**Capa de Transporte - Parte I Redes y comunicaciones - 2022**

**Práctica 5**

**Capa de Transporte - Parte I**

**13, 14, 15a**

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

* MUX/DEMUX App. to App. (puertos, Ports). El nº de puerto identifica los procesos que corren y usan servicios de puerto
* Soporte de datos de tamanos arbitrarios. IP no ofrece eso, maneja paquetes autocontenidos (Datagramas)
* Control y Deteccion de Errores, pérdida, duplicación, se corrompen.
* ¿Como enviar info sobre la red de acuerdo al estado de la misma? ¿Cuándo y Cómo una App. debe
* enviar datos? 🡺 Cuando el receptor puede recibirla y la red mandarla
  + Control de Flujo.
  + Control de Congestion. ´
* Dos modelos:
  + Modelo Confiable: TCP. 🡪 No da seguridad
  + Modelo NO Confiable: UDP. 🡪 Implementa lo mínimo sobre IP

Su función es la de proveer un mecanismo de comunicación lógico entre procesos, aunque se ejecuten en máquinas distintas interconectadas mediante alguna red.

Los datos de las aplicaciones del usuario se empaquetan, transportan y entregan al servidor, o aplicación de destino. Los procesos de la capa de Transporte aceptan datos de la capa de aplicación y los preparan para el direccionamiento en la capa de red, es decir, es responsable de la transferencia de extremo a extremo de los datos de aplicación. Lo que se busca es lograr “una comunicación lógica entre procesos”, es decir que, desde la perspectiva de la aplicación, es como si los hosts que ejecutan los procesos estuvieran conectados directamente.

La capa de Transporte permite la segmentación de datos y brinda el control necesario para reensamblar las partes dentro de los distintos streams de comunicación. Las responsabilidades que debe cumplir son:

- Seguimiento de la comunicación individual entre aplicaciones en los host origen y destino.

- Segmentación de datos y gestión de cada porción.

- Reensamble de segmentos en flujos de datos de aplicación.

- Identificación de las diferentes aplicaciones. (los datos de las distintas aplicaciones que están ejecutándose en la misma computadora no deben mezclarse).

Existen varios protocolos para la capa de Transporte. Para algunas aplicaciones, los segmentos deben

llegar en una secuencia específica. En algunos casos, todos los datos deben recibirse para ser utilizados. En otros casos, una aplicación puede tolerar cierta pérdida de datos durante la transmisión a través de la red.

En general, todos los protocolos de la capa incluyen:

- Segmentación y reensamblaje: se dividen los datos de las aplicaciones en bloques de tamaño adecuado para poder enviarlos. En el destino, deben volver a ensamblarse.

- Multiplexación de conversaciones: debido a que puede haber varias aplicaciones corriendo en la misma computadora, a cada una se le asigna una dirección (puerto) para que la capa de Transporte pueda determinar con qué aplicación o servicio se identifican los datos.

Estas funciones son posibles gracias a los encabezados de la información que se comparte.

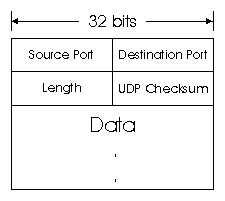
**2. Describa la estructura del segmento TCP y UDP.**

UDP

Datagrama (UDP)

Puertos: MUX/DEMUX.

Longitud: UDP HDR + Payload.



* Source port: 16 bits, puerto origen. Opcional, ya que UDP no solicita respuestas; si no se usa puede ponerse en 0.
* Destination port: 16 bits, puerto destino
* Length: 16 bits, longitud total del segmento (encabezados + datos). Valor mínimo: 8 bytes.
* UDP checksum: 16 bits son la suma de verificación del segmento. Evita errores relacionados a datos que se corrompieron o bits que se cambiaron, en este caso se descartan silenciosamente. Evita errores de un bit, mas no, hay otros controles mejores en capa de enlace, asi que puede deshabilitarse en IPV4 si se pone 0x0000. Se calcula en origen y destino.

Checksum

• Calculo Ca1, Opcional. 0 = Sin checksum. ´

• Calculado HDR + PseudoHDR + Payload.

• PseudoHDR: IP.SRC + IP.DST + Zero + IP.PROTO + UDP.LENGTH.

• PseudoHDR: proteccion contra paquetes mal enrutados. ´

• Aplicaciones de LAN por eficiencia lo podr´ıan deshabilitar.

• Si tiene error se descarta silenciosamente.

TCP

Puertos: MUX/DEMUX.

No tiene Longitud total, si de HDR LEN (variable, max 60B Unit=4B).

Total LEN se computa para PseudoHDR, no viaja en el segmento.

Checksum:

• Calculo Ca1. Obligatorio, calculado, igual que UDP. ´

• Si tiene error podría pedir retransmisión, implementación de TCP descarta y espera RTO

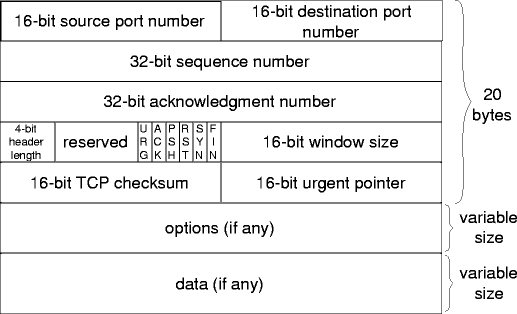
(Retransmission Timer).

Necesidad de manejar Timers, RTO (tmout. por cada segmento). (implementaciones lo manejan mas

eficiente). ´

Permite Opciones y Negociacion. ´

TCP entrega y envía lo datos agrupados o separados de forma dis-asociada de la aplicacion: ´



* Source port number: 16 bits, puerto origen
* Destination port number: 16 bits, puerto destino.

Estos dos se usan para multiplexar y demultiplexar los datos y para las aplicaciones de la capa

superior.

* Sequence number: 32 bits, número de secuencia del segmento. Se usa para asegurar que los datos llegan en el orden deseado. Al inicio de la conexión, se le asigna un valor aleatorio
* Acknowledgment number:32 bits, número de recibo del segmento. Almacena el número de segmento, del siguiente byte que el receptor espera recibir.

Cuando el receptor envía el valor X, significa que pudo recibir correctamente todos los índices menores a X. Cuando se recibe cada paquete, toma numSecuencia + 1 y lo envía al emisor

* Header length: 4 bits, longitud del encabezado (Se aclara porque una parte es opcional).
* Reserved: 6 bits reservados, no se usan
* Flag URG: 1 bit. Si está en 1, significa que los datos recibidos en el segmento son urgentes
* Flag ACK (Campo de Control de Errores): Si está en 1, significa que el número de recibo es válido. Vale 1 en todos los segmentos, menos el de solicitud (al iniciar una conexión usando saludo de 3 vías)
* Flag PSH: Si está en 1, el receptor enviará inmediatamente el buffer entero a la capa de aplicación (sin esperar a que se llene). Luego, lo vaciará. No prioriza los datos, y éstos pueden llegar desordenados a la capa de aplicación, si aún no fueron ordenados. Ejemplo de la información que maneja es la pulsación de teclas.
* Flag RST: Si está en 1, el receptor finalizará la conexión de forma abrupta (Condición anómala). Ambos liberarán sus recursos, y pueden perderse los datos que estaban viajando. Se usa cuando hay errores irrecuperables, y no se puede finalizar la conexión de forma limpia.
* Flag SYN (Campo de sesión): Si está en 1, significa que el número de secuencia es el inicial de la conexión, se quiere establecer una conexión nueva. Vale 0 en todos los segmentos, menos el de solicitud (al iniciar una conexión usando saludo de tres vías)
* Flag FIN (Campo de sesión): Si está en 1, significa que el emisor quiere finalizar la conexión limpiamente. Vale 0 en todos los segmentos, menos el de finalización.
* Windows size (Campo de Control de Flujo): 16 bits, longitud de la ventana de recepción del emisor. Almacena la cantidad máxima de bytes que el emisor puede recibir sin enterarse explícitamente. Controles de flujo y envió de información eficiente. Se incrementa hasta detectar congestiones, donde se reduce para evitar perder paquetes.
* TCP checksum (Campo de Deteccion de Errores): 16 bits, suma de verificación del segmento. Lo mismo que UDP, pero acá es obligatorio. Se usa para verificar si los datos llegaron correctamente. Sino, el receptor lo rechaza y solicita que se vuelvan a enviar. Se calcula usando CRC
* Urgent pointer: 16 bits, puntero urgente. Almacena la cantidad de bytes urgentes, contenidos en el segmento. Al sumarlo al número de secuencia, se obtiene el primer byte no-urgente. Se usa sólo cuando el flag URG es 1
* El campo opciones es opcional y de longitud variable. Se utiliza cuando un emisor y un receptor negocian el tamaño máximo de segmento (MSS) o como un factor de escala de la ventana en las redes de alta velocidad. También se define una opción de marca temporal.
* Opcionalmente, pueden incluirse otras opciones (Tamaño variable):
  + Timestamp: Sirve para ordenar segmentos, cuando más de uno tiene el mismo número de secuencia. También se usa para que ambos extremos calculen el RTT con el otro
  + Extensión de longitud de ventana: Permite que el emisor notifique que puede recibir más de 2^16-1 bytes sin enterarse (máximo 2^32-1)
  + Negociación de parámetros: El emisor y el receptor establecen al inicio su longitud máxima de segmento, etc
  + Relleno: Bytes nulos que se agregan al final, para que la longitud total de los headers sea múltiplo de 4
* Data: Tamaño variable, datos que se desean enviar

