

Práctica 5 RyC

1. ¿Cuál es la función de la capa de transporte?	3
2. Describa la estructura del segmento TCP y UDP	3
Segmento UDP	3
Segmento TCP	3
3. ¿Cuál es el objetivo del uso de puertos en el modelo TCP/IP?	4
Multiplexación y Demultiplexación	5
Números de Puerto	5
4. Compare TCP y UDP en cuanto a:	5
a. Confiabilidad	5
b. Multiplexación	6
c. Orientado a la conexión	6
d. Controles de congestión	6
e. Utilización de puertos	7
5. La PDU de la capa de transporte es el segmento. Sin embargo, en algunos contextos suele utilizarse el término datagrama. Indique cuando	7
6. Describa el saludo de tres vías de TCP. ¿UDP tiene esta característica?	7
UDP no tiene un saludo de tres vías	8
7. Investigue qué es el ISN (Initial Sequence Number). Relaciónelo con el saludo de tres vías.	8
8. Investigue qué es el MSS. ¿Cuándo y cómo se negocia?	9
9. Utilice el comando ss (reemplazo de netstat) para obtener la siguiente información de su PC:	10
a. Para listar las comunicaciones TCP establecidas	10
b. Para listar las comunicaciones UDP establecidas	11
c. Obtener sólo los servicios TCP que están esperando comunicaciones	11
d. Obtener sólo los servicios UDP que están esperando comunicaciones	11
e. Repetir los anteriores para visualizar el proceso del sistema asociado a la conexión	12
f. Obtenga la misma información planteada en los ítems anteriores usando el comando netstat	13
10. ¿Qué sucede si llega un segmento TCP con el flag SYN activo a un host que no tiene ningún proceso esperando en el puerto destino de dicho segmento (es decir, el puerto destino no está en estado LISTEN)?	14
a. Utilice hping3 para enviar paquetes TCP al puerto destino 22 de la máquina virtual con el flag SYN activado	14
b. Utilice hping3 para enviar paquetes TCP al puerto destino 40 de la máquina virtual con el flag SYN activado	15
c. ¿Qué diferencias nota en las respuestas obtenidas en los dos casos anteriores? ¿Puede explicar a qué se debe? (Ayuda: utilice el comando ss visto anteriormente).	15
11. ¿Qué sucede si llega un datagrama UDP a un host que no tiene ningún proceso esperando en el puerto destino de dicho datagrama (es decir, que dicho puerto no está en estado LISTEN)?	16

a. Utilice hping3 para enviar datagramas UDP al puerto destino 5353 de la máquina virtual.....	16
b. Utilice hping3 para enviar datagramas UDP al puerto destino 40 de la máquina virtual.....	17
c. ¿Qué diferencias nota en las respuestas obtenidas en los dos casos anteriores? ¿Puede explicar a qué se debe? (Ayuda: utilice el comando ss visto anteriormente).	17
12. Investigue los distintos tipos de estado que puede tener una conexión TCP.....	17
13. Dada la siguiente salida del comando ss, responda:.....	19
a. ¿Cuántas conexiones hay establecidas?.....	19
b. ¿Cuántos puertos hay abiertos a la espera de posibles nuevas conexiones?.....	19
c. El cliente y el servidor de las comunicaciones HTTPS (puerto 443), ¿residen en la misma máquina?.....	20
d. El cliente y el servidor de la comunicación SSH (puerto 22), ¿residen en la misma máquina?.....	20
e. Liste los nombres de todos los procesos asociados con cada comunicación. Indique para cada uno si se trata de un proceso cliente o uno servidor.....	21
f. ¿Cuáles conexiones tuvieron el cierre iniciado por el host local y cuáles por el remoto?.....	22
g. ¿Cuántas conexiones están aún pendientes por establecerse?.....	22
14. Dadas las salidas de los siguientes comandos ejecutados en el cliente y el servidor, responder:.....	23
a. ¿Qué segmentos llegaron y cuáles se están perdiendo en la red?.....	23
b. ¿A qué protocolo de capa de aplicación y de transporte se está intentando conectar el cliente?.....	23
c. ¿Qué flags tendría seteado el segmento perdido?.....	24
15. Use CORE para armar una topología como la siguiente, sobre la cual deberá realizar:.....	25
a. En ambos equipos inspeccionar el estado de las conexiones y mantener abiertas ambas ventanas con el comando corriendo para poder visualizar los cambios a medida que se realiza el ejercicio. Ayuda: watch -n1 'ss -nat'.....	26
b. En Servidor, utilice la herramienta ncat para levantar un servicio que escuche en el puerto 8001/TCP. Utilice la opción -k para que el servicio sea persistente. Verifique el estado de las conexiones.....	28
c. Desde CLIENTE1 conectarse a dicho servicio utilizando también la herramienta ncat. Inspeccione el estado de las conexiones.....	29
d. Iniciar otra conexión desde CLIENTE1 de la misma manera que la anterior y verificar el estado de las conexiones. ¿De qué manera puede identificar cada conexión?.....	29
e. En base a lo observado en el ítem anterior, ¿es posible iniciar más de una conexión desde el cliente al servidor en el mismo puerto destino? ¿Por qué? ¿Cómo se garantiza que los datos de una conexión no se mezclarán con los de la otra?.....	30
f. Analice en el tráfico de red, los flags de los segmentos TCP que ocurren cuando:.....	30
i. Cierra la última conexión establecida desde CLIENTE1. Evalúe los estados de las conexiones en ambos equipos.....	30
ii. Corta el servicio de ncat en el servidor (Ctrl+C). Evalúe los estados de las conexiones en ambos equipos.....	31
iii. Cierra la conexión en el cliente. Evalúe nuevamente los estados de las conexiones.	31

1. ¿Cuál es la función de la capa de transporte?

La capa de transporte es fundamental en la arquitectura de red, situada entre las capas de aplicación y red. Facilita la comunicación entre aplicaciones en hosts diferentes, simulando una conexión directa entre ellos. Usa el servicio de entrega host a host de la capa de red para recibir y entregar segmentos a los procesos de aplicación, mediante multiplexación y demultiplexación.

Además, puede ofrecer servicios adicionales como transferencia de datos confiable, cifrado, control de flujo y control de congestión. En resumen, la capa de transporte asegura una comunicación confiable y eficiente entre aplicaciones, simplificando su desarrollo.

2. Describa la estructura del segmento TCP y UDP.

Segmento UDP

El segmento UDP tiene una estructura simple que consta de una cabecera de 8 bytes y un campo de datos de longitud variable. La cabecera contiene cuatro campos de 2 bytes cada uno:

- **Puerto de origen:** identifica el puerto del proceso de la aplicación en el host emisor.
- **Puerto de destino:** identifica el puerto del proceso de la aplicación en el host receptor.
- **Longitud:** indica la longitud total del segmento UDP en bytes, incluyendo la cabecera y los datos.
- **Suma de comprobación:** se utiliza para la detección de errores. Se calcula sobre la cabecera UDP, los datos y algunos campos de la cabecera IP.

El campo de datos contiene los datos de la aplicación que se van a transmitir. El tamaño del campo de datos puede variar, pero está limitado por el tamaño máximo del datagrama IP.

Segmento TCP

El segmento TCP es más complejo que el segmento UDP, con una cabecera de al menos 20 bytes y un campo de datos de longitud variable. La cabecera TCP contiene varios campos que proporcionan un control más preciso sobre la transmisión de datos y permiten implementar un servicio de transferencia confiable:

- **Puerto de origen:** identifica el puerto del proceso de la aplicación en el host emisor.

- **Puerto de destino:** identifica el puerto del proceso de la aplicación en el host receptor.
- **Número de secuencia:** identifica el primer byte del segmento dentro del flujo de bytes de la conexión TCP. Se utiliza para garantizar la entrega ordenada de los datos.
- **Número de reconocimiento:** indica el siguiente byte que el receptor espera recibir en el flujo de datos. Se utiliza para confirmar la recepción de segmentos y para indicar la ventana de recepción.
- **Longitud de cabecera:** indica la longitud de la cabecera TCP en palabras de 32 bits.
- **Reservado:** campo de 6 bits reservado para uso futuro.
- **Flags (bits de control):** un conjunto de bits que indican diferentes aspectos del segmento, como SYN (para iniciar una conexión), ACK (para confirmar la recepción), FIN (para terminar una conexión), PSH (para indicar datos urgentes), RST (para reiniciar la conexión), y URG (para indicar datos urgentes).
- **Ventana de recepción:** indica el número de bytes que el receptor está dispuesto a aceptar en su buffer. Se utiliza para controlar el flujo de datos entre el emisor y el receptor.
- **Suma de comprobación:** se utiliza para la detección de errores. Se calcula sobre la cabecera TCP, los datos y algunos campos de la cabecera IP.
- **Puntero urgente:** se utiliza para indicar la posición de los datos urgentes dentro del flujo de datos.
- **Opciones:** campo de longitud variable que puede contener información adicional, como el tamaño máximo de segmento (MSS).
- **Relleno:** se utiliza para asegurar que la cabecera TCP tenga una longitud múltiplo de 32 bits.

El campo de datos del segmento TCP contiene los datos de la aplicación que se van a transmitir. El tamaño del campo de datos está limitado por el tamaño máximo de segmento (MSS), que se negocia durante el establecimiento de la conexión.

3. ¿Cuál es el objetivo del uso de puertos en el modelo TCP/IP?

Los puertos son esenciales en el modelo TCP/IP, ya que **permiten la comunicación entre múltiples aplicaciones en un mismo host**. Son cruciales para distinguir entre diferentes aplicaciones que operan simultáneamente en un dispositivo, actuando como **puntos finales de comunicación**. Cada puerto se asocia con una aplicación específica, permitiendo que el sistema operativo entregue paquetes a la aplicación correspondiente según el número de puerto.

