

Computación Centrada en Redes

BGP y políticas entre redes

Routing

Gerson Garrido

Semana 6

UNIVERSIDAD NACIONAL DE INGENIERÍA

Hoy

Sesión A: el protocolo

1. Sistemas autónomos
2. Vector de caminos
3. Sesiones y mensajes
4. Leer una tabla BGP
5. eBGP, iBGP y OSPF
6. Elegir la mejor ruta

Sesión B: la política

7. Quién le paga a quién
8. Importación y exportación
9. Caminos valle-libre
10. `route-map` en FRRouting
11. Anycast
12. Presencia en Internet

Lectura

Kurose y Ross, secs. 5.4.1, 5.4.2 y 5.4.3 (sesión A); 5.4.4, 5.4.5 y 5.4.6 (sesión B). CS168: *BGP* y *BGP Implementation and Issues*. FRRouting: *BGP*. Complementaria: RFC 4271.

SECCIÓN 01

Sistemas autónomos

01

Dos problemas distintos

	Dentro de un AS	Entre AS
Protocolo	OSPF, IS-IS	BGP
Quién manda	Una administración	Cada AS, solo en el suyo
Qué se busca	El menor costo	Lo que convenga según la política
Qué se comparte	El mapa entero	Qué prefijos se alcanzan, y por dónde
Tamaño	Cientos o miles de routers	Toda Internet

Sistema autónomo ***(autonomous system)***

Conjunto de routers bajo una misma administración, que frente al resto del mundo se presenta con una sola política de routing. Lo identifica un número de AS. En este curso, privados: `64512`–`65534`.

¿Por qué no OSPF para todo?

- **Escala.** OSPF inunda cada cambio a todos los routers. Con toda Internet, sería imposible.
- **Autonomía.** Un costo solo tiene sentido en la red que lo definió, y nadie quiere publicar su mapa.
- **Política.** Un AS puede tener un camino corto y no querer usarlo, o no querer ofrecerlo.

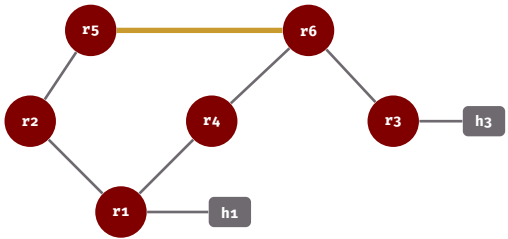
> **Idea clave**
>
> Dentro de un AS manda la eficiencia. Entre sistemas autónomos manda la política. BGP no resuelve un problema de optimización: deja que cada AS aplique sus reglas, y aun así todo funciona.

SECCIÓN 02

Vector de caminos

02

Los seis AS de esta semana

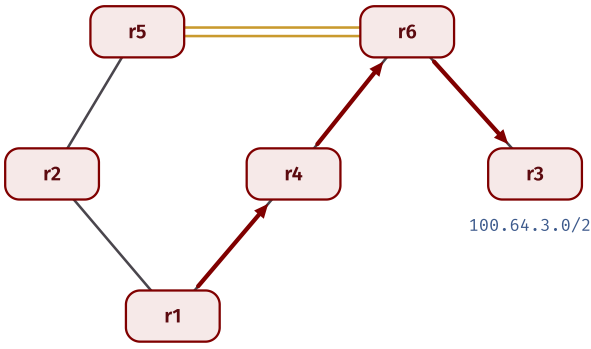


- `rN` es el AS `6460N`.
- Cada uno es owner de `100.64.N.0/24`.
- Hosts en la parte baja, enlaces en la alta.
- Los enlaces se numeran con el bloque del proveedor: `traceroute` los puede mostrar.

code/week-06

```
$ make up-ases
```

Un anuncio, salto a salto



AS_PATH QUE RECIBE CADA AS

r6	64603
r4	64606 64603
r5	64606 64603
r2	64605 64606 64603
r1	64604 64606 64603
r1	64602 64605 64606 64603

traceroute: h1 → r1 → r4 → r6 → r3 → h3

De vuelta en 100.64.3.0/24: el anuncio subió por un lado, y el tráfico va en sentido contrario, r1 → r4 → r6 → r3. Una ruta anunciada es una promesa.

Path vector

Reglas de AS_PATH

1. El owner anuncia el prefijo con AS_PATH igual a su número.
2. Quien reenvía agrega su número al principio.
3. Quien recibe una ruta con su propio número la **descarta**.

- La regla 3 elimina los ciclos, sin mapa global. AS_PATH no dice nada de retardo.
- Se anuncia solo el camino elegido: un AS sabe lo que sus vecinos quieren contarle.

Error común

BGP no “calcula” como OSPF. Cada AS elige entre lo que le anunciaron, y nadie ve el camino completo.

