upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

Tecnologías Avanzadas de Red

Área de Ingeniería Telemática

Routing: Protocolos *Link State*

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

Grado en Ingeniería en Tecnologías de Telecomunicación, 3º

Objetivos

- Conocer los principios de diseño de los protocolos Link State
- Comprender los problemas y soluciones asociados a la distribución de los mensajes por inundación

Link-State: Introducción

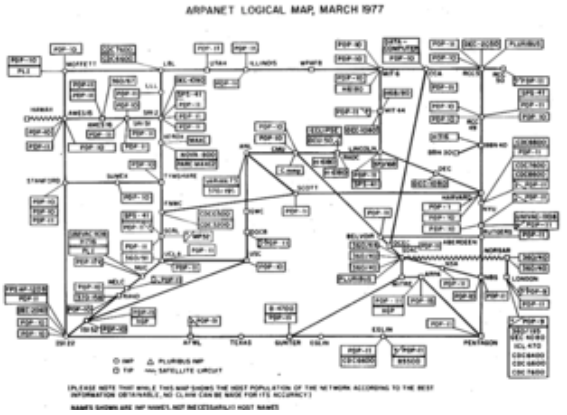
Routing en ARPANET

1969-1978

- Distance Vector
- El coste es la longitud de cola
- Bellman-Ford
- Problemas
  - Elevados tiempos de convergencia (cuentas a infinito)
  - No tiene en cuenta el BW (había enlaces de 56Kbps y de 1.5Mbps)

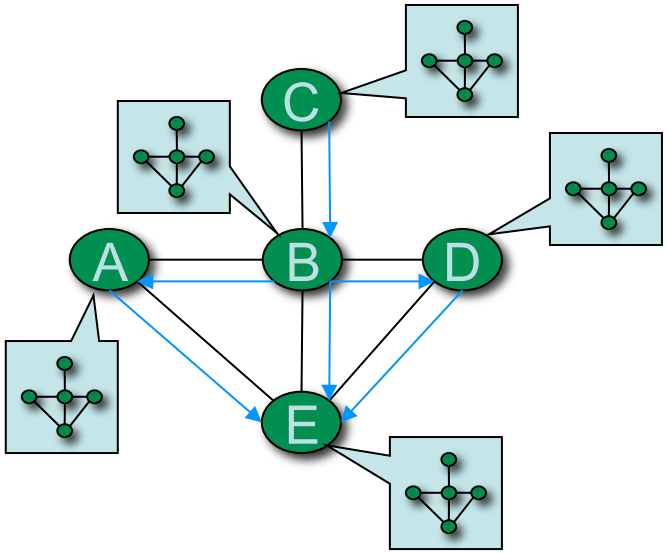
1979-1986

- Link State
- El coste es el retardo medido en un intervalo de 10s
- Algoritmo de Dijkstra
- Problemas
  - Oscilaciones
  - Mayor variación en el retardo
  - Updates más frecuentes



Características globales LS

- B.D. distribuida replicada
- Cada router posee información global sobre la red: nodos y enlaces existentes
- ¿Cómo?
  - Informan de sus enlaces a redes activas y con routers vecinos
  - "Inundan" la red con esta información
  - "**Cómo**" hacer esta inundación es uno de los principales problemas de estos protocolos
- Todos los routers tienen una imagen (grafo) de la red
- A partir de ella eligen los caminos
- Menor tiempo de convergencia que DV ante cambios en la red
- Permiten calcular caminos con diferentes requisitos de métrica
- Ejemplos: OSPF, IS-IS, PNNI, DEC's DNA Phase V, Novell's NLSP



