Redes de ordenadores

TCP

Grupo de sistemas y comunicaciones
Juan Jesús Muñoz Esteban
jjmunoz@gsyc.inf.uc3m.es

3. Nivel de transporte: TCP y UDP

El nivel 2 permite en control de errores y de flujo entre máquinas directamente accesibles.

Cuando se producen comunicaciones entre redes, no estando los ordenadores directamente conectados unos a otros, el control de errores (IP sólo comprueba su cabecera) y de flujo desaparecen. Los datagramas pueden llegar por rutas distintas y en consecuencia desordenados, llegar duplicados, perderse en colas saturadas...

El nivel 4 se encarga de un control de errores y de flujo extremo a extremo, recuperandose de los problemas que se producen en un nivel de red no fiable como IP: datagramas perdidos, duplicados, desordenados...

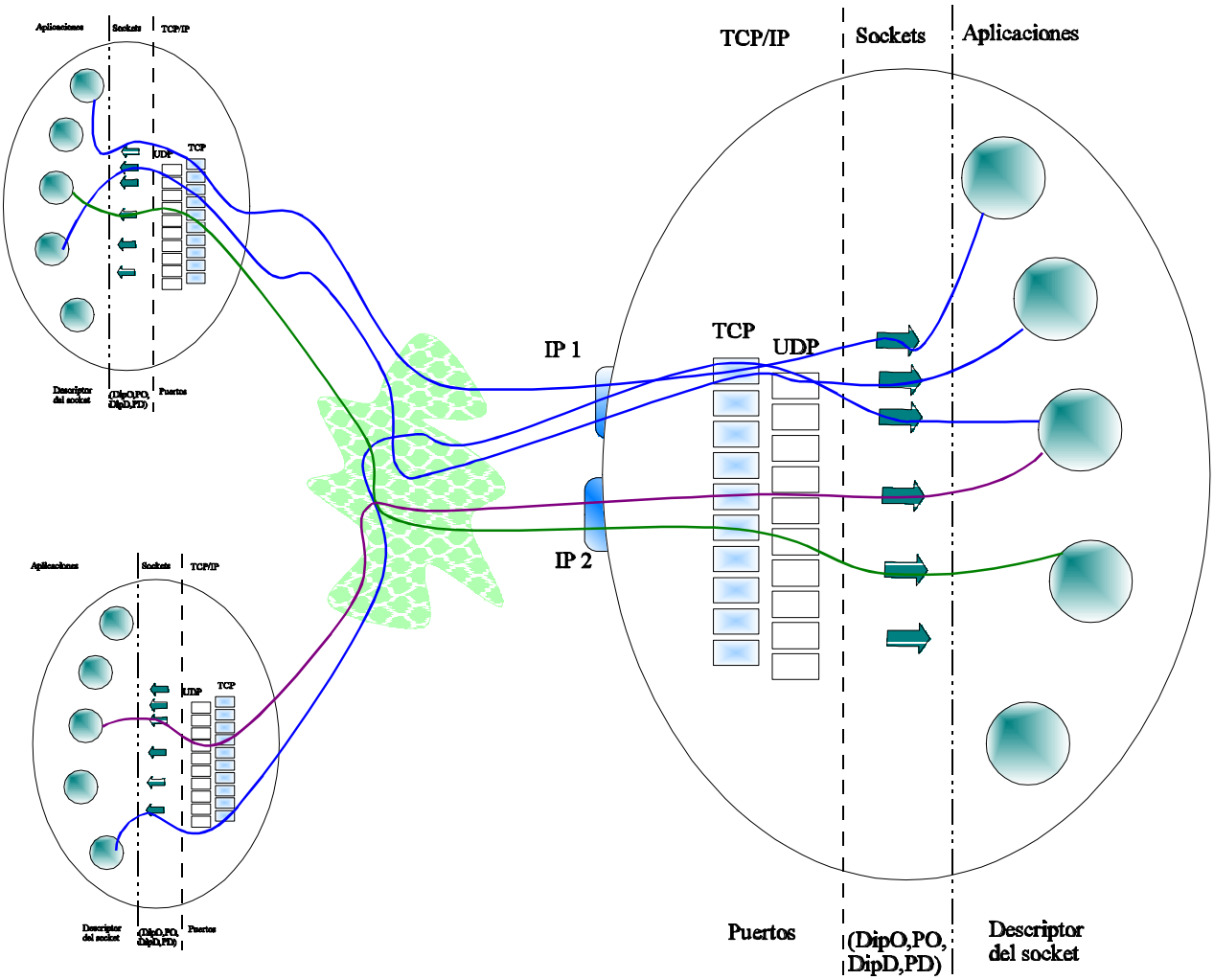
En todo caso el nivel 4 puede ser un mero transmisor de los datos recibidos por la red, y dejar que sean las propias aplicaciones quienes realicen el control de errores que necesiten.

Los ordenadores albergan en general diversas aplicaciones concurrentemente, por lo que el nivel de transporte también extiende el direccionamiento incorporando el concepto de puerto, de manera que una dirección IP se puede multiplexar permitiendo que varias aplicaciones de un mismo ordenador puedan simultáneamente estar comunicandose con otras aplicaciones en otros ordenadores

.Los puertos permiten a varias aplicaciones acceder simultáneamente a las comunicaciones. Hay servicios fijos, cuyos números de puerto están estandarizados, y el resto de los puertos se asignan dinámicamente por el sistema operativo según son solicitados.

Una conexión supone un enlace bidireccional entre dos puertos de la misma o diferente máquina. Cada par (dirección IP, número de puerto) define una conexión, de manera que un mismo puerto permite a una aplicación mantener distintas conexiones con otras máquinas y puertos.

3.1 Puertos y aplicaciones



3.2 Puertos vs sockets

Los programas se crean y se destruyen continuamente en los ordenadores. Es imposible establecer un mecanismo rígido que asigne direcciones fijas a los procesos. Se necesita que el sistema operativo multiplexe las entradas IP para repartirlas entre las aplicaciones, a las que debe asignar descriptores con los que manejar sus puntos finales de comunicación.

