Tema 3: Capa de Transporte en Internet

Aclaración

Todo el contenido de estos apuntes viene de las diapositivas de la asignatura FR del profesor José Camacho Páez http://wdb.ugr.es/~josecamacho/ más las notas que he ido tomando en clase.

Contenido

1. Introducción

UDP -> solo utiliza los puertos. **TCP** -> tiene todo lo que se implementa en la capa de transporte . Control de conexión significa que se necesita que haya un inicio de conexión y un fin de conexión.

Funciones y servicios de la capa de transporte:

- Comunicacion *extremo a extremo* (end to end).
- *Multiplexación/demultiplexacíón* de aplicaciones -> puerto.

Protocolo UDP:

- *Multiplexación/demultiplexación* de aplicaciones.
- Servicio *no orientado a conexión,no fiable*.

Protocolo TCP:

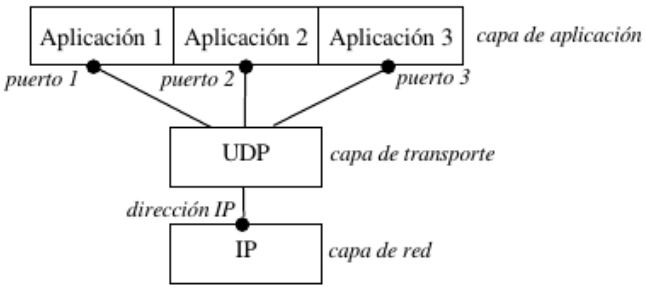
- *Multiplexación/demultiplexación* de aplicaciones.
- Servicio *orientado a conexión,fiable*: Control de *errores y de flujo*. Control de la *conexión*. Control de *congestión*.

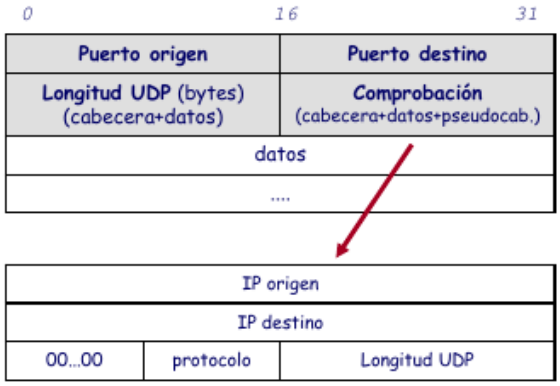
2. Protocolo de datagrama de usuario(UDP)

User Datagram Protocol (UDP)

Funcionalidad "*best effort*":

- Servicio *no orientado a conexión*.
- Servicio *no fiable*.
- *No* hay garantías de *entrega ordenada*.
- *No* hay *control de congestión*: entrega tan rápida como se pueda.
- *Multiplexación/demultiplexación*: transportar las TPDU al proceso correcto





Longitud UDP nos da el valor de la cabecera + el payload.

payload -> donde tenemos toda la información de la capa de aplicación.
comprobación(checksum) -> nos permite comprobar ciertas cosas. En la comprobación se meten las IP para que se compruebe que si ha ocurrido algún error de que ese paquete no iba dirigido a ti.

El checksum se calcula en UCP, en TCP, es el complemento a 1 de la la suma complemento a 1 de las palabras de 16 bits.

Multiplexación/demultiplexación: transportar las TPDU al proceso correcto.

Ejemplos de puertos UDP preasignados

Puerto	Aplicación/Servicio	Descripción
53	DNS	Servicio de nombres de domino
69	TFTP	Transferencia simple de ficheros
123	NTP	Protocolo de tiempo de red
161	SNMP	Protocolo simple de administración de red
520	RIP	Protocolo de información de encaminamiento

UDP se usa frecuentemente para **aplicaciones multimedia**: tolerantes a fallos y sensibles a retardos.

Cada segmento UDP se **encapsula** en un datagrama IP.

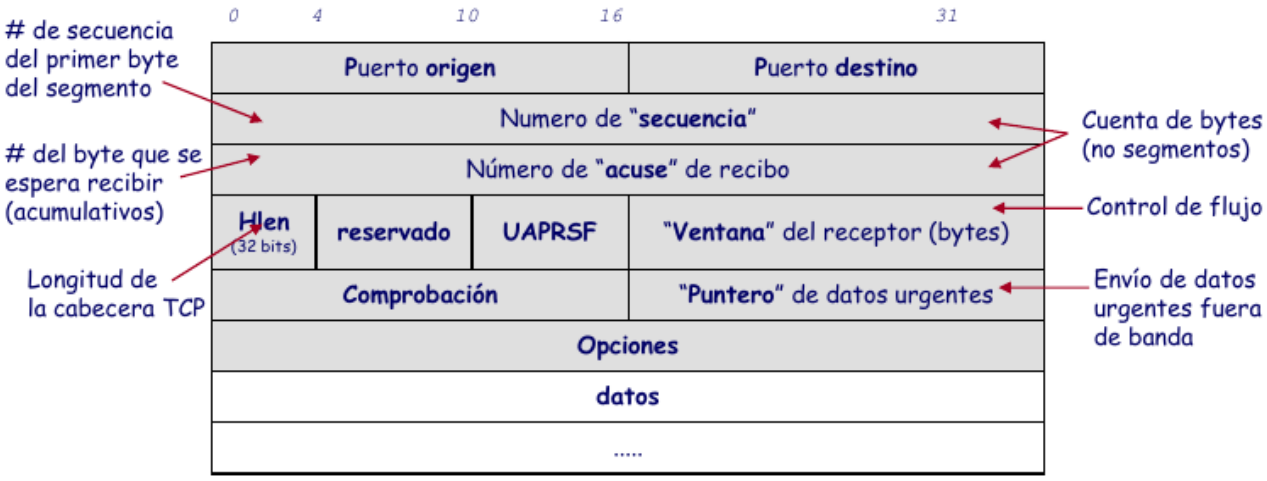
3. Protocol de control de transmisión(TCP)

