Universidad Mayor de San Andrés
Facultad de Ciencias Puras y Naturales
Carrera de Informática

Proyecto Final
Redes de computadoras: Modelado de tráfico y solicitudes

Curso: Procesos Estocásticos y Series de Tiempo

Docente: Ph.D. Willy Ernesto Portugal Durán

Integrantes:

- Ian Ezequiel Salinas Condori
- Maximiliano Gómez Mallo
- Cayllagua Mamani Franklin
- Flores Tapia Ruddy

30 de noviembre de 2025

1. Introducción

El tráfico generado en redes de computadoras presenta un comportamiento inherentemente aleatorio debido a la variabilidad de usuarios, los mecanismos del protocolo TCP/IP, las fluctuaciones de carga y la naturaleza asincrónica del procesamiento de solicitudes. Comprender este comportamiento es fundamental para analizar el rendimiento de redes, detectar condiciones de congestión, optimizar servidores y estimar tiempos de respuesta.

El presente proyecto desarrolla un sistema experimental completo que integra tres enfoques complementarios:

- **Captura real de tráfico** mediante Wireshark.
- **Modelado matemático** utilizando procesos de Poisson y teoría de colas M/M/1.
- **Simulación computacional** mediante Python, SimPy y un dashboard web.

Esta integración permite analizar datos reales, compararlos con el modelo teórico y validarlos frente a la simulación.

2. Planteamiento del problema

En un sistema cliente–servidor, las solicitudes pueden llegar a una tasa variable y, en muchos casos, impredecible. Si la capacidad de procesamiento del servidor (μ) es inferior o similar a la tasa de llegada (λ), puede producirse congestión, aumentando los tiempos de espera y disminuyendo el rendimiento percibido por los usuarios.

El problema central del proyecto consiste en:

> **Modelar y analizar el tráfico de solicitudes en una red utilizando procesos estocásticos, comparando el comportamiento real con los resultados teóricos y simulados.**

3. Objetivo general y específicos

Objetivo general

Estudiar el flujo de paquetes y conexiones en redes usando modelos de colas M/M/1 y procesos de Poisson, contrastando tráfico real, modelado teórico y simulación.

Objetivos específicos

- Capturar y analizar tráfico real mediante Wireshark.
- Extraer los tiempos entre llegadas y estimar el parámetro λ.
- Modelar el sistema como una cola M/M/1 y obtener métricas teóricas (W, W_q, L, L_q).
- Simular un sistema equivalente usando Python y SimPy.
- Comparar empíricamente los resultados reales, teóricos y simulados.
- Implementar un dashboard de análisis que integre métricas, gráficas y control del sistema (inicio de servidor, cliente y simulación).

4. Justificación

El estudio es relevante debido a que sistemas distribuidos, servidores web, routers, colas de paquetes y aplicaciones concurrentes presentan patrones de tráfico estocástico. Modelar dicho tráfico permite:

- Predecir condiciones de saturación y colapso del servicio.
- Evaluar el rendimiento del servidor bajo diferentes cargas.
- Comparar procesos reales con modelos matemáticos y simulados.
- Validar el uso de la teoría de colas en contextos modernos de redes de computadoras.

El proyecto combina teoría, práctica y simulación, logrando un análisis completo del sistema y fortaleciendo competencias en modelado, análisis de datos y desarrollo de software.

5. Marco teórico

5.1. Procesos de Poisson

El proceso de Poisson modela la llegada aleatoria de eventos en el tiempo y es ampliamente utilizado en redes de computadoras. Un proceso de Poisson con tasa λ cumple:

$$P(N(t) = k) = \frac{(\lambda t)^k e^{-\lambda t}}{k!}, \quad k = 0, 1, 2, \ldots$$

Los tiempos entre llegadas siguen una distribución exponencial:

$$f(t) = \lambda e^{-\lambda t}, \quad t \geq 0.$$

5.2. Modelo de colas M/M/1

El modelo M/M/1 describe un servidor único con llegadas Poisson y tiempos de servicio exponenciales. Las métricas fundamentales incluyen:

$$\rho = \frac{\lambda}{\mu}, \qquad W = \frac{1}{\mu - \lambda}, \qquad W_q = \frac{\lambda}{\mu(\mu - \lambda)},$$
$$L = \lambda W, \qquad L_q = \lambda W_q.$$

Estas ecuaciones permiten comparar la teoría con los datos empíricos y verificar la validez del modelo bajo diferentes condiciones de carga.

