

CAPA DE TRANSPORTE

SERVICIOS CAPA DE TRANSPORTE

Facilita comunicación lógica entre procesos (aplicaciones) corriendo en diferentes hosts.

Los protocolos de transporte se ejecutan en los sistemas finales

→ LADO EMISOR: trocea/agrega mensajes en segmentos, que pasa a la capa de red.

→ LADO RECEPTOR: reensambla segmentos que pasa a la capa de apli

TRANSPORTE VS RED

Red: comunicación lógica entre hosts no adyacentes.

Transporte: comunicación lógica entre procesos.

DIFERENTES PROTOCOLOS

- Confiable, recepción en orden → TCP
 - control de flujo
 - control de congestión
 - requiere setup inicial
- No confiable, recepción desordenada → UDP (extensión "best effort" IP)

Servicios no implementados:

- garantía de retardos
- garantía de ancho de banda

MULTIPLEXACIÓN/DEMULTIPLEXACIÓN

- Multiplexación en host que envía: encapsular el tráfico convenientemente y enviarlo al nivel de red.
- Demultiplexación en el host receptor: distribuir los segmentos recibidos al socket correcto.

⊗ En general es posible usar direcciones IP y n° de puerto para identificar un socket.

DEMÚLTIPLEXACIÓN SIN CONEXIÓN (UDP)

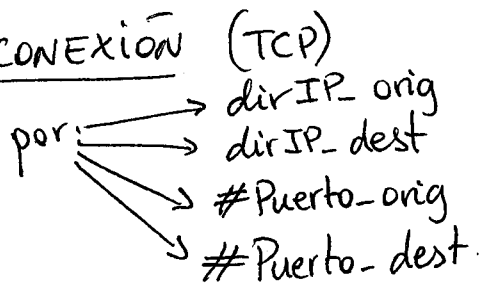
Una vez asignados los puertos podemos describir de forma precisa las tareas de multiplexación y demultiplexación en UDP.

Es importante ver que un socket UDP queda completamente identificado por una tupla (dirIP-dest , \#Puerto-dest). Aunque tengan diferentes dirIP-orig y/o \#Puerto-orig el segmento irá al mismo socket.

El \#Puerto-orig solo sirve como identificación del socket de retorno.

DEMÚLTIPLEXACIÓN ORIENTADO A LA CONEXIÓN (TCP)

Un socket TCP queda identificado por:
El host receptor usa estos 4 valores para direccionar al socket correcto.



UDP: USER DATAGRAM PROTOCOL

Servicio "Best Effort". Los segmentos UDP pueden ser $\begin{matrix} \nearrow \text{perdidos} \\ \searrow \text{entregados fuera de orde} \end{matrix}$

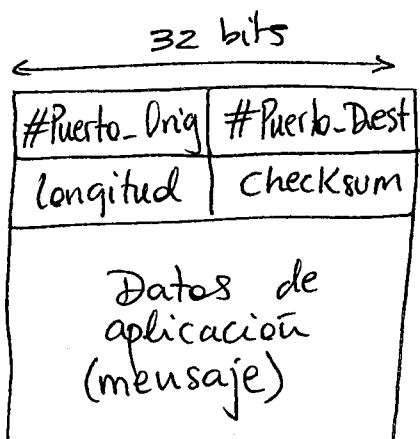
Sin conexión: no establecimiento de la conexión emisor/receptor

Cada segmento UDP es manejado independientemente al resto y como entidad

¿POR QUÉ UDP?

- En TCP el establecimiento de la conexión añade retardo
- Cabecera más pequeña
- Cuando el retardo supera en importancia a las pérdidas
- Sin control de flujo/congestión