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

Brindar múltiples canales de comunicación a través de una misma conexión física. Un proceso de la máquina A se comunicará con uno de la B usando uno de estos canales

Un proceso puede tener uno o más sockets, puertas por las que pasan los datos de la red al proceso, y viceversa. Por tanto, la capa de transporte del host receptor realmente no entrega los datos directamente a un proceso, sino a un socket intermedio, que a su vez tiene un identificador único (ya que puede haber más de uno por host).

Cada segmento de la capa de transporte contiene un conjunto de campos destinados para identificar el socket apropiado (entre otros). En el extremo receptor, la capa de transporte examina estos campos para identificar el socket receptor y, a continuación, envía el segmento a dicho socket. Esta tarea de entregar los datos contenidos en un segmento de la capa de transporte al socket correcto es lo que se conoce como demultiplexación. La tarea de reunir los fragmentos de datos en el host de origen desde los diferentes sockets (en caso de que el proceso cuente con diferentes sockets), encapsulando cada fragmento de datos

con la información de cabecera (el cual se utilizará después en el proceso de demultiplexación) para crear los segmentos y pasarlos a la capa de red es lo que se denomina multiplexación. Es importante darse cuenta de que estas técnicas son necesarias siempre que un único protocolo en una capa (cualquiera) sea utilizado por varios protocolos de la capa inmediatamente superior.

Sabemos que la operación de multiplexación que se lleva a cabo en la capa de transporte requiere que los sockets tengan identificadores únicos y que cada segmento tenga campos especiales que indiquen el socket al que tiene que entregarse el segmento. Estos campos especiales, son el campo número de puerto origen y número de puerto destino.

En Resumen: se usan los puertos para poder identificar los procesos (socket) con los que se mantiene una conexión y se intercambian mensajes.

**4. Compare TCP y UDP en cuanto a:**

**a. Confiabilidad.**

**b. Multiplexación.**

**c. Orientado a la conexión.**

**d. Controles de congestión.**

**e. Utilización de puertos.**

**a. Confiabilidad.**

El protocolo IP proporciona una comunicación lógica entre hosts. IP hace todo lo que puede por entregar los segmentos entre los hosts que se están comunicando, pero no garantiza la entrega. Por lo tanto, IP es un servicio NO FIABLE.

|  |  |
| --- | --- |
| UDP | TCP |
| ● Servicio no fiable.  ● No garantiza que los segmentos lleguen al proceso destino.  ● Tampoco garantiza que los lleguen en orden y estén libres de errores.  ● La comprobación de errores es responsabilidad de la aplicación. | ● Transferencia de datos fiable mediante control de flujo, números de secuencia, mensajes de reconocimiento y temporizadores.  ● Garantiza que los datos transmitidos por el proceso emisor sean entregados al proceso receptor correctamente y en orden. Continuará reenviando un segmento hasta que la recepción del mismo haya sido confirmada por el destino. |

**b. Multiplexación.**

|  |  |
| --- | --- |
| UDP | TCP |
| ● Utiliza multiplexación y demultiplexación SIN conexión.  ● Creamos sockets indicando el número de puerto.  ● El segmento (de UDP + IP) identifica el socket destino mediante dos campos de cabecera: Dirección IP Destino y Nro. Puerto Destino.  ● El segmento tiene una “dirección de retorno”, por si el receptor desea devolver un segmento al emisor.  ● Si dos segmentos UDP con diferentes sockets de origen (Dirección IP y/o Nro. Puerto) poseen la misma dirección IP destino y el mismo nro. de puerto de destino, entonces los dos segmentos se enviarán al mismo proceso de destino a través del mismo socket de destino.  ● Cabeceras de 8 bytes. Más pequeñas. | ● Utiliza multiplexación y demultiplexación orientada a la conexión.  ● El socket, se identifica por cuatro elementos: Dirección IP Origen, Nro. Puerto Origen, Dirección IP Destino, Nro. Puerto Destino.  ● Dos segmentos entrantes con direcciones IP de origen o números de puerto de origen diferentes (con la excepción de un segmento TCP que transporte la solicitud original de establecimiento de conexión) serán dirigidos a dos sockets distintos.  ● Cuando un proceso cliente se quiere comunicar con un servidor por un socket, este debe enviar una solicitud en un puerto de destino y bits de establecimiento de conexión en la cabecera. Cuando el SO del host servidor recibe el segmento de entrada de solicitud de conexión con ese puerto de destino, localiza el proceso de servidor que está esperando una conexión en dicho puerto 6789. El proceso de servidor crea un nuevo puerto. Además, la capa de transporte toma nota de los cuatro valores contenidos en el segmento de solicitud de conexión (IPO, PO, IPD/su propia IP, PD). El socket de conexión recién creado queda identificado por estos cuatro valores, así, todos los segmentos que lleguen después con estos valores serán enviados a este socket. Establecida la conexión, el cliente y el servidor podrán enviarse datos.  ● Cabecera de 20 bytes (es variable). Más del doble que UDP. |

**c. Orientado a la conexión.**

|  |  |
| --- | --- |
| UDP | TCP |
| ● Protocolo sin conexión. No hay una fase de establecimiento de la conexión entre las entidades de la capa de transporte emisora y receptora previa al envío del segmento.  ● Inicia la transmisión sin formalidades preliminares, no añade ningún retarde por establecimiento de una conexión.  ● No mantiene información del estado de la conexión.  ● Suele soportar más clientes activos cuando la aplicación se ejecuta sobre UDP. | ● Proceso de establecimiento de la conexión en tres fases antes de la transferencia de datos.  ● Mantiene información acerca del estado de la conexión, la cual reside completamente en los sistemas terminales y no en los elementos intermedios de la red (routers y switches).  ● Servicio full-dupex: si existe una conexión entre el proceso A que se ejecuta en un host y el proceso B que se ejecuta en otro host, entonces los datos de la capa de aplicación pueden fluir desde el proceso A al proceso B en el mismo instante que los datos de la capa de aplicación fluyen del proceso B al proceso A.  ● Conexión (casi siempre) punto a punto, es decir, entre un único emisor y un único receptor. La “multidifusión” (transferencia de datos desde un emisor a muchos receptores en una operación) no es posible.  ● Handshaking (intercambio de mensajes de control). Inicializa ambos procesos antes del intercambio de datos.  ● Cada lado de la conexión tiene su propio buffer de emisión y su propio buffer de recepción.  ● Consta de buffers, variables y un socket de conexión a un proceso en un host, y otro conjunto de buffers, variables y un socket de conexión a un proceso en otro host. |

