


Computación Centrada en Redes

# Los problemas de BGP y la estructura de Internet

## Routing

**Gerson Garrido**

Semana 7

UNIVERSIDAD NACIONAL DE INGENIERÍA

# Hoy

## Sesión A: los problemas de BGP

1. BGP cree lo que le dicen
2. Prefix hijack, y YouTube en 2008
3. Route leak, y junio de 2019
4. Filtros y límites
5. RPKI, y lo que no protege
6. Facebook, en octubre de 2021

## Sesión B: la Internet real y repaso

7. Tránsito y peering
8. IXPs: lo medimos
9. El Perú y sus cables
10. Repaso para el parcial

**Lectura**

Kurose y Ross, sec. 5.4.5 (sesión A) y 1.3.3 (sesión B). CS168: *BGP Implementation and Issues* y *Model for Inter-Domain Routing*. RFC 7454, sección *Prefix Filtering*. Complementarias: RFC 6480 y RFC 6811.

SECCIÓN 01

# BGP cree lo que le dicen

01

# Qué hace un router con un anuncio

Un UPDATE dice: “a este prefijo se llega por mí, por este `AS_PATH`”.

1. Lo descarta si ve su propio AS: habría un ciclo.
2. Le aplica su política de importación.
3. Si es la mejor ruta, la instala y la reanuncia.

# Qué hace un router con un anuncio

Un UPDATE dice: “a este prefijo se llega por mí, por este `AS_PATH`”.

1. Lo descarta si ve su propio AS: habría un ciclo.
2. Le aplica su política de importación.
3. Si es la mejor ruta, la instala y la reanuncia.

**Lo que nadie comprueba**

Que el AS de origen tenga derecho a ese prefijo. Que el camino exista. El `AS_PATH` es solo una lista de números que alguien escribió.

# Qué hace un router con un anuncio

Un UPDATE dice: “a este prefijo se llega por mí, por este `AS_PATH`”.

1. Lo descarta si ve su propio AS: habría un ciclo.
2. Le aplica su política de importación.
3. Si es la mejor ruta, la instala y la reanuncia.

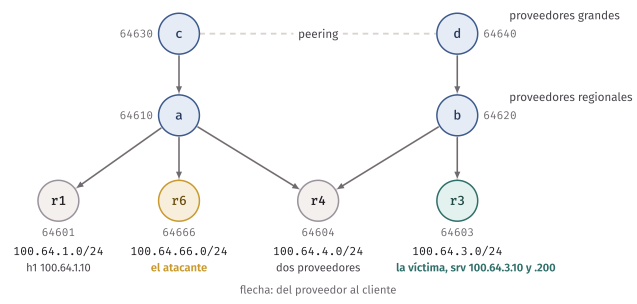
**Lo que nadie comprueba**

Que el AS de origen tenga derecho a ese prefijo. Que el camino exista. El `AS_PATH` es solo una lista de números que alguien escribió.

**La sesión no es el mensaje**

La sesión sobre TCP se puede proteger, pero eso cuida al mensajero: un vecino legítimo puede anunciar cualquier cosa.

# La red de esta sesión




- `r1`, AS 64601: `100.64.1.0/24`
- `r3`, la víctima, AS 64603: `100.64.3.0/24`
- `r4`: cliente con dos proveedores
- `r6`: el atacante

**Sin local-preference**

Decide el largo del camino: se puede predecir contando. Al final de cada caso vemos qué cambia con política.

# El punto de partida

el camino normal

```
$ docker exec clab-w07-bgp-h1 traceroute -n -w 1 100.64.3.10
 1  100.64.1.1  0.003 ms  0.002 ms  0.002 ms
 2  10.207.4.2  0.001 ms  0.001 ms  0.003 ms
 3  10.207.2.2  0.002 ms  0.003 ms  0.001 ms
 4  10.207.1.2  0.001 ms  0.002 ms  0.003 ms
 5  10.207.3.1  0.001 ms  0.002 ms  0.002 ms
 6  10.207.8.1  0.001 ms  0.002 ms  0.002 ms
 7  100.64.3.10  0.001 ms  0.002 ms  0.002 ms
$ docker exec clab-w07-bgp-a vtysh -c "show bgp ipv4 unicast"
     Network          Next Hop            Metric LocPrf Weight Path
 *>  100.64.1.0/24    10.207.4.1               0             0 64601 i
 *>  100.64.3.0/24    10.207.2.2                             0 64630 64640 64620 64603 i
 *>  100.64.4.0/24    10.207.6.1               0             0 64604 i
 *>  100.64.66.0/24   10.207.5.1               0             0 64666 i
```

Sube por `a` y `c`, cruza el peering y baja por `d` y `b`. Tiempos casi cero: todo corre en una computadora.

SECCIÓN 02

# Prefix hijack

02

# Prefix hijack

**Definición**

Un AS origina una ruta hacia un prefijo que no le fue asignado, y otras redes la aceptan. El tráfico de esas redes termina en el AS equivocado.

El atacante puede competir de dos maneras, y se comportan muy distinto:

- **Con el mismo prefijo.** Cada AS elige con su proceso de decisión. La red se parte en dos.
- **Con un prefijo más largo.** Lo decide longest prefix match. Gana en todo lugar donde lo acepten.

# Mismo prefijo: predigan antes de correrlo

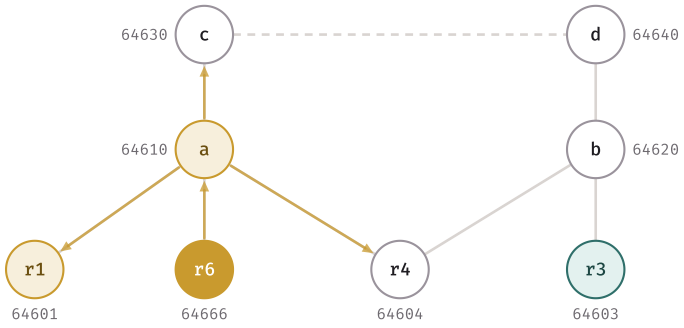
| AS | Hacia la víctima | Largo | Hacia el atacante | Largo |
|---|---|---|---|---|
| r1 | 64610 64630 64640 64620 64603 | 5 | 64610 64666 | 2 |
| a | 64630 64640 64620 64603 | 4 | 64666 | 1 |
| c | 64640 64620 64603 | 3 | 64610 64666 | 2 |
| d | 64620 64603 | 2 | 64630 64610 64666 | 3 |
| b | 64603 | 1 | 64640 64630 64610 64666 | 4 |
| r4 | 64620 64603 | 2 | 64610 64666 | 2 |

# Mismo prefijo: predigan antes de correrlo

| AS | Hacia la víctima | Largo | Hacia el atacante | Largo |
|---|---|---|---|---|
| r1 | 64610 64630 64640 64620 64603 | 5 | 64610 64666 | 2 |
| a | 64630 64640 64620 64603 | 4 | 64666 | 1 |
| c | 64640 64620 64603 | 3 | 64610 64666 | 2 |
| d | 64620 64603 | 2 | 64630 64610 64666 | 3 |
| b | 64603 | 1 | 64640 64630 64610 64666 | 4 |
| r4 | 64620 64603 | 2 | 64610 64666 | 2 |

- `r1`, `a` y `c` caen; `d` y `b` no.
- `r4` empata, y se queda con la ruta que ya tenía (`Older Path`). Ese resultado depende del orden de los eventos.

# La red se parte en dos

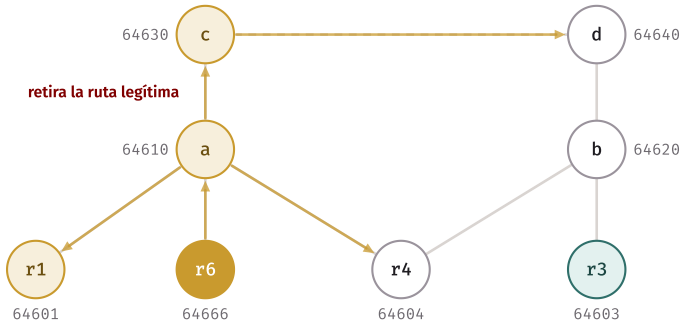



largo del camino

| | vícт. | atac. |
|---|---|---|
| a | 4 | 1 atacante |
| r1 | 5 | 2 atacante |
| c | | |
| d | | |
| b | | |
| r4 | | |

r6 anuncia el /24 de la víctima sin que nadie le pida prueba. a y r1 lo prefieren: su camino es más corto.