Tareas principales del proceso

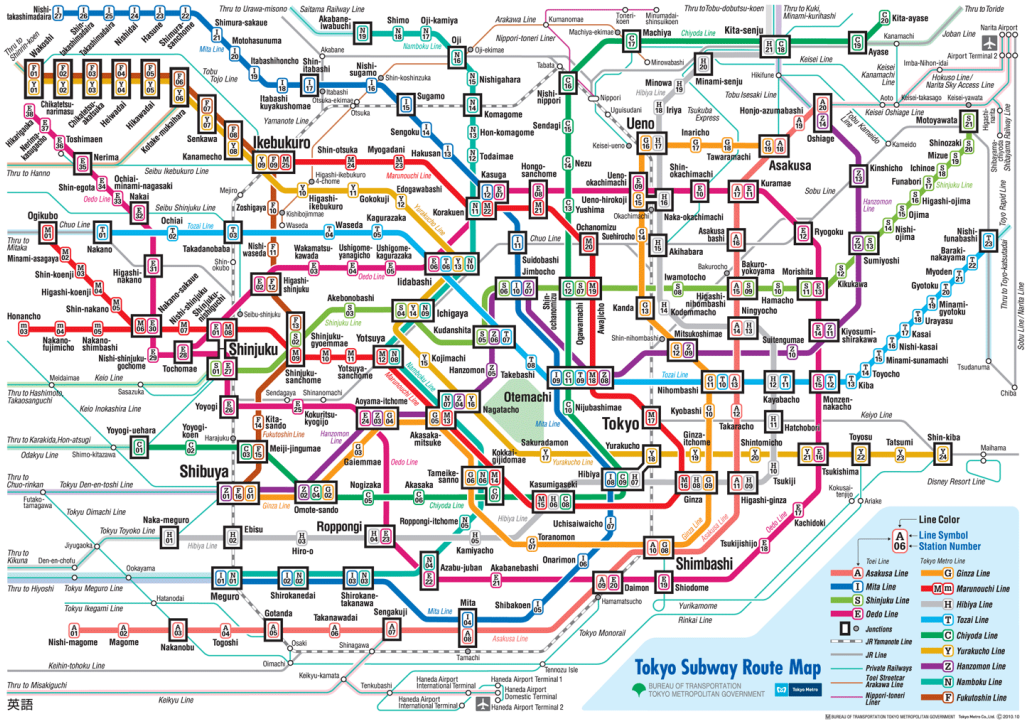
- *Meeting neighbors*
  - En enlaces punto-a-punto transmiten un paquete identificándose
  - En LANs transmiten periódicamente un paquete especial a una dirección del grupo
- Construir un LSP (Link State Packet) cuando
  - Descrubre un nuevo vecino
  - Cambia el coste de un enlace con un vecino
  - Un enlace con un vecino desaparece
  - Periódicamente
- Diseminar el LSP a todos los routers de la red
- Calcular las rutas

LSP

Nodo origen
Vecinos

Cálculo de las rutas

- Cada router tiene conocimiento completo de la red
- Algoritmo de Dijkstra para calcular los caminos
- Se pueden calcular rutas alternativas para diferentes clases de servicio

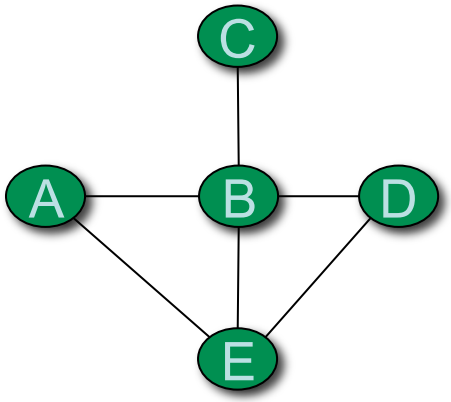


Edsger Wybe Dijkstra
(1930-2002)

Problema de diseminación de LSPs

Diseminación de LSPs

- Objetivos:
  - Todos los routers posean la misma B.D. de LSPs
  - Evitar saturar la red en el proceso
- Si se hace mal:
  - Diferentes LSPs en cada router → calculan rutas incoherentes
  - Puede volverse cancerígeno
- Ojo, ¡¡ No puede usar la tabla de rutas para hacer llegar el LSP a todos los routers !!
- Solución básica: *flooding* (inundación)

