MANUAL NS-3

NS-3 (Network Simulator 3) es una herramienta de simulación de redes de código abierto ampliamente utilizada en investigación académica y desarrollo tecnológico. Permite modelar y analizar el comportamiento de redes de comunicación a nivel de paquetes, desde entornos cableados hasta redes inalámbricas avanzadas. Está diseñado en C++ y Python, y ofrece una arquitectura modular que facilita la construcción de escenarios personalizados, el estudio del rendimiento de protocolos, y la evaluación de métricas como throughput, delay, jitter y packet loss.

El simulador incluye módulos que emulan tecnologías como WiFi (802.11), LTE, 5G NR, TCP/IP, vehicular networking (V2X), y muchas otras, lo que lo convierte en una herramienta versátil para estudiar redes tanto tradicionales como emergentes.

Propósito de la simulación

En esta simulación se implementa un escenario de network slicing sobre una red WiFi, con el fin de demostrar cómo se puede segmentar lógicamente una misma infraestructura física para ofrecer distintos niveles de servicio.

El experimento se centra en dos slices inalámbricos:

- El Slice 1, configurado con baja capacidad de transmisión y baja latencia, representa servicios que requieren estabilidad y tiempos de respuesta cortos (por ejemplo, IoT o control industrial).
- El Slice 2, configurado con alta capacidad y mayor carga, emula aplicaciones que priorizan el ancho de banda, como transmisión de video o transferencia de datos intensiva.

Elementos utilizados del simulador

Para construir el escenario, se emplean los siguientes componentes principales del entorno ns-3:

- **YansWifiChannel y YansWifiPhyHelper:** definen las propiedades físicas del canal inalámbrico.
- **WifiMacHelper y WifiHelper:** configuran el estándar WiFi, el modo de operación y la capa MAC.
- **NodeContainer y NetDeviceContainer:** crean los nodos (estaciones y punto de acceso) y las interfaces de red correspondientes.
- **MobilityHelper:** establece posiciones fijas para los dispositivos dentro del área de simulación.
- **OnOffHelper y PacketSinkHelper:** generan tráfico UDP entre los nodos y permiten medir el rendimiento.
- **FlowMonitorHelper:** recopila estadísticas de rendimiento como throughput, delay, jitter y pérdida de paquetes.

Objetivo

El objetivo principal es evaluar el desempeño diferenciado de cada slice dentro de un mismo entorno inalámbrico, analizando métricas clave de calidad de servicio (KPIs) para validar la efectividad del network slicing en la separación de tráfico. Esto permite observar cómo distintas configuraciones de red pueden coexistir y comportarse de forma aislada, incluso cuando comparten el mismo medio físico.

GUIA DE INSTALACION DE NS-3 EN MAQUINA VIRTUAL

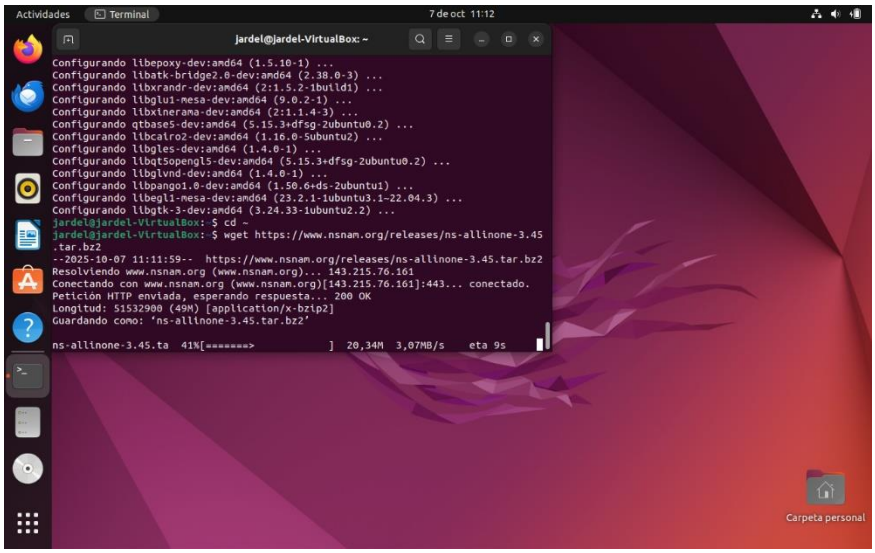
1) Instalar dependencias necesarias de Boost y desarrollo general

