Seguridad, control de acceso y conformado a través de L3 VPNs

Departamento de Física Teórica
Universidad Autónoma de Madrid

Pablo Collado Soto & José del Peso Malagón - 18/10/2022

Nuestro laboratorio

Fuente

- Gestionamos un laboratorio con más de 50 máquinas:
  - Muchas de ellas prestan servicio al experimento ATLAS.
  - Otras son empleadas por usuarios para tareas de cómputo.
- Debemos garantizar el acceso tanto a usuarios como administradores.
- Todas las máquinas tienen `IPv4/IPv6` públicas…
  - Exponer puertos directamente nos parece arriesgado…
- ¿Solución?

¿Qué venimos haciendo?

Fuente

- No contamos con *firewalls* a «físicos».
- Todas nuestras máquinas emplean distribuciones GNU/Linux.
- Por tanto, aplicamos políticas de seguridad **en cada máquina**.
- Las políticas son, en esencia, reglas de `iptables(8)`.
- Esto plantea varios problemas:
    - ¿Cómo gestionamos las reglas de manera centralizada?
    - ¿Cómo permitimos a `IPv4/IPv6s` específicas conectarse desde fuera?

Espera ¿iptables?

Fuente

- A día de hoy nuestras máquinas siguen usando `iptables`.
- El proyecto *Netfilter* ha desarrollado una alternativa mejor: `nftables`.
- La sintaxis de `ntftables` **no** es compatible con `iptables`.
  - Sin embargo, `nftables` ofrece «capas» de retrocompatibilidad.
- En algún momento acometeremos esta migración…
- … pero esta charla se centra en cómo hacemos uso de `iptables`.
- Todos los ejemplos y conceptos son totalmente transferibles a `nftables`.

¿Qué proponemos?

Fuente

- En vez de diseminar las reglas de protección, centralicémoslas.
- Todas las conexiones de usuarios externas deben atravesar un punto.
- Es ahí donde aplicaremos las medidas y acciones oportunas.
- Conseguimos:
  - **Seguridad**: el tráfico del cliente a este punto va cifrado en capa 3.
  - **Control de acceso**: cada cliente aparece asociado a una `IPv4`.
  - **QoS**: podemos conformar el tráfico en este punto.

WireGuard VPN

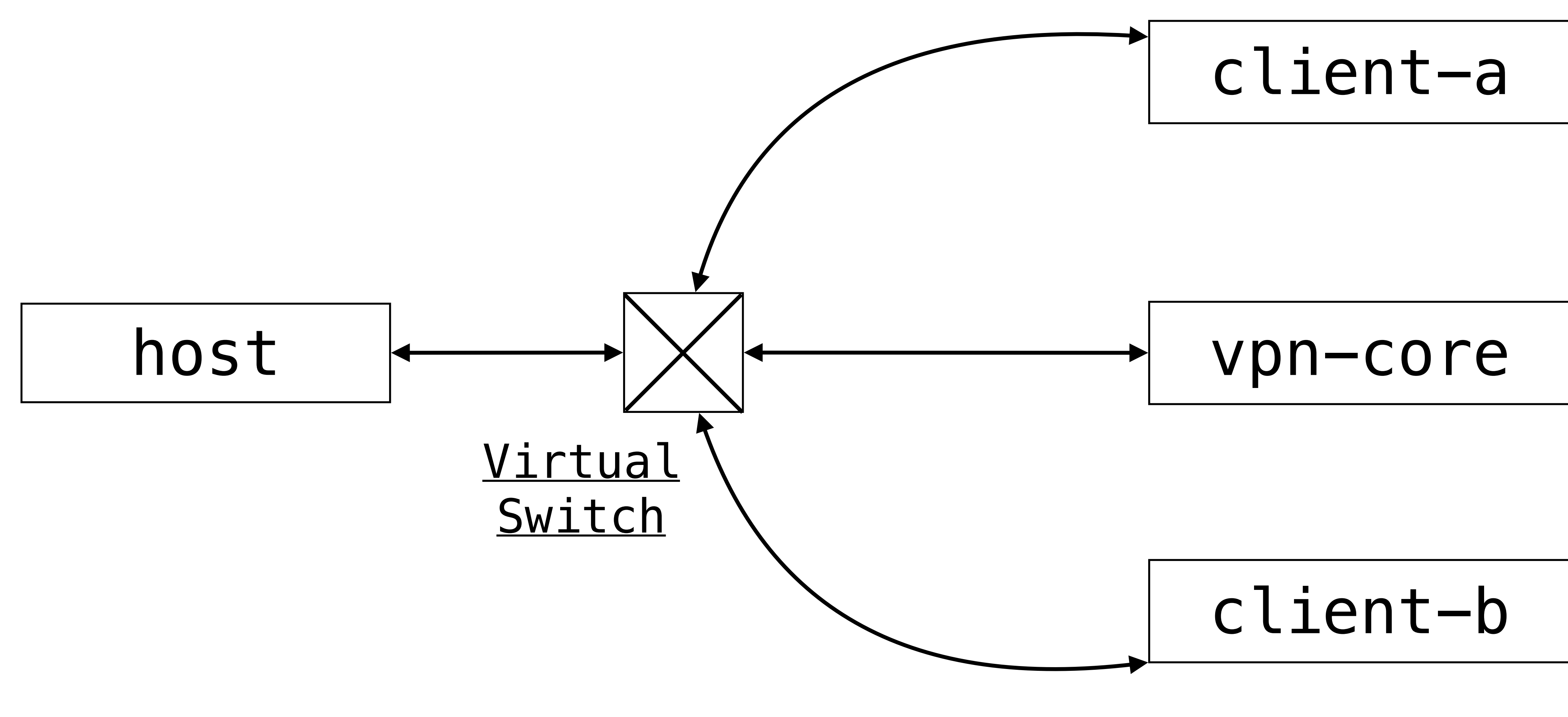
Fuente

- Para conseguir centralizar el tráfico usaremos una VPN.
- Nosotros empleamos `WireGuard` porque:
  - Se ejecuta en espacio de kernel, no de usuario: es muy rápido.
  - Se registra como una interfaz virtual: no requiere un *daemon*.
  - Se gestiona a través de `rtnetlink(7)` con `ip(8)` (i.e. `iproute2`).
  - Ofrece dos niveles de cifrado:
    - Cifrado asimétrico (Curve25519 ECDH): vulnerable al algoritmo de Shor.
    - Cifrado simétrico (ChaCha20Poly1305): resistente al algoritmo de Grover.

Antes de seguir… Vagrant

Fuente

- Llegados a este punto ya podemos discutir topologías **lógicas**.
- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.
- Hemos escogido Vagrant para levantarla a través de VirtualBox.
- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en pcolladosoto/wg-ebpf.



Antes de seguir… Vagrant

HashiCorp Vagrant

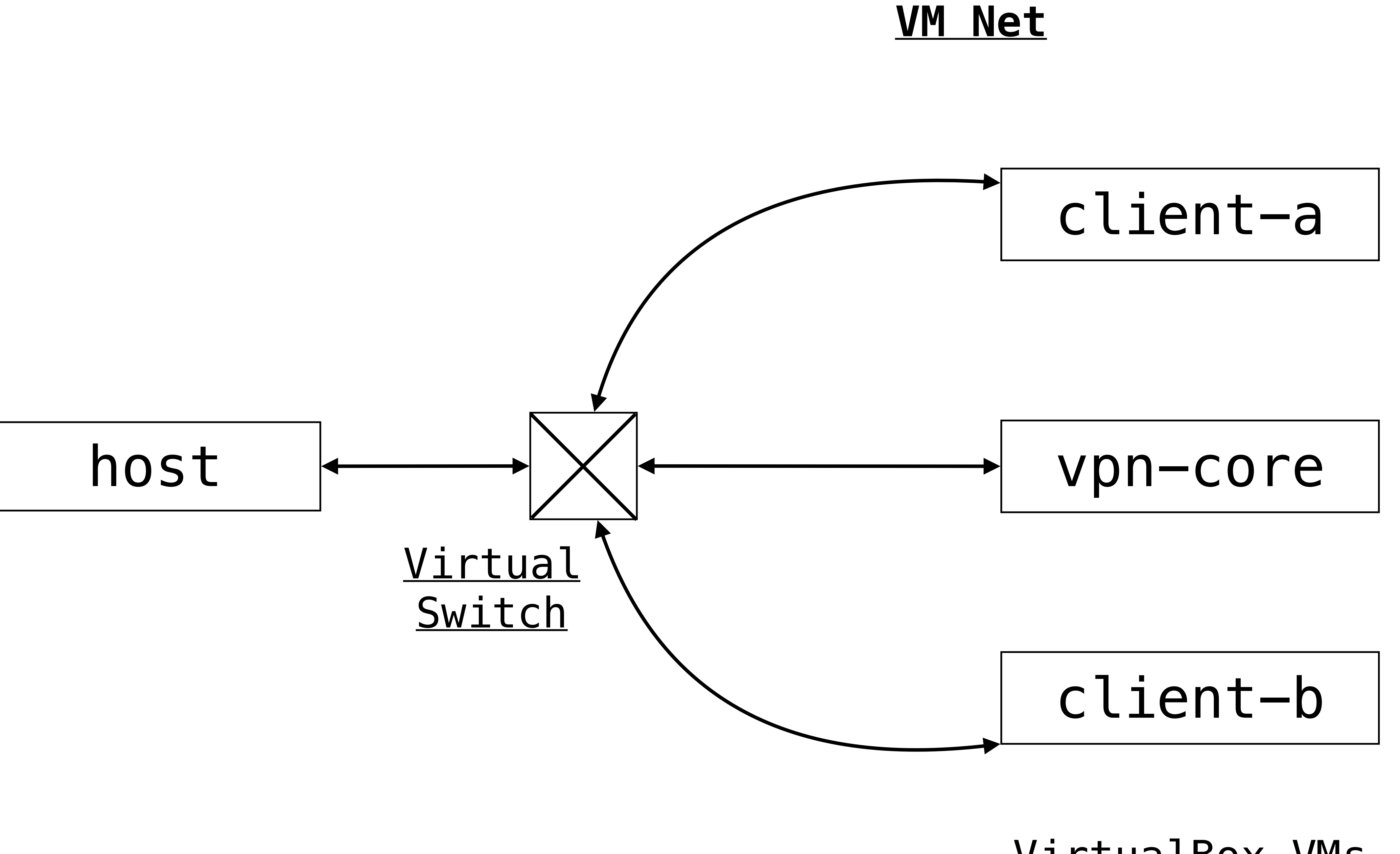
Fuente

- Llegados a este punto ya podemos discutir topologías **lógicas**.

- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.

- Hemos escogido Vagrant para levantarla a través de VirtualBox.

- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en pcolladosoto/wg-ebpf.

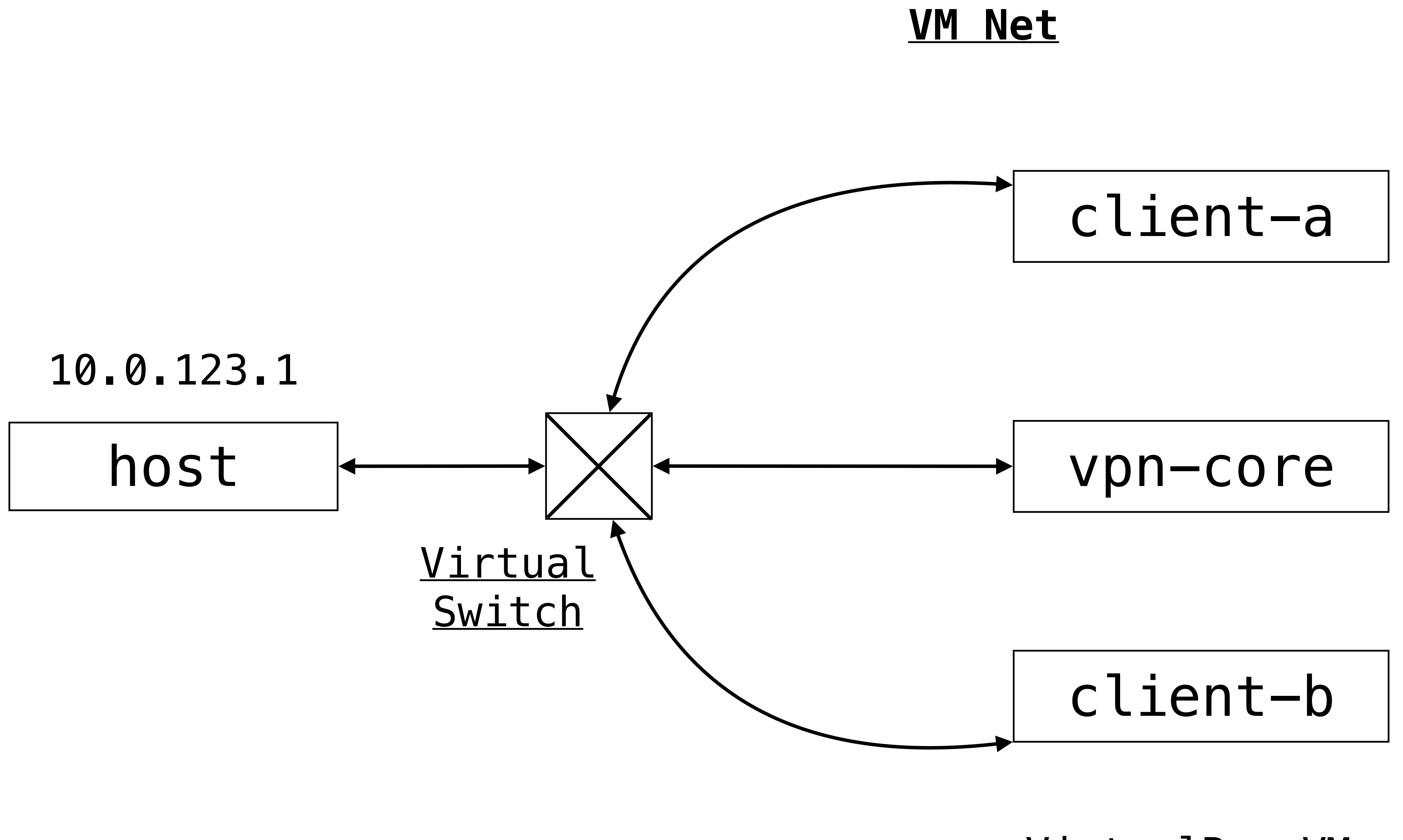


Antes de seguir… Vagrant

HashiCorp Vagrant

[Fuente]

- Llegados a este punto ya podemos discutir topologías **lógicas**.
- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.
- Hemos escogido Vagrant para levantarla a través de VirtualBox.
- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en pcolladosoto/wg-ebpf.

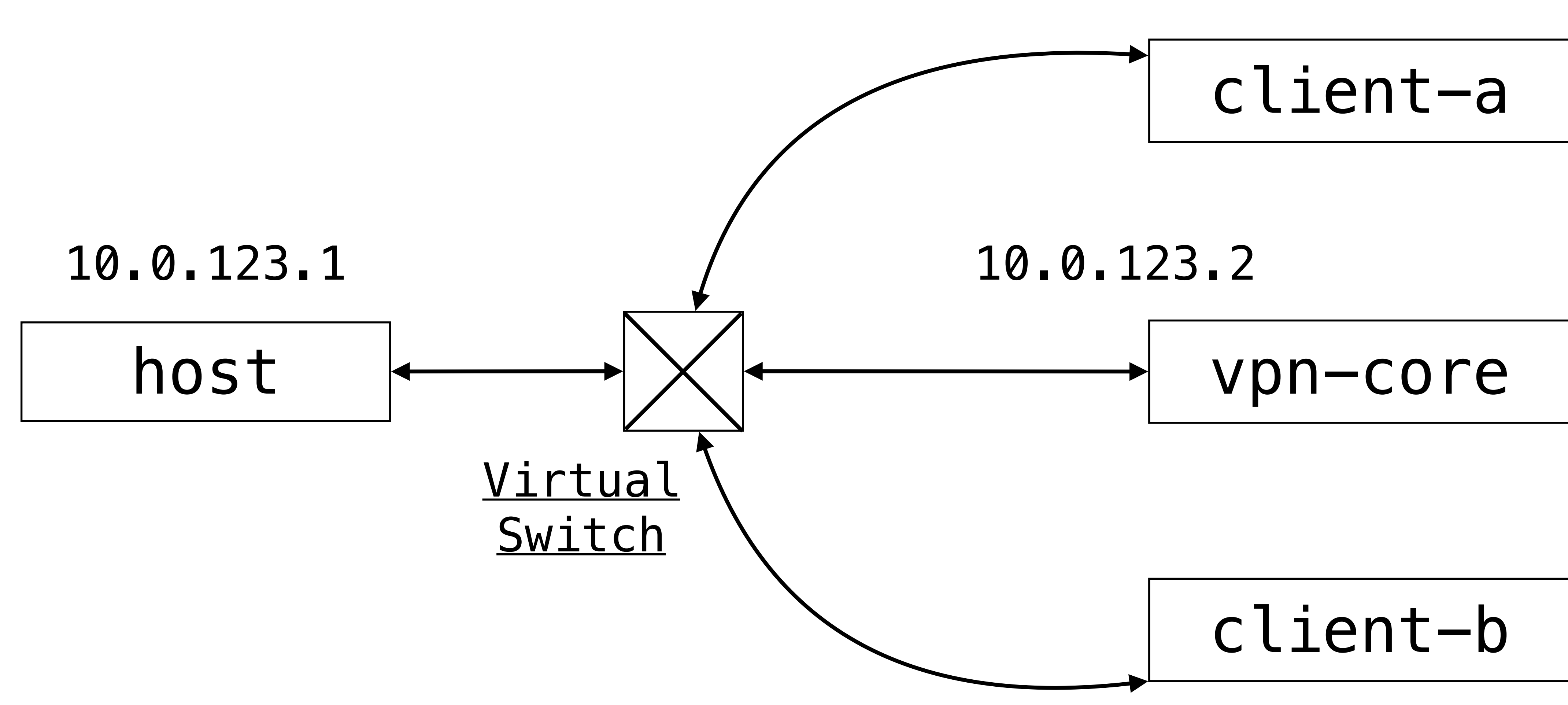


Antes de seguir… Vagrant

HashiCorp Vagrant

Fuente

- Llegados a este punto ya podemos discutir topologías **lógicas**.
- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.
- Hemos escogido Vagrant para levantarla a través de VirtualBox.
- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en pcolladosoto/wg-ebpf.

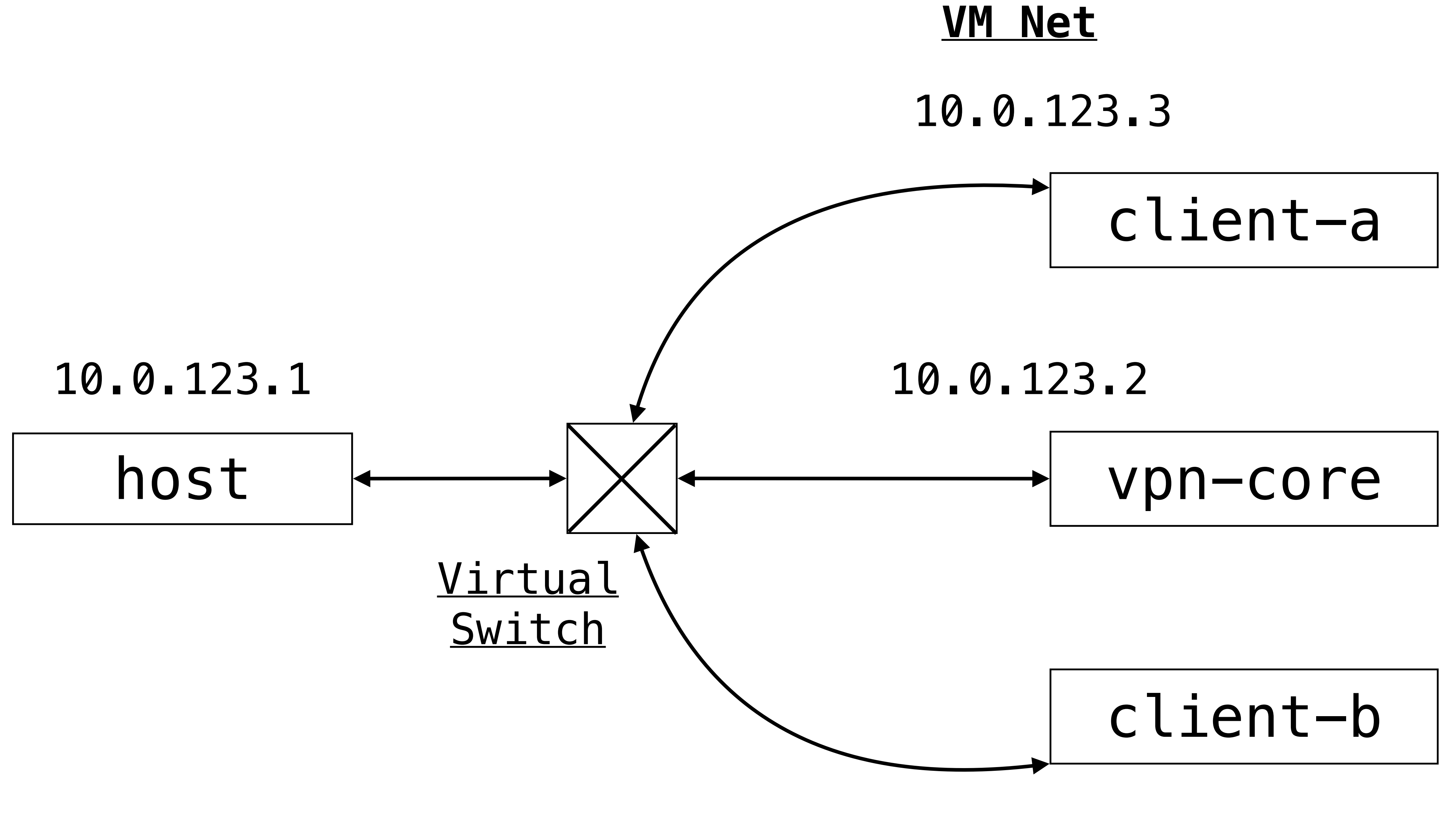


Antes de seguir... Vagrant

HashiCorp Vagrant

Fuente

- Llegados a este punto ya podemos discutir topologías **lógicas**.
- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.
- Hemos escogido Vagrant para levantarla a través de VirtualBox.
- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en pcolladosoto/wg-ebpf.

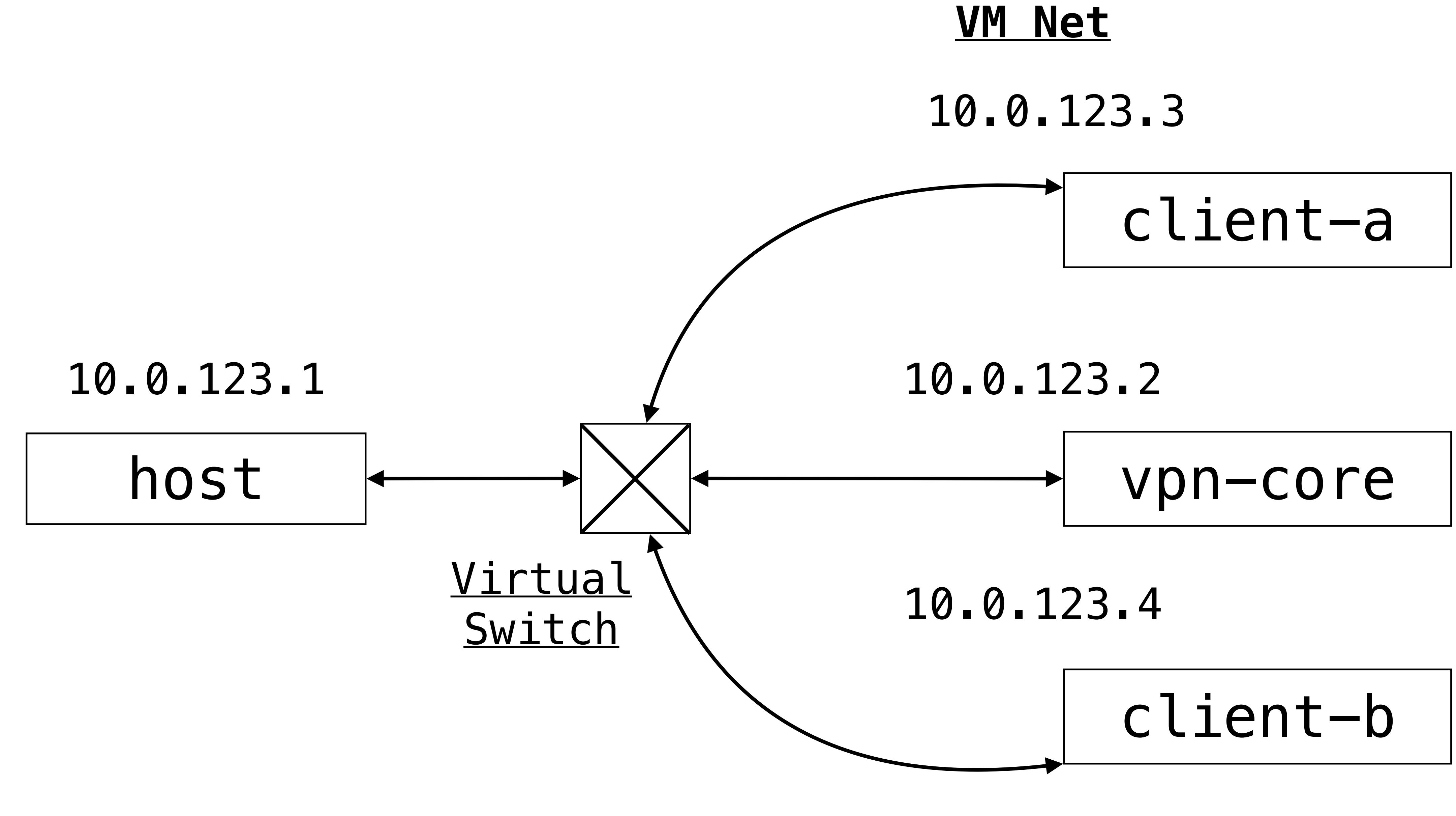


Antes de seguir… Vagrant

HashiCorp Vagrant

Fuente

- Llegados a este punto ya podemos discutir topologías **lógicas**.
- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.
- Hemos escogido Vagrant para levantarla a través de VirtualBox.
- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en pcolladosoto/wg-ebpf.

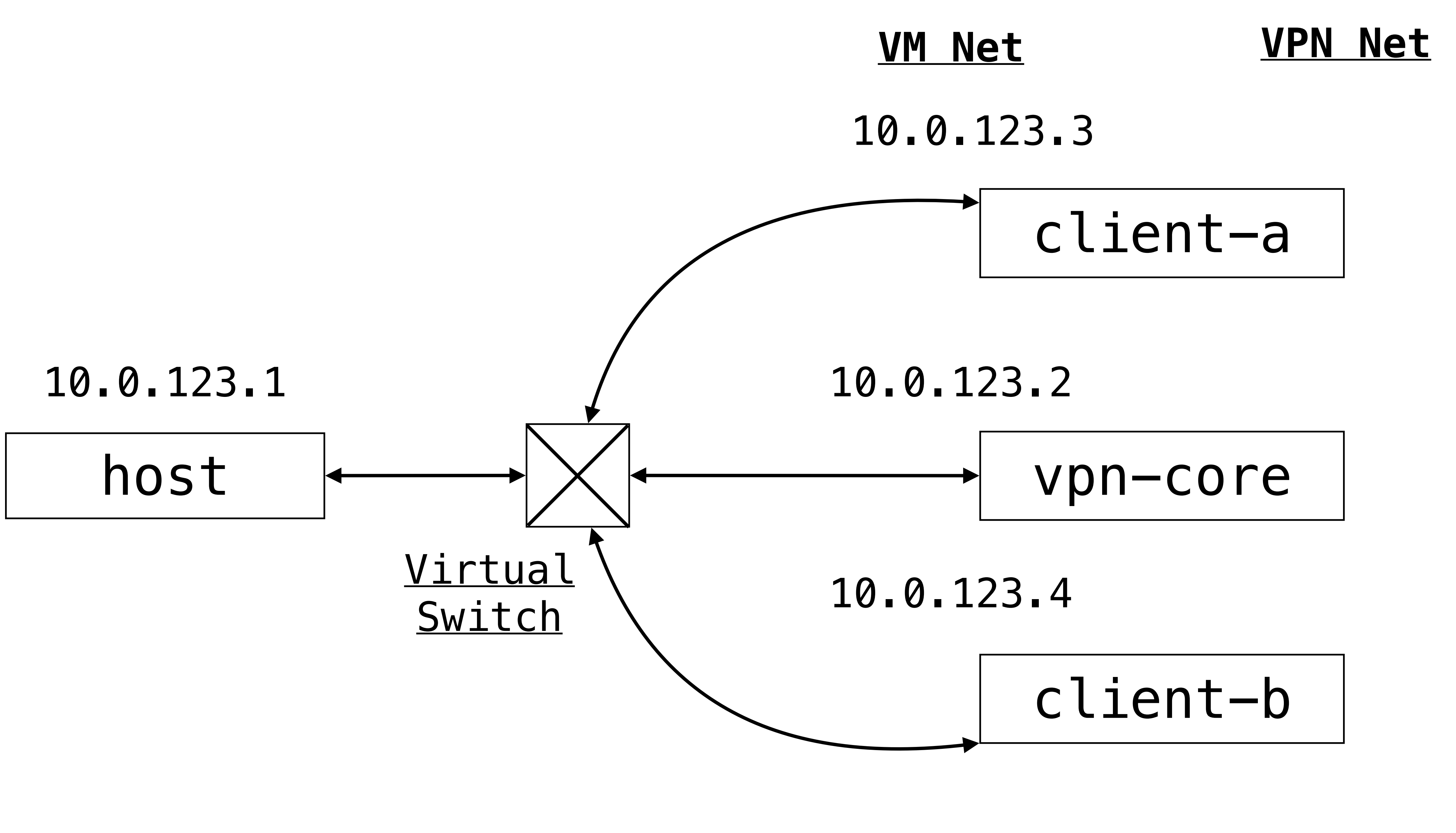


Antes de seguir... Vagrant

HashiCorp Vagrant

Fuente

- Llegados a este punto ya podemos discutir topologías **lógicas**.
- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.
- Hemos escogido Vagrant para levantarla a través de VirtualBox.
- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en pcolladosoto/wg-ebpf.

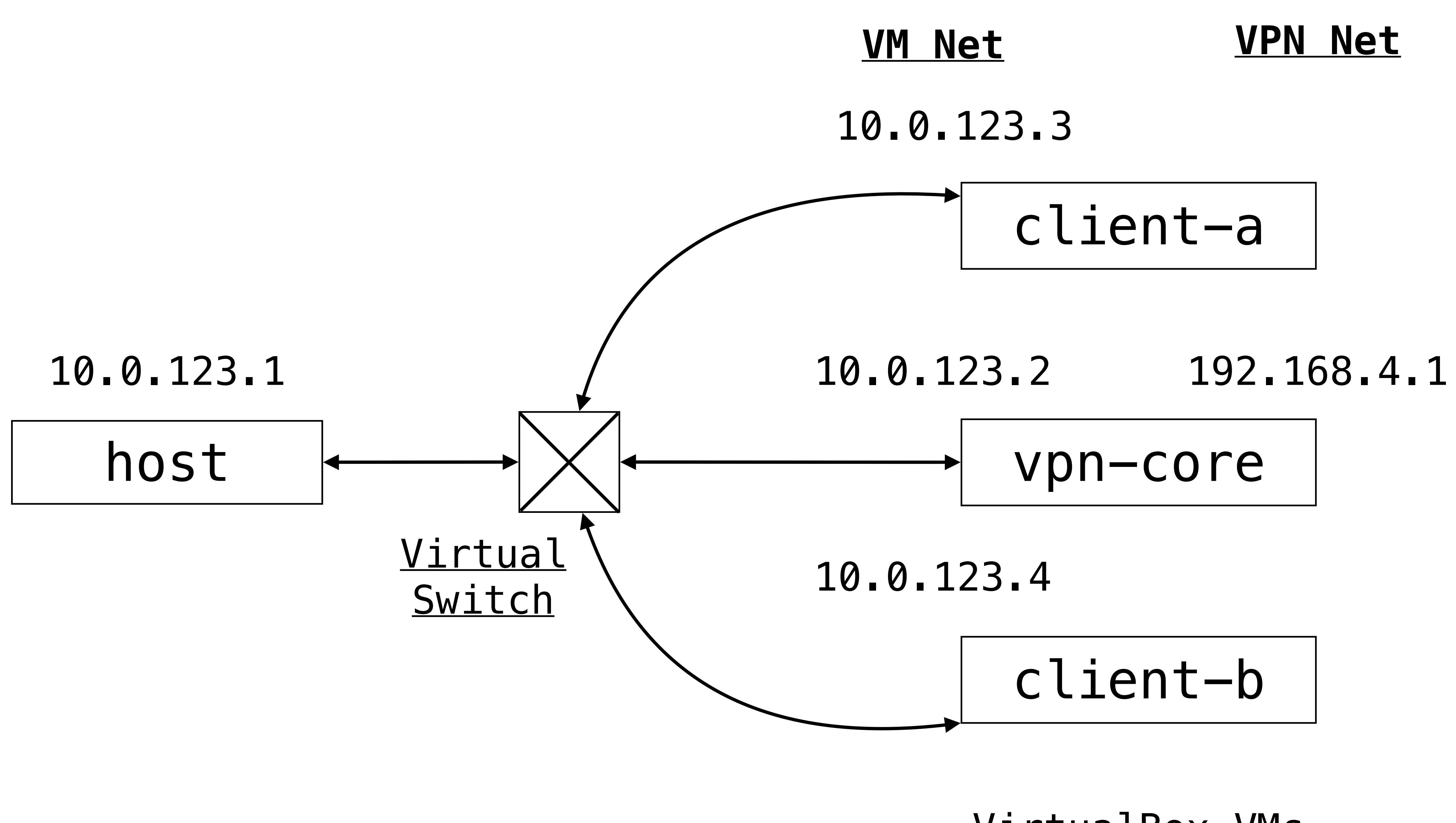


Antes de seguir... Vagrant

HashiCorp Vagrant

[Fuente]

- Llegados a este punto ya podemos discutir topologías **lógicas**.
- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.
- Hemos escogido [Vagrant] para levantarla a través de [VirtualBox].
- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en [pcolladosoto/wg-ebpf].



Antes de seguir… Vagrant

HashiCorp Vagrant

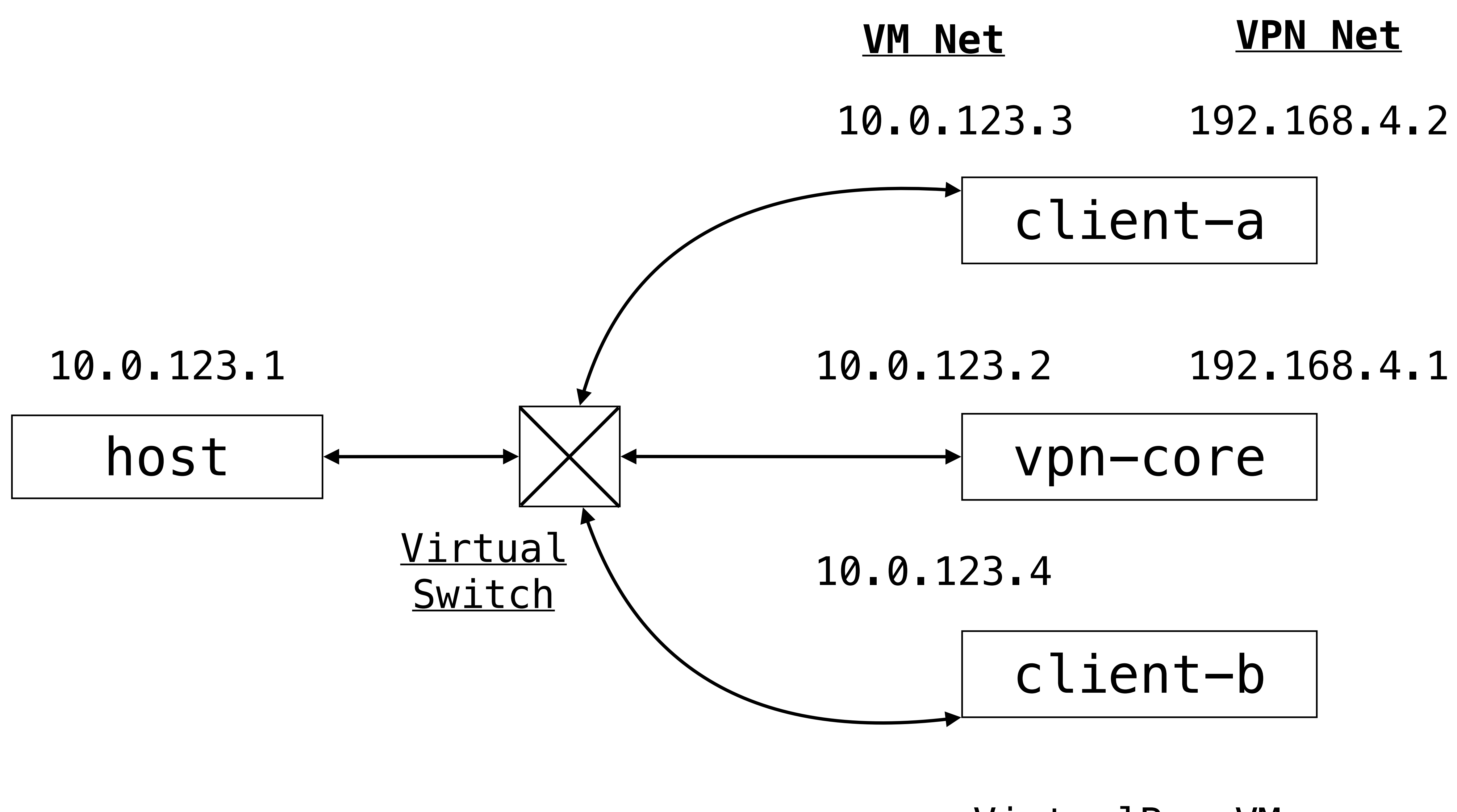
Fuente

- Llegados a este punto ya podemos discutir topologías **lógicas**.

- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.

- Hemos escogido Vagrant para levantarla a través de VirtualBox.

- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en pcolladosoto/wg-ebpf.

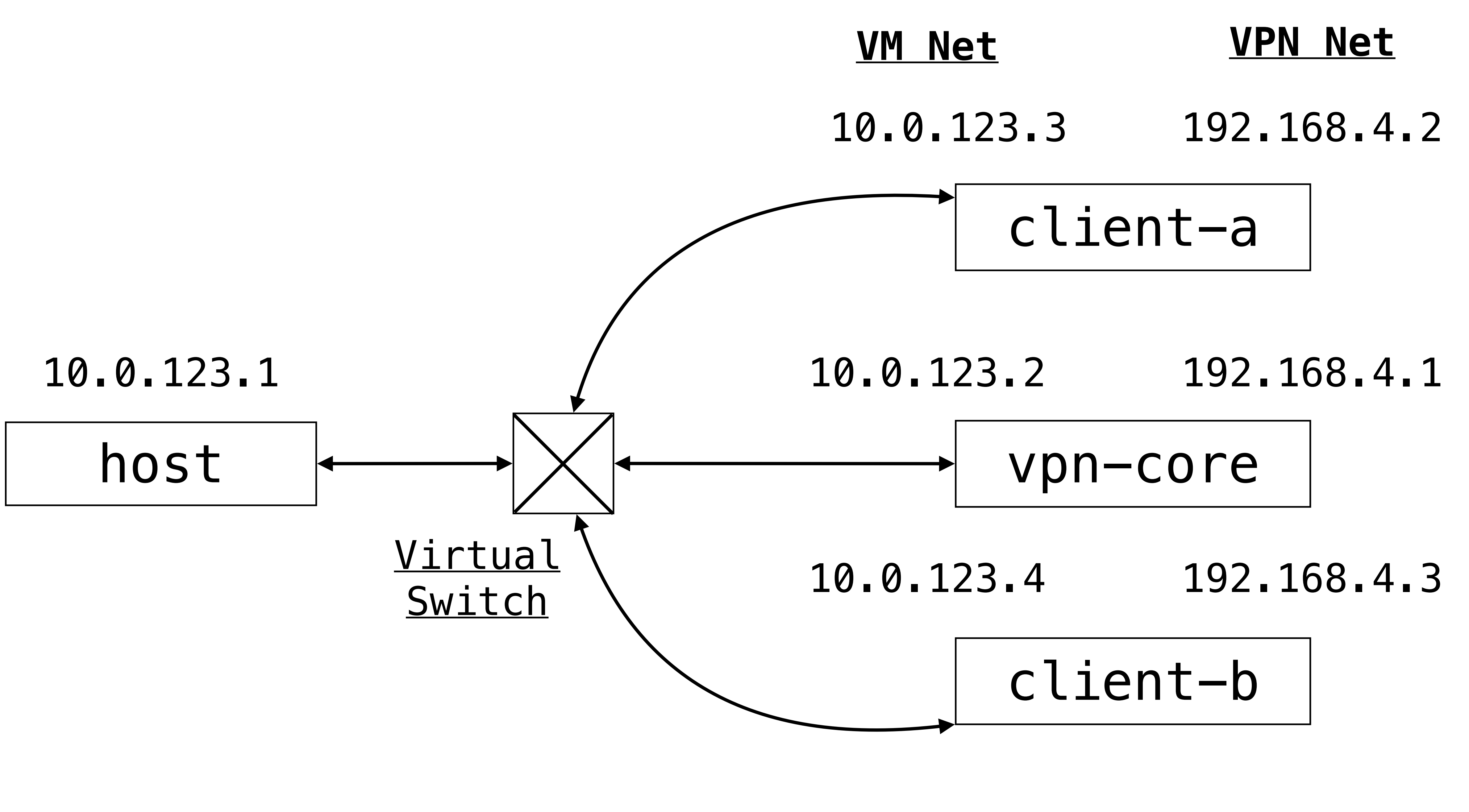


Antes de seguir… Vagrant

HashiCorp Vagrant

Fuente

- Llegados a este punto ya podemos discutir topologías **lógicas**.
- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.
- Hemos escogido Vagrant para levantarla a través de VirtualBox.
- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en pcolladosoto/wg-ebpf.

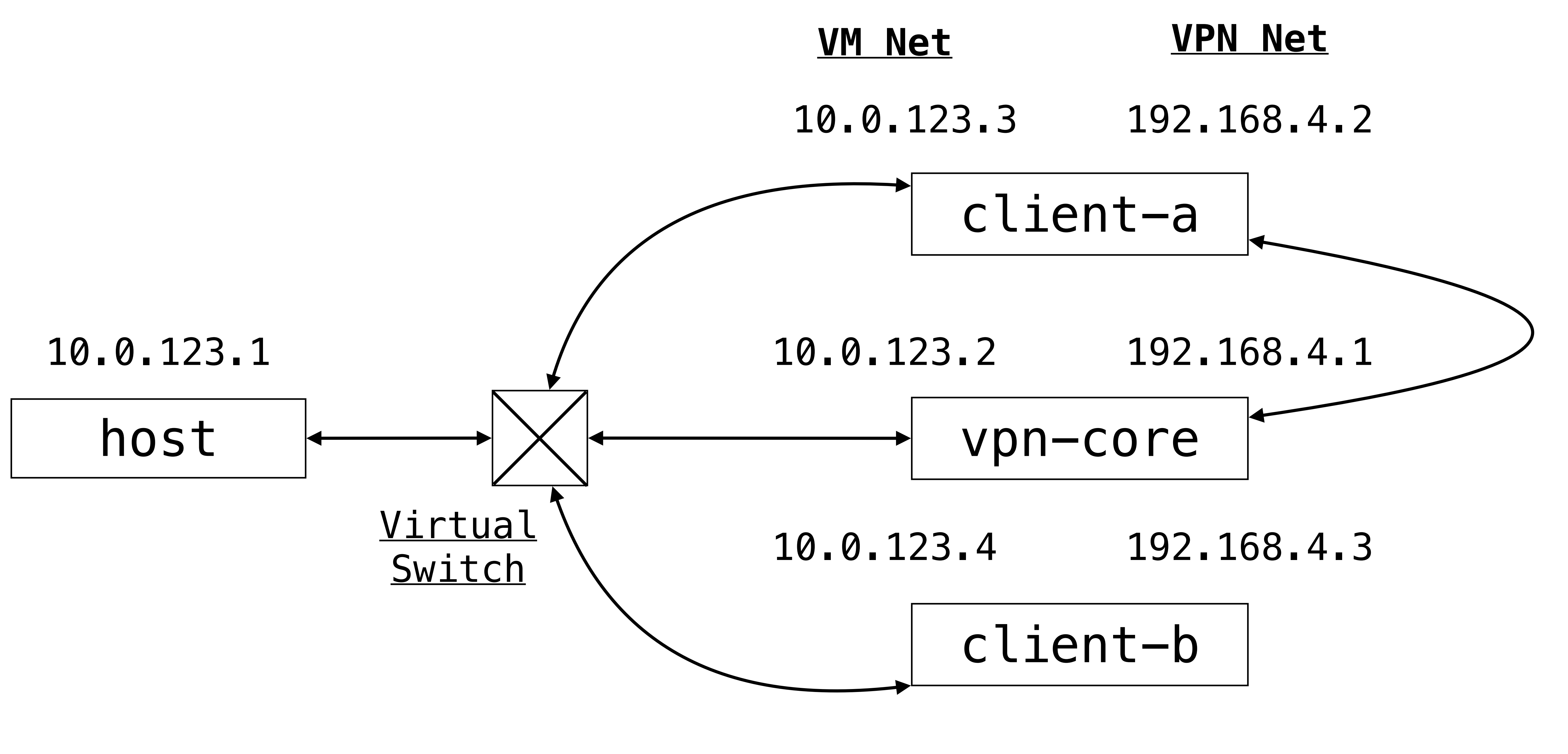


Antes de seguir... Vagrant

HashiCorp Vagrant

[Fuente]

- Llegados a este punto ya podemos discutir topologías **lógicas**.
- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.
- Hemos escogido [Vagrant] para levantarla a través de [VirtualBox].
- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en [pcolladosoto/wg-ebpf].

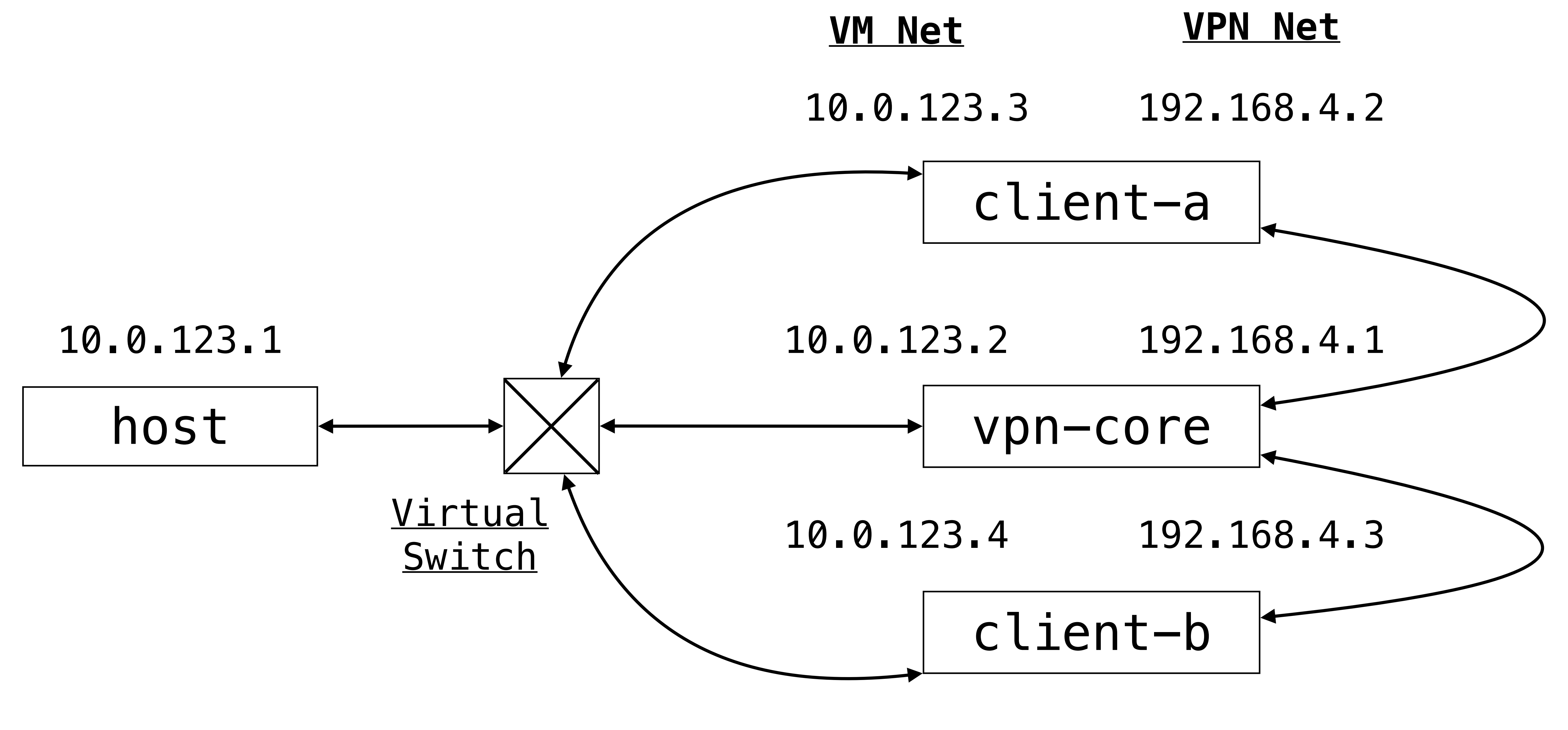


Antes de seguir... Vagrant

HashiCorp Vagrant

Fuente

- Llegados a este punto ya podemos discutir topologías **lógicas**.
- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.
- Hemos escogido Vagrant para levantarla a través de VirtualBox.
- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en pcolladosoto/wg-ebpf.

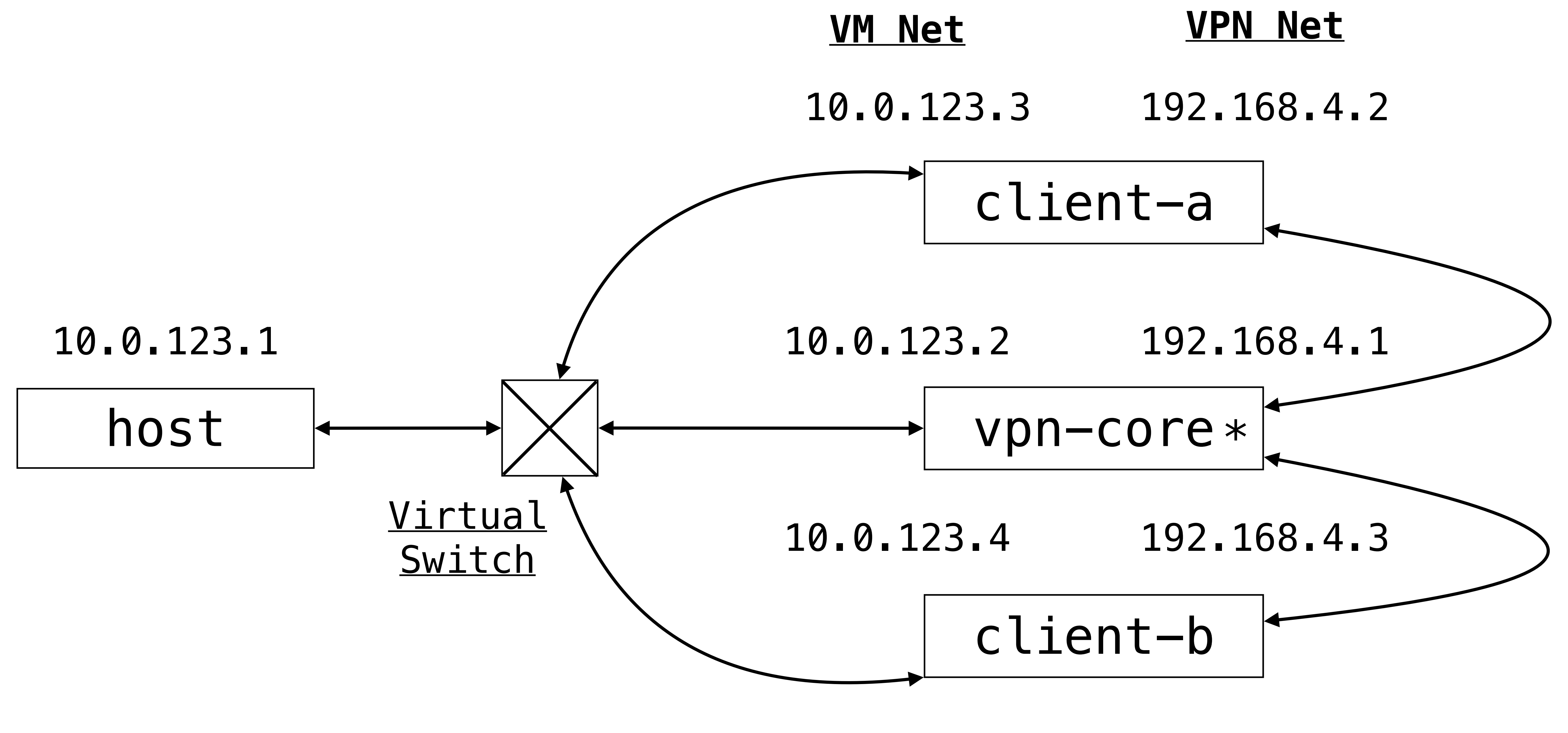


Antes de seguir… Vagrant

HashiCorp Vagrant

Fuente

- Llegados a este punto ya podemos discutir topologías **lógicas**.
- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.
- Hemos escogido Vagrant para levantarla a través de VirtualBox.
- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en pcolladosoto/wg-ebpf.

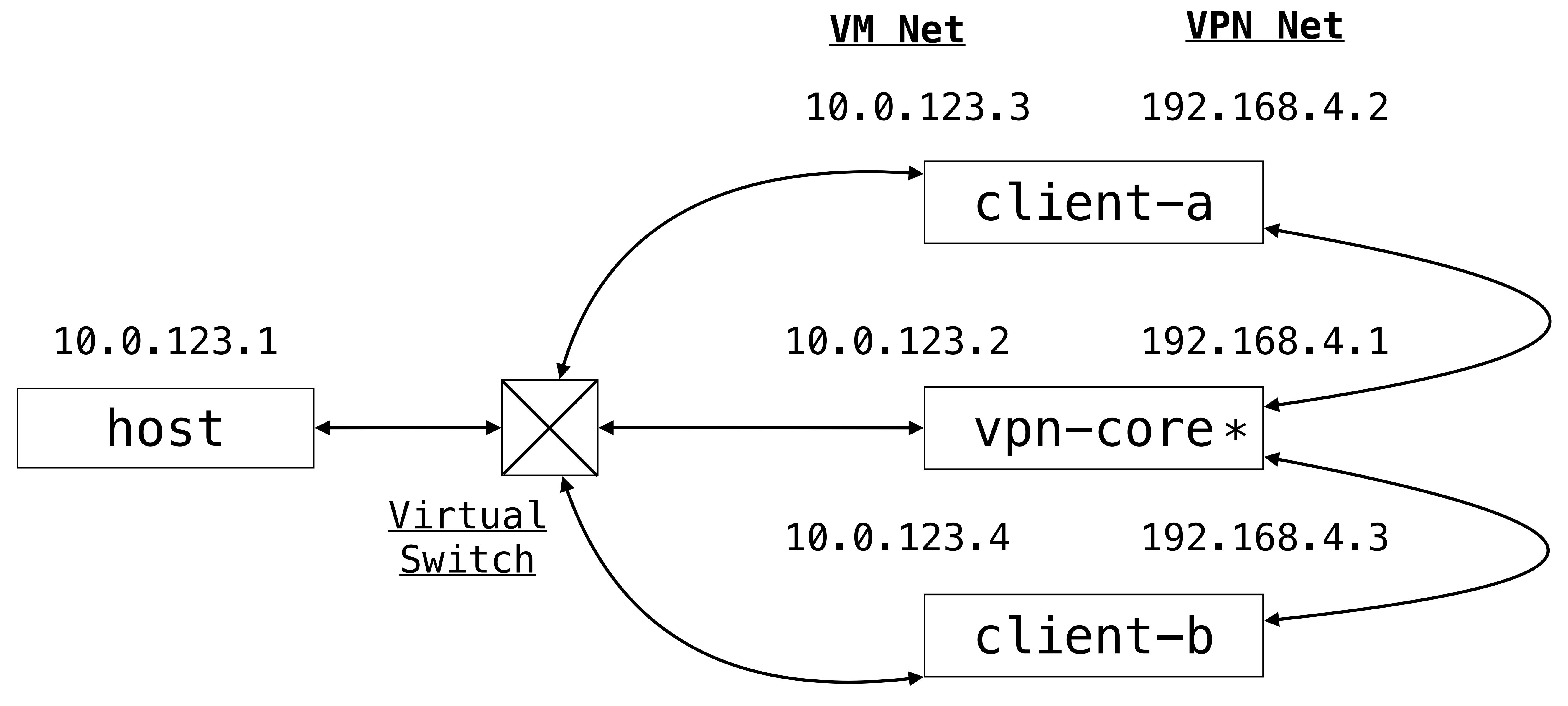


Antes de seguir… Vagrant

HashiCorp Vagrant

Fuente

- Llegados a este punto ya podemos discutir topologías **lógicas**.
- Vamos a basarnos en una topología «sintética» con 3 máquinas a partir de ahora.
- Hemos escogido Vagrant para levantarla a través de VirtualBox.
- Podéis encontrar el `Vagrantfile` junto a todo lo demás que iremos viendo en pcolladosoto/wg-ebpf.



Desplegando WireGuard

- El despliegue de la VPN es muy sencillo:
  - Generamos un archivo de configuración para cada miembro.
  - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
  - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
  - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

```
###################
# vpn-core's conf #
###################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

###################
# client-a's conf #
###################
[Interface]
PrivateKey = iLgkq+1tSZ6BKMAeAoqwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJOuN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0
Endpoint = 10.0.123.2:51820
```

Esta y las demás imágenes con código y configuración se han elaborado con Carbon.

Desplegando WireGuard

- El despliegue de la VPN es muy sencillo:
  - Generamos un archivo de configuración para cada miembro.
  - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
  - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
  - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

```
###################
# vpn-core's conf #
###################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

###################
# client-a's conf #
###################
[Interface]
PrivateKey = iLgkq+1tSZ6BKMAeAoqwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJOuN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0
Endpoint = 10.0.123.2:51820
```

Esta y las demás imágenes con código y configuración se han elaborado con Carbon.

Desplegando WireGuard

- El despliegue de la VPN es muy sencillo:
    - Generamos un archivo de configuración para cada miembro.
    - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
    - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
    - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

```
###################
# vpn-core's conf #
###################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

###################
# client-a's conf #
###################
[Interface]
PrivateKey = iLgkq+1tSZ6BKMAeAoqwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJOuN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0
Endpoint = 10.0.123.2:51820
```

Esta y las demás imágenes con código y configuración se han elaborado con Carbon.

Desplegando WireGuard

- El despliegue de la VPN es muy sencillo:
  - Generamos un archivo de configuración para cada miembro.
  - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
  - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
  - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

```
###################
# vpn-core's conf #
###################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

###################
# client-a's conf #
###################
[Interface]
PrivateKey = iLgkq+1tSZ6BKMAeAoqwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJ0uN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0
Endpoint = 10.0.123.2:51820
```

Esta y las demás imágenes con código y configuración se han elaborado con Carbon.

Desplegando WireGuard

- El despliegue de la VPN es muy sencillo:
  - Generamos un archivo de configuración para cada miembro.
  - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
  - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
  - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

```
###################
# vpn-core's conf #
###################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

###################
# client-a's conf #
###################
[Interface]
PrivateKey = iLqkq+1tSZ6BKMAeAoqwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJOuN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0
Endpoint = 10.0.123.2:51820
```

Esta y las demás imágenes con código y configuración se han elaborado con Carbon.

Desplegando WireGuard

- El despliegue de la VPN es muy sencillo:
  - Generamos un archivo de configuración para cada miembro.
  - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
  - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
  - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

```
###################
# vpn-core's conf #
###################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

###################
# client-a's conf #
###################
[Interface]
PrivateKey = iLqkq+1tSZ6BKMAeAoqwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJOuN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0
Endpoint = 10.0.123.2:51820
```

Esta y las demás imágenes con código y configuración se han elaborado con [Carbon](#).

Desplegando WireGuard

- El despliegue de la VPN es muy sencillo:
  - Generamos un archivo de configuración para cada miembro.
  - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
  - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
  - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

```
###################
# vpn-core's conf #
###################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

###################
# client-a's conf #
###################
[Interface]
PrivateKey = iLgkq+1tSZ6BKMAeAoqwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJOuN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0
Endpoint = 10.0.123.2:51820
```

`Esta y las demás imágenes con código y configuración se han elaborado con Carbon.`

Desplegando WireGuard

- El despliegue de la VPN es muy sencillo:
  - Generamos un archivo de configuración para cada miembro.
  - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
  - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
  - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

```
###################
# vpn-core's conf #
###################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

###################
# client-a's conf #
###################
[Interface]
PrivateKey = iLgkq+1tSZ6BKMAeAoqwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJOuN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0
Endpoint = 10.0.123.2:51820
```

Esta y las demás imágenes con código y configuración se han elaborado con Carbon.

Desplegando WireGuard

- El despliegue de la VPN es muy sencillo:
  - Generamos un archivo de configuración para cada miembro.
  - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
  - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
  - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

```
###################
# vpn-core's conf #
###################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

###################
# client-a's conf #
###################
[Interface]
PrivateKey = iLgkq+1tSZ6BKMAeAoqwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJOuN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0
Endpoint = 10.0.123.2:51820
```

Esta y las demás imágenes con código y configuración se han elaborado con Carbon.

Desplegando WireGuard

- El despliegue de la VPN es muy sencillo:
  - Generamos un archivo de configuración para cada miembro.
  - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
  - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
  - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

```
###################
# vpn-core's conf #
###################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

###################
# client-a's conf #
###################
[Interface]
PrivateKey = iLgkg+1tSZ6BKMAeAogwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJOuN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVWO7VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0
Endpoint = 10.0.123.2:51820
```

Esta y las demás imágenes con código y configuración se han elaborado con Carbon.

Desplegando WireGuard

- El despliegue de la VPN es muy sencillo:
  - Generamos un archivo de configuración para cada miembro.
  - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
  - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
  - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

```
###################
# vpn-core's conf #
###################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVW07VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

###################
# client-a's conf #
###################
[Interface]
PrivateKey = iLgkq+1tSZ6BKMAeAoqwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJOuN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVW07VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0
Endpoint = 10.0.123.2:51820
```

Esta y las demás imágenes con código y configuración se han elaborado con Carbon.

Desplegando WireGuard

- El despliegue de la VPN es muy sencillo:
  - Generamos un archivo de configuración para cada miembro.
  - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
  - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
  - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

```
###################
# vpn-core's conf #
###################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVW07VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

###################
# client-a's conf #
###################
[Interface]
PrivateKey = iLgkq+1tSZ6BKMAeAoqwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJOuN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVW07VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0
Endpoint = 10.0.123.2:51820
```

Esta y las demás imágenes con código y configuración se han elaborado con Carbon.

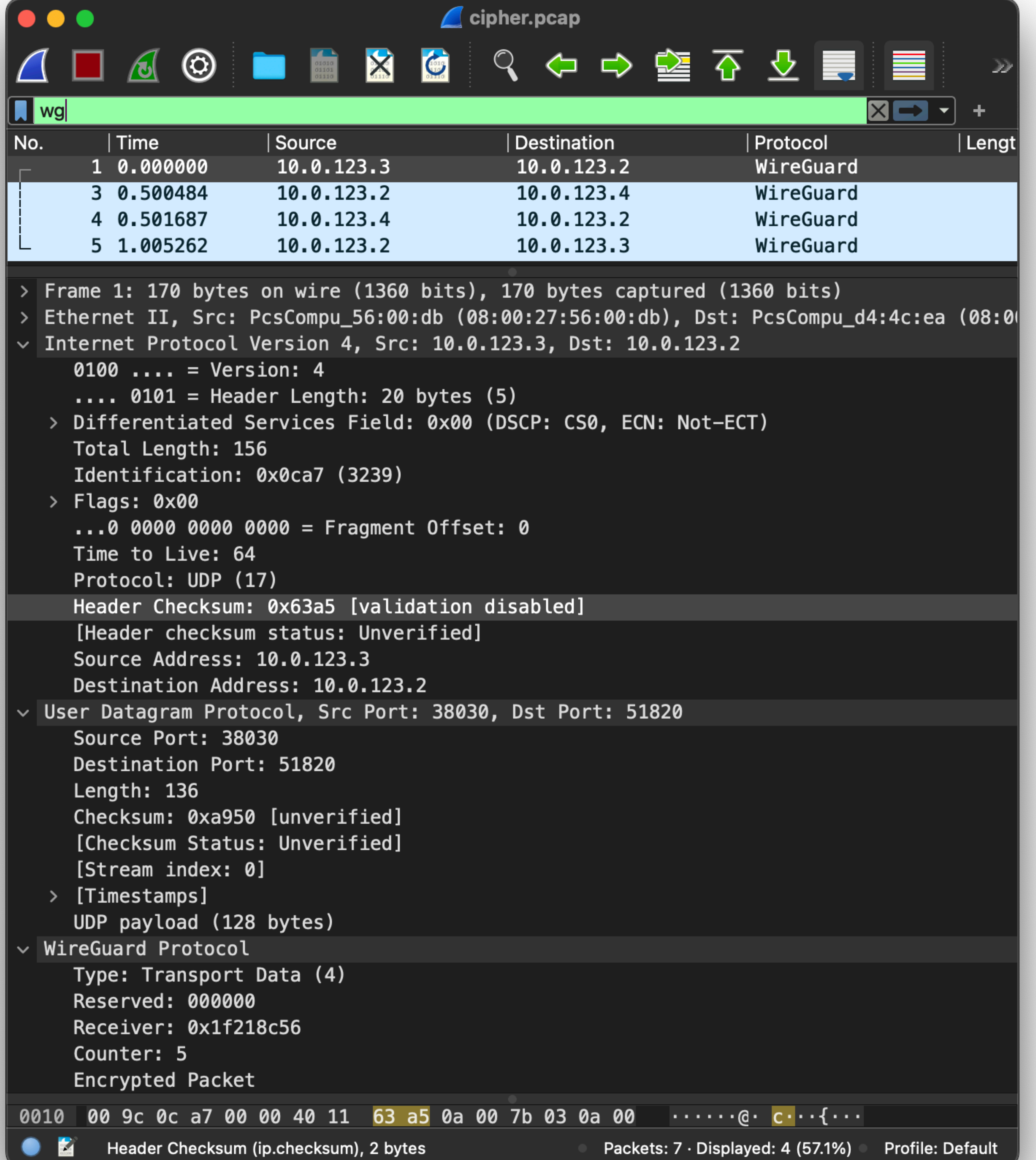
Desplegando WireGuard

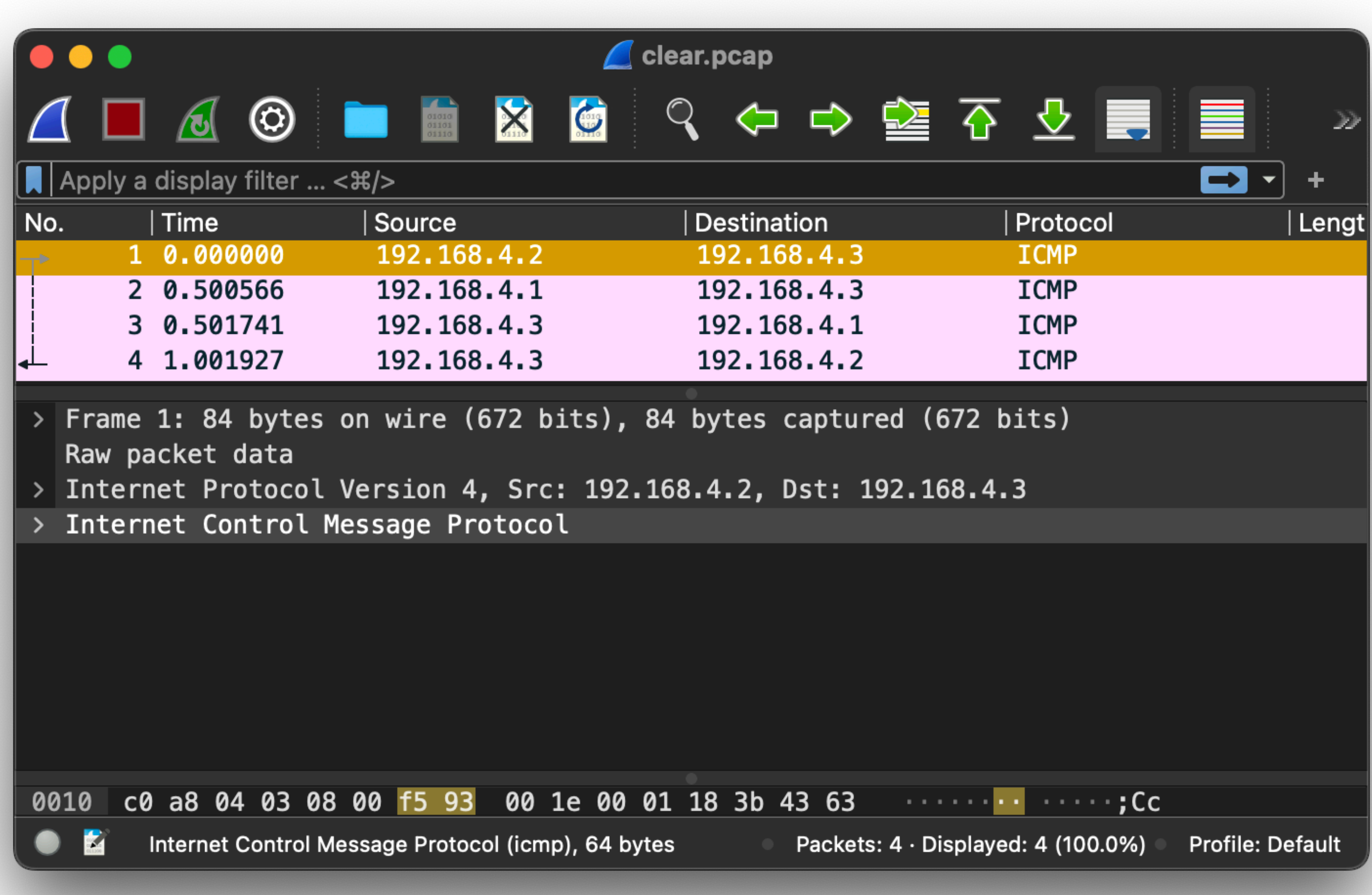
- El despliegue de la VPN es muy sencillo:
  - Generamos un archivo de configuración para cada miembro.
  - El archivo se almacena en **`/etc/wireguard/ifname.conf`**.
  - Gestionamos la VPN con **`wg-quick {up, down} ifname`**.
- Cada cliente tiene asociada una `IPv4` de un rango privado.
  - En definitiva, generamos una red «overlay» sobre la topología física.
- También podemos controlar la «cantidad» de tráfico a tunelar.

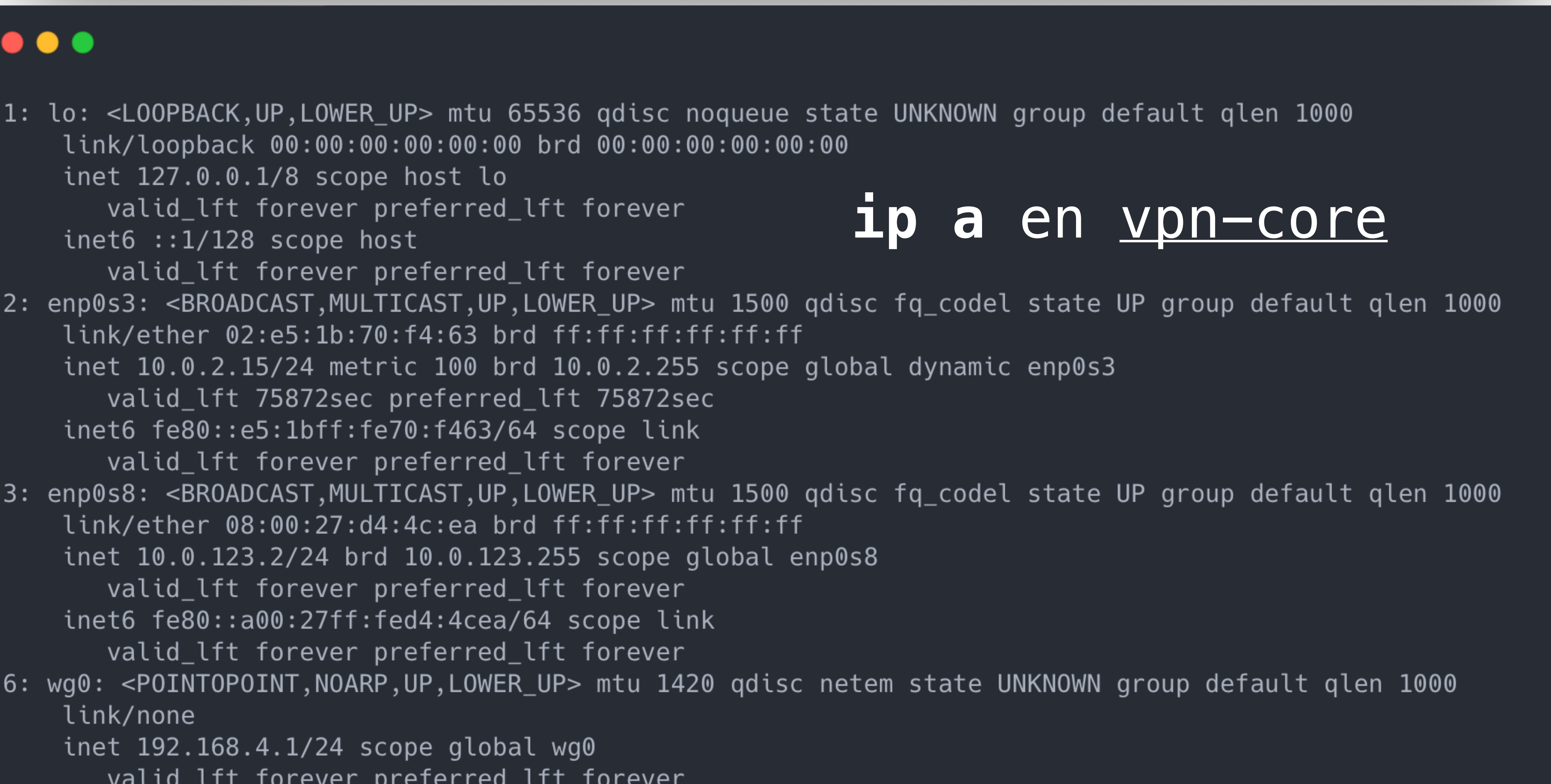
```
####################
# vpn-core's conf #
####################
[Interface]
PrivateKey = wJlBjCdC4GS6jV3adhgljSKXRMXbpiYfLmEpGNr743E=
Address = 192.168.4.1/24
ListenPort = 51820


# client-a
[Peer]
PublicKey = QF20FvY3hkToQFRAIP1G+UELDXX9mMLipFchtRtAly0=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVW07VLTVC59wdDXGDw=
AllowedIPs = 192.168.4.2/32

####################
# client-a's conf #
####################
[Interface]
PrivateKey = iLqkq+1tSZ6BKMAeAoqwRHYlnJ5xDw/h66AsVsvb7HA=
Address = 192.168.4.2/24

[Peer]
PublicKey = dddNjX8TSiXx9V+HK0+E4Qd//k8N3FHYAJ0uN8PxBhM=
PresharedKey = XDGYygoz5sN1wcU8Tf2B3HcudVW07VLTVC59wdDXGDw=
AllowedIPs = 0.0.0.0/0          ¡Tunelamos todo!
Endpoint = 10.0.123.2:51820
```

Esta y las demás imágenes con código y configuración se han elaborado con Carbon.

