



UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
Año 2025 - 1^{er} cuatrimestre

REDES DE COMUNICACIONES (TB067)

TRABAJO PRÁCTICO 1 - NIVEL DE APLICACIONES Y DE TRANSPORTE
DEL MODELO TCP/IP

ESTUDIANTES:

| | |
|----------------------------------|--------|
| Del Rio, Francisco | 110761 |
| <code>fadelrio@fi.uba.ar</code> | |
| Hurtado Giraldo, Juan Daniel | 110917 |
| <code>jdhurtado@fi.uba.ar</code> | |

Índice

| | |
|--|----------|
| 1. Introducción | 2 |
| 2. Servidores TCP | 2 |
| 2.1. Puertos de los servidores | 2 |
| 3. Limite de conexiones simultaneas | 2 |
| 4. Conexiones simultaneas | 3 |
| 5. Cierre de sesiones | 3 |
| 6. Funciones TCP | 4 |
| 7. Conclusiones | 6 |

1. Introducción

En este trabajo se utilizan dos servidores TCP para ver el funcionamiento del protocolo, comunicándose a través de telnet y capturando los paquetes con wireshark. Se abarcan desde la asignación de puertos hasta las formas de cerrar las sesiones, todo con sus debidas capturas de paquetes.

2. Servidores TCP

Para el desarrollo de este trabajo práctico se utilizan dos servidores TCP implementados en **Python**. Uno realiza un eco y el otro realiza una suma; en ambos casos esto sucede cuando reciben un comando valido, de no ser así, el servidor cierra la conexión. Ambos servidores sufren modificaciones a lo largo del desarrollo del trabajo, para cumplir con las consignas dadas.

2.1. Puertos de los servidores

Antes de ejecutar los servidores, hubo que chequear que puertos estaban disponibles, para esto se ejecutó el comando `nmap localhost` en una consola y se obtuvo lo siguiente:

```
Starting Nmap 7.92 ( https://nmap.org ) at 2025-05-14 16:28 -03
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00030s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 999 closed tcp ports (conn-refused)
PORT STATE SERVICE
631/tcp open ipp
Nmap done: 1 IP address (1 host up) scanned in 0.12 seconds
```

Se puede observar que hay un único abierto, por lo que no se puede utilizar, si se intenta iniciar alguno de los servidores en este puerto se obtiene el error:

```
OSError: [Errno 98] Address already in use
```

Finalmente, los servidores se ejecutarán en los puertos 5000 y 5001.

3. Limite de conexiones simultaneas

Se modificaron los servidores para aceptar una única conexión a la vez, y se intento establecer conexión a través de dos instancias distintas de `telnet`. En la primera instancia el servidor funciona con normalidad, sin embargo en la segunda, si bien `telnet` no registra ningún error, al enviar los comandos correctos no se obtiene respuesta, ni se cierra la sesión. Analizando los paquetes en wireshark, se ve lo siguiente:

| | | | | | | | | |
|----|--------------|-----------|-----------|-----|----|--------------|------------|-------------------------------------|
| 1 | 0.000000000 | ::1 | ::1 | TCP | 94 | 48124 → 5001 | [SYN] | Seq=0 Win=65476 Len=0 MSS=65476 SAC |
| 2 | 0.000017613 | ::1 | ::1 | TCP | 74 | 5001 → 48124 | [RST, ACK] | Seq=1 Ack=1 Win=0 Len=0 |
| 3 | 0.000097552 | 127.0.0.1 | 127.0.0.1 | TCP | 74 | 49214 → 5001 | [SYN] | Seq=0 Win=65495 Len=0 MSS=65495 SAC |
| 4 | 0.000117579 | 127.0.0.1 | 127.0.0.1 | TCP | 74 | 5001 → 49214 | [SYN, ACK] | Seq=0 Ack=1 Win=65483 Len=0 MS |
| 5 | 0.000132868 | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 49214 → 5001 | [ACK] | Seq=1 Ack=1 Win=65536 Len=0 TSval=3 |
| 6 | 10.266257187 | ::1 | ::1 | TCP | 94 | 46378 → 5001 | [SYN] | Seq=0 Win=65476 Len=0 MSS=65476 SAC |
| 7 | 10.266281763 | ::1 | ::1 | TCP | 74 | 5001 → 46378 | [RST, ACK] | Seq=1 Ack=1 Win=0 Len=0 |
| 8 | 10.266384395 | 127.0.0.1 | 127.0.0.1 | TCP | 74 | 42412 → 5001 | [SYN] | Seq=0 Win=65495 Len=0 MSS=65495 SAC |
| 9 | 10.266406787 | 127.0.0.1 | 127.0.0.1 | TCP | 74 | 5001 → 42412 | [SYN, ACK] | Seq=0 Ack=1 Win=65483 Len=0 MS |
| 10 | 10.266424380 | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 42412 → 5001 | [ACK] | Seq=1 Ack=1 Win=65536 Len=0 TSval=3 |
| 11 | 59.911801771 | 127.0.0.1 | 127.0.0.1 | TCP | 77 | 42412 → 5001 | [PSH, ACK] | Seq=1 Ack=1 Win=65536 Len=11 T |
| 12 | 59.911821137 | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 5001 → 42412 | [ACK] | Seq=1 Ack=12 Win=65536 Len=0 TSval= |

