Enrutamiento (2)

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

Arquitectura de Redes, Sistemas y Servicios
3º Ingeniería de Telecomunicación

Basadas en el material docente de Lawrie Brown sobre el libro de William Stallings (Data and Computer Communications)

Temario

1. Introducción
2. Protocolos y arquitectura
3. Redes de área local
4. Protocolos de Internet
5. Conmutación de paquetes
6. Conmutación de circuitos
7. Gestión de recursos en conmutadores
8. Protocolos de control de acceso al medio

Temario

1. Introducción
2. Protocolos y arquitectura
3. Redes de área local
4. Protocolos de Internet
5. Conmutación de paquetes
   - Principios
   - **Problemas básicos**
     - **Encaminamiento (y van 2)**
     - Transporte fiable
     - Control de flujo
     - Control de congestión
6. Conmutación de circuitos
7. Gestión de recursos en conmutadores
8. Protocolos de control de acceso al medio

En clases anteriores…

- Conmutación de circuitos y de paquetes
- El problema del enrutamiento
- Técnicas de enrutamiento básicas
  - Enrutamiento estático
  - Inundación
  - Enrutamiento aleatorio
  - Enrutamiento de mínimo coste
    - Algoritmo de Dijkstra (de mínimo coste)

Hoy

- Algoritmo de Bellman-Ford
  (el otro algoritmo clásico de mínimo coste)
- Historia del enrutamiento en ARPANET
- De los algoritmos de grafos al enrutamiento
- Enrutamiento en Internet

Algoritmo de Bellman-Ford

- Encontrar los **caminos mas cortos** desde un nodo origen dado, sujetos a la restricción de que **como mucho tengan un enlace (1 salto)**
- Encontrar los **caminos mas cortos** desde un nodo origen dado, sujetos a la restriccion de que **como mucho tengan dos enlace (2 saltos)**
- Encontrar los **caminos mas cortos** desde un nodo origen dado, sujetos a la restriccion de que **como mucho tengan h enlaces (h saltos)**

  Es facil si se conocen los caminos para el caso h-1
- …

Algoritmo de Bellman-Ford

- Variables del algoritmo
- Conjunto de nodos E={n_i nodos en el grafo}
- Pesos de los enlaces

  $w(n_i, n_j)$ peso del enlace de n_i a n_j

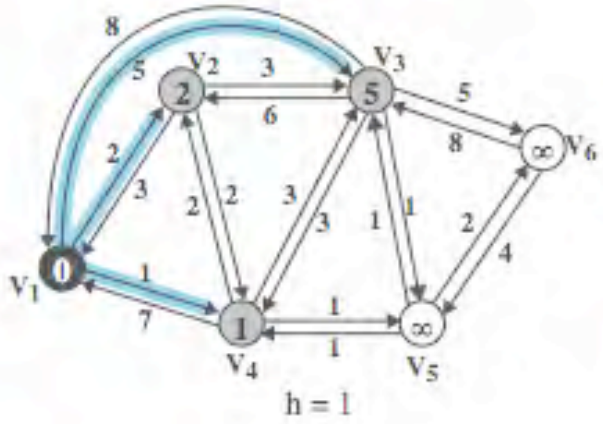
  en general distinto de $w(n_i, n_j)$

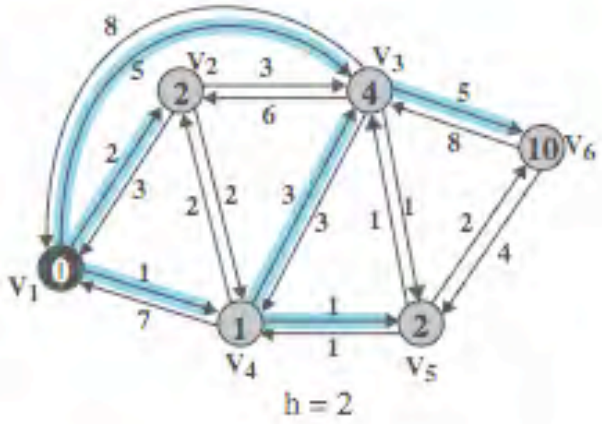
  $w(n_i, n_j)$ = infinito si no hay enlace
- s nodo origen del que pretendemos calcular los caminos
- h número de iteración del algoritmo = máximo número de enlaces que se consideran
- $L_h(n_i)$ distancia mínima de s a n_i considerando solo caminos que tengan como mucho h enlaces