Sin los puertos, sería imposible que un solo host manejara múltiples aplicaciones que utilizan la red al mismo tiempo, ya que no habría forma de diferenciar los datos para cada aplicación.

Multiplexación y Demultiplexación

Los puertos permiten la **multiplexación** y **demultiplexación** de datos en la capa de transporte:

- **Multiplexación:** La capa de transporte combina datos de diferentes aplicaciones en un único flujo, asignando un puerto de origen a cada aplicación.
- **Demultiplexación:** La capa de transporte utiliza el puerto de destino para entregar datos a la aplicación correcta en el host receptor.

Números de Puerto

Los puertos se identifican por un número de 16 bits (0 a 65535):

- **Puertos bien conocidos:** Reservados para aplicaciones específicas (e.g., 80 para HTTP, 21 para FTP).
- Otros puertos son asignados dinámicamente según las necesidades de las aplicaciones.

4. Compare TCP y UDP en cuanto a:

a. Confiabilidad

- **TCP:** TCP proporciona un servicio de transferencia de datos **confiable**. Esto significa que garantiza que los datos enviados por un proceso emisor lleguen al proceso receptor sin errores, duplicaciones o pérdidas, y en el orden correcto. Para lograr esta confiabilidad, TCP utiliza mecanismos como:
 - **Números de secuencia:** Cada byte en el flujo de datos TCP se identifica con un número de secuencia único, lo que permite al receptor reordenar los datos si llegan desordenados.
 - **Reconocimientos:** El receptor envía reconocimientos (ACK) al emisor para confirmar la recepción de segmentos.
 - **Temporizadores:** El emisor utiliza temporizadores para retransmitir segmentos que no han sido reconocidos dentro de un tiempo determinado.
 - **Control de flujo:** TCP ajusta la velocidad de transmisión para evitar que el emisor sature al receptor con datos.
- **UDP:** UDP, por otro lado, ofrece un servicio **no confiable**. No garantiza la entrega de los datos, ni el orden de llegada. Si un segmento UDP se pierde, no se retransmite. Si los segmentos llegan desordenados, el receptor no los reordena. Es responsabilidad de la aplicación manejar la pérdida, duplicación o desorden de datos.

b. Multiplexación

- **TCP y UDP:** Ambos protocolos utilizan la **multiplexación** y **demultiplexación** para permitir que múltiples aplicaciones en un mismo host compartan la misma conexión de red.
 - La **multiplexación** se refiere al proceso de combinar datos de diferentes aplicaciones en el host emisor, asignando a cada flujo de datos un puerto de origen único.
 - La **demultiplexación** ocurre en el host receptor, donde la capa de transporte utiliza el puerto de destino especificado en el paquete para entregar los datos a la aplicación correcta.
- **Diferencias en la implementación:**
 - En **UDP**, la multiplexación y demultiplexación se basan únicamente en los números de puerto de origen y destino.
 - En **TCP**, el proceso es más complejo. Un socket TCP se identifica por una tupla de cuatro elementos: dirección IP de origen, puerto de origen, dirección IP de destino y puerto de destino. Esto permite una mayor precisión al dirigir los segmentos a la aplicación correcta.

c. Orientado a la conexión

- **TCP:** TCP es un protocolo **orientado a la conexión**. Antes de que los datos puedan intercambiarse, el cliente y el servidor deben establecer una conexión mediante un proceso de **acuerdo en tres fases (handshake)**. Este proceso implica el intercambio de segmentos especiales (SYN, SYN-ACK, ACK) para sincronizar los números de secuencia y confirmar que ambos lados están listos para la transmisión. Una vez establecida la conexión, los datos se transmiten como un flujo continuo de bytes. Al finalizar la comunicación, la conexión se cierra mediante un proceso de cierre.
- **UDP:** UDP es un protocolo **sin conexión**. No hay un proceso de establecimiento de conexión. Los datos se envían en paquetes individuales (datagramas) sin una secuencia o conexión establecida. Cada datagrama se trata de forma independiente.

d. Controles de congestión

- **TCP:** TCP implementa mecanismos de **control de congestión** para evitar la congestión de la red. Ajusta dinámicamente la velocidad de transmisión en función de la congestión percibida en la red.
 - TCP utiliza una **ventana de congestión** para limitar la cantidad de datos no reconocidos que un emisor puede tener en un momento dado.

- El tamaño de la ventana de congestión se ajusta en función de la pérdida de paquetes y otros factores.
- El control de congestión de TCP tiene como objetivo lograr una **equidad**, es decir, que las conexiones TCP que comparten un enlace congestionado obtengan una parte justa del ancho de banda disponible.
- **UDP:** UDP no implementa ningún mecanismo de control de congestión. Las aplicaciones UDP pueden enviar datos a la velocidad que deseen, lo que puede contribuir a la congestión de la red si no se gestiona adecuadamente.

e. Utilización de puertos

- **TCP y UDP:** Ambos protocolos utilizan **puertos** para identificar las aplicaciones que envían y reciben datos.
 - Los puertos son números de 16 bits que se especifican en la cabecera del segmento.
 - Permiten la **multiplexación** y **demultiplexación**, es decir, que múltiples aplicaciones en un mismo host puedan utilizar la red al mismo tiempo.

5. La PDU de la capa de transporte es el segmento. Sin embargo, en algunos contextos suele utilizarse el término datagrama. Indique cuando.

La PDU (Unidad de Datos de Protocolo) de la capa de transporte se conoce como **segmento**. Sin embargo, el término **datagrama** a veces se usa indistintamente, especialmente cuando se habla de **paquetes UDP**.

- **En la literatura de Internet (RFCs):** Se observa una tendencia a referirse al paquete de la capa de transporte de TCP como **segmento** y al paquete de UDP como **datagrama**.
- **Para evitar confusiones:** Dado que el término "datagrama" también se usa para los paquetes de la capa de red, es recomendable **reservar "datagrama" para los paquetes de la capa de red y usar "segmento" para los paquetes TCP y UDP**.

Esta convención ayuda a **mantener la claridad** en la terminología y a evitar confusiones cuando se discuten diferentes capas de la pila de protocolos.

6. Describa el saludo de tres vías de TCP. ¿UDP tiene esta característica?

El saludo de tres vías de TCP es un proceso de establecimiento de conexión **orientado a la conexión** que garantiza que ambos extremos, cliente y servidor, estén listos para la transmisión de datos antes de que comience el intercambio de información. Este proceso es fundamental para la fiabilidad que ofrece TCP.

A continuación, se describe el proceso paso a paso:

1. **SYN (Solicitud de sincronización):** El cliente envía un segmento TCP al servidor con el flag SYN establecido en 1. Este segmento no contiene datos de la capa de aplicación, pero incluye un número de secuencia aleatorio elegido por el cliente (cliente_isn).
2. **SYN-ACK (Sincronización-Reconocimiento):** El servidor, al recibir el segmento SYN, asigna los recursos necesarios para la conexión (buffers, variables, etc.). Luego, envía un segmento SYN-ACK al cliente. Este segmento también tiene el flag SYN establecido en 1, confirmando la recepción del SYN del cliente. Además, el servidor elige su propio número de secuencia aleatorio (servidor_isn) y lo incluye en el segmento. Finalmente, el campo de reconocimiento (ACK) se establece en cliente_isn+1, indicando que el servidor está listo para recibir el siguiente byte de datos del cliente.
3. **ACK (Reconocimiento):** El cliente, al recibir el segmento SYN-ACK, envía un segmento ACK al servidor. Este segmento tiene el flag ACK establecido en 1 y el campo de reconocimiento se establece en servidor_isn+1. Con este paso, el cliente confirma la recepción del SYN-ACK del servidor y ambos lados tienen sus números de secuencia sincronizados.

Una vez completado el saludo de tres vías, la conexión TCP se considera establecida y el intercambio de datos de la capa de aplicación puede comenzar. La conexión es full-duplex, lo que significa que ambos lados pueden enviar y recibir datos simultáneamente.

UDP no tiene un saludo de tres vías

A diferencia de TCP, **UDP es un protocolo sin conexión**, lo que significa que no hay un proceso de establecimiento de conexión antes de que comience la transmisión de datos. Los datagramas UDP se envían de forma independiente sin ninguna garantía de entrega o orden. Por lo tanto, UDP no tiene una característica similar al saludo de tres vías de TCP.

7. Investigue qué es el ISN (Initial Sequence Number). Relaciónelo con el saludo de tres vías.

El ISN (Initial Sequence Number), o Número de Secuencia Inicial, es un número aleatorio de 32 bits que se utiliza en el protocolo TCP para **identificar y numerar**

cada byte en un flujo de datos. Este número **se genera al establecer una conexión TCP** y juega un papel crucial en la **fiabilidad del protocolo**, permitiendo al receptor detectar segmentos perdidos, duplicados o desordenados.

El ISN está estrechamente **relacionado con el saludo de tres vías**, el proceso de establecimiento de la conexión en TCP. Durante este proceso, **tanto el cliente como el servidor eligen un ISN aleatorio** para la conexión:

1. **SYN:** El cliente envía un segmento SYN al servidor con su ISN (cliente_isn).
2. **SYN-ACK:** El servidor responde con un segmento SYN-ACK, incluyendo su propio ISN (servidor_isn) y un reconocimiento del ISN del cliente (cliente_isn + 1).
3. **ACK:** El cliente confirma la recepción del SYN-ACK enviando un segmento ACK, reconociendo el ISN del servidor (servidor_isn + 1).