Características del "Transmission Control Protocol"

- Servicio *orientado a conexión*.["Hand-shaking"]
- *Entrega ordenada*.
- *Full-duplex* : se recibe y se envía en el mismo canal.
- Mecanismo de control de flujo y detección y recuperación de errores:
  - confirmaciones positicas (*ACKs*) y acumulativas: paquetes de control que confirman la correcta llegada. Esto se combina con temporizadores, si se agota y no hay confirmación se vuelve a enviar.
  - incorporación de confirmaciones("*piggybacking*"): se intenta maximizar el número de paquetes que se envían de control.
  - *timeouts* adaptables.
  - *Ventanas* adaptables : puedo utilizarlas para enviar varios segmentos, y cuando los han confirmado todos, envío otro pack de segmentos.
- Mecanismo de control de congestión.
- *Servicio fiable* : control de congestión y control de flujo.

LA CABECERA DE TCP SUELE CAER EN EL EXÁMEN

TDPU TCP = **Segmento TCP**:



- *Número de secuencia* : por donde veo lo que yo he enviado.
- *Reservado* : para investigación.
- *UAPRSF* : campo de flags. Se mira para conmprobar el resto de los campos.
- *Puntero de datos urgentes* : permite diferenciar en el **payload** los datos que merecen más prioridad que otros.

El campo de opciones es variable, muchas veces en el comienzo de una comunicación se utilizan para negociar ciertos campos de la comunicación.

Multiplexación/demultiplexación de aplicaciones:

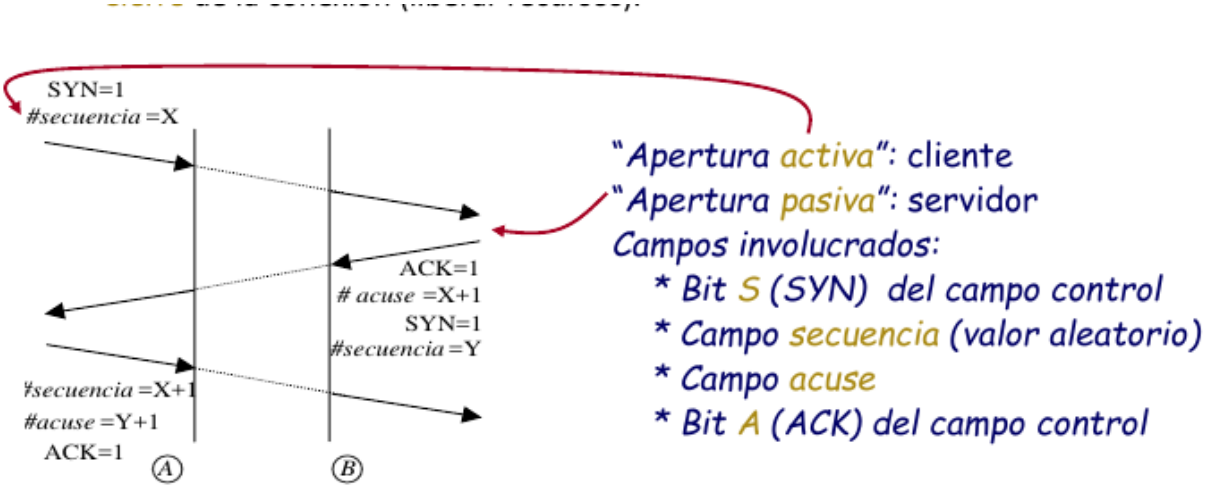
Puerto	Aplicación/Servicio	Descripción
20	FTP-DATA	Transferencia de ficheros: datos
21	FTP	Transferencia de ficheros: control
22	SSH	Terminal Seguro
23	TELNET	Acceso remoto
25	SMTP	Correo electrónico
53	DNS	Servicio de nombres de domino
80	HTTP	Acceso hipertexto (web)
110	POP3	Descarga de correo

La "conexión TCP" se identifica por: puerto e IP origen y puerto e IP destino (y opcionalmente el protocolo).

