FACULTAD DE INGENIERIA
Universidad de Buenos Aires

TP N°1: File Transfer

[75.43] Introducción a los Sistemas Distribuidos
Cátedra Hamelin
Segundo Cuatrimestre 2025

Alumno:	Número de Padrón	Email
ALDONATE, Lucas	100030	ladonate@fi.uba.ar
CASTAÑO, Mateo	110114	macastano@fi.uba.ar
RUIZ DIAZ Carolina	111442	cruiz@fi.uba.ar
ALVAREZ, Cristobal	110308	calvarez@fi.uba.ar
THOME, Milagros	110684	mthome@fi.uba.ar

Índice

1. Resumen / Abstract

El presente trabajo práctico tiene como objetivo la creación de una aplicación de red basada en UDP, implementando un protocolo de transferencia de datos confiable. Como requisito, se desarrollaron dos versiones del protocolo: una utilizando **Stop & Wait** y otra basada en el mecanismo de **error-recovery Go-Back-N**. El trabajo permitió reforzar y profundizar en diversos conceptos de la capa de red, tales como: *TCP*, *UDP*, *reliable data transfer*, *sockets* y *packet loss*.

2. Objetivos

2.1. Objetivo general

El objetivo principal del trabajo es permitir la **transferencia de archivos binarios**, tanto en modalidad *UPLOAD* como *DOWNLOAD*, mediante el desarrollo de una aplicación de red confiable sobre el protocolo **UDP**. Para ello, se implementó una arquitectura **cliente-servidor** que posibilita la comunicación y la gestión de dichas operaciones.

2.2. Objetivos específicos

- Implementar y comparar dos mecanismos de transferencia confiable: *Stop & Wait* y *Go-Back-N*.
- Diseñar la estructura de paquetes y su manejo de errores/retransmisiones.
- Validar la transferencia de archivos de distintos tamaños en escenarios con y sin pérdida de paquetes.
- Analizar el desempeño en términos de tiempo de transferencia y eficiencia de los protocolos.

3. Marco Teórico

3.1. Protocolo UDP

El **User Datagram Protocol (UDP)** es un protocolo perteneciente a la capa de transporte. Se caracteriza por ser **no orientado a la conexión** y por ofrecer un servicio de envío de datagramas sin mecanismos de control de flujo ni de retransmisión automática. Esto lo convierte en un protocolo liviano y eficiente en términos de latencia, pero sin garantías de entrega, orden o ausencia de duplicados.

3.2. Limitaciones

Las principales limitaciones de UDP radican en que:

- No asegura la entrega de los paquetes (posible pérdida).
- No garantiza el orden de llegada de los datagramas.
- No detecta ni corrige errores más allá del checksum básico.

Por estas razones, cuando se requiere confiabilidad, deben implementarse mecanismos adicionales en la capa de aplicación.

3.3. Teoría de Stop-and-Wait y Go-Back-N

Los mecanismos de **reliable data transfer** buscan garantizar la entrega correcta y ordenada de los datos sobre un medio no confiable como UDP.

- **Stop-and-Wait:** el emisor envía un paquete y espera la confirmación (ACK) antes de enviar el siguiente. Este método es simple de implementar, pero puede resultar ineficiente ante enlaces de alta latencia, ya que mantiene al emisor inactivo mientras espera el ACK.
- **Go-Back-N:** el emisor puede transmitir múltiples paquetes sin esperar confirmaciones inmediatas, utilizando una *ventana deslizante*. Si se detecta pérdida de un paquete, el receptor solicita la retransmisión desde el paquete perdido en adelante. Esto mejora la eficiencia, pero puede aumentar la cantidad de retransmisiones si la tasa de pérdida es alta.

3.4. Comparación entre ambos

- **Eficiencia:** Go-Back-N es más eficiente que Stop-and-Wait, ya que permite un mayor aprovechamiento del canal.
- **Complejidad:** Stop-and-Wait es más sencillo de implementar, mientras que Go-Back-N requiere mayor manejo de buffers y control de ventana.
- **Escenarios de uso:** Stop-and-Wait es adecuado para entornos de baja latencia y archivos pequeños, mientras que Go-Back-N es preferible en conexiones más largas o donde se transfieren archivos grandes.

4. Diseño e Implementación

4.1. Arquitectura del sistema

4.1.1. Cliente-servidor

El sistema implementa una arquitectura cliente-servidor tradicional sobre UDP. El servidor (`src/lib/server/server.py`) actúa como un receptor que puede manejar múltiples clientes concurrentemente utilizando threads separados para cada conexión. Cada cliente (`src/lib/client/client.py`) se conecta al servidor para realizar operaciones de upload o download de archivos.

Características principales:

- **Servidor concurrente**: Utiliza threading para manejar múltiples clientes simultáneamente
- **Comunicación UDP**: Implementa su propio protocolo de confiabilidad sobre UDP
- **Cola de escritura**: El servidor utiliza una cola thread-safe (`queue.Queue`) para gestionar el envío de paquetes
- **Gestión de conexiones**: Cada cliente recibe un thread dedicado y una cola de mensajes individual

4.1.2. Flujo de datos y comunicación

El flujo de comunicación sigue un patrón de tres fases:

1. **Handshake inicial**: Negociación del tipo de operación (UPLOAD/DOWNLOAD), protocolo (SW/GBN) y nombre del archivo
2. **Transferencia de datos**: Implementación del protocolo seleccionado con control de flujo y recuperación de errores
3. **Finalización**: Confirmación de transferencia completa y cierre de conexión

4.2. Implementación de protocolos

4.2.1. Triple-handshake de request

El sistema implementa un handshake de tres pasos para establecer la conexión:

1. **Tipo de conexión**: El cliente envía "U" (Upload) o "D" (Download)
2. **Protocolo**: El cliente especifica "SW" (Stop-and-Wait) o "GBN" (Go-Back-N)
3. **Nombre del archivo**: El cliente envía el nombre del archivo a transferir

Cada paso utiliza Stop-and-Wait con timeout de 0.1 segundos y máximo 70 reintentos para garantizar la entrega.