**d. Controles de congestión.**

|  |  |
| --- | --- |
| UDP | TCP |
| ● No posee control de congestión | ● Regula el flujo del emisor TCP de la capa de transporte cuando uno o más de los enlaces entre los hosts de origen y de destino están excesivamente congestionados.  ● Control de flujo para sus aplicaciones para eliminar la posibilidad de que el emisor desborde el buffer del receptor🡪Servicio de adaptación de velocidades (adapta la velocidad a la que el emisor transmite con la de la que la aplicación receptora lee).  ● El emisor dispone (y determina el tamaño) de una ventana de recepción. Proporciona al emisor una idea de cuánto espacio libre hay en el buffer receptor. Debido a que la conexión es full dúplex, cada proceso dispone de una ventana de recepción diferente. |

**e. Utilización de puertos.**

|  |  |
| --- | --- |
| UDP | TCP |
| ● Socket definido por el número de puerto destino y la dirección IP del destino.  ● Muchos clientes pueden enviar datos por un mismo socket. | ● Socket definido para cada extremo de la conexión por los cuatro campos (IPOrigen, PuertoOrigen, IPDestino, PuertoDestino). Esto quiere decir que  por cada socket pueden intercambiar datos únicamente dos procesos (conexión  punto a punto).  ● Debido a que la conexión solo puede ser punto a punto (porque identifica por esos 4 campos, no puede usar multicast). |

**REVISAR**

**Los sockets pueden verse como las puertas de los procesos.**

Tabla resumen:

|  |  |  |
| --- | --- | --- |
|  | **TCP** | **UDP** |
| **Confiabilidad** | Alta | Baja |
| **Multiplexación** | Soportada | Soportada |
| **Orientado a la conexión** | Sí | No |
| **Controles de congestión** | Sí | No |
| **Utilización de puertos** | Independiente a TCP, y se usa para multiplexación | Independiente a UDP, y se usa para multiplexación |

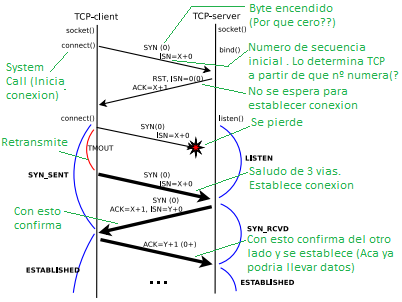
**5. La PDU de la capa de transporte es el segmento. Sin embargo, en algunos contextos suele utilizarse el**

**término datagrama. Indique cuando.**

PDU: Unidad de datos de protocolo.

Se llama datagrama cuando usamos el protocolo UDP (User Datagram Protocol), que técnicamente sigue siendo un segmento, pero al tener tan poca información se le dice Datagrama. También con SCTP.

**6. Describa el saludo de tres vías de TCP. ¿Se utiliza algo similar en UDP?**

De la teoría:

TCP Establecimiento de Conexion ´

* Flags: SYN (Synchronize), ACK (Acknowledge) y RST (Reset).
* En el 3 segmento se puede enviar info.
* el ISN (Initial Sequence Number), se utiliza un contador que se incrementa cada 4 mseg.
* RST si no hay proceso en estado LISTEN.
* Open Pasivo (servidor) y Activo (cliente).
* Open simultaneo

De la practica larga:

Cuando dos hosts se comunican mediante TCP, se establece una conexión antes de que puedan intercambiarse los datos. Luego de que se completa la comunicación, se cierran las sesiones y la conexión finaliza. Los mecanismos de conexión y sesión habilitan la función de confiabilidad del TCP. Para establecer la conexión los hosts realizan un protocolo de enlace de tres vías. Los bits de control en el encabezado TCP indican el progreso y estado de la

conexión.

Aclaración: SYN se utiliza solo en el saludo de 3 vías. ACK se utiliza siempre.

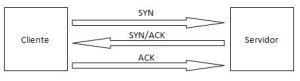
El cliente primero envía un segmento TCP especial; el servidor responde con un segundo segmento TCP especial y, por último, el cliente responde de nuevo con tercer segmento especial. Los dos primeros segmentos no transportan ninguna carga útil; es decir, no transportan datos de la capa de aplicación; el tercero de estos segmentos es el que puede llevar la carga útil. Puesto que los tres segmentos son intercambiados entre dos hosts, este procedimiento de establecimiento de la conexión suele denominarse acuerdo en tres fases.

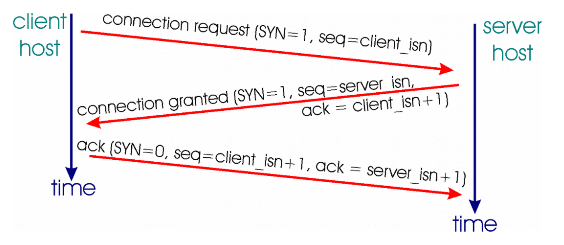
Paso 1: Un cliente TCP comienza el enlace de tres vías enviando un segmento con el señalizador de control SYN (Sincronizar números de secuencia) establecido, indicando un valor inicial en el campo de número de secuencia del encabezado. Este valor inicial para el número de secuencia, conocido como número de secuencia inicial (ISN), se elige de manera aleatoria y se utiliza para comenzar a rastrear el flujo de datos desde el cliente al servidor para esta sesión.

Paso 2: El servidor TCP necesita reconocer la recepción del segmento SYN del cliente para establecer la sesión de cliente a servidor. Para hacerlo, el servidor envía un segmento al cliente con el señalizador ACK establecido indicando que el número de acuse de recibo es significativo. Con este señalizador establecido en el segmento, el cliente interpreta estocomo acuse de recibo de que el servidor ha recibido el SYN del cliente TCP. El servidor manda su propio SYN.

Paso 3: Por último, el cliente TCP responde con un segmento que contiene un ACK que actúa como respuesta al SYN de TCP enviado por el servidor. Pueden no existir datos de usuario en este segmento. El valor del campo número de acuse de recibo contiene uno más que el número de secuencia inicial (es decir, ACK = SEQ\_SV + 1) recibido del servidor. Una vez establecidas ambas sesiones entre el cliente y el servidor, todos los segmentos adicionales que se intercambien en la comunicación tendrán establecido el señalizador ACK.

Una vez completados estos 3 pasos, los hosts cliente y servidor pueden enviarse segmentos que contengan datos el uno al otro.





ACK o “acknowledge” (1 bit): Si está activo entonces el campo con el número de acuse de recibo es válido (si no, es ignorado). Es un mensaje que se envía para confirmar que un mensaje o un conjunto de mensajes han llegado.

SYN o “synchronize” (1 bit): Activa/desactiva la sincronización de los números de secuencia. Se usa para sincronizar los números de secuencia en tres tipos de segmentos: petición de conexión, confirmación de conexión (con ACK activo) y la recepción de la confirmación (con ACK activo).

RST o “reset” (1 bit): Si llega a 1, termina la conexión sin esperar respuesta. Es un bit que se encuentra en el campo del código en el protocolo TCP, y se utiliza para reiniciar la conexión. Un ejemplo práctico de utilización es el que realiza un servidor cuando le llega un paquete a un puerto no válido: este responde con el RST activado.

