Link State

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

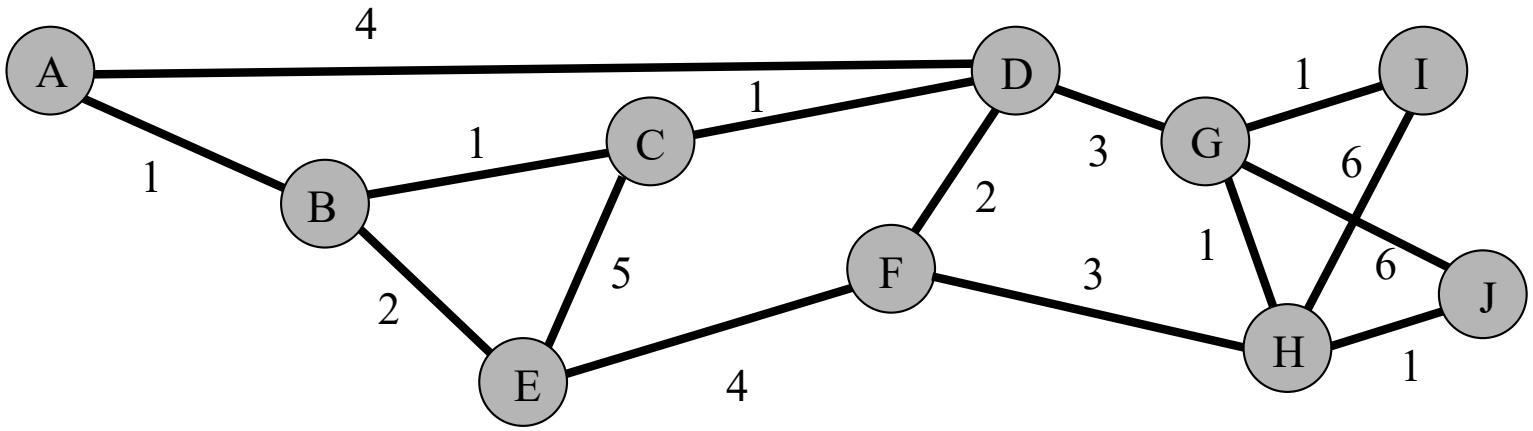
Arquitectura de Redes, Sistemas y Servicios

Enrutamiento link state

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

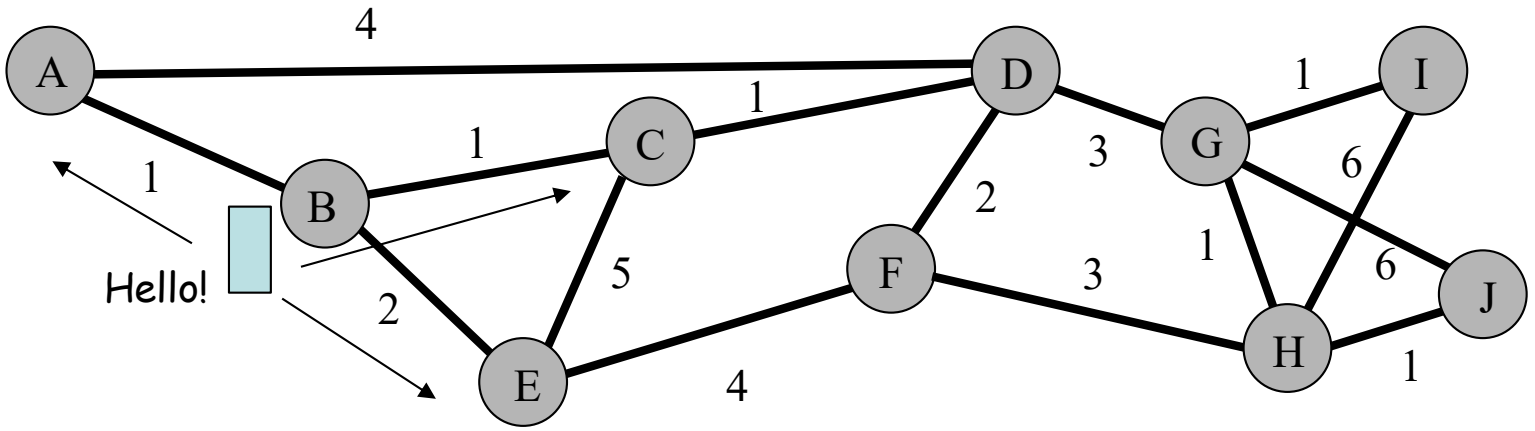
Idea general

- Cada nodo conoce a sus vecinos y el coste de enlaces con ellos
- Cada nodo hace llegar esa información a todos los demás
- Con eso cada nodo conoce el grafo completo de la topología
- Puede emplear un algoritmo para calcular los sucesores para ir a cada destino
- Este será normalmente el algoritmo de Dijkstra



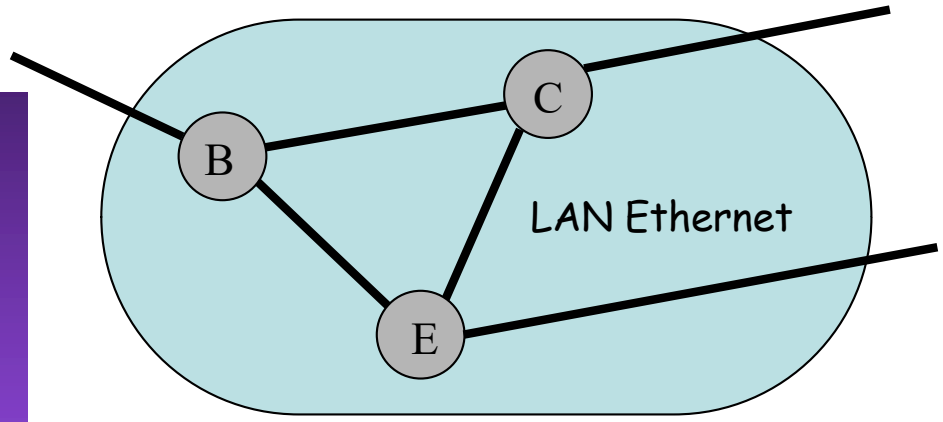
Descubrimiento de vecinos

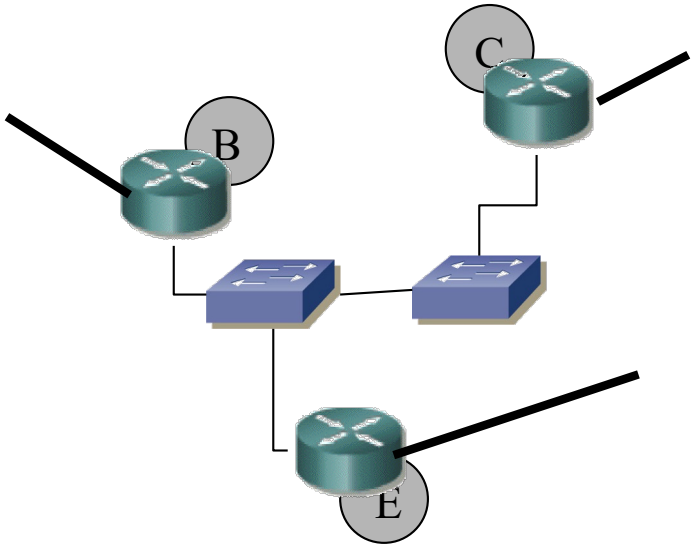
- Mediante algún protocolo por el que cada uno anuncie su existencia a sus vecinos
- En enlaces punto-a-punto es sencillo
- En redes multiacceso (Ethernet) suele hacerse el envío a una dirección de multicast o broadcast (no se inunda)
- Envío periódico permite recordar a vecino que sigue ahí y descubrir aprox. cuándo se pierde conectividad con él

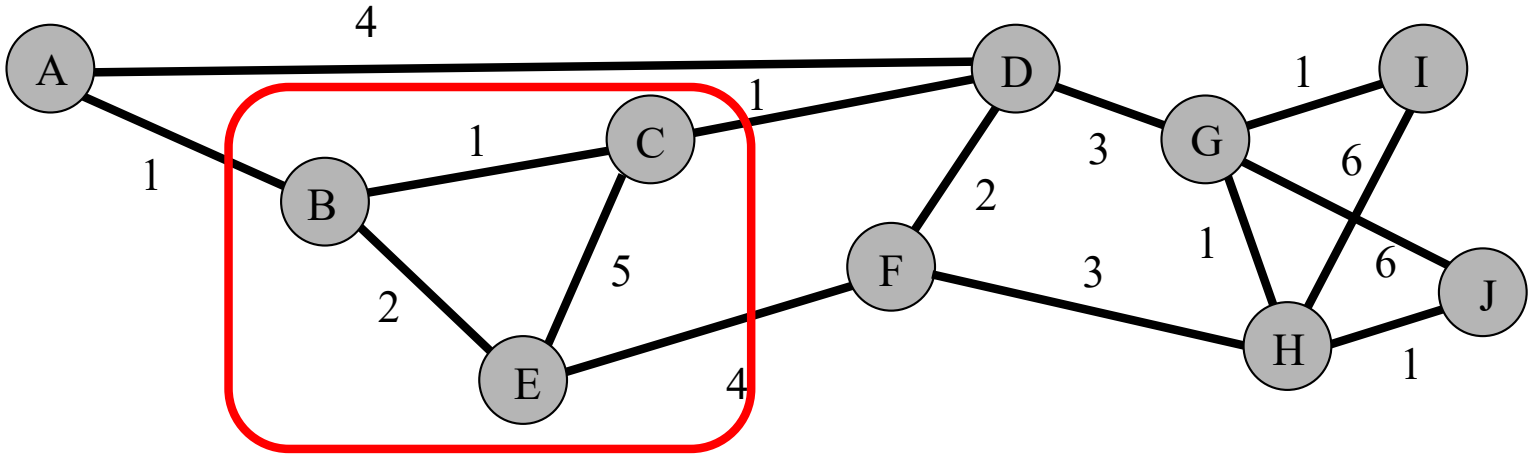


Red multiacceso

- Por ejemplo, lo que en el grafo representábamos como una unión de todos-con-todos
- Podría ser una LAN Ethernet

