

Computación Centrada en Redes

Routing dentro de una red

Routing

Gerson Garrido

Semana 5

UNIVERSIDAD NACIONAL DE INGENIERÍA

Hoy

Sesión A: los algoritmos

1. El routing como un problema de grafos
2. Estado de enlace: Dijkstra
3. Vector de distancias: Bellman-Ford
4. Buenas y malas noticias
5. Comparación
6. Dos tablas que no son la misma

Sesión B: OSPF

7. Por qué un solo algoritmo no alcanza
8. OSPF con FRRouting
9. Áreas y backbone
10. Convergencia: lo lento es enterarse
11. Cuando dos routers no se hablan

Lectura

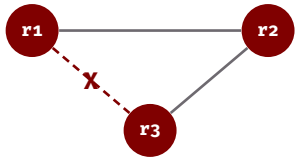
Kurose y Ross, secs. 5.1 y 5.2 (sesión A); 5.3 (sesión B). CS168: *Model for Intra-Domain Routing*, *Distance Vector*, *Link State* y *Model for Inter-Domain Routing*. FRRouting: *OSPFv2*. Complementaria: RFC 2328.

SECCIÓN 01

El routing como un problema de grafos

01

La pregunta de la semana pasada



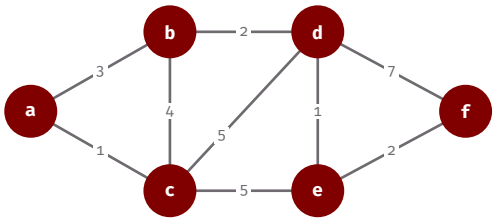
Se cae r1–r3. Con rutas a mano:

- Nadie avisa.
- Los paquetes se pierden hasta que una persona entra a cada router.

Esta semana

Reemplazamos a esa persona por un protocolo. El plano de datos no cambia: sigue siendo longest prefix match. Cambia quién escribe la tabla.

Una red es un grafo



Definición

G = (N, E): routers y enlaces. Cada enlace tiene un **costo** c(x, y) positivo.
El costo de un camino es la suma de sus enlaces.

¿Qué mide el costo? Lo que el administrador decida: retardo, capacidad, dinero, o 1 para contar saltos.

¿Qué es una ruta válida?

Elegimos un destino y seguimos los next hop desde cada router.



Bien: llega al destino.



Callejón sin salida *(dead end)*: se descarta.



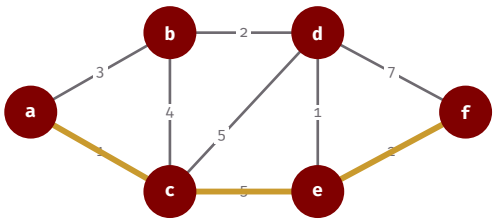
Ciclo *(loop)*: gira hasta agotar el TTL.

Definición

Un estado de routing es **válido** para un destino si, desde cada router, seguir los next hop llega al destino: sin callejones y sin ciclos.

Es una propiedad global: ningún router la puede verificar mirando solo su propia tabla.

Por qué basta con guardar el próximo salto



Todo tramo de un camino óptimo es óptimo

Si el mejor camino de `a` a `f` pasa por `e`, el tramo de `e` a `f` tiene que ser el mejor. Si no, lo usaríamos.

Por eso las rutas hacia un destino forman un **árbol**, y un router solo necesita su next hop.

Dónde se calcula, y con qué información

	Distribuido	**Centralizado**
Quién calcula	Cada router, conversando	Un controlador instala las tablas
Dónde lo vemos	OSPF y BGP	SDN, data centers

	Estado de enlace *(link-state)*	**Vector de distancias** *(distance-vector)*
Qué conoce	Todo el grafo	Sus enlaces y lo que dicen sus vecinos
Qué comunica	Sus enlaces, a todos	Sus distancias, a sus vecinos
Qué calcula	Dijkstra, sobre el mapa	Bellman-Ford, poco a poco
Protocolo	OSPF, IS-IS	RIP

SECCIÓN 02

Estado de enlace

02

Dos pasos

1. Inundar
Cada router le cuenta a toda la red qué enlaces tiene y cuánto cuestan: un **LSA**.

- Cada vecino lo reenvía por todos sus enlaces, menos por el que llegó.
- **Número de secuencia**: solo se reenvía lo nuevo. Si no, daría vueltas para siempre.
- **Confiable**: cada salto confirma, y se retransmite si no.

2. Calcular
Cada router, con el mapa completo, corre Dijkstra **desde sí mismo**.

Resultado: una **LSDB** igual en todos, y una forwarding table distinta en cada uno.

Costo
Inundar todo cuesta del orden de n·e. Aquí, como máximo, 6 × 2 × 9 = 108 copias de LSA al arrancar. Después, solo lo que cambia.

Dijkstra

pseudocódigo

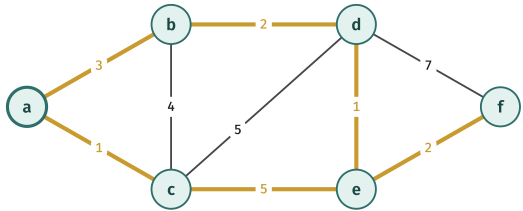
```
N' = {u}
para cada nodo v:  D(v) = c(u, v) si es vecino de u; si no, infinito

repetir hasta que N' tenga todos los nodos:
    w = el nodo fuera de N' con el menor D(w)
    agregar w a N'
    para cada vecino v de w fuera de N':
        D(v) = min( D(v), D(w) + c(w, v) )
```

Por qué funciona

El más cercano de los que faltan ya no puede mejorar: cualquier otro camino saldría de N' por un nodo igual o más lejano, y sumaría costos positivos.