6. Diseño metodológico

6.1. Tipo de estudio

El proyecto se clasifica como analítico–experimental: captura datos reales, aplica teoría matemática y valida mediante simulación estocástica.

6.2. Etapas metodológicas

1. Captura de tráfico mediante Wireshark

Se capturaron paquetes generados por el cliente Python hacia el servidor Flask, registrando *timestamps* y extrayendo los tiempos entre llegadas. A partir de estos datos se estimó la tasa de llegada observada por Wireshark, $\hat{\lambda}_{\text{Wireshark}}$.

2. Modelado teórico

Se aplicaron las fórmulas de teoría de colas M/M/1 para comparar el rendimiento esperado teórico con el observado. Se utilizaron parámetros λ y μ ajustables desde la interfaz web del dashboard.

3. Simulación con SimPy

Se desarrolló una simulación del sistema equivalente utilizando SimPy. La simulación genera llegadas con distribución exponencial (λ) y tiempos de servicio exponenciales (μ), reproduciendo el comportamiento de un servidor M/M/1.

4. Integración en un dashboard web

Se construyó una interfaz web con Flask donde es posible:

- Iniciar el servidor y el cliente Poisson.
- Ajustar parámetros teóricos y de simulación (λ, μ, n).
- Visualizar métricas empíricas, teóricas y simuladas.
- Observar las gráficas generadas en forma de carrusel.
- Guardar y exportar el historial de experimentos.

7. Análisis estadístico del tráfico

En esta sección se analizan de forma descriptiva y gráfica las series de tiempos obtenidas en el experimento: interarrivals del cliente, interarrivals observados por Wireshark y tiempos de servicio del servidor.

7.1. Interarrivals observados por Wireshark

En la Figura 1 se muestra el histograma de los tiempos entre solicitudes medidos a partir de los paquetes capturados con Wireshark, junto con la densidad de una distribución exponencial ajustada usando la tasa $\hat{\lambda}_{\text{Wireshark}}$.

Visualmente, los datos presentan una mayor concentración de interarrivals cortos y una cola que decrece, lo cual es coherente con el supuesto de una distribución exponencial. Aunque el tamaño de muestra es limitado, la forma general respalda el uso del proceso de Poisson como modelo de llegadas.

