**Redes y comunicaciones 2009 - Práctica 3**

**Capa De Transporte**

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

La función de la capa de **Transporte** es proveer de una comunicación lógica entre *procesos de aplicación corriendo en diferentes hosts*, permitiendo el envío de *segmentos* de datos como si estuviesen conectados directamente, sin intermediarios ni preocupaciones por la infraestructura de envío de mensajes. Se implementa en los hosts terminales, y se ubica por sobre la capa de **Red** (que se ocupa de la comunicación lógica, pero entre hosts remotos en vez de entre procesos).  
  
**2. Describa la estructura del segmento TCP y UDP**   
  
 **Segmento TCP:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| + | [Bits](file:///C:\Users\Nicolas\wiki\Bit) 0 - 3 | 4 - 7 | 8 - 15 | 16 - 31 |
| 0 | Puerto Origen | | | Puerto Destino |
| 32 | Número de Secuencia | | | |
| 64 | Número de Acuse de Recibo (ACK) | | | |
| 96 | longitud cabecera TCP | Reservado | Flags | Ventana de recepcion |
| 128 | Suma de Verificación (Checksum) | | | Puntero Urgente |
| 160 | Opciones + Relleno (opcional) | | | |
| 224 | Datos | | | |

#port fuente, #port destino, # de secuencia de los datos enviados, # de confirmación de la última secuencia recibida, tamaño del header, un campo actualmente no utilizado, flags, tamaño de la ventana en el receptor, checksum, puntero a la posición donde comienzan los datos urgentes, campo de opciones de longitud variable y los datos en sí que se transmiten -opcionales, por si sólo se envía información de control-.

* TCP percibe los datos como flujos de bytes no estructurados, pero ordenados. La utilización de TCP de los números de secuencia refleja esta visión, ya que los *números de secuencia lo son sobre el flujo de bytes*, y *no sobre la serie de segmentos transmitidos*.  
  El **numero de secuencia de un segmento** es: el **numero del primer byte en el segmento dentro del flujo de bytes**.  
  Los numeros de reconocimiento: como TCP es full duplex, es decir, el host A puede estar recibiendo datos desde el host B mientras esta mandando datos al host B (dentro de la misma conexión TCP). Cada uno de los segmentos que llegan desde el host B tienen un numero de secuencia para los datos que fluyen de B a A. **El numero de reconocimiento que el host A pone en su segmento es el numero de secuencia del siguiente byte esperado por el host A desde el host B***.*Ambos numeros son de 32 bits.
* El campo de **longitud de cabecera**, de 4 bits, especifica la longitud de la cabecera TCP en palabras de 32 bits. La cabecera TCP puede ser variable debido al campo *opciones*, por esta razón se precisa indicar el tamaño en un campo especial. Normalmente el campo de opciones se encuentra vacio, de forma que la longitud de una cabecera tipica es de 20 bytes.
* El campo flags incluye 6 flags dintintos: URGent data -indica que se envían datos urgentes-, ACKnowlodge -confirmación-, PUSH -entregar datos al proceso en forma inmediata-, RST -negación de pedido de conexión-, SYN -utilizado para establecimiento de la conexión en el **three-way handshake**- y FIN -utilizado para el cierre de conexión-. En la práctica, URG y PUSH no se utilizan.
* El campo **ventana de recepcion**, de 16 bits, se refiere a la cantidad de bytes que el receptor está dispuesto a aceptar, permitiendo el control de flujo.
* El checksum se calcula sobre los datos y el encabezado.
* El puntero a los datos urgentes no se utiliza en la práctica.
* El campo opciones, de longitud variable y opcional, que se utiliza cuando el emisor y el receptor nogocian el tamaño maximo del segmento (MMS) o el factor de escala de la ventana utilizado en las redes de alta velocidad. Tambien permite enviar *timestamps.*

* ***Segmento UDP****: #port fuente, #port destino, largo de los datos, checksum y los datos en sí que se transmiten.*

3. ¿Cuál es el objetivo del uso de puertos en el modelo TCP/IP?   
  
El uso de puertos en el modelo TCP/IP permite la **multiplexación** de los paquetes que la capa de transporte recibe de los distintos procesos y pasa a la capa de Red en el emisor, y la **demultiplexación** de los segmentos recibidos desde la capa de Red en paquetes que se envían a los procesos adecuados en el receptor.