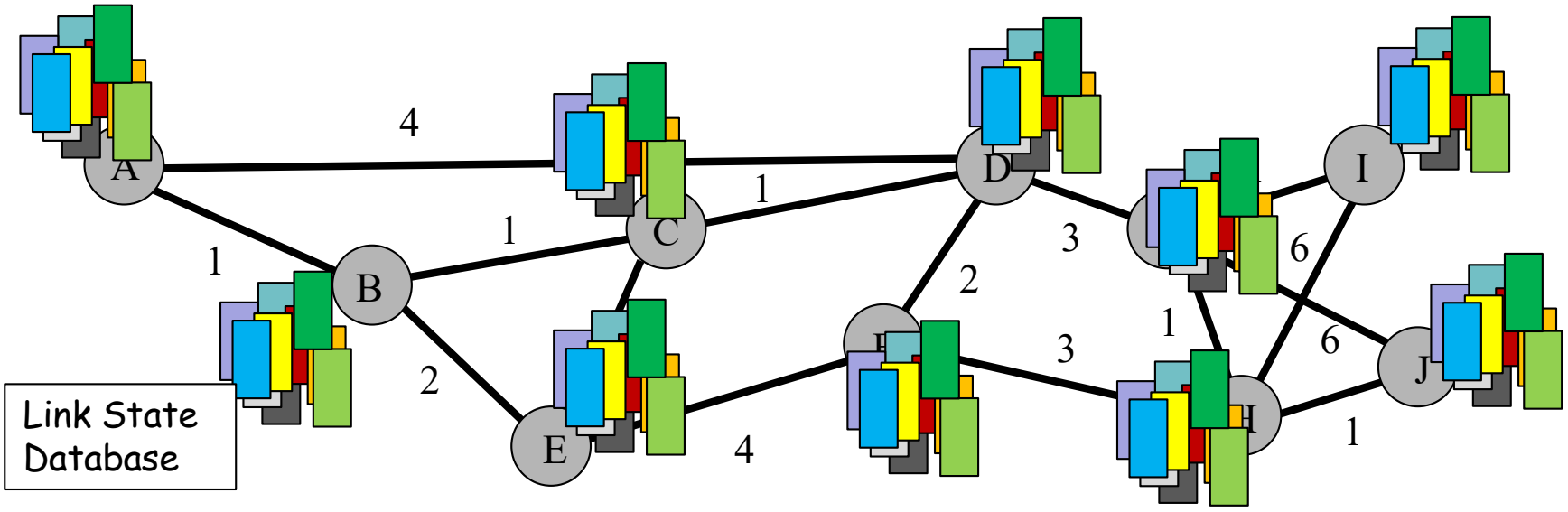






Difusión de información

- Cada nodo es capaz de construir un mensaje describiendo a los vecinos descubiertos (y enlaces con ellos)
- El nombre concreto depende del protocolo, pero se suele conocer como el “Link State Packet” (LSP)
- Ese LSP debe hacerse llegar a todos los demás nodos
- Si todos los nodos hacen esto tendremos una base de datos de LSPs replicada en todos ellos
- Envío periódico y/o ante cambios: aparecer o desaparecer vecinos, cambiar costes
- Con la información de todos los LSPs cada nodo conoce el grafo



Inundación

- Cada nodo conoce solo a sus vecinos
- No sabe a quiénes debe hacer llegar la información
- Ni los nodos conocen caminos para llegar a ellos
- Se suele emplear un mecanismo de inundación
- Problemas a resolver:
  - Bucles (¿TTL?)
  - Recibir múltiples copias (¿guardar y comparar?)
  - Se pueden perder (¿confirmarlos?)
  - Se pueden desordenar (¿numerarlos?)
  - Desconexión y reconexión de dos partes de la red
  - Etc.
- Lo vemos más adelante con protocolos concretos (OSPF, IS-IS, PNNI)

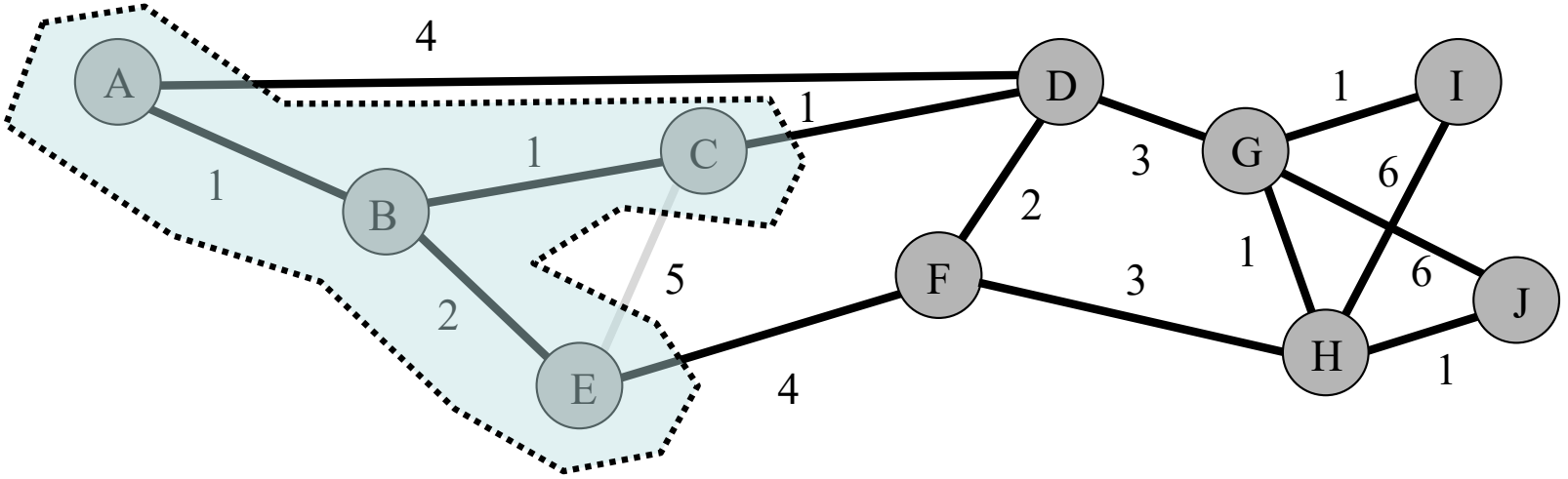
Cálculo de caminos

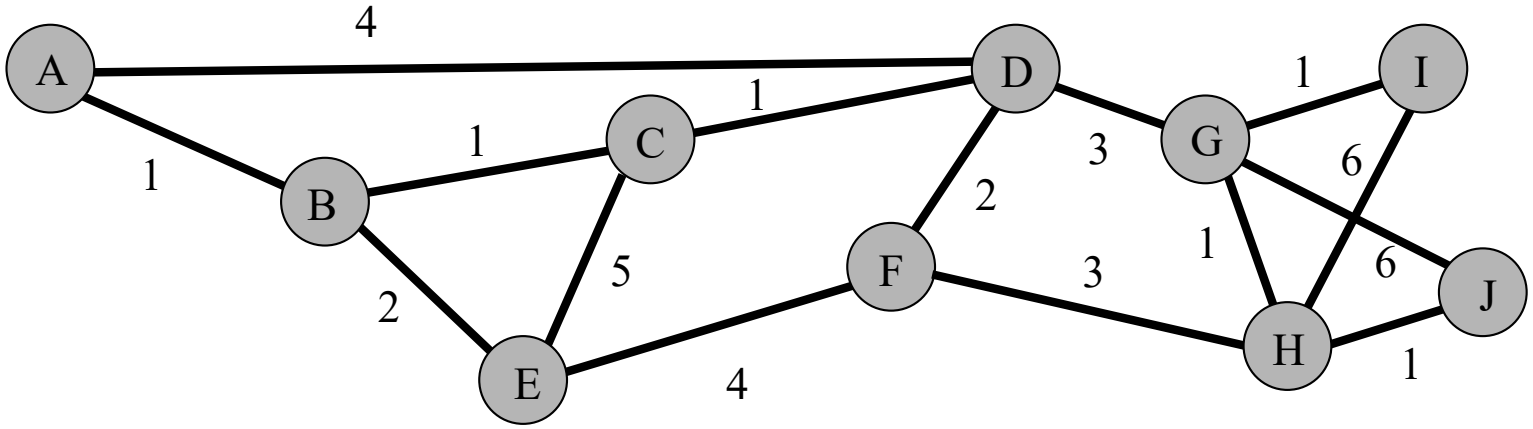
- Calcular árboles a partir del grafo
- Algoritmo de Dijkstra

Algoritmo de Dijkstra

Dijkstra: Idea general

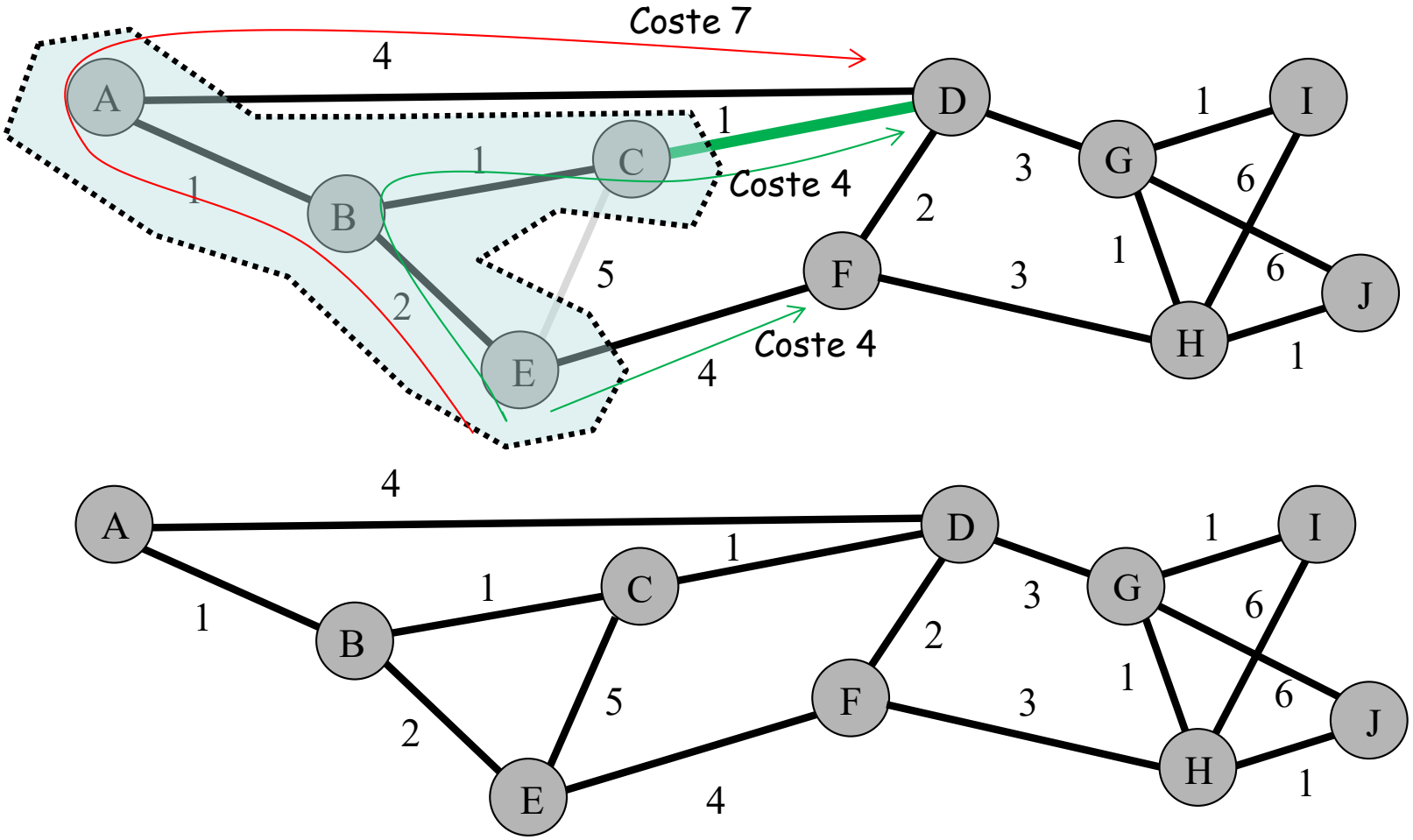
- Vamos construyendo el árbol de caminos de mínimo coste
- En cierto momento hemos construido una parte del árbol
- Añadimos el nodo de fuera del árbol adyacente a uno del árbol para el que tengamos el menor coste del camino