Configurar un AS

ases/r2/frr.conf

```
hostname r2
interface lo
 ip address 100.64.2.1/32
interface eth1
 ip address 100.64.2.129/30
interface eth2
 ip address 100.64.5.134/30
ip route 100.64.2.0/24 Null0
router bgp 64602
 bgp router-id 100.64.2.1
 no bgp ebgp-requires-policy
 neighbor 100.64.2.130 remote-as 64601
 neighbor 100.64.5.133 remote-as 64605
 address-family ipv4 unicast
  network 100.64.2.0/24
```

- Los vecinos no se descubren: cada sesión es un contrato.
- `network` anuncia solo lo que **ya está** en la tabla.
- `Null0` sostiene el agregado. Lo más específico le gana.
- `ebgp-requires-policy` (RFC 8212): sin política, nada entra ni sale.

SECCIÓN 03

Sesiones y mensajes

03

Una sesión sobre TCP

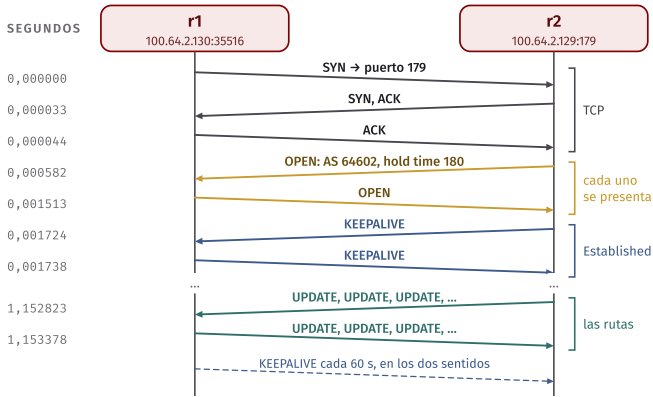
Conexión **TCP al puerto 179**. Sin refrescos: una ruta vale hasta que se retira o cae la sesión.

Tipo	Mensaje	Para qué
1	OPEN	Versión, AS, hold time, identificador y capacidades
2	UPDATE	Rutas nuevas con atributos, y rutas retiradas
3	NOTIFICATION	Un error o un cierre: la sesión termina
4	KEEPALIVE	“Sigo aquí”. Solo el header de 19 bytes

Hold time

Sin nada del vecino durante ese tiempo, la sesión se da por muerta y sus rutas se retiran. KEEPALIVE por lo general va cada tercio. Aquí: 180 y 60 segundos.

Cómo se abre



Estados: `Idle`, `Connect`, `Active`, `OpenSent`, `OpenConfirm`, `Established`. `Active` es mala señal: sigue intentando.

Seguir una sesión

show bgp summary, en r2

```
BGP router identifier 100.64.2.1, local AS number 64602 VRF default vrf-id 0

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
100.64.2.130    4      64601         9         9        6    0    0 00:00:10            4        6 N/A
100.64.5.133    4      64605         9        10        6    0    0 00:00:10            4        6 N/A
```

la misma tabla en r1, con una sesión apagada

```
Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
100.64.2.129    4      64602         9        11        0    0    0 00:00:02 Idle (Admin)        0 N/A
100.64.4.129    4      64604        11        13        9    0    0 00:00:12            5        6 N/A
```

Si `State/PfxRcd` muestra un número, está en `Established` y ese número son los prefijos recibidos.

Capturar el inicio

se apaga la sesión y se captura en r1

```
$ vtysh ... -c 'neighbor 100.64.2.129 shutdown'
$ tcpdump -U -i eth2 -w /tmp/s.pcap tcp port 179 &
$ vtysh ... -c 'no neighbor 100.64.2.129 shutdown'
```

tshark -Y bgp: trama, origen, TTL, tipo, largo

```
4   100.64.2.129  1  1              99
6   100.64.2.130  1  1              99
8   100.64.2.129  1  4              19
9   100.64.2.130  1  4              19
12  100.64.2.129  1  2,2,2,2,2,2,2  64,55,60,60,52,56,23
14  100.64.2.130  1  2,2,2,2,2,2,2  55,64,60,52,60,56,23
16  100.64.2.130  1  2,2            52,56
18  100.64.2.129  1  2,2            52,56
```

- OPEN en cada sentido, luego KEEPALIVE, y recién entonces las rutas.
- KEEPALIVE: 19 bytes, solo el header.
- **TTL 1**: una sesión eBGP directa no debe cruzar ningún router.

OPEN y UPDATE

tshark -O bgp: OPEN de r2

```
Border Gateway Protocol - OPEN Message
    Length: 99
    Type: OPEN Message (1)
    Version: 4
    My AS: 64602
    Hold Time: 180
    BGP Identifier: 100.64.2.1
    Optional Parameters Length: 70
```

Si `My AS` no coincide con el `remote-as` del otro, la sesión no sube.

tshark -O bgp: UPDATE de r1 (recortado)

```
Border Gateway Protocol - UPDATE Message
    Length: 55
    Withdrawn Routes Length: 0
    Total Path Attribute Length: 28
    Path attributes
        Path Attribute - ORIGIN: IGP
        Path Attribute - AS_PATH: 64601
        Path Attribute - NEXT_HOP: 100.64.2.130
        Path Attribute - MULTI_EXIT_DISC: 0
    Network Layer Reachability Information (NLRI)
        100.64.1.0/24
```

Una ruta hacia mí mismo

tcpdump -vv: un UPDATE que r2 le manda a r1

```
Update Message (2), length: 64
  Origin (1), length: 1, Flags [T]: IGP
  AS Path (2), length: 22, Flags [TE]: 64602 64605 64606 64604 64601
  Next Hop (3), length: 4, Flags [T]: 100.64.2.129
  Updated routes:
    100.64.1.0/24
```

La regla 3, en una captura real

`r2` le ofrece a `r1` su propio prefijo, por el camino largo. `r1` ve `64601` y lo descarta. En `show bgp summary`, `r1` registra solo 4 de los 6 prefijos que `r2` le manda. El otro descartado es `100.64.4.0/24`, al que `r2` llega a través de `r1`.