Si un terminal tuviese dos procesos haciendo solicitudes a un mismo servidor Web, ambos se comunicarían al port 80 -utilizado por el servidor Web en el host destino-.  
El servidor Web se “forkea”, es decir crea un nuevo proceso (hilo) para atender cada nueva solicitud (en este ejemplo tendria dos procesos) y asi tener el proceso central escuchando el port 80. El port destino permite a la capa de Transporte del receptor entregar los paquetes (demultiplexacion) al servidor Web y no a otro proceso que esté corriendo en ese host (servidor). Del mismo modo, cada proceso del emisor tendrá asociado un port de origen diferente -comúnmente aleatorio-, permitiendo al servidor Web encaminar sus respuestas en dirección contraria a un proceso específico del host (cliente) que realizó las solicitudes.

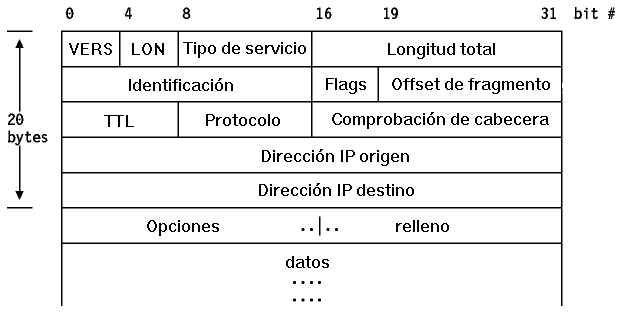
En el caso que dos clientes (distintos, es decir, diferentes host) escojan un mismo port origen, el servidor podrá diferenciar a cada proceso utilizando la IP de origen, en el datagrama de capa de Red. Así, la tripla *(port origen, port destino, IP origen)* permite al host receptor encaminar los datos al socket correspondiente.

**4. Compare TCP y UDP en cuanto a: Confiabilidad , Multiplexación , Orientado a la conexión , Controles de congestión y Utilización de puertos.**

La capa de Transporte ofrece dos protocolos, cada uno con distintos servicios: **TCP** y **UDP.**   
Por estar construídos sobre el único protocolo que ofrece la capa de Red, *IP*, algunos servicios no pueden ser ofrecidos por ninguno de ellos: garantías de retardo ni de ancho de banda -básicamente porque IP utiliza *conmutación de paquetes* y no de *circuitos*-. Sin embargo, otras características no soportadas por *IP*, como la transmisión confiable, sí pueden ser ofrecidas por la capa de Transporte.

|  |  |  |  |
| --- | --- | --- | --- |
| **Característica** | **Definición** | **TCP** | **UDP** |
| **Confiabilidad** | Asegurar la entrega de todos los paquetes enviados, permitiendo mantener en el receptor el orden en que le fueron enviados (utilizando confirmación de recepción, timeout, reenvío de paquetes y números de secuencia) y detectar errores en la recepción (mediante *checksum*). | **SI** (utiliza establecimiento de conexión de 3 pasos) | **NO** (sin conexión, sólo ofrece el *best effort* de IP) (\*) |
| **Multiplexación** | Extender la comunicación host-host que ofrece IP permitiendo diferenciar los procesos en el receptor y en el emisor, de modo que varios procesos en un host se comuniquen con varios procesos en otros hosts, sin confundir los paquetes (utilizando direcciones IP y números de puerto que identifican a los procesos). | **SI** Socket: (*IP Orig, Port Orig, IP Dest, Port Dest*) | **SI** Socket: (*IP Dest, Port Dest*) (\*\*) |
| **Control de Congestión** | Controlar la velocidad de transmisión del emisor para **no sobrecargar la** **red**. | **SI** | **NO** |
| **Control de Flujo** | Controlar la velocidad de transmisión del emisor para **no sobrecargar al receptor** (según el tamaño de su buffer, ventana de recepcion) | **SI** | **NO** |
| **Utilización de puertos** | Los ports destino y origen son precisos tanto en TCP como en UDP por fines de multiplexación/demultiplexación, aunque UDP sólo precisa el port destino para definir un socket (al ser sin conexión se envían segmentos individuales, mientras que TCP divide un mensaje en múltiples segmentos y precisa reconocer el proceso emisor para recomponer el mensaje original) | **Port Origen y Port Destino** | **Port Origen y Port Destino** |
| **Orientado a Conexión** | Si se establece una conexión previa entre los host | **SI** | **NO** |

(\*) Es posible un Transporte Confiable Sobre UDP -TCsU-, pero dependiente de la aplicación -será esta quien determine cómo recuperarse ante fallos-. Del mismo modo que TCP ofrece transmisión confiable sobre IP, una aplicación puede lograr transmisión confiable sobre UDP. Si bien podría utilizarse TCP en estos casos y simplificar la aplicación, el uso de TCsU ofrece a la aplicación una ventaja: UDP no posee control de congestión, de modo que puede transmitir a cualquier tasa de transferencia aunque perjudique el desempeño total de la red -con TCP, se comienza enviando datos a una tasa baja que aumenta gradualmente; por cada segmento no confirmado se baja la tasa por detección de congestión y en conjunto con IP se logra un control de congestión preventivo.   
  
