

Enrutamiento

Link-state...

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

Arquitectura de Redes, Sistemas y Servicios
Grado en Ingeniería en Tecnologías de Telecomunicación, 2º

Temario

1. Introducción
2. Arquitecturas de conmutación y protocolos
3. Introducción a las tecnologías de red
4. Control de acceso al medio
5. Conmutación de circuitos
6. Transporte fiable
7. Encaminamiento

Temario

1. Introducción
2. Arquitecturas de conmutación y protocolos
3. Introducción a las tecnologías de red
4. Control de acceso al medio
5. Conmutación de circuitos
6. Transporte fiable
7. **Encaminamiento**

Resumen hasta ahora

- Algoritmo distance-vector teórico
- 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?

Enrutamiento Link-State

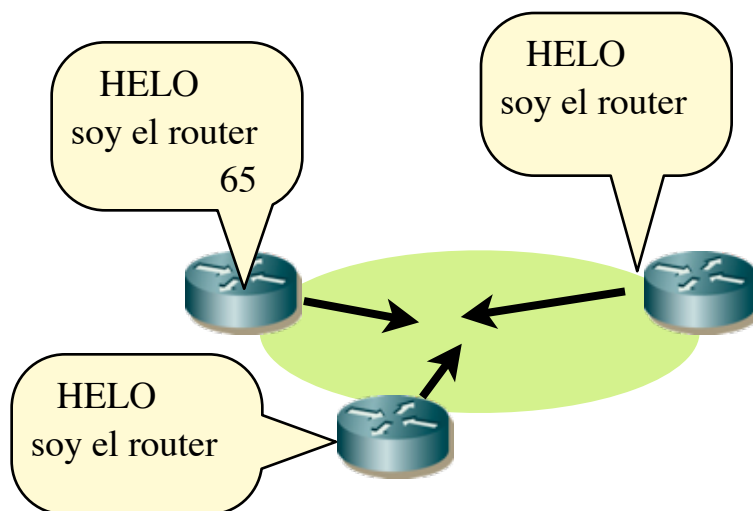
- Idea de funcionamiento
 - Cada router es responsable de reconocer a sus vecinos y aprenderse su nombre (identidad)
 - Cada router construye un paquete llamado LSP (link state packet) conteniendo una lista de todos sus vecinos y el coste asociado
 - El LSP se envía a todos los demás routers de la red. Cada router recuerda el ultimo LSP recibido de cada otro router
 - Cada router con el mapa completo de la topología calcula las rutas a cada destino posible y actualiza su tabla de rutas
- Como de difícil es cada una de estas fases?

Link-State: 1 descubrir vecinos

- Descubrimiento de vecinos

Mensaje HELO

- Contiene una identidad del router
- Periódico a la red de área local o al enlace punto a punto
- Recordar los últimos HELOs vistos (lista de identidades de los vecinos)
- Tiempo entre mensajes
- Tiempo para dar al router por desaparecido



Vecinos	Visto hace
65	6s
42	14s

Link-State: construir LSPs

- Construir LSPs
 - Periódicamente
 - Cada vez que veo un nuevo vecino
 - Cada vez que cambia el coste a un vecino
 - Cada vez que dejo de ver a un vecino (ha pasado el tiempo definido sin escuchar HELOs de ese vecino)
- El LSP contiene