SECCIÓN 04

Leer una tabla BGP

04



La tabla

```
show bgp ipv4, en r1

     Network          Next Hop            Metric LocPrf Weight Path
*>   100.64.1.0/24    0.0.0.0                  0         32768 i
*>   100.64.2.0/24    100.64.2.129             0             0 64602 i
*>   100.64.3.0/24    100.64.4.129                           0 64604 64606 64603 i
*                     100.64.2.129                           0 64602 64605 64606 64603 i
*>   100.64.4.0/24    100.64.4.129             0             0 64604 i
*>   100.64.5.0/24    100.64.2.129                           0 64602 64605 i
*                     100.64.4.129                           0 64604 64606 64605 i
*>   100.64.6.0/24    100.64.4.129                           0 64604 64606 i
*                     100.64.2.129                           0 64602 64605 64606 i
```

- `*` válido, `>` elegido. Sin prefijo: otro camino al mismo destino.
- `Path`: el primero es el vecino, el último el origen. `r1` no se incluye a sí mismo.
- `Metric` es el MED: solo aparece en lo que originó el vecino directo. No cruza al siguiente AS.
- `Weight` es local al router. No es un atributo de BGP.

Por qué ganó

show bgp ipv4 100.64.3.0/24, en r1

```
Paths: (2 available, best #1, table default)
  64604 64606 64603
    100.64.4.129 from 100.64.4.129 (100.64.4.1)
      Origin IGP, valid, external, best (AS Path)
  64602 64605 64606 64603
    100.64.2.129 from 100.64.2.129 (100.64.2.1)
      Origin IGP, valid, external
```

traceroute a h3, en h1

```
1  100.64.1.1  0.006 ms
2  100.64.4.129  0.003 ms
3  100.64.6.129  0.003 ms
4  100.64.6.134  0.003 ms
5  100.64.3.10  0.002 ms
```

Lean esa palabra

`best (...)` es la razón por la que ganó ese camino. Los saltos del `traceroute` corresponden al `AS_PATH` elegido. No lean los tiempos: los enlaces son virtuales.

Atributos

Atributo	Qué dice	Alcance
`ORIGIN`	De dónde salió: `i`, `e` o `?`	Viaja por todas partes
`AS_PATH`	Los AS que cruzó	Cada AS lo alarga al anunciar a otro AS
`NEXT_HOP`	A qué dirección mandar los paquetes	Cambia en eBGP. En iBGP no, sin que se pida
`LOCAL_PREF`	Cuánto le gusta a este AS	Solo iBGP: nunca sale del AS
`MED`	Por cuál enlace prefiero que me entren	Lo pone el vecino. No pasa al siguiente AS
Comunidad	Una etiqueta de 32 bits	Viaja mientras nadie la quite

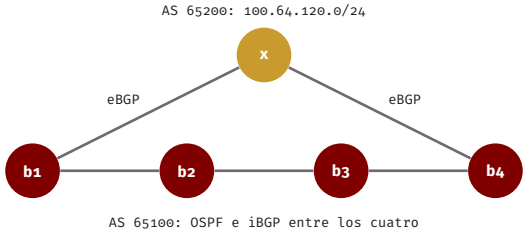
SECCIÓN 05

eBGP, iBGP y OSPF

05



BGP dentro de un AS



OSPF

Cómo llegar a los **demás routers** del AS.

iBGP

Qué **prefijos externos** se alcanzan, y por cuál borde.

Dos búsquedas: BGP dice “está detrás de X”; OSPF dice “hacia X, por esta interfaz”.

El problema del NEXT_HOP

```
show bgp ipv4 100.64.120.0/24, en b2

Paths: (2 available, no best path)
  65200
    10.207.4.2 (inaccessible, import-check enabled) from 100.64.100.4 (100.64.100.4)
      Origin IGP, metric 0, localpref 100, invalid, internal
  65200
    10.207.1.2 (inaccessible, import-check enabled) from 100.64.100.1 (100.64.100.1)
      Origin IGP, metric 0, localpref 100, invalid, internal
```

Por iBGP, `NEXT_HOP` no cambia: sigue siendo la dirección de `x`. Ese enlace no está en OSPF, así que `b2` no sabe llegar.

ibgp/nhs_b1.conf

```
1  router bgp 65100
2   address-family ipv4 unicast
3    neighbor 100.64.100.2 next-hop-self
4    neighbor 100.64.100.3 next-hop-self
5    neighbor 100.64.100.4 next-hop-self
```