Dijkstra desde a, a mano



paso	N'	b	c	d	e	f
0	a	3,a	1,a	-	-	-
1	ac	3,a	*	6,c	6,c	-
2	acb	*	*	5,b	6,c	-
3	acbd	*	*	*	6,cd	12,d
4	acbde	*	*	*	*	8,e
5	acbdef	*	*	*	*	*

Paso 5. Solo queda f, con 8. Terminamos: el árbol en oro es lo que usa la forwarding table de a.

El mismo cálculo, con `dijkstra.c`

code/week-05

```
$ ./dijkstra graphs/six.txt a
6 nodos, 9 enlaces, origen a

paso N'      b        c        d        e        f
0    a       3,a      1,a      -        -        -
1    ac      3,a      *        6,c      6,c      -
2    acb     *        *        5,b      6,c      -
3    acbd    *        *        *        6,cd     12,d
4    acbde   *        *        *        *        8,e
5    acbdef  *        *        *        *        *

forwarding table de a
destino  costo  next hop
b        3      b
c        1      c
d        5      b
e        6      b c
f        8      b c
```

El next hop de un nodo es el de su predecesor.

`e` tiene dos caminos de costo 6: a-c-e y a-b-d-e. `f` los hereda.

Usar los dos a la vez se llama **ECMP** *(equal-cost multipath)*.

Error común: Dijkstra da la ruta del nodo de origen, no "la de la red". Cada router corre el suyo.

SECCIÓN 03

Vector de distancias

03

Sin mapa: la ecuación de Bellman-Ford

Bellman-Ford

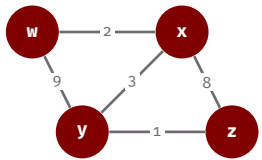
$$d(x, y) = \text{el mínimo, sobre todos los vecinos } v \text{ de } x, \text{ de } c(x, v) + d(v, y)$$

- Para ir de x a y hay que salir por algún vecino. Pago el enlace y lo que le cueste a él.
- Cada router mantiene un **vector de distancias** y se lo envía a sus vecinos.
- Sin coordinación ni orden: si los costos no cambian, llegan a los valores correctos.

Las dos reglas de `dv_sim.c`

1. Lo que diga mi **next hop actual** se acepta siempre, aunque sea peor.
2. Lo que diga **otro vecino** se acepta solo si mejora lo que tengo.

Iteración por iteración



Celda: `costo/next hop`

ronda 2

	w	x	y	z
w	0	2/x	5/x	6/x
x	2/w	0	3/y	4/y
y	5/x	3/x	0	1/z
z	6/y	4/y	1/y	0

La noticia avanza un salto por ronda. Ahora la red **convergió**.

Llega a lo mismo que Dijkstra

dv_sim, sin mapa y sin Dijkstra

```
$ ./dv_sim graphs/six.txt | tail -9
     a      b      c      d      e      f
a    0      3/b    1/c    5/b    6/c    8/c
b    3/a    0      4/c    2/d    3/d    5/d
c    1/a    4/b    0      5/d    5/e    7/e
d    5/b    2/b    5/c    0      1/e    3/e
e    6/c    3/d    5/c    1/d    0      2/f
f    8/e    5/e    7/e    3/e    2/e    0

convergió: 2 rondas, 54 anuncios
```

La fila de `a`: 3, 1, 5, 6, 8, los mismos costos.
En los empates `dv_sim` se queda con un next hop, para mantener el programa simple. Un protocolo real puede guardar los dos.

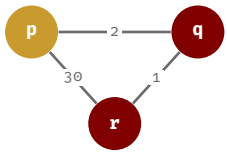
Hasta aquí: con los costos quietos, las dos familias dan la misma respuesta. La diferencia aparece cuando algo cambia.

SECCIÓN 04

Buenas y malas noticias

04

Las buenas noticias viajan rápido



p-q baja de 2 a 1

```
$ ./dv_sim graphs/triangle.txt --dest p --inf 100 --set p q 1
antes          q=2/p    r=3/q
--- el enlace p-q pasa a costar 1
evento         q=1/p    r=3/q
ronda 1        q=1/p    r=2/q
volvió a converger: 1 ronda, 12 anuncios
```

Una mejora se acepta al primer anuncio.

Las malas noticias viajan lento

p-q sube de 2 a 20

```
$ ./dv_sim graphs/triangle.txt --dest p --inf 100 --set p q 20
antes           q=2/p    r=3/q
--- el enlace p-q pasa a costar 20
evento          q=20/p   r=3/q
ronda 1         q=4/r    r=21/q    ciclo: q > r > q
ronda 2         q=20/p   r=5/q
ronda 3         q=6/r    r=21/q    ciclo: q > r > q
ronda 4         q=20/p   r=7/q
ronda 5         q=8/r    r=21/q    ciclo: q > r > q
... (y así, subiendo de a dos)
ronda 14        q=20/p   r=17/q
ronda 15        q=18/r   r=21/q    ciclo: q > r > q
ronda 16        q=20/p   r=19/q
ronda 17        q=20/p   r=21/q
volvió a converger: 17 rondas, 108 anuncios
```

Ronda 1: `q` ve su enlace en 20, y oye a `r` decir "llego con 3". Calcula 4.

`q` no puede saber que el camino de `r` **pasaba por él**.

Un vector trae **costos, no caminos**.

Y mientras tanto hay un **ciclo real**.