La utilización de ISN aleatorios y el intercambio de reconocimientos durante el saludo de tres vías permite:

- **Sincronizar los números de secuencia:** El cliente y el servidor conocen el ISN del otro y pueden numerar correctamente los bytes del flujo de datos.
- **Prevenir ataques de reproducción de conexiones:** Al utilizar ISN aleatorios, se evita que un atacante pueda reproducir una secuencia de segmentos antigua para establecer una conexión fraudulenta.
- **Detectar segmentos perdidos:** Si el receptor recibe un segmento con un número de secuencia que no esperaba, puede identificar que se ha perdido un segmento anterior.

El ISN se incrementa con cada byte de datos transmitido, asegurando que cada byte tenga un número de secuencia único dentro del flujo de datos.

8. Investigue qué es el MSS. ¿Cuándo y cómo se negocia?

El MSS (Maximum Segment Size), o Tamaño Máximo de Segmento, es un parámetro en el protocolo TCP que define la **cantidad máxima de datos de la capa de aplicación que puede contener un único segmento TCP**. Este valor se especifica en bytes y **se negocia durante el saludo de tres vías de TCP**, lo que asegura que ambos extremos, cliente y servidor, estén de acuerdo en el tamaño de los segmentos que intercambiarán.

Negociación del MSS

La negociación del MSS ocurre **durante el segundo paso del saludo de tres vías**, cuando el servidor responde al segmento SYN del cliente con un segmento SYN-ACK. En este segmento, el servidor incluye una opción TCP llamada "**Maximum Segment**

Size", que indica el valor MSS que el servidor está dispuesto a aceptar. El cliente también puede incluir esta opción en su segmento SYN inicial, proponiendo su propio valor MSS.

El **valor final del MSS** para la conexión será el **menor** entre los dos valores propuestos, el del cliente y el del servidor. Esto garantiza que ambos extremos puedan manejar los segmentos sin problemas, evitando la fragmentación en la capa IP.

Relación con la MTU

El valor MSS está estrechamente **relacionado con la MTU (Maximum Transmission Unit)**, que es el **tamaño máximo de trama que puede transmitir un enlace de red**. Generalmente, el MSS se establece de manera que un segmento TCP, incluyendo la cabecera TCP/IP (generalmente 40 bytes), **quepa en una única trama de la capa de enlace**.

Por ejemplo, en redes Ethernet y PPP, la MTU es de 1500 bytes. Un valor común de MSS es 1460 bytes, lo que deja 40 bytes para las cabeceras TCP/IP.

Importancia del MSS

El uso del MSS tiene varias ventajas:

- **Evita la fragmentación:** Al asegurar que los segmentos TCP quepan en las tramas de la capa de enlace, se evita la fragmentación en la capa IP, lo que reduce la sobrecarga y mejora el rendimiento.
- **Optimiza el uso del ancho de banda:** Permite a los extremos utilizar el tamaño de segmento más eficiente para la conexión, lo que maximiza la cantidad de datos que se pueden transmitir en cada segmento.
- **Simplifica la implementación:** El cliente y el servidor solo necesitan manejar segmentos de un tamaño conocido, lo que simplifica la lógica de procesamiento de paquetes.

9. Utilice el comando ss (reemplazo de netstat) para obtener la siguiente información de su PC:

a. Para listar las comunicaciones TCP establecidas.

```
redes@debian:~/Desktop$ ss -t -a
```

```
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
```

```
LISTEN 0 128 0.0.0.0:ssh 0.0.0.0:*
```

```
LISTEN 0 128 127.0.0.1:ipp 0.0.0.0:*
```

```

LISTEN 0 5 127.0.0.1:4038 0.0.0.0:*
LISTEN 0 128 [::]:ssh [::]:*
LISTEN 0 128 [::1]:ipp [::]:*
LISTEN 0 4096 [::1]:50051 [::]:*
LISTEN 0 4096 [::ffff:127.0.0.1]:50051 :

```

b. Para listar las comunicaciones UDP establecidas.

```

redes@debian:~/Desktop$ ss -u -a
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
UNCONN 0 0 0.0.0.0:631 0.0.0.0:*
UNCONN 0 0 0.0.0.0:41742 0.0.0.0:*
UNCONN 0 0 127.0.0.1:4038 0.0.0.0:*
ESTAB 0 0 10.0.2.15%enp0s3:bootpc 10.0.2.2:bootps
UNCONN 0 0 0.0.0.0:mdns 0.0.0.0:*
UNCONN 0 0 [::]:57359 [::]:*
UNCONN 0 0 [::]:mdns [::]:*

```

c. Obtener sólo los servicios TCP que están esperando comunicaciones

```

redes@debian:~/Desktop$ ss -t -l
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
LISTEN 0 128 0.0.0.0:ssh 0.0.0.0:*
LISTEN 0 128 127.0.0.1:ipp 0.0.0.0:*
LISTEN 0 5 127.0.0.1:4038 0.0.0.0:*
LISTEN 0 128 [::]:ssh [::]:*
LISTEN 0 128 [::1]:ipp [::]:*
LISTEN 0 4096 [::1]:50051 [::]:*
LISTEN 0 4096 [::ffff:127.0.0.1]:50051 :

```

d. Obtener sólo los servicios UDP que están esperando comunicaciones.

```

redes@debian:~/Desktop$ ss -u -l
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
UNCONN 0 0 0.0.0.0:631 0.0.0.0:*
UNCONN 0 0 0.0.0.0:41742 0.0.0.0:*
UNCONN 0 0 127.0.0.1:4038 0.0.0.0:*
UNCONN 0 0 0.0.0.0:mdns 0.0.0.0:*

```

```
UNCONN 0 0 [::]:57359 [::]:*
UNCONN 0 0 [::]:mdns [::]:*
```

e. Repetir los anteriores para visualizar el proceso del sistema asociado a la conexión.

```
redes@debian:~/Desktop$ ss -u -p -l
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
UNCONN 0 0 0.0.0.0:631 0.0.0.0:*
UNCONN 0 0 0.0.0.0:41742 0.0.0.0:*
UNCONN 0 0 127.0.0.1:4038 0.0.0.0:*
UNCONN 0 0 0.0.0.0:mdns 0.0.0.0:*
UNCONN 0 0 [::]:57359 [::]:*
UNCONN 0 0 [::]:mdns [::]:*
```

```
redes@debian:~/Desktop$ ss -u -p -a
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
UNCONN 0 0 0.0.0.0:631 0.0.0.0:*
UNCONN 0 0 0.0.0.0:41742 0.0.0.0:*
UNCONN 0 0 127.0.0.1:4038 0.0.0.0:*
ESTAB 0 0 10.0.2.15%enp0s3:bootpc 10.0.2.2:bootps
UNCONN 0 0 0.0.0.0:mdns 0.0.0.0:*
UNCONN 0 0 [::]:57359 [::]:*
UNCONN 0 0 [::]:mdns [::]:*
```

```
redes@debian:~/Desktop$ ss -t -p -a
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
LISTEN 0 128 0.0.0.0:ssh 0.0.0.0:*
LISTEN 0 128 127.0.0.1:ipp 0.0.0.0:*
LISTEN 0 5 127.0.0.1:4038 0.0.0.0:*
ESTAB 0 0 10.0.2.15:54876 162.159.136.232:https
users:(("x-www-browser",pid=3395,fd=91))
ESTAB 0 0 10.0.2.15:40208 35.186.224.45:https
users:(("x-www-browser",pid=3395,fd=149))
ESTAB 0 0 10.0.2.15:45644 34.149.100.209:https
users:(("x-www-browser",pid=3395,fd=45))
ESTAB 0 0 10.0.2.15:33822 184.31.2.16:http
users:(("x-www-browser",pid=3395,fd=106))
ESTAB 0 0 10.0.2.15:49426 162.159.136.234:https
users:(("x-www-browser",pid=3395,fd=80))
```

```

ESTAB 0 0 10.0.2.15:33830 184.31.2.16:http
users:(("x-www-browser",pid=3395,fd=116))
ESTAB 0 0 10.0.2.15:33840 184.31.2.16:http
users:(("x-www-browser",pid=3395,fd=133))
ESTAB 0 0 10.0.2.15:33836 184.31.2.16:http
users:(("x-www-browser",pid=3395,fd=119))
ESTAB 0 0 10.0.2.15:45140 34.107.243.93:https
users:(("x-www-browser",pid=3395,fd=132))
ESTAB 0 0 10.0.2.15:41842 34.117.121.53:https
users:(("x-www-browser",pid=3395,fd=90))
LISTEN 0 128 [::]:ssh [::]:*
LISTEN 0 128 [::1]:ipp [::]:*
LISTEN 0 4096 [::1]:50051 [::]:*
LISTEN 0 4096 [::ffff:127.0.0.1]:50051 :
redes@debian:~/Desktop$ ss -t -p -l
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
LISTEN 0 128 0.0.0.0:ssh 0.0.0.0:*
LISTEN 0 128 127.0.0.1:ipp 0.0.0.0:*
LISTEN 0 5 127.0.0.1:4038 0.0.0.0:*
LISTEN 0 128 [::]:ssh [::]:*
LISTEN 0 128 [::1]:ipp [::]:*
LISTEN 0 4096 [::1]:50051 [::]:*
LISTEN 0 4096 [::ffff:127.0.0.1]:50051 :

```

f. Obtenga la misma información planteada en los items anteriores usando el comando netstat.

```
netstat -t -a
```