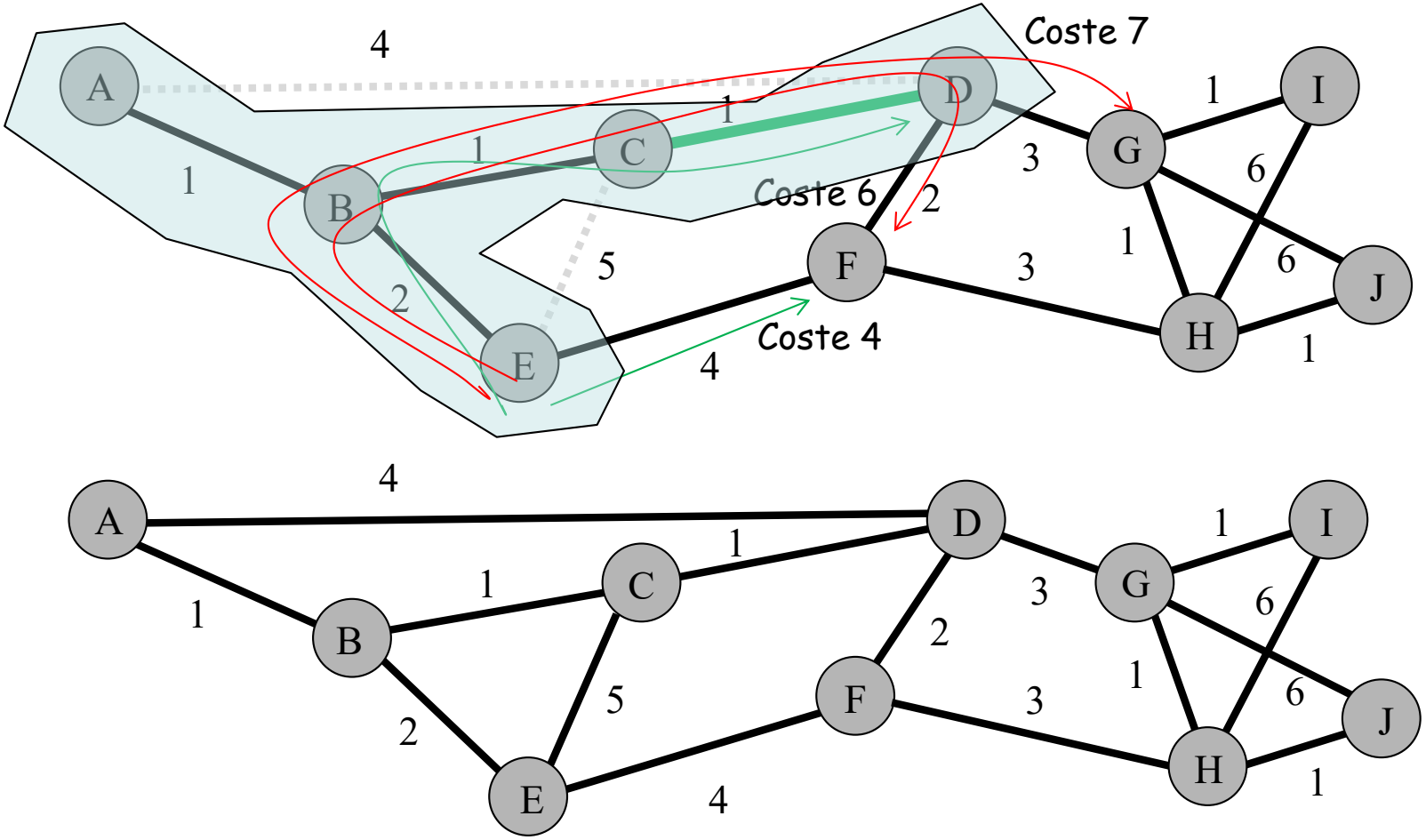
Dijkstra: Idea general

- En este caso hay 2 nodos que hacen frontera con el árbol parcial que tenemos: D y F, uno de ellos con dos caminos
- Añadimos al árbol F con el enlace a E o añadimos D con el enlace a C (mismo coste total)



Dijkstra: Idea general

- Ahora los nodos adyacentes son F y G
- A F coste 4 o coste 6, a G coste 7, añadiremos F con el enlace a E



Dijkstra

- s = nodo origen (source)
- N = conjunto total de nodos
- E = conjunto total de vértices (enlaces)
- V = vértices añadidos al árbol
- T = conjunto de nodos incorporados al árbol hasta el momento
- G = {T, V} = árbol formado por el conjunto de nodos y el conjunto de vértices (enlaces) en el árbol en un momento
- $w(i,j)$ = coste del enlace directo entre nodos i y j. Si no hay enlace directo el coste es ∞. De un nodo a sí mismo coste 0.
- $L(n)$ = coste del camino de menor coste desde el nodo s al nodo n conocido hasta el momento
- Inicialización:

$$\begin{cases} T = \{s\} \\ L(n) = w(s,n) \quad \forall n \in N, n \neq s \end{cases}$$

- Actualización: Encontrar el nodo vecino a algún nodo de T, que no esté en T con el menor coste desde nodo de T a él e incorpórese a T, junto con el enlace al nodo que ya estaba en T al conjunto V

$$Encontrar\ x \epsilon N, tal\ que\ x \notin T\ y\ L(x) = \min_{j \notin T} L(j)$$

- Actualizar costes mínimos para nuevos nodos n adyacentes al árbol:

$$L(n) = min[L(n), L(x) + w(x,n)], x \epsilon T$$

- Condición de parada: Cuando todos los nodos estén en T

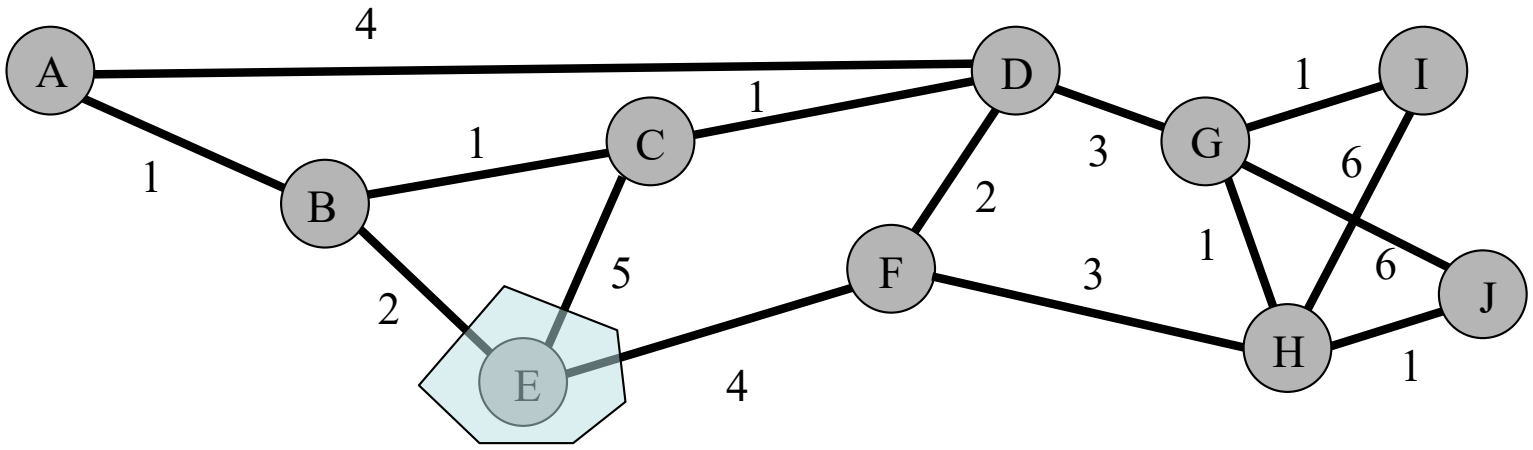
Ejemplo

T={E}

- Vecinos a nodos de T son B, C y F
- Mínimo coste es a B

L(n)									
A	B	C	D	E	F	G	H	I	J
∞	2	5	∞	0	4	∞	∞	∞	∞

$$Encontrar\ x \epsilon N, tal\ que\ x \notin T\ y\ L(x) = \min_{j \notin T} L(j)$$