Otra practica:

La secuencia es:

* El cliente envía un segmento de solicitud al servidor (SYN = true; ACK = false) y espera un segmento de respuesta. El segmento de solicitud tiene Número de secuencia inicial (ISN), que es un número aleatorio de 32 bits, generado por el cliente.
* Luego de enviarse, queda incrementado en 1, porque el hecho de enviarlo “consume” una unidad
* Se almacena en su campo respectivo
* Flag SYN en 1: Le indica al servidor que este segmento contiene el ISN usado por el cliente, y se lo está enviando para sincronizarlos
* Longitud máxima de segmento: Cantidad máxima de bytes que el cliente puede enviar y recibir. Se almacena en el campo Options
* Longitud de la ventana de recepción: Almacena la cantidad máxima de bytes que el cliente puede recibir sin enterarse explícitamente. Se almacena en su campo respectivo, y puede usar Options para representar un número mayor
* El servidor lo recibe y le responde con el segmento de respuesta (SYN = true; ACK = true), que contiene: Número de secuencia inicial (ISN), que ss un número aleatorio de 32 bits, generado por el servidor. Luego de enviarse, queda incrementado en 1, porque el hecho de enviarlo “consume” una unidad. **VER ESTO POR QUE EN EL OTRO LO INCREMENTA EN EL TERCERO**
* Se almacena en su campo respectivo
* Flag SYN en 1: Le indica al cliente que este segmento contiene el ISN usado por el servidor, y se lo está enviando para sincronizarlos
* Longitud máxima de segmento: Cantidad máxima de bytes que el servidor puede enviar y recibir. Se almacena en el campo Options
* Longitud de la ventana de recepción:
* Almacena la cantidad máxima de bytes que el servidor puede recibir sin enterarse explícitamente
* Se almacena en su campo respectivo, y puede usar Options para representar un número mayor
* Número de recibo del segmento: Almacena el número de segmento, del siguiente byte que el receptor espera recibir. El valor que le pone el servidor, es el recibido por el cliente, incrementado en 1. Se almacena en su campo respectivo
* Flag ACK en 1: Le indica al servidor que el número de recibo de este segmento es válido
* El cliente lo recibe y le envía un segmento de ACK puro (SYN = false; ACK = true), o también puede enviar los datos directamente (se sobreentiende)
* La conexión Full Duplex queda establecida.

En UDP no se usa, ya que es un protocolo sin conexión previa

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

El ISN es un número de 32 bits (0 - 4.294.967.265), generado aleatoriamente por el emisor, que se asigna para cada nueva conexión TCP (Al inicio de un saludo de tres vías). Es único para cada conexión y cada dispositivo este número se le asigna al ACK, a partir de ahí el ACK va a ir aumentando.

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

MSS, o tamaño máximo de segmento, es la cantidad máxima de bytes o carga útil de datos más grande que un dispositivo aceptará de una conexión de red, es decir, lo máximo que se puede enviar o recibir.

Si se recibe una cantidad mayor, se produce fragmentación, lo cual suma overhead.

Se envía dentro de Options, ya que sólo se negocia una vez por conexión.

Según la RFC-879, el valor mínimo recomendado es de 536 bytes, usando una Unidad de Transmisión Máxima (MTU) de 576 bytes, y 40 bytes de cabeceras TCP/IP

MTU - (encabezado TCP + encabezado IP) = MSS

El MSS se negocia en el saludo de 3 vias.

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

**a. Para listar las comunicaciones TCP establecidas.**

ss –t

ss --tcp –processes

**b. Para listar las comunicaciones UDP establecidas.**

ss –u

ss --udp –processes

**c. Obtener sólo los servicios TCP que están esperando comunicaciones**

ss -t –l

ss --tcp --processes –listening

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

ss -u –l

ss --udp --processes –listening

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

ss -t -l -p

ss -u -l -p

El parámetro -p: Show process using socket. Para que muestre la información de interés es necesario ser sudo.

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

Inciso A: netstat –t

Inciso B: netstat -u

Inciso C: netstat -t -l

Inciso D: netstat -u -l

Inciso E: netstat -t -l -p

netstat -u -l –p

netstat --all --tcp

netstat --all --udp

netstat --listening --tcp

netstat --listening --udp

**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, que dicho puerto no está en estado LISTEN)?**

No se podrá efectuar la conexión, ya que debe haber un proceso (con un socket) escuchando para aceptar el segmento SYN, dando comienzo al saludo de tres vías.

El host enviará al origen un segmento especial de reinicio. Este segmento TCP tiene el bit indicador RST

puesto a 1. Por tanto, cuando un host envía un segmento de reinicio, le está diciendo al emisor “No tengo un socket para ese segmento. Por favor, no reenvies el segmento.”

Agrego caso UDP: Cuando un host recibe un paquete UDP en el que el número de puerto de destino no se

corresponde con un socket UDP activo, el host envía un datagrama ICMP especial (puerto inalcanzable).

Hping3 es una aplicación de terminal para Linux que nos va a permitir analizar y ensamblar fácilmente paquetes TCP/IP. A diferencia de un Ping convencional que se utiliza para enviar paquetes ICMP, esta aplicación permite el envío de paquetes TCP, UDP y RAW-IP.

Los campos de información que se maneja con el comando hping3 son:

● id es el campo ip id.

● win representa el tamaño de la ventana TCP. Indica la capacidad que tiene la aplicación del otro lado para recibir datos. Por ejemplo, win=20 indica que la aplicación puede recibir 20 datos. Al momento de enviarle datos a esta aplicación, el valor de win se irá decrementando (porque está recibiendo, y debe procesarlos). A medida que los procesa, el valor de win se incrementará y puede alcanzar su límite máximo. Esto se da porque la aplicación cuenta con un buffer con capacidad limitada.

● rtt es el round trip time en milisegundos.

● If the reply contains DF the IP header has the don't fragment bit set.

● len is the size, in bytes, of the data captured from the data link layer excluding the

data link header size.

● flags are the TCP flags, R for RESET, S for SYN, A for ACK, F for FIN, P for PUSH,

U for URGENT, X for not standard 0x40, Y for not standard 0x80.

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

**activado.**

*sudo hping3 localhost --destport 22 –syn //o 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=7.7 ms

len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=**SA** seq=1 win=65495 rtt=3.5 ms

len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=22 flags=**SA** seq=2 win=65495 rtt=3.2 ms

^C

--- localhost hping statistic ---

3 packets transmitted, 3 packets received, 0% packet loss

round-trip min/avg/max = 3.2/4.8/7.7 ms

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

**activado.**

*sudo hping3 localhost --destport 40 –syn //o 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=3.9 ms

len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=**RA** seq=1 win=0 rtt=3.7 ms

len=40 ip=127.0.0.1 ttl=64 DF id=0 sport=40 flags=**RA** seq=2 win=0 rtt=7.4 ms

^C

--- localhost hping statistic ---

3 packets transmitted, 3 packets received, 0% packet loss

round-trip min/avg/max = 3.7/5.0/7.4 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).**

En el primer caso, el puerto está abierto. El servidor responde con los flags **SA** (SYN y ACK) activados, lo que significa que es un segmento de respuesta, y que ese puerto esta abierto → se puede establecer la conexión

En el segundo caso, el puerto está cerrado. El servidor responde con los flags **RA** (RST y ACK) activados, lo que significa que es un segmento de cierre de conexión forzada, por que ese puerto esta cerrado→ no se establece la conexión porque no hay nadie escuchando

Len: En el primer caso el tamaño en bytes es 44, y en el segundo 40.

Sport: la diferencia se dá por la solitud en el comando. En el primer caso se pide el puerto 22 y en el segundo el 40.

Id: en el primer caso, siempre es 0. En el segundo caso va variando.

Win: en el primer caso, nunca es 0. En el segundo si.

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

Cuando un host recibe un paquete UDP en el que el número de puerto de destino no se corresponde con un socket UDP activo, el host envía un datagrama ICMP especial (port unreachable, o puerto inalcanzable).

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

*sudo hping3 localhost --destport 5353 –udp //o sudo hping3 localhost -p 68 -2 (2 es para ponerlo en udp)*

HPING localhost (lo 127.0.0.1): udp mode set, 28 headers + 0 data bytes

^C

--- localhost hping statistic ---

3 packets transmitted, 0 packets received, 100% packet loss

round-trip min/avg/max = 0.0/0.0/0.0 ms

Nota: No hay respuesta.