# La red se parte en dos

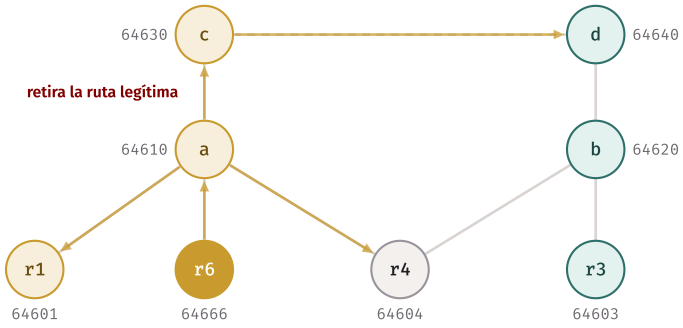



largo del camino

| | vícт. | atac. |
|---|---|---|
| a | 4 | 1 atacante |
| r1 | 5 | 2 atacante |
| c | 3 | 2 atacante |
| d | | |
| b | | |
| r4 | | |

c también cae (2 contra 3). Su mejor ruta ya pasa por a, así que le retira a a la ruta legítima.

# La red se parte en dos

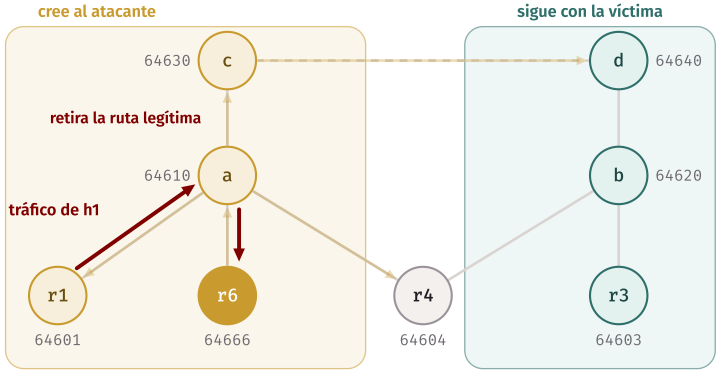



largo del camino

| | vícт. | atac. |
|---|---|---|
| a | 4 | 1 atacante |
| r1 | 5 | 2 atacante |
| c | 3 | 2 atacante |
| d | 2 | 3 víctima |
| b | 1 | 4 víctima |
| r4 | 2 | 2 empate |

d y b no caen: la víctima les queda más cerca. r4 empata en 2 y se queda con la que ya tenía.

# La red se parte en dos




largo del camino

| | vícт. | atac. | |
|---|---|---|---|
| a | 4 | 1 | atacante |
| r1 | 5 | 2 | atacante |
| c | 3 | 2 | atacante |
| d | 2 | 3 | víctima |
| b | 1 | 4 | víctima |
| r4 | 2 | 2 | empate |

La red se parte en dos. curl desde h1 llega al atacante, sin ninguna señal de error, y la tabla de r3 no cambió.

# python3 scenarios.py same

desde h1 no hay ninguna señal de error

```
$ docker exec clab-w07-bgp-h1 curl -s http://100.64.3.10:8080/
ATACANTE: AS 64666 recibio esta conexion
$ docker exec clab-w07-bgp-h1 traceroute -n -w 1 100.64.3.10
 1  100.64.1.1  0.003 ms  0.001 ms  0.003 ms
 2  10.207.4.2  0.001 ms  0.004 ms  0.002 ms
 3  100.64.3.10  0.002 ms  0.002 ms  0.002 ms
$ docker exec clab-w07-bgp-srv curl -s http://100.64.3.10:8080/
servidor legitimo de AS 64603
```

- Tres saltos en vez de siete, y la misma dirección al final.
- Si el cliente no comprueba con quién habla, no se entera. Eso es lo que hace TLS.
- La tabla de la víctima no cambió. Un hijack se detecta **mirando desde afuera**.

# Dos ASes con las mismas rutas, dos decisiones

Es la salida de `show bgp ipv4 unicast 100.64.3.0/24` en cada uno, sin las líneas de direcciones.

c elige al atacante

```
64610 64666
    Origin IGP, valid, external,
    best (AS Path)
64640 64620 64603
    Origin IGP, valid, external
```

d elige a la víctima

```
64620 64603
    Origin IGP, valid, external,
    best (AS Path)
64630 64610 64666
    Origin IGP, valid, external
```

# Dos ASes con las mismas rutas, dos decisiones

Es la salida de `show bgp ipv4 unicast 100.64.3.0/24` en cada uno, sin las líneas de direcciones.

c elige al atacante

```
64610 64666
    Origin IGP, valid, external,
    best (AS Path)
64640 64620 64603
    Origin IGP, valid, external
```

d elige a la víctima

```
64620 64603
    Origin IGP, valid, external,
    best (AS Path)
64630 64610 64666
    Origin IGP, valid, external
```

**¿Y a? Ya no ve la ruta buena**

La mejor ruta de c ahora pasa por a, así que c se la retira. Un AS solo reanuncia su mejor ruta. Mientras dure el ataque, a no sabe que existe un camino a la víctima.

# Con un prefijo más específico

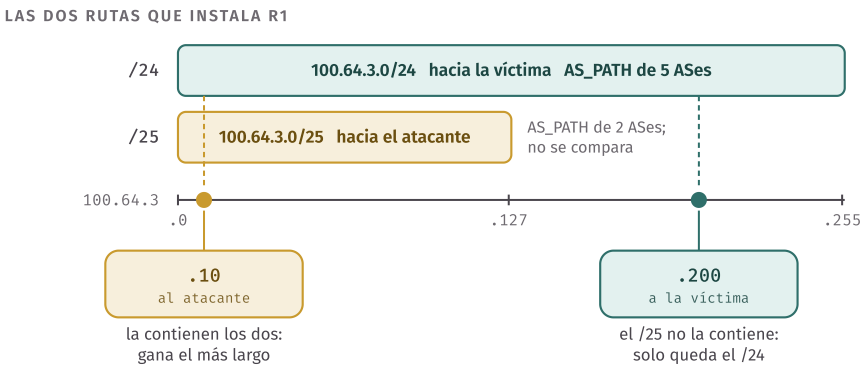
Un `/25` y un `/24` no son dos rutas al mismo destino: son **dos prefijos**.

- El proceso de decisión de BGP nunca los compara.
- Los dos se instalan en la forwarding table.
- El forwarding elige el más largo, sin mirar el `AS_PATH` ni nada más.

**Error común**

"Gana el `AS_PATH` más corto". Aquí el `AS_PATH` nunca llega a compararse.

# Decide cada dirección



# `python3 scenarios.py specific`

también b y d, y hasta la víctima

```
$ docker exec clab-w07-bgp-b vtysh -c "show ip route 100.64.3.10"
Routing entry for 100.64.3.0/25
  * 10.207.3.2, via eth1, weight 1
$ docker exec clab-w07-bgp-r3 vtysh -c "show ip route 100.64.3.10"
Routing entry for 100.64.3.0/25
  * 10.207.8.2, via eth2, weight 1
$ docker exec clab-w07-bgp-h1 curl -s http://100.64.3.10:8080/
ATACANTE: AS 64666 recibio esta conexion
$ docker exec clab-w07-bgp-h1 curl -s http://100.64.3.200:8080/
servidor legitimo de AS 64603
```

- `r3` tiene el servidor en una interfaz, y lo manda afuera: el `/24` conectado es más corto que el `/25`.
- La distancia administrativa solo compara rutas al **mismo** prefijo.
- `.200` no está en el `/25`: el hijack llega hasta donde llega el prefijo.