El router de identidad X tiene esta lista de vecinos

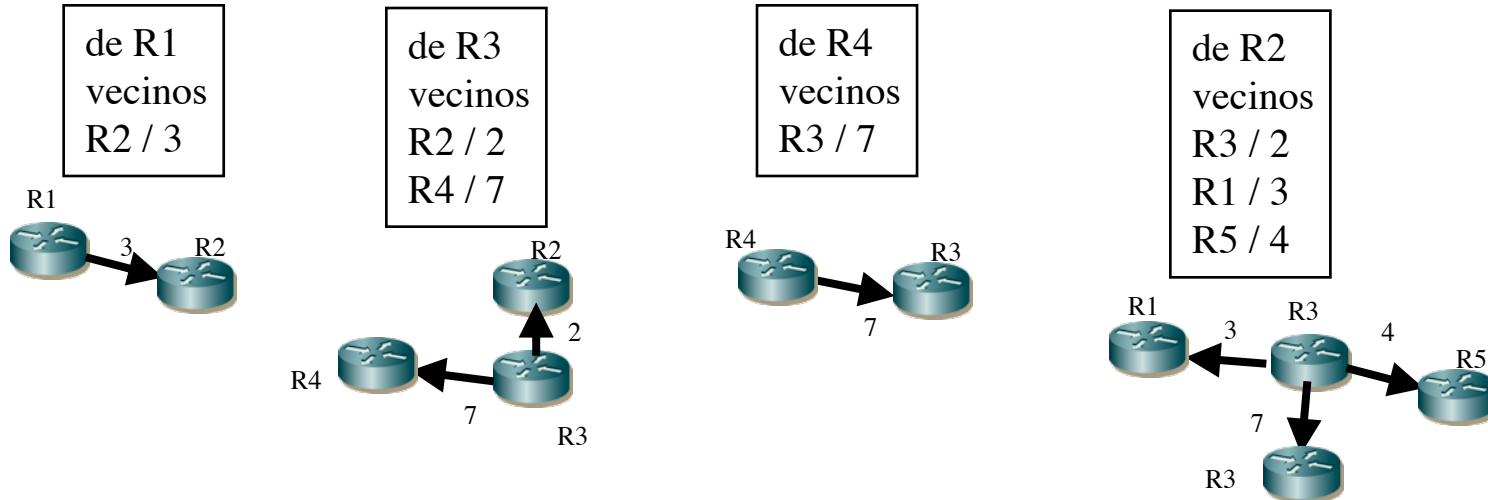
Vecino 1: Y con coste c_1

Vecino 2: Z con coste c_2

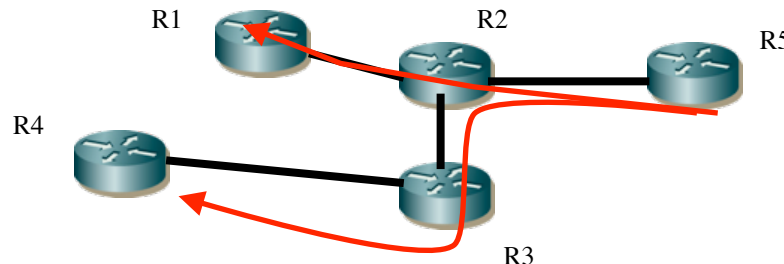
Vecino 3 ..
- En que se diferencia de los distance-vector??

Link-State: 3 distribuir LSPs

- Cada router necesita los LSPs de todos los demás, no solo el de sus vecinos
- Recibo estos



- Base de datos de nodos y enlaces en la red
- Usando el algoritmo de Dijkstra calculo caminos



Link-State: 3 distribuir LSPs

- Si hay un cambio en la red los routers que lo ven envían el cambio a todos
- El resto de los enlaces están en la base de datos
- Con el cambio de ese enlace se recalculan los caminos en todos
- No hay cuentas a infinito
- Reaccion muy rapida

Link-State: 3 distribuir LSPs

- El destino de los LSPs
Son todos los routers de la red
Es facil enviar a los vecinos pero no a cualquier router de la red... (no se las direcciones de todos, no puedo basarme en la tabla de rutas para llegar a ellos)
- Es necesario un sistema de envío a cualquier destino que no requiera que funcionen las tablas de rutas.
Inundación?
- La distribución de LSPs es el punto critico de los algoritmos de tipo link-state
 - Si no llega la información a todos, las rutas son incoherentes y puede haber ciclos de enrutamiento (recordar cuentas a infinito con un solo enlace en desacuerdo)
 - Si no tenemos cuidado la inundación puede causar mucho trafico extra, los routers pueden saturarse y dedicar gran parte de su cpu simplemente a recalcular rutas tras recibir

Link-State: 3 distribuir LSPs

- Inundando LSPs
- Cada router escucha LSPs de sus vecinos y reenvía
- Como limitar el tráfico
 - TTLs?
 - Con LSPs ya estamos guardando estado (cada router recuerda el ultimo LSP de cada destino) usar esto para limitar la inundacion (ignorar un LSP si ya lo he visto)