Ejecutar el siguiente código:

```
sudo apt update
sudo apt install -y build-essential gcc g++ python3 python3-dev python3-pip \
pkg-config sqlite3 cmake ninja-build git \
libboost-all-dev libgtk-3-dev qtbase5-dev \
libxml2 libxml2-dev libsqlite3-dev libssl-dev
```

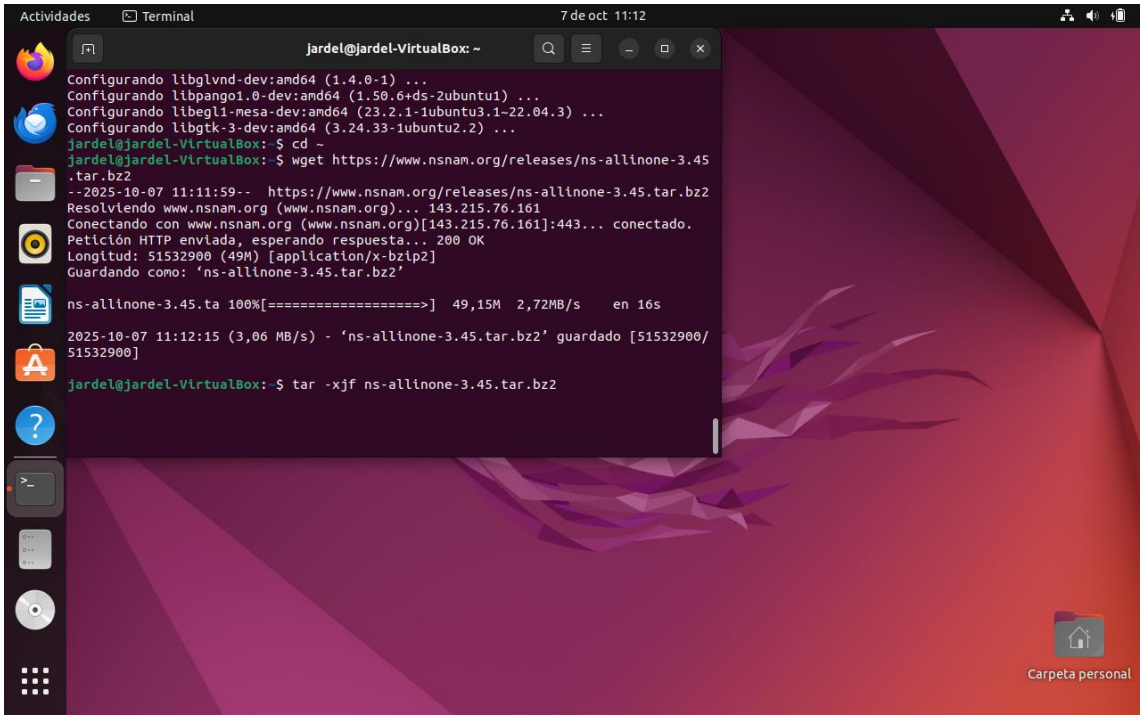
Segunda tanda de dependencias:

```
sudo apt install -y g++ python3 python3-dev cmake ninja-build git pkg-config \
sqlite3 mercurial libgtk-3-dev vtun lxc uml-utilities \
qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools
```



2) Descargar e instalar NS-3.45

```
cd ~

wget https://www.nsnam.org/releases/ns-allinone-3.45.tar.bz2

tar -xjf ns-allinone-3.45.tar.bz2

cd ns-allinone-3.45/ns-3.45
```



