
inf + 0	1 + 1	0 + inf	inf + 1	2 + inf

Este es B tampoco es vecino

Bellman-Ford ejemplo

h	d(B) / s(B)	d(C) / s(C)	d(D) / s(D)	d(E) / s(E)	d(F) / s(F)
0	infinito / desc	infinito / desc	0 / soy yo	infinito / desc	infinito / desc
1	infinito / desc	1 / D	0 / soy yo	infinito / desc	2 / D
2	2 / C	1 / D			
3					
4					
5 ...					



Este es C no es un candidato

Me vuelve a salir mejor el camino via D

- Para cada nodo por ejemplo para el C

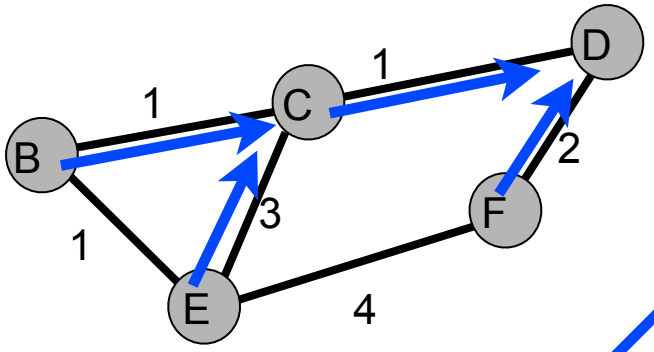
$d_2(C)$

$d_1(B)+w(C,B)$ ~~$d_1(C)+w(C,C)$~~ $d_1(D)+w(C,D)$ $d_1(E)+w(C,E)$ ~~$d_1(F)+w(C,F)$~~

inf + 1 ~~1 + 0~~ 0 + 1 inf + 3 ~~2 + inf~~

Bellman-Ford ejemplo

h	d(B) / s(B)	d(C) / s(C)	d(D) / s(D)	d(E) / s(E)	d(F) / s(F)
0	infinito / desc	infinito / desc	0 / soy yo	infinito / desc	infinito / desc
1	infinito / desc	1 / D	0 / soy yo	infinito / desc	2 / D
2	2 / C	1 / D	0 / soy yo	4 / C	
3					
4					
5 ...					



E elige entre
ir a traves de C 3+d(C) = 4 <<<< este
ir a traves de F 4+d(F) = 6

- Para cada nodo por ejemplo para el E

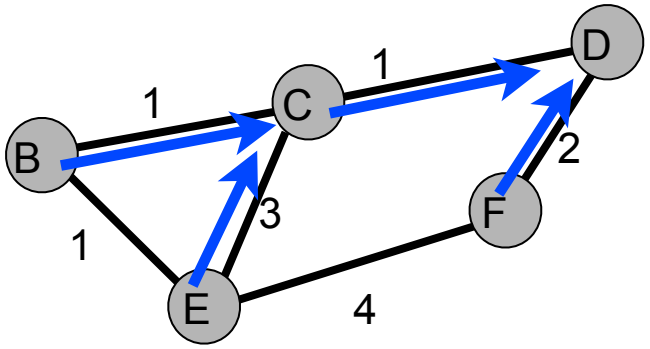
$d_1(B)+w(E,B)$ $d_1(C)+w(E,C)$ $d_1(D)+w(E,D)$ $d_1(E)+w(E,E)$ $d_1(F)+w(E,F)$

inf + 1 1 + 3 0 + inf inf + 0 2 + 4

B no es un candidato porque el no sabe como llegar en la iteracion 1

Bellman-Ford ejemplo

h	d(B) / s(B)	d(C) / s(C)	d(D) / s(D)	d(E) / s(E)	d(F) / s(F)
0	infinito / desc	infinito / desc	0 / soy yo	infinito / desc	infinito / desc
1	infinito / desc	1 / D	0 / soy yo	infinito / desc	2 / D
2	2 / C	1 / D	0 / soy yo	4 / C	2 / D
3					
4					
5 ...					



- Al final de la segunda iteración tenemos el arbol de caminos mínimos a D usando como mucho 2 saltos
- Con este ya tenemos un camino desde cada nodo !!!
- Aunque esta claro que no son los mejores. Por ejemplo de E llegaríamos antes via B
- Cuando acaba el algoritmo???

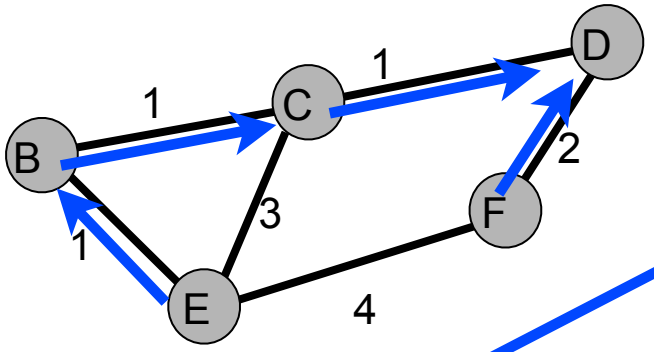
Bellman-Ford ejemplo

h	d(B) / s(B)	d(C) / s(C)	d(D) / s(D)	d(E) / s(E)	d(F) / s(F)
0	infinito / desc	infinito / desc	0 / soy yo	infinito / desc	infinito / desc
1	infinito / desc	1 / D	0 / soy yo	infinito / desc	2 / D
2	2 / C	1 / D	0 / soy yo	4 / C	2 / D
3					
4					
5 ...					

- **Cuando acaba el algoritmo???**
- No vale decir cuando tengamos todos los caminos con coste mínimo. Como hacemos que un programa haga la comprobación?
- El objetivo del algoritmo es saber cuales son los caminos de coste minimo, asi que no podemos usar el conocimiento de cual es el coste minimo para resolverlo
- **Entonces**
- El algoritmo calcula cada fila d_h en funcion de la fila anterior dh-1
- Una vez que una fila se repita... va a seguir repitiendose eternamente...
- Esa es la condición que nos dice que el algoritmo ya no va a encontrar caminos mejores
- Bellman-Ford demostraron que cuando pasa eso ya ha encontrado los mejores

Bellman-Ford ejemplo

h	d(B) / s(B)	d(C) / s(C)	d(D) / s(D)	d(E) / s(E)	d(F) / s(F)
0	infinito / desc	infinito / desc	0 / soy yo	infinito / desc	infinito / desc
1	infinito / desc	1 / D	0 / soy yo	infinito / desc	2 / D
2	2 / C	1 / D	0 / soy yo	4 / C	2 / D
3	2 / C	1 / D	0 / soy yo	3 / B	
4					
5 ...					



Hay un mejor camino por B con distancia 3

- Iteración 3 para el E

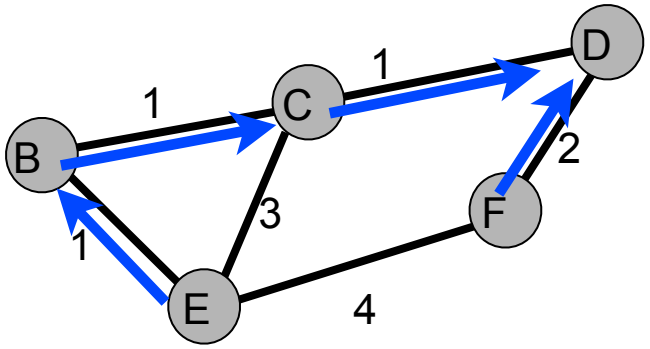
$d_3(B)+w(E,B)$	$d_3(C)+w(E,C)$	$d_3(D)+w(E,D)$	$d_3(E)+w(E,E)$	$d_3(F)+w(E,F)$
2 + 1	1 + 3	0 + inf	4 + 0	2 + 4

E elige entre ir por cualquiera de sus vecinos que ya tienen una distancia a D

Aunque en este caso es ya el mínimo no tendría por que ser

Bellman-Ford ejemplo

h	d(B) / s(B)	d(C) / s(C)	d(D) / s(D)	d(E) / s(E)	d(F) / s(F)
0	infinito / desc	infinito / desc	0 / soy yo	infinito / desc	infinito / desc
1	infinito / desc	1 / D	0 / soy yo	infinito / desc	2 / D
2	2 / C	1 / D	0 / soy yo	4 / C	2 / D
3	2 / C	1 / D	0 / soy yo	3 / B	2 / D
4	2 / C	1 / D	0 / soy yo	3 / B	2 / D
5 ...					



