GUIA INSTALACION SIMU 5G

1) INSTALACION Y CONFIGURACION DE UBUNTU

1.1 Crear la VM

REQUISITOS:

Software: VirtualBox o VMware (el que prefieras).

SO invitado: Ubuntu 22.04 LTS (64-bit).

Recursos:

CPU: 4 vCPU (mejor 6–8 si puedes)

RAM: 8–16 GB (mejor 16 si compilas rápido),

Disco: 60–80 GB (la compilación ocupa).

Activar VT-x/AMD-V (suele venir por defecto).

NOTA: *Seguir la guía de instalación de Ubuntu 22.04 en VM*

1.2 Abrir la Terminal:

Ctrl + Alt + T

1.3 Actualizar el sistema:

```
sudo apt update && sudo apt -y upgrade
```

1.4 Instalar herramientas base (compilación y utilidades):

```
sudo apt update
sudo apt upgrade -y


# Herramientas y dependencias comunes para OMNeT++/INET/Simu5G
sudo apt install -y build-essential curl git wget ca-certificates \
python3 python3-pip python3-venv pkg-config cmake \
libxml2-dev zlib1g-dev doxygen graphviz \
qt6-base-dev qt6-base-dev-tools libqt6svg6 qt6-wayland \
libwebkit2gtk-4.1-0
```

2) Instalar **Nix** (lo usa **opp_env** para instalaciones reproducibles)

```
sh <(curl -L https://nixos.org/nix/install)
. ~/.nix-profile/etc/profile.d/nix.sh
nix –version
```

3) Instalar opp_env **y crear el workspace reproducible de** Simu5G

```
# Instala opp_env para tu usuario
pip3 install --user opp_env

# Asegura que ~/.local/bin está en el PATH (solo para esta sesión)
export PATH="$HOME/.local/bin:$PATH"

# Crea el workspace
mkdir -p ~/simu5g_workspace
cd ~/simu5g_workspace

# Inicializa opp_env (crea archivos de entorno)
opp_env init
```

4) Instalar Simu5G 1.4.0 **(trae OMNeT++ 6.2.0 e INET 4.5.4)**

```
# Instala la release recomendada y compatible
opp_env install simu5g-1.4.0
```

4.1 Entrar al entorno:

```
opp_env shell
```

5) Verificación de herramientas (incluida la de exportación de resultados)

```
which opp_scavetool
```

6) Ejecutar un ejemplo base para validar la instalación

```
~/simu5g_workspace/simu5g-1.4.0/simulations/nr/standalone
```

7) Crear el escenario a partir del standalone oficial

```
opp_env Shell
cd ~/simu5g_workspace/simu5g-1.4.0/simulations/nr
cp -r standalone My3Slices
cd My3Slices
ls
```

Verificamos que existan los siguientes archivos:

```
demo.xml  omnetpp.ini  results  run
```

8) Reemplaza por completo **el `omnetpp.ini` de `My3Slices`**

Editar el archivo `omnetpp.ini`

Abre el archivo con **`nano`:**

nano omnetpp.ini

Borra todo y coloca el siguiente código (si no deseas hacerlo con nano también puedes buscar la ubicación del archivo, abrirlo con block de notas y borrar y pegar con más facilidad).

```
[General]
network = simu5g.simulations.nr.networks.SingleCell_Standalone
sim-time-limit = 30s
warmup-period = 1s

# Archivos de salida
output-scalar-file = results/slices_results.sca
output-vector-file = results/slices_results.vec
record-eventlog = true

# =========================
# CONFIGURACIÓN GENERAL 5G
# =========================
**.numBands = 50
**.targetBler = 0.01
**.blerShift = 5
**.vector-recording = true
**.scalar-recording = true

*.gnb.mobility.initialX = 450m
*.gnb.mobility.initialY = 300m

*.numUe = 3

# Ubicación UE por slice
```

```
*.ue[0].mobility.initialX = 300m   # eMBB
*.ue[0].mobility.initialY = 500m

*.ue[1].mobility.initialX = 450m   # URLLC
*.ue[1].mobility.initialY = 500m

*.ue[2].mobility.initialX = 600m   # mMTC
*.ue[2].mobility.initialY = 500m

# Identidad de celdas
*.ue[*].macCellId = 0
*.ue[*].masterId = 0
*.ue[*].nrMacCellId = 1
*.ue[*].nrMasterId = 1

# Assign slices
*.ue[0].nrCell->sliceId = 0   # eMBB
*.ue[1].nrCell->sliceId = 1   # URLLC
*.ue[2].nrCell->sliceId = 2   # mMTC

# =========================
# CONFIGURACIÓN DE SCHEDULING POR SLICE
# =========================
# Más RBs para eMBB (máximo throughput)
*.slice[0].numBands = 40
*.slice[0].priority = 2

# RBs moderados, latencia mínima para URLLC
*.slice[1].numBands = 20
*.slice[1].priority = 5   # mayor prioridad -> menor delay
```

```
# RBs mínimos para mMTC
*.slice[2].numBands = 5
*.slice[2].priority = 1

# =========================
# CONFIGURACIÓN DE APLICACIONES CBR
# =========================
*.ue[*].numApps = 1
*.server.numApps = 3

# ------------- eMBB: >10 Mbps --------------------
*.server.app[0].typename = "CbrSender"
*.server.app[0].destAddress = "ue[0]"
*.server.app[0].destPort = 4000
*.server.app[0].packetLength = 5000B
*.server.app[0].sendInterval = 0.0004s   # ~100 packets/s → ~40 Mbps

# ------------- URLLC: 5–8 Mbps, delay mínimo ------
*.server.app[1].typename = "CbrSender"
*.server.app[1].destAddress = "ue[1]"
*.server.app[1].destPort = 4001
*.server.app[1].packetLength = 2000B
*.server.app[1].sendInterval = 0.0008s   # ~20 packets/s → ~10 Mbps
# HARQ rápidas
*.ue[1].**.harqEnabled = true

# ------------- mMTC: <1 Mbps, jitter alto ----------
*.server.app[2].typename = "CbrSender"
*.server.app[2].destAddress = "ue[2]"
*.server.app[2].destPort = 4002
*.server.app[2].packetLength = 200B
```

```
*.server.app[2].sendInterval = exponential(0.02s)   # tráfico bursty IoT

# Simulación de red congestionada IoT

*.ue[2].**.bufferSize = 20kB

*.ue[2].**.queue.typename = "DropTailQueue"
```