(\*\*) UDP: Segmentos con distinto IP y port de origen pueden ser dirigidos a un mismo socket, dado que sólo lo determina el IP y port destino. El port origen provee una dirección de retorno, por si el receptor desea responder de algún modo al emisor, pero esto dependerá del protocolo de aplicación, no es propio de UDP -TCP en cambio, envía siempre un paquete de acuse de recibo o ACK-.**¿Cuál es el campo del datagrama IP y los valores que se utilizan en este para diferenciar que se transporta TCP o UDP? (Ayuda: buscar en /etc/protocols y contrastarlo con una captura de tráfico)**

Es el campo **PROTOCOLO:** TCP/UDP/ICMP

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

Datagrama → UDP

Un datagrama es una cabecera + datos, tal que la cabecera ofrece la información necesaria para que los datos puedan ser encaminados y alcancen un destino.

Se suele hablar de *Datagrama* en referencia a los segmentos en el protocolo *UDP* -*Protocolo de Datagrama de Usuario*-. Sin embargo, como *datagrama* es el nombre formal de los *PDU* de la capa de *Red* -protocolo IP- es preferible evitar esta denominación.

**6. Describa el saludo de tres vías de TCP.**

Ver ejercicio 15. a. i

**7.¿Qué sucede si llega un segmento TCP a un host que no tiene a ningún proceso esperando en el puerto destino de dicho segmento?**

Si cliente envía SYN y el servidor le niega el servicio -por ejemplo, cliente intenta acceder al puerto #80 y no hay servidor web corriendo-, el servidor responde con RESET

**8. ¿Qué sucede si llega un datagrama UDP a un host que no tiene a ningún proceso esperando en el puerto destino de dicho datagrama?**

Llega un mensaje ICMP de tipo 3 (destination unreachable) y codigo 3 (port unreachable)

**9. Investigue que es muticast**

La tecnología multicast representa un servicio de red en el cual un único flujo de datos, proveniente de una determinada fuente, se puede enviar simultáneamente a diversos receptores interesados. Cabe a la infraestructura de red transportar este flujo de datos, replicándolo cuando sea necesario, para todos los receptores que registren interés en recibir estos datos.

En redes TCP/IP, estos receptores son representados por una dirección de grupo o dirección multicast. Esta dirección de grupo corresponde a una dirección IP que pertenece a la antigua clase D, es decir, en la franja entre 224.0.0.0 y 239.255.255.255. Cada fuente envía paquetes hacia una dirección de grupo (por ejemplo: 233.7.124.1), en el cual estarán asociados diversos receptores. Estos receptores, a su vez se pueden vincular y desvincular en forma dinámica. Cabe a los dispositivos de la red y en particular a los enrutadores, determinar cuáles de sus interfaces poseen receptores interesados en un grupo multicast y cuáles deberán recibir una copia de los paquetes enviados para ese grupo.

El multicast está orientado hacia aplicaciones del tipo "uno para muchos" y "muchos para muchos". En estos casos, presenta claras ventajas cuando se lo compara con los mecanismos de transmisión unicast y broadcast. En unicast, es necesario que la fuente replique varios flujos de datos idénticos con el objeto de transmitirlos a cada uno de los receptores, generando desperdicio de banda. Por otro lado, el sistema broadcast envía los datos a toda la red de forma indiscriminada. Esto también da como resultado el desperdicio de recursos, pues implica el transporte de datos para todas las estaciones de la red, aunque el número de receptores deseosos de que ese contenido sea reducido. Con multicast, la fuente de tránsito envía una única copia de los paquetes hacia una dirección de grupo multicast. La infraestructura de red replica estos paquetes de forma inteligente, encaminando los datos de acuerdo con la topología de receptores interesados en esa información.

Entre las diversas aplicaciones que pueden obtener ganancias con el uso de multicast están: videoconferencia; aprendizaje a distancia; distribución de software, noticias e informaciones de mercado; conciertos al vivo; actualización de bases de datos; juegos distribuidos; procesamiento competidor; simulacros distribuidos etc...

**¿Se podría adaptar para que funcione sobre el otro protocolo de capa de transporte? ¿por qué?**

TCP es orientado a conexión, el establecimiento de la misma de tres vias es uno a uno (punto a punto). Con UDP si es posible multicast.  
  