En la Figura 1 se observa el intercambio inicial de mensajes (*handshake*) correspondiente a una transferencia *upload* bajo el protocolo Stop & Wait. Durante esta etapa, el cliente (`10.0.0.2`) inicia la comunicación enviando una solicitud de carga (`Upload Request`) al servidor (`10.0.0.1`). El servidor responde con una confirmación de recepción y la selección del protocolo a utilizar (`Stop-and-Wait Protocol`). A continuación, se intercambian los mensajes de negociación del nombre de archivo (`uploadsw.png`) y los primeros acuses de recibo (`ACK 0`, `ACK 1`, `ACK 2`), lo que confirma el establecimiento exitoso de la sesión de transferencia.

6	0.940227	10.0.0.2	10.0.0.1	UDPFT	43 Handshake: Upload Request
7	0.940823	10.0.0.1	10.0.0.2	UDPFT	46 ACK 0
8	0.940845	10.0.0.2	10.0.0.1	UDPFT	44 Handshake: Stop-and-Wait Protocol
9	0.941043	10.0.0.1	10.0.0.2	UDPFT	46 ACK 1
10	0.941057	10.0.0.2	10.0.0.1	UDPFT	54 Handshake: uploadsw.png
12	1.041223	10.0.0.2	10.0.0.1	UDPFT	54 Handshake: uploadsw.png
13	1.041490	10.0.0.1	10.0.0.2	UDPFT	46 ACK 2
14	1.041500	10.0.0.1	10.0.0.2	UDPFT	46 ACK 2

Figura 1: Intercambio de mensajes durante el *handshake* inicial en una transferencia *upload* con Stop & Wait.

4.2.2. Stop-and-Wait

La implementación de Stop-and-Wait se basa en el principio de enviar y esperar confirmación. Es analogo a Go Back N con un tamaño de ventana n=1. Cada paquete incluye:

- **Número de secuencia**: Alterna entre valores incrementales (3, 4, 5, ...)
- **Timeout**: 0.05 segundos por paquete
- **Reintentos**: Hasta 70 intentos por paquete

1. **Upload Stop-and-Wait:**
   El cliente envía paquetes secuencialmente y espera ACK antes de continuar. El servidor:
   - Procesa paquetes en orden secuencial
   - Envía ACK del siguiente paquete esperado (`pkg_id`)
   - Para paquetes fuera de orden, reenvía el `pkg_id` del paquete que espera.
2. **Download Stop-and-Wait:**
   El servidor envía paquetes y el cliente confirma cada uno. El cliente:
   - Procesa solo paquetes en orden secuencial.
   - Envía ACK del siguiente paquete esperado.
   - Ignora paquetes fuera de orden y reenvía el último ACK válido.
3. **Ventajas y Desventajas:**
   - **Ventajas**: Simplicidad, confiabilidad garantizada, control estricto de orden.
   - **Desventajas**: Baja utilización del ancho de banda, ineficiente en enlaces de alta latencia.

4.2.3. Go-Back-N

La implementación utiliza una ventana deslizante con tamaño configurable (por defecto 10 paquetes). Características:

Upload Go-Back-N:

- **Ventana deslizante**: Envía hasta 10 paquetes sin esperar ACK.
- **Control de flujo**: Mantiene buffer de paquetes sin confirmar.
- **Recuperación**: En caso de timeout, reenvía todos los paquetes sin ACK.
- **Timeout**: 0.05 segundos para ventana completa.

Download Go-Back-N:

- **Recepción ordenada**: Procesa solo paquetes en secuencia.
- **ACK acumulativo**: Envía ACK del próximo paquete esperado.
- **Descarte**: Ignora paquetes fuera de orden, en su lugar envía el `pkg_id` del próximo paquete esperado.

Ventajas y Desventajas:

- **Ventajas**: Mayor throughput, mejor utilización del ancho de banda
- **Desventajas**: Complejidad mayor, posibles reenvíos innecesarios

4.2.4. Flujo de envío de datos en Go-Back-N

La Figura 2 muestra un extracto de la captura de red obtenida durante una transferencia *upload* utilizando el protocolo Go-Back-N. En el flujo se observa el envío continuo de múltiples segmentos de datos (`DATA`) desde el cliente (`10.0.0.3`) hacia el servidor (`10.0.0.1`), correspondientes a los identificadores de paquete `ID=3` a `ID=21`. Simultáneamente, el servidor responde con los acuses de recibo (`ACK`) que confirman la correcta recepción de los paquetes. La secuencia evidencia el comportamiento característico del protocolo Go-Back-N: mantiene una **ventana deslizante** de transmisión que permite enviar varios paquetes antes de recibir sus respectivos `ACK`, optimizando así la utilización del canal de comunicación.

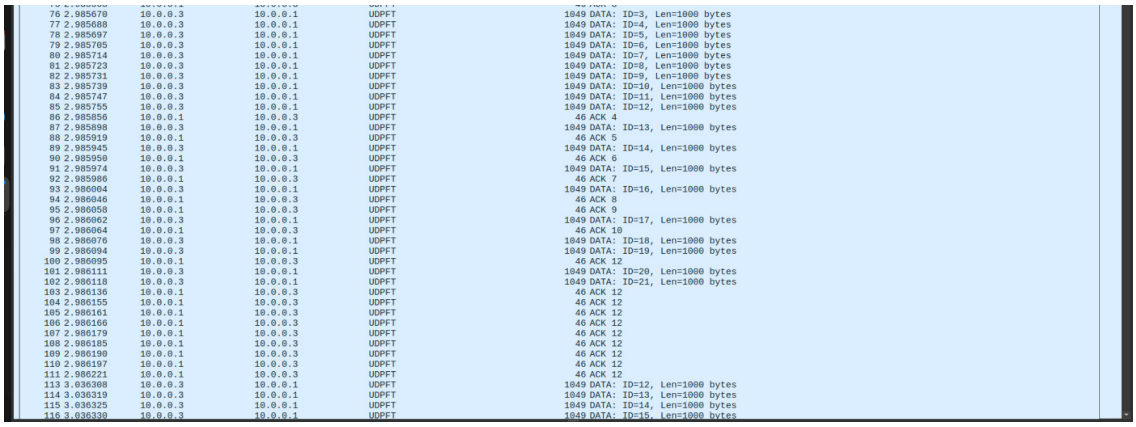

Figura 2: Flujo de envío y recepción de paquetes en una transferencia *upload* mediante Go-Back-N.