Figura 1: Captura de wireshark de dos conexiones simultaneas no soportadas por el servidor

Se puede apreciar, en las capturas 3 a 5 y 8 a 10, que en ambas instancias se completó el **handshake** (SYN - SYN,ACK - ACK), por lo que a nivel TCP la conexión se establece en ambos casos. Sin embargo, en las capturas 11 y 12 se puede apreciar como al enviar datos desde el segundo cliente, se recibe el ACK por parte del servidor, pero no se obtiene la respuesta de (en este caso) el eco.

4. Conexiones simultaneas

Ahora se modificó el servidor para que acepte más de una conexión simultanea. Esta vez, al seguir la misma secuencia de operaciones de el apartado anterior, se obtuvo la respuesta esperada del servidor en ambos casos. A continuación se ilustra lo obtenido a través de la captura de paquetes de wireshark:

| | | | | | | | |
|----|--------------|-----------|-----------|-----|----|--------------|---|
| 1 | 0.000000000 | :::1 | :::1 | TCP | 94 | 37578 → 5000 | [SYN] Seq=0 Win=65476 Len=0 MSS=65476 SA= |
| 2 | 0.000027813 | :::1 | :::1 | TCP | 74 | 5000 → 37578 | [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 3 | 0.000111199 | 127.0.0.1 | 127.0.0.1 | TCP | 74 | 45380 → 5000 | [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SA= |
| 4 | 0.000154631 | 127.0.0.1 | 127.0.0.1 | TCP | 74 | 5000 → 45380 | [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MS |
| 5 | 0.000173907 | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 45380 → 5000 | [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3 |
| 6 | 5.019965505 | :::1 | :::1 | TCP | 94 | 53936 → 5000 | [SYN] Seq=0 Win=65476 Len=0 MSS=65476 SA= |
| 7 | 5.019992065 | :::1 | :::1 | TCP | 74 | 5000 → 53936 | [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 8 | 5.020095609 | 127.0.0.1 | 127.0.0.1 | TCP | 74 | 52588 → 5000 | [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SA= |
| 9 | 5.020120566 | 127.0.0.1 | 127.0.0.1 | TCP | 74 | 5000 → 52588 | [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MS |
| 10 | 5.020145874 | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 52588 → 5000 | [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3 |
| 11 | 14.529744060 | 127.0.0.1 | 127.0.0.1 | TCP | 77 | 45380 → 5000 | [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=11 1 |
| 12 | 14.529779206 | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 5000 → 45380 | [ACK] Seq=1 Ack=12 Win=65536 Len=0 TSval= |
| 13 | 14.529928737 | 127.0.0.1 | 127.0.0.1 | TCP | 71 | 5000 → 45380 | [PSH, ACK] Seq=1 Ack=12 Win=65536 Len=5 1 |
| 14 | 14.529950067 | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 45380 → 5000 | [ACK] Seq=12 Ack=6 Win=65536 Len=0 TSval= |
| 15 | 23.093922926 | 127.0.0.1 | 127.0.0.1 | TCP | 77 | 52588 → 5000 | [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=11 1 |
| 16 | 23.093957300 | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 5000 → 52588 | [ACK] Seq=1 Ack=12 Win=65536 Len=0 TSval= |
| 17 | 23.094093527 | 127.0.0.1 | 127.0.0.1 | TCP | 71 | 5000 → 52588 | [PSH, ACK] Seq=1 Ack=12 Win=65536 Len=5 1 |
| 18 | 23.094114937 | 127.0.0.1 | 127.0.0.1 | TCP | 66 | 52588 → 5000 | [ACK] Seq=12 Ack=6 Win=65536 Len=0 TSval= |