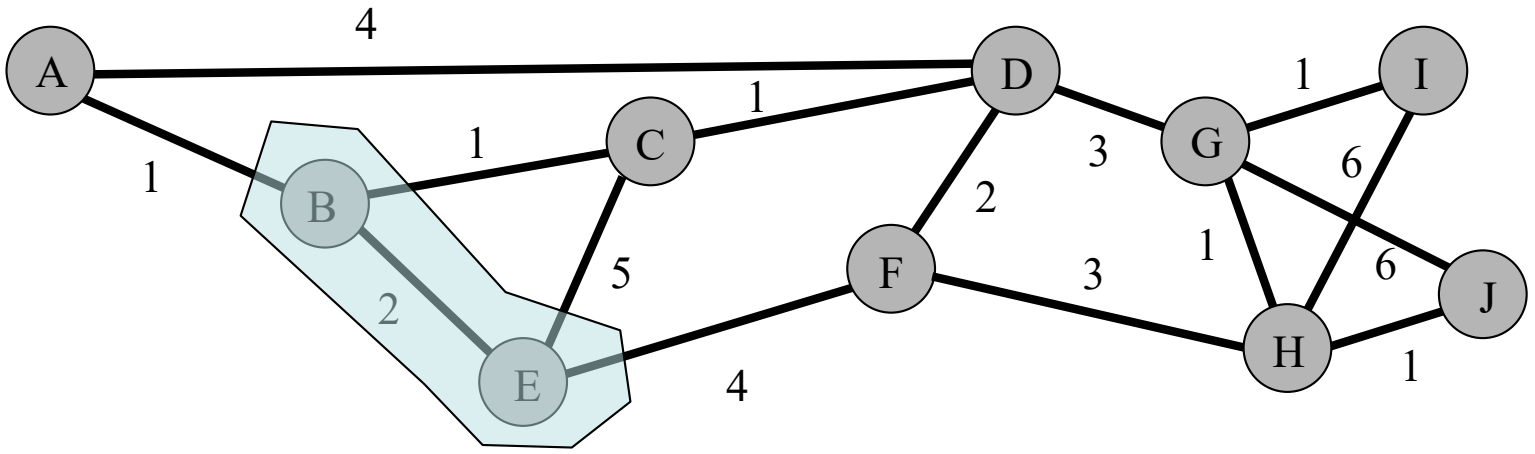


T={B,E}

- Recalculamos los L(n) de los nodos fuera de T

L(n)									
A	B	C	D	E	F	G	H	I	J
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞

$$L(n) = min[L(n), L(x) + w(x,n)], x \epsilon T$$

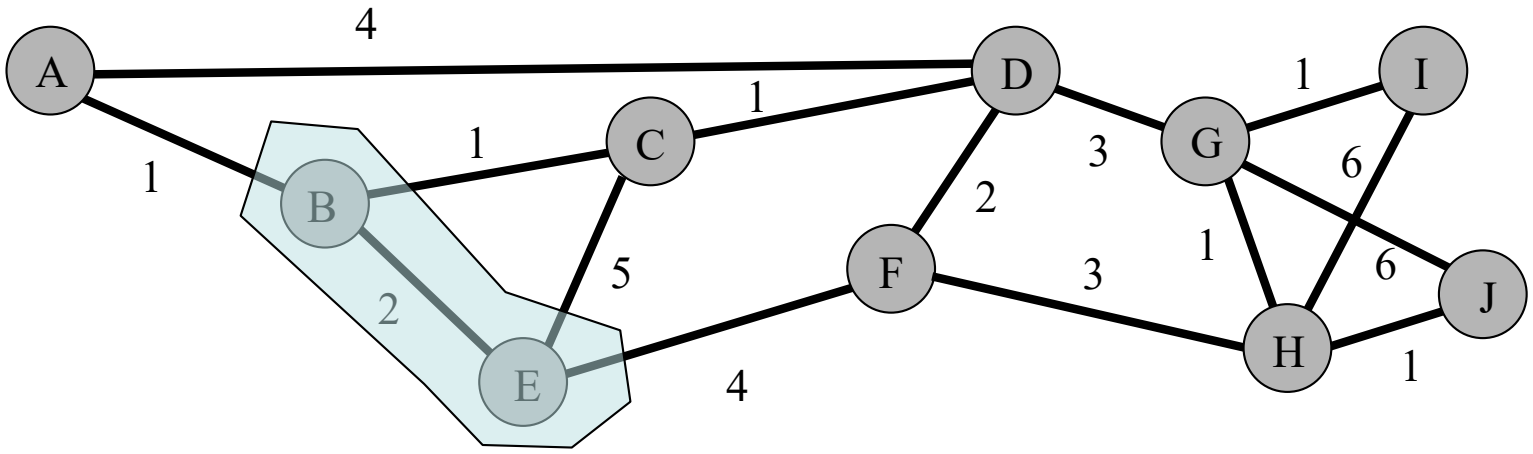


T={B,E}

- Vecinos de T son A, C y F
- Mínimo coste es a A o C (empate), añadimos A

L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞

$$Encontrar\ x \epsilon N, tal\ que\ x \notin T\ y\ L(x) = \min_{j \notin T} L(j)$$

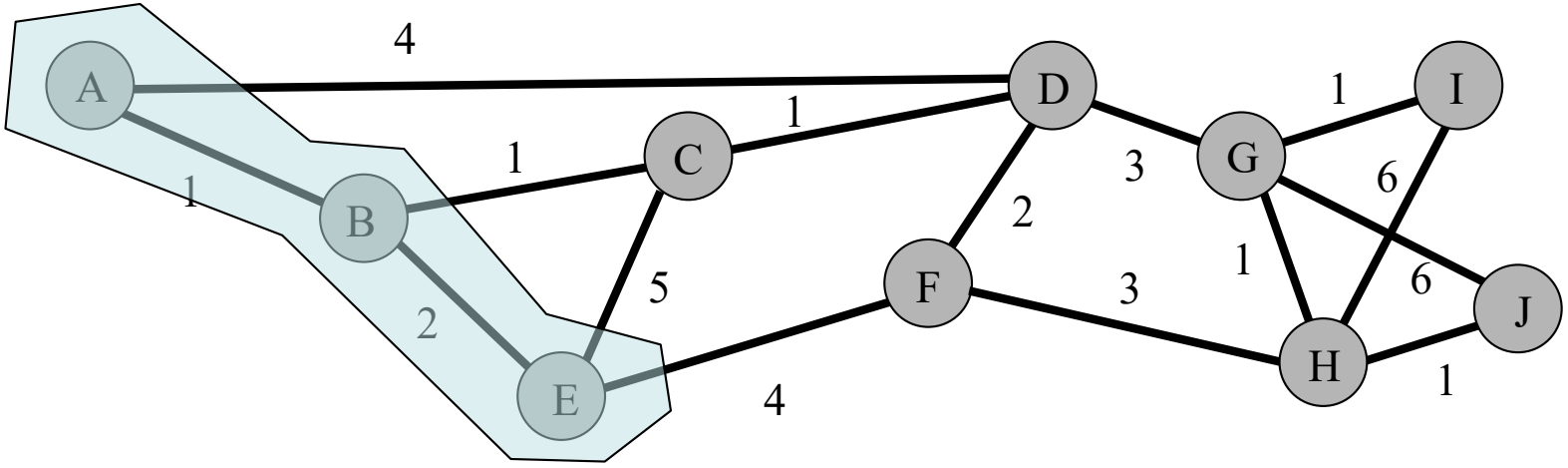


T={A,B,E}

- Recalculamos los L(n) de los nodos fuera de T

L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞

$$L(n) = min[L(n), L(x) + w(x,n)], x \epsilon T$$

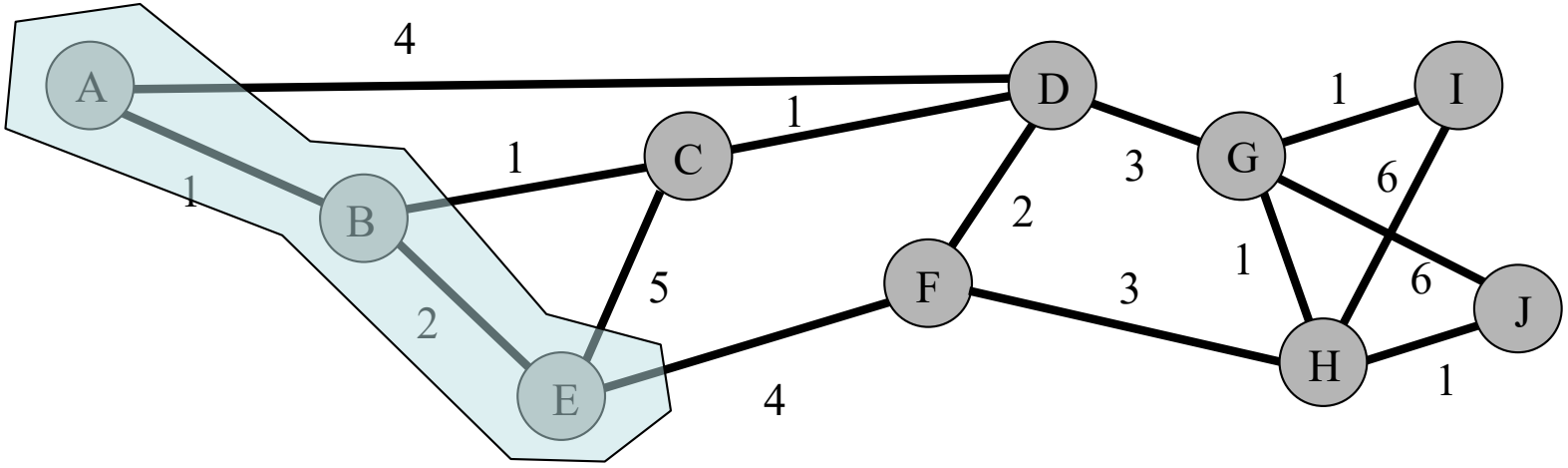


T={A,B,E}

- Vecinos de T son C, D y F
- Mínimo coste a C, lo añadimos

L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞

$$Encontrar\ x \epsilon N, tal\ que\ x \notin T\ y\ L(x) = \min_{j \notin T} L(j)$$