3) Configurar y compilar

```
./ns3 configure --build-profile=optimized --enable-examples --enable-tests

./ns3 build
```

4) Verificar instalación

```
./ns3 run hello-simulator
```

Nota: Al correrlo debe salir algo como “Hello Simulator”

5) Crear simulación personalizada

```
cd ~/ns-allinone-3.45/ns-3.45/scratch

nano network-slicing-wifi.cc
```

6) Pegar el siguiente codigo en la ventana emergente que salió

```
/* network-slicing-wifi.cc
 *
 * Two-slice WiFi example for ns-3.45 (or similar).
 * Slice1: low-rate, low-latency target
 * Slice2: high-rate, higher-load -> likely higher delay
 *
 * Avoids using helpers' ::Default() to be compatible with ns-3.45 builds.
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/wifi-helper.h"
#include "ns3/wifi-mac-helper.h"
#include "ns3/ssid.h"
#include "ns3/applications-module.h"
#include "ns3/flow-monitor-module.h"

#include <fstream>
#include <iomanip>

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("NetworkSlicingWiFi");

int
main (int argc, char *argv[])
{
```

```
double simTime = 15.0;
CommandLine cmd;
cmd.AddValue ("simTime", "Simulation time (s)", simTime);
cmd.Parse (argc, argv);

// --- NODES -------------------------------------------------------------
NodeContainer staSlice1; staSlice1.Create (2); // two STAs slice1
NodeContainer staSlice2; staSlice2.Create (2); // two STAs slice2
NodeContainer apNode; apNode.Create (1);       // single AP node

// --- CHANNEL & PHY -----------------------------------------------------
YansWifiChannelHelper channelHelper = YansWifiChannelHelper::Default ();
Ptr<YansWifiChannel> channel = channelHelper.Create ();

// Two PHY helpers that will share the same channel object
YansWifiPhyHelper phy1; phy1.SetChannel (channel);
YansWifiPhyHelper phy2; phy2.SetChannel (channel);

// --- WIFI HELPERS (different station managers per slice) ---------------
WifiHelper wifi1; // slice 1 -> low MCS (low PHY rate)
wifi1.SetStandard (WIFI_STANDARD_80211ac);
wifi1.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                               "DataMode", StringValue ("VhtMcs0"),
                               "ControlMode", StringValue ("VhtMcs0"));

WifiHelper wifi2; // slice 2 -> high MCS (high PHY rate)
wifi2.SetStandard (WIFI_STANDARD_80211ac);
wifi2.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                               "DataMode", StringValue ("VhtMcs7"),
                               "ControlMode", StringValue ("VhtMcs0"));

// --- MACs & SSIDs ------------------------------------------------------
Ssid ssid1 = Ssid ("slice-1-ssid");
```

```
Ssid ssid2 = Ssid ("slice-2-ssid");

WifiMacHelper macSta1;
macSta1.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid1));
NetDeviceContainer devSta1 = wifi1.Install (phy1, macSta1, staSlice1);

WifiMacHelper macSta2;
macSta2.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid2));
NetDeviceContainer devSta2 = wifi2.Install (phy2, macSta2, staSlice2);

// AP devices: install two AP devices (one per SSID) on the same AP node
WifiMacHelper macAp;
macAp.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid1));
NetDeviceContainer devAp1 = wifi1.Install (phy1, macAp, apNode);

macAp.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid2));
NetDeviceContainer devAp2 = wifi2.Install (phy2, macAp, apNode);

// --- MOBILITY ------------------------------------------------------
MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (0.0),
                               "MinY", DoubleValue (0.0),
                               "DeltaX", DoubleValue (5.0),
                               "DeltaY", DoubleValue (10.0),
                               "GridWidth", UintegerValue (3),
                               "LayoutType", StringValue ("RowFirst"));
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (staSlice1);
mobility.Install (staSlice2);
mobility.Install (apNode);

// --- IP stack & addressing -----------------------------------------
```

```
 InternetStackHelper internet;
 internet.Install (staSlice1);
 internet.Install (staSlice2);
 internet.Install (apNode);

 Ipv4AddressHelper addr1;
 addr1.SetBase ("10.1.1.0", "255.255.255.0");
 Ipv4InterfaceContainer ifSta1 = addr1.Assign (devSta1);
 Ipv4InterfaceContainer ifAp1  = addr1.Assign (devAp1);

 Ipv4AddressHelper addr2;
 addr2.SetBase ("10.1.2.0", "255.255.255.0");
 Ipv4InterfaceContainer ifSta2 = addr2.Assign (devSta2);
 Ipv4InterfaceContainer ifAp2  = addr2.Assign (devAp2);

 // --- APPLICATIONS: PacketSinks on AP for each slice ----------------------
 uint16_t port1 = 5001;
 uint16_t port2 = 6001;

 PacketSinkHelper sink1 ("ns3::UdpSocketFactory", InetSocketAddress (Ipv4Address::GetAny
(), port1));
 ApplicationContainer sinkApp1 = sink1.Install (apNode.Get (0));
 sinkApp1.Start (Seconds (0.5));
 sinkApp1.Stop (Seconds (simTime));

 PacketSinkHelper sink2 ("ns3::UdpSocketFactory", InetSocketAddress (Ipv4Address::GetAny
(), port2));
 ApplicationContainer sinkApp2 = sink2.Install (apNode.Get (0));
 sinkApp2.Start (Seconds (0.5));
 sinkApp2.Stop (Seconds (simTime));

 // --- TRAFFIC GENERATORS (OnOff) --------------------------------------
 // Slice1: very low data rate (e.g., 100 kb/s) -> low throughput, low queuing
```

```
OnOffHelper onoff1 ("ns3::UdpSocketFactory", Address ());
AddressValue remoteAddress1 (InetSocketAddress (ifAp1.GetAddress (0), port1));
onoff1.SetAttribute ("Remote", remoteAddress1);
onoff1.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));
onoff1.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));
onoff1.SetAttribute ("DataRate", DataRateValue (DataRate ("100kbps"))); // <--- low
onoff1.SetAttribute ("PacketSize", UintegerValue (200));

// Install on all STAs in slice1
ApplicationContainer clientApps1 = onoff1.Install (staSlice1);
clientApps1.Start (Seconds (1.0));
clientApps1.Stop (Seconds (simTime));

// Slice2: high data rate (e.g., 20 Mb/s) -> high throughput, more contention
OnOffHelper onoff2 ("ns3::UdpSocketFactory", Address ());
AddressValue remoteAddress2 (InetSocketAddress (ifAp2.GetAddress (0), port2));
onoff2.SetAttribute ("Remote", remoteAddress2);
onoff2.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));
onoff2.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));
onoff2.SetAttribute ("DataRate", DataRateValue (DataRate ("20Mbps"))); // <--- high
onoff2.SetAttribute ("PacketSize", UintegerValue (1400));

ApplicationContainer clientApps2 = onoff2.Install (staSlice2);
clientApps2.Start (Seconds (2.0));
clientApps2.Stop (Seconds (simTime));

// --- FLOW MONITOR ----------------------------------------------------
FlowMonitorHelper flowmonHelper;
Ptr<FlowMonitor> flowmon = flowmonHelper.InstallAll ();

// --- RUN SIM ---------------------------------------------------------
Simulator::Stop (Seconds (simTime));
Simulator::Run ();
```

