



UBA
1821 Universidad
de Buenos Aires

.UBAfiuba 
FACULTAD DE INGENIERÍA

TB067

REDES DE COMUNICACIONES

Nivel de Aplicaciones y de Transporte del Modelo TCP/IP

Autores:

Falcon Luciana B.
Juncal Franco

Padrón:

107316
106448

Fecha:

12/05/2025

Índice

1. Introducción	2
2. Resumen	3
3. Desarrollo	4
3.1. Análisis de Puertos No Disponibles para Servidores	4
3.2. Manejo de Conexiones Concurrentes en un Servidor	5
3.3. Asignación de puertos para múltiples sesiones	6
3.4. Análisis de Finalización de Sesiones Telnet	6
3.5. Puertos origen del lado del cliente	8
3.6. Funciones del Protocolo TCP	8
3.6.1. Establecimiento de sesión	8
3.6.2. Orden en los datos transmitidos	10
3.6.3. Control de flujo	11
3.6.4. Control de congestión	12
3.6.5. Calculo de RTT y RTO	13
4. Conclusiones	14
5. Bibliografía	15

1. Introducción

Se implementaron dos servidores TCP que ofrecen servicios básicos de suma y eco de mensajes. Cada servidor escuchó en un puerto diferente, permitiendo la conexión de clientes a través de la herramienta de texto Telnet.

El objetivo principal fue gestionar consultas en una y en múltiples sesiones TCP, validando los comandos recibidos y asegurando la correcta respuesta. En caso de recibir una solicitud inválida, el servidor cerrará inmediatamente la sesión, asegurando así un control sencillo pero robusto de las conexiones.

2. Resumen

Se desarrollaron dos servidores TCP independientes: uno encargado de realizar operaciones de suma (/suma [número 1] [número 2]) y otro de repetir mensajes (/eco [texto]).

Ambos servidores fueron programados para:

- Mantener las sesiones abiertas mientras las consultas sean válidas.
- Cerrar la conexión ante cualquier comando inválido.
- Escuchar y atender solicitudes simultáneamente de uno y múltiples clientes.

Se realizaron pruebas utilizando Telnet y Wireshark, analizando sus respuestas y confirmando el correcto funcionamiento de los servicios, la estabilidad de las conexiones múltiples, y el adecuado manejo de errores y cierres de sesión.

3. Desarrollo

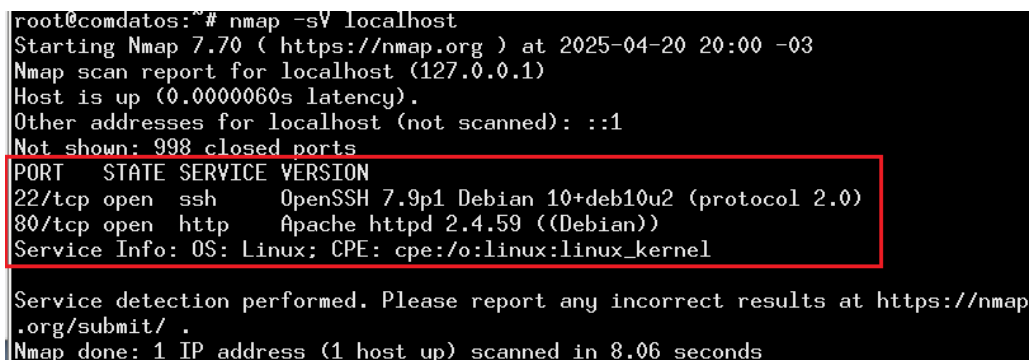
3.1. Análisis de Puertos No Disponibles para Servidores

Los puertos de escucha para los servidores se distinguen en dos categorías:

- 0 al 1023: Well-known ports o puertos del sistema. Son puertos reservados para servicios del sistema o aplicaciones ampliamente utilizadas (como HTTP en el puerto 80 o SSH en el 22).
- > 1023: Dynamics ports o puertos efímeros. Se asignan temporalmente para establecer conexiones cliente, de manera automática, y no suelen utilizarse para escuchar conexiones entrantes, osea los usan los clientes para conectarse a los servidores.

Se puede elegir cualquiera de dichos puertos de escucha para los servidores siempre y cuando no este en uso por otra aplicación o proceso que ya haya abierto ese mismo puerto en esa misma IP (por ejemplo, si en la máquina ya está corriendo un servidor web en el puerto 80).