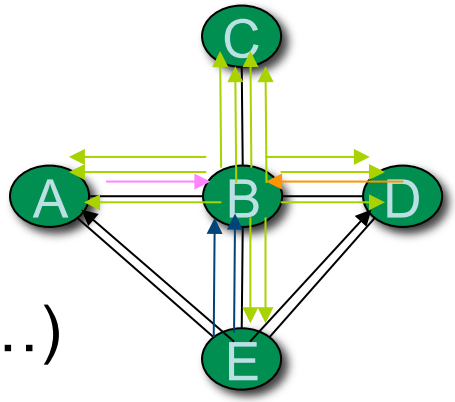


Flooding

- Esquema:
  - ***Cada LSP recibido/generado se transmite a cada vecino excepto a aquel del cual se recibió*** (. . .)
  - Se incluye un TTL para evitar que nunca deje de reenviarse
- No depende de las tablas de rutas
- Problema:
  - ¡ Se crea un número exponencial de copias del LSP !

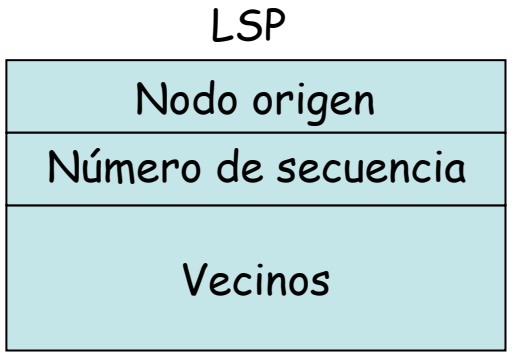
- Mejora:
  - Cada router guarda una copia del LSP
  - Si le llega de nuevo ve que ya lo ha transmitido y no lo retransmite
  - Así cada LSP solo aparece una vez en cada enlace
- Problema:
  - Si recibe un LSP de un router que no es igual al que tiene, ¿cuál es el más moderno?

(etc…)



Flooding y timestamps

- Para reconocer al LSP más reciente se incluye un timestamp en el mismo
- Problema:
  - ¿Y si accidentalmente se genera un LSP con un timestamp de un futuro muy lejano?
  - No se aceptarían LSPs de ese router en mucho tiempo
- Solución 1:
  - Timestamps de tiempo absoluto.
  - Si está muy desviado del instante actual se ignora
- Problema:
  - Requiere sincronización entre los routers
  - En sí mismo más complejo que el algoritmo de distribuir LSPs
- Solución 2:
  - Usar un número de secuencia



Flooding, secuencia

Timestamp → nº de secuencia

- Cuando un router genera un nuevo LSP usa un nuevo número de secuencia
- Cuando recibe un LSP comprueba que el número de secuencia sea mayor

Problema:

- Tamaño finito (32bits) ¿Y cuando se alcance el máximo?

Solución 1:

- Que sean 64 bits (muy difícil desbordarse)

Problema:

- Corromperse o generarse mal un valor muy alto

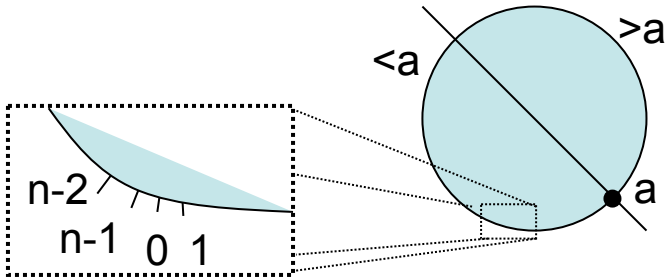
Solución 2:

- Reset o wrap, ej a < b si
  - a < b y | a - b | ≤ n/2

  o
  - a > b y | a - b | > n/2

Problema:

- Si un router se reinicia y
- Comienza de nuevo con secuencia 0
- O se particiona la red y se repara mucho más tarde