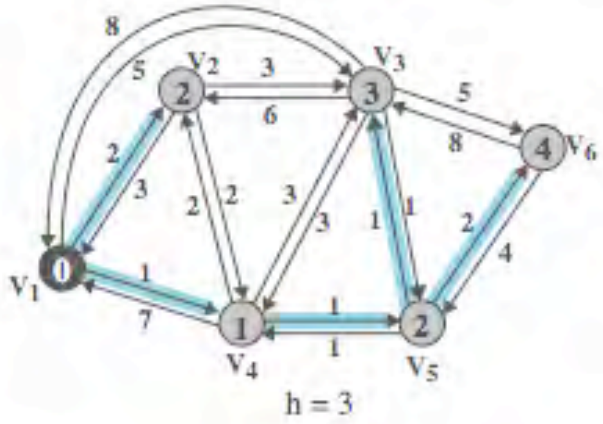
Algoritmo de Bellman-Ford

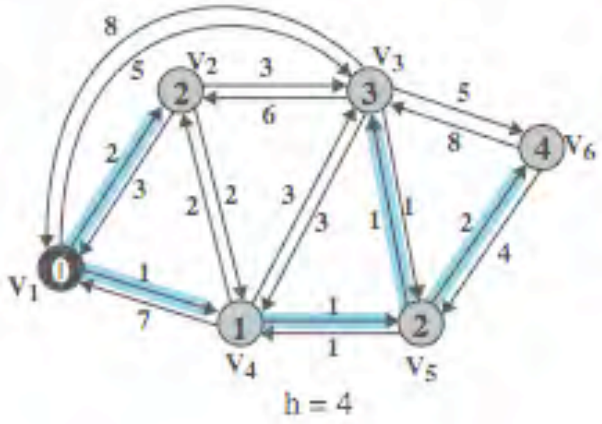
- Paso 1 [Inicialización]
  - $L_0(n) = \infty$, para todo $n \neq s$
  - $L_h(s) = 0$, para todo h
- Paso 2 [Actualización]
  - Para cada $h \geq 0$ sucesivo
    - Para cada $n \neq s$, : $L_{h+1}(n)=\min_j[L_h(j)+w(j,n)]$
  - Conectar n con el predecesor j que de menor camino
  - Eliminar la conexión de n con predecesores de una iteracion anterior
  - El camino nuevo es el camino para llegar al nodo elegido j al que se ha añadido el enlace de j a n
  - Si en una iteración no hacemos ningún cambio el algoritmo ha terminado

Ejemplo

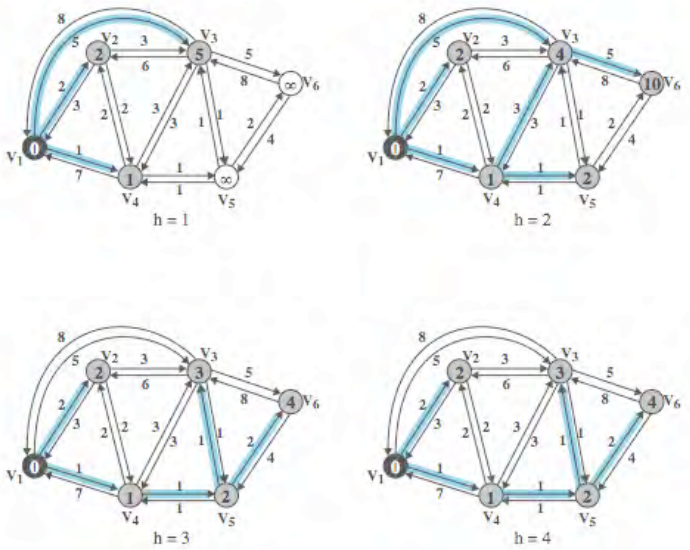








Ejemplo



h	$L_h(2)$	Path	$L_h(3)$	Path	$L_h(4)$	Path	$L_h(5)$	Path	$L_h(6)$	Path
0	∞	-	∞	-	∞	-	∞	-	∞	-
1	2	1-2	5	1-3	1	1-4	∞	-	∞	-
2	2	1-2	4	1-4-3	1	1-4	2	1-4-5	10	1-3-6
3	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6
4	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

Resumen Bellman-Ford

- Funciona, si hay camino más corto lo calcula (Si el grafo es conexo)
- Calcula los mismos caminos que Dijkstra (También da un spanning tree centrado en el nodo que queremos)
- También necesita la topología completa, pero…
- Cual es más rápido?

  Se acepta que desde un punto de vista algorítmico Dijkstra es más rápido, acaba en menos operaciones (no necesariamente en menos iteraciones)
- Pero hay otros factores…

Prestaciones para uso en enrutamiento

- Depende de:
  - Tiempo de proceso de los algoritmos
  - Cantidad de información requerida de otros nodos
- Es especifico de la implementación del algoritmo

Esta demostrado

- Los dos convergen bajo condiciones de topología estática (no cambian los enlaces ni sus pesos)
- Los dos convergen a la misma solución
- Si los enlaces cambian los algoritmos tratan de recalcular los caminos (Spanning-Tree)
  - Dijkstra tiene que empezar de nuevo las iteraciones
- Si los costes dependen del tráfico en los enlaces (enrutamiento sensible a la congestion), el tráfico depende a su vez de los caminos elegidos lo que lleva a inestabilidades