Control de la conexión

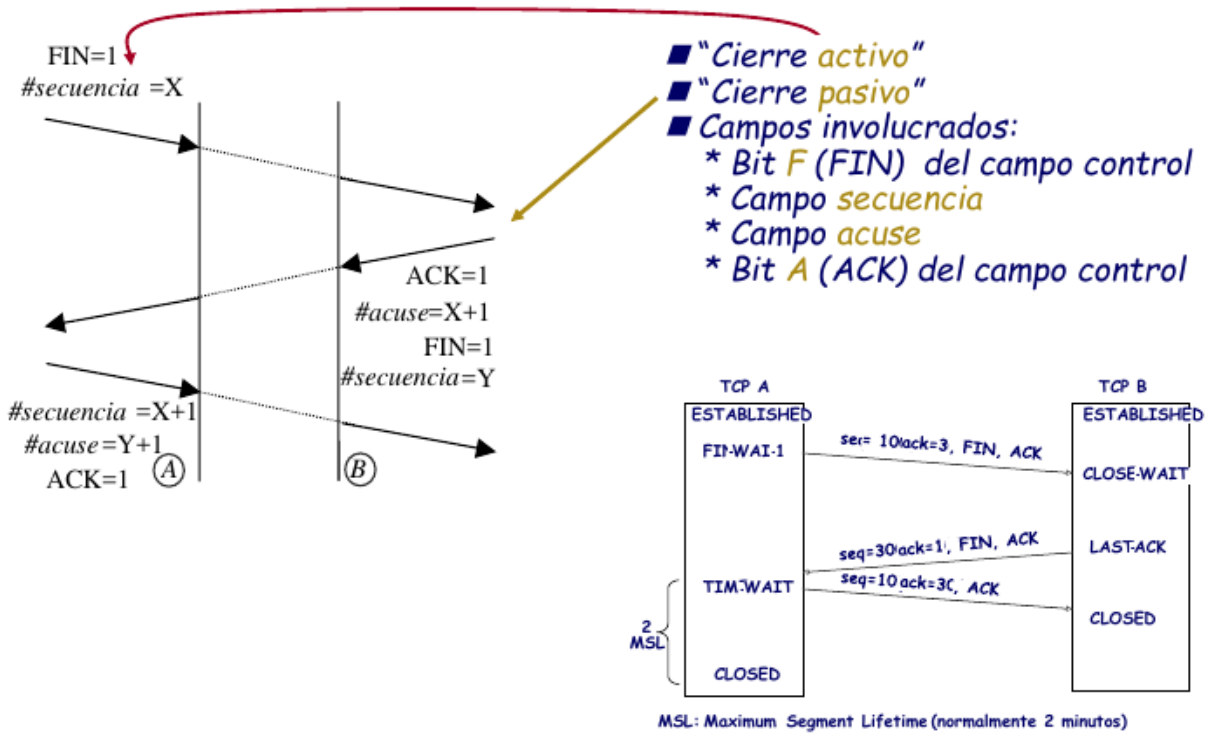
El intercambio de información tiene **tres fases**:

- *1º paquete* : se le pone el flag de sincronización a 1. El nº de secuencia se inicializará a un número pseudoaleatorio y se manda. Este llega a un puerto del servidor.
- *2º paquete* : devuelve ACK = 1. Como tenemos conexión full-duplex el servidor también pone el SYN = 1 e inicializa el nº de secuencia a este número pseudoaleatrio. El acuse se pone a x +- 1.
- *3º paquete* : secuencia = X + 1; acuse = Y + 1.; ACK = 1. Cierre de la conexión (liberar recursos).



Cierre de la conexión, liberación de recursos

El que quiere desconectar manda un segmento con el flag FIN=1. Cuando el segmento llega al otro confirma, si quiere, y pone también el flag FIN=1 y lo mandará. Y finalmente se confirma ese segmento. El siguiente segmento será con Y+1. Espera cin el temporizador ya que puede que no todos los paquetes hayan llegado, ni que el otro haya cerrado, por lo tanto se queda un ratito escuchando por si acaso.



Control de conexión. Números de secuencia

- Campo de 32 bits : 2^32 valores. Cuando en un flujo TCP se agotan los valores se da la vuelta.
- Inicialización(#secuencia = ISN)
    - Empieza en el *ISN*(Initial Sequence Number), elegido por el sistema.
    - Para el ISN, el estándar sugiere utilizar un contador entero incrementado en 1 cada 4 microsegundos aproximadamente.
    - El ISN no se inicializa a 0 para evitar confluencias de flujos.
    - Protege de coincidencias, pero no de sabotajes.
- **Número de secuencia** aumenta el número de bytes en el payload.
- Incremento (#secuencia = #secuencia + Nbytes)
    - Se incrementa según los bytes de carga útil (payload).
    - Los flags *SYN* y *FIN* incrementan en 1 el número de secuencia.