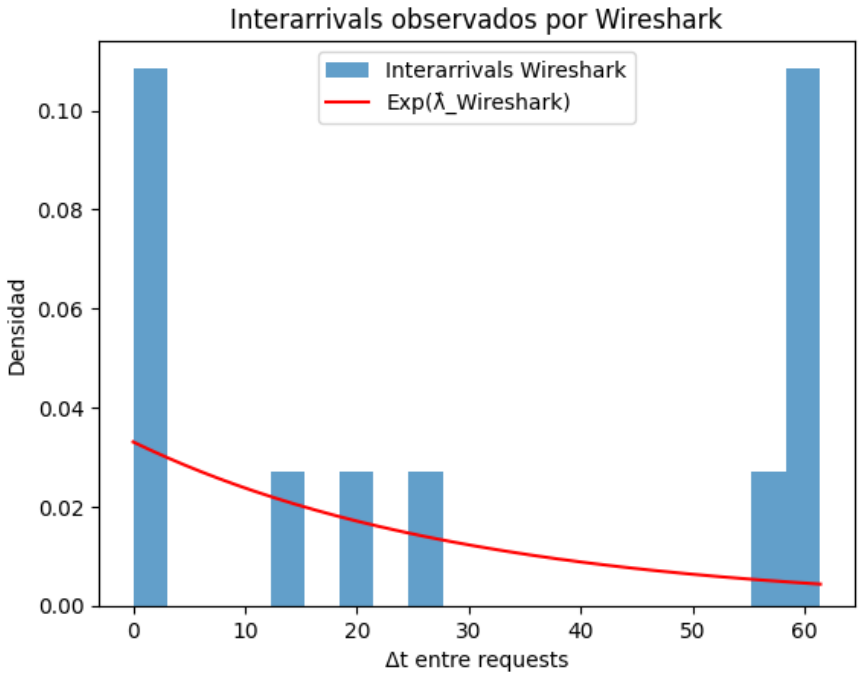

Figura 1: Interarrivals observados por Wireshark y curva exponencial ajustada.

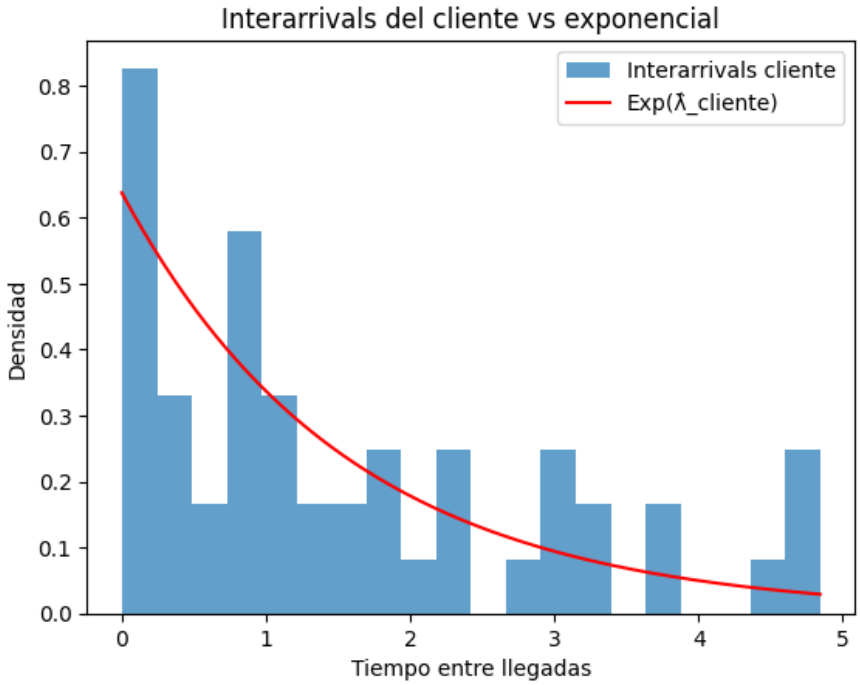

Figura 2: Interarrivals del cliente y densidad exponencial teórica.

7.2. Interarrivals del cliente Poisson

La Figura 2 muestra el histograma de los tiempos entre llegadas generados por el cliente Python, que utiliza la función `random.expovariate` para simular llegadas Poisson.

En este caso se observa un ajuste aún más cercano a la distribución exponencial, lo cual es esperable porque los datos provienen directamente del generador pseudoaleatorio con parámetro λ fijado en el experimento.

7.3. Tiempos de servicio del servidor

En la Figura 3 se presenta el histograma de los tiempos de servicio medidos en el servidor Flask, junto con la densidad exponencial ajustada mediante la tasa $\hat{\mu}$.