7) Ejecutar tu simulación

```
cd ~/simu5g_workspace/simu5g/simulations/NR/My3Slices


# Modo consola (rápido, útil en VM)

./run -u Cmdenv
```

8) Extraer sólo las métricas de los UEs (no del servidor)

```
opp_scavetool export -T s -F CSV-R \

--filter '(module=~"**.ue[*].app[0]" AND (name=~"cbrReceivedThroughput:mean" OR name=~"cbrFrameDelay:mean" OR name=~"cbrJitter:mean" OR name=~"cbrFrameLoss:mean" OR name=~"cbrReceivedBytes:sum"))' \

-o slices_metrics.csv slices_results.sca
```

9) Script Python para graficar resultados por slice

```
import pandas as pd

import matplotlib.pyplot as plt

import os


# === Configuración ===

CSV_FILE = "slices_metrics.csv"

OUT_DIR = "plots"

os.makedirs(OUT_DIR, exist_ok=True)


df = pd.read_csv(CSV_FILE)


# Limpiar columnas

df = df[df["type"] == "scalar"]

df = df[["module", "name", "value"]].dropna()

df["value"] = pd.to_numeric(df["value"], errors="coerce")
```

```
# Identificar UE (slice)
df["slice"] = df["module"].str.extract(r"ue\[(\d+)\]").astype(int)
slice_labels = {0: "Slice 0 – eMBB", 1: "Slice 1 – URLLC", 2: "Slice 2 – mMTC"}
df["slice_name"] = df["slice"].map(slice_labels)

# --- Agrupar métricas ---
metrics = {
    "cbrReceivedThroughput:mean": "Throughput (Mbps)",
    "cbrFrameDelay:mean": "Delay (ms)",
    "cbrJitter:mean": "Jitter (ms)",
    "cbrFrameLoss:mean": "Frame Loss (%)"
}

for metric, ylabel in metrics.items():
    subset = df[df["name"] == metric].copy()
    if subset.empty:
        print(f"⚠ Métrica no encontrada: {metric}")
        continue

    # Conversión de unidades
    if "Throughput" in ylabel:
        subset["value"] = subset["value"] / 1e6   # bits/s → Mbps
    elif "Delay" in ylabel or "Jitter" in ylabel:
        subset["value"] = subset["value"] * 1e3   # s → ms
    elif "Loss" in ylabel:
        subset["value"] = subset["value"] * 100   # proporción → %

    # Graficar
    plt.figure(figsize=(6, 4))
    plt.bar(subset["slice_name"], subset["value"], color=["#1f77b4", "#ff7f0e", "#2ca02c"])
```

```
    plt.ylabel(ylabel)
    plt.title(f"Comparación por Slice: {ylabel}")
    plt.tight_layout()
    outfile = os.path.join(OUT_DIR, f"{metric.replace(':','_')}.png")
    plt.savefig(outfile)
    plt.close()
    print(f"✅ Gráfico guardado: {outfile}")

print("\n🎯 Listo. Gráficos disponibles en carpeta:", OUT_DIR)
```

10) Ejecutar el archivo

```
python3 plot_slices.py
```