4.2.5. Flujo de envío de datos en Stop and Wait

La Figura 3 ilustra el comportamiento del protocolo Stop & Wait durante una transferencia *upload*. En la captura puede observarse el intercambio secuencial entre el cliente (`10.0.0.4`) y el servidor (`10.0.0.1`), donde cada paquete de datos (`DATA`) es seguido por su respectivo acuse de recibo (`ACK`) antes de que se envíe el siguiente. Este patrón confirma la naturaleza del protocolo: solo un paquete se mantiene "en vuelo" en cada instante, lo que limita el aprovechamiento del canal pero garantiza un control estricto de errores y orden de entrega.

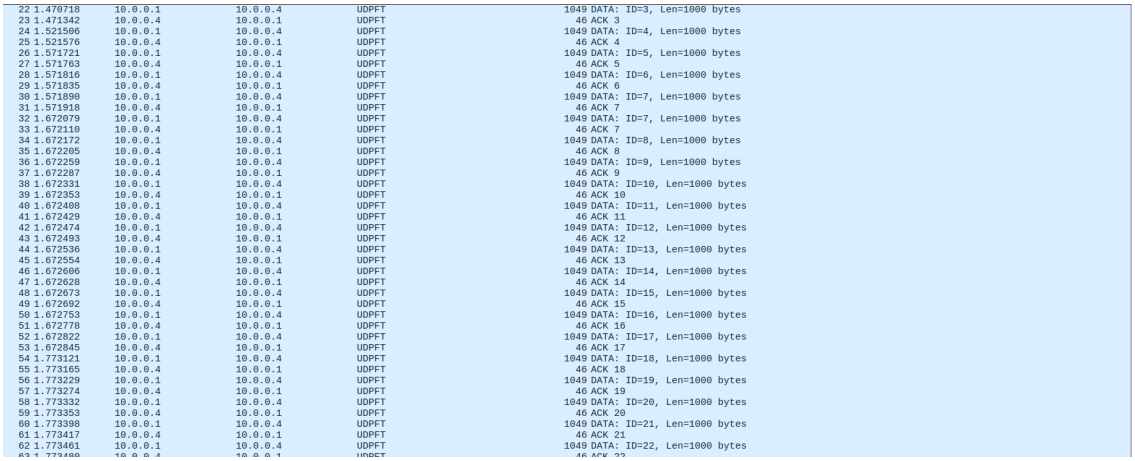

Figura 3: Flujo de envío y recepción de paquetes en una transferencia *download* mediante Stop and Wait

4.3. Finalización

La finalización de la transferencia se realiza mediante el envío de un paquete especial que contiene el bit de fin de archivo (end flag) activado. Para garantizar la entrega confiable de este paquete crítico, se implementa un mecanismo de retransmisión que envía el paquete de finalización hasta 10 veces consecutivas. El proceso de finalización se completa exitosamente cuando se recibe la confirmación (ACK) correspondiente del receptor, o alternativamente, después de haber realizado los 10 intentos de envío, asumiendo que la transferencia ha sido completada.

La Figura 4 muestra el cierre de una transferencia *upload* utilizando el protocolo Go-Back-N. En esta etapa, el cliente (`10.0.0.2`) envía el último paquete de datos (`ID=5247`) seguido del paquete de control con el flag `END=1` (`ID=5248`), que indica al servidor el final del archivo. El servidor (`10.0.0.1`) responde con múltiples acuses de recibo (`ACK 5249`) como mecanismo de confirmación redundante, asegurando la correcta recepción del último segmento antes de cerrar la sesión.

Este comportamiento, visible en la repetición de ACKs en la captura, permite garantizar la confiabilidad en la etapa de finalización, evitando pérdidas de sincronización entre cliente y servidor.

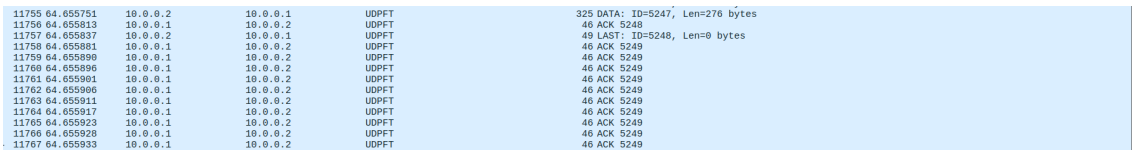

Figura 4: Secuencia de cierre de la transmisión en *upload* mediante Go-Back-N.

4.4. Interfaz de terminal

4.4.1. Menús y opciones

Cliente:

```
# Modo interactivo
python3 client.py
Enter command: upload -s archivo.png -n nombre.png -r SW -H 127.0.0.1 -p 5000 -v

# Modo directo
python3 client.py upload -s archivo.png -n nombre.png -r GBN -H 127.0.0.1 -p 5005
```

Servidor:

```
python3 server.py start-server -H 127.0.0.1 -p 5000 -s storage -v
```

Opciones disponibles:

- `-s, -src`: Archivo fuente (upload)
- `-n, -name`: Nombre del archivo
- `-r, -protocol`: Protocolo (SW/GBN)
- `-H, -host`: Dirección del servidor
- `-p, -port`: Puerto del servidor
- `-v, -verbose`: Modo verbose
- `-q, -quiet`: Modo silencioso

4.5. Manejo de errores y retransmisiones

Estrategias implementadas:

1. **Timeouts adaptativos**: 0.05 segundos para ACK.
2. **Reintentos limitados**: Máximo 70 reintentos por paquete.
3. **Timeout global**: 5 minutos máximo por transferencia completa.
4. **Recuperación de errores**:
   - Stop-and-Wait: Reenvío individual
   - Go-Back-N: Reenvío de ventana completa
5. **Manejo de paquetes perdidos**: Por timeout o desigualdad de tamaño.

4.6. Estructura de los paquetes

Formato del paquete de datos:

```
[flag_end:1bit][data_len:2bytes][pkg_id:4bytes][data:variable]
```

Campos:

- **flag_end (1 bit)**: Indica si es el último paquete (1) o datos normales (0)
- **data_len (2 bytes)**: Tamaño de los datos en bytes
- **pkg_id (4 bytes)**: Número de secuencia del paquete

- **data (variable)**: Datos del archivo (máximo 1000 bytes por paquete)

Formato del ACK:

```
[ack_number:4bytes]
```

Constantes del sistema:

- **Tamaño de paquete**: 1000 bytes de datos
- **Timeout ACK**: 0.05 segundos
- **Ventana GBN**: 10 paquetes
- **Ventana SW**: 1 paquete
- **Puerto servidor**: 5005 (configurable)
- **Puerto cliente**: 5006 (configurable)

Esta implementación proporciona un sistema robusto de transferencia de archivos UDP con dos protocolos de confiabilidad diferentes, permitiendo comparar su rendimiento en diferentes condiciones de red.