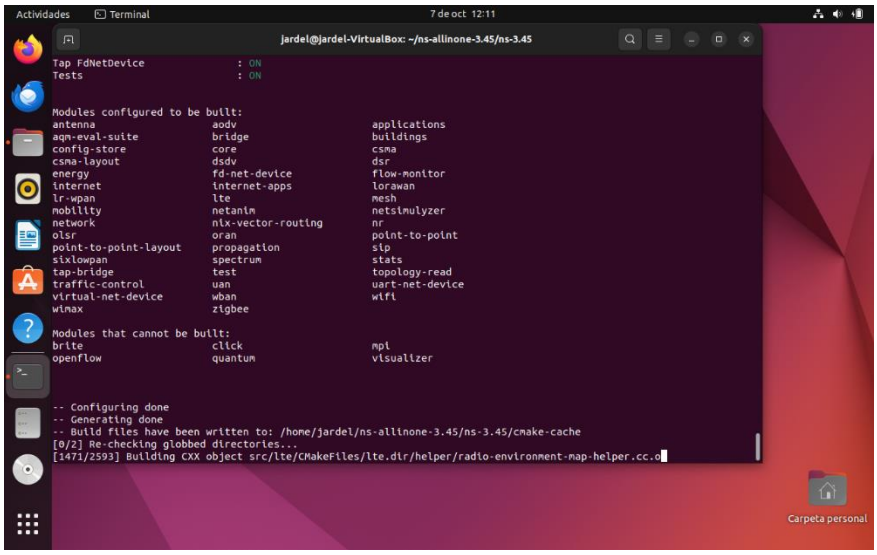
```
 // --- COLLECT STATS & WRITE CSV ----------------------------------------

 flowmon->CheckForLostPackets ();

 Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
(flowmonHelper.GetClassifier ());

 std::map<FlowId, FlowMonitor::FlowStats> stats = flowmon->GetFlowStats ();


 std::ofstream csv ("metrics_slicing_wifi.csv");

 csv <<
"FlowID,Source,Destination,TxBytes,RxBytes,Throughput_Mbps,Delay_s,MeanJitter_s,TxPacke
ts,RxPackets,LostPackets,PacketLossRate\n";

 std::cout << std::fixed << std::setprecision (6);


 for (auto const &entry : stats)
  {
    FlowId fid = entry.first;
    const FlowMonitor::FlowStats &f = entry.second;
    Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (fid);


    double duration = 0.0;
    if (!f.timeLastRxPacket.IsZero () && !f.timeFirstTxPacket.IsZero ())
     {
       duration = f.timeLastRxPacket.GetSeconds () - f.timeFirstTxPacket.GetSeconds ();
     }


    double throughput_mbps = 0.0;
    if (duration > 0.0)
     {
       throughput_mbps = (f.rxBytes * 8.0 / duration) / 1e6;
     }


    double meanDelay = 0.0;
    double meanJitter = 0.0;
    if (f.rxPackets > 0)
```