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

*sudo hping3 localhost --destport 40 –udp//o sudo hping3 localhost -p 40 -2*

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=2330 seq=0

**ICMP Port Unreachable from ip=127.0.0.1 name=localhost**

status=0 port=2331 seq=1

**ICMP Port Unreachable from ip=127.0.0.1 name=localhost**

status=0 port=2332 seq=2

^C

--- localhost hping statistic ---

3 packets transmitted, 3 packets received, 0% packet loss

round-trip min/avg/max = 3.8/6.6/8.3 ms

Nota: en este caso sí hay respuesta, la cual es: “ ICMP Port Unreachable from ip=127.0.0.1 name=localhost”

**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).**

En el primer caso “hay un servicio escuchando en el puerto”, por lo que se transmitieron todos los paquetes, pero no hay forma de saber si los recibió correctamente o se perdieron en el camino ya que UDP no tiene mensajes de retorno por reconocimiento

En el segundo caso, como no hay un servicio escuchando en el puerto por lo que no se puede alcanzar este y UDP responde informando esto con un paquete ICMP Port Unreachable

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

**Ver** [**https://users.cs.northwestern.edu/~agupta/cs340/project2/TCPIP\_State\_Transition\_Diagram.pdf**](https://users.cs.northwestern.edu/~agupta/cs340/project2/TCPIP_State_Transition_Diagram.pdf)

Literalmente el link de arriba traducido:

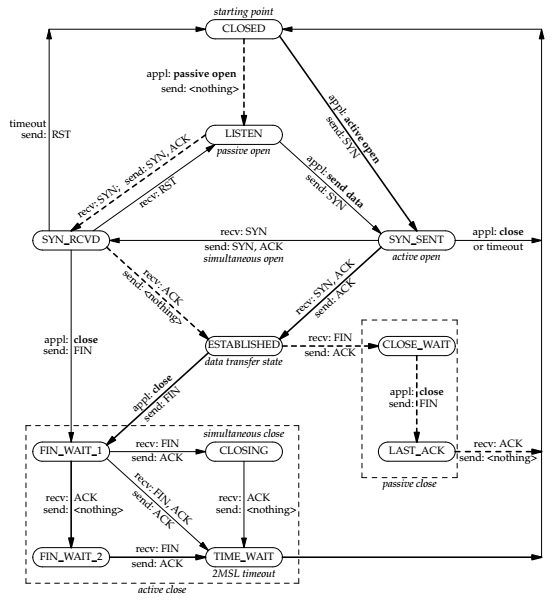
Una conexión progresa a través de una serie de estados durante su vida útil. Los estados son: LISTEN, SYN-SENT, SYNRECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT y el estado ficticio CLOSED. Este es ficticio porque representa el estado cuando no hay TCB, y por lo tanto, no hay conexión. Significados de los estados:

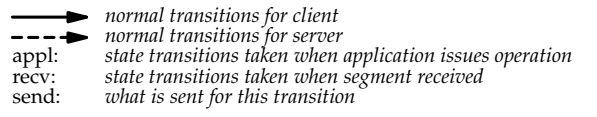
* LISTEN representa la espera de una solicitud de conexión desde cualquier puerto y TCP remoto.
* SYN-SENT representa la espera de una solicitud de conexión coincidente después de haber enviado una solicitud de conexión.
* SYN-RECEIVED representa la espera de un reconocimiento de solicitud de conexión de confirmación después de tener ambos recibido y enviado una solicitud de conexión.
* ESTABLISHED representa una conexión abierta, los datos recibidos pueden ser entregados al usuario. el estado normal para la fase de transferencia de datos de la conexión.
* FIN-WAIT-1 representa la espera de una solicitud de terminación de conexión del TCP remoto, o un acuse de recibo de la solicitud de terminación de conexión enviada previamente.
* FIN-WAIT-2 representa la espera de una solicitud de terminación de conexión del TCP remoto.
* CLOSE-WAIT representa la espera de una solicitud de terminación de conexión del usuario local.
* El CIERRE representa la espera de una confirmación de solicitud de terminación de conexión del control remoto TCP.
* LAST-ACK representa la espera de un reconocimiento de la solicitud de finalización de la conexión previamente enviado al TCP remoto (que incluye un reconocimiento de su terminación de conexión

pedido).

* TIME-WAIT representa esperar a que pase suficiente tiempo para asegurarse de que el TCP remoto recibió el acuse de recibo de su solicitud de terminación de conexión.
* CERRADO representa ningún estado de conexión en absoluto.

Una conexión TCP progresa de un estado a otro en respuesta a eventos. Los eventos son las llamadas del usuario, OPEN, SEND, RECEIVE, CLOSE, ABORT y STATUS; los segmentos entrantes, particularmente aquellos que contienen las banderas SYN, ACK, RST y FIN; y tiempos de espera.





Traducción de las flechitas: transiciones normales para el cliente

transiciones normales para el servidor

appl: transiciones de estado tomadas cuando la aplicación emite una operación

recv: transiciones de estado tomadas cuando se recibió el segmento

enviar: lo que se envía para esta transición

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

**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’.**

root@Servidor:/tmp/pycore.34739/Servidor.conf# ncat --listen --keep-open 8001

root@Servidor:/tmp/pycore.34739/Servidor.conf# ss -nat

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

root@Cliente:/tmp/pycore.34739/Cliente.conf# ss -nat

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

\*-\*-\*-\*-

NOTAS:

- ss is another utility to investigate sockets

- ss is used to dump socket statistics. It allows showing information similar to

netstat. It can display more TCP and state informations than other tools.- El comando watch ejecuta un comando repetidas veces (en este caso “n1” = “cada

un segundo”). Haciendo “watch -n1 ’ss -nat’” se está ejecutando el comando ss para

verificar el estado de las conexiones.

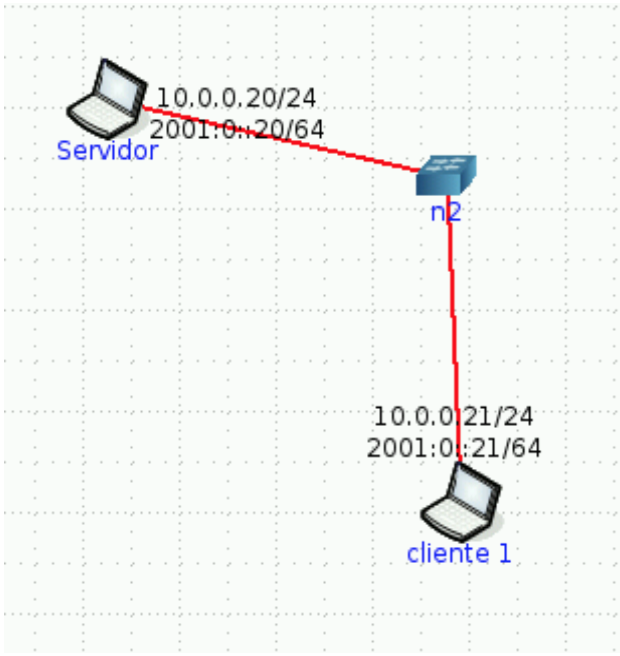
- Una vez graficada la topologìa le di play al ‘run session’ y al hacer click derecho

sobre cada host optè por la opción view shell para mandarle el comando

especificado tanto al cliente como al server.

**b. En Servidor, utilice la herramienta ncat para levantar un servicio que escuche en el puerto 8001/TCP.**

**Utilice la opcion -k para que el servicio sea persistente. Verifique el estado de las conexiones.**



root@Cliente:/tmp/pycore.34739/Cliente.conf# ncat 10.0.0.10 8001

root@Servidor:/tmp/pycore.34739/Servidor.conf# ss -nat

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 [::]:\*

\*-\*-\*-\*-\*

Se utilizó el comando ncat -l -k 8001 (en una nueva terminal).