5. Pruebas y Validación

5.1. Tipos de pruebas

Se realizaron tres niveles de pruebas: **unitarias**, **de integración** y **de red**.

Las pruebas unitarias verificaron el correcto funcionamiento de los módulos principales del sistema (envío, recepción, control de secuencia, ACKs y retransmisiones). Las pruebas de integración comprobaron la comunicación entre los módulos del cliente y servidor, garantizando la coherencia de los mensajes intercambiados. Finalmente, las pruebas de red validaron el comportamiento del sistema completo bajo diferentes condiciones de conectividad simuladas con Mininet, midiendo rendimiento, confiabilidad y estabilidad del protocolo implementado.

5.2. Escenarios de prueba

Se diseñaron distintos escenarios para evaluar el desempeño de los protocolos implementados:

- **Sin pérdida:** entorno ideal, sin pérdida ni retardo, para evaluar el rendimiento máximo posible de cada protocolo.
- **Con pérdida:** se introdujo una pérdida del 10 % de paquetes en los enlaces mediante, observando la capacidad de recuperación de cada protocolo.

Las mediciones no se basaron en repeticiones múltiples, sino en el análisis de las capturas de red (`.pcap`) obtenidas en cada host de Mininet durante una ejecución controlada de la red. Dichas capturas fueron procesadas con `PyShark`, extrayendo los tiempos de inicio y finalización de la transferencia, el total de bytes enviados y las retransmisiones registradas, a partir de las cuales se calcularon los valores de duración y throughput efectivos.

5.3. Mediciones de rendimiento

5.3.1. Metodología

Las mediciones se realizaron sobre las capturas de red (`.pcap`) generadas automáticamente por `tcpdump` en cada host de Mininet.

Para cada prueba se registraron:

- **Tiempo total de transferencia:** medido como la diferencia entre el primer y último paquete UDP del archivo capturado.
- **Throughput efectivo:** calculado como la cantidad total de bytes transferidos dividida por la duración de la transferencia.

El análisis automatizado se realizó con `PyShark`, procesando las capturas almacenadas en la carpeta `wireshark_files/`, y generando un archivo consolidado de métricas (`metricas_pcap.csv`) y gráficos comparativos (`duration_comparison.png` y `throughput_comparison.png`).

Las pruebas se ejecutaron bajo el siguiente esquema de red:

- **h1:** servidor.
- **h2:** cliente upload Stop & Wait.
- **h3:** cliente upload Go Back N.
- **h4:** cliente download Stop & Wait.
- **h5:** cliente download Go Back N.

5.3.2. Resultados sin pérdidas de paquetes

En esta etapa se midió el tiempo total de transferencia y el throughput efectivo (KiB/s) de los dos protocolos implementados: Stop & Wait (SW) y Go-Back-N (GBN), tanto para las operaciones de *upload* como de *download*sin pérdida de paquetes en el envío de un archivo de 5 MB.

Protocolo	**Operación**	**Duración (s)**	**Throughput (KiB/s)**
Stop & Wait	Upload	1.658	3089.04
Go Back N	Upload	1.341	3837.518
Stop & Wait	Download	1.645	3123.206
Go Back N	Download	1.274	4018.951

Cuadro 1: Resultados promedio de duración y throughput por protocolo.

A continuación se presentan los gráficos comparativos generados automáticamente:

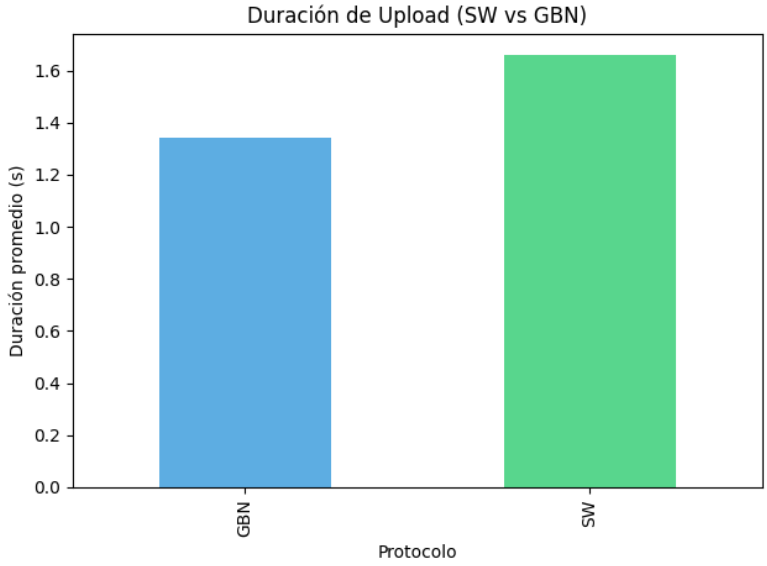

Figura 5: Comparación de duracion en Upload entre Stop & Wait y Go-Back-N.

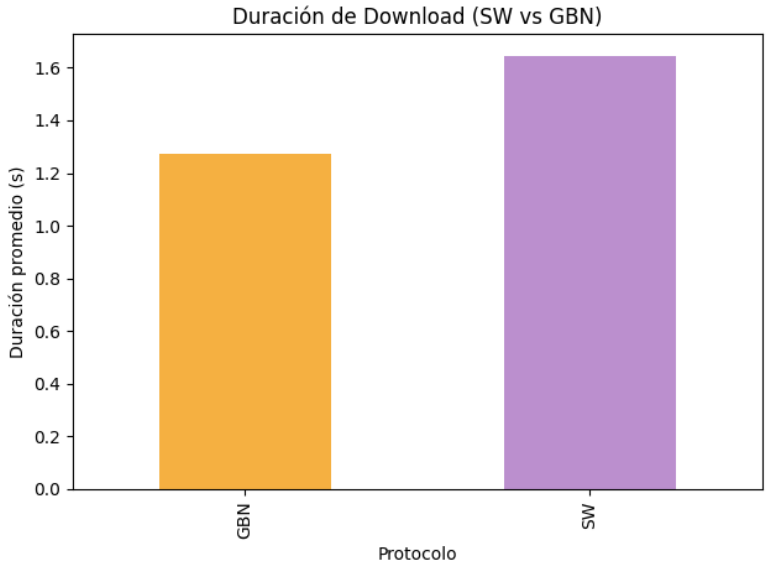

Figura 6: Comparación de duracion en Download entre Stop & Wait y Go-Back-N.