Con `next-hop-self`

```
show bgp ipv4 100.64.120.0/24, en b2

Paths: (2 available, best #1, table default)
  65200
    100.64.100.1 (metric 10) from 100.64.100.1 (100.64.100.1)
      Origin IGP, metric 0, localpref 100, valid, internal, best (IGP Metric)
  65200
    100.64.100.4 (metric 20) from 100.64.100.4 (100.64.100.4)
      Origin IGP, metric 0, localpref 100, valid, internal
```

```
show ip ospf route, en b2: de 100.64.120.0/24 no sabe nada

N    100.64.100.1/32       [10] area: 0.0.0.0
                           via 10.207.12.1, eth1
N    100.64.100.4/32       [20] area: 0.0.0.0
                           via 10.207.23.2, eth2
```

`(metric 10)` es la distancia OSPF al next hop. Los dos protocolos, en una sola línea.

iBGP no reenvía a iBGP

- Dentro del AS, `AS_PATH` no detecta ciclos.
- Lo aprendido por iBGP **no** se pasa a otro vecino iBGP.
- Hace falta **malla completa**: `n(n-1)/2` sesiones.
- De ahí los route reflectors (RFC 4456).

b3, sin sesiones con b1 ni con b4 (recortado)

```
Neighbor        V         AS  Up/Down State/PfxRcd
100.64.100.1    4      65100 00:00:03 Idle (Admin)
100.64.100.2    4      65100 00:00:19            0
100.64.100.4    4      65100 00:00:03 Idle (Admin)

$ vtysh -c 'show bgp ipv4 100.64.120.0/24'
 % Network not in table
$ ping -c 1 -W 1 -I 100.64.100.3 100.64.120.1
ping: sendto: Network unreachable
```

show bgp ipv4, en b2

```
 *>i 100.64.120.0/24  100.64.100.1            0    100      0 65200 i
 * i                  100.64.100.4            0    100      0 65200 i
```

SECCIÓN 06

Elegir la mejor ruta

06

La lista, en orden

	Criterio	Gana	Razón en FRRouting
1	Weight (local al router)	El mayor	`Weight`
2	**`LOCAL_PREF`**	**El mayor**	`Local Pref`
3	Originada localmente	La propia	`Local Route`
4	**Largo de `AS_PATH`**	**El menor**	`AS Path`
5	`ORIGIN`	`i`, luego `e`, luego `?`	`Origin`
6	`MED`	El menor, si vienen del mismo AS	`MED`
7	eBGP contra iBGP	eBGP	`Peer Type`
8	Costo IGP al `NEXT_HOP`	El menor	`IGP Metric`
9	Desempates	El más antiguo, menor router ID…	`Older Path`, `Router ID`

Idea clave

Primero lo que el AS **quiere**. Después lo que sugieren los demás. Al final, lo más barato dentro del AS.

Hot potato

traceroute a x desde b2

```
1  10.207.12.1  0.009 ms
2  100.64.120.1  0.002 ms
```

traceroute a x desde b3

```
1  10.207.34.2  0.004 ms
2  100.64.120.1  0.001 ms
```

show bgp ipv4 100.64.120.0/24, en b2

```
100.64.100.1 (metric 10) from 100.64.100.1 (100.64.100.1)
  Origin IGP, metric 0, localpref 100, valid, internal, best (IGP Metric)
```

Mismo AS, mismo destino, puntas opuestas. En `b1` gana su ruta externa: `best (Peer Type)`.

MED: x pide que le entren por b4

ibgp/med_x.conf

```
route-map MED_LOW permit 10
 set metric 10
route-map MED_HIGH permit 10
 set metric 50
```

ibgp/med_x.conf

```
router bgp 65200
 address-family ipv4 unicast
  neighbor 10.207.4.1 route-map MED_LOW out
  neighbor 10.207.1.1 route-map MED_HIGH out
```

show bgp ipv4 100.64.120.0/24, en b1

```
65200
  100.64.100.4 (metric 30) from 100.64.100.4 (100.64.100.4)
    Origin IGP, metric 10, localpref 100, valid, internal, best (MED)
65200
  10.207.1.2 from 10.207.1.2 (100.64.120.1)
    Origin IGP, metric 50, valid, external
```

Cold potato

```
show bgp ipv4 100.64.120.0/24, en b2

Paths: (1 available, best #1, table default)
  65200
    100.64.100.4 (metric 20) from 100.64.100.4 (100.64.100.4)
      Origin IGP, metric 10, localpref 100, valid, internal, best (First path received)
```

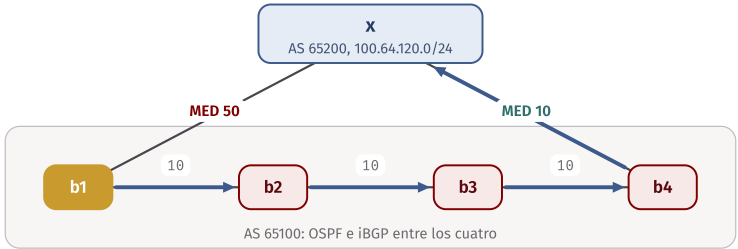
```
traceroute a x
desde b1

 1  10.207.12.2  0.004 ms
 2  10.207.23.2  0.002 ms
 3  10.207.34.2  0.001 ms
 4  100.64.120.1  0.002 ms
```

Sorprendente

`b1` tiene un enlace directo con `x`, y aun así cruza todo el AS: MED va antes que eBGP y que IGP. Y como su mejor camino ya no es externo, deja de ofrecérselo a los demás: `b2` se queda con uno solo.