El conteo al infinito

```
./dv_sim graphs/line.txt --dest z --fail y z --order x:y,y:x

evento          x=2/y    y=inf
paso 1 x>y      x=2/y    y=3/x     ciclo: x > y > x
paso 2 y>x      x=4/y    y=3/x     ciclo: x > y > x
paso 3 x>y      x=4/y    y=5/x     ciclo: x > y > x
paso 4 y>x      x=6/y    y=5/x     ciclo: x > y > x
paso 5 x>y      x=6/y    y=7/x     ciclo: x > y > x
paso 6 y>x      x=8/y    y=7/x     ciclo: x > y > x
...
paso 13 x>y     x=14/y   y=15/x    ciclo: x > y > x
paso 14 y>x     x=inf    y=15/x
paso 15 x>y     x=inf    y=inf
```

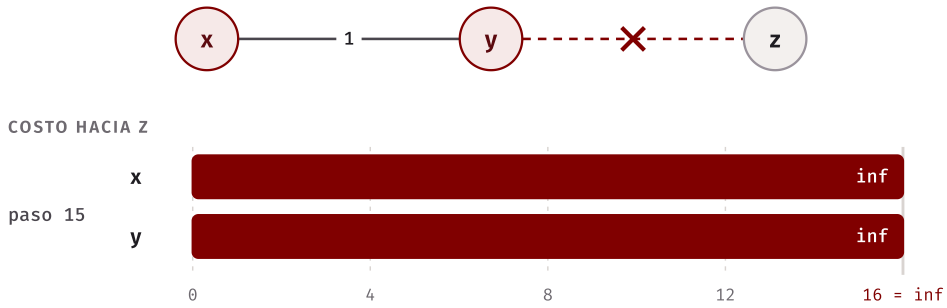
`z` ya no se alcanza, y nadie dará una mejor noticia.

- Paso 1: `x` dice "llego con 2". `y` calcula 3 y apunta a `x`.
- Paso 2: `x` oye subir a su next hop, y sube a 4.
- Y así, como una subasta.

Infinito = 16

Sin tope, no termina nunca. Como RIP, `dv_sim` trata 16 como inalcanzable. RIP solo sirve hasta 15 saltos.

La cuenta, paso a paso



Paso 14. 15 + 1 = 16, y 16 significa inalcanzable: x se rinde. En el paso 15 se rinde y. Convergió después de 15 pasos.

Split horizon y poison reverse

- **Split horizon:** no le anuncio una ruta al vecino por el que la aprendí.
- **Poison reverse:** se la anuncio, pero con costo infinito. "Por mí no vayas".

Split horizon tiene que esperar a que la ruta venza. Poison reverse no, a cambio de mensajes más grandes.

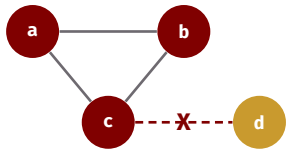
la línea, con `--mode split`: 2 pasos

```
evento          x=2/y    y=inf
paso 1 x>y      x=2/y    y=inf
paso 2 y>x      x=inf    y=inf
paso 3 x>y      x=inf    y=inf
paso 4 y>x      x=inf    y=inf
volvió a converger: 2 pasos, 4 anuncios
```

el triángulo, con `--mode poison`: 1 ronda, en vez de 17

```
evento          q=20/p   r=3/q
ronda 1         q=20/p   r=21/q
volvió a converger: 1 ronda, 12 anuncios
```

Los límites: un triángulo con cola



`a` y `b` no pasan uno por el otro, así que se anuncian su costo de verdad: "llego con 2".

Poison reverse mira dos nodos. El ciclo tiene tres.

```
./dv_sim graphs/tail.txt –dest d –mode poison
–fail c d

evento          a=2/c    b=2/c    c=inf
ronda 1         a=3/b    b=3/a    c=inf     ciclo: a > b > a
ronda 2         a=inf    b=inf    c=4/a
ronda 3         a=inf    b=5/c    c=inf
ronda 4         a=6/b    b=inf    c=inf
ronda 5         a=inf    b=inf    c=7/a
... (y así, subiendo de a dos)
ronda 14        a=inf    b=inf    c=inf
volvió a converger: 14 rondas, 90 anuncios
```

Un ciclo de tres nodos, con poison reverse

```
–mode poison –fail c d –order c:a,b:a,a:c,c:b –max 12, primeros pasos

evento          a=2/c    b=2/c    c=inf
paso 1 c>a      a=inf    b=2/c    c=inf
paso 2 b>a      a=3/b    b=2/c    c=inf
paso 3 a>c      a=3/b    b=2/c    c=4/a     ciclo: a > b > c > a
paso 4 c>b      a=3/b    b=5/c    c=4/a     ciclo: a > b > c > a
paso 5 c>a      a=3/b    b=5/c    c=4/a     ciclo: a > b > c > a
paso 6 b>a      a=6/b    b=5/c    c=4/a     ciclo: a > b > c > a
```

Ningún mensaje viola la regla. Cada uno envenena hacia su next hop, y le cuenta la ruta a quien no lo es. Al recorrer el círculo, la información vuelve al origen.

Lo que faltaría

Para detectar ciclos de cualquier largo, hay que saber **por dónde** pasa la ruta, no solo cuánto cuesta. Es la idea del vector de caminos de BGP, la próxima semana.

SECCIÓN 05

Comparación

05

Estado de enlace y vector de distancias

	Estado de enlace	**Vector de distancias**
Guarda	El grafo entero	Distancia y next hop por destino
Mensajes	Del orden de $n \cdot e$ al arrancar; después, lo que cambia, a toda la red	Vectores a los vecinos; un cambio se propaga o no
Cálculo	Dijkstra: del orden de n^2 en la versión simple	Una comparación por vecino y destino
Convergencia	Inundar y recalcular; ciclos solo mientras las bases difieren	Lenta con malas noticias; puede contar al infinito
Si alguien se equivoca	Daña sobre todo lo suyo	Puede arrastrar a toda la red

¿Y si un router se equivoca?

e anuncia que llega a a con costo 0

```
$ ./dv_sim graphs/six.txt --dest a --lie e a 0
antes          b=3/a    c=1/a    d=5/b    e=6/c    f=8/e
--- e anuncia a con costo 0
evento         b=3/a    c=1/a    d=5/b    e=0      f=8/e
ronda 1        b=3/a    c=1/a    d=1/e    e=0      f=2/e
volvió a converger: 1 ronda, 36 anuncios
```

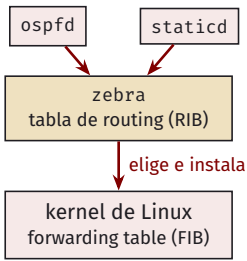
- `d` y `f` mandan el tráfico a `e`, que no lleva a ningún lado. Y lo vuelven a anunciar, mezclado con su costo.
- En estado de enlace, cada LSA lleva la firma de quien lo originó, y todos calculan con datos correctos sobre los demás.
- **No es inmune**: un router también puede anunciar como propio un prefijo ajeno. Pero el error no se transforma al avanzar.