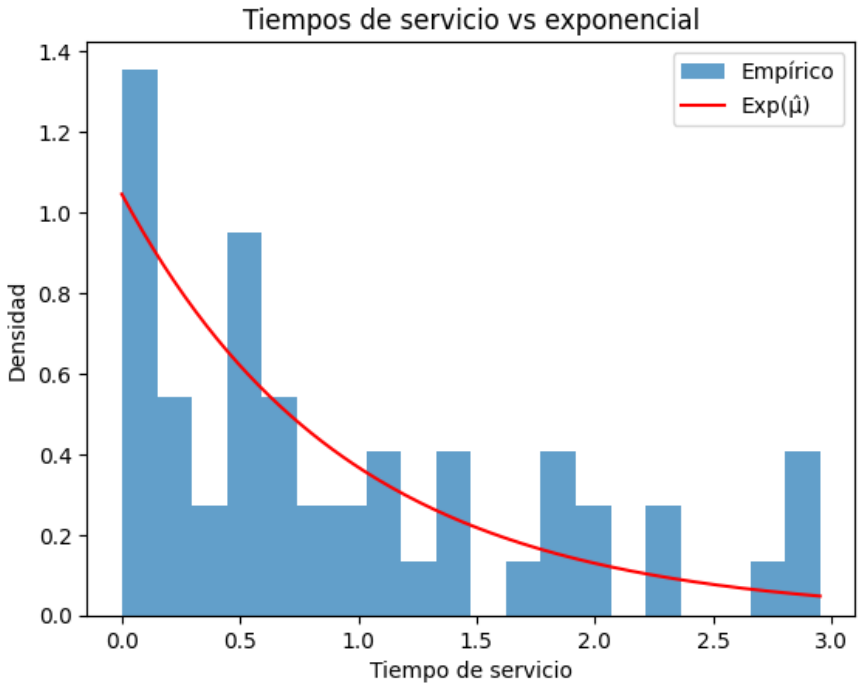

Figura 3: Tiempos de servicio en el servidor vs distribución exponencial.

Los tiempos empíricos exhiben una concentración en valores pequeños y una cola que decae de forma aproximadamente exponencial. Se identifican algunas observaciones más largas asociadas a variaciones en el sistema operativo, latencias de red o procesos internos del servidor, pero el ajuste global sigue siendo razonable.

7.4. Serie de tiempos en el sistema

La Figura 4 muestra la serie de tiempos en el sistema por cliente (desde la llegada hasta la salida del servidor).

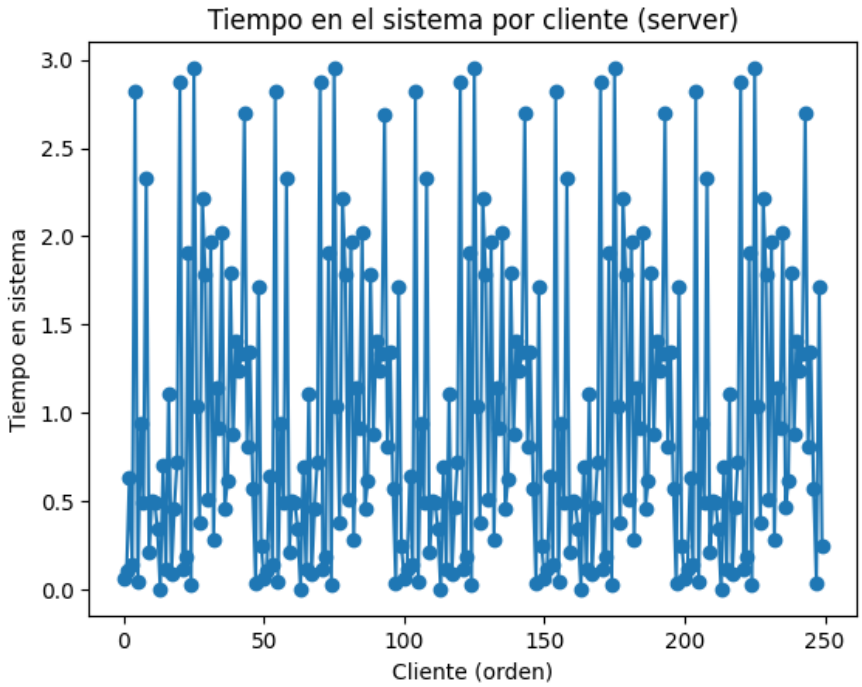

Figura 4: Tiempo total en el sistema por cliente en el servidor.