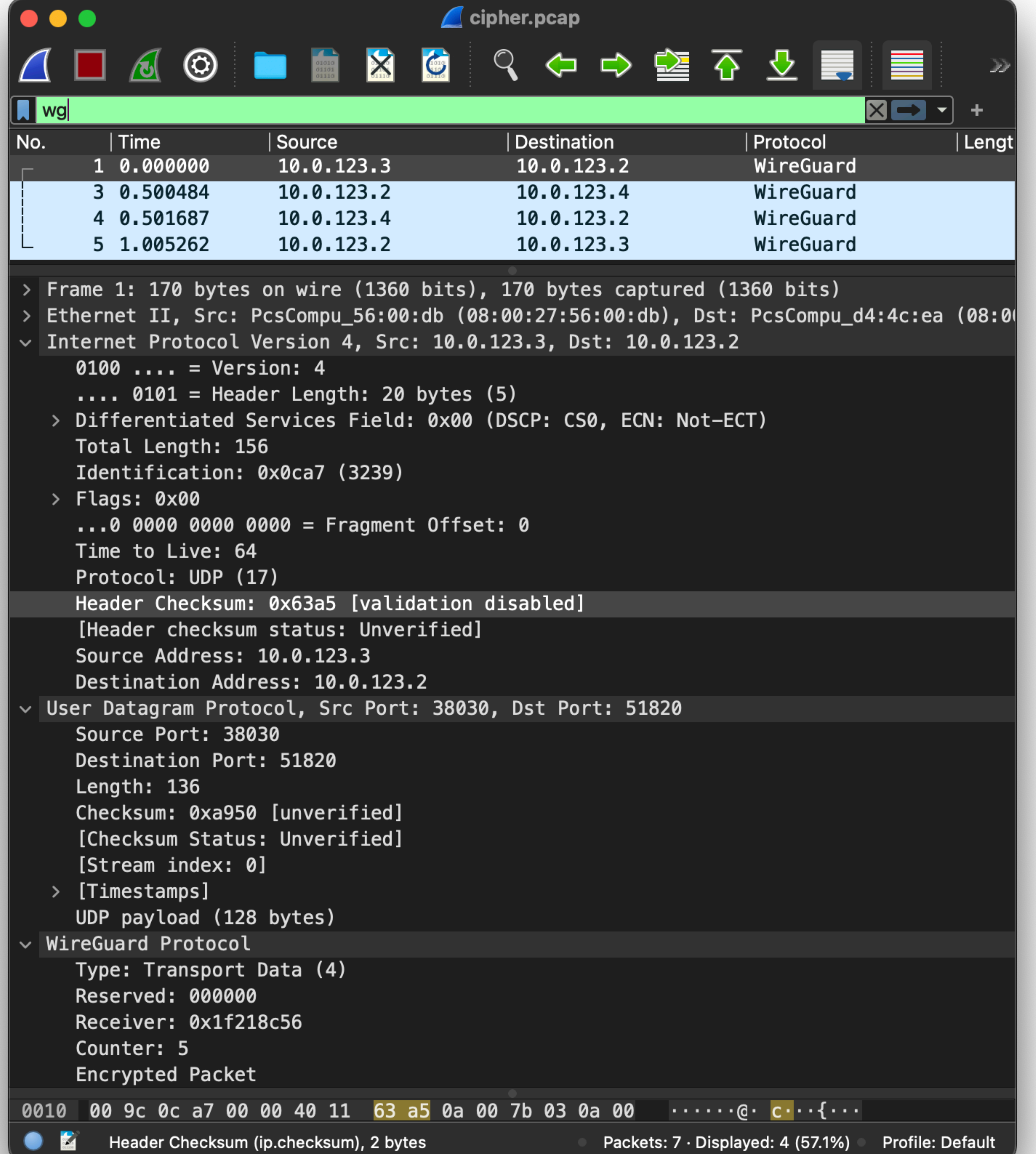
Pila de protocolos

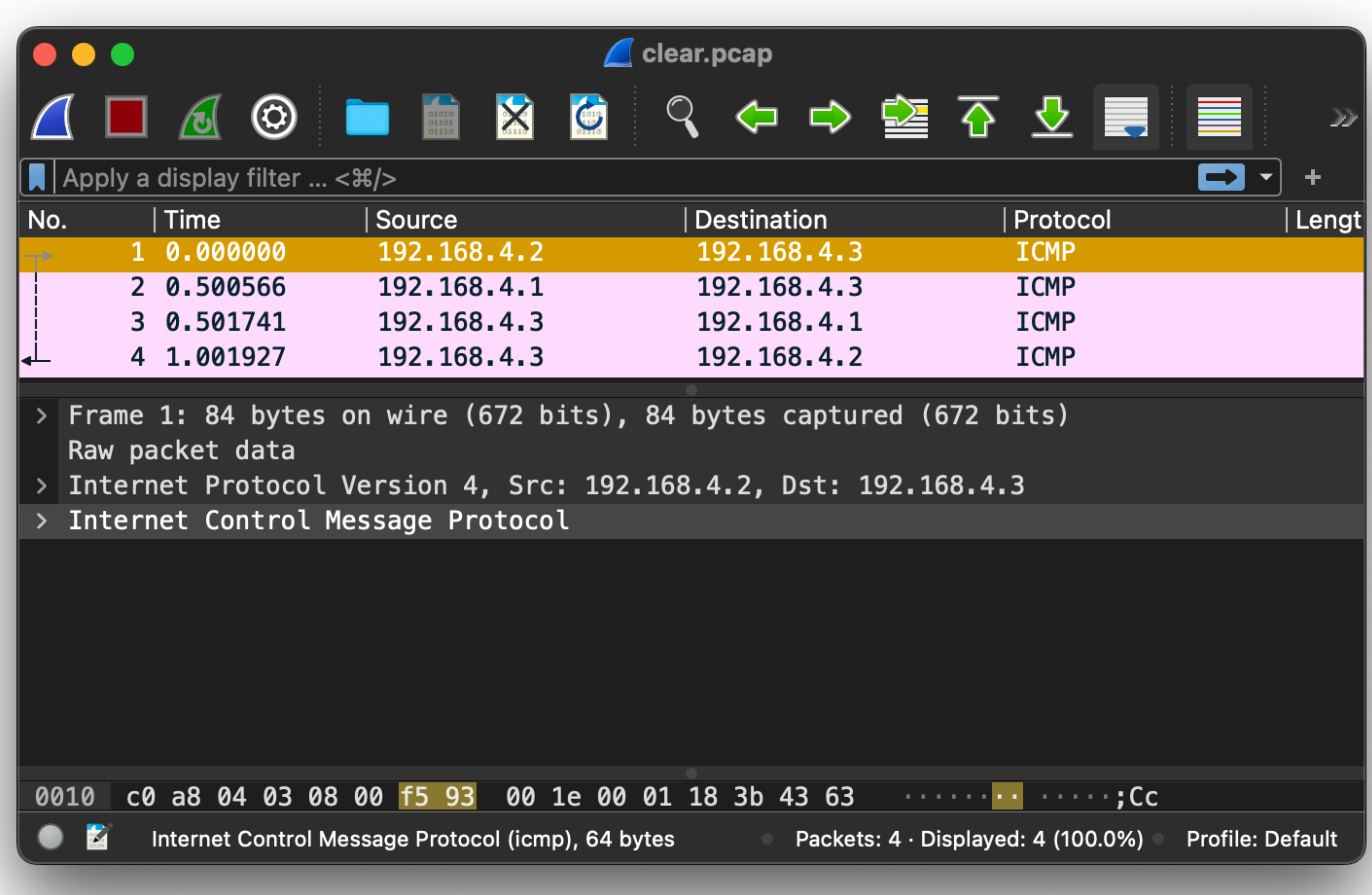


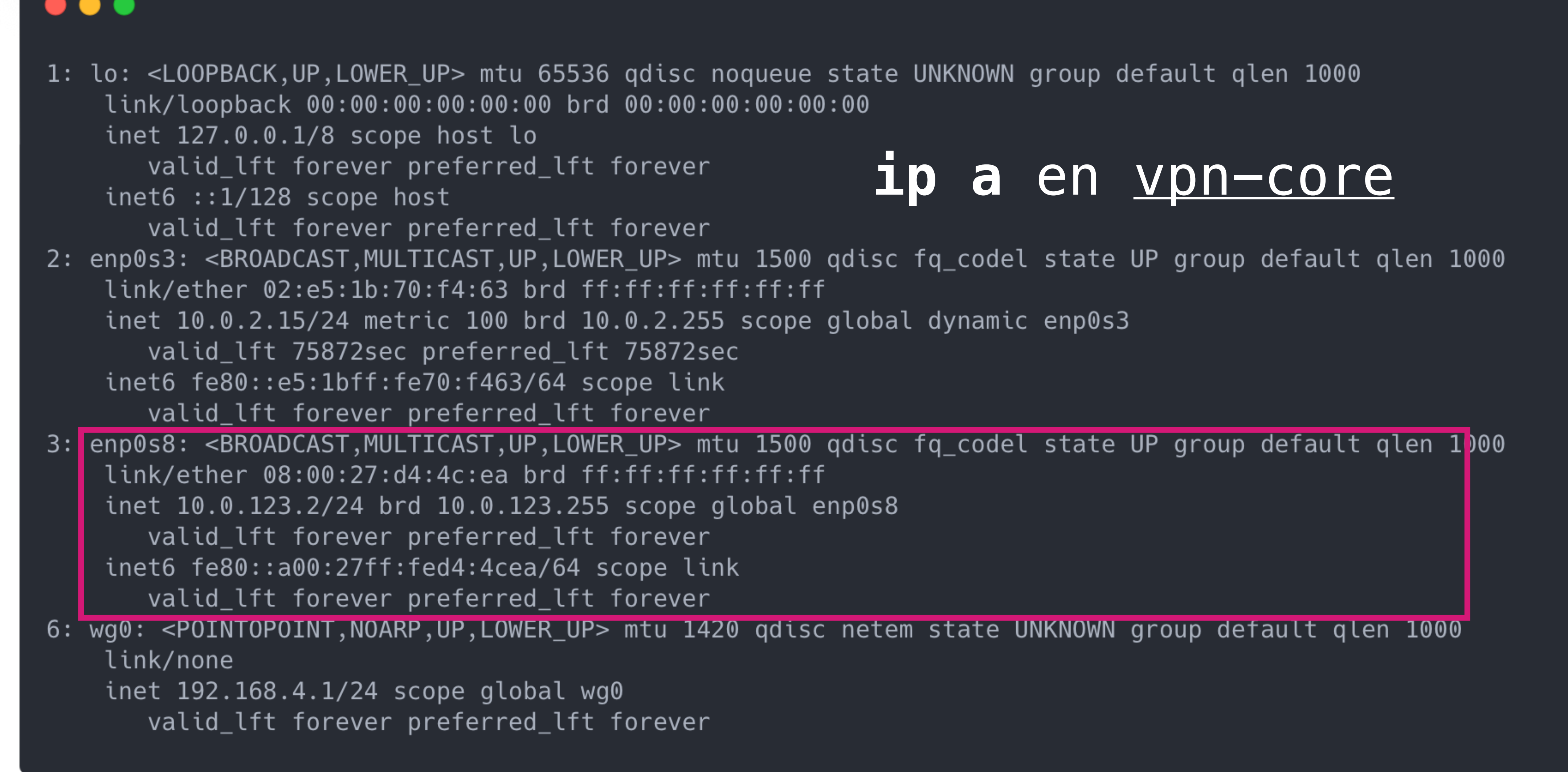




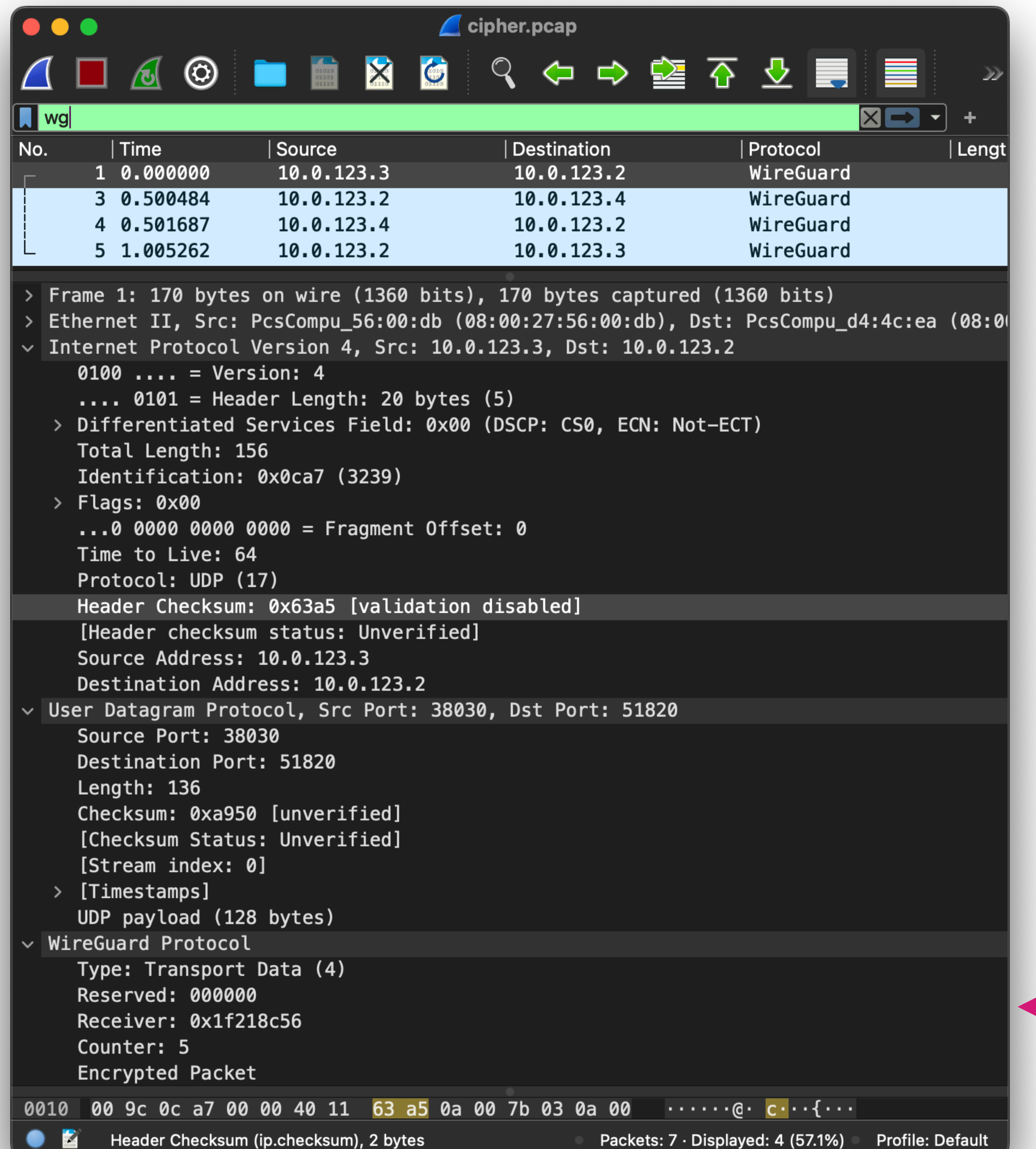
Pila de protocolos

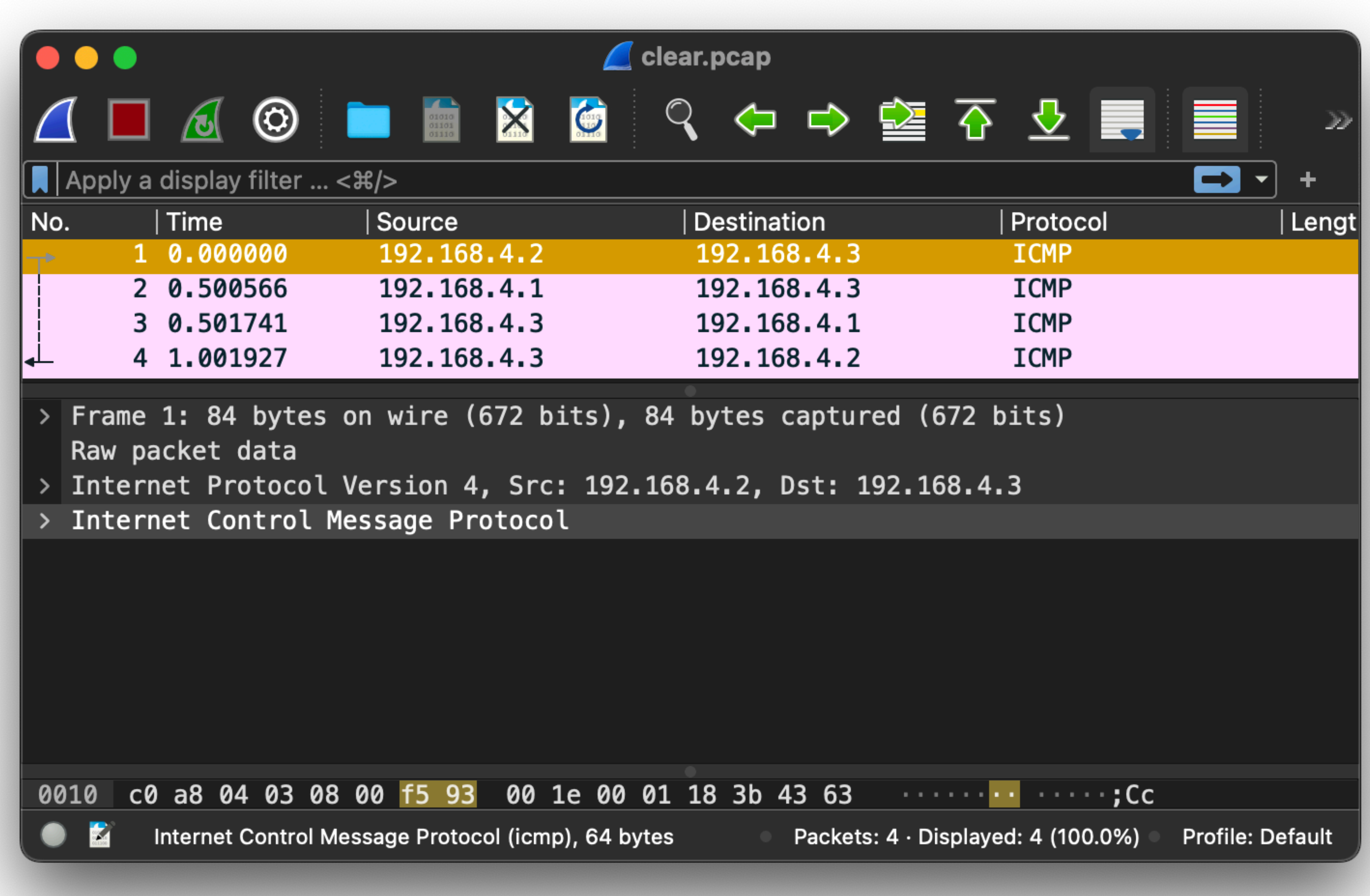


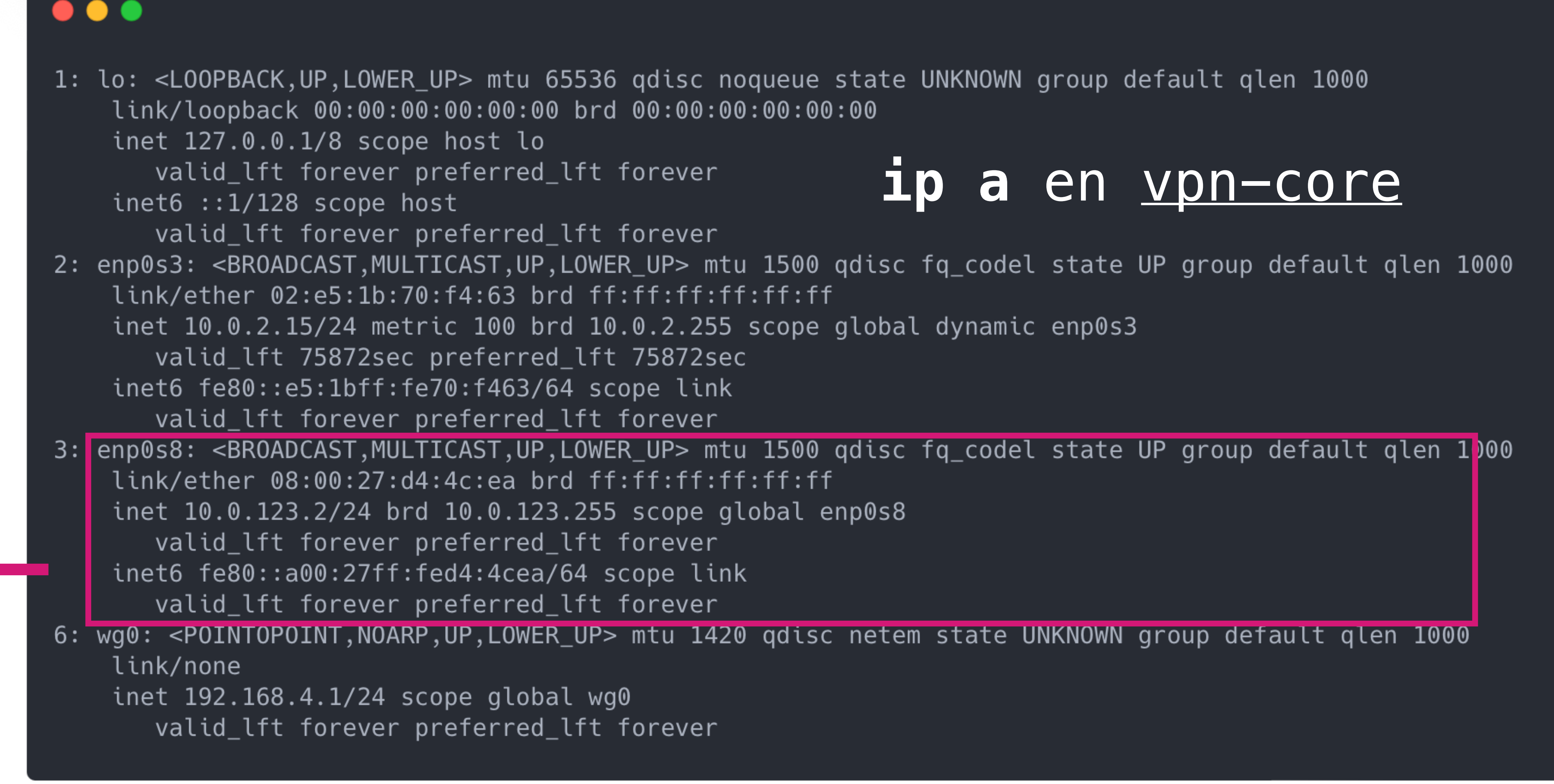




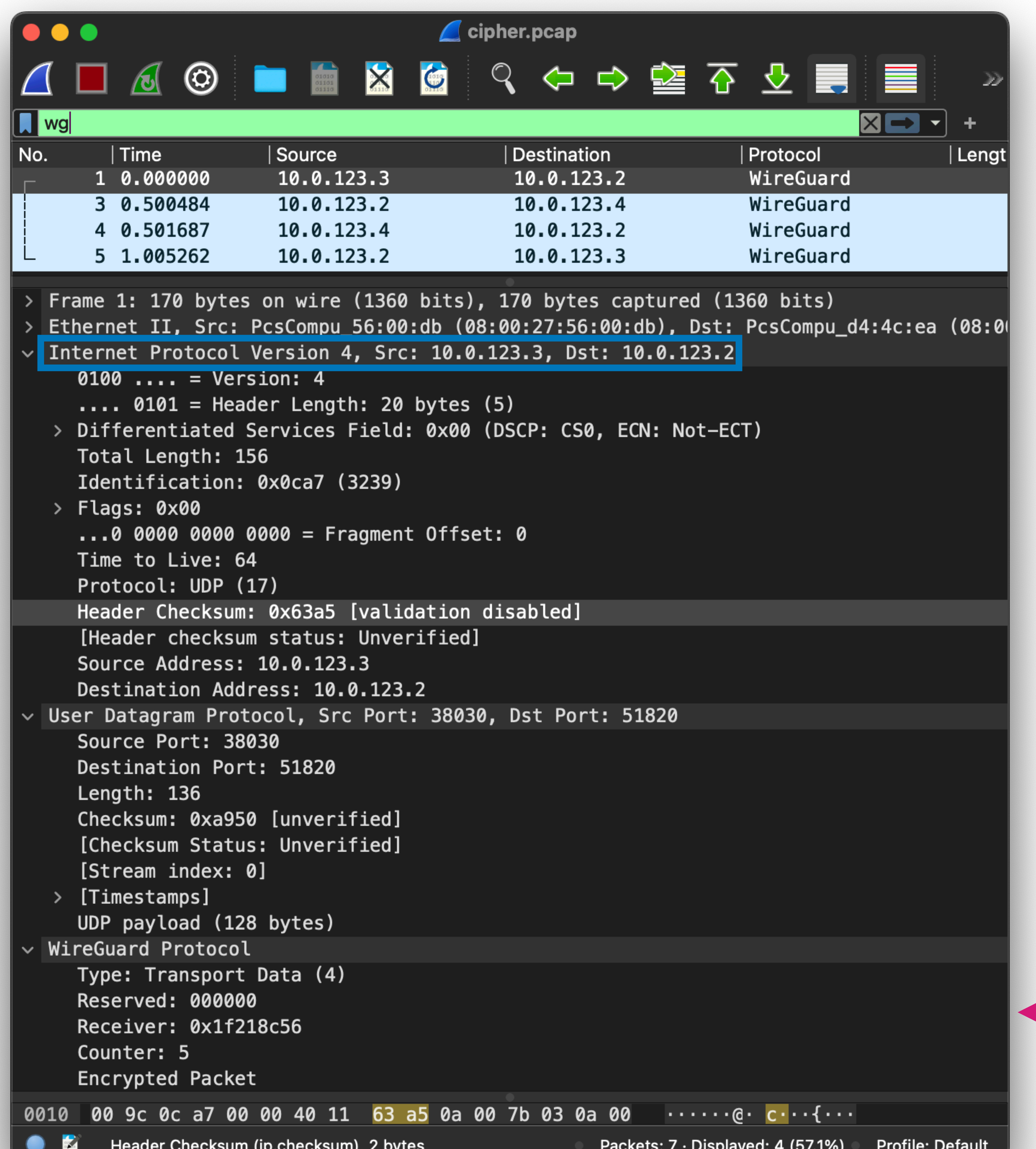
Pila de protocolos

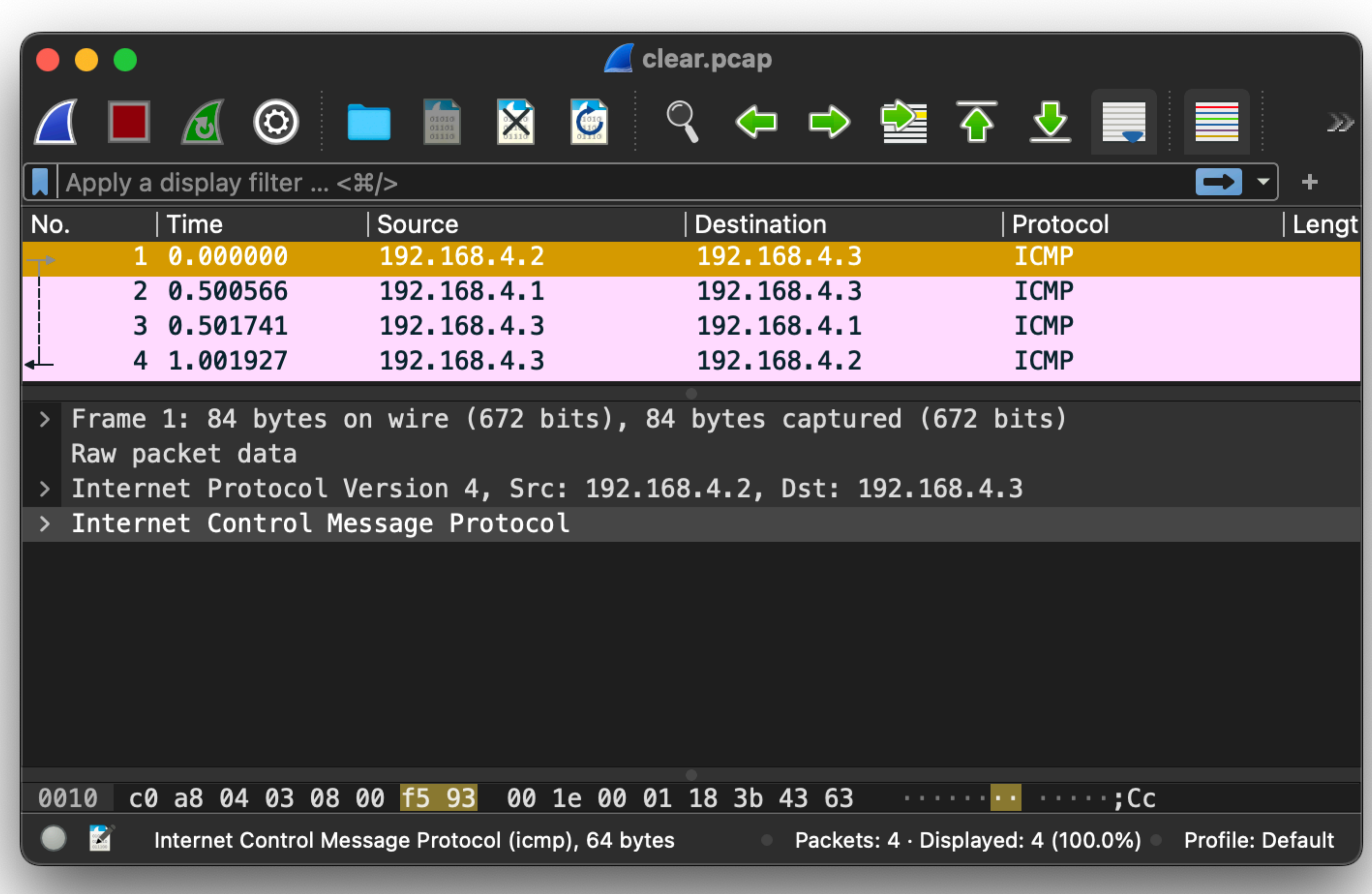


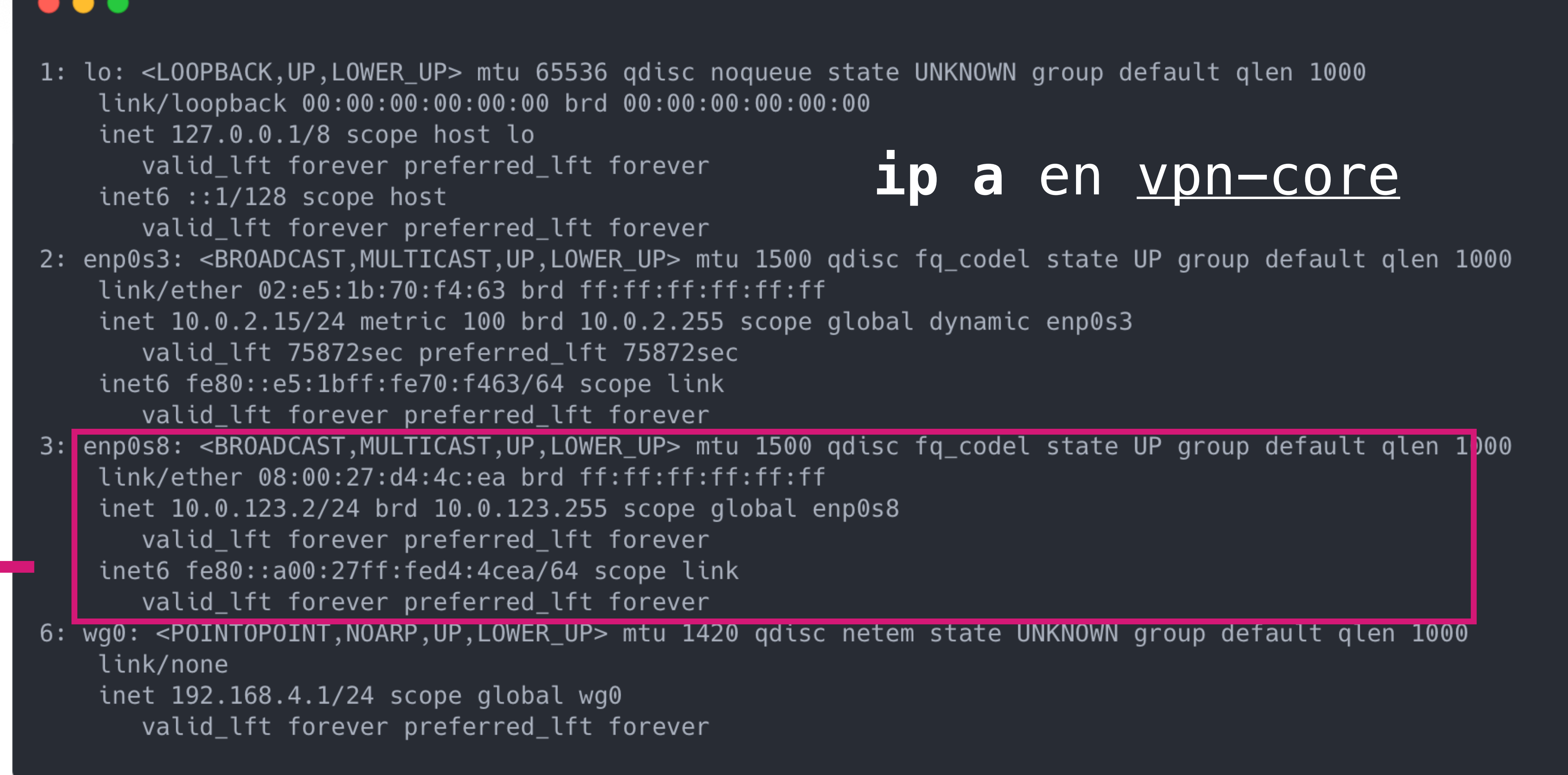




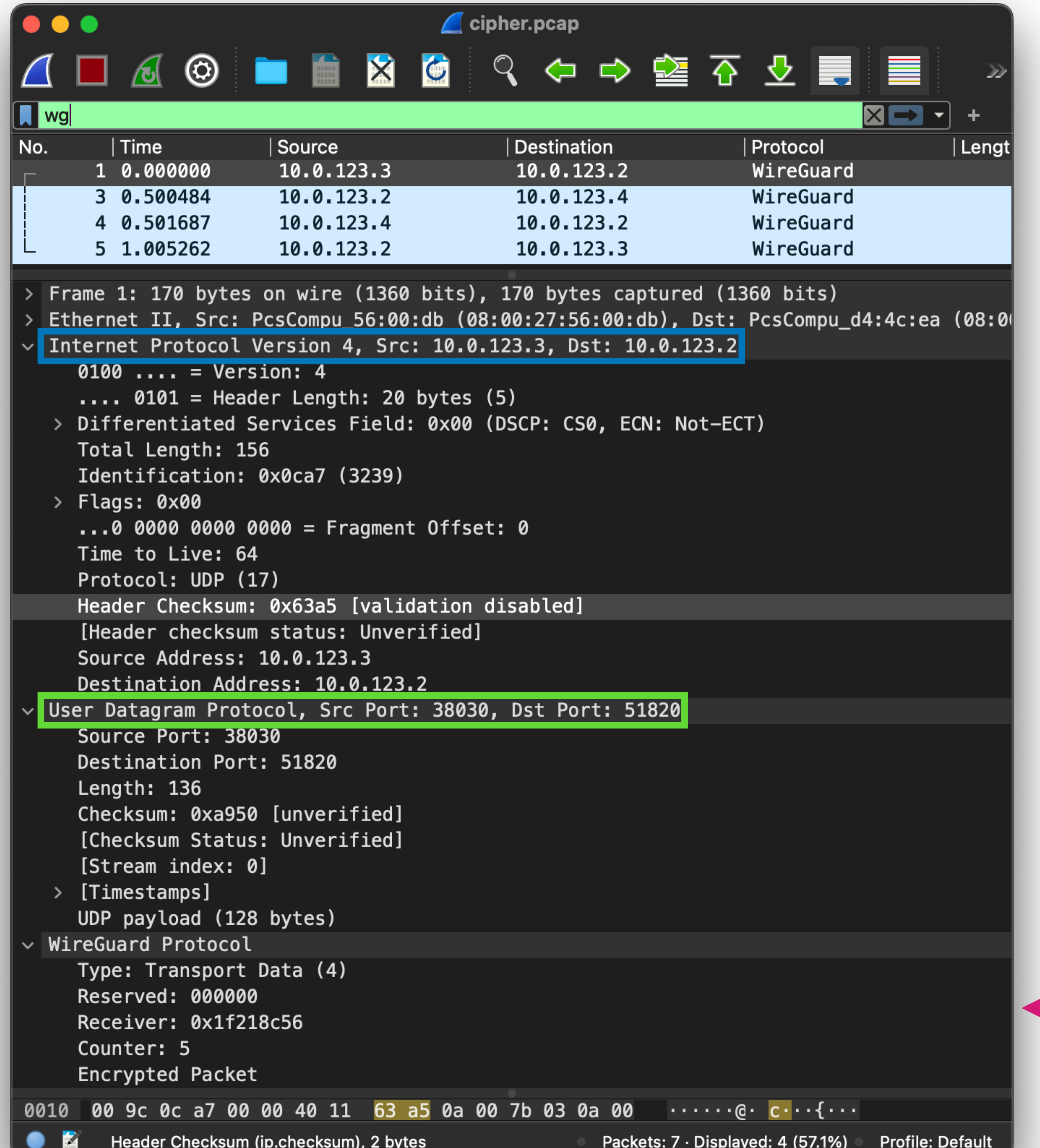
Pila de protocolos

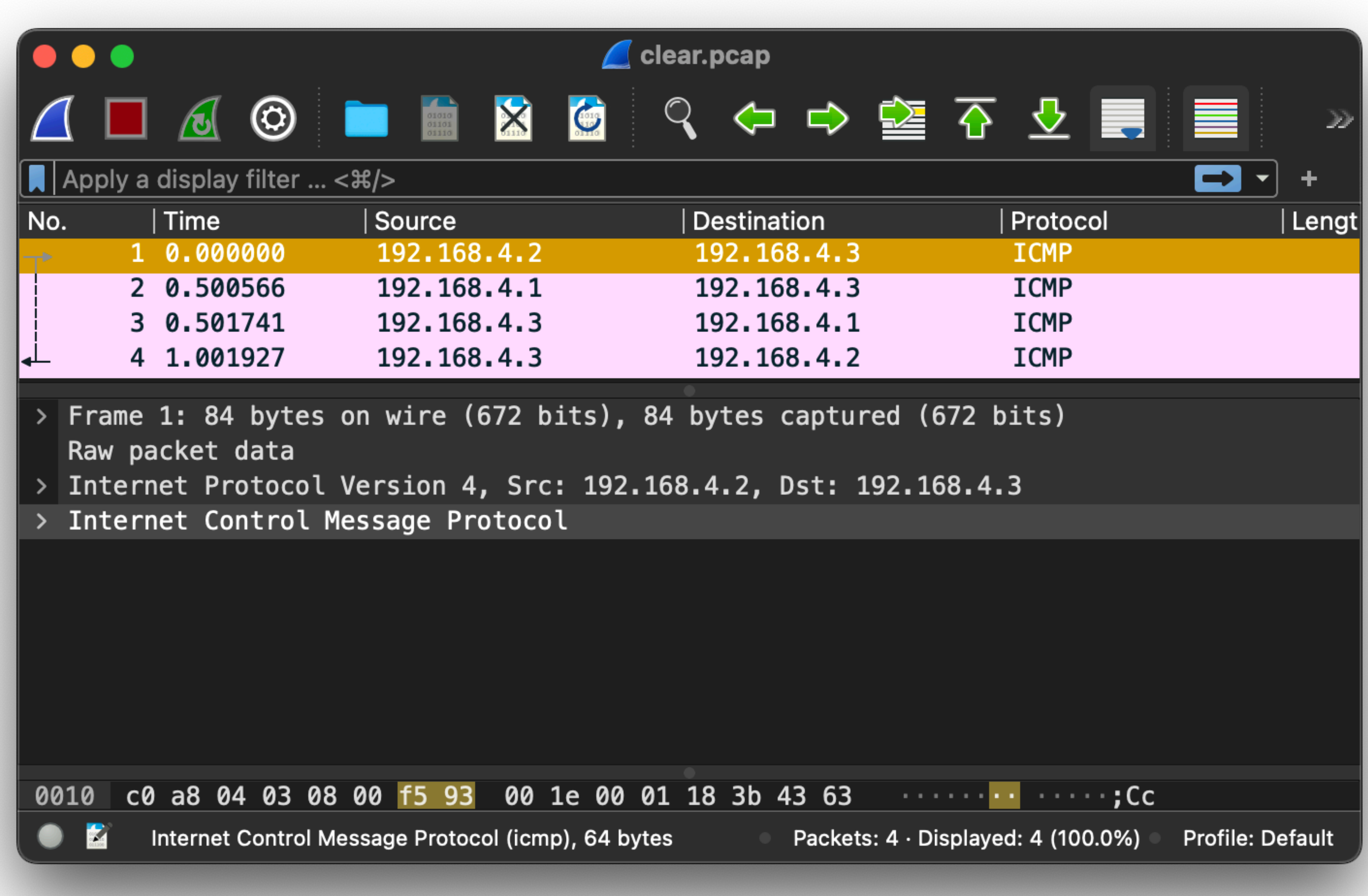


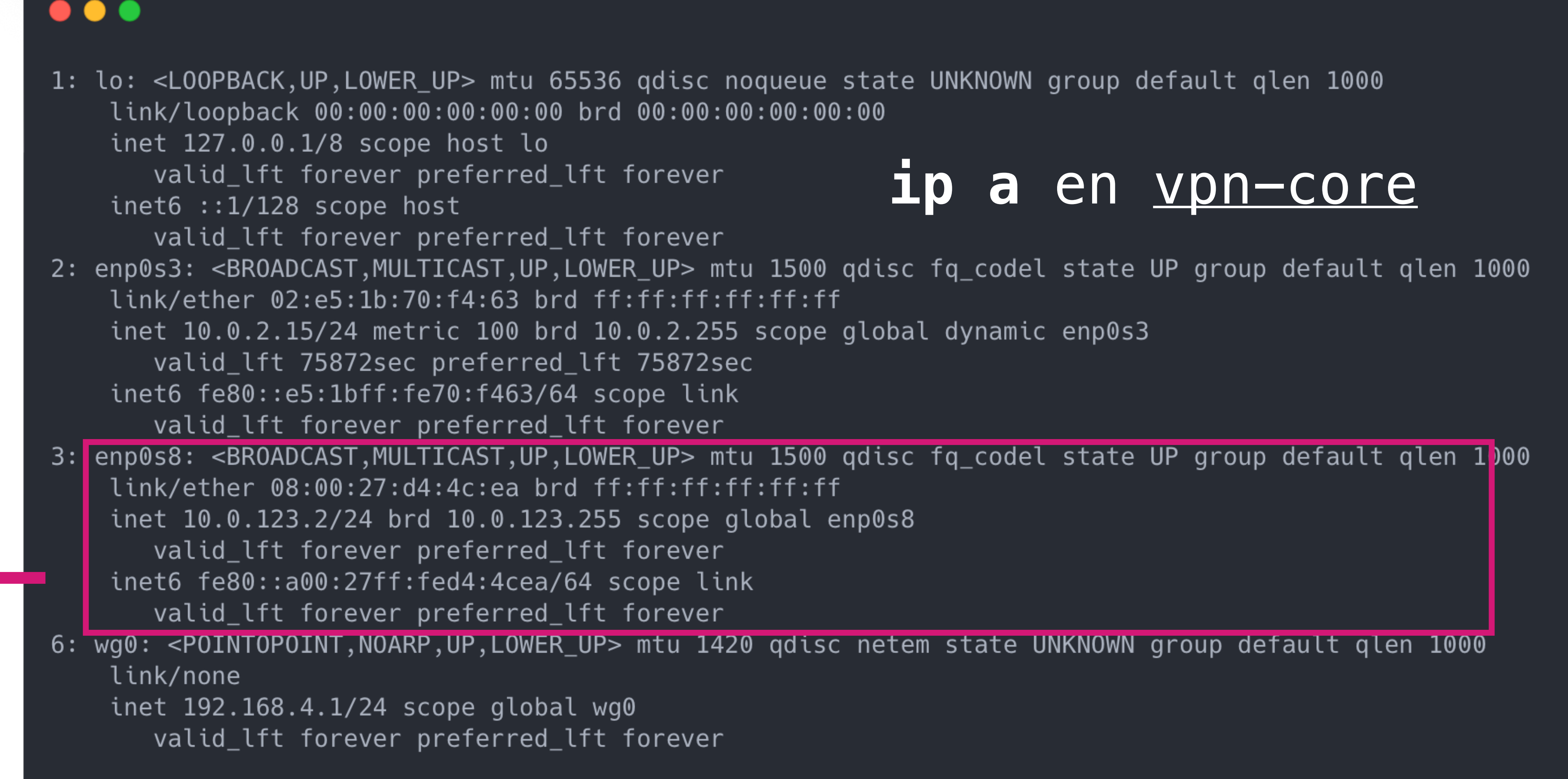




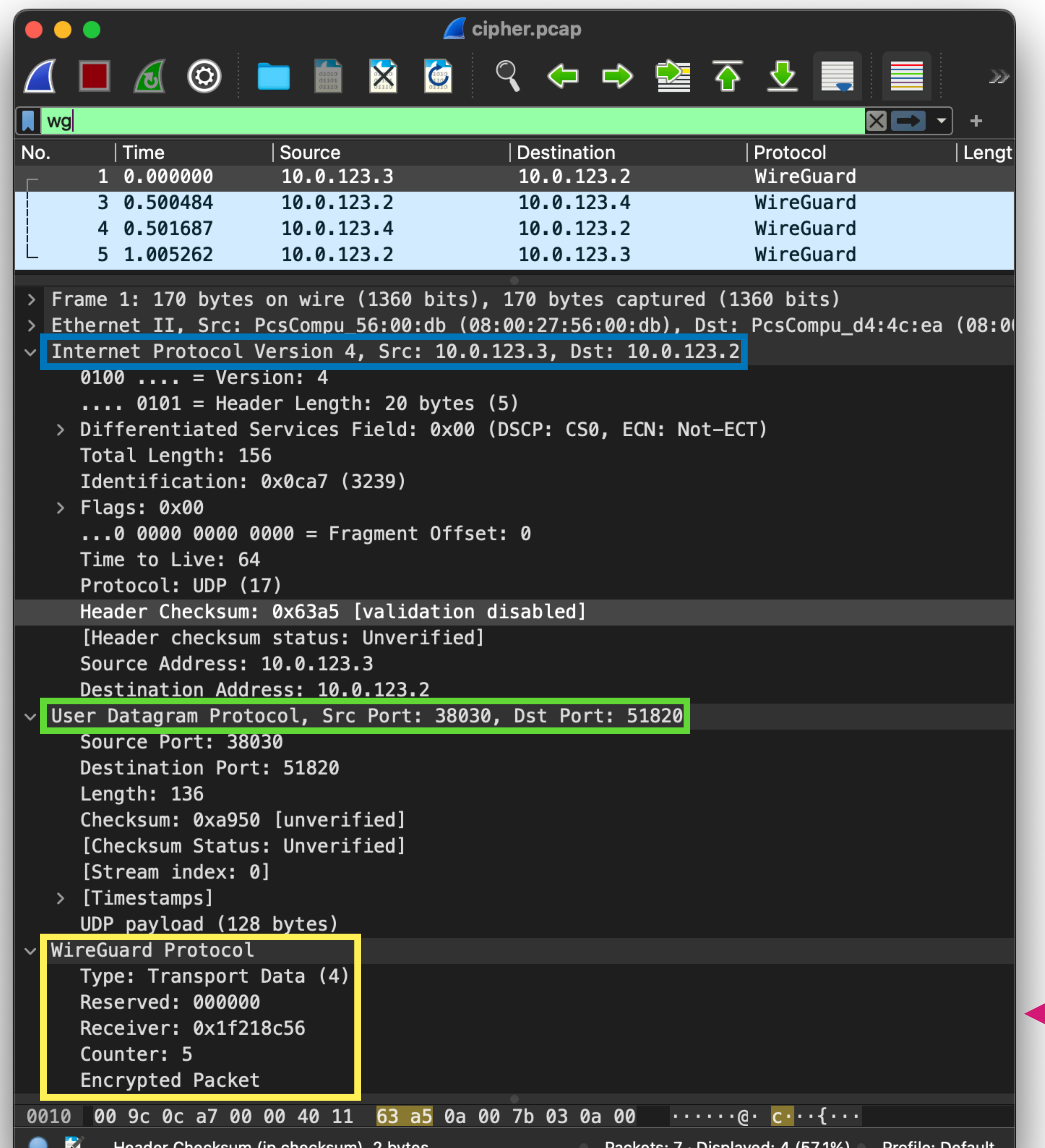
Pila de protocolos

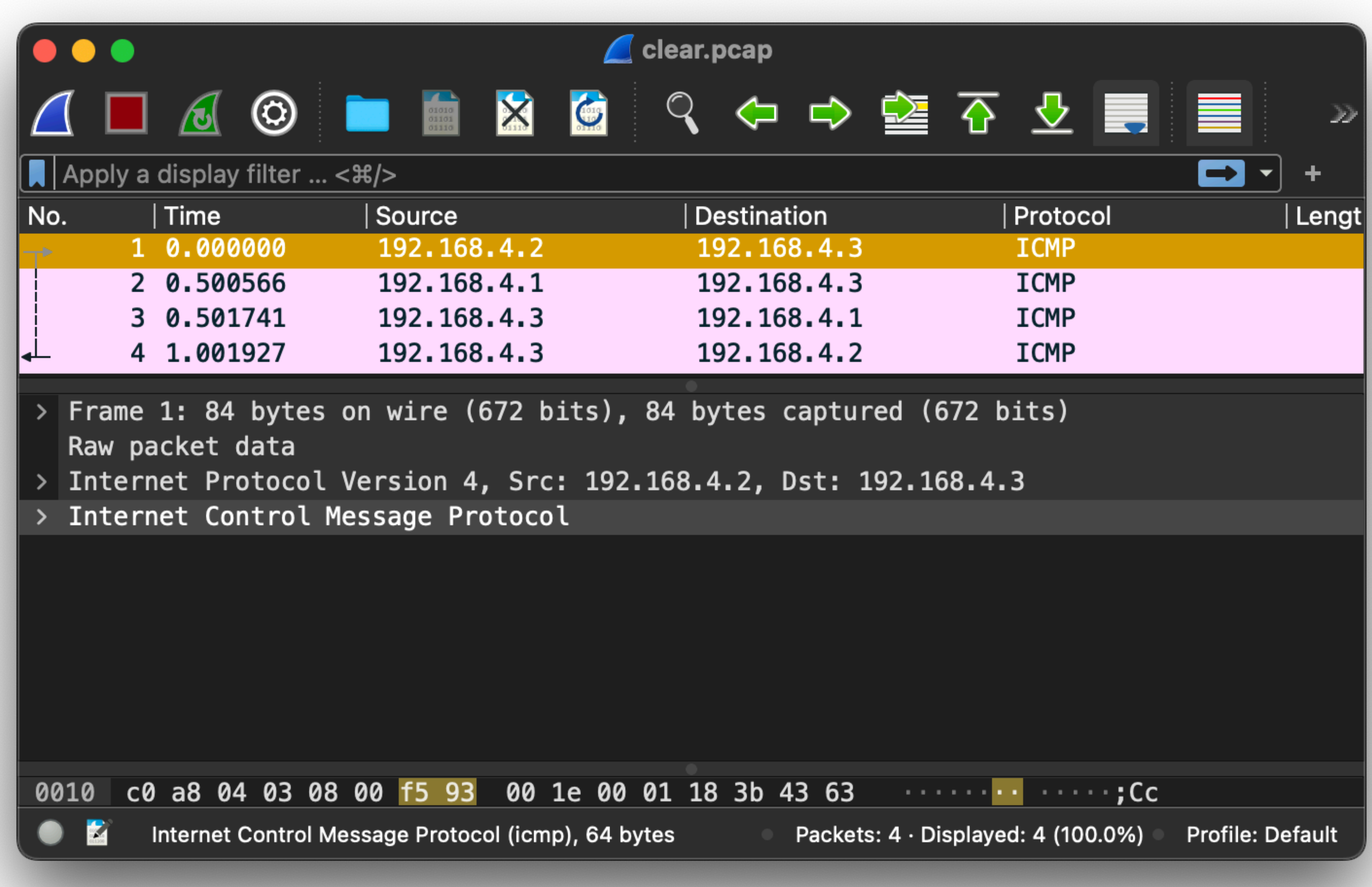


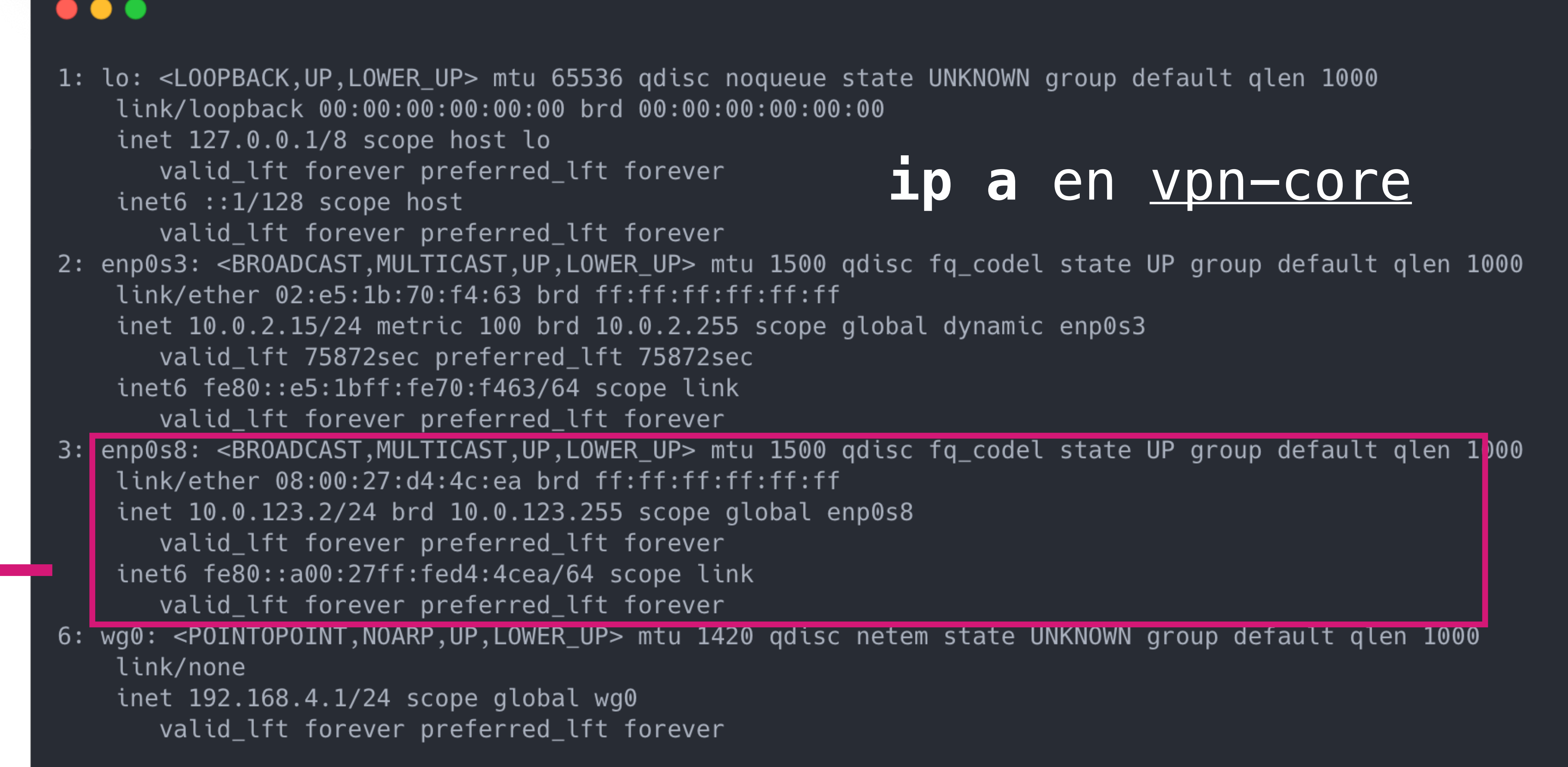




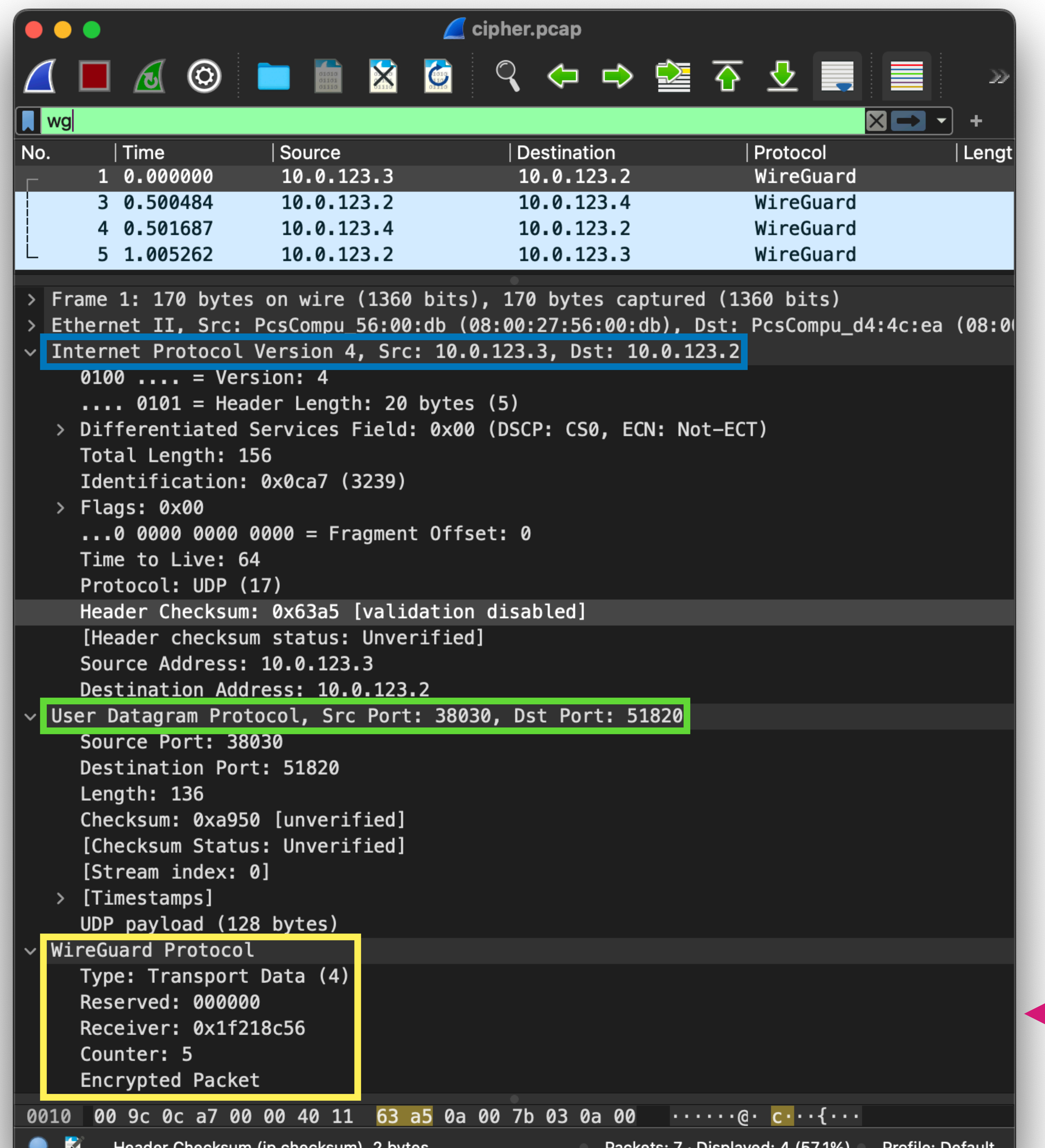
Pila de protocolos

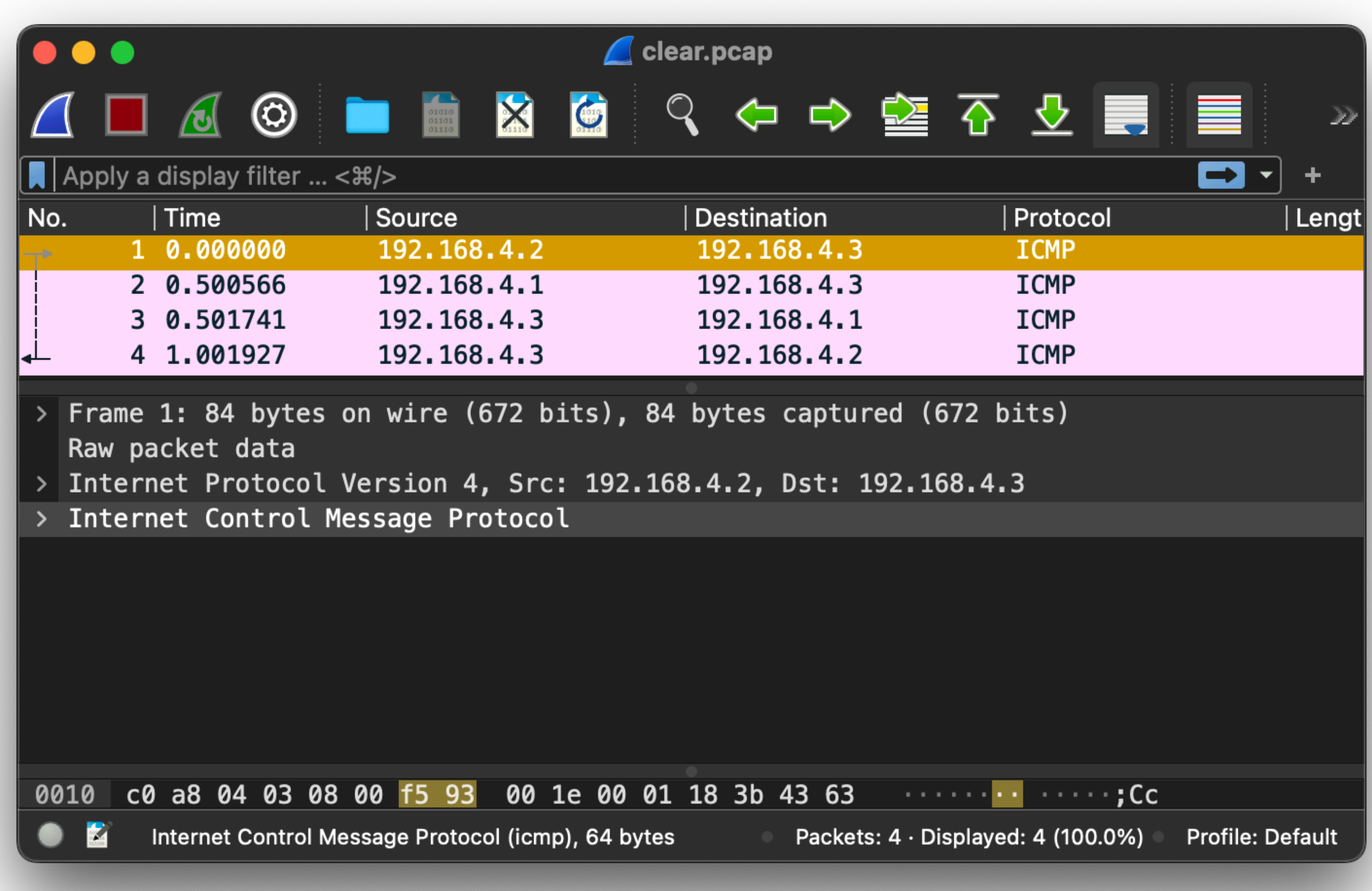


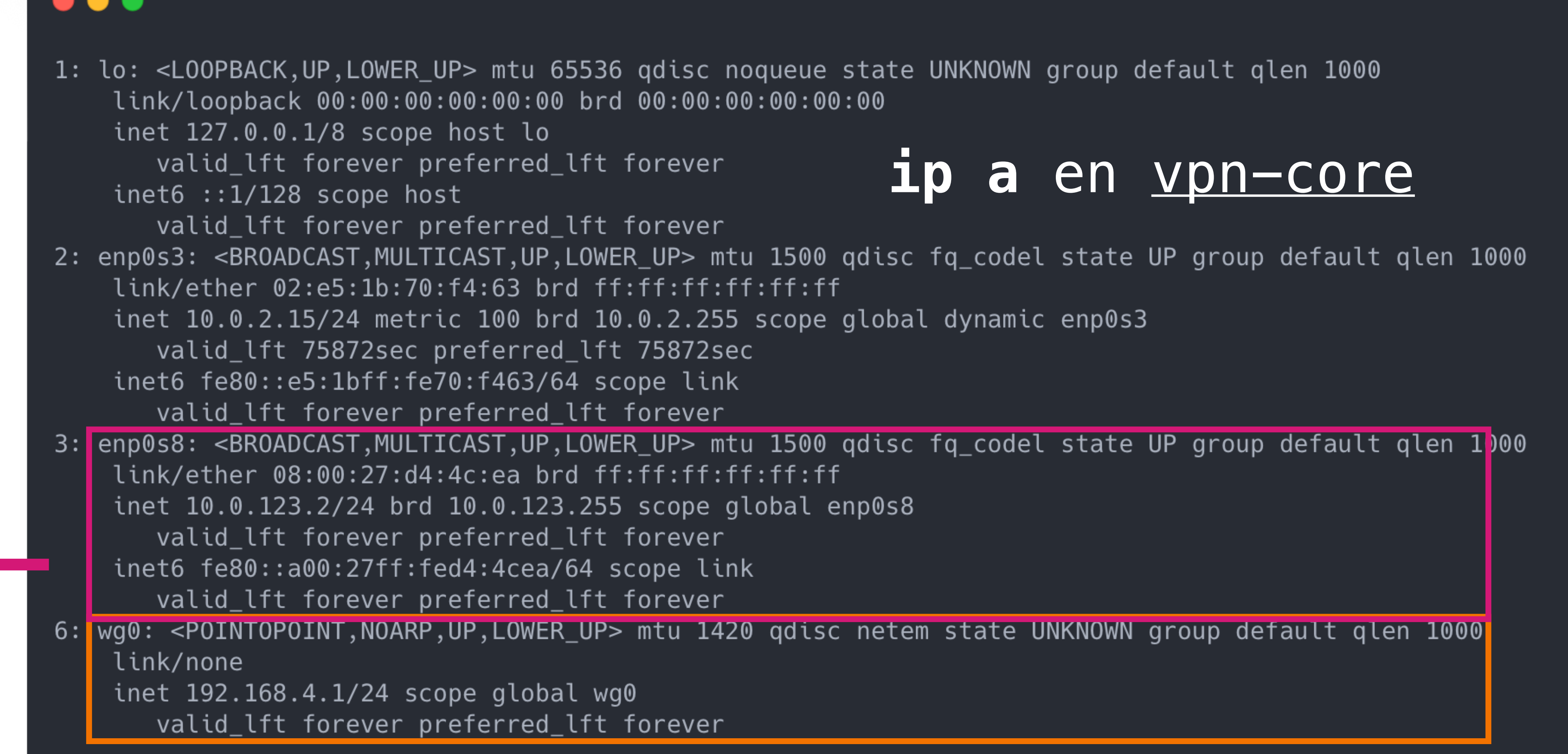




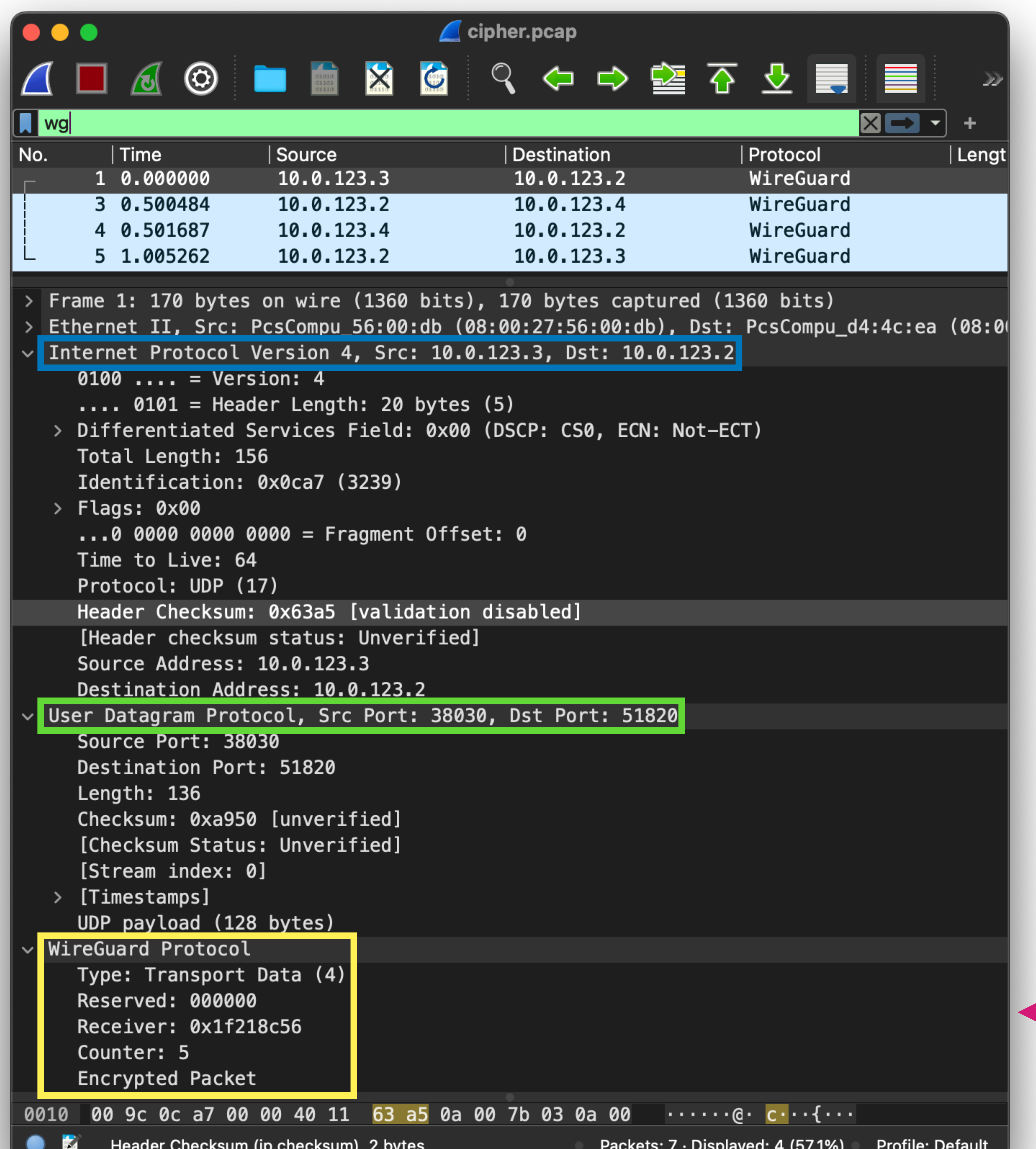
Pila de protocolos

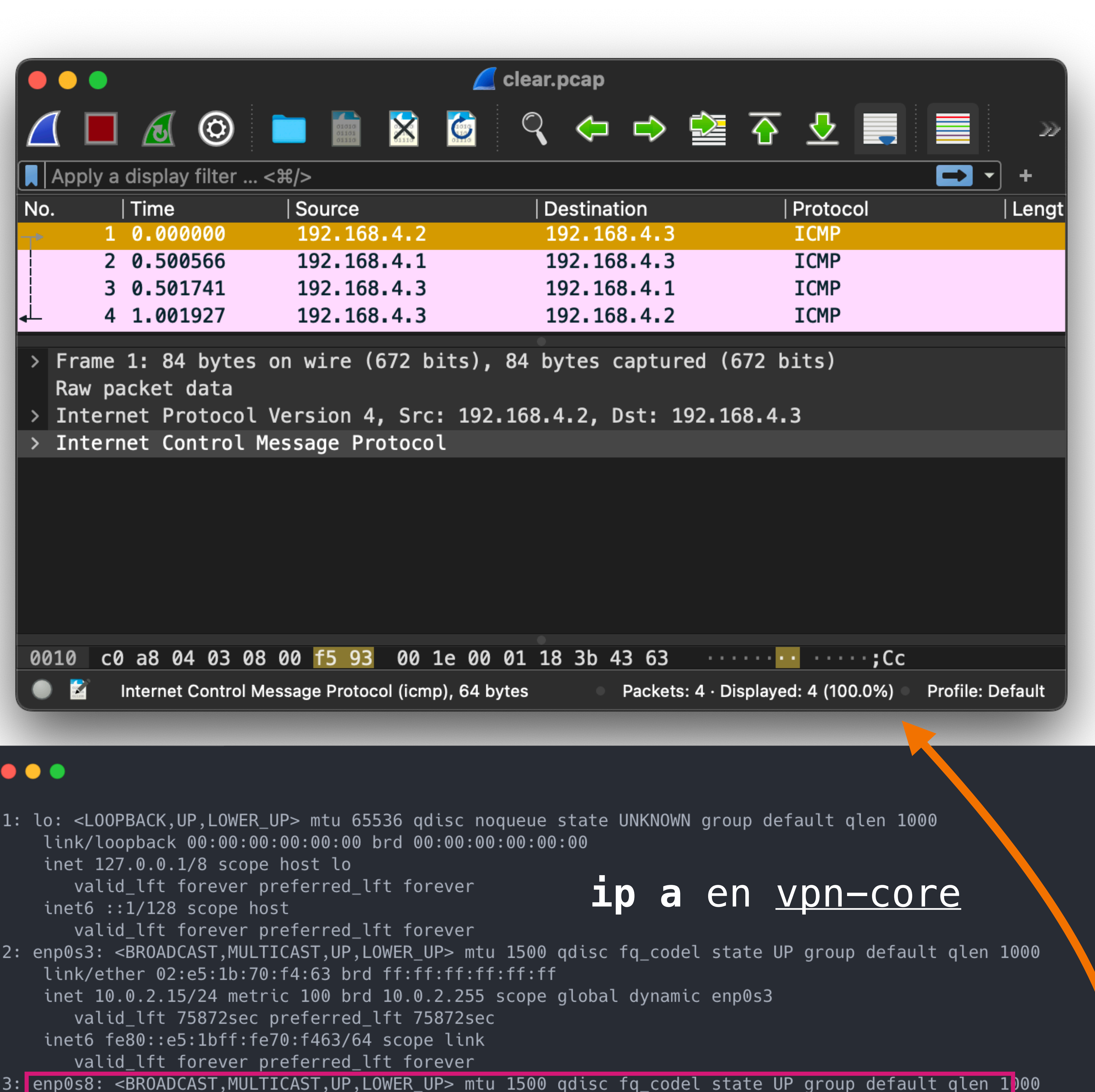




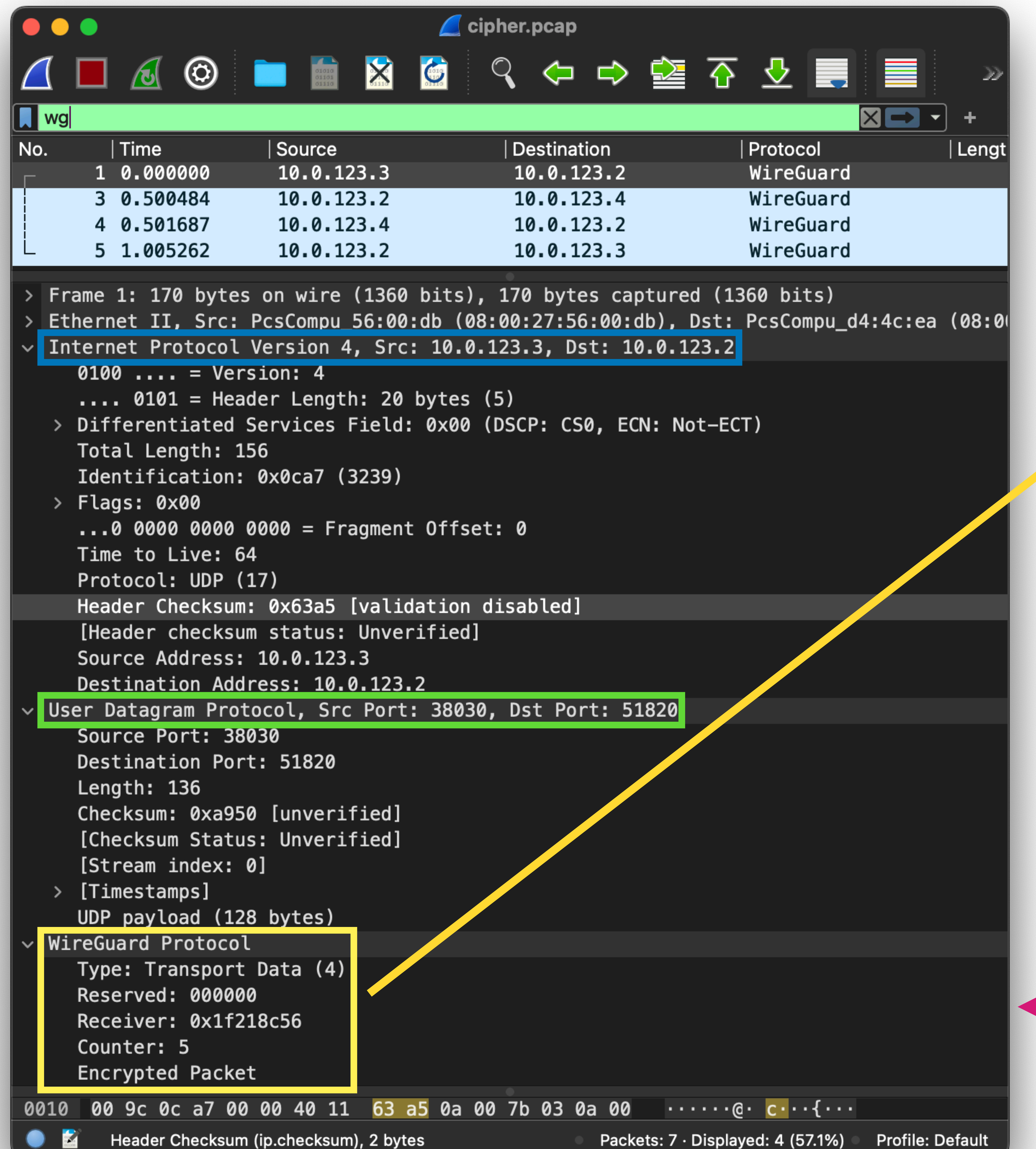


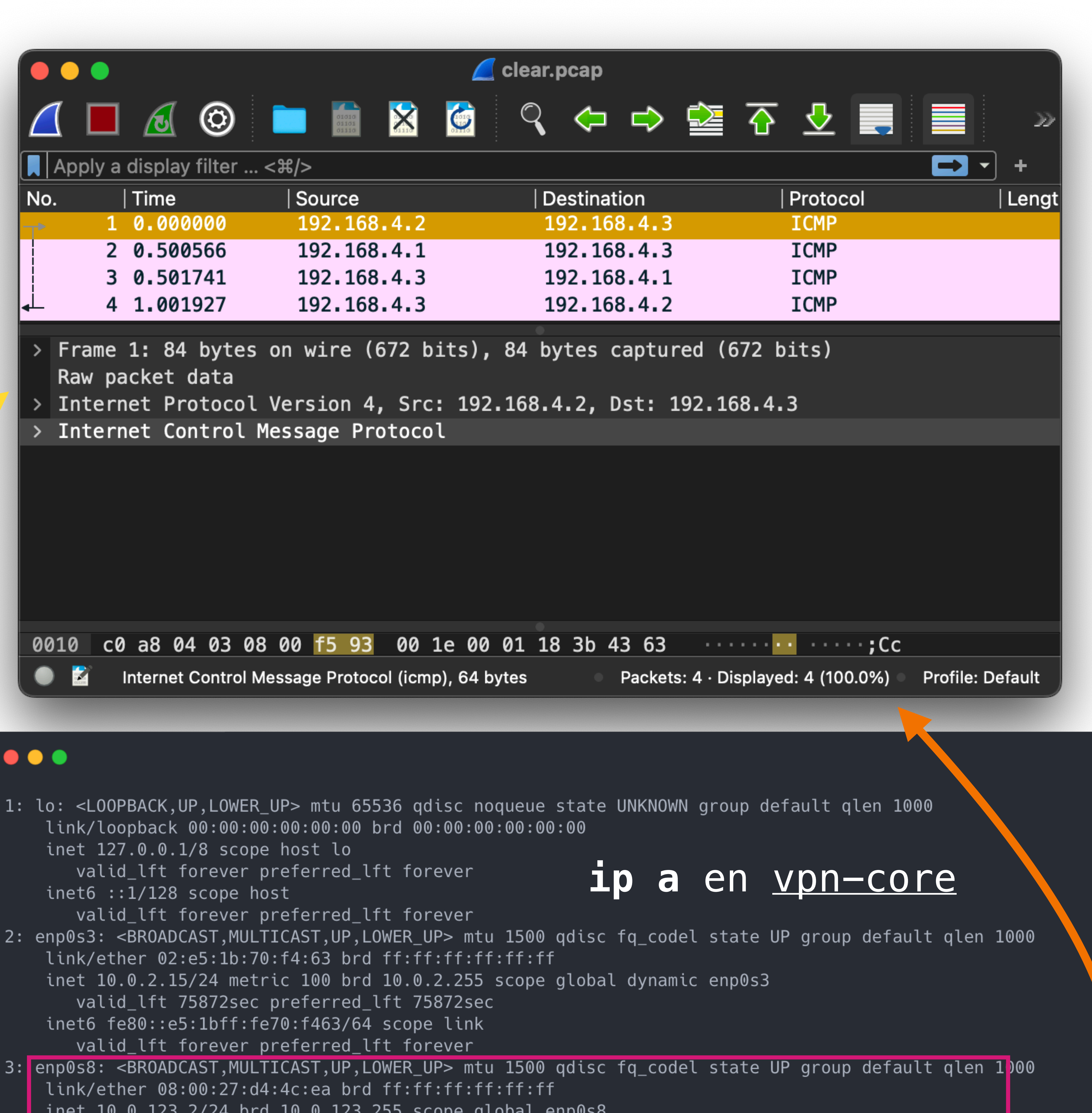
Pila de protocolos



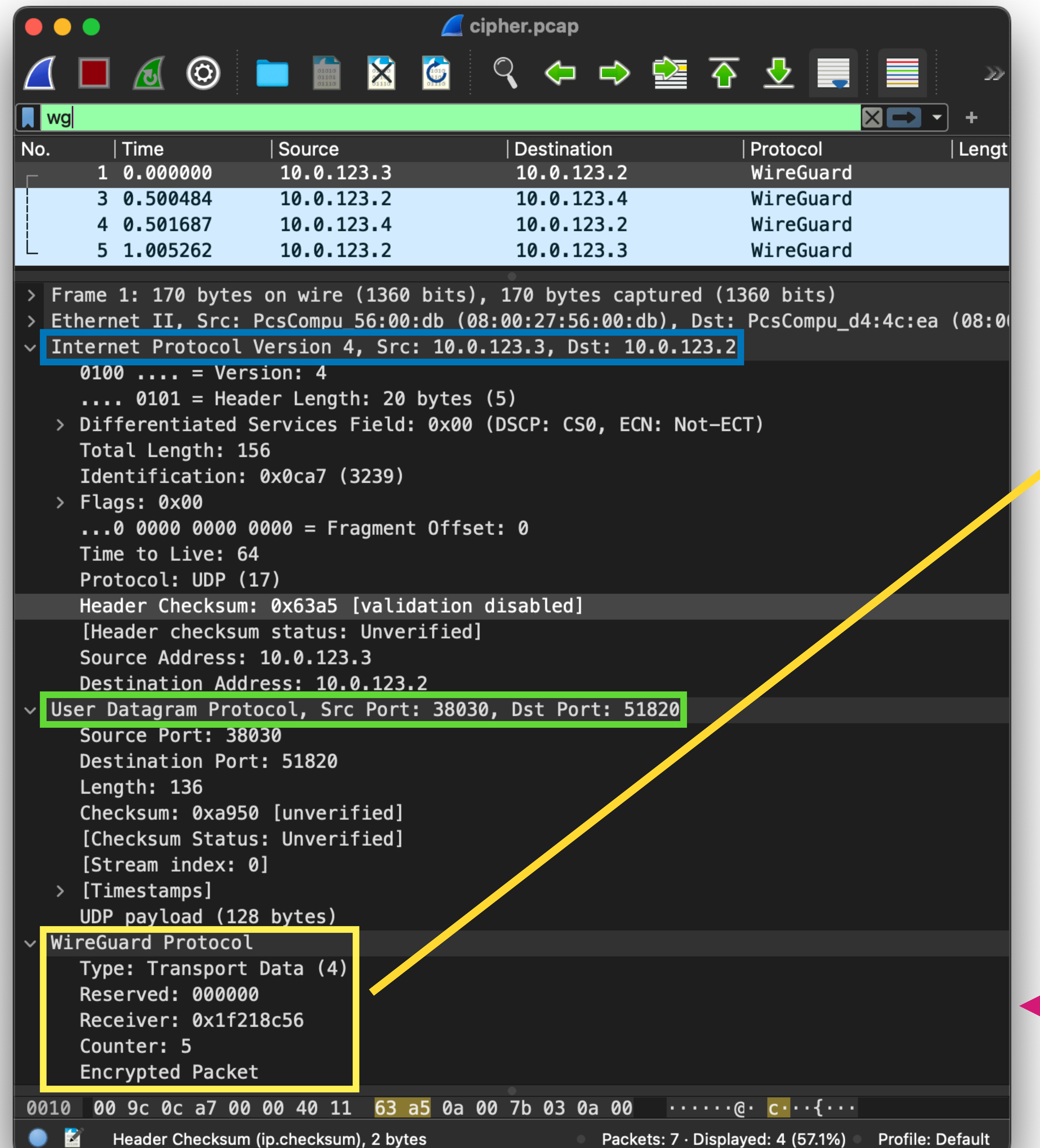


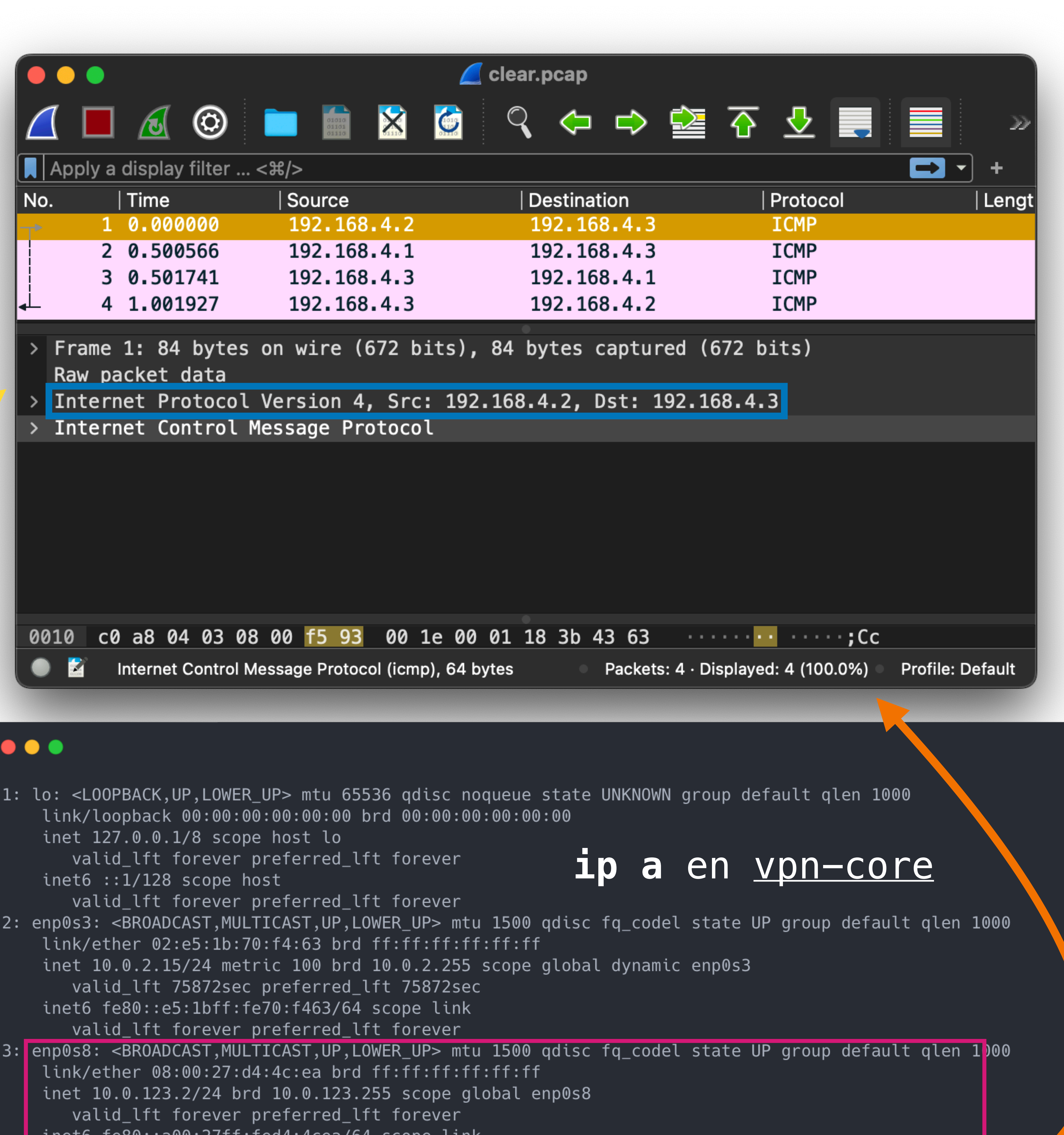
Pila de protocolos



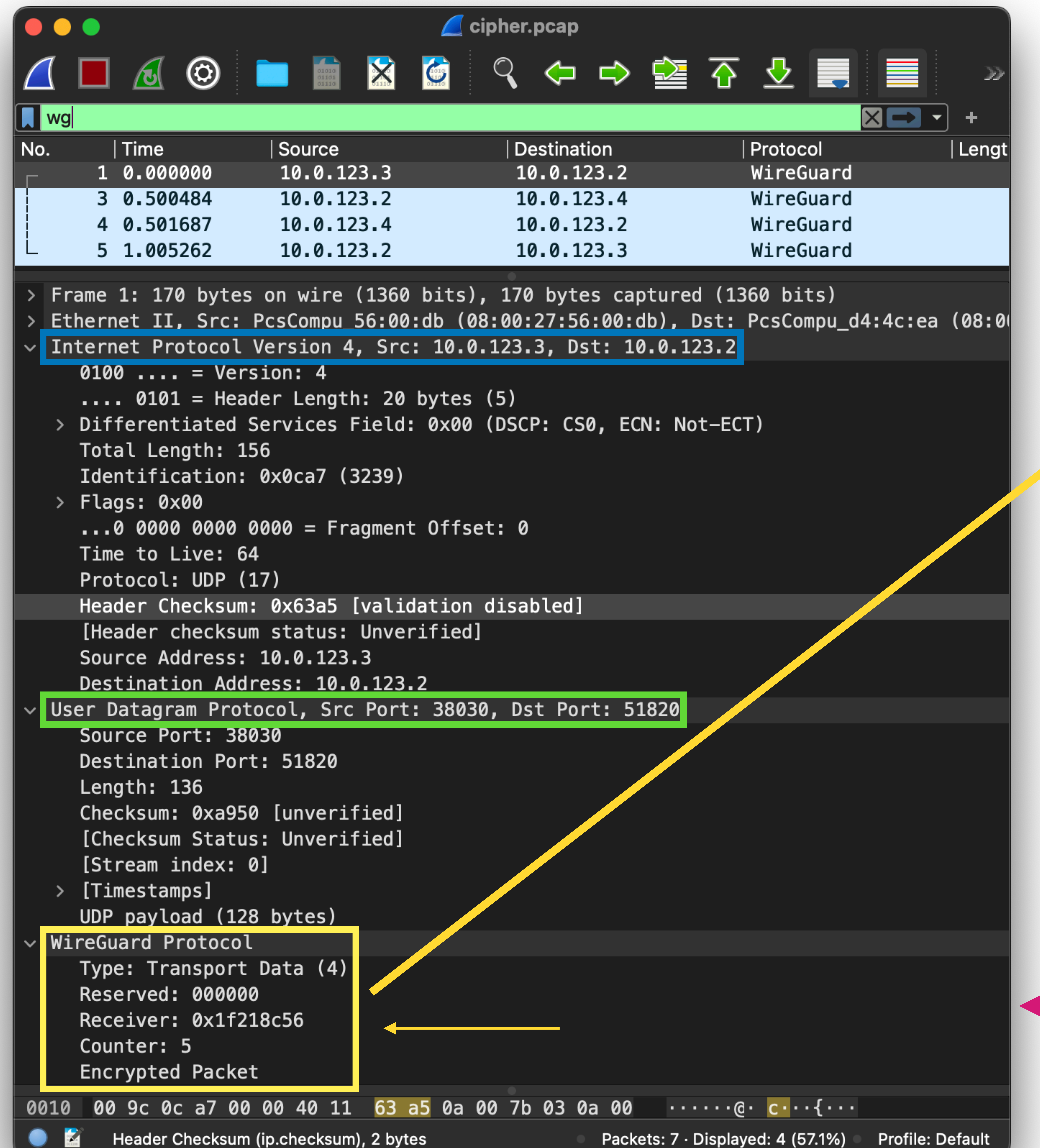


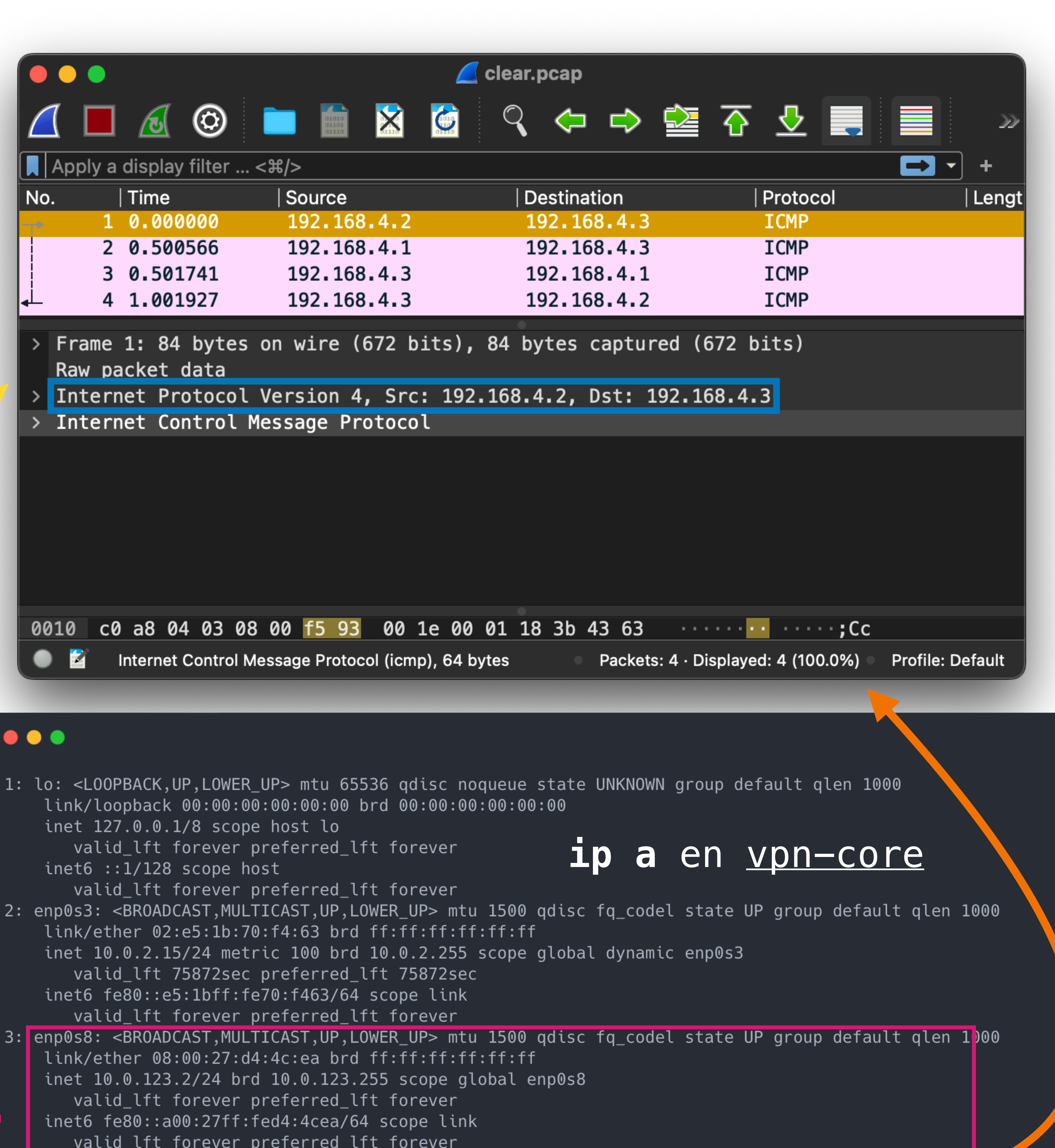
Pila de protocolos



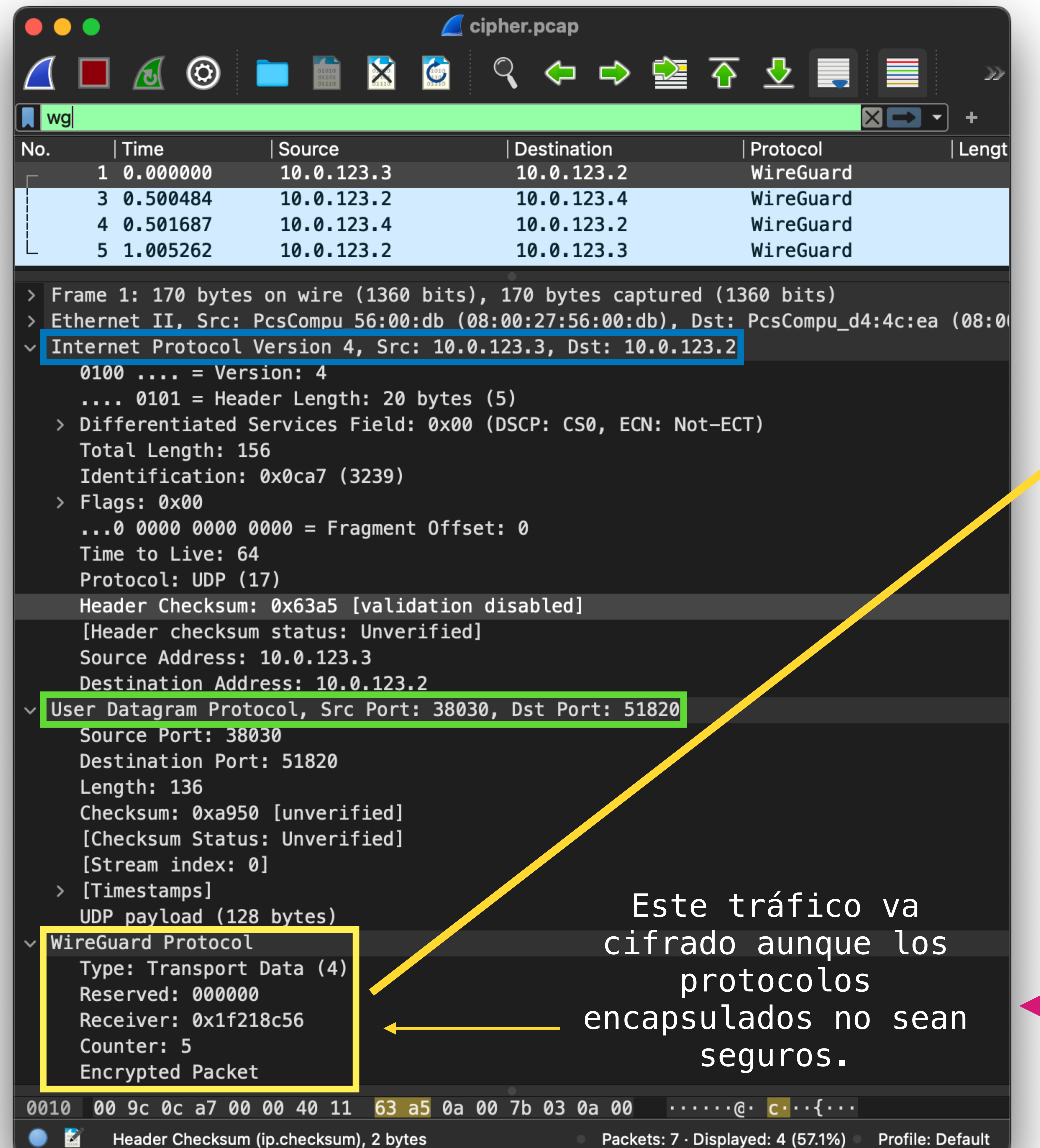


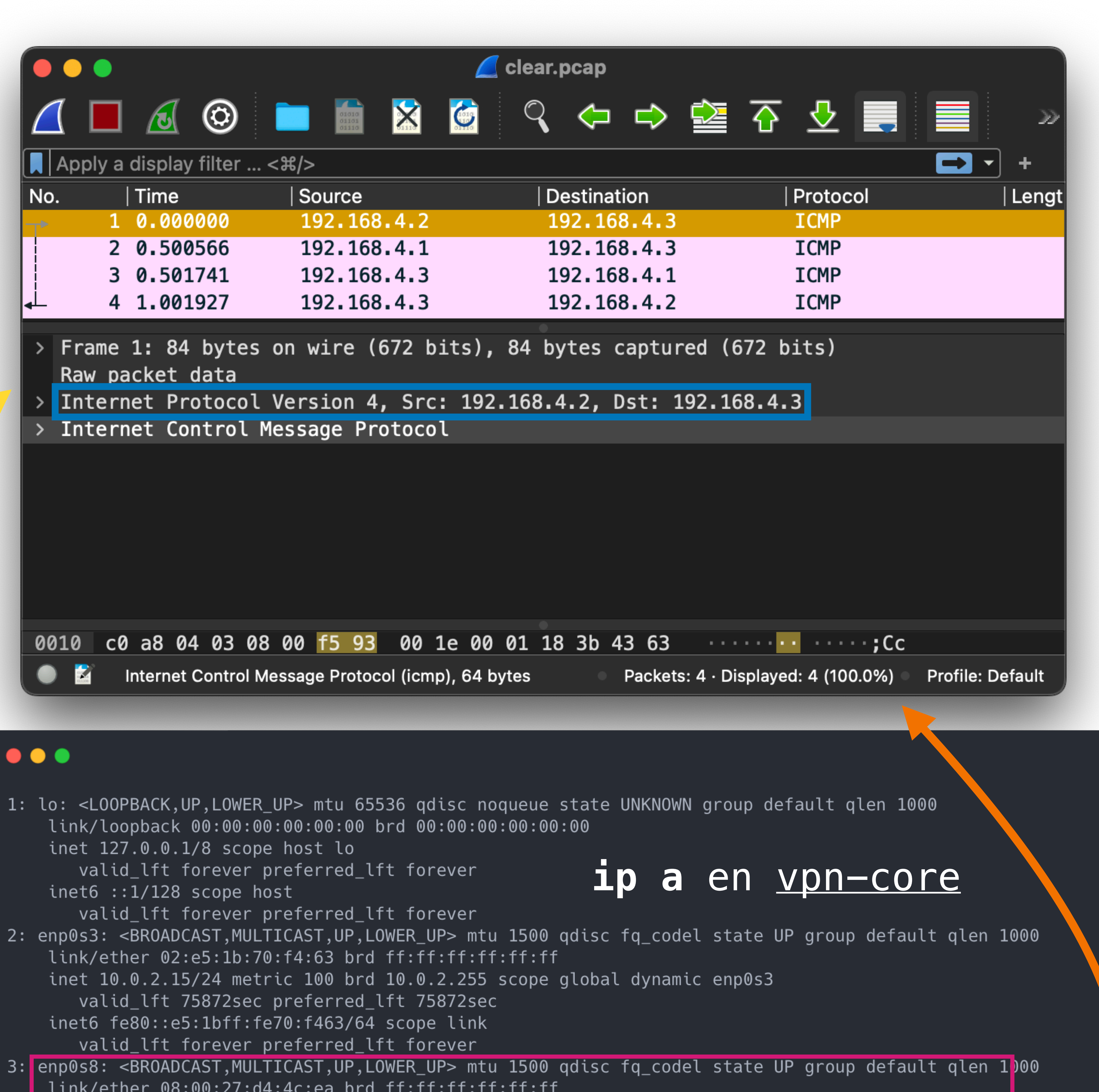
Pila de protocolos



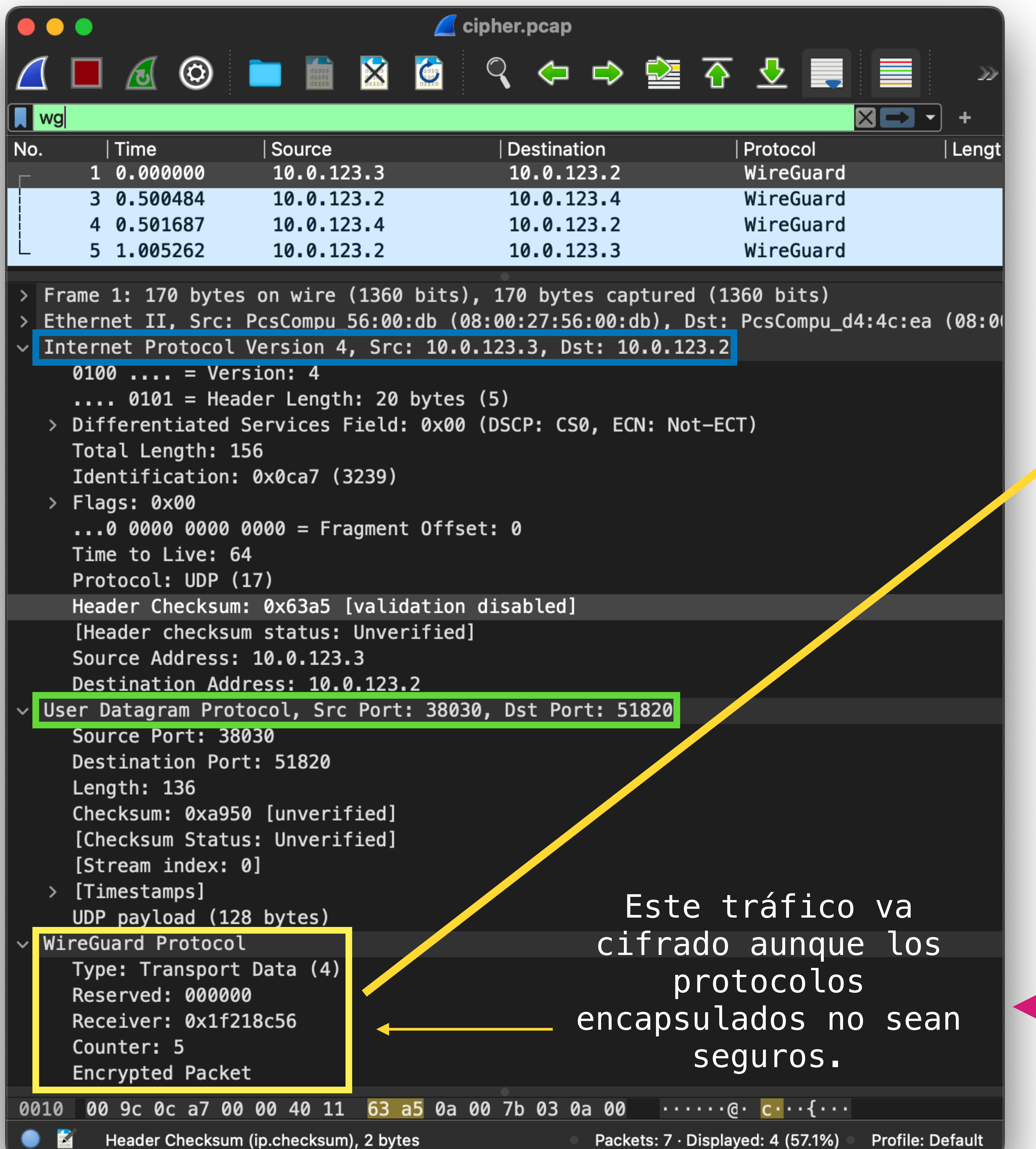


Pila de protocolos





Pila de protocolos

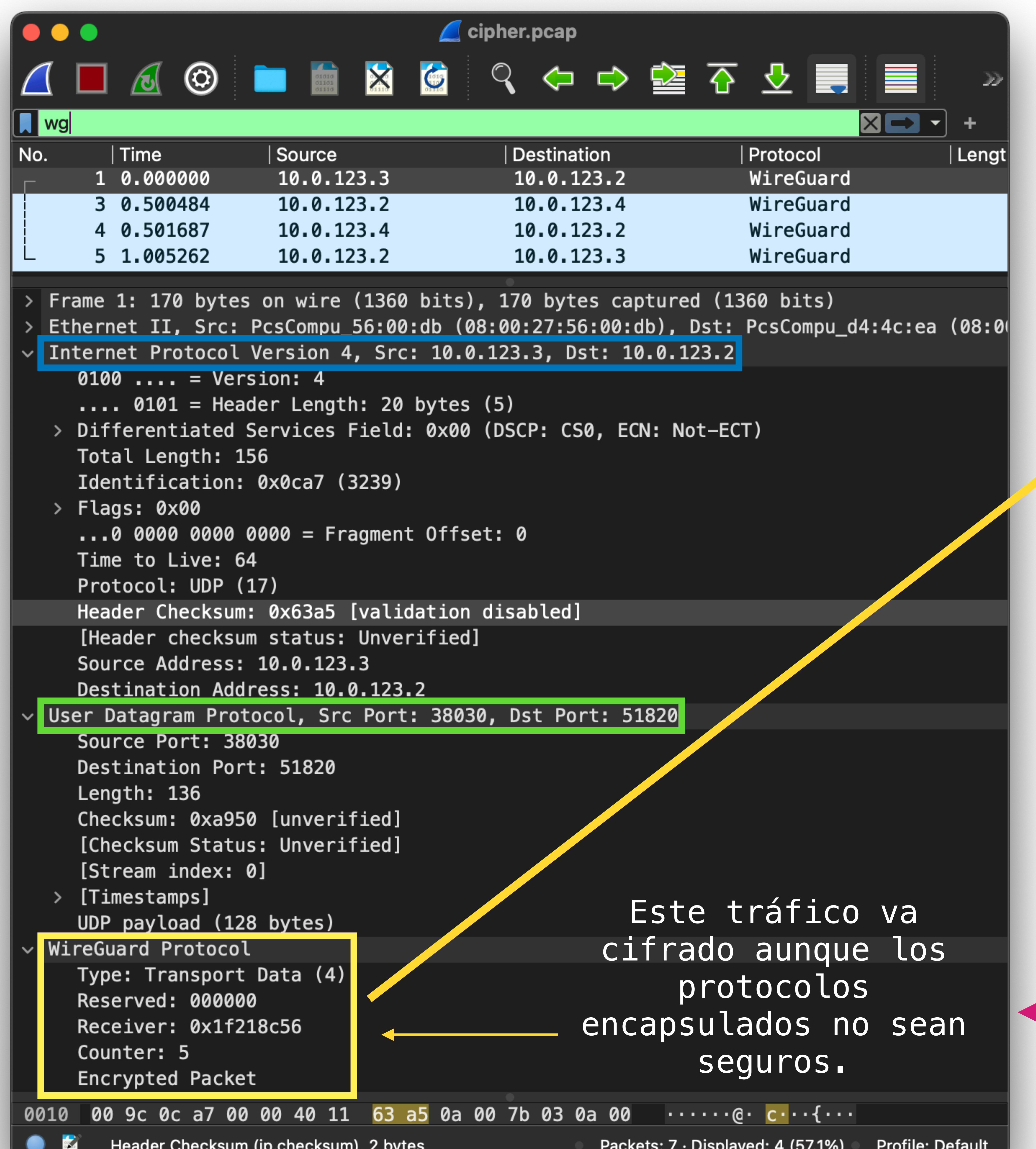



```
clear.pcap
Apply a display filter ... <⌘/>
No. | Time | Source | Destination | Protocol | Lengt
1 0.000000 192.168.4.2 192.168.4.3 ICMP
2 0.500566 192.168.4.1 192.168.4.3 ICMP
3 0.501741 192.168.4.3 192.168.4.1 ICMP
4 1.001927 192.168.4.3 192.168.4.2 ICMP
> Frame 1: 84 bytes on wire (672 bits), 84 bytes captured (672 bits)
  Raw packet data
> Internet Protocol Version 4, Src: 192.168.4.2, Dst: 192.168.4.3
> Internet Control Message Protocol
0010  c0 a8 04 03 08 00 f5 93  00 1e 00 01 18 3b 43 63   ;Cc
Internet Control Message Protocol (icmp), 64 bytes   Packets: 4 · Displayed: 4 (100.0%)   Profile: Default
```

ip a en vpn-core

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 02:e5:1b:70:f4:63 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 metric 100 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 75872sec preferred_lft 75872sec
    inet6 fe80::e5:1bff:fe70:f463/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:d4:4c:ea brd ff:ff:ff:ff:ff:ff
    inet 10.0.123.2/24 brd 10.0.123.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fed4:4cea/64 scope link
       valid_lft forever preferred_lft forever
6: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc netem state UNKNOWN group default qlen 1000
    link/none
    inet 192.168.4.1/24 scope global wg0
       valid_lft forever preferred_lft forever
```

Pila de protocolos



clear.pcap

Apply a display filter ... <⌘/>

No.	Time	Source	Destination	Protocol
1	0.000000	192.168.4.2	192.168.4.3	ICMP
2	0.500566	192.168.4.1	192.168.4.3	ICMP
3	0.501741	192.168.4.3	192.168.4.1	ICMP
4	1.001927	192.168.4.3	192.168.4.2	ICMP

```
> Frame 1: 84 bytes on wire (672 bits), 84 bytes captured (672 bits)
  Raw packet data
> Internet Protocol Version 4, Src: 192.168.4.2, Dst: 192.168.4.3
> Internet Control Message Protocol
```

Este datagrama pertenece a la red «overlay» definida en la configuración de la VPN.

```
0010  c0 a8 04 03 08 00 f5 93  00 1e 00 01 18 3b 43 63   ········ ·····;Cc
```

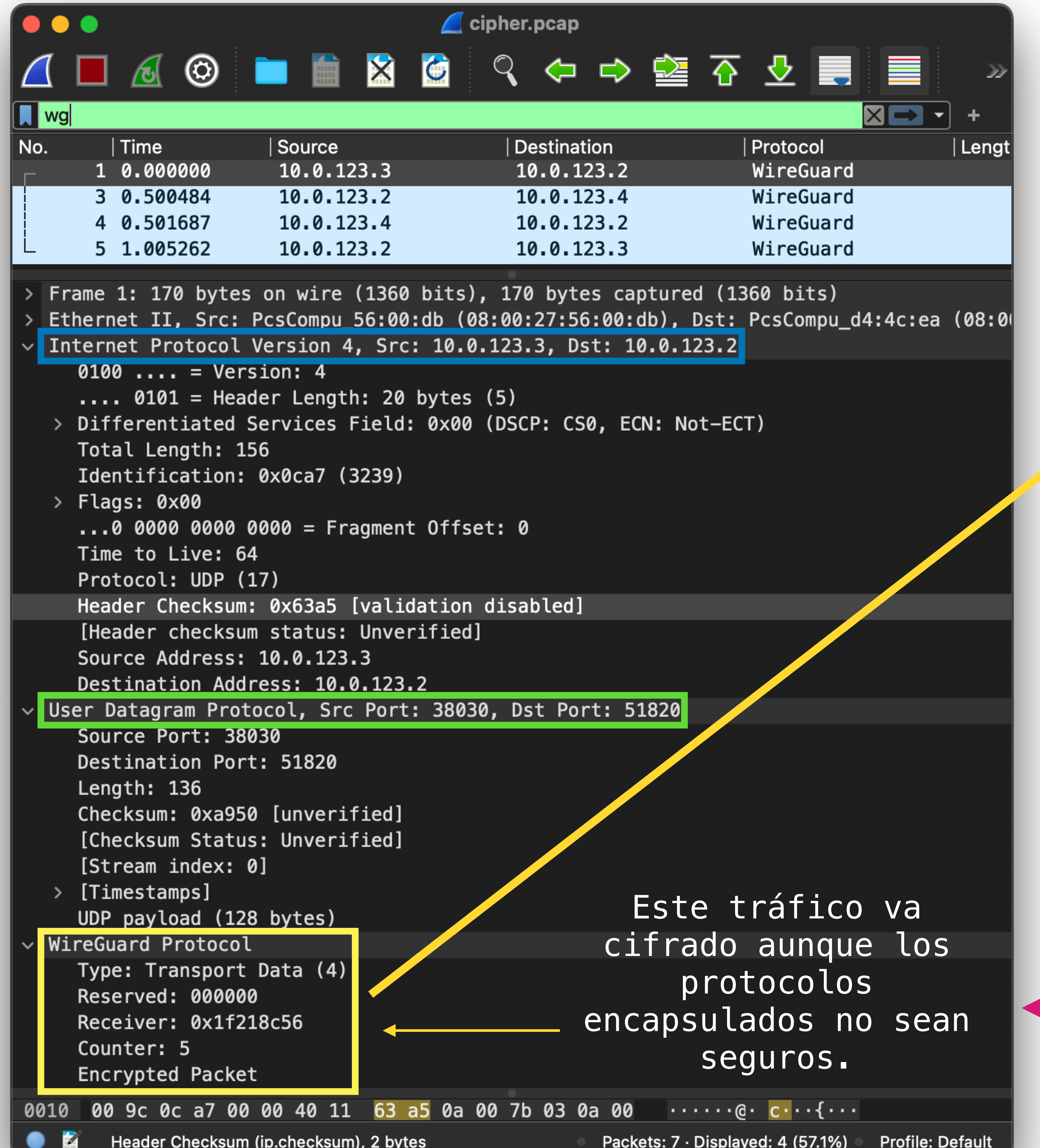
Internet Control Message Protocol (icmp), 64 bytes · Packets: 4 · Displayed: 4 (100.0%) · Profile: Default

ip a en vpn-core

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 02:e5:1b:70:f4:63 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 metric 100 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 75872sec preferred_lft 75872sec
    inet6 fe80::e5:1bff:fe70:f463/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:d4:4c:ea brd ff:ff:ff:ff:ff:ff
    inet 10.0.123.2/24 brd 10.0.123.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fed4:4cea/64 scope link
       valid_lft forever preferred_lft forever
6: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc netem state UNKNOWN group default qlen 1000
    link/none
    inet 192.168.4.1/24 scope global wg0
       valid_lft forever preferred_lft forever
```

Pila de protocolos

Capturas visualizadas con WireShark