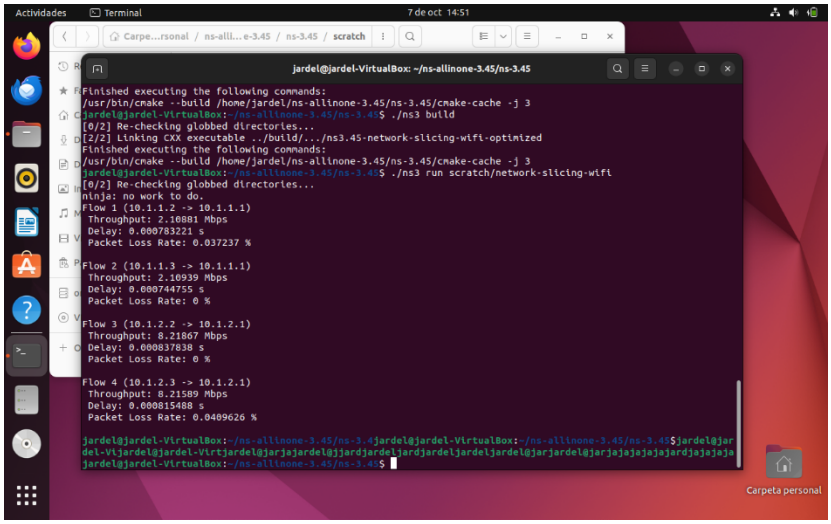
```
      {
        meanDelay = f.delaySum.GetSeconds () / f.rxPackets;
        meanJitter = f.jitterSum.GetSeconds () / f.rxPackets;
      }

    uint64_t txPackets = f.txPackets;
    uint64_t rxPackets = f.rxPackets;
    uint64_t lostPackets = f.lostPackets;
    double lossRate = (txPackets > 0) ? (double) lostPackets / (double) txPackets : 0.0;

    csv << fid << "," << t.sourceAddress << "," << t.destinationAddress << ","
        << f.txBytes << "," << f.rxBytes << "," << throughput_mbps << ","
        << meanDelay << "," << meanJitter << ","
        << txPackets << "," << rxPackets << "," << lostPackets << "," << lossRate << "\n";

    std::cout << "Flow " << fid << " (" << t.sourceAddress << " -> " << t.destinationAddress <<
")\n"
              << " Throughput: " << throughput_mbps << " Mbps\n"
              << " Delay (mean): " << meanDelay << " s\n"
              << " Jitter (mean): " << meanJitter << " s\n"
              << " Packet Loss Rate: " << (lossRate * 100.0) << " %\n\n";
   }

  csv.close ();

  Simulator::Destroy ();
  return 0;
}
```