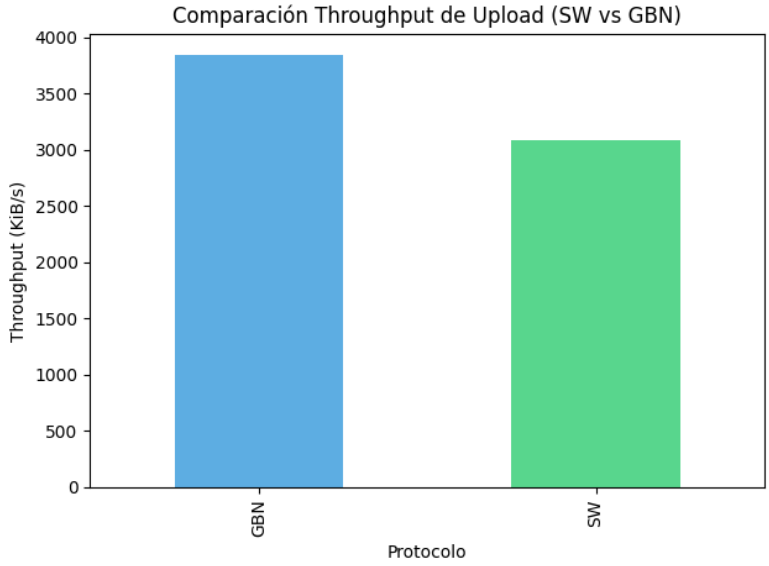

Figura 7: Comparación de throughput en Upload entre Stop & Wait y Go-Back-N.

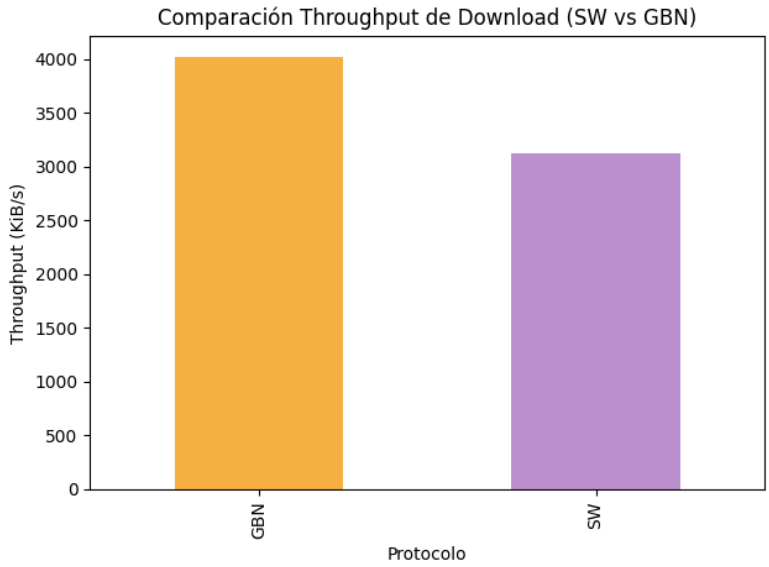

Figura 8: Comparación de throughput en Download entre Stop & Wait y Go-Back-N.

5.3.3. Resultados con 10 % de pérdida de paquetes

En esta etapa se midió el tiempo total de transferencia y el throughput efectivo (KiB/s) de los dos protocolos implementados: Stop & Wait (SW) y Go-Back-N (GBN), tanto para las operaciones de *upload* como de *download* con un 10 % pérdida de paquetes en el envío de un archivo de 5 MB.

Los resultados obtenidos muestran diferencias notables entre ambos protocolos. En las Figuras 9 y 10 se observa que GBN alcanza un throughput entre tres y cuatro veces superior a SW. De forma inversa, las Figuras 11 y 12 confirman que el tiempo total de transferencia con SW es aproximadamente el doble que con GBN. Los resultados obtenidos se resumen en la Tabla 2, donde se muestra la duración total, bytes transferidos y throughput promedio por protocolo.

Protocolo	Operación	Duración (s)	Throughput (KiB/s)
Stop & Wait	Upload	63.325	80.357
Go Back N	Upload	30.673	166.991
Stop & Wait	Download	60.781	78.621
Go Back N	Download	30.167	164.319

Cuadro 2: Resultados de duración y throughput por protocolo con pérdida de paquetes.

A continuación se presentan los gráficos comparativos generados automáticamente:

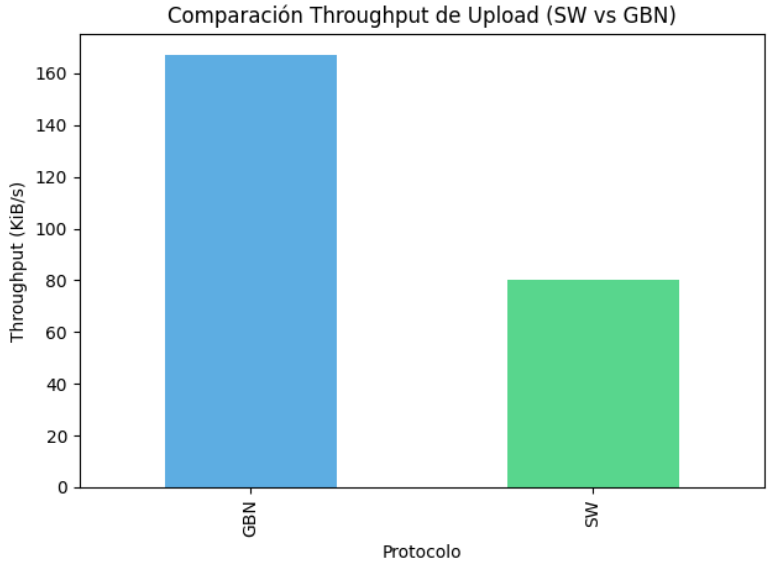

Figura 9: Comparación de throughput en Upload entre Stop & Wait y Go-Back-N.