Figura 2: Captura de wireshark de dos conexiones simultaneas soportadas por el servidor

Nuevamente, se puede observar que en las líneas 3 a 5 y 8 a 10 se realiza el **handshake**. Una instancia se conecta desde el puerto 45380 y la otra desde el 52588. Estos puertos son asignados por el sistema operativo, y dependen de la disponibilidad en el momento de la conexión, por lo que no necesariamente se utilizará el mismo en una reconexión. El único puerto que se mantiene constante es el 5000, que es el utilizado por el servidor.

5. Cierre de sesiones

Vale aclarar, esta parte del trabajo se hizo con una computadora Windows. En este sistema operativo la aplicación Telnet no permite el envío de comandos o secuencia de caracteres; efectúa el envío de un carácter por vez al servidor con lo cual no funcionan bien los servidores programados. Es así que fue necesario la instalación y utilización de Netcat para el desarrollo de la práctica.

A la hora de evaluar el cierre de sesión hubo dos casos que se analizaron. El primero fue cuando se enviaba un comando equivocado, generando así el cierre por parte del servidor; el segundo fue cuando el cierre lo efectúa el usuario. Adjunto se incluye las capturas hecha con Wireshark de los paquetes enviados para cada uno de los casos.

| | | | | | | | |
|----|-----------|-----------|-----------|-----|----|--------------|---|
| 8 | 8.711835 | 127.0.0.1 | 127.0.0.1 | TCP | 55 | 20855 → 5000 | [PSH, ACK] Seq=1 Ack=1 Win=10233 Len=11[Malformed Packet] |
| 9 | 8.711848 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 5000 → 20855 | [ACK] Seq=1 Ack=12 Win=10233 Len=0 |
| 10 | 8.723166 | 127.0.0.1 | 127.0.0.1 | TCP | 57 | 5000 → 20855 | [PSH, ACK] Seq=1 Ack=12 Win=10233 Len=13 |
| 11 | 8.723197 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 20855 → 5000 | [ACK] Seq=12 Ack=14 Win=10233 Len=0 |
| 22 | 24.283791 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 20855 → 5000 | [FIN, ACK] Seq=12 Ack=14 Win=10233 Len=0 |
| 23 | 24.283804 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 5000 → 20855 | [ACK] Seq=14 Ack=13 Win=10233 Len=0 |
| 24 | 24.283849 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 5000 → 20855 | [FIN, ACK] Seq=14 Ack=13 Win=10233 Len=0 |
| 25 | 24.283858 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 20855 → 5000 | [ACK] Seq=13 Ack=15 Win=10233 Len=0 |

Figura 3: Captura de cierre por parte del cliente

En el primer caso (Figura 3) se puede evidenciar lo siguiente:

- En la línea 8 el usuario envía la información al servidor. En total, el mensaje enviado consta de 11 bytes como consta el flag **Len**.
- En la línea 9 el servidor hace el acknowledge de la información enviada. Por este motivo el flag **Ack** tiene un valor de 12, indicando que 11 bytes fueron recibidos.
- En la línea 10 el servidor hace el envío de la respuesta al usuario. En total la respuesta consta de 13 bytes.
- En la línea 11 el usuario manda el acknowledge al servidor de que la información fue recibida. El flag **Ack** tiene un valor igual a 14, con lo cual indica que se recibieron de manera exitosa los 13 bytes enviados.

- En la línea 22 se ve como el usuario inicia el cierre enviando la solicitud al puerto 5000.
- En la línea 23 el servidor envía al usuario el acknowledgment de la solicitud de cierre.
- En la línea 24 el servidor envía su solicitud de cierre de conexión al usuario.
- En la línea 25 el usuario envía al servidor el acknowledgment de la solicitud recibida, dando así formalmente por cerrada la conexión entre ambos.