clear.pcap

Apply a display filter ... <⌘/>

No.	Time	Source	Destination	Protocol	Lengt
1	0.000000	192.168.4.2	192.168.4.3	ICMP	
2	0.500566	192.168.4.1	192.168.4.3	ICMP	
3	0.501741	192.168.4.3	192.168.4.1	ICMP	
4	1.001927	192.168.4.3	192.168.4.2	ICMP	

Frame 1: 84 bytes on wire (672 bits), 84 bytes captured (672 bits)
Raw packet data
Internet Protocol Version 4, Src: 192.168.4.2, Dst: 192.168.4.3
Internet Control Message Protocol

Este datagrama pertenece a la red «overlay» definida en la configuración de la VPN.

0010 c0 a8 04 03 08 00 f5 93 00 1e 00 01 18 3b 43 63 ·······;Cc

Internet Control Message Protocol (icmp), 64 bytes · Packets: 4 · Displayed: 4 (100.0%) · Profile: Default

ip a en vpn-core

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 02:e5:1b:70:f4:63 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 metric 100 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 75872sec preferred_lft 75872sec
    inet6 fe80::e5:1bff:fe70:f463/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:d4:4c:ea brd ff:ff:ff:ff:ff:ff
    inet 10.0.123.2/24 brd 10.0.123.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fed4:4cea/64 scope link
       valid_lft forever preferred_lft forever
6: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc netem state UNKNOWN group default qlen 1000
    link/none
    inet 192.168.4.1/24 scope global wg0
       valid_lft forever preferred_lft forever
```

A day in the life…

client-a

vpn-core

client-b

A day in the life…

client-a

vpn-core

client-b

ICMP

A day in the life…

client-a

vpn-core

client-b

ICMP

IPv4

A day in the life…

client-a

vpn-core

client-b

ICMP

IPv4

A day in the life…

client-a

vpn-core

client-b

ICMP

IPv4

Ciphertext

A day in the life…

client-a

vpn-core

client-b

ICMP

IPv4

Ciphertext

UDP

A day in the life...

client-a

vpn-core

client-b

ICMP

IPv4

Ciphertext

UDP

IPv4

A day in the life…

client-a

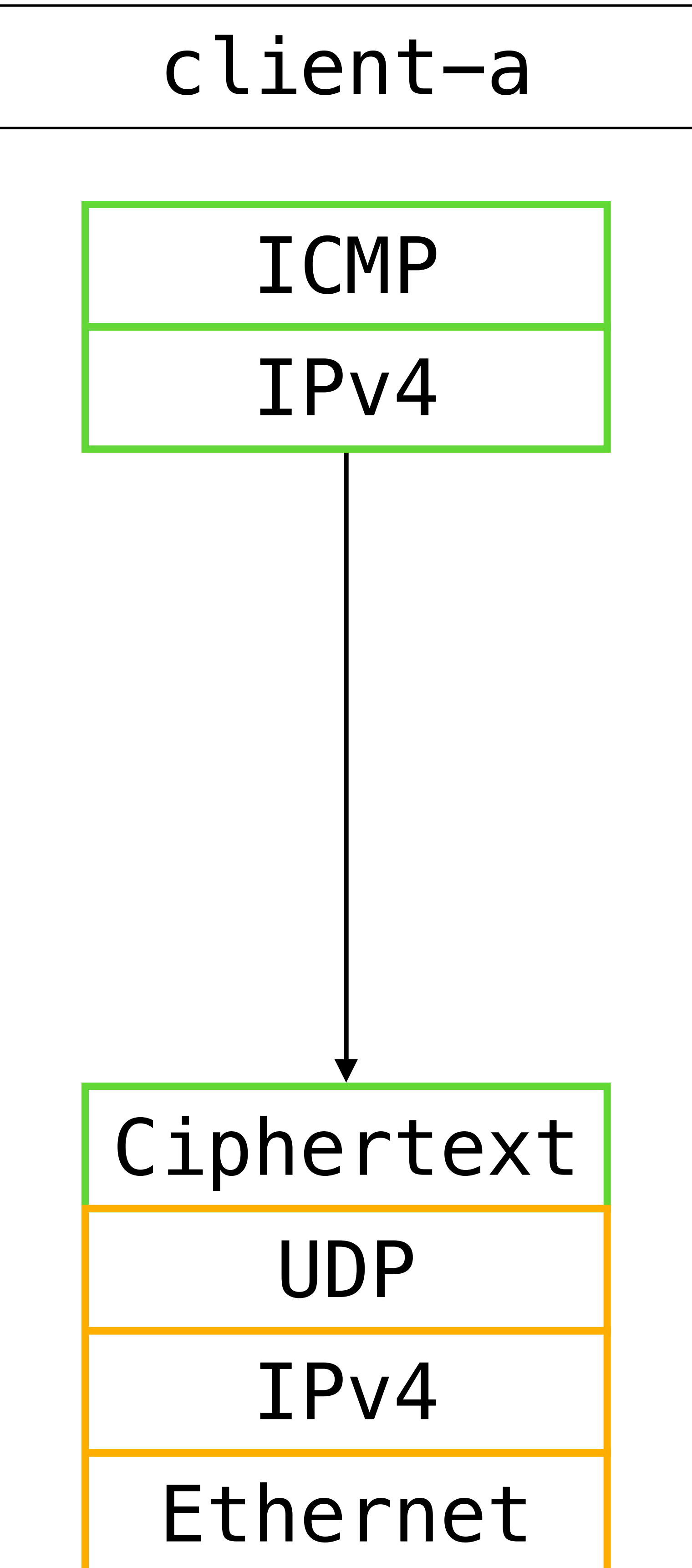
vpn-core

client-b

ICMP
IPv4

Ciphertext
UDP
IPv4
Ethernet

A day in the life…

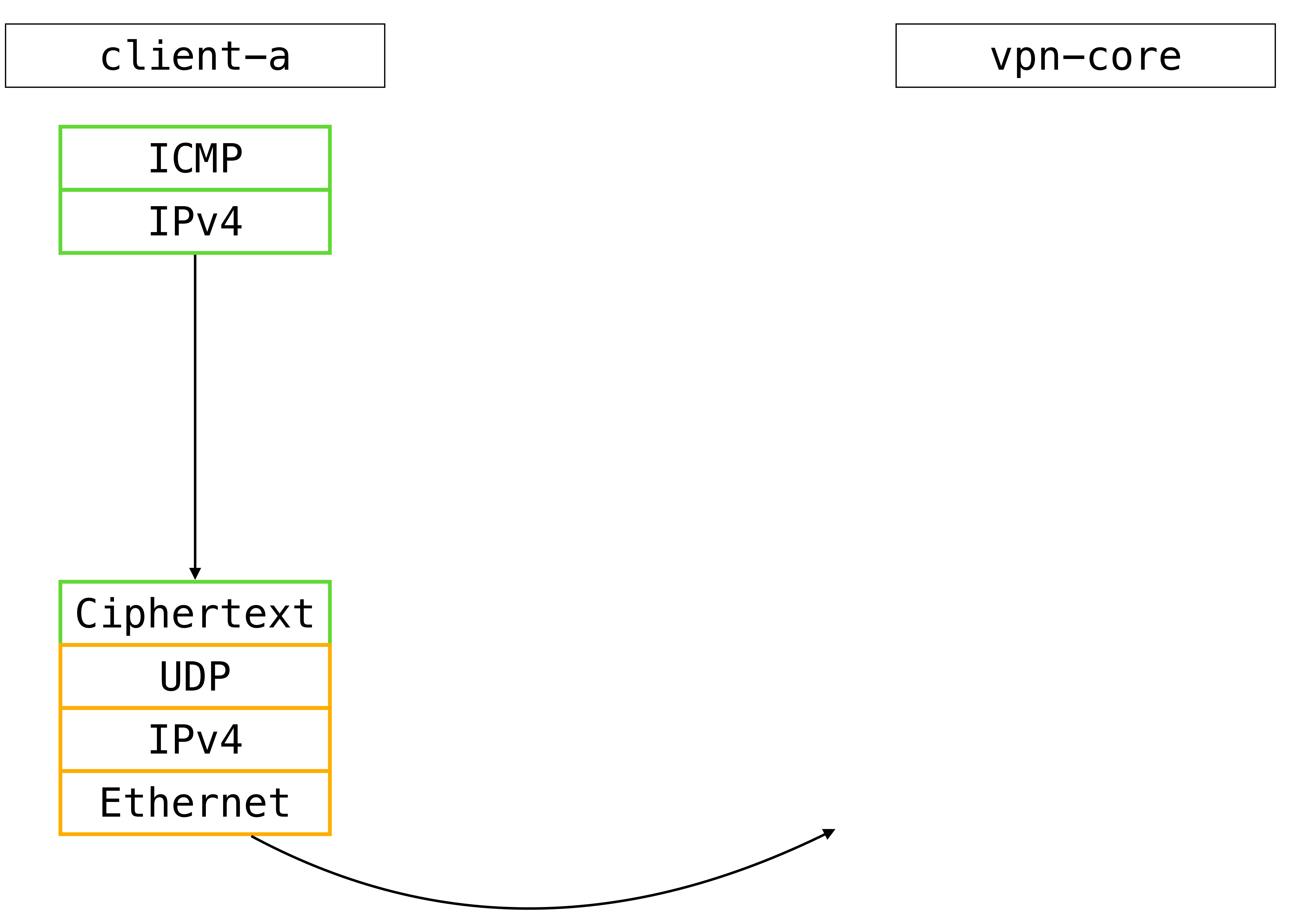


vpn-core

client-b

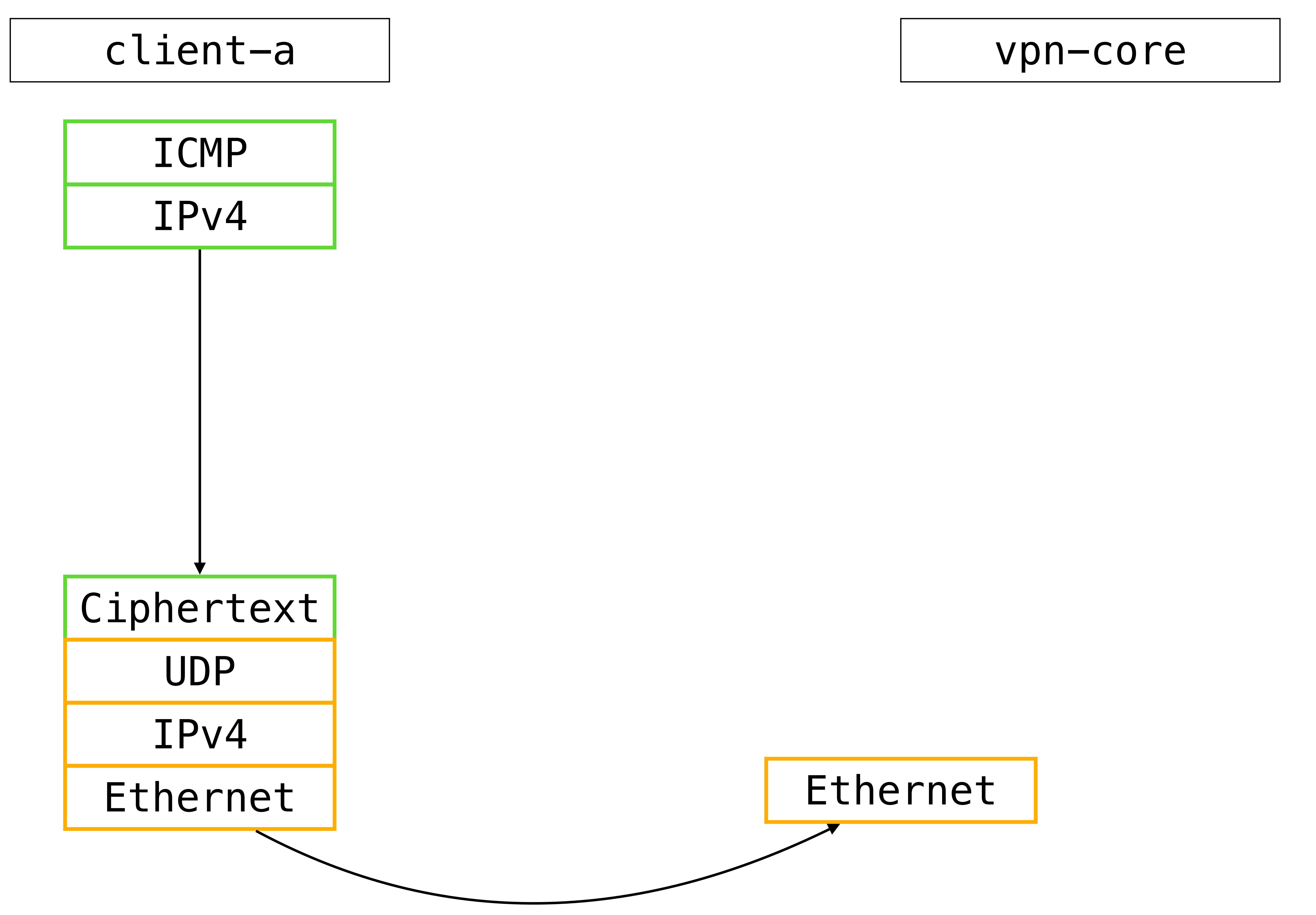
Topología Física

A day in the life…



Topología Física

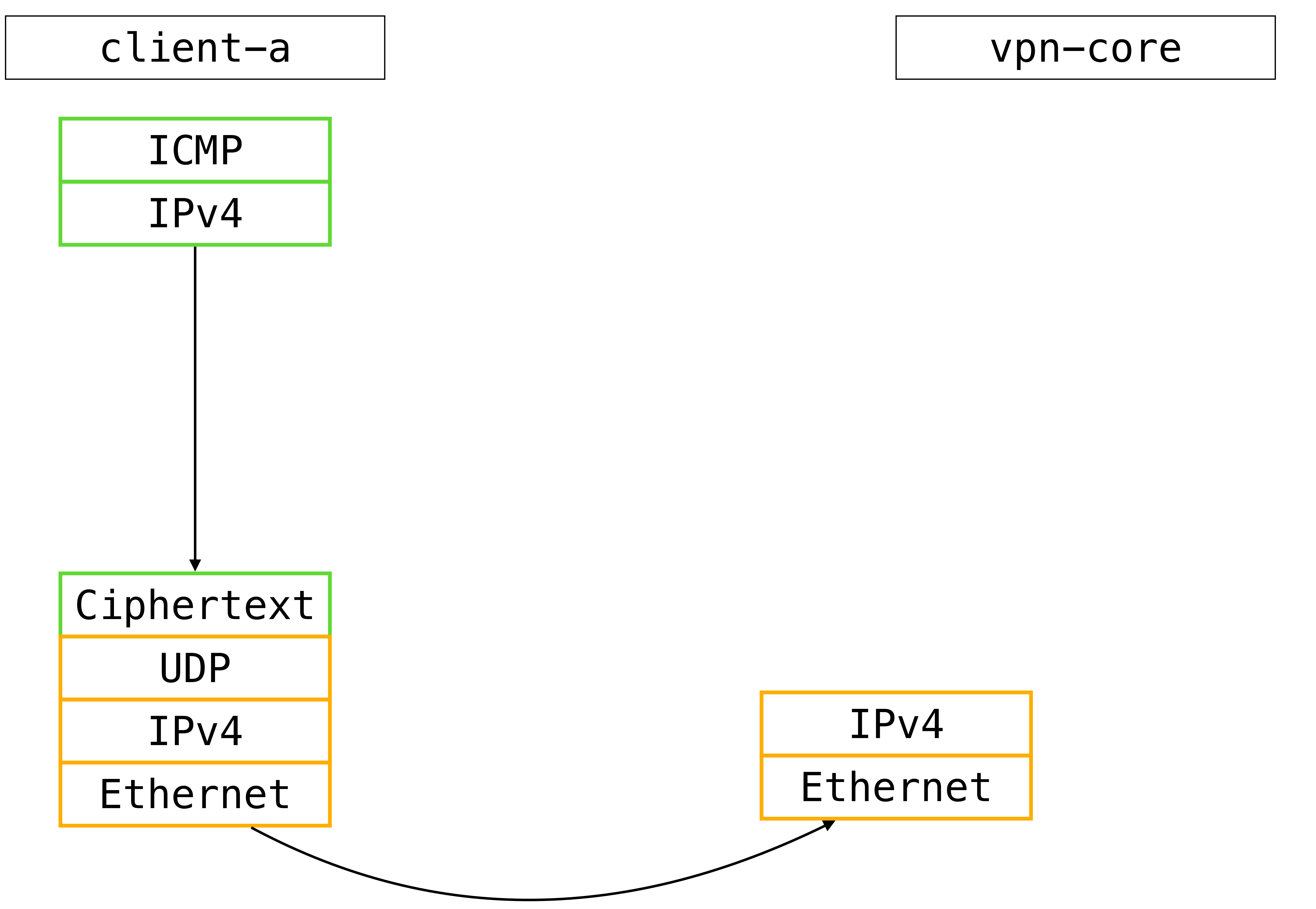
A day in the life…



client-b

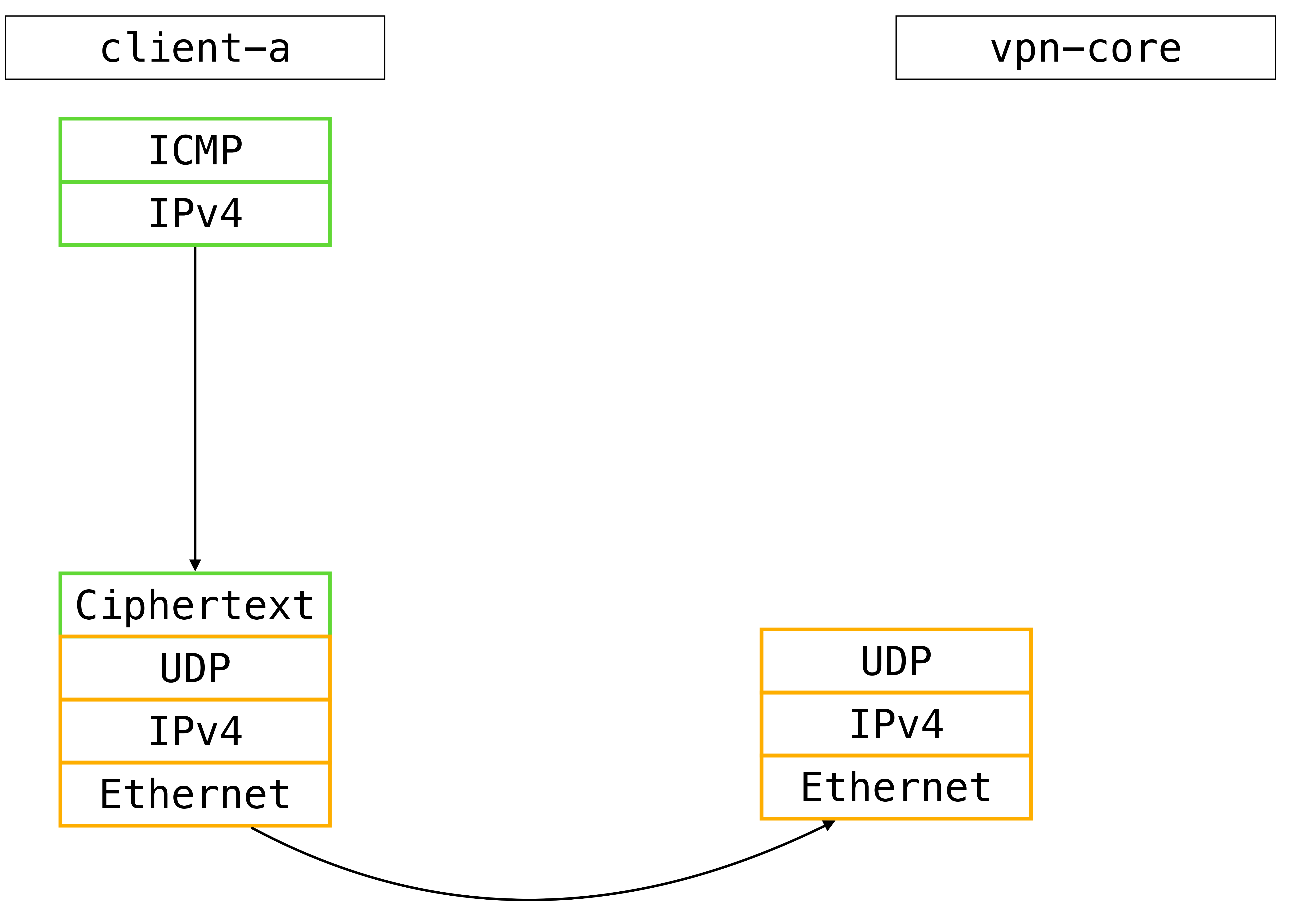
Topología Física

A day in the life…



Topología Física

A day in the life…

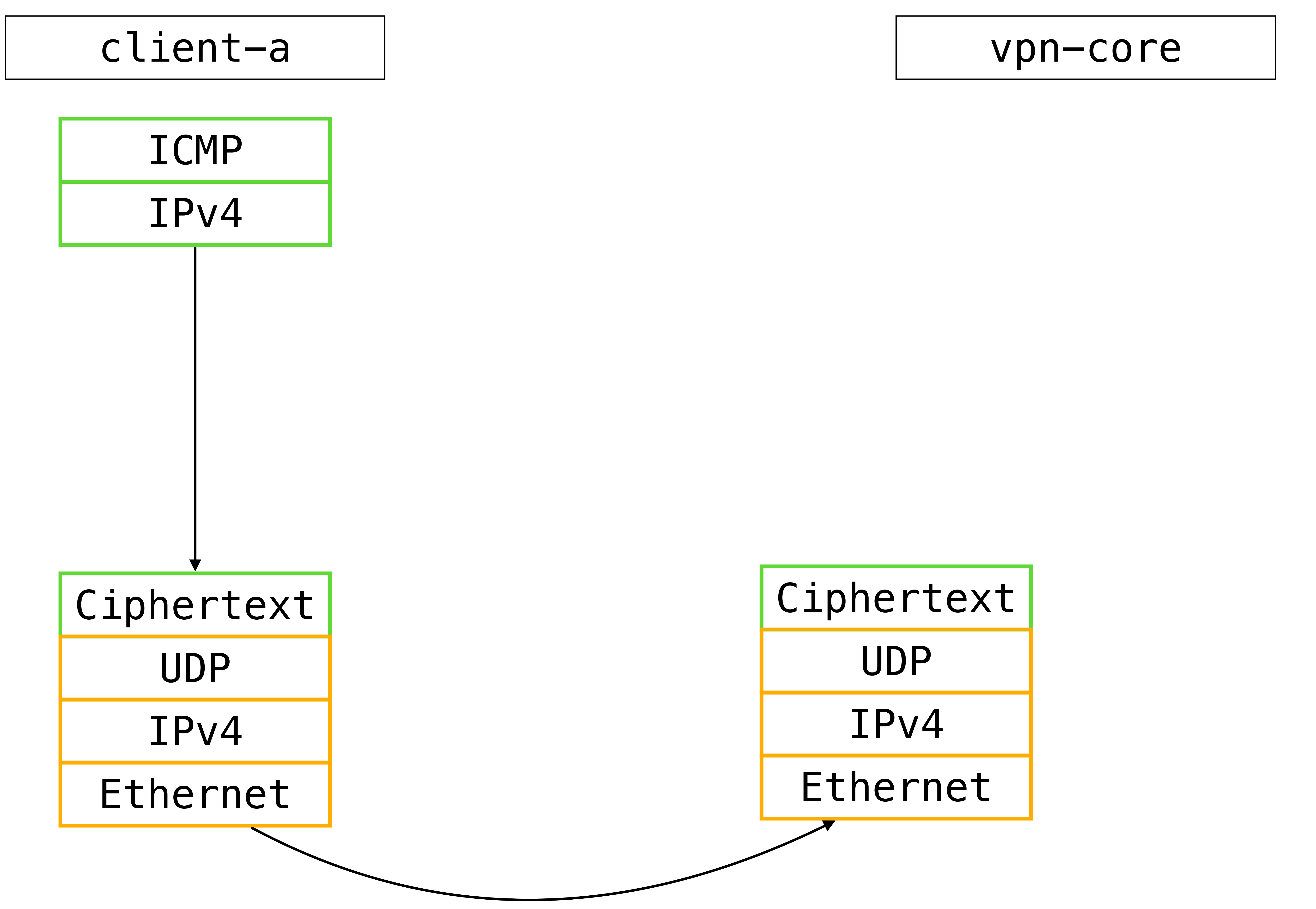


vpn-core

client-b

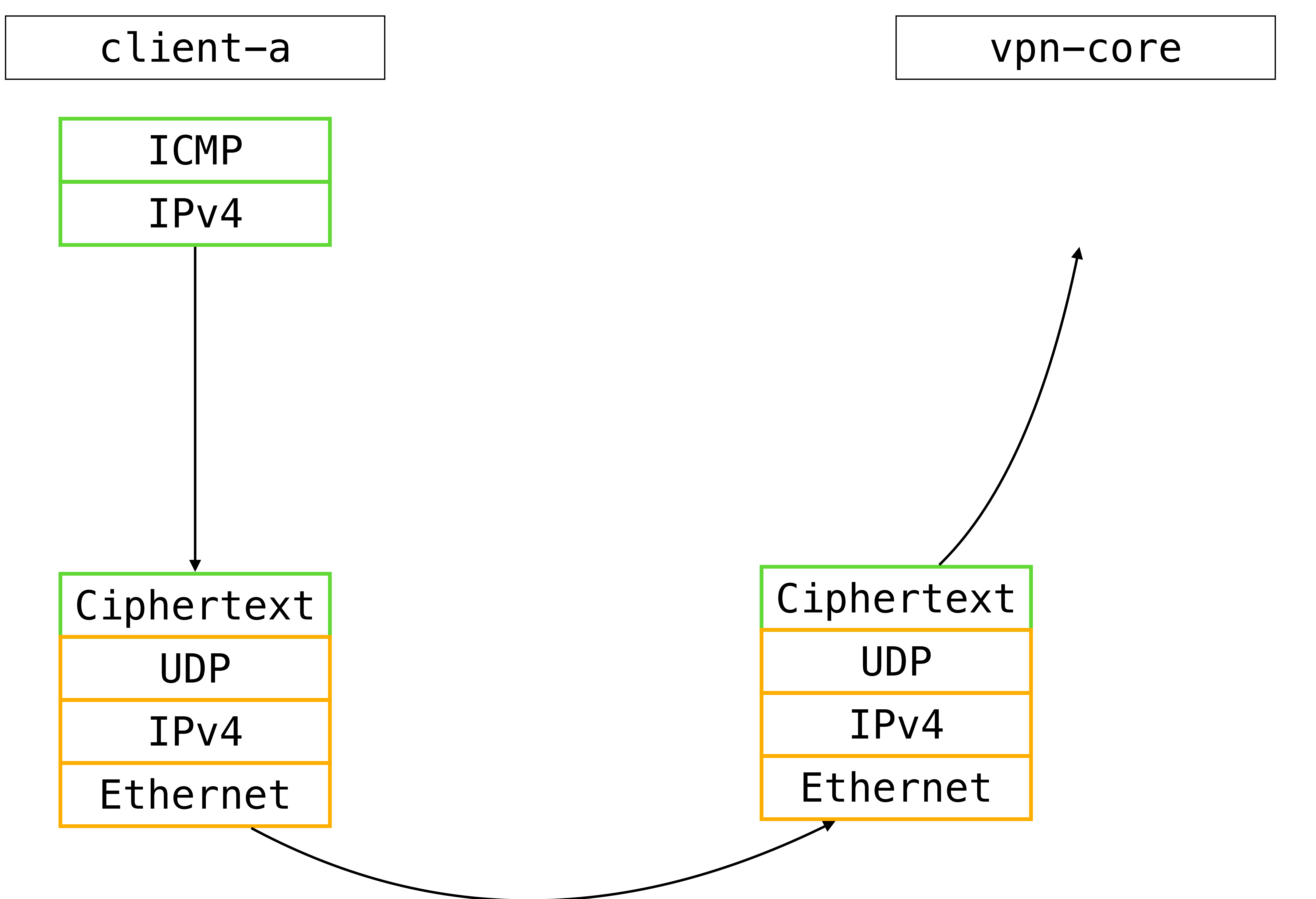
Topología Física

A day in the life...



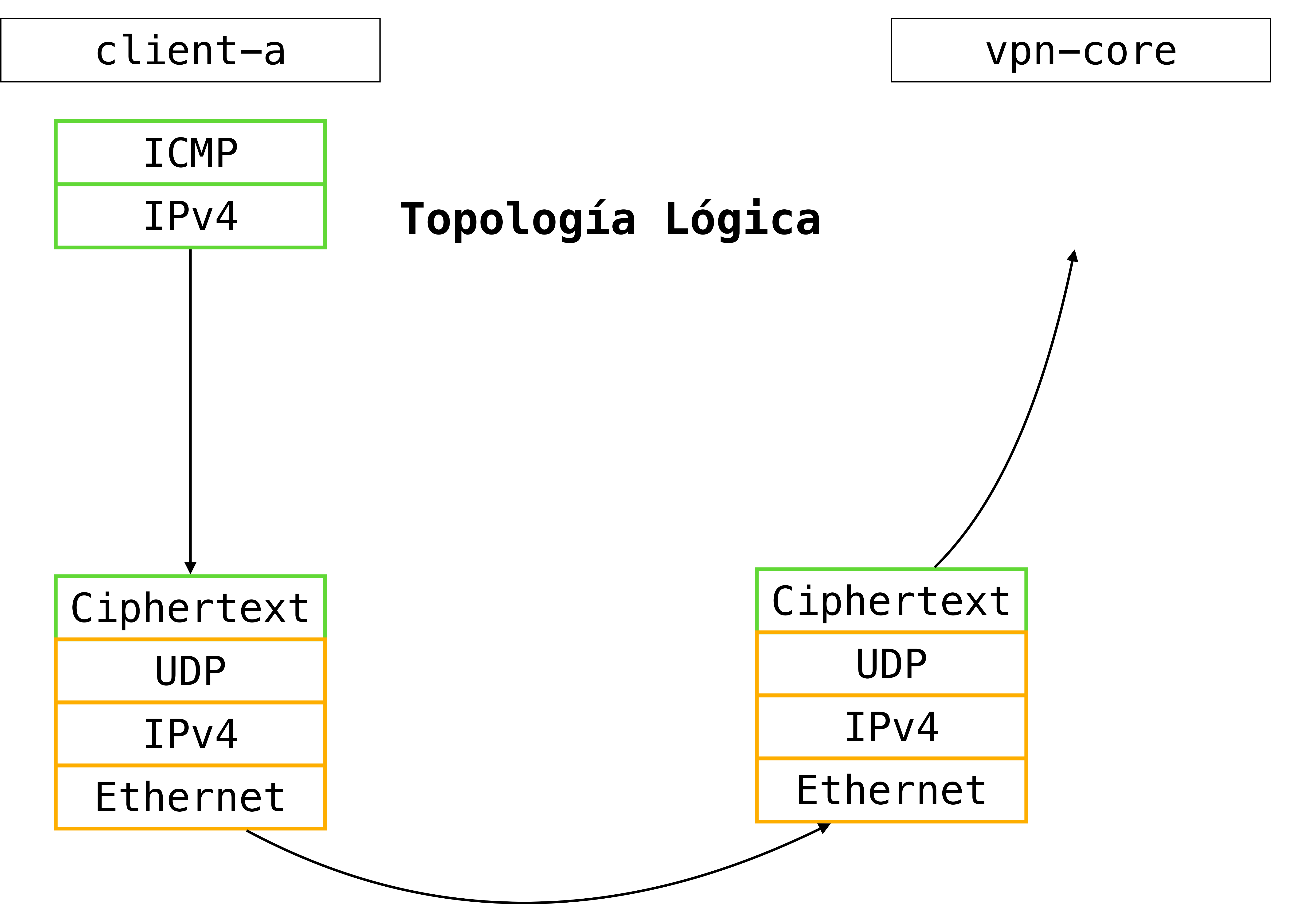
Topología Física

A day in the life…

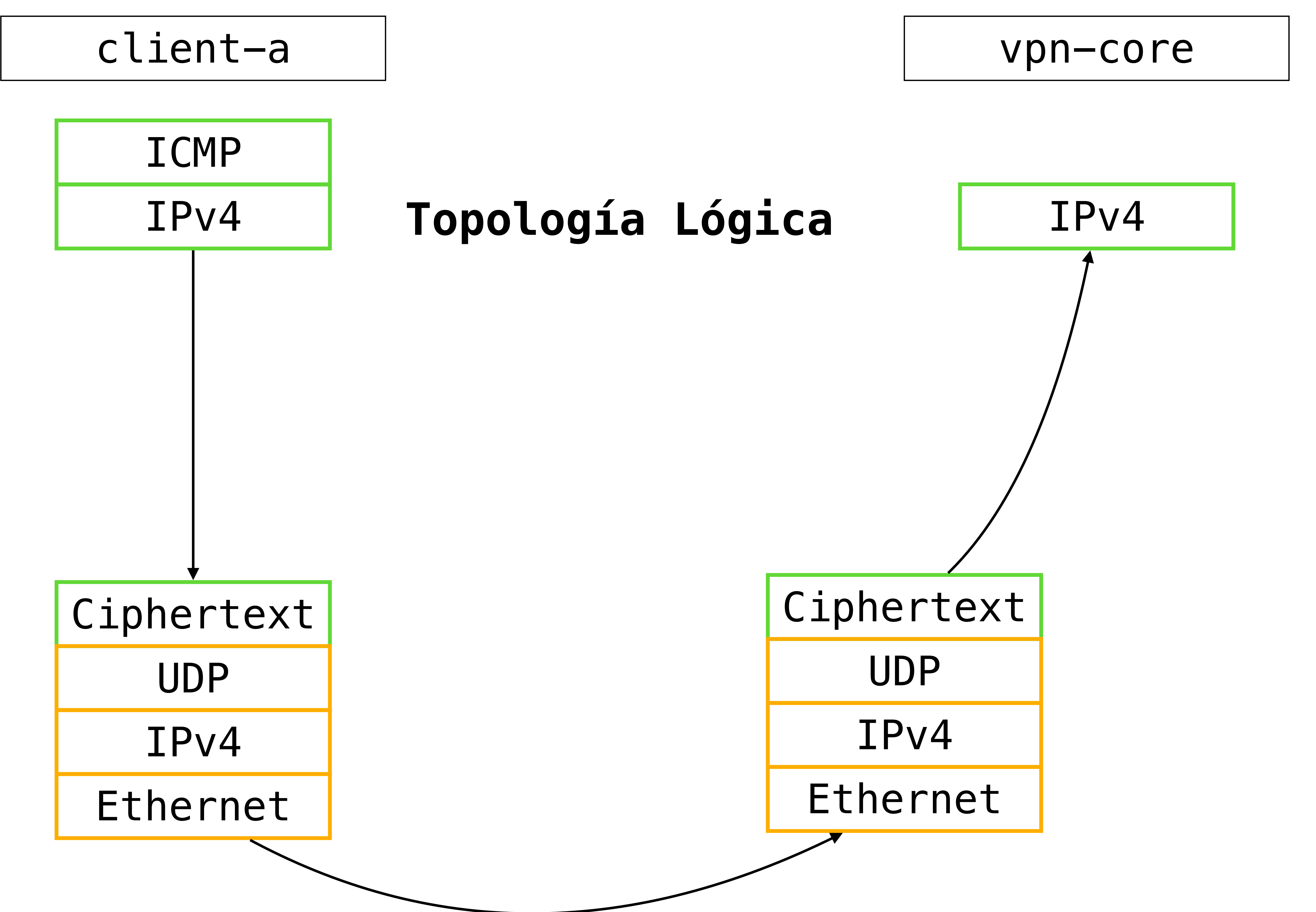


Topología Física

A day in the life...



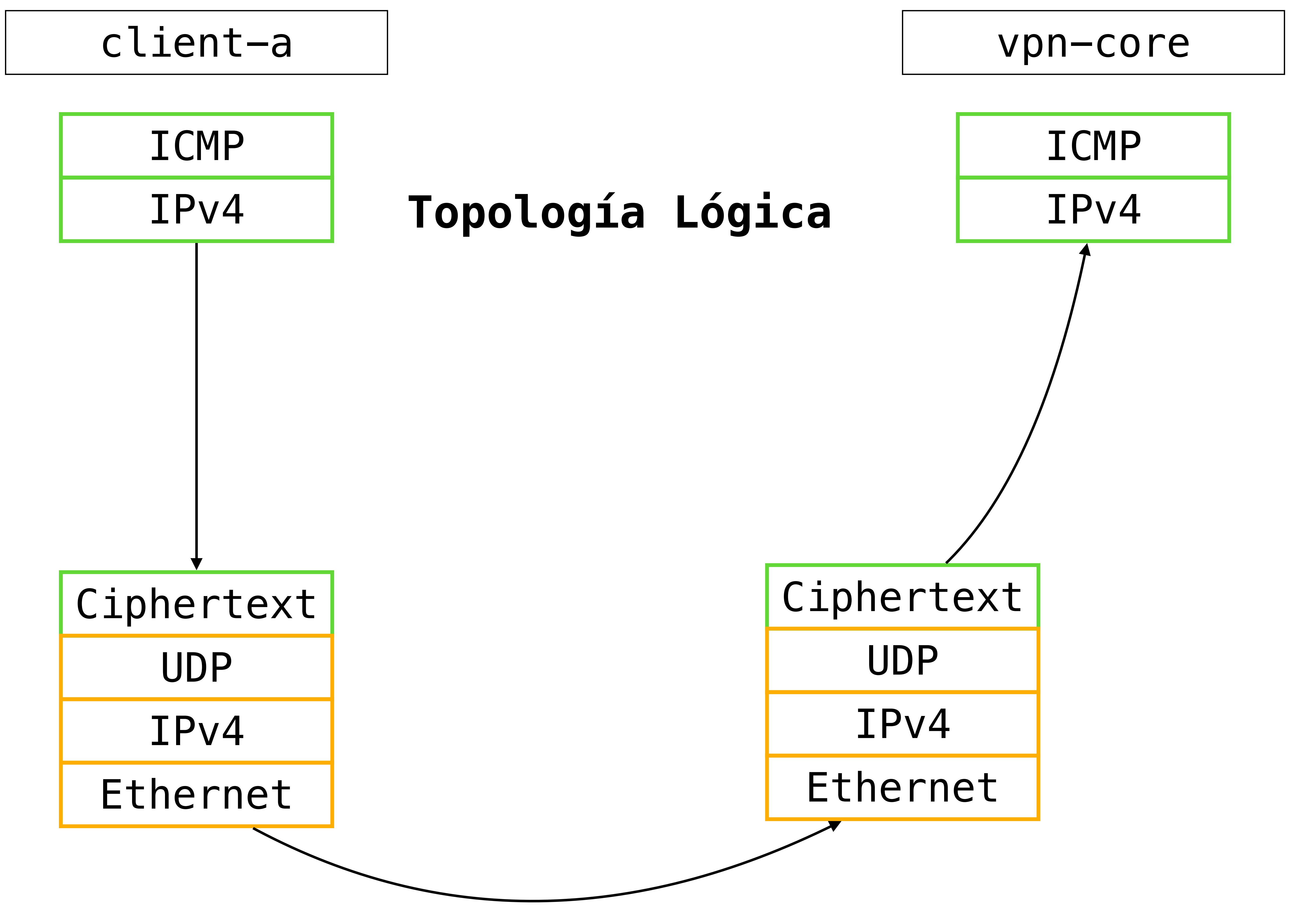
A day in the life…



client-b

Topología Física

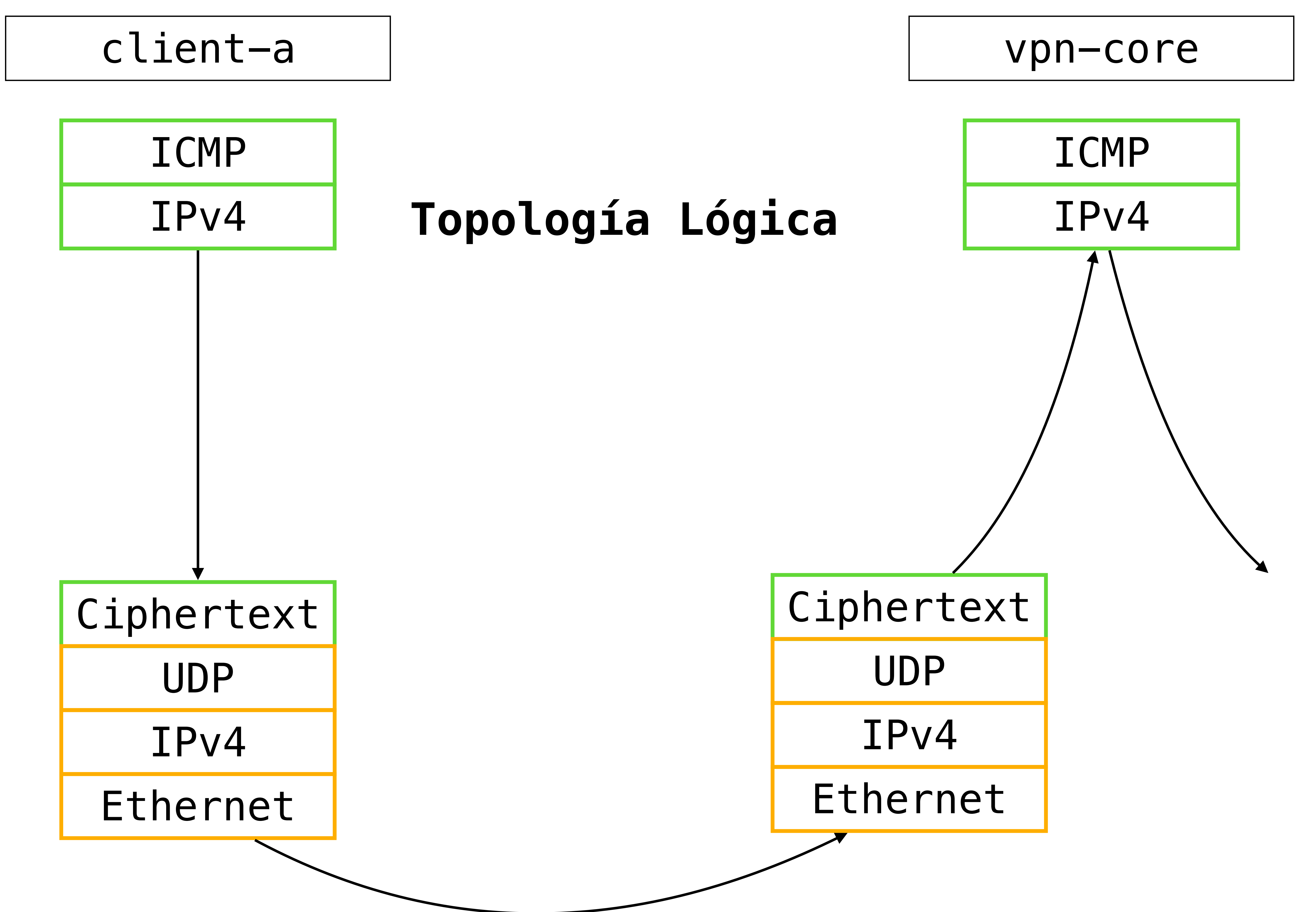
A day in the life...



client-b

Topología Física

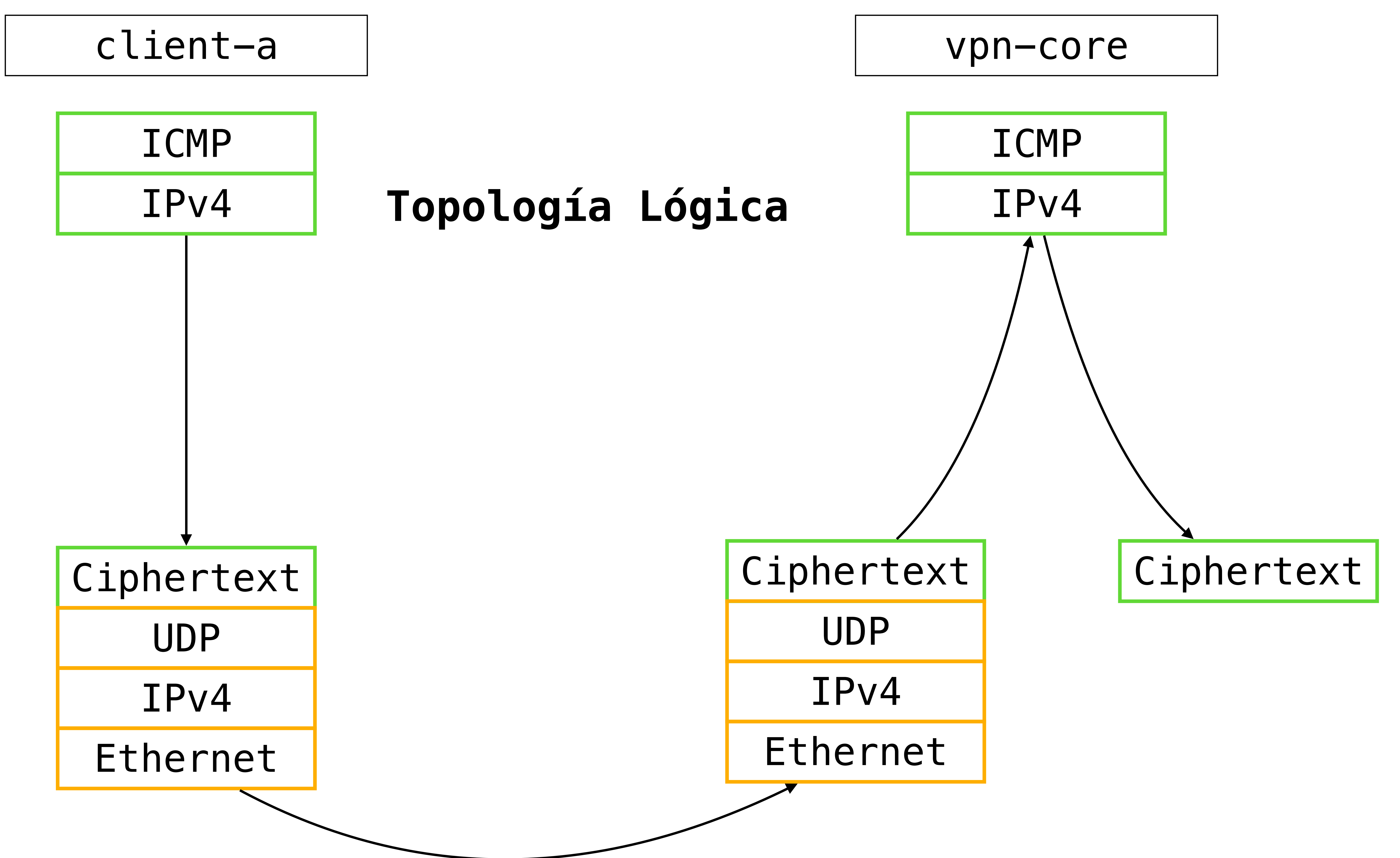
A day in the life...



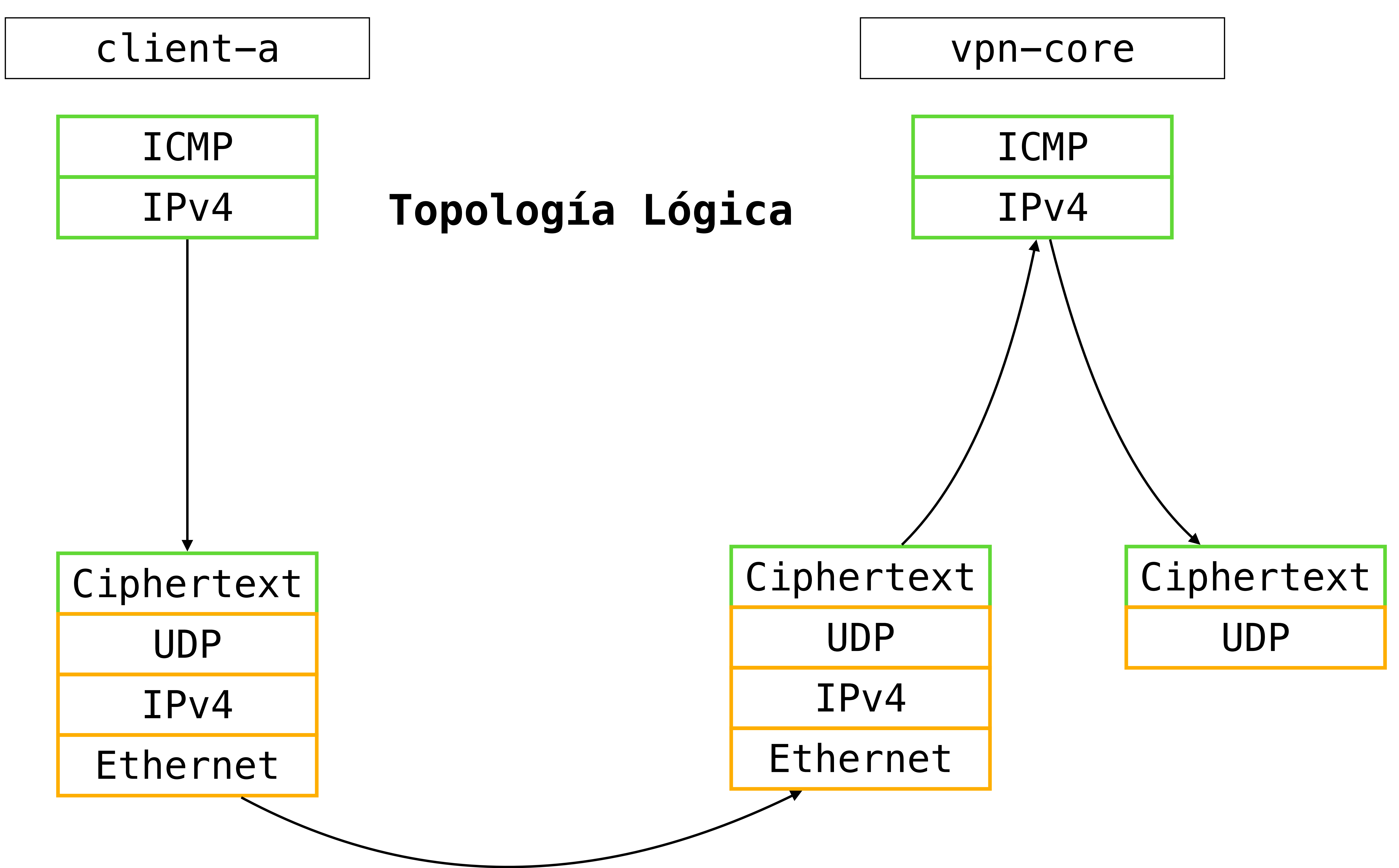
client-b

Topología Física

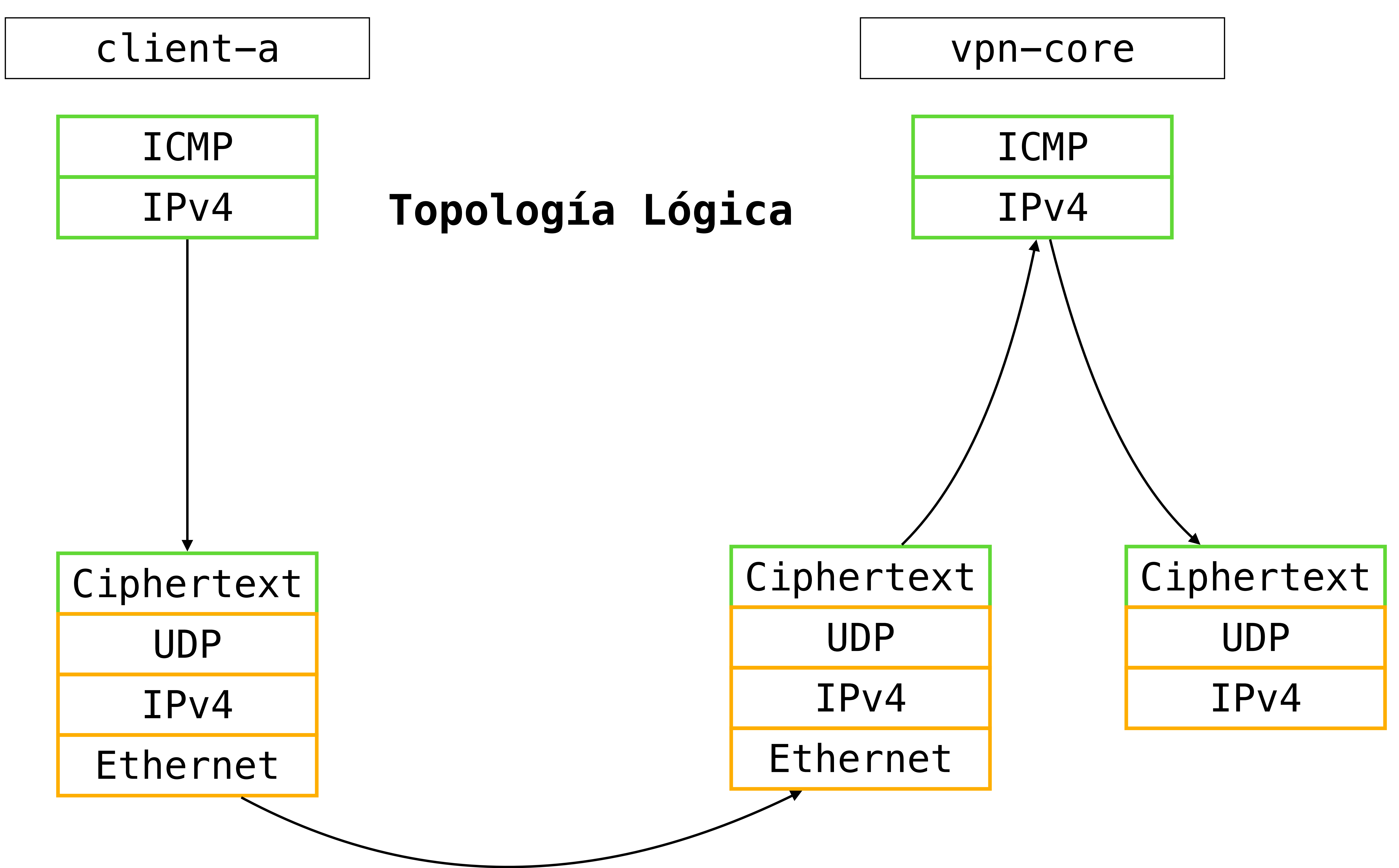
A day in the life…



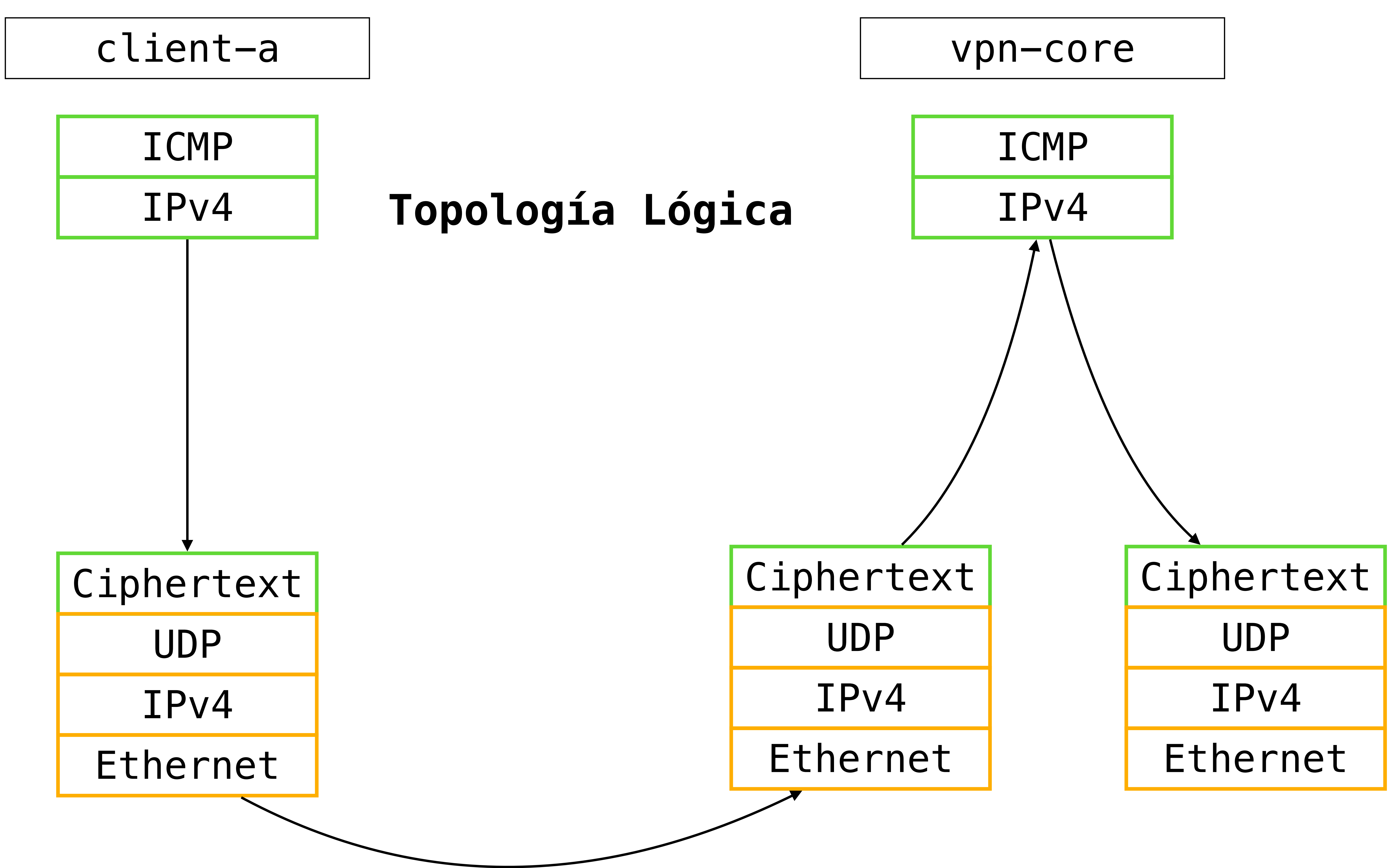
A day in the life...



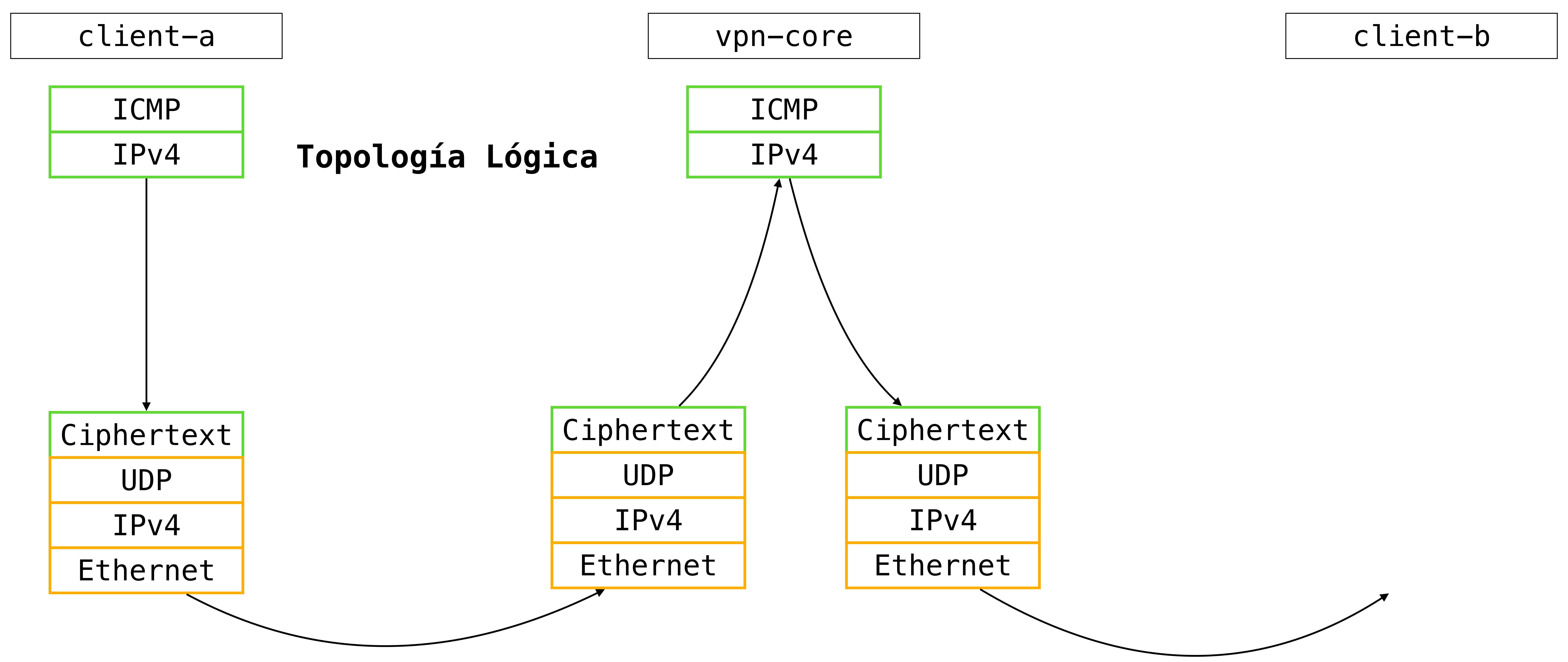
A day in the life…



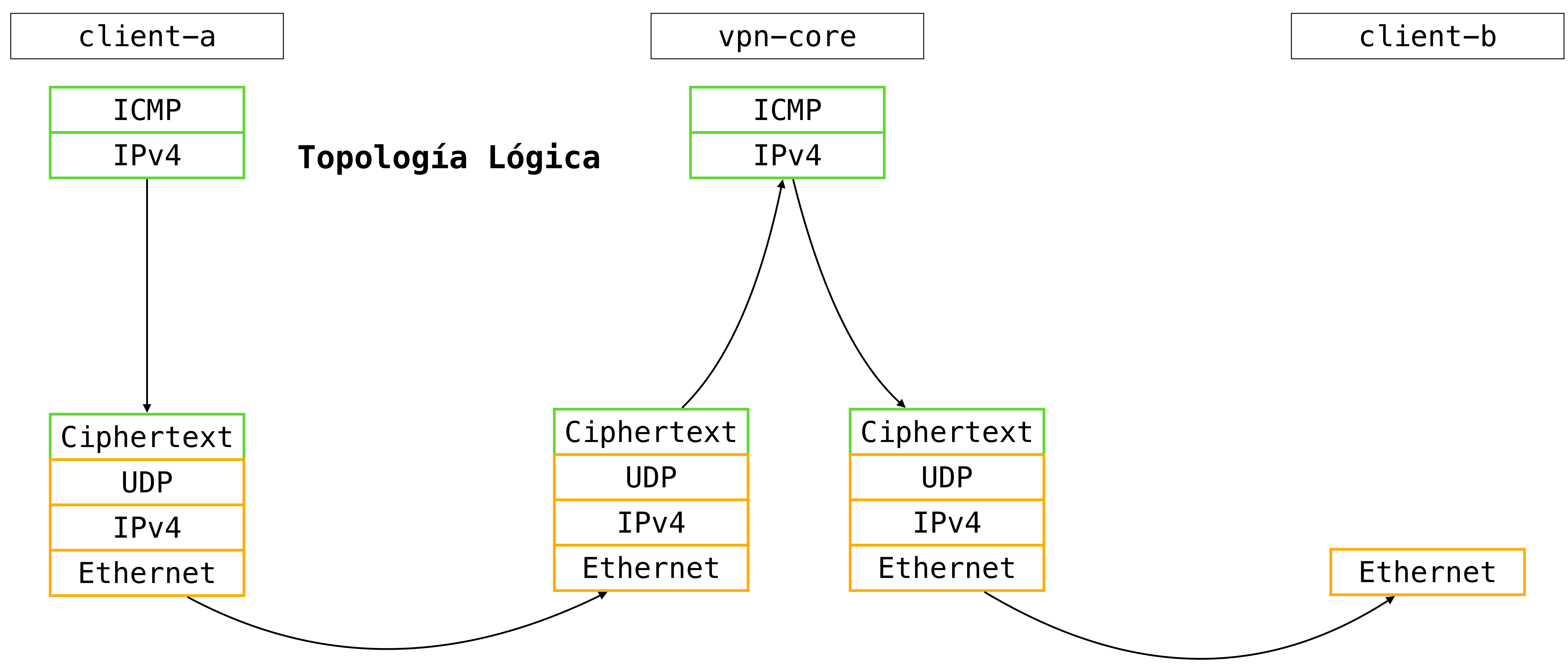
A day in the life...



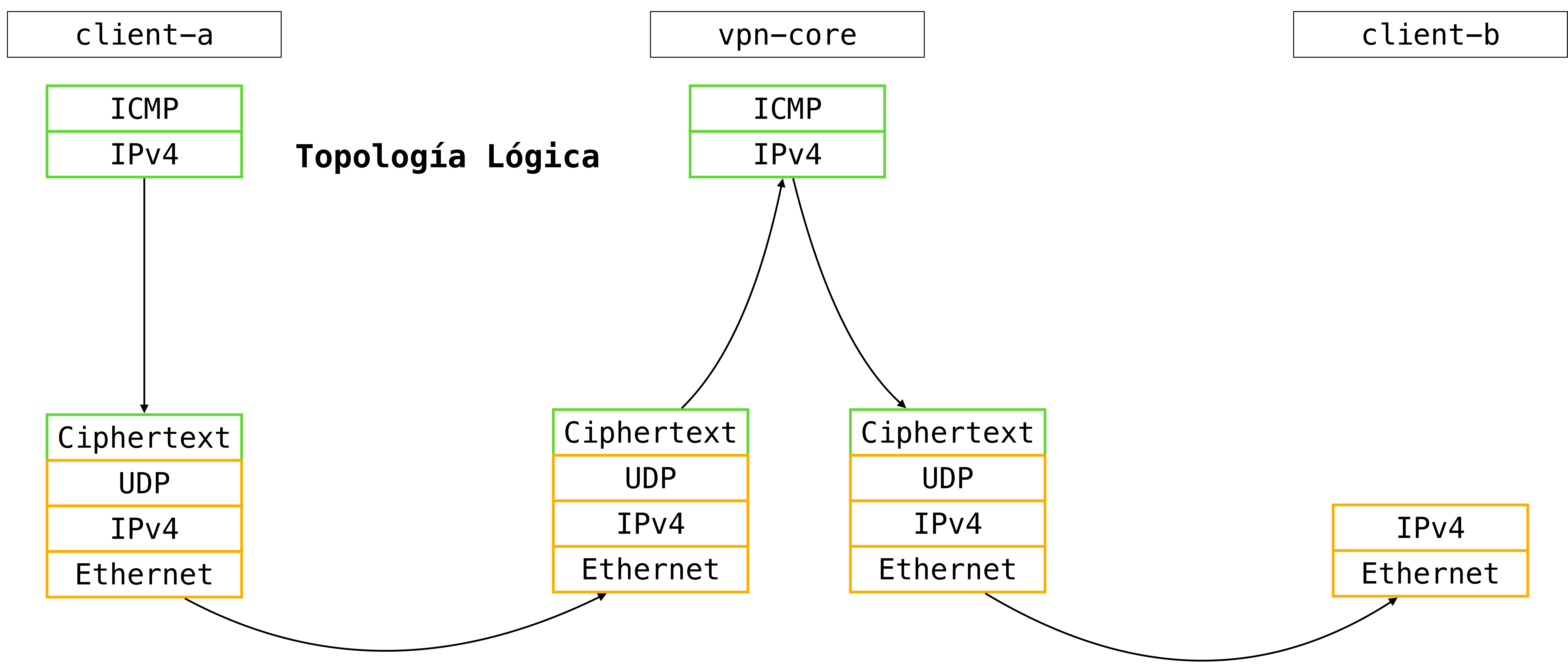
A day in the life...



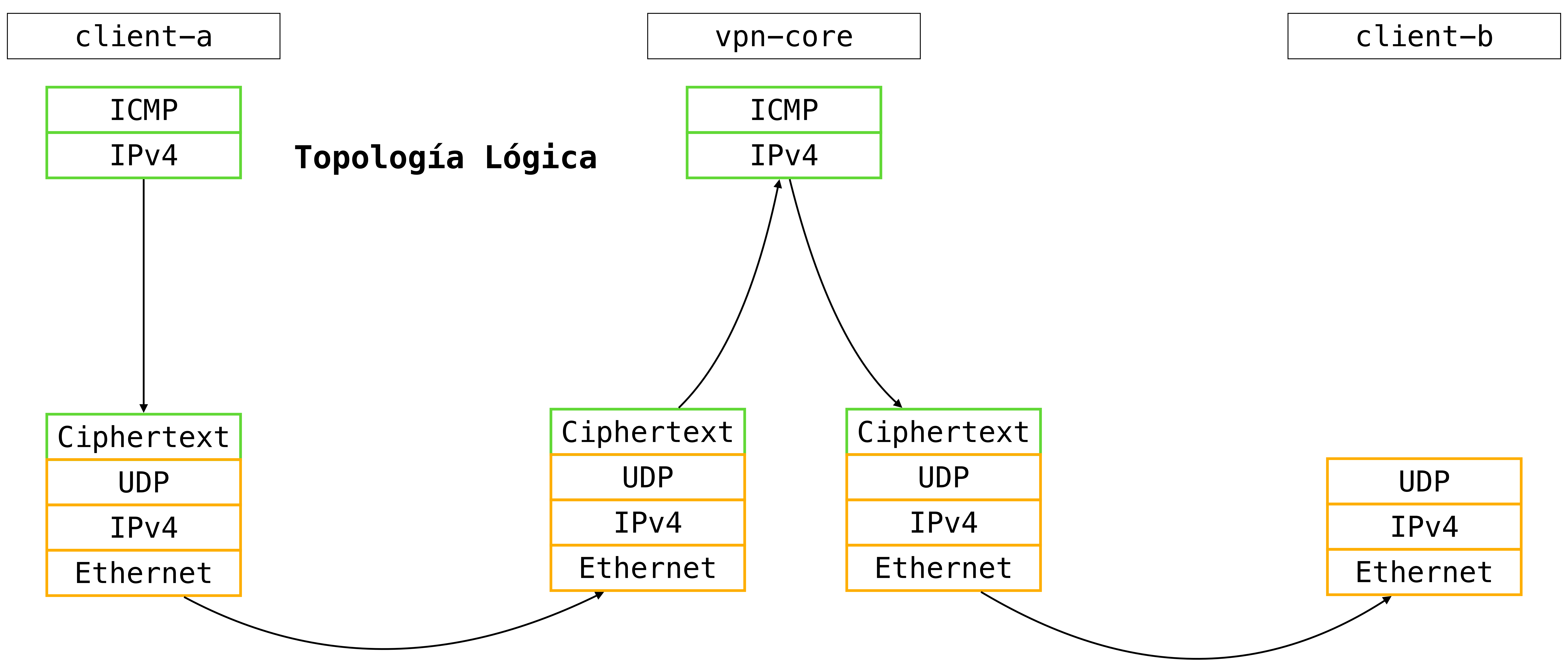
A day in the life...



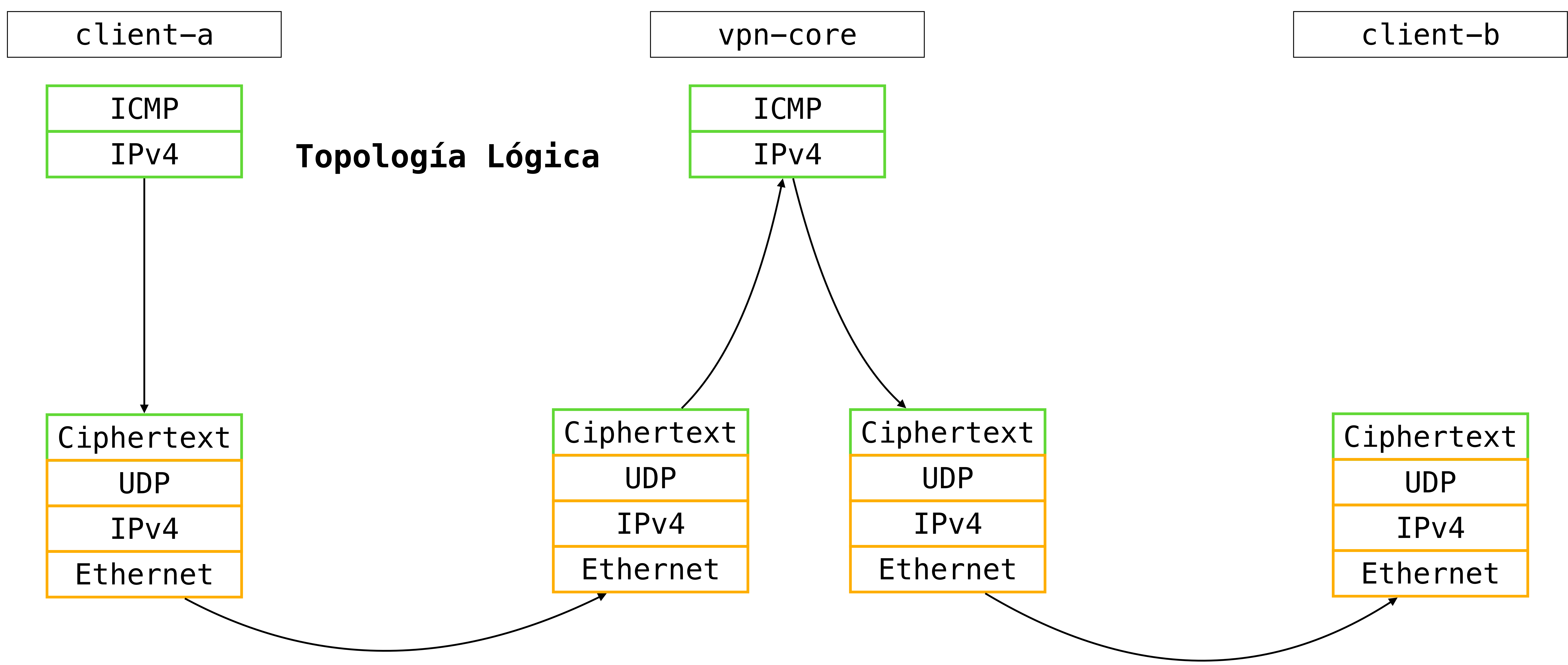
A day in the life...



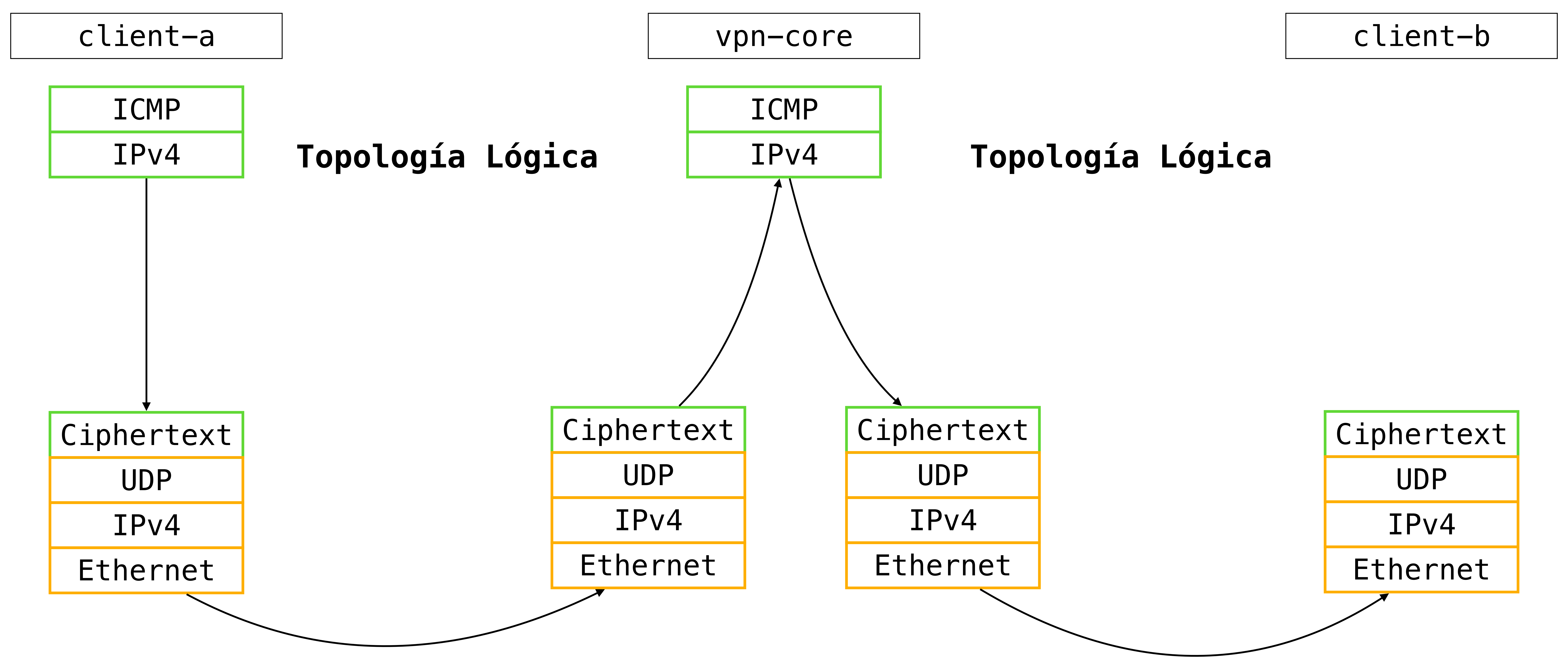
A day in the life...



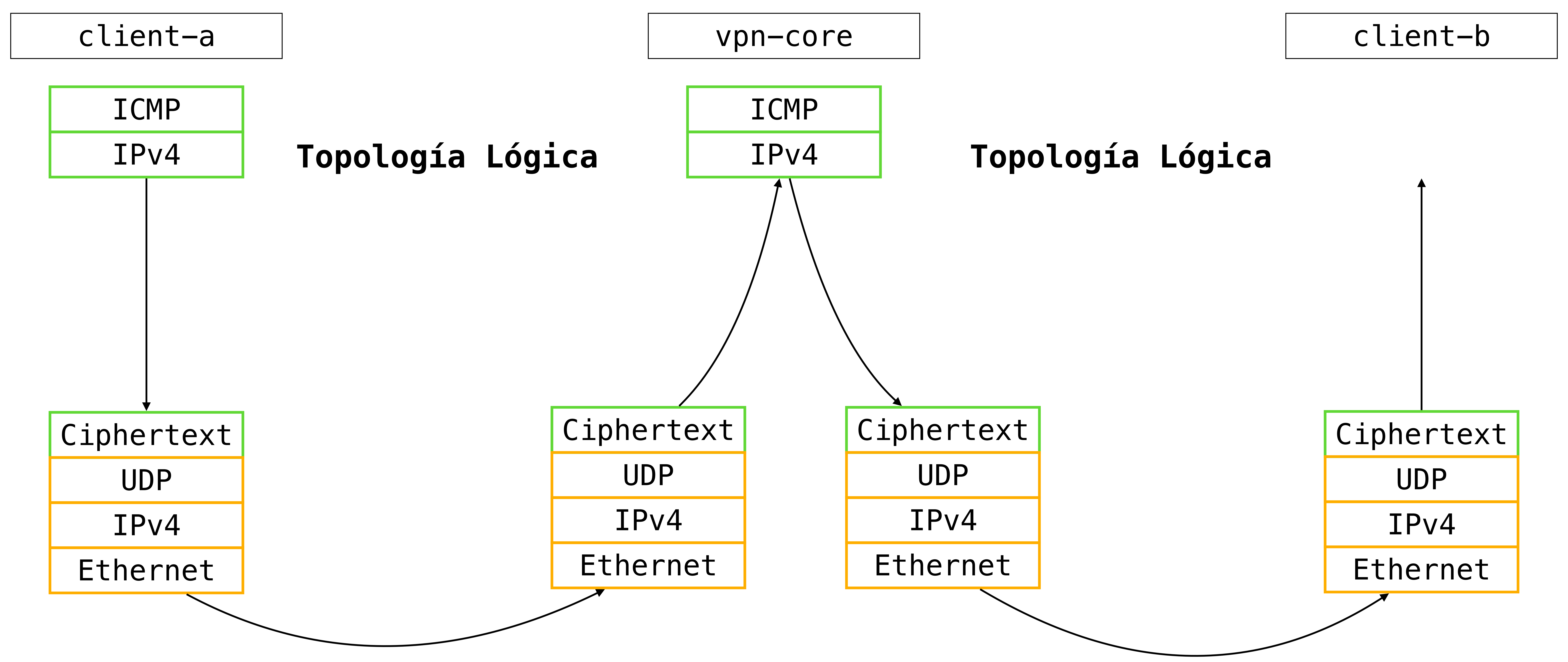
A day in the life...



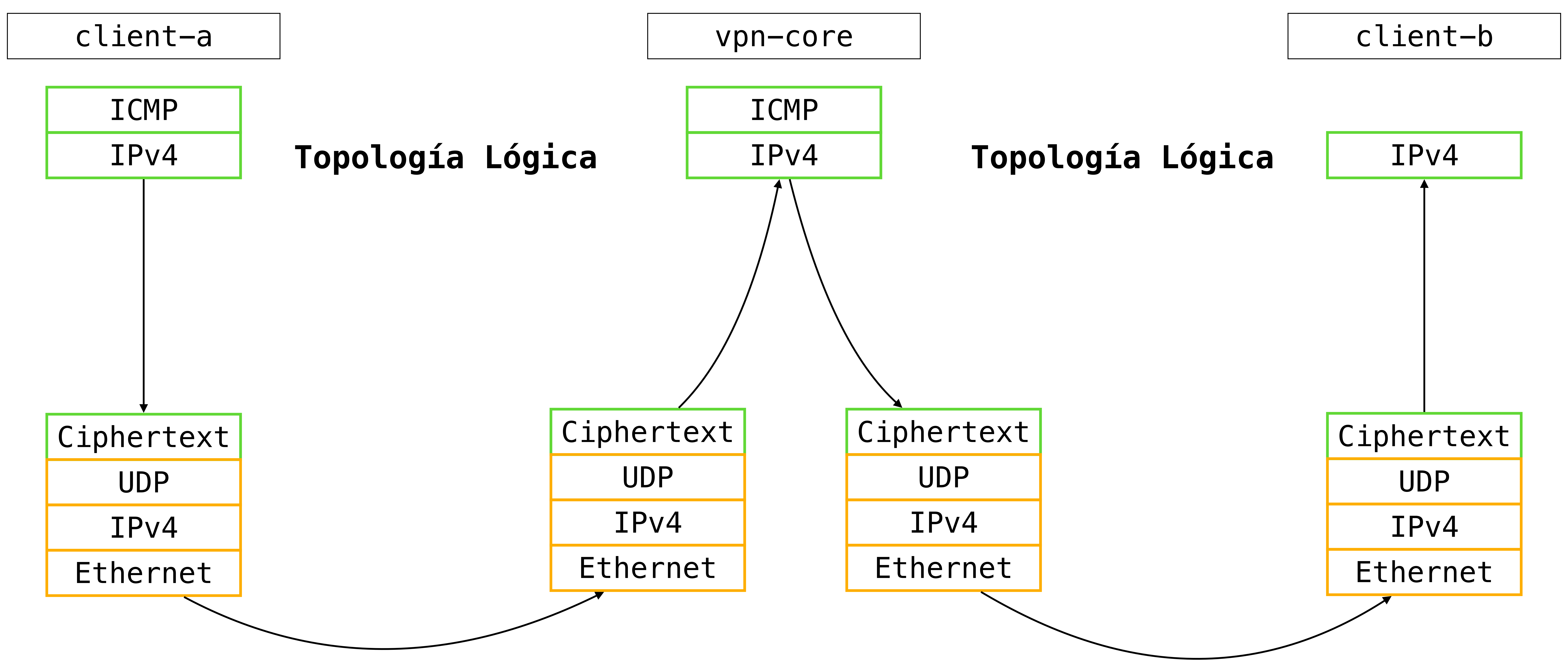
A day in the life…



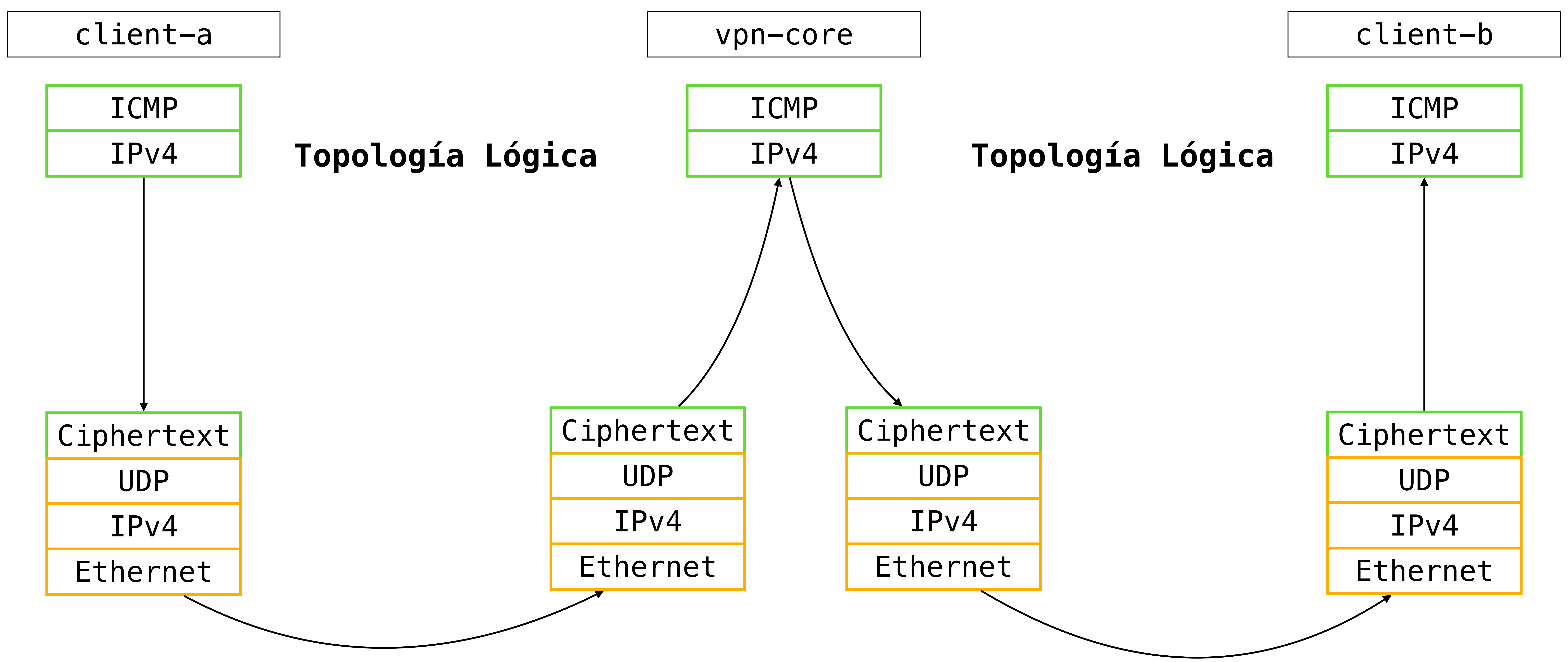
A day in the life…



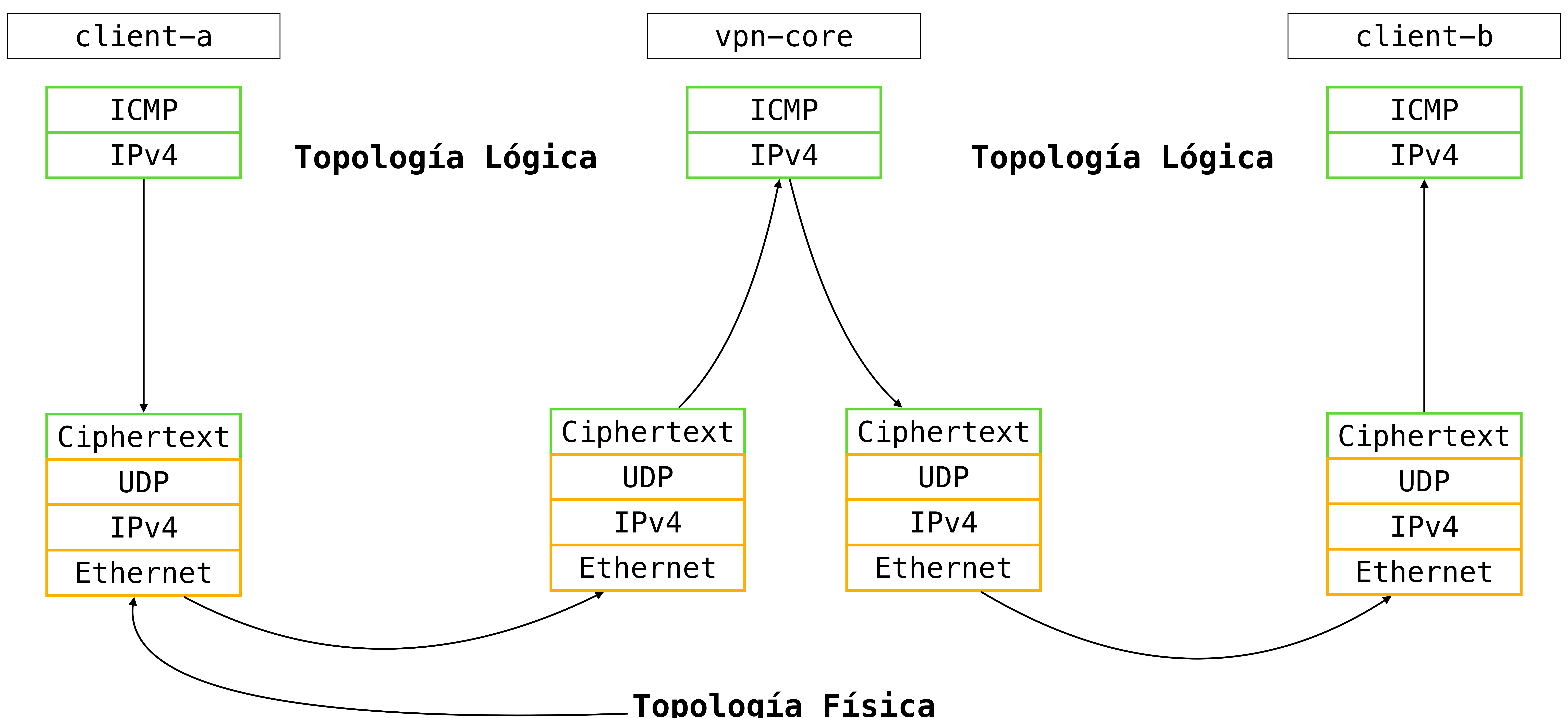
A day in the life...



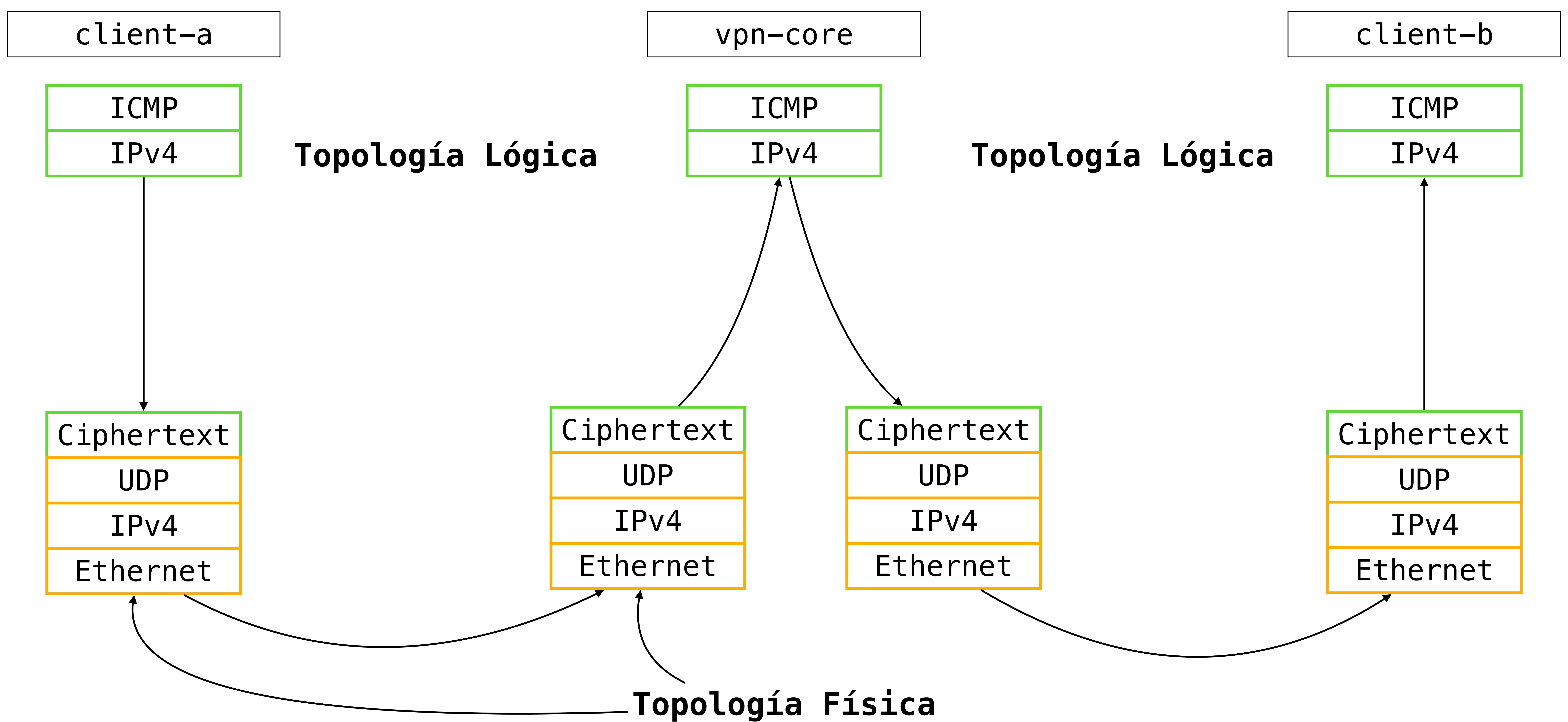
A day in the life...



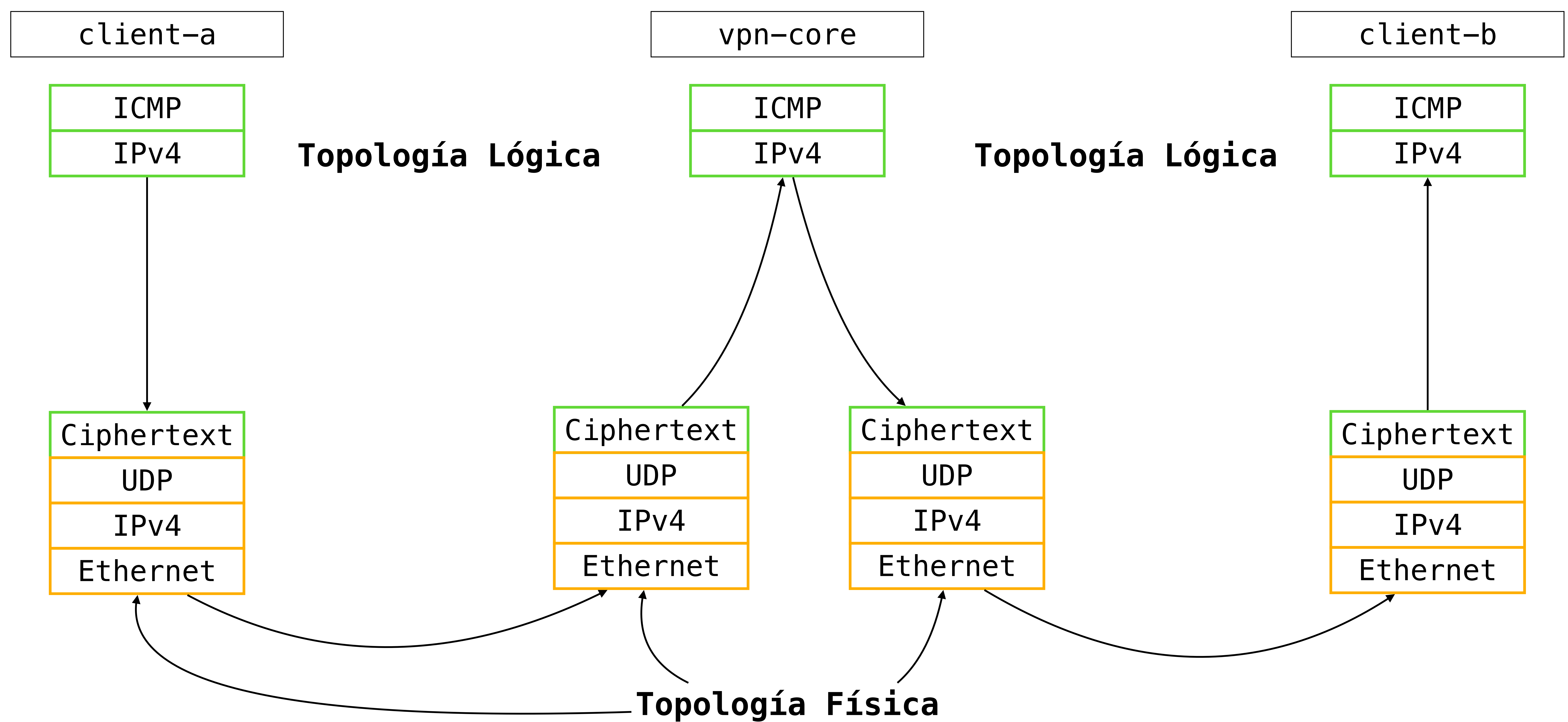
A day in the life…



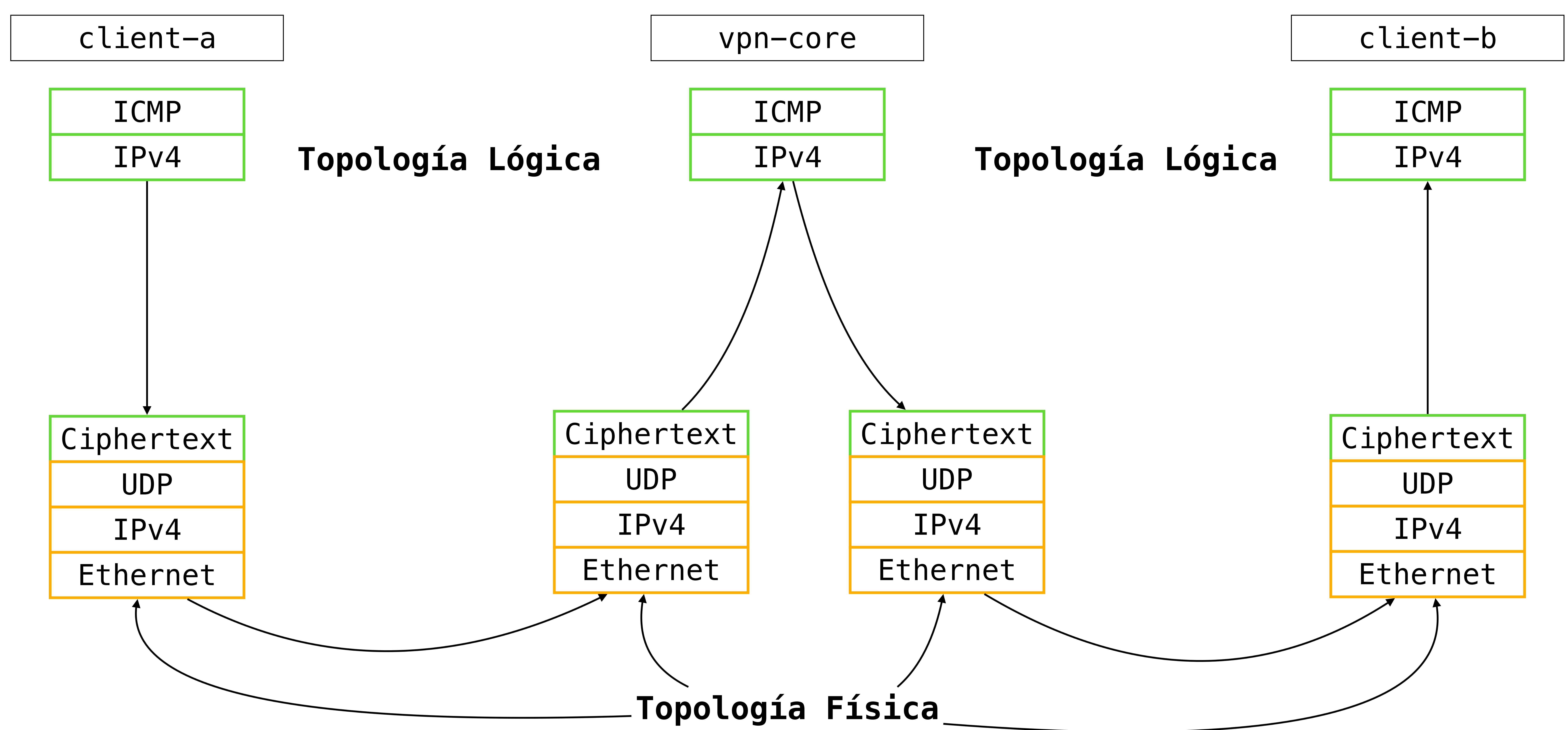
A day in the life...



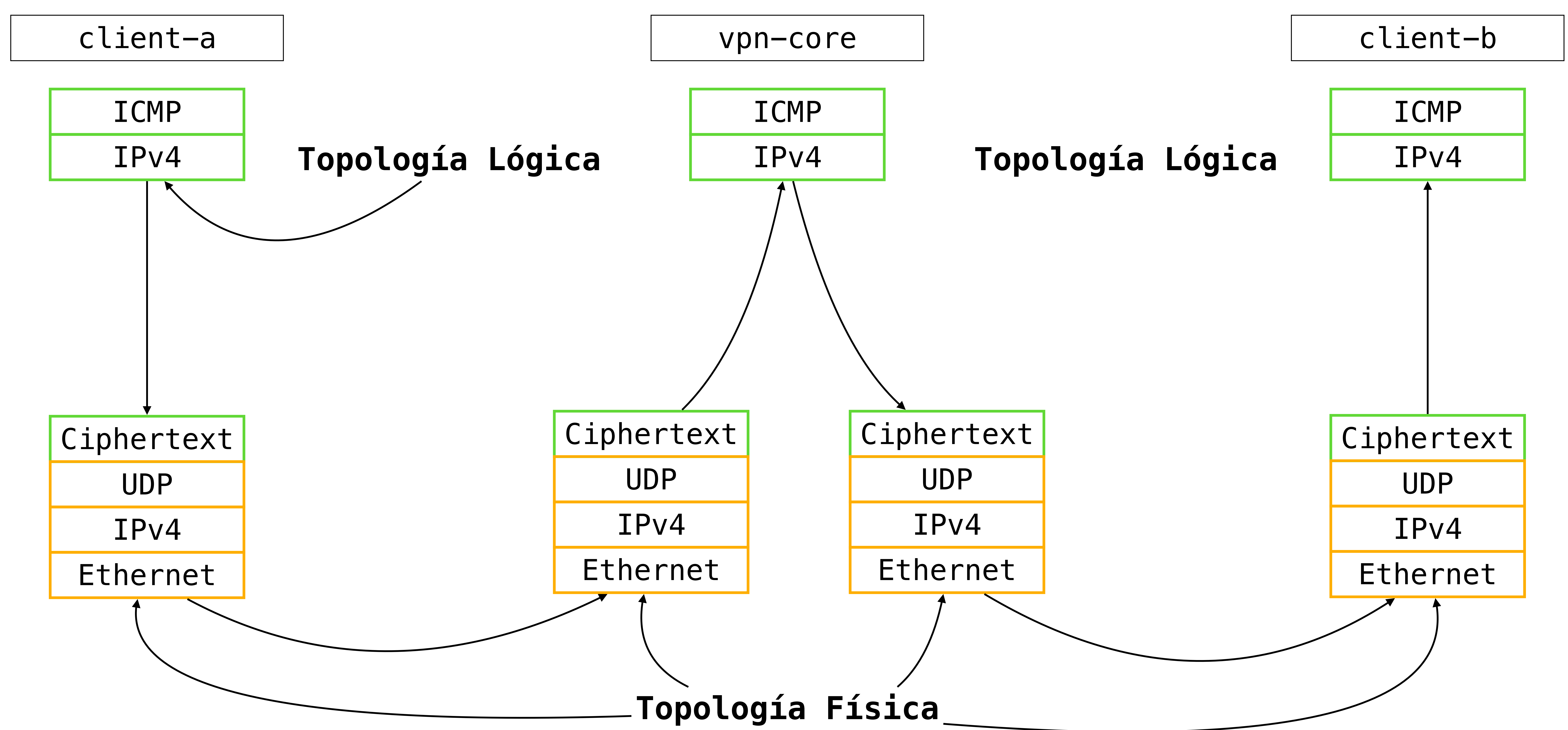
A day in the life...



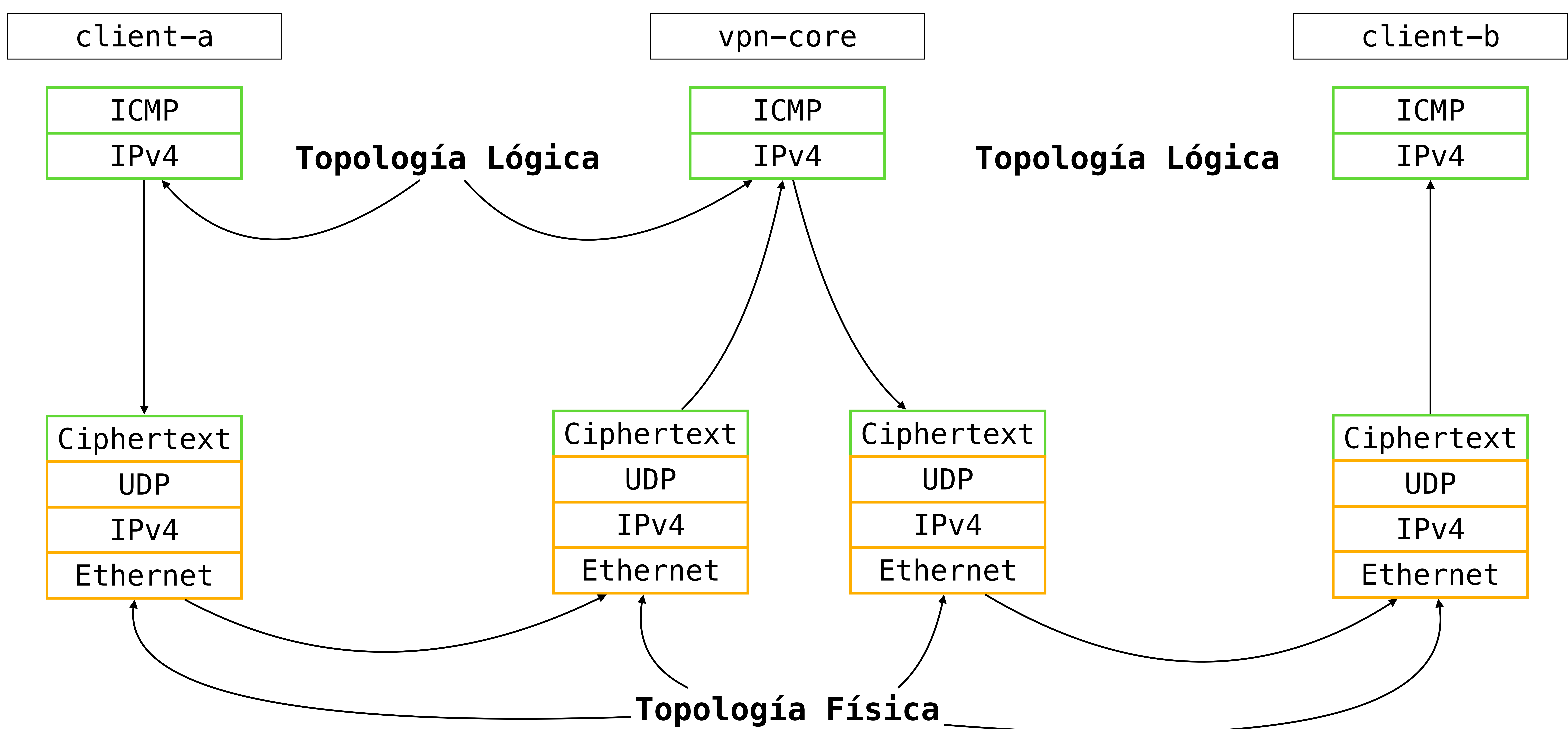
A day in the life…



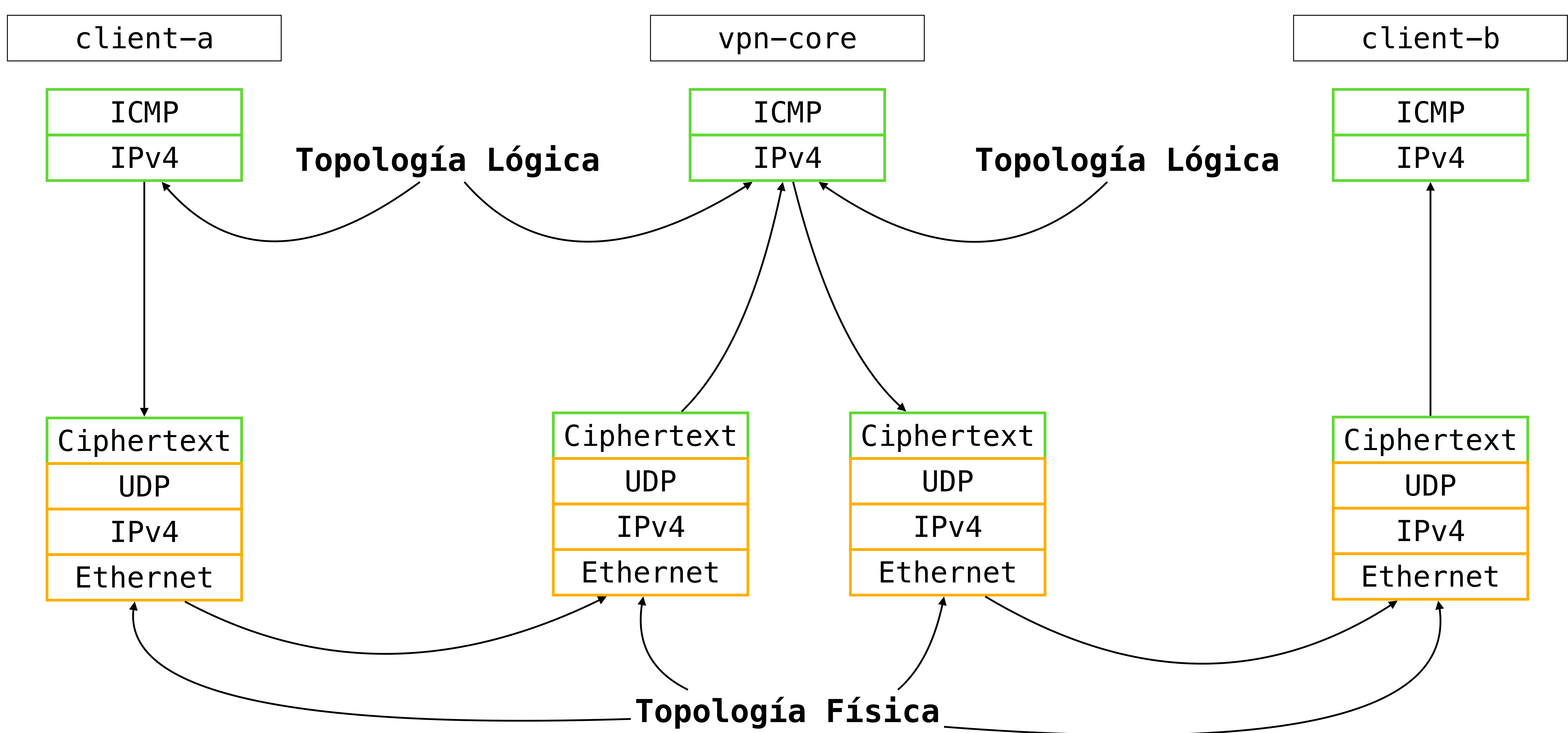
A day in the life…



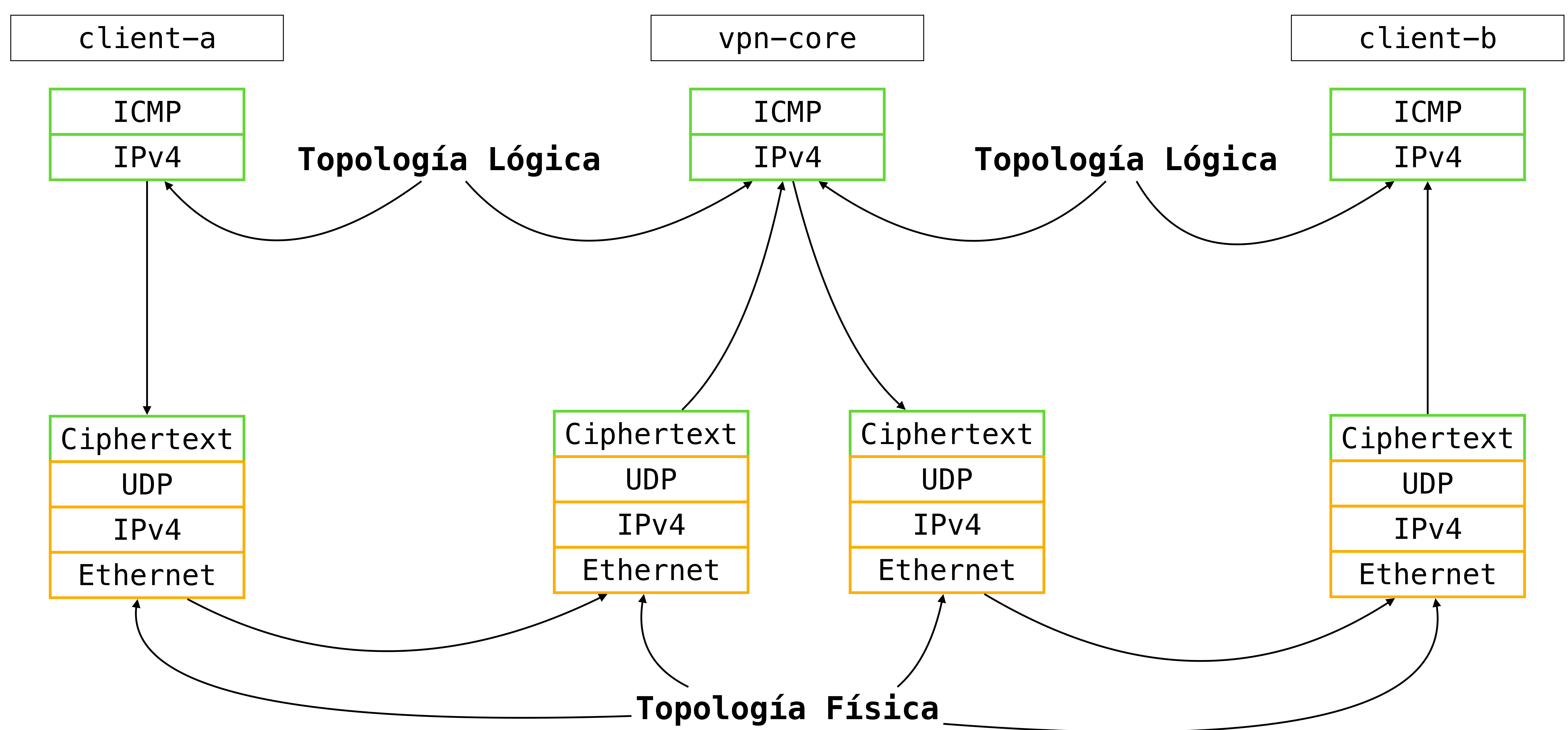
A day in the life...



A day in the life...



A day in the life…



Asegurando servicios

- Vamos a empezar por asegurar un servicio en uno de los clientes.
- Desplegaremos un «servidor» con `nc(1)` en `client-b`:

```
nc -l 1234
```

Fuente

- Lo aseguraremos con una regla de `iptables`. Podemos:
  - Solo permitir tráfico recibido de `vpn-core` (i.e `10.0.123.2`).
  - Solo permitir tráfico perteneciente a la VPN.
- Para verificar el despliegue, trataremos de conectarnos desde client-a:

```
nc 10.0.123.4 1234
nc 192.168.4.3 1234
```

Asegurando servicios

- Vamos a empezar por asegurar un servicio en uno de los clientes.
- Desplegaremos un «servidor» con `nc(1)` en `client-b`:

```
nc -l 1234
```

- Lo aseguraremos con una regla de `iptables`. Podemos:
  - Solo permitir tráfico recibido de `vpn-core` (i.e `10.0.123.2`).
  - Solo permitir tráfico perteneciente a la VPN.
- Para verificar el despliegue, trataremos de conectarnos desde client-a:

```
nc 10.0.123.4 1234 ←
nc 192.168.4.3 1234
```

Fuente

Asegurando servicios

- Vamos a empezar por asegurar un servicio en uno de los clientes.
- Desplegaremos un «servidor» con `nc(1)` en `client-b`:

```
nc -l 1234
```

- Lo aseguraremos con una regla de `iptables`. Podemos:
  - Solo permitir tráfico recibido de `vpn-core` (i.e `10.0.123.2`).
  - Solo permitir tráfico perteneciente a la VPN.
- Para verificar el despliegue, trataremos de conectarnos desde client-a:

```
nc 10.0.123.4 1234    ← Primera estrategia
nc 192.168.4.3 1234
```

Fuente

Asegurando servicios

- Vamos a empezar por asegurar un servicio en uno de los clientes.
- Desplegaremos un «servidor» con `nc(1)` en `client-b`:

```
nc -l 1234
```

- Lo aseguraremos con una regla de `iptables`. Podemos:
  - Solo permitir tráfico recibido de `vpn-core` (i.e `10.0.123.2`).
  - Solo permitir tráfico perteneciente a la VPN.
- Para verificar el despliegue, trataremos de conectarnos desde client-a:

```
nc 10.0.123.4 1234   ←   Primera estrategia
nc 192.168.4.3 1234  ←
```

Fuente

Asegurando servicios

- Vamos a empezar por asegurar un servicio en uno de los clientes.
- Desplegaremos un «servidor» con `nc(1)` en `client-b`:

```
nc -l 1234
```

Fuente

- Lo aseguraremos con una regla de `iptables`. Podemos:
  - Solo permitir tráfico recibido de `vpn-core` (i.e `10.0.123.2`).
  - Solo permitir tráfico perteneciente a la VPN.
- Para verificar el despliegue, trataremos de conectarnos desde client-a:

```
nc 10.0.123.4 1234     ← Primera estrategia
nc 192.168.4.3 1234    ← Segunda estrategia
```

Filtrado por IP de origen

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

- El tráfico se «autentica» atravesando `vpn-core`:
    - Al hacerlo el tráfico es NATeado…
    - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en `vpn-core`

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en vpn-core

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter.**

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplicará una vez que el paquete vaya a ser reenviado.

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplicará una vez que el paquete vaya a ser reenviado.

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

La regla se aplicará una vez que el paquete vaya a ser reenviado.

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

La regla se aplicará una vez que el paquete vaya a ser reenviado.

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplicará una vez que el paquete vaya a ser reenviado.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplicará una vez que el paquete vaya a ser reenviado.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplicará una vez que el paquete vaya a ser reenviado.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

La regla se aplicará una vez que el paquete vaya a ser reenviado.

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplicará una vez que el paquete vaya a ser reenviado.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Ejecutado en client-b

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplicará una vez que el paquete vaya a ser reenviado.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Ejecutado en client-b

Filtrado por IP de origen

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplicará una vez que el paquete vaya a ser reenviado.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Ejecutado en client-b

Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

La regla se aplicará una vez que el paquete vaya a ser reenviado.

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Ejecutado en client-b

Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplicará una vez que el paquete vaya a ser reenviado.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Ejecutado en client-b

Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.

La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

La regla se aplicará una vez que el paquete vaya a ser reenviado.

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Ejecutado en client-b

Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.

La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplicará una vez que el paquete vaya a ser reenviado.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Ejecutado en client-b

Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.

La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).

La regla se aplica a aquellos datagramas cuya dirección de origen **NO** sea la de vpn-core.

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

La regla se aplicará una vez que el paquete vaya a ser reenviado.

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Ejecutado en client-b

Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.

La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).

La regla se aplica a aquellos datagramas cuya dirección de origen **NO** sea la de vpn-core.

Filtrado por IP de origen

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

La regla se aplicará una vez que el paquete vaya a ser reenviado.

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Ejecutado en client-b

Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.

La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).

La regla se aplica a aquellos datagramas cuya dirección de origen **NO** sea la de vpn-core.

La regla se aplica al puerto **1234**.

Filtrado por IP de origen

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

Ejecutado en vpn-core

Añadimos la regla a la tabla de **nat**, no a la de **filter**.

La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…

La regla se aplicará una vez que el paquete vaya a ser reenviado.

… pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.

Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

Ejecutado en client-b

Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.

La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).

La regla se aplica a aquellos datagramas cuya dirección de origen **NO** sea la de vpn-core.

La regla se aplica al puerto **1234**.

Filtrado por IP de origen

- Esta estrategia se basa en el NATeo de conexiones.

```
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 ! -o wg0 -j MASQUERADE
```

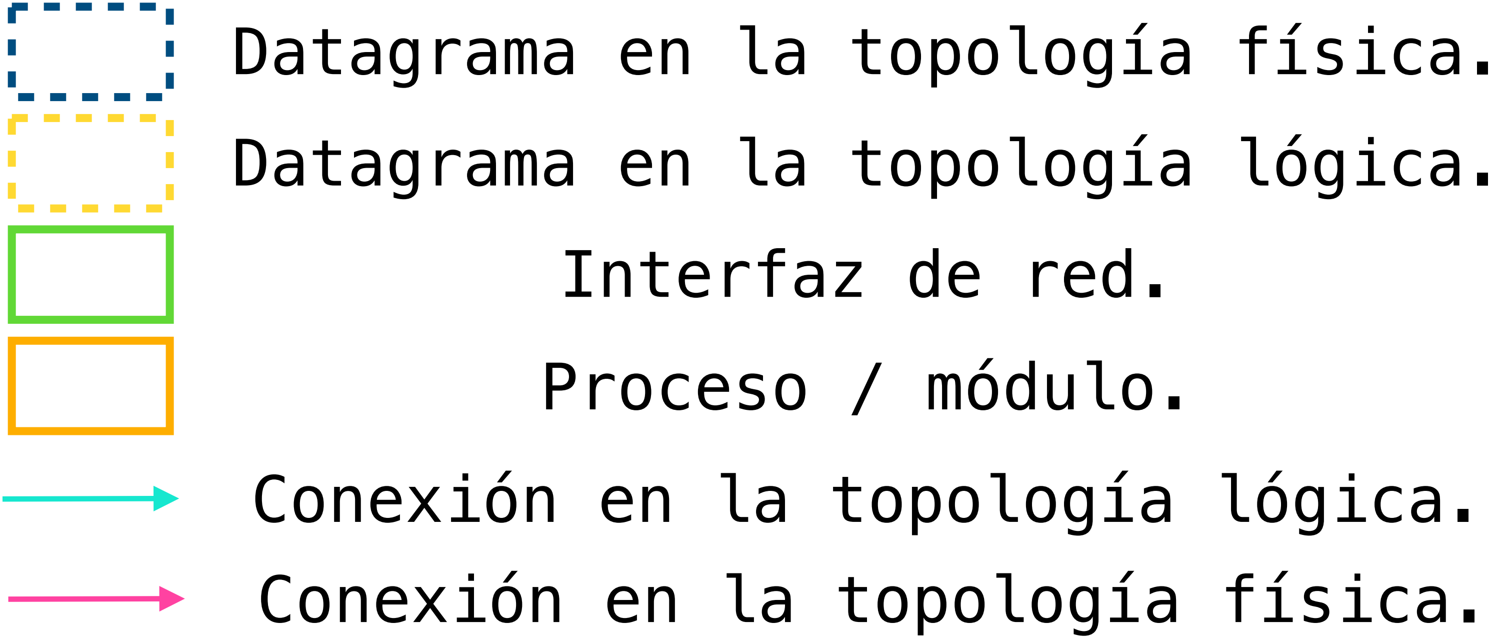
- `iptables`: Ejecutado en vpn-core
- `-t nat`: Añadimos la regla a la tabla de **nat**, no a la de **filter**.
- `-A POSTROUTING`: La regla se aplicará una vez que el paquete vaya a ser reenviado.
- `-s 192.168.4.0/24`: La regla se aplica a todos aquellos paquetes que pertenezcan a la VPN…
- `! -o wg0`: … pero que no vayan destinados a otro miembro de la VPN. Así evitamos NATear tráfico «puro» de la VPN.
- `-j MASQUERADE`: Los paquetes que cumplan las condiciones serán enmascarados (i.e. NATeados).

- El tráfico se «autentica» atravesando `vpn-core`:
  - Al hacerlo el tráfico es NATeado…
  - … con lo que a dirección de origen de los datagramas es la de `vpn-core`.
- La solución requiere que el núcleo de la VPN tenga una `IPv4` estática…
- Aseguramos los servicios filtrando por `IPv4` de origen: la de `vpn-core`.