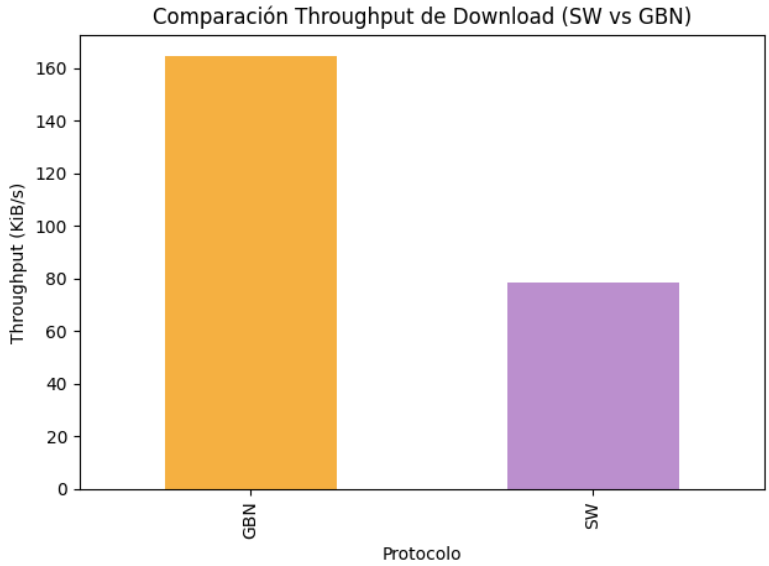

Figura 10: Comparación de throughput en Download entre Stop & Wait y Go-Back-N.

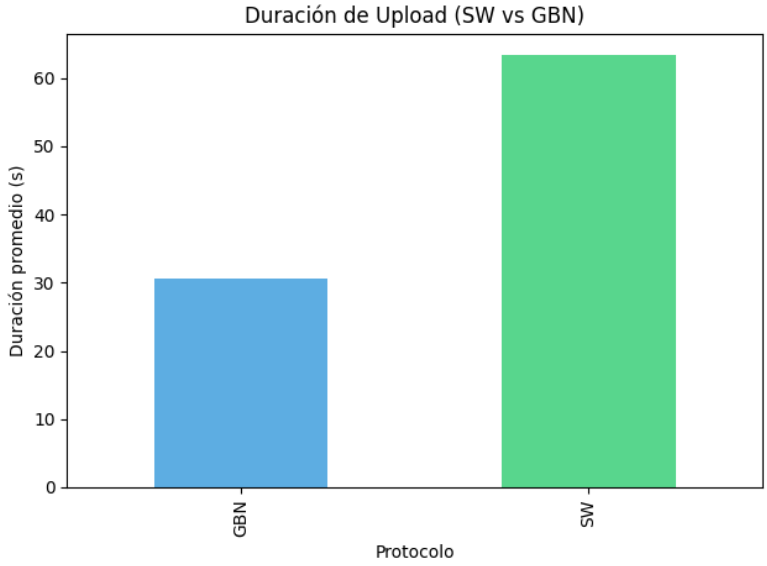

Figura 11: Duración promedio en Upload entre Stop & Wait y Go-Back-N.

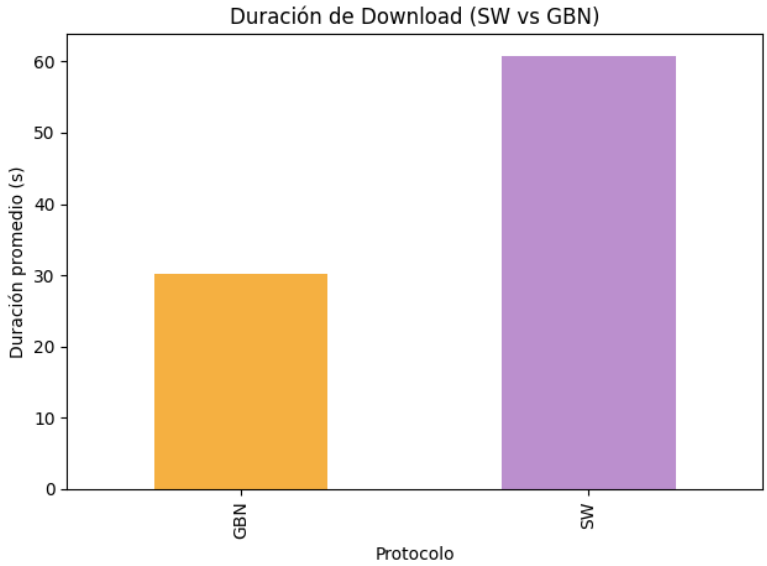

Figura 12: Duración promedio en Download entre Stop & Wait y Go-Back-N.

El protocolo Go-Back-N presentó una mejora de aproximadamente *250 %* en throughput respecto a Stop-and-Wait bajo condiciones de pérdida del 10 %, tanto en operaciones de *upload* como de *download*. Asimismo, redujo el tiempo total de transferencia en cerca de un *50 %* en comparación con Stop-and-Wait. En cambio, Stop-and-Wait mostró un aprovechamiento del canal significativamente menor debido a su ventana efectiva de un solo paquete y a los tiempos de espera por cada ACK, lo que lo hace menos eficiente en escenarios con pérdida.

5.3.4. Escenario con alta pérdida de paquetes (20 %)

Con el objetivo de analizar el comportamiento de los protocolos en condiciones más exigentes, se realizaron pruebas adicionales con una pérdida simulada del **20 %** de los paquetes. Este escenario permite observar cómo cada protocolo responde frente a un entorno con retransmisiones frecuentes y tiempos de espera prolongados.

En estas condiciones, el impacto de la pérdida es especialmente notorio en Stop & Wait, donde cada paquete perdido implica un nuevo ciclo de espera por timeout y retransmisión, aumentando considerablemente la duración total de la transferencia. Por el contrario, Go-Back-N mantiene una mayor eficiencia al aprovechar su ventana deslizante, lo que le permite continuar el envío mientras gestiona los paquetes pendientes de confirmación.