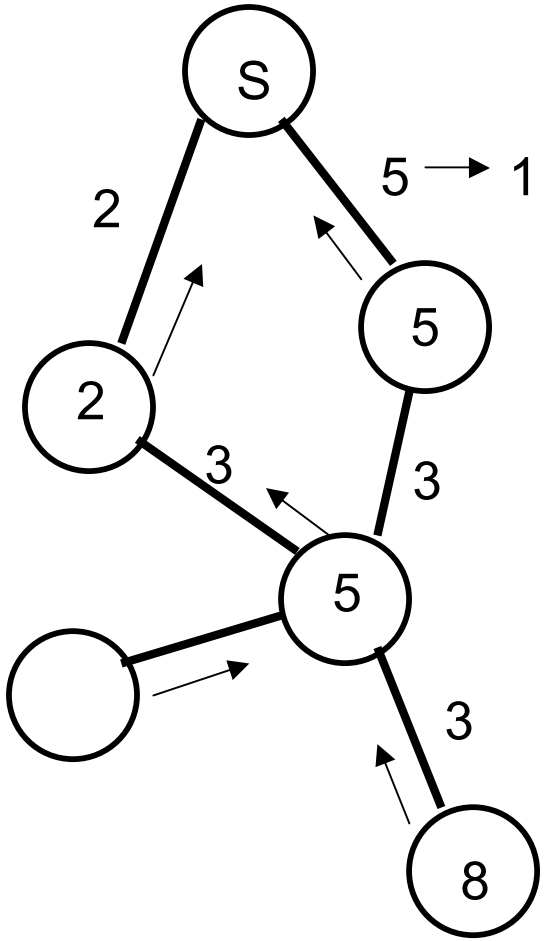
Comparación

- Los dos algoritmos dan el mismo resultado
- Quizas Dijkstra incluso lo calcula antes pero…
- Bellman-Ford
  - El calculo para el nodo n necesita el coste de los enlaces desde los vecinos de n mas el coste total del camino desde s a los vecinos de n
  - Cada nodo puede mantener el coste y el camino a cada otro nodo (en realidad vale con el siguiente salto)
  - Es suficiente con intercambiar información entre nodos vecinos
  - Se puede actualizar la el coste y el camino en cada nodo basado en la información de los vecinos y los costes de los enlaces a los vecinos
- Dijkstra
  - Cada nodo necesita toda la topología
  - Necesita conocer los costes de todos los enlaces en la red
  - Debe intercambiar información con el resto de nodos

Topología variable

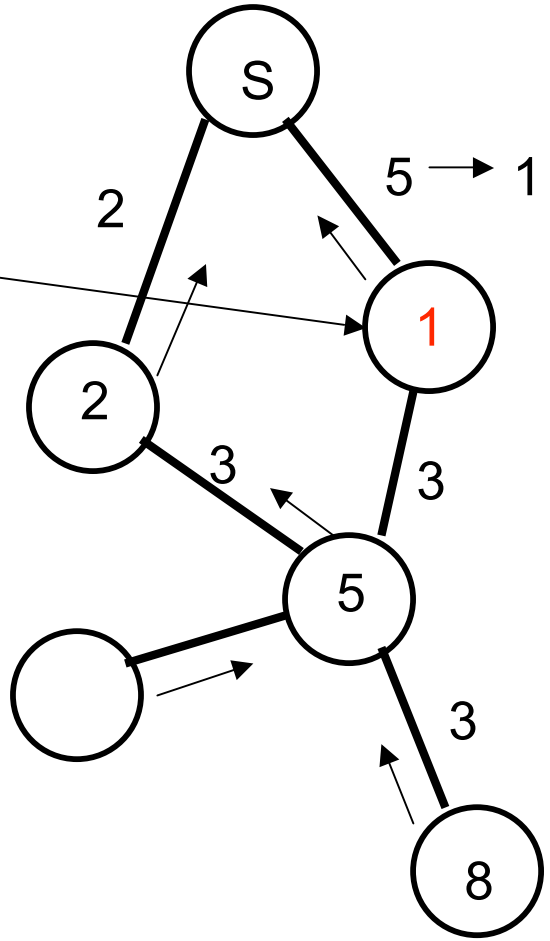
¿qué pasa si cambia un enlace?

- Dijkstra: reinicio el algoritmo y al añadir ese nodo a T se tendrá en cuenta el nuevo enlace
- Bellman-Ford: Aunque hemos detenido el algoritmo al dejar de producirse cambios. Las iteraciones de Bellman-Ford se pueden mantener periódicamente.
- Si no hay cambios reelegirán los mismos caminos, si hay cambios se adaptara a los cambios



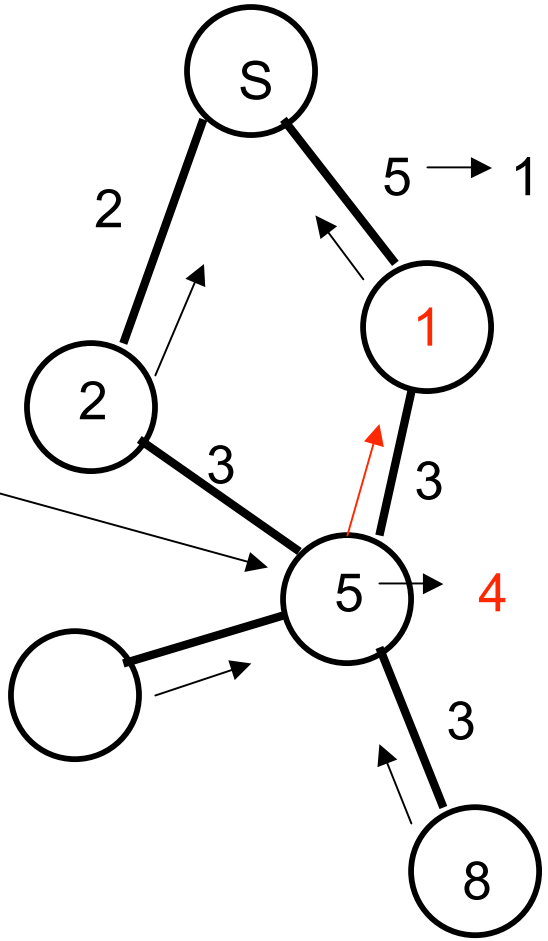
Topología variable

- Aplicando la iteración de Bellman-Ford se cambia el camino de este nodo
- La distancia más corta es ahora la de su vecino S + el nuevo peso
- El camino se mantiene