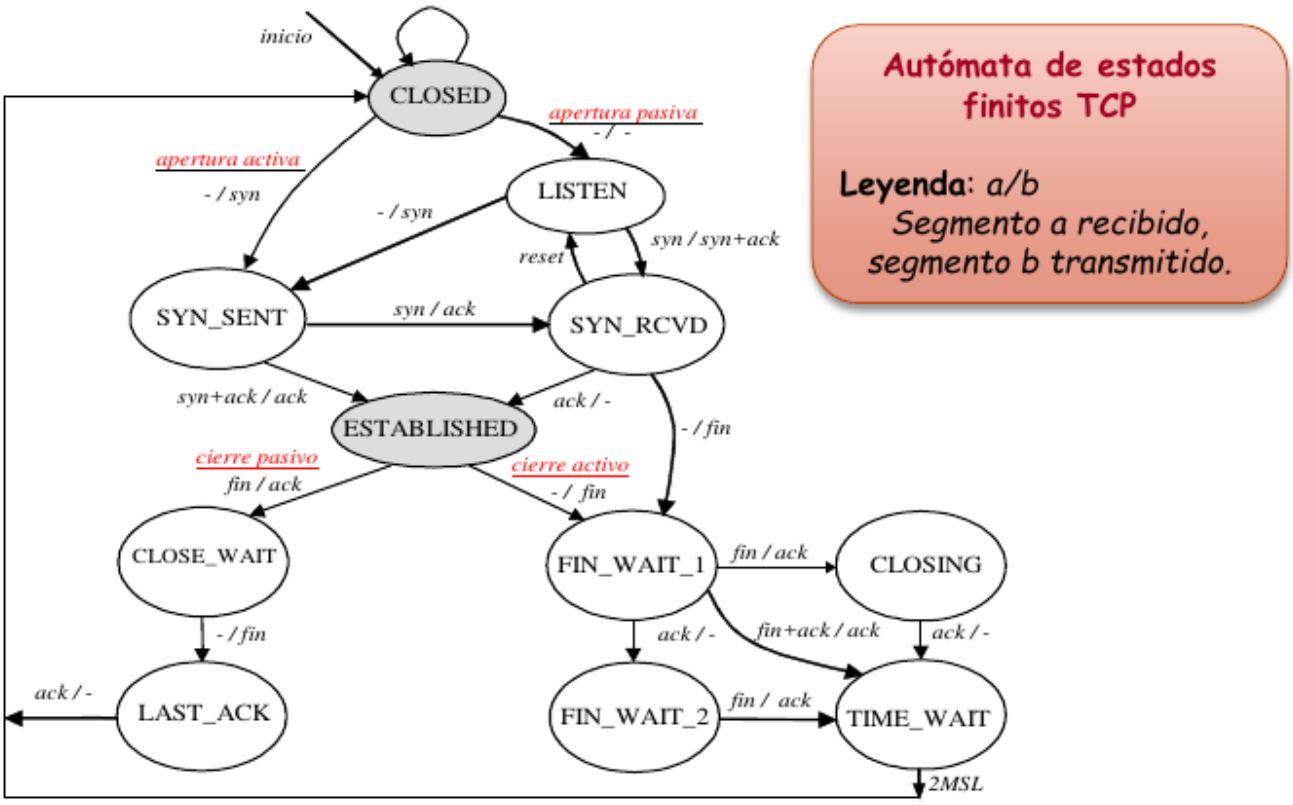
Cuando se conecta:

#secuencia = x -> se obtiene del ISN. Se saca por como aumenta el contador cada 4 microsegundos.

Cuando se desconecta:

#secuencia = x -> la x por a que empezó más todos los bytes que se han transferido.

Lo mismo para la Y. Se acaba en el número de secuencia + 1 por el flag FIN.