```
iptables -A INPUT -p tcp ! -s 10.0.123.2/32 --dport 1234 -j DROP
```

- `iptables`: Ejecutado en client-b
- `-A INPUT`: Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.
- `-p tcp`: La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).
- `! -s 10.0.123.2/32`: La regla se aplica a aquellos datagramas cuya dirección de origen **NO** sea la de vpn-core.
- `--dport 1234`: La regla se aplica al puerto **1234**.
- `-j DROP`: Los paquetes que cumplan la regla se **descartan**.

Another day in the life…

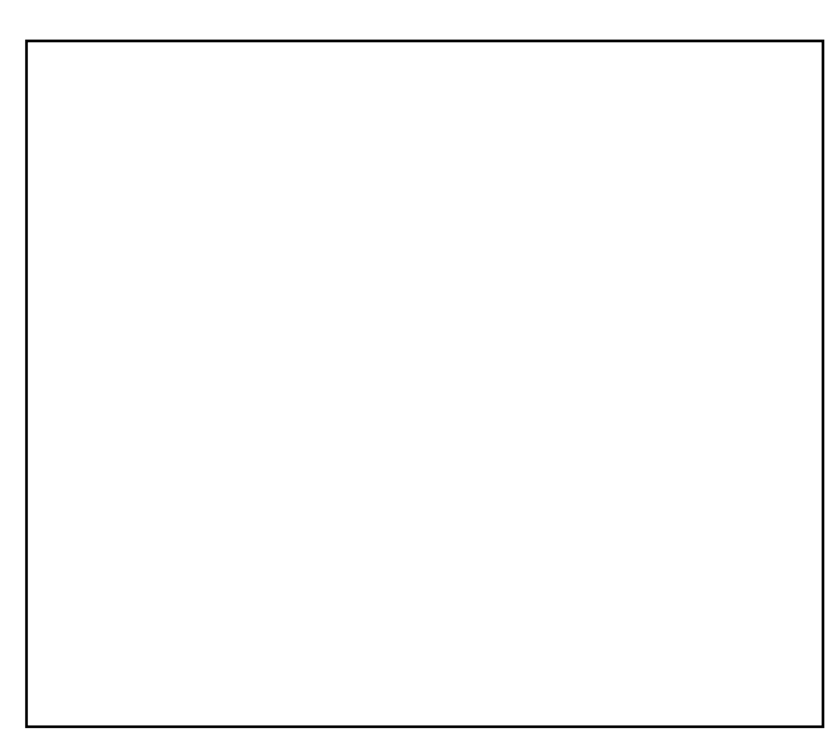


client-a

10.0.123.3

client-b

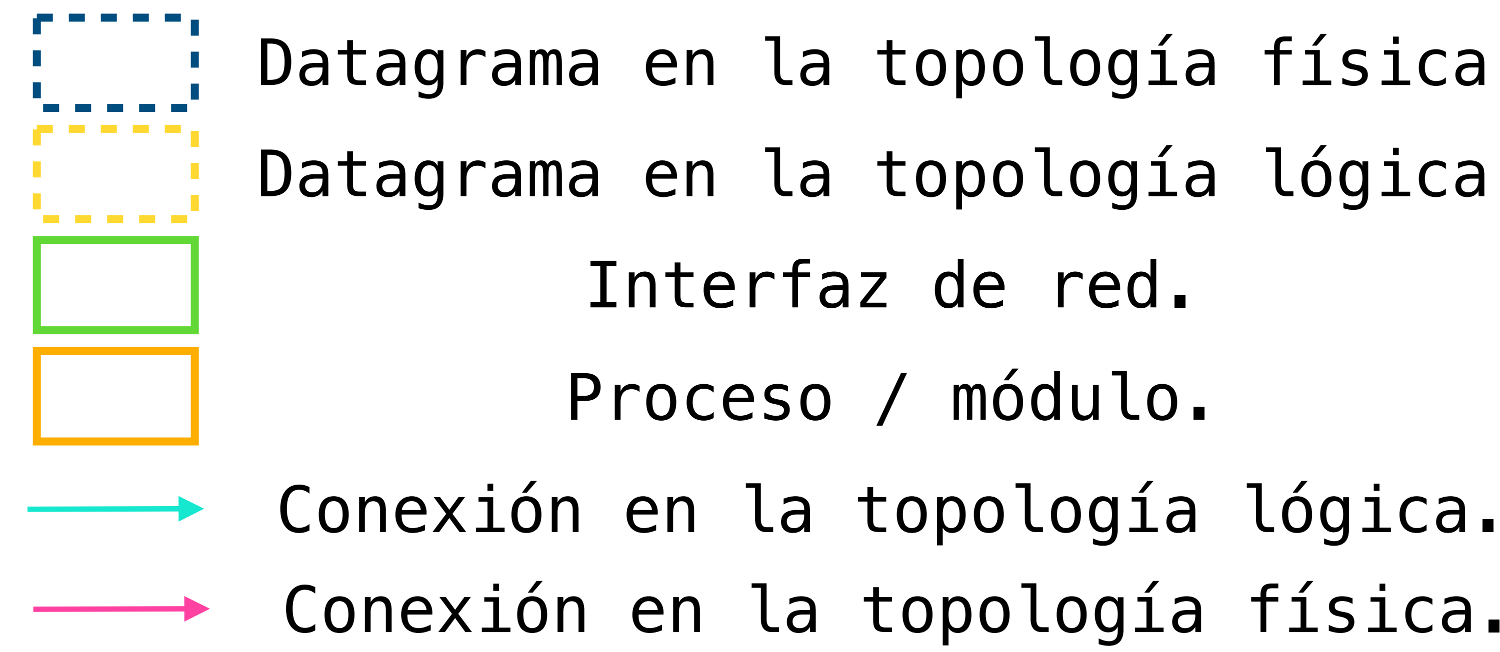
10.0.123.4



10.0.123.2

vpn-core

Another day in the life...



client-a

10.0.123.3

nc

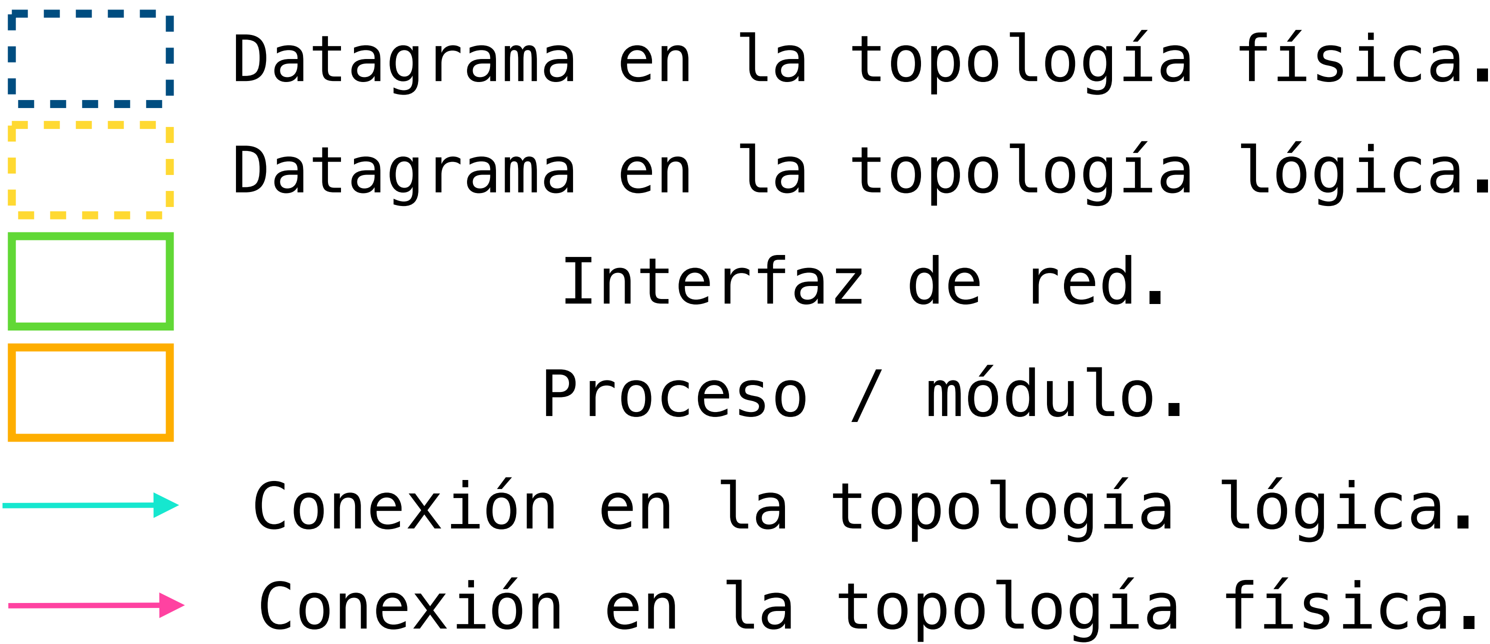
10.0.123.2

vpn-core

client-b

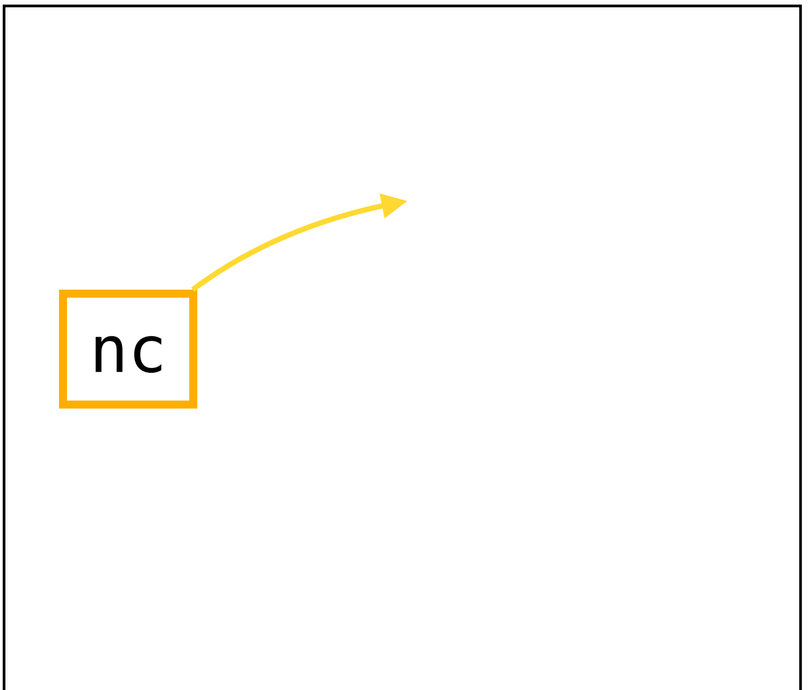
10.0.123.4

Another day in the life…



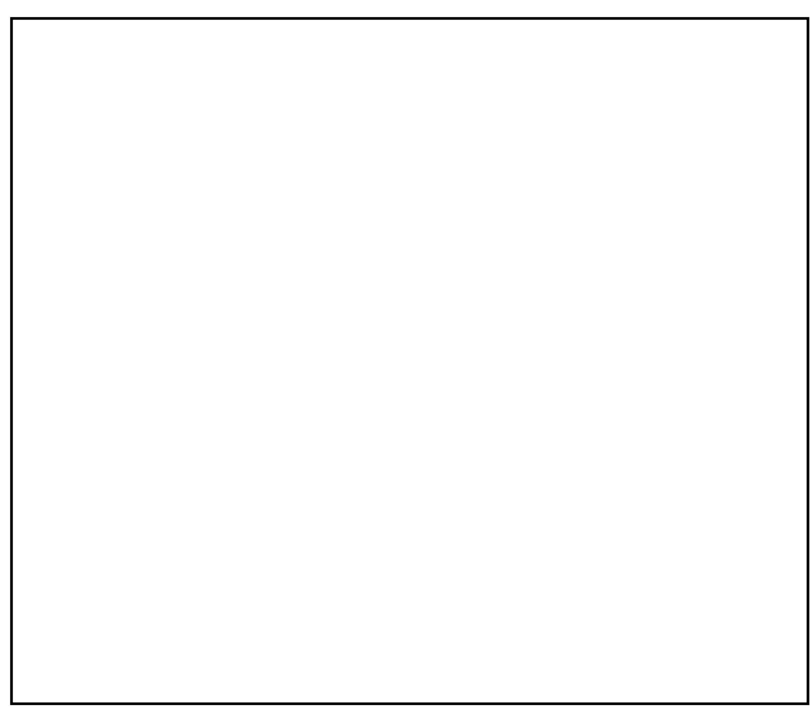
client-a

10.0.123.3



client-b

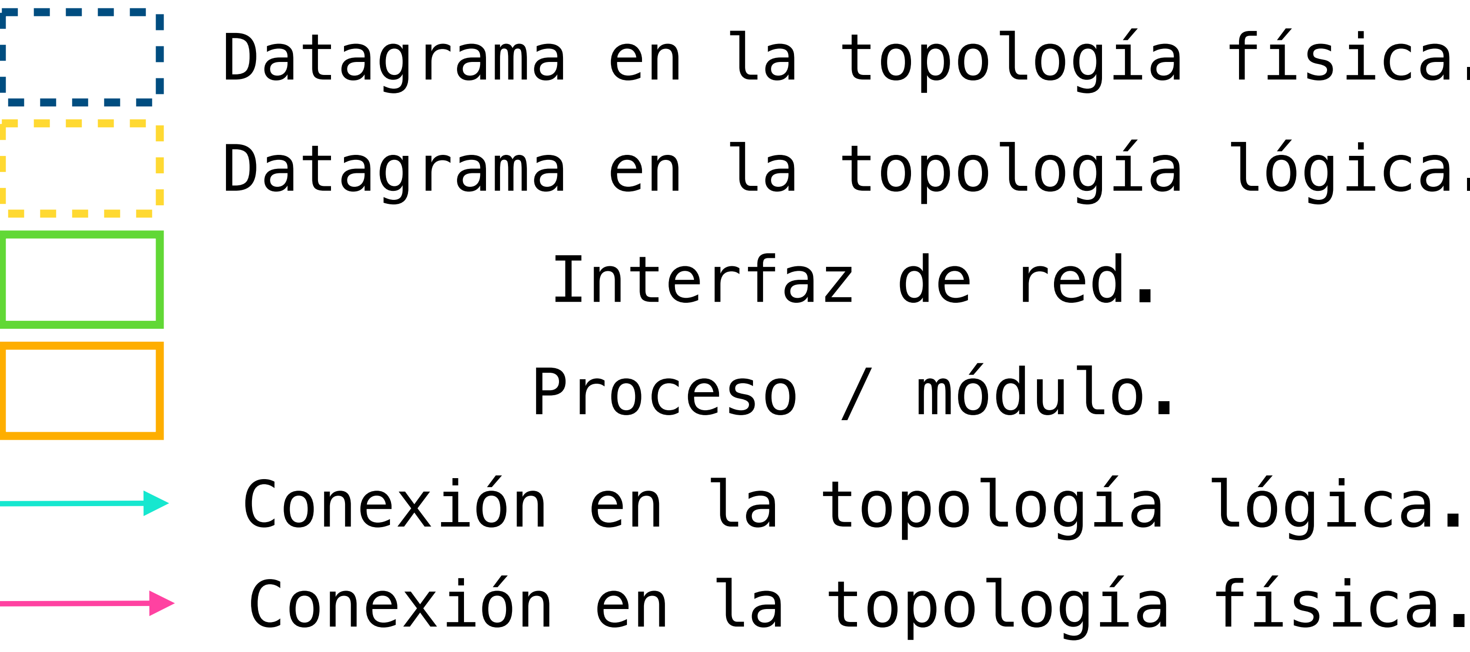
10.0.123.4



10.0.123.2

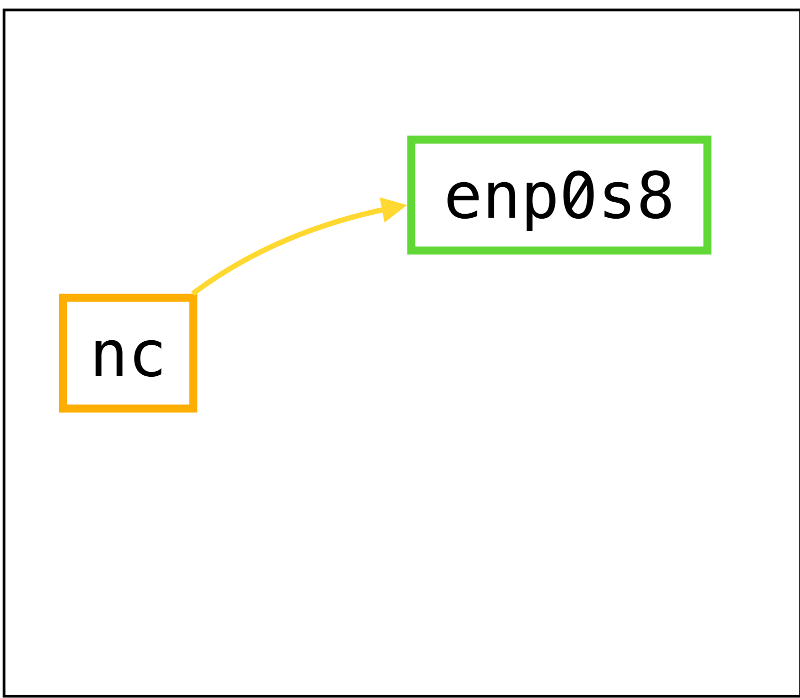
vpn-core

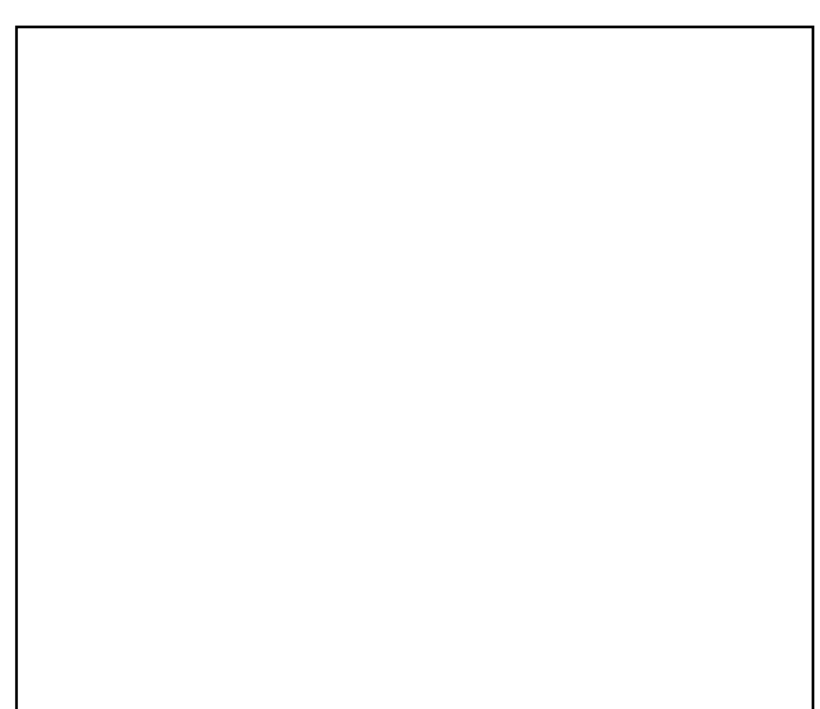
Another day in the life...



client-a

10.0.123.3





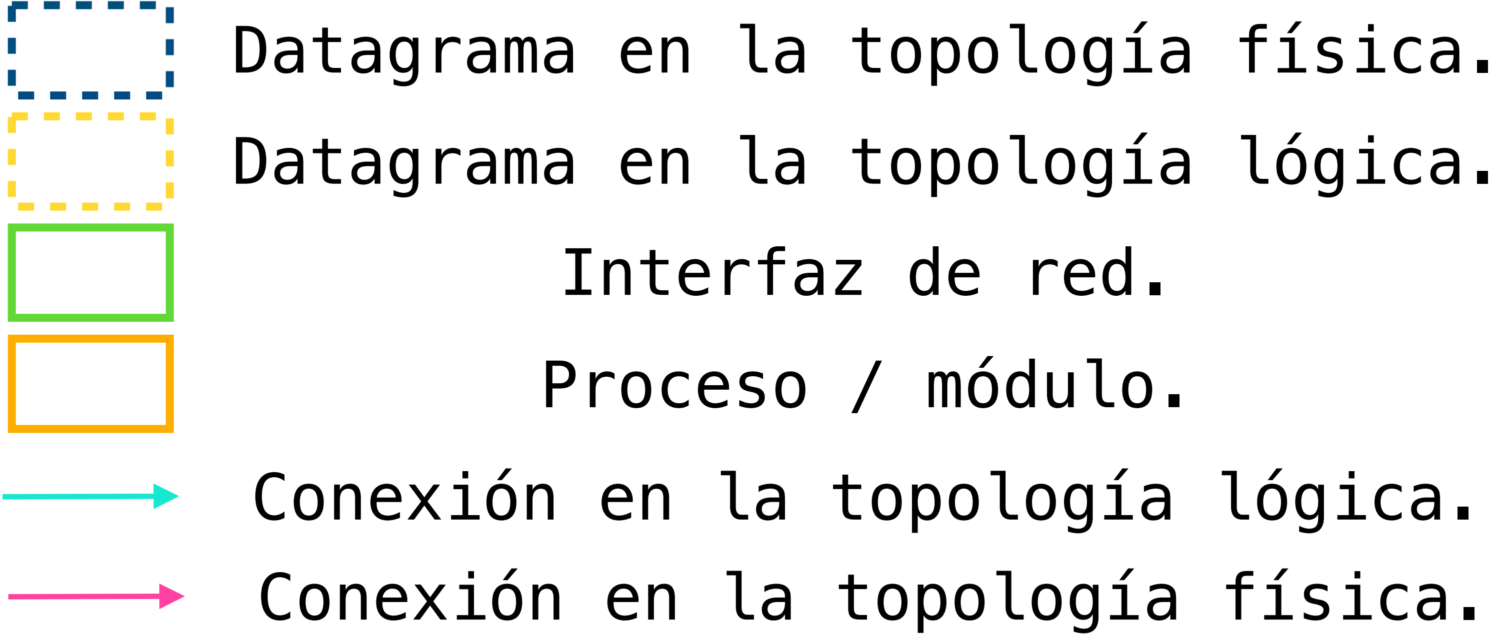
10.0.123.2

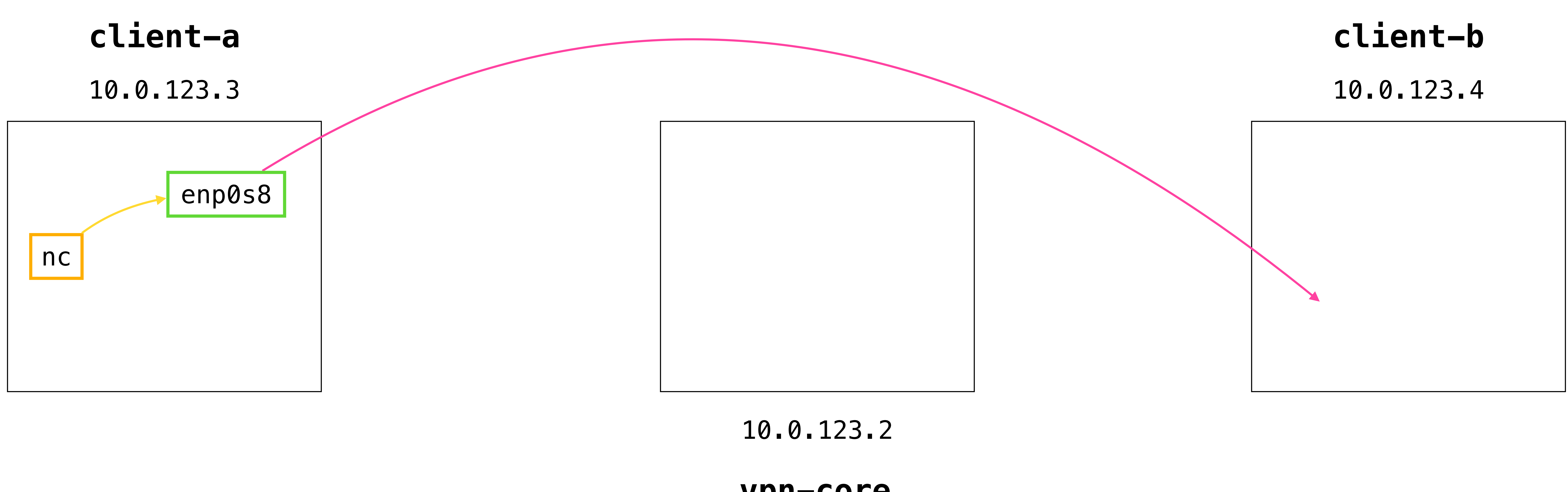
vpn-core

client-b

10.0.123.4

Another day in the life...





Another day in the life...

Datagrama en la topología física.
Datagrama en la topología lógica.
Interfaz de red.
Proceso / módulo.
Conexión en la topología lógica.
Conexión en la topología física.

From: 10.0.123.3
To: 10.0.123.4

client-a

10.0.123.3

enp0s8

nc

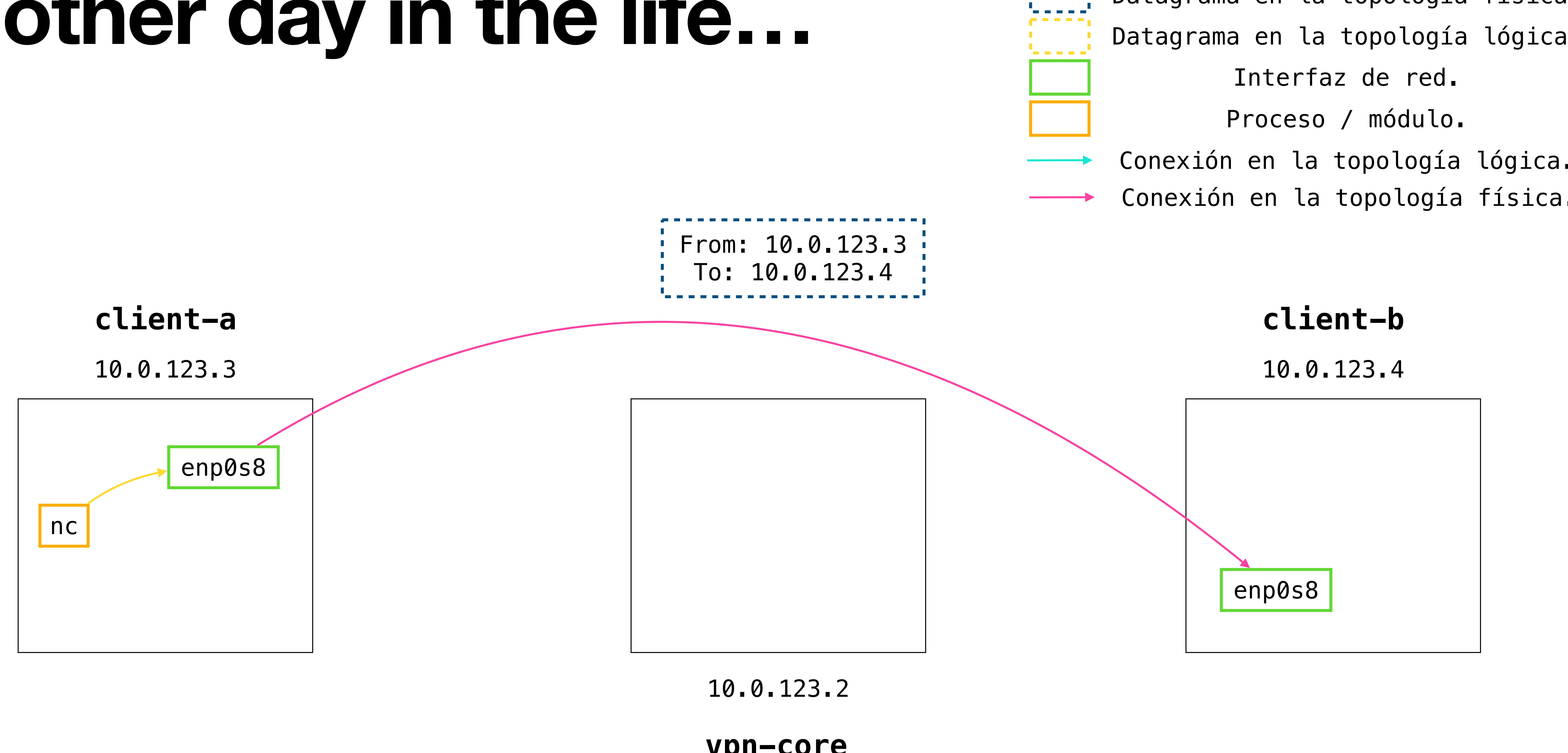
10.0.123.2

vpn-core

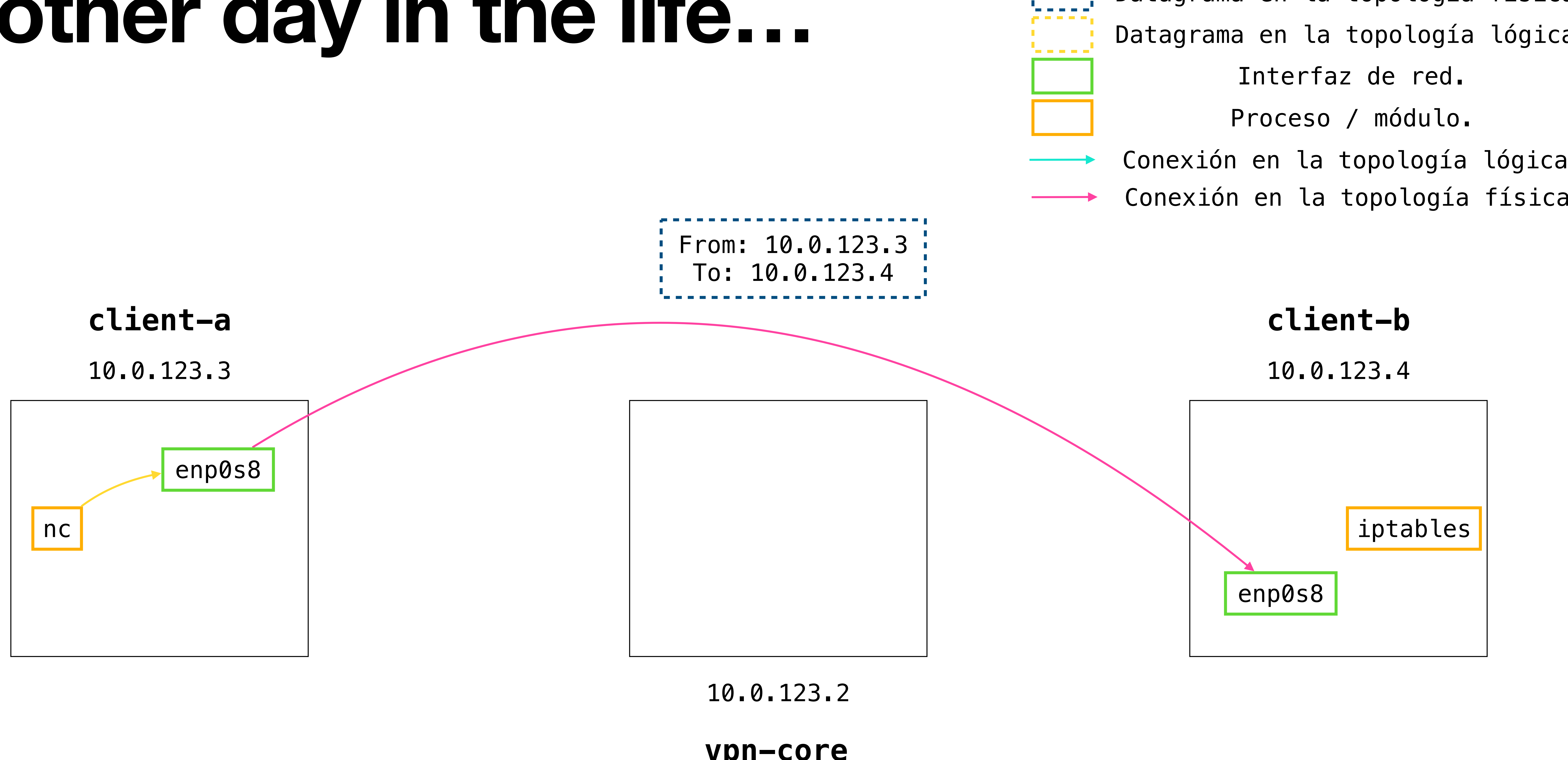
client-b

10.0.123.4

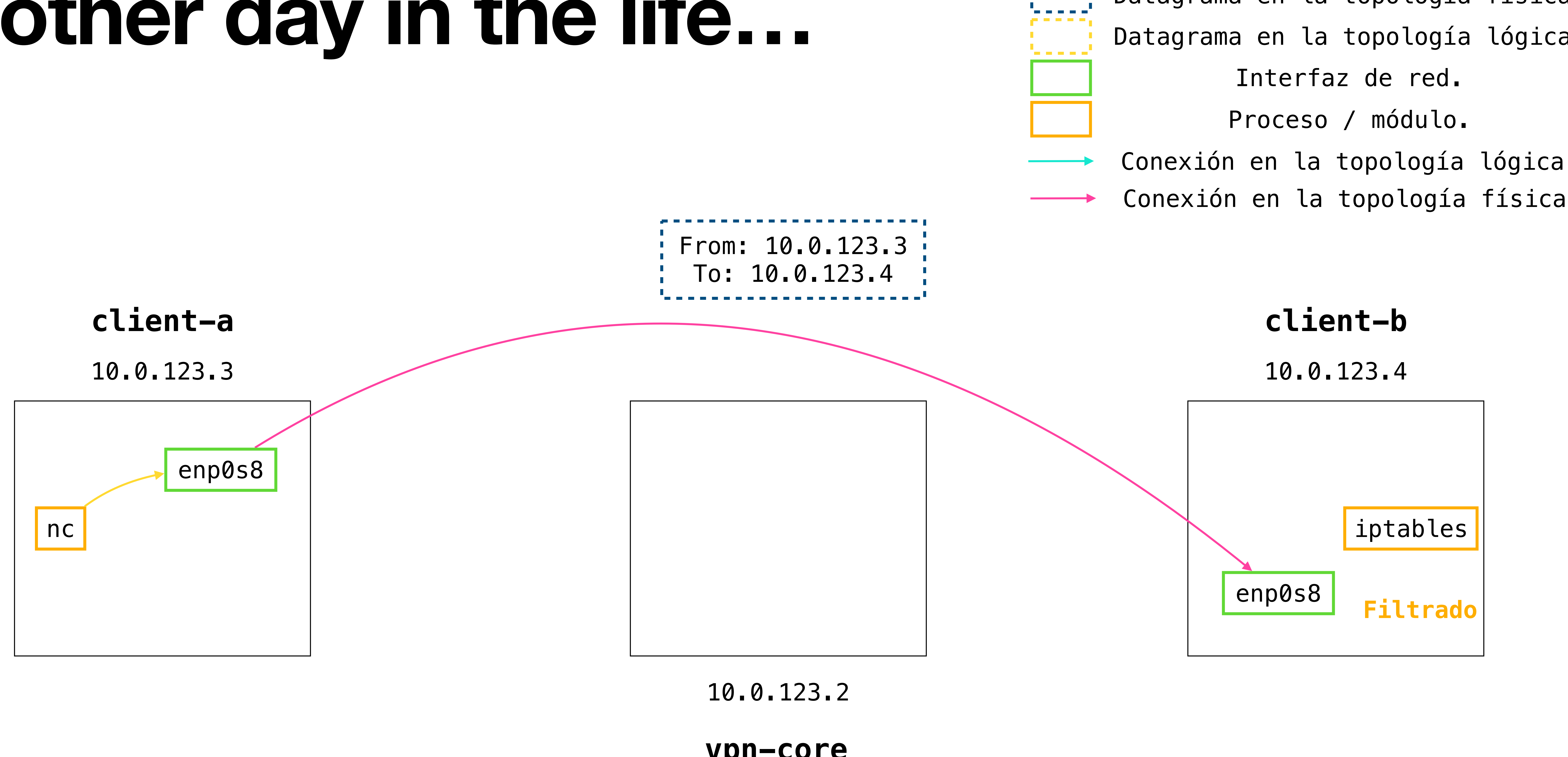
Another day in the life...



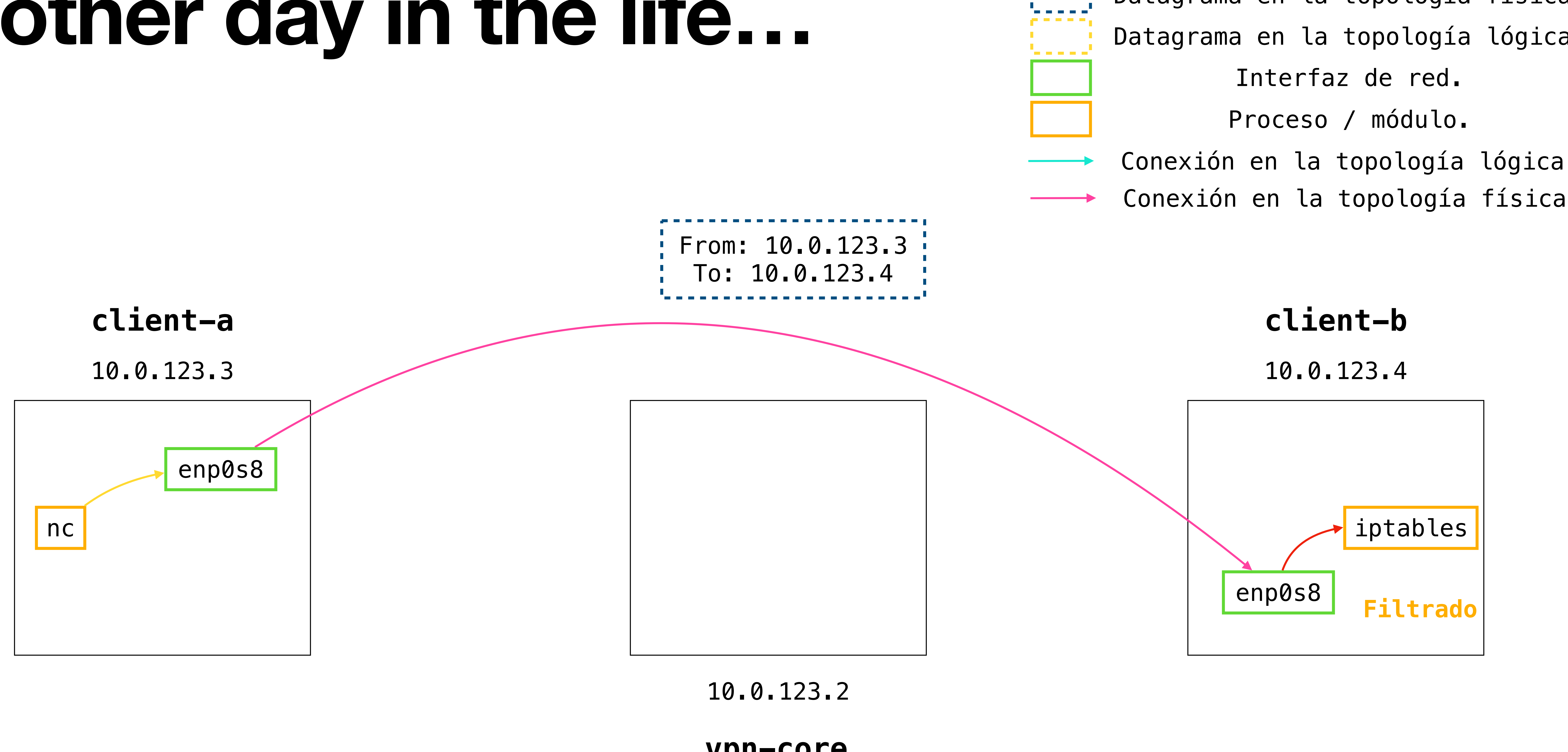
Another day in the life...



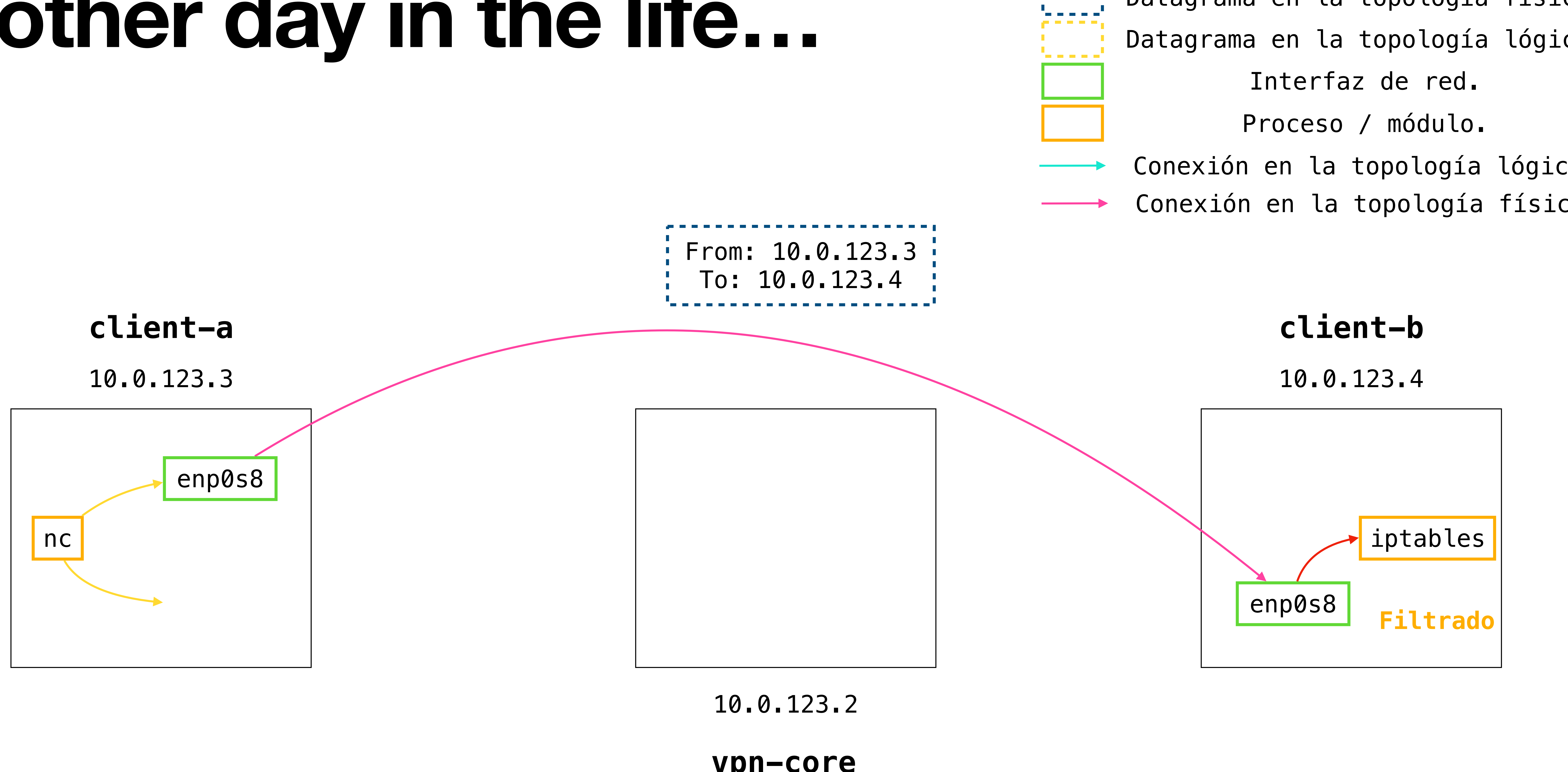
Another day in the life...



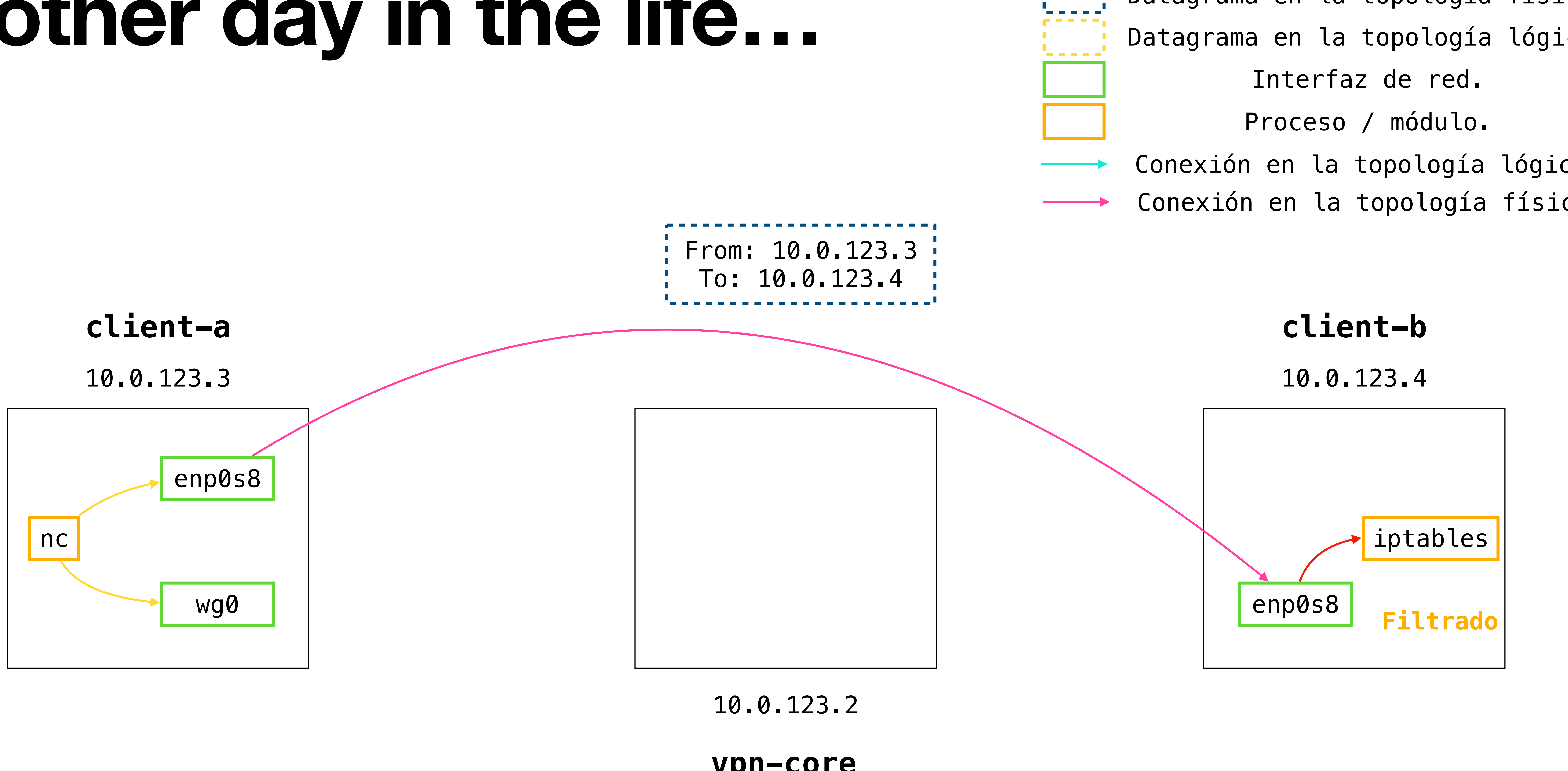
Another day in the life...



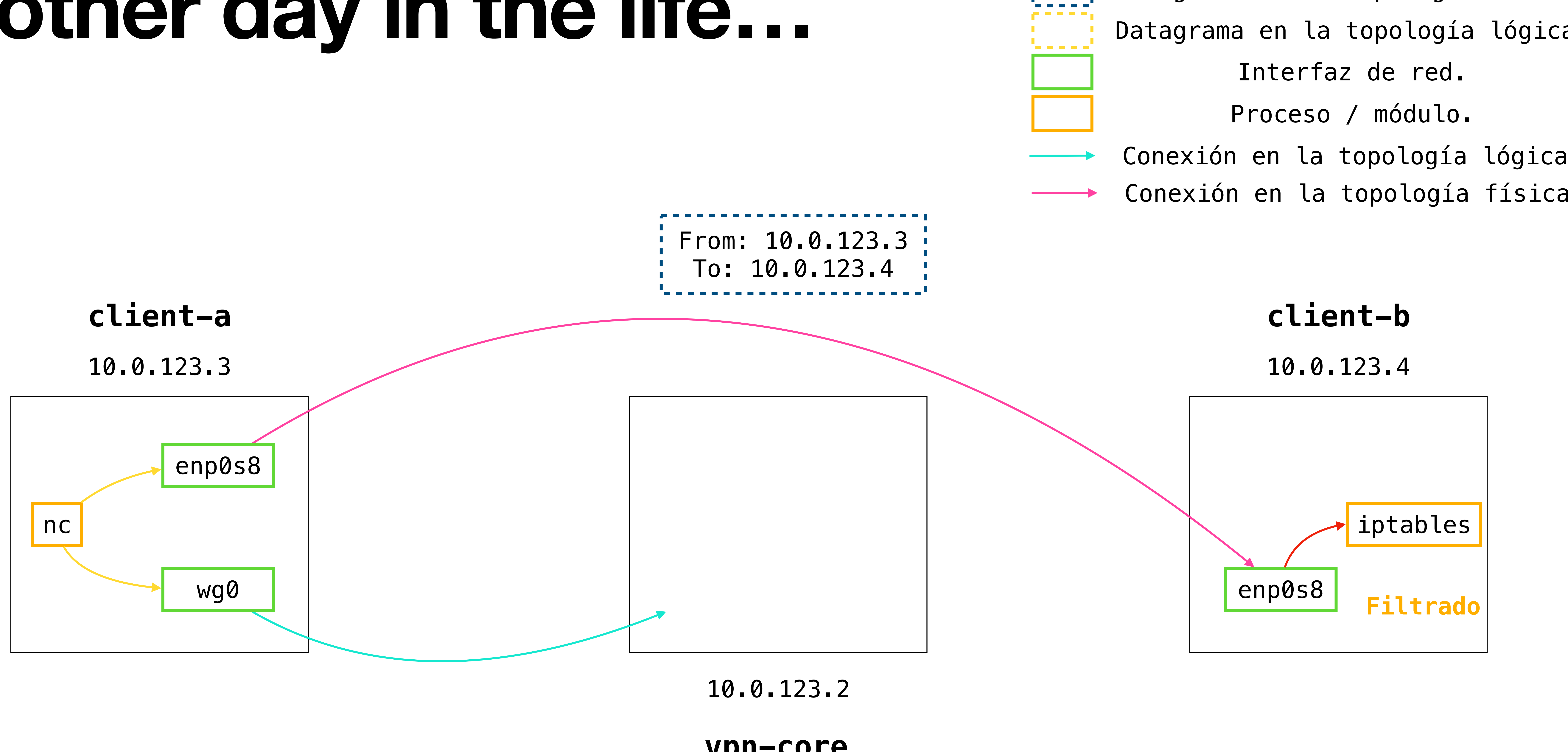
Another day in the life...



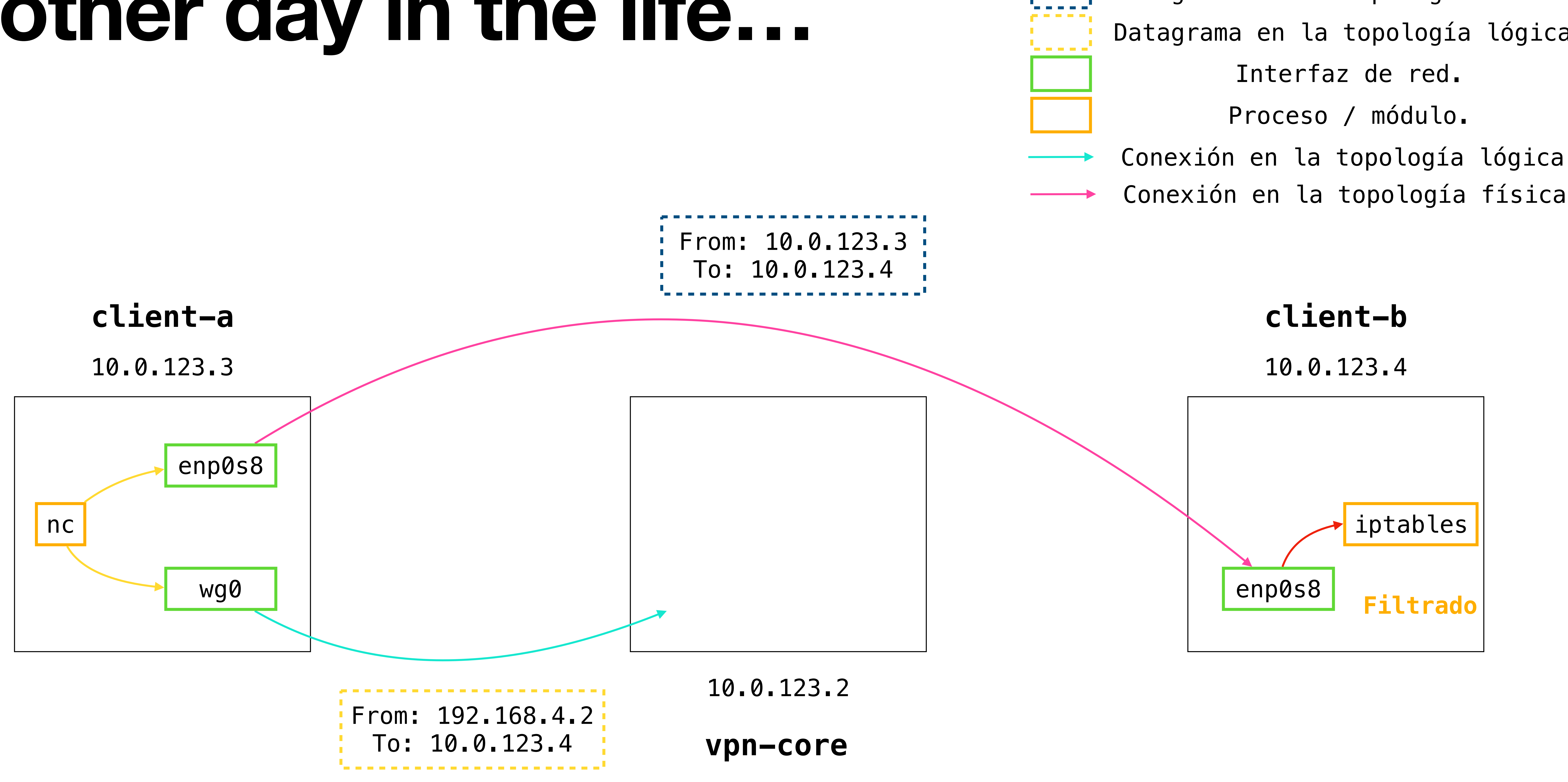
Another day in the life...



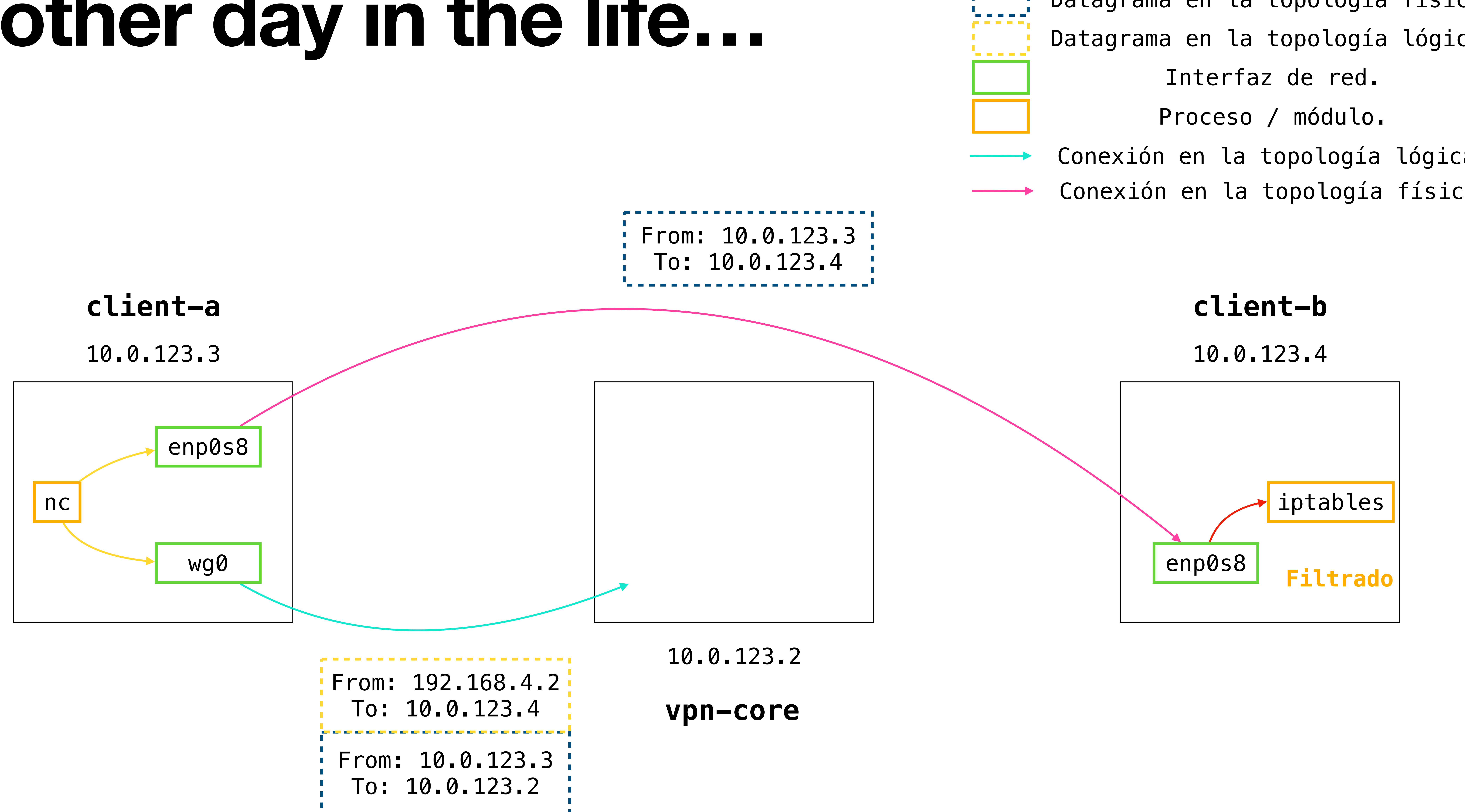
Another day in the life...



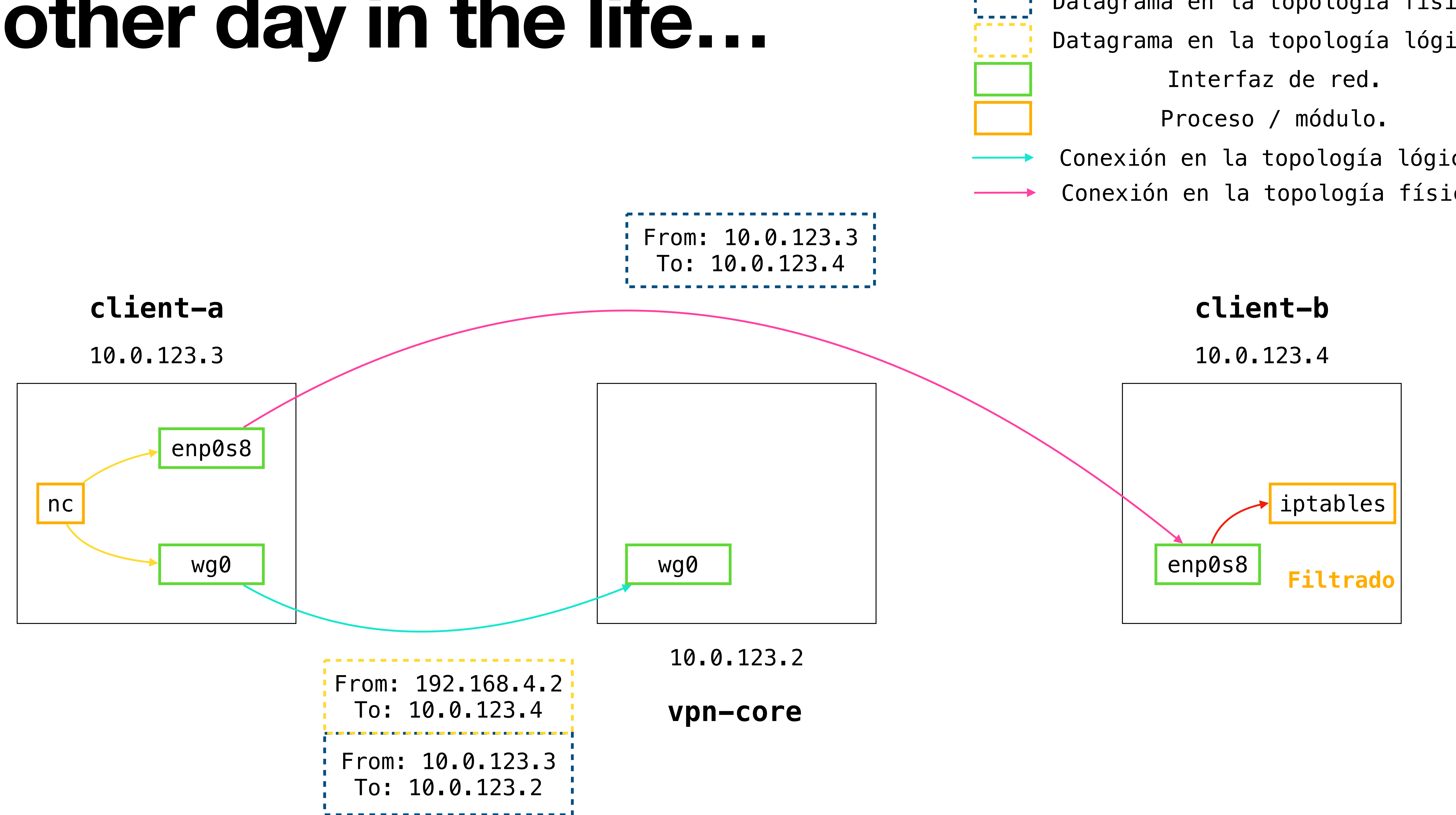
Another day in the life...



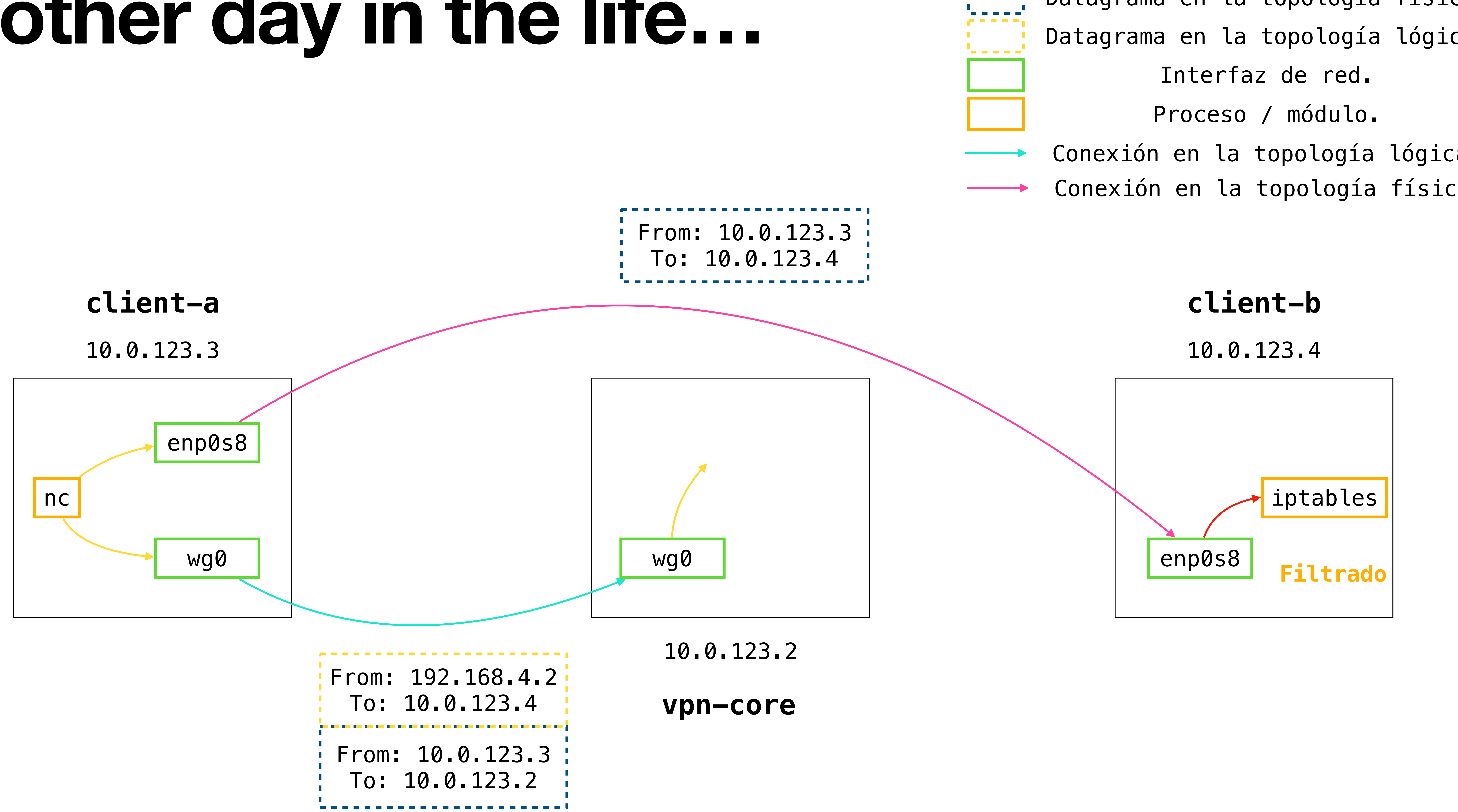
Another day in the life...



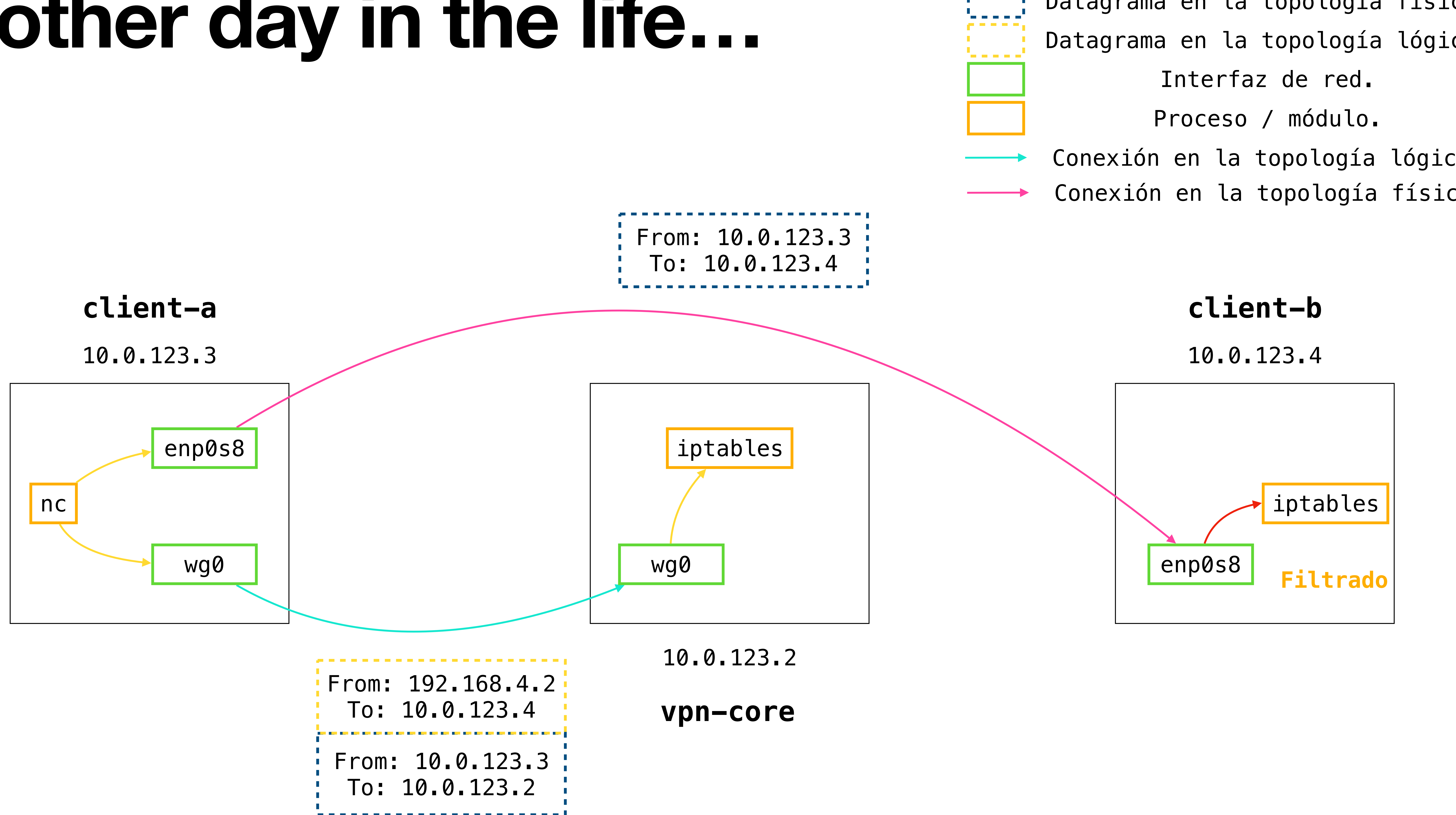
Another day in the life...



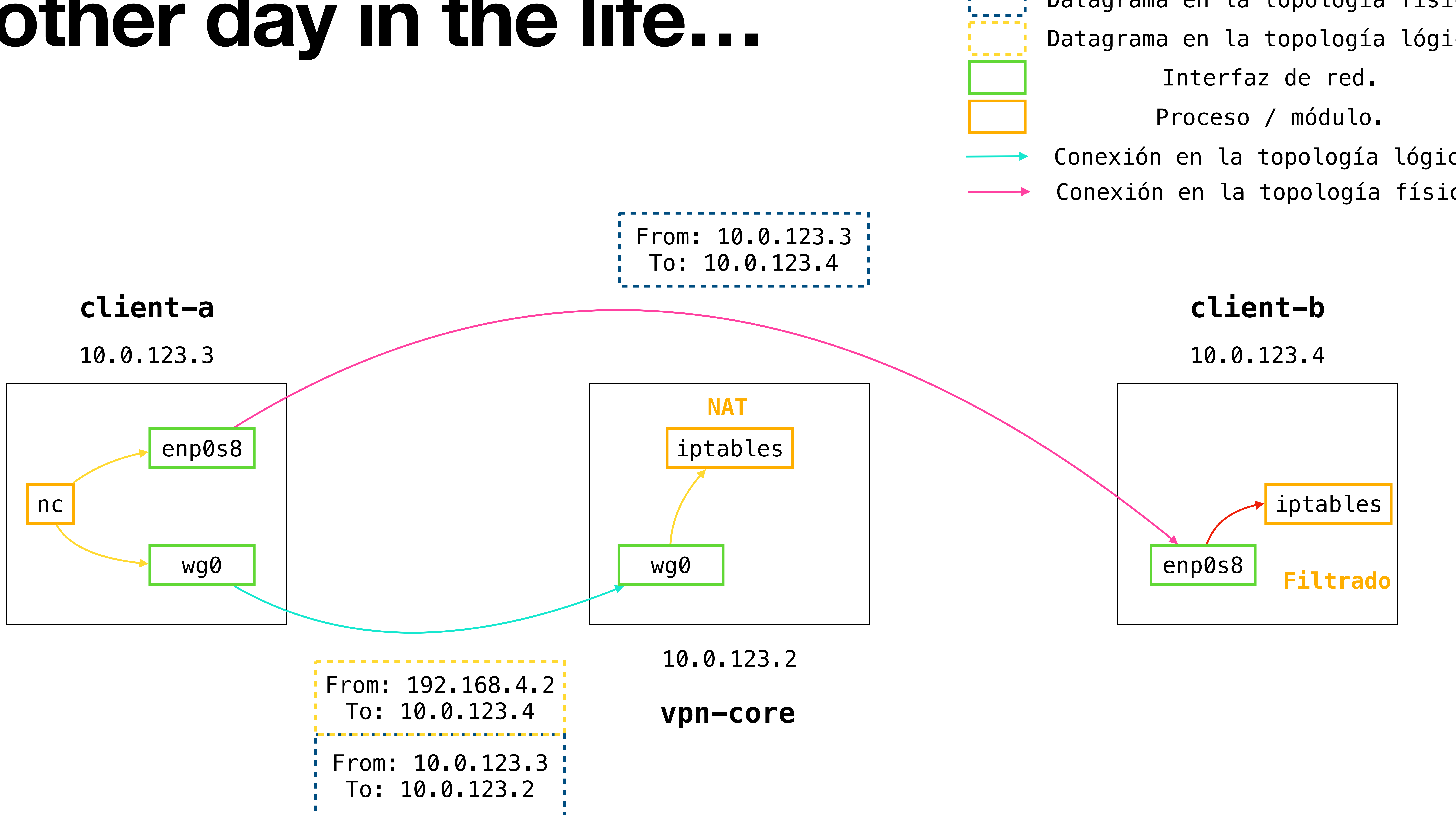
Another day in the life...



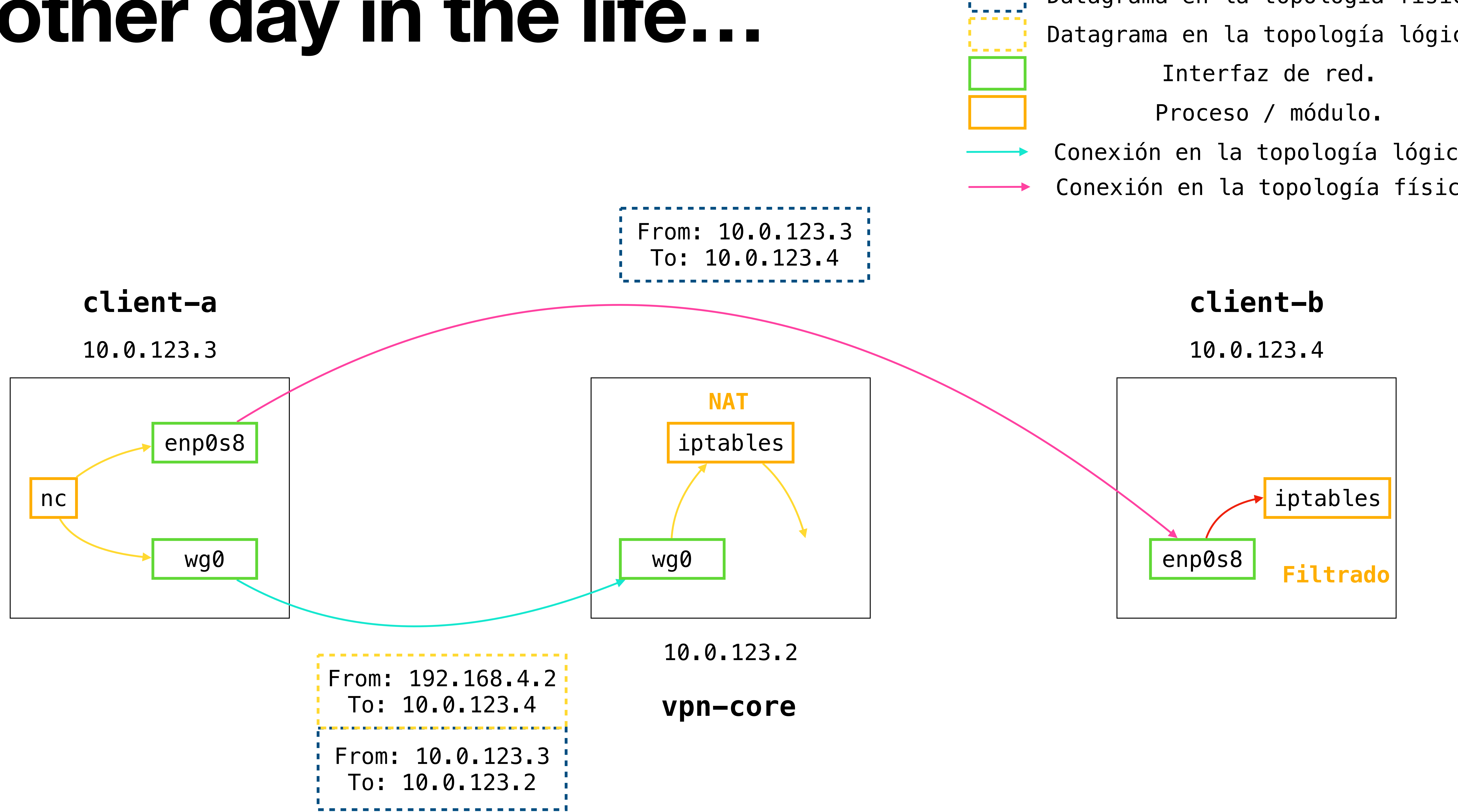
Another day in the life...



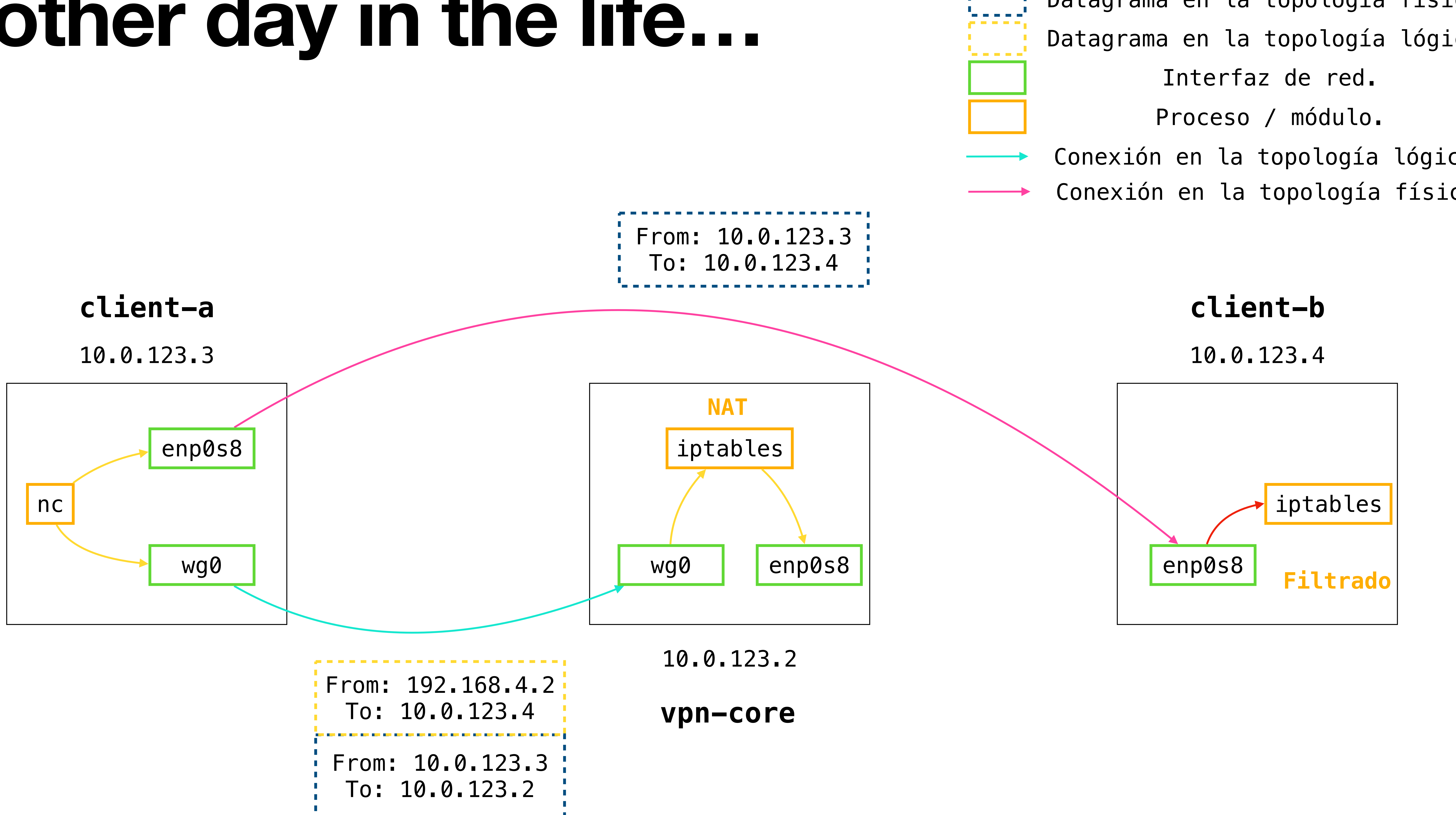
Another day in the life...



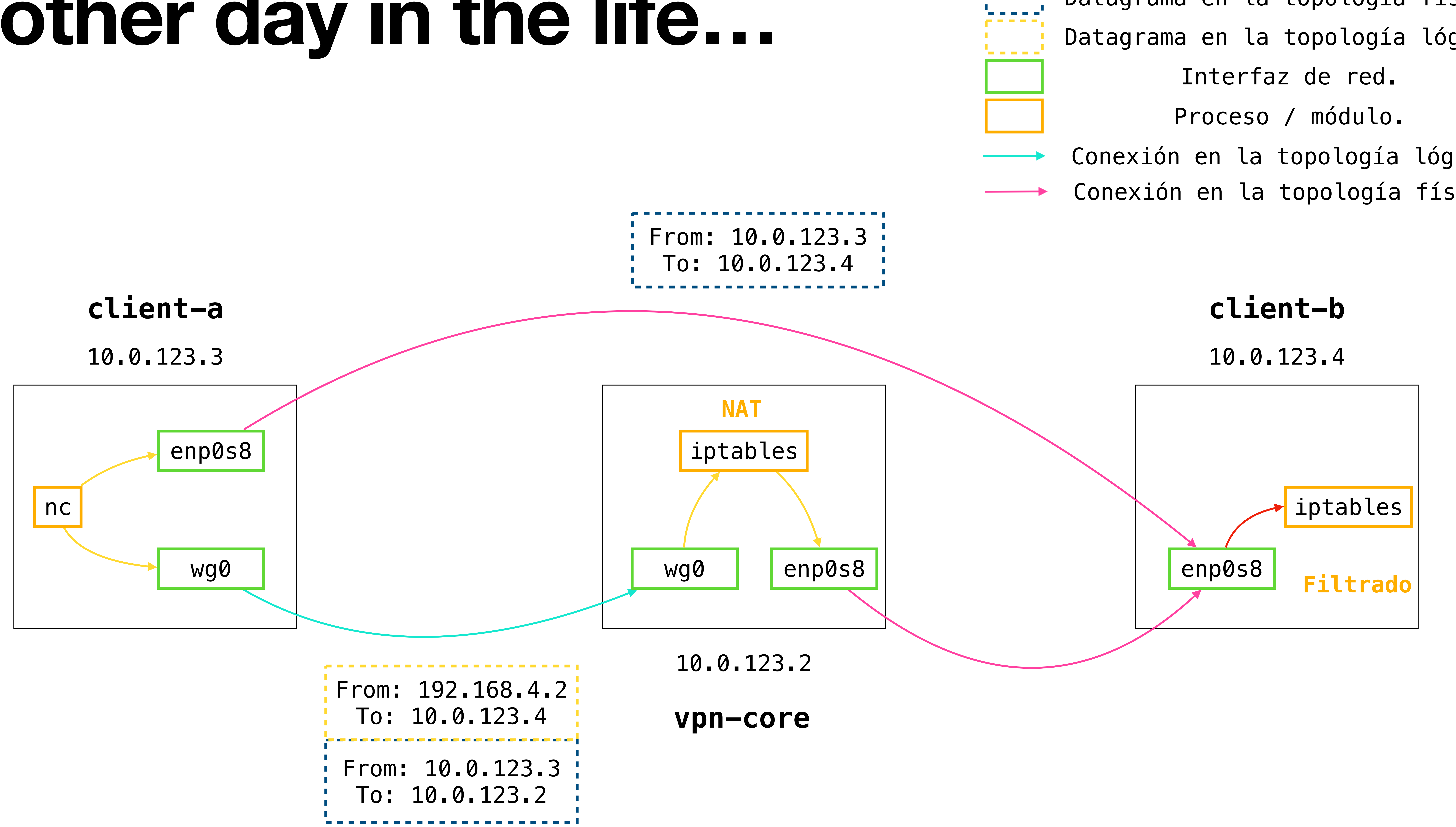
Another day in the life…

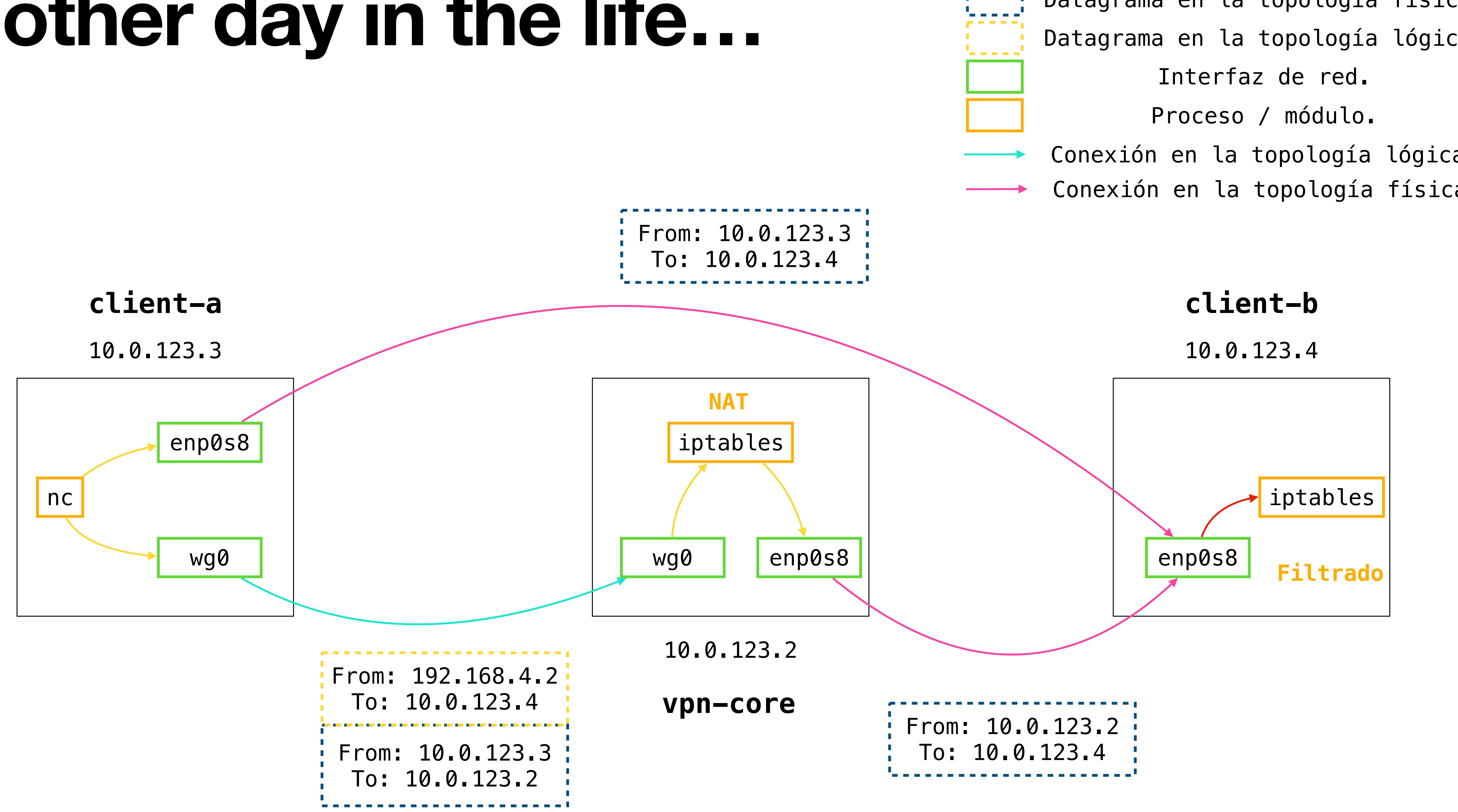


Another day in the life…



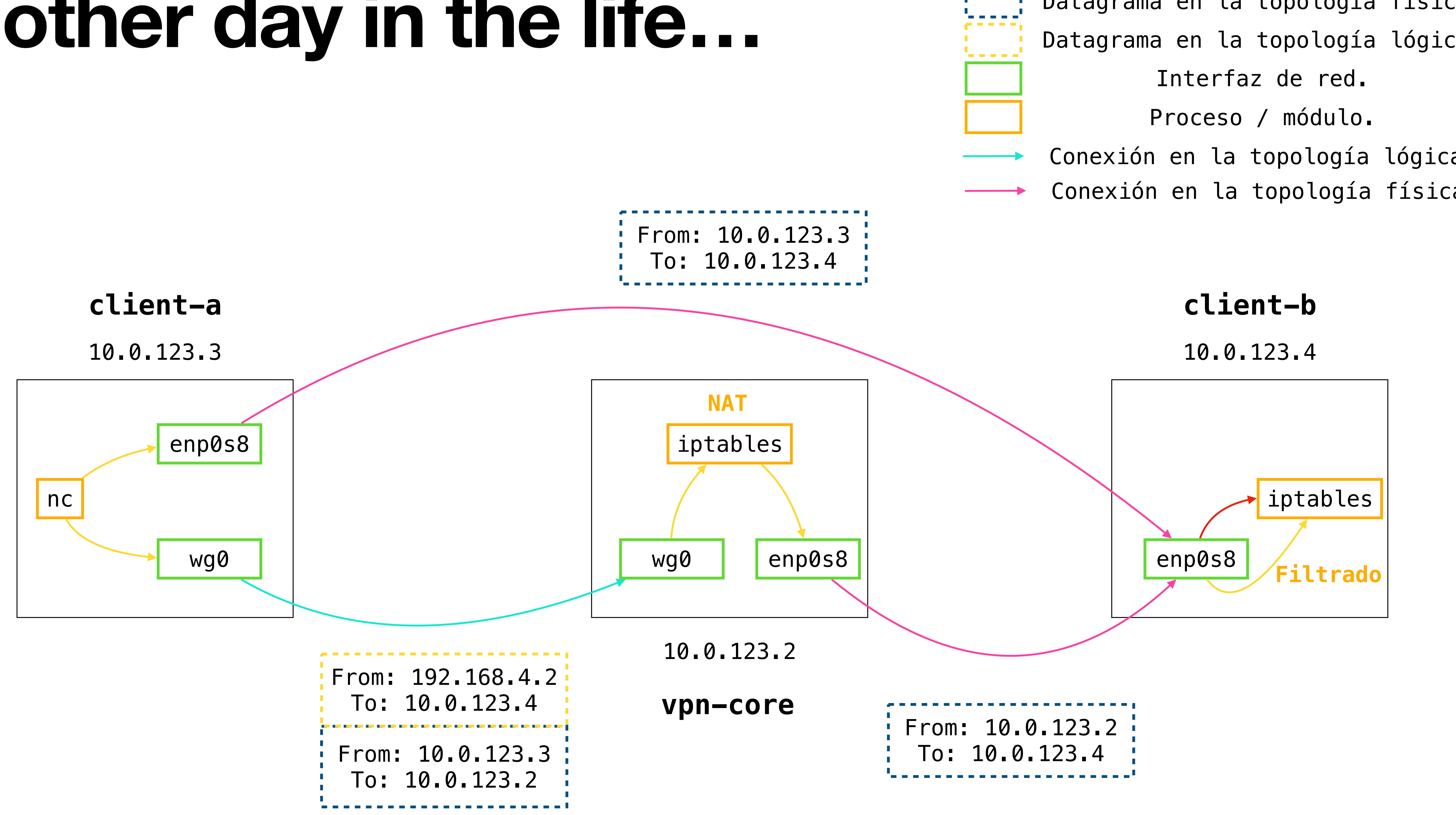
Another day in the life...



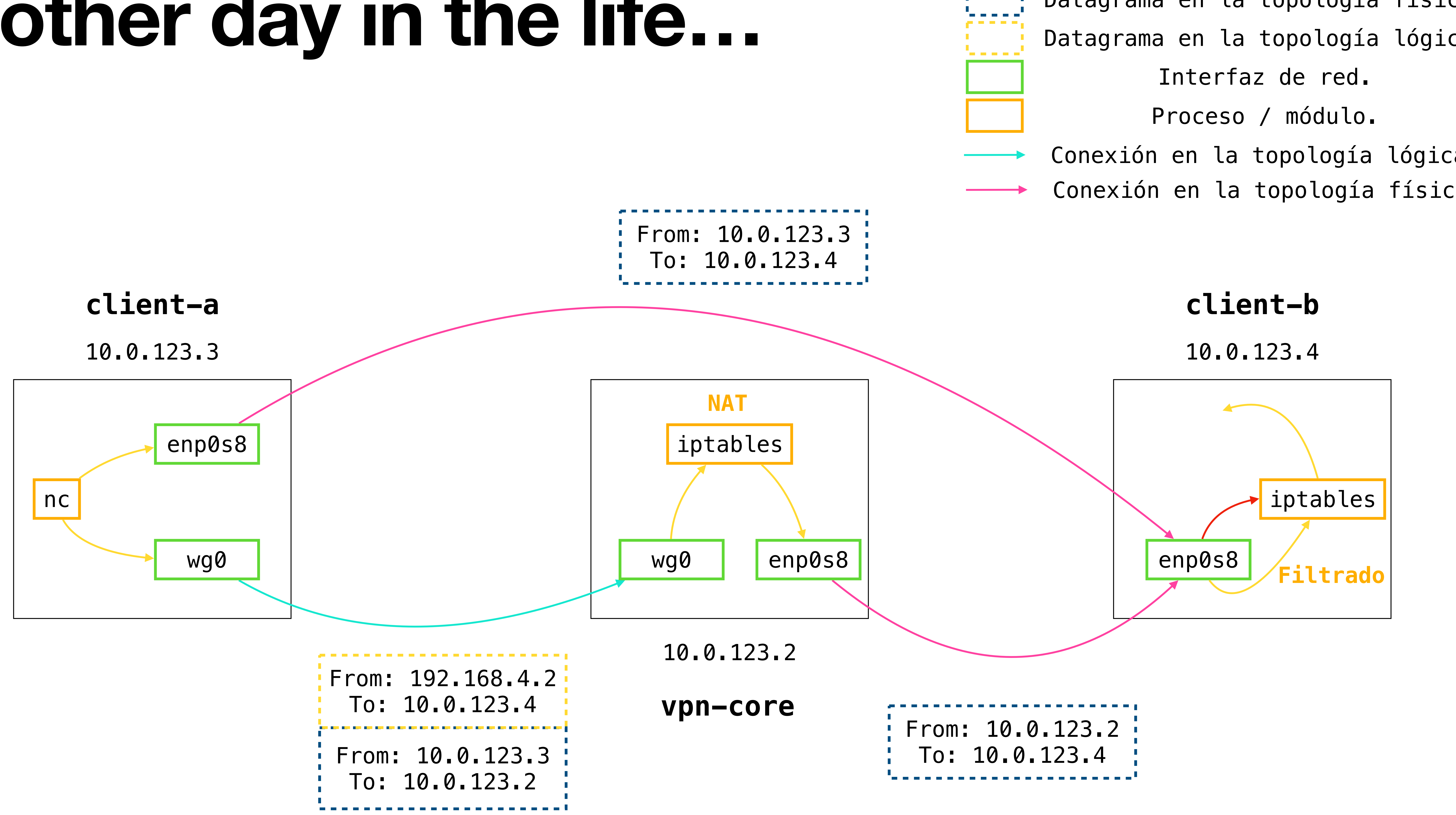


Another day in the life...
Datagrama en la topología física.
Datagrama en la topología lógica.
Interfaz de red.
Proceso / módulo.
Conexión en la topología lógica.
Conexión en la topología física.
From: 10.0.123.3
To: 10.0.123.4
client-a
10.0.123.3
enp0s8
nc
wg0
NAT
iptables
wg0
enp0s8
client-b
10.0.123.4
iptables
enp0s8
Filtrado
From: 192.168.4.2
To: 10.0.123.4
From: 10.0.123.3
To: 10.0.123.2
10.0.123.2
vpn-core
From: 10.0.123.2
To: 10.0.123.4

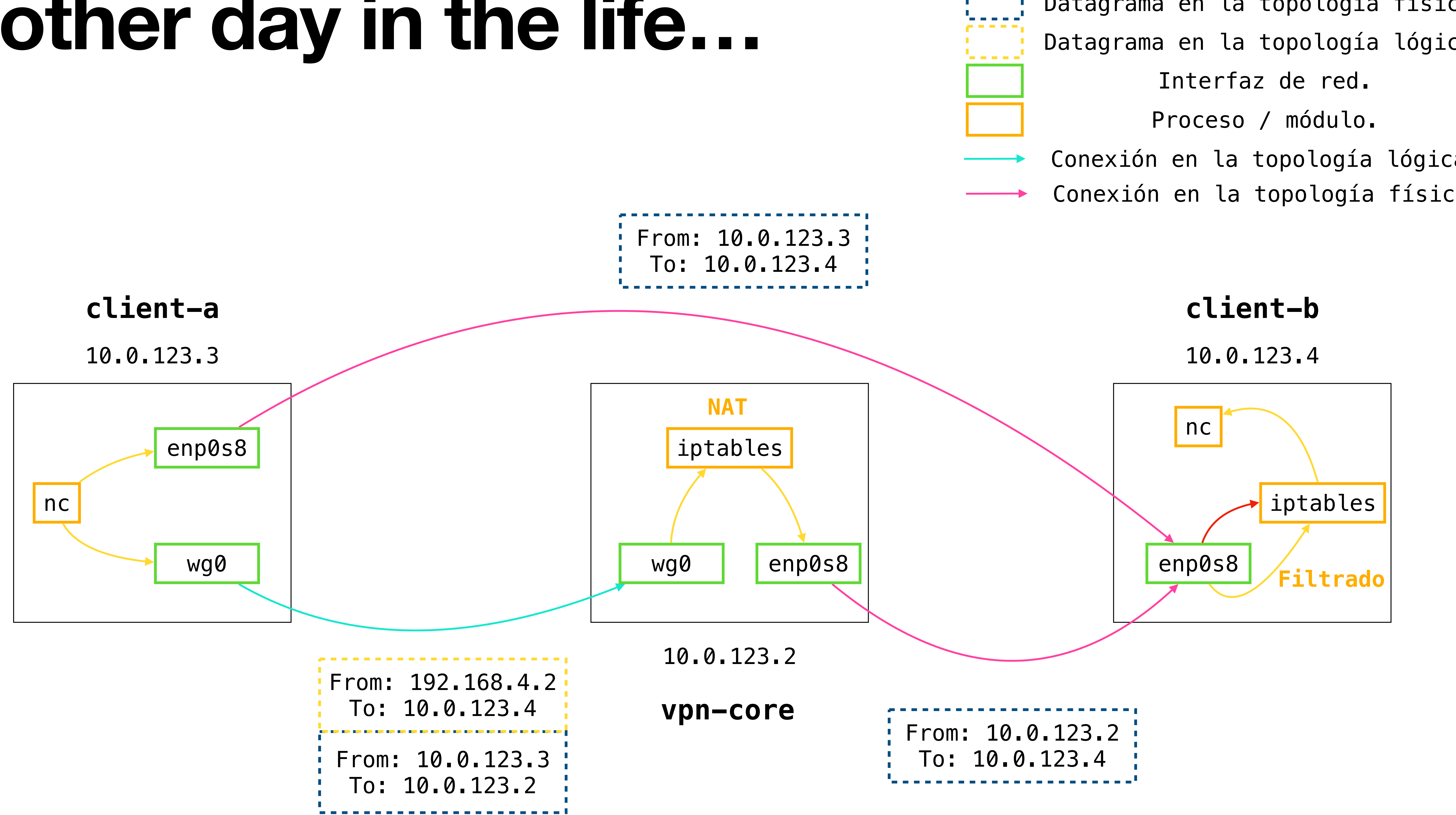
Another day in the life...



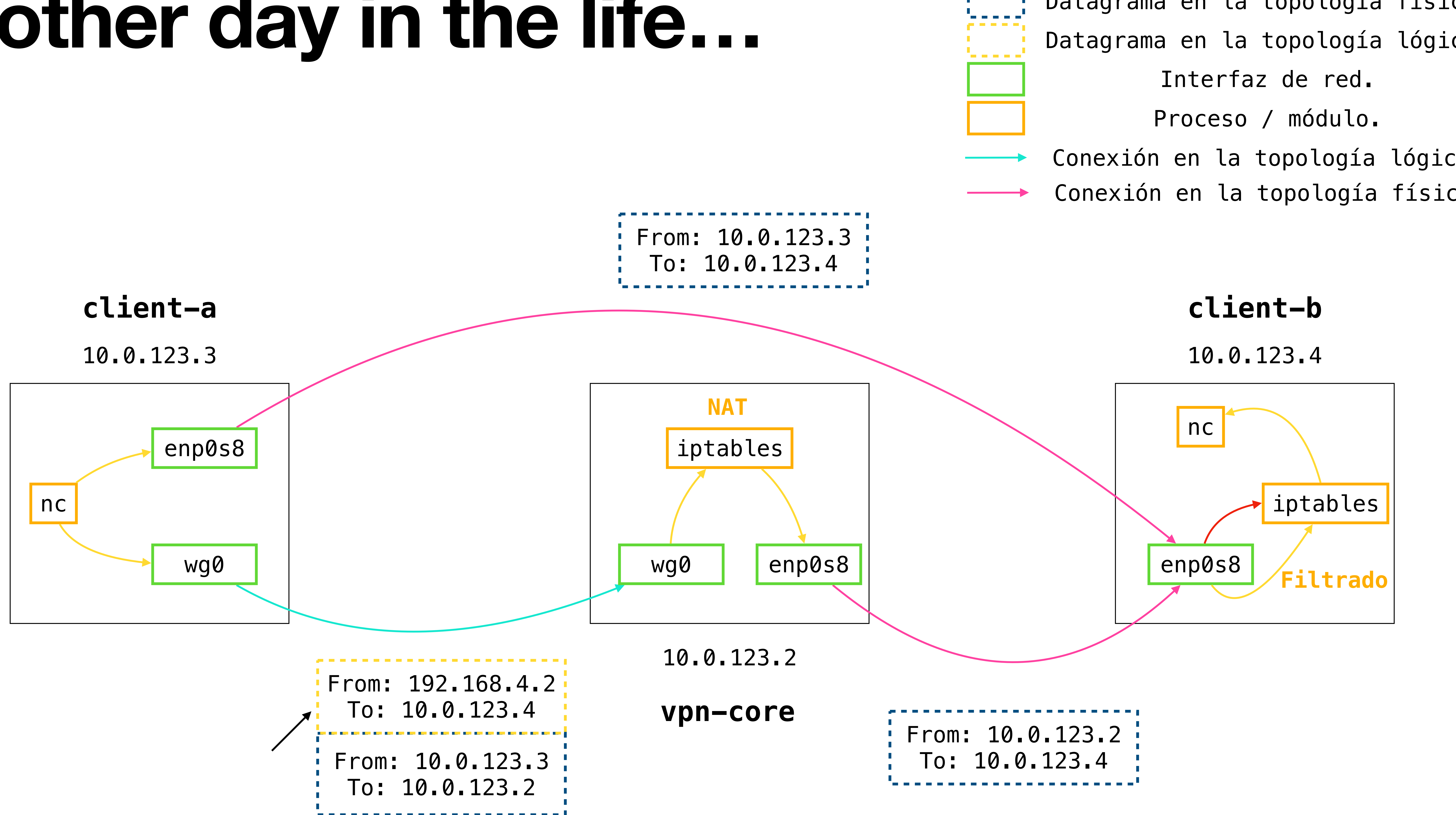
Another day in the life...



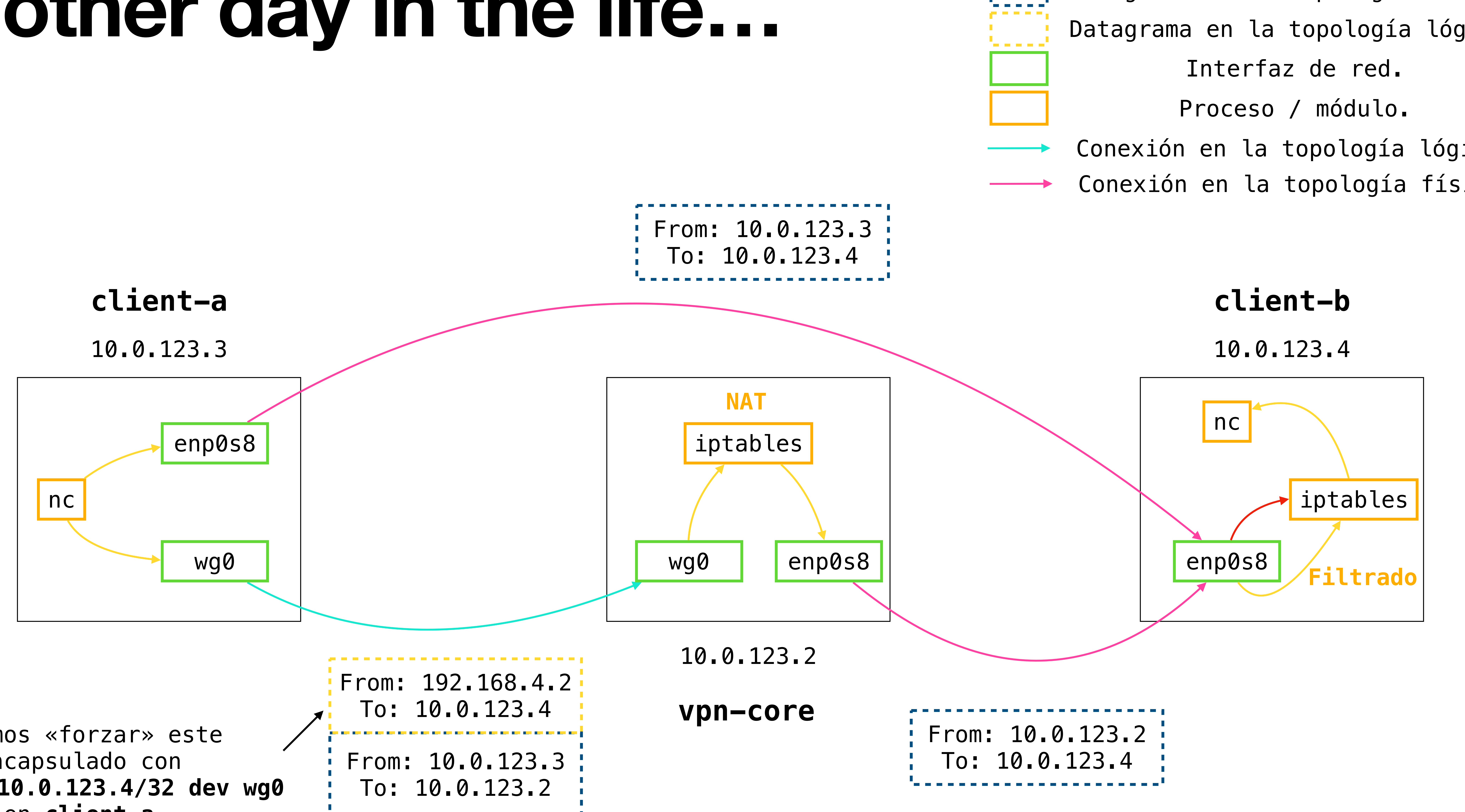
Another day in the life...



Another day in the life...



Another day in the life...



Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

Ejecutado en
client-b

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

Ejecutado en client-b

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

Ejecutado en client-b

Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

Ejecutado en client-b

Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

Ejecutado en client-b

Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.

La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

- `iptables`: Ejecutado en client-b
- `-A INPUT`: Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.
- `-p tcp`: La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

- `iptables`: Ejecutado en client-b
- `-A INPUT`: Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.
- `-p tcp`: La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).
- `! -i wg0`: La regla se aplica a aquellos datagramas cuya interfaz de ingreso **NO** haya sido wg0.

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

- `iptables`: Ejecutado en client-b
- `-A INPUT`: Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.
- `-p tcp`: La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).
- `! -i wg0`: La regla se aplica a aquellos datagramas cuya interfaz de ingreso **NO** haya sido wg0.

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

- `iptables`: Ejecutado en client-b
- `-A INPUT`: Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.
- `-p tcp`: La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).
- `! -i wg0`: La regla se aplica a aquellos datagramas cuya interfaz de ingreso **NO** haya sido wg0.
- `--dport 1234`: La regla se aplica al puerto **1234**.

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

- `iptables`: Ejecutado en client-b
- `-A INPUT`: Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.
- `-p tcp`: La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).
- `! -i wg0`: La regla se aplica a aquellos datagramas cuya interfaz de ingreso **NO** haya sido wg0.
- `--dport 1234`: La regla se aplica al puerto **1234**.
- `-j DROP`

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

`iptables`	`-A INPUT`	`-p tcp`	`! -i wg0`	`--dport 1234`	`-j DROP`
Ejecutado en client-b	Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.	La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).	La regla se aplica a aquellos datagramas cuya interfaz de ingreso **NO** haya sido wg0.	La regla se aplica al puerto **1234.**	Los paquetes que cumplan la regla se **descartan.**

- También podríamos filtrar con **`! -s 192.168.4.0/24`**: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Filtrado por pertenencia a la VPN

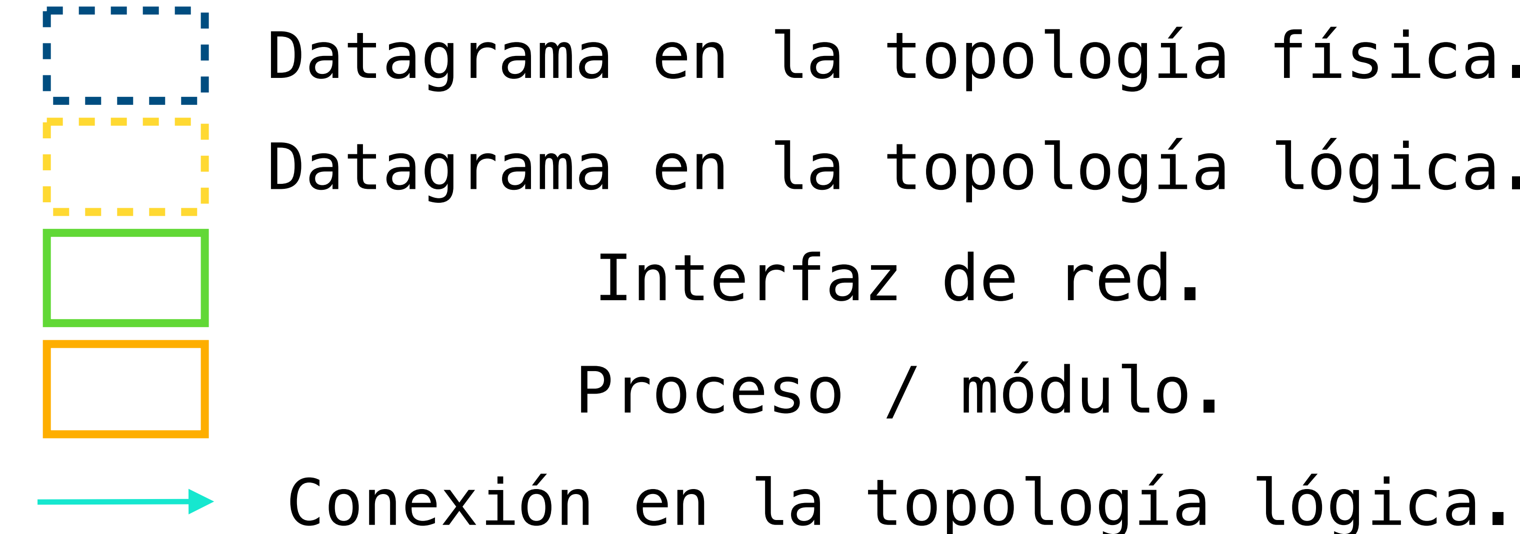
- En este caso solo permitiremos acceso al tráfico de la VPN.
- Esto se traduce a una sola regla de `iptables`:

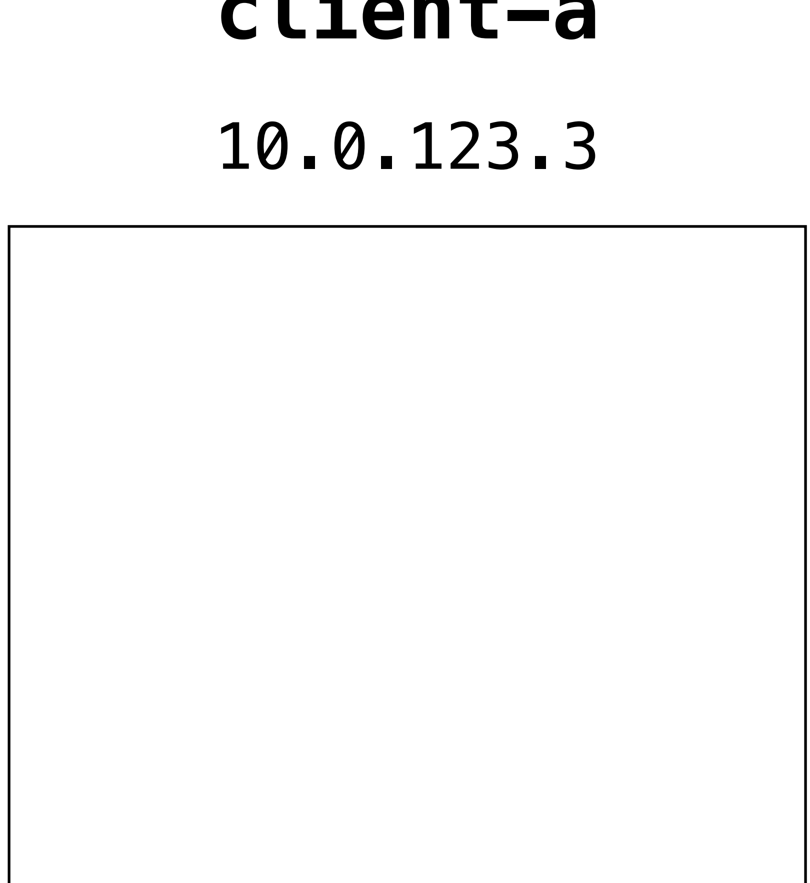
```
iptables -A INPUT -p tcp ! -i wg0 --dport 1234 -j DROP
```

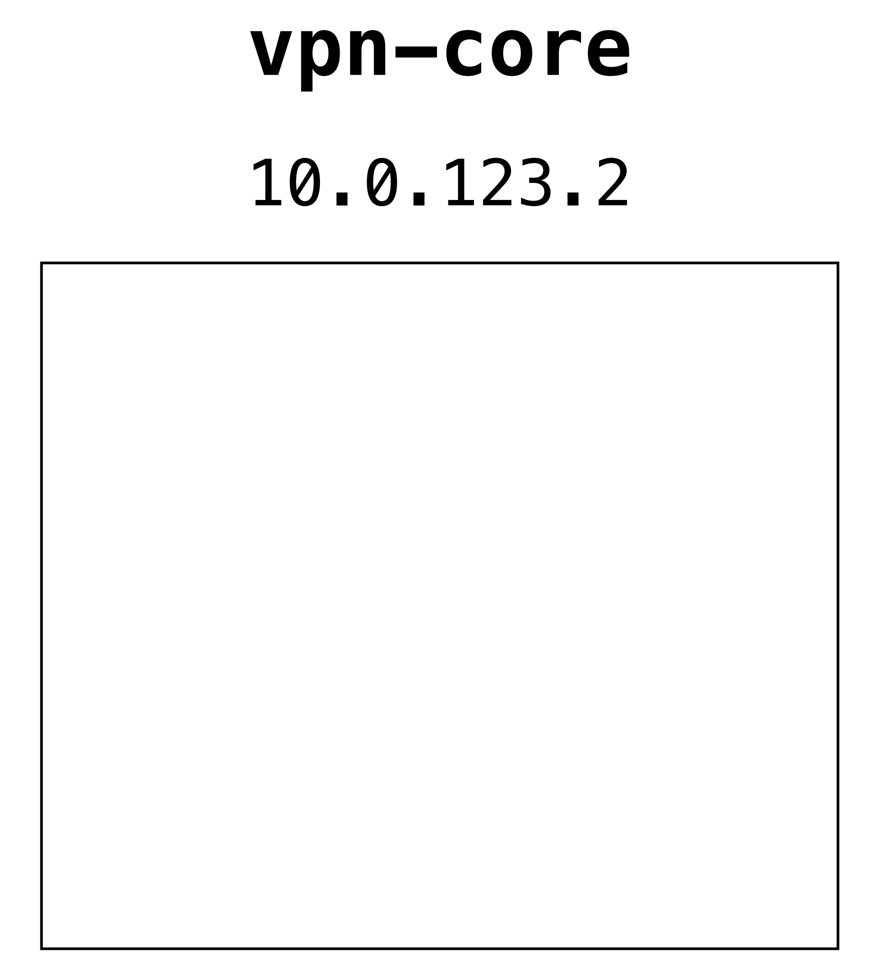
`iptables`	`-A INPUT`	`-p tcp`	`! -i wg0`	`--dport 1234`	`-j DROP`
Ejecutado en client-b	Añadimos la regla a la «cadena» de paquetes dirigidos a nosotros.	La regla solo atañe al protocolo TCP (el que usa nc(1) por defecto).	La regla se aplica a aquellos datagramas cuya interfaz de ingreso **NO** haya sido wg0.	La regla se aplica al puerto **1234.**	Los paquetes que cumplan la regla se **descartan.**

- También podríamos filtrar con `! -s 192.168.4.0/24`: ¡no tan robusto!
- Esta solución requiere que tanto el cliente como servidor sean parte de la VPN…

Yet another day in the life...



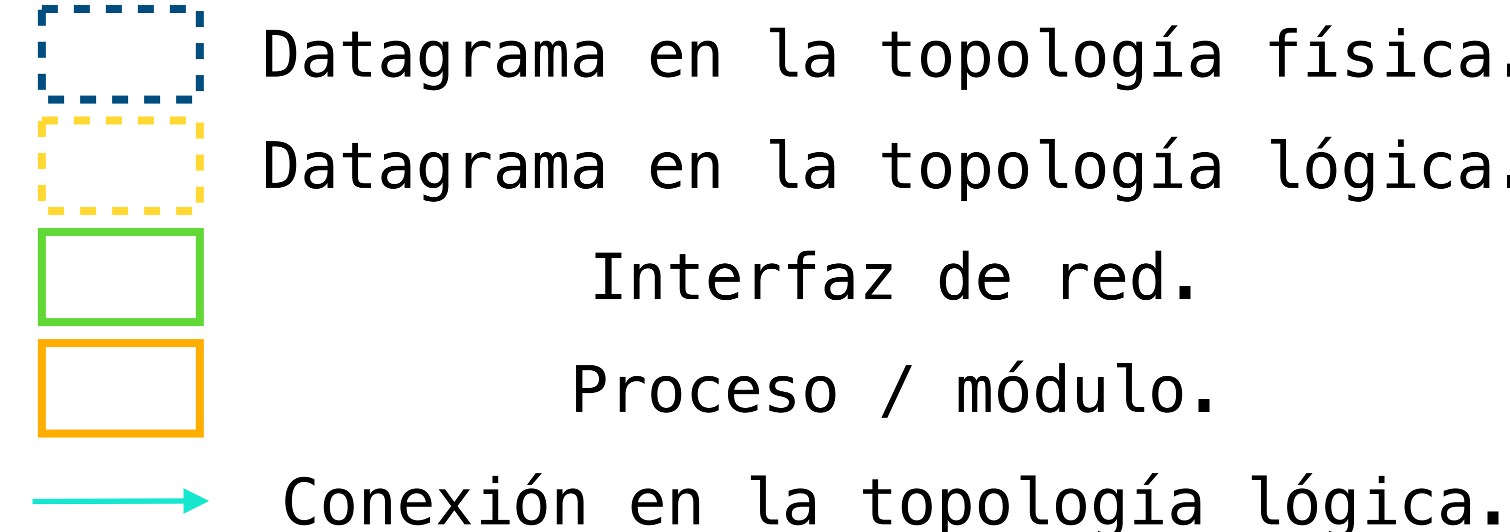




client-b

10.0.123.4

Yet another day in the life...



client-a

10.0.123.3

nc

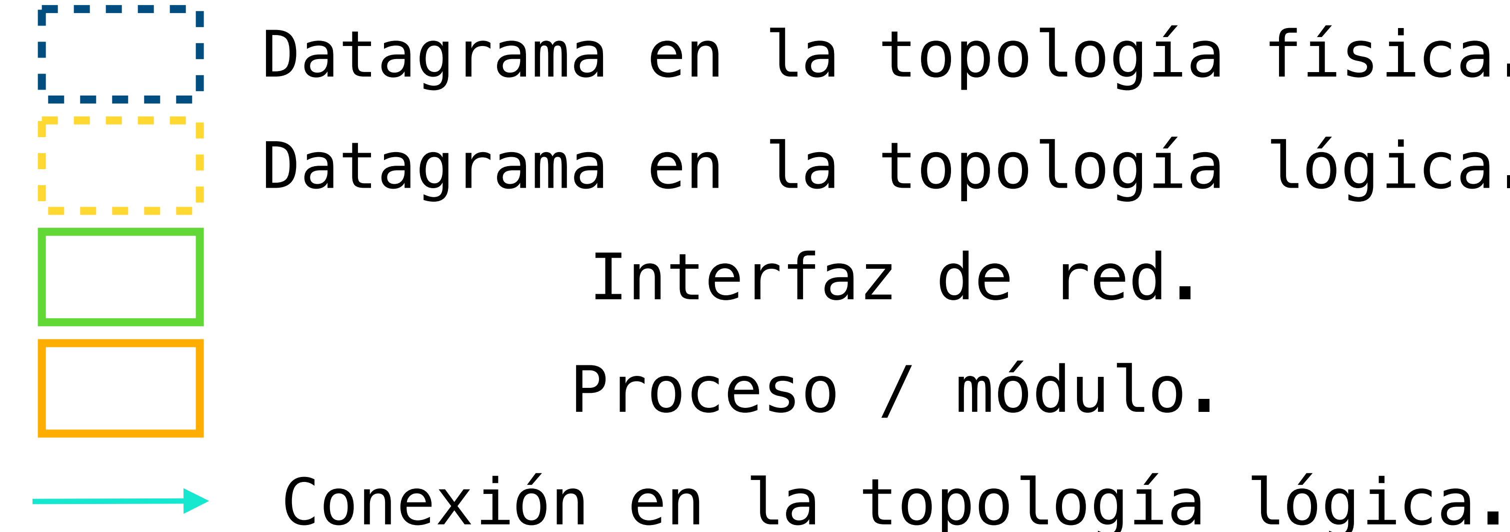
vpn-core

10.0.123.2

client-b

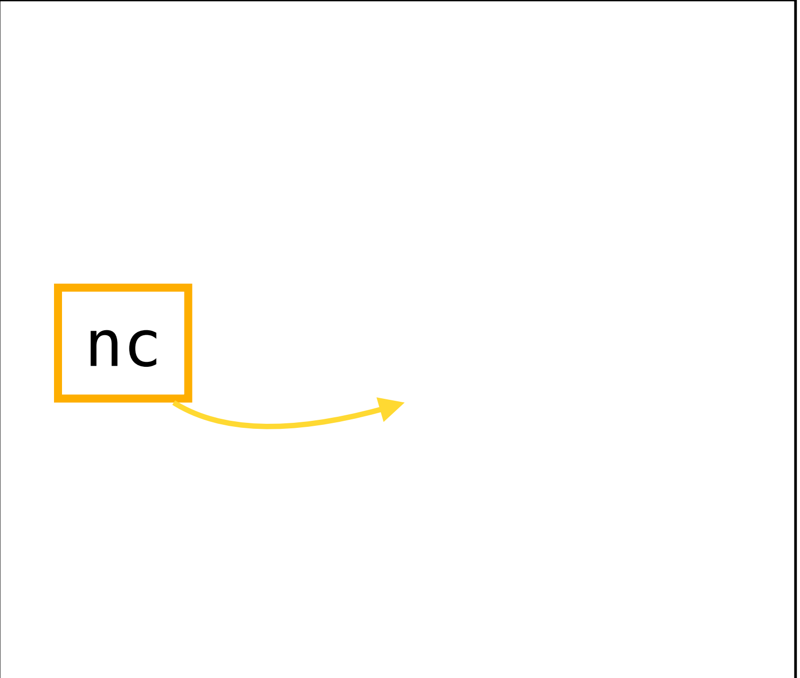
10.0.123.4

Yet another day in the life...



client-a

10.0.123.3



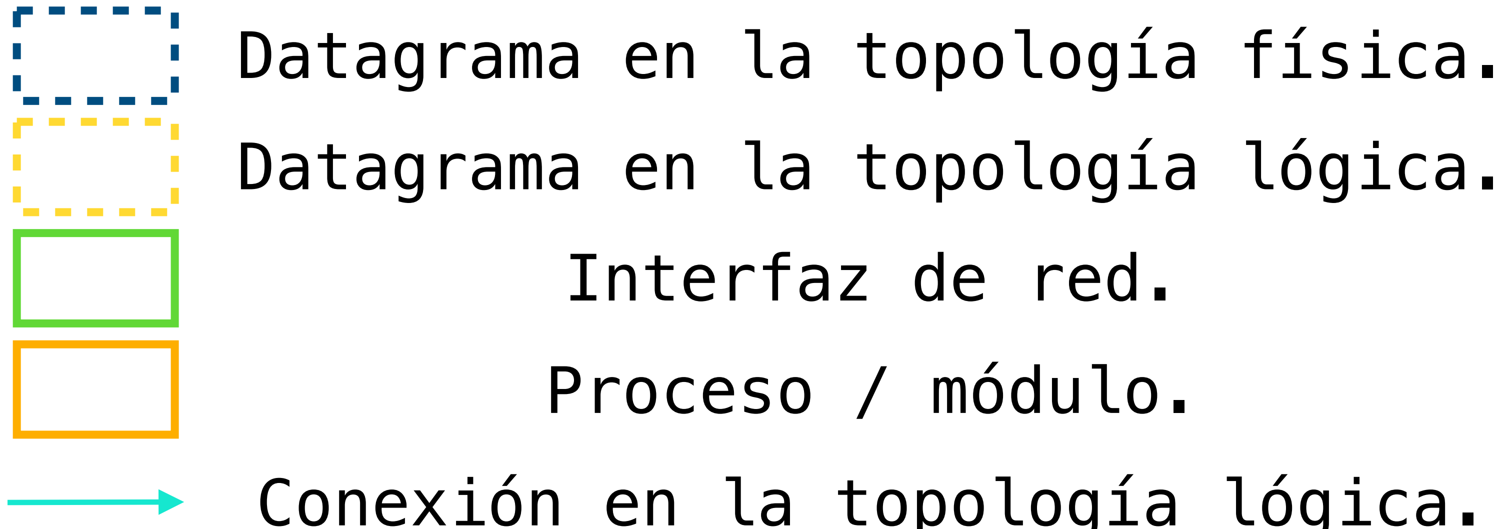
vpn-core

10.0.123.2

client-b

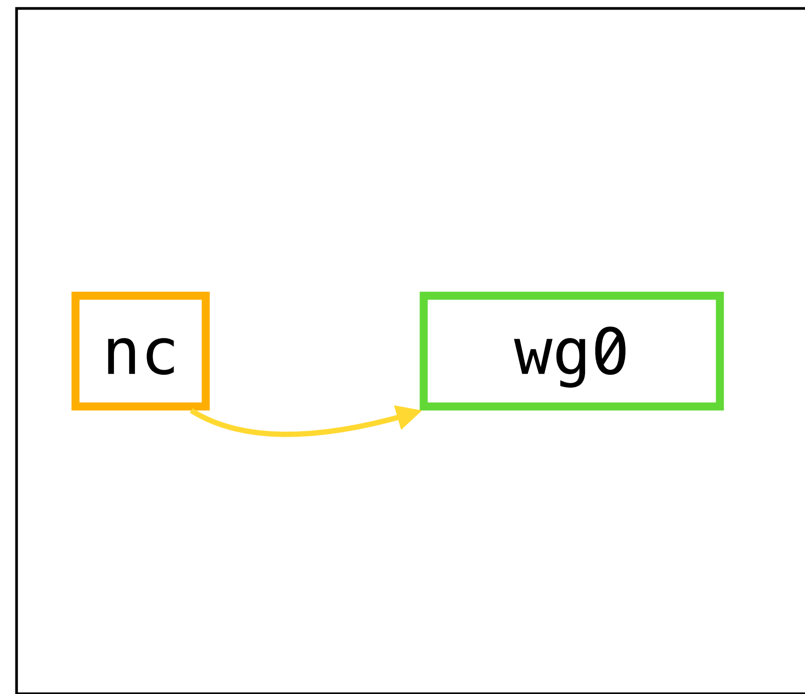
10.0.123.4

Yet another day in the life...



client-a

10.0.123.3



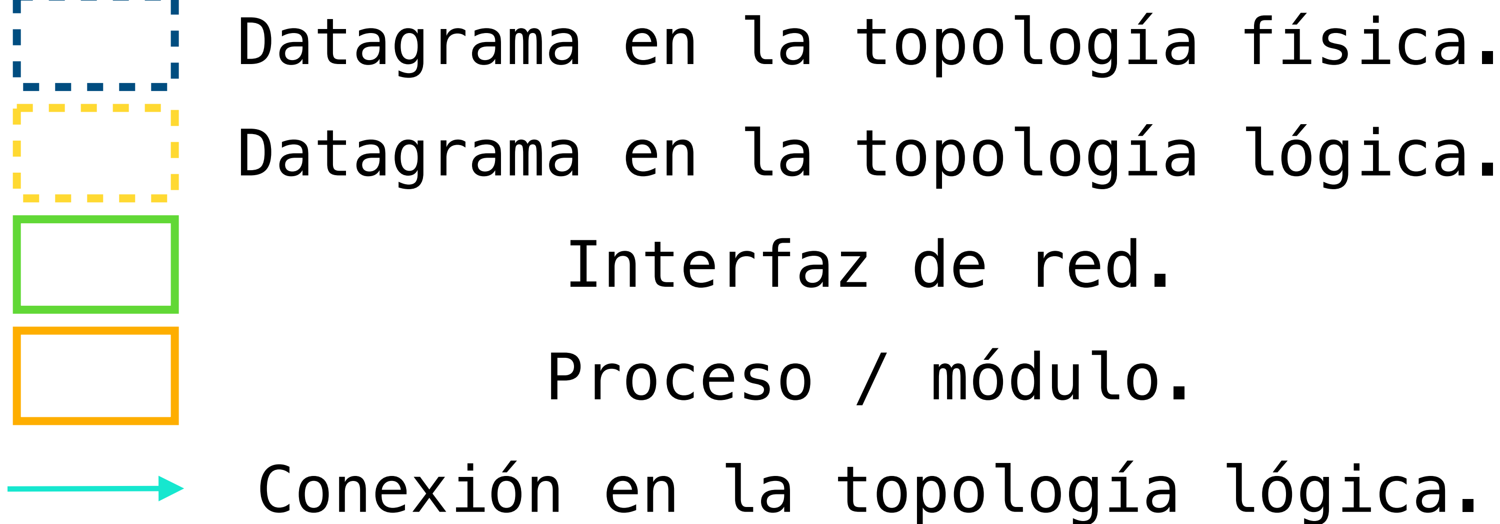
vpn-core

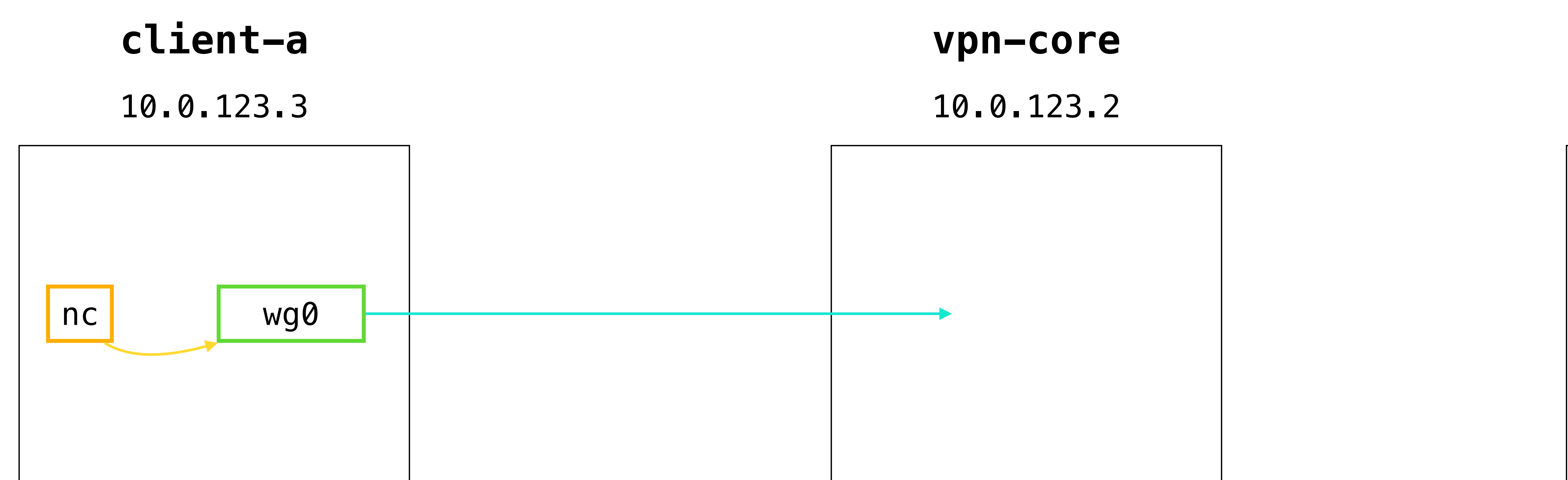
10.0.123.2

client-b

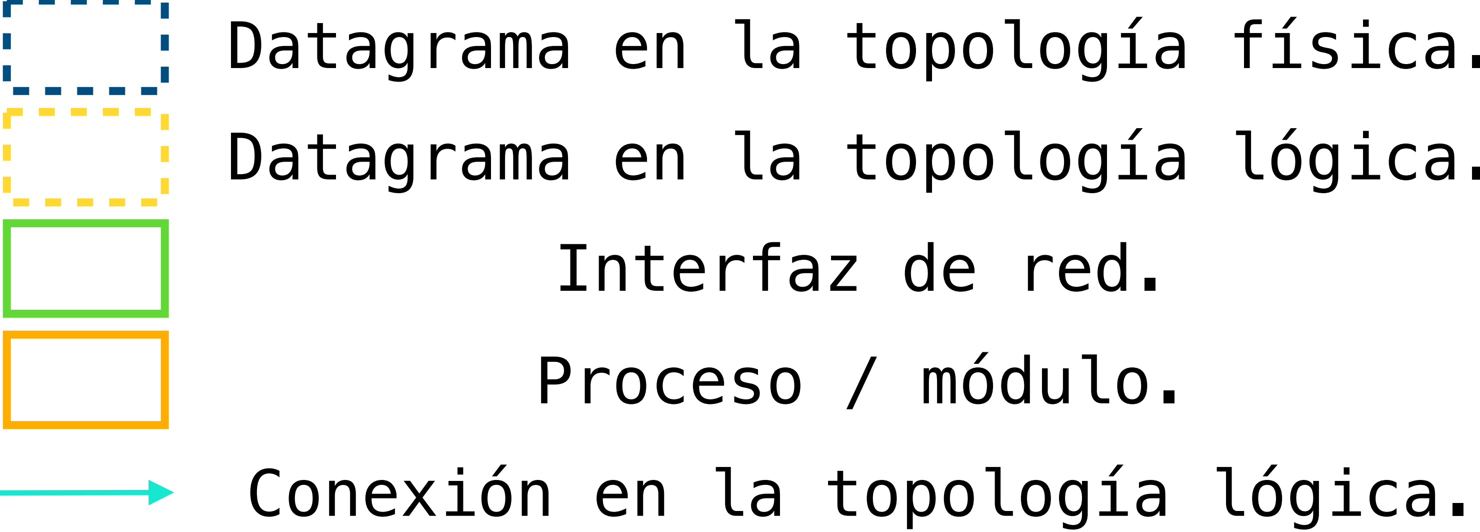
10.0.123.4

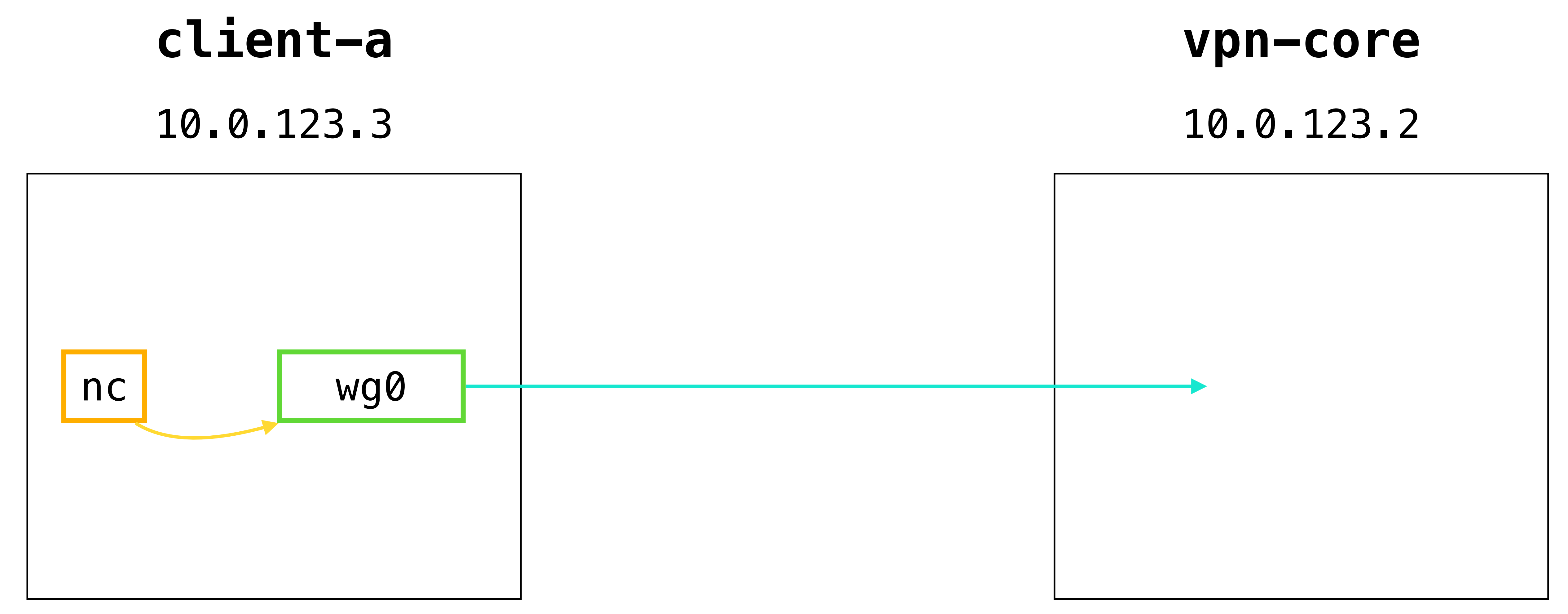
Yet another day in the life...



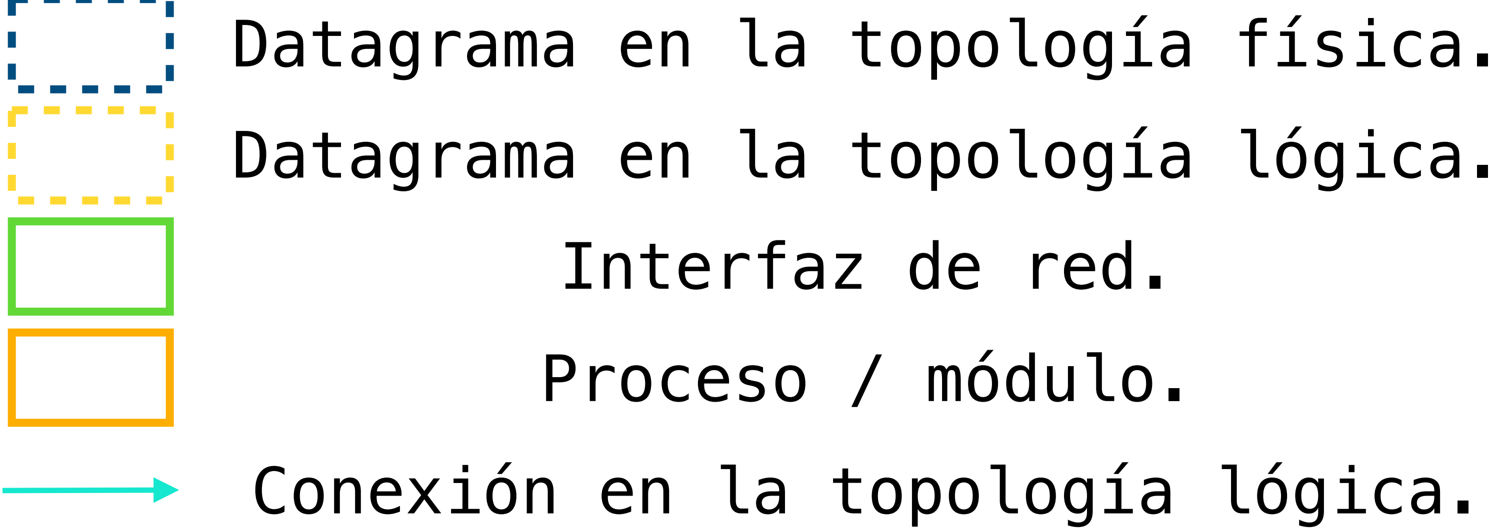


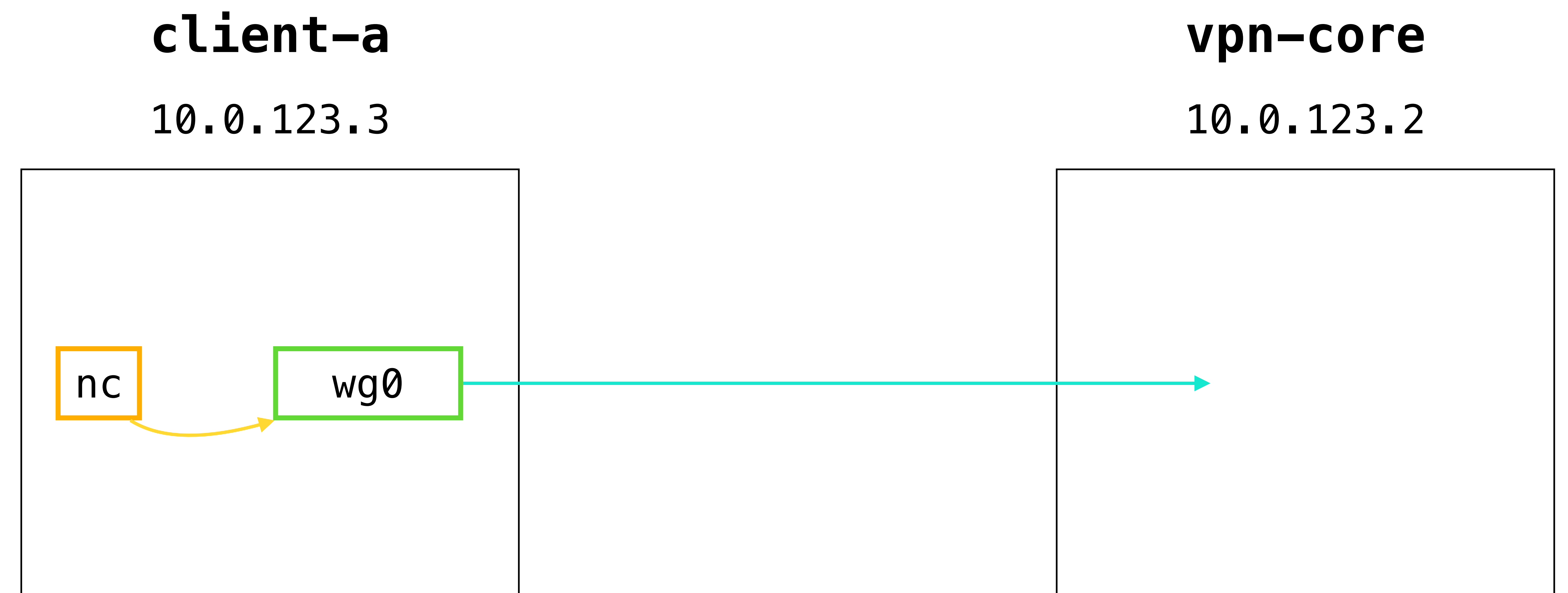
Yet another day in the life...





Yet another day in the life...





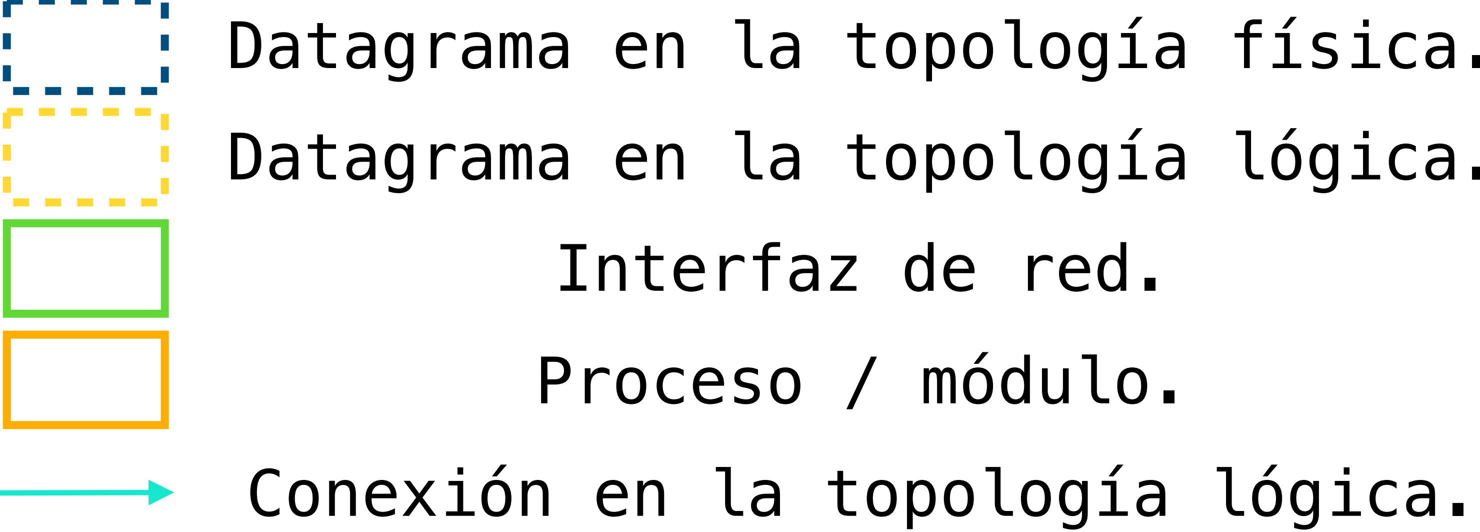