Flooding, secuencia y edad

Edad

- Comienza en un valor
- Va siendo decrementado por los routers a medida que permanece en memoria
- Cuando alcanza 0 se elimina el LSP
- Se aceptará otro independiente del número de secuencia

LSP

Nodo origen
Número de secuencia
Edad
Vecinos

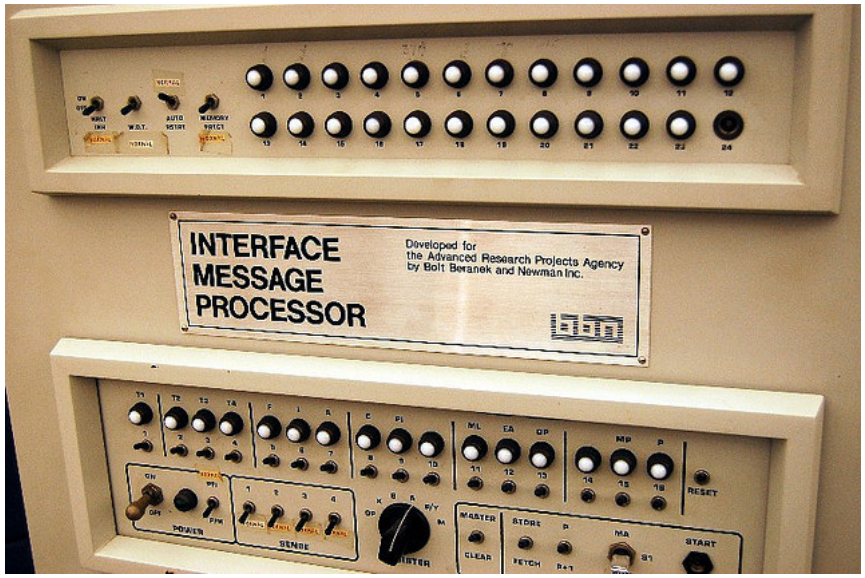
Nuevo enrutamiento en ARPANET

- Secuencia (6 bits) +1 en cada nuevo LSP
- Nuevo LSP cada 60 seg
- Al recibir LSP de router R:
  - Aceptar si mayor número de secuencia (aritmética módulo)
  - Si edad del almacenado = 0, aceptar
  - Si lo acepta: reenviar a vecinos
- Edad:
  - 3 bits, unidades de 8 seg
  - Comienza en el máximo (56 seg)
  - Cuando alcanza 0 ya no se propaga pero se guarda
- Tras arrancar espera 90 seg antes de mandar el primer LSP

The ARPANET “incident”

The ARPANET "incident"

- 27 de Octubre de 1980
- ARPANET *deja de funcionar*
- La red es incapaz de cursar tráfico
- Los IMPs están saturados con LSPs
- Todos provienen del mismo nodo (IMP 50)
- Los mismos 3 números de secuencia repetidos: 8, 40 y 44
- Y en este orden: 8, 40, 44, 8, 40, 44, 8, 40…
- ¿Qué estaba sucediendo? ¿Por qué?



The ARPANET "incident"

¿Qué estaba sucediendo?

- 8, 40, 44, 8, 40, 44, 8, 40
- Cada uno reemplazado por el siguiente
- No hay tiempo de que se agote su edad

¿Por qué?

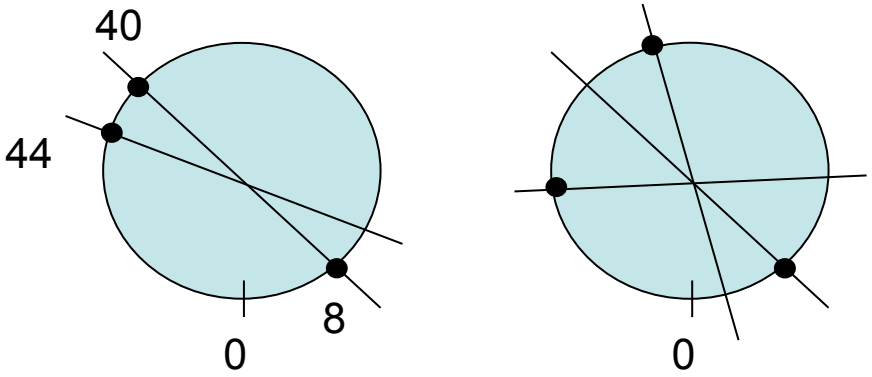
- Los 3 LSPs idénticos salvo por el nº de secuencia (…)

