

Quiero que me expliques con lujo de detalle como funciona example4.py para poder crear dos slices con diferentes ancho de banda y latencias en cada uno, quiero que me indiques en donde esta efecutando estos cambios en el open5gs y ueransim, has de cuenta que no entiendo nada

ChatGPT Plus

Pensó durante 31s

Perfecto — te lo desgloso **con mucho detalle paso a paso** y con los comandos exactos que puedes copiar/pegar. La idea es que entiendas **qué hace example4.py**, **dónde** se configuran los *slices* (en Open5GS y en UERANSIM) y **cómo** implementar dos slices con **diferente latencia y ancho de banda**. Voy a dividirlo en: (A) explicación general de cómo funciona el script y la plataforma, (B) qué archivos/servicios modifican la lógica de slice, y (C) instrucciones prácticas para crear 2 slices con control de latencia y banda, con comprobaciones y comandos para medir.

1. Qué hace `example4.py` (visión de alto nivel)

- `example4.py` crea una topología virtual con **Containernet** (Mininet extendido que puede lanzar hosts Docker):
 - Añade 5 *DockerHost*: cp (Open5GS control plane), `upf_cld` (UPF cloud), `upf_mec` (UPF MEC), `gnb` (gNB), `ue` (UE).
 - Crea 3 switches (`s1`, `s2`, `s3`) y enlaces entre ellos usando `TCLink` (esto permite fijar `bw` y `delay` en cada enlace).
 - Arranca la red y abre la CLI de Mininet para interactuar.
- Control plane (Open5GS):
 - El script usa la clase `Open5GS` (archivo `python_modules/Open5GS.py`) para programáticamente insertar un **subscriber** en la base de datos MongoDB de Open5GS (esto es equivalente a “crear un UE” desde la WebUI).
 - Cuando el UE se registra, el control plane (AMF/SMF) crea las PDU sessions para los S-NSSAI/DNN definidos en el perfil del suscriptor, asignando cada sesión al UPF correspondiente (según la configuración).
- User plane (UERANSIM + UPF):
 - `open5gs_ue_init.sh` (ejecutado en el contenedor UE) arranca UERANSIM y crea las PDU sessions que originan las **interfaces tun/uesimtunX** dentro del contenedor UE (cada sesión aparece como `uesimtun0`, `uesimtun1`, ... con direcciones 10.45.0.2, 10.46.0.2 en el ejemplo).
 - El UPF (contenedor `upf_cld` / `upf_mec`) crea su extremo del túnel (10.45.0.1, 10.46.0.1).
- En resumen: **las sesiones (slices)** se definen por:
 1. El **perfil del suscriptor** (control plane — Open5GS / MongoDB) que dice qué S-NSSAI y qué DNN/session debe crear.
 2. El **UE** (UERANSIM) que solicita esas sesiones según su configuración.
 3. Los **UPF** que forman los extremos user-plane y transportan paquetes.

2. ¿Dónde se aplican los cambios para crear/ajustar slices?

A — En Open5GS (control plane)

- **Dónde está:** los datos de suscriptores y sesiones están en **MongoDB** que usa Open5GS (normalmente DB open5gs).
- **Cómo se cambia:** example4.py usa python_modules/Open5GS.py para conectarse al MongoDB (host/puerto: en el ejemplo 172.17.0.2:27017) y crear/eliminar suscriptores.
- **Qué editar:**
 - python_modules/subscriber_profile.json — aquí defines el perfil del subscriber (IMSI, APN/DNN, sesiones PDU, S-NSSAI, etc.).
 - Si quieres cambiar qué slices crea el control plane, editas ese JSON (agregar sesiones con S-NSSAI/DNN y parámetros) y vuelves a ejecutar example4.py para insertar el perfil en la DB.

Cómo inspeccionar los suscriptores en MongoDB (comandos):

```
# entrar al contenedor CP
docker exec -it cp bash

# dentro del contenedor cp, abrir mongo shell
mongo --port 27017

# en mongo shell:
use open5gs
db.subscribers.find().pretty()
```

Esto te mostrará lo que está guardado y las sesiones (busca campos relacionados con sessions, dnn o slice / s-nssai).