Cuando se recibe un datagrama se analiza si el valor del campo protocolo de la cabecera es TCP o UDP. Luego se mira el número de puerto y se entrega el contenido a la aplicación que esté conectada a ese puerto. Los puertos multiplexan a nivel de transporte. Una dirección final es por tanto una dirección de máquina mas un puerto.

En UNIX hay un demonio (inetd) que escucha por una serie de puertos definidos (/etc/inetd.conf). Cuando llega un mensaje para el puerto en cuestión, lanza la aplicación relacionada con el mismo y le pasa el mensaje.

Los sockets son el estándar de facto para el acceso a los protocolos Internet, pero no son parte de TCP/IP. Los sockets son puntos de acceso a las comunicaciones. Se manejan como descriptores de ficheros y el S.O. los asigna como puntos terminales de comunicaciones bidireccionales. Se desarrollaron en BSD Unix, pero como mecanismo independiente del sistema operativo. Existen otros APIs para acceder a TCP/IP: TLI (transport layer interface) de AT&T y su superconjunto XTI de X/Open.

Los sockets son puntos de acceso a las comunicaciones (no protocolos). Definen un servicio de sesión-transporte. Un socket es un extremo de una comunicación bidireccional entre dos aplicaciones (de la misma u otra máquina). Admiten comunicación fiable y no fiable. Hay unas librerías de funciones para programar directamente sobre los sockets, y existen multitud de ejemplos sobre cómo construir clientes y servidores con este sencillo interfaz.

3.3 Asignación de puertos

Los puertos se identifican por números de 16 bits. Los de TCP y los de UDP se manejan por separado (el datagrama IP ya distingue el tipo de protocolo de nivle 4 que encapsula). Se asignan con el siguiente criterio:

1. Well Known Ports

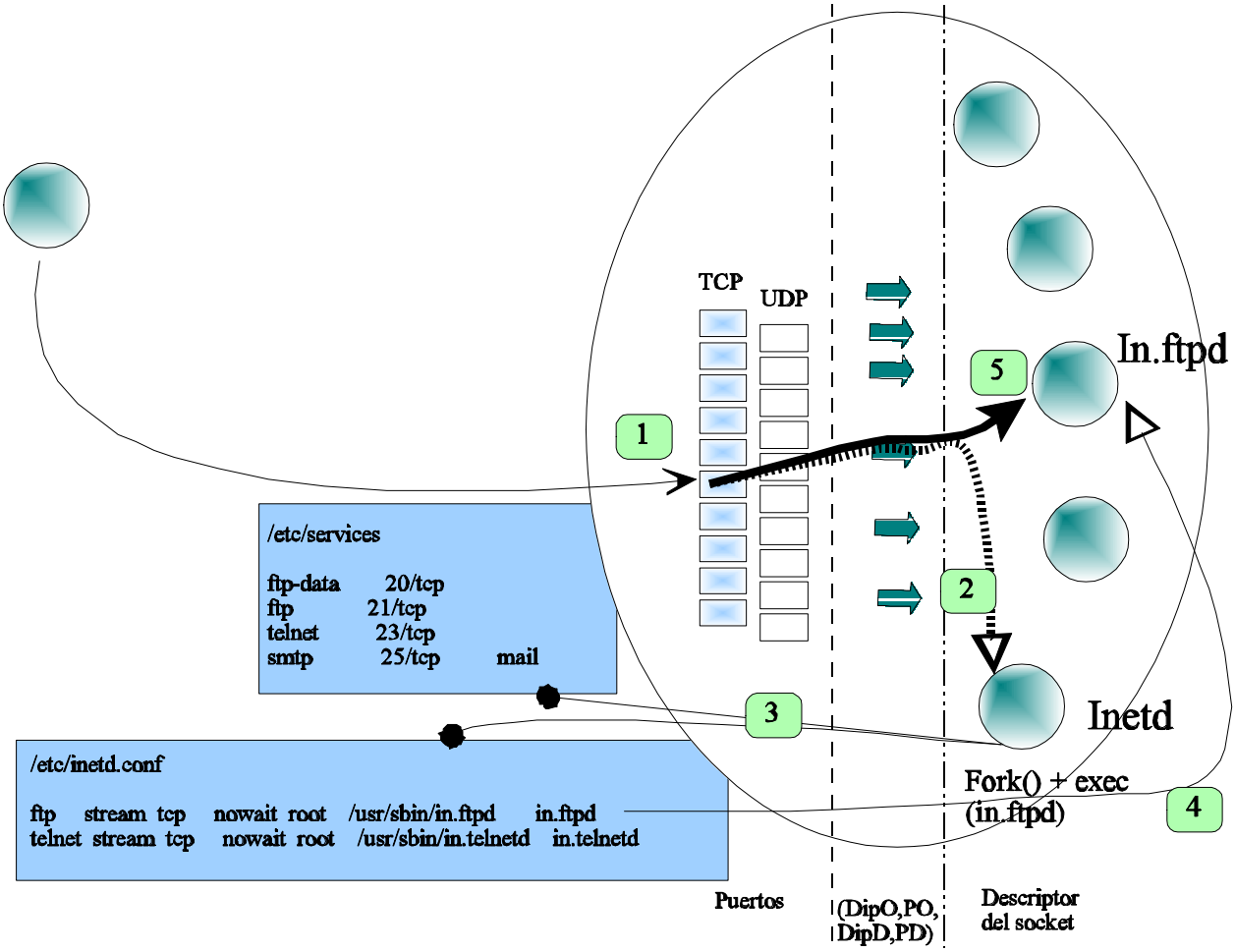
Los números de puerto 1 a 255 están reservados, y tienen una asignación universal (/etc/services)

```
# Network services, Internet style
#
tcpmux          1/tcp
echo            7/tcp   (la misma funcionalidad tiene el mismo número en tcp
echo            7/udp    que en udp, pero ésto no es obligatorio)
discard         9/tcp           sink null
discard         9/udp           sink null
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
ftp-data        20/tcp
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
name            42/udp          nameserver
whois           43/tcp          nicname         # usually to sri-nic
hostnames       101/tcp         hostname        # usually to sri-nic
sunrpc          111/udp         rpcbind
sunrpc          111/tcp         rpcbind
snmp            161/udp         # Simple Network Mgmt Protocol (SNMP)
```

2. Dynamic binding

El resto de los números los gestiona el software de red (integrado en el S.O), y los asigna dinámicamente según son solicitados por las aplicaciones (no se da un mismo puerto a dos aplicaciones, e incluso si una nueva instanciación de una aplicación pide un número fijo de puerto, este sólo se le asigna tras un plazo de reutilizción). Además puede haber un programa (inetd) que escucha por todos los puertos de los servicios definidos y lanza los procesos servidores adecuados)

3.4 Enlace de puertos y aplicaciones



Una apertura pasiva es la que se produce cuando una aplicación informa al sistema operativo de que está dispuesta a aceptar una comunicación entrante. Se produce una asignación de puerto para ella, y queda en espera.