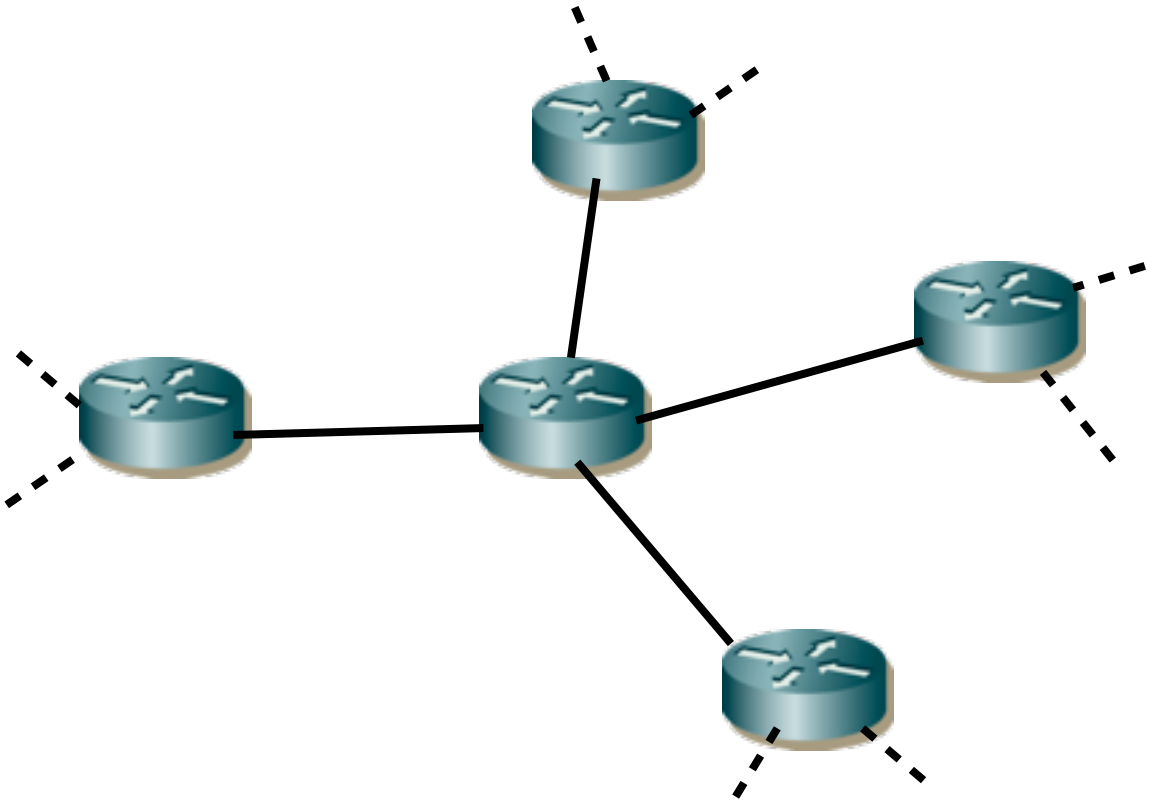
- Tras la tercera iteración ya tenemos el arbol de caminos minimos con 3 saltos
- Que ya es el definitivo pero el algoritmo no lo sabe
- Al hacer la iteración 4 no hay ningún cambio
- Esa es la condición de que ya ha terminado

Algoritmos de caminos mínimos

- Dijkstra vs Bellman-Ford
  - Los dos calculan el árbol de expansión mínimo para una raiz dada
  - Los dos algoritmos dan el mismo resultado
  - El resultado no tiene por que ser único (probad a hacer los ejemplos anteriores cambiando el peso de C-E a 2)
- Cuál es más rápido?
  - Parece que Bellman-Ford hace menos iteraciones pero las iteraciónes de Dijkstra parecen más cortas y rápidas
- Cuál preferiríais programar?
- Normalmente se suele considerar mejor el algoritmo de Dijkstra para resolver el problema de los caminos en un grafo
- Pero no todo es la velocidad del algoritmo...

Como usar esto en la realidad

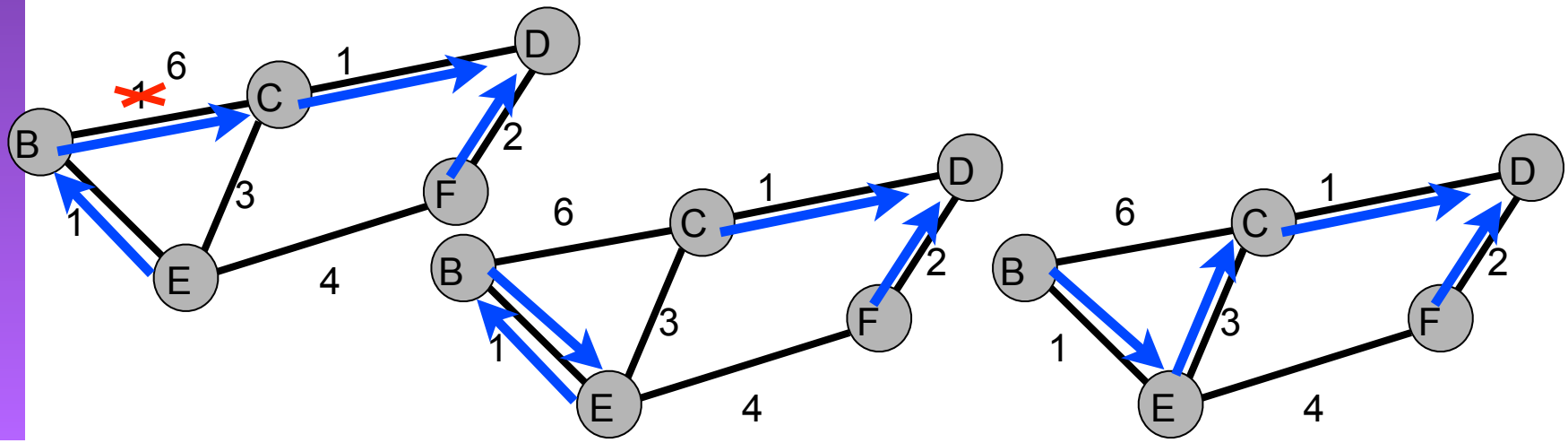
- El problema del cruce de caminos... En cada router
- Como calculo las rutas a todos los destinos?
- Solo con información mía o que pueda obtener de los vecinos
- Ni siquiera se cuales son todos los destinos !!!



Bellman-Ford continuo...

h	d(B) / s(B)	d(C) / s(C)	d(D) / s(D)	d(E) / s(E)	d(F) / s(F)
12032	2 / C	1 / D	0 / soy yo	3 / B	2 / D
12033	2 / C	1 / D	0 / soy yo	3 / B	2 / D
12034	4 / E	1 / D	0 / soy yo	3 / B	2 / D
12035	4 / E	1 / D	0 / soy yo	4 / C	2 / D
12036	5 / E	1 / D	0 / soy yo	4 / C	2 / D
12037...	5 / E	1 / D	0 / soy yo	4 / C	2 / D

- Si hay un cambio? Enlace B-C cambia a peso 6
- parece que se adapta a el ... en unos pocos turnos



Distance-Vector

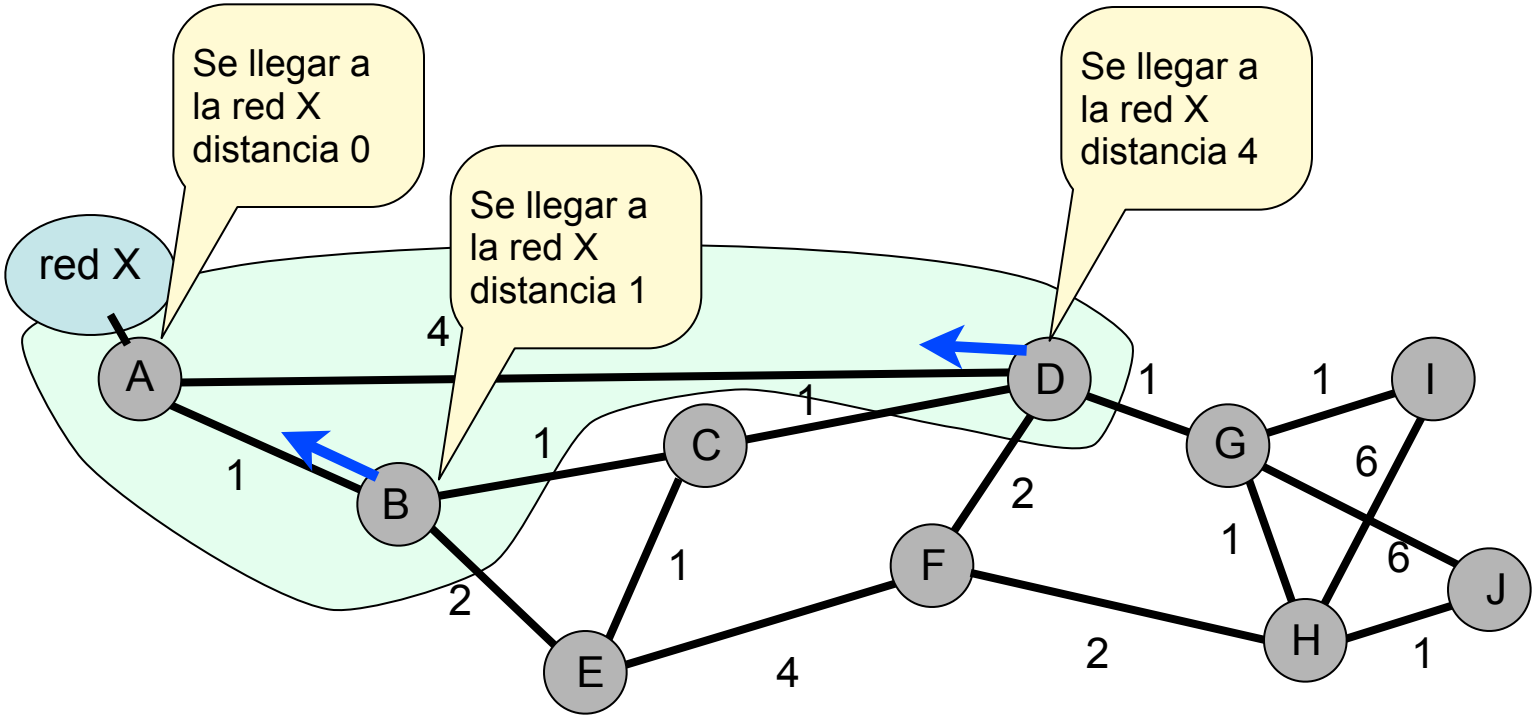
- La idea es la base para construir algoritmos de enrutamiento distribuidos
- Enrutamiento Distance-Vector
  - Cada router esta configurado con un identificador
  - Cada router esta configurado con los enlaces a sus vecinos y cada enlace tiene por configuración un numero para usar como coste del enlace
  - Cada router mantiene un vector de distancias a cada destino, se inicializa con 0 para si mismo y infinito para cualquier otro destino
  - Mantiene también otro vector con el siguiente salto a cada destino (tabla de rutas)
  - Cada router es capaz de comunicarse con sus vecinos y envía el vector de distancias a sus vecinos (cada vez que hay un cambio o periodicamente)
  - Cada router almacena el vector de distancias más reciente que ha recibido de cada vecino y utiliza la iteración de Bellman-Ford para decidir cual es el la mejor distancia y el mejor siguiente salto a ese destino.
  - Con eso actualiza su distancia y siguiente salto hacia ese destino
  - Cuando se recalcula el vector?
    - Cada vez que recibo de la red información nueva (nuevo vector)
    - Cada vez que cambia el peso de un enlace, o desaparece (cambio a infinito)
- Es una aproximación distribuida al algoritmo de Bellman-Ford
  - Salvo que no hay exactamente iteraciones o turnos...
  - De hecho (solo mensajes de algunos vecinos) parece razonable
  - Pero hay algunos problemas...