7) Compilar y Ejecutar

cd ~/ns-allinone-3.45/ns-3.45

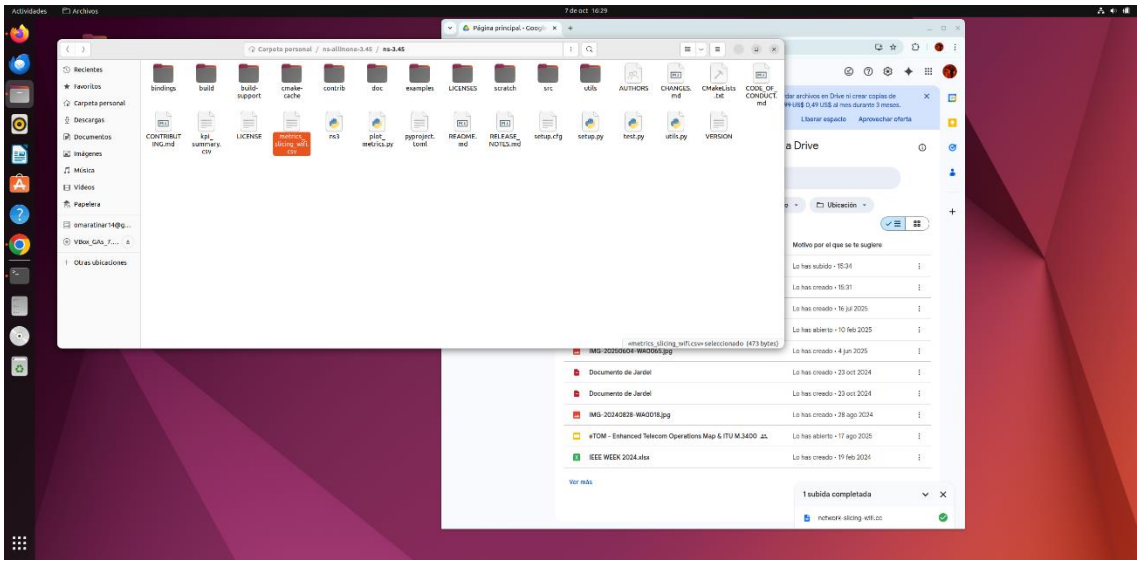
```
./ns3 build
./ns3 run scratch/network-slicing-wifi
```





8) Visualizar las metricas en CSV

Para ello ir a la carpeta /home/$USER/ns-allinone-3.45 y buscarla con el nombre metrics_slicing_wifi.csv