No es posible establecer comunicaciones multicast sobre TCP, las mismas se realizan sobre UDP debido a que no son comunicaciones punto a punto.  
Con TCP antes de comenzar el envío de datos, debería establecerse una comunicación mediante el *handshake* de tres vías. Ese establecimiento de conexión es con una cardinalidad uno a uno, lo que no resultaría práctico para el establecimiento de conexión uno a muchos (lo que es, en forma sencilla, la idea de multicast). Por esto, se utiliza UDP para multicasting, debido a que no es necesario un establecimiento de conexión previo al envío de datos.

**10. Netstat**

Netstat (network statistics) es una herramienta de línea de comandos que muestra un listado de las conexiones activas de un ordenador, tanto entrantes como salientes. La información que resulta del uso del comando incluye el protocolo en uso, las direcciones IP tanto locales como remotas, los puertos locales y remotos utilizados y el estado de la conexión.

t: TCP  
u: UDP  
a: all  
n: numeric  
p: program  
  
-r, --route Muestra la tabla de enrutamiento.  
-i, --interfaces Muestra la tabla de interfaces  
-g, --groups Muestra los miembros del grupo de multidifusión  
-s, --statistics Muestra estadísticas de red (como SNMP)  
-M, --masquerade Muestra conexiones enmascaradas  
-v, --verbose Muestra más información en la salida  
-n, --numeric No resuelve nombres en general  
--numeric-hosts No resuelve el nombre de los hosts  
--numeric-ports No resuelve el nombre de los puertos  
--numeric-users No resuelve los nombres de usuarios  
-N, --symbolic Muestra los nombres del hardware de red  
-e, --extend Muestra otra/mas información.  
-p, --programs Muestra PID o nombre del programa por cada socket  
-c, --continuous Muestra continuamente las estadísticas de red (hasta que se interrumpa el programa)  
-l, --listening Muestra los server sockets que están es modo escucha  
-a, --all, --listening Muestra todos los sockets (por defecto únicamente los que están en modo conectado)  
-o, --timers Muestra los timers  
-F, --fib Muestra el Forwarding Information Base (por defecto)  
-C, --cache Mostrar el cache de enrutamiento en ves del FIB

● netstat -nat

● netstat -natp

● netstat -nlt

● netstat -nau

● netstat -naup

**11. En maquina**

12. COMPLETAR

13. COMPLETAR

14. COMPLETAR

**15. Utilizando el Live CD. Use el analizador de paquetes Wireshark para capturar los paquetes enviados y recibidos en cada uno de los siguientes casos. Arranque la captura antes de realizar cada una de las acciones indicadas:**

**A) Abra un navegador e ingrese a la URL:** [**www.redes.unlp.edu.ar**](http://www.redes.unlp.edu.ar/)

**i. Analice la secuencia de mensajes que ocurren antes de que se intercambien mensajes HTTP. ¿Cuál es el objetivo? ¿Qué flags se utilizan en cada caso? ¿Qué indica cada uno?**

Tras las consultas DNS para obtener la IP de www.redes.unlp.edu.ar, se envían tres segmentos TCP para establecer la conexión -**three-way handshake**-.

1. Cliente#35150 a Servidor#80, flag SYN: Es el pedido de conexión del cliente al servidor. El cliente indica su *secuence number* inicial (0), su *tamaño de ventana*, y a través de *options* algunos parámetros para la conexión (como el *maximum segment size* (16K) y el *window scale* (6)).
2. Servidor#80 a Cliente#35150, flag ACK,SYN: Es la aceptación de la conexión del servidor al cliente. El servidor confirma la conexión -flag *ACK* y *acknowledgement number* en 1(*0+1*)- y le pide al cliente que él reserve también recursos para la conexión -flag *SYN* y *secuence number* inicial del servidor en 0-.
3. Cliente#35150 a Servidor#80, flag ACK: Es el acuse de recibo del cliente por el segmento SYNACK que envió el servidor - *acknowledgement number* en 1 (*0+1*), confirma al segmento con número de secuencia 0 del servidor-.

**B) Cierre el navegador:  
i. Analice la secuencia de mensajes que ocurren al hacerlo ¿Cuál es el objetivo? ¿Qué flags se utilizan en cada caso? ¿Qué indica cada uno?**

Al cerrar el navegador se produce el cierre de la conexión TCP, en cuatro pasos (dado que se cierra por separado desde cada host -***half close***-, pudiendo quedar "medio abierta" si sólo uno de ellos la cierra). En el caso de la captura, el Servidor había iniciado el cierre de conexión cuando se cerró el browser:

1. Servidor#80 a Cliente#35150, flag FIN, ACK: El servidor solicita el fin de la conexión. La presencia de ACK se debe a que en el mismo segmento, confirmó la recepción de un segmento anterior del cliente, pero nada tiene que ver con el cierre de conexión.
2. Cliente#35150 a Servidor#80, flag ACK: El cliente da acuse de recibo del pedido de cierre de conexión del servidor. El servidor queda esperando el pedido de cierre desde el cliente.
3. Cliente#35150 a Servidor#80, flag FIN: El cliente solicita el fin de la conexión al servidor.
4. Servidor#80 a Cliente#35150, flag ACK: El servidor da acuse de recibo del pedido de cierre de conexión del cliente.