Distance-Vector

- La idea es la base para construir algoritmos de enrutamiento distribuidos
- Enrutamiento Distance-Vector
  - Cada router mantiene un vector con la distancia a cada destino
  - Y otro con el siguiente salto a cada destino
  - Cada router es capaz de comunicarse con sus vecinos y envía regularmente el vector de distancias a sus vecinos
  - Cuando un router acumula las distancias de todos sus vecinos a un destino puede utilizar la iteración de Bellman-Ford para decidir cual es el mejor siguiente salto a ese destino.
  - Con eso actualiza su distancia y siguiente salto hacia ese destino
  - Y lo repite para todos los
- Es una aproximación distribuida al algoritmo de Bellman-Ford
  - Salvo que nos esforcemos en sincronizar los envíos de mensajes no hay exactamente iteraciones o turnos... pero más o menos funciona
  - Incluso con información incompleta (solo mensajes de algunos vecinos) parece razonable
  - Pero hay algunos problemas...

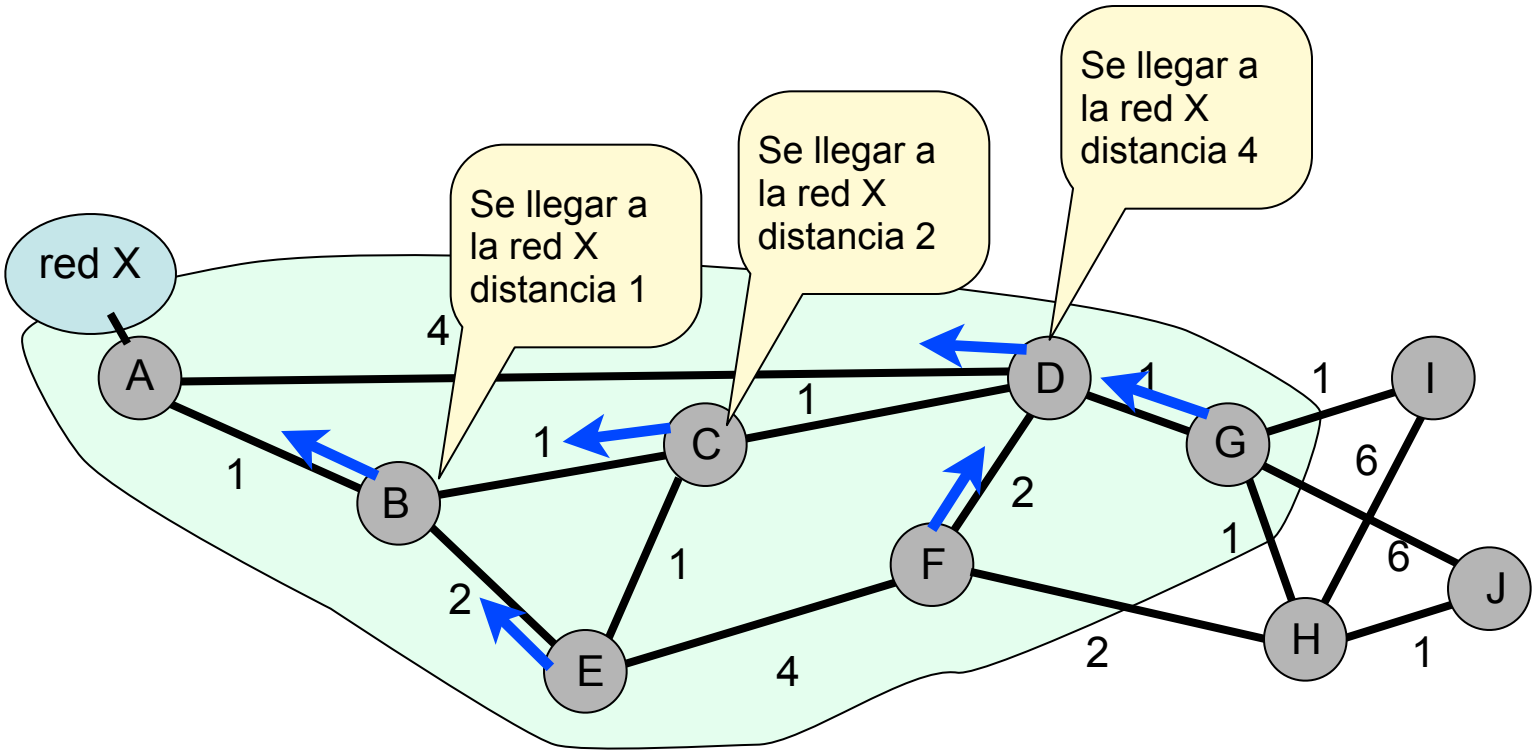
Y esto funciona?

- La información para cada destino se propaga desde los routers que la saben...
-



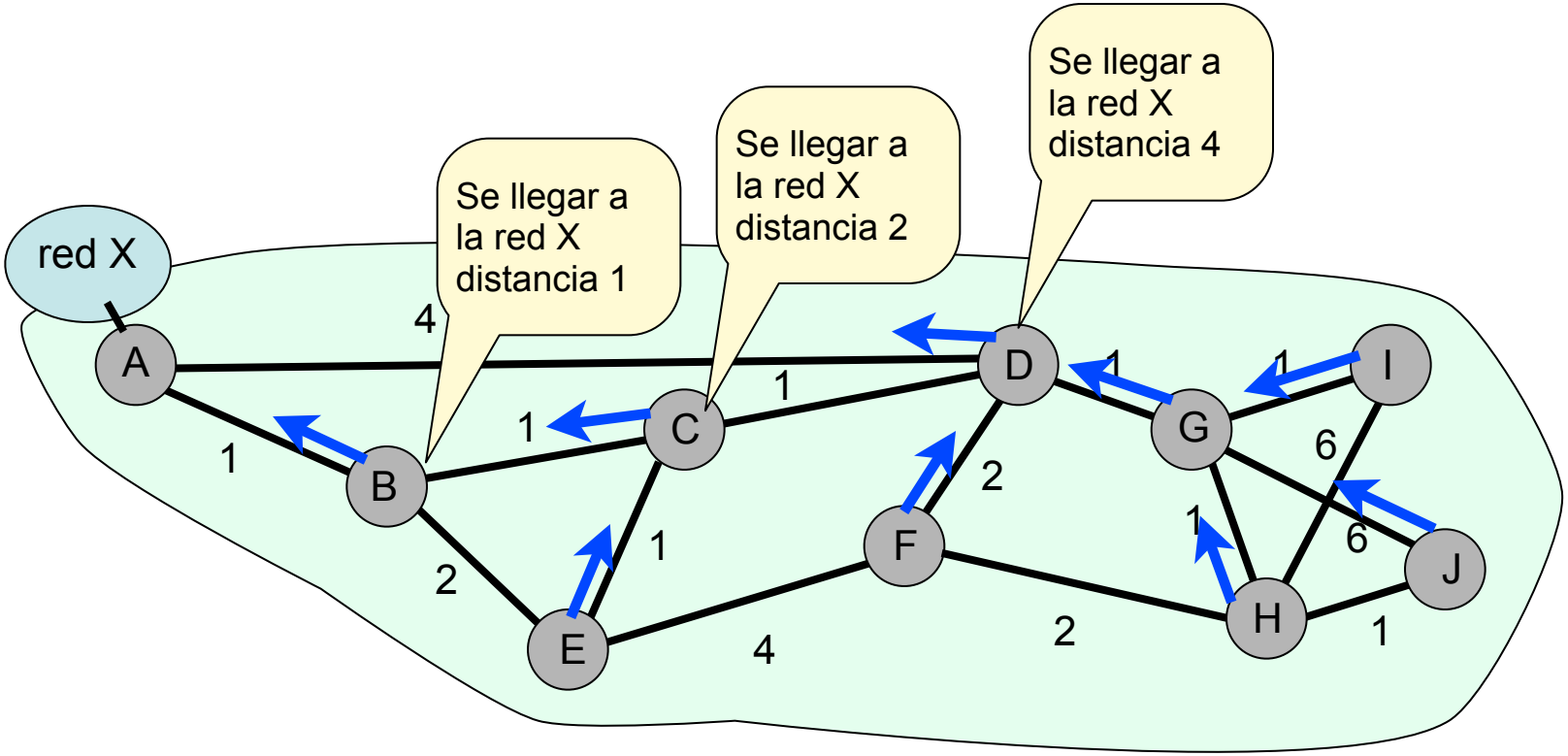
Y esto funciona?