SECCIÓN 06

Dos tablas que no son la misma

06

La tabla de routing y la forwarding table



- La **RIB** guarda todas las candidatas, en el plano de control.
- La **FIB** guarda solo las elegidas, en el plano de datos.
- Se ven con `show ip route` e `ip route`.

Error común

Una ruta sin `*` en `show ip route` no mueve ningún paquete.

El grafo hecho red: `make up-six`

a: vtysh -c 'show ip route ospf'

```
O>* 10.205.0.4/32 [110/5] via 10.205.12.2, eth1, weight 1, 00:00:02
O>* 10.205.0.5/32 [110/6] via 10.205.12.2, eth1, weight 1, 00:00:02
  *                       via 10.205.13.2, eth2, weight 1, 00:00:02
O>* 10.205.0.6/32 [110/8] via 10.205.12.2, eth1, weight 1, 00:00:02
  *                       via 10.205.13.2, eth2, weight 1, 00:00:02
```

a: ip route

```
10.205.0.5 nhid 23 proto ospf metric 20
    nexthop via 10.205.13.2 dev eth2 weight 1
    nexthop via 10.205.12.2 dev eth1 weight 1
```

- **Costos 3, 1, 5, 6, 8**: los de nuestra tabla a mano, con ECMP. `test_six.py` lo compara.
- `>` es seleccionada, `*` está en la FIB.
- El `metric 20` del kernel **no es el costo**: es un valor fijo de `zebra`.

Tres respuestas para la misma pregunta

Agregamos una ruta a mano: `ip route 10.205.0.4/32 10.205.13.2`

a: show ip route 10.205.0.4/32 (la RIB), recortado

```
Known via "static", distance 1, metric 0, best
* 10.205.13.2, via eth2, weight 1

Known via "ospf", distance 110, metric 5
  10.205.12.2, via eth1, weight 1
```

a: show ip ospf route (ospfd)

```
N    10.205.0.4/32          [5] area: 0.0.0.0
                            via 10.205.12.2, eth1
```

a: ip route show 10.205.0.4 (la FIB)

```
10.205.0.4 nhid 17 via 10.205.13.2 dev eth2 proto 196 metric 20
```

- La **distancia administrativa** compara orígenes, no costos: estática 1, OSPF 110.
- Al borrar la estática vuelve OSPF: nunca se fue de la RIB.

SECCIÓN 07

Por qué un solo algoritmo no alcanza

07

Escala y autonomía

Escala
Un mapa de todo el planeta, y cualquier enlace que se caiga obligaría a todos a recalcular.

Autonomía
Cada organización decide cómo funciona su red, y no quiere publicar su topología. Entre redes, las decisiones no son técnicas.

Sistema autónomo (AS)

Routers bajo una misma administración, con una política de routing común y un número.

- **Dentro de un AS**: OSPF, IS-IS. Cada red elige, y manda el desempeño.
- **Entre AS**: BGP, el mismo para todos. Manda la política.

SECCIÓN 08

OSPF

08

De la Sesión A a OSPF

Concepto	En OSPF
Descubrir vecinos	Hello
Sincronizar el mapa con un vecino nuevo	Adyacencia: Database Description, LS Request
Inundar un cambio	LS Update, confirmado con LS Ack
El anuncio	LSA
El mapa	LSDB
Calcular	SPF, que es Dijkstra
Costo	Lo configura el administrador, por interfaz

Dos rasgos

Corre **directo sobre IP**, protocolo 89, con su propia entrega confiable. Y usa multicast a `224.0.0.5` con TTL 1: un paquete OSPF nunca sale de su enlace.

Configurarlo en FRRouting

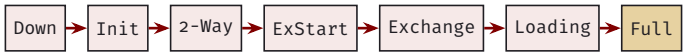
six/c/frr.conf, recortado

```
frr defaults traditional
hostname c
log file /tmp/ospfd.log
!
interface lo
 ip address 10.205.0.3/32
!
interface eth1
 ip address 10.205.13.2/30
 ip ospf network point-to-point
 ip ospf cost 1
!
router ospf
 ospf router-id 10.205.0.3
 log-adjacency-changes detail
 network 10.205.0.0/16 area 0
```

- `ip ospf cost`: el costo de **salir** por esa interfaz. Puede ser distinto en cada dirección.
- `router-id`: el nombre del router en OSPF.
- `network`: **selecciona interfaces**, no anuncia ese prefijo.
- `log-adjacency-changes`: registra cada cambio de estado.

`show ip ospf neighbor`	Vecinos y estado
`show ip ospf interface`	Área, costo, temporizadores
`show ip ospf database`	El mapa
`show ip route ospf`	Lo que quedó en la RIB

Estados de una adyacencia



`Init`	Oí su Hello, pero él no me menciona.
`2-Way`	Me menciona: la comunicación funciona en las dos direcciones.
`ExStart`	Acordamos quién dirige el intercambio.
`Exchange`	Nos mandamos los índices de nuestras bases de datos.
`Loading`	Pido los LSA que me faltan.
`Full`	Tenemos la misma base. Ahora sí podemos usar el enlace.