En este caso, el servidor hizo un *cierre activo* -fue quien inicio el cierre- y el cliente un *cierre pasivo*; bien pudo ser al revés, o pudieron realizar cierres simultáneos.

**C) Abra un navegador e ingrese a la URL: www.redes.unlp.edu.ar/ y abra otra instancia o solapa del navegador en ingrese a la URL: www.redes.unlp.edu.ar/  
i. Observe: IP origen, IP destino, Puerto Origen, Puerto destino, Protocolo de transporte de los paquetes pertenecientes a cada una de las conexiones ¿Cómo es posible conectarse 2 veces en forma simultánea al mismo lugar? ¿Qué distingue una conexión de otra?**

Los protocolos, IPs y port del Servidor coinciden. La diferencia está en el port que el cliente asigna a cada conexión, lo que le permite multiplexarlas.

**D) Desde la consola de root use el servicio tftp:  
i. Ejecute tftp localhost y copie un archivo cualquiera desde su PC al servidor, a través de la opción put.  
ii. Borre el archivo de su PC y obtengalo ahora del servidor a través de la opción get.  
iii. ¿Qué diferencias encuentra con los puntos A y B en cuanto a mensajes intercambiados?  
iv. ¿Qué diferencias y similitudes encuentra con los puntos A y B respecto a datos incluidos en los campos de control: IP Origen, IP Destino, Puerto Origen, Puerto destino, Protocolo de Transporte?**

* **TFTP: Trivial File Transfer Protocol.** Es un protocolo de transferencia de archivos muy simple, que funciona sobre UDP puerto #69 en vez de TCP #20 y #21 como FTP. Al no ser orientado a conexión, no tiene autentificación de usuario, encriptación, y no puede listar contenidos de directorios. Utiliza un segmento inicial indicando el nombre del archivo, el tipo de operación y el modo de transferencia. Los archivos se transmiten en segmentos numerados de 512 bytes, excepto el último que necesariamente debe ser inferior para indicar el fin de la transmisión (si el archivo tiene un tamaño múltiplo de 512, se envía un segmento con 0 bytes de datos para indicar el fin).

**~$ ls -l >listado.txt** #genero un archivo para probar  
**~$ tftp localhost** #me conecto -sin usuario ni nada en TFTP  
**tftp> ?**  
Commands may be abbreviated.  Commands are:  
  
connect         connect to remote tftp  
mode            set file transfer mode  
put             send file  
get             receive file  
quit            exit tftp  
verbose         toggle verbose mode  
trace           toggle packet tracing  
status          show current status  
binary          set mode to octet  
ascii           set mode to netascii  
rexmt           set per-packet retransmission timeout  
timeout         set total retransmission timeout  
?               print help information  
**tftp> put listado.txt listadoEnServidor.txt**  
Sent 239 bytes in 0.0 seconds  
**tftp> get listadoEnServidor.txt list2.txt**  
Received 239 bytes in 0.0 seconds  
**tftp> quit**  
**~$ ls**  
captura.pcap  Desktop  list2.txt  listado.txt  Maildir

**~$ find / -name listadoEnServidor.txt 2>/dev/null          #busco donde guardó el servidor el archivo que subí por TFTP**  
/redes/tftpserver/listadoEnServidor.txt  
/fll/cow/redes/tftpserver/listadoEnServidor.txt  
/fll/aufs/redes/tftpserver/listadoEnServidor.txt  
**~$ ls /fll/aufs/redes/tftpserver/listadoEnServidor.txt -l  #me fijo con qué usuario queda**  
-rw-rw-rw- 1 **nobody nogroup** 234 may 18 17:11 /fll/aufs/redes/tftpserver/listadoEnServidor.txt

**En cuanto a los mensajes intercambiados:** Con TFTP no hay establecimiento de conexión ni confirmaciones. Los mensajes fueron: pedido de escritura del cliente, acuse de recibo del servidor, envío del archivo desde el cliente en bloque #1, acuse de recibo del bloque #1 desde el servidor, pedido de lectura, bloque de datos #1 desde el servidor, acuse de recibo del bloque #1 desde el cliente.