- La información para cada destino se propaga desde los routers que la saben...
-



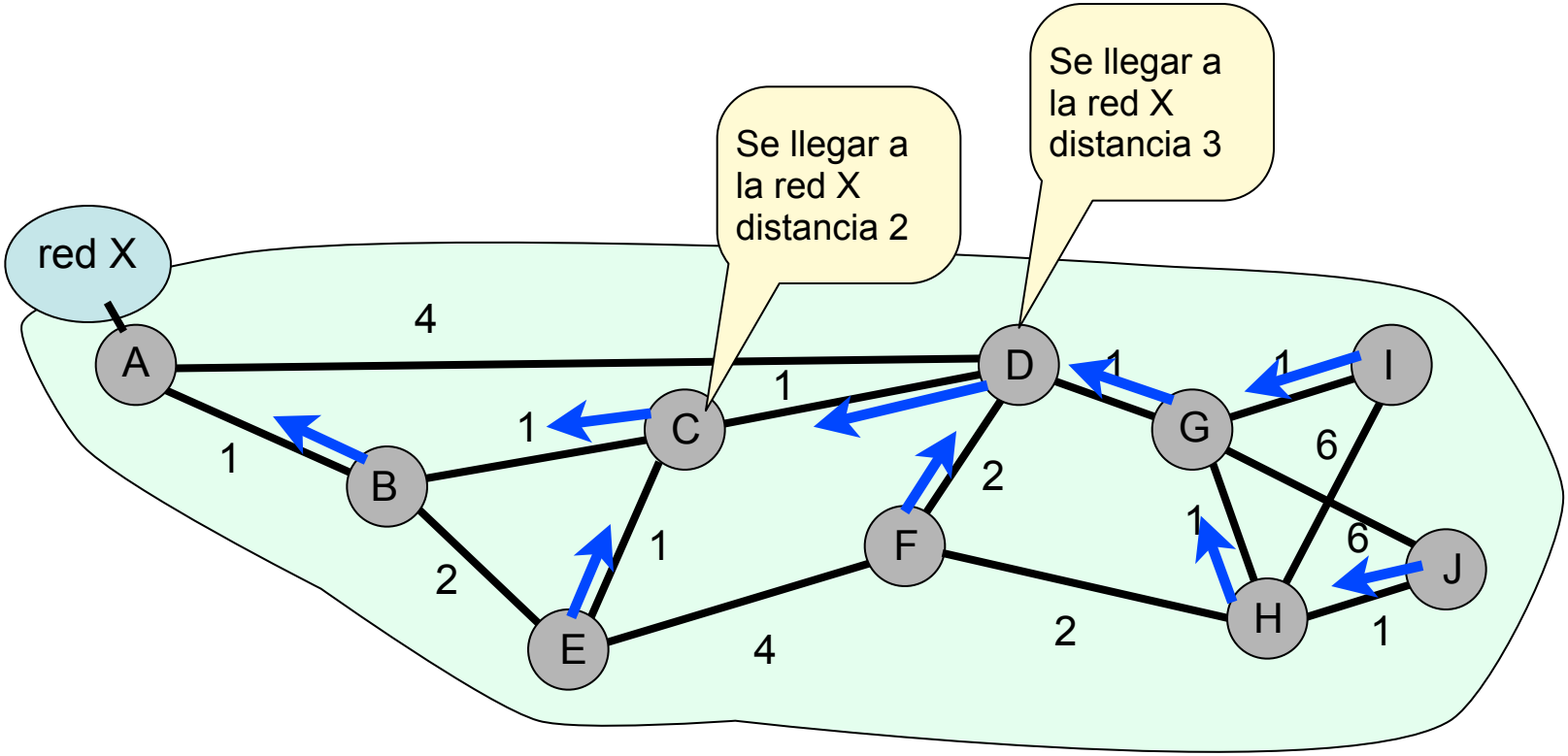
Y esto funciona?

- La información para cada destino se propaga desde los routers que la saben...
- El tiempo de propagación depende de cuantos routers hay hasta el destino
- El tiempo de propagación no es tanto si cada router envía la información a sus vecinos cada vez que hay cambios (triggered updates)



Y esto funciona?

- La información para cada destino se propaga desde los routers que la saben...
- No necesariamente la mejor ruta es la que oigo la primera vez por eso el algoritmo debe funcionar de forma continua
- Lo mismo pasa a la vez con todos los demas destinos



Y esto funciona?

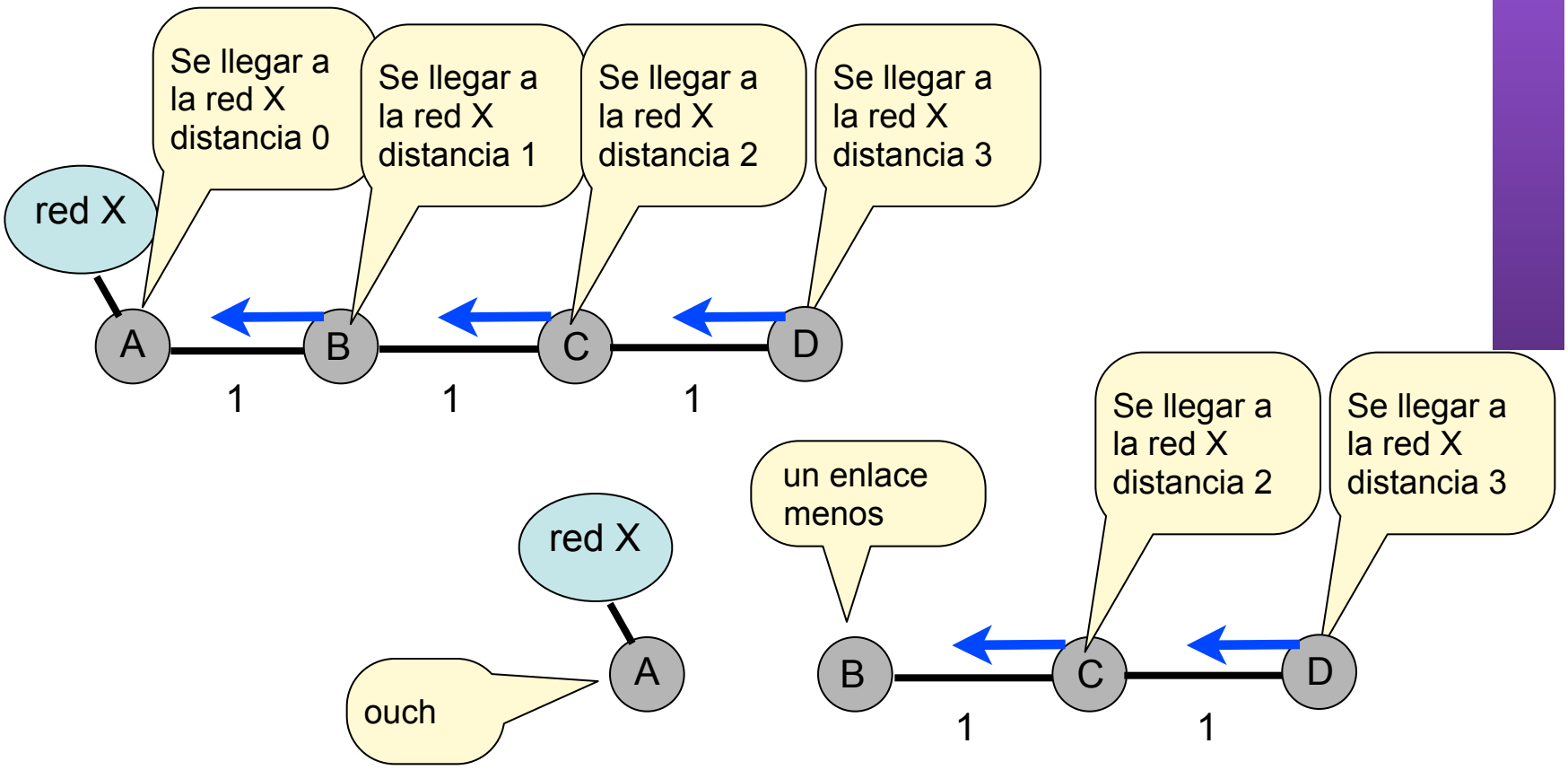
- El algoritmo no sabe cuando ha terminado
- Pero no importa mucho que se ejecute de forma continua
- Si usamos envío de información periódica controlamos el trafico de enrutamiento que se envía... pero el tiempo en propagar rutas es mas largo
- Si enviamos en cuanto hay cambios (triggered updates) la propagación es rapida y se envía más tráfico cuando hay un cambio pero es self-stoping se autocontrola y deja de enviar cuando las rutas se estabilizan