  8 = 001000

  40 = 101000

  44 = 101100

¡ Cambio de un bit !



¡¡ 8 < 40 < 44 < 8 < 40 < 44 < 8 … !!

The ARPANET "incident"

¿Cómo recuperarse?

- ¿Reiniciar un nodo cualquiera? Vuelve a contaminarse
- ¿Apagar todos y después empezar a encenderlos? Complicado y largo

¿Qué se hizo?

- Apagar el IMP generador del error y arreglarlo
- Modificar el S.O. de los IMPs para que ignoren esos LSPs
  - Cargarlo en el IMP local
  - Luego en los vecinos, etc.
- La red queda limpia del "gusano"
- Volver a la versión original del S.O.
- ¡ Y replantearse el algoritmo de distribución !

Requisitos para un protocolo LS

- Distribuir LSPs
  - Correctamente (sin alteraciones)
  - Completamente (a todos)
- Si no, diferentes grafos de la red
  - Cálculo de rutas inconsistentes
  - *Routing loops*
- Pero esto no es suficiente

Requisitos para un protocolo LS

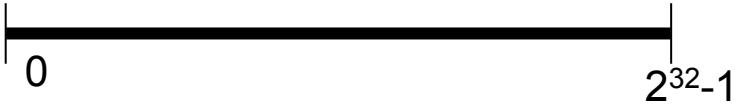
- **Self-stabilization:**
  - Recuperarse de paquetes corruptos, equipo defectuoso, ataques…
  - *Pase lo que pase, tras desconectar todo el equipamiento defectuoso la red volverá a un estado normal en un periodo tolerable de tiempo (aprox 1h)*
- **Eficiencia:**
  - No generar demasiados paquetes ni agotar recursos de routers
  - Hay regeneración periódica de LSPs pero de frecuencia muy baja (1h, para recuperarse de eventos raros)
- ***Responsiveness*:**
  - No hay requisito de esperar 90 segs antes de empezar
  - Salvo que se le acaben los números de secuencia, en cuyo caso debe esperar a que caduquen sus LSPs

Distribución de LSPs actual

Esquema de distribución actual

Número de secuencia:

- Espacio lineal desde 0
- Al alcanzar el máximo no se aceptan LSPs de esa fuente hasta que los que hay caducan
- Grande para no alcanzarlo salvo ante fallos (32bits)



Edad (*age*):

- Inicializado por el router generador del LSP (~1h)
- Todo router que los procesa lo decrementa
- Se decrementa mientras está en memoria

***DECnet Phase V*, IS-IS, OSPF, PNNI**

Send flags:

- Un LSP que se debe retansmitir no se encola inmediatemente
- Se marca (*flag*) como que debe enviarse
- Un flag por cada enlace por el que debiera salir
- LSPs más modernos del mismo router pueden sobreencribir el antiguo sin haberlo enviado
- Si hay BW disponible, RR por las colas enviándolos (LSPs o ACKs)

Confirmaciones:

- Vecinos confirman LSPs
- *Flag* indicando que se debe enviar *ack* de un LSP por un enlace

Empleo de los *flags* (I)

- *Send flags* y *ACK flags*
- Por cada LSP y enlace
- (2 x nº links) flags por cada LSP
- No pueden estar activos simultáneamente

Se acepta un LSP (…):

- Activar todos los *send flags* y desactivar por donde se recibió (…)
- Desactivar *ack flags* y activar por donde se recibió (…)