Usando la aplicación nmap en la terminal como se muestra en la figura 1, se obtuvo un listado de puertos no disponibles en la computadora para ser usados como servidor.



```
root@comdatos:~# nmap -sV localhost
Starting Nmap 7.70 ( https://nmap.org ) at 2025-04-20 20:00 -03
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000060s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
80/tcp    open  http     Apache httpd 2.4.59 ((Debian))
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 8.06 seconds
```

Figura 1: Puertos no disponibles, mediante nmap.

Al estar en uso el puerto 80, se intento ejecutar los servidores en el mismo, pero las aplicaciones no pudieron iniciar. En la figura 2 se muestra el mensaje de error correspondiente, con el código 98 (del sistema Linux/Unix), que indica que el puerto especificado ya está en uso.

```

root@comdatos:~/Downloads# ls
servidor_eco_unica_5001.py      servidor_suma_unica_5000.py
servidores_multiples_eco_5001.py  'socket'$'\r'
servidores_multiples_suma_5000.py

root@comdatos:~/Downloads# python3 servidor_suma_unica_5000.py
Traceback (most recent call last):
  File "servidor_suma_unica_5000.py", line 55, in <module>
    main()
  File "servidor_suma_unica_5000.py", line 37, in main
    s.bind((HOST, PORT))
OSError: [Errno 98] Address already in use
root@comdatos:~/Downloads#

```

Figura 2: Terminal - Puerto no disponible.

3.2. Manejo de Conexiones Concurrentes en un Servidor

Mediante código en Python, se limitó a los servidores a aceptar solamente una conexión a la vez para analizar que sucede cuando habiendo una conexión establecida arriba un nuevo intento de conexión.

En la figura 3 se muestran los paquetes capturados en Wireshark, donde en la acción número 11, remarcada en rojo, al intentar establecer una segunda conexión, se activa el flag RST del lado del servidor, rechazando la conexión.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	76	45776 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=156801596 TSecr=0 WS=128
2	0.000012727	127.0.0.1	127.0.0.1	TCP	76	5000 → 45776 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=156801596 TSecr=0
3	0.000021521	127.0.0.1	127.0.0.1	TCP	68	45776 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=156801596 TSecr=156801596
4	0.000421001	127.0.0.1	127.0.0.1	IP	113	unknown 0x72
5	0.000441751	127.0.0.1	127.0.0.1	TCP	68	45776 → 5000 [ACK] Seq=1 Ack=46 Win=65536 Len=0 TSval=156801596 TSecr=156801596
6	23.671039595	127.0.0.1	127.0.0.1	IP	80	unknown 0x75
7	23.671095293	127.0.0.1	127.0.0.1	TCP	68	5000 → 45776 [ACK] Seq=46 Ack=13 Win=65536 Len=0 TSval=156825255 TSecr=156825255
8	23.671298616	127.0.0.1	127.0.0.1	IP	81	unknown 0x2b
9	23.671304159	127.0.0.1	127.0.0.1	TCP	68	45776 → 5000 [ACK] Seq=13 Ack=59 Win=65536 Len=0 TSval=156825255 TSecr=156825255
10	41.748033937	127.0.0.1	127.0.0.1	TCP	76	45778 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=156843315 TSecr=0 WS=128
11	41.748044322	127.0.0.1	127.0.0.1	TCP	56	5000 → 45778 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figura 3: Conexiones a servidor de una única sesión activa.

Como ya existe una sesión activa ocupando el puerto de escucha, al recibir un nuevo intento de conexión, el servidor responde enviando un paquete con el flag RST (Reset) activado, rechazando así la nueva solicitud de conexión.

Este comportamiento es coherente con un servidor diseñado para manejar únicamente una conexión a la vez, cerrando o reseteando cualquier intento adicional hasta que la sesión activa finalice.

3.3. Asignación de puertos para múltiples sesiones

Nuevamente, modificando el código del servidor, se permitió que hayan dos conexiones activas simultáneamente, esto es para poder observar como cambian los puertos del lado de los clientes. Se ejecutaron 3 sesiones en total, 2 conexiones con un "cliente 1" y otra aparte con otro "cliente 2", esto se muestra en la figura 4.

Figura 4: Terminales de clientes y servidor en múltiples conexiones.