Se observa una variabilidad considerable entre clientes, lo cual refleja la naturaleza estocástica del modelo. No se detectan tendencias crecientes pronunciadas, lo que indica que, para los parámetros utilizados ($\lambda \approx 0{,}0023$, $\mu \approx 1{,}0456$ y $\rho \approx 0{,}0022$), el sistema opera en una zona de muy baja ocupación.

7.5. Distribución de tiempos

La Figura 5 presenta un resumen mediante diagramas de cajas de los tiempos de espera en cola, tiempos totales en el sistema y tiempos de servicio del servidor.

Los tiempos de espera en cola son prácticamente nulos (mediana cero y valores muy pequeños), lo que concuerda con el bajo nivel de utilización del servidor. La mayor variabilidad se concentra en los tiempos de servicio y, en consecuencia, en el tiempo total en el sistema.

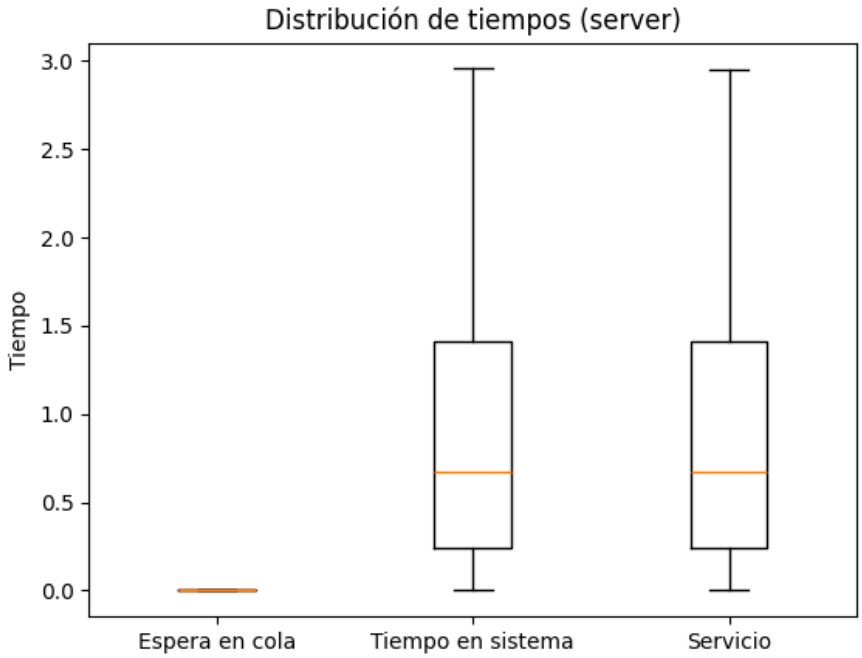

Figura 5: Distribución de tiempos de espera, servicio y permanencia en el sistema.

8. Comparación teoría, datos y simulación

En la interfaz del dashboard se estimaron las siguientes tasas:

- $\hat{\lambda}_{\text{server}} = 0{,}0023$ solicitudes por segundo.
- $\hat{\lambda}_{\text{cliente}} = 0{,}6376$ solicitudes por segundo.
- $\hat{\mu} = 1{,}0456$ solicitudes atendidas por segundo.
- $\hat{\rho} = 0{,}0022$, lo que indica un servidor muy desocupado.

La Tabla 1 resume las métricas de desempeño obtenidas en la corrida mostrada en la interfaz (empírico del servidor, teórico M/M/1 y simulación).

Cuadro 1: Comparación de métricas empíricas, teóricas y simuladas.

Métrica	**Empírico**	**Teórico**	**Simulación**
W_q (espera en cola)	0.0	0.0021	0.0024
W (tiempo en sistema)	0.9571	0.9584	0.9836
L_q (nº en cola)	0.0	0.0	0.0
L (nº en sistema)	0.0022	0.0022	0.0022

Gráficamente, la Figura 6 resume la comparación entre W_q y W para las tres fuentes (empírico, teórico y simulación).