# Caso real: Pakistan Telecom y YouTube

Domingo 24 de febrero de 2008, UTC. Fuente: estudio de caso del RIPE NCC, con datos de RIS.

| | |
|---|---|
| Antes | AS36561 (YouTube) anuncia `208.65.152.0/22`. |
| 18:47 | AS17557 (Pakistan Telecom) anuncia `208.65.153.0/24`. AS3491 (PCCW Global), uno de sus proveedores, lo propaga. |
| 20:07 | YouTube anuncia el mismo `/24`. Ahora compiten por política. |
| 20:18 | YouTube anuncia dos `/25`: gana por longest prefix match donde lleguen. |
| 20:51 | Los anuncios, incluido el `/24` de AS17557, se ven con otro 17557 antepuesto. El camino más largo favorece a YouTube. |
| 21:01 | PCCW retira los prefijos originados por AS17557. Fin del hijack. |

# Caso real: Pakistan Telecom y YouTube

Domingo 24 de febrero de 2008, UTC. Fuente: estudio de caso del RIPE NCC, con datos de RIS.

| | |
|---|---|
| Antes | AS36561 (YouTube) anuncia `208.65.152.0/22`. |
| 18:47 | AS17557 (Pakistan Telecom) anuncia `208.65.153.0/24`. AS3491 (PCCW Global), uno de sus proveedores, lo propaga. |
| 20:07 | YouTube anuncia el mismo `/24`. Ahora compiten por política. |
| 20:18 | YouTube anuncia dos `/25`: gana por longest prefix match donde lleguen. |
| 20:51 | Los anuncios, incluido el `/24` de AS17557, se ven con otro 17557 antepuesto. El camino más largo favorece a YouTube. |
| 21:01 | PCCW retira los prefijos originados por AS17557. Fin del hijack. |

Según el estudio, Pakistán buscaba bloquear el sitio. Un `/24` dentro de un `/22`: no importó la distancia.

# Caso real: Pakistan Telecom y YouTube

Domingo 24 de febrero de 2008, UTC. Fuente: estudio de caso del RIPE NCC, con datos de RIS.

| | |
|---|---|
| Antes | AS36561 (YouTube) anuncia `208.65.152.0/22`. |
| 18:47 | AS17557 (Pakistan Telecom) anuncia `208.65.153.0/24`. AS3491 (PCCW Global), uno de sus proveedores, lo propaga. |
| 20:07 | YouTube anuncia el mismo `/24`. Ahora compiten por política. |
| 20:18 | YouTube anuncia dos `/25`: gana por longest prefix match donde lleguen. |
| 20:51 | Los anuncios, incluido el `/24` de AS17557, se ven con otro 17557 antepuesto. El camino más largo favorece a YouTube. |
| 21:01 | PCCW retira los prefijos originados por AS17557. Fin del hijack. |

Según el estudio, Pakistán buscaba bloquear el sitio. Un `/24` dentro de un `/22`: no importó la distancia.

# Caso real: Pakistan Telecom y YouTube

Domingo 24 de febrero de 2008, UTC. Fuente: estudio de caso del RIPE NCC, con datos de RIS.

| | |
|---|---|
| Antes | AS36561 (YouTube) anuncia `208.65.152.0/22`. |
| 18:47 | AS17557 (Pakistan Telecom) anuncia `208.65.153.0/24`. AS3491 (PCCW Global), uno de sus proveedores, lo propaga. |
| 20:07 | YouTube anuncia el mismo `/24`. Ahora compiten por política. |
| 20:18 | YouTube anuncia dos `/25`: gana por longest prefix match donde lleguen. |
| 20:51 | Los anuncios, incluido el `/24` de AS17557, se ven con otro 17557 antepuesto. El camino más largo favorece a YouTube. |
| 21:01 | PCCW retira los prefijos originados por AS17557. Fin del hijack. |

Según el estudio, Pakistán buscaba bloquear el sitio. Un `/24` dentro de un `/22`: no importó la distancia.

# Cómo responde la víctima

el atacante anuncia el /24

```
$ python3 scenarios.py same
$ docker exec clab-w07-bgp-h1 curl ...
ATACANTE: AS 64666 recibio esta conexion
$ python3 scenarios.py counter
$ docker exec clab-w07-bgp-h1 curl ...
servidor legitimo de AS 64603
```

1. Anunciar más específicos.
2. Llamar a los proveedores del atacante.

No controla los filtros de los demás.

# Cómo responde la víctima

el atacante anuncia el /24

```
$ python3 scenarios.py same
$ docker exec clab-w07-bgp-h1 curl ...
ATACANTE: AS 64666 recibio esta conexion
$ python3 scenarios.py counter
$ docker exec clab-w07-bgp-h1 curl ...
servidor legitimo de AS 64603
```

1. Anunciar más específicos.
2. Llamar a los proveedores del atacante.

No controla los filtros de los demás.

**Esta defensa tiene techo**

RFC 7454, sec. 6.1.3: los prefijos IPv4 más largos que `/24` “generalmente ni se anuncian ni se aceptan”. En una consulta del estudio de RIPE: el `/22` se veía desde 112 peers, el `/24` desde 105 y el `/25` solo desde 21.

SECCIÓN 03

# Route leak

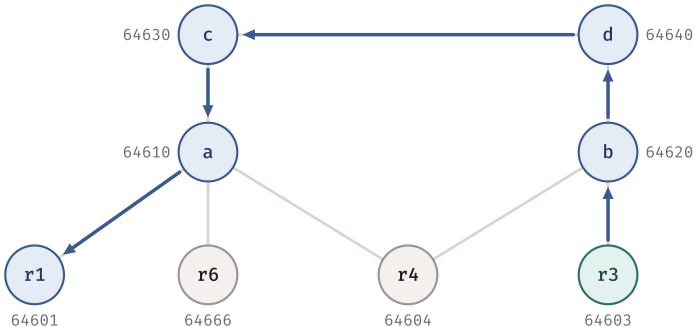
03

# Route leak

**Definición (RFC 7908)**
La propagación de anuncios de routing más allá de su alcance previsto.

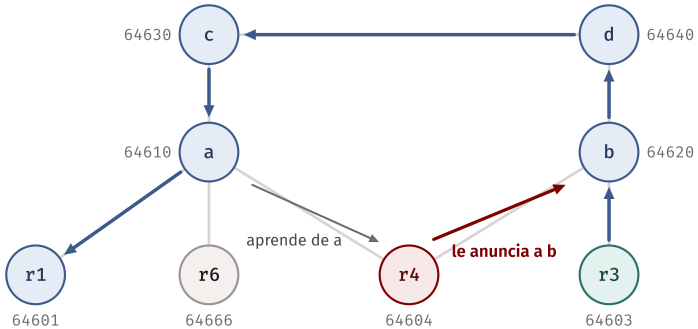
- El origen puede ser totalmente correcto. Lo que está mal es **por dónde** pasa.
- Viola la regla de la semana 6: lo de un proveedor solo va a clientes.
- **Tipo 1, Hairpin Turn:** un AS aprende de un proveedor y le anuncia a otro.

# La fuga de r4, paso a paso




Normal: r4 solo anuncia lo suyo. El tráfico de srv hacia h1 sube por b y d, cruza el peering y baja por c y a.