- En **rojo** esta la primer conexión del "cliente 1". Puerto: 52478
- En **verde** esta la conexión del "cliente 2". Puerto: 52480
- En **azul** esta la reconexion del "cliente 1". Puerto: 52482

Para confirmar lo visto en la terminal, en la figura 5 se muestra la captura de Wireshark de las sesiones.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	52478 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1793849996 TSecr=0 WS=128
2	0.000000387	127.0.0.1	127.0.0.1	TCP	74	5000 → 52478 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1793849996 TSecr=1793849996 WS=128
3	0.000010283	127.0.0.1	127.0.0.1	TCP	66	52478 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1793849996 TSecr=1793849996
4	0.000355460	127.0.0.1	127.0.0.1	IP	111	unknown 0x72
5	0.000365291	127.0.0.1	127.0.0.1	TCP	66	52478 → 5000 [ACK] Seq=1 Ack=46 Win=65536 Len=0 TSval=1793849996 TSecr=1793849996
6	12.083234389	127.0.0.1	127.0.0.1	TCP	74	52480 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1793862079 TSecr=0 WS=128
7	12.083339556	127.0.0.1	127.0.0.1	TCP	74	5000 → 52480 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1793862079 TSecr=1793862079 WS=128
8	12.083349980	127.0.0.1	127.0.0.1	TCP	66	52480 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1793862079 TSecr=1793862079
9	12.083562241	127.0.0.1	127.0.0.1	IP	111	unknown 0x72
10	12.083571433	127.0.0.1	127.0.0.1	TCP	66	52480 → 5000 [ACK] Seq=1 Ack=46 Win=65536 Len=0 TSval=1793862079 TSecr=1793862079
11	13.429870899	127.0.0.1	127.0.0.1	TCP	66	52482 → 5000 [FIN, ACK] Seq=1 Ack=46 Win=65536 Len=0 TSval=1793884426 TSecr=1793884426
12	13.429871710	127.0.0.1	127.0.0.1	TCP	66	5000 → 52478 [FIN, ACK] Seq=46 Ack=2 Win=65536 Len=0 TSval=1793884426 TSecr=1793884426
13	13.429879337	127.0.0.1	127.0.0.1	TCP	66	52478 → 5000 [ACK] Seq=2 Ack=47 Win=65536 Len=0 TSval=1793884426 TSecr=1793884426
14	17.585028496	127.0.0.1	127.0.0.1	TCP	74	52482 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1793887581 TSecr=0 WS=128
15	17.585038361	127.0.0.1	127.0.0.1	TCP	74	5000 → 52482 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1793887581 TSecr=1793887581 WS=128
16	17.585045402	127.0.0.1	127.0.0.1	TCP	66	52482 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1793887581 TSecr=1793887581
17	17.585366668	127.0.0.1	127.0.0.1	IP	111	unknown 0x72
18	17.585333467	127.0.0.1	127.0.0.1	TCP	66	52482 → 5000 [ACK] Seq=1 Ack=46 Win=65536 Len=0 TSval=1793887581 TSecr=1793887581

Figura 5: Múltiples conexiones capturadas en Wireshark.

De nuevo, en **rojo**, **verde** y **azul** están resaltadas las interacciones de cada sesión con el servidor. Como se puede ver, aunque el cliente sea el mismo (cliente 1) si estuvo en dos sesiones distintas, estas dos sesiones tendrán puertos distintos.

3.4. Análisis de Finalización de Sesiones Telnet

Las sesiones pueden concluir tanto por forzar un comando inválido como cerrando la sesión vía telnet. Se procedió a realizar ambos casos como se muestran en las figuras

6 y 7.

```

root@comdatos:~# telnet 127.0.0.1 5000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^'.
Servidor de suma listo. Usa /suma num1 num2
^]
telnet> quit
Connection closed.
root@comdatos:~#
[suma] escuchando en puerto 5000
--> Conexión aceptada de ('127.0.0.1', 52448)
Conexión desde ('127.0.0.1', 52448)
Cerrada conexión con ('127.0.0.1', 52448)
[suma] escuchando en puerto 5000

```

Figura 6: Terminales cuando el cliente finaliza la conexión.

```

root@comdatos:~# telnet 127.0.0.1 5000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^'.
Servidor de suma listo. Usa /suma num1 num2
este es un comando invalido
Comando invalido. Cerrando conexión.
Connection closed by foreign host.
root@comdatos:~#
[suma] escuchando en puerto 5000
--> Conexión aceptada de ('127.0.0.1', 52450)
Conexión desde ('127.0.0.1', 52450)
> ('127.0.0.1', 52450) -> 'este es un comando invalido'
Cerrada conexión con ('127.0.0.1', 52450)
[suma] escuchando en puerto 5000

```

Figura 7: Terminales cuando el servidor finaliza la conexión.