Una apertura activa genera una serie de mensajes por la red, a fin de conseguir una respuesta del otro extremo mediante un protocolo de conexión.

3.5 UDP

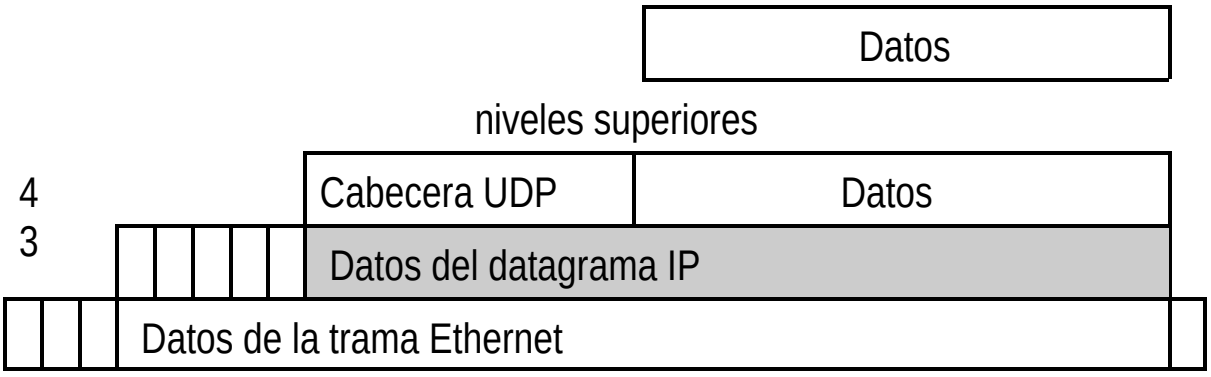
El servicio no fiable tiene unas características idénticas a las de IP, pero añadiendo los números de puerto.

Las direcciones se representan: (128.110.56.9,111) (ver ftp)

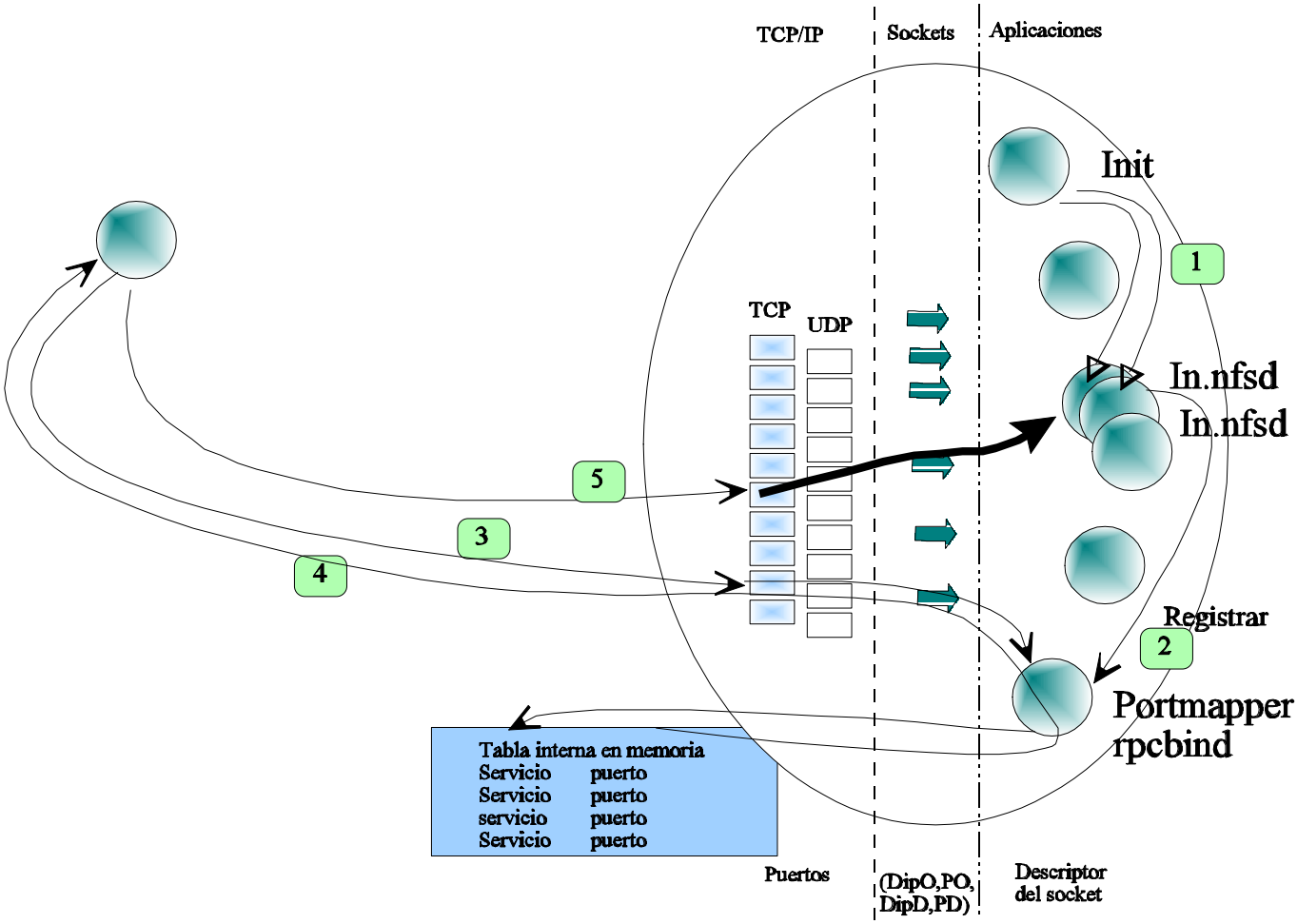
No es orientado a conexión: los mensajes se pueden perder, duplicarse o llegar desordenados.