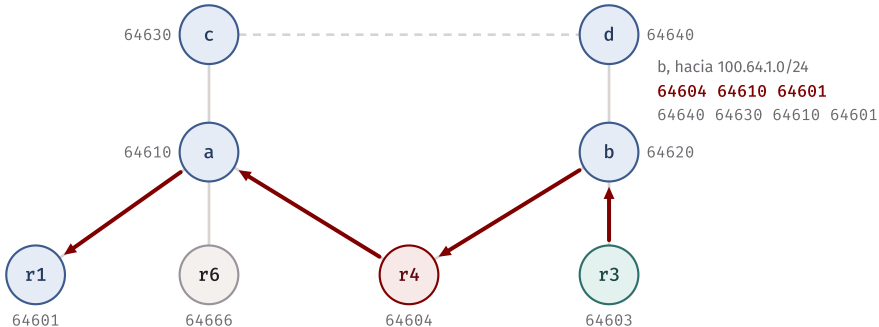
# La fuga de r4, paso a paso




prefijos que r4 le anuncia a b: de 1 a 4

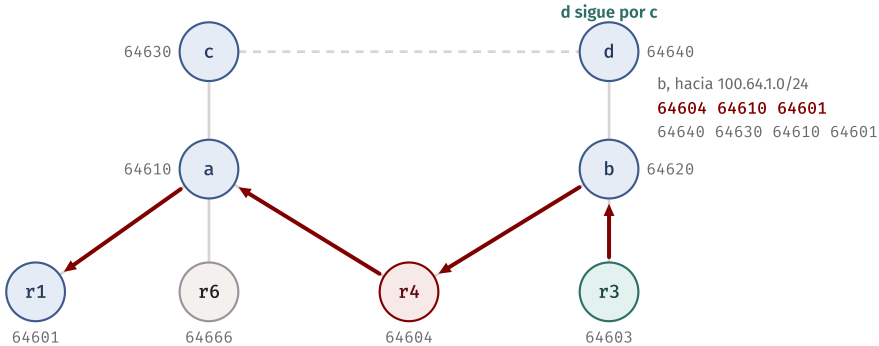
r4 pierde su regla de exportación: lo que aprende de un proveedor se lo pasa al otro. El camino tiene un valle.

# La fuga de r4, paso a paso




b compara hacia r1: 64604 64610 64601 son 3 ASes, contra 4 por d. Elige la fuga. a hace lo mismo en el otro sentido.

# La fuga de r4, paso a paso




d no cambia (3 contra 4). Todo lo que va entre las dos mitades pasa ahora por r4, un cliente chico.

# `python3 scenarios.py leak`

Antes, `r4` le anunciaba a `b` solo su `100.64.4.0/24`.

después: todo

```
     Network          Next Hop            Metric LocPrf Weight Path
 *>  100.64.1.0/24    10.207.6.2                             0 64610 64601 i
 *>  100.64.3.0/24    10.207.7.2                             0 64620 64603 i
 *>  100.64.4.0/24    0.0.0.0                  0         32768 i
 *>  100.64.66.0/24   10.207.6.2                             0 64610 64666 i
```

y b la elige: es más corta

```
$ docker exec clab-w07-bgp-b vtysh -c "show bgp ipv4 unicast 100.64.1.0/24"
  64604 64610 64601
      Origin IGP, valid, external, best (AS Path)
  64640 64630 64610 64601
      Origin IGP, valid, external
```

# El cliente chico se volvió tránsito

el salto 3 es r4, con 5 ms y 20 Mbit/s por enlace (slow_r4)

```
$ docker exec clab-w07-bgp-srv traceroute -n -w 1 100.64.1.10
 1  100.64.3.1  0.003 ms  0.004 ms  0.002 ms
 2  10.207.8.2  0.002 ms  0.002 ms  0.002 ms
 3  10.207.7.1  5.798 ms  6.708 ms  5.994 ms
 4  10.207.6.2  12.010 ms  12.706 ms  11.089 ms
 5  10.207.4.1  11.884 ms  12.185 ms  14.302 ms
 6  100.64.1.10  12.932 ms  14.381 ms  12.146 ms
```

| | Sin fuga | Con fuga |
|---|---|---|
| RTT `srv` a `h1` | 0,09 ms | 12 ms |
| Throughput, 3 s de `iperf3` | 4,57 Gbit/s | 18,9 Mbit/s |

Los 4,57 Gbit/s dependen de la computadora. Lo que importa es la tendencia. Con política real es peor: `b` ve a `r4` como cliente y lo prefiere, aunque el camino sea más largo.

# Caso real: la fuga del 24 de junio de 2019

Análisis publicado por Cloudflare ese día. Empezó a las 10:30 UTC.

1. **AS33154 (DQE)** usaba un "optimizador de BGP" que divide prefijos: `104.20.0.0/20` pasó a `104.20.0.0/21` y `104.20.8.0/21`.
2. DQE se los anunció a su cliente, **AS396531 (Allegheny)**.
3. Allegheny los pasó a su otro proveedor, **AS701 (Verizon)**.
4. Verizon se los anunció al resto de Internet.

# Caso real: la fuga del 24 de junio de 2019

Análisis publicado por Cloudflare ese día. Empezó a las 10:30 UTC.

1. **AS33154 (DQE)** usaba un "optimizador de BGP" que divide prefijos: `104.20.0.0/20` pasó a `104.20.0.0/21` y `104.20.8.0/21`.
2. DQE se los anunció a su cliente, **AS396531 (Allegheny)**.
3. Allegheny los pasó a su otro proveedor, **AS701 (Verizon)**.
4. Verizon se los anunció al resto de Internet.

**Dos ingredientes**

El viraje: como nuestro `r4`. Los más específicos: no compiten por política. Por eso fue tan grave.

**Impacto**

Afectó a servicios de Cloudflare, Amazon y Linode. Cloudflare perdió cerca del 15 % de su tráfico global en el peor momento.

SECCIÓN 04

# Filtros y límites

04

# Qué recomienda el RFC 7454

| Sesión | Filtro de entrada |
|---|---|
| Cliente | Aceptar solo los prefijos verificados del cliente; descartar el resto. |
| Peer | Estricto, contra registros de routing; o laxo: nada no enrutable globalmente, demasiado específico, propio, de la LAN del IXP ni ruta por defecto. |
| Proveedor | Si se pide la tabla completa, lo mismo que con un peer, sin el filtro de la ruta por defecto. |

# Qué recomienda el RFC 7454

| Sesión | Filtro de entrada |
|---|---|
| Cliente | Aceptar solo los prefijos verificados del cliente; descartar el resto. |
| Peer | Estricto, contra registros de routing; o laxo: nada no enrutable globalmente, demasiado específico, propio, de la LAN del IXP ni ruta por defecto. |
| Proveedor | Si se pide la tabla completa, lo mismo que con un peer, sin el filtro de la ruta por defecto. |

**El punto más poderoso es el cliente**

Un proveedor **sabe qué bloques tiene su cliente**: está en el contrato. PCCW en 2008 y Verizon en 2019 estaban en ese lugar.

# Cómo se lee una `prefix-list`

| **Línea** | `198.51.100.0/24` | `198.51.100.0/25` | `198.51.0.0/16` |
|---|---|---|---|
| `permit 198.51.100.0/24` | sí | no | no |
| `permit 198.51.100.0/24 le 25` | sí | sí | no |
| `permit 198.51.100.0/24 ge 25` | no | sí | no |
| `permit 198.51.0.0/16 le 32` | sí | sí | sí |

- Sin `ge` ni `le`, coincide solo con ese largo exacto.
- **Termina en un deny implícito**: lo que no coincide con ninguna línea se descarta.

**Error común**

Poner `le 32` “para no molestar al cliente”. Abre la puerta a los más específicos, que son justamente lo que hizo tanto daño en 2019.

# El filtro lo escriben ustedes; aquí el resultado

Con `lab/filters-a.conf` y `lab/filters-b.conf` completos, el ataque y la fuga juntos:

r6 anuncia lo que quiere; a se queda solo con lo legítimo

```
$ python3 scenarios.py same
$ python3 scenarios.py leak
$ docker exec clab-w07-bgp-a vtysh -c "show bgp ipv4 unicast neighbors 10.207.5.1 routes"
     Network          Next Hop            Metric LocPrf Weight Path
 *>  100.64.66.0/24   10.207.5.1               0             0 64666 i
$ docker exec clab-w07-bgp-a vtysh -c "show bgp ipv4 unicast"
     Network          Next Hop            Metric LocPrf Weight Path
 *>  100.64.1.0/24    10.207.4.1               0             0 64601 i
 *>  100.64.3.0/24    10.207.2.2                             0 64630 64640 64620 64603 i
 *>  100.64.4.0/24    10.207.6.1               0             0 64604 i
 *>  100.64.66.0/24   10.207.5.1               0             0 64666 i
$ docker exec clab-w07-bgp-h1 curl -s http://100.64.3.10:8080/
servidor legitimo de AS 64603
```

# Límite de prefijos: `maximum-prefix`

En b: `neighbor 10.207.7.1 maximum-prefix 2`. No depende de saber qué prefijos son, solo cuántos (RFC 7454, sec. 8). Si se excede, la sesión se corta.