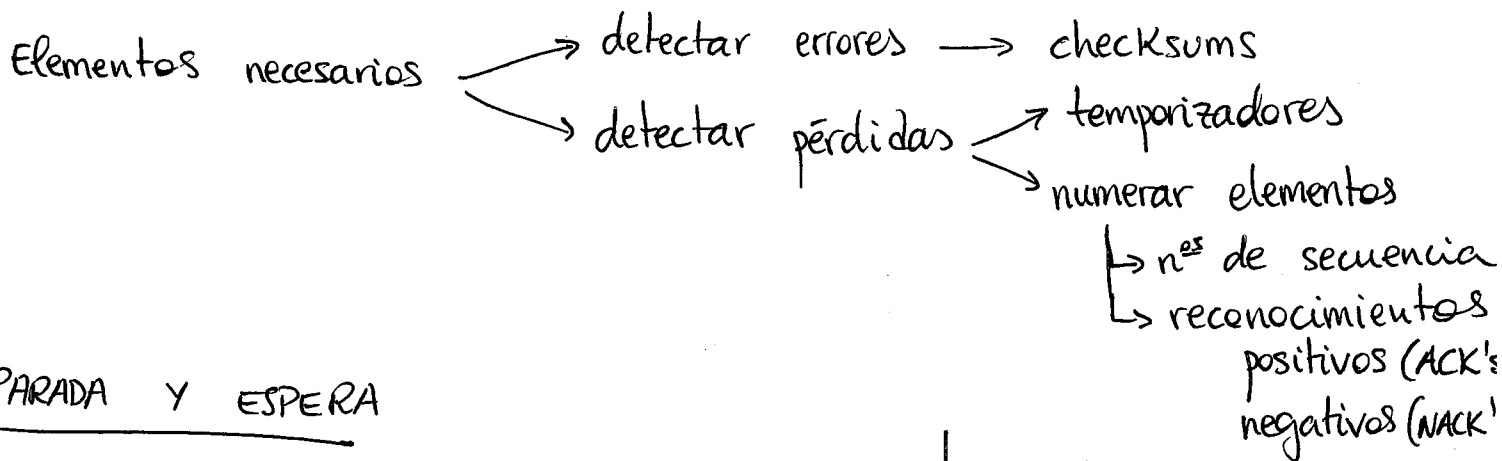
→ Usos $\begin{matrix} \nearrow \text{aplicaciones multimedia (tolerantes a las pérdidas y sensibles al caudal/latencia).} \\ \searrow \text{DNS} \end{matrix}$



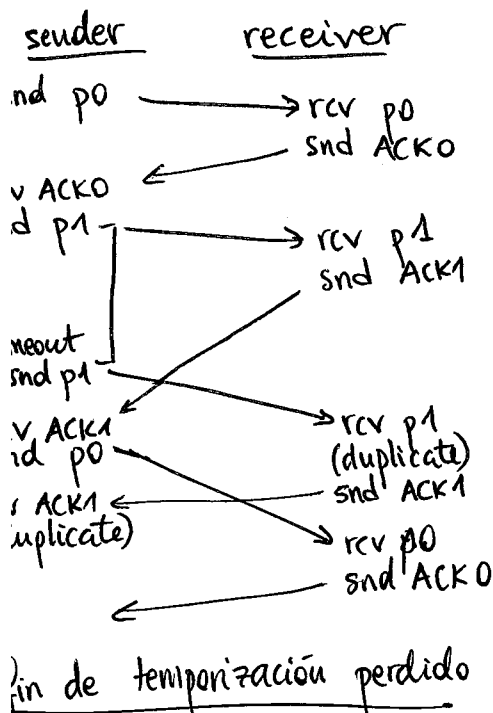
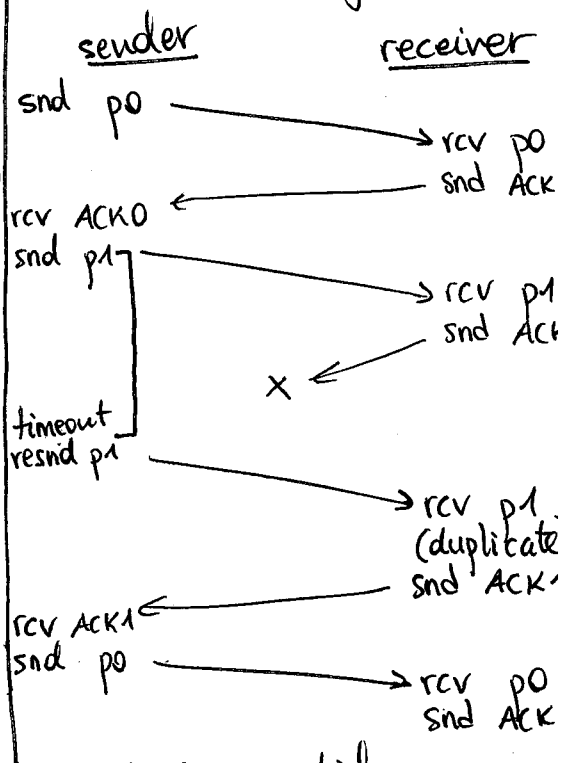
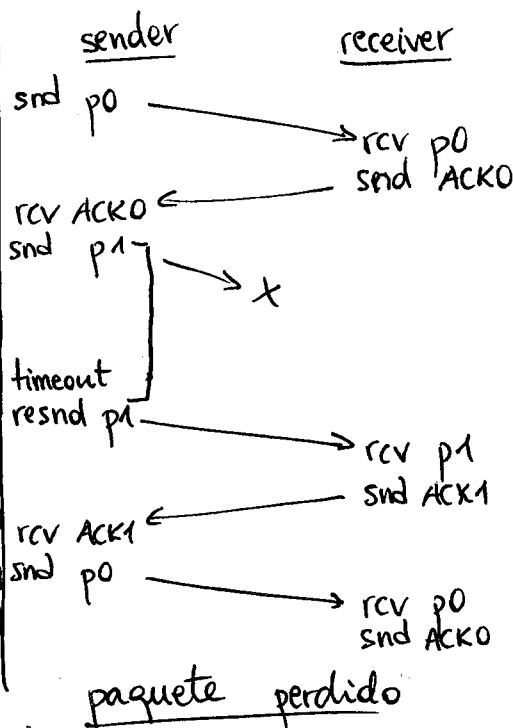
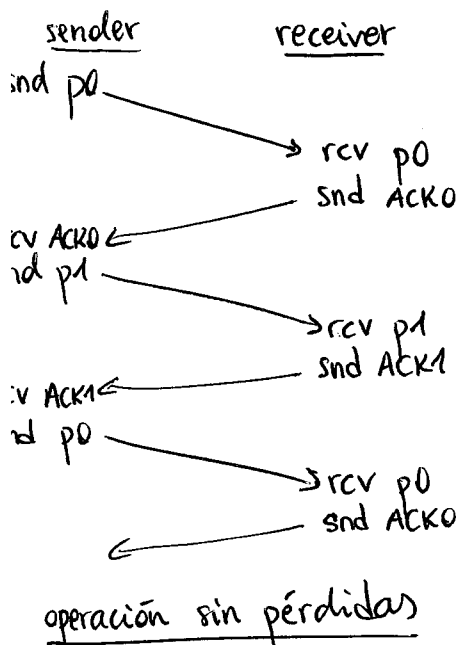
Longitud: longitud en bytes del segmento UDP incluyendo cabecera