- Normalmente se utiliza triggered updates con y envio periodico no demasiado frecuente
  - Envío rapido de cambios
  - El envio periodico ayuda si se pierden mensajes o para descubrir vecinos cuando un router se conecta a la red

- Parece razonable. Funciona, se propaga rapido y no crea mucho trafico...
- Y entonces por que es el sistama de **enrutamiento *antiguo* de Internet**

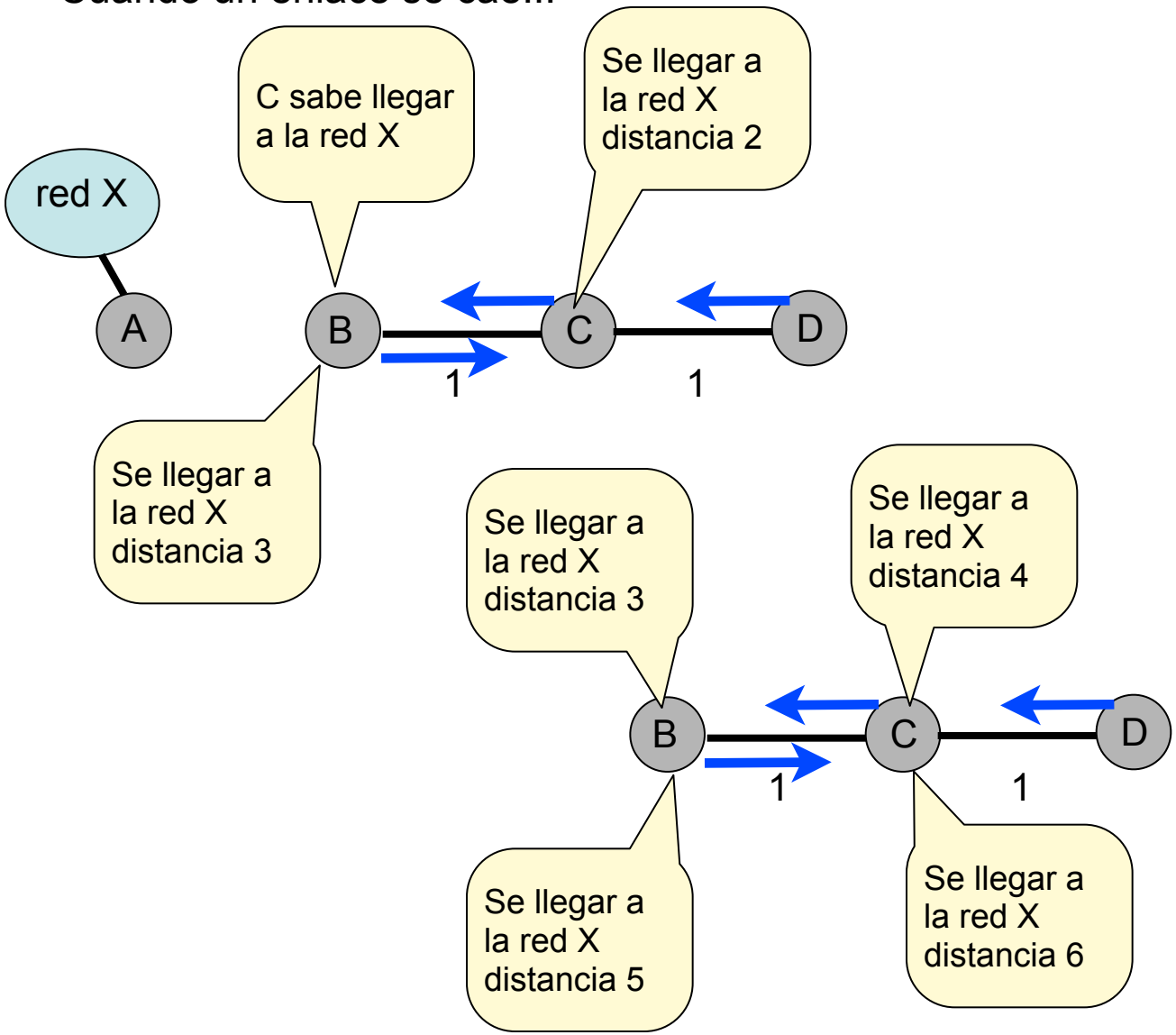
Problemas

- Es lento en reaccionar !!!
- No todos los cambios se propagan rápido hay situaciones anómalas
- Un caso muy simple y bien conocido
- Qué pasa si se cae el primer enlace?



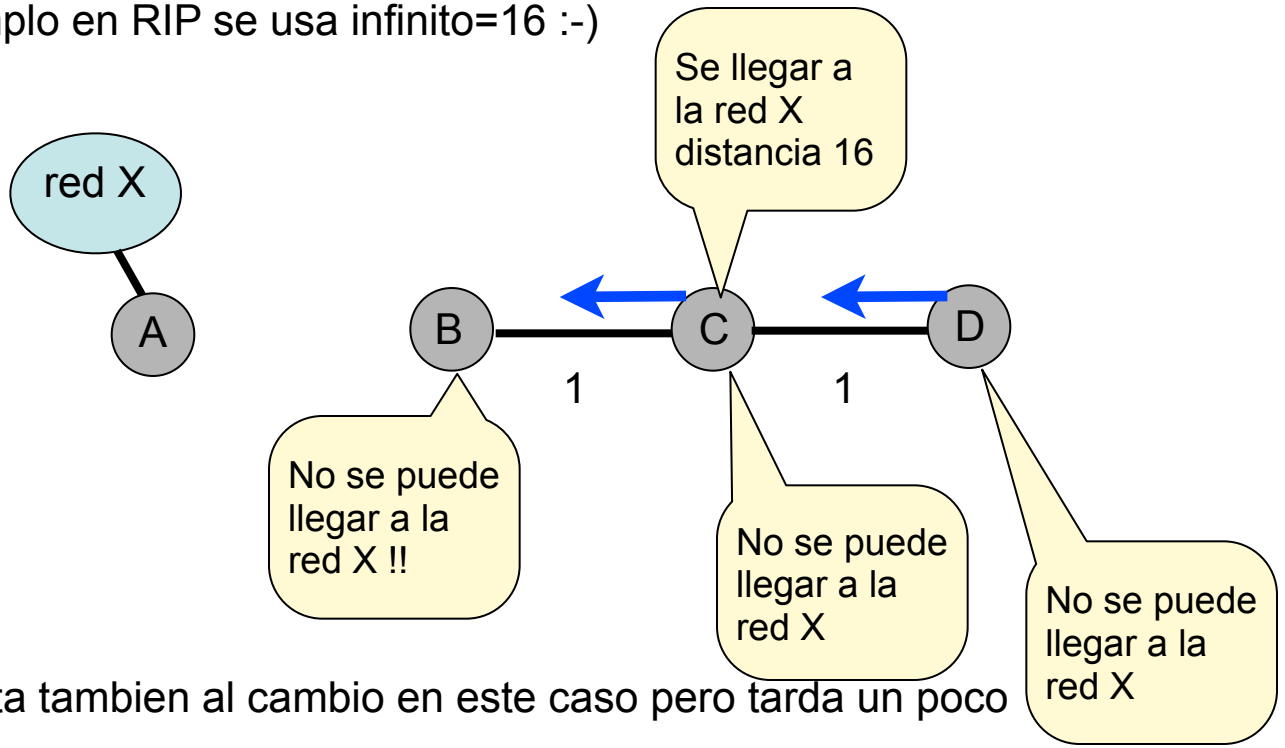
La cuenta a infinito

- Cuando un enlace se cae...