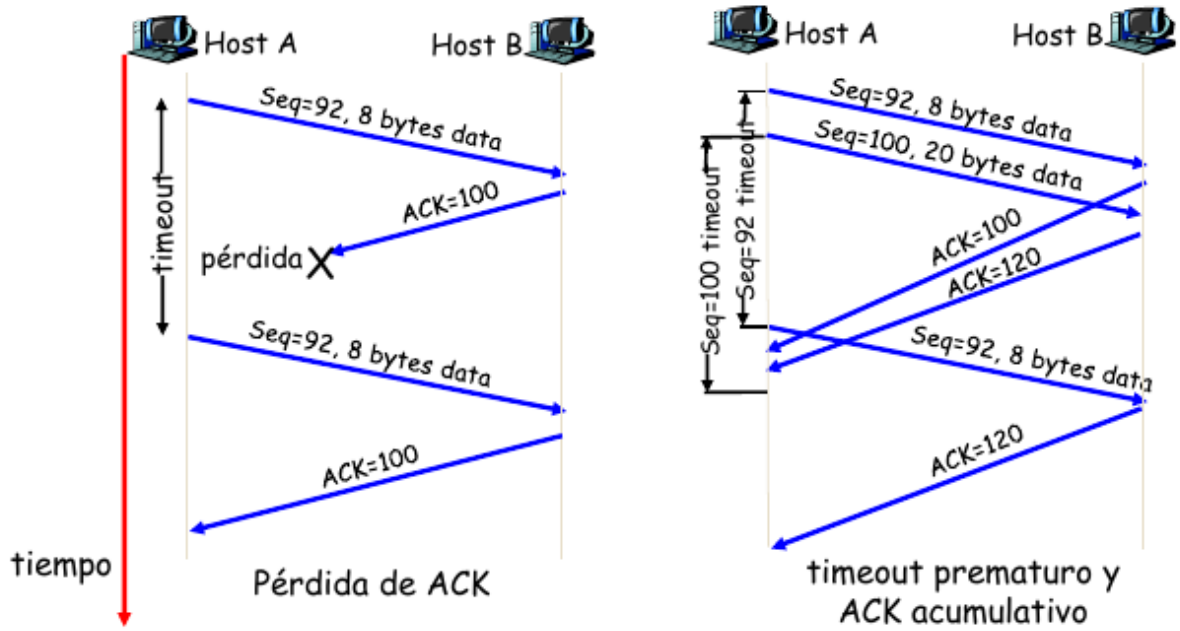


Control de errores y de flujo

Hacen reducir la velocidad. *Control de errores*: esquema ARQ con confirmaciones positivas y acumulativas. Campos involucrados:

- Campo secuencia: offset (en bytes) dentro del mensaje.
- Campo acuse: número de byte esperado en el receptor.
- Bit A (ACK) del campo de control.
- Campo comprobación: checksum de todo el segmento y uso de pseudo-cabecera.

Escenarios de retransmisión:



El segundo es una ventana de dos segmentos. Vuelve a enviar el ACK = 120 porque este contiene el ACK = 100, ya que son acumulativos. Generación de ACKs:

Evento	Acción del TCP receptor
Llegada ordenada de segmento, sin discontinuidad, todo lo anterior ya confirmado.	Retrasar ACK. Esperar recibir al siguiente segmento hasta 500 mseg. Si no llega, enviar ACK.
Llegada ordenada de segmento, sin discontinuidad, hay pendiente un ACK retrasado.	Inmediatamente enviar un único ACK acumulativo.
Llegada desordenada de segmento con # de sec. mayor que el esperado, discontinuidad detectada.	Enviar un ACK duplicado, indicando el # de sec. del siguiente byte esperado.
Llegada de un segmento que completa una discontinuidad parcial o totalmente.	Confirmar ACK inmediatamente si el segmento comienza en el extremo inferior de la discontinuidad.

¿Cómo estimar los timeouts?

- Mayor que el tiempo de ida y vuelta (RTT).
- Si es demasiado pequeño: timeouts prematuros.
- Si es demasiado grande: reacción lenta a pérdida de segmentos.
- Para situaciones cambiantes la mejor solución es la adaptable, es decir, dependiendo de cómo esté la red.

Las condiciones de una red son cambiantes. Si la red cambiantes. Si la red cambia un temporizador que me vale ahora puede ser que no valga en 10 minutos. Necesito adaptarme a la red lo más rápido posible.

Kurose & Ross

RTTmedido: *tiempo desde la emisión de un segmento hasta la recepción del ACK.*

$$RTTnuevo = (1-\alpha) \times RTTviejo + \alpha \times RTTmedido \ , \ \alpha \ \& \ \beta \in [0,1]$$

$$Desviacion_{nueva} = (1-\beta) \times Desviacion_{vieja} + \beta \times | RTTmedido - RTTnuevo |$$

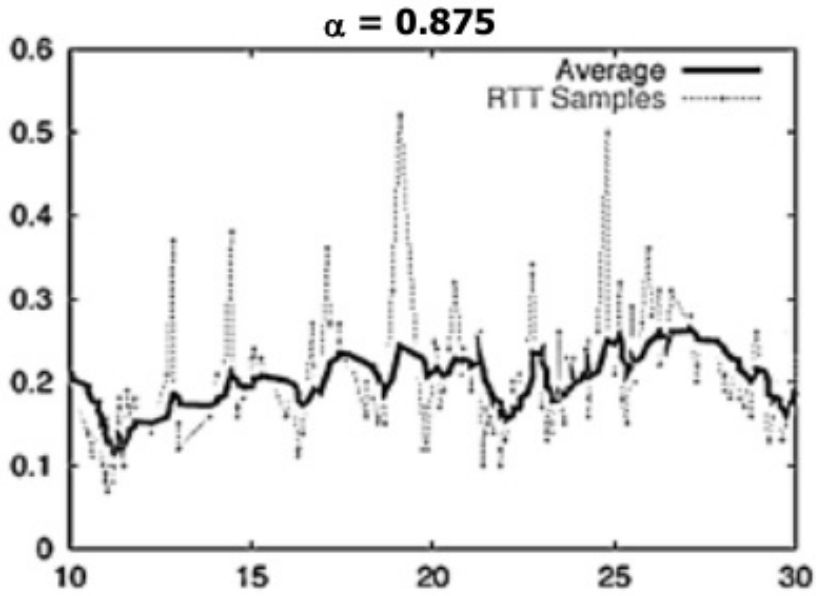
$$Timeout = RTTnuevo + 4 * Desviacion$$

Problema con ACKs repetidos: ambigüedad en la interpretación.