Checksum: datos + cabecera

TRANSFERENCIA FIABLE



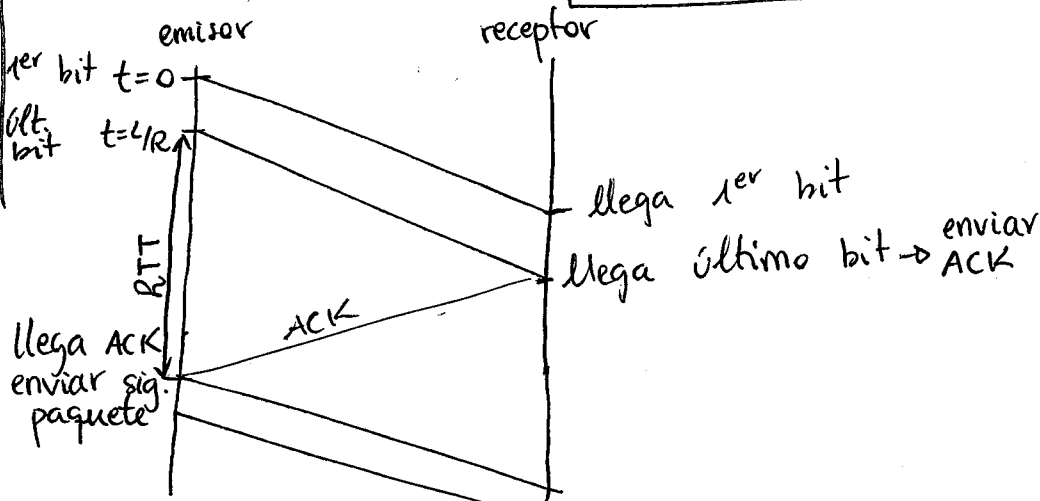
PARADA Y ESPERA



Parada y espera funciona pero mal rendimiento
 RTT: retardo de propagación de ida y vuelta
 R: velocidad de transmisión
 L: tamaño paquete

$$d_{trans} = \frac{L}{R}$$

$U_{emisor} := \text{utilización emisor} \Rightarrow U_{emisor} = \frac{L/R}{RTT + L/R}$