Se capturaron los paquetes en Wireshark y se observó que, cuando la finalización de la conexión es iniciada por el cliente, figura 8, este envía un paquete al servidor con el flag FIN activo. El servidor responde con su propio paquete FIN, y finalmente el cliente cierra el proceso enviando un paquete ACK.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	52446 -> 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1788967338 TSecr=0 WS=128
2	0.000003937	127.0.0.1	127.0.0.1	TCP	74	5000 -> 52446 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1788967338 TSecr=1788967338 WS=128
3	0.00016559	127.0.0.1	127.0.0.1	TCP	66	52446 -> 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1788967338 TSecr=1788967338
4	0.000165725	127.0.0.1	127.0.0.1	IP	111	unknown 0x72
5	0.00017367	127.0.0.1	127.0.0.1	TCP	66	52446 -> 5000 [ACK] Seq=1 Ack=46 Win=65536 Len=0 TSval=1788967338 TSecr=1788967338
6	0.00038783	127.0.0.1	127.0.0.1	TCP	66	52446 -> 5000 [FIN, ACK] Seq=1 Ack=46 Win=65536 Len=0 TSval=1788976895 TSecr=1788967338
7	0.00055660	127.0.0.1	127.0.0.1	TCP	66	5000 -> 52446 [FIN, ACK] Seq=46 Ack=2 Win=65536 Len=0 TSval=1788976895 TSecr=1788976895
8	0.000564306	127.0.0.1	127.0.0.1	TCP	66	52446 -> 5000 [ACK] Seq=1 Ack=47 Win=65536 Len=0 TSval=1788976895 TSecr=1788976895

Figura 8: Fin de conexión dada por el cliente, capturado en Wireshark.

En cambio, cuando la conexión es finalizada por el servidor, figura 9, es éste quien envía primero el paquete con el flag FIN activado. Posteriormente, el cliente responde enviando también un paquete con el flag FIN, y el servidor finaliza la comunicación completando el intercambio con un paquete ACK.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	52446 -> 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1790592519 TSecr=0 WS=128
2	0.000012016	127.0.0.1	127.0.0.1	TCP	74	5000 -> 52450 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1790592519 TSecr=1790592519 WS=128
3	0.000024361	127.0.0.1	127.0.0.1	TCP	66	52450 -> 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1790592519 TSecr=1790592519
4	0.000222794	127.0.0.1	127.0.0.1	IP	111	unknown 0x72
5	0.000237310	127.0.0.1	127.0.0.1	IP	66	52450 -> 5000 [ACK] Seq=1 Ack=46 Win=65536 Len=0 TSval=1790592519 TSecr=1790592519
6	0.00038783	127.0.0.1	127.0.0.1	IP	95	unknown 0x74
7	0.000390530	127.0.0.1	127.0.0.1	TCP	66	5000 -> 52450 [ACK] Seq=46 Ack=30 Win=65536 Len=0 TSval=1790602517 TSecr=1790602517
8	0.00072163	127.0.0.1	127.0.0.1	IP	104	unknown 0x6d
9	0.00077385	127.0.0.1	127.0.0.1	TCP	66	52450 -> 5000 [ACK] Seq=30 Ack=44 Win=65536 Len=0 TSval=1790602517 TSecr=1790602517
10	0.00110313	127.0.0.1	127.0.0.1	TCP	66	5000 -> 52450 [FIN, ACK] Seq=46 Ack=30 Win=65536 Len=0 TSval=1790602517 TSecr=1790602517
11	0.00194127	127.0.0.1	127.0.0.1	TCP	66	52450 -> 5000 [FIN, ACK] Seq=30 Ack=85 Win=65536 Len=0 TSval=1790602517 TSecr=1790602517
12	0.00199893	127.0.0.1	127.0.0.1	TCP	66	5000 -> 52450 [ACK] Seq=35 Ack=31 Win=65536 Len=0 TSval=1790602517 TSecr=1790602517

Figura 9: Fin de conexión dada por el servidor, capturado en Wireshark.

3.5. Puertos origen del lado del cliente

Cuando un cliente Telnet (o cualquier cliente TCP) inicia una conexión hacia un servidor, el puerto origen lo elige automáticamente el sistema operativo del cliente, dicho puerto es efímero (o dinámico) automático, con lo cual su valor es > 1023 . Particularmente la Internet Assigned Numbers Authority, IANA, (y en general en sistemas Linux modernos), sugiere el rango desde 49152 hasta 65535 para los puertos efímeros.