LSPs	*Flags*	Vecinos				
R1 Links'	*Send*	√	√	√	R1 Links'	√
	ACK					√
R2 Links	*Send*	√			√	
	ACK					√
R3 Links	*Send*		√			√
	ACK			√		
…	…	…	…	…	…	…

Empleo de los *flags* (II)

- *Send flags* y *ACK flags*
- Por cada LSP y enlace
- (2 x nº links) flags por cada LSP
- No pueden estar activos simultáneamente

Se recibe un LSP duplicado (…):

- Borrar el *send flag* del enlace por donde se recibió (…)
- Activar el *ack flag* de ese enlace (…)

LSPs	*Flags*	Vecinos				
R1 Links'	*Send*	√	√	√		√
	ACK				√	
R2 Links	*Send*	R2 Links			√	
	ACK					√
R3 Links	*Send*		√			√
	ACK			√		
…	…	…	…	…	…	…

Empleo de los *flags* (III)

- *Send flags* y *ACK flags*
- Por cada LSP y enlace
- (2 x nº links) flags por cada LSP
- No pueden estar activos simultáneamente

Se recibe un ACK (…):

- Borrar el *ack flag* del enlace por donde se recibió
- Borrar el *send flag* del enlace por donde se recibió (… …)

LSPs	*Flags*	Vecinos				
R1 Links'	*Send*	√	√	√		√
	ACK				√	
R2 Links	*Send*				√	
	ACK	√				√
R3 Links	*Send*		√	ACK √		ACK
	ACK					
…	…	…	…	…	…	…

Empleo de los *flags* (IV)

- *Send flags* y *ACK flags*
- Por cada LSP y enlace
- (2 x nº links) flags por cada LSP
- No pueden estar activos simultáneamente

Si hay BW disp. en un enlace:

- RR por la B.D.
- *Send flag?* ⇒ Enviar LSP (… ...)
- *ACK flag?* ⇒ Enviar ACK (… …)

LSPs	*Flags*	Vecinos				
R1 Links'	*Send*	√	√	√		R1 Links'
	ACK				√	
R2 Links	*Send*				√	ACK √
	ACK	√				
R3 Links	*Send*		√			
	ACK					
…	…	…	…	…	…	…

Empleo de los *flags* (y V)

- *Send flags* y *ACK flags*
- Por cada LSP y enlace
- (2 x nº links) flags por cada LSP
- No pueden estar activos simultáneamente

La edad de un LSP llega a 0 (…):

- Borrar los enlaces (…)
- Queremos que caduque a la vez en todos
- Reenviarlo y borrarlo tras *ack* (…)
- Es decir, se re-inunda al llegar a 0 y se borra al ser confirmado por todos
- Si se tiene un LSP y recibe otro de mismo origen con age=0 lo acepta aunque tenga igual secuencia

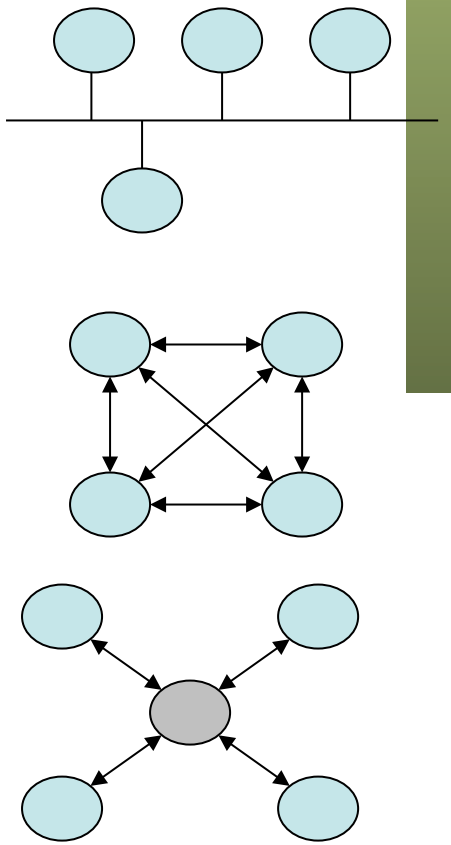
LSPs	*Flags*	Vecinos				
R1 Links'	*Send*	√	√	√		
	ACK				√	
R2	*Send*	√	√	√	√	√
	ACK	√				
R3 Links	*Send*		√			
	ACK					
…	…	…	…	…	…	…