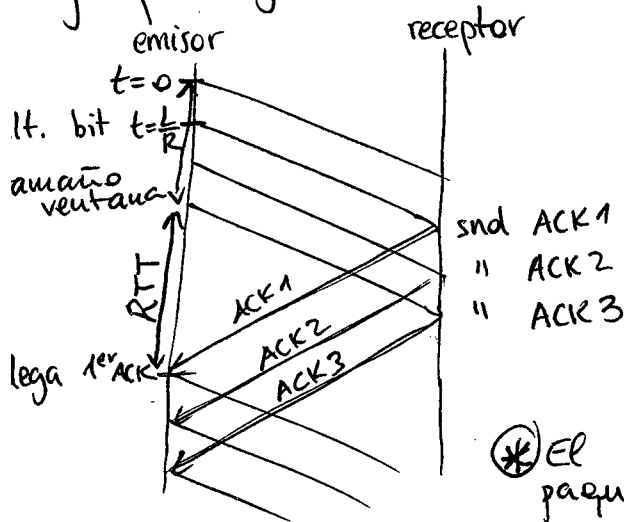
PROTOCOLOS PIPELINED (procesamiento en cadena)

El emisor permite múltiples envíos "al vuelo" antes de llegar los correspondientes ACK.

El número de secuencia debe abarcar más bits e incrementarse (== ventana deslizante).

Se requiere buffer en emisor y/o receptor

Ejemplos: go-Back-N (GBN), repetición selectiva, ...



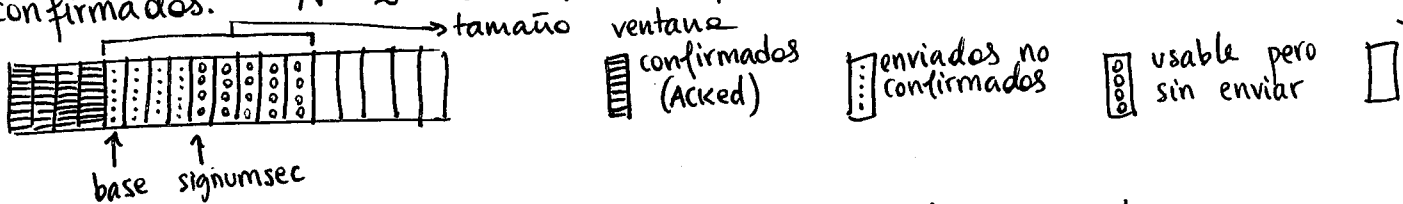
$$U_{em} = \frac{3 * L/R}{RTT + L/R}$$

$$U_{emisor} = \frac{\text{tamVentana} * L/R}{RTT + L/R}$$

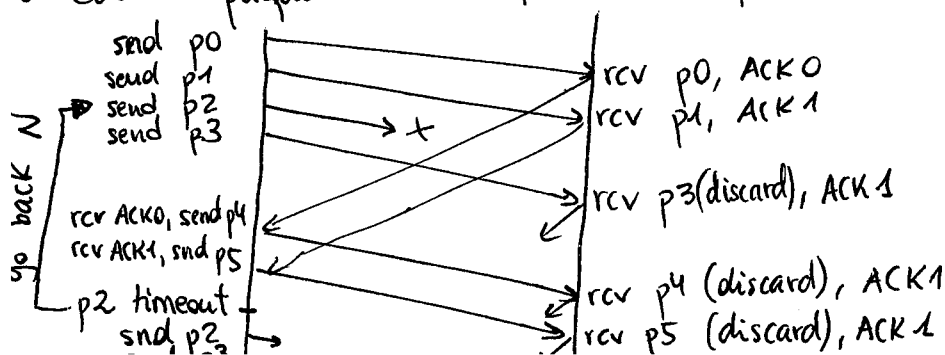
* El tamaño de la ventana es el nº de paquetes que se pueden enviar sin confirmar 3 en el ejemplo
go-Back-N

VENTANA DESLIZANTE (GBN)

- Número de secuencia limitado (m bits para el nº de secuencia)
- Tamaño de la ventana deslizante hasta un máximo de N segmentos no confirmados. $N = 2^m - 1 \Rightarrow N = |\text{nº de secuencia}| - 1$.