la fuga con el límite puesto

```
$ docker exec clab-w07-bgp-b vtysh -c "show bgp summary"
Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.207.7.1      4      64604        83       110        0    0    0 00:00:05 Idle (PfxCt)        0 r4 (cliente)
$ docker exec clab-w07-bgp-a vtysh -c "show bgp summary"
Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.207.6.1      4      64604        93       121       29    0    0 00:00:14            1        4 r4 (cliente)
```

# Límite de prefijos: `maximum-prefix`

En `b`: `neighbor 10.207.7.1 maximum-prefix 2`. No depende de saber qué prefijos son, solo cuántos (RFC 7454, sec. 8). Si se excede, la sesión se corta.

la fuga con el límite puesto

```
$ docker exec clab-w07-bgp-b vtysh -c "show bgp summary"
Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.207.7.1      4      64604        83       110        0    0    0 00:00:05 Idle (PfxCt)        0 r4 (cliente)
$ docker exec clab-w07-bgp-a vtysh -c "show bgp summary"
Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.207.6.1      4      64604        93       121       29    0    0 00:00:14            1        4 r4 (cliente)
```

- `a`, con el mismo límite, no cortó: solo le llegaron dos. **Detecta fugas grandes**.
- `r4` se quedó con un proveedor, y esa sesión no vuelve sola.
- Filtros y límites protegen a **los demás**, no a quien los configura: ese es el problema de incentivos.

SECCIÓN 05

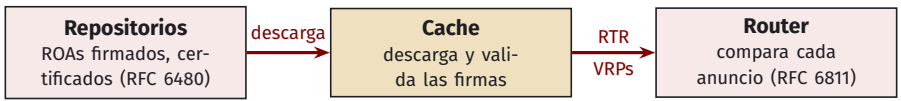
# RPKI

05



# RPKI: las piezas

¿El AS 64666 tiene derecho a originar `100.64.3.0/24`? Las direcciones se asignan en jerarquía: se puede certificar.




**ROA (Route Origin Authorization)**

Autorización firmada por quien tiene el prefijo. Contiene **un** AS y una lista de prefijos, cada uno con un largo máximo opcional.

*"El owner de `100.64.3.0/24` autoriza al AS 64603 a originarlo, hasta `/24`."*

# Los tres resultados (RFC 6811)

Para una ruta, se toma el prefijo y el **origen**: el último AS del `AS_PATH`.

- Un VRP **cubre** la ruta: su prefijo es igual o más corto, y los bits coinciden.
- Un VRP **hace match**: la cubre, el largo no pasa del máximo y el origen es su AS.

| | |
|---|---|
| **valid** | Al menos un VRP hace match. |
| **invalid** | Al menos un VRP cubre la ruta, y ninguno hace match. |
| **not found** | Ningún VRP cubre la ruta. |

# Los tres resultados (RFC 6811)

Para una ruta, se toma el prefijo y el **origen**: el último AS del `AS_PATH`.

- Un VRP **cubre** la ruta: su prefijo es igual o más corto, y los bits coinciden.
- Un VRP **hace match**: la cubre, el largo no pasa del máximo y el origen es su AS.

| | |
|---|---|
| **valid** | Al menos un VRP hace match. |
| **invalid** | Al menos un VRP cubre la ruta, y ninguno hace match. |
| **not found** | Ningún VRP cubre la ruta. |

**Lo que no dicen**

Invalid no significa ataque: puede ser un ROA mal hecho. Not found no es sospechoso: nadie publicó un ROA. Valid no significa que el camino sea bueno.

# Ejemplos: un ROA con largo máximo 24, AS 64600

```
./rov vrps-example.txt prefijo as

$ ./rov vrps-example.txt 198.51.100.0/24 64600
  cubierto por 198.51.100.0/24 max 24 AS 64600: largo ok, origen ok
valid
$ ./rov vrps-example.txt 198.51.100.0/24 64666
  cubierto por 198.51.100.0/24 max 24 AS 64600: largo ok, origen no coincide
invalid
$ ./rov vrps-example.txt 198.51.100.0/25 64600
  cubierto por 198.51.100.0/24 max 24 AS 64600: largo demasiado largo, origen ok
invalid
$ ./rov vrps-example.txt 198.51.0.0/16 64600
not found
$ ./rov vrps-example.txt 198.51.101.0/24 64600
not found
```

- Tercero: el AS correcto, y aun así invalid: el freno a los más específicos. Cuarto: el `/16` es más corto que el ROA, y nadie lo cubre.

# RPKI en el laboratorio

**Qué es real y qué no**

Real: routers, RTR, clasificación y descarte. No real: `rtr_cache.py` es nuestro, lee una lista fija de VRPs y no comprueba firmas.

a conectado al cache

```
$ docker exec clab-w07-bgp-a vtysh -c "show rpki prefix-table"
Prefix                                  Prefix Length  Origin-AS
100.64.3.0                                 24 -  24   64603
100.64.1.0                                 24 -  24   64601
$ docker exec clab-w07-bgp-a vtysh -c "show bgp ipv4 unicast"
     Network          Next Hop            Metric LocPrf Weight Path
V*>  100.64.1.0/24    10.207.4.1               0             0 64601 i
V*>  100.64.3.0/24    10.207.2.2                             0 64630 64640 64620 64603 i
N*>  100.64.4.0/24    10.207.6.1               0             0 64604 i
N*>  100.64.66.0/24   10.207.5.1               0             0 64666 i
```

# Marcar no es descartar

con el hijack, sin una regla

```
$ docker exec clab-w07-bgp-a vtysh -c "show bgp ipv4 unicast"
     Network          Next Hop            Metric LocPrf Weight Path
V*>  100.64.1.0/24    10.207.4.1               0             0 64601 i
I*>  100.64.3.0/24    10.207.5.1               0             0 64666 i
N*>  100.64.4.0/24    10.207.6.1               0             0 64604 i
N*>  100.64.66.0/24   10.207.5.1               0             0 64666 i
$ docker exec clab-w07-bgp-h1 curl -s http://100.64.3.10:8080/
ATACANTE: AS 64666 recibio esta conexion
```

**La `I` está ahí, y el tráfico sigue yendo al atacante**

RFC 6811: una implementación no excluye una ruta por su estado a menos que se configure explícitamente. Es política local.

# La regla: Route Origin Validation (ROV)

```
{en a}
route-map ROV deny 10
 match rpki invalid
route-map ROV permit 20
router bgp 64610
 address-family ipv4 unicast
  neighbor 10.207.5.1 route-map ROV in
```

- Sin el `permit 20`, se pierde también lo válido y lo que no tiene ROA.
- El not found debe pasar, o caen las redes sin ROA.

con rov_a: queda la ruta legítima

```
$ docker exec clab-w07-bgp-h1 curl -s http://100.64.3.10:8080/
servidor legitimo de AS 64603
```

# Lo que no protege: el origen forjado

RFC 6811, sec. 6: ROV no valida el camino; *al atacante solo le hace falta poner el AS “válido”.*

r6 agrega 64603 al AS_PATH, con ROV en a

```
$ docker exec clab-w07-bgp-a vtysh -c "show bgp ipv4 unicast 100.64.3.0/24"
  64666 64603
      Origin IGP, metric 0, valid, external, best (AS Path), rpki validation-state: valid
  64630 64640 64620 64603
      Origin IGP, valid, external, rpki validation-state: valid
$ docker exec clab-w07-bgp-h1 curl -s http://100.64.3.10:8080/
ATACANTE: AS 64666 recibio esta conexion
```

# Lo que no protege: el origen forjado

RFC 6811, sec. 6: ROV no valida el camino; *al atacante solo le hace falta poner el AS "válido"*.

r6 agrega 64603 al AS_PATH, con ROV en a

```
$ docker exec clab-w07-bgp-a vtysh -c "show bgp ipv4 unicast 100.64.3.0/24"
  64666 64603
      Origin IGP, metric 0, valid, external, best (AS Path), rpki validation-state: valid
  64630 64640 64620 64603
      Origin IGP, valid, external, rpki validation-state: valid
$ docker exec clab-w07-bgp-h1 curl -s http://100.64.3.10:8080/
ATACANTE: AS 64666 recibio esta conexion
```

**El propio RFC lo dice**

Hay que verlo como *una protección contra errores de configuración*, más que como "seguridad" en sentido fuerte.