Los tres caminos, en la misma red



TRACEROUTE A X DESDE B1

```
1  10.207.12.2          2  10.207.23.2                    best (MED)
3  10.207.34.2          4  100.64.120.1
```

x pide entrar por b4 con un MED menor (10 contra 50). El MED va antes que eBGP y que IGP, así que b1 cruza todo el AS aunque tenga un enlace directo con x. Eso es cold potato.

Hot potato y cold potato

	Hot potato	**Cold potato**
Regla	Salgo por la salida más cercana a mí	Llevo hasta la entrada que prefiere el otro
Quién carga la distancia	El siguiente AS	Yo
A quién favorece	Al AS que manda	Al AS que recibe, y a menudo al usuario
Efecto típico	Caminos asimétricos	Caminos más controlados

Error común

Creer que BGP busca el menor retardo. Ningún atributo lo mide. El `AS_PATH` más corto ni siquiera es el primer criterio, y solo cuenta sistemas autónomos: un AS puede ser un edificio o un continente.

SECCIÓN 07

Quién le paga a quién

07

Dos relaciones

Cliente y proveedor *(customer, provider)*

El cliente paga para que el proveedor lo conecte con **todo Internet**. El servicio se llama transit.

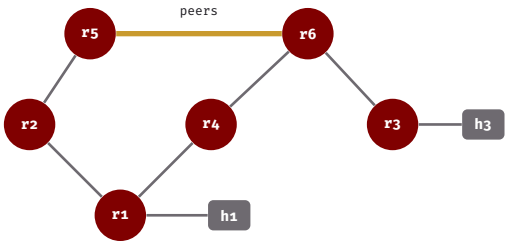
Peers *(peering)*

Intercambian tráfico **entre sus redes y sus clientes**, por lo general sin pagarse. No es salida a Internet.

Tráfico en un enlace con mi…	Efecto
cliente	**Gano**, en cualquier sentido
peer	Ni gano ni pierdo, salvo el costo del enlace
proveedor	**Pago**, en cualquier sentido

Anunciar una ruta a un vecino es ofrecerle transportar sus paquetes: exportar es una decisión de negocio.

Los seis AS, otra vez



Arriba el proveedor, abajo el cliente.

	Prov.	Peer	Clientes
r1	r2, r4		
r2	r5		r1
r3	r6		
r4	r6		r1
r5		r6	r2
r6		r5	r3, r4

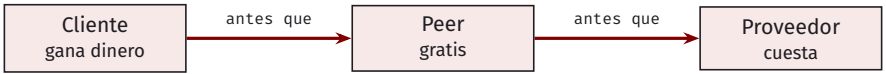
r1 es **multihomed**: dos proveedores, para no depender de uno solo.

SECCIÓN 08

Importación y exportación

08

Importación: a qué ruta le doy preferencia



- Sin importar el largo del `AS_PATH`.
- Se implementa con `local-preference`, que gana antes de mirar el largo.

Pregunta

Un AS recibe el mismo prefijo de un proveedor, con `AS_PATH` corto, y de un cliente, con `AS_PATH` largo. ¿Cuál prefiere, y por qué?

Exportación: qué le cuento a quién

La ruta viene de mi…	¿A un cliente?	¿A un peer?	¿A un proveedor?
propia red	Sí	Sí	Sí
cliente	Sí	Sí	Sí
peer	Sí	**No**	**No**
proveedor	Sí	**No**	**No**

- Una ruta de cliente, a todos: el tráfico termina en un enlace que genera ingresos.
- Una ruta de proveedor a un peer: yo pagaría por llevarle su tráfico. Sería transit gratis.
- A los clientes, todo: es el servicio por el que pagan.

Error común

Un peer no da acceso a todo Internet. Si `r5` y `r6` dejan de tener peering, los clientes de uno no alcanzan a los del otro.

Sin políticas: encuentren el problema

show bgp ipv4 100.64.4.0/24, en r2

```
Paths: (2 available, best #1, table default)
  64601 64604
    100.64.2.130 from 100.64.2.130 (100.64.1.1)
      Origin IGP, valid, external, best (AS Path)
  64605 64606 64604
    100.64.5.133 from 100.64.5.133 (100.64.5.1)
      Origin IGP, valid, external
```

traceroute -s 100.64.2.1 100.64.4.1, en r2

```
 1  100.64.2.130  0.012 ms
 2  100.64.4.1  0.003 ms
```

¿Qué dice ese camino, si `r1` es cliente de `r2` y de `r4`?