Topología variable

- Aplicando la iteración de Bellman-Ford una segunda vez se evalúa el camino para este nodo
- Ahora el camino por el nodo de la izquierda tiene distancia 5 pero el de la derecha sería 4
- El camino se cambia al nodo de la derecha
- Las siguiente iteración no cambia de nuevo



Ventajas de Bellman-Ford

- Las iteraciónes de Bellman-Ford se pueden ir aplicando continuamente.
- Se obtiene un algoritmo que se adapta a la topología cambiante
- Se puede mantener la información de los caminos a todos los nodos en cada nodo.
- Se pueden ir aplicando correcciones a esos caminos con información procedente sólo de los vecinos.
- Algoritmo de Bellman-Ford distribuido
- Algoritmo distribuido y que utiliza información de nodos vecinos

Encaminamiento en ARPANET

- 1969
- Enrutamiento distribuido adaptativo
  - Usando la ocupación de la cola como estimación del retardo
- Basado en el algoritmo de Bellman-Ford
- Cada nodo intercambia su vector de retardos con los vecinos (varias veces por segundo cada 128ms)
- Se recalcula la tabla de rutas basada en la información nueva (usando la ecuación de Bellman-Ford)
- Problemas:
  - No considera la velocidad del enlace solo el tamaño de la cola
  - El tamaño de cola no es buena estimación del retardo
  - Responde despacio a la congestión (Para cuando envio un paquete el tamaño de cola ha cambiado mucho)

Encaminamiento en ARPANET 2ª generación

- 1979
- Algoritmo distribuido y adaptativo basado en el retardo medido a los vecinos
  - Etiquetando los paquetes con el tiempo de salida, de llegada y confirmación de paquetes anteriores
- Se calcula el promedio de retardo de paquetes cada 10 segundos
- Los cambios se envían por inundación al resto de nodos
- Se calculan las rutas usando el algoritmo de Dijkstra, en cada nodo
- Bueno en condiciones de carga baja o media
- Con carga alta las tablas de rutas cambian al mismo tiempo en todos los nodos, con lo que en cuanto se calculan ya están obsoletas: Oscilaciones

Encaminamiento en ARPANET 3ª generación

- 1987
- Se cambia el cálculo del coste de los enlaces
- link cost calculations changed
  - Para reducir las oscilaciones
  - Reducir la carga extra de mensajes de erutamiento
- Se mide el retardo medio cada 10 segundos y se transforma siguiendo un modelo matematico de colas a una estimación de la utilización (que % de tiempo esta el enlace transmitiendo)
- Se normaliza y se utiliza una función para obtener la utilización media
- El coste cambia mas despacio y no se producen cambios bruscos de todas las rutas

Enrutamiento en Internet

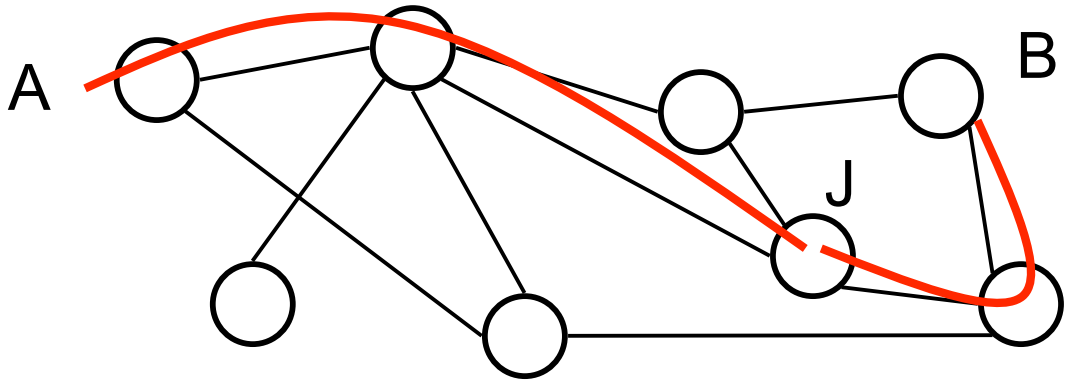
- Hasta ahora hemos visto algoritmos para calcular caminos entre nodos
- Pero en Internet el destino del enrutamiento no es calcular como llegar a un nodo sino calcular como llegar a una dirección IP
- Los routers conocen los rangos de direcciones IPs a las que están conectados y los almacenan con forma de subredes

  dirección de red + mascara

  dirección de red / numero de bits del prefijo
- Los routers almacenaran una lista de redes en forma de condiciones direccion/mascara y el siguiente router al que deben reenviar los paquetes que van a las direcciónes que cumplan esa condición

Teorema:

- Si el camino óptimo de A a B pasa por J, el camino óptimo de J a B es comun y el camino óptimo de A a B se construye concatenando el camino óptimo de A a J con el camino óptimo de J a B



- Esto implica que los caminos hacia un destino son un arbol (Spanning-tree)
- Y tambien que cada nodo vale con que se encargue del siguiente salto del camino hacia un destino. Es suficiente con almacenar un siguiente salto (entre los nodos vecinos) para cada destino

Enrutamiento en Internet

- Como parte del nivel 3 (de red) de la pila de protocolos de Internet (TCP/IP) existen protocolos que permiten comunicarse a los routers con otros routers vecinos para decidir las tablas de rutas
- Hay varios protocolos de enrutamiento de diferentes tipos que se pueden activar según convenga
  - Protocolos de enrutamiento de tipo **Distance-Vector**
  - Protocolos de enrutamiento de tipo **Link-State**
  - Protocolos de enrutamiento Intra-Sistema-Autonomo (**IntraAS**)
  - Protocolos de enrutamiento Inter-Sistema-Autónomo (**InterAS**)