Por su parte, en el segundo caso (Figura 4) la secuencia es la siguiente:

| | | | | | | | |
|----|------------|-----------|-----------|-----|----|--------------|--|
| 38 | 111.661858 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 19614 → 5000 | [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 51 | 134.938011 | 127.0.0.1 | 127.0.0.1 | TCP | 63 | 19614 → 5000 | [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=19 |
| 52 | 134.938024 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 5000 → 19614 | [ACK] Seq=1 Ack=20 Win=2619648 Len=0 |
| 53 | 134.938206 | 127.0.0.1 | 127.0.0.1 | TCP | 81 | 5000 → 19614 | [PSH, ACK] Seq=1 Ack=20 Win=2619648 Len=37 |
| 54 | 134.938243 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 19614 → 5000 | [ACK] Seq=20 Ack=38 Win=2619648 Len=0 |
| 55 | 134.938260 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 5000 → 19614 | [FIN, ACK] Seq=38 Ack=20 Win=2619648 Len=0 |
| 56 | 134.938264 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 19614 → 5000 | [ACK] Seq=20 Ack=39 Win=2619648 Len=0 |
| 57 | 134.938331 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 19614 → 5000 | [FIN, ACK] Seq=20 Ack=39 Win=2619648 Len=0 |
| 58 | 134.938361 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 5000 → 19614 | [ACK] Seq=39 Ack=21 Win=2619648 Len=0 |

Figura 4: Captura de cierre por parte del servidor

- En la línea 51 el usuario envía la información al servidor. En total, el mensaje enviado consta de 19 bytes como consta el flag **Len**.
- En la línea 52 el servidor hace el acknowledge de la información enviada. Por este motivo el flag **Ack** tiene un valor de 02, indicando que 19 bytes fueron recibidos.
- En la línea 153 el servidor hace el envío de la respuesta al usuario. En total la respuesta consta de 37 bytes. Haciendo el análisis a partir del código, estos 37 bytes corresponden al largo del mensaje enviado cuando se recibe un comando inadecuado. Es así que podemos asegurar que en este punto el servidor envía el mensaje de error al usuario.
- En la línea 54 el usuario manda el acknowledge al servidor de que la información fue recibida. El flag **Ack** tiene un valor de 38, con lo cual indica que se recibieron de manera exitosa los 37 bytes enviados.
- En la línea 55 se ve como el servidor inicia el cierre enviando la solicitud al puerto 19614, correspondiente al del usuario.
- En la línea 56 el usuario devuelve al servidor el acknowledgment de la solicitud de cierre.
- En la línea 57 el usuario envía su solicitud de cierre de conexión al servidor.
- En la línea 58 el servidor envía el acknowledgment de la solicitud recibida, dando así formalmente por cerrada la conexión entre ambos.

Comparando ambos casos se evidencia que el cierre tiene la misma estructura en la cual ambas partes envían la solicitud de cierre y la conexión se termina formalmente con el último acknowledgment; además todos los mensajes cuentan con la flag **FIN**. La principal diferencia reside en quien comienza y termina el proceso de cierre; por su parte en el ejemplo 1 se ve como la solicitud de cierre es comenzada y terminada el usuario (líneas 22 y 25 respectivamente) mientras que en el ejemplo 2 la solicitud es comenzada por y terminada por el servidor (líneas 55 y 58 respectivamente).

Por otro lado se pudo evidenciar que con cada conexión al servidor el puerto utilizado por el usuario cambia. Es así que se comprobó como cada vez que el usuario busca conectarse al servidor asigna de forma aleatoria un puerto a partir del cual establecer la conexión.

6. Funciones TCP

En primer lugar es importante entender como se establece la conexión mediante el protocolo TCP. Para establecer la conexión entre usuario y servidor se utiliza un procedimiento llamado **three way handshake**. Para empezar el usuario envía un mensaje con el flag **SYN** buscando comenzar la conexión con el servidor; este a su vez responde con un mensaje con las flags **SYN** y **ACK** confirmando la llegada de la solicitud. Por último, el usuario envía un último acknowledgment estableciendo así finalmente la

conexión. Este proceso se puede evidenciar en la captura 5 en la cual se evidencian los tres mensajes con sus correspondientes flags.