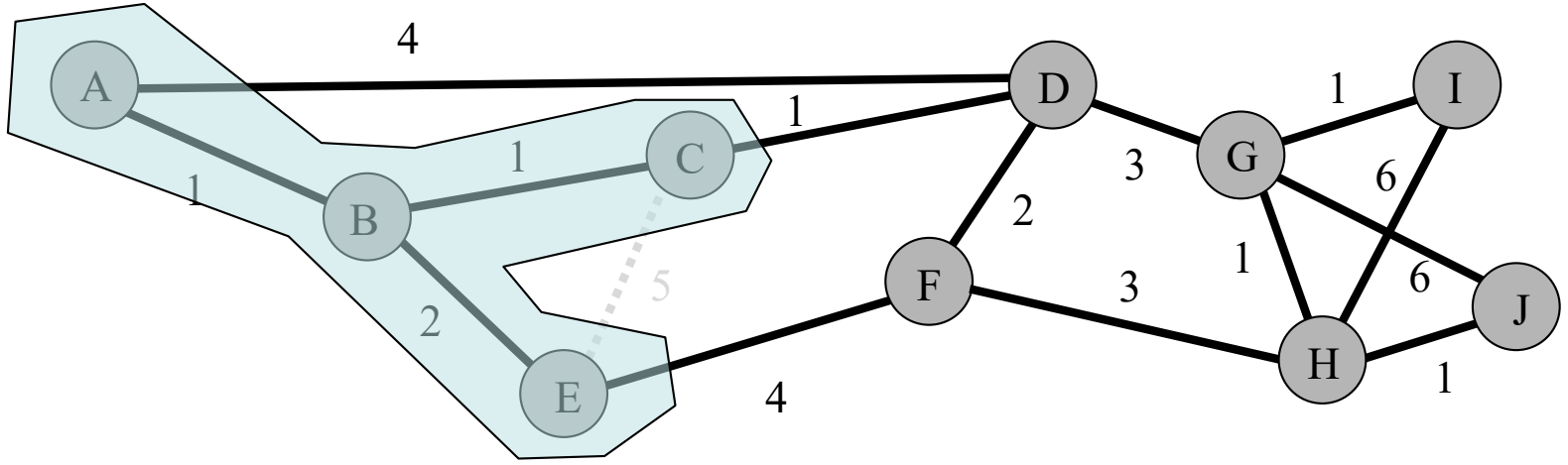


T={A,B,C,E}

- Recalculamos los L(n) de los nodos fuera de T

L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞

$$L(n) = min[L(n), L(x) + w(x,n)], x \epsilon T$$

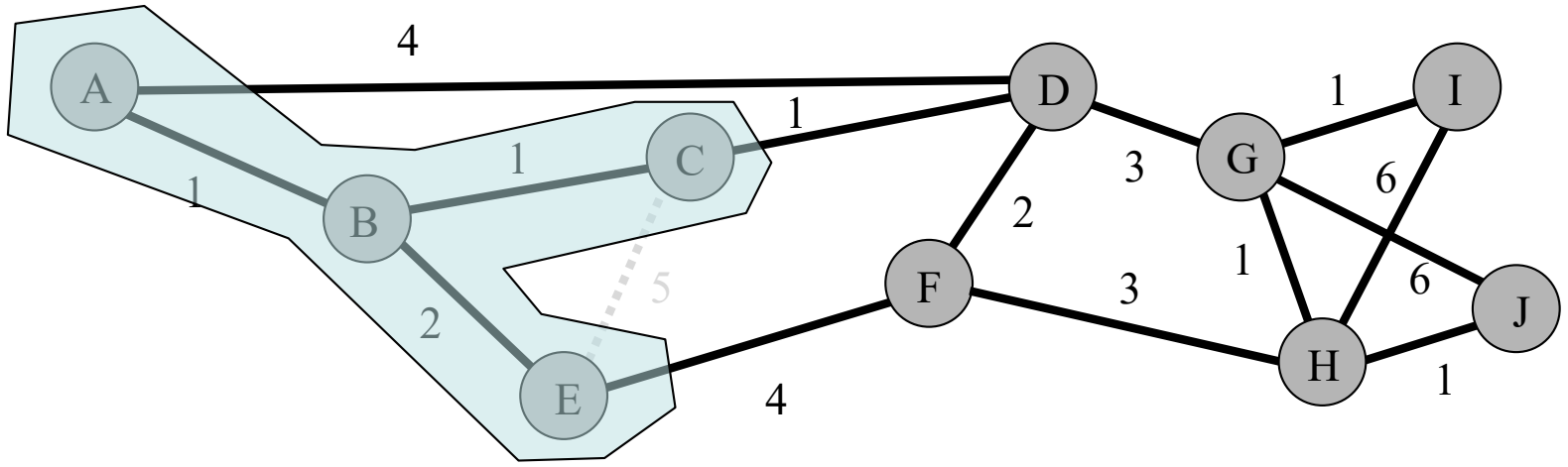


T={A,B,C,E}

- Vecinos de T son D (coste 4) y F (coste 4)
- Mínimo coste a F, lo añadimos (podríamos haber añadido D)

L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞

$$Encontrar\ x \epsilon N, tal\ que\ x \notin T\ y\ L(x) = \min_{j \notin T} L(j)$$

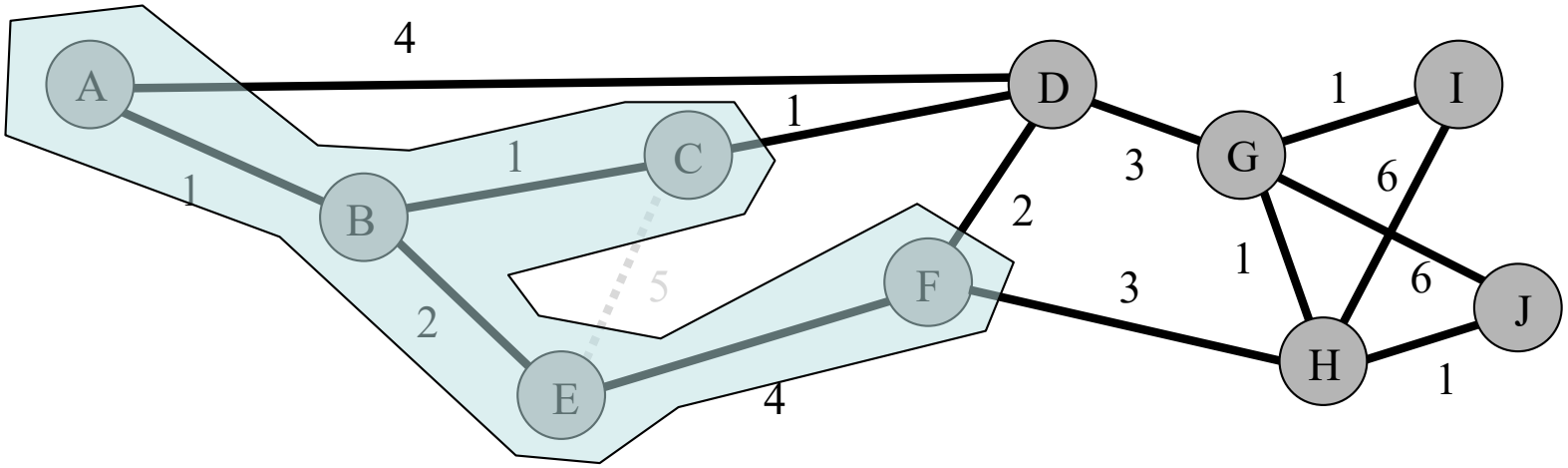


T={A,B,C,E,F}

- Recalculamos los L(n) de los nodos fuera de T

L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	7	∞	∞

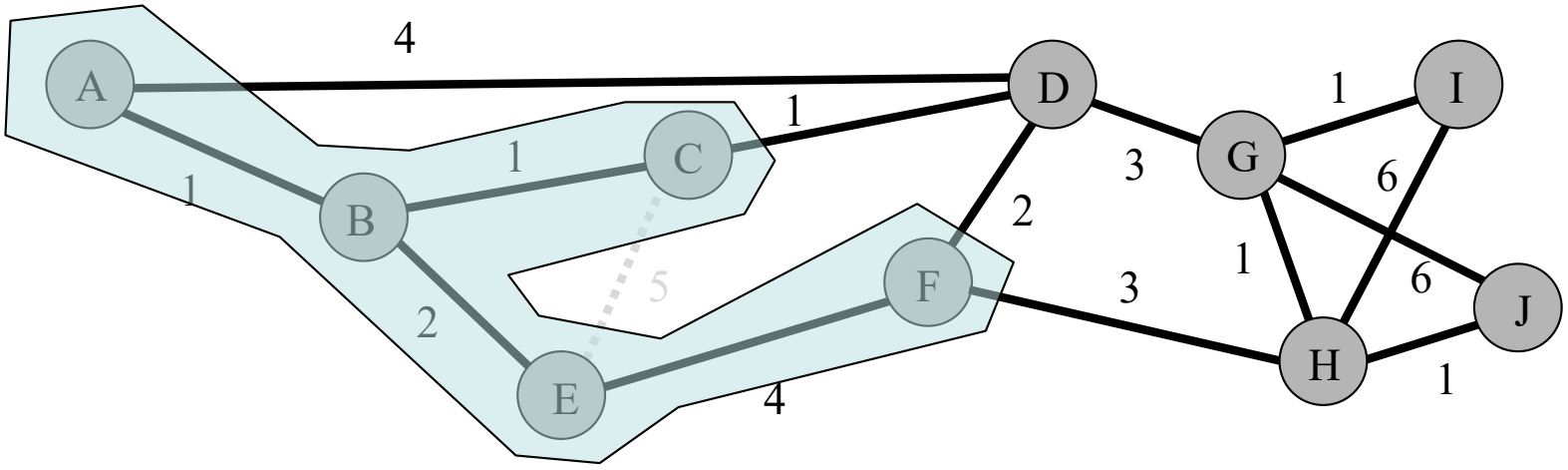
$$L(n) = min[L(n), L(x) + w(x,n)], x \epsilon T$$



T={A,B,C,E,F}

- Vecinos de T son D (coste 4) y H (coste 7)
- Mínimo coste a D, lo añadimos

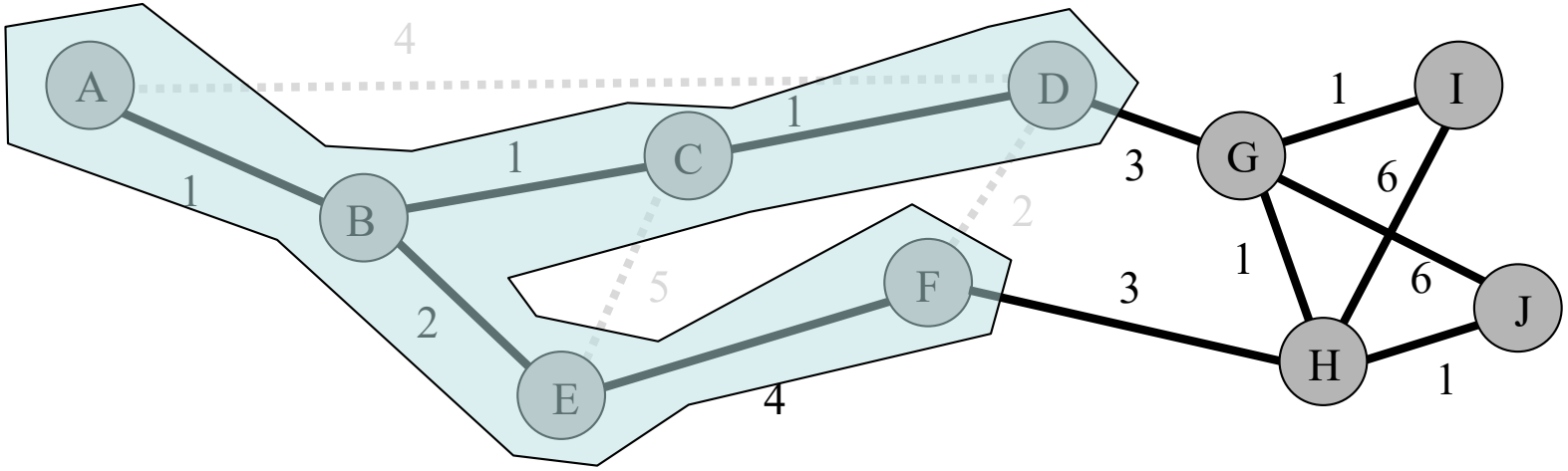
L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	7	∞	∞