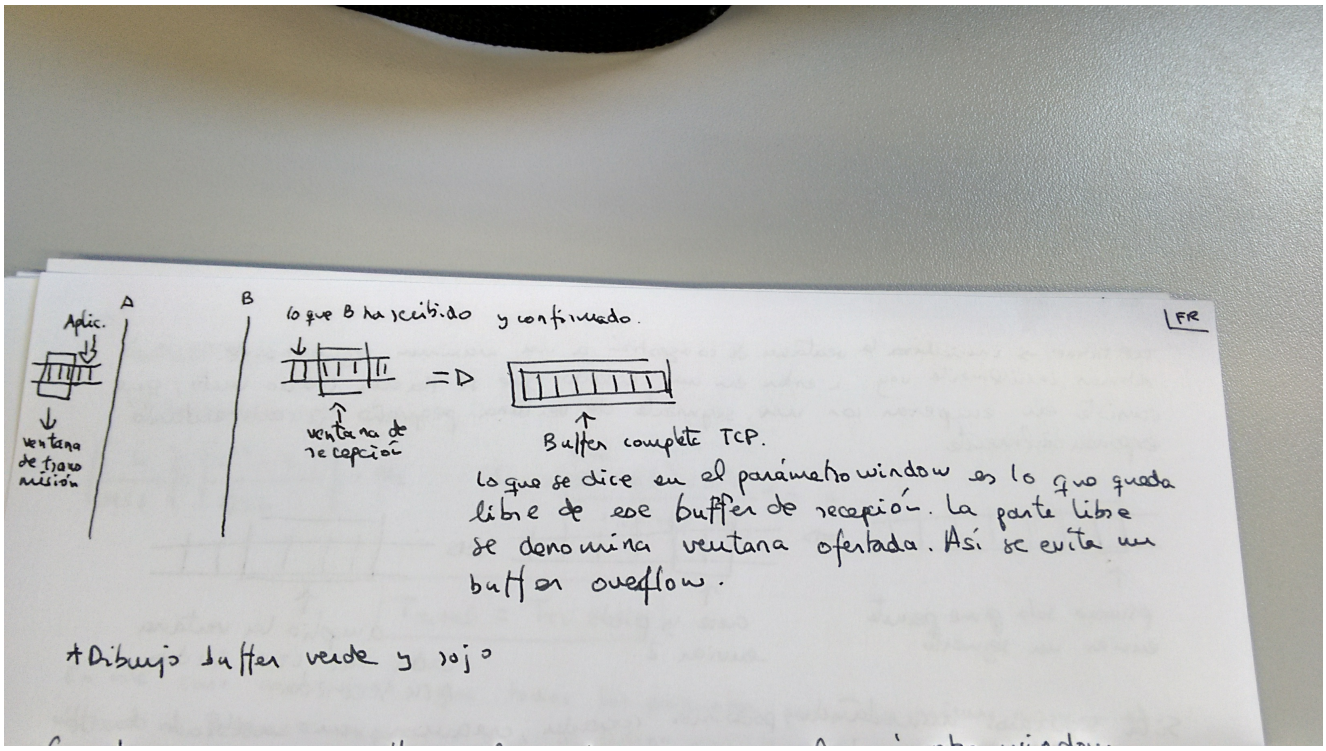
Solución: Algoritmo de Karn:

- *actualizar el RTT sólo para los no repetidos*
- si hay que repetir un segmento incrementar el timeout: $tout_{nuevo} = \gamma . tout_{viejo}$, $\gamma = 2$.

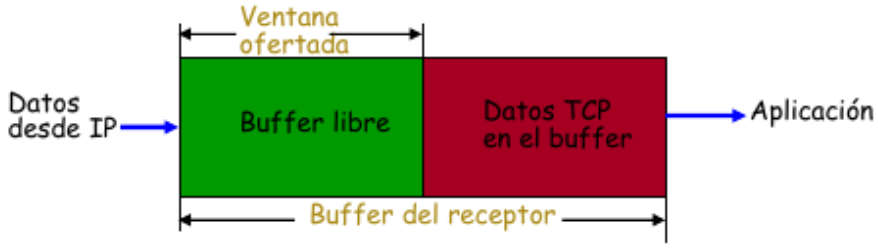
Ejemplo de RTT medidos y estimados entre Amherst, Massachusetts y St. Louis, Missouri.



Control de flujo: sirve para no saturar al receptor. Aunque a la ve que hace eso controla la recepción en orden de paquetes. No se manda un segmento cada vez, es decir, por cada segmento no esperamos al ACK, creamos ventanas. COn las ventanas enviamos más de un segmento a la vez. Así reducimos el tiempo muerto y aumentamos el tiempo activo. Las confirmaciones son por paquete, ya que si no tendría que volver a enviar la ventana completa, y aumentaría el tiempo muerto otra vez. El control de flujo de TCP es un sistema crediticio, con esto se intenta no saturar al receptor. Con los créditos que nos da, nos indica cuánto le podemos enviar. Esto lo da el parámetro window. Tiene relación con la comunicación de B a A. Lo que dice el parámetro window es lo que queda libre de ese buffer de recepción. La parte libre se denomina ventana ofertada. Así se evita un buffer overflow.