No es fiable (lo cual no significa que camufle datos erróneos): El servicio cuenta con que las respuestas necesitan comprobación extra (responsabilidad de la aplicación).

Es más ligero que TCP (en una LAN, donde hay pocos fallos y se aplica el CRC, supone más velocidad), pero el programador tiene que implementar sus propios mecanismos de comprobación de errores.

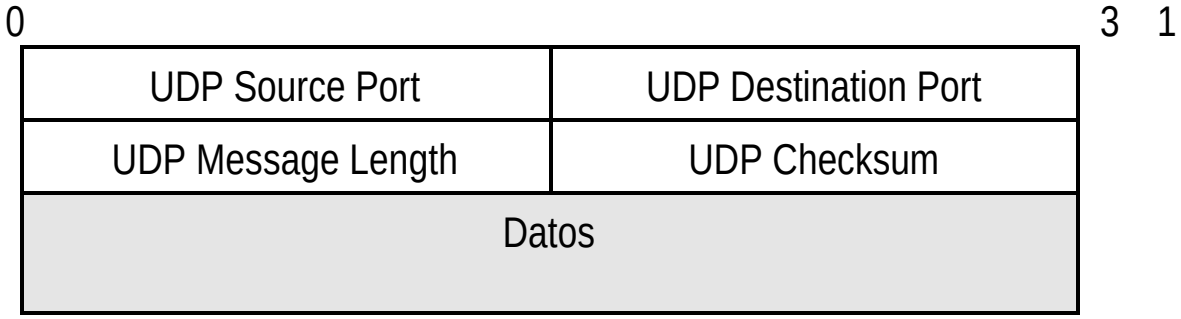


3.6 Localización de servidores: portmapper



Cuando el servicio dura menos tiempo que la creación de un proceso servidor para atenderlo, el sistema de puertos prefijados no es adecuado. Existe un modelo basado en un puerto conocido en el que preguntar en qué puerto hay un servidor disponible, de manera que el cliente antes de usar por primera vez el servicio, pregunte por un puerto de uno de los servidores disponibles.

3.7 Formato del paquete

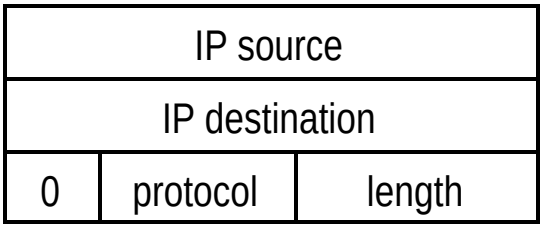


El puerto origen es opcional (si no se especifica dónde se esperan las respuestas, debe ponerse a cero).

La longitud del mensaje se especifica en octetos, incluyendo la cabecera UDP (8 es el valor mínimo)

El checksum es opcional (si no lo usas no tienes más garantía que el CRC de nivel 2). 0 significa que no se calcula. Si tras calcularlo el resultado es 0, se representa con todos los bits a 1 (también es 0).

Para calcularlo, se dividen la cabecera, una pseudocabecera y los datos en trozos de 16 bits (se rellena con ceros si falta al final), de los que se calcula su complemento a uno y los suma. Utiliza una pseudocabecera además de la de UDP:



De esta manera se salta el modelo por niveles para comprobar que ha llegado al sitio correcto (la cabecera de UDP sólo lleva el número de puerto, que existe en todas las máquinas).

3.8 TCP

Transmission Control Protocol es un protocolo de comunicación, no un componente software. Define un servicio fiable tipo stream. Es una especificación de los formatos de los datos intercambiados, la secuencia de los reconocimientos de recepción...

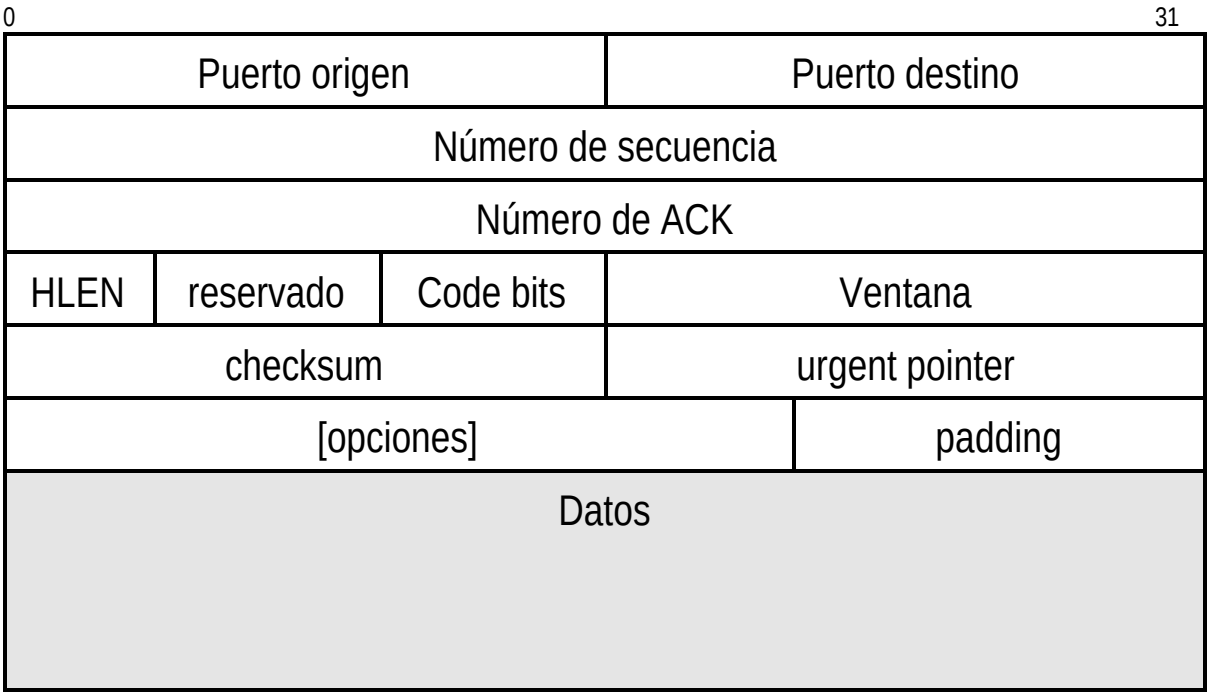
No asume nada sobre el sistema de comunicación subyacente (no está ligado a IP), y es la base sobre la que se ha desarrollado TP4. Tampoco define los detalles de cómo los programas de aplicación deben acceder a él, para que se pueda implementar sobre cualquier S.O.