B — En UERANSIM (UE / gNB)

- **Dónde está:** configuración en ~/comnetsemu_5Gnet/ueransim/config/ (por ejemplo open5gs-ue.yaml o ue.yaml dependiendo del repo).
- **Cómo se cambia:** en el archivo de configuración del UE (UERANSIM) defines las PDU sessions que el UE iniciará al arrancar (o el script open5gs_ue_init.sh puede forzar la creación de sesiones).
- **Qué editar:** parámetros como dnn, s-nssai o PDU session list dentro del YAML del UE. Al iniciar UERANSIM el UE pedirá al AMF/SMF esas sesiones y el control plane/SMF asignará los UPFs y establecerá user-plane.

Comprobación dentro del contenedor UE:

```
docker exec -it ue bash
# ver logs de UERANSIM (según dónde lo coloque el script)
tail -f /mnt/log/ueransim.log
# ver interfaces
ifconfig
# buscar uesimtun0 / uesimtun1
```

3. Cómo implementar dos slices con distinta latencia y ancho (pasos concretos)

Existen dos niveles donde impones diferencia:

- **A. Topología/latencia:** haces que la ruta a cada UPF tenga diferentes delays poniendo `delay` en los enlaces de Mininet (`net.addLink(..., delay="Xms")`). Eso ya está muy cerca en tu `example4.py` ($s1-s2 = 10\text{ms}$, $s2-s3 = 50\text{ms}$). La ruta a `upf_mec` ($\text{ue} \rightarrow s1 \rightarrow s2 \rightarrow \text{upf_mec}$) será más corta que a `upf_cld` ($\text{ue} \rightarrow s1 \rightarrow s2 \rightarrow s3 \rightarrow \text{upf_cld}$) → mayor RTT.
- **B. Ancho de banda (b/w):** debes aplicar *shaping* (`tc/htb/tbf`) en la interfaz adecuada para limitar la tasa por slice. Esto lo puedes aplicar:
 - En **el UE** (sobre `uesimtun0 / uesimtun1`) para limitar el tráfico saliente del UE (útil para pruebas de uplink).
 - En **el UPF** (en la interfaz que va hacia la UE) para limitar el tráfico descendente (downlink).
 - O en uno de los enlaces de Mininet (pero un enlace puede ser compartido por ambos slices si pasa por el mismo path).

A continuación te doy **dos recetas**: A) cambiar topología (latencia) y B) aplicar limitación de ancho por slice con `tc` en UPF (downlink) y en UE (uplink).

3.A Cambiar latencias (editar `example4.py`)

Abre `example4.py` y edita los `net.addLink` para que la **ruta al MEC tenga menos delay** y la **ruta al Cloud tenga más delay**.

En tu fichero actual ya hay:

```
net.addLink(s1, s2, bw=1000, delay="10ms", ...)
net.addLink(s2, s3, bw=1000, delay="50ms", ...)
```

Esto hace que:

- UE → gNB → s1 → s2 → upf_mec \approx 1ms + 10ms + 1ms = ~12ms (baja latencia)
- UE → ... → s3 → upf_cld \approx 1ms + 10ms + 50ms + 1ms = ~62ms (alta latencia)

Si quieres cambiar valores, edítalos así (ejemplos):

```
# camino MEC: poner s1-s2 = 5ms
net.addLink(s1, s2, bw=1000, delay="5ms", intfName1="s1-s2", intfName2="s2-s1")
# camino cloud: poner s2-s3 = 80ms
net.addLink(s2, s3, bw=1000, delay="80ms", intfName1="s2-s3", intfName2="s3-s2")
```

Guarda y reinicia `example4.py`. Verifica con `ping` desde UE a los UPF para ver RTT.

3.B Limitar ancho de banda por slice (forma recomendada: aplicar `tc` en UPF para el downlink)

Razonamiento: las PDU sessions tienen rangos IP específicos. En el ejemplo típico:

- UPF cloud IP: 10.45.0.1 ← UE uesmtun0 = 10.45.0.2
- UPF mec IP: 10.46.0.1 ← UE uesmtun1 = 10.46.0.2

Podemos limitar tráfico `dest = 10.45.0.2` a 2 Mbps, y `dest = 10.46.0.2` a 10 Mbps en el UPF contenedor (para downlink). También conviene limitar en UE en la interfaz `uesmtunX` para el uplink.