client-b

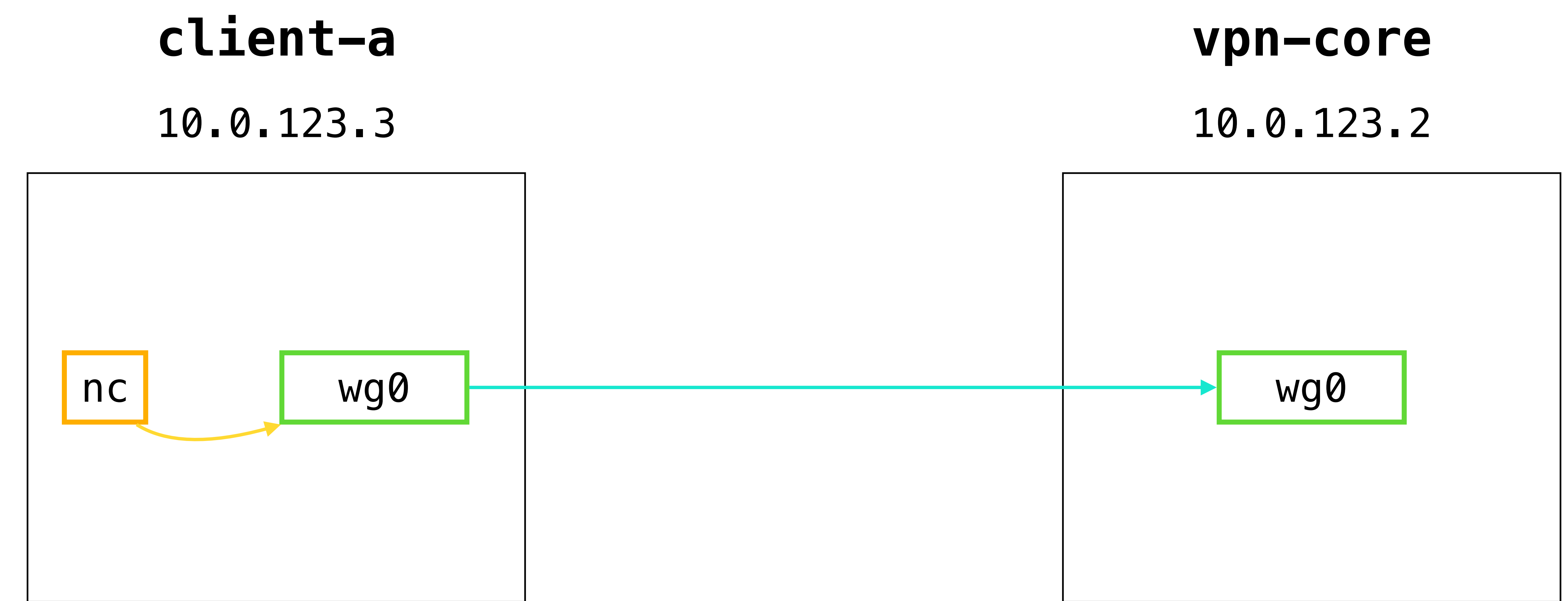
10.0.123.4

From: 192.168.4.2
To: 192.168.4.3

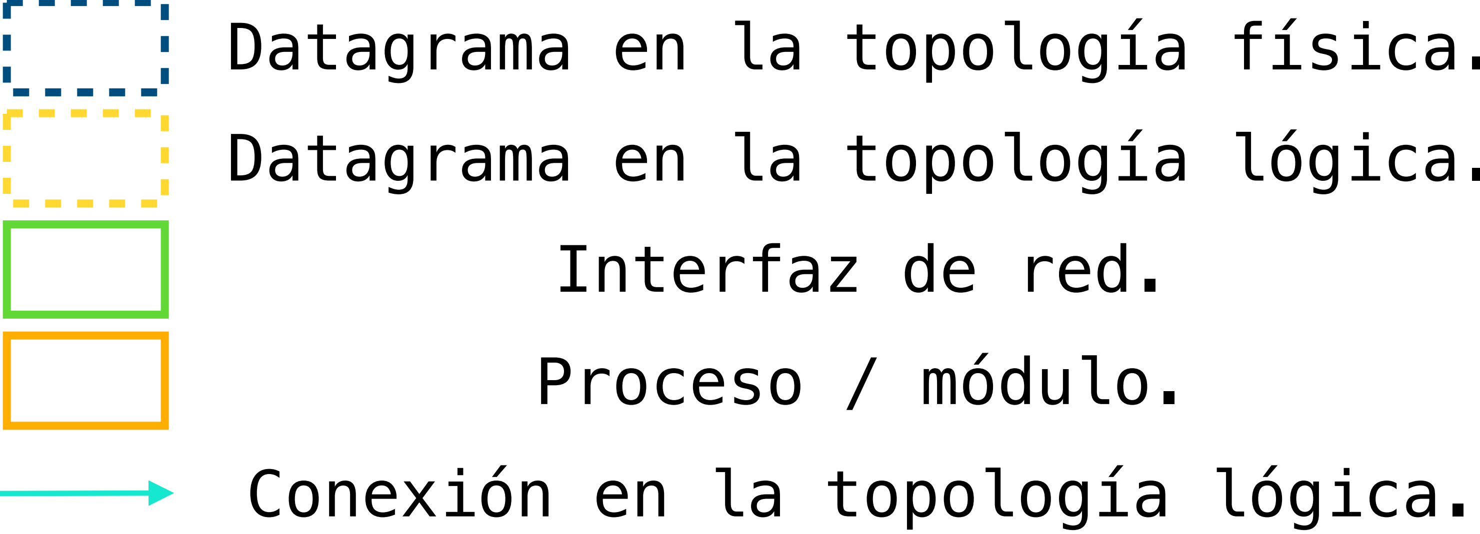
From: 10.0.123.3
To: 10.0.123.2

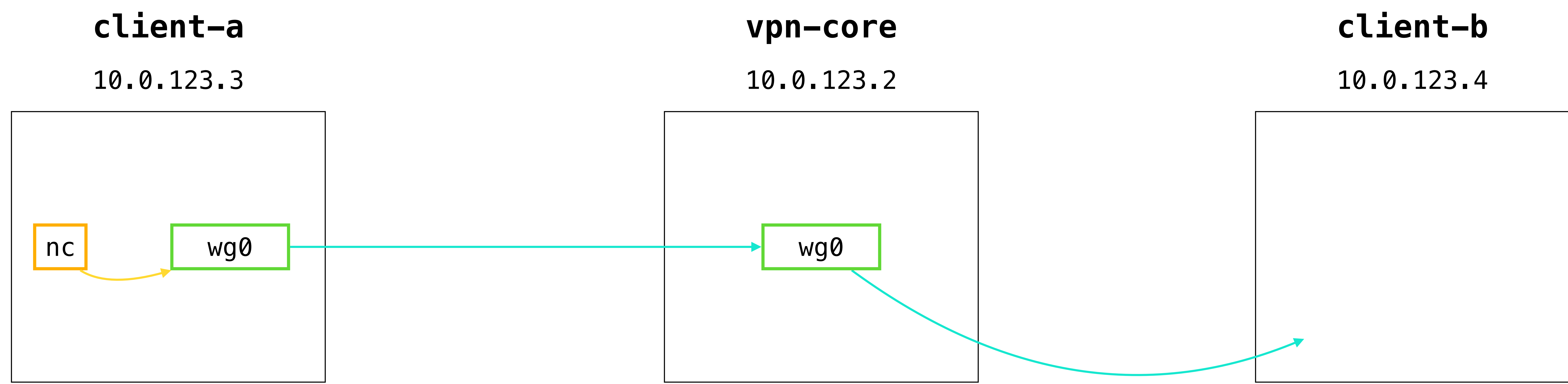
Yet another day in the life...





Yet another day in the life...

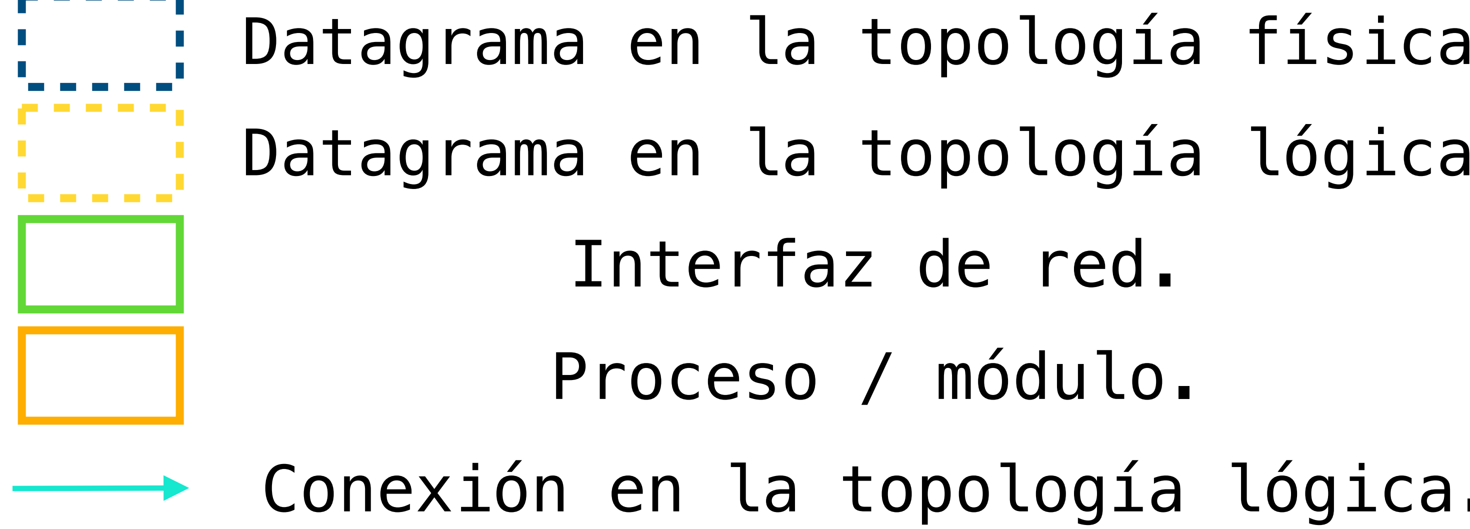




From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.3
To: 10.0.123.2

Yet another day in the life…



client-a

10.0.123.3

nc

wg0

vpn-core

10.0.123.2

wg0

client-b

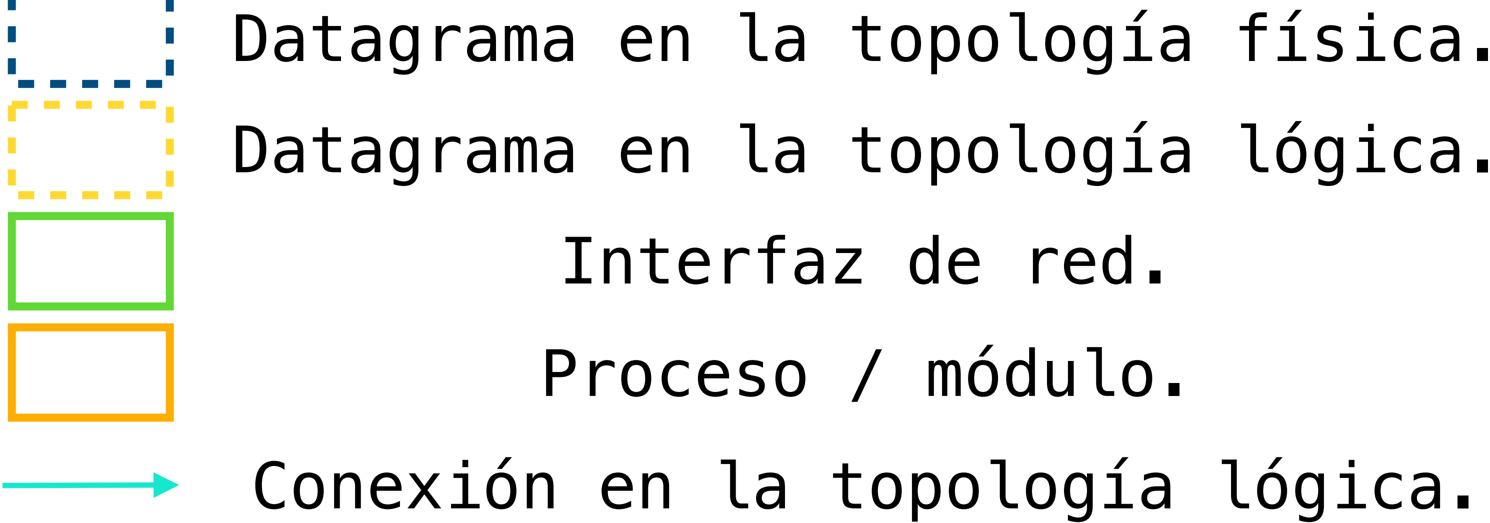
10.0.123.4

From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.3
To: 10.0.123.2

From: 192.168.4.2
To: 192.168.4.3

Yet another day in the life…



client-a
10.0.123.3
nc
wg0

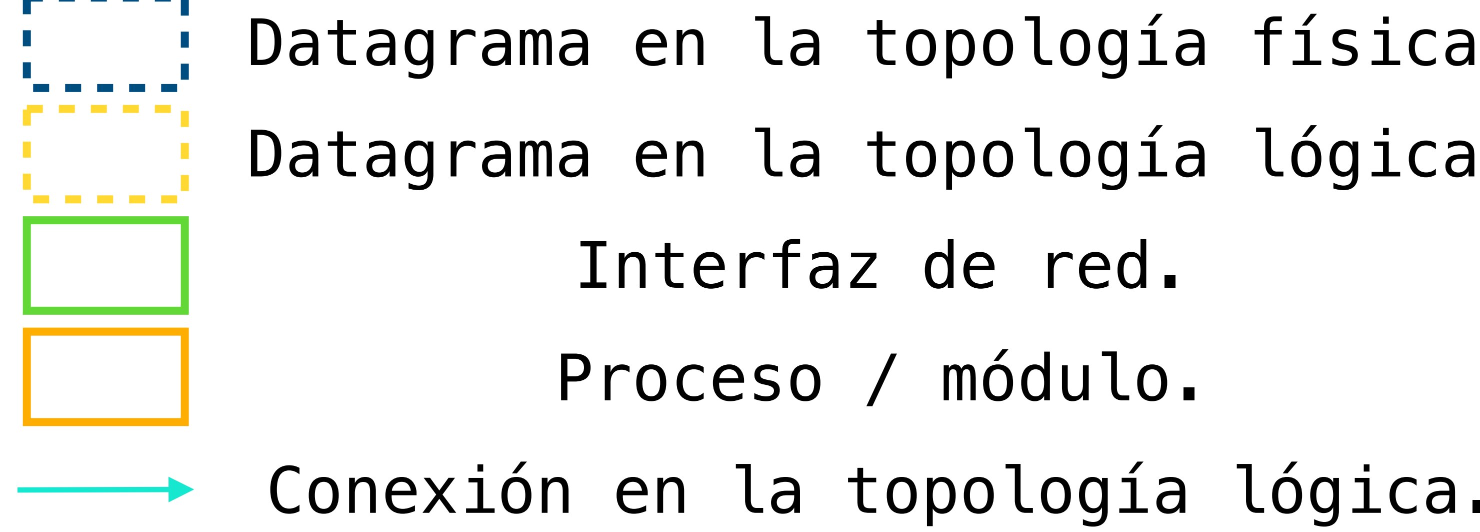
vpn-core
10.0.123.2
wg0

client-b
10.0.123.4

From: 192.168.4.2
To: 192.168.4.3
From: 10.0.123.3
To: 10.0.123.2

From: 192.168.4.2
To: 192.168.4.3
From: 10.0.123.2
To: 10.0.123.4

Yet another day in the life...



client-a

10.0.123.3

nc

wg0

vpn-core

10.0.123.2

wg0

client-b

10.0.123.4

wg0

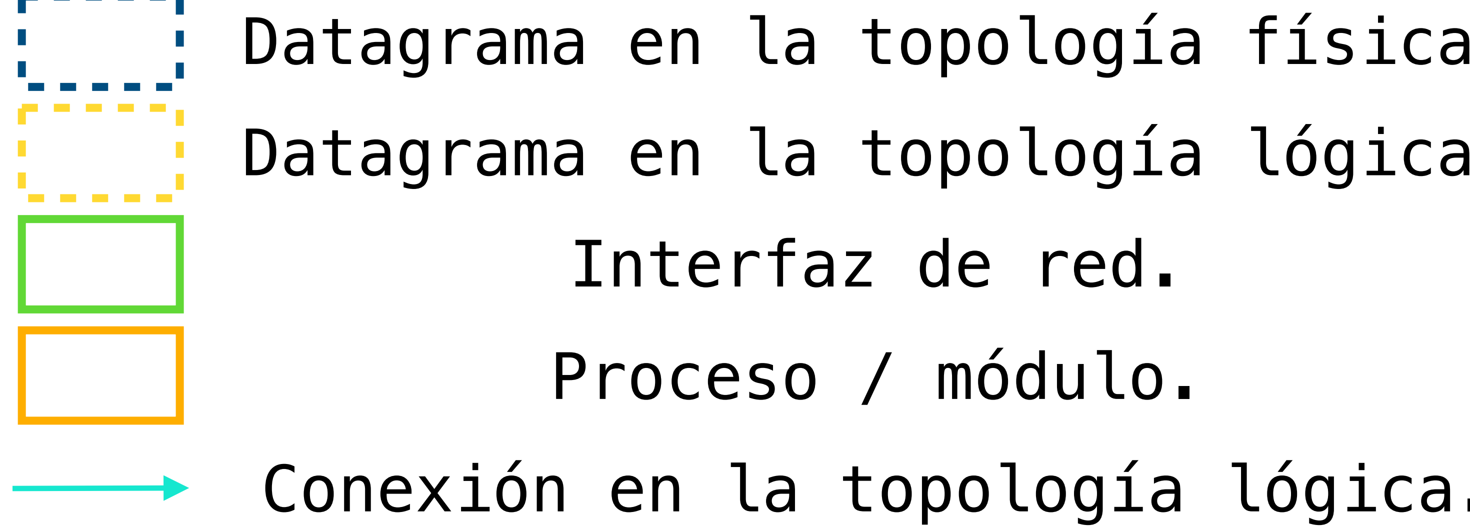
From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.3
To: 10.0.123.2

From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.2
To: 10.0.123.4

Yet another day in the life…



client-a

10.0.123.3

nc

wg0

vpn-core

10.0.123.2

wg0

client-b

10.0.123.4

iptables

wg0

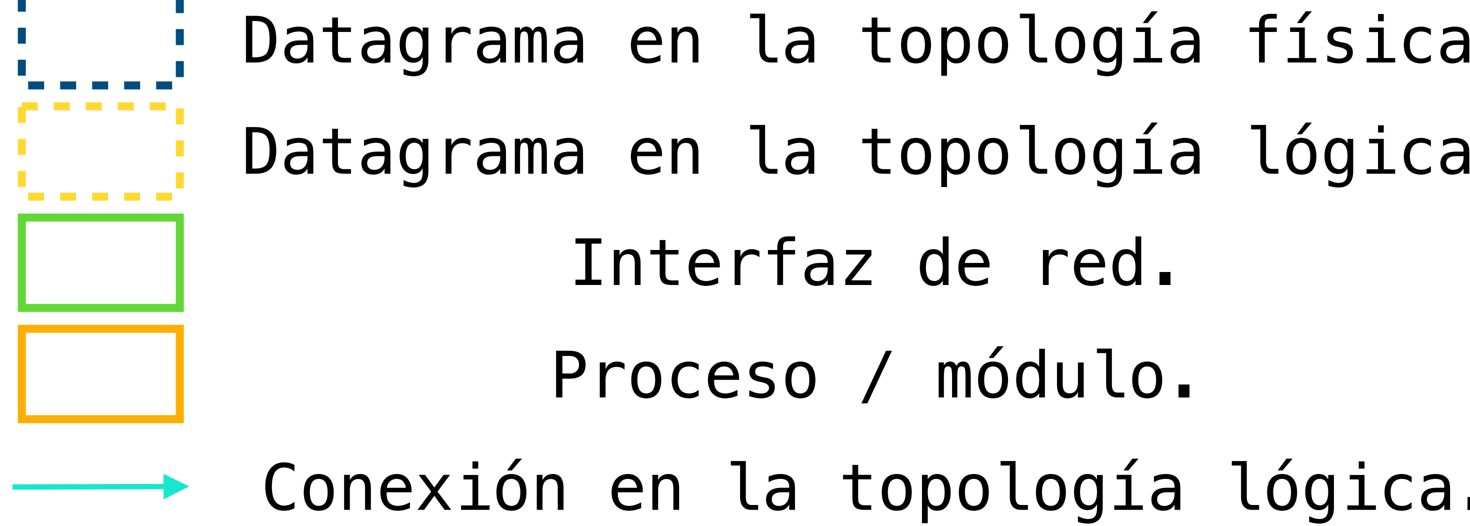
From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.3
To: 10.0.123.2

From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.2
To: 10.0.123.4

Yet another day in the life...



client-a

10.0.123.3

nc

wg0

vpn-core

10.0.123.2

wg0

client-b

10.0.123.4

iptables

wg0

Filtrado

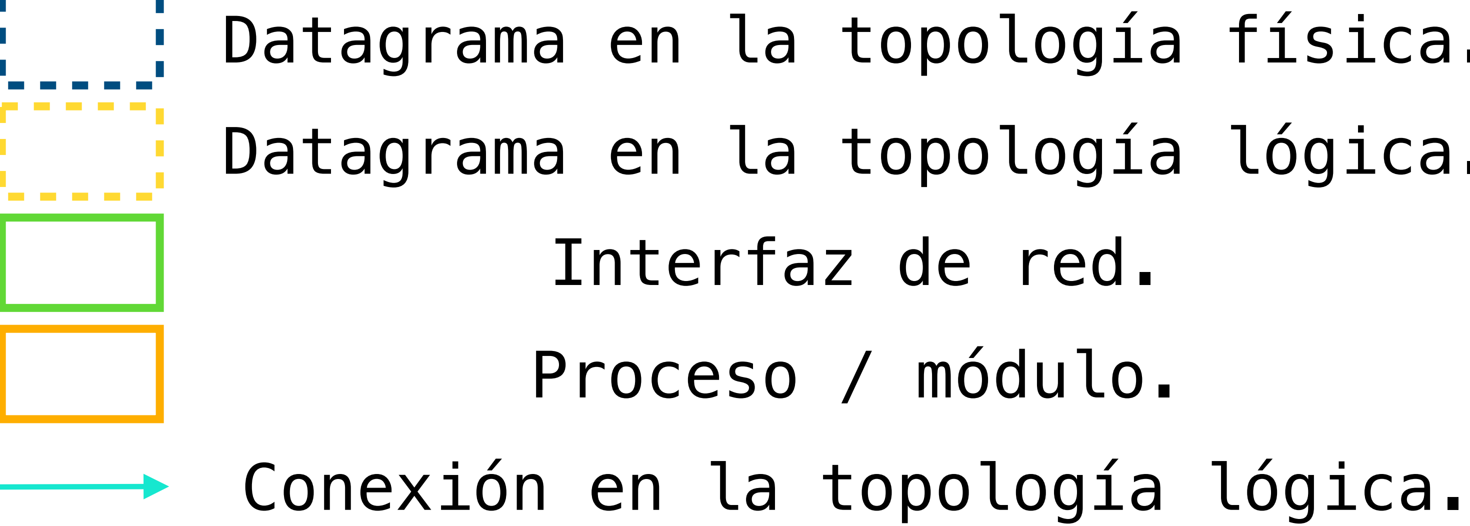
From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.3
To: 10.0.123.2

From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.2
To: 10.0.123.4

Yet another day in the life...



client-a

10.0.123.3

nc

wg0

vpn-core

10.0.123.2

wg0

client-b

10.0.123.4

iptables

wg0

Filtrado

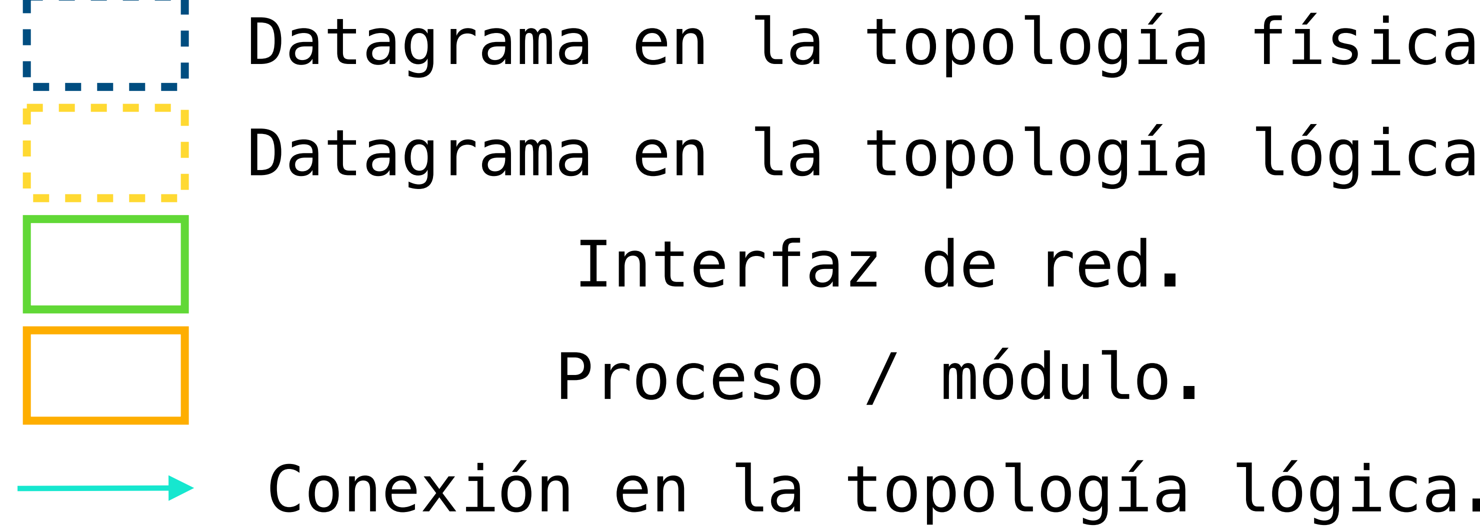
From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.3
To: 10.0.123.2

From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.2
To: 10.0.123.4

Yet another day in the life…



client-a

10.0.123.3

nc

wg0

vpn-core

10.0.123.2

wg0

client-b

10.0.123.4

iptables

wg0

Filtrado

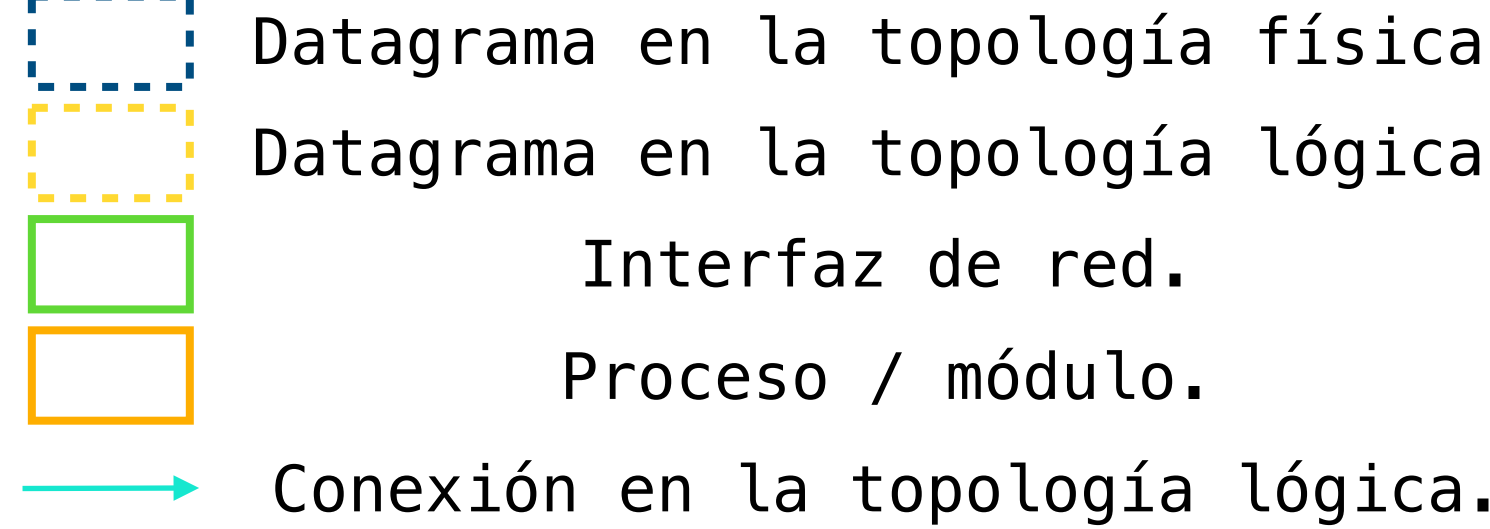
From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.3
To: 10.0.123.2

From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.2
To: 10.0.123.4

Yet another day in the life...



client-a
10.0.123.3
nc
wg0

vpn-core
10.0.123.2
wg0

client-b
10.0.123.4
nc
iptables
wg0
Filtrado

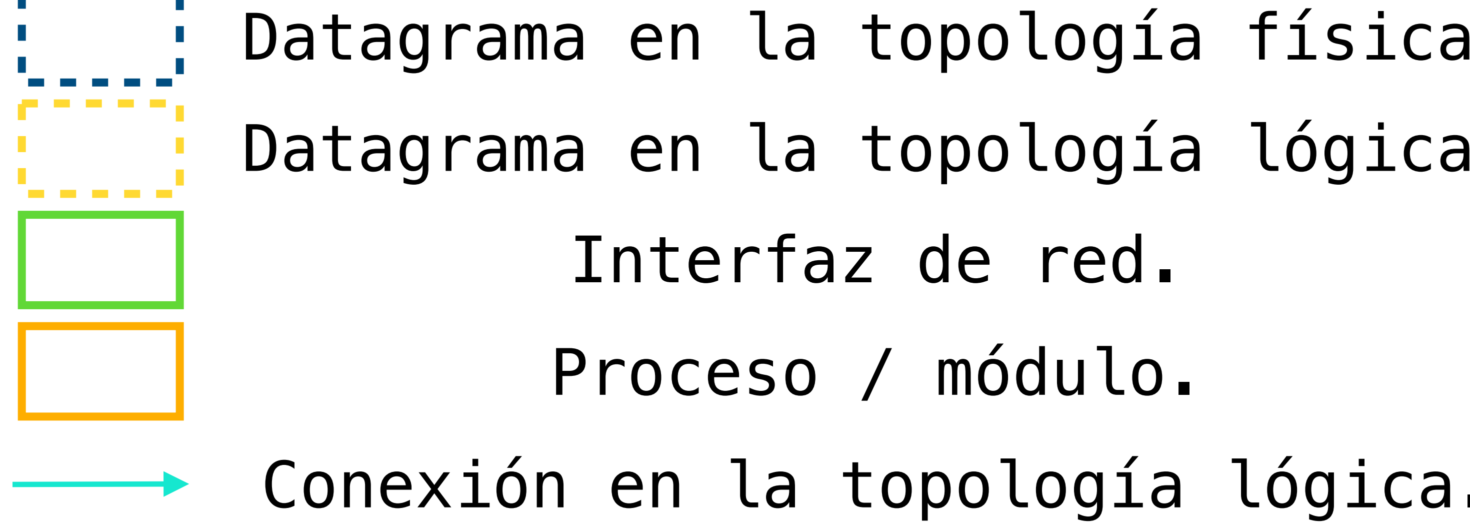
From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.3
To: 10.0.123.2

From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.2
To: 10.0.123.4

Yet another day in the life...



client-a
10.0.123.3
nc
wg0

vpn-core
10.0.123.2
wg0

client-b
10.0.123.4
nc
iptables
wg0
Filtrado

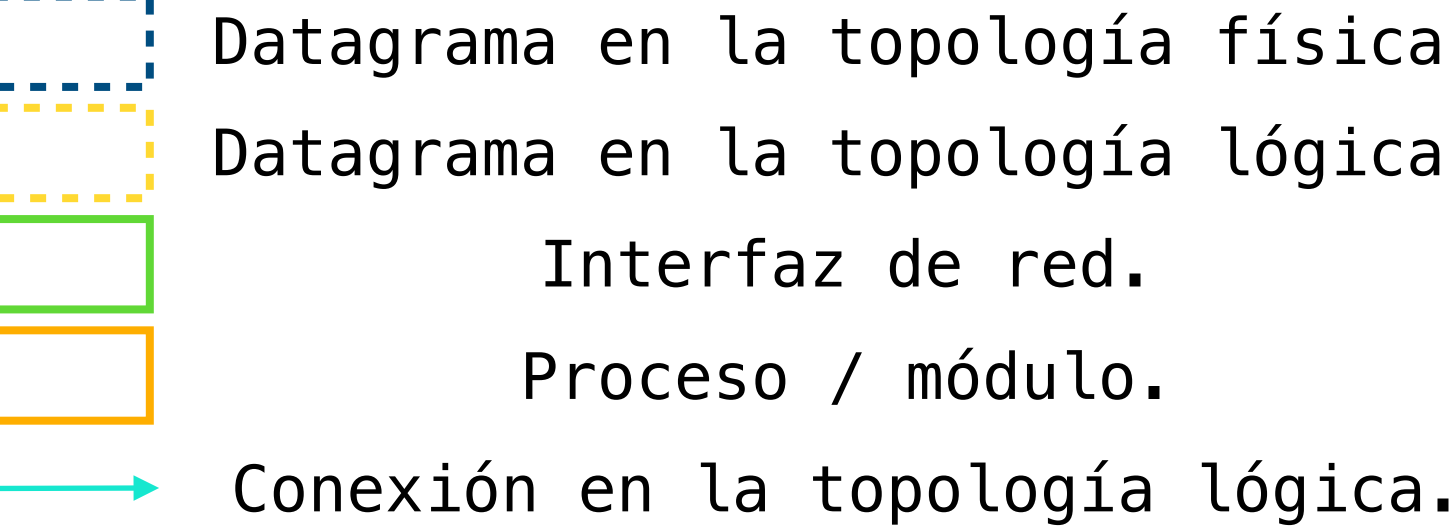
From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.3
To: 10.0.123.2

From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.2
To: 10.0.123.4

Yet another day in the life...



client-a

10.0.123.3

nc

wg0

vpn-core

10.0.123.2

wg0

client-b

10.0.123.4

nc

iptables

wg0

Filtrado

From: 192.168.4.2
To: 192.168.4.3

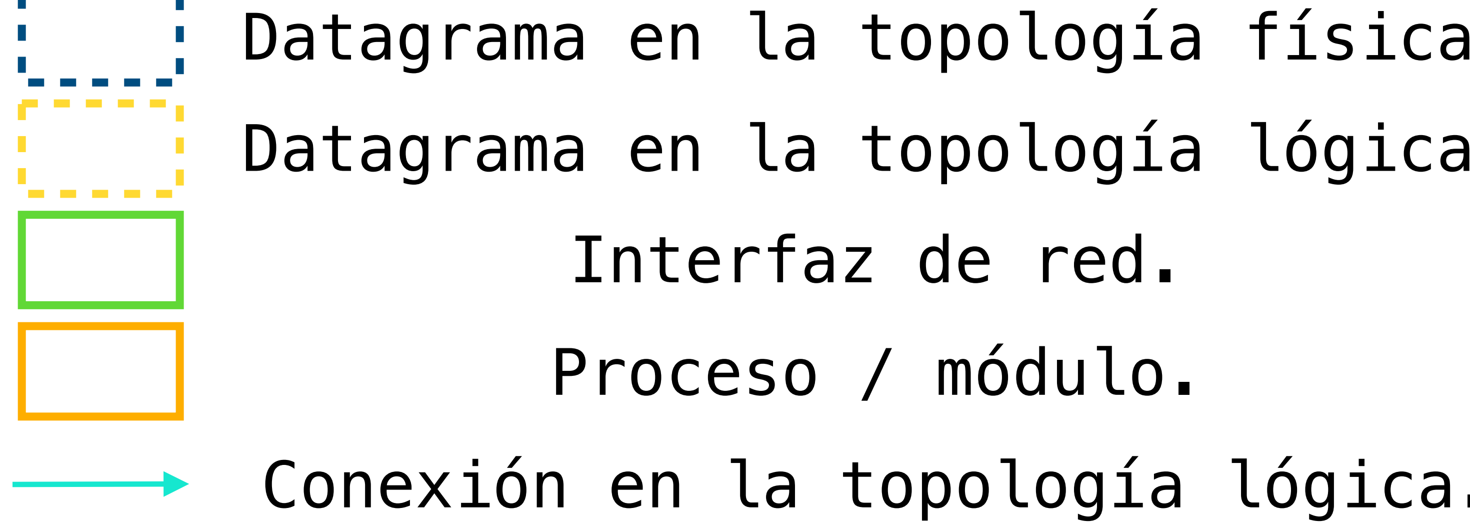
From: 10.0.123.3
To: 10.0.123.2

El tráfico
capturado aquí
aparece
«duplicado»

From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.2
To: 10.0.123.4

Yet another day in the life…



client-a

10.0.123.3

nc

wg0

vpn-core

10.0.123.2

wg0

client-b

10.0.123.4

nc

iptables

wg0

Filtrado

From: 192.168.4.2
To: 192.168.4.3

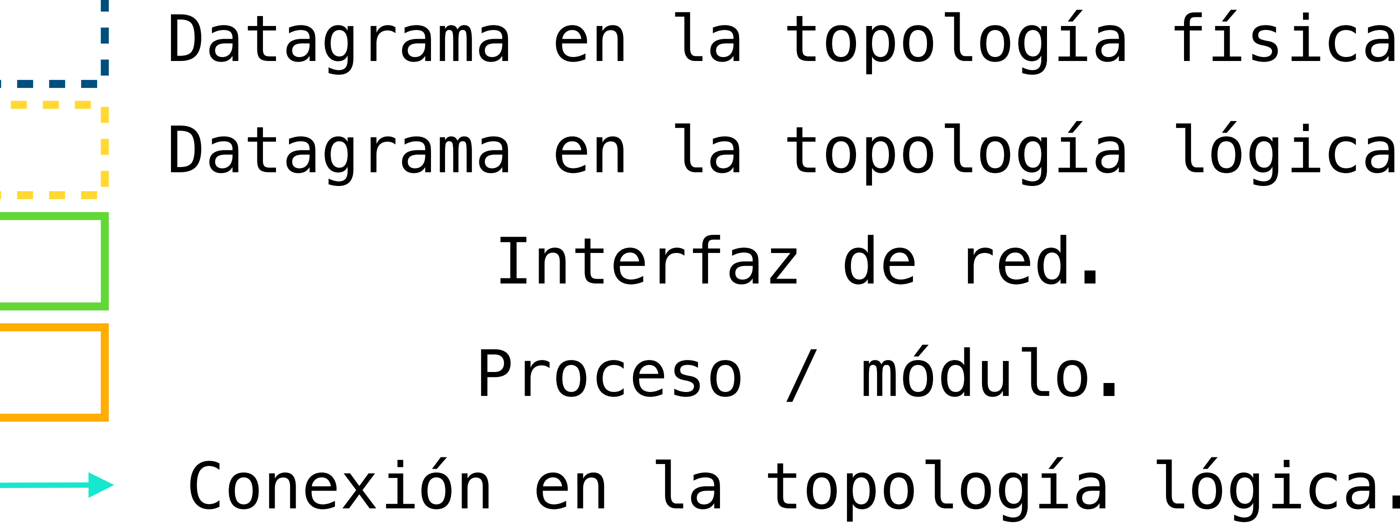
From: 10.0.123.3
To: 10.0.123.2

El tráfico
capturado aquí
aparece
«duplicado»

From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.2
To: 10.0.123.4

Yet another day in the life...



client-a

10.0.123.3

nc

wg0

vpn-core

10.0.123.2

wg0

client-b

10.0.123.4

nc

iptables

wg0

Filtrado

From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.3
To: 10.0.123.2

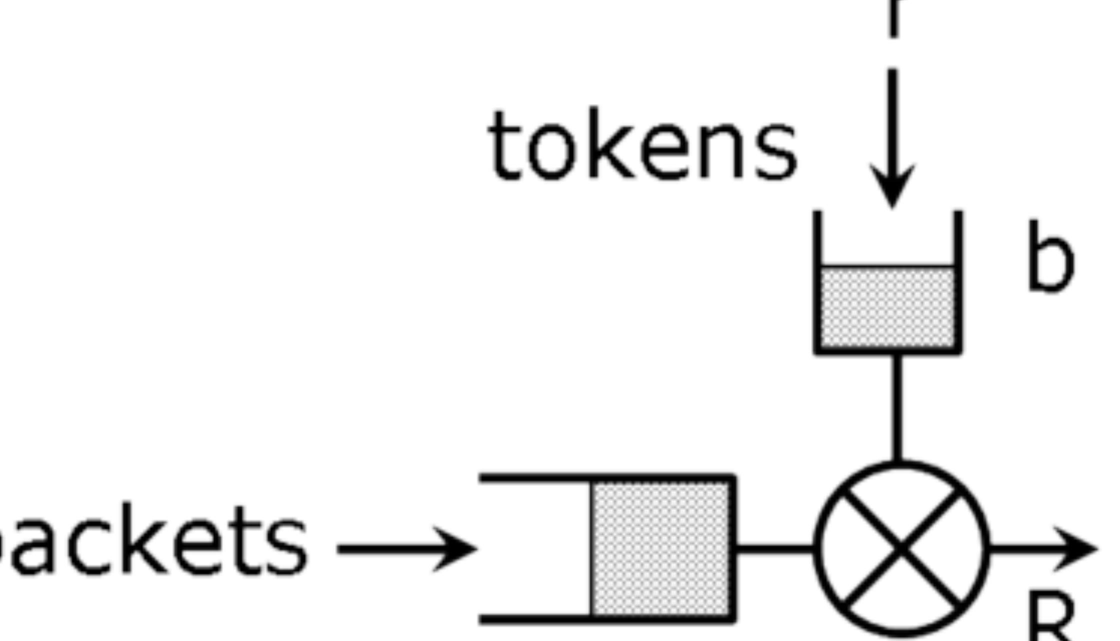
El tráfico capturado aquí aparece «duplicado»

From: 192.168.4.2
To: 192.168.4.3

From: 10.0.123.2
To: 10.0.123.4

La captura se hace con:
tcpdump -i wg0 [-w foo.pcap]

Aplicando técnicas QoS



Fuente

- Nuestro laboratorio ofrece servicio de almacenamiento.
- Muchos de los archivos son tremendamente pesados…
- Para garantizar un buen rendimiento podemos aplicar técnicas QoS.
- Linux tiene implementados gran cantidad de algoritmos QoS.
- Haciendo uso de `tc(8)` podemos conformar y controlar el tráfico:
  - Esta herramienta es parte de la suite `iproute2`.
  - Hay documentación excelente en LARTC.

Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.

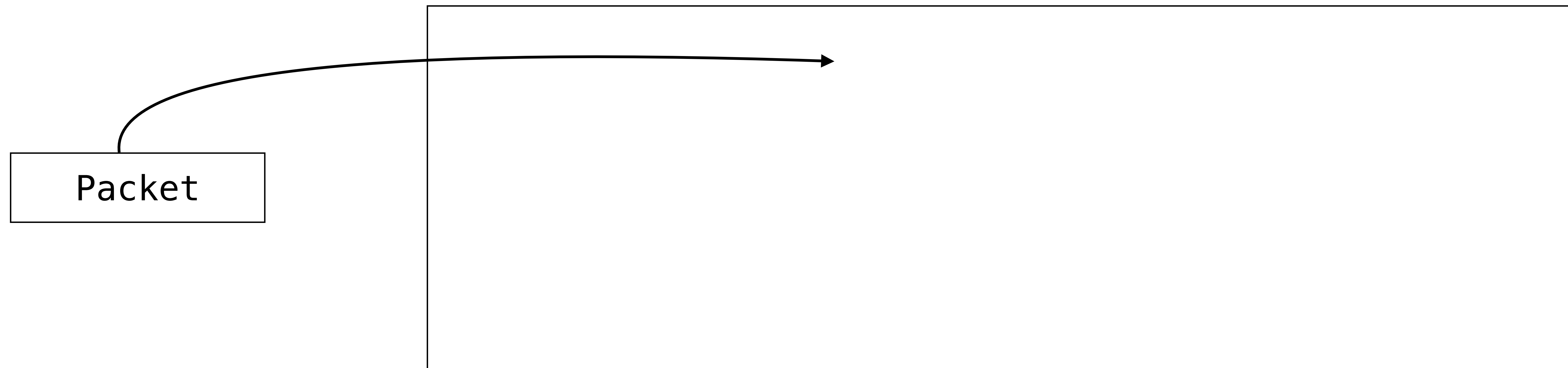
`Traffic Control for an Interface`

Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.

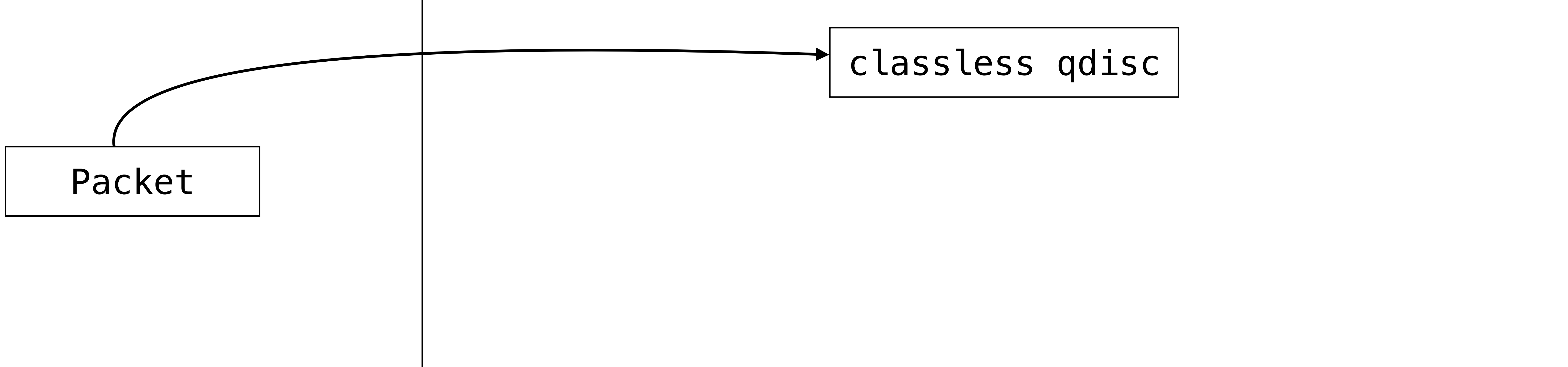
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



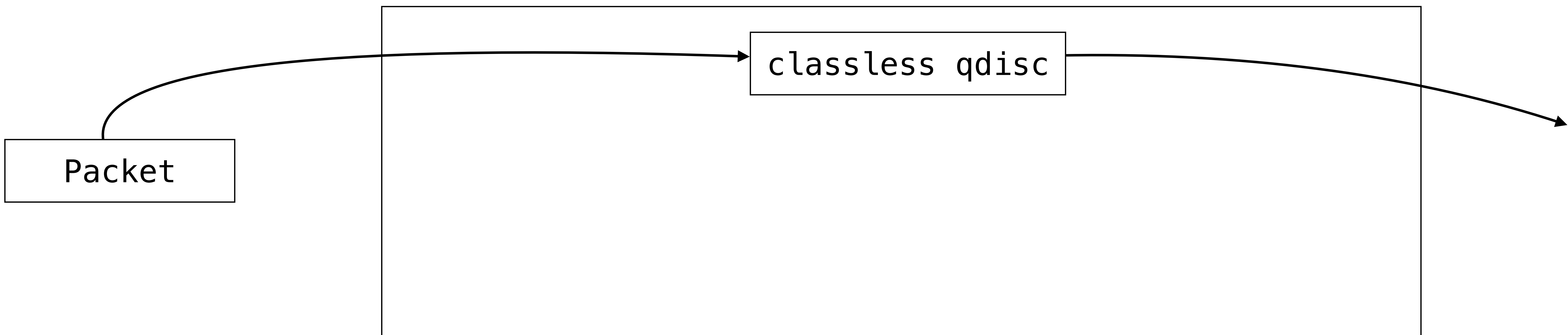
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



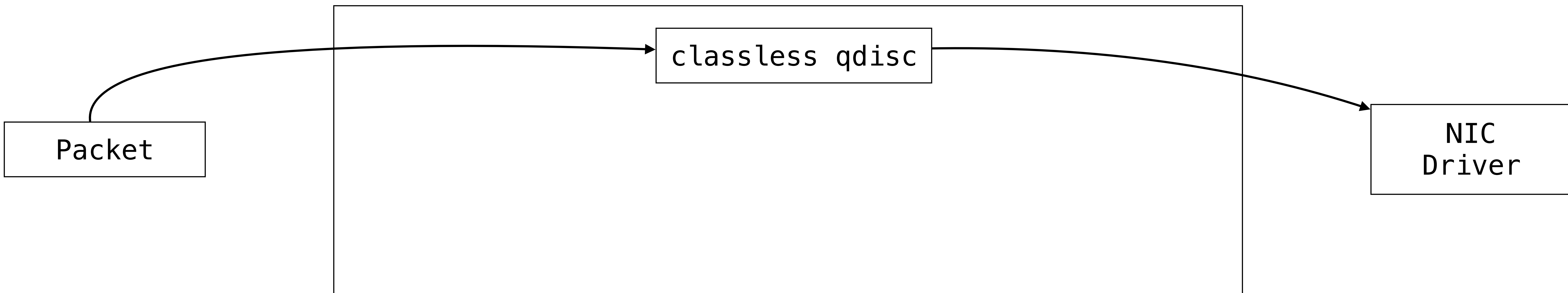
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



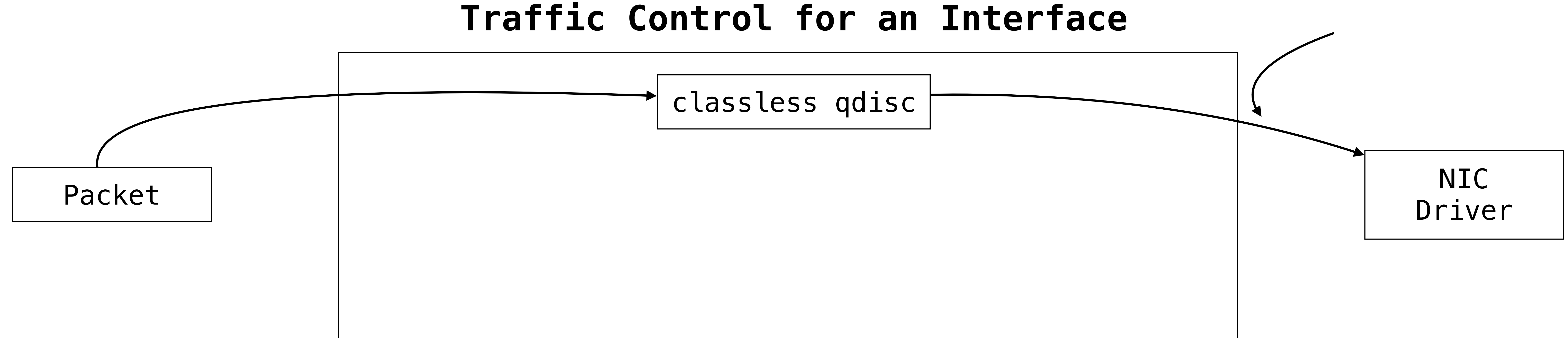
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



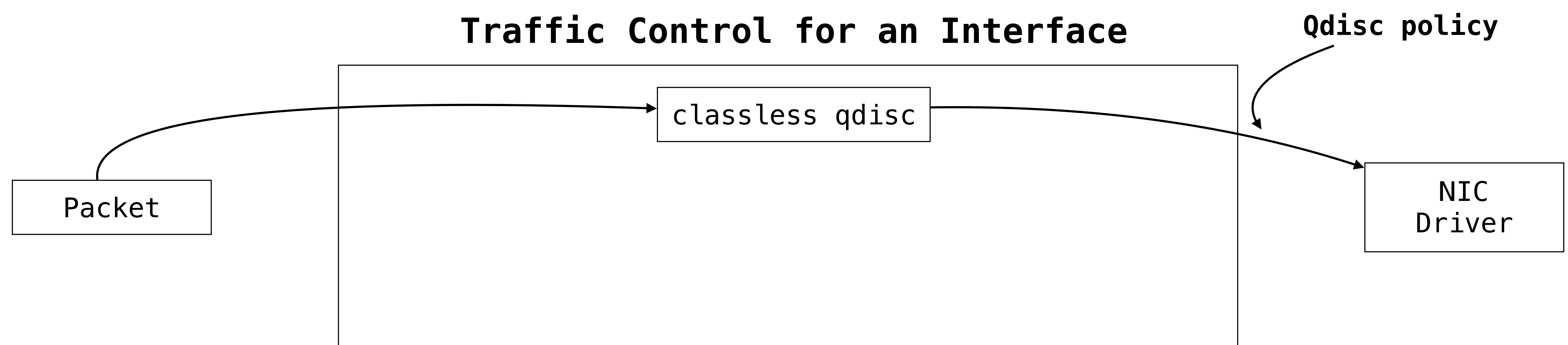
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



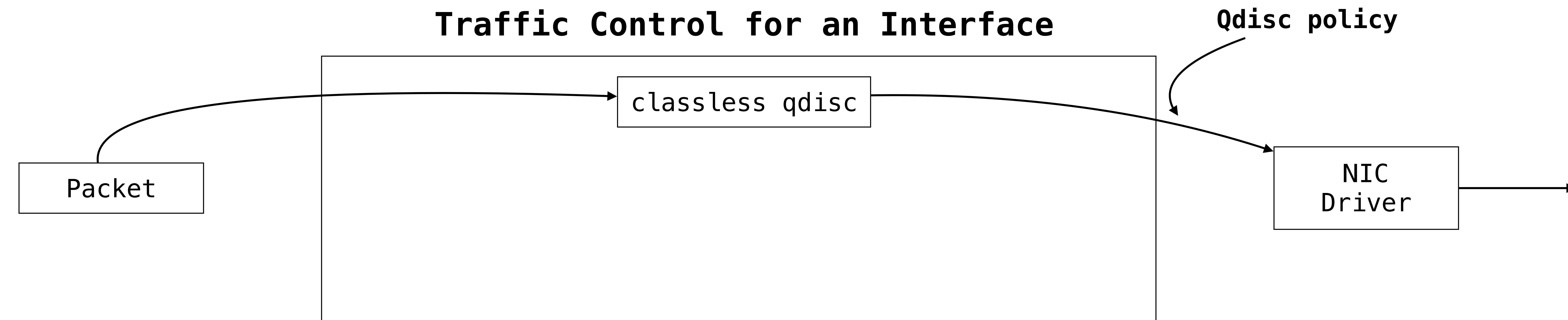
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



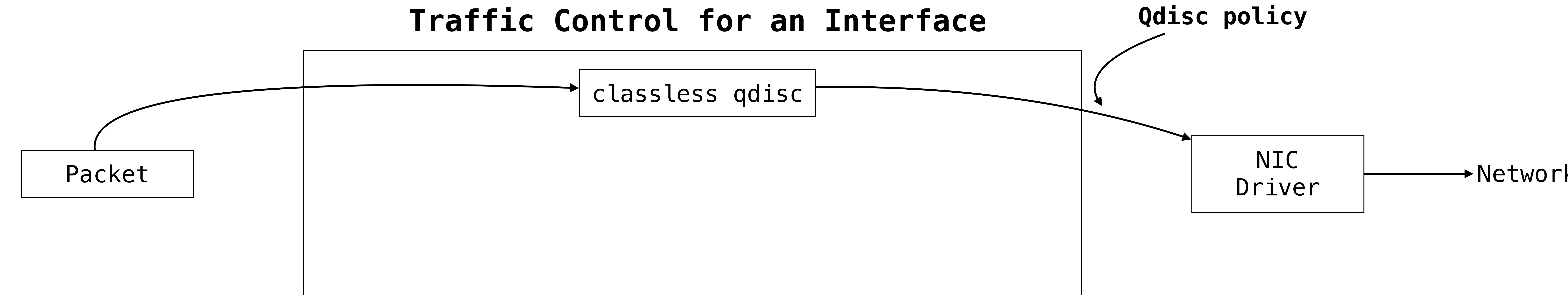
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



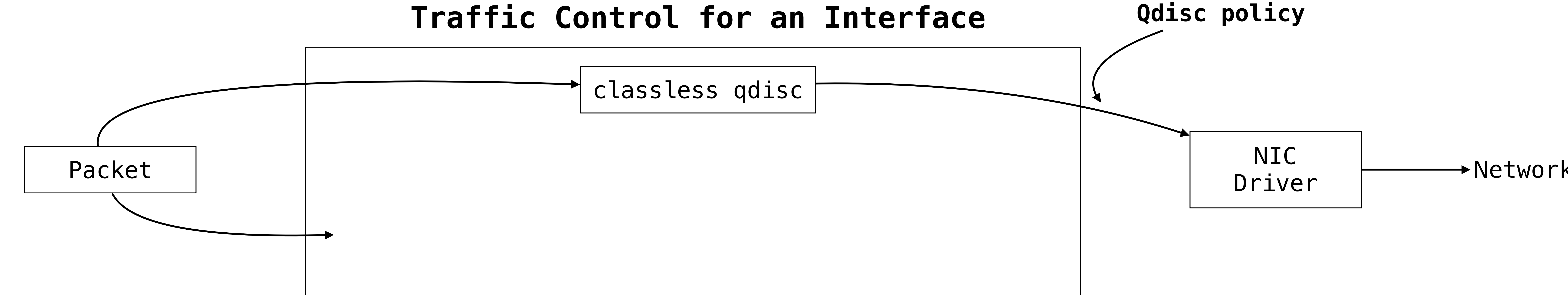
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



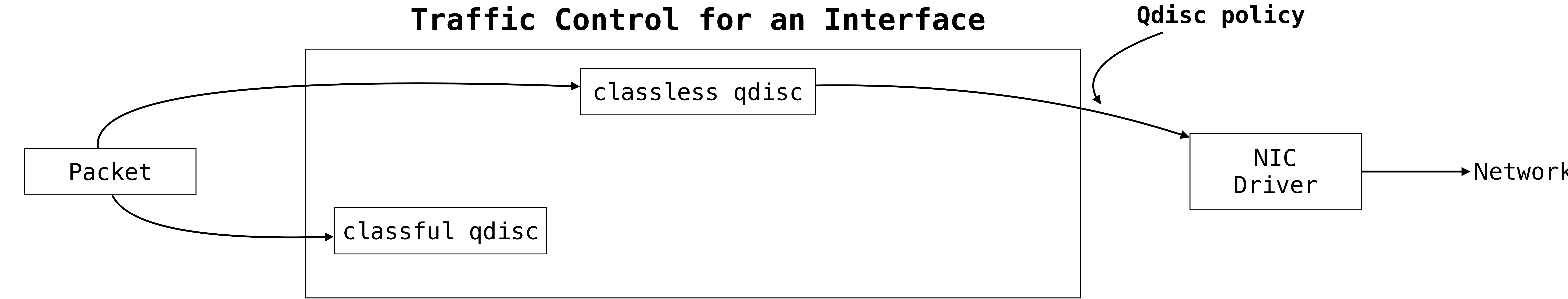
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



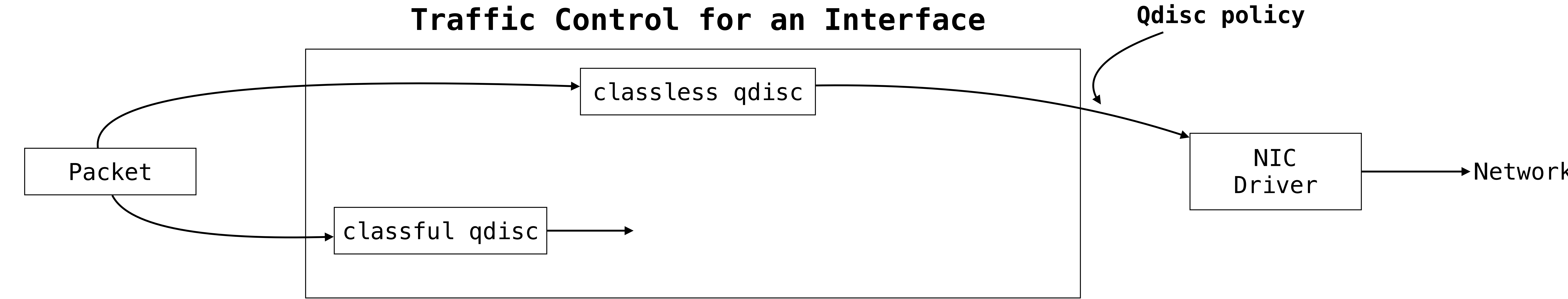
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



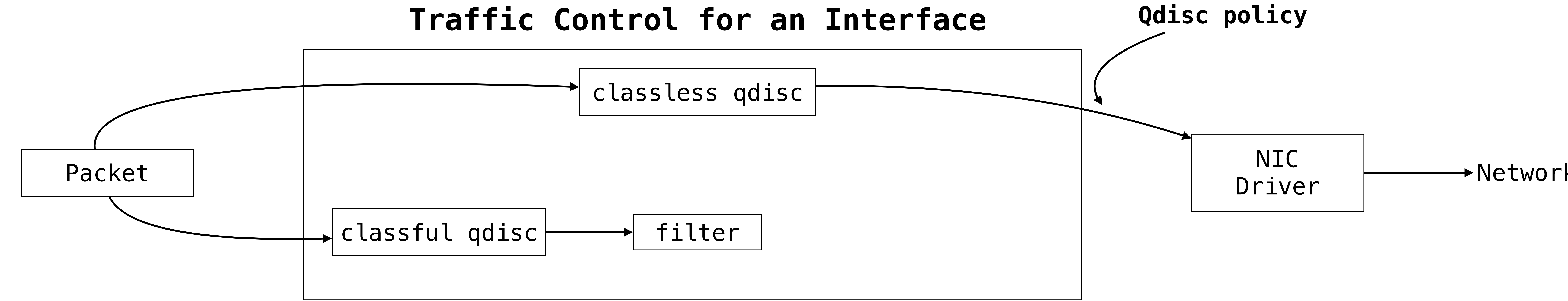
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



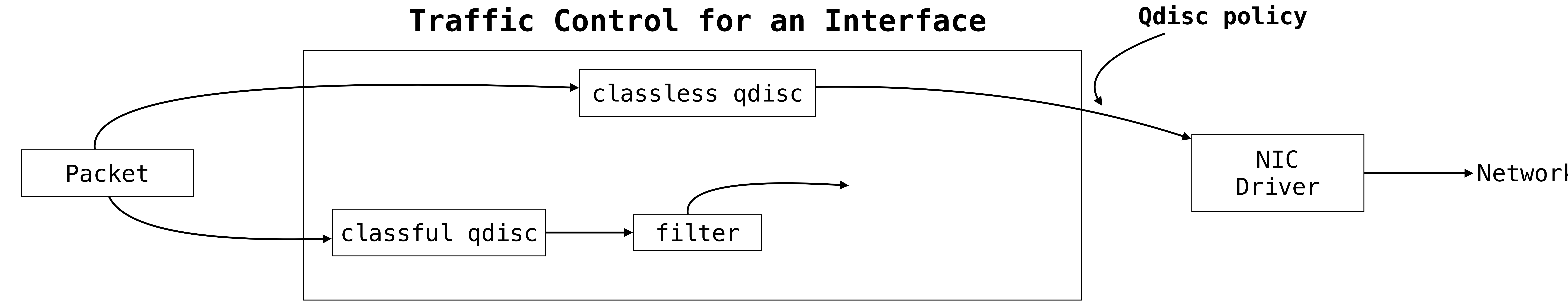
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



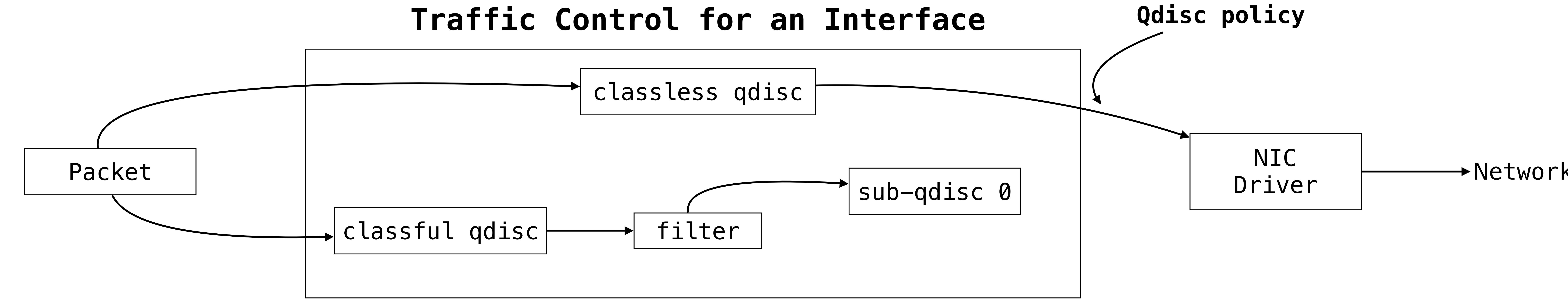
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



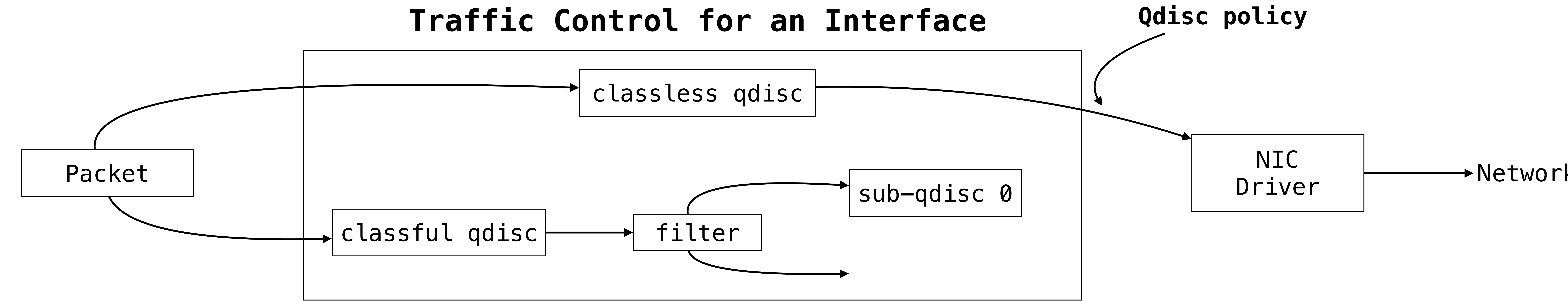
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



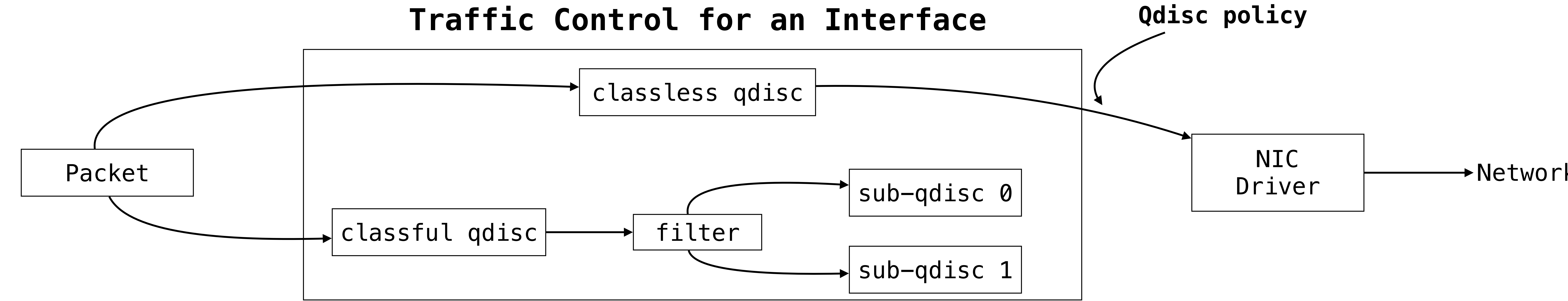
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



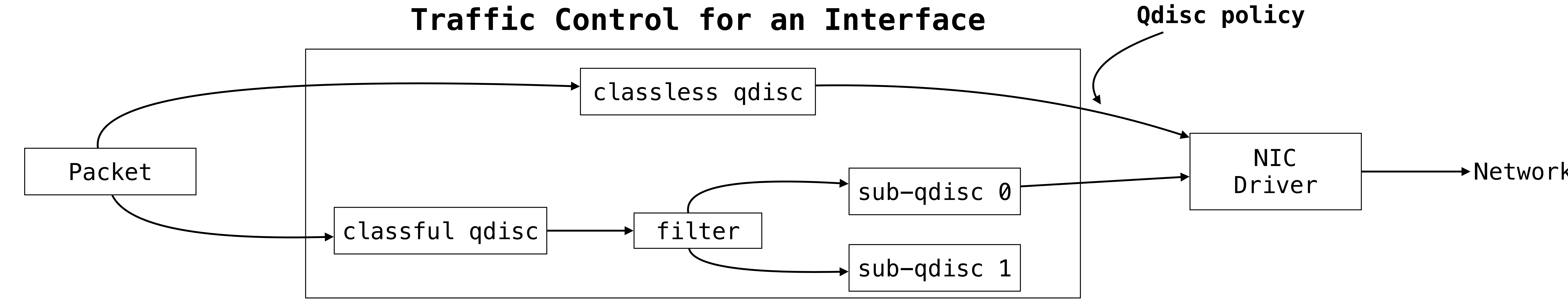
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



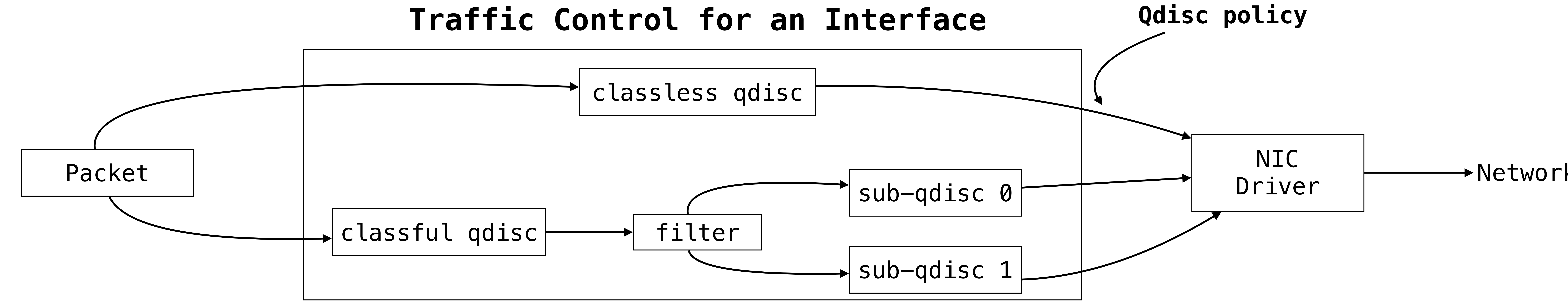
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



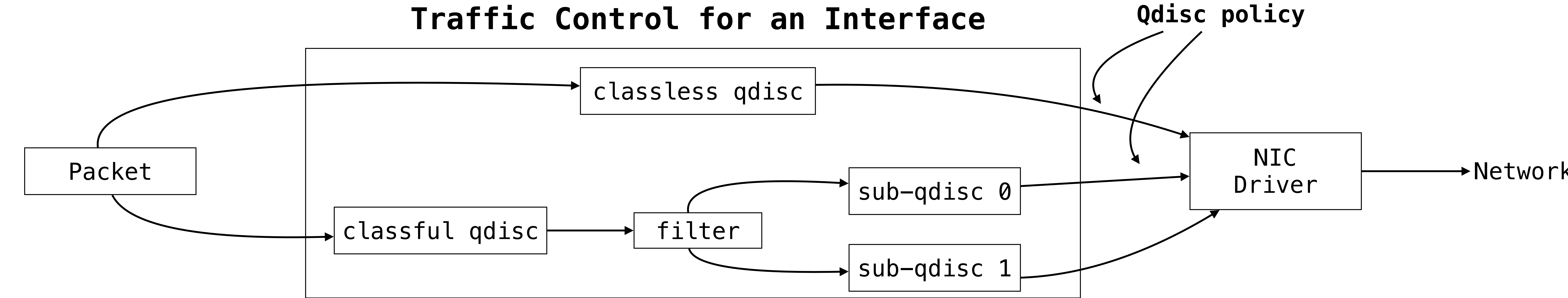
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



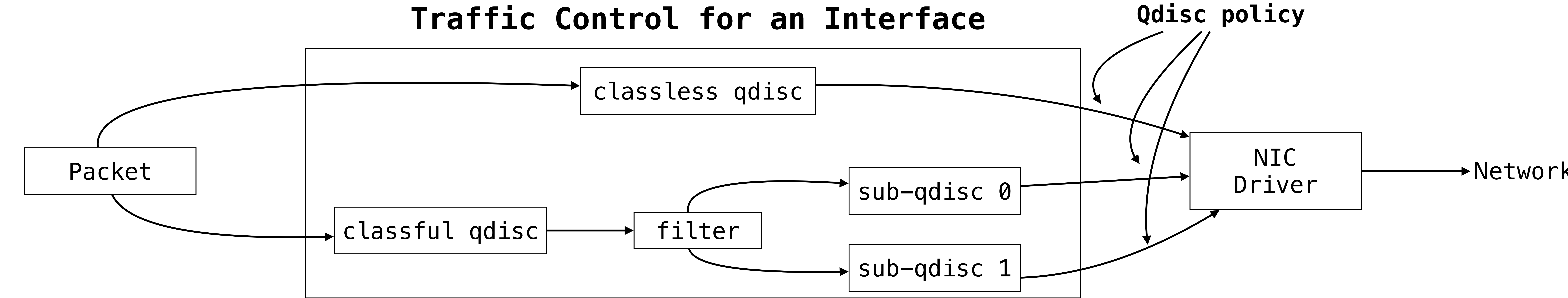
Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



Control de tráfico en Linux

- El control de tráfico es complejo: existe una jerarquía para definirlo:
  - Cada interfaz de red tiene asociada una cola con disciplina (**`qdisc`**).
  - El kernel intenta obtener paquetes de las colas para enviarlos.
  - Las políticas de las colas (i.e. `qdiscs`) determinan si un paquete puede «salir».
  - Existen `qdiscs` «**con clase**»: a su vez comprenden varias `qdiscs`.
    - En este caso, un **filtro** elige en que `qdisc` «interna» se encolan paquetes.



Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
    - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: `netem(8)`:
    - Nos permite añadir retardos, jitter, pérdidas…
    - Hace uso de la pila `QoS`/`DiffServ` del kernel.

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila `QoS`/[`DiffServ`] del kernel.

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: `netem(8)`:
  - Nos permite añadir retardos, jitter, pérdidas…
  - Hace uso de la pila `QoS`/`DiffServ` del kernel.

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila `QoS`/[`DiffServ`] del kernel.

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila `QoS`/[`DiffServ`] del kernel.