# Lo que decide el largo máximo

origen forjado, /25, max 24

```
$ docker exec clab-w07-bgp-a vtysh \
  -c "show bgp ipv4 unicast 100.64.3.0/25"
% Network not in table
$ docker exec clab-w07-bgp-h1 curl ...
servidor legitimo de AS 64603
```

origen forjado, /25, max 25

```
$ docker exec clab-w07-bgp-a vtysh \
  -c "show bgp ipv4 unicast 100.64.3.0/25"
  64666 64603
    valid, rpki validation-state: valid
$ docker exec clab-w07-bgp-h1 curl ...
ATACANTE: AS 64666 recibio esta conexion
```

**Un ROA “por si acaso” le da espacio al atacante**

Un largo máximo holgado autoriza /25 que nadie anuncia. El RFC 9319 recomienda evitar el largo máximo salvo en casos específicos.

La salida de la derecha está resumida: en el laboratorio sale la entrada completa.

# Tampoco: adopción parcial y fugas

### Si solo `d` valida

El `/25` pasa `a` y `c`, y se detiene en `d`.

rov_d, specific

```
$ ... r1 vtysh -c "show ip route 100.64.3.10"
Routing entry for 100.64.3.0/25
$ ... b vtysh -c "show ip route 100.64.3.10"
Routing entry for 100.64.3.0/24
$ ... h1 curl -s http://100.64.3.10:8080/
ATACANTE: AS 64666 recibio esta conexion
```

La víctima publicó su ROA, y `h1` cae igual.

### La fuga conserva el origen

La mejor ruta tiene un valle, y es valid.

rov_a, rov_d, leak

```
     Network          Next Hop    Path
V*>  100.64.1.0/24    10.207.4.1  64601 i
V*>  100.64.3.0/24    10.207.6.1  64604 64620 64603 i
V*                    10.207.2.2  64630 64640 64620 64603 i
N*>  100.64.4.0/24    10.207.6.1  64604 i
N*>  100.64.66.0/24   10.207.5.1  64666 i
```

Salidas resumidas.

# ¿La detiene ROV?

| Situación | ¿La detiene? |
|---|---|
| Origen ajeno, mismo prefijo, con ROA del owner | Sí, si quien la recibe descarta |
| Más específico con origen ajeno, con ROA del owner | Sí |
| Prefijo sin ROA | No: not found |
| Origen forjado, dentro del largo máximo | No |
| Origen forjado, más largo que el máximo | Sí |
| Fuga con el mismo largo | No: el origen no cambió |
| Fuga con más específicos, más largos que el máximo | Sí, por el largo |

**Error común**

Creer que RPKI cifra BGP o evita todo hijack. No cifra nada, y un ROA es un objeto firmado y público. Solo ayuda a validar quién puede originar, y solo donde se aplica.

# ¿Y en los casos reales?

**YouTube, 2008**

RPKI se publicó en 2012 (RFC 6480): no era una opción. *Con* un ROA de YouTube para el `/22`, el `/24` de AS17557 habría sido invalid: por origen y por largo.

**La fuga de 2019**

Cloudflare firmaba con largo máximo 20: sus `/21` habrían sido invalid. Quien hace el trabajo es el **largo**, no el camino. Sin los más específicos, habría sido valid, como en el laboratorio.

**Más allá del origen.** BGPsec (RFC 8205): cada AS del camino firma. RFC 9234: los vecinos declaran su relación en el OPEN y marcan rutas en el UPDATE.

SECCIÓN 06

# Facebook, 2021

06

# Facebook, 4 de octubre de 2021: en orden

1. Un **comando de mantenimiento**, para evaluar la capacidad del backbone, lo dio de baja.
2. Un **bug en la herramienta de auditoría** dejó pasar el comando.
3. Los servidores DNS **retiraron sus propios anuncios de BGP**: si no alcanzan a los data centers, se declaran enfermos.
4. Quedaron funcionando, y nadie los alcanzaba. Los resolvers daban `SERVFAIL`.
5. Sin red ni DNS, tampoco había acceso remoto ni herramientas: hubo que ir al data center.
6. Se restauró de a pocos: la carga de golpe podía causar otra falla.

Desde afuera, según Cloudflare: pico de cambios de AS32934 cerca de las 15:40 UTC; prefijos DNS ausentes a las 15:58 UTC; `facebook.com` de vuelta a las 21:20 UTC. Fuentes: informe de ingeniería de Facebook y análisis de Cloudflare.

# Facebook, 4 de octubre de 2021: en orden

1. Un **comando de mantenimiento**, para evaluar la capacidad del backbone, lo dio de baja.
2. Un **bug en la herramienta de auditoría** dejó pasar el comando.
3. Los servidores DNS **retiraron sus propios anuncios de BGP**: si no alcanzan a los data centers, se declaran enfermos.
4. Quedaron funcionando, y nadie los alcanzaba. Los resolvers daban `SERVFAIL`.
5. Sin red ni DNS, tampoco había acceso remoto ni herramientas: hubo que ir al data center.
6. Se restauró de a pocos: la carga de golpe podía causar otra falla.

Desde afuera, según Cloudflare: pico de cambios de AS32934 cerca de las 15:40 UTC; prefijos DNS ausentes a las 15:58 UTC; `facebook.com` de vuelta a las 21:20 UTC. Fuentes: informe de ingeniería de Facebook y análisis de Cloudflare.

# BGP no se equivocó

**Ningún filtro ni RPKI habrían ayudado**

No hubo anuncio falso, ni fuga, ni hijack. Los prefijos se retiraron de manera legítima, siguiendo una regla razonable, en el peor momento posible.

Las lecciones son de otra clase:

- **Dependencias circulares.** Las herramientas para arreglar la red dependen de la red.
- **Una protección puede ampliar la falla.** Retirar un servidor enfermo está bien. Si todos deciden lo mismo a la vez, desaparece el servicio.
- **Un error local tiene alcance global**, en minutos.

SECCIÓN 07

# La estructura real de Internet

07

# De la jerarquía a una red más plana

- **Tier 1:** sin proveedores, peering con los demás Tier 1. Unos 20, según CS168.
- La mayoría son **stubs**: solo sus propios hosts.

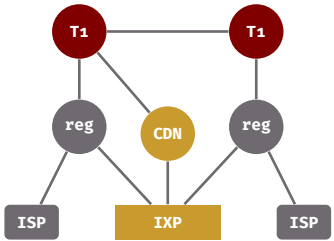
# De la jerarquía a una red más plana

- **Tier 1:** sin proveedores, peering con los demás Tier 1. Unos 20, según CS168.
- La mayoría son **stubs**: solo sus propios hosts.

Dos fuerzas lo aplanan:

- **Peering** entre redes del mismo nivel.
- **Redes de contenido**: llegan directo a muchos ISPs.

Semana 1: `www.ox.ac.uk` respondió en 55 ms, desde `cdn.cloudflare.net`. Oxford exigiría más de 100 ms.



# Tránsito y peering: un negocio

| | **Tránsito** | **Peering** |
|---|---|---|
| ¿Qué se compra? | Acceso a toda Internet | La red del otro y sus clientes |
| ¿Quién paga? | El cliente, a su proveedor | Normalmente nadie al otro |
| ¿Cómo? | Por volumen, normalmente | Equipo, puerto y conexión al lugar |

# Tránsito y peering: un negocio

| | Tránsito | Peering |
|---|---|---|
| ¿Qué se compra? | Acceso a toda Internet | La red del otro y sus clientes |
| ¿Quién paga? | El cliente, a su proveedor | Normalmente nadie al otro |
| ¿Cómo? | Por volumen, normalmente | Equipo, puerto y conexión al lugar |

**Con números inventados: solo mostramos la cuenta**

Tránsito a 1 dólar por Mbps al mes. Conectarse a un punto común: 1200 dólares al mes.
**3000 Mbps hacia otro ISP:** 3000 de tránsito contra 1200: ahorra 1800.
**500 Mbps:** 500 de tránsito contra 1200: no le conviene, salvo que el punto común dé acceso a muchas otras redes.