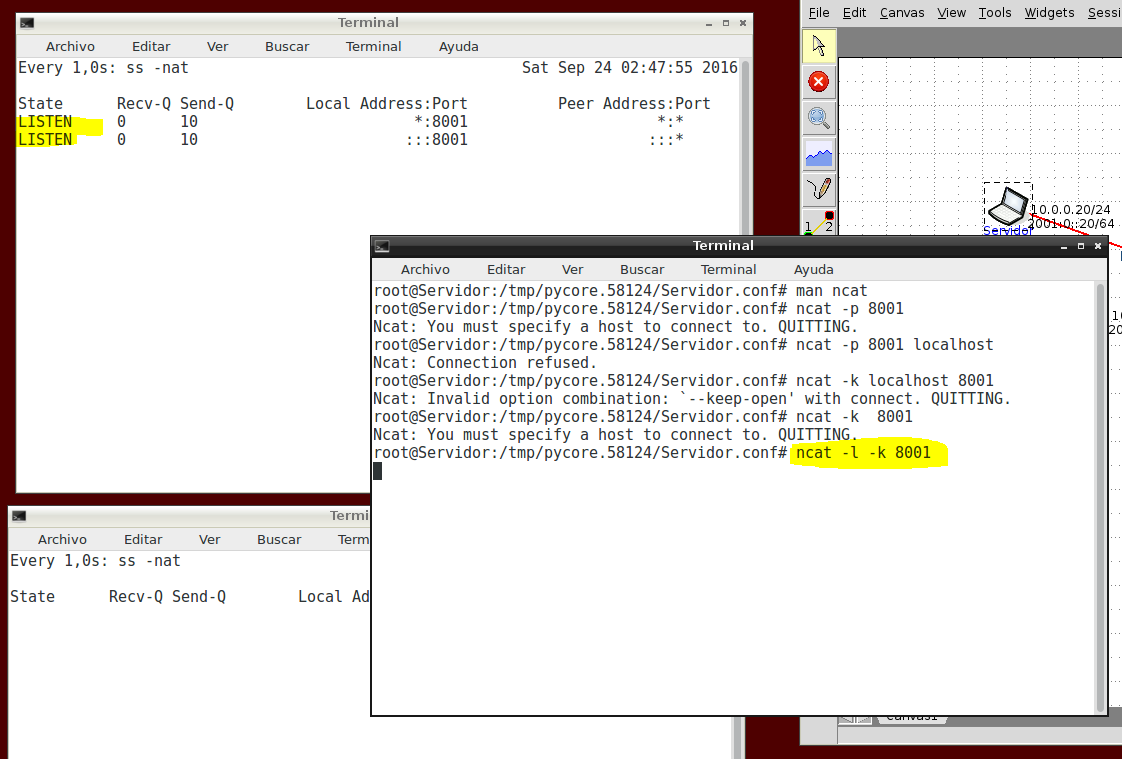
→ -l es para LISTEN

→ -k es para que las conexiones sean persistentes

→ Puerto 8001

→ Protocolo TCP (por defecto)

Ahora el servidor aparece escuchando en el puerto 8001



**c. Desde CLIENTE1 conectarse a dicho servicio utilizando también la herramienta ncat. Inspeccione**

**el estado de las conexiones.**

root@Cliente:/tmp/pycore.34739/Cliente.conf# ncat 10.0.0.10 8001

root@Servidor:/tmp/pycore.34739/Servidor.conf# ss -nat

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

LISTEN 0 10 0.0.0.0:8001 0.0.0.0:\*

ESTAB 0 0 10.0.0.10:8001 10.0.0.20:46858

LISTEN 0 10 [::]:8001 [::]:\*

root@Cliente:/tmp/pycore.34739/Servidor.conf# ss -nat

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

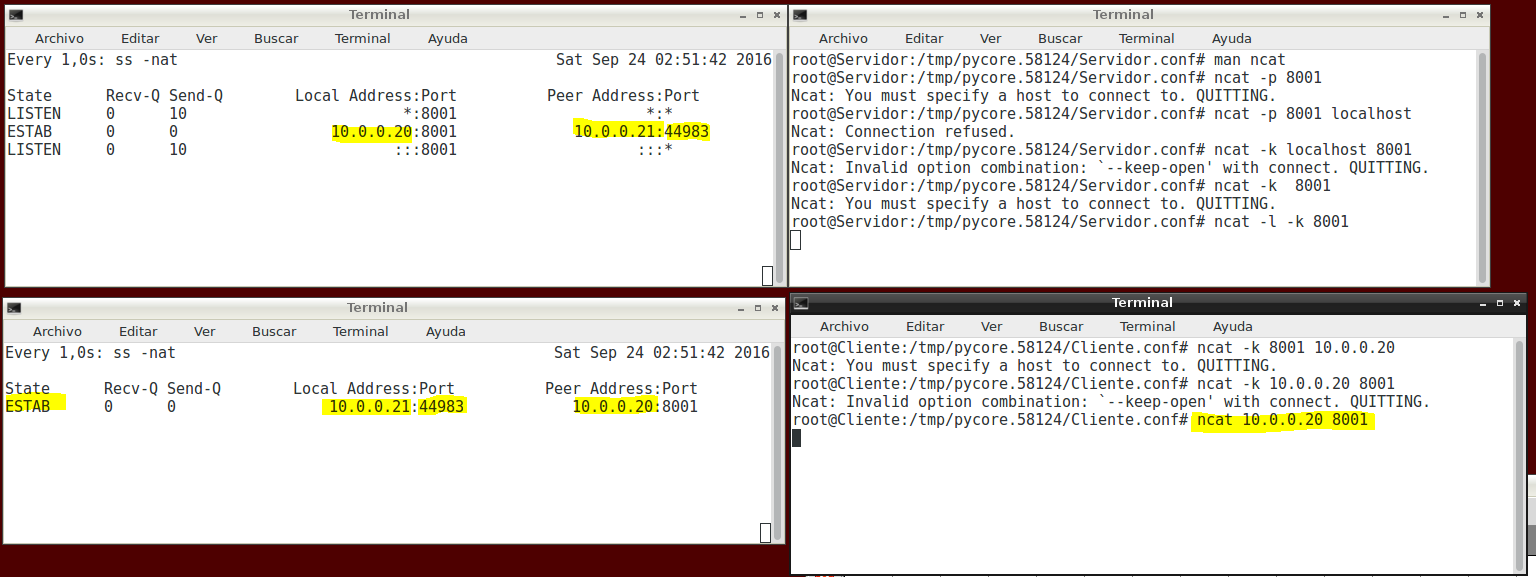
ESTAB 0 0 10.0.0.20:46858 10.0.0.10:8001

\*-\*-\*-\*-\*-\*-

Se utilizó de nuevo el comando ncat pero esta vez indicando el host destino y el puerto:

Ncat 10.0.0.20 8001

NOTA: ncat - Concatenate and redirect sockets



**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?**

root@Cliente:/tmp/pycore.34739/Cliente.conf# ncat 10.0.0.10 8001

root@Servidor:/tmp/pycore.34739/Servidor.conf# ss -nat

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

LISTEN 0 10 0.0.0.0:8001 0.0.0.0:\*

ESTAB 0 0 10.0.0.10:8001 10.0.0.20:46858

ESTAB 0 0 10.0.0.10:8001 10.0.0.20:37716

LISTEN 0 10 [::]:8001 [::]:\*

root@Cliente:/tmp/pycore.34739/Cliente.conf# ss -nat

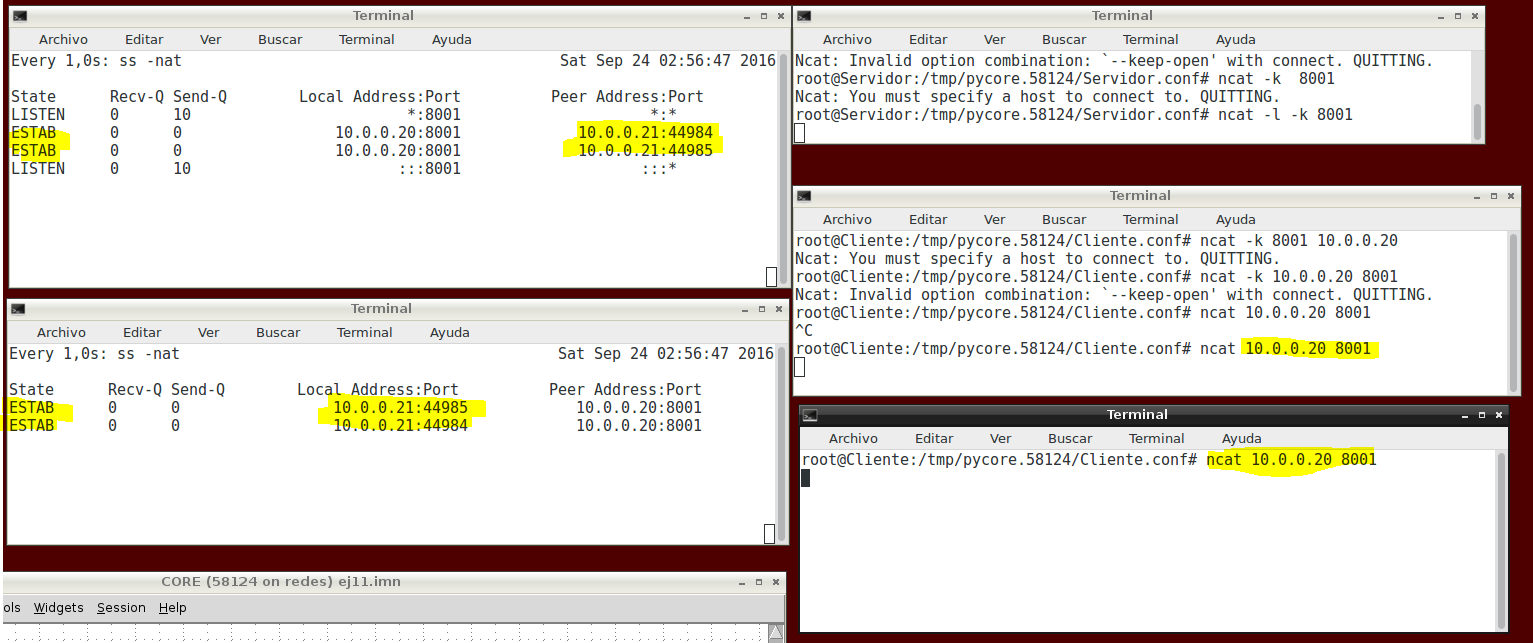
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process