Link-State: 4 calcular rutas

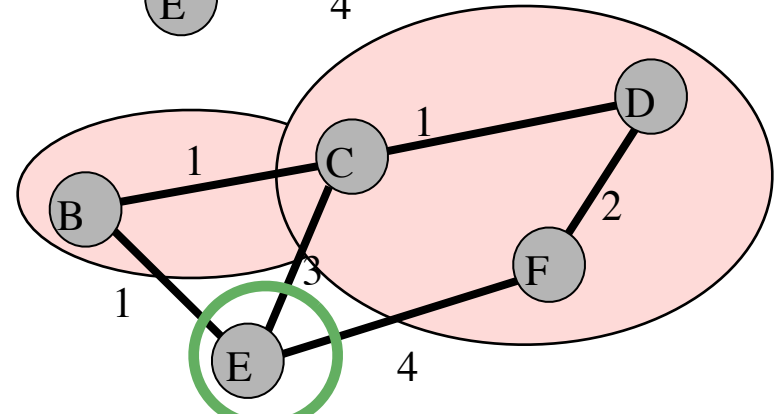
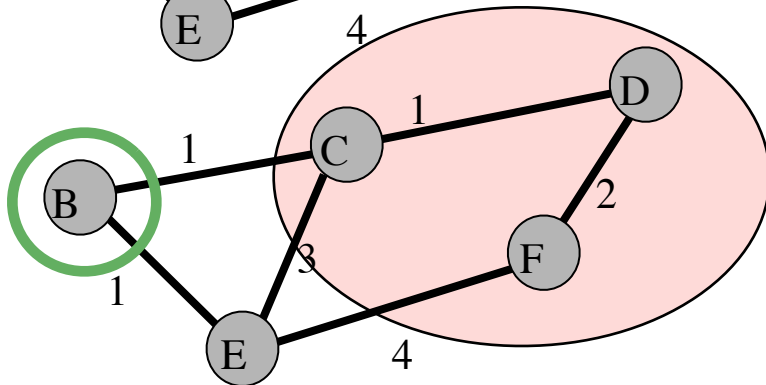
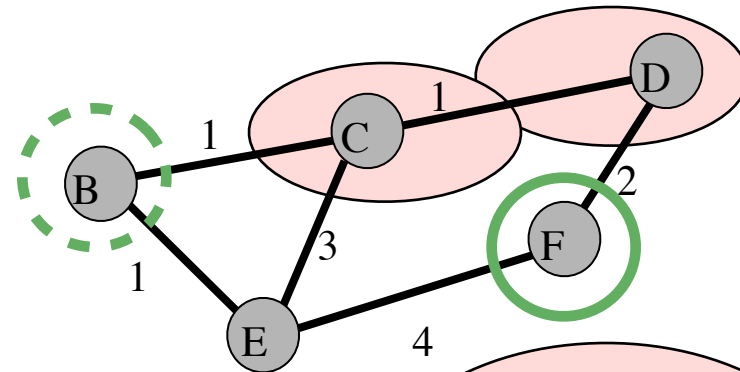
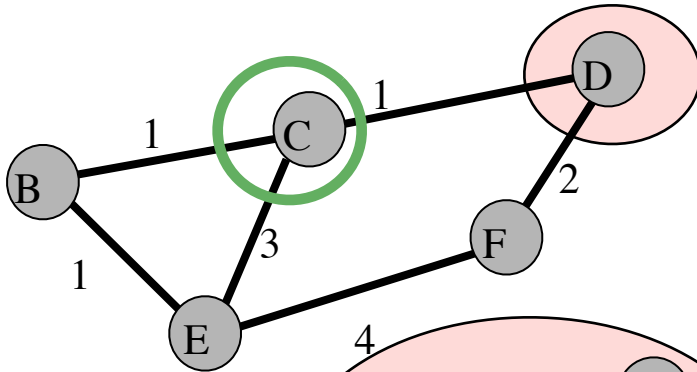
- Los LSPs se guardan en una link-state database
- Se utiliza el algoritmo de Dijkstra para calcular caminos
- Se colocan las rutas de este nodo teniendo en cuenta esos caminos
- Si la red es muy grande todos los routers necesitan mas recursos que en el caso de distance-vector
 - Deben almacenar todos los enlaces de la red
(mucha mas memoria que el vector de distancias)
 - Deben calcular el algoritmo de Dijkstra cada vez que ven un cambio (mucho mas complicado que la iteracion del Bellman-Ford)
- La convergencia es rapida porque los LSPs llegan a todos y no hay ambigüedades de caminos ni cuentas a infinito.

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
 $d(i)=\infty$ $d(r)=0$ si el nodo i es vecino de r [$w(i,r)<\infty$] $d(i)=w(i,r)$
 $T=\{r\}$
 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

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

$T=\{r\}$

$s(i)=\text{desconocido}$ si i es vecino de r [$w(i,r)<\infty$] $s(i)=r$

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 .