**Ese es el motivo de los IXPs:** un solo puerto, para hablar con muchos.

# Qué es un IXP

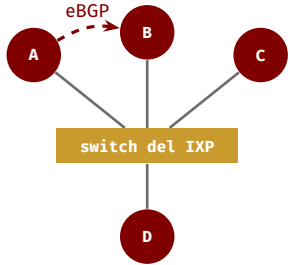
- Un **switch grande** en un edificio. Cada red trae un router.
- Quedan en una misma LAN, y cada par abre su sesión eBGP.
- **No** es un router, no decide caminos, no tiene política.

# Qué es un IXP

- Un **switch grande** en un edificio. Cada red trae un router.
- Quedan en una misma LAN, y cada par abre su sesión eBGP.
- **No** es un router, no decide caminos, no tiene política.

**Route server**

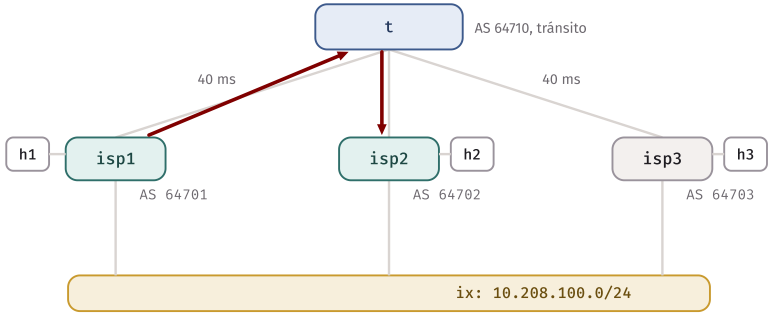
Con N redes, $N(N-1)/2$ sesiones: para 100, son 4950. Un route server reenvía los anuncios, y cada red abre una sola sesión. No está en el camino de los paquetes.



# Lo medimos: `code/week-07/ixp/`

- Tres ISPs en una ciudad, un proveedor de tránsito lejos y un IXP local al que los tres ya llegan.
- El proveedor está lejos: 40 ms por sentido, como el cable a Miami.
- Es un retardo que elegimos con `tc netem`.
- El IXP ya está, pero al comienzo nadie tiene sesiones.

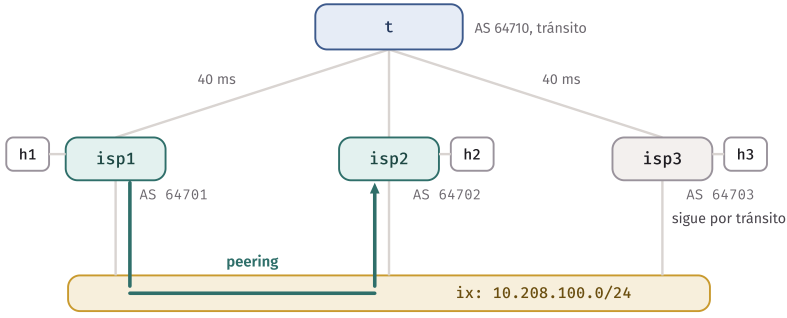
# El peering cambia el camino




| | | | |
|---|---|---|---|
| RTT h1 a h2 | 163 ms | por el enlace pagado | 5 484 023 bytes |
| saltos | 4 | de una transferencia de 5 MiB | |

Sin peering, h1 llega a h2 por el proveedor: 40 ms por sentido en cada enlace, y todo pasa por el enlace pagado.

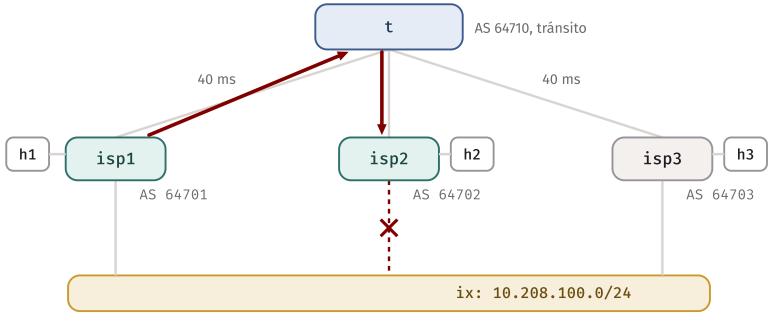
# El peering cambia el camino




| | | | |
|---|---|---|---|
| RTT h1 a h2 | menos de 1 ms | por el enlace pagado | 151 bytes |
| saltos | 3 | los mensajes de BGP con el proveedor | |

Con una sesión eBGP por la LAN del IXP, el camino queda en la ciudad. El tránsito sigue allí, sin usarse.

# El peering cambia el camino




RTT h1 a h2 163 ms

el enlace de isp1 sigue arriba: no se da cuenta enseguida

isp2 se desconecta del switch. Vence el hold timer, de 9 s, y el tráfico vuelve solo al camino de tránsito.

# Antes y después del peering

sin peering

```
$ docker exec clab-w07-ixp-h1 \
    traceroute -n -w 1 100.64.72.10
 1  100.64.71.1    0.003 ms
 2  10.208.1.2    81.855 ms
 3  10.208.2.1   162.883 ms
 4  100.64.72.10 163.813 ms
```

sh ixp/peering.sh

```
$ docker exec clab-w07-ixp-h1 \
    traceroute -n -w 1 100.64.72.10
 1  100.64.71.1   0.005 ms
 2  10.208.100.2  0.002 ms
 3  100.64.72.10  0.002 ms
```

| | **Sin peering** | **Con peering** |
|---|---|---|
| RTT `h1` a `h2` | 163 ms | menos de 1 ms |
| Bytes de 5 MiB por el enlace pagado | 5 484 023 | 151 |

Salidas resumidas a un tiempo por salto. Los 151 bytes son BGP con el proveedor. `isp3`, sin peering, sigue en 164 ms. **Baja el retardo porque cambia el camino:** no hace el enlace más rápido; quita distancia.

# En el Perú: las trazas de la semana 1

hacia la UNI: se queda cerca

```
$ traceroute -n -q 3 -w 2 -m 22 www.uni.edu.pe
 5  181.177.224.6  5.601 ms  5.040 ms  6.637 ms
 7  45.183.47.179  5.896 ms  4.633 ms  6.118 ms
```

hacia Sudáfrica: dos saltos grandes

```
$ traceroute -q 3 -w 2 -m 14 www.uct.ac.za
 6  8.243.120.5 (8.243.120.5)  10.167 ms  6.497 ms  6.092 ms
 8  4.68.38.70 (4.68.38.70)  82.491 ms  83.350 ms  82.815 ms
10  et-0-0-1-66.rt04.bb.ampath.net (170.39.8.30)  82.700 ms  82.417 ms  82.540 ms
11  187-185-103-190.ampath.net (190.103.185.187)  241.758 ms  241.256 ms  241.834 ms
```

Se omiten algunos saltos. El RTT sube y **se queda** alto: ahí hay distancia real, y un cable. La física da 42 ms hasta Miami; medimos 76 de aumento.

# Por qué importa en un país como el Perú

- **NAP Perú:** según su sitio, agrupa a los principales operadores y proveedores de acceso a Internet del país.
- **Cables:** según Submarine Networks, Lurín es la puerta de entrada internacional; SAm-1 tiene otra estación en Máncora. Según Telxius, el Mistral, de unos 7300 km entre Guatemala y Chile, también aterriza en Lurín.

| | |
|---|---|
| **Retardo** | Física. Solo baja trayendo el contenido cerca y evitando que el tráfico local salga. |
| **Costo** | Cada mega que se queda en el país deja de pagar tránsito internacional. |
| **Resiliencia** | Pocos puntos físicos de salida son pocos puntos de falla. |

SECCIÓN 08

# Repaso para el parcial

08

# Repaso: retardos y CIDR

**Semana 1**

1500 bytes, 100 Mbps, 4200 km a $2 \times 10^8$ m/s. ¿Total? ¿Y a 1 Gbps?