ESTAB 0 0 10.0.0.20:46858 10.0.0.10:8001

ESTAB 0 0 10.0.0.20:37716 10.0.0.10:8001

-\*-\*-\*-\*-

Cambia el puerto origen, así es como se identifican.



**e. En base a lo observado en el item 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?**

Es posible, porque en los headers de los segmentos transferidos en cada conexión, se guarda el puerto origen y destino (que entre ambas forman un identificador unívoco)

\*-\*-\*-\*-\*

Sí, es posible iniciar más de una conexión desde el cliente al servidor en el mismo puerto destino, ya que lo que termina identificando a cada una es el puerto origen, el cual es distinto. Es por esto que no se mezclan los datos de una con otra, ya que en la cabecera de los segmentos figuran ips origen destino y puertos origen destino.Aclaración: Anteriormente vimos que en UDP el campo puerto origen es opcional.

Esto NO es algo normal. Si no tenemos el puerto destino, no podemos enviarle información, y por ende, siempre se configura.

**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.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Paquete** | **Sentido** | **URG** | **ACK** | **PSH** | **RST** | **SYN** | **FIN** |
| 1 | C → S | false | false | false | false | true | false |
| 2 | S → C | false | true | false | false | true | false |
| 3 | C → S | false | true | false | false | false | false |

El estado de las conexiones es similar a 13.C)

-\*-\*-\*-\*--\*-

Apreté control C en una de las terminales del cliente donde había hecho la conexión y esa

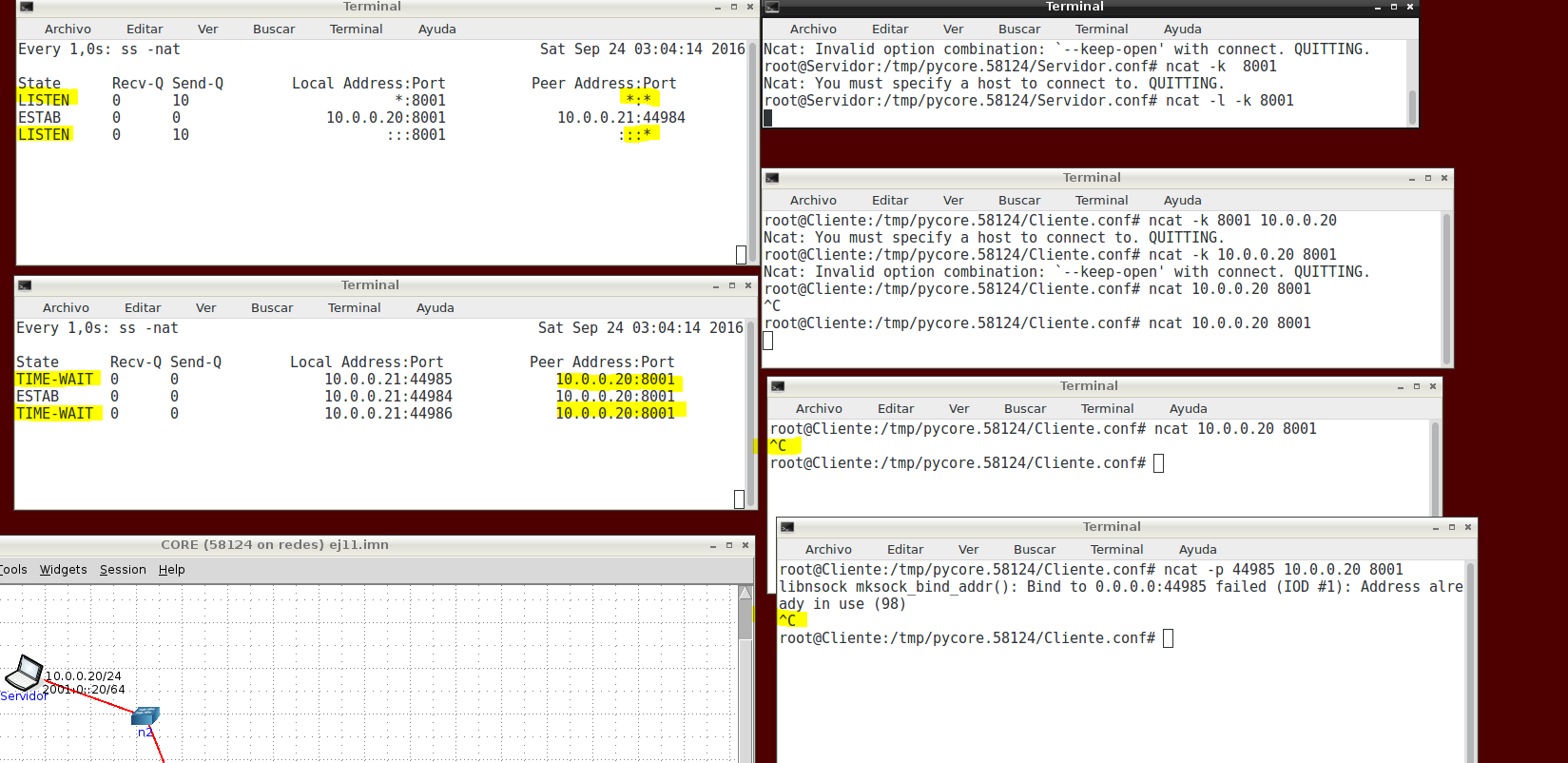
fue la forma de cerrarla.

El servidor dejo de mostrar estado de ESTABLISHED en ese puerto origen y el cliente

mostró durante unos minutos el estado TIME-WAIT en la conexión y luego nada.

Si el cliente cierra la conexión, desde el lado del cliente se puede ver el estado TIME-WAIT

por unos segundos hasta que desaparece de la lista.