9) Generar archivo en Python para mejor visualización de KPIs

En la misma carpeta generar un nuevo documento de texto en el que introducirán el siguiente código:

```
#!/usr/bin/env python3

# plot_metrics.py

# Lee metrics_slicing_wifi.csv y crea gráficos KPI (Throughput, Delay, Packet Loss %, Jitter, Tx/Rx bytes).

# Salidas: /mnt/data/kpi_*.png (en el entorno Jupyter). En una VM local las imágenes se guardan en la carpeta actual.


import os

import pandas as pd

import matplotlib.pyplot as plt


csv_path = "metrics_slicing_wifi.csv"  # Cambia la ruta si tu CSV está en otra carpeta


if not os.path.exists(csv_path):

    raise FileNotFoundError(f"'{csv_path}' no encontrado. Coloca el CSV en la misma carpeta o ajusta csv_path.")


# Leer CSV

df = pd.read_csv(csv_path)
```

```
# Forzar tipos numéricos (si alguna columna falta, la crea como NaN)
numeric_cols = ['Throughput_Mbps', 'Delay_s', 'LostPackets', 'PacketLossRate', 'MeanJitter_s', 'TxBytes', 'RxBytes']
for col in numeric_cols:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
    else:
        df[col] = pd.NA

# Crear columna PacketLossPct (en %). Si el CSV trae fracción (0..1) la multiplica por 100.
if df['PacketLossRate'].notna().any():
    if df['PacketLossRate'].max(skipna=True) <= 1.0:
        df['PacketLossPct'] = df['PacketLossRate'] * 100.0
    else:
        df['PacketLossPct'] = df['PacketLossRate']
else:
    df['PacketLossPct'] = pd.NA

# Convertir FlowID a str para las etiquetas
if 'FlowID' in df.columns:
    labels = df['FlowID'].astype(str)
else:
    # si no hay FlowID, usar índices
    labels = df.index.astype(str)
    df['FlowID'] = df.index

# Helper para guardar figuras en la carpeta actual
def save_fig(fig, filename):
    out = os.path.join(os.getcwd(), filename)
    fig.savefig(out, bbox_inches='tight')
    print(f"Guardado: {out}")
```

```
    plt.close(fig)
    return out

# 1) Throughput (Mbps)
fig = plt.figure(figsize=(8,4))
plt.bar(labels, df['Throughput_Mbps'])
plt.xlabel("FlowID")
plt.ylabel("Throughput (Mbps)")
plt.title("KPI: Throughput por Flow")
plt.grid(axis='y', linestyle='--', linewidth=0.5)
save_fig(fig, "kpi_throughput.png")

# 2) Delay (s)
fig = plt.figure(figsize=(8,4))
plt.bar(labels, df['Delay_s'])
plt.xlabel("FlowID")
plt.ylabel("Delay (s)")
plt.title("KPI: Delay promedio por Flow")
plt.grid(axis='y', linestyle='--', linewidth=0.5)
save_fig(fig, "kpi_delay.png")

# 3) Packet Loss (%)
fig = plt.figure(figsize=(8,4))
plt.bar(labels, df['PacketLossPct'])
plt.xlabel("FlowID")
plt.ylabel("Packet Loss (%)")
plt.title("KPI: Tasa de pérdida de paquetes por Flow")
plt.grid(axis='y', linestyle='--', linewidth=0.5)
save_fig(fig, "kpi_packetloss.png")

# 4) Mean Jitter (s)
```

```
fig = plt.figure(figsize=(8,4))
plt.bar(labels, df['MeanJitter_s'])
plt.xlabel("FlowID")
plt.ylabel("Mean Jitter (s)")
plt.title("KPI: Jitter medio por Flow")
plt.grid(axis='y', linestyle='--', linewidth=0.5)
save_fig(fig, "kpi_jitter.png")

# 5) Tx / Rx bytes (gráfica comparativa - barras apiladas)
fig = plt.figure(figsize=(8,4))
tx = df['TxBytes'].fillna(0)
rx = df['RxBytes'].fillna(0)
ind = range(len(df))
plt.bar(ind, tx, label='TxBytes')
plt.bar(ind, rx, bottom=tx, label='RxBytes')
plt.xticks(ind, labels)
plt.xlabel("FlowID")
plt.ylabel("Bytes")
plt.title("Tx / Rx Bytes por Flow (stacked)")
plt.legend()
plt.grid(axis='y', linestyle='--', linewidth=0.5)
save_fig(fig, "kpi_tx_rx_bytes.png")

# Resumen simple
summary = {
   "total_throughput_Mbps": float(df['Throughput_Mbps'].sum(skipna=True)),
   "mean_delay_s": float(df['Delay_s'].mean(skipna=True)),
   "mean_packet_loss_pct": float(df['PacketLossPct'].mean(skipna=True)),
   "mean_jitter_s": float(df['MeanJitter_s'].mean(skipna=True))
}
print("\nResumen KPI (agregado):")
```

```
for k, v in summary.items():
    print(f" - {k}: {v}")


# También exportar resumen a CSV
summary_df = pd.DataFrame([summary])
summary_df.to_csv("kpi_summary.csv", index=False)
print("Guardado: kpi_summary.csv")


print("\nHecho. Revisa los PNG generados en la carpeta actual.")
```