T={A,B,C,D,E,F}

- Recalculamos los L(n) de los nodos fuera de T

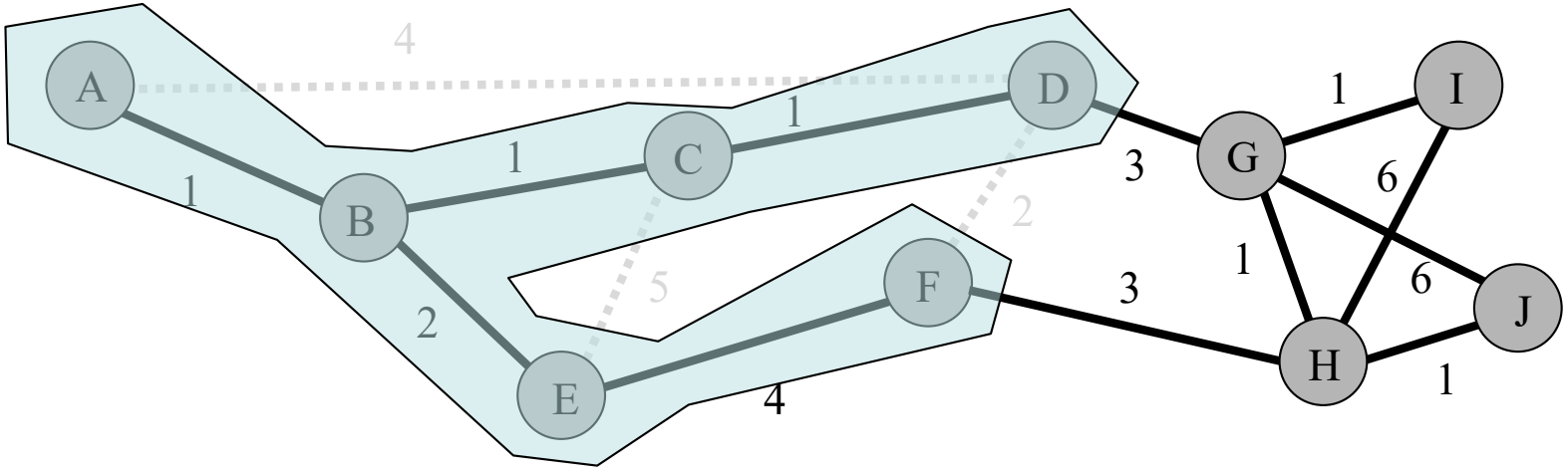
L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	7	7	∞	∞



T={A,B,C,D,E,F}

- Vecinos de T son G (coste 7) y H (coste 7)
- Mínimo coste a G (empate), lo añadimos

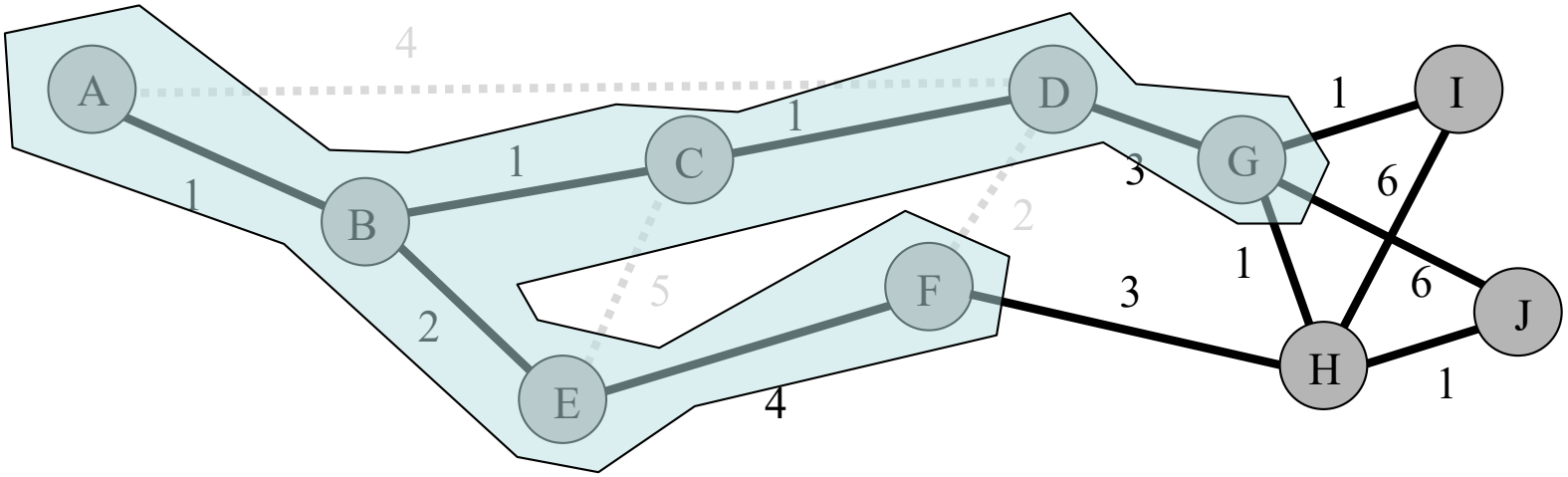
L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	7	7	∞	∞



T={A,B,C,D,E,F,G}

- Recalculamos los L(n) de los nodos fuera de T

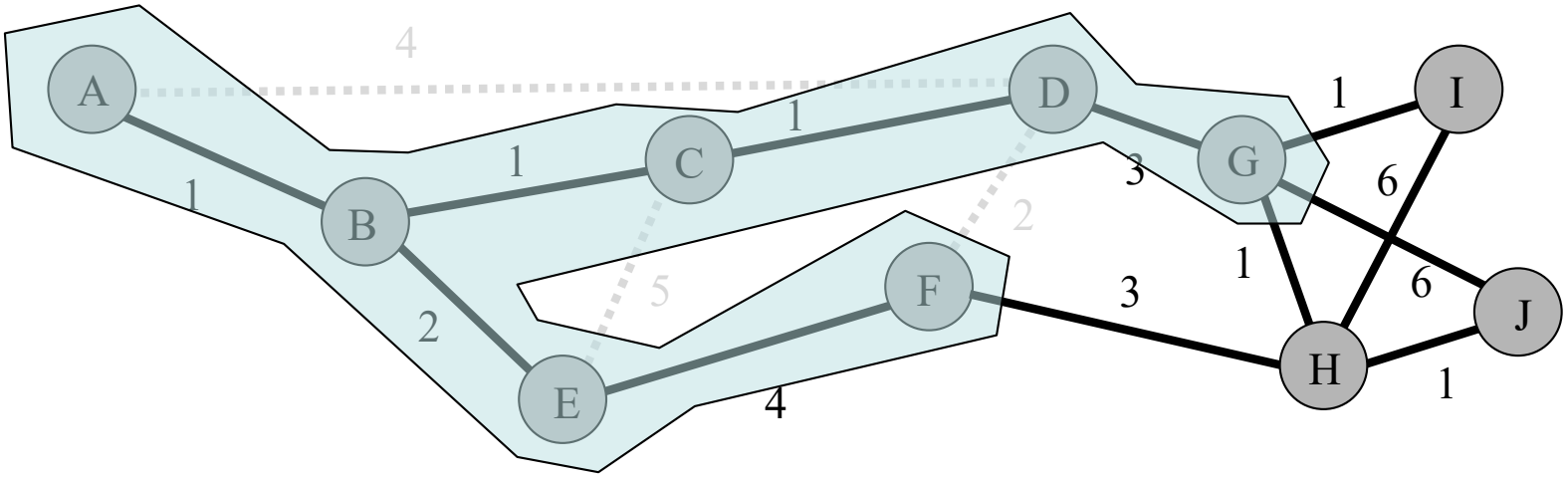
L(n)									
A	B	C	D	E	F	G	H	I	J
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	7	7	∞	∞
3	2	3	4	0	4	7	7	8	14



T={A,B,C,D,E,F,G}

- Vecinos de T son H (coste 7), I (coste 8) y J (coste 14)
- Mínimo coste a H, lo añadimos

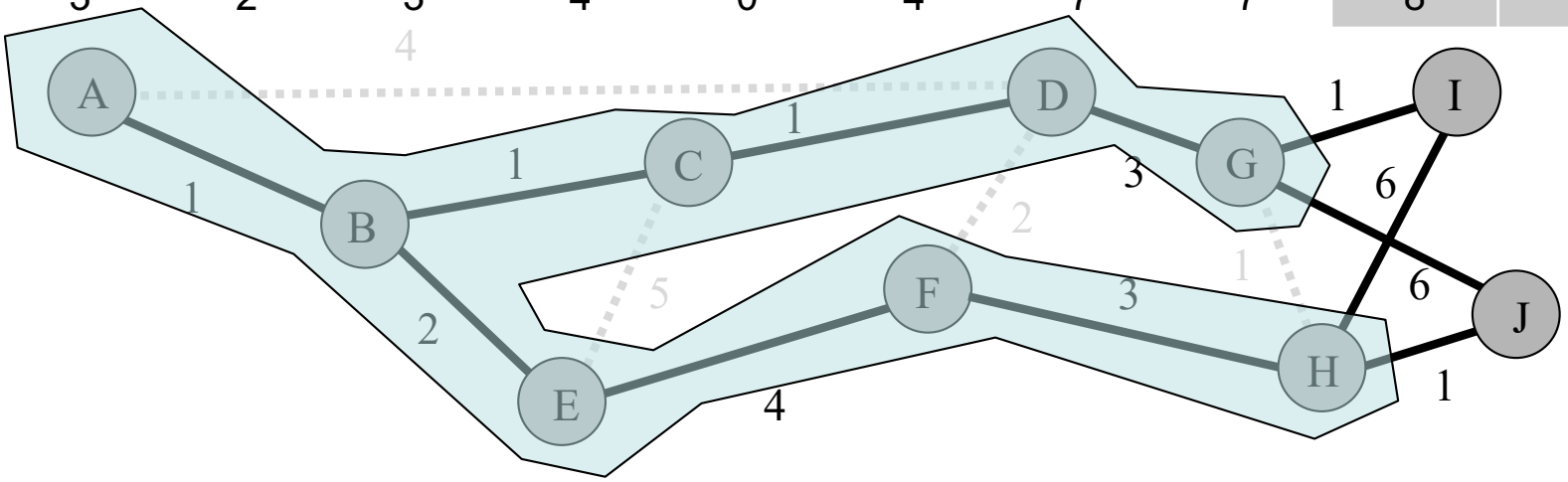
L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	7	7	∞	∞
3	2	3	4	0	4	7	7	8	14