Protocolo	**Operación**	**Duración (s)**	**Throughput (KiB/s)**
Stop & Wait	Upload	146.538	34.94
Go Back N	Upload	69.707	73.45
Stop & Wait	Download	153.241	33.411
Go Back N	Download	67.539	75.807

Cuadro 3: Resultados promedio de duración y throughput por protocolo con 20 % de pérdida de paquetes.

Los resultados muestran que Go-Back-N conserva un throughput aproximadamente **2,2 veces superior** al de Stop & Wait en ambos sentidos de transferencia, y reduce el tiempo total en alrededor de un **55 %**. Esto evidencia que, aun con una pérdida considerable, Go-Back-N mantiene un rendimiento más estable y una utilización más eficiente del canal, mientras que Stop & Wait ve degradado su desempeño debido a su naturaleza secuencial y dependiente del reconocimiento individual de cada paquete.

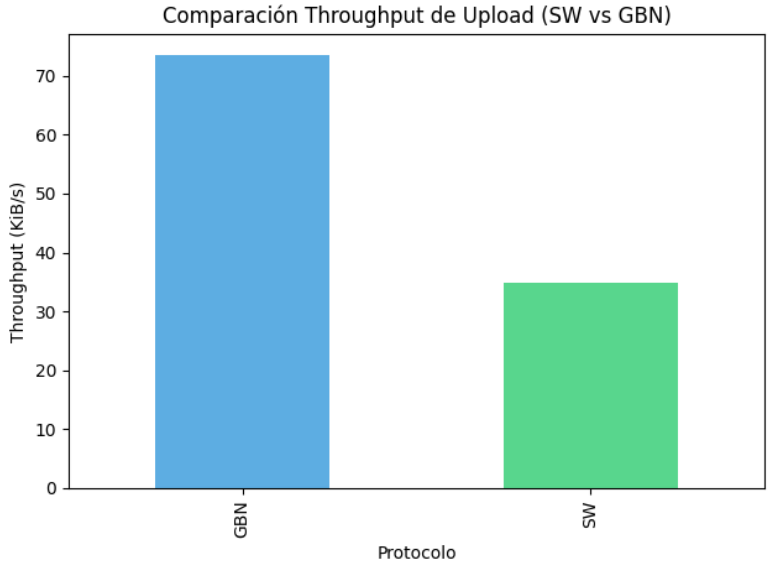

Figura 13: Comparación de throughput en Upload entre Stop & Wait y Go-Back-N.

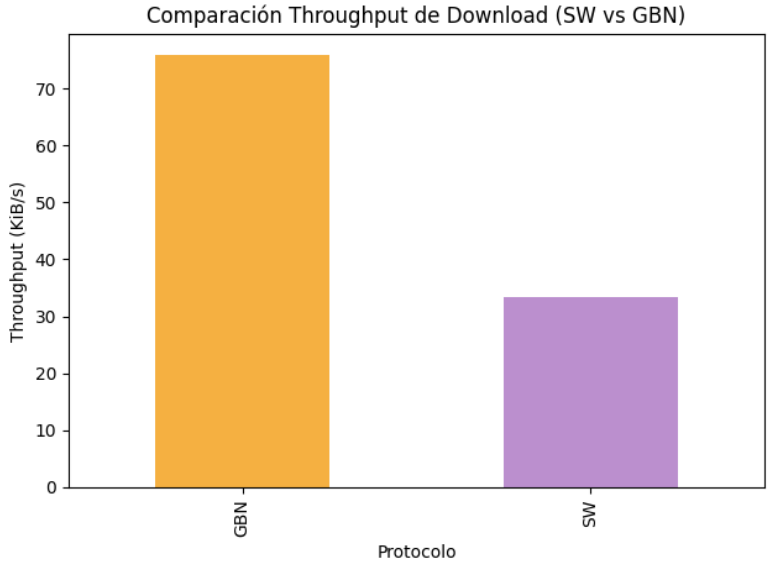

Figura 14: Comparación de throughput en Download entre Stop & Wait y Go-Back-N.

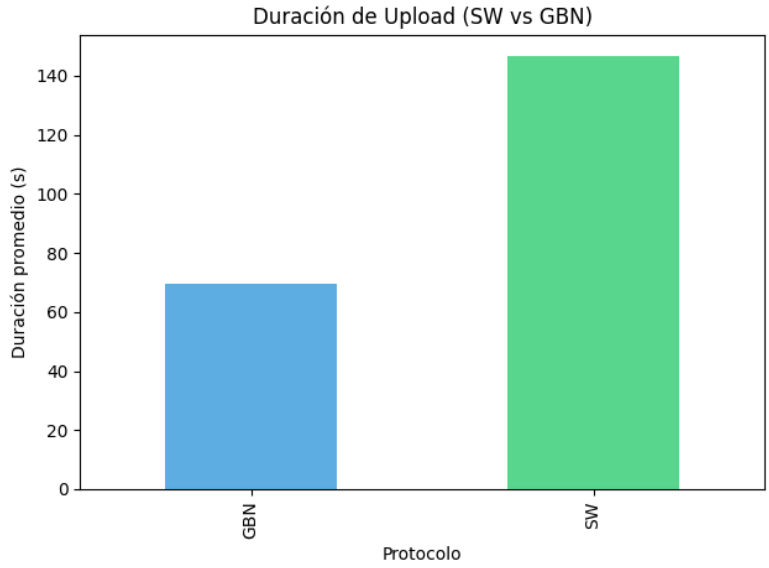

Figura 15: Duración promedio en Upload entre Stop & Wait y Go-Back-N.

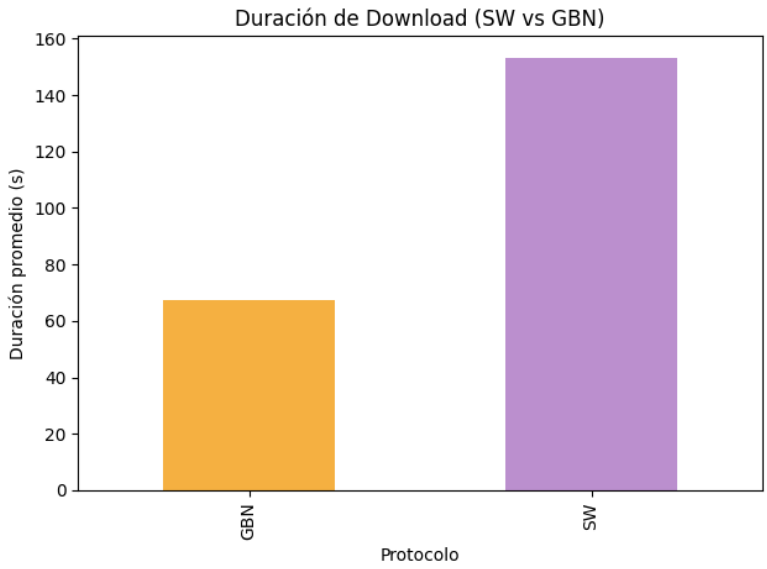

Figura 16: Duración promedio en Download entre Stop & Wait y Go-Back-N.