Paso 1 — Averigua las direcciones de los túneles

En la UE:

```
# en la VM
docker exec -it ue bash
ifconfig
# toma nota de las IPs de uesmtun0 y uesmtun1 (ej. 10.45.0.2 y 10.46.0.2)
exit
```

En el UPF cloud:

```
docker exec -it upf_cld bash
ifconfig
# identifica la interfaz que conecta a la red de túneles; anota IP (ej.
10.45.0.1)
exit
```

En el UPF mec:

```

docker exec -it upf_mec bash
ifconfig
# identifica la interfaz (ej. 10.46.0.1)
exit

```

Paso 2 — Limitar downlink en UPF cloud a 2 Mbps (ejemplo)

En la VM (fuera de Mininet):

```

# suponer que upf_cld tiene interfaz 'upf-s3' conectada al switch s3
docker exec -it upf_cld bash

# instalar tc si no está (dentro del contenedor, si falla, hazlo en la imagen
de UPF)
apt-get update && apt-get install -y iproute2 iutils-ping

# aplica HTB: limitar tráfico que tiene como destino 10.45.0.2 a 2mbit
tc qdisc add dev upf-s3 root handle 1: htb default 30
tc class add dev upf-s3 parent 1: classid 1:1 htb rate 1000mbit ceil 1000mbit
tc class add dev upf-s3 parent 1:1 classid 1:10 htb rate 2mbit ceil 2mbit
tc class add dev upf-s3 parent 1:1 classid 1:30 htb rate 1000mbit ceil
1000mbit

# aplicamos un filtro que coincida con paquetes destinados a la IP de la UE
en slice cloud
tc filter add dev upf-s3 protocol ip parent 1: prio 1 u32 match ip dst
10.45.0.2/32 flowid 1:10

```

Nota: `upf-s3` es el nombre de la interfaz que definiste en Mininet (`intfName1="upf-s3"`). Usa el nombre correcto mostrado por `ifconfig`.

Paso 3 — Limitar downlink en UPF MEC a 10 Mbps

```

docker exec -it upf_mec bash

tc qdisc add dev upf_mec-s2 root handle 1: htb default 30
tc class add dev upf_mec-s2 parent 1: classid 1:1 htb rate 1000mbit ceil
1000mbit
tc class add dev upf_mec-s2 parent 1:1 classid 1:10 htb rate 10mbit ceil
10mbit
tc class add dev upf_mec-s2 parent 1:1 classid 1:30 htb rate 1000mbit ceil
1000mbit

tc filter add dev upf_mec-s2 protocol ip parent 1: prio 1 u32 match ip dst
10.46.0.2/32 flowid 1:10

```

Paso 4 — Limitar uplink en UE (opcional)

En el contenedor UE limita la interfaz `uesimtun0` a 2mbit y `uesimtun1` a 10mbit:

```
docker exec -it ue bash
```

```
# uesimtun0 -> 2 Mbps
tc qdisc add dev uesimtun0 root handle 1: htb default 10
tc class add dev uesimtun0 parent 1: classid 1:1 htb rate 2mbit ceil 2mbit

# uesimtun1 -> 10 Mbps
tc qdisc add dev uesimtun1 root handle 1: htb default 10
tc class add dev uesimtun1 parent 1: classid 1:1 htb rate 10mbit ceil 10mbit
```

Verificación (iperf3)

- Arranca iperf3 server en cada UPF (si no está corriendo, en el UPF container ejecuta iperf3 -s)
- Desde UE:

```
docker exec -it ue bash
iperf3 -c 10.45.0.1 -B 10.45.0.2 -t 5    # debe acercarse a ~2 Mbps
iperf3 -c 10.46.0.1 -B 10.46.0.2 -t 5    # debe acercarse a ~10 Mbps
```

4. Dónde editar la configuración “de slices” en los archivos del proyecto

A — `python_modules/subscriber_profile.json`

- Ese es el archivo que `example4.py` carga y que el Open5GS Python helper inserta en MongoDB.
- Abrelo y fíjate en estructuras como `sessions`, `dnn`, `s-nssai` o `slice`. Para crear dos sesiones asegúrate que el JSON tenga dos objetos de sesión con `dnn` distinto y/o `s-nssai` distinto. Ejemplo (PSEUDOCÓDIGO JSON):

```
{
  "imsi": "001010000000001",
  "subscriber": {
    "sessions": [
      {
        "dnn": "internet",
        "s_nssai": {"sst": 1, "sd": "010203"},
        "pduSessionType": "IPv4",
        "ipv4": "10.45.0.2"
      },
      {
        "dnn": "mec",
        "s_nssai": {"sst": 2, "sd": "010204"},
        "pduSessionType": "IPv4",
        "ipv4": "10.46.0.2"
      }
    ]
  }
}
```

IMPORTANTE: la estructura exacta puede variar con la versión de Open5GS; abre tu `subscriber_profile.json` y copia el formato que ya exista para añadir la segunda sesión.