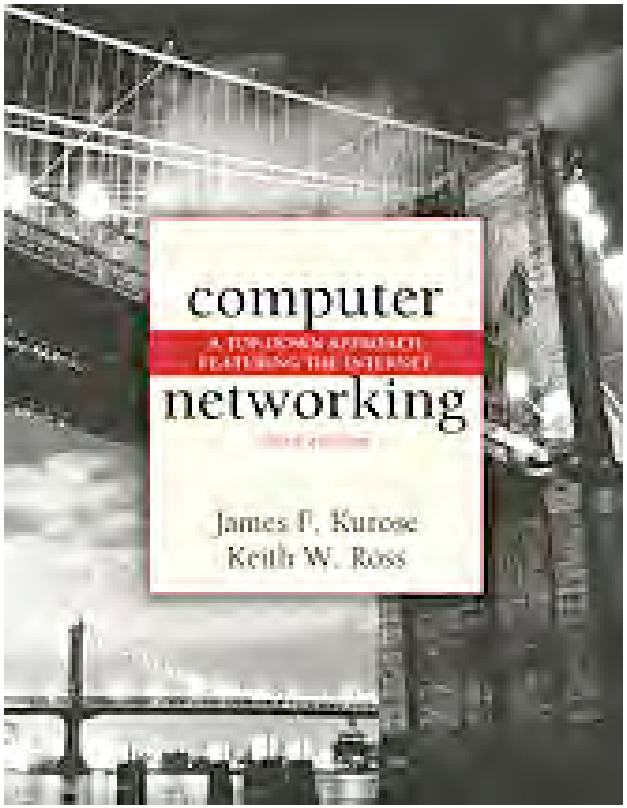
Bibliografía de lo que sigue

- Kurose & Ross,

  "Computer Networking a top-down approach featuring the Internet"

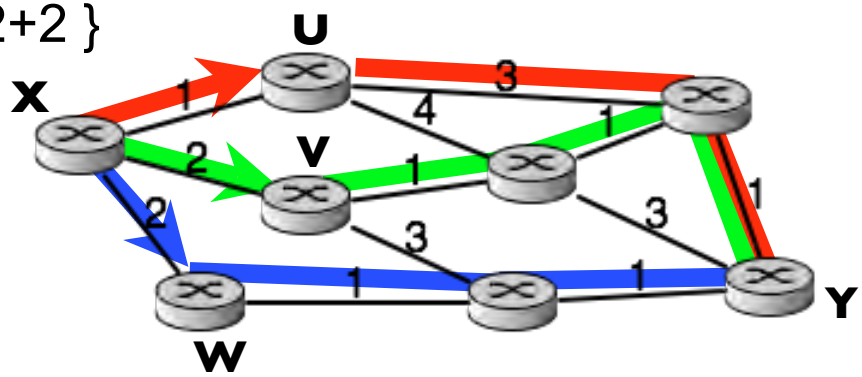
  Addison Wesley

  Capítulo 4



Enrutamiento distance-vector

- Basados en el algoritmo de Bellman-Ford distribuido (Ecuación de Bellman-Ford
  $d_x(y)$: coste mínimo del camino de x a y
  Entonces:
  $d_x(y) = \min_v \{ c(x,v) + d_v(y) \}$
- Ejemplo:
  - $d_u(y)=4 \quad d_v(y)=3 \quad d_w(y)=2$
  - $d_x(y)=\min \{ \; c(x,u) + d_u(y),$
    $c(x,v) + d_v(y),$
    $c(x,w) + d_w(y) \}$
  - $d_x(y)=\min \{ 1+4, 2+3, 2+2 \}$