5.3.5. Evaluación del cumplimiento de requisitos

El requisito funcional especificaba que el sistema debía permitir la **transferencia de archivos de 5 MB en menos de 2 minutos** bajo condiciones normales de red.
Los resultados obtenidos muestran que:

- Con Go-Back-N: la transferencia de 5 MB, tanto para la carga como para la descarga, se completó en aproximadamente **30.67 segundos**.
- Con Stop-and-Wait: la transferencia tomó aproximadamente **60 segundos**.

En escenarios con pérdida, el tiempo de transferencia aumentó proporcionalmente al porcentaje de pérdida, aunque Go-Back-N logró mantener la estabilidad del throughput gracias a su ventana deslizante. En ambos casos, a pesar de la notable diferencia de tiempos, se cumple el requisito pedido. Se completa la transferencia en tiempo menor a 2 minutos. Ambos protocolos logran recuperarse de las condiciones de la red, asegurando que los paquetes lleguen completos, en orden y sin errores.

6. Preguntas

6.1. Arquitectura Cliente-Servidor

La arquitectura Cliente-Servidor es un modelo de diseño de aplicaciones distribuidas donde las funciones se dividen entre dos tipos de entidades:
Servidor:

- Es un host que está siempre activo y disponible
- Tiene una dirección IP fija y bien conocida
- Espera pasivamente las solicitudes de conexión
- Provee servicios o recursos a los clientes
- Puede atender múltiples clientes simultáneamente

Cliente:

- Inicia la comunicación con el servidor
- Puede conectarse y desconectarse de forma intermitente
- No necesita tener una dirección IP fija
- Generalmente no se comunica directamente con otros clientes
- Depende del servidor para obtener servicios o recursos

Funcionamiento: El cliente envía una solicitud (*request*) al servidor, quien la procesa y devuelve una respuesta (*response*). Esta comunicación sigue un patrón de petición-respuesta donde el servidor permanece a la escucha en un puerto específico y el cliente conoce la dirección del servidor para iniciar la conexión.

6.2. Función de un Protocolo de Capa de Aplicación

Un protocolo de capa de aplicación define las reglas y el formato de comunicación entre procesos de aplicación que se ejecutan en diferentes hosts. Sus funciones principales son:

Especifica:

- **Tipos de mensajes intercambiados:** por ejemplo, mensajes de solicitud (*request*) y respuesta (*response*)
- **Sintaxis de los mensajes:** estructura de los campos y cómo se delimitan
- **Semántica de los campos:** significado de la información en cada campo
- **Reglas de comunicación:** cuándo y cómo un proceso envía y responde mensajes

Características:

- Define el formato exacto de los datos que se transmiten
- Establece la secuencia de mensajes entre cliente y servidor
- Especifica las acciones a tomar ante diferentes eventos o errores
- Puede ser de dominio público (como HTTP, SMTP, FTP) o propietario

Los protocolos de aplicación operan sobre los servicios que proporciona la capa de transporte (TCP o UDP) y son específicos para cada tipo de aplicación (web, correo electrónico, transferencia de archivos, etc.).

6.3. Protocolo de Aplicación Desarrollado

El protocolo de aplicación desarrollado en este trabajo fue descrito previamente en detalle en la Sección 4, donde se especificaron los tipos de mensajes, el formato de los campos, la sintaxis y semántica de la comunicación entre cliente y servidor, así como las reglas de intercambio de datos para el funcionamiento de la aplicación implementada.

6.4. Protocolos TCP y UDP de la Capa de Transporte

6.4.1. TCP (Transmission Control Protocol)

Servicios que provee:

- **Conexión orientada:** establece una conexión antes de transferir datos (*handshake* de tres vías)
- **Transferencia confiable:** garantiza que los datos lleguen sin errores y en orden
- **Control de flujo:** evita que el emisor sobrecargue al receptor
- **Control de congestión:** regula la velocidad de transmisión según el estado de la red

Características:

- Orientado a la conexión
- Entrega garantizada y ordenada de datos
- Detección y retransmisión de paquetes perdidos
- Mayor *overhead* por los mecanismos de control
- Más lento que UDP debido a las verificaciones

Cuándo utilizarlo:

- Transferencia de archivos (FTP, HTTP)
- Correo electrónico (SMTP)
- Navegación web
- Cualquier aplicación donde la integridad de los datos sea crítica

6.4.2. UDP (User Datagram Protocol)

Servicios que provee:

- **Transferencia no confiable:** envío de datagramas sin garantía de entrega
- **Sin conexión:** no establece una conexión previa
- **Multiplexación:** mediante números de puerto

Características:

- No orientado a la conexión
- No garantiza entrega ni orden de los paquetes
- Sin control de flujo ni congestión
- Menor *overhead*, más rápido y eficiente
- Los paquetes pueden perderse, duplicarse o llegar desordenados

Cuándo utilizarlo:

- *Streaming* de audio y video (donde la velocidad es más importante que la perfección)
- Videojuegos en línea (baja latencia es crítica)
- DNS (consultas rápidas y simples)
- VoIP (tolerante a pequeñas pérdidas)
- Aplicaciones en tiempo real donde la latencia es más crítica que la confiabilidad

6.4.3. Comparación

Aspecto	**TCP**	**UDP**
Confiabilidad	Alta (garantizada)	Baja (no garantizada)
Velocidad	Más lento	Más rápido
Conexión	Orientado a conexión	Sin conexión
Orden	Garantizado	No garantizado
Overhead	Mayor	Menor
Uso típico	Datos críticos	Tiempo real

Cuadro 4: Comparación entre TCP y UDP