Universidad de Buenos Aires
Facultad de Ingeniería
Año 2025 - 1er cuatrimestre

Redes de comunicaciones (TB067)

Trabajo Práctico 1 - Nivel de Aplicaciones y de Transporte del Modelo TCP/IP

ESTUDIANTES:

Del Rio, Francisco	110761
fadelrio@fi.uba.ar	
Hurtado Giraldo, Juan Daniel	110917
jdhurtado@fi.uba.ar	

Índice

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:

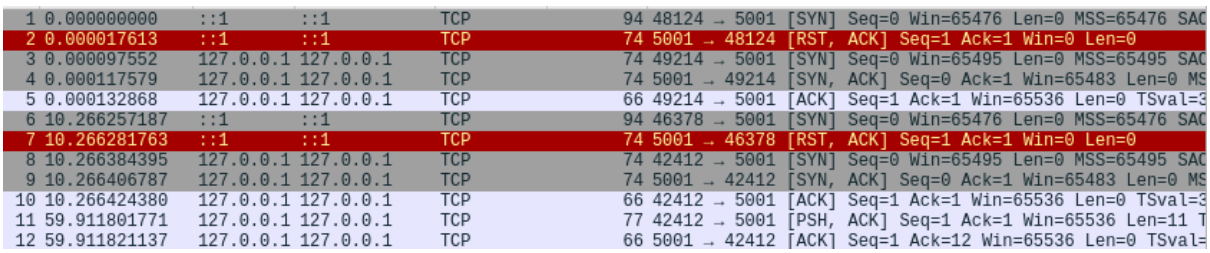

Figura 1: Captura de wireshark de dos conexiones simultaneas no soportadas por el servidor

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 SAC
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 SAC
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 SAC
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 SAC
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 T
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 T
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 T
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 T
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 lineas 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 usurio. 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 la 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:

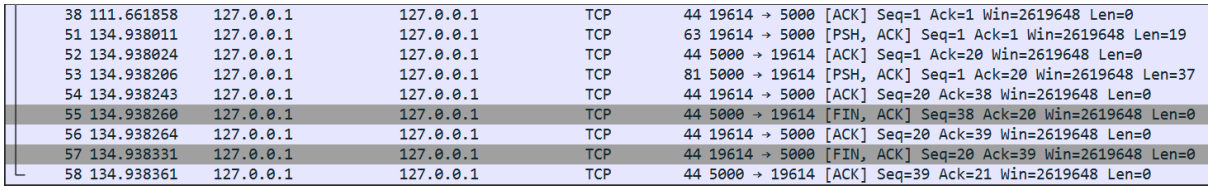

Figura 4: Captura de cierre por parte del servidor

- 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 usurio. En total la respuesta consta de 37 bytes. Haciendo el análisis a partir del código, esots 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 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 fomra aleatoria un puerto a partir del cual establecer la conexión.

6. Funciones TCP

En primer ligar 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. Est eproceso se puede evidenciar en la captura 5 en la cual se evidencian los tres mensajes con sus correspondientes flags.

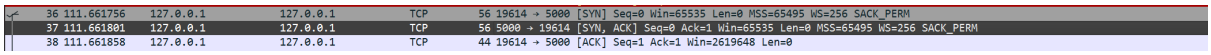

Figura 5: Establecimiento de la conexión en TCP

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 **??**.

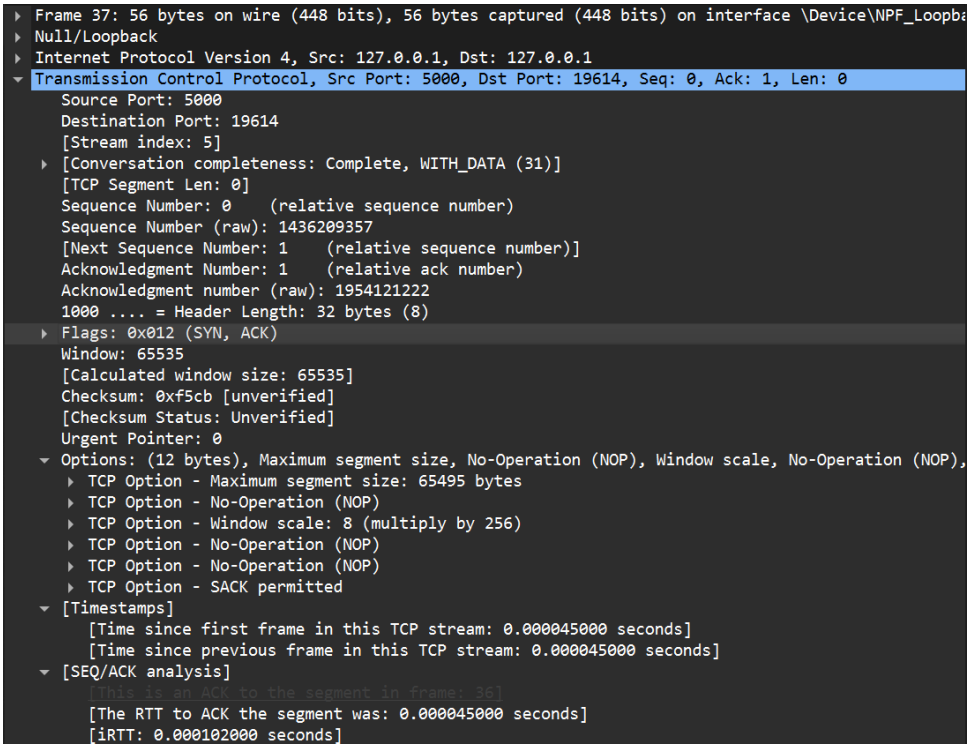

Figura 6: Detalles del paquete de acknowledge

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 perdida 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.

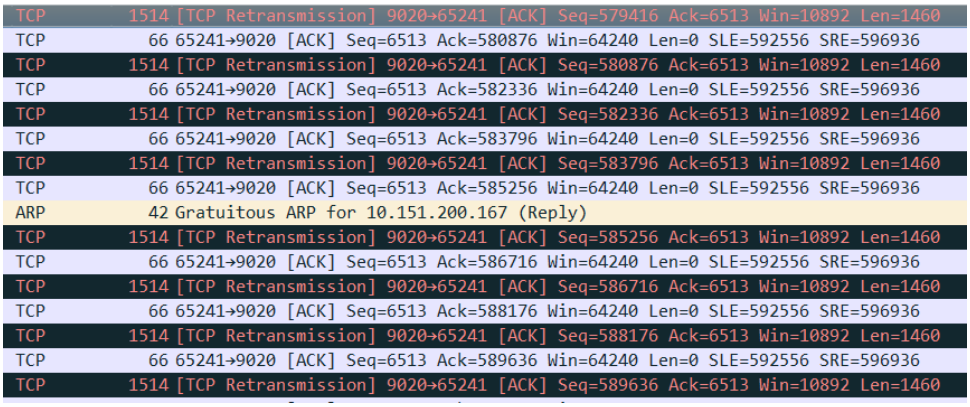

Figura 7: Retransmisión de paquetes en TCP

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.

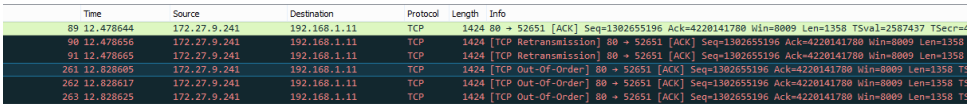

Figura 8: Paquetes enviados fuera de orden

Muy parecido a como se establece la conexión, el cierre de esta utiliza un `four way handshake` el cual ya se discutrió 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.