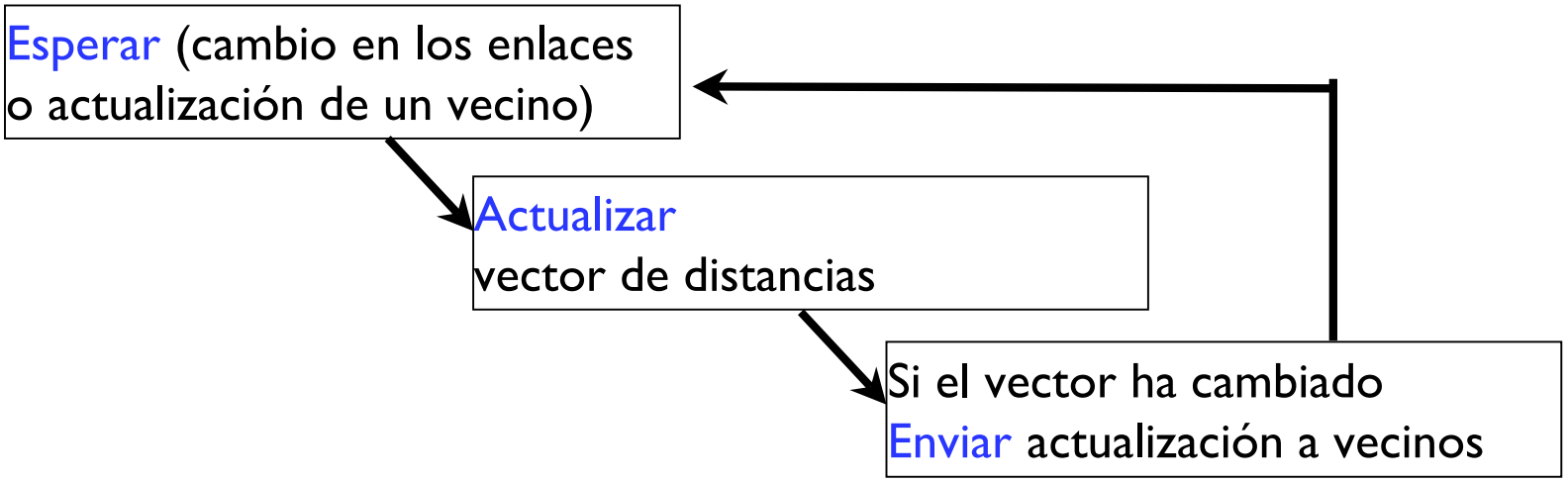


Algoritmo Distance-Vector

- Algoritmo distribuido
  - Cada nodo mantiene un **vector** de estimaciones de distancias
    - D_x=[$D_x(y)$ para todos los y]
  - Cada nodo tiene los costes de enlaces a los vecinos c(x,v)
    Para minimizar los saltos simplemente se supone c(x,v)=1
- Funcionamiento
  - Periódicamente cada nodo envia su vector D_x a sus vecinos
  - Cuando recibe una actualización de estimacion de un vecino v
    - Actualiza su propia estimación (eq de Bellman-Ford)
      $D_x(y) = \min_v\{ c(x,v)+D_v(y) \}$
- Converge hacia $d_x(y)$
  - con condiciones mínimas (no actualizar todos a la vez)

Algoritmo Distance-Vector

- Iterativo asíncrono
  - El envío de información a los vecinos se dispara por
    - Cambio en el coste de un enlace local
    - Cambio en el coste al actualizar con información de otro nodo
  - Los cambios se envían rápido
  - El algoritmo es self-stopping: cuando no hay cambios no manda
  - Las implementaciones reales mandan periodicamente por si acaso
- Ejemplo típico: RIP (Routing Information Protocol)



$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0 , 3+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$

node x table

from x	to x	to y	to z
	0	2	7
	∞	∞	∞
	∞	∞	∞

from x	to x	to y	to z
	0	2	3
	2	0	1
	7	1	0

from x	to x	to y	to z
	0	2	3
	2	0	1
	3	1	0

node y table

from y	to x	to y	to z
	∞	∞	∞
	2	0	1
	∞	∞	∞

from y	to x	to y	to z
	0	2	7
	2	0	1
	7	1	0

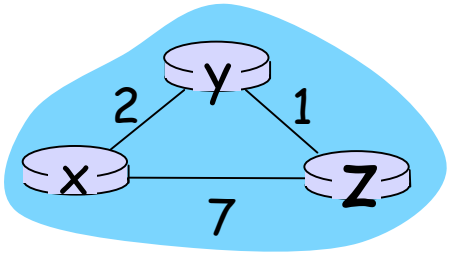
from y	to x	to y	to z
	0	2	3
	2	0	1
	3	1	0

node z table

from z	to x	to y	to z
	∞	∞	∞
	∞	∞	∞
	7	1	0

from z	to x	to y	to z
	0	2	7
	2	0	1
	3	1	0

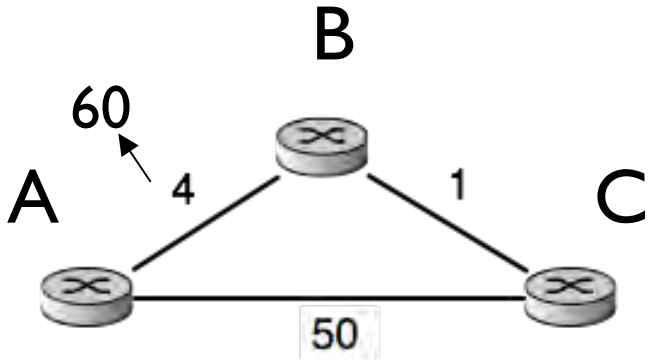
from z	to x	to y	to z
	0	2	3
	2	0	1
	3	1	0



time

Algoritmo Distance-Vector

Problema de los Cambios a peor en Distance-Vector



	B
A	60
C	1

	C
A	5 viaB
B	1

$D_B(A)=60$

$D_C(A)+1=6$!!

	B
A	6 viaC
C	1

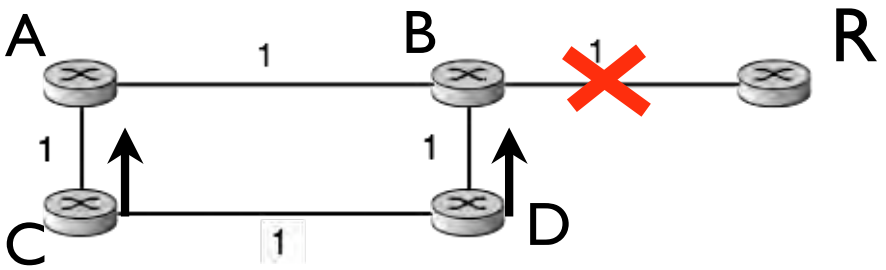
	C
A	7 viaB
B	1

	B
A	8 viaC
C	1

Bucle de enrutamiento
Hasta cuando???
count to infinity

Problemas distance-vector

- Count to infinity, inestabilidad y bucles
- Soluciones:
  - Hold-down. Si un enlace se cae no lo actualizes en un tiempo para que de tiempo a todos los routers a enterarse
  - Enviar no sólo la distancia sino también el camino. Más estado
  - Split-horizon: No anunciar distancias a B de los caminos que envío a través de B (resuelve el ejemplo pero no todas las cuentas a infinito)
  - Poison reverse. Todos los caminos que sean a través de un vecino los anuncio a ese vecino como distancia infinita
  - Triggered updates: enviar los cambios en cuanto se produzcan
- Ninguna elimina completamente las cuentas a infinito
  - Si cae el enlace B-R observe las rutas de C y D a R