Tránsito gratis

lo que r1 le anuncia a r2, sin políticas (recortado)

```
     Network          Next Hop            Path
*>   100.64.1.0/24    0.0.0.0             i
*>   100.64.2.0/24    100.64.2.129        64602 i
*>   100.64.3.0/24    100.64.4.129        64604 64606 64603 i
*>   100.64.4.0/24    100.64.4.129        64604 i
*>   100.64.5.0/24    100.64.2.129        64602 64605 i
*>   100.64.6.0/24    100.64.4.129        64604 64606 i
```

El AS más chico paga dos veces

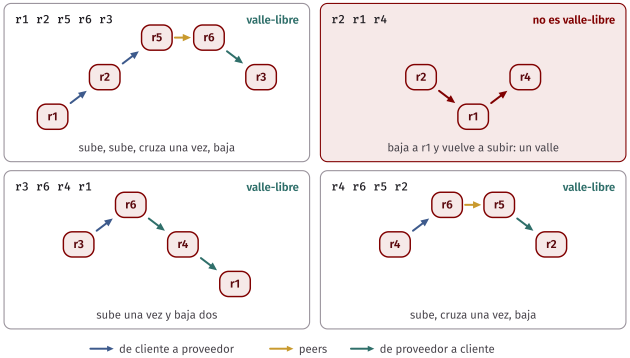
`r2` llega a `r4` a través de `r1`. `r1` paga los dos enlaces para llevar tráfico entre sus proveedores. BGP hizo lo que se le pidió: es falta de política. Cuando pasa en la realidad, es un **route leak**.

SECCIÓN 09

Caminos valle-libre

09

Una montaña, no un valle



Valle-libre *(valley-free)*: cero o más subidas de cliente a proveedor, a lo sumo un enlace de peers, y cero o más bajadas de proveedor a cliente. Las reglas de exportación lo garantizan.

Por qué dan estabilidad

Gao y Rexford (2001)

Si se cumple todo esto, BGP converge a un estado estable, en cualquier orden:

1. No hay ciclos de proveedores.
2. Cada AS prefiere rutas de clientes antes que de peers y proveedores.
3. Se cumple la tabla de exportación.

- Ninguna condición pide coordinación global: cada uno las cumple porque le conviene.
- **Es condicional.** Basta que un AS no cumpla. BGP no verifica nada.
- **Estable no significa bueno.** El resultado puede ser largo, lento o caro para alguien.

No es automático: `spvp.py`

Destino 0, cliente de 1, 2 y 3. Ciclo: 1 es proveedor de 2, 2 de 3, y 3 de 1. Cada uno prefiere a su cliente del anillo.

```
python3 spvp.py

Anillo con un ciclo de proveedores:
  vuelta 1:  1 0   2 0   3 1 0
  vuelta 2:  1 2 0   2 3 1 0   3 1 2 0
  vuelta 3:  1 0   2 0   3 1 0
  repite un estado ya visto: en este orden no termina nunca

Anillo sin ciclo de proveedores:
  vuelta 1:  1 0   2 0   3 0
  vuelta 2:  1 2 0   2 3 0   3 0
  vuelta 3:  1 3 0   2 3 0   3 0
  vuelta 4:  1 3 0   2 3 0   3 0
  estable
```

Cada uno persigue al siguiente, y nadie descansa: es un *bad gadget* (Griffin, Shepherd y Wilfong). Basta invertir un enlace del ciclo.

SECCIÓN 10

Políticas en FRRouting

10



route-map

forma general

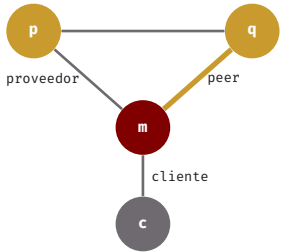
```
route-map NOMBRE permit 10
 match <condición>
 set <cambio>
route-map NOMBRE permit 20
 match <otra condición>
router bgp 65300
 address-family ipv4 unicast
  neighbor 100.64.130.130 route-map
      NOMBRE in
```

`in`: importación. `out`: exportación. Un vecino, un `route-map` por dirección.

1. Se recorren las entradas por número.
2. Sin `match`, aplica a todas. Con varios, deben cumplirse todos.
3. En la primera que aplica: con `permit`, se hacen los `set` y pasa. Con `deny`, se bota.
4. **Si ninguna aplica, se bota.**

`prefix-list`: nombra prefijos.
Comunidad: etiqueta `AS:valor`.

Un ejemplo, pieza por pieza



- m: AS 65300, `100.64.130.0/24`.
- p: 65310, `100.64.131.0/24`.
- q: 65320, `100.64.132.0/24`.
- c: 65330, `100.64.133.0/24`.

Un ejemplo, no el conjunto completo de reglas. Eso es el laboratorio.

code/week-06

```
$ make up-poli
```

Paso 1: `local-preference`

poli/step1_localpref.conf

```
route-map FROM_Q permit 10
 set local-preference 150
router bgp 65300
 address-family ipv4 unicast
  neighbor 100.64.130.130 route-map FROM_Q in
```

show bgp ipv4 100.64.131.0/24, en m

```
65320 65310
  100.64.130.130 from 100.64.130.130 (100.64.132.1)
    Origin IGP, localpref 150, valid, external, best (Local Pref)
65310
  100.64.131.129 from 100.64.131.129 (100.64.131.1)
    Origin IGP, metric 0, valid, external
```

Dos AS le ganan a uno. `LOCAL_PREF` no sale del AS: en `C`, la ruta llega sin él.