Cuando fuesen necesarias grandes transferencias de datos, los programadores tendrían mucho trabajo repetitivo. TCP se lo da hecho proporcionando un servicio de entrega fiable:

1. Orientado a stream: chorro de bytes.
2. Conexión virtual: hay una fase de establecimiento lógico de la conexión y otra de transferencia. Es fiable porque se recupera de fallos temporales (pérdidas de datagramas, replicados, reordenamiento...), aunque si el problema es persistente notifica el error al nivel superior.
3. La transferencia se apoya en buffers: Las aplicaciones envían y reciben grupos de bytes de un flujo contínuo, que se almacenan en memoria y se transmiten por la red en segmentos independientes, bien cuando hay suficientes datos o bien cuando se obliga expresamente.
4. El flujo de datos no tiene estructura predeterminada. Los extremos deben estar de acuerdo en la interpretación de una secuencia de bytes no dividida en registros.
5. Conexión Full Duplex: ambos sentidos simultáneamente.

Piggybacking: Los asentimientos de recepción van en las tramas de envío de datos en el sentido opuesto, para reducir el tráfico.

3.9 Formato del segmento



Los puertos son números de 16 bits que seleccionan la aplicación o servicio al que se accede. Son independientes de los de UDP.

El número de secuencia es un entero que indica cual es la posición del primer byte del segmento dentro del stream. El valor inicial, al establecerse la conexión, debe ser aleatorio.

ACK: número de secuencia del siguiente byte que se espera recibir (asiente la llegada de todos los anteriores).

HLEN: Longitud de la cabecera (múltiplo de 32 bits). Estrictamente significa "offset del inicio de los datos".

Los 6 bytes reservados para uso futuro deben estar a 0 .

La ventana especifica los bytes que el receptor está dispuesto a aceptar. Se utiliza para el control de flujo.

Code Bits:

URG: urgent pointer es válido
ACK: ACK is valid
PSH: Este segmento necesita entregarse cuanto antes
RST: Reset conexión
SYN: Sincronizar los números de secuencia
FIN: Quien envía ha llegado al final de su stream.

El checksum se calcula a partir de una pseudocabecera como la de UDP, para comprobar que el destino es correcto: un puerto local puede tener varias conexiones a distintas máquinas en el mismo número de puerto simultáneamente: debe distinguir a cual pertenece cada segmento.

Se calcula el complemento a 1 de la suma de los complementos a 1 descomponiendo el segmento en bloques de 16 bits. El checksum se calcula sobre la cabecera, la pseudocabecera y los datos (se asume que el cheksum es 0 durante el cálculo). Observar que en la pseudocabecera se especifica el protocolo (no asume IP).

Las opciones se usan como en IP: Van una detrás de otra, con su clase y valor codificados en bytes. Se emplea para negociar el tamaño máximo de segmento (en función del MTU y de la velocidad relativa de los ordenadores conectados).

El PADDING es relleno para expresar HLEN en palabras de 32bits

Tipos de Segmentos:

1. De establecimiento de conexión
2. De datos
3. ACKs (normalmente usa piggybacking, salvo que el receptor retrase los acks)
4. Tamaños de ventana
5. Cierre de conexión

3.10 Segmentos, streams y números de secuencia

Los datos se envían en segmentos de tamaño variable (conviene que se adapten a las longitudes de trama de los niveles inferiores para aumentar las prestaciones).

Los números de secuencia se refieren a bytes. Cada segmento especifica cual es el número de orden del primer byte, y cuantos se envían. El receptor con esa información podrá recomponer byte a byte el stream, reordenando los segmentos lleguen como lleguen.

Ejemplo

EN_
UN_LU
GAR (se pierde en primera instancia)
_D
E_LA_M

542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561
E	N	_	U	N	_	L	U	G	A	R	_	D	E	_	L	A	_	M	A

Y el receptor enviará ACK=550 (primer número de secuencia que no tiene).
Si posteriormente se le envía un datagrama con

GAR_DE_LA

Pondrá los caracteres en su sitio y devolverá ACK=561

3.11 Asentimientos positivos y retransmisiones

Los ACKS indican el número de orden del siguiente octeto que se desea recibir (reconociendo que se han recibido correctamente los anteriores).

No hace falta reconocer cada uno de los segmentos recibidos. Si se envía un ACK indicando que se han recibido bien 4000 caracteres, se asume que el primer segmento con los 1000 primeros fue correctamente recibido.

Los segmentos TCP envían datos en segmentos de longitud variable. Los segmentos retransmitidos pueden contener más datos que los originales, por lo que los ACKs deben referirse a caracteres, no a segmentos, y el receptor debe reconstruir la secuencia exacta, con independencia del orden de recepción.

El reconocimiento se aplica siempre al mayor bloque contiguo recibido (la pérdida de ACKs no fuerza una retransmisión), contando desde el último reconocido. No existen reconocimientos negativos ni rechazo selectivo (no se pueden reconocer los 10000 primeros caracteres a excepción del rango 200-276), lo que puede generar retransmisiones innecesarias.

La transmisión siempre es dirigida por la iniciativa del emisor, que retransmite únicamente cuando vencen los plazos y no tiene reconocimiento explícito de que un carácter haya llegado a destino, y retoma desde ahí toda la tarea hasta que reciba un asentimiento que pueda incluir otros caracteres previamente enviados y que ya no necesitará retransmitir.