Link-state

- Algoritmo de enrutamiento centralizado
  - Los nodos envian información del estado de sus enlaces (link-state)
    - A un nodo central
    - A todos los nodos
  - Con la información sobre todos los enlaces (el grafo completo) se buscan las mejores rutas en el nodo central o en cada nodo
    - La búsqueda de rutas no es distribuida (pero hay que distribuir la información)
- Para encontrar los caminos de mínimo coste se emplea el **algoritmo de Dijkstra**
- Protocolos de enrutamiento Link-State

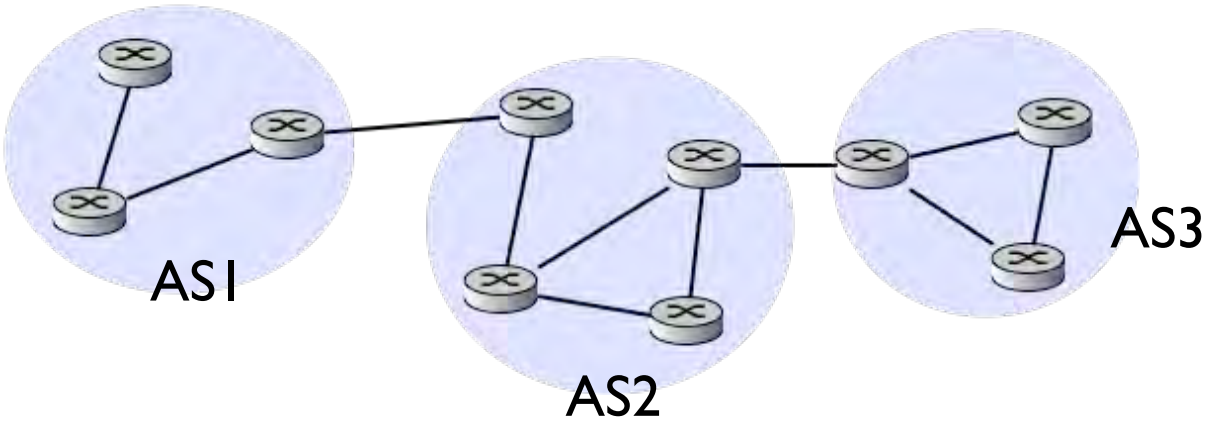
  Más conocido: OSPF (Open Shortest Path First)

Enrutamiento en Internet

- El uso de algoritmos distance-vector o link-state para toda la red es difícil
  - **Escala**
    Las tablas de rutas, el numero de nodos y el numero de subredes a manejar... podría resolverse
  - **Administración**
    Los administradores de diferentes empresas quieren controlar el enrutamiento teniendo en cuenta quien es el propietario de cada red/router
    Pueden querer ocultar la estructura interna de sus redes
- RIP, OSPF están pensados para una única entidad

Enrutamiento en Internet

- A alto nivel en Internet se definen **Sistemas Autónomos** (**AS**)
- Conjunto de routers administrados por una misma entidad (ISP o empresa...)
  - Dentro de cada AS se usan los algoritmos de enrutamiento vistos hasta ahora (RIP, OSPF...) que se llamarán **Intra-AS**
  - Como se enruta entre diferentes sistemas autónomos?

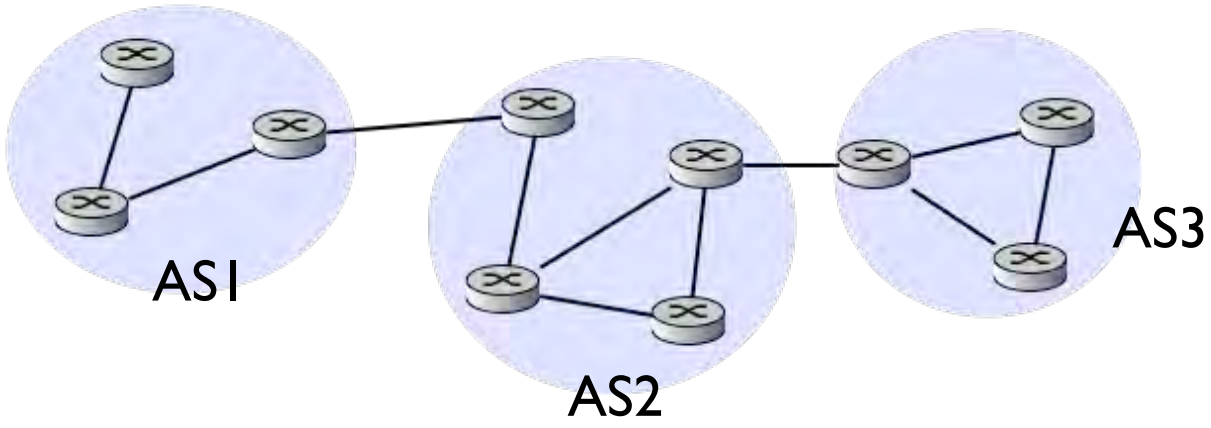


Enrutamiento en Internet

- Internet se puede ver como un conjunto interconectado de Autonomous Systems (AS)
  AS: Red bajo un mismo control administrativo y técnico
- Enrutamiento a dos niveles
  - **Intra-AS** (protocolos llamados **Interior Gateway Protocols**)
    Enrutamiento dentro de un AS
    Ejemplos típicos: RIP, OSPF
  - **Inter-AS** (protocolos llamados **Exterior Gateway Protocols**)
    Encontrar rutas entre ASs diferentes
    Ejemplos típicos: EGP, BGP