Paso 2: comunidades

poli/step2_community.conf

```
route-map FROM_C permit 10
 set community 65300:30
router bgp 65300
 address-family ipv4 unicast
  neighbor 100.64.130.134 route-map FROM_C in
```

show bgp ipv4 100.64.133.0/24, en m

```
65330
  100.64.130.134 from 100.64.130.134 (100.64.133.1)
    Origin IGP, metric 0, valid, external, best (First path received)
    Community: 65300:30
```

Una vez en la tabla, el `AS_PATH` no dice qué tipo de vecino era: se marca al entrar y se lee al salir. El `30` lo interpreta el operador.

Una comunidad sí viaja

```
show bgp ipv4 100.64.133.0/24, en p

65300 65330
  100.64.131.130 from 100.64.131.130 (100.64.130.1)
    Origin IGP, valid, external, best (AS Path)
    Community: 65300:30
```

Por eso reemplazar

`set community` sin más opciones **reemplaza** las que traía la ruta: un vecino no debería poder entrar en la lógica interna de otro AS mandando una comunidad con su número. Para agregar sin borrar: `additive`.

Paso 3: exportación

poli/step3_export.conf

```
ip prefix-list MINE seq 5 permit 100.64.130.0/24
bgp community-list standard TAGGED_C permit 65300:30
route-map TO_P permit 10
 match ip address prefix-list MINE
route-map TO_P permit 20
 match community TAGGED_C
router bgp 65300
 address-family ipv4 unicast
  neighbor 100.64.131.129 route-map TO_P out
```

Dos `permit`, y todo lo demás cae en el `deny` implícito. Se usa comunidad y no una `prefix-list` del cliente porque se pone una vez, donde se conoce la relación, y sirve en todas las salidas.

Paso 3: el resultado

m hacia p, antes (columnas recortadas)

```
     Network          Next Hop            Path
*>   100.64.130.0/24  0.0.0.0              i
*>   100.64.131.0/24  100.64.131.129      65310 i
*    100.64.131.0/24  100.64.130.130      65320 65310 i
*>   100.64.132.0/24  100.64.130.130      65320 i
*    100.64.132.0/24  100.64.131.129      65310 65320 i
*>   100.64.133.0/24  100.64.130.134      65330 i
```

m hacia p, después

```
     Network          Path
*>   100.64.130.0/24  i
*>   100.64.133.0/24  65330 i
```

Hacia p ya no sale la ruta del peer q. Con esas tres piezas ya se puede escribir cualquier política de la tabla de exportación.

El deny implícito

m: route-map de salida hacia q con un nombre que no existe

```
The route-map 'TYPO_NAME' does not exist.

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
100.64.130.130  4      65320        26        32        9    0    0 00:01:09            2        0 N/A
100.64.130.134  4      65330        35        37        9    0    0 00:00:49            1        4 N/A
100.64.131.129  4      65310        12        18        9    0    0 00:01:09            2        2 N/A
```

Error común

Un `route-map` que no deja pasar nada deja la sesión arriba y no anuncia nada, sin ningún error. Si el vecino no recibe nada, miren los `route-map` y `advertised-routes`.

SECCIÓN 11

Con las políticas puestas

11

Lo que deben observar en el laboratorio

traceroute de r2 a r4

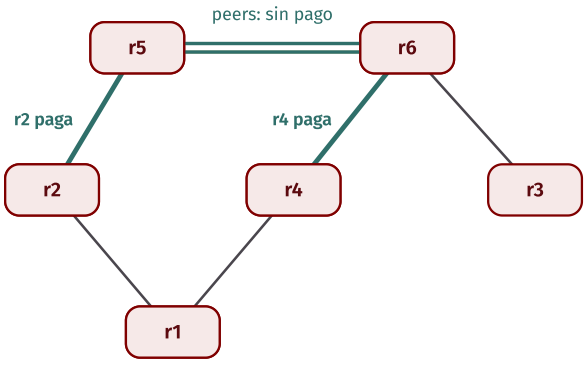
```
 1  100.64.5.133  0.007 ms
 2  100.64.5.130  0.002 ms
 3  100.64.4.1    0.002 ms
```

show bgp ipv4 100.64.1.0/24, en r6

```
64604 64601
  100.64.6.130 from 100.64.6.130 (100.64.4.1)
    Origin IGP, localpref 300, valid, external, best (Local Pref)
    Community: 64606:100
64605 64602 64601
  100.64.5.129 from 100.64.5.129 (100.64.5.1)
    Origin IGP, localpref 200, valid, external
    Community: 64606:200
```

- `r1` solo anuncia su propio prefijo. El camino de `r2` a `r4` pasa por arriba, y es valle-libre.
- Es un salto más, y es lo correcto: ahora paga quien debe pagar.
- `make test` compara las tablas reales con lo que predice `spvp.py`.