3.12 Control de flujo: Ventanas deslizantes

El envío consecutivo de mensajes, asentidos uno a uno, necesita de un protocolo que garantice la fiabilidad. El más sencillo de ellos es el protocolo del bit alternante.

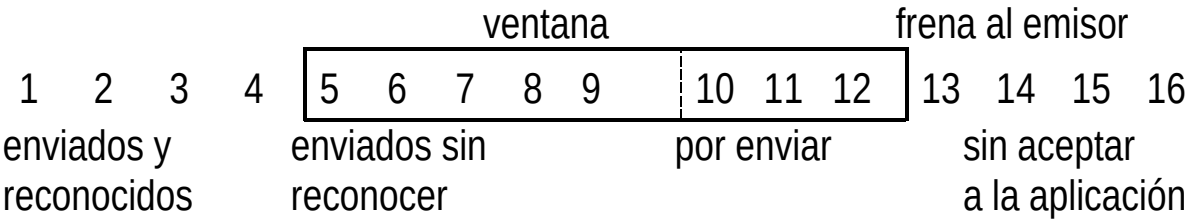
Para aumentar la eficiencia de la transmisión y realizar control del flujo extremo a extremo, se permite que el emisor emita segmentos sin haber obtenido los ACKs de los anteriores. Esto exige la existencia de memoria en el emisor y el receptor.

El emisor guarda en un número acotado de buffers, con los segmentos enviados y aún no reconocidos. Cuando los buffers se llenan, para al emisor.

El receptor almacena en sus buffers aquellos segmentos recibidos pero aún no solicitados por la aplicación. Si la aplicación no consume datos, la cola se llena y deja de reconocer nuevos segmentos.

Cada conexión lleva asociadas dos ventanas por sentido, una en el emisor y la otra en el receptor.

La implementación de cada ventana supone 3 punteros, que indican un número de secuencia referido a caracteres del stream:



El tamaño de la ventana en TCP varía con el tiempo. En cada ACK el campo WINDOW especifica la cantidad de datos que el receptor está dispuesto a recibir. Si el tamaño es mayor que el actual, se crean nuevos buffers. Si es menor, reduce el número de buffers (lógicamente no puede perder datos, por lo que la reducción del tamaño sólo se envía cuando se producen ACKs que liberan).

En el caso extremo, el receptor saturado reduce a cero la ventana para detener al emisor (si sigue enviando llena de tráfico la red para nada). Por si luego se perdiesen los mensajes aumentando la ventana, cada cierto tiempo el emisor incrementa el tamaño de la ventana.

Con este tipo de mecanismos, aunque no se puede conocer qué router está congestionado, se pueden tratar problemas de congestión. Si las colas de los routers se llenan, y comienzan a desechar datagramas y a enviar errores con ICMP, una buena implementación de TCP puede detectar el problema y recuperarse.

Aunque la ventana sea cero, el emisor puede transmitir datos que tengan puesto el bit de urgente.

3.13 Datos fuera de Banda

Aunque TCP/IP es un protocolo orientado a stream, a veces es importante enviar datos "fuera de banda", sin esperar a que el receptor lea el resto de datos que han llegado antes.

El envío de estos caracteres con caracter de "urgentes" se realiza gracias al puntero de urgencias, habilitado por el bit de código URG. Esto es especialmente interesante en el envío de interrupciones (^C).

Debe notificarse (según cada sistema operativo) a la aplicación receptora de la llegada de caracteres enviados de esta manera, sin importar su posición en el stream.

No hay que confundir este mecanismo con la el bit PUSH. Este bit sirve para indicar al receptor que debe procurarse entregar los datos a la aplicación cuanto antes. Por ejemplo, si se envía una petición de una URL que ocupa más de un datagrama, el primero no tiene por qué entregarse a la aplicación, ya que esta nada podrá hacer con solo esa parte de los datos. Pero el último si debería tener el PUSH para que cuanto antes la aplicación lo procese, habida cuenta de que ya tiene todo lo que necesita para tomar sus acciones.

3.14 Timeouts y retransmisiones

TCP espera que el destinatario reconozca los segmentos cada cierto tiempo (si tiene datos que enviar, con piggybacking, y si no con un segmento ACK).

Cada vez que se envía un segmento, se inicia un temporizador y se espera un plazo, a cuyo vencimiento se asume que se perdió el segmento y se procede a su retransmisión.

Es imposible conocer a priori cuanto tardará el enlace con la otra máquina (el número de routers atravesados y su congestión es imprevisible). TCP debe acomodarse a las diferencias entre los más rápidos y los más lentos, usando un algoritmo adaptativo: mide las prestaciones obtenidas y en función de ellas prevé su futuro.

TCP computa el tiempo entre que se envió un segmento y se recibió su ACK, y va calculando la media en el tiempo.

$$RTT = (a*viejo\_RTT) + ((1-a)*nuevo\_RTT) \qquad (0<a<1)$$

Y asume que el timeout debe valer b*RTT.

Si b es pequeño, puedes reenviar muchos paquetes innecesariamente (sólo estaban algo retrasados). Si b es grande, puedes esperar mucho antes de notar que se perdió. Recomendación: b=2.

La medida del RTT debe considerar si el ACK se produce por la llegada del paquete original (que tardó) o por el retransmitido (que fue rápido). Considerar lo primero retrasará el rendimiento, considerar lo segundo puede conducir a exigir un RTT demasiado pequeño, lo que también generará retransmisiones innecesarias.

3.15 Algoritmo de Karn

Karn propone para evaluar el RTT lo siguiente:

Los segmentos para los que se ha realizado retransmisión deben considerarse a la hora de actualizar el RTT.

Pero esto podría conducir a que nunca se ajuste: si se comienza por uno pequeño, que obliga a retransmitir, y no se ajusta el RTT, siempre será pequeño y obligará a la retransmisión.

Por ello siempre que se retransmite un segmento, TCP incrementa el timeout (normalmente se multiplica por 2, aunque Berkeley UNIX por ejemplo tiene una tabla con factores). De esta manera siempre se llegará a un segmento que llegue sin necesidad de retransmisión y que reduzca el RTT.