```
Mostramos la qdisc configurada en wg0

vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila `QoS`/[`DiffServ`] del kernel.

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: `netem(8)`:
  - Nos permite añadir retardos, jitter, pérdidas…
  - Hace uso de la pila `QoS`/`DiffServ` del kernel.

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila `QoS`/[`DiffServ`] del kernel.

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, jitter, pérdidas…
  - Hace uso de la pila `QoS`/`DiffServ` del kernel.

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, jitter, pérdidas…
  - Hace uso de la pila QoS/`DiffServ` del kernel.

Añadimos la qdisc principal

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, jitter, pérdidas…
  - Hace uso de la pila QoS/`DiffServ` del kernel.

Añadimos la qdisc principal

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila QoS/[`DiffServ`] del kernel.

Añadimos la qdisc principal

La qdisc a añadir es netem

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: `netem(8)`:
  - Nos permite añadir retardos, jitter, pérdidas…
  - Hace uso de la pila `QoS`/`DiffServ` del kernel.

Añadimos la qdisc principal

La qdisc a añadir es netem

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila QoS/[`DiffServ`] del kernel.

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Añadimos la qdisc principal

La qdisc a añadir es netem

Añadimos un retardo de 500000 us = 500 ms

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila QoS/[`DiffServ`] del kernel.

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Añadimos la qdisc principal

La qdisc a añadir es netem

Añadimos un retardo de 500000 us = 500 ms

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: `netem(8)`:
  - Nos permite añadir retardos, jitter, pérdidas…
  - Hace uso de la pila `QoS/DiffServ` del kernel.

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Añadimos la qdisc principal

La qdisc a añadir es netem

Añadimos un retardo de 500000 us = 500 ms

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc netem 8003: root refcnt 2 limit 1000 delay 500ms
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila `QoS/DiffServ` del kernel.

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Añadimos la qdisc principal

La qdisc a añadir es netem

Añadimos un retardo de 500000 us = 500 ms

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc netem 8003: root refcnt 2 limit 1000 delay 500ms
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila QoS/[`DiffServ`] del kernel.

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Añadimos la qdisc principal

La qdisc a añadir es netem

Añadimos un retardo de 500000 us = 500 ms

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc netem 8003: root refcnt 2 limit 1000 delay 500ms
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, jitter, pérdidas…
  - Hace uso de la pila QoS/`DiffServ` del kernel.

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Añadimos la qdisc principal

La qdisc a añadir es netem

Añadimos un retardo de 500000 us = 500 ms

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc netem 8003: root refcnt 2 limit 1000 delay 500ms
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila `QoS`/[`DiffServ`] del kernel.

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Añadimos la qdisc principal

La qdisc a añadir es netem

Añadimos un retardo de 500000 us = 500 ms

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Hemos configurado netem como qdisk

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc netem 8003: root refcnt 2 limit 1000 delay 500ms
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila QoS/[`DiffServ`] del kernel.

Añadimos la qdisc principal

La qdisc a añadir es netem

Añadimos un retardo de 500000 us = 500 ms

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Hemos configurado netem como qdisk

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc netem 8003: root refcnt 2 limit 1000 delay 500ms
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila QoS/[`DiffServ`] del kernel.

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Añadimos la qdisc principal

La qdisc a añadir es netem

Añadimos un retardo de 500000 us = 500 ms

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Hemos configurado netem como qdisk

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc netem 8003: root refcnt 2 limit 1000 delay 500ms
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Vemos el retardo de 500 ms

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila `QoS/`[`DiffServ`] del kernel.

Añadimos la qdisc principal

La qdisc a añadir es netem

Añadimos un retardo de
500000 us = 500 ms

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Esta qdisc equivale a
«no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Hemos configurado
netem como qdisk

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc netem 8003: root refcnt 2 limit 1000 delay 500ms
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Vemos el retardo de 500 ms

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: `netem(8)`:
  - Nos permite añadir retardos, jitter, pérdidas…
  - Hace uso de la pila QoS/`DiffServ` del kernel.

```
vagrant@vpn-core:~$ sudo tc qdisc del dev wg0 root
```

Esta qdisc equivale a «no hacer nada».

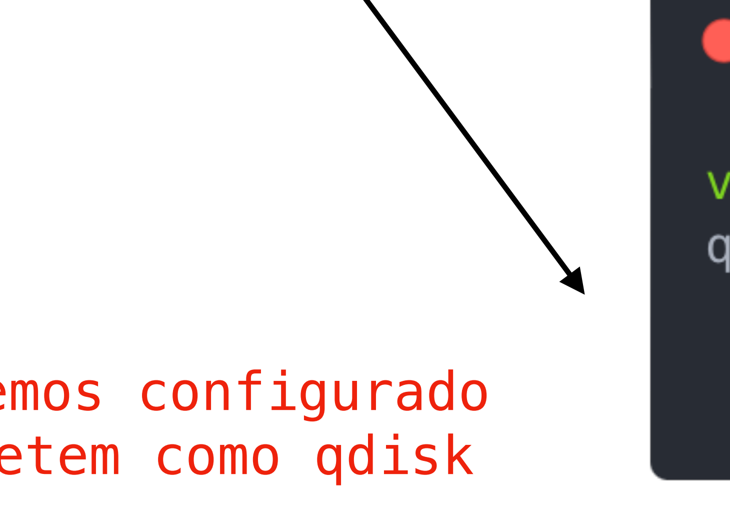
```
Mostramos la qdisc configurada en wg0
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

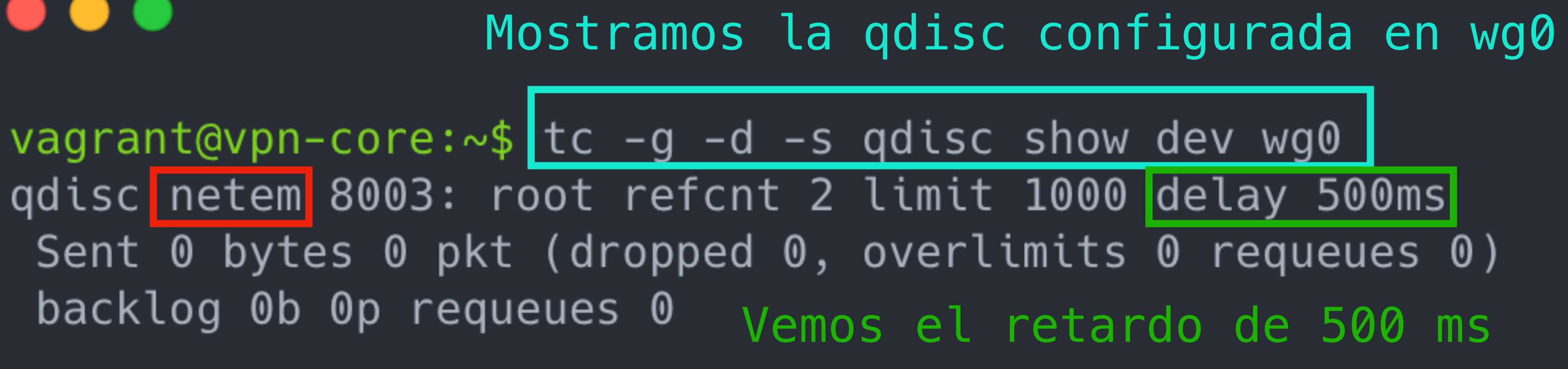
```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Añadimos la qdisc principal

La qdisc a añadir es netem

Añadimos un retardo de 500000 us = 500 ms





Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila QoS/`DiffServ` del kernel.

```
vagrant@vpn-core:~$ sudo tc qdisc del dev wg0 root
```

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Añadimos la qdisc principal

La qdisc a añadir es netem

Añadimos un retardo de 500000 us = 500 ms

Esta qdisc equivale a «no hacer nada».

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Hemos configurado netem como qdisk

Mostramos la qdisc configurada en wg0

```
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc netem 8003: root refcnt 2 limit 1000 delay 500ms
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Vemos el retardo de 500 ms

Nuestra prueba de concepto

- Queremos demostrar que WireGuard se integra con `tc`:
  - La interfaz `wg0` tiene asociada una `qdisc` como todas las demás.
- Usaremos una qdisc simple: [`netem(8)`]:
  - Nos permite añadir retardos, [jitter], pérdidas…
  - Hace uso de la pila `QoS`/[`DiffServ`] del kernel.

```
Volvemos a dejar la qdisc por defecto
vagrant@vpn-core:~$ sudo tc qdisc del dev wg0 root
```

Añadimos la qdisc principal

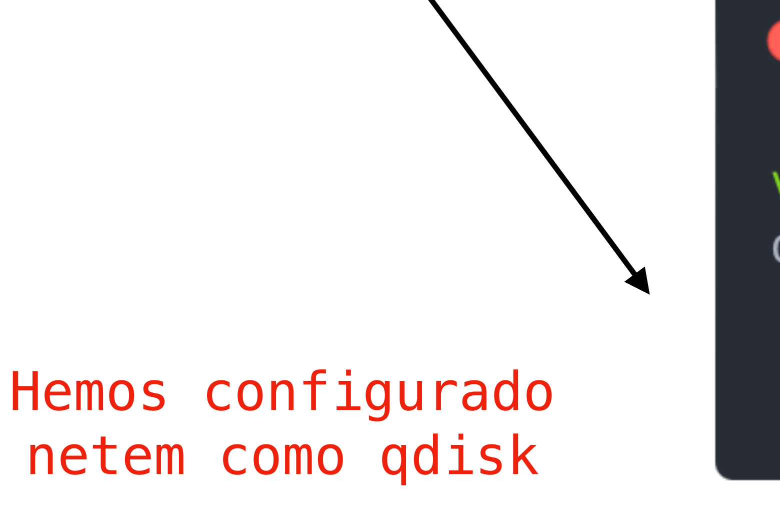
La qdisc a añadir es netem

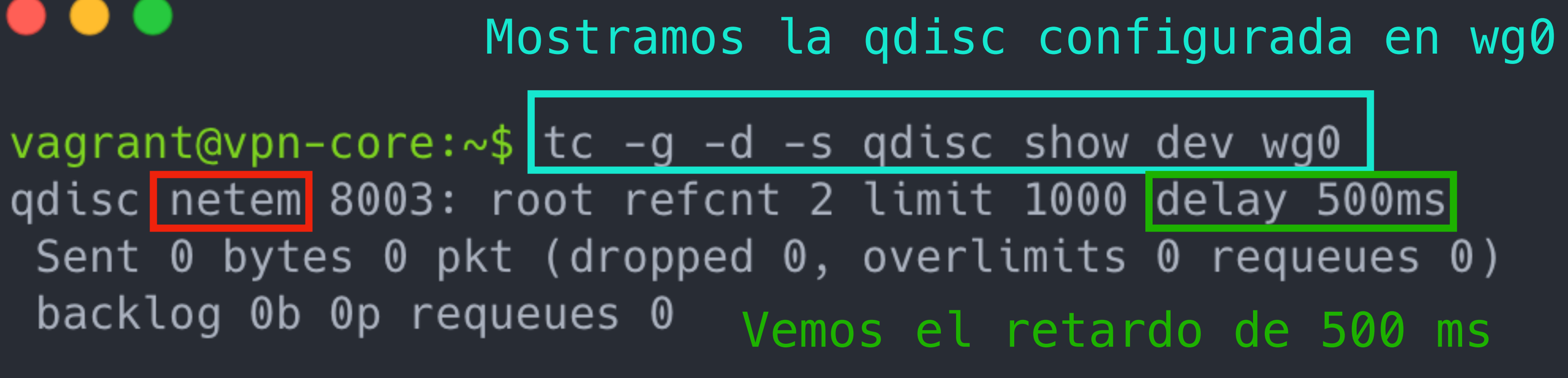
Añadimos un retardo de 500000 us = 500 ms

```
vagrant@vpn-core:~$ sudo tc qdisc add dev wg0 root netem delay 500000
```

Esta qdisc equivale a «no hacer nada».

```
Mostramos la qdisc configurada en wg0
vagrant@vpn-core:~$ tc -g -d -s qdisc show dev wg0
qdisc noqueue 0: root refcnt 2
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```





Comprobando los retardos

```
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.
64 bytes from 192.168.4.3: icmp_seq=1 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=2 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=3 ttl=63 time=1003 ms

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2010ms
rtt min/avg/max/mdev = 1003.343/1004.430/1005.247/0.800 ms
```

Comprobando los retardos

```
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.
64 bytes from 192.168.4.3: icmp_seq=1 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=2 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=3 ttl=63 time=1003 ms

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2010ms
rtt min/avg/max/mdev = 1003.343/1004.430/1005.247/0.800 ms
```

Comprobando los retardos

```
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.
64 bytes from 192.168.4.3: icmp_seq=1 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=2 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=3 ttl=63 time=1003 ms

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2010ms
rtt min/avg/max/mdev = 1003.343/1004.430/1005.247/0.800 ms
```

Comprobando los retardos

client-b

```
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.
64 bytes from 192.168.4.3: icmp_seq=1 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=2 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=3 ttl=63 time=1003 ms

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2010ms
rtt min/avg/max/mdev = 1003.343/1004.430/1005.247/0.800 ms
```

Comprobando los retardos

client-b

```
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.
64 bytes from 192.168.4.3: icmp_seq=1 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=2 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=3 ttl=63 time=1003 ms

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2010ms
rtt min/avg/max/mdev = 1003.343/1004.430/1005.247/0.800 ms
```

Comprobando los retardos

client-b

```
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.
64 bytes from 192.168.4.3: icmp_seq=1 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=2 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=3 ttl=63 time=1003 ms

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2010ms
rtt min/avg/max/mdev = 1003.343/1004.430/1005.247/0.800 ms
```

Comprobando los retardos

client-b

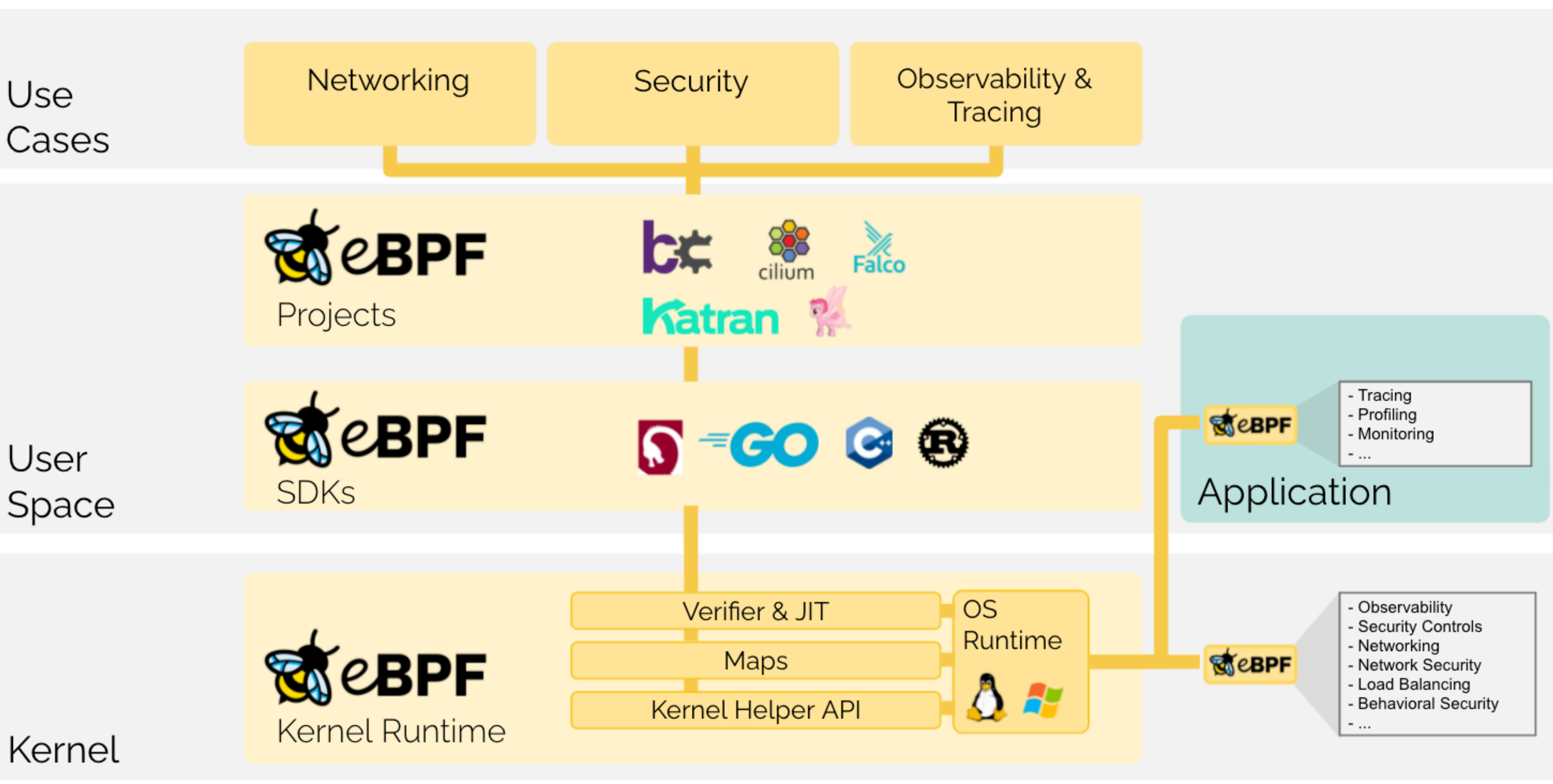
```
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.
64 bytes from 192.168.4.3: icmp_seq=1 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=2 ttl=63 time=1005 ms
64 bytes from 192.168.4.3: icmp_seq=3 ttl=63 time=1003 ms

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2010ms
rtt min/avg/max/mdev = 1003.343/1004.430/1005.247/0.800 ms
```

$$d_{RTT} = d_{a \to core} + d_{netem} + d_{core \to b} + d_{b \to core} + d_{netem} + d_{core \to a} \approx 2 \cdot d_{netem} = 1000\ ms$$

Un paso más allá con eBPF

Fuente

- eBPF es una tecnología que nos permite:
  - Ejecutar programas en un «sandbox» en espacio de kernel.
  - Nos permite extender el kernel sin módulos ni recompilaciones.
- La comunicación entre el espacio de usuario y kernel se hace a través de **mapas**.
- Es aplicable a múltiples casos de uso.

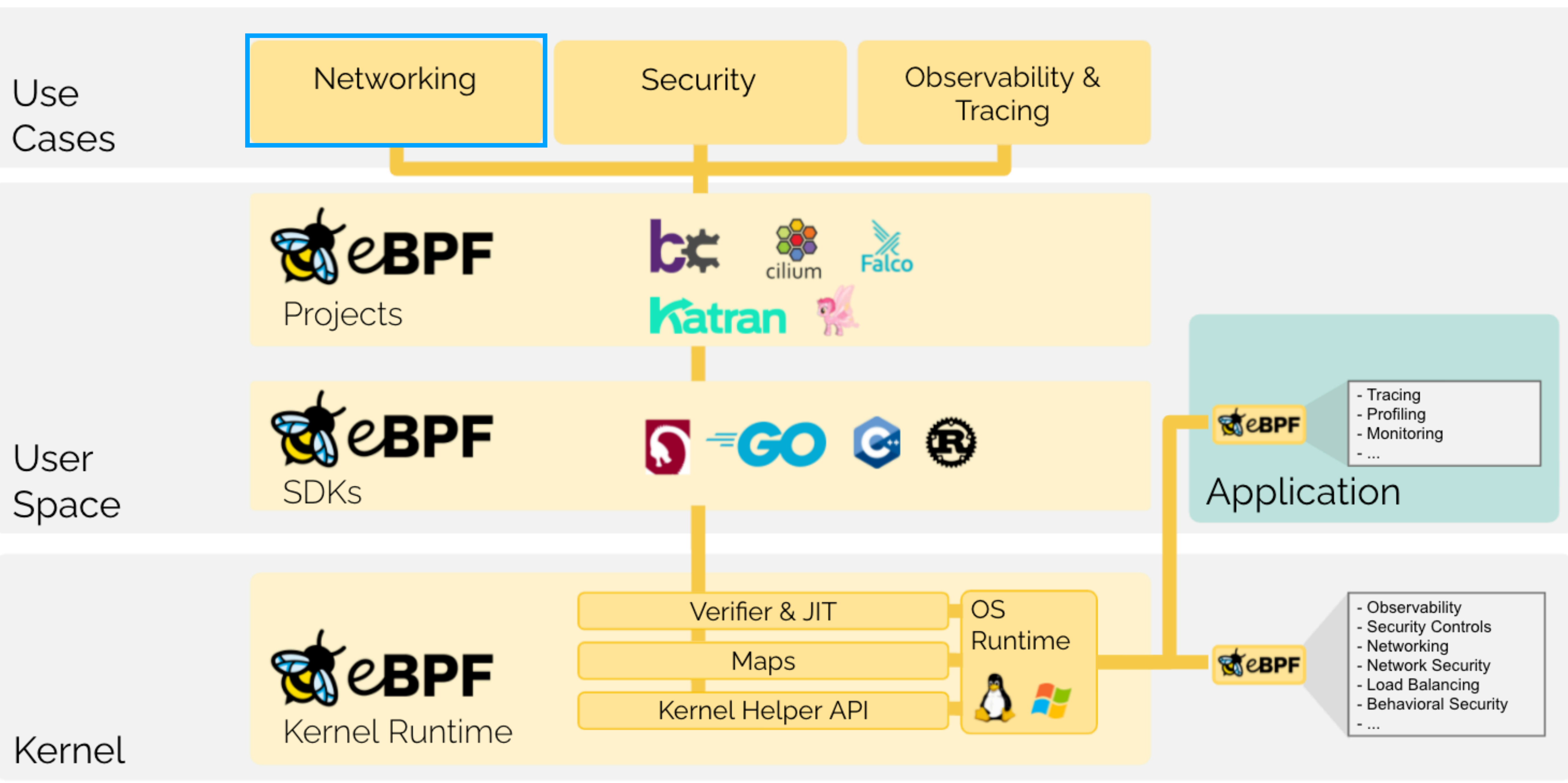


Fuente

Un paso más allá con eBPF

Fuente

- eBPF es una tecnología que nos permite:
    - Ejecutar programas en un «sandbox» en espacio de kernel.
    - Nos permite extender el kernel sin módulos ni recompilaciones.
- La comunicación entre el espacio de usuario y kernel se hace a través de **mapas**.
- Es aplicable a múltiples casos de uso.

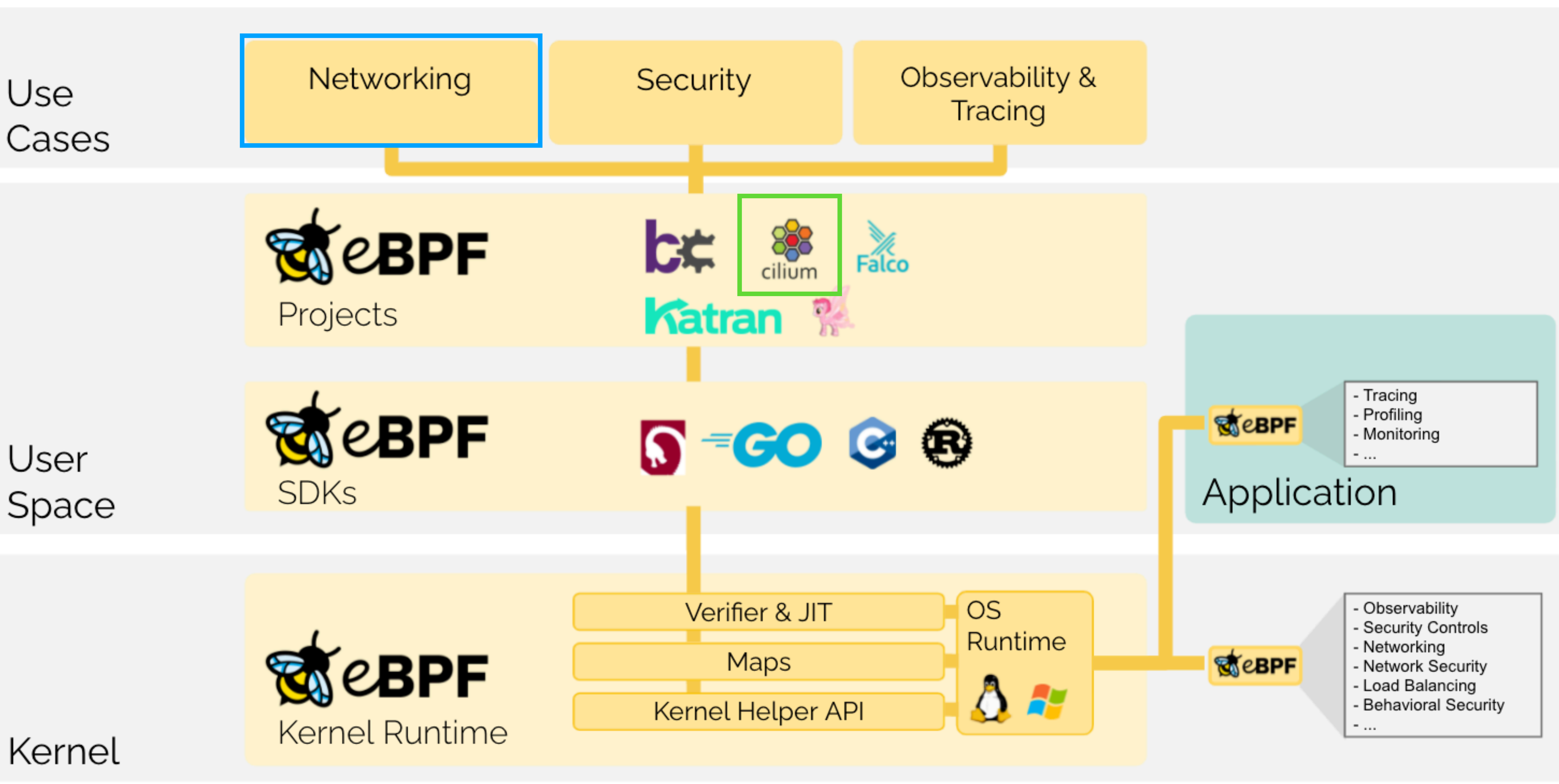


Fuente

Un paso más allá con eBPF

Fuente

- eBPF es una tecnología que nos permite:
  - Ejecutar programas en un «sandbox» en espacio de kernel.
  - Nos permite extender el kernel sin módulos ni recompilaciones.
- La comunicación entre el espacio de usuario y kernel se hace a través de **mapas**.
- Es aplicable a múltiples casos de uso.

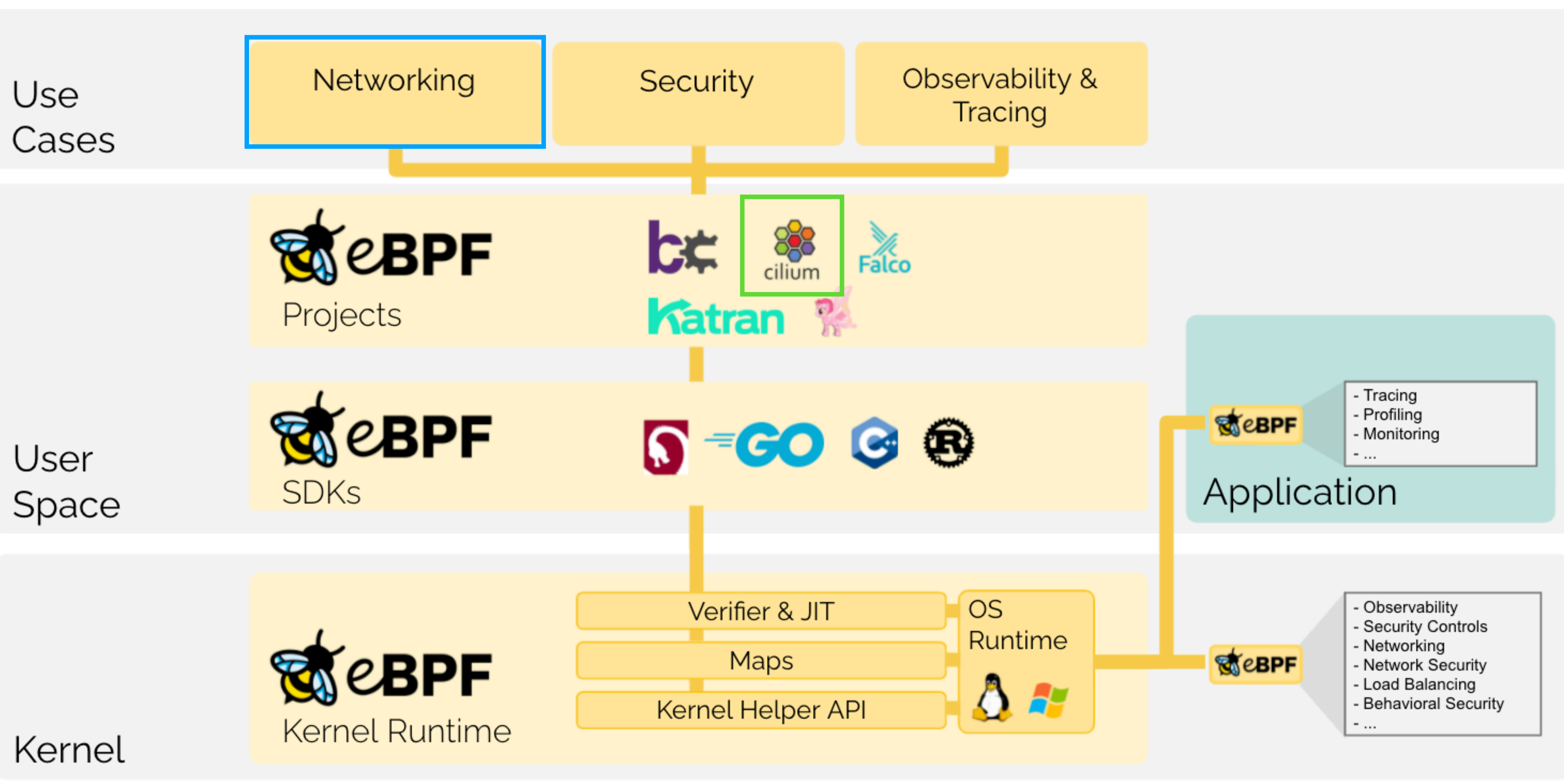


Fuente

Un paso más allá con eBPF

Fuente

- eBPF es una tecnología que nos permite:
    - Ejecutar programas en un «sandbox» en espacio de kernel.
    - Nos permite extender el kernel sin módulos ni recompilaciones.
- La comunicación entre el espacio de usuario y kernel se hace a través de **mapas**.
- Es aplicable a múltiples casos de uso.



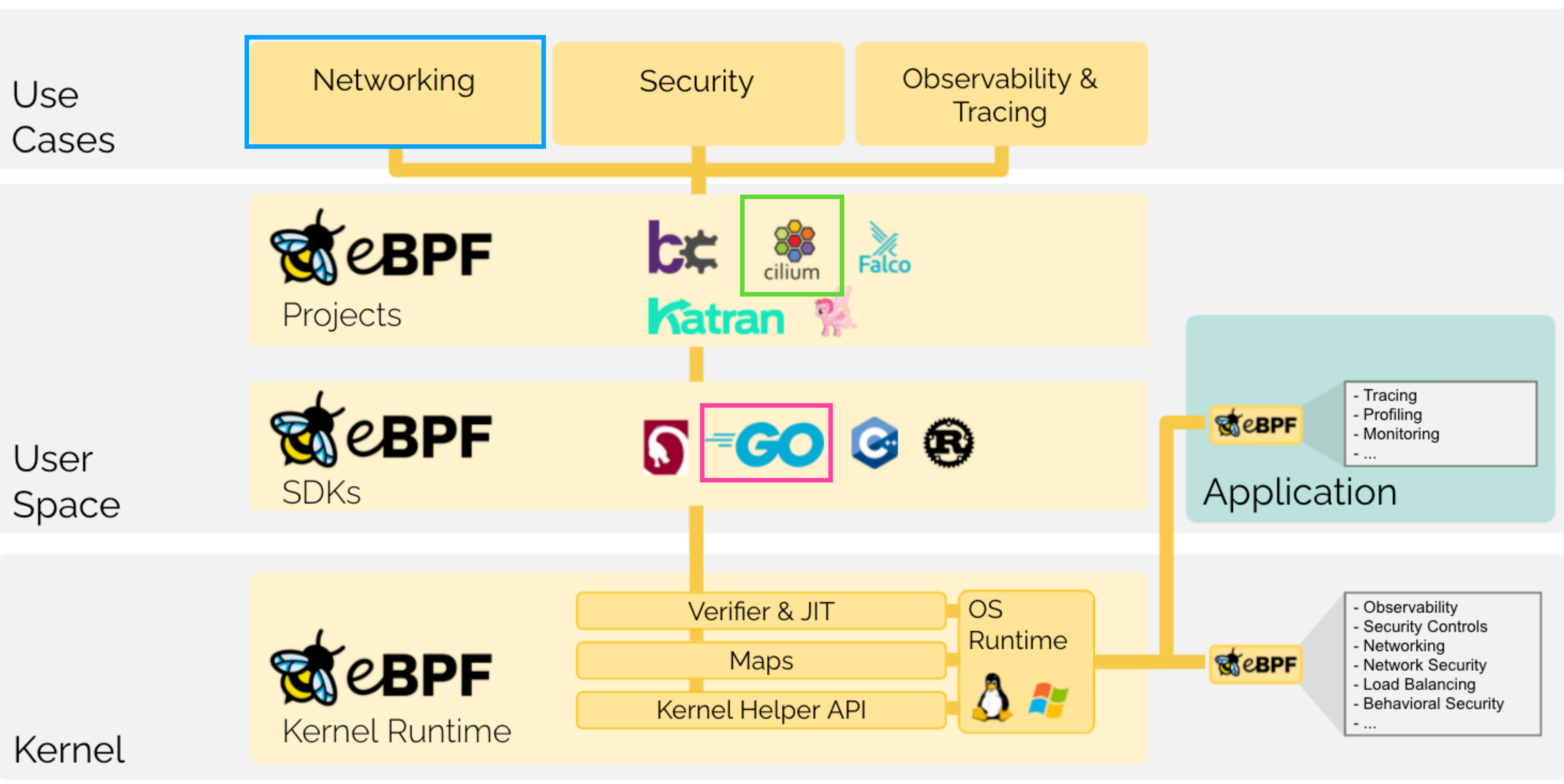
Nosotros hacemos uso de cilium/ebpf. Este proyecto nos permite **compilar** y **cargar** nuestros programas eBPF.

Fuente

Un paso más allá con eBPF

Fuente

- eBPF es una tecnología que nos permite:
  - Ejecutar programas en un «sandbox» en espacio de kernel.
  - Nos permite extender el kernel sin módulos ni recompilaciones.
- La comunicación entre el espacio de usuario y kernel se hace a través de **mapas**.
- Es aplicable a múltiples casos de uso.



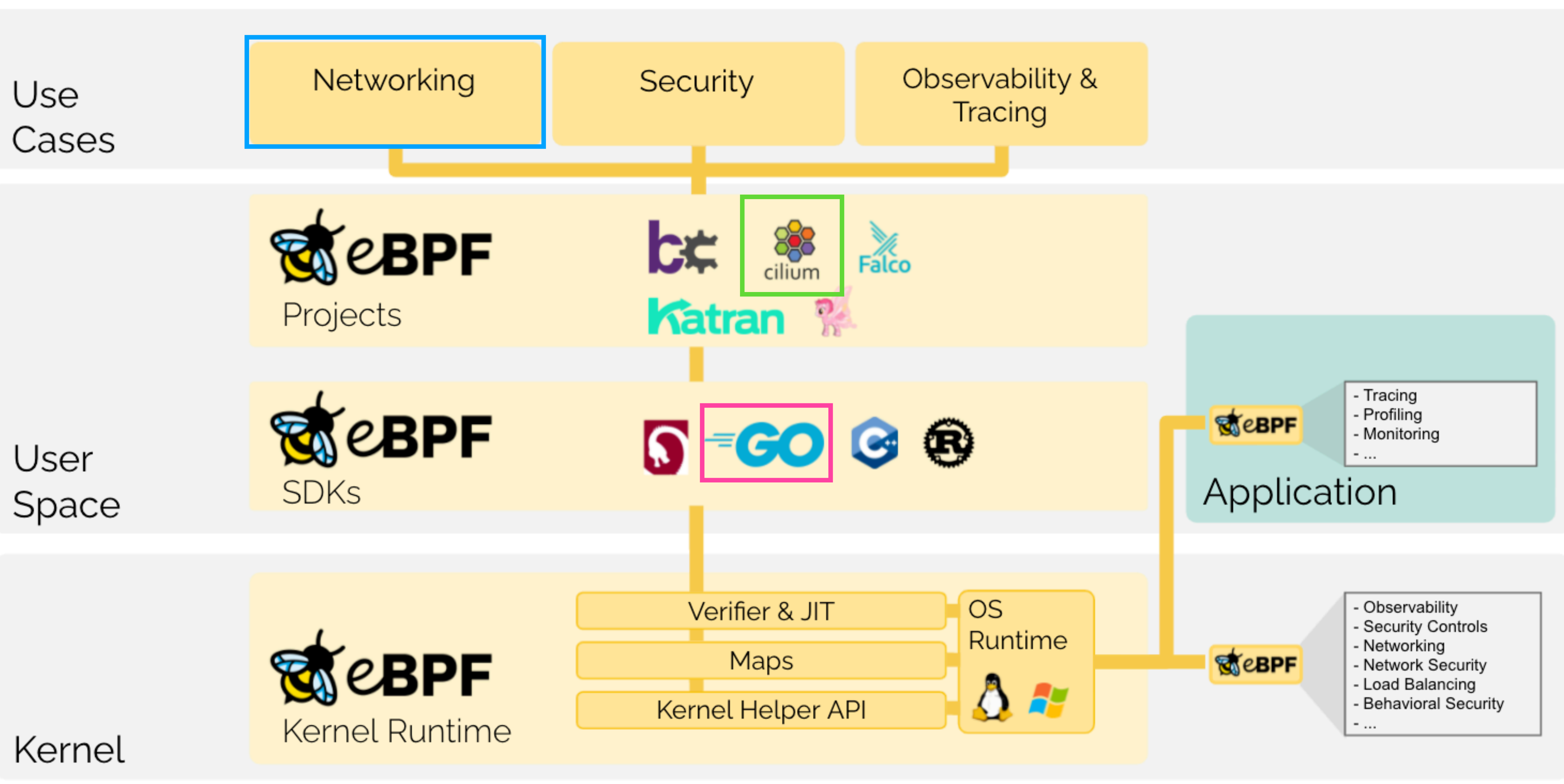
Nosotros hacemos uso de cilium/ebpf. Este proyecto nos permite **compilar** y **cargar** nuestros programas eBPF.

Fuente

Un paso más allá con eBPF

Fuente

- eBPF es una tecnología que nos permite:
  - Ejecutar programas en un «sandbox» en espacio de kernel.
  - Nos permite extender el kernel sin módulos ni recompilaciones.
- La comunicación entre el espacio de usuario y kernel se hace a través de **mapas**.
- Es aplicable a múltiples casos de uso.



Nosotros hacemos uso de cilium/ebpf. Este proyecto nos permite **compilar** y **cargar** nuestros programas eBPF.

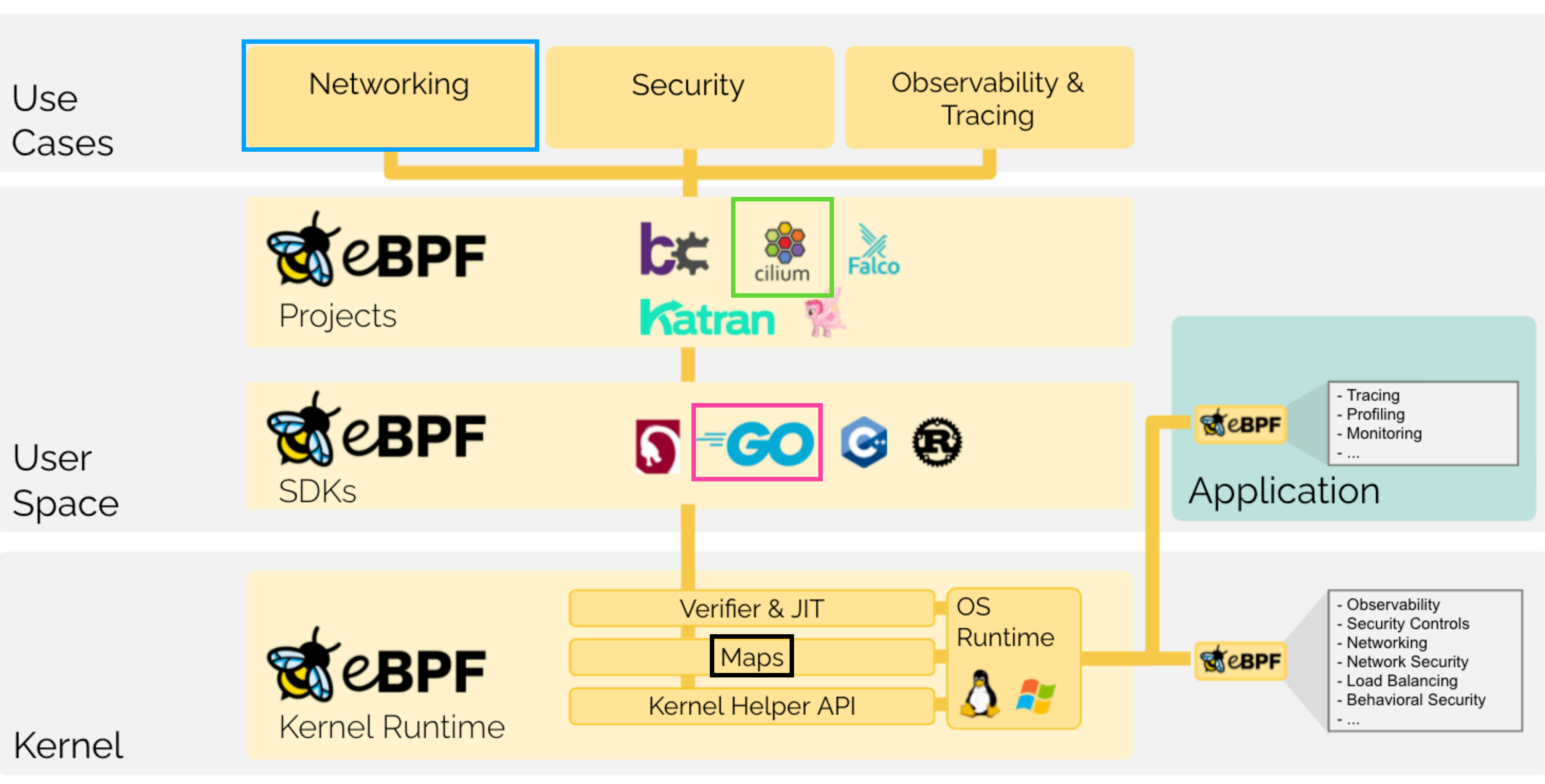
El proyecto está implementado en Go.

Fuente

Un paso más allá con eBPF

Fuente

- eBPF es una tecnología que nos permite:
  - Ejecutar programas en un «sandbox» en espacio de kernel.
  - Nos permite extender el kernel sin módulos ni recompilaciones.
- La comunicación entre el espacio de usuario y kernel se hace a través de **mapas**.
- Es aplicable a múltiples casos de uso.

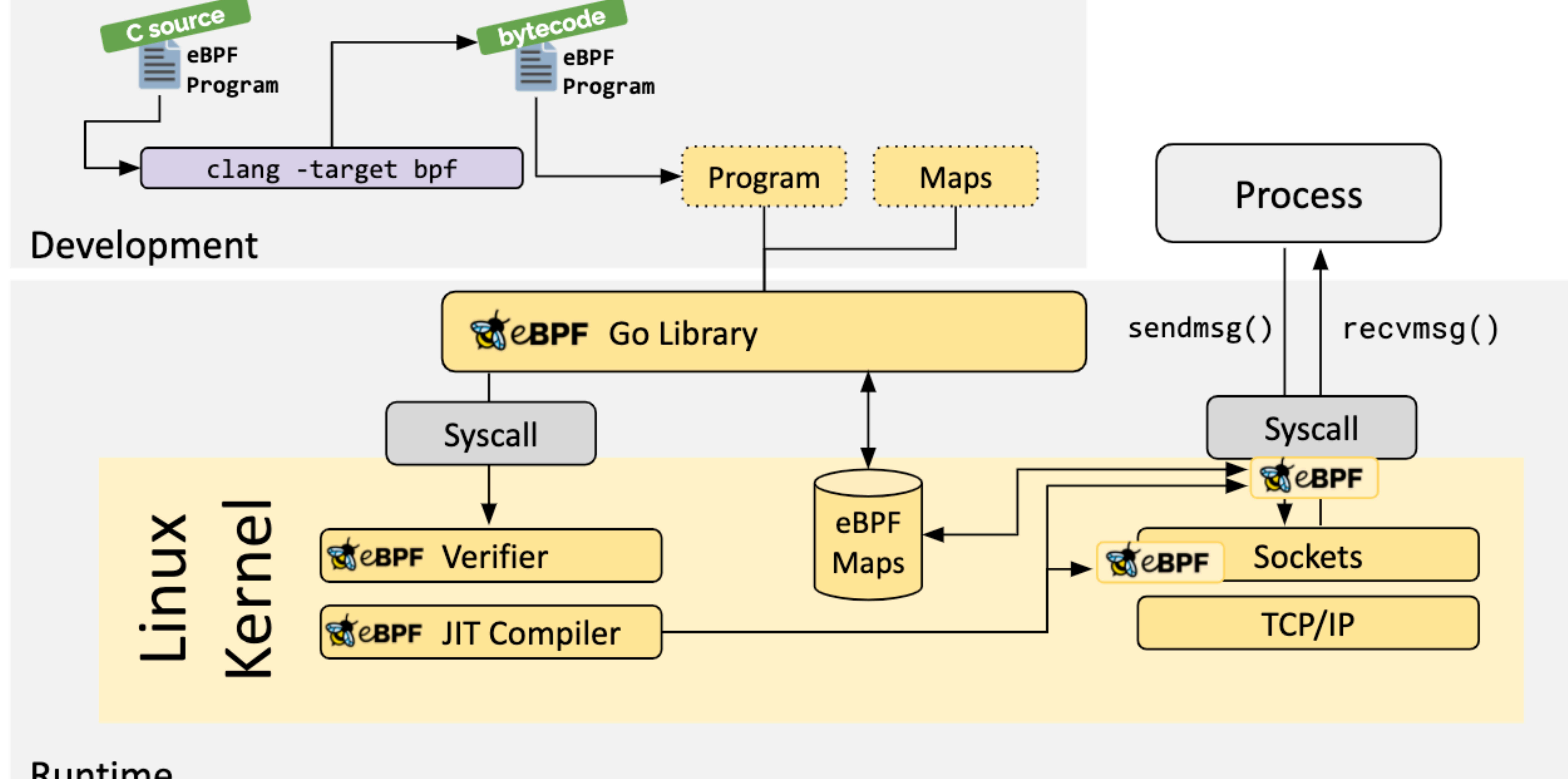


Nosotros hacemos uso de cilium/ebpf. Este proyecto nos permite **compilar** y **cargar** nuestros programas eBPF.

El proyecto está implementado en Go.