| | | | | | | |
|----|------------|-----------|-----------|-----|----|--|
| 36 | 111.661756 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 19614 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 37 | 111.661801 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 5000 → 19614 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 38 | 111.661858 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 19614 → 5000 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |

Figura 5: Establecimiento de la conexión en TCP

Adicionalmente, en el segundo mensaje (correspondiente a la línea 37 en la captura 5) es posible analizar dos comportamientos característicos del protocolo TCP. Para ello se deben revisar los detalles del paquete que brinda Wireshark, estos se muestran en la captura ??.

```

> Frame 37: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_{...}
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 5000, Dst Port: 19614, Seq: 0, Ack: 1, Len: 0
  Source Port: 5000
  Destination Port: 19614
  [Stream index: 5]
  > [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 0 (relative sequence number)
  Sequence Number (raw): 1436209357
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 1 (relative ack number)
  Acknowledgment number (raw): 1954121222
  1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x012 (SYN, ACK)
  Window: 65535
  [Calculated window size: 65535]
  Checksum: 0xf5cb [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP),
    > TCP Option - Maximum segment size: 65495 bytes
    > TCP Option - No-Operation (NOP)
    > TCP Option - Window scale: 8 (multiply by 256)
    > TCP Option - No-Operation (NOP)
    > TCP Option - No-Operation (NOP)
    > TCP Option - SACK permitted
  > [Timestamps]
    [Time since first frame in this TCP stream: 0.000045000 seconds]
    [Time since previous frame in this TCP stream: 0.000045000 seconds]
  > [SEQ/ACK analysis]
    [This packet is the first in the stream]
    [The RTT to ACK the segment was: 0.000045000 seconds]
    [iRTT: 0.000102000 seconds]

```

Figura 6: Detalles del paquete de acknowledge

Por un lado se puede ver el parámetro Window, el cual establece el máximo número de bytes que se pueden enviar por segmento. Este es un parámetro característico del protocolo TCP en el cual el usuario y el servidor establecen un tope con respecto al número de bytes enviados por segmento de forma tal que se puede controlar y evitar en mayor medida la congestión de la red. Para nuestro caso este tope se estableció en 65535 bytes por segmento. Un dato que vale la pena revisar es el RTT que existe entre el envío del segmento y la llegada de su acknowledge el cual se puede ver en la última parte de la captura; en nuestro caso el RTT fue de 0,102 mili segundos, haciéndolo despreciable en términos macro.

Una de las funciones más importantes es la transmisión sin pérdida de datos. TCP asegura que los datos enviados lleguen al destino sin errores mediante el uso de números de secuencia y verificación de bytes. Cuando un segmento no llega al receptor, el emisor lo retransmite automáticamente. Este comportamiento puede observarse en la captura 7, donde se muestra una serie de paquetes marcados como retransmisión.

| | | | | | | | | |
|-----|------|-----------------------------------|------------|----------|------------|-----------|-----------|-----------------------|
| TCP | 1514 | [TCP Retransmission] | 9020→65241 | [ACK] | Seq=579416 | Ack=6513 | Win=10892 | Len=1460 |
| TCP | 66 | 65241→9020 | [ACK] | Seq=6513 | Ack=580876 | Win=64240 | Len=0 | SLE=592556 SRE=596936 |
| TCP | 1514 | [TCP Retransmission] | 9020→65241 | [ACK] | Seq=580876 | Ack=6513 | Win=10892 | Len=1460 |
| TCP | 66 | 65241→9020 | [ACK] | Seq=6513 | Ack=582336 | Win=64240 | Len=0 | SLE=592556 SRE=596936 |
| TCP | 1514 | [TCP Retransmission] | 9020→65241 | [ACK] | Seq=582336 | Ack=6513 | Win=10892 | Len=1460 |
| TCP | 66 | 65241→9020 | [ACK] | Seq=6513 | Ack=583796 | Win=64240 | Len=0 | SLE=592556 SRE=596936 |
| TCP | 1514 | [TCP Retransmission] | 9020→65241 | [ACK] | Seq=583796 | Ack=6513 | Win=10892 | Len=1460 |
| TCP | 66 | 65241→9020 | [ACK] | Seq=6513 | Ack=585256 | Win=64240 | Len=0 | SLE=592556 SRE=596936 |
| ARP | 42 | Gratuitous ARP for 10.151.200.167 | (Reply) | | | | | |
| TCP | 1514 | [TCP Retransmission] | 9020→65241 | [ACK] | Seq=585256 | Ack=6513 | Win=10892 | Len=1460 |
| TCP | 66 | 65241→9020 | [ACK] | Seq=6513 | Ack=586716 | Win=64240 | Len=0 | SLE=592556 SRE=596936 |
| TCP | 1514 | [TCP Retransmission] | 9020→65241 | [ACK] | Seq=586716 | Ack=6513 | Win=10892 | Len=1460 |
| TCP | 66 | 65241→9020 | [ACK] | Seq=6513 | Ack=588176 | Win=64240 | Len=0 | SLE=592556 SRE=596936 |
| TCP | 1514 | [TCP Retransmission] | 9020→65241 | [ACK] | Seq=588176 | Ack=6513 | Win=10892 | Len=1460 |
| TCP | 66 | 65241→9020 | [ACK] | Seq=6513 | Ack=589636 | Win=64240 | Len=0 | SLE=592556 SRE=596936 |
| TCP | 1514 | [TCP Retransmission] | 9020→65241 | [ACK] | Seq=589636 | Ack=6513 | Win=10892 | Len=1460 |