En un enlace point-to-point no se ve `2-Way`: siempre se forma adyacencia, y se pasa de frente a `ExStart`.

Lo vemos en vivo

a: grep AdjChg /tmp/ospfd.log, acortado

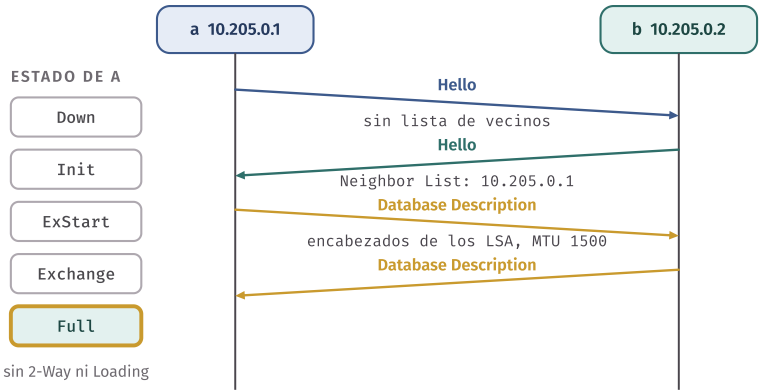
```
08:03:39 OSPF: AdjChg: Nbr 10.205.0.2 ... on eth1:10.205.12.1: Down -> Init (HelloReceived)
08:03:39 OSPF: AdjChg: Nbr 10.205.0.2 ... on eth1:10.205.12.1: Init -> ExStart (2-WayReceived)
08:03:39 OSPF: AdjChg: Nbr 10.205.0.2 ... on eth1:10.205.12.1: Exchange -> Full (ExchangeDone)
```

a: tcpdump -n -v -i eth1 ip proto 89

```
10.205.12.2 > 224.0.0.5: OSPFv2, Hello, length 48
Router-ID 10.205.0.2, Backbone Area, Authentication Type: none (0)
Options [External]
  Hello Timer 10s, Dead Timer 40s, Mask 255.255.255.252, Priority 1
  Neighbor List:
    10.205.0.1
```

- La **lista de vecinos** prueba las dos direcciones. TTL 1, protocolo 89, multicast.
- Viajan área, temporizadores y máscara. Hello cada **10 s**; sin oírlo por el Dead, **40 s**, el vecino se da por muerto.

La adyacencia, paquete por paquete



a acababa de bajar el enlace y conservaba el mapa: no tenía nada que pedir, por eso no pasa por Loading. Con la misma base de datos, queda en Full.

Point-to-point y broadcast

En una red **broadcast** se elige un router designado *(DR)* y uno de respaldo *(BDR)*. Todos forman adyacencia completa solo con ellos, y el DR origina un **Network LSA**.

broadcast: el vecino es el DR del enlace

```
Neighbor ID     Pri State           Up Time         Dead Time Address         Interface
10.205.0.2        1 Full/DR         0.430s            39.575s 10.205.12.2     eth1:10.205.12.1
```

point-to-point: sin DR

```
Neighbor ID     Pri State           Up Time         Dead Time Address         Interface
10.205.0.2        1 Full/-          5.909s            39.469s 10.205.12.2     eth1:10.205.12.1
```

- OSPF trata a Ethernet como broadcast, aunque solo haya dos routers. Espera el **Wait** (40 s) antes de elegir.
- En los `/30` entre routers usamos `ip ospf network point-to-point`: sin elección, y cada LSA lista directo a sus vecinos.

El mismo mapa en dos routers

a: show ip ospf database, recortado

Link ID	ADV Router	Age	Seq#	CkSum	Links
10.205.0.1	10.205.0.1	6	0x80000005	0x64d9	5
10.205.0.2	10.205.0.2	7	0x80000007	0x0d36	7
10.205.0.3	10.205.0.3	13	0x80000009	0x9b91	9
10.205.0.4	10.205.0.4	7	0x80000009	0xbc11	9
10.205.0.5	10.205.0.5	8	0x80000007	0xeeac	7
10.205.0.6	10.205.0.6	8	0x80000005	0x5034	5

d: show ip ospf database, recortado

Link ID	ADV Router	Age	Seq#	CkSum	Links
10.205.0.1	10.205.0.1	8	0x80000005	0x64d9	5
10.205.0.2	10.205.0.2	6	0x80000007	0x0d36	7
10.205.0.3	10.205.0.3	11	0x80000009	0x9b91	9
10.205.0.4	10.205.0.4	6	0x80000009	0xbc11	9
10.205.0.5	10.205.0.5	6	0x80000007	0xeeac	7
10.205.0.6	10.205.0.6	6	0x80000005	0x5034	5

- **Secuencia y checksum coinciden línea por línea.** Solo cambia la edad.
- Hay un LSA por router, originado por él mismo.
- Un router renueva los suyos cada 30 minutos, y un LSA que llega a la hora sin renovarse se elimina.
- Si dos routers de un área tienen bases distintas después de un rato, algo está mal.

Un Router LSA es el grafo

a: show ip ospf database router

10.205.0.2, recortado

```
  LS Type: router-LSA
  Advertising Router: 10.205.0.2
  LS Seq Number: 80000007

   Number of Links: 7

    Link connected to: Stub Network
     (Link ID) Net: 10.205.0.2
       TOS 0 Metric: 0

    Link connected to: another Router (point-to-point)
     (Link ID) Neighboring Router ID: 10.205.0.1
       TOS 0 Metric: 3

    Link connected to: Stub Network
     (Link ID) Net: 10.205.12.0
       TOS 0 Metric: 3
```

- `b` dice: tengo un vecino `a` a costo 3. Y lo mismo para los otros dos.
- Son 7 líneas: 1 de su loopback y 2 por cada uno de sus 3 enlaces.
- Una **red stub** es una hoja del grafo: sirve para llegar a sus direcciones, y por ahí no se pasa.
- Un enlace se usa solo si **ambos** lo anuncian.