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

**equipos.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Paquete** | **Sentido** | **URG** | **ACK** | **PSH** | **RST** | **SYN** | **FIN** |
| 1 | S → C | false | true | false | false | false | true |
| 2 | S → C | false | true | false | true | false | false |
| 3 | C → S | false | true | false | false | false | false |

Estado de las conexiones:

root@Servidor:/tmp/pycore.34739/Servidor.conf# ss -nat

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

root@Cliente:/tmp/pycore.34739/Servidor.conf# ss -nat

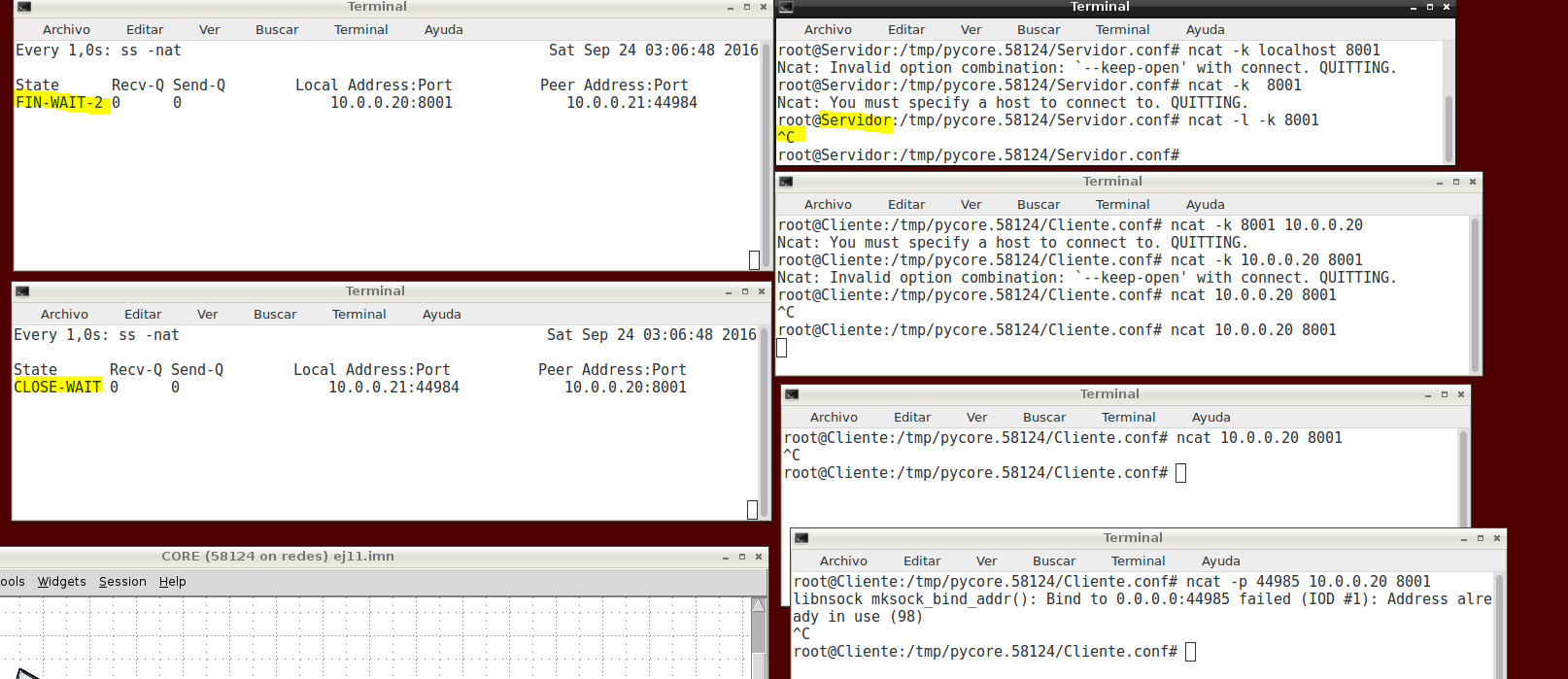
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process

CLOSE-WAIT 0 0 10.0.0.20:43456 10.0.0.10:8001

CLOSE-WAIT 0 0 10.0.0.20:41442 10.0.0.10:8001

CLOSE-WAIT 0 0 10.0.0.20:46724 10.0.0.10:8001

--\*-\*-\*-\*-\*-\*-\*-



Si la conexión se cierra primero desde el servidor, entonces se queda en estado FIN-WAIT y

hasta que no se cierra desde el cliente no se deja de listar; desde el cliente se puede ver el

estado CLOSE-WAIT.

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Paquete** | **Sentido** | **URG** | **ACK** | **PSH** | **RST** | **SYN** | **FIN** |
| 1 | C → S | false | true | false | false | false | true |
| 2 | S → C | false | true | false | false | false | true |
| 3 | C → S | false | true | false | false | false | false |

El estado de la conexión cliente es similar a 13.A)

El de servidor, no:

root@Servidor:/tmp/pycore.34739/Servidor.conf# ss -nat

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

LISTEN 0 128 0.0.0.0:22 0.0.0.0:\*

LISTEN 0 10 0.0.0.0:8001 0.0.0.0:\*

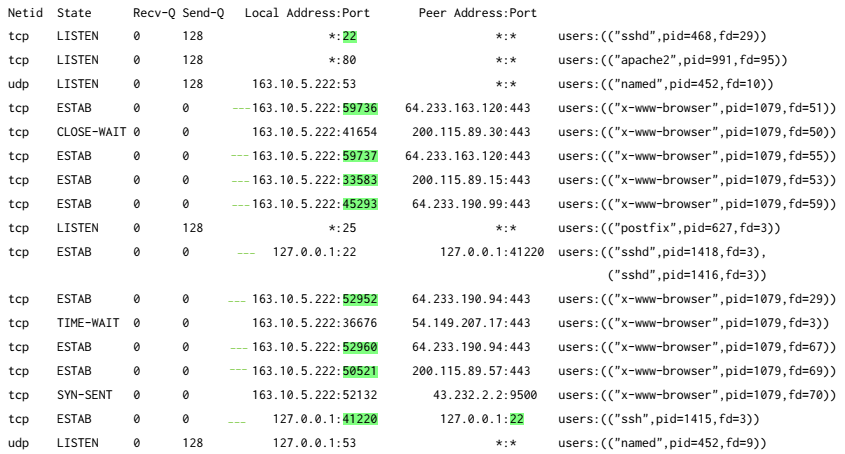
CLOSE-WAIT 1 0 10.0.0.10:8001 10.0.0.20:60136

LISTEN 0 128 [::]:22 [::]:\*

LISTEN 0 10 [::]:8001 [::]:\*

Uno puede quedar en TIME-WAIT, porque espera recibir la confirmación de cierre del otro extremo (segmento que aún no llegó), o por si hay segmentos aún viajando (si justo se abre una nueva conexion en ese puerto, para que no se confundan los paquetes)

**14. Dada la siguiente salida del comando ss, responda:**



**a. ¿Cuántas conexiones hay establecidas?**

8. Una no se cuenta porque es la misma conexión -> Clase jueves

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

5 o 6?

3 TCP y 1 UDP -> Clase jueves. Por que???? Si son los que están en listen :/

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

No, el cliente y el servidor no residen en la misma máquina, ya que el local address y el

peer address son diferentes (Distintas IPs)

443 se conecta a un servidor remoto

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

Residen en la misma, ya que tienen misma IP (127.0.0.1, o localhost)

**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.**

|  |  |  |
| --- | --- | --- |
| **Proceso** | **Puerto local** | **Rol** |
| x-www-browser | 59736 | Cliente |
| x-www-browser | 59737 | Cliente |
| x-www-browser | 33583 | Cliente |
| x-www-browser | 45293 | Cliente |
| sshd | 22 | Servidor |
| x-www-browser | 52952 | Cliente |
| x-www-browser | 52960 | Cliente |
| x-www-browser | 50521 | Cliente |
| ssh | 41220 | Cliente |

**Apache es servidor y hay uno de mail(?**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Proceso** | **Puerto local** | **Estado** | **Cerrado por** |
| x-www-browser | 41654 | CLOSE-WAIT (local) | Cliente |
| x-www-browser | 36676 | TIME-WAIT (remoto) | Servidor |

**g. ¿Cuántas conexiones están aún pendientes por establecerse?**

1, el syn-send -> Clase jueves

**15. 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?**

El primer mensaje (cliente a servidor) llega correctamente, el servidor lo procesa

La respuesta del servidor es la que se pierden en el camino o está en tránsito. -> Clase jueves

Cuando llegue pasa a establecida, si no intenta un par de veces hasta devolver que no se puede conectar.

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

El protocolo de la capa de aplicación y la de transporte, te lo indica el número de puerto al

que se están conectando en este caso es el 110: POP3

De aplicación: POP

De transporte: TCP (porque existe el saludo de tres vías, con sus estados, y además la primera columna lo dice explícitamente)

**c. ¿Qué flags tendría seteado el segmento perdido?**

SYN y ACK en 1.