**En cuanto a las IP, puertos y protocolos:** Las IPs corresponden a cliente y servidor. El protocolo de transporte usado es UDP en vez de TCP. Los puertos no se mantienen como en los casos anteriores: varían para cada archivo a transmitir; el cliente se conecta al puerto #69 del servidor, pero recibe las respuestas inmediatamente desde un puerto no reservado (>1023), que se continúa utilizando para la transmisión del primer archivo. Para descargar el segundo archivo, el cliente utiliza un nuevo puerto no reservado y el servidor le responde también desde otro puerto. Sólo se utiliza el puerto #69 para el primer segmento de cada transmisión.

(D es diferente)

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

**17. En máquina**

**18. ¿Cual es el puerto por defecto que se utiliza en los siguientes servicios?**

* **Web** 80 o 8080
* **SSH** 22
* **DNS** 5353
* **Web Seguro** 443
* **POP3** 110
* **IMAP** 143
* **SMTP** 25

Investigue en que lugar en Linux y en Windows está descripta la asociación utilizada por defecto para cada servicio

**Windows**: ?

**Linux**: /etc/services

**19. Stop-and-Wait**

COMPLETAR

Al ser **Stop-and-Wait** (es decir, el emisor envia un paquete y queda en el estado de espera de un ACK. Cuando el emisor esta en éste estado no puede conseguir mas datos desde la capa superior hasta que reciba un ACK y deje este estado, o se venza el tiempo de espera de respuesta).  
Por utilizarse S**top-and-Wait**, sólo hay en cada momento un segmento enviado esperando confirmación. El rango de los números de secuencia es sólo [0;1] (es un protocolo sin *pipelining*).

**20. Go Back N**

Un protocolo **Go Back** - **N** (GNB) es un protocolo **ARQ** con *pipelining* (es decir, soporta múltiples segmentos en tránsito pendientes de confirmación, implementando buffers en el emisor; el rango de los números de secuencia debe aumentarse), por esto el emisor puede transmitir varios paquetes sin esperar ningun reconocimiento, pero tiene restringido el numero maximo, *N,* de paquetes no reconocidos en el entubado, llamado **ventana** del emisor. El emisor tambien tiene un **timer** para controlar el timeout.   
La recepción del ACK de un segmento, confirma a todos los anteriores pendientes -**acuse de recibo acumulado**-. El vencimiento del timer para un segmento implica la retransmisión de todos los segmentos siguientes en la ventana. La ventana se desplaza a medida que se reciben ACK's.

El receptor envía el ACK sólo del segmento recibido correctamente de mayor número de secuencia en orden -es decir, que no le falte ninguno-. Puede generar acuses duplicados. No precisa buffers, ya que descarta los segmentos fuera de orden y duplicados -al descartar reenvía el ACK del mayor número de secuencia que recibió-.

**21.**

COMPLETAR

**22.**

COMPLETAR

**23. ¿Qué restricción existe sobre el tamaño de ventanas en el protocolo Selective Repeat?**  
  
**Selective Repeat** resuelve el problema de **GBN** de que el emisor llene la tuberia con paquetes (por un gran tamaño de la ventana y el retardo debido al ancho de banda son grandes) un simple error en un paquete puede ocacionar que GBN retransmita un gran numero de paquetes, muchos de los cuales resultan innecesatrios.  
**Selective Repeat** (SR) es otro protocolo ARQ con *pipelining*. Éste evita la retransmicion innecesaria de paquetes haciendo que el emisor retransmita unicamente aquellos paquetes sospechosos de haber sido recibidos con errores (los que fueron perdidos o corrumpidos) por el receptor. Esta retransmicion individual, requerira que el receptor reconozca los paquetes recibidos de forma *invididualizada*. Tanto el emisor como el receptor manejan una ventana, enviando el receptor ACK's individuales para cada segmento. Esto evita retransmisiones innecesarias, dado que el receptor no descarta tantos segmentos y puede pedir la retransmisión de uno específico.