Inundar un cambio

Se apaga b-d. Lo que ve `f`, que no toca ese enlace:

`f: tcpdump -n -i eth2, sin los Hello: primeros paquetes`

```
08:08:33.128338 IP 10.205.56.1 > 224.0.0.5: OSPFv2, LS-Update, length 112
08:08:34.129375 IP 10.205.56.2 > 224.0.0.5: OSPFv2, LS-Ack, length 44
```

`a: tcpdump -n -v -i eth2: un LS-Update que a reenvía`

```
10.205.13.1 > 224.0.0.5: OSPFv2, LS-Update, length 112
  Advertising Router 10.205.0.2, seq 0x80000020, age 2s, length 64
      Neighbor Router-ID: 10.205.0.3, Interface Address: 10.205.23.1
      Neighbor Router-ID: 10.205.0.1, Interface Address: 10.205.12.2
```

- Lo reenvía `a`, y lo originó `b`: secuencia mayor, y ya no menciona a `d`.
- Nadie les dijo qué ruta usar: cada uno corre SPF. A los 5 s sale otra tanda, con secuencia mayor: no es retransmisión.

Costos: mejor explícitos

a: sin ip ospf cost, en una veth de 10000 Mbit

```
ifindex 8779, MTU 1500 bytes, BW 10000 Mbit <UP,LOWER_UP,BROADCAST,RUNNING,MULTICAST>
Router ID 10.205.0.1, Network Type POINTOPOINT, Cost: 10
```

c: con bandwidth 1000, y luego con auto-cost reference-bandwidth 1000

```
ifindex 8192, MTU 1500 bytes, BW 1000 Mbit <UP,LOWER_UP,BROADCAST,RUNNING,MULTICAST>
Router ID 10.205.0.3, Network Type POINTOPOINT, Cost: 100
ifindex 8192, MTU 1500 bytes, BW 1000 Mbit <UP,LOWER_UP,BROADCAST,RUNNING,MULTICAST>
Router ID 10.205.0.3, Network Type POINTOPOINT, Cost: 1
```

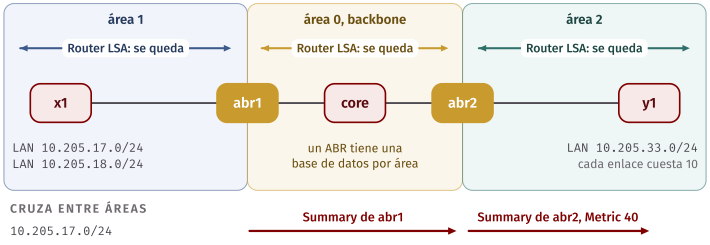
- OSPF divide una referencia entre el ancho de banda. De lo medido sale 100 000 / Mbit/s.
- El manual de FRRouting habla de una referencia de 100 Mbit/s: **no lo den por hecho**.
- En los laboratorios ponemos `ip ospf cost` en cada enlace: una decisión escrita.

SECCIÓN 09

Áreas y backbone

09

Áreas



- Toda área se conecta al área 0. Un **ABR** tiene una base de datos por área.
- Entre áreas, OSPF funciona como vector de distancias, con el backbone en el centro para evitar ciclos.

Lo que ve `y1`: `make up-areas`

```
y1: show ip ospf database

                Router Link States (Area 0.0.0.2)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.205.0.14     10.205.0.14      12 0x80000003 0x612c 2
10.205.0.15     10.205.0.15      12 0x80000003 0xc1b7 3

                Summary Link States (Area 0.0.0.2)

Link ID         ADV Router      Age  Seq#       CkSum  Route
10.205.1.0      10.205.0.14        7 0x80000001 0xd1bb 10.205.1.0/30
10.205.2.0      10.205.0.14       22 0x80000001 0x6234 10.205.2.0/30
10.205.16.0     10.205.0.14        7 0x80000001 0x90e3 10.205.16.0/30
10.205.17.0     10.205.0.14        7 0x80000001 0xfb6a 10.205.17.0/24
10.205.18.0     10.205.0.14        7 0x80000001 0xf074 10.205.18.0/24
```

- `y1` conoce a dos routers: él mismo y `abr2`. De `x1` no tiene ningún Router LSA.
- Se apaga una LAN de `x1`: a `y1` solo le llega que un Summary dejó de valer. Su contador de SPF igual sube, de 5 a 7: evitamos la inundación del detalle, no todo cálculo.

Resumir, y el precio: `make up-range`

areas/abr1/frr-range.conf, recortado

```
1  router ospf
2   ospf router-id 10.205.0.12
3   network 10.205.1.0/30 area 0
4   network 10.205.16.0/20 area 1
5   area 1 range 10.205.16.0/20
```

y1: un solo prefijo por el área 1

```
Link ID      ADV Router    Age Seq#       Route
10.205.1.0   10.205.0.14     7 0x80000001 10.205.1.0/30
10.205.2.0   10.205.0.14    21 0x80000001 10.205.2.0/30
10.205.16.0  10.205.0.14     7 0x80000001 10.205.16.0/20
```

Se apaga la misma LAN. El contador de SPF de `y1` se queda en 5, y su base no cambia.

y1, con 10.205.17.0/24 apagada

```
$ ping -c 1 -W 1 10.205.17.1
1 packets transmitted, 0 packets received, 100% packet loss
```

En FRRouting 10.3.1, `area range` escrito con `vtysh` en vivo tumbó a `ospfd` (`signal 11`). En `frr.conf`, funciona.