Retrasos variables

Hay implementaciones de TCP que tienen en cuenta no sólo un tiempo medio para RTT, sino una varianza o evaluación de cual es la diferencia entre el segmento más rápido y el más lento.

3.16 Control de la congestión

La congestión se manifiesta en grandes retardos causados por el exceso de tráfico en los servidores: encolan lo que no pueden retransmitir hasta que se llenan las colas y pierden lo que les llega.

Los retrasos aumentan las retransmisiones, y por tanto supone más sobrecarga, hasta colapsar la red.

Lo adecuado es bajar la tasa de emisión. El estándar TCP recomienda actualmente 2 técnicas: slow start y multiplicative decrease, que permiten mejorar las prestaciones entre 2 y 10 veces.

Para ello se mantiene una ventana de congestión:

ventana_permitida=min(ventana_anunciada, ventana_congestión)

Cuando no hay congestión, ambas ventanas son iguales. Cuando hay congestión, la segunda se reduce:

multiplicative decrease: cuando se pierde un segmento, se reduce la ventana de congestión a la mitad (hasta un mínimo), y se aumentan los timers de los segmentos que quedan.

Como la ventana decrece exponencialmente, se reduce el tráfico exponencialmente, para bajar pronto la congestión. Pero si cuando acaba la congestión se dobla la ventana, se llegaría pronto a otra congestión:

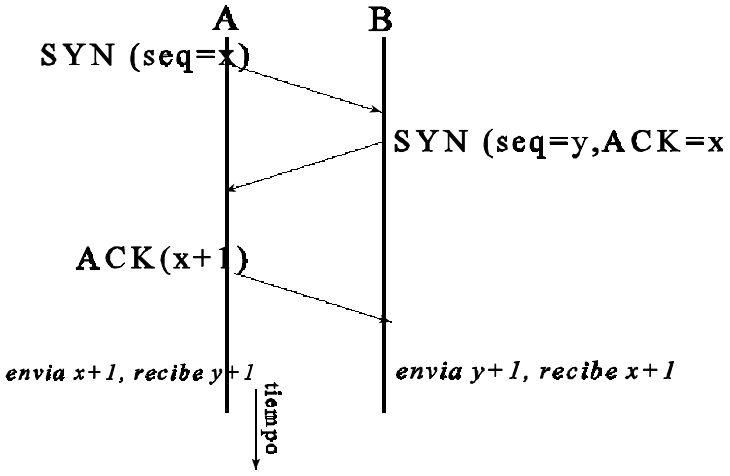
slow start: la ventana de congestión se aumenta en un segmento cada vez que se recibe un ACK.

Aún así el tráfico crece rápidamente de nuevo: envía 1, recibe su ACK, aumenta la ventana a 2 y recibe 2 ACKs, aumenta 1 por cada ACK y puede enviar 4, luego 8... Cuando la ventana de congestión llega a la mitad de su valor original, se frena la tasa de aumento de la ventana.

3.17 Establecimiento de la conexión

Como los reconocimientos asumen la aceptación de todos los datos a partir del último reconocido con anterioridad, es necesario acordar un número de secuencia inicial para emisor y receptor.

Sin el establecimiento de conexión, A podría enviar 1000bytes, que se pierden, luego otros 1000, y al recibir el ACK de 2001 (siguiente caracter que espera B) dar por recibidos los 1000 primeros.



Three-way handshake

TCP ignora peticiones de conexión una vez que ha conectado.

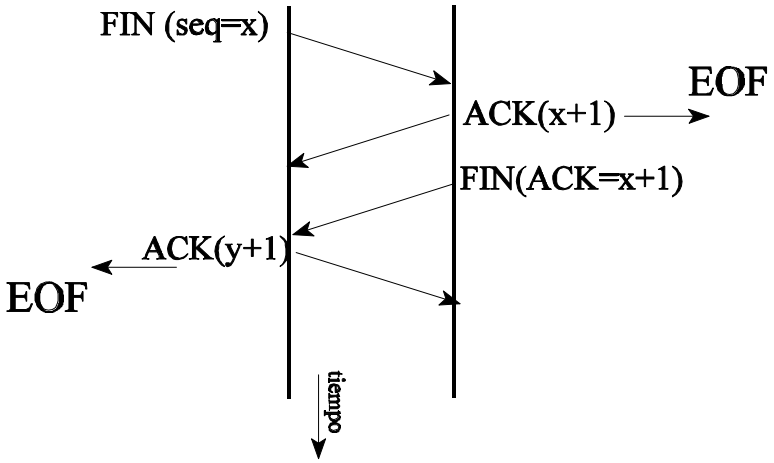
¿Qué ocurre si ambos extremos comienzan a la vez? Que cuando B recibe un SYN pero no le asienten y, usa ese mismo valor y para responder: Asocia los números de secuencia a la cuadrupla (dirección IP origen, puerto origen,dirección IP destino, puerto destino)

¿Y se pierden o duplican mensajes, o llegan desordenados?(el retransmitido y el original pueden llegar seguidos): Al final siempre se determina cual es el número de secuencia del primer carácter del stream.

3.18 Cierre de la conexión

Dos programas pueden acordar terminar su conversación de forma amigable. Para ello emplearán un protocolo de tres pasos. Como hay dos streams, uno en cada sentido, una aplicación puede decirle a la otra que no desea enviarle más datos.

Para cerrar un sentido de la conexión, se termina de transmitir lo que falte, se espera su reconocimiento (no se cierra la transmisión en un sentido mientras quedan caracteres sin asentir) y manda un segmento con el bit FIN. El receptor envía en ACK del segmento con bit FIN, e informa a la aplicación de que no hay más datos disponibles (típicamente EOF).