La cuenta a infinito

- Como acaba esto?
- Se hace que el campo para indicar la distancia tendrá bits limitados
  Cuando llegue al máximo se considera infinito y la ruta se descarta
  Establcer un valor de infinito pequeño hace que estos casos se detecten antes
  Pero entonces solo podremos calcular distancias menores a ese valor
  Por ejemplo en RIP se usa infinito=16 :-)



- Se adapta tambien al cambio en este caso pero tarda un poco y genera unos cuantos mensajes de actualización en el proceso

Problemas distance-vector

- Las cuentas a infinito hacen que los protocolos distance-vector puedan tardar en bastante en converger a la solución
- Mientras convergen las rutas puede no ser buenas

  (ciclos de enrutamiento y perdidas)

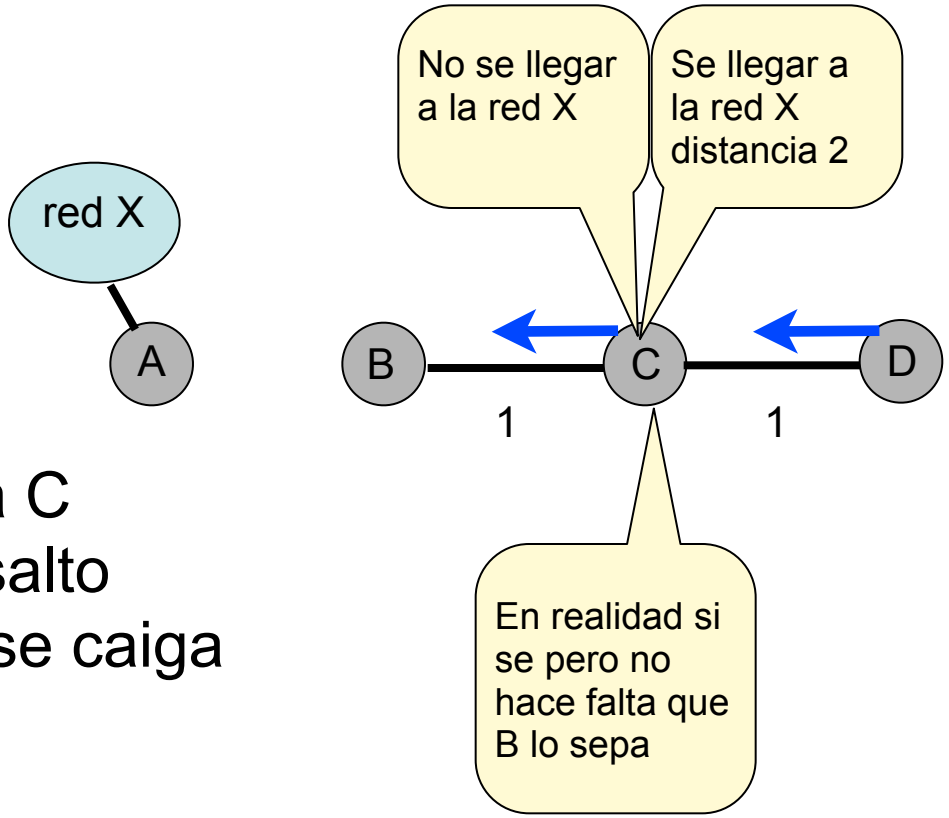
- Se puede resolver el problema de las cuentas a infinito?

  Hay algunas optimizaciónes que parecen obvias...
  - Mejor no anunciar una ruta a un nodo si la ruta pasa por el (split-horizon)
  - Cuando una ruta se vuelve es descartada no aceptar nuevas (hold-down)
  - ...

Soluciones: split horizon

- No anuncio las rutas a un destino a mi siguiente salto para ese destino

  El deberia saber como llegar porque yo le estoy mandando a el los paquetes que van a ese destino
- Esto resuelve el escenario anterior

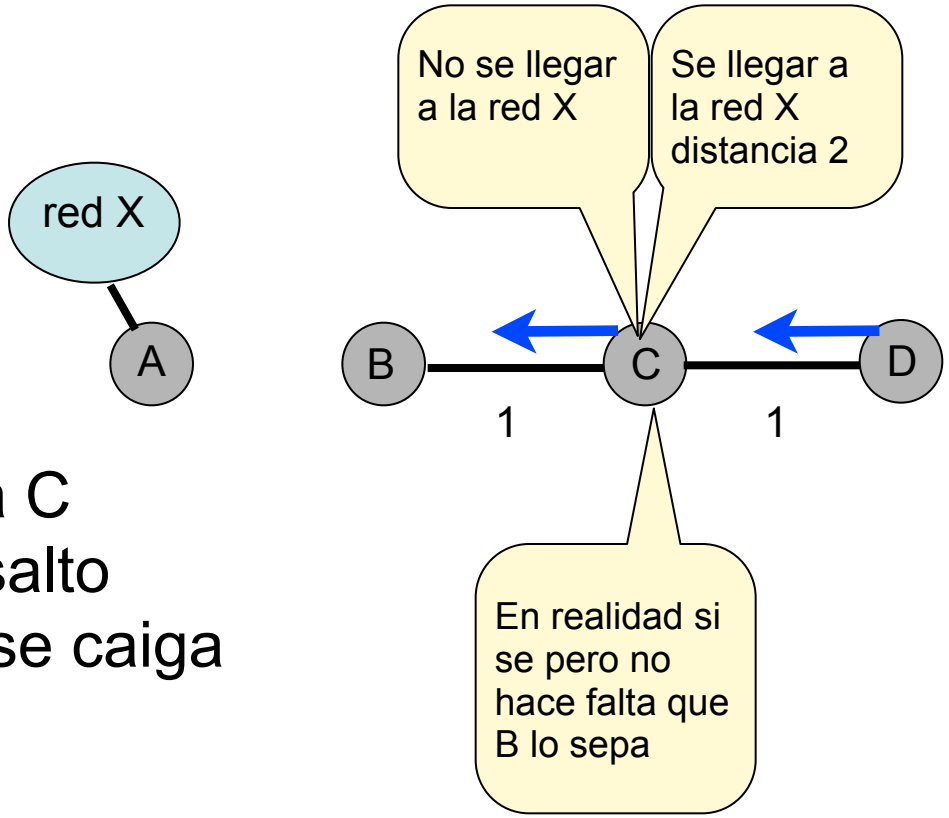


B ya no elegira a C como siguiente salto hacia X cuando se caiga el enlace

Soluciones: split horizon

- No anuncio las rutas a un destino a mi siguiente salto para ese destino

  El deberia saber como llegar porque yo le estoy mandando a el los paquetes que van a ese destino
- Esto resuelve el escenario anterior



B ya no elegira a C como siguiente salto hacia X cuando se caiga el enlace

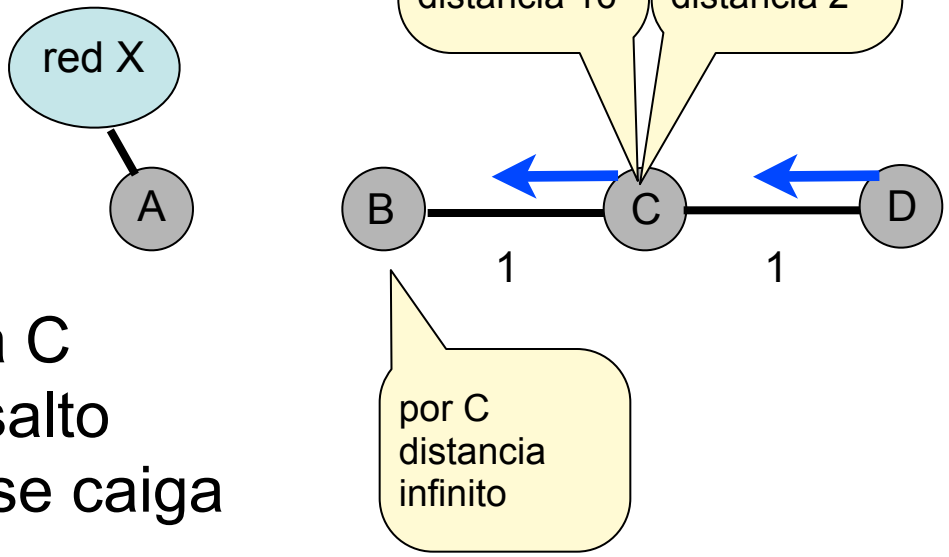
Soluciones: poison reverse

- Al siguiente salto para un destino le miento y le digo que la distancia es infinito

  Parecido a Split-horizon

  Un poco mejor porque si por alguna otra cosa habia un ciclo de enrutamiento lo rompe