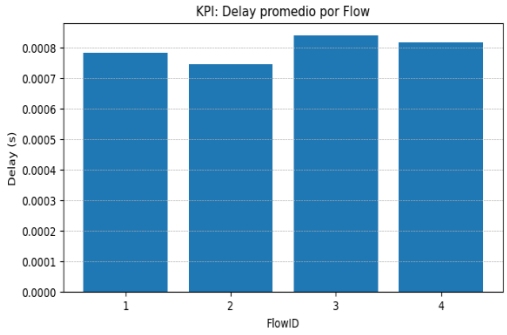
Al terminar **guardar** el archivo como **“plot_metrics.py”.**

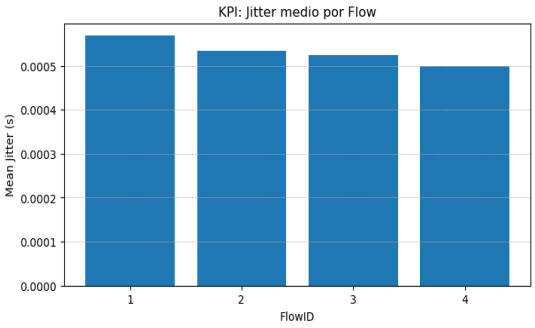
9) EJECUTAR SCRIPT DE PYTHON

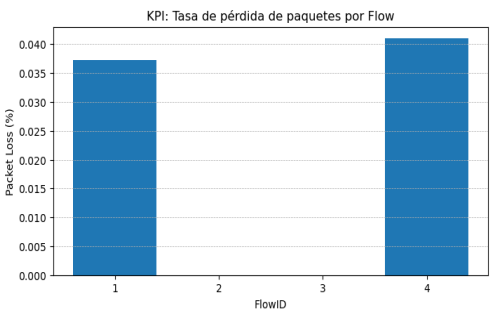
- Coloca metrics_slicing_wifi.csv y plot_metrics.py en la misma carpeta (por ejemplo: ~/ns-allinone-3.45/ns-3.45).
- Abre terminal en esa carpeta.
- Ejecuta:

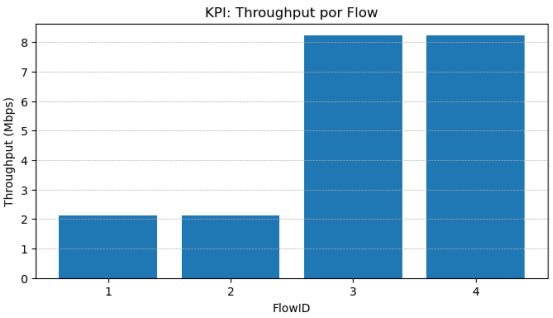
```
python3 plot_metrics.py
```

Luego podras visualizar las diferentes métricas en imágenes .PNG tal y como se muestra en las siguientes imágenes:






KPI: Tasa de pérdida de paquetes por Flow
Packet Loss (%)
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
1
2
3
4
FlowID


KPI: Throughput por Flow
Throughput (Mbps)
8
7
6
5
4
3
2
1
0
1
2
3
4
FlowID