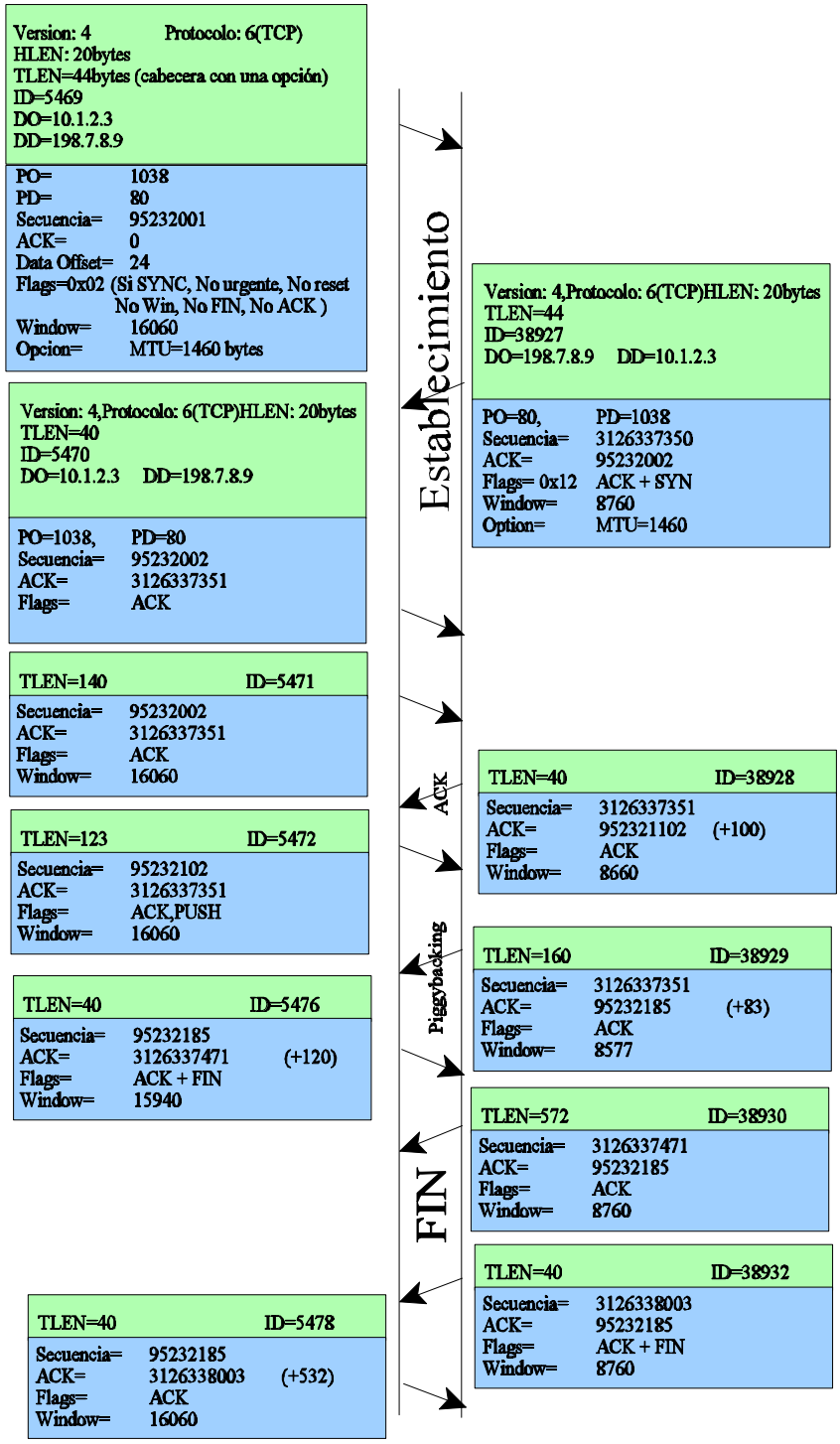


La transmisión del ACK correspondiente al FIN evita que A retransmita, ya que el receptor puede tardar en enviar su FIN.

Hay otro tipo de desconexión más expeditiva.

¿Que pasa si el último ACK no llega? Que A ha dado por finalizada completamente la conexión y la aplicación desaparece, pero B no ha obtenido el ACK y nunca podrá obtenerlo aunque reenvíe su último datragrama: espera un tiempo y abandona.

3.19 Ejemplo


Version: 4 Protocolo: 6(TCP)
HLEN: 20bytes
TLEN=44bytes (cabecera con una opción)
ID=5469
DO=10.1.2.3
DD=198.7.8.9
PO= 1038
PD= 80
Secuencia= 95232001
ACK= 0
Data Offset= 24
Flags=0x02 (Si SYNC, No urgente, No reset No Win, No FIN, No ACK)
Window= 16060
Opcion= MTU=1460 bytes
Establecimiento
Version: 4,Protocolo: 6(TCP)HLEN: 20bytes
TLEN=44
ID=38927
DO=198.7.8.9 DD=10.1.2.3
PO=80, PD=1038
Secuencia= 3126337350
ACK= 95232002
Flags= 0x12 ACK + SYN
Window= 8760
Option= MTU=1460
Version: 4,Protocolo: 6(TCP)HLEN: 20bytes
TLEN=40
ID=5470
DO=10.1.2.3 DD=198.7.8.9
PO=1038, PD=80
Secuencia= 95232002
ACK= 3126337351
Flags= ACK
TLEN=140 ID=5471
Secuencia= 95232002
ACK= 3126337351
Flags= ACK
Window= 16060
ACK
TLEN=40 ID=38928
Secuencia= 3126337351
ACK= 952321102 (+100)
Flags= ACK
Window= 8660
TLEN=123 ID=5472
Secuencia= 95232102
ACK= 3126337351
Flags= ACK,PUSH
Window= 16060
Piggybacking
TLEN=160 ID=38929
Secuencia= 3126337351
ACK= 95232185 (+83)
Flags= ACK
Window= 8577
TLEN=40 ID=5476
Secuencia= 95232185
ACK= 3126337471 (+120)
Flags= ACK + FIN
Window= 15940
TLEN=572 ID=38930
Secuencia= 3126337471
ACK= 95232185
Flags= ACK
Window= 8760
FIN
TLEN=40 ID=38932
Secuencia= 3126338003
ACK= 95232185
Flags= ACK + FIN
Window= 8760
TLEN=40 ID=5478
Secuencia= 95232185
ACK= 3126338003 (+532)
Flags= ACK
Window= 16060

ÍNDICE