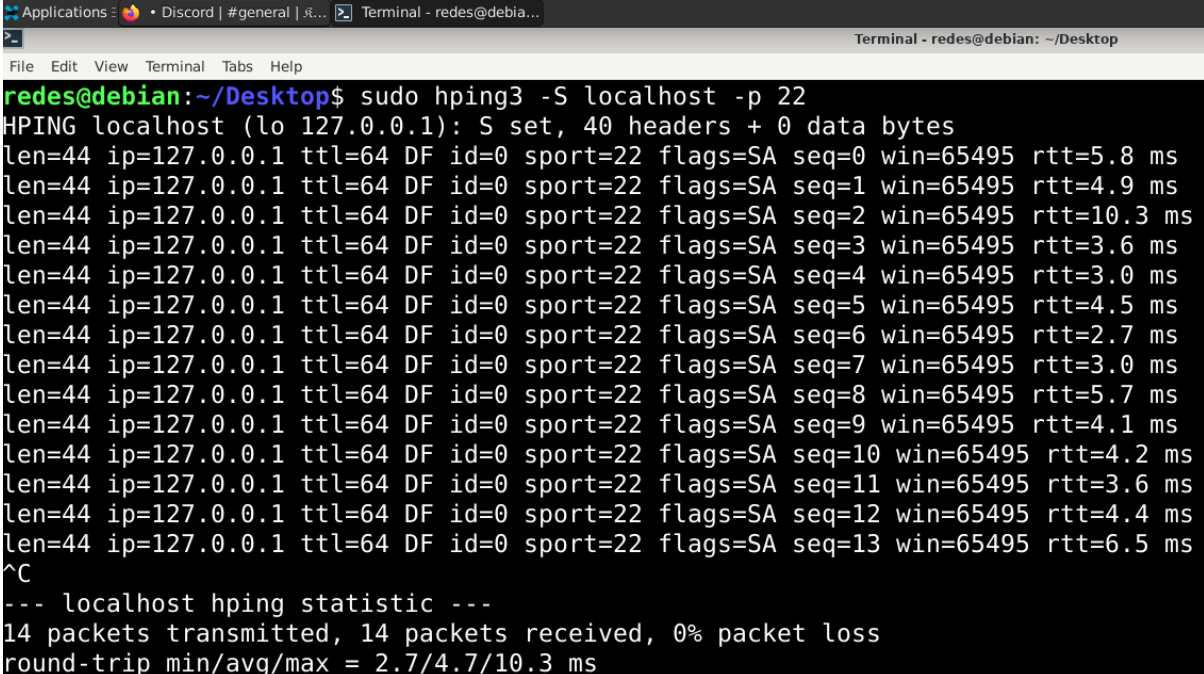
```
netstat -t -l
```

```
netstat -u -a
```

```
netstat -t -l
```

10. ¿Qué sucede si llega un segmento TCP con el flag SYN activo a un host que no tiene ningún proceso esperando en el puerto destino de dicho segmento (es decir, el puerto destino no está en estado LISTEN)?

a. Utilice hping3 para enviar paquetes TCP al puerto destino 22 de la máquina virtual con el flag SYN activado.

A screenshot of a terminal window titled "Terminal - redes@debian: ~/Desktop". The terminal shows the command "sudo hping3 -S localhost -p 22" being executed. The output displays 14 SYN packets sent to localhost (127.0.0.1) on port 22. Each line shows packet details: length (44), IP (127.0.0.1), TTL (64), DF flag, ID (0), sport (22), flags (SA), sequence number (seq=0 to seq=13), window size (win=65495), and round-trip time (rtt). The rtt values vary, with the highest being 10.3 ms. At the end, a summary shows "14 packets transmitted, 14 packets received, 0% packet loss" and "round-trip min/avg/max = 2.7/4.7/10.3 ms".

```
redes@debian:~/Desktop$ sudo hping3 -S localhost -p 22
HPING localhost (lo 127.0.0.1): S set, 40 headers + 0 data bytes
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=0 win=65495 rtt=5.8 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=1 win=65495 rtt=4.9 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=2 win=65495 rtt=10.3 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=3 win=65495 rtt=3.6 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=4 win=65495 rtt=3.0 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=5 win=65495 rtt=4.5 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=6 win=65495 rtt=2.7 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=7 win=65495 rtt=3.0 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=8 win=65495 rtt=5.7 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=9 win=65495 rtt=4.1 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=10 win=65495 rtt=4.2 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=11 win=65495 rtt=3.6 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=12 win=65495 rtt=4.4 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=SA seq=13 win=65495 rtt=6.5 ms
^C
--- localhost hping statistic ---
14 packets transmitted, 14 packets received, 0% packet loss
round-trip min/avg/max = 2.7/4.7/10.3 ms
```

b. Utilice hping3 para enviar paquetes TCP al puerto destino 40 de la máquina virtual con el flag SYN activado.

```
redes@debian:~/Desktop$ sudo hping3 -S localhost -p 40
HPING localhost (lo 127.0.0.1): S set, 40 headers + 0 data bytes
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=0 win=0 rtt=8.7 ms
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=1 win=0 rtt=7.3 ms
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=2 win=0 rtt=2.8 ms
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=3 win=0 rtt=7.3 ms
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=4 win=0 rtt=6.4 ms
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=5 win=0 rtt=12.1 ms
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=6 win=0 rtt=3.9 ms
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=7 win=0 rtt=3.2 ms
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=8 win=0 rtt=7.2 ms
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=9 win=0 rtt=6.4 ms
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=10 win=0 rtt=1.7 ms
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=11 win=0 rtt=0.4 ms
len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=RA seq=12 win=0 rtt=16.4 ms
^C
--- localhost hping statistic ---
13 packets transmitted, 13 packets received, 0% packet loss
round-trip min/avg/max = 0.4/6.4/16.4 ms
```

c. ¿Qué diferencias nota en las respuestas obtenidas en los dos casos anteriores? ¿Puede explicar a qué se debe? (Ayuda: utilice el comando ss visto anteriormente).

Las diferencia se da en las flags que retorna, SA dice que el puerto está abierto, RA que puerto está cerrado.

11. ¿Qué sucede si llega un datagrama UDP a un host que no tiene ningún proceso esperando en el puerto destino de dicho datagrama (es decir, que dicho puerto no está en estado LISTEN)?

a. Utilice hping3 para enviar datagramas UDP al puerto destino 5353 de la máquina virtual.

```
redes@debian:~/Desktop$ sudo hping3 --udp localhost -p 5353
HPING localhost (lo 127.0.0.1): udp mode set, 28 headers + 0 data bytes
^C
--- localhost hping statistic ---
12 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```


b. Utilice hping3 para enviar datagramas UDP al puerto destino 40 de la máquina virtual.

```
redes@debian:~/Desktop$ sudo hping3 --udp localhost -p 40
HPING localhost (lo 127.0.0.1): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from ip=127.0.0.1 name=localhost
status=0 port=1085 seq=0
ICMP Port Unreachable from ip=127.0.0.1 name=localhost
status=0 port=1086 seq=1
ICMP Port Unreachable from ip=127.0.0.1 name=localhost
status=0 port=1087 seq=2
ICMP Port Unreachable from ip=127.0.0.1 name=localhost
status=0 port=1088 seq=3
ICMP Port Unreachable from ip=127.0.0.1 name=localhost
status=0 port=1089 seq=4
ICMP Port Unreachable from ip=127.0.0.1 name=localhost
status=0 port=1090 seq=5
ICMP Port Unreachable from ip=127.0.0.1 name=localhost
status=0 port=1091 seq=6
ICMP Port Unreachable from ip=127.0.0.1 name=localhost
status=0 port=1092 seq=7
ICMP Port Unreachable from ip=127.0.0.1 name=localhost
status=0 port=1093 seq=8
ICMP Port Unreachable from ip=127.0.0.1 name=localhost
status=0 port=1094 seq=9
^C
--- localhost hping statistic ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 0.5/5.3/13.0 ms
```

c. ¿Qué diferencias nota en las respuestas obtenidas en los dos casos anteriores? ¿Puede explicar a qué se debe? (Ayuda: utilice el comando ss visto anteriormente).

La diferencia es que en 5353 hay un proceso escuchando, en el otro te avisa que no hay nada, (al ser UDP 5353 no indica si llegó o no).

12. Investigue los distintos tipos de estado que puede tener una conexión TCP.

Ver

https://users.cs.northwestern.edu/~agupta/cs340/project2/TCP_IP_State_Transition_Diagram.pdf

Algunos de los estados más importantes en una conexión TCP son:

- **LISTEN:** El servidor se encuentra en este estado cuando está esperando conexiones entrantes en un puerto específico. En este estado, el servidor está dispuesto a aceptar un segmento SYN de un cliente.
- **SYN-SENT:** El cliente se encuentra en este estado después de haber enviado un segmento SYN al servidor y está esperando la respuesta SYN-ACK.
- **SYN-RECEIVED:** El servidor se encuentra en este estado después de haber recibido un segmento SYN del cliente y haber enviado un segmento SYN-ACK. El servidor está esperando un segmento ACK final del cliente para establecer completamente la conexión.
- **ESTABLISHED:** Ambos extremos, cliente y servidor, se encuentran en este estado cuando la conexión TCP está completamente establecida y se pueden intercambiar datos de la capa de aplicación.
- **FIN-WAIT-1:** El cliente se encuentra en este estado después de haber enviado un segmento FIN al servidor, indicando su deseo de cerrar la conexión. El cliente espera un segmento ACK del servidor, confirmando la recepción del FIN.
- **FIN-WAIT-2:** El cliente se encuentra en este estado después de haber recibido un segmento ACK del servidor en respuesta a su segmento FIN. El cliente ahora espera un segmento FIN del servidor, indicando que el servidor también desea cerrar la conexión.
- **TIME-WAIT:** El cliente se encuentra en este estado después de haber recibido un segmento FIN del servidor y haber enviado un segmento ACK en respuesta. Este estado, también conocido como "estado de espera de 2MSL", asegura que el último segmento ACK enviado por el cliente llegue al servidor y evita que se establezcan conexiones duplicadas con los mismos números de secuencia.
- **CLOSING:** Ambos extremos se encuentran en este estado cuando han enviado un segmento FIN y están esperando el segmento ACK final del otro lado.
- **CLOSE-WAIT:** El servidor se encuentra en este estado después de haber recibido un segmento FIN del cliente y haber enviado un segmento ACK en respuesta. El servidor está esperando que la aplicación cierre la conexión, momento en el cual enviará un segmento FIN al cliente.
- **LAST-ACK:** El servidor se encuentra en este estado después de haber enviado un segmento FIN al cliente y está esperando un segmento ACK final del cliente.
- **CLOSED:** La conexión TCP se encuentra en este estado cuando se ha cerrado completamente y no se están intercambiando datos.

IMPORTANTE:

- Los estados "**semiabierto**" ocurren cuando un lado de la conexión ha iniciado el proceso de cierre, pero el otro lado aún no ha respondido o completado su parte del cierre.
- Los estados "**inactivos**" se refieren a conexiones establecidas donde no se están intercambiando datos activamente. Estos estados no se representan explícitamente en los diagramas de estados de TCP, pero son importantes para la gestión de recursos.

13. Dada la siguiente salida del comando ss, responda:

```
Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
tcp LISTEN 0 128 *:22 *:~ users:(("sshd",pid=468,fd=29))
tcp LISTEN 0 128 *:80 *:~ users:(("apache2",pid=991,fd=95))
udp LISTEN 0 128 163.10.5.222:53 *:~ users:(("named",pid=452,fd=10))
tcp ESTAB 0 0 163.10.5.222:59736 64.233.163.120:443
users:(("x-www-browser",pid=1079,fd=51))
tcp CLOSE-WAI T 0 0 163.10.5.222:41654 200.115.89.30:443
users:(("x-www-browser",pid=1079,fd=50))
tcp ESTAB 0 0 163.10.5.222:59737 64.233.163.120:443
users:(("x-www-browser",pid=1079,fd=55))
tcp ESTAB 0 0 163.10.5.222:33583 200.115.89.15:443
users:(("x-www-browser",pid=1079,fd=53))
tcp ESTAB 0 0 163.10.5.222:45293 64.233.190.99:443
users:(("x-www-browser",pid=1079,fd=59))
tcp LISTEN 0 128 *:25 *:~ users:(("postfix",pid=627,fd=3))
tcp ESTAB 0 0 127.0.0.1:22 127.0.0.1:41220 users:(("sshd",pid=1418,fd=3),
("sshd",pid=1416,fd=3))
tcp ESTAB 0 0 163.10.5.222:52952 64.233.190.94:443
users:(("x-www-browser",pid=1079,fd=29))
tcp TIME-WAIT 0 0 163.10.5.222:36676 54.149.207.17:443
users:(("x-www-browser",pid=1079,fd=3))
tcp ESTAB 0 0 163.10.5.222:52960 64.233.190.94:443
users:(("x-www-browser",pid=1079,fd=67))
tcp ESTAB 0 0 163.10.5.222:50521 200.115.89.57:443
users:(("x-www-browser",pid=1079,fd=69))
tcp SYN-SENT 0 0 163.10.5.222:52132 43.232.2.2:9500
users:(("x-www-browser",pid=1079,fd=70))
tcp ESTAB 0 0 127.0.0.1:41220 127.0.0.1:22 users:(("ssh",pid=1415,fd=3))
udp LISTEN 0 128 127.0.0.1:53 *:~ users:(("named",pid=452,fd=9))
```

a. ¿Cuántas conexiones hay establecidas?

En la salida del comando **ss**, las conexiones establecidas se indican con el estado **ESTAB**. Contando las líneas que muestran este estado, encontramos que hay **9 conexiones establecidas**.

b. ¿Cuántos puertos hay abiertos a la espera de posibles nuevas conexiones?

Los puertos abiertos a la espera de nuevas conexiones se identifican con el estado **LISTEN**. En la salida, se observan los siguientes puertos en este estado:

- `tcp LISTEN *:22`: Puerto 22 para SSH.
- `tcp LISTEN *:80`: Puerto 80 para HTTP.
- `udp LISTEN 163.10.5.222:53`: Puerto 53 para DNS (UDP).
- `tcp LISTEN *:25`: Puerto 25 para SMTP (correo electrónico).
- `udp LISTEN 127.0.0.1:53`: Puerto 53 para DNS (UDP) en la interfaz de loopback.

En total, hay **5 puertos abiertos** a la espera de nuevas conexiones.

c. El cliente y el servidor de las comunicaciones HTTPS (puerto 443), ¿residen en la misma máquina?

Para determinar si el cliente y el servidor de las comunicaciones HTTPS residen en la misma máquina, debemos analizar las direcciones IP. HTTPS utiliza el puerto 443.

Observando las líneas con `:443` en la columna `Peer Address:Port`, vemos que todas las conexiones se originan desde la dirección IP `163.10.5.222`. Ninguna de las conexiones HTTPS tiene como dirección de origen `127.0.0.1`, que es la dirección de loopback que se utiliza para referirse a la propia máquina.

Por lo tanto, podemos concluir que **el cliente y el servidor de las comunicaciones HTTPS no residen en la misma máquina**.

d. El cliente y el servidor de la comunicación SSH (puerto 22), ¿residen en la misma máquina?

Sí, hay una comunicación SSH donde cliente y servidor residen en la misma máquina. Se observa en la línea:

```
tcp ESTAB 0 0 127.0.0.1:22 127.0.0.1:41220
users:(("sshd",pid=1418,fd=3), ("sshd",pid=1416,fd=3))
```

- La dirección `127.0.0.1` es la dirección de loopback, utilizada para referirse a la propia máquina.
- El puerto 22 es el utilizado para SSH.
- El estado `ESTAB` indica una conexión establecida.

e. Liste los nombres de todos los procesos asociados con cada comunicación. Indique para cada uno si se trata de un proceso cliente o uno servidor.

Puerto	Protocolo	Proceso (PID)	Rol
22	TCP	sshd (468)	Servidor
80	TCP	apache2 (991)	Servidor
53	UDP	named (452)	Servidor
443	TCP	x-www-browser (1079)	Cliente
443	TCP	x-www-browser (1079)	Cliente
443	TCP	x-www-browser (1079)	Cliente
443	TCP	x-www-browser (1079)	Cliente
443	TCP	x-www-browser (1079)	Cliente
25	TCP	postfix (627)	Servidor
22	TCP	sshd (1418), sshd (1416)	Servidor
22	TCP	ssh (1415)	Cliente

443	TCP	x-www-browser (1079)	Cliente
443	TCP	x-www-browser (1079)	Cliente
443	TCP	x-www-browser (1079)	Cliente
443	TCP	x-www-browser (1079)	Cliente
9500	TCP	x-www-browser (1079)	Cliente
53	UDP	named (452)	Servidor

f. ¿Cuáles conexiones tuvieron el cierre iniciado por el host local y cuáles por el remoto?

La salida de `ss` no proporciona información suficiente para determinar qué lado inició el cierre de una conexión. Para saberlo, se necesitaría analizar el intercambio de paquetes con una herramienta como Wireshark y buscar los flags FIN y ACK.

g. ¿Cuántas conexiones están aún pendientes por establecerse?

Hay una conexión en estado `SYN-SENT`, lo que significa que el cliente ha enviado un SYN pero aún no ha recibido un SYN-ACK del servidor.

```
tcp SYN-SENT 0 0 163.10.5.222:52132 43.232.2.2:9500
users:(("x-www-browser",pid=1079,fd=70))
```

Por lo tanto, hay **1 conexión pendiente por establecerse**.

14. Dadas las salidas de los siguientes comandos ejecutados en el cliente y el servidor, responder:

```
servidor# ss -natu | grep 110
```

```
tcp LISTEN 0 0 *:110 *:*
```

```
tcp SYN-RECV 0 0 157.0.0.1:110 157.0.11.1:52843
```

```
cliente# ss -natu | grep 110
```

```
tcp SYN-SENT 0 1 157.0.11.1:52843 157.0.0.1:110
```

a. ¿Qué segmentos llegaron y cuáles se están perdiendo en la red?

Analizando las salidas de los comandos **ss** en el servidor y el cliente, podemos deducir la siguiente secuencia de eventos:

1. **El cliente envió un segmento SYN al servidor en el puerto 110.** Esto se evidencia por el estado **SYN-SENT** en la salida del cliente. [Cliente: tcp SYN-SENT]
2. **El servidor recibió el segmento SYN y respondió con un segmento SYN-ACK.** El estado **SYN-RECV** en el servidor indica que se ha recibido un SYN y se ha enviado un SYN-ACK. [Servidor: tcp SYN-RECV]

En este punto, el cliente debería haber enviado un segmento ACK al servidor para completar el establecimiento de la conexión TCP (handshake de tres vías). Sin embargo, el cliente todavía se encuentra en estado **SYN-SENT**, lo que significa que **el segmento SYN-ACK enviado por el servidor se está perdiendo en la red.** [Cliente: tcp SYN-SENT]

b. ¿A qué protocolo de capa de aplicación y de transporte se está intentando conectar el cliente?

El puerto 110 es el puerto bien conocido para el protocolo **POP3** (Post Office Protocol version 3), utilizado para **recibir correos electrónicos** desde un servidor.

El protocolo de transporte utilizado es **TCP**, como se indica en la salida del comando **ss**. [Cliente: tcp SYN-SENT, Servidor: tcp LISTEN, Servidor: tcp SYN-RECV]

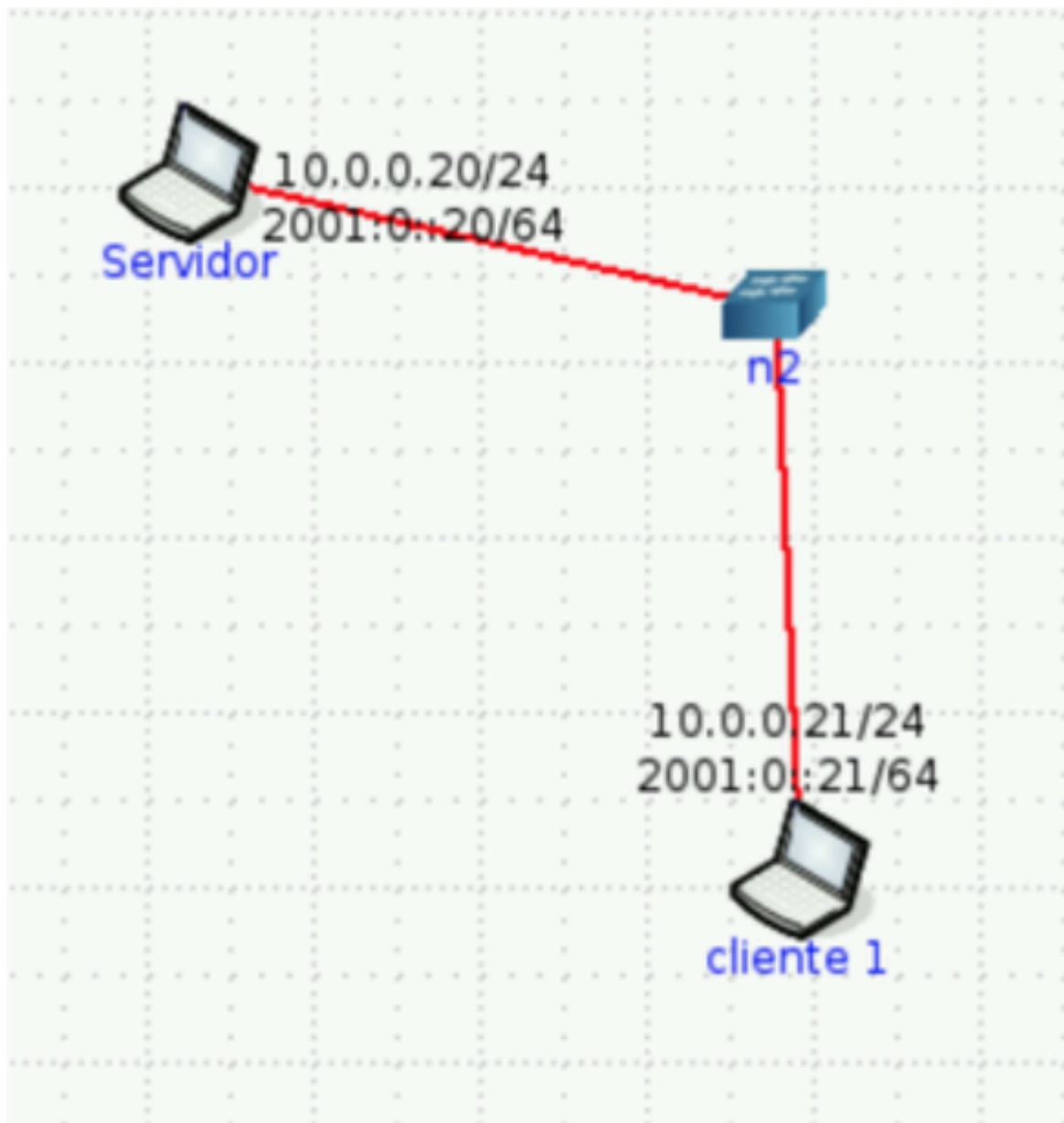
c. ¿Qué flags tendría seteado el segmento perdido?

El segmento perdido es el **SYN-ACK** enviado por el servidor al cliente. Este segmento tiene los siguientes flags seteados:

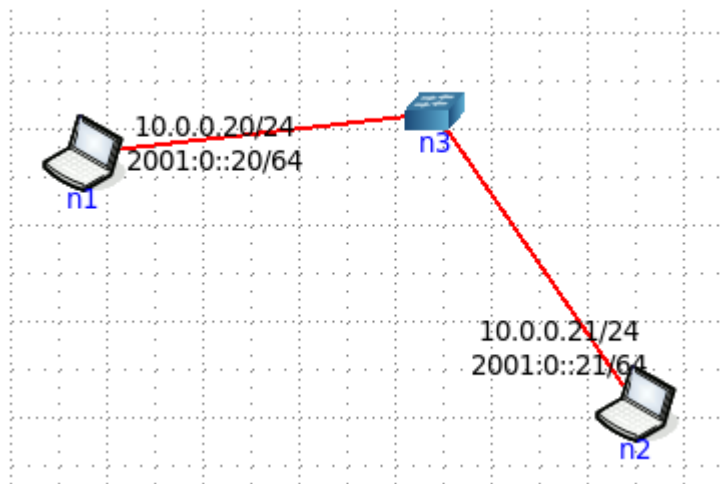
- **SYN:** Para indicar la sincronización de números de secuencia.
- **ACK:** Para confirmar la recepción del segmento SYN del cliente.

Es importante recordar que el handshake de tres vías de TCP requiere el intercambio de estos flags para establecer correctamente la conexión.

15. Use CORE para armar una topología como la siguiente, sobre la cual deberá realizar:

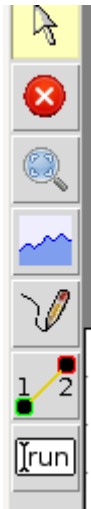


MI CREACION:

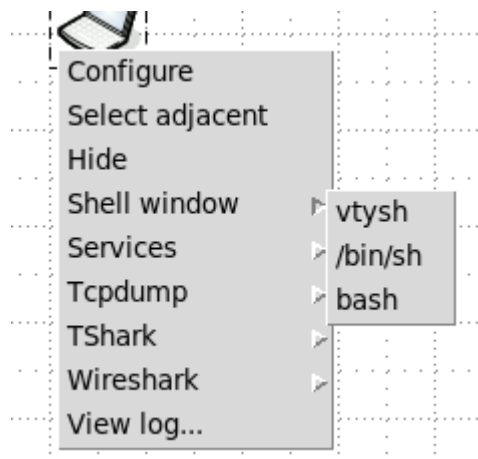


a. En ambos equipos inspeccionar el estado de las conexiones y mantener abiertas ambas ventanas con el comando corriendo para poder visualizar los cambios a medida que se realiza el ejercicio. Ayuda: `watch -n1 'ss -nat'`.

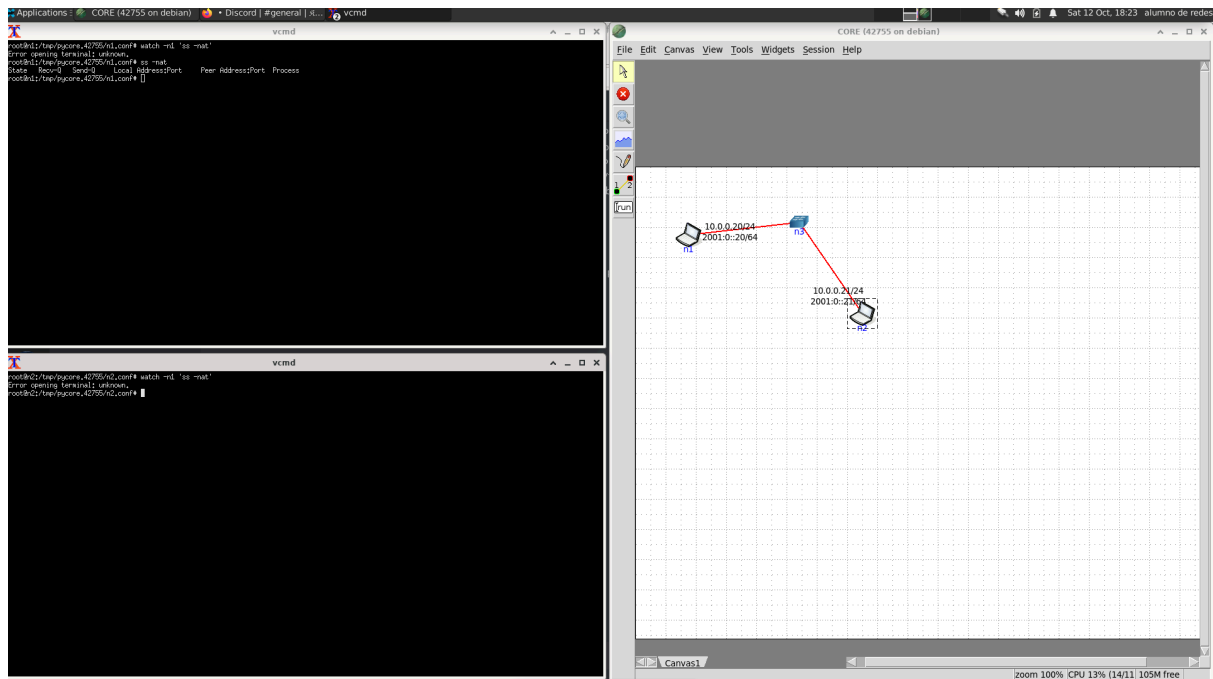
Presioné:



(Run [flechita verde])

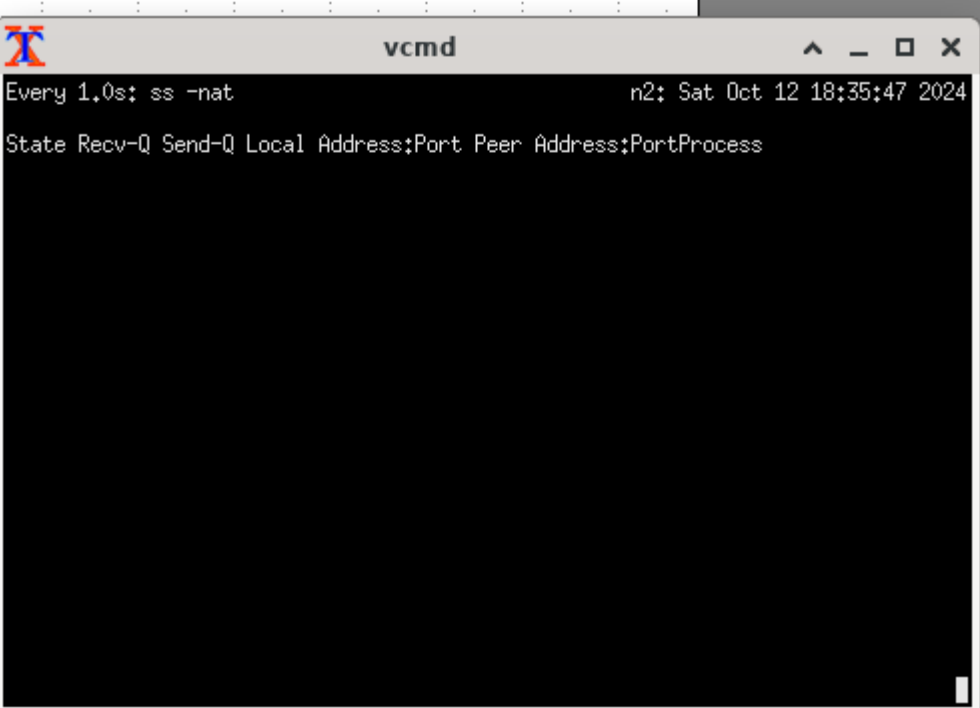


Acá elegí bash



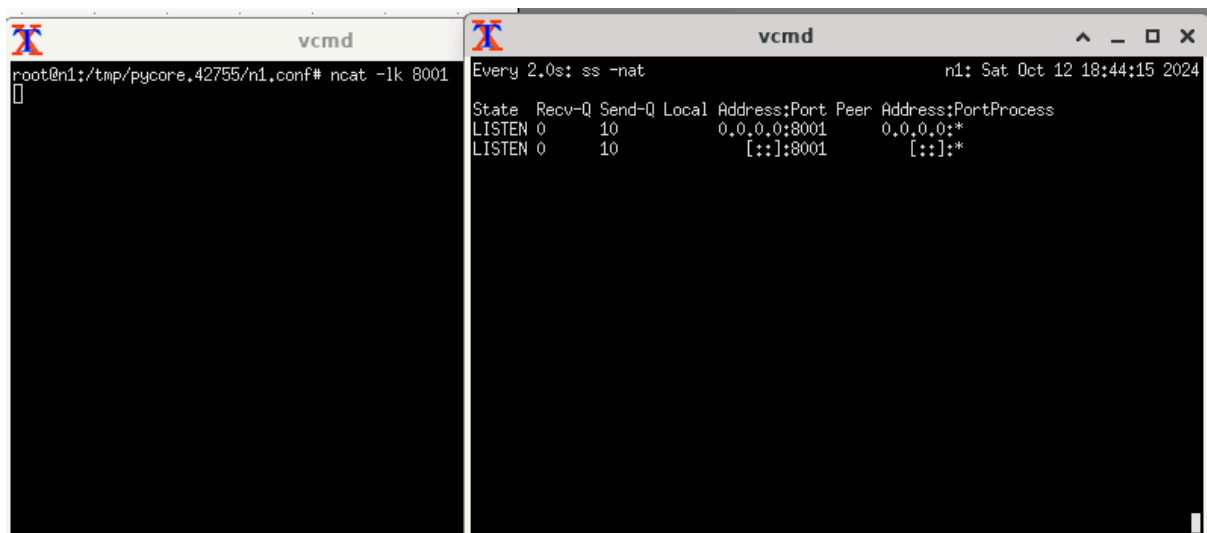
El comando no sirve 'watch' no anda acá, así que tire directamente:

```
root@n2:/tmp/pycore.42755/n2.conf# export TERM=xterm
root@n2:/tmp/pycore.42755/n2.conf# watch -n1 "ss -nat"
```



```
vcmd
Every 1.0s: ss -nat n2: Sat Oct 12 18:35:47 2024
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
```

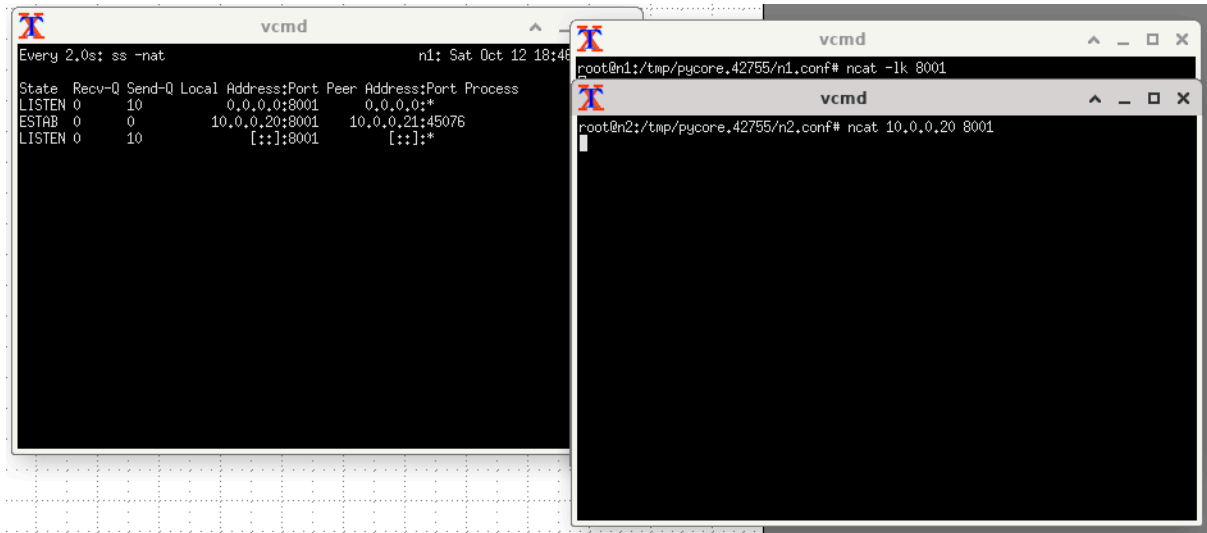
b. En Servidor, utilice la herramienta ncat para levantar un servicio que escuche en el puerto 8001/TCP. Utilice la opción -k para que el servicio sea persistente. Verifique el estado de las conexiones.



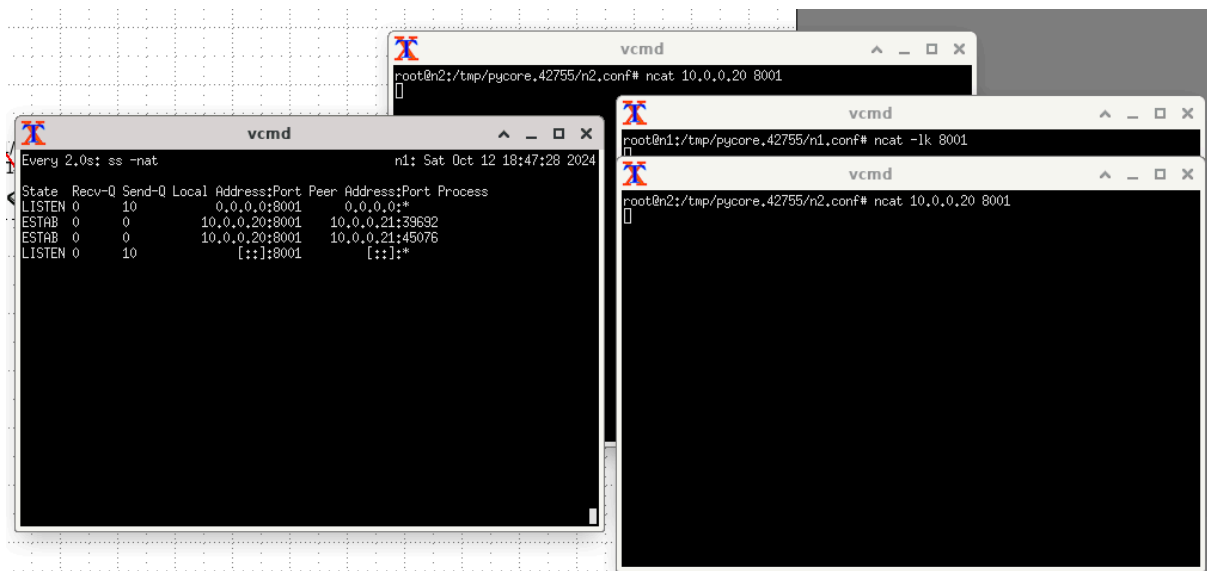
```
vcmd
root@n1:/tmp/pycore.42755/n1.conf# ncat -lk 8001
[]