El puerto destino es el puerto del servidor. Es fijo, programado. Por ejemplo 5000 para suma o 5001 para eco.

3.6. Funciones del Protocolo TCP

RFC 793 Std 7

El protocolo TCP, además de ofrecer la multiplexación de canales, como se observó en el punto 3.3, ofrece una amplia cantidad de ventajas, algunas de estas son:

- Establecimiento de sesión.
- Orden en los datos transmitidos.
- Control de flujo.
- Control de congestión.
- Cálculo de tiempo de retransmisión (RTT).

Para dejar más en claro la utilidad de cada una de estas implementaciones al protocolo, se procede a explicar su funcionamiento.

3.6.1. Establecimiento de sesión

El establecimiento de sesión en TCP presenta una ventaja clave frente a UDP por su confiabilidad, orden y control, aunque a costa de algo más de complejidad y latencia. Esta sesión se establece mediante:

Inicio de sesión:

El establecimiento de sesión en TCP se basa en la creación de una conexión confiable entre los dispositivos (cliente y servidor) antes de comenzar a intercambiar datos. Este proceso se conoce como el "three-way handshake" (intercambio de tres vías) y consta de tres pasos como se muestra en la figura 10 a continuación:

- Se intercambian 3 segmentos
- Se envía el MSS

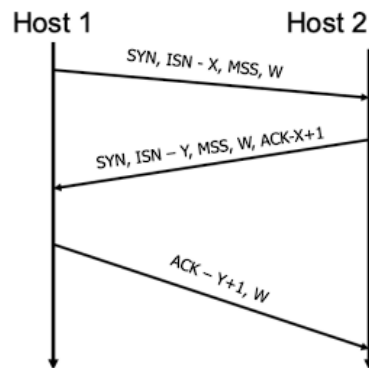


Figura 10: Establecimiento de sesión.

La secuencia de segmentos es:

- Segmento 1: SYN (synchronize).
El cliente inicia la conexión enviando un segmento TCP con el bit $SYN = 1$ y un número de secuencia inicial ($ISN-X$).
- Segmento 2: SYN-ACK (synchronize-acknowledgment).
El servidor responde con un segmento que tiene $SYN = 1$ y $ACK = 1$.
El servidor incluye su propio número de secuencia ($ISN-Y$.) y confirma el número de secuencia del cliente.
- Segmento 3: ACK (acknowledgment):
El cliente responde con un segmento con $ACK = 1$, confirmando el número de secuencia del servidor.

Una vez completado este intercambio, la sesión TCP está establecida y ambos extremos pueden comenzar a enviarse datos de manera segura y confiable.

Fin de sesión:

Para el cierre de la sesión se intercambian 4 segmentos: FIN, ACK, FIN, ACK, como se muestra en la figura 11:

- Se intercambian 4 segmentos(default)

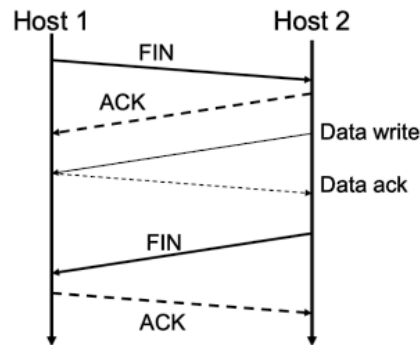


Figura 11: Fin de sesión.

La secuencia de segmentos es:

- Segmento 1: FIN (de Host 1 a Host 2).
- Segmento 2: ACK (de Host 2 a Host 1).
- Segmento 3: FIN (de Host 2 a Host 1).
- Segmento 4: ACK (de Host 1 a Host 2).

Un ejemplo de establecimiento de sesión se puede ver en la figura 5 donde se muestran múltiples conexiones capturadas en Wireshark, donde los recuadros rojos corresponden al inicio y fin.

3.6.2. Orden en los datos transmitidos

TCP garantiza que los datos lleguen en el mismo orden en que fueron enviados, incluso si los segmentos individuales se transmiten por diferentes rutas en la red y llegan desordenados.

Para lograr la llegada de los datos en orden intervienen:

1. Número de secuencia:
Cada byte de datos enviado en una conexión TCP tiene un número de secuencia, permitiendo al receptor reordenar los datos si llegan fuera de orden.
2. Buffer:
Si un segmento llega antes que otro anterior, el receptor lo guarda temporalmente hasta que pueda reconstruir la secuencia completa.

3. ACK:

El receptor confirma la recepción del último byte recibido en orden, mediante el el flag de ACK. Si faltan segmentos, el emisor puede reenviarlos.