T={A,B,C,D,E,F,G,H}

- Recalculamos los L(n) de los nodos fuera de T

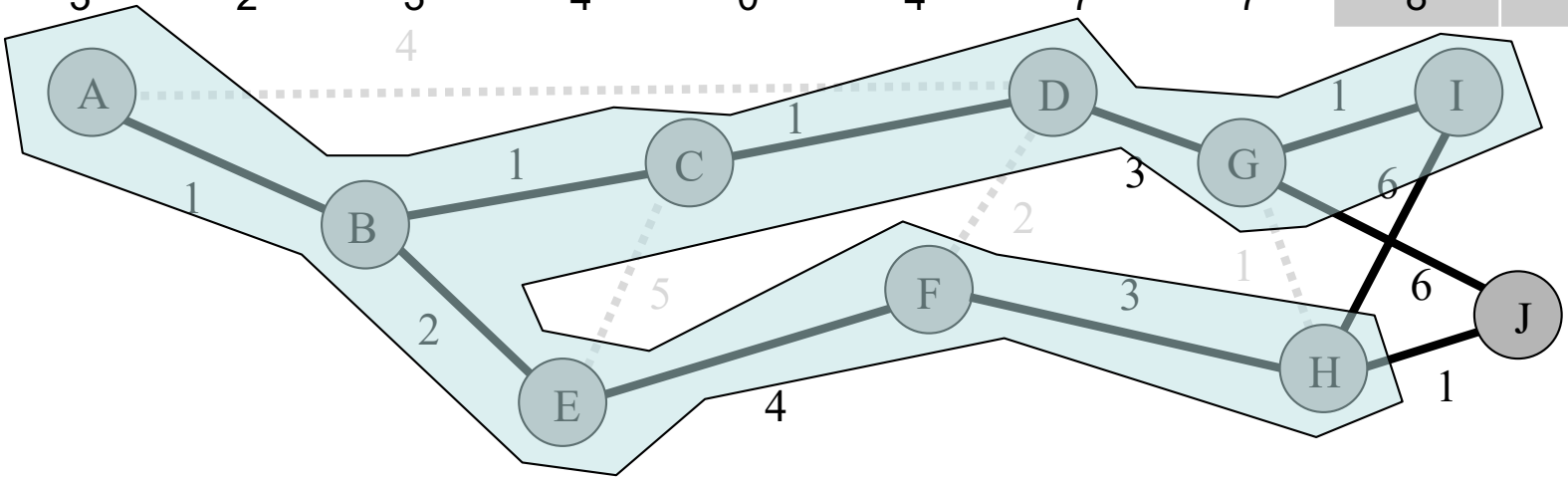
L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	7	7	∞	∞
3	2	3	4	0	4	7	7	8	14
3	2	3	4	0	4	7	7	8	8



T={A,B,C,D,E,F,G,H,I}

- Vecinos de T son I (coste 8) y J (coste 14)
- Mínimo coste a I (empate), lo añadimos

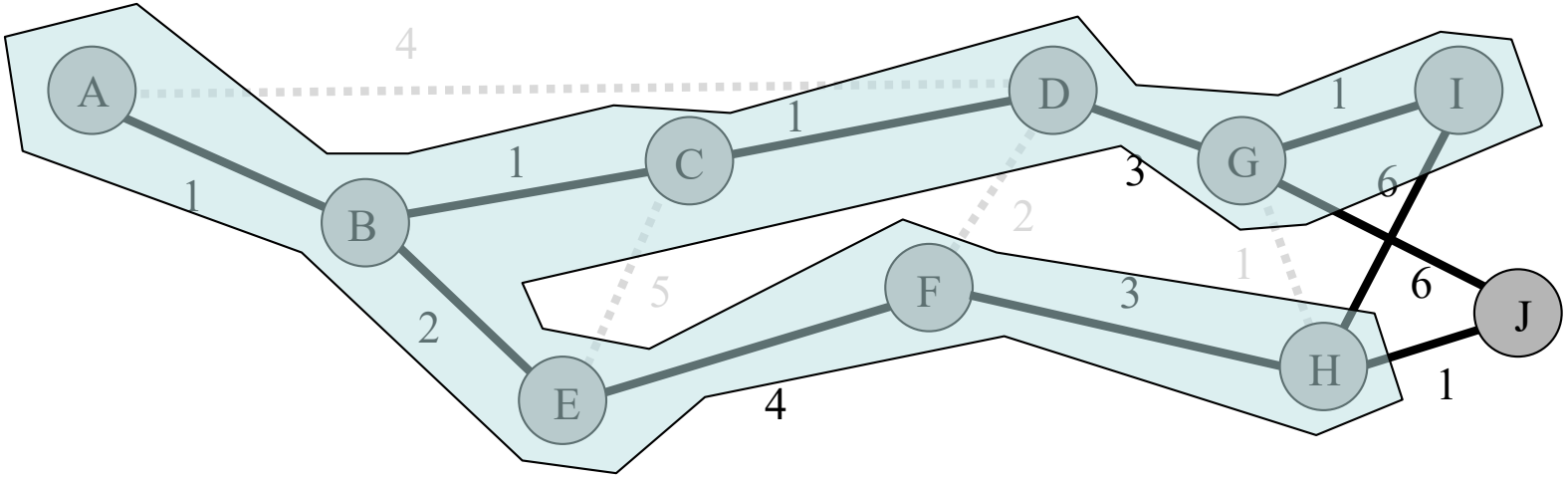
L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	7	7	∞	∞
3	2	3	4	0	4	7	7	8	14
3	2	3	4	0	4	7	7	8	8



T={A,B,C,D,E,F,G,H,I}

- Recalculamos los L(n) de los nodos fuera de T

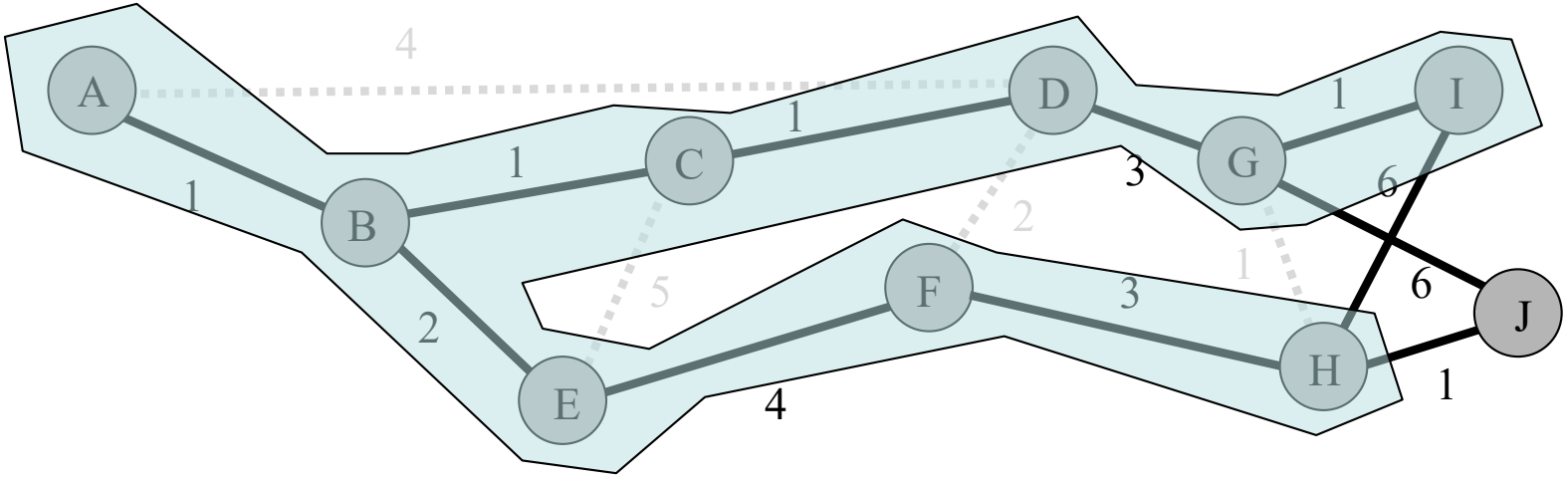
L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	7	7	∞	∞
3	2	3	4	0	4	7	7	8	14
3	2	3	4	0	4	7	7	8	8
3	2	3	4	0	4	7	7	8	8



T={A,B,C,D,E,F,G,H,I}

- Añadimos J

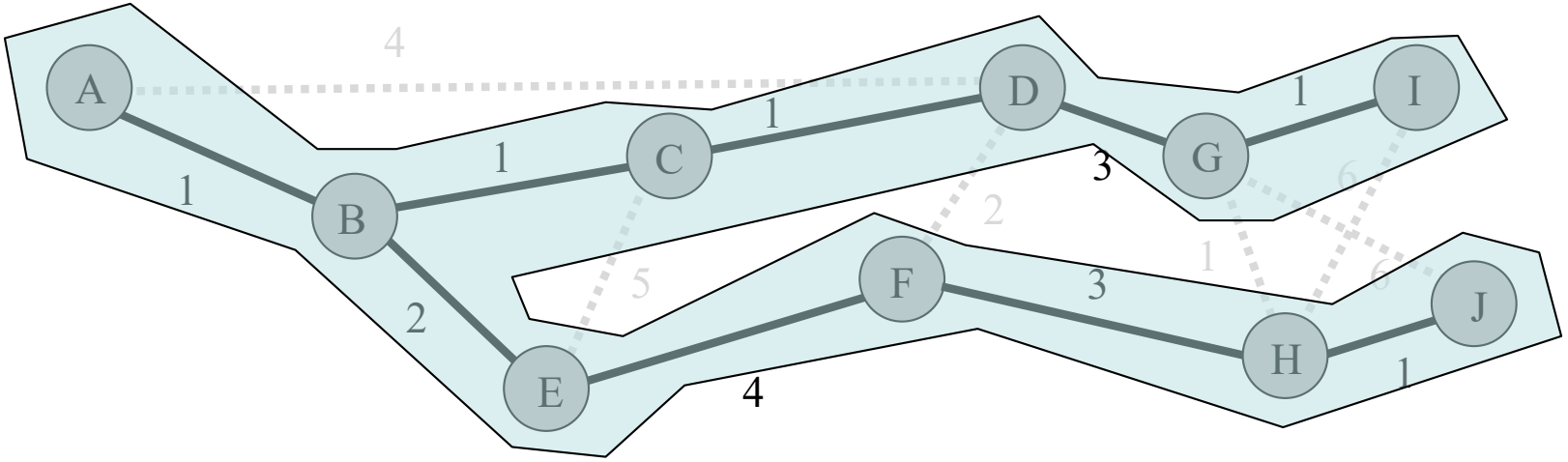
L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	7	7	∞	∞
3	2	3	4	0	4	7	7	8	14
3	2	3	4	0	4	7	7	8	8
3	2	3	4	0	4	7	7	8	8