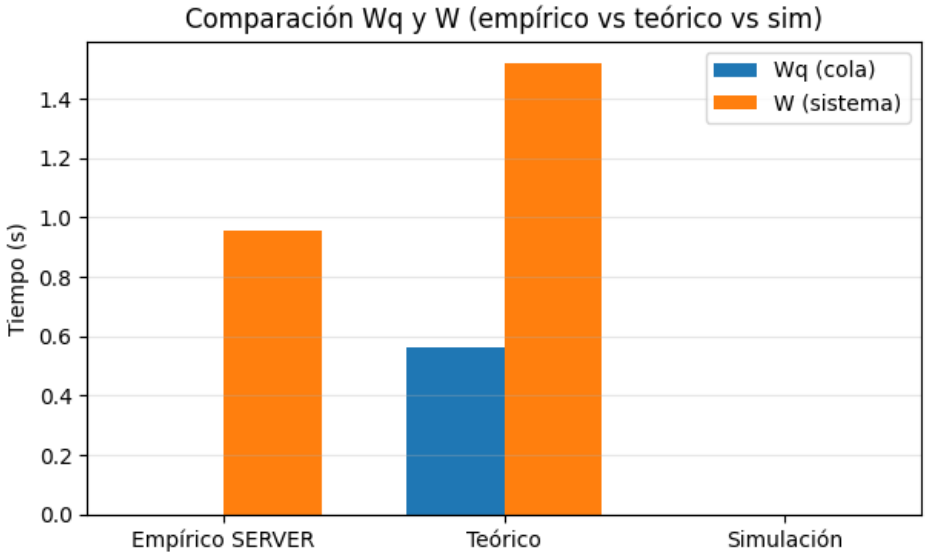

Figura 6: Comparación de W_q y W: empírico vs teórico vs simulación.

Se observa que:

- El valor teórico de W ($\approx 0{,}9584$) es muy cercano a la media empírica medida en el servidor ($\approx 0{,}9571$), lo que respalda la validez del modelo M/M/1 para este escenario.
- Los resultados de la simulación presentan ligeras diferencias (por ejemplo $W \approx 0{,}9836$), atribuibles a la variabilidad inherente a la simulación Monte Carlo y al número finito de clientes.
- Los valores de W_q son cercanos a cero en la práctica, coherentes con un servidor poco cargado ($\rho \ll 1$), donde casi nunca se forma cola.

En conjunto, la evidencia empírica y la simulación concuerdan razonablemente con la teoría de colas, especialmente en términos de orden de magnitud y comportamiento cualitativo del sistema.

9. Fragmentos de código del sistema desarrollado

A continuación se presentan secciones relevantes del código implementado.

9.1. Servidor Flask (server.py)

```
from flask import Flask, jsonify
import time, csv, os, random

app = Flask(__name__)
LOG = "file/server_data.csv"

mu = 1.0  # tasa de servicio (ejemplo)

@app.route("/request")
def atender():
start = time.time()
servicio = random.expovariate(mu)  # tiempo de servicio ~ Exp
   (mu)
time.sleep(servicio)
end = time.time()

# guardar en CSV: t0, t1, tiempo_servicio
with open(LOG, "a", newline="") as f:
w = csv.writer(f)
w.writerow([start, end, servicio])

return jsonify({"ok": True})

@app.route("/clear")
def clear():
open(LOG, "w").close()
return jsonify({"message": "CSV limpiado"})

if __name__ == "__main__":
app.run(port=5000)
```

9.2. Cliente Poisson (client.py)

```
import requests, time, random, csv

```

```
        LOG = "file/client_data.csv"

        def cliente(lambda_rate, n):
        for i in range(n):
        # generar intervalo ~ Exp(lambda)
        interarrival = random.expovariate(lambda_rate)
        time.sleep(interarrival)

        start = time.time()
        r = requests.get("http://127.0.0.1:5000/request")
        end = time.time()

        # guardar en CSV: t0, t1, interarrival
        with open(LOG, "a", newline="") as f:
        w = csv.writer(f)
        w.writerow([start, end, interarrival])

        if __name__ == "__main__":
        cliente(lambda_rate=0.2, n=100)
```