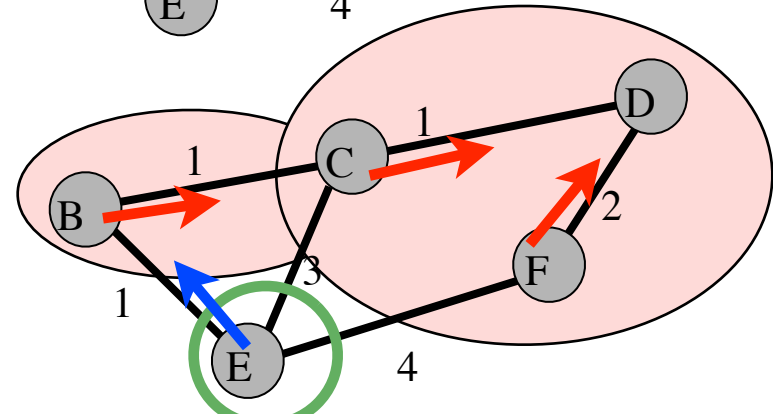
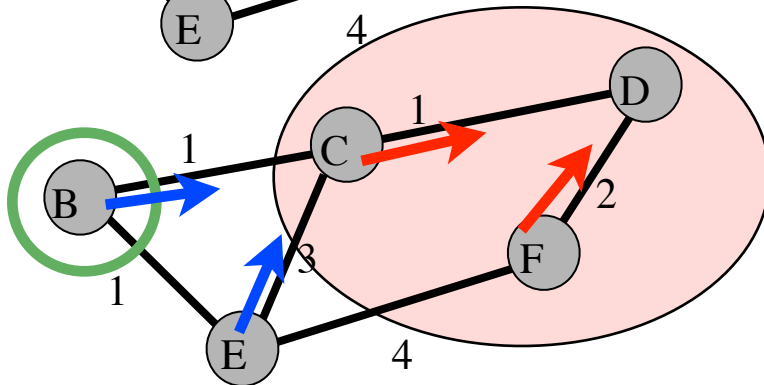
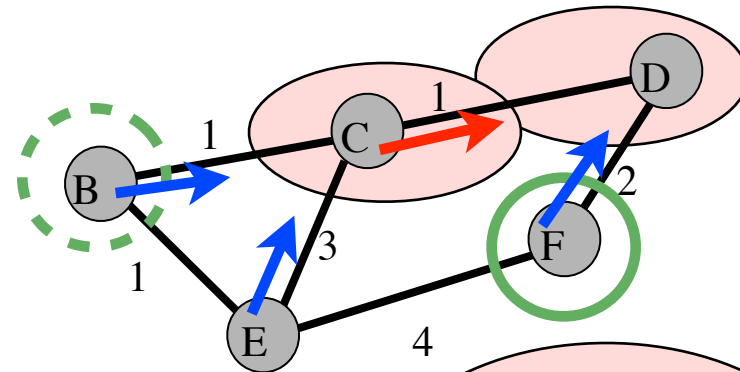
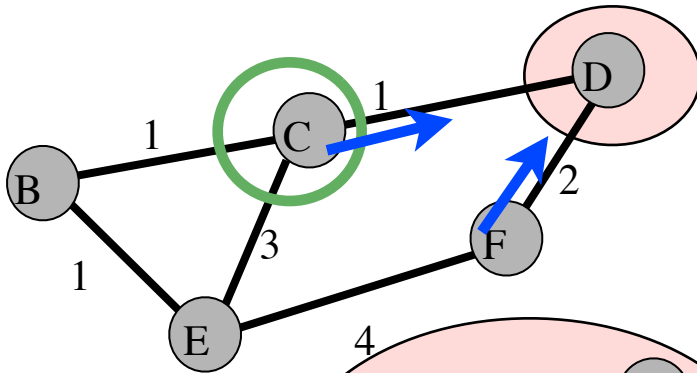
 Si $d(i)+w(k,i) < d(k)$ entonces $d(k)=d(i)+w(k,i)$