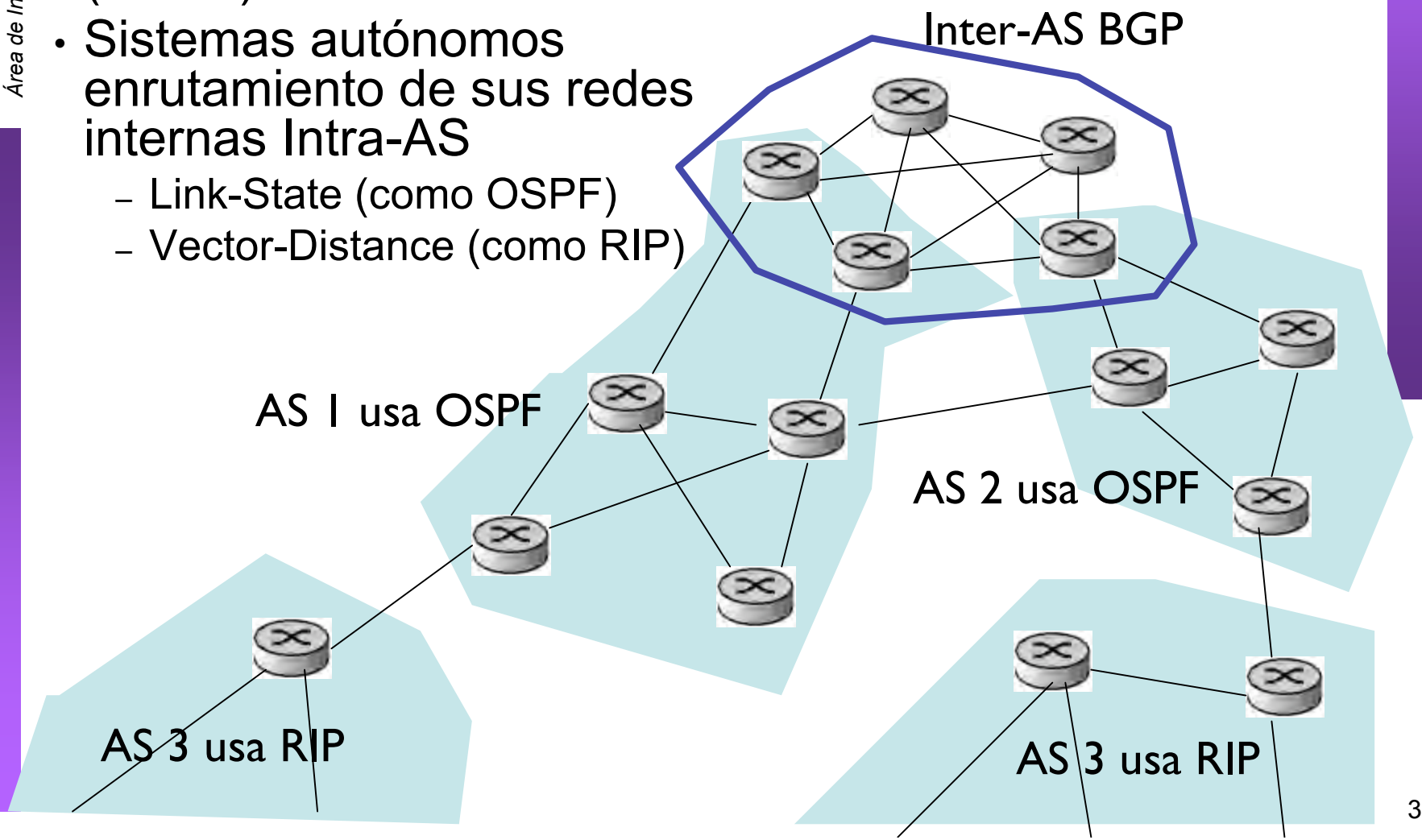
Enrutamiento en Internet

- Protocolos de enrutamiento **inter-AS**
  - Enviar al resto de sistemas autónomos información sobre las redes internas
  - Utilizar la información del resto de sistemas autónomos para decidir que router de salida de nuestro AS debemos usar según el destino del paquete
- Los sistemas autónomos deben usar el mismo protocolo de enrutamiento inter-AS



Enrutamiento en Internet

- Núcleo muy interconectado enrutamiento Inter-AS (BGP4)
- Sistemas autónomos enrutamiento de sus redes internas Intra-AS
  - Link-State (como OSPF)
  - Vector-Distance (como RIP)



Conclusiones

- Algoritmo de Bellman-Ford
- Algoritmo de Bellman-Ford distribuido
- Enrutamiento en ARPANET
  - 3 generaciones
- Enrutamiento en Internet
  - Distance-vector
  - Link-state
  - Enrutamiento jerarquico
  - IntraAS e InterAS