Redes multi-acceso

- Este esquema asume que los enlaces son punto-a-punto

Si están conectados a una LAN

- Depende del protocolo cómo se resuelva
- Una posibilidad es tratar la LAN como una malla (…)
  - Muchos LSPs a enviar
- Otra es tratar la LAN como un nodo (…)
  - Se elige a un router de esa LAN que crea un LSP por ella



Resumen

- Se crean adyacencias
- Se distribuyen LSPs
- Se calculan las rutas
- El algoritmo de distribución es delicado
- Requiere poder ordenar los LSPs de un origen
- Y poder hacerlos caducar
- Se confirman
- En general converge más rápido y genera menos tráfico que un DV
- Aunque hay protocolos DV bastante sofisticados que no distan mucho

DS vs LS

Comparativa: Memoria

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

***Distance-vector*:**

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

Con jerarquía:

- En niveles altos no es necesario ruta a todos los destinos
- Rutas agregadas
- El consumo sería menor con DV

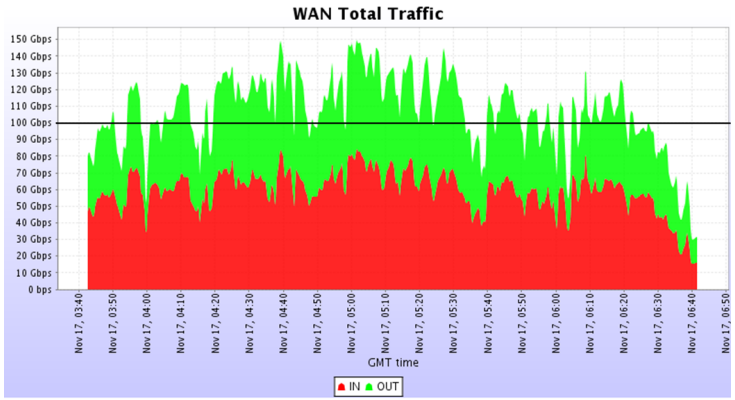
En general consumo *similar*

Comparativa: BW

- Según la topología es mejor uno u otro

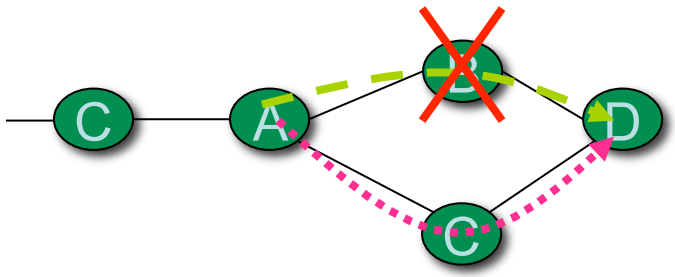
Ante fallo de enlace:

- ***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 cambio 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.
  - Routers que anuncien tener enlaces que no tienen
  - Routers que no anuncian enlaces que tienen
  - Que generen una secuencia extraña de números de secuencia
  - Que calculen mal las rutas
  - Que no reenvíen o confirmen LSPs
  - Que corrompan la edad de los LSPs
  - Que corrompan los datos de LSPs al reenviarlos
  - Que corrompan los números de secuencia al reenviar 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 la cuenta a infinito
  - Las técnicas para resolverlo (*hold-down interval*) también llevan tiempo
  - Aunque se resolviera, es más lento porque no puede reenviar la información de enrutado hasta hacer sus cálculos locales