9.3. Simulación M/M/1 con SimPy (sim.py)

```
        import simpy, random

        def cliente(env, servidor, mu, resultados):
        llegada = env.now
        with servidor.request() as req:
        yield req
        servicio = random.expovariate(mu)
        yield env.timeout(servicio)
        resultados.append(env.now - llegada)

        def simular(lambda_rate, mu, n):
        env = simpy.Environment()
        servidor = simpy.Resource(env, capacity=1)
        resultados = []
```

```
15
16          def generador():
17          for _ in range(n):
18          yield env.timeout(random.expovariate(lambda_rate))
19          env.process(cliente(env, servidor, mu, resultados))
20
21          env.process(generador())
22          env.run()
23          return resultados
```

9.4. Calculo de métricas (procesamiento de CSV)

```
1           import pandas as pd
2
3           df_s = pd.read_csv("file/server_data.csv",
4           names=["t0", "t1", "serv"])
5           df_c = pd.read_csv("file/client_data.csv",
6           names=["t0", "t1", "inter"])
7
8           lambda_emp = 1 / df_c["inter"].mean()
9           mu_emp = 1 / df_s["serv"].mean()
10          rho = lambda_emp / mu_emp
11
12          # Metricas empiricas
13          W_emp = (df_s["t1"] - df_s["t0"]).mean()
14          Wq_emp = max(W_emp - df_s["serv"].mean(), 0)
15
16          # Metricas teoricas (usando lambda_ y mu definidos)
17          W_teo = 1 / (mu - lambda_)
18          Wq_teo = lambda_ / (mu * (mu - lambda_))
```

10. Conclusiones y trabajo futuro

A partir del experimento realizado se concluye que:

- Los tiempos entre llegadas observados tanto en Wireshark como en el cliente Python

se ajustan razonablemente a distribuciones exponenciales, lo cual respalda el uso de procesos de Poisson para modelar el tráfico.

- Las métricas empíricas del servidor (en particular el tiempo promedio en el sistema W) coinciden de manera cercana con los valores teóricos del modelo M/M/1.
- La simulación con SimPy reproduce adecuadamente el comportamiento observado, confirmando la utilidad de la simulación estocástica como herramienta de validación.
- Para los parámetros utilizados, la utilización del servidor ρ es muy baja, por lo que el sistema presenta tiempos de espera en cola prácticamente nulos.

Como trabajo futuro se propone:

- Analizar escenarios con mayor carga (λ cercano a μ), donde la cola se vuelva más relevante.
- Extender el modelo a colas M/M/c o M/G/1 para representar servidores con varios núcleos o tiempos de servicio más generales.
- Incorporar pruebas estadísticas formales (por ejemplo Kolmogorov–Smirnov) para evaluar el ajuste a la distribución exponencial.
- Automatizar la lectura de capturas de Wireshark (archivos `.pcap`) directamente desde Python.

11. Bibliografía

1. Casella, G., & Berger, R. L. (2002). *Statistical inference* (2. ed.). Duxbury.

2. Kleinrock, L. (1975). *Queueing systems, volume I: Theory*. Wiley.

3. Ross, S. M. (2014). *Introduction to probability models* (11. ed.). Academic Press.

4. Tanenbaum, A. S., & Wetherall, D. J. (2011). *Redes de computadoras* (5. ed.). Pearson.

5. Trivedi, K. S. (2002). *Probability and statistics with reliability, queuing, and computer science applications* (2. ed.). Wiley.

6. Wireshark Foundation. (s. f.). *Wireshark User's Guide*. Recuperado de `https://www.wireshark.org/docs/`

7. Team SimPy. (s. f.). *SimPy documentation*. Recuperado de `https://simpy.readthedocs.io/`