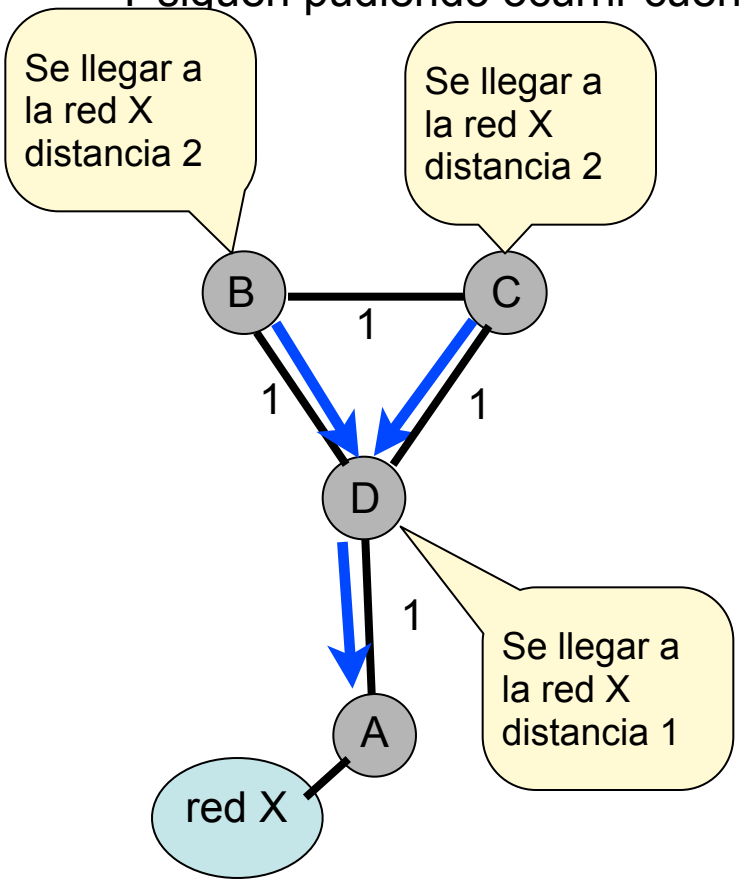
- Tambien resuelve el escenario anterior

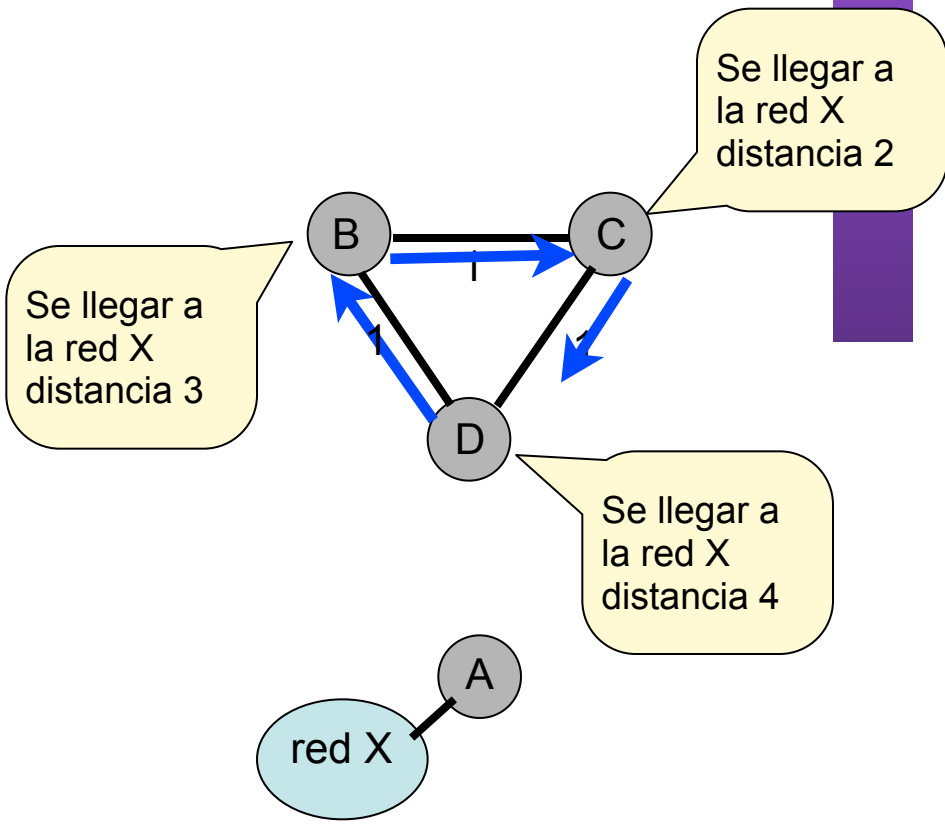


B ya no elegira a C como siguiente salto hacia X cuando se caiga el enlace

Split-horizon/poison-reverse

- Fáciles de implementar y resuelven el escenario anterior
- Tienen problemas en redes de area local cuando envio anuncios a broadcast como enveneno las rutas a mi siguiente salto pero no a los demas
- Y siguen pudiendo ocurrir cuentas a infinito de otros tipos





Más soluciones

- Hold-down: si una ruta se hace invalida no aceptar nuevos caminos a ese destino en un tiempo de hold-down

  Confia en que en ese tiempo la no alcanzabilidad se extienda a toda la red. si no da tiempo y algún nodo tiene aun ruta al destino puede crearse una cuenta a infinito
- Anunciar la distancia y el siguiente salto ~parecido a poison reverse
- Anunciar la distancia y el camino entero

  Este funciona bien pero requiere que los routers almacenen todo el camino para cada destino. No escala bien
- Usar dos métricas y mantener dos distancias

  Una para comparar caminos y la otra con el numero de saltos para las cuentas a infinito
- DUAL (Diffusing Update Algorithm)

  razonamientos con las distancias para decidir si es posible o no que el camino anunciado pase por ti

  i.e. si la distancia que anuncia C es menor que la que yo tenia al destino antes de hacerse invalida la ruta es seguro cambiar a C

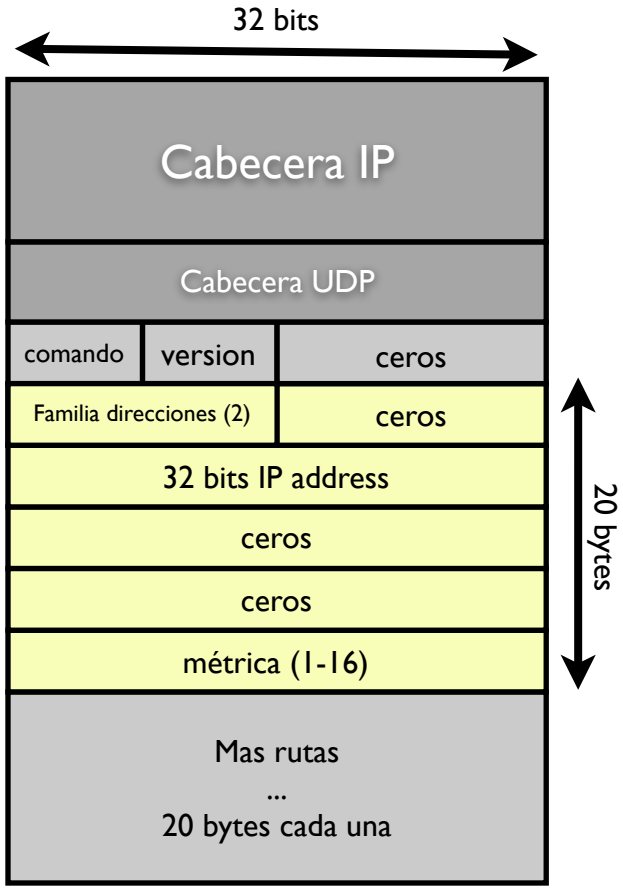
Resumen hasta ahora

- Algoritmo distance-vector teorico
- Calculo de rutas distribuidas
- Adaptación a los cambios
- Convergencia rápida a los cambios y auto-parada en algunos casos
  - Normalmente los cambios a mejor se propagan rápido
- Problemas de convergencia/estabilidad y cuentas a infinito en otros casos
  - Normalmente los cambios a peor se propagan despacio
- Soluciones que alivian estos problemas pero no los resuelven totalmente

- Lo suficiente para que los algoritmos distance-vector sean útiles
- Qué protocolos reales distance-vector se utilizan? (RIP, IGRP, EIGRP...)
- Como conseguir algoritmos con menos problemas de convergencia?
  (La semana que viene)