El tránsito gratis, antes y después



TRACEROUTE DE R2 A R4

```
1  100.64.5.133
2  100.64.5.130
3  100.64.4.1
```

r2 → r5 → r6 → r4: valle-libre

Con las políticas, r1 solo anuncia su prefijo. r2 sube por r5, cruza el peering hasta r6 y baja a r4. Es un salto más, y ahora el que paga es quien debe pagar.

Preferir un proveedor

show bgp ipv4 100.64.3.0/24, en r1: un LOCAL_PREF mayor en lo que entra desde r2

```
64602 64605 64606 64603
  100.64.2.129 from 100.64.2.129 (100.64.2.1)
    Origin IGP, localpref 120, valid, external, best (Local Pref)
    Community: 64601:300
64604 64606 64603
  100.64.4.129 from 100.64.4.129 (100.64.4.1)
    Origin IGP, localpref 100, valid, external
    Community: 64601:300
```

Un camino más largo en AS gana porque `r1` lo quiere así. Si cae el enlace con `r2`, el tráfico pasa solo a `r4`, y vuelve cuando el enlace regresa.

La ida no decide la vuelta

traceroute desde h1 hacia h3

```
1  100.64.1.1   0.005 ms
2  100.64.2.129  0.003 ms
3  100.64.5.133  0.002 ms
4  100.64.6.129  0.002 ms
5  100.64.6.134  0.001 ms
6  100.64.3.10   0.005 ms
```

traceroute desde h3 hacia h1

```
1  100.64.3.1   0.006 ms
2  100.64.6.133  0.002 ms
3  100.64.6.130  0.002 ms
4  100.64.2.130  0.001 ms
5  100.64.1.10   0.003 ms
```

Idea clave

`LOCAL_PREF` controla por dónde **sale** el tráfico. La vuelta sigue entrando por `r4` (`100.64.6.130`). En el salto 4, `r1` responde con la dirección de la interfaz por la que saldrá la respuesta. Para la entrada solo hay sugerencias: MED, prefijos más específicos, prepending.

SECCIÓN 12

Anycast

12



La misma dirección en muchos lugares

Anycast

Varios servidores con la misma dirección IP. El routing lleva cada paquete a uno, normalmente el más cercano según BGP.

- Servidores raíz de DNS.
- CDN.
- Reparte los ataques de DoS.

la dirección 100.64.200.1, en r2 y en r6

```
$ ases/anycast.sh r2 64602 on
$ ases/anycast.sh r6 64606 on
$ docker exec clab-w06-ases-h1 sh -c 'nc ... 5300'
r2
$ docker exec clab-w06-ases-h3 sh -c 'nc ... 5300'
r6
$ ases/anycast.sh r2 64602 off
$ docker exec clab-w06-ases-h1 sh -c 'nc ... 5300'
r6
```

Cercano, ¿para quién?

show bgp ipv4 100.64.200.0/24, en r5, a la misma distancia de los dos

```
64602
  100.64.5.134 from 100.64.5.134 (100.64.2.1)
    Origin IGP, metric 0, valid, external, best (Older Path)
64606
  100.64.5.130 from 100.64.5.130 (100.64.6.1)
    Origin IGP, metric 0, valid, external
```

Error común

“Cercano” es para BGP, no en kilómetros. Y si la ruta cambia en medio de una conexión TCP, otra instancia no conoce esa conexión y la corta.

SECCIÓN 13

Presencia en Internet

13

Una empresa nueva en Lima

1. **Direcciones**: de su proveedor, o propias, del registro regional (LACNIC).
2. **Número de AS**: si tendrá política propia, por ejemplo con dos proveedores.
3. **Proveedores**: sesión eBGP con cada uno. Anunciar solo lo propio.
4. **Red interna**: OSPF, iBGP entre bordes, y NAT o direcciones públicas.
5. **DNS**: dominio registrado, un NS en el dominio superior, y un registro A hacia el servidor.

Cómo se enlazan

Sin DNS no te encuentran. Sin BGP no te alcanzan. Sin OSPF el paquete no cruza tu red. Y sin direcciones no hay nada que anunciar.

SECCIÓN 14

Laboratorio

14

Conectar sistemas autónomos con políticas

1. Una sesión hasta `Established`. Capturar OPEN y UPDATE.
2. Leer un `AS_PATH` desde otro AS.
3. Encontrar el tránsito gratis, y explicar quién paga.
4. Políticas completas: `lab/r1.conf` a `lab/r6.conf`.
5. Preferir un proveedor: `lab/prefer.conf`.
6. Entregar **P1** y leer el enunciado de **P2**.



Las pruebas de `lab/` se saltan hasta que borren las líneas `TODO`.

Para la próxima clase

- Terminar el laboratorio y empezar **P2**.
- Leer Kurose y Ross, sec. 5.4.5, CS168: *BGP Implementation and Issues*, y la sección *Prefix Filtering* de la RFC 7454.

Pregunta

BGP le cree a su vecino lo que anuncia. Si `r1` empieza a anunciar `100.64.5.0/24`, el prefijo de `r5`, ¿qué elige `r2`, que tiene a `r1` de cliente? ¿Quién podría impedirlo?