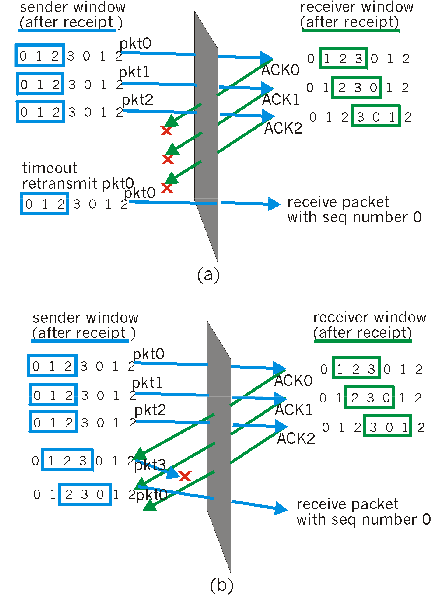
La llegada de un segmento del comienzo de la ventana del receptor correctamente hace que se envíen a la capa aplicación todos los datos disponibles en orden, desplazando la ventana.  
En el emisor, la ventana puede contener segmentos con y sin ACK's -desplazará la ventana cuando los primeros estén con ACK. Deberá manejar timers individuales para cada paquete sin ACK.   
Si la ventana del receptor se desplaza completa y recibe un segmento anterior a la ventana -retransmisión-, el receptor debe reenviar el ACK correspondiente -o el emisor seguirá reenviándolo cada vez que venza su timer, trabando al desplazamiento de su ventana-.

En GBN, si el delay de transmisión y el tamaño de la ventana son grandes, el emisor puede estar mucho tiempo esperando los ACK's, y un solo acuse NACK puede producir la retransmisión de muchos segmentos.

**Selective Repeat** es otro protocolo ARQ con *pipelining*. Tanto el emisor como el receptor manejan una ventana, enviando el receptor ACK's individuales para cada segmento. Esto evita retransmisiones innecesarias, dado que el receptor no descarta tantos segmentos y puede pedir la retransmisión de uno específico.

Restriccion del tamaño de la ventana: en este ejemplo el rango de cuatro numeros de secuencia (0,1,2,3) y un numero de ventana de tres. Suponga que se transmiten los paquetes del 0 al 2 y son recibidos correctamente y reconocidos por el receptor. En este punto, la ventana del receptor esta sobre los paquetes cuarto, quinto y sexto, los cuales tienen los numeros de secuecia 3, 0 y 1, respectivamente. Ahora concidere dos escenarios. El primero, la figura (a), se pierden los ACK de los tres primeros paquetes (y el emisor retransmite estos paquetes). El receptor, por lo tanto, recibe un paquete con numero de secuencia 0 (una copia del primer paquete enviado).  
En el segundo escenario, la figura (b), los ACK de los tres primeros paquetes se entregan correctamente. El emisor avanza su ventana y envia el cuarto, el quinto y el sexto paquete, con numero de secuencia 3, 0 y 1, respectivamente. El paquete con numero de secuencia 3 se pierde, pero el paquete con numero de secuencia 0 (que contiene datos nuevos) llega correctamente.

No hay forma de distinguir la retransmicion del primer paquete de la transmision original del quinto. Resulta evidente que un tamaño de ventana menor en una unidad que el tamaño del espacio de numeros de secuencia no va a funcionar. Pero ¿cuanto menor tiene que ser el tamaño de ventana? **El tamaño de la ventana debe ser menor o igual a la mitad del tamaño del espacio de numeros de secuencia**

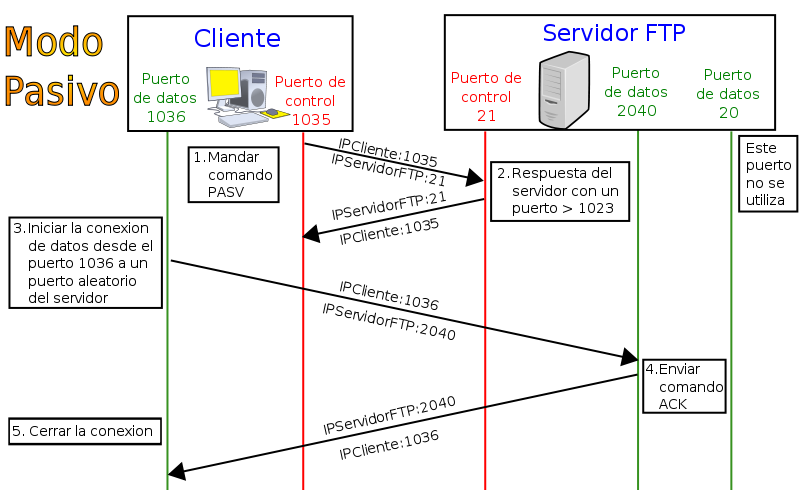
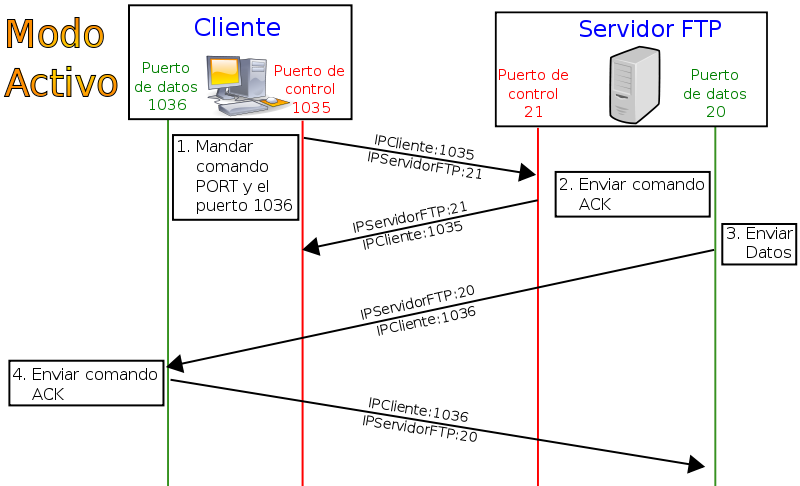
(al final del capitulo del libro dice que esta)