En el receptor (paso 1):

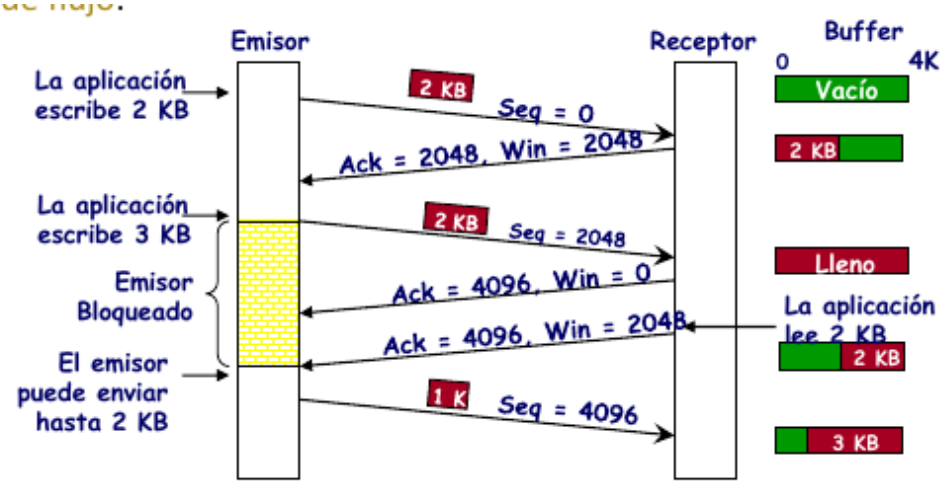


En el emisor (paso 2):

ventana útil emisor = ventana ofertada receptor – bytes en tránsito

Cuando ese mensaje llega al emisor, se coge el parámetro window y se restan los bytes en tránsito.

Control de flujo:



Posible problema: *síndrome de la ventana tonta (RFC 813) si se utilizan segmentos muy pequeños.*

Posible mejora: *la ventana optimista* (*RFC 813*)

ventana útil emisor = ventana ofertada receptor

Es posible hacer entregas "no ordenadas": Bit *U* (URG), campo *puntero.*

Solicitar una entrega inmediata a la aplicación: bit *P* (PSH).

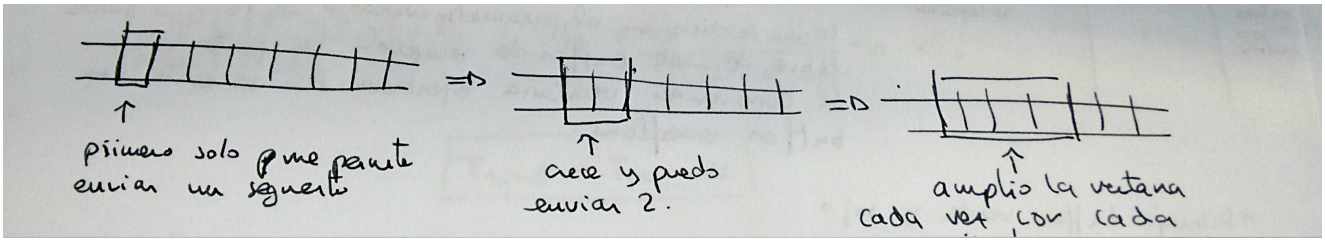
Podemos observar como se llena el buffer del receptor y el cliente queda bloqueado, hasta que el receptor envía un ACK duplicado con el mismo ACK number con el windows que queda libre. En el three-way handshake si dice el tamaño de buffer que se usa. Normalmente el buffer se va a atomizar mucho, y se van a enviar segmentos muy pequeños. Esto hace que la proporción de cabeceras con las de datos sea muy descompasada.

Para ello tenemos la ventana optimista, donde si tengo una buena sincronización con el receptor, dejámos de considerar los bytes en tránsito y así reducimos el tamaño de la ventana. Una solución por parte del receptor sería que no te envíe su ventana hasta que tenga suficiente tamaño. Con los datos urgentes tiene sentido en el receptor, este se salta la cola de recepción y los envía a la capa de aplicación directamente. Cuando una información tiene que enviarse se pone el bit P (PSH) a 1 para que el receptor lo trate como datos urgentes. Se utiliza más que la anterior.