SECCIÓN 10

Convergencia

10

Las cinco partes de converger

1. **Detectar** la falla
2. Originar un LSA nuevo
3. Inundarlo por el área
4. Calcular SPF en cada router
5. Instalar en la FIB

Calcular y avisar, en microsegundos y milisegundos. ¿Y detectar? Depende de **cómo muere** el enlace.

a: show ip ospf, recortado

```
Initial SPF scheduling delay 0 millisec(s)
Minimum hold time between consecutive SPFs 50 millisec(s)
Maximum hold time between consecutive SPFs 5000 millisec(s)
Hold time multiplier is currently 2
SPF algorithm last executed 1.441s ago
Last SPF duration 67 usecs
SPF timer is inactive
LSA minimum interval 5000 msecs
LSA minimum arrival 1000 msecs
```

Los `hold time` y el `LSA minimum interval` son frenos contra la inestabilidad.

Cuando el enlace avisa

Si se corta un cable, la interfaz pierde **carrier**, y `ospfd` se entera al instante. Ping de `a` a `d`, a 5 paquetes por segundo, y a los 2 s apagamos b-d:

```
ip link set eth3 down en b
```

```
$ ip link show eth1      # en d, el otro extremo
8765: eth1@if8764: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
$ ping -c 40 -i 0.2 -W 1 -q -I 10.205.0.1 10.205.0.4
40 packets transmitted, 40 packets received, 0% packet loss
$ traceroute -n -q1 -s 10.205.0.1 10.205.0.4
 1  10.205.13.2  0.005 ms
 2  10.205.0.4  0.001 ms
```

- **Ningún paquete perdido**, las tres veces, y el tráfico cambió de camino.
- Los temporizadores no tuvieron nada que ver. Apagar una interfaz es **el caso más favorable**: da una idea equivocada de la convergencia.

Cuando el enlace no avisa

Algo falla en medio: un switch, un proveedor. Lo imitamos con `tc qdisc add dev eth3 root netem loss 100%` en los dos extremos.

justo después

```
$ ip link show eth3    # en b
8764: eth3@if8765: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc netem state UP ...
$ ping -c 3 -W 1 -I 10.205.0.1 10.205.0.4    # en a
3 packets transmitted, 0 packets received, 100% packet loss
$ vtysh -c 'show ip ospf neighbor' | grep 10.205.0.4   # en b
10.205.0.4        1 Full/-          5.931s           34.068s 10.205.24.2     eth3:10.205.24.1
```

El peor estado que existe

El plano de control dice que todo va bien, y el de datos lo pierde todo. Dura hasta que se agota el Dead: `Full -> Deleted (InactivityTimer)`.

Lo medimos

Falla	Hello/Dead	Cambio de ruta en a	Perdidos	Sin servicio
Interfaz abajo	10 / 40	Inmediato	0, 0, 0 de 40	0 s
Silenciosa	10 / 40	38,2; 35,7; 35,0 s	190, 177, 174 de 275	35 a 38 s
Silenciosa	2 / 8	6,8; 7,7; 7,3 s	34, 38, 36 de 75	7 a 8 s

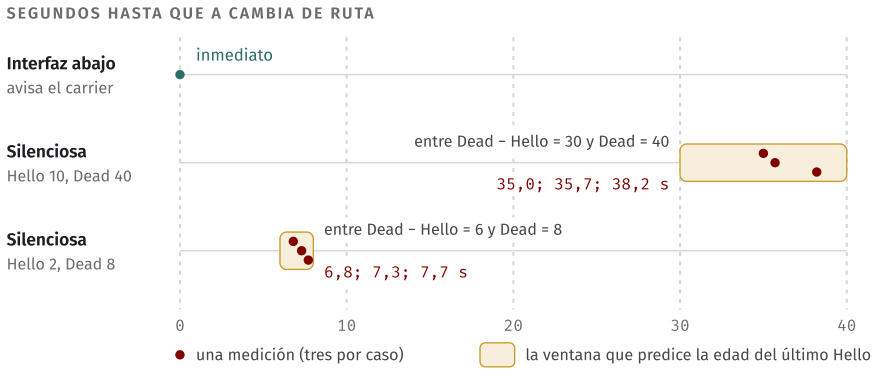
Ping a 5 por segundo: los segundos son paquetes perdidos × 0,2. Las cifras dependen de la máquina; el orden, no.

El último Hello tiene una edad entre 0 y un Hello. Por eso se detecta entre **Dead** − **Hello** y **Dead**.

La idea de la semana

Detectar la falla suele ser lo más lento. Dijkstra toma microsegundos e inundar, milisegundos. Esperar el Dead toma decenas de segundos. No hace falta un mejor algoritmo, sino enterarse antes.

Por qué es casi el Dead



Las mediciones caen dentro de la ventana que predice la edad del último Hello.

Lo que arriesgan los temporizadores cortos

Un corte de solo 12 s: con 10 / 40, la adyacencia no se entera (0 cambios en el log). Con 2 / 8:

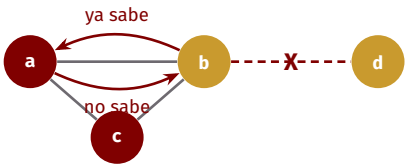
b: /tmp/ospfd.log

```
08:09:42 OSPF: AdjChg: Nbr 10.205.0.4 ... on eth3:10.205.24.1: Full -> Deleted (InactivityTimer)
08:09:50 OSPF: AdjChg: Nbr 10.205.0.4 ... on eth3:10.205.24.1: Exchange -> Full (ExchangeDone)
```

	Largos	**Cortos**
Falla silenciosa	Mucho tiempo sin servicio	Poco tiempo sin servicio
Parpadeo	Pasa inadvertido	Mueve toda la red
Tráfico de control	Casi nada	Más Hello, rara vez el problema
Un router ocupado	Aguanta	Puede perder vecinos

Un **flap** puede tener a toda el área recalculando. Para detectar en milisegundos existe BFD.