**24. FTP**

FTP es out-of-band porque utiliza dos conexiones TCP paralelas para la transmisión de archivos, una de **control/comandos** y otra para **datos** -puertos #21 y #20 del lado servidor respectivamente-, aunque el puerto de datos puede variar en conexiones en modo pasivo. *Lo habitual es que comandos y datos se envíen por una misma conexión, aunque FTP no es el único que trabaja out-of-band.*

**Modo activo:**   
En modo Activo, el servidor siempre crea el canal de datos en su puerto 20, mientras que en el lado del cliente el canal de datos se asocia a un puerto aleatorio mayor que el 1024. Para ello, el cliente manda un comando PORT al servidor por el canal de control indicándole ese número de puerto, de manera que el servidor pueda abrirle una conexión de datos por donde se transferirán los archivos y los listados, en el puerto especificado.  
Lo anterior tiene un grave problema de seguridad, y es que la máquina cliente debe estar dispuesta a aceptar cualquier conexión de entrada en un puerto superior al 1024, con los problemas que ello implica si tenemos el equipo conectado a una red insegura como Internet. De hecho, los cortafuegos que se instalen en el equipo para evitar ataques seguramente rechazarán esas conexiones aleatorias. Para solucionar esto se desarrolló el modo Pasivo.

**Modo pasivo:  
Cuando el cliente envía un comando PASV sobre el canal de control, el servidor FTP le indica por el canal de control el puerto ( mayor a 1023 del servidor. Ej:2040 ) al que debe conectarse el cliente. El cliente inicia una conexión desde el puerto siguiente al puerto de control (Ej: 1036) hacia el puerto del servidor especificado anteriormente (Ej: 2040).**Antes de cada nueva transferencia, tanto en el modo Activo como en el Pasivo, el cliente debe enviar otra vez un comando de control (PORT o PASV, según el modo en el que haya conectado), y el servidor recibirá esa conexión de datos en un nuevo puerto aleatorio (si está en modo pasivo) o por el puerto 20 (si está en modo activo).



**25. Usando FTP: activo vs pasivo**

**Activo:**Origen es C Puerto Origen Puerto Destino   
  *33714 21 → puerto de control*Se logue etc...  
  
El cliente manda un 'ls', pero antes (automaticamente) manda el comando PORT, indicando al servidor cual es su puerto activo (47722), despues si envia el 'ls'.  
Origen es C Puerto Origen Puerto Destino   
  *33714 21*El servidor inicia la conexión desde su puerto 20 (datos) al puerto indicado por el cliente (47722) y envia los datos. Luego de la transferencia el servidor cierra la conexión.Origen es S Puerto Origen Puerto Destino  
  *20 47722*Se repite lo mismo para mas comandos.

**Pasivo:**

Origen es C Puerto Origen Puerto Destino

*34348 21 → puerto de control*Se loguea...etc

Manda el comando PASSIVE por el puerto de control: (passive mode on)  
Origen es C Puerto Origen Puerto Destino

*34348 21*

El servidor le responde que el puerto de datos es el nº 10632

Origen es S Puerto Origen Puerto Destino

*21 34348*El cliente manda un 'ls'. Primero crea la conexión en el nuevo puerto de datos  
Origen es C Puerto Origen Puerto Destino   
 *54335 10632*  
  
Y despues envia el comando 'ls' por el puerto 21, el servidor acepta la conexión y envia los datos por el puerto de datos (10632). La conexión es cerrada por el servidor.  
Si el cliente manda otros comandos, se repite que: el cliente envia (automaticamente) el comando passive, el servidor le indica el nº de puerto de datos al cual conectarse, se transfieren los datos y se cierra la conexión.