T={A,B,C,D,E,F,G,H,I,J}

- Ya están todos los nodos en T. Árbol completado

L(n)									
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
∞	2	5	∞	0	4	∞	∞	∞	∞
3	2	3	∞	0	4	∞	∞	∞	∞
3	2	3	7	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	∞	∞	∞	∞
3	2	3	4	0	4	7	7	∞	∞
3	2	3	4	0	4	7	7	8	14
3	2	3	4	0	4	7	7	8	8
3	2	3	4	0	4	7	7	8	8



Resumen Link State

- Se crean adyacencias
- Se distribuyen LSPs
- Se calculan las rutas
- El algoritmo de distribución es delicado
- En general converge más rápido y genera menos tráfico que un DV
- Aunque hay protocolos DV bastante sofisticados que no distan mucho

Link State

Distance Vector vs Link State

Comparativa

- Los dos calculan un árbol de expansión de coste mínimo (*minimum spanning tree*)
- Dan el mismo resultado (salvo empates)
- ¿Cuál es más simple de programar? Probadlo

Comparativa: Memoria

- Supongamos una red con *n nodos*
- Cada nodo de la red tiene ***k*** vecinos

Distance-vector:

- Cada nodo almacena vector con la distancia desde cada vecino
- Vector consume memoria en orden *O(n)*
- Cada nodo consume *O(k x n)*

Link-state:

- Cada nodo guarda *n* LSPs
- Cada LSP contiene información de *k* vecinos: *O(k)*
- Cada nodo consume *O(k x n)*

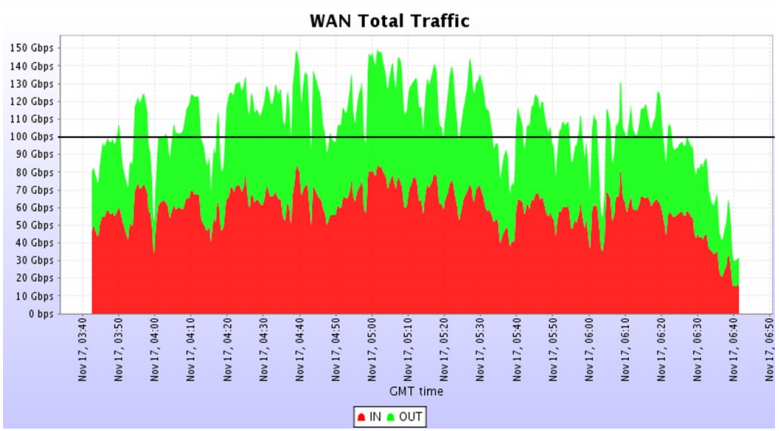
En general consumo *similar*

Comparativa: BW

- Según la topología es mejor uno u otro

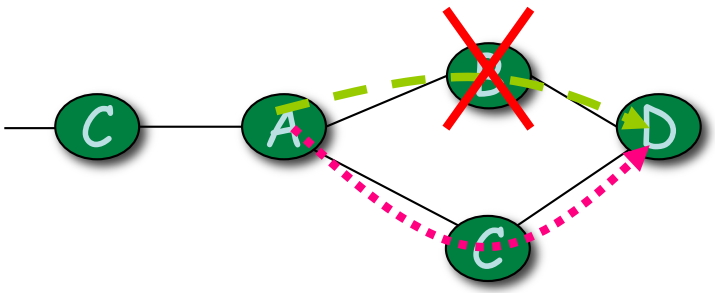
Ante fallo de enlace:

- ***Distance Vector***
  - Solo se propagará hasta donde se "note" el cambio
  - Muchos paquetes por cuentas a infinito
  - Pueden cambiar muchos costes en rutas que por tanto hay que anunciar (paquete grande)



- ***Link State***
  - El cambio se informa a toda la red
  - LSP circula una vez por cada enlace
  - El tamaño del LSP es fijo

En general consumo *modesto*



Comparativa: Cálculo

- ***n*** nodos, cada uno ***k*** vecinos

***Distance-vector*:**

- BF requiere recorrer la matriz de vectores distancia: *O(n x k)*
- No siempre hay que recorrer la matriz (ej: solo cambia el coste)
- Un nuevo nodo extremo requiere cálculos en muchos nodos
- Más que puede haber varias iteraciones

***Link-state*:**

- Dijkstra lleva *O(n x k x log(n))*
- Se puede hacer un cálculo *incremental* del árbol de expansión mínimo
- Un nuevo nodo extremo implica cambios mínimos

Coste de cálculo en ambos es *razonable* y *modesto*

El resultado no es “tan urgente”

Comparativa: Robustez

- Ninguno de los dos es mejor ante fallos soft/hard, errores de configuración, sabotaje, etc.
  - Nodos que anuncien tener enlaces que no tienen
  - Nodos que no anuncian enlaces que tienen
  - Que calculen mal las rutas
  - Que corrompan los LSPs
  - Que no reenvíen los paquetes o los reenvíen por el camino incorrecto
  - Etc
- Depende de lo probable que sea cada tipo de error y los daños que cause
- DV es un poco más robusto porque cada uno realiza su cálculo de rutas

Comparativa

- **Funcionalidad**
  - LS ofrece conocimiento de la topología completa de la red con solo preguntar a un router cualquiera
  - Se puede emplear *source-routing*
  - Calcular rutas diferentes para distintas clases de servicio con mayor sencillez
- **Velocidad de convergencia**
  - DV tiene el problema de las cuentas a infinito
  - Las técnicas para resolverlo también llevan tiempo
  - Aunque se resolviera, es más lento porque no puede reenviar la información de enrutado hasta hacer sus cálculos locales

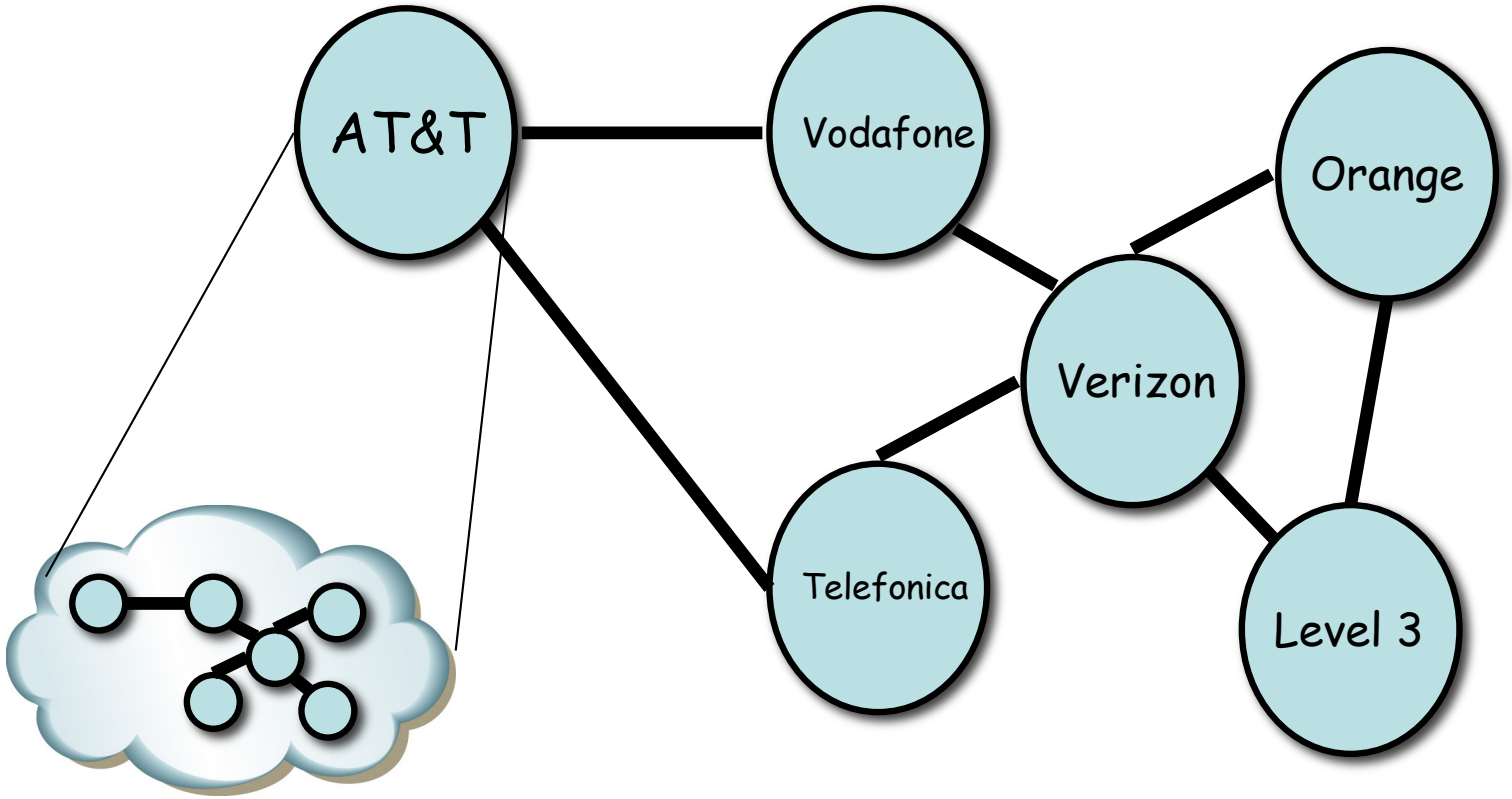
upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

ARQUITECTURA DE REDES, SISTEMAS Y SERVICIOS
Área de Ingeniería Telemática

Encaminamiento en Internet

Autonomous Systems

- Ya hemos comentado que Internet es la interconexión de un conjunto de Sistemas Autónomos (ASs)
- Cada uno gestión independiente
- Cada uno decide cómo hacer el encaminamiento interno
- Los protocolos que se usan se llaman IGPs (*Interior Gateway Protocols*)



Autonomous Systems

- El encaminamiento **entre** esos sistemas autónomos sí necesita coordinación
- Comparten información equipos de gestores independientes
- El mecanismo empleado es similar a *distance vector,* aunque se anuncia en realidad el camino (ASs por los que pasa)
- Se llama *path vector*