En la Figura 12 se muestra un ejemplo del orden de los datos capturados por Wireshark, donde, luego del primer paquete enviado por el transmisor, el receptor puede determinar cuál es el siguiente número de secuencia que debe utilizar para mantener el orden de los datos. Éste se obtiene sumando el número de secuencia anterior (seq1) con la longitud (length) del mismo segmento.

No.	Time	Source	Destination	Protocol	Length	Info
2	0.022414	128.119.245.12	192.168.86.68	TCP	74	80 → 55639 [SYN, ACK] Seq=1068969752 Ack=4236649188 Win=28960 Len=0 MS...
3	0.022585	192.168.86.68	128.119.245.12	TCP	66	55639 → 80 [ACK] Seq=4236649188 Ack=1068969753 Win=131712 Len=0 TSval=3...
4	0.024047	128.119.245.12	192.168.86.68	TCP	1514	55639 → 80 [ACK] Seq=4236649188 Ack=1068969753 Win=131712 Len=1448 TSv...
5	0.024048	192.168.86.68	128.119.245.12	TCP	1514	55639 → 80 [ACK] Seq=4236650636 Ack=1068969753 Win=131712 Len=1448 TSv...
6	0.024049	192.168.86.68	128.119.245.12	TCP	1514	55639 → 80 [ACK] Seq=4236652084 Ack=1068969753 Win=131712 Len=1448 TSv...
7	0.052671	192.168.86.68	128.119.245.12	TCP	66	80 → 55639 [ACK] Seq=1068969753 Ack=4236650636 Win=131712 Len=0 TSval=3...
8	0.052676	128.119.245.12	192.168.86.68	TCP	66	80 → 55639 [ACK] Seq=1068969753 Ack=4236652084 Win=131712 Len=0 TSval=3...
9	0.052774	192.168.86.68	128.119.245.12	TCP	1514	55639 → 80 [ACK] Seq=4236653532 Ack=1068969753 Win=131712 Len=1448 TSv...
10	0.052775	192.168.86.68	128.119.245.12	TCP	1514	55639 → 80 [ACK] Seq=4236654980 Ack=1068969753 Win=131712 Len=1448 TSv...

Figura 12: Orden de los datos capturados con Wireshark.

3.6.3. Control de flujo

El control de flujo define cuánta información puede enviarse al receptor sin saturar su buffer, lo que provocaría pérdida de datos. Este mecanismo impone una cota importante a la velocidad de transferencia de datos: incluso si se dispone de un enlace con gran ancho de banda, el rendimiento podría verse limitado por el tamaño del buffer y la capacidad de procesamiento del receptor.

La cantidad máxima de datos que el receptor puede aceptar sin confirmar está especificada principalmente en dos elementos de la sesión TCP. El primero es el campo Window Size, presente en el encabezado TCP. Este campo, de 16 bits, indica una cantidad de bytes (con un valor máximo de 65.535) y es actualizado por el receptor en cada segmento que envía, informando al emisor sobre su ventana de recepción disponible.

Sin embargo, en contextos modernos, 65 kB resultan insuficientes. Por ello, el RFC 1323 introdujo la opción Window scale, que permite escalar el valor del window Size mediante un factor de multiplicación. Este factor se negocia al inicio de la conexión, dentro de las opciones TCP del segmento SYN, y permite alcanzar un window size efectivo de hasta 1 GB.

En la figura 13 se observa la ubicación de estos campos en una captura realizada con Wireshark:

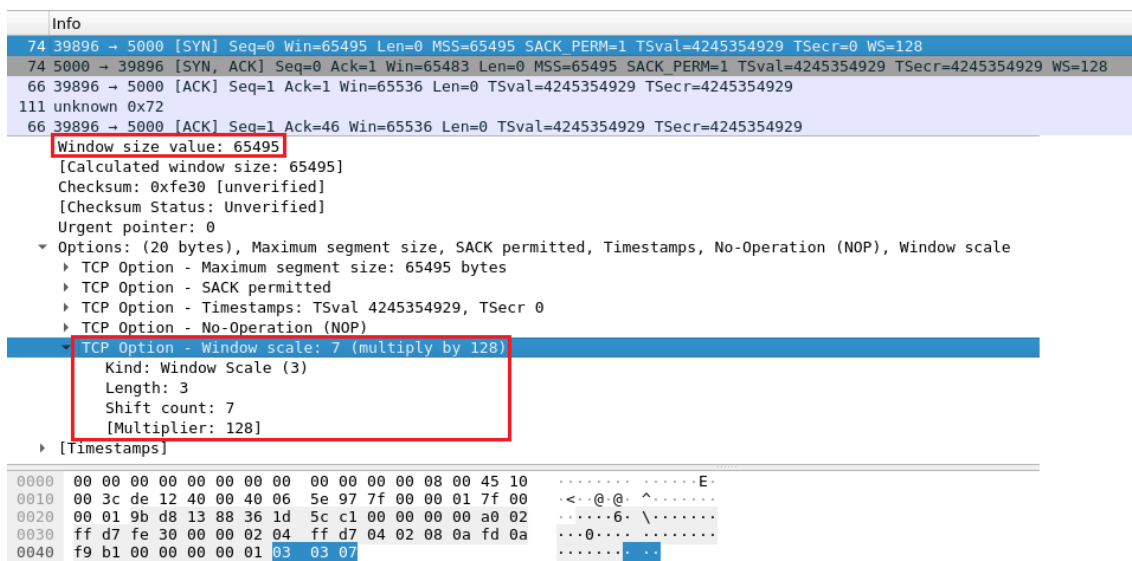


Figura 13: Campos Window Size y Window Scale en un servidor de suma, capturados con Wireshark.

3.6.4. Control de congestión

A diferencia del control de flujo que viene limitado por los extremos de la red, el control de congestión se regula en base a la congestión de la red.

Este control se maneja mediante ambos extremos de la sesión y la información que estos pueden recolectar sobre la red, lo que genera una independencia de la red a costo de complejidad de manejo.

TCP implementa una serie de algoritmos para ajustar la cwnd (ventana de congestión), estos son slow start, congestion avoidance, fast retransmit y fast recovery. La cwnd nos delimita cuantos segmentos podremos enviar antes sin recibir un ACK.

Principalmente una sesión comienza en slow start, haciendo crecer cwnd exponencialmente hasta hallar un limite determinado, este limite puede ser el ssthresh (slow start threshold) o una perdida de datos. Si se llega a ssthresh pasa a ejecutarse congestion avoidance, que genera una crecimiento lineal en la cwnd, pero si se perdieron datos pueden suceder 2 cosas, o solo fue un extravió de algunos paquetes o sucedió un timeout.

Un timeout quiere decir que se supero un RTO dado, o sea, nunca llego o tardo un tiempo considerable la respuesta del transmisor. Como respuesta a este problema, para evitar la saturación de la red, el algoritmo settea la cwnd en 1 MSS y comienza de nuevo desde slow start con una ssthresh reducido.

Por otro lado, si solo se extraviaron algunos paquetes, entran en juego el fast retransmit y fast recovery, estos se activan una vez se reciben tres ACKs repetidos. El fast retransmit retransmite los segmentos que no se pudieron confirmar y el fast recovery evita que se vuelva a slow start, disminuyendo el sstresh y actualizando el cwnd.

Todo este funcionamiento, a grandes rasgos, compone al TCP Reno (RFC 5681), a continuación se muestra la figura 14 donde se ve como después de tres ACKs duplicados, actual el TCP Fast retransmission.

3356	4.126675	0.000103	192.168.10.184	192.168.10.196	TCP	1460	210240	262144	52343 → 5201 [ACK] Seq=4555802 Ack=1 Win=262144 Len=1460
3357	4.126676	0.000001	192.168.10.184	192.168.10.196	TCP	1460	211700	262144	52343 → 5201 [ACK] Seq=4557262 Ack=1 Win=262144 Len=1460
3358	4.191494	0.064818	192.168.10.196	192.168.10.184	TCP	0		212992	[TCP Dup ACK 3355#1] 5201 → 52343 [ACK] Seq=1 Ack=4347022 Win=212992 Le
3359	4.192319	0.000825	192.168.10.196	192.168.10.184	TCP	0		212992	[TCP Dup ACK 3355#2] 5201 → 52343 [ACK] Seq=1 Ack=4347022 Win=212992 Le
3360	4.193054	0.000735	192.168.10.196	192.168.10.184	TCP	0		212992	[TCP Dup ACK 3355#3] 5201 → 52343 [ACK] Seq=1 Ack=4347022 Win=212992 Le
3361	4.193149	0.000095	192.168.10.184	192.168.10.196	TCP	1460	211700	262144	[TCP Fast Retransmission] 52343 → 5201 [ACK] Seq=4347022 Ack=1 Win=2621

Figura 14: Fast retransmit y ACKs duplicados, capturados con Wireshark.