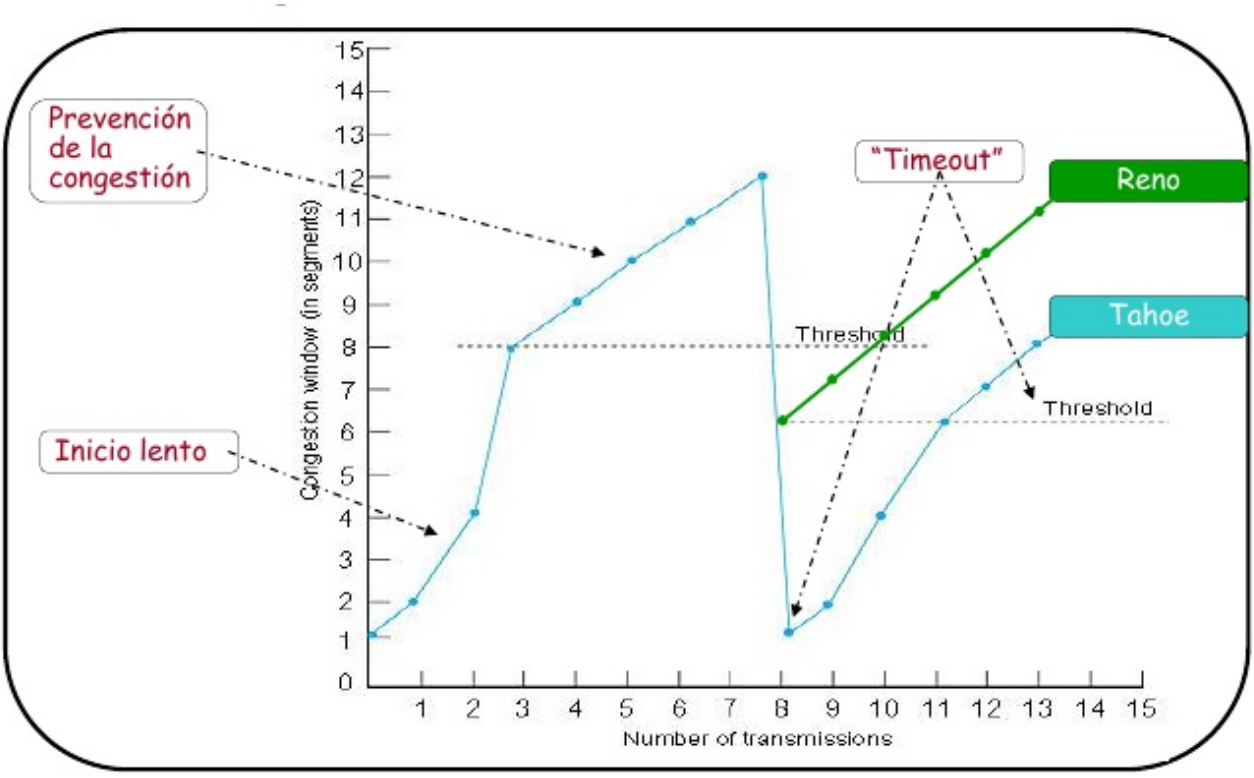
Control de congestión: Trabaja en colaboración con el control de errores y de flujo. Mismo objetivo que el de flujo pero aquí cambia el agente. No hay nadie que diga que la red está congestionada, tengo que darme cuenta. Una forma de ver si hay congestión, es ver si se ha perdido ACK. Si se excede el timeout significa que seguramente la red está congestionada. TCP Tahoe -> inicializa la ventana de congestión a un maximun segment size. Además, inicialmente voy a estar en un estado que se llama inicio lento, que consiste en empezar con un segmento de ventana pequeño y aumentarlo exponencialmente.

Si *VCongestion* < *Umbral*, por cada ACK recibido
VCongestion += MSS **(crecimiento exponencial)**
Inicio lento

Si *VCongestion* > *Umbral*, por cada ventana completada (todos ACKs recibidos)
VCongestion += MSS (**crecimiento lineal**)
Prevención de la congestión



Si la seguimos aumentando podemos cagarla, creamos un umbral donde empieza a crecer de forma lineal. Por cada vez que recibo la ventana completa, aumento un MSS(va aumentando de uno en uno).



Combinación de CTrl de flujo y congestión:

Combinación del Ctrl de flujo y congestión:

El emisor elige el mínimo de las ventanas correspondientes:

```
Bytes_permitidos_enviar =
            min{VCongestion,VentanaOfertadaReceptor}
```

```
ventana útil emisor = Bytes_permitidos_enviar - bytes en tránsito
```

En cada momento sabes el window y no puedes enviar más de lo que te diga, así el control de congestión se eleva y al final cogemos la ventana más restrictiva.

4.Extensiones TCP.

- TCP se define con múltiples *sabores*.
- Los diferentes sabores no afectan a la *interoperabilidad* dentre extremos.
- Desde cualquier versión de Linux con kernel mayor que la 2.6.19 se usa por defecto *TCP CuBIC*

Adaptación de TCP a redes actuales (RFC 1323, 2018).

Ventana escalada:

Opción TCP en segmentos SYN:

Hasta $2^{14}x2^{16}$ bytes (=2^{30} bytes=1GB) autorizados.

Estimación RTT:

Opción TCP de *sello de tiempo*, en todos los segmentos.

PAWS ("Protect Against Wrapped Sequence numbers"):

Sello de tiempo y rechazo de segmentos duplicados.

SACK:

Confirmaciones selectivas.