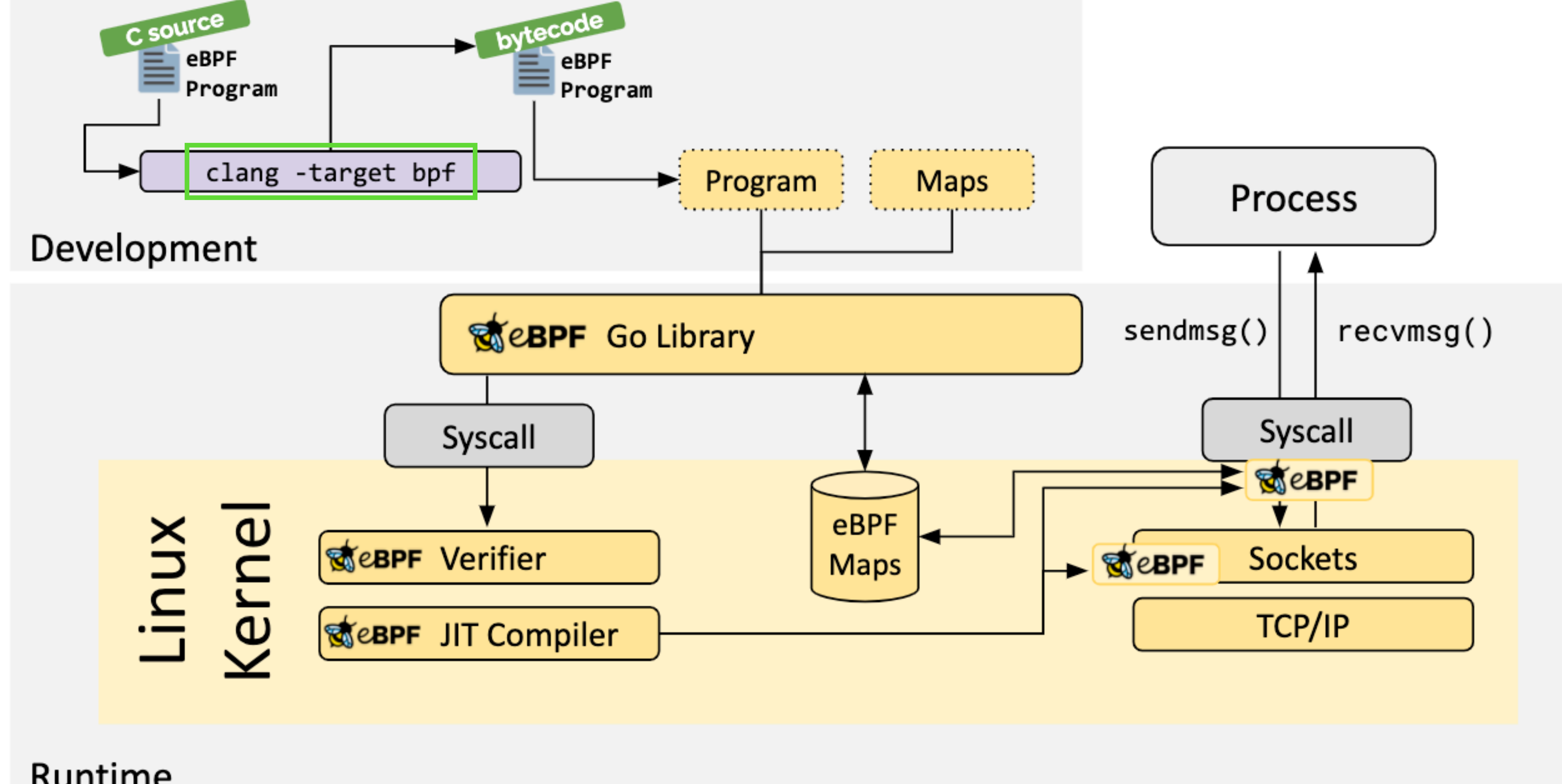
Fuente

Un poco más sobre eBPF



Fuente

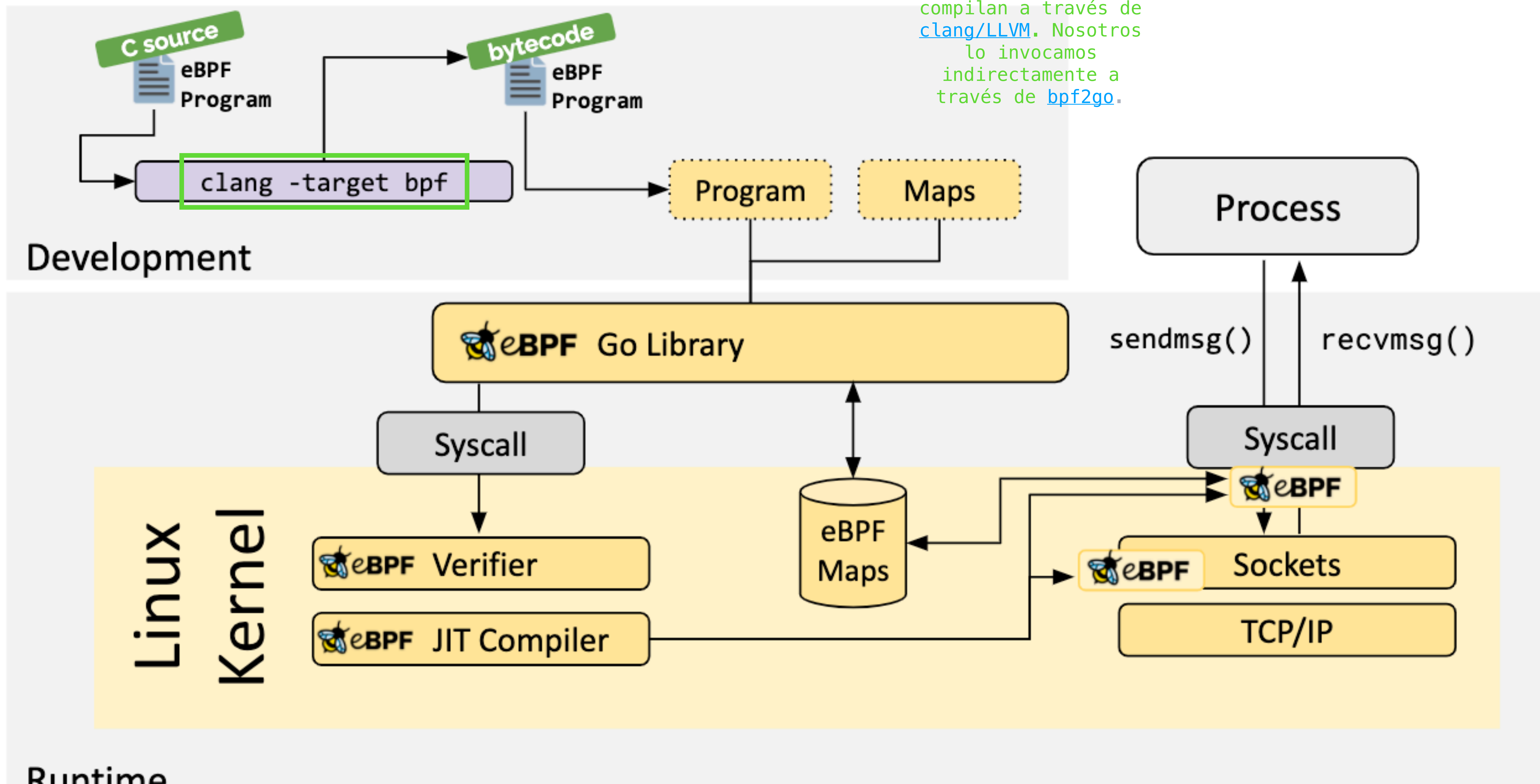
Un poco más sobre eBPF



Fuente

Un poco más sobre eBPF

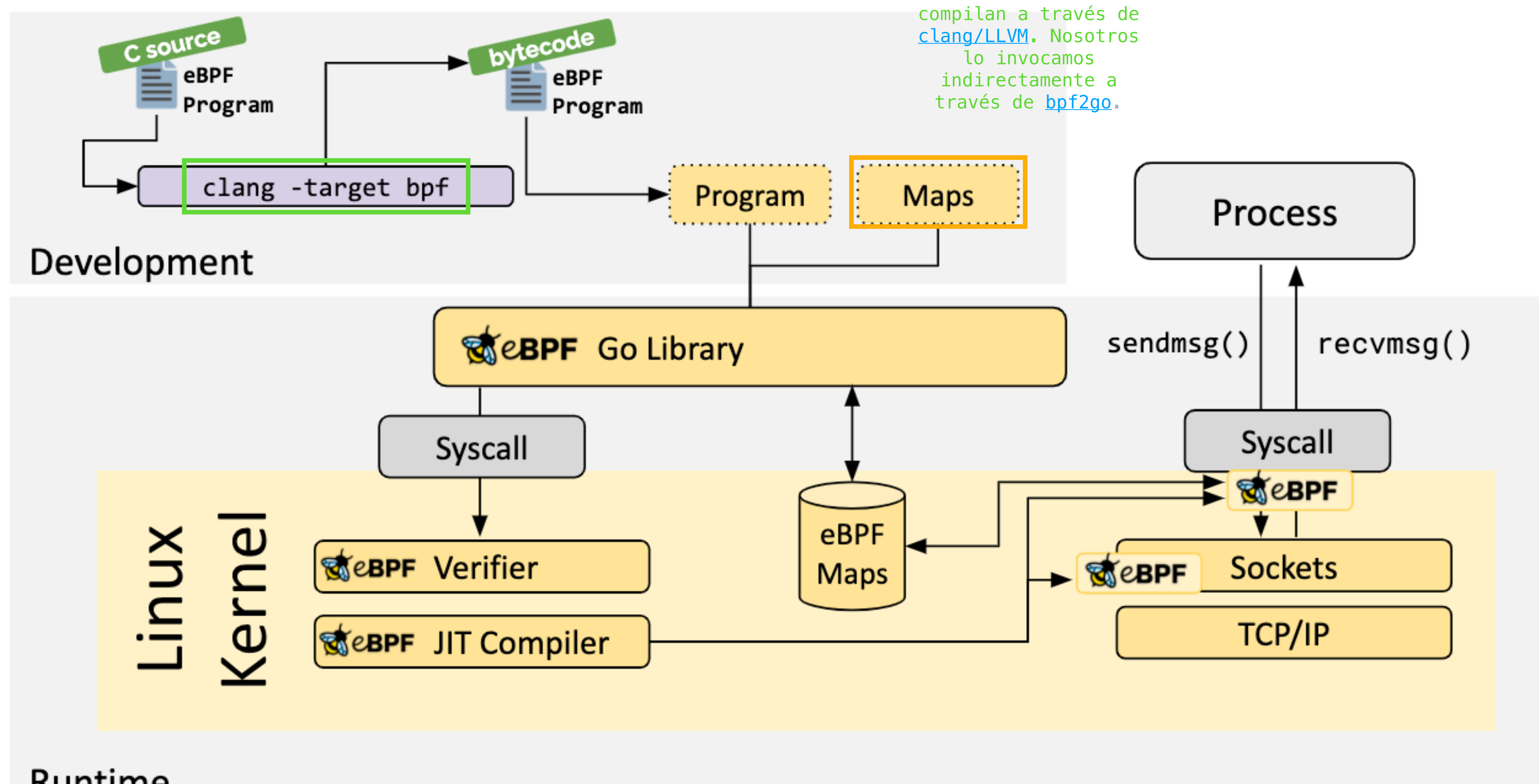
Las fuentes se compilan a través de clang/LLVM. Nosotros lo invocamos indirectamente a través de bpf2go.



Fuente

Un poco más sobre eBPF

Las fuentes se compilan a través de clang/LLVM. Nosotros lo invocamos indirectamente a través de bpf2go.

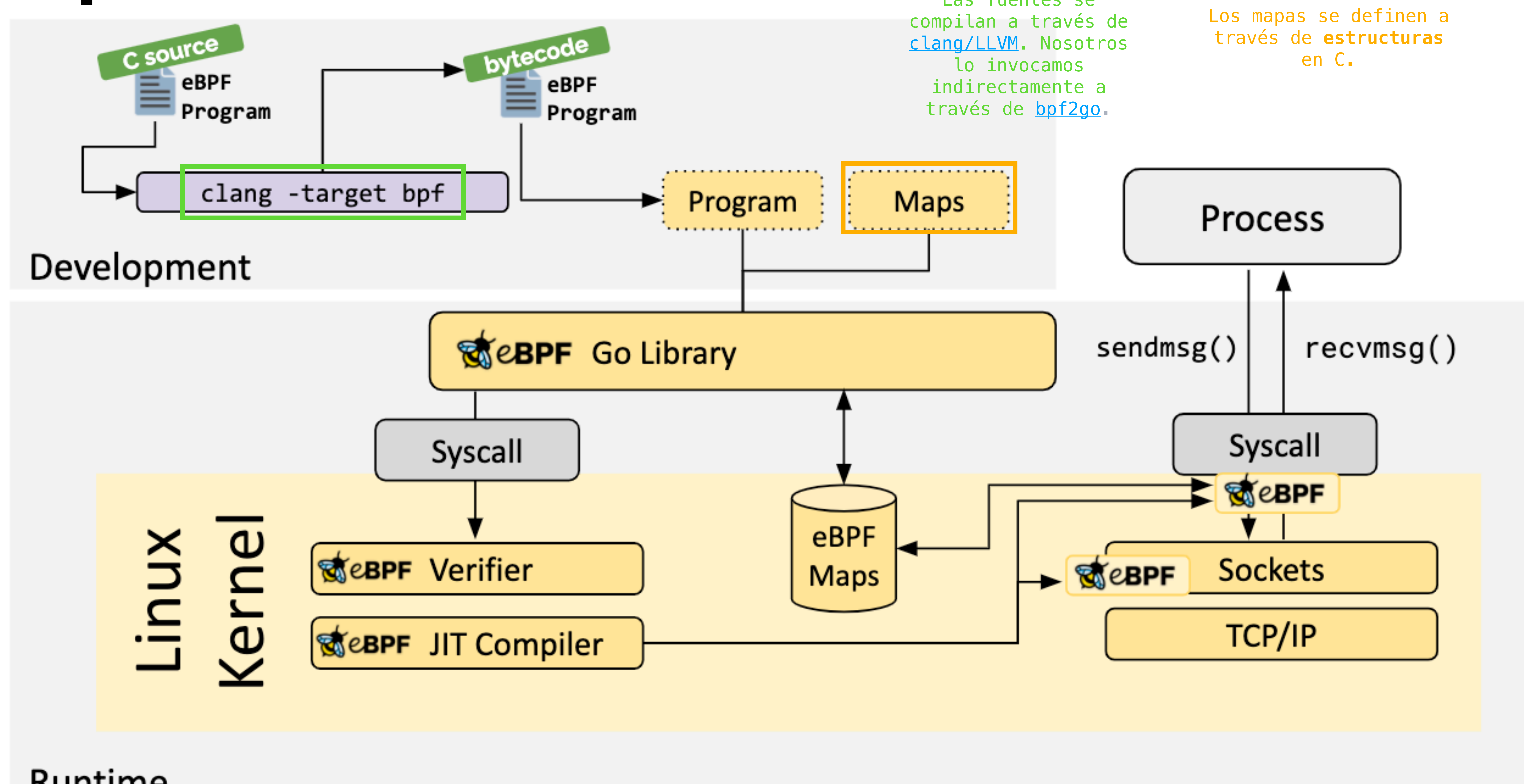


Fuente

Un poco más sobre eBPF

Las fuentes se compilan a través de clang/LLVM. Nosotros lo invocamos indirectamente a través de bpf2go.

Los mapas se definen a través de **estructuras** en C.

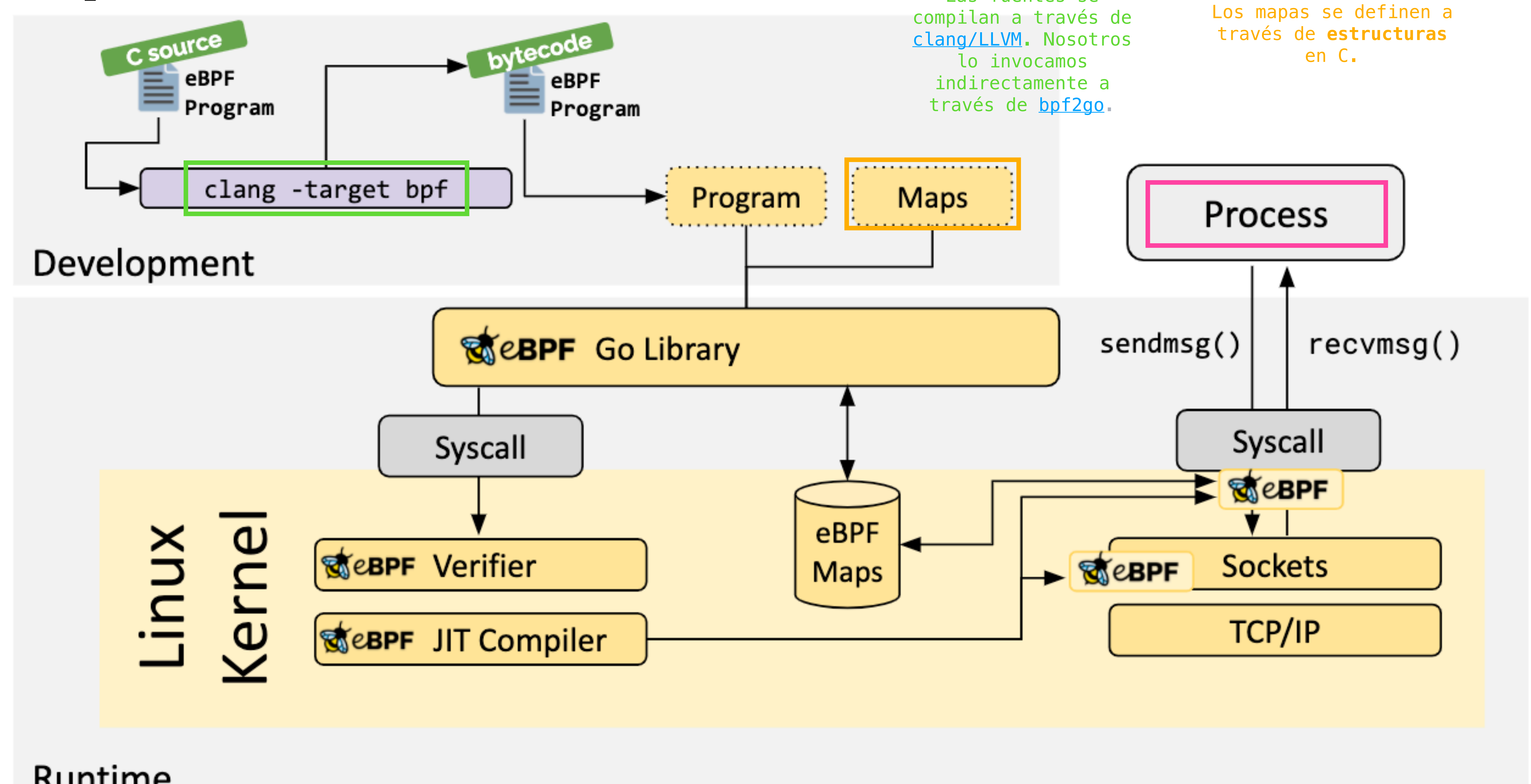


Fuente

Un poco más sobre eBPF

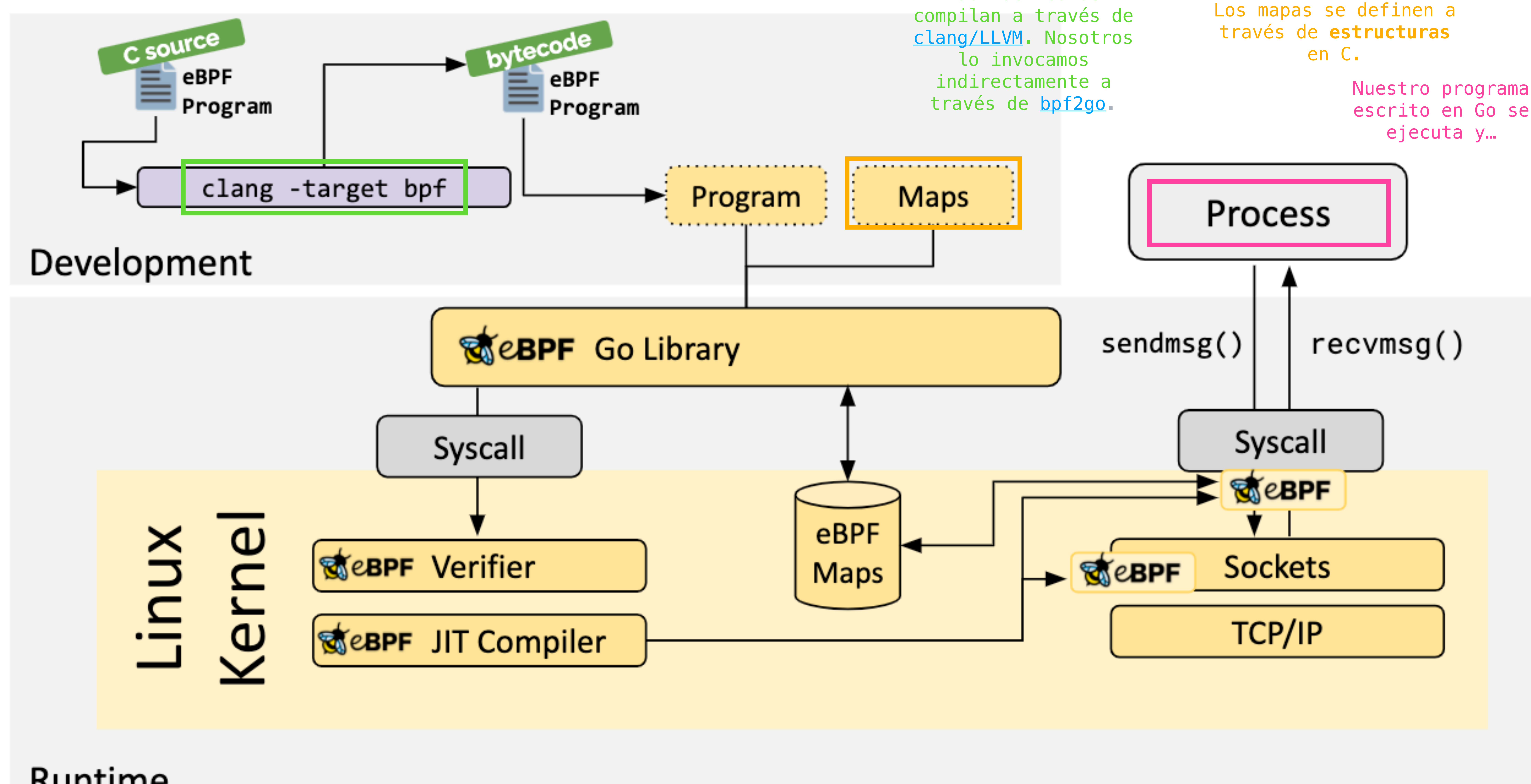
Las fuentes se compilan a través de [clang/LLVM](). Nosotros lo invocamos indirectamente a través de [bpf2go]().

Los mapas se definen a través de **estructuras** en C.



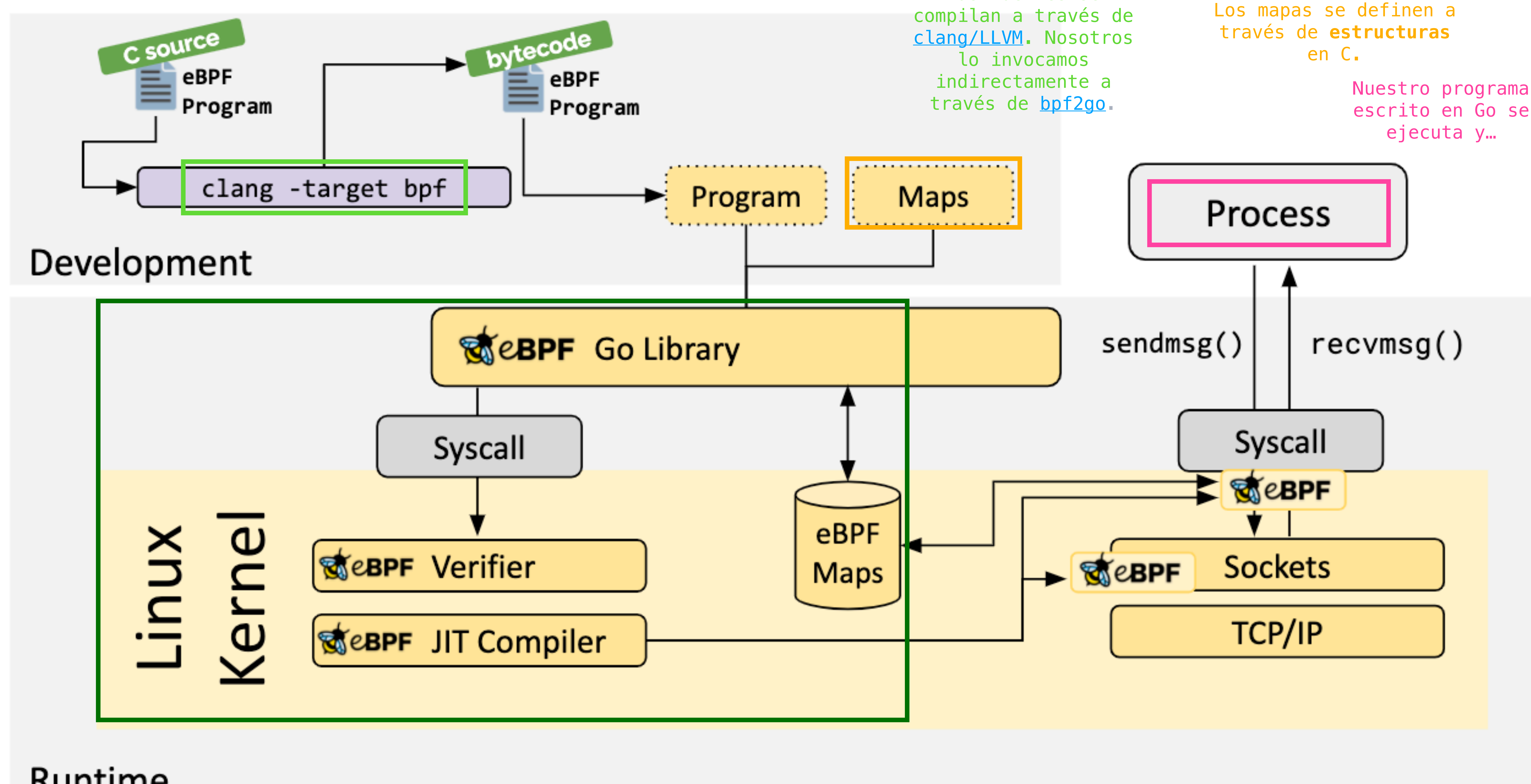
[Fuente]()

Un poco más sobre eBPF



Fuente

Un poco más sobre eBPF



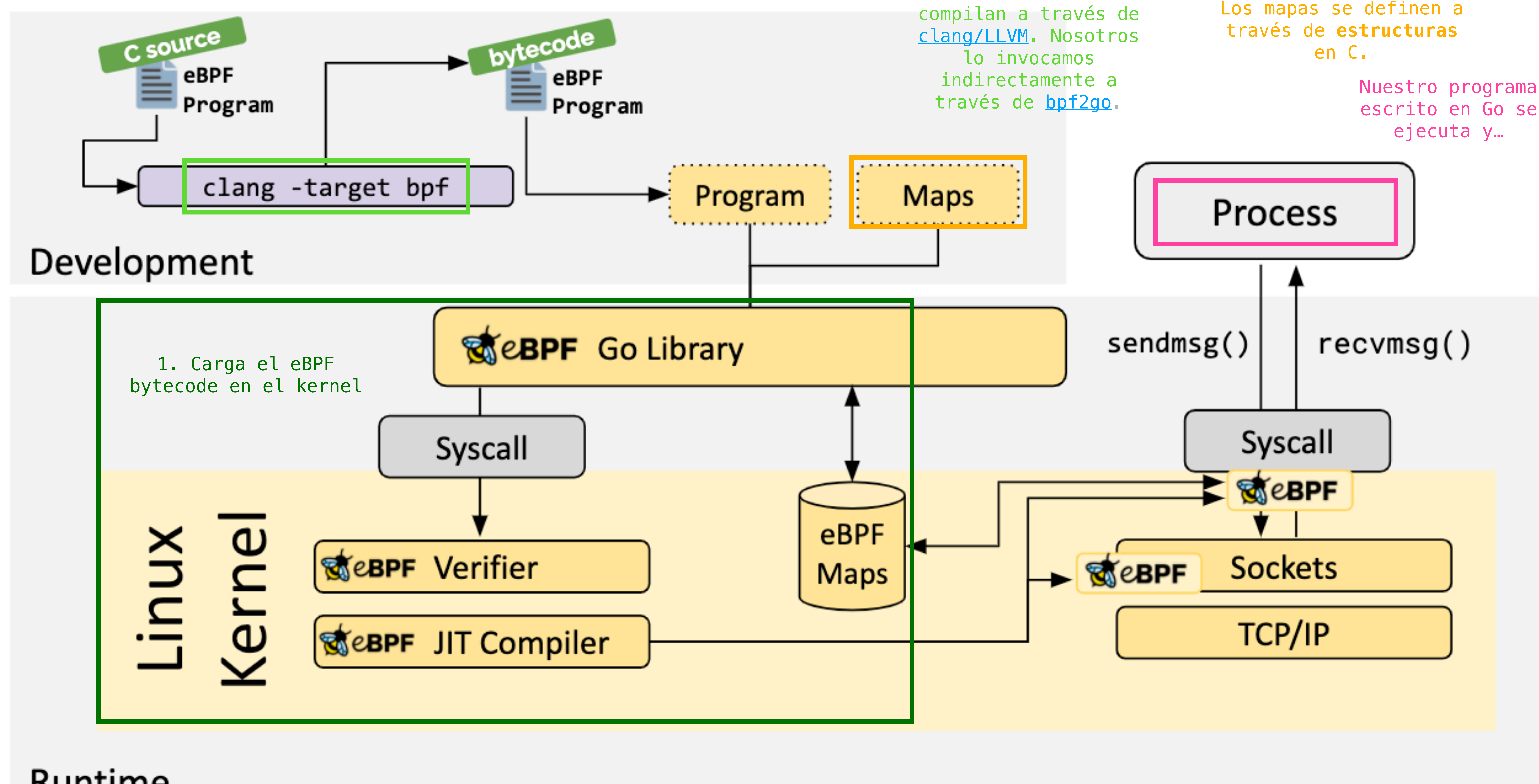
Las fuentes se compilan a través de clang/LLVM. Nosotros lo invocamos indirectamente a través de bpf2go.

Los mapas se definen a través de **estructuras** en C.

Nuestro programa escrito en Go se ejecuta y…

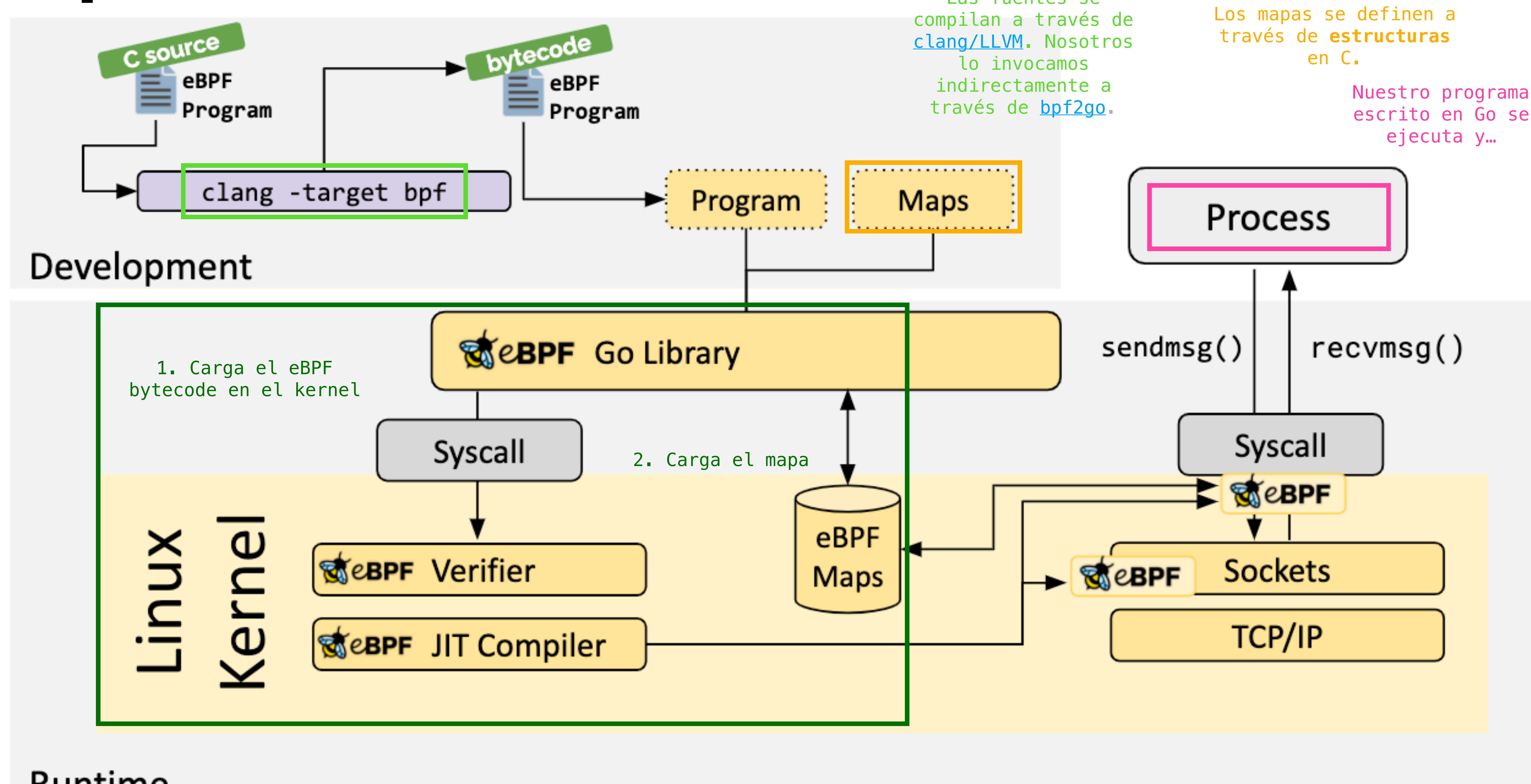
Fuente

Un poco más sobre eBPF



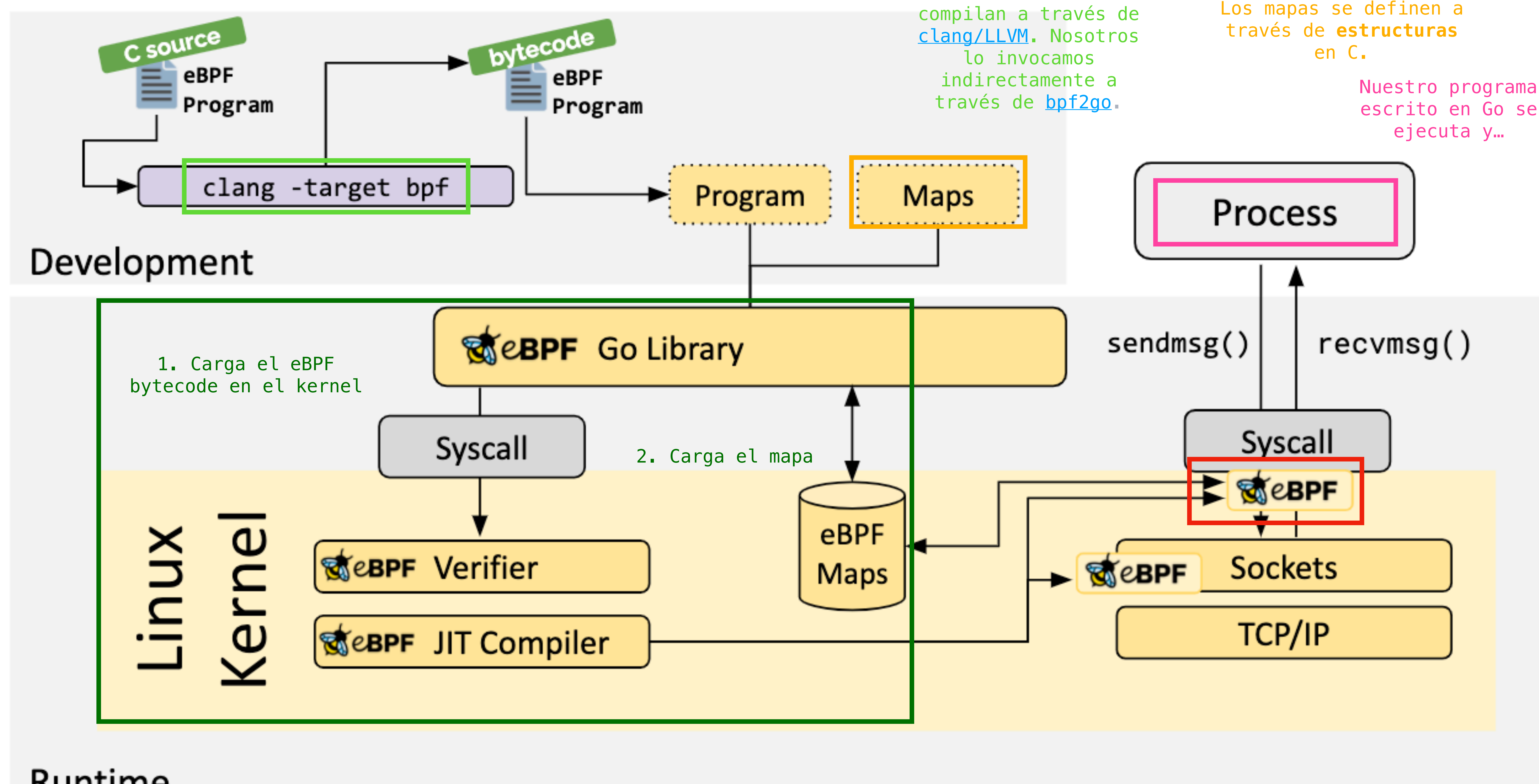
Fuente

Un poco más sobre eBPF



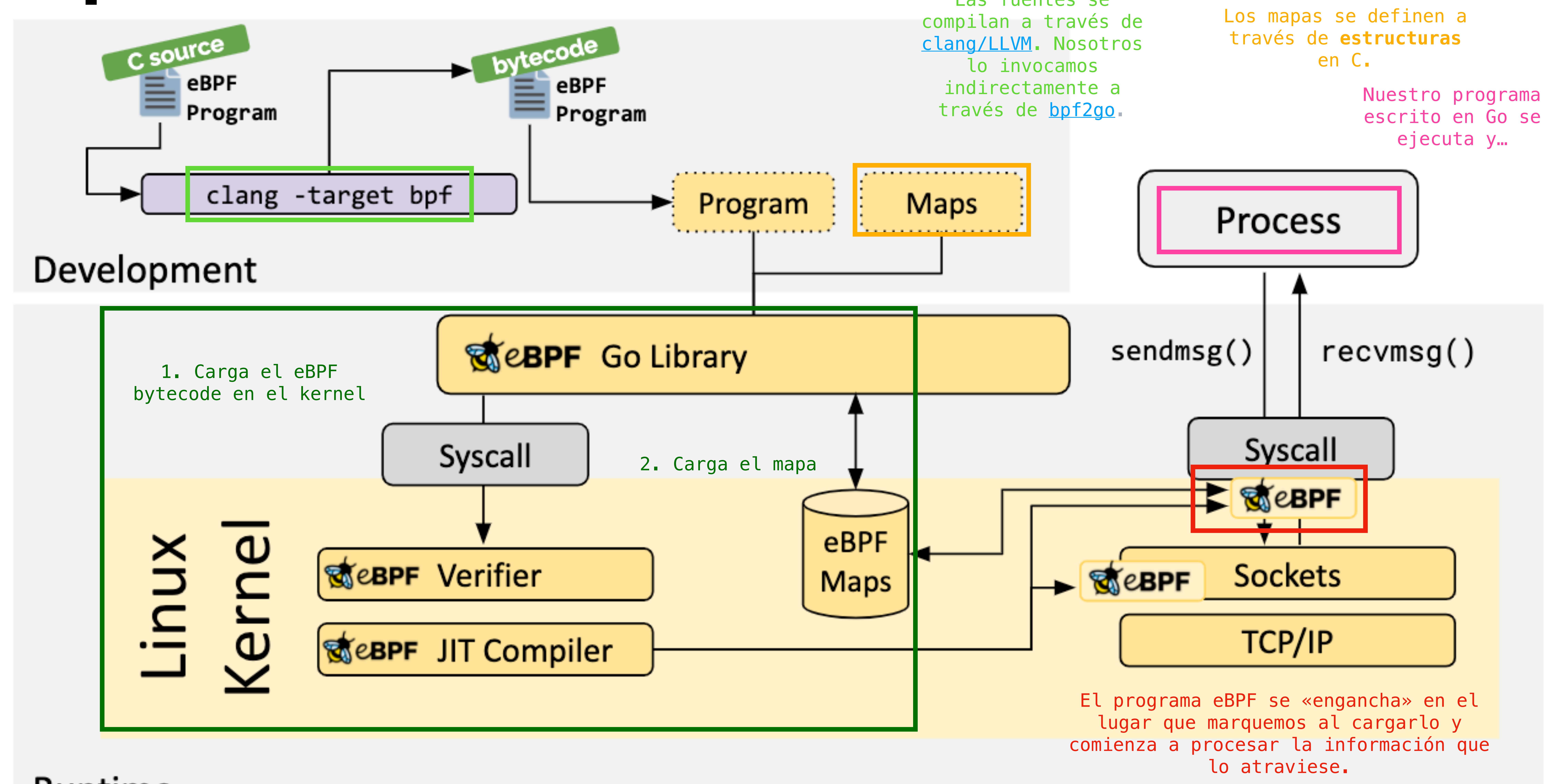
Fuente

Un poco más sobre eBPF



Fuente

Un poco más sobre eBPF

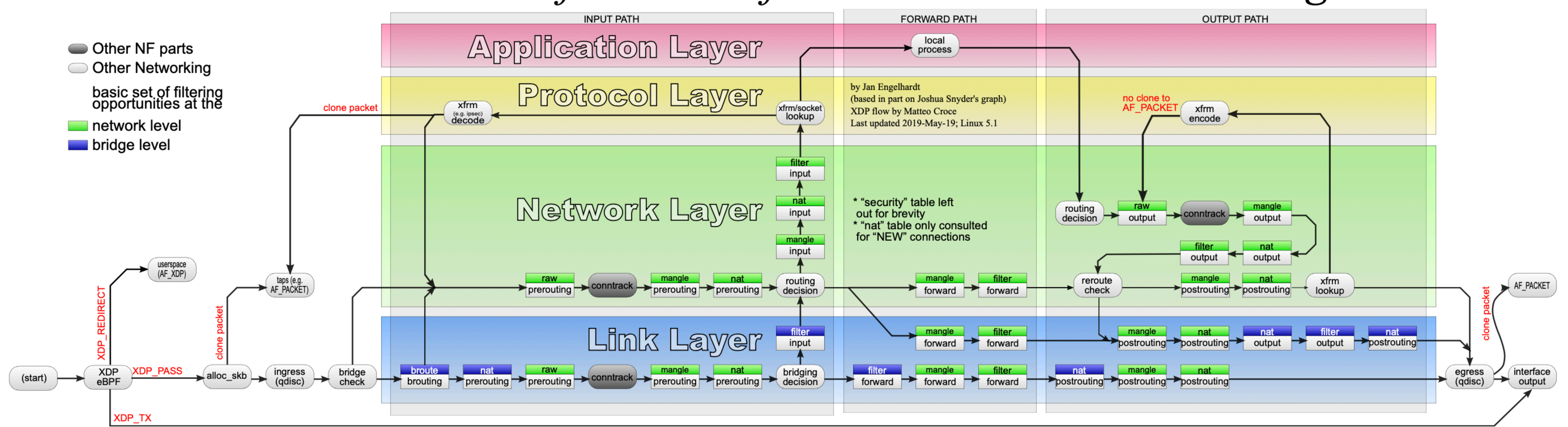


Fuente

eXpress Data Path

- El objetivo es que los programas eBPF puedan actuar pronto.
- Para ello se define el [`eXpress Data Path`]() (XDP):
  - Lo primero que se «encuentra» una trama al salir de la NIC es un programa eBPF.
  - Los programas eBPF deciden el destino del paquete (hay más opciones):
    - **`XDP_PASS`**: Dejamos pasar el paquete a la pila de red.
    - **`XDP_DROP`**: Descartamos el paquete.
    - **`XDP_REDIRECT`**: Redirigimos el paquete a otra NIC o a un socket.

Packet flow in Netfilter and General Networking



[Fuente]()

Nuestro ejemplo: C

- Se basa enormemente en uno de `cilium/ebpf`.
- Llevamos la cuenta de la cantidad de datagramas por `IPv4`.
- También descartamos datagramas provenientes de `192.168.4.3`.
- Ojo con el *endianness* de los datos…

```c
/* Define an LRU hash map for storing packet count by source IPv4 address */
struct {
    __uint(type, BPF_MAP_TYPE_LRU_HASH);
    __uint(max_entries, MAX_MAP_ENTRIES);
    __type(key, __u32);                    // Source IPv4 address
    __type(value, __u32);                  // Packet count
} xdp_stats_map SEC(".maps");

// 192.168.4.3: 192 == 0xc0; 168 == 0xa8; 4 == 0x04; 3 == 0x03
#define BANNED_IP 0xc0a80403

SEC("xdp")
int xdp_prog_func(struct xdp_md *ctx) {
  __u32 ip;

  /* Parse the IPv4 address: ctx offers access to the raw link-layer frame */
  parse_wg_ip_src_addr(ctx, &ip);

  /* Retrieve current count for the IPv4 address stored on ip */
  __u32* pkt_count = bpf_map_lookup_elem(&xdp_stats_map, &ip);
  if (!pkt_count) {
    /* No entry existed: initialize a new one to 1 */
    __u32 init_pkt_count = 1;
    bpf_map_update_elem(&xdp_stats_map, &ip, &init_pkt_count, BPF_ANY);
  } else {
    /* Otherwise incrment the existing entry by one */
      __sync_fetch_and_add(pkt_count, 1);
  }

  /* If the source IP is the banned one, drop the packet */
  if (ip == bpf_htonl(BANNED_IP))
        return XDP_DROP;

  /* Otherwise let it through to the kernel's network stack */
  return XDP_PASS;
}
```

Nuestro ejemplo: C

- Se basa enormemente en uno de `cilium/ebpf`.
- Llevamos la cuenta de la cantidad de datagramas por `IPv4`.
- También descartamos datagramas provenientes de `192.168.4.3`.
- Ojo con el *endianness* de los datos…

```
/* Define an LRU hash map for storing packet count by source IPv4 address */
struct {
    __uint(type, BPF_MAP_TYPE_LRU_HASH);
    __uint(max_entries, MAX_MAP_ENTRIES);
    __type(key, __u32);                 // Source IPv4 address
    __type(value, __u32);               // Packet count
} xdp_stats_map SEC(".maps");

// 192.168.4.3: 192 == 0xc0; 168 == 0xa8; 4 == 0x04; 3 == 0x03
#define BANNED_IP 0xc0a80403

SEC("xdp")
int xdp_prog_func(struct xdp_md *ctx) {
  __u32 ip;

  /* Parse the IPv4 address: ctx offers access to the raw link-layer frame */
  parse_wg_ip_src_addr(ctx, &ip);

  /* Retrieve current count for the IPv4 address stored on ip */
  __u32* pkt_count = bpf_map_lookup_elem(&xdp_stats_map, &ip);
  if (!pkt_count) {
    /* No entry existed: initialize a new one to 1 */
    __u32 init_pkt_count = 1;
    bpf_map_update_elem(&xdp_stats_map, &ip, &init_pkt_count, BPF_ANY);
  } else {
    /* Otherwise incrment the existing entry by one */
      __sync_fetch_and_add(pkt_count, 1);
  }

  /* If the source IP is the banned one, drop the packet */
  if (ip == bpf_htonl(BANNED_IP))
        return XDP_DROP;

  /* Otherwise let it through to the kernel's network stack */
  return XDP_PASS;
}
```

Nuestro ejemplo: C

- Se basa enormemente en uno de `cilium/ebpf`.
- Llevamos la cuenta de la cantidad de datagramas por `IPv4`.
- También descartamos datagramas provenientes de `192.168.4.3`.
- Ojo con el *endianness* de los datos…

```
/* Define an LRU hash map for storing packet count by source IPv4 address */
struct {
    __uint(type, BPF_MAP_TYPE_LRU_HASH);   ¡Los rectángulos contienen enlaces!
    __uint(max_entries, MAX_MAP_ENTRIES);
    __type(key, __u32);                    // Source IPv4 address
    __type(value, __u32);                  // Packet count
} xdp_stats_map SEC(".maps");

// 192.168.4.3: 192 == 0xc0; 168 == 0xa8; 4 == 0x04; 3 == 0x03
#define BANNED_IP 0xc0a80403

SEC("xdp")
int xdp_prog_func(struct xdp_md *ctx) {
  __u32 ip;

  /* Parse the IPv4 address: ctx offers access to the raw link-layer frame */
  parse_wg_ip_src_addr(ctx, &ip);

  /* Retrieve current count for the IPv4 address stored on ip */
  __u32* pkt_count = bpf_map_lookup_elem(&xdp_stats_map, &ip);
  if (!pkt_count) {
    /* No entry existed: initialize a new one to 1 */
    __u32 init_pkt_count = 1;
    bpf_map_update_elem(&xdp_stats_map, &ip, &init_pkt_count, BPF_ANY);
  } else {
    /* Otherwise incrment the existing entry by one */
      __sync_fetch_and_add(pkt_count, 1);
  }

  /* If the source IP is the banned one, drop the packet */
  if (ip == bpf_htonl(BANNED_IP))
        return XDP_DROP;

  /* Otherwise let it through to the kernel's network stack */
  return XDP_PASS;
}
```

Nuestro ejemplo: C

- Se basa enormemente en uno de `cilium/ebpf`.
- Llevamos la cuenta de la cantidad de datagramas por `IPv4`.
- También descartamos datagramas provenientes de `192.168.4.3`.
- Ojo con el *endianness* de los datos…

```c
/* Define an LRU hash map for storing packet count by source IPv4 address */
struct {
    __uint(type, BPF_MAP_TYPE_LRU_HASH);    ¡Los rectángulos contienen enlaces!
    __uint(max_entries, MAX_MAP_ENTRIES);
    __type(key, __u32);                     // Source IPv4 address
    __type(value, __u32);                   // Packet count
} xdp_stats_map SEC(".maps");

// 192.168.4.3: 192 == 0xc0; 168 == 0xa8; 4 == 0x04; 3 == 0x03
#define BANNED_IP 0xc0a80403

SEC("xdp")
int xdp_prog_func(struct xdp_md *ctx) {
  __u32 ip;

  /* Parse the IPv4 address: ctx offers access to the raw link-layer frame */
  parse_wg_ip_src_addr(ctx, &ip);

  /* Retrieve current count for the IPv4 address stored on ip */
  __u32* pkt_count = bpf_map_lookup_elem(&xdp_stats_map, &ip);
  if (!pkt_count) {
    /* No entry existed: initialize a new one to 1 */
    __u32 init_pkt_count = 1;
    bpf_map_update_elem(&xdp_stats_map, &ip, &init_pkt_count, BPF_ANY);
  } else {
    /* Otherwise incrment the existing entry by one */
      __sync_fetch_and_add(pkt_count, 1);
  }

  /* If the source IP is the banned one, drop the packet */
  if (ip == bpf_htonl(BANNED_IP))
        return XDP_DROP;

  /* Otherwise let it through to the kernel's network stack */
  return XDP_PASS;
}
```

Nuestro ejemplo: C

- Se basa enormemente en uno de cilium/ebpf.
- Llevamos la cuenta de la cantidad de datagramas por IPv4.
- También descartamos datagramas provenientes de 192.168.4.3.
- Ojo con el *endianness* de los datos…

¡Los rectángulos contienen enlaces!

```c
/* Define an LRU hash map for storing packet count by source IPv4 address */
struct {
    __uint(type, BPF_MAP_TYPE_LRU_HASH);
    __uint(max_entries, MAX_MAP_ENTRIES);
    __type(key, __u32);                     // Source IPv4 address
    __type(value, __u32);                   // Packet count
} xdp_stats_map SEC(".maps");

// 192.168.4.3: 192 == 0xc0; 168 == 0xa8; 4 == 0x04; 3 == 0x03
#define BANNED_IP 0xc0a80403

SEC("xdp")
int xdp_prog_func(struct xdp_md *ctx) {
  __u32 ip;

  /* Parse the IPv4 address: ctx offers access to the raw link-layer frame */
  parse_wg_ip_src_addr(ctx, &ip);

  /* Retrieve current count for the IPv4 address stored on ip */
  __u32* pkt_count = bpf_map_lookup_elem(&xdp_stats_map, &ip);
  if (!pkt_count) {
    /* No entry existed: initialize a new one to 1 */
    __u32 init_pkt_count = 1;
    bpf_map_update_elem(&xdp_stats_map, &ip, &init_pkt_count, BPF_ANY);
  } else {
    /* Otherwise incrment the existing entry by one */
      __sync_fetch_and_add(pkt_count, 1);
  }

  /* If the source IP is the banned one, drop the packet */
  if (ip == bpf_htonl(BANNED_IP))
        return XDP_DROP;

  /* Otherwise let it through to the kernel's network stack */
  return XDP_PASS;
}
```

Nuestro ejemplo: C

- Se basa enormemente en uno de cilium/ebpf.
- Llevamos la cuenta de la cantidad de datagramas por IPv4.
- También descartamos datagramas provenientes de 192.168.4.3.
- Ojo con el *endianness* de los datos…

¡Los rectángulos contienen enlaces!

```c
/* Define an LRU hash map for storing packet count by source IPv4 address */
struct {
    __uint(type, BPF_MAP_TYPE_LRU_HASH);
    __uint(max_entries, MAX_MAP_ENTRIES);
    __type(key, __u32);                     // Source IPv4 address
    __type(value, __u32);                   // Packet count
} xdp_stats_map SEC(".maps");

// 192.168.4.3: 192 == 0xc0; 168 == 0xa8; 4 == 0x04; 3 == 0x03
#define BANNED_IP 0xc0a80403

SEC("xdp")
int xdp_prog_func(struct xdp_md *ctx) {
  __u32 ip;

  /* Parse the IPv4 address: ctx offers access to the raw link-layer frame */
  parse_wg_ip_src_addr(ctx, &ip);

  /* Retrieve current count for the IPv4 address stored on ip */
  __u32* pkt_count = bpf_map_lookup_elem(&xdp_stats_map, &ip);
  if (!pkt_count) {
    /* No entry existed: initialize a new one to 1 */
    __u32 init_pkt_count = 1;
    bpf_map_update_elem(&xdp_stats_map, &ip, &init_pkt_count, BPF_ANY);
  } else {
    /* Otherwise incrment the existing entry by one */
      __sync_fetch_and_add(pkt_count, 1);
  }

  /* If the source IP is the banned one, drop the packet */
  if (ip == bpf_htonl(BANNED_IP))
        return XDP_DROP;

  /* Otherwise let it through to the kernel's network stack */
  return XDP_PASS;
}
```

Nuestro ejemplo: C

- Se basa enormemente en uno de `cilium/ebpf`.
- Llevamos la cuenta de la cantidad de datagramas por `IPv4`.
- También descartamos datagramas provenientes de `192.168.4.3`.
- Ojo con el *endianness* de los datos…

```c
/* Define an LRU hash map for storing packet count by source IPv4 address */
struct {
    __uint(type, BPF_MAP_TYPE_LRU_HASH);   ¡Los rectángulos contienen enlaces!
    __uint(max_entries, MAX_MAP_ENTRIES);
    __type(key, __u32);                    // Source IPv4 address
    __type(value, __u32);                  // Packet count
} xdp_stats_map SEC(".maps");

// 192.168.4.3: 192 == 0xc0; 168 == 0xa8; 4 == 0x04; 3 == 0x03
#define BANNED_IP 0xc0a80403

SEC("xdp")
int xdp_prog_func(struct xdp_md *ctx) {
  __u32 ip;

  /* Parse the IPv4 address: ctx offers access to the raw link-layer frame */
  parse_wg_ip_src_addr(ctx, &ip);

  /* Retrieve current count for the IPv4 address stored on ip */
  __u32* pkt_count = bpf_map_lookup_elem(&xdp_stats_map, &ip);
  if (!pkt_count) {
    /* No entry existed: initialize a new one to 1 */
    __u32 init_pkt_count = 1;
    bpf_map_update_elem(&xdp_stats_map, &ip, &init_pkt_count, BPF_ANY);
  } else {
    /* Otherwise incrment the existing entry by one */
     __sync_fetch_and_add(pkt_count, 1);
  }

  /* If the source IP is the banned one, drop the packet */
  if (ip == bpf_htonl(BANNED_IP))
        return XDP_DROP;

  /* Otherwise let it through to the kernel's network stack */
  return XDP_PASS;
}
```

Nuestro ejemplo: C

- Se basa enormemente en uno de [`cilium/ebpf`].
- Llevamos la cuenta de la cantidad de datagramas por `IPv4`.
- También descartamos datagramas provenientes de `192.168.4.3`.
- Ojo con el *endianness* de los datos…

Esta instrucción es un *builtin* de LLVM que se traduce a `BPF_STX | BPF_ATOMIC | BPF_ADD | BPF_W` en base al set de instrucciones de eBPF. Fuente.

```c
/* Define an LRU hash map for storing packet count by source IPv4 address */
struct {
    __uint(type, BPF_MAP_TYPE_LRU_HASH);   ¡Los rectángulos contienen enlaces!
    __uint(max_entries, MAX_MAP_ENTRIES);
    __type(key, __u32);                    // Source IPv4 address
    __type(value, __u32);                  // Packet count
} xdp_stats_map SEC(".maps");

// 192.168.4.3: 192 == 0xc0; 168 == 0xa8; 4 == 0x04; 3 == 0x03
#define BANNED_IP 0xc0a80403

SEC("xdp")
int xdp_prog_func(struct xdp_md *ctx) {
  __u32 ip;

  /* Parse the IPv4 address: ctx offers access to the raw link-layer frame */
  parse_wg_ip_src_addr(ctx, &ip);

  /* Retrieve current count for the IPv4 address stored on ip */
  __u32* pkt_count = bpf_map_lookup_elem(&xdp_stats_map, &ip);
  if (!pkt_count) {
    /* No entry existed: initialize a new one to 1 */
    __u32 init_pkt_count = 1;
    bpf_map_update_elem(&xdp_stats_map, &ip, &init_pkt_count, BPF_ANY);
  } else {
    /* Otherwise incrment the existing entry by one */
     __sync_fetch_and_add(pkt_count, 1);
  }

  /* If the source IP is the banned one, drop the packet */
  if (ip == bpf_htonl(BANNED_IP))
        return XDP_DROP;

  /* Otherwise let it through to the kernel's network stack */
  return XDP_PASS;
}
```

Nuestro ejemplo: C

- Se basa enormemente en uno de [cilium/ebpf](#).
- Llevamos la cuenta de la cantidad de datagramas por `IPv4`.
- También descartamos datagramas provenientes de `192.168.4.3`.
- Ojo con el [*endianness*](#) de los datos…

Esta instrucción es un [*builtin*](#) de LLVM que se traduce a BPF_STX | BPF_ATOMIC | BPF_ADD | BPF_W en base al [set de instrucciones](#) de eBPF. [Fuente](#).

```c
/* Define an LRU hash map for storing packet count by source IPv4 address */
struct {
    __uint(type, BPF_MAP_TYPE_LRU_HASH);   ¡Los rectángulos contienen enlaces!
    __uint(max_entries, MAX_MAP_ENTRIES);
    __type(key, __u32);                    // Source IPv4 address
    __type(value, __u32);                  // Packet count
} xdp_stats_map SEC(".maps");

// 192.168.4.3: 192 == 0xc0; 168 == 0xa8; 4 == 0x04; 3 == 0x03
#define BANNED_IP 0xc0a80403

SEC("xdp")
int xdp_prog_func(struct xdp_md *ctx) {
  __u32 ip;

  /* Parse the IPv4 address: ctx offers access to the raw link-layer frame */
  parse_wg_ip_src_addr(ctx, &ip);

  /* Retrieve current count for the IPv4 address stored on ip */
  __u32* pkt_count = bpf_map_lookup_elem(&xdp_stats_map, &ip);
  if (!pkt_count) {
    /* No entry existed: initialize a new one to 1 */
    __u32 init_pkt_count = 1;
    bpf_map_update_elem(&xdp_stats_map, &ip, &init_pkt_count, BPF_ANY);
  } else {
    /* Otherwise incrment the existing entry by one */
    __sync_fetch_and_add(pkt_count, 1);
  }

  /* If the source IP is the banned one, drop the packet */
  if (ip == bpf_htonl(BANNED_IP))
      return XDP_DROP;

  /* Otherwise let it through to the kernel's network stack */
  return XDP_PASS;
}
```

Nuestro ejemplo: Go

- En Go escribimos el código que:
  - Compila el programa en C a bytecode de eBPF.
  - Carga este *`bytecode`* en el kernel.
  - Lo «engancha» al comienzo del `XDP`.
  - Lee el mapa para conocer el conteo de `IPv4`s.
  - Descarga el *`bytecode`* de eBPF al finalizar.
- Hemos automatizado el proceso en un `Makefile`: **`make run`** se ocupa de todo.

```go
//go:generate go run bpf2go -cc $BPF_CLANG -cflags $BPF_CFLAGS bpf xdp.c -- -I./headers

func main() {
    // Look up the network interface by name.
    iface, err := net.InterfaceByName(os.Args[1])

    // Load pre-compiled programs into the kernel.
    objs := bpfObjects{}
    loadBpfObjects(&objs, nil)
    defer objs.Close()

    // Attach the program.
    l, _ := link.AttachXDP(link.XDPOptions{
        Program:   objs.XdpProgFunc,
        Interface: iface.Index,
    })
    defer l.Close()

    // Print the contents of the BPF hash map (source IP address -> packet count).
    ticker := time.NewTicker(1 * time.Second)
    defer ticker.Stop()
    for range ticker.C {
        s, _ := formatMapContents(objs.XdpStatsMap)
        log.Printf("Map contents:\n%s", s)
    }
}

func formatMapContents(m *ebpf.Map) (string, error) {
    var (sb strings.Builder; key []byte; val uint32)
    iter := m.Iterate()
    for iter.Next(&key, &val) {
        // The key is the source IPv4 address!
        sb.WriteString(fmt.Sprintf("\t%s => %d\n", net.IP(key), val))
    }
    return sb.String(), iter.Err()
}
```

Nuestro ejemplo: Go

- En Go escribimos el código que:
  - Compila el programa en C a bytecode de eBPF.
  - Carga este *`bytecode`* en el kernel.
  - Lo «engancha» al comienzo del `XDP`.
  - Lee el mapa para conocer el conteo de `IPv4s`.
  - Descarga el *`bytecode`* de eBPF al finalizar.
- Hemos automatizado el proceso en un `Makefile`: **`make run`** se ocupa de todo.

```go
//go:generate go run bpf2go -cc $BPF_CLANG -cflags $BPF_CFLAGS bpf xdp.c -- -I./headers

func main() {
    // Look up the network interface by name.
    iface, err := net.InterfaceByName(os.Args[1])

    // Load pre-compiled programs into the kernel.
    objs := bpfObjects{}
    loadBpfObjects(&objs, nil)
    defer objs.Close()

    // Attach the program.
    l, _ := link.AttachXDP(link.XDPOptions{
        Program:   objs.XdpProgFunc,
        Interface: iface.Index,
    })
    defer l.Close()

    // Print the contents of the BPF hash map (source IP address -> packet count).
    ticker := time.NewTicker(1 * time.Second)
    defer ticker.Stop()
    for range ticker.C {
        s, _ := formatMapContents(objs.XdpStatsMap)
        log.Printf("Map contents:\n%s", s)
    }
}

func formatMapContents(m *ebpf.Map) (string, error) {
    var (sb strings.Builder; key []byte; val uint32)
    iter := m.Iterate()
    for iter.Next(&key, &val) {
        // The key is the source IPv4 address!
        sb.WriteString(fmt.Sprintf("\t%s => %d\n", net.IP(key), val))
    }
    return sb.String(), iter.Err()
}
```

Nuestro ejemplo: Go

- En Go escribimos el código que:
  - Compila el programa en C a bytecode de eBPF.
  - Carga este *bytecode* en el kernel.
  - Lo «engancha» al comienzo del XDP.
  - Lee el mapa para conocer el conteo de IPv4s.
  - Descarga el *bytecode* de eBPF al finalizar.
- Hemos automatizado el proceso en un Makefile: **`make run`** se ocupa de todo.

```go
//go:generate go run bpf2go -cc $BPF_CLANG -cflags $BPF_CFLAGS bpf xdp.c -- -I./headers

func main() {
    // Look up the network interface by name.
    iface, err := net.InterfaceByName(os.Args[1])

    // Load pre-compiled programs into the kernel.
    objs := bpfObjects{}
    loadBpfObjects(&objs, nil)
    defer objs.Close()

    // Attach the program.
    l, _ := link.AttachXDP(link.XDPOptions{
        Program:   objs.XdpProgFunc,
        Interface: iface.Index,
    })
    defer l.Close()

    // Print the contents of the BPF hash map (source IP address -> packet count).
    ticker := time.NewTicker(1 * time.Second)
    defer ticker.Stop()
    for range ticker.C {
        s, _ := formatMapContents(objs.XdpStatsMap)
        log.Printf("Map contents:\n%s", s)
    }
}

func formatMapContents(m *ebpf.Map) (string, error) {
    var (sb strings.Builder; key []byte; val uint32)
    iter := m.Iterate()
    for iter.Next(&key, &val) {
        // The key is the source IPv4 address!
        sb.WriteString(fmt.Sprintf("\t%s => %d\n", net.IP(key), val))
    }
    return sb.String(), iter.Err()
}
```

Esta orden compila el *bytecode* eBPF y genera el código *.go para su carga

Nuestro ejemplo: Go

- En Go escribimos el código que:
  - Compila el programa en C a bytecode de eBPF.
  - Carga este *bytecode* en el kernel.
  - Lo «engancha» al comienzo del XDP.
  - Lee el mapa para conocer el conteo de IPv4s.
  - Descarga el *bytecode* de eBPF al finalizar.
- Hemos automatizado el proceso en un Makefile: **make run** se ocupa de todo.

```go
//go:generate go run bpf2go -cc $BPF_CLANG -cflags $BPF_CFLAGS bpf xdp.c -- -I./headers

func main() {
    // Look up the network interface by name.
    iface, err := net.InterfaceByName(os.Args[1])

    // Load pre-compiled programs into the kernel.
    objs := bpfObjects{}
    loadBpfObjects(&objs, nil)
    defer objs.Close()

    // Attach the program.
    l, _ := link.AttachXDP(link.XDPOptions{
        Program:   objs.XdpProgFunc,
        Interface: iface.Index,
    })
    defer l.Close()

    // Print the contents of the BPF hash map (source IP address -> packet count).
    ticker := time.NewTicker(1 * time.Second)
    defer ticker.Stop()
    for range ticker.C {
        s, _ := formatMapContents(objs.XdpStatsMap)
        log.Printf("Map contents:\n%s", s)
    }
}

func formatMapContents(m *ebpf.Map) (string, error) {
    var (sb strings.Builder; key []byte; val uint32)
    iter := m.Iterate()
    for iter.Next(&key, &val) {
        // The key is the source IPv4 address!
        sb.WriteString(fmt.Sprintf("\t%s => %d\n", net.IP(key), val))
    }
    return sb.String(), iter.Err()
}
```

Esta orden compila el *bytecode* eBPF y genera el código *.go para su carga

Nuestro ejemplo: Go

- En Go escribimos el código que:
  - Compila el programa en C a bytecode de eBPF.
  - Carga este *bytecode* en el kernel.
  - Lo «engancha» al comienzo del XDP.
  - Lee el mapa para conocer el conteo de IPv4s.
  - Descarga el *bytecode* de eBPF al finalizar.
- Hemos automatizado el proceso en un [Makefile]: **make run** se ocupa de todo.

```go
//go:generate go run bpf2go -cc $BPF_CLANG -cflags $BPF_CFLAGS bpf xdp.c -- -I./headers

func main() {
    // Look up the network interface by name.
    iface, err := net.InterfaceByName(os.Args[1])

    // Load pre-compiled programs into the kernel.
    objs := bpfObjects{}
    loadBpfObjects(&objs, nil)
    defer objs.Close()

    // Attach the program.
    l, _ := link.AttachXDP(link.XDPOptions{
        Program:   objs.XdpProgFunc,
        Interface: iface.Index,
    })
    defer l.Close()

    // Print the contents of the BPF hash map (source IP address -> packet count).
    ticker := time.NewTicker(1 * time.Second)
    defer ticker.Stop()
    for range ticker.C {
        s, _ := formatMapContents(objs.XdpStatsMap)
        log.Printf("Map contents:\n%s", s)
    }
}

func formatMapContents(m *ebpf.Map) (string, error) {
    var (sb strings.Builder; key []byte; val uint32)
    iter := m.Iterate()
    for iter.Next(&key, &val) {
        // The key is the source IPv4 address!
        sb.WriteString(fmt.Sprintf("\t%s => %d\n", net.IP(key), val))
    }
    return sb.String(), iter.Err()
}
```

Esta orden compila el *bytecode* eBPF y genera el código *.go para su carga

Nuestro ejemplo: Go

- En Go escribimos el código que:
  - Compila el programa en C a bytecode de eBPF.
  - Carga este *`bytecode`* en el kernel.
  - Lo «engancha» al comienzo del `XDP`.
  - Lee el mapa para conocer el conteo de `IPv4`s.
  - Descarga el *`bytecode`* de eBPF al finalizar.
- Hemos automatizado el proceso en un `Makefile`: **`make run`** se ocupa de todo.

```go
//go:generate go run bpf2go -cc $BPF_CLANG -cflags $BPF_CFLAGS bpf xdp.c -- -I./headers

func main() {
    // Look up the network interface by name.
    iface, err := net.InterfaceByName(os.Args[1])

    // Load pre-compiled programs into the kernel.
    objs := bpfObjects{}
    loadBpfObjects(&objs, nil)
    defer objs.Close()

    // Attach the program.
    l, _ := link.AttachXDP(link.XDPOptions{
        Program:   objs.XdpProgFunc,
        Interface: iface.Index,
    })
    defer l.Close()

    // Print the contents of the BPF hash map (source IP address -> packet count).
    ticker := time.NewTicker(1 * time.Second)
    defer ticker.Stop()
    for range ticker.C {
        s, _ := formatMapContents(objs.XdpStatsMap)
        log.Printf("Map contents:\n%s", s)
    }
}

func formatMapContents(m *ebpf.Map) (string, error) {
    var (sb strings.Builder; key []byte; val uint32)
    iter := m.Iterate()
    for iter.Next(&key, &val) {
        // The key is the source IPv4 address!
        sb.WriteString(fmt.Sprintf("\t%s => %d\n", net.IP(key), val))
    }
    return sb.String(), iter.Err()
}
```

Esta orden compila el *bytecode* eBPF y genera el código *.go para su carga

Nuestro ejemplo: Go

- En Go escribimos el código que:
  - Compila el programa en C a bytecode de eBPF.
  - Carga este *bytecode* en el kernel.
  - Lo «engancha» al comienzo del XDP.
  - Lee el mapa para conocer el conteo de IPv4s.
  - Descarga el *bytecode* de eBPF al finalizar.
- Hemos automatizado el proceso en un Makefile: **make run** se ocupa de todo.

```go
//go:generate go run bpf2go -cc $BPF_CLANG -cflags $BPF_CFLAGS bpf xdp.c -- -I./headers

func main() {
    // Look up the network interface by name.
    iface, err := net.InterfaceByName(os.Args[1])

    // Load pre-compiled programs into the kernel.
    objs := bpfObjects{}
    loadBpfObjects(&objs, nil)
    defer objs.Close()

    // Attach the program.
    l, _ := link.AttachXDP(link.XDPOptions{
        Program:   objs.XdpProgFunc,
        Interface: iface.Index,
    })
    defer l.Close()

    // Print the contents of the BPF hash map (source IP address -> packet count).
    ticker := time.NewTicker(1 * time.Second)
    defer ticker.Stop()
    for range ticker.C {
        s, _ := formatMapContents(objs.XdpStatsMap)
        log.Printf("Map contents:\n%s", s)
    }
}

func formatMapContents(m *ebpf.Map) (string, error) {
    var (sb strings.Builder; key []byte; val uint32)
    iter := m.Iterate()
    for iter.Next(&key, &val) {
        // The key is the source IPv4 address!
        sb.WriteString(fmt.Sprintf("\t%s => %d\n", net.IP(key), val))
    }
    return sb.String(), iter.Err()
}
```

Esta orden compila el *bytecode* eBPF y genera el código *.go para su carga

Nuestro ejemplo: Go

- En Go escribimos el código que:
  - Compila el programa en C a bytecode de eBPF.
  - Carga este *bytecode* en el kernel.
  - Lo «engancha» al comienzo del XDP.
  - Lee el mapa para conocer el conteo de IPv4s.
  - Descarga el *bytecode* de eBPF al finalizar.
- Hemos automatizado el proceso en un `Makefile`: **`make run`** se ocupa de todo.

```go
//go:generate go run bpf2go -cc $BPF_CLANG -cflags $BPF_CFLAGS bpf xdp.c -- -I./headers

func main() {
    // Look up the network interface by name.
    iface, err := net.InterfaceByName(os.Args[1])

    // Load pre-compiled programs into the kernel.
    objs := bpfObjects{}
    loadBpfObjects(&objs, nil)
    defer objs.Close()

    // Attach the program.
    l, _ := link.AttachXDP(link.XDPOptions{
        Program:   objs.XdpProgFunc,
        Interface: iface.Index,
    })
    defer l.Close()

    // Print the contents of the BPF hash map (source IP address -> packet count).
    ticker := time.NewTicker(1 * time.Second)
    defer ticker.Stop()
    for range ticker.C {
        s, _ := formatMapContents(objs.XdpStatsMap)
        log.Printf("Map contents:\n%s", s)
    }
}

func formatMapContents(m *ebpf.Map) (string, error) {
    var (sb strings.Builder; key []byte; val uint32)
    iter := m.Iterate()
    for iter.Next(&key, &val) {
        // The key is the source IPv4 address!
        sb.WriteString(fmt.Sprintf("\t%s => %d\n", net.IP(key), val))
    }
    return sb.String(), iter.Err()
}
```

Esta orden compila el *bytecode* eBPF y genera el código *.go para su carga

Si te pierdes… ¡mira el mapa!

```
vagrant@vpn-core:~$ sudo ./wg-ebpf wg0
2022/10/10 15:00:02 Attached XDP program to iface "wg0" (index 6)
2022/10/10 15:00:02 Press Ctrl-C to exit and remove the program
2022/10/10 15:00:12 Map contents:
        192.168.4.2 => 1
2022/10/10 15:00:13 Map contents:
        192.168.4.2 => 2
2022/10/10 15:00:14 Map contents:
        192.168.4.2 => 3
2022/10/10 15:00:17 Map contents:
        192.168.4.2 => 4
2022/10/10 15:00:18 Map contents:
        192.168.4.3 => 1
        192.168.4.2 => 5
2022/10/10 15:00:19 Map contents:
        192.168.4.3 => 2
        192.168.4.2 => 6
2022/10/10 15:00:20 Map contents:
        192.168.4.3 => 3
        192.168.4.2 => 6
^C
```

```
vagrant@client-a:~$ ping -c 3 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=64 time=503 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=64 time=501 ms

--- 192.168.4.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 501.371/502.023/503.220/0.847 ms
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
```

Si te pierdes… ¡mira el mapa!

```
vagrant@vpn-core:~$ sudo ./wg-ebpf wg0
2022/10/10 15:00:02 Attached XDP program to iface "wg0" (index 6)
2022/10/10 15:00:02 Press Ctrl-C to exit and remove the program
2022/10/10 15:00:12 Map contents:
        192.168.4.2 => 1
2022/10/10 15:00:13 Map contents:
        192.168.4.2 => 2
2022/10/10 15:00:14 Map contents:
        192.168.4.2 => 3
2022/10/10 15:00:17 Map contents:
        192.168.4.2 => 4
2022/10/10 15:00:18 Map contents:
        192.168.4.3 => 1
        192.168.4.2 => 5
2022/10/10 15:00:19 Map contents:
        192.168.4.3 => 2
        192.168.4.2 => 6
2022/10/10 15:00:20 Map contents:
        192.168.4.3 => 3
        192.168.4.2 => 6
^C
```

```
vagrant@client-a:~$ ping -c 3 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=64 time=503 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=64 time=501 ms

--- 192.168.4.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 501.371/502.023/503.220/0.847 ms
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
```

Si te pierdes… ¡mira el mapa!

```
vagrant@vpn-core:~$ sudo ./wg-ebpf wg0
2022/10/10 15:00:02 Attached XDP program to iface "wg0" (index 6)
2022/10/10 15:00:02 Press Ctrl-C to exit and remove the program
2022/10/10 15:00:12 Map contents:
        192.168.4.2 => 1
2022/10/10 15:00:13 Map contents:
        192.168.4.2 => 2
2022/10/10 15:00:14 Map contents:
        192.168.4.2 => 3
2022/10/10 15:00:17 Map contents:
        192.168.4.2 => 4
2022/10/10 15:00:18 Map contents:
        192.168.4.3 => 1
        192.168.4.2 => 5
2022/10/10 15:00:19 Map contents:
        192.168.4.3 => 2
        192.168.4.2 => 6
2022/10/10 15:00:20 Map contents:
        192.168.4.3 => 3
        192.168.4.2 => 6
^C
```

```
vagrant@client-a:~$ ping -c 3 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=64 time=503 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=64 time=501 ms

--- 192.168.4.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 501.371/502.023/503.220/0.847 ms
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
```

Si te pierdes… ¡mira el mapa!

```
vagrant@vpn-core:~$ sudo ./wg-ebpf wg0
2022/10/10 15:00:02 Attached XDP program to iface "wg0" (index 6)
2022/10/10 15:00:02 Press Ctrl-C to exit and remove the program
2022/10/10 15:00:12 Map contents:
        192.168.4.2 => 1
2022/10/10 15:00:13 Map contents:
        192.168.4.2 => 2
2022/10/10 15:00:14 Map contents:
        192.168.4.2 => 3
2022/10/10 15:00:17 Map contents:
        192.168.4.2 => 4
2022/10/10 15:00:18 Map contents:
        192.168.4.3 => 1
        192.168.4.2 => 5
2022/10/10 15:00:19 Map contents:
        192.168.4.3 => 2
        192.168.4.2 => 6
2022/10/10 15:00:20 Map contents:
        192.168.4.3 => 3
        192.168.4.2 => 6
^C
```

```
vagrant@client-a:~$ ping -c 3 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=64 time=503 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=64 time=501 ms

--- 192.168.4.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 501.371/502.023/503.220/0.847 ms
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
```

Si te pierdes… ¡mira el mapa!

```
vagrant@vpn-core:~$ sudo ./wg-ebpf wg0
2022/10/10 15:00:02 Attached XDP program to iface "wg0" (index 6)
2022/10/10 15:00:02 Press Ctrl-C to exit and remove the program
2022/10/10 15:00:12 Map contents:
        192.168.4.2 => 1
2022/10/10 15:00:13 Map contents:
        192.168.4.2 => 2
2022/10/10 15:00:14 Map contents:
        192.168.4.2 => 3
2022/10/10 15:00:17 Map contents:
        192.168.4.2 => 4
2022/10/10 15:00:18 Map contents:
        192.168.4.3 => 1
        192.168.4.2 => 5
2022/10/10 15:00:19 Map contents:
        192.168.4.3 => 2
        192.168.4.2 => 6
2022/10/10 15:00:20 Map contents:
        192.168.4.3 => 3
        192.168.4.2 => 6
^C
```

```
vagrant@client-a:~$ ping -c 3 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=64 time=503 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=64 time=501 ms

--- 192.168.4.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 501.371/502.023/503.220/0.847 ms
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
```

Si te pierdes… ¡mira el mapa!

```
vagrant@vpn-core:~$ sudo ./wg-ebpf wg0
2022/10/10 15:00:02 Attached XDP program to iface "wg0" (index 6)
2022/10/10 15:00:02 Press Ctrl-C to exit and remove the program
2022/10/10 15:00:12 Map contents:
        192.168.4.2 => 1
2022/10/10 15:00:13 Map contents:
        192.168.4.2 => 2
2022/10/10 15:00:14 Map contents:
        192.168.4.2 => 3
2022/10/10 15:00:17 Map contents:
        192.168.4.2 => 4
2022/10/10 15:00:18 Map contents:
        192.168.4.3 => 1
        192.168.4.2 => 5
2022/10/10 15:00:19 Map contents:
        192.168.4.3 => 2
        192.168.4.2 => 6
2022/10/10 15:00:20 Map contents:
        192.168.4.3 => 3
        192.168.4.2 => 6
^C
```

```
vagrant@client-a:~$ ping -c 3 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=64 time=503 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=64 time=501 ms

--- 192.168.4.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 501.371/502.023/503.220/0.847 ms
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
```

Si te pierdes… ¡mira el mapa!

```
vagrant@vpn-core:~$ sudo ./wg-ebpf wg0
2022/10/10 15:00:02 Attached XDP program to iface "wg0" (index 6)
2022/10/10 15:00:02 Press Ctrl-C to exit and remove the program
2022/10/10 15:00:12 Map contents:
        192.168.4.2 => 1
2022/10/10 15:00:13 Map contents:
        192.168.4.2 => 2
2022/10/10 15:00:14 Map contents:
        192.168.4.2 => 3
2022/10/10 15:00:17 Map contents:
        192.168.4.2 => 4
2022/10/10 15:00:18 Map contents:
        192.168.4.3 => 1
        192.168.4.2 => 5
2022/10/10 15:00:19 Map contents:
        192.168.4.3 => 2
        192.168.4.2 => 6
2022/10/10 15:00:20 Map contents:
        192.168.4.3 => 3
        192.168.4.2 => 6
^C
```

```
vagrant@client-a:~$ ping -c 3 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=64 time=503 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=64 time=501 ms

--- 192.168.4.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 501.371/502.023/503.220/0.847 ms
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
```

Si te pierdes… ¡mira el mapa!

```
vagrant@vpn-core:~$ sudo ./wg-ebpf wg0
2022/10/10 15:00:02 Attached XDP program to iface "wg0" (index 6)
2022/10/10 15:00:02 Press Ctrl-C to exit and remove the program
2022/10/10 15:00:12 Map contents:
        192.168.4.2 => 1
2022/10/10 15:00:13 Map contents:
        192.168.4.2 => 2
2022/10/10 15:00:14 Map contents:
        192.168.4.2 => 3
2022/10/10 15:00:17 Map contents:
        192.168.4.2 => 4
2022/10/10 15:00:18 Map contents:
        192.168.4.3 => 1
        192.168.4.2 => 5
2022/10/10 15:00:19 Map contents:
        192.168.4.3 => 2
        192.168.4.2 => 6
2022/10/10 15:00:20 Map contents:
        192.168.4.3 => 3
        192.168.4.2 => 6
^C
```

```
vagrant@client-a:~$ ping -c 3 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=64 time=503 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=64 time=501 ms

--- 192.168.4.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 501.371/502.023/503.220/0.847 ms
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
```

Si te pierdes… ¡mira el mapa!

```
vagrant@vpn-core:~$ sudo ./wg-ebpf wg0
2022/10/10 15:00:02 Attached XDP program to iface "wg0" (index 6)
2022/10/10 15:00:02 Press Ctrl-C to exit and remove the program
2022/10/10 15:00:12 Map contents:
        192.168.4.2 => 1
2022/10/10 15:00:13 Map contents:
        192.168.4.2 => 2
2022/10/10 15:00:14 Map contents:
        192.168.4.2 => 3
2022/10/10 15:00:17 Map contents:
        192.168.4.2 => 4
2022/10/10 15:00:18 Map contents:
        192.168.4.3 => 1
        192.168.4.2 => 5
2022/10/10 15:00:19 Map contents:
        192.168.4.3 => 2
        192.168.4.2 => 6
2022/10/10 15:00:20 Map contents:
        192.168.4.3 => 3
        192.168.4.2 => 6
^C
```

```
vagrant@client-a:~$ ping -c 3 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=64 time=503 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=64 time=501 ms

--- 192.168.4.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 501.371/502.023/503.220/0.847 ms
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
```

Si te pierdes… ¡mira el mapa!

```
vagrant@vpn-core:~$ sudo ./wg-ebpf wg0
2022/10/10 15:00:02 Attached XDP program to iface "wg0" (index 6)
2022/10/10 15:00:02 Press Ctrl-C to exit and remove the program
2022/10/10 15:00:12 Map contents:
        192.168.4.2 => 1
2022/10/10 15:00:13 Map contents:
        192.168.4.2 => 2
2022/10/10 15:00:14 Map contents:
        192.168.4.2 => 3
2022/10/10 15:00:17 Map contents:
        192.168.4.2 => 4
2022/10/10 15:00:18 Map contents:
        192.168.4.3 => 1
        192.168.4.2 => 5
2022/10/10 15:00:19 Map contents:
        192.168.4.3 => 2
        192.168.4.2 => 6
2022/10/10 15:00:20 Map contents:
        192.168.4.3 => 3
        192.168.4.2 => 6
^C
```

```
vagrant@client-a:~$ ping -c 3 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=64 time=503 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=64 time=501 ms

--- 192.168.4.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 501.371/502.023/503.220/0.847 ms
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
```

Si te pierdes… ¡mira el mapa!

```
vagrant@vpn-core:~$ sudo ./wg-ebpf wg0
2022/10/10 15:00:02 Attached XDP program to iface "wg0" (index 6)
2022/10/10 15:00:02 Press Ctrl-C to exit and remove the program
2022/10/10 15:00:12 Map contents:
        192.168.4.2 => 1
2022/10/10 15:00:13 Map contents:
        192.168.4.2 => 2
2022/10/10 15:00:14 Map contents:
        192.168.4.2 => 3
2022/10/10 15:00:17 Map contents:
        192.168.4.2 => 4
2022/10/10 15:00:18 Map contents:
        192.168.4.3 => 1
        192.168.4.2 => 5
2022/10/10 15:00:19 Map contents:
        192.168.4.3 => 2
        192.168.4.2 => 6
2022/10/10 15:00:20 Map contents:
        192.168.4.3 => 3
        192.168.4.2 => 6
^C
```

```
vagrant@client-a:~$ ping -c 3 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=64 time=503 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=64 time=501 ms

--- 192.168.4.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 501.371/502.023/503.220/0.847 ms
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
```

Si te pierdes… ¡mira el mapa!

```
vagrant@vpn-core:~$ sudo ./wg-ebpf wg0
2022/10/10 15:00:02 Attached XDP program to iface "wg0" (index 6)
2022/10/10 15:00:02 Press Ctrl-C to exit and remove the program
2022/10/10 15:00:12 Map contents:
        192.168.4.2 => 1
2022/10/10 15:00:13 Map contents:
        192.168.4.2 => 2
2022/10/10 15:00:14 Map contents:
        192.168.4.2 => 3
2022/10/10 15:00:17 Map contents:
        192.168.4.2 => 4
2022/10/10 15:00:18 Map contents:
        192.168.4.3 => 1
        192.168.4.2 => 5
2022/10/10 15:00:19 Map contents:
        192.168.4.3 => 2
        192.168.4.2 => 6
2022/10/10 15:00:20 Map contents:
        192.168.4.3 => 3
        192.168.4.2 => 6
^C
```

```
vagrant@client-a:~$ ping -c 3 192.168.4.1
PING 192.168.4.1 (192.168.4.1) 56(84) bytes of data.
64 bytes from 192.168.4.1: icmp_seq=1 ttl=64 time=503 ms
64 bytes from 192.168.4.1: icmp_seq=2 ttl=64 time=501 ms
64 bytes from 192.168.4.1: icmp_seq=3 ttl=64 time=501 ms

--- 192.168.4.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 501.371/502.023/503.220/0.847 ms
vagrant@client-a:~$ ping -c 3 192.168.4.3
PING 192.168.4.3 (192.168.4.3) 56(84) bytes of data.

--- 192.168.4.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
```

Las respuestas se descartan en vpn-core cuando llegan desde client-b según la regla que hemos impuesto.

Resumiendo…

- El modelo de seguridad con `WireGuard/iptables` ha funcionado fenomenal.
- La integración de WireGuard con el kernel permite gran cantidad de usos.
- El conformado del tráfico que atraviese WireGuard es viable.
- Con eBPF podemos añadir una gran capa de programabilidad.
- Juntando todo lo anterior, este despliegue es tremendamente flexible.
- Esperamos que la charla haya sido útil y si no… ¡por lo menos interesante!
- ¡Muchas gracias por vuestra atención!

¿Preguntas?

Fuente