B — `ueransim/config/*` (UE config)

- Busca el archivo `open5gs-ue.yaml` o `ue.yaml` en `~/comnetsemu_5Gnet/ueransim/config/`.
- Allí define **qué PDU sessions** el UE solicitará al iniciar. Añade dos entradas, una para cada DNN/S-NSSAI.
- Ejemplo conceptual (YAML):

```
ue:  
  imsi: "00101000000001"  
  pduSessions:  
    - dnn: "internet"  
      sst: 1  
      sd: "010203"  
    - dnn: "mec"  
      sst: 2  
      sd: "010204"
```

Nuevamente, comprueba el formato exacto consultando el archivo de ejemplo en el directorio `ueransim/config/`.

5. Logs y debug (dónde mirar si algo falla)

- **Open5GS logs** dentro del contenedor CP:
 - `docker exec -it cp bash`
 - `tail -f /open5gs/install/var/log/open5gs/*.log`

Revisa AMF/SMF logs para ver si PDU sessions se crearon.

- **UPF logs:**
 - `docker exec -it upf_cld bash`
 - `tail -f /open5gs/install/var/log/open5gs/*.log`
- **UERANSIM logs (UE & gNB):**
 - `docker exec -it ue bash`
 - `tail -f /mnt/log/ueransim-ue.log` # o el nombre que use el script
- **Ver interfaces:**
 - `docker exec -it ue bash`
 - `ifconfig` # busca uesimtun0/1
 -
 - `docker exec -it upf_cld bash`
 - `ifconfig` # busca interfaz upf-s3 o el nombre correcto
- **Ver subscribers en MongoDB** (desde CP contenedor):

-
- docker exec -it cp bash
 - mongo --port 27017 --eval 'db = db.getSiblingDB("open5gs"); printjson(db.subscribers.find().toArray())'
-

6. Flujo completo sugerido (comandos listos para pegar)

1. Edita subscriber_profile.json añadiendo dos sesiones (s-nssai/dnn).
 2. Edita ueransim/config para que el UE pida ambas PDU sessions.
 3. (Opcional) ajusta example4.py delays si quieres otras latencias.
 4. Limpia estado previo:
 5. sudo mn -c
 6. docker rm -f \$(docker ps -aq) || true
 7. sudo systemctl restart docker
 8. Levanta la topología:
 9. cd ~/comnetsemu_5Gnet
 10. sudo python3 example4.py
 11. En otra terminal, asegúrate de los logs y comprobar que el UE registró:
 12. ./enter_container.sh ue
 13. ifconfig
 14. # espera que aparezcan uesimtun0 y uesimtun1
 15. exit
 16. Aplica shaping en UPF (como mostré arriba con tc) para 2Mbps/10Mbps.
 17. Ejecuta iperf:
 18. docker exec -it ue bash
 19. iperf3 -c 10.45.0.1 -B 10.45.0.2 -t 5
 20. iperf3 -c 10.46.0.1 -B 10.46.0.2 -t 5
 21. Captura:
 22. ./start_tcpdump.sh upf_cld
 23. ./start_tcpdump.sh upf_mec
 24. # o desde UE:
 25. docker exec -it ue bash
 26. tcpdump -i uesimtun0 -w /tmp/uesimtun0.pcap &
 27. tcpdump -i uesimtun1 -w /tmp/uesimtun1.pcap &
 28. # generar tráfico y luego Ctrl+C
-

7. Resumen / idea clave

- **Control plane (Open5GS)** decide y crea las sesiones/slices según lo que pongas en **subscriber_profile.json** (insertado en MongoDB).
- **UE (UERANSIM)** solicita las PDU sessions según su config (ueransim/config), y crea las uesimtunX user-plane interfaces.
- **UPF** crea su extremo de túnel y enruta el user-plane.
- **Latencia** se logra principalmente con delay en los enlaces de Mininet (net.addLink(... delay="Xms")).

- **Ancho de banda** se logra colocando shaping (`tc`, HTB/TBF) en las interfaces correctas (UPF para downlink, UE para uplink) o configurando `bw` en `TCLink` si quieres limitar físicamente un enlace — pero para slices diferenciadas conviene aplicar `tc` por túnel/IP (como mostré).