3.6.5. Calculo de RTT y RTO

Debido a que TCP nos ofrece un control en la entrega de cada paquete, es posible calcular cuanto fue el tiempo de retransmisión (RTT) que es muy útil para comprender la congestión que sufre la red.

Mediante el conocimiento contaste del RTT, se puede derivar un tiempo de espera de retransmisión (RTO), el RTO es un referencia crucial en la modulación de la velocidad de transmisión, y es necesario definir como se estima.

El RTO debe de ser una función dependiente de RTT, ya que este varia según el estado de las redes, las cuales son muy inestables, dados determinados casos. Entonces una muy buena manera de determinar un factor inestable es usar probabilidad, mas específicamente el teorema de Chebyshev, que nos otorga un valor ajustado de RTO en base a un estimado y la desviación de RTT. Generalmente en los protocolo TCP estudiados se usa una ecuación que tiene una forma:

$$RTO = RTT_{estimado} + 4RTT_{desviacion} \quad (1)$$

Esto ofrece un timeout con tasa de error $< 6\%$, que según la aplicación, se puede variar el valor k , en este caso se ocupo un $k = 4$.

En la figura 15 se encuentra el caso mas simple para poder ver un RTT. Corresponde al primer ACK de cualquier sesión TCP, en el three way handshake, donde el tiempo de arribo de este primer ACK concuerda con el tiempo de sesión activa hasta ese momento.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	39254 → 5000 [SYN] Seq=0 Win=65495 Len=0
2	0.000013877	127.0.0.1	127.0.0.1	TCP	74	5000 → 39254 [SYN, ACK] Seq=0 Ack=1 Win=6
3	0.000023650	127.0.0.1	127.0.0.1	TCP	66	39254 → 5000 [ACK] Seq=1 Ack=1 Win=65536

Acknowledgment number: 1 (relative ack number) 1010 = Header Length: 40 bytes (10) ▶ Flags: 0x012 (SYN, ACK) Window size value: 65483 [Calculated window size: 65483] Checksum: 0xfe30 [unverified] [Checksum Status: Unverified] Urgent pointer: 0 ▶ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale ▶ [SEQ/ACK analysis] [This is an ACK to the segment in frame: 1] [The RTT to ACK the segment was: 0.000013877 seconds] [iRTT: 0.000023650 seconds] ▶ [Timestamps] [Time since first frame in this TCP stream: 0.000013877 seconds] [Time since previous frame in this TCP stream: 0.000013877 seconds]	
---	--

0000	00 00 00 00 00 00 00 00	00 00 00 00 08 00 45 00E.
0010	00 3c 00 00 40 00 40 06	3c ba 7f 00 00 01 7f 00	<.@.<.....
0020	00 01 13 88 99 56 3a f9	7b 60 33 61 11 af a0 12	...V: { 3a...
0030	ff cb fe 30 00 00 02 04	ff d7 04 02 08 0a a1 6f	...0.....o
0040	3a f5 a1 6f 3a f5 01 03	03 07	...o:....

Figura 15: tiempo de retransmisión capturado en Wireshark

4. Conclusiones

A lo largo del desarrollo del trabajo práctico se pudo observar el funcionamiento del protocolo TCP desde una perspectiva práctica, utilizando las herramientas Telnet y Wireshark para verificar su comportamiento esperado.

Las funcionalidades implementadas en los servidores permitieron experimentar con el control de sesiones, el manejo de errores y el análisis de conexiones múltiples. Además, se profundizó en los mecanismos internos del protocolo, como el control de flujo, congestión y el orden de los datos, validando su utilidad para garantizar una comunicación confiable en redes.

El uso de puertos, la identificación de sesiones mediante IP y la lógica de los flags TCP (como SYN, ACK, FIN y RST) se volvieron elementos clave en el análisis.

Finalmente, se afianzaron los conceptos teóricos y se ganó experiencia práctica en el manejo de puertos, de sesiones, análisis de paquetes e inspección de errores.

5. Bibliografía

- [1] Ross Kurose. *Redes de computadoras, un enfoque descendiente*. Pearson, 7ma edition, 2017.
- [2] Marrone Luis. Nivel de transporte - udp, tcp, optimizacion y extenciones. Facultad de ingenieria, Universidad de Buenos Aires, 2022.
- [3] Ing. Lovato Silvia. Nivel de aplicacion: Generalidades y http, smtp, dns. Facultad de ingenieria, Universidad de Buenos Aires, 2024.