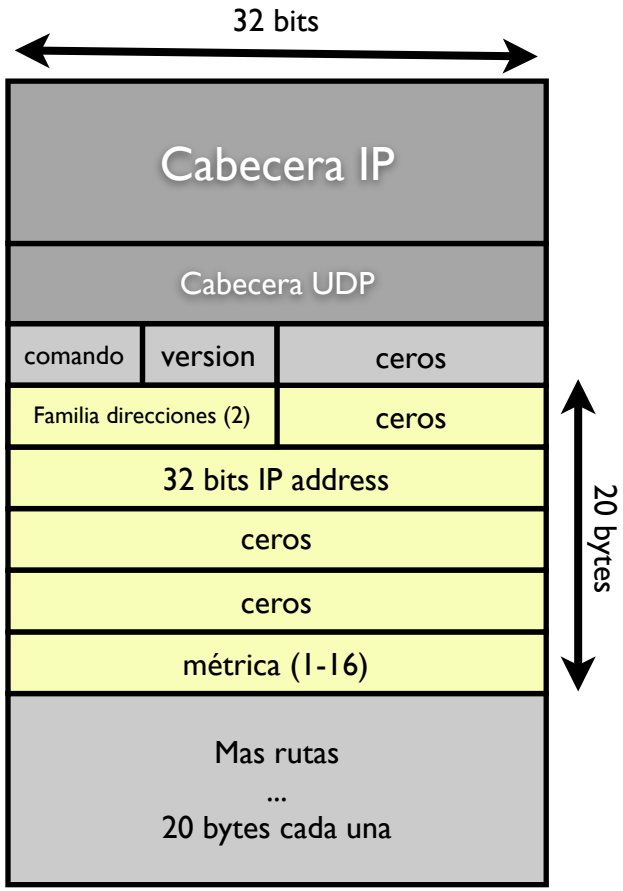
Protocolos reales: RIP

- RIP Routing Information Protocol (RC 1058)
- Protocolo simple de enrutamiento tipo distance-vector con estos principios. Ya existía para cuando se escribio el RFC. Cambios menores entre implementaciones que interoperan entre si. Distance-vector es simple y robusto
- Formato de mensaje del protocolo
- Versión: 1 o 2
- Comando =1 **request** =2 **response**
  - Enviar vector
  - Pedir vector o unos destinos concretos
- La **response** se envían
  - Periodicamente (30 s)
    si un router no anuncia un destino
    durante un tiempo se elimina
  - En respuesta a una query
  - Opcionalmente cuando cambia el vector
    (triggered update) opcional en RIPv1



Protocolos reales: RIPv1

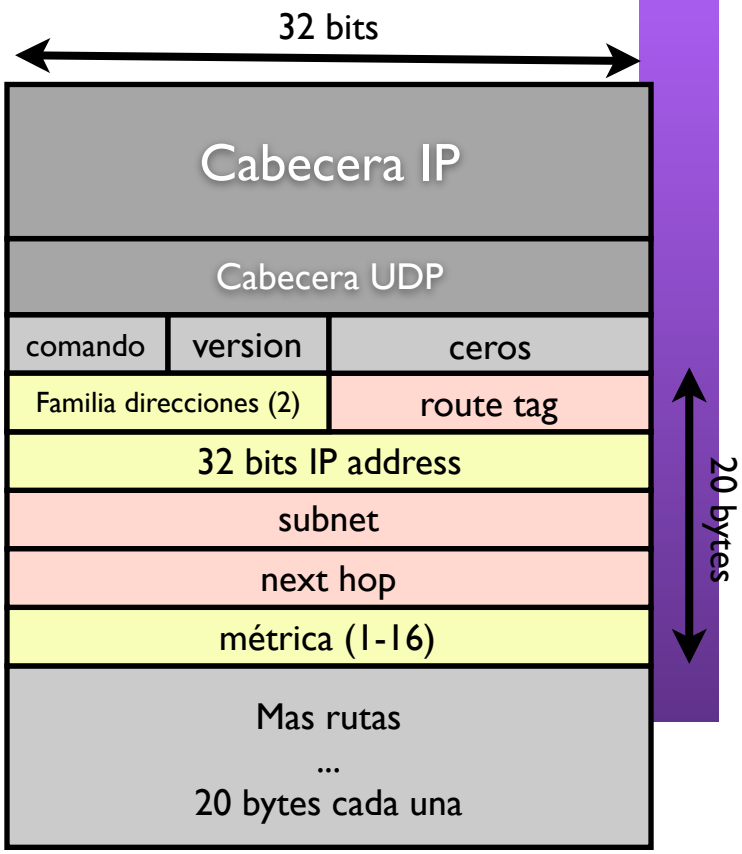
- Hasta 25 elementos de 20bytes
  - Familia de direcciones: AF_INET=2
  - Dirección destino (14 bytes)
    en IPv4 solo se usan 4
    Es la dirección de una subred
    Sin máscara !!!
    Eran los tiempos de las direcciones con clases A, B, C...
  - Métrica: distancia (4bytes)
    limitada a 1-16 por las cuentas a infinito



- Los mensajes se envian sobre UDP al puerto 520 (y desde el 520 salvo las resuestas solicitadas) y a la dirección de broadcast para que la oigan todos los vecinos
- Es aceptable un agente RIP silencioso que solo escucha, si no eres un router
- RIPv1 muy popular aunque un poco lento sobre si no implementa triggered-updates

RIPv2

- Compatible con v1
- Anuncia red/mascara/siguiente salto
  - Soporta mascara variable
  - Soporta anunciar rutas de parte de otro
- Soporta etiquetar las rutas (route tag)
  - Distinguir rutas generadas por RIP de rutas prestadas por otro protocolo y que se están anunciando por RIP
- Soporta autentificación
  - primera entrada se marca con familia=0xFFFF
    2 bytes tipo de auth
    16 bytes auth data



- Los mensajes se envían a una dirección de multicast (por defecto la 224.0.0.9 pero se puede cambiar) para no cargar a los vecinos no interesados en el enrutamiento.

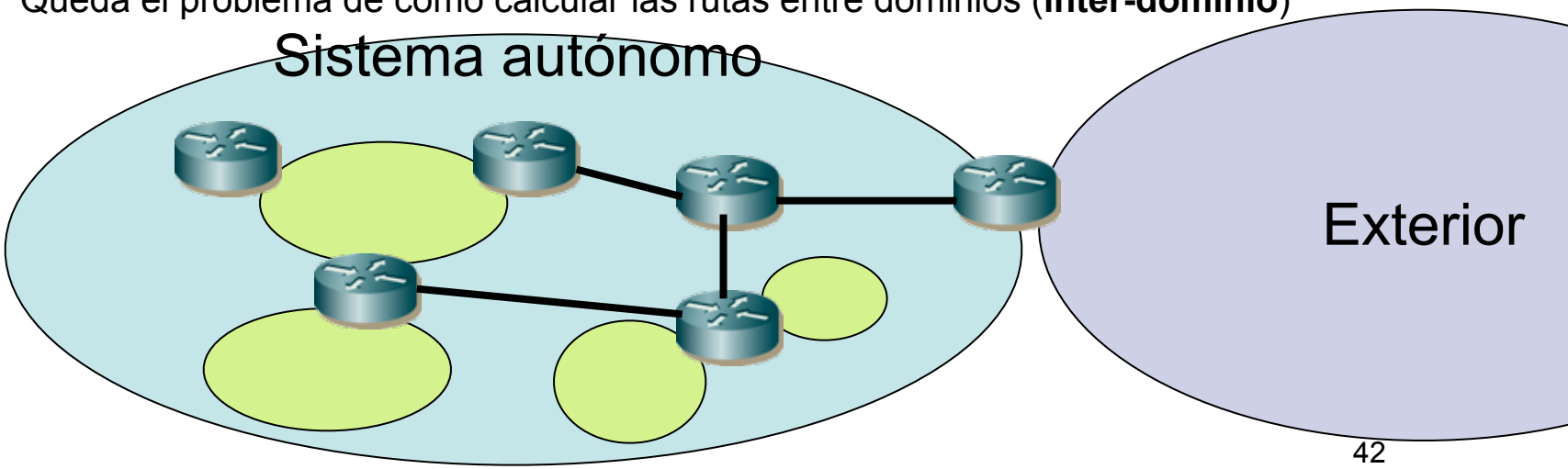
Otros protocolos distance-vector

- IGRP: Interior Gateway Routing Protocol
  - De CISCO
  - Distance vector
  - Metrica configurable en función de propiedades del enlace (ancho de banda, retardo, MTU, carga...)
  - Sin máscara: direcciones con clases
- EIGRP: Extended Interior Gateway Routing Protocol
  - Basado en el algoritmo DUAL (Diffusing Update Algorithm)

    Vector-distance pero con técnicas de evitación de ciclos y convergencia rápida

Intradomain routing

- Los algoritmos distance-vector son útiles en redes no demasiado complicadas ni grandes.
  - Añadir routers y enlaces automaticamente sin tener que configurar tablas de rutas
  - Reaccionar a los cambios, aunque tarden un poco. Si la red no es muy complicada tampoco hay tantas cuentas a infinito
- Una red autonoma puede usar un algoritmo distance-vector para propagar sus destinos internos. Los routers frontera simplemente anuncian el exterior como un destino único
  - Con eso configuro automaticamente todas las rutas de una red de una entidad
  - Sistema autónomo
  - El enrutamiento de este tipo se llama **intra-dominio**
- Queda el problema de como calcular las rutas entre dominios (**inter-dominio**)



Conclusiones

- Protocolos distance-vector permiten construir enrutamiento adaptativo distribuido
  - Basados en el algoritmo de Bellman-Ford
  - Con ciertos problemas de convergencia
  - Y soluciones parciales
  - Son útiles para redes no demasiado complicadas
  - Se utilizan como intra-domain routing
- Protocolos reales
  - RIPv1 y RIPv2
  - IGRP y EIGRP
- Próxima clase:

  Otro tipo de algoritmos de enrutamiento: link-state