# Repaso: retardos y CIDR

**Semana 1**

1500 bytes, 100 Mbps, 4200 km a $2 \times 10^8$ m/s. ¿Total? ¿Y a 1 Gbps?

**Respuesta**

0,12 ms + 21 ms = 21,12 ms. A 1 Gbps: 21,012 ms. Manda la propagación.

# Repaso: retardos y CIDR

**Semana 1**

1500 bytes, 100 Mbps, 4200 km a $2 \times 10^8$ m/s. ¿Total? ¿Y a 1 Gbps?

**Respuesta**

0,12 ms + 21 ms = 21,12 ms. A 1 Gbps: 21,012 ms. Manda la propagación.

**Semana 4**

Rutas: `10.207.0.0/16`→A, `10.207.16.0/24`→B, `10.207.16.128/25`→C, `0.0.0.0/0`→D. ¿Por dónde salen `10.207.16.200`, `10.207.17.1` y `10.208.0.1`?

# Repaso: retardos y CIDR

**Semana 1**

1500 bytes, 100 Mbps, 4200 km a $2 \times 10^8$ m/s. ¿Total? ¿Y a 1 Gbps?

**Respuesta**

0,12 ms + 21 ms = 21,12 ms. A 1 Gbps: 21,012 ms. Manda la propagación.

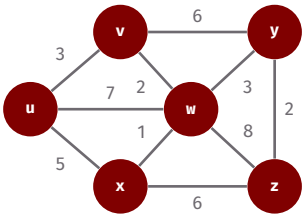
**Semana 4**

Rutas: `10.207.0.0/16`→A, `10.207.16.0/24`→B, `10.207.16.128/25`→C, `0.0.0.0/0`→D. ¿Por dónde salen `10.207.16.200`, `10.207.17.1` y `10.208.0.1`?

**Respuesta**

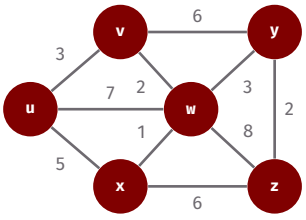
C (coincide con A, B y C; gana el más largo). A. D.

# Repaso: Dijkstra desde u



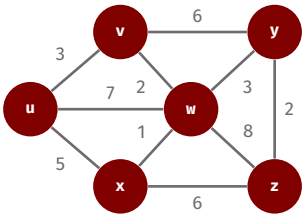

| Agrega | v | w | x | y | z |
|---|---|---|---|---|---|
| u | 3,u | 7,u | 5,u | $\infty$ | $\infty$ |

# Repaso: Dijkstra desde u



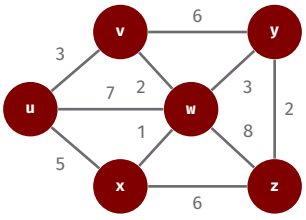

| Agrega | v | w | x | y | z |
|---|---|---|---|---|---|
| u | 3,u | 7,u | 5,u | $\infty$ | $\infty$ |
| v | | 5,v | 5,u | 9,v | $\infty$ |

# Repaso: Dijkstra desde u



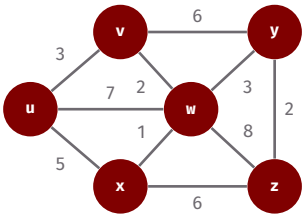

| Agrega | v | w | x | y | z |
|---|---|---|---|---|---|
| u | 3,u | 7,u | 5,u | $\infty$ | $\infty$ |
| v | | 5,v | 5,u | 9,v | $\infty$ |
| w | | | 5,u | 8,w | 13,w |

# Repaso: Dijkstra desde u



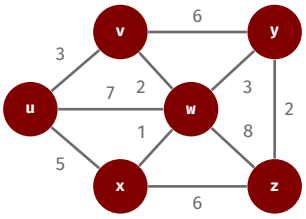

| Agrega | v | w | x | y | z |
|---|---|---|---|---|---|
| u | 3,u | 7,u | 5,u | ∞ | ∞ |
| v | | 5,v | 5,u | 9,v | ∞ |
| w | | | 5,u | 8,w | 13,w |
| x | | | | 8,w | 11,x |

# Repaso: Dijkstra desde u




| Agrega | v | w | x | y | z |
|---|---|---|---|---|---|
| u | 3,u | 7,u | 5,u | $\infty$ | $\infty$ |
| v | | 5,v | 5,u | 9,v | $\infty$ |
| w | | | 5,u | 8,w | 13,w |
| x | | | | 8,w | 11,x |
| y | | | | | 10,y |

# Repaso: Dijkstra desde u




| Agrega | v | w | x | y | z |
|---|---|---|---|---|---|
| u | 3,u | 7,u | 5,u | $\infty$ | $\infty$ |
| v | | 5,v | 5,u | 9,v | $\infty$ |
| w | | | 5,u | 8,w | 13,w |
| x | | | | 8,w | 11,x |
| y | | | | | 10,y |

El empate de `w` y `x` no cambia el resultado. Final: `v` 3, `w` 5, `x` 5, `y` 8, `z` 10, por `u-v-w-y-z`.

# Repaso: políticas de BGP

**Semanas 6 y 7**

P es proveedor de X e Y; X e Y son peers; X es proveedor de $C_1$ e Y de $C_2$. ¿Qué camino usa $C_1$ hacia $C_2$? ¿X le anuncia a P esa ruta?

# Repaso: políticas de BGP

**Semanas 6 y 7**

P es proveedor de X e Y; X e Y son peers; X es proveedor de $C_1$ e Y de $C_2$. ¿Qué camino usa $C_1$ hacia $C_2$? ¿X le anuncia a P esa ruta?

**Respuesta**

$C_1$ usa X Y $C_2$: el peer gana al proveedor. X no se la anuncia a P: lo aprendido de un peer solo va a clientes. Si lo hiciera, sería un route leak.

# Repaso: leer una salida

**Leer una salida, de a con ROV y una fuga**

```
V*>  100.64.3.0/24  10.207.6.1  0 64604 64620 64603 i
V*                  10.207.2.2  0 64630 64640 64620 64603 i
```

¿Cuál se usa? ¿Quién la origina? ¿Por qué RPKI no ayuda?

# Repaso: leer una salida

**Leer una salida, de a con ROV y una fuga**

```
V*>  100.64.3.0/24  10.207.6.1  0 64604 64620 64603 i
V*                  10.207.2.2  0 64630 64640 64620 64603 i
```

¿Cuál se usa? ¿Quién la origina? ¿Por qué RPKI no ayuda?

**Respuesta**

La primera (>), por más corta. Las dos las origina 64603, y por eso son valid: RPKI no ve el valle.

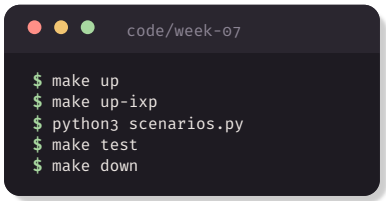
SECCIÓN 09

# Laboratorio

09

# Provocar un route hijack y defenderse

1. Línea base: `h1` (AS 64601) a `srv` (AS 64603).
2. `same`: ¿qué ASes cambiaron y por qué?
3. `specific`: el tráfico se desvía, y explicar por qué.
4. `leak`: seguir el tráfico, y medir.
5. **El filtro:** `lab/filters-a.conf` y `filters-b.conf`.
6. RPKI: `rov_a`, `rov_d`, `forged`, `rov_a_lax`. ¿Qué tendría que cambiar para que hubiera detenido cada ataque?
7. IXP: medir antes y después.
8. Avanzar con **P2**.




Predigan antes de ejecutar. Vuelvan al inicio con `reset`.

Las pruebas de los filtros se saltan hasta que quiten el `TODO`.

# Para la próxima clase

- Terminar el laboratorio, y seguir con **P2**: su red es otra, con varios routers e iBGP; este filtro no se copia.
- Examen parcial: semanas 1 a 7.
- Estudiar con las preguntas de repaso de cada guía y los ejercicios de estos apuntes.

**Pregunta**

Un compañero dice: “publicamos nuestro ROA, así que ya estamos protegidos contra hijacks”. ¿Qué le responden? Mencionen por lo menos dos situaciones del laboratorio en las que seguiría cayendo.