Mientras tanto: ciclos pasajeros



Tres routers ya actualizaron su tabla, y dos todavía no.

- **Ciclo pasajero**: uno actualizado reenvía a uno atrasado, que le devuelve el paquete.
- **Callejón pasajero**: uno atrasado envía por el enlace muerto.

Ruta válida, otra vez

Cada tabla es correcta con lo que sabe. Es el conjunto el que no es válido. Con estado de enlace termina cuando la inundación llega a todos; con vector de distancias, puede durar lo que tarde el conteo.

SECCIÓN 11

Cuando dos routers no se hablan

11

Si no coincide, el Hello se descarta en silencio

`eth1` de `a` en el área 1, y `eth1` de `b` en el área 0:

```
show ip ospf interface eth1 | grep -E 'Internet|Neighbor Count'

$ en a
  Internet Address 10.205.12.1/30, Broadcast 10.205.12.3, Area 0.0.0.1
  Neighbor Count is 0, Adjacent neighbor count is 0
$ en b
  Internet Address 10.205.12.2/30, Broadcast 10.205.12.3, Area 0.0.0.0
  Neighbor Count is 1, Adjacent neighbor count is 1
```

```
a: /tmp/ospfd.log

08:10:41 OSPF: [Y9FH4-5CVMQ][EC 134217741] interface eth1: ospf_read invalid Area ID 0.0.0.0
```

- `show ip ospf interface`, en **los dos** routers. El log, si está activado, dice la causa.
- En `b`, el vecino sigue ahí hasta que le venza el Dead: los dos lados no se enteran al mismo tiempo.

Temporizadores: el efecto llega después

a: ip ospf dead-interval 20, y de inmediato

```
$ vtysh -c 'show ip ospf neighbor' | grep 10.205.0.2
10.205.0.2        1 Full/-          2.791s           37.207s 10.205.12.2     eth1:10.205.12.1
```

Sigue en `Full`. Se corta cuando pasan 20 s sin un Hello aceptable.

a: 20 s después

```
  Timer intervals configured, Hello 10s, Dead 20s, Wait 20s, Retransmit 5
  Neighbor Count is 0, Adjacent neighbor count is 0
Packet 10.205.0.2 [Hello:RECV]: RouterDeadInterval mismatch on eth1:10.205.12.1 (expected 20, but received 40).
```

Error común: mirar de inmediato, ver `Full` y concluir que no pasa nada.

Máscara: tiene letra chica

`/29` en `a` y `/30` en `b`, en un enlace point-to-point:

a: forma adyacencia igual

```
10.205.0.2        1 Full/-          4.881s           35.118s 10.205.12.2     eth1:10.205.12.1
  Router ID 10.205.0.1, Network Type POINTOPOINT, Cost: 3
```

RFC 2328, sec. 10.5: en redes point-to-point y enlaces virtuales, la máscara del Hello se ignora. Mismo cambio, sin `point-to-point` en los dos extremos:

a: broadcast, el vecino no aparece

```
  Router ID 10.205.0.1, Network Type BROADCAST, Cost: 3
  Neighbor Count is 0, Adjacent neighbor count is 0
Packet ... NetworkMask mismatch on eth1:10.205.12.1 (configured prefix length is 29, but hello packet indicates 30).
```

MTU, y la tabla completa

a: MTU 1400 aquí y 1500 en b. Detenido para siempre en ExStart

```
10.205.0.2        1 ExStart/-       25.069s           34.934s 10.205.12.2     eth1:10.205.12.1
Packet[DD]: Neighbor 10.205.0.2 MTU 1500 is larger than [eth1:10.205.12.1]'s MTU 1400
```

Síntoma	Causa probable	Dónde se ve
El vecino no aparece	Área distinta	`Area`; log `invalid Area ID`
El vecino no aparece	Hello o Dead distintos	`Timer intervals`; log `mismatch`
No aparece, en broadcast	Máscara distinta	`Internet Address`; log `NetworkMask`
El vecino no aparece	`passive-interface`, o sin `network`	`show ip ospf interface`
Detenido en `ExStart`	MTU distinto	Log `MTU ... is larger`
`Full`, pero faltan rutas	El otro extremo no anuncia	`show ip ospf database`

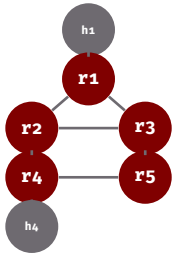
SECCIÓN 12

Laboratorio

12

OSPF con FRRouting: una falla en vivo

1. OSPF en `r1` y `r2`. Seguir los estados hasta `Full`.
2. Dijkstra a mano desde `r1`: `lab/r1_table.txt`. Terminar: `lab/ospf.sh`.
3. Mover el tráfico por `r3` y `r5`, en las dos direcciones: `lab/cost.sh`.
4. Ping en vivo: apagar un enlace y, luego, una falla sin aviso con `netem`.
5. Hello 2 / Dead 8: `lab/timers.sh`. ¿Qué se gana y qué se arriesga?
6. Provocar una adyacencia que no sube, y diagnosticarla.
7. Avanzar con **P1**.



code/week-05

```
$ make up-ospf
$ make test
$ make down
```

Para la próxima clase

- Terminar el laboratorio, y avanzar con el OSPF de P1.
- Leer Kurose y Ross, secs. 5.4.1, 5.4.2 y 5.4.3, y CS168: *BGP* y *BGP Implementation and Issues*.

Pregunta

Hoy cada router buscó el camino de menor costo. Si dos redes compiten y una te paga por llevar su tráfico, ¿el camino de menor costo sigue siendo el que quieres? ¿Qué tendría que anunciar un router para que nadie forme ciclos, sin publicar su mapa?