 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



Algoritmos de caminos mínimos

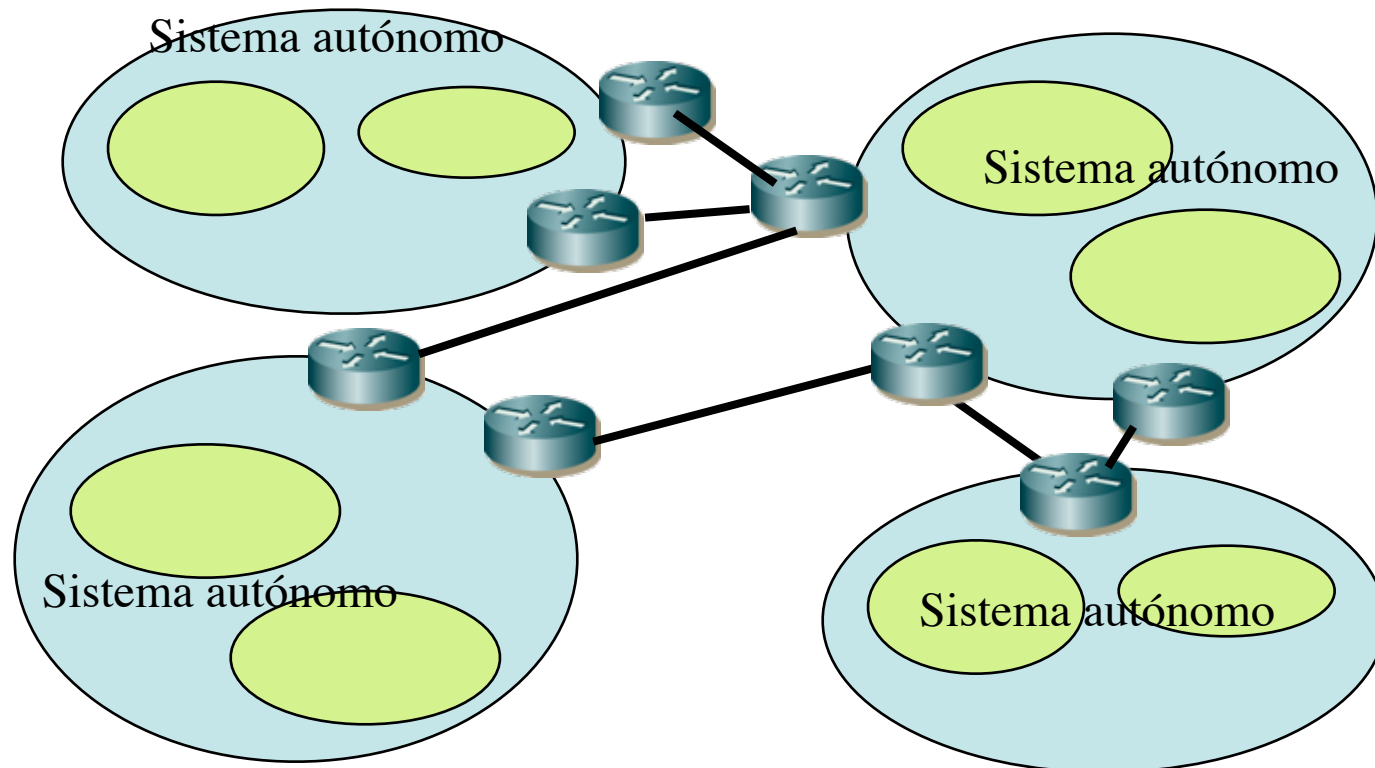
- Dijkstra vs Bellman-Ford
 - Los dos calculan el árbol de expansión mínimo para una raíz dada
 - Los dos algoritmos dan el mismo resultado
 - El resultado no tiene por que ser único (probad a hacer los ejemplos anteriores cambiando el peso de C-E a 2)
- Cuál es más rápido?
 - Parece que Bellman-Ford hace menos iteraciones pero las iteraciones 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
- Bellman-Ford permite construir algoritmos distribuidos basados en el concepto distance-vector

Distance-vector vs Link-state

- En uso de memoria: menor distance-vector
- En tráfico de red: controvertido distance-vector no hace inundacion pero la inestabilidad dura mas
- En carga de CPU: menor distance-vector
- En robustez: no esta claro. link-state es mas robusto frente a fallos no maliciosos
- En funcionalidad: link-state tiene muchas ventajas, ya que todos los routers conocen la topologia entera, es mas facil de depurar las situaciones de fallo, permite source-routing, caminos paralelos...
- Velocidad de convergencia: calramente mejor link-state

Intradomain vs Interdomain

- Internet es un conjunto de sistemas autonomos con enrutamiento independiente
- Cada uno puede utilizar los protocolos de enrutamiento que elija internamente **intra-dominio**
- Hay protocolos para compartir las rutas entre dominios (**inter-dominio**)
 - Un protocolo común para hablar con cualquier otro sistema autonomo



Conclusiones

- Protocolos de enrutamiento link-state
 - Mas complejos
 - pero en general mejores que distance-vector
- El problema de la red completa
 - Enrutamiento jerarquico
 - Enrutamiento intradominio e interdominio