vcmd
Every 2.0s: ss -nat n1: Sat Oct 12 18:44:15 2024
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
LISTEN 0 10 0.0.0.0:8001 0.0.0.0:*
LISTEN 0 10 [::]:8001 [::]:*
```

c. Desde CLIENTE1 conectarse a dicho servicio utilizando también la herramienta ncat. Inspeccione el estado de las conexiones.



d. Iniciar otra conexión desde CLIENTE1 de la misma manera que la anterior y verificar el estado de las conexiones. ¿De qué manera puede identificar cada conexión?



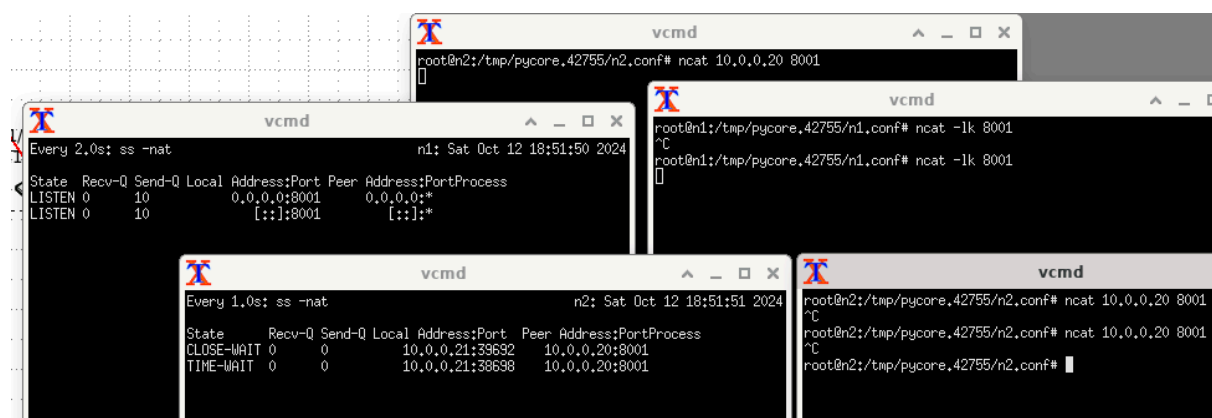
Se pueden identificar gracias a que los clientes están en puertos distintos.

e. En base a lo observado en el ítem anterior, ¿es posible iniciar más de una conexión desde el cliente al servidor en el mismo puerto destino? ¿Por qué? ¿Cómo se garantiza que los datos de una conexión no se mezclarán con los de la otra?

Si, es posible ya que los distintos clientes tienen distintos puertos orígenes y eso garantiza que sea posible que se inicie más de una conexión desde el cliente al servidor en el mismo puerto destino ya que cada conexión se identifica de manera única por la combinación de la dirección IP y el número de puerto local y remoto. Cada conexión se gestiona de manera individual y se mantiene separada de las demás gracias a la combinación única de direcciones IP y puertos.

f. Analice en el tráfico de red, los flags de los segmentos TCP que ocurren cuando:

i. Cierra la última conexión establecida desde CLIENTE1. Evalúe los estados de las conexiones en ambos equipos.



The image shows four terminal windows titled 'vcmd' running on different hosts. The top-left window shows the output of 'ss -nat' on host 'n1', displaying two listening sockets on port 8001. The top-right window shows the execution of 'ncat -lk 8001' on host 'n1'. The bottom-left window shows the output of 'ss -nat' on host 'n2', displaying two connections in 'CLOSE-WAIT' and 'TIME-WAIT' states. The bottom-right window shows the execution of 'ncat 10.0.0.20 8001' on host 'n2'.

```
vcmd
root@n2:/tmp/pycore.42755/n2.conf# ncat 10.0.0.20 8001

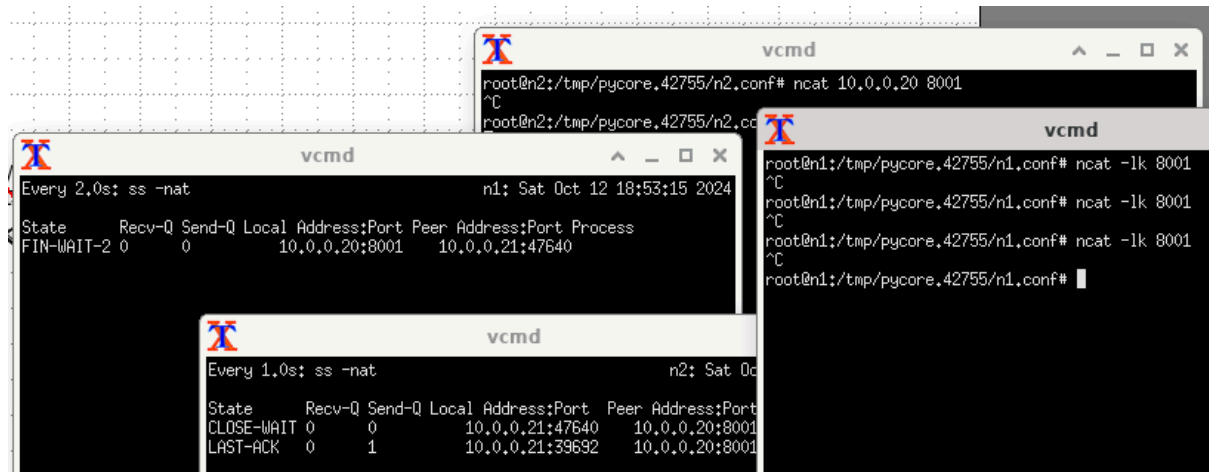
vcmd
Every 2.0s: ss -nat                               n1: Sat Oct 12 18:51:50 2024
State  Recv-Q  Send-Q  Local Address:Port  Peer Address:PortProcess
LISTEN 0        10      0.0.0.0:8001        0.0.0.0:*
LISTEN 0        10      [::]:8001          [::]:*

vcmd
root@n1:/tmp/pycore.42755/n1.conf# ncat -lk 8001
^C
root@n1:/tmp/pycore.42755/n1.conf# ncat -lk 8001

vcmd
Every 1.0s: ss -nat                               n2: Sat Oct 12 18:51:51 2024
State  Recv-Q  Send-Q  Local Address:Port  Peer Address:PortProcess
CLOSE-WAIT 0        0      10.0.0.0.21:39692   10.0.0.20:8001
TIME-WAIT 0        0      10.0.0.0.21:38698   10.0.0.20:8001

vcmd
root@n2:/tmp/pycore.42755/n2.conf# ncat 10.0.0.20 8001
^C
root@n2:/tmp/pycore.42755/n2.conf# ncat 10.0.0.20 8001
^C
root@n2:/tmp/pycore.42755/n2.conf#
```

ii. Corta el servicio de ncat en el servidor (Ctrl+C). Evalúe los estados de las conexiones en ambos equipos.



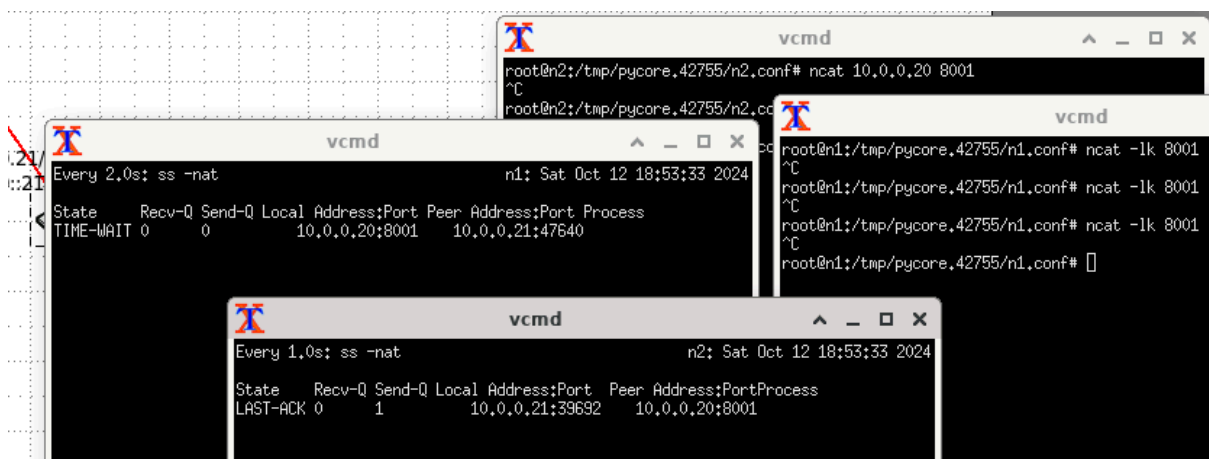
The screenshot shows three terminal windows titled 'vcmd'. The top-right window is the server terminal (n2) running 'ncat 10.0.0.20 8001'. The bottom-left window is the client terminal (n1) running 'ss -nat' with output:

```
Every 2.0s: ss -nat n1: Sat Oct 12 18:53:15 2024
State      Recv-Q Send-Q Local Address:Port Peer Address:Port Process
FIN-WAIT-2 0      0      10.0.0.20:8001    10.0.0.21:47640
```

The middle window is another client terminal (n2) running 'ss -nat' with output:

```
Every 1.0s: ss -nat n2: Sat Oct 12 18:53:15 2024
State      Recv-Q Send-Q Local Address:Port Peer Address:Port Process
CLOSE-WAIT 0      0      10.0.0.21:47640   10.0.0.20:8001
LAST-ACK   0      1      10.0.0.21:39692   10.0.0.20:8001
```

iii. Cierra la conexión en el cliente. Evalúe nuevamente los estados de las conexiones.



The screenshot shows the same three terminal windows after the client connection has been closed. The top-right window (server n2) remains the same. The bottom-left window (client n1) shows the 'ss -nat' output:

```
Every 2.0s: ss -nat n1: Sat Oct 12 18:53:33 2024
State      Recv-Q Send-Q Local Address:Port Peer Address:Port Process
TIME-WAIT  0      0      10.0.0.20:8001    10.0.0.21:47640
```

The middle window (client n2) shows the 'ss -nat' output:

```
Every 1.0s: ss -nat n2: Sat Oct 12 18:53:33 2024
State      Recv-Q Send-Q Local Address:Port Peer Address:Port Process
LAST-ACK   0      1      10.0.0.21:39692   10.0.0.20:8001
```