Figura 7: Retransmisión de paquetes en TCP

Otra característica fundamental es la entrega ordenada. Dado que los paquetes pueden llegar fuera de orden debido a la naturaleza de las redes, TCP utiliza los números de secuencia para reordenarlos antes de entregarlos a la aplicación. Este mecanismo se evidencia en la captura 8, donde se identifica paquetes recibidos fuera de orden.

| Time | Source | Destination | Protocol | Length | Info |
|---------------|--------------|--------------|----------|--------|---|
| 89.12.478644 | 172.27.9.241 | 192.168.1.11 | TCP | 1424 | 80 → 52651 [ACK] Seq=1302655196 Ack=4220141780 Win=8009 Len=1358 TSval=2587437 TSecr=4220141780 |
| 90.12.478656 | 172.27.9.241 | 192.168.1.11 | TCP | 1424 | [TCP Retransmission] 80 → 52651 [ACK] Seq=1302655196 Ack=4220141780 Win=8009 Len=1358 |
| 91.12.478665 | 172.27.9.241 | 192.168.1.11 | TCP | 1424 | [TCP Retransmission] 80 → 52651 [ACK] Seq=1302655196 Ack=4220141780 Win=8009 Len=1358 |
| 261.12.828695 | 172.27.9.241 | 192.168.1.11 | TCP | 1424 | [TCP Out-Of-Order] 80 → 52651 [ACK] Seq=1302655196 Ack=4220141780 Win=8009 Len=1358 |
| 262.12.828617 | 172.27.9.241 | 192.168.1.11 | TCP | 1424 | [TCP Out-Of-Order] 80 → 52651 [ACK] Seq=1302655196 Ack=4220141780 Win=8009 Len=1358 |
| 263.12.828625 | 172.27.9.241 | 192.168.1.11 | TCP | 1424 | [TCP Out-Of-Order] 80 → 52651 [ACK] Seq=1302655196 Ack=4220141780 Win=8009 Len=1358 |

Figura 8: Paquetes enviados fuera de orden

Muy parecido a como se establece la conexión, el cierre de esta utiliza un **four way handshake** el cual ya se discutió en la sección anterior, pero que cuenta con cuatro mensajes con los cuales se da por cerrada la conexión.

Finalmente TCP realiza la detección de errores. Cada segmento incluye un campo **Checksum** que permite verificar la integridad de los datos transmitidos. En entornos locales, como en las capturas realizadas sobre la interfaz localhost, Wireshark no puede validar el checksum y lo marca como **unverified**. Este fue nuestro caso durante la práctica, lo se puede notar claramente en la captura 6 en el cual se ve el resultado del checksum en forma hexadecimal con la aclaración **[unverified]** por parte del Wireshark.

7. Conclusiones

Se pudieron afianzar los conocimientos sobre el funcionamiento del protocolo TCP mediante el uso de los de los servidores y capturas de wireshark. Se corroboró el funcionamiento con dos clientes simultáneos, las formas de terminar la conexión y las características principales de la comunicación utilizando el protocolo.