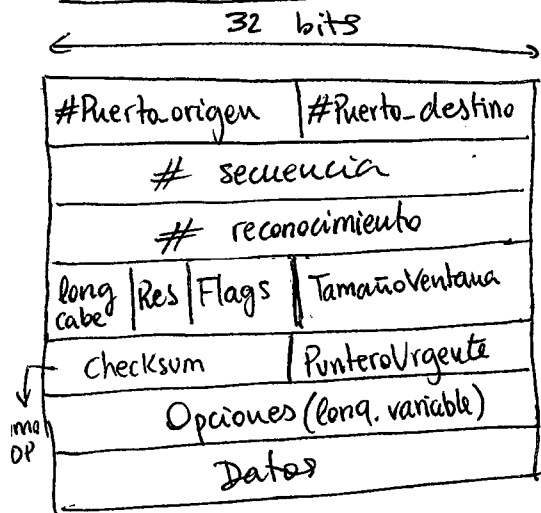
- ACK(n) es acumulativo: se confirman todos los paquetes con nº de secuencia menor o igual a n (buffer de recepción igual a 1).
- El extremo receptor en caso de pérdidas/desorden devuelve la confirmación del último segmento recibido en secuencia
- Cada paquete es confirmado por su respectivo ACK.



TCP: Transmission Control Protocol

- Punto a punto: 1 emisor 1 receptor
- Flujo de bytes ordenado (confiable)
- El control de ventana es fijado por el control de flujo y congestión
- Buffers en ambos extremos
- Orientado a la conexión: handshaking (intercambio de mensajes de control acuerdo en tres fases) para inicialización.
- MSS: Maximum segment size (cantidad máxima de datos de la capa de aplic.

ESTRUCTURA SEGMENTO TCP



Los n° de secuencia/reconocimiento hacen referencia al flujo de bytes transmitidos

datos 1st segm. datos 2º segm.

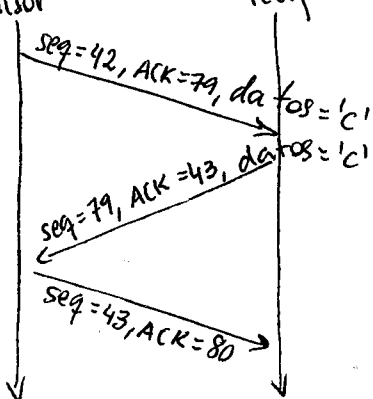
0	1	...	1000	1001	...	1999	...
---	---	-----	------	------	-----	------	-----

Número Secuencia: Bytes ya enviados desde ese extremo

Número Reconocimiento (ACK): Bytes recibidos hasta ese momento en ese extremo

▷ Reconocimiento acumulativo

emisor receptor



Nº secuencia (32 bits): identifica el byte del flujo de datos enviado por el emisor TCP que representa el primer byte de datos del segmento

Nº reconocimiento (32 bits): contiene el valor siguiente del nº de secuencia que el emisor espera recibir.

LongCabecera (4 bits): longitud cabecera en palabras 3

Reservado (3 bits): sin uso actual, debe estar a cero.

Flags (9 bits):

- NS
- CWR: congestion window reduced
- ECE: para indicar sobre congestión
- URG: el puntero urgente es válido (adelantar buffer)
- ACK: nº de reconocimiento válido
- PSH: el emisor/receptor debe pasar los datos tan rápido como pueda a la aplicación/nvl inferior
- RST: reset de la conexión.
- SYN: sincronizar nº de secuencia
- FIN: el emisor no tiene más datos que manda

Tamaño Ventana (16 bits): nº máx. de bytes que pueden ser metidos en el buffer de recepción, i.e., nº máx. de bytes pendientes de asentimientos.

Puntero Urgente (16 bits): cantidad de bytes desde el nº de secuencia que indica el lugar donde acaban los datos urgentes.

TCP RTT Y TIMEOUT

¿Cómo fijar el timeout?

¿Cómo estimar el RTT?

↳ Con un $RTT_{muestra}$ (medir t desde envío hasta recepción ACK) → $RTT_{muestra}$ varía ⇒ promedio

objetivo: mayor RTT (pero este varía)

↳ muy pequeño: retransmisiones innecesarias

↳ muy grande: lenta reacción a pérdida

FIABILIDAD EN LA TRANSMISIÓN DE DATOS.

TCP da un servicio de transferencia fiable sobre un canal que no lo es.

TCP usa un único temporizador de transmisión (salvo que se especifique lo contrario)

Las retransmisiones suceden cuando $\begin{cases} \rightarrow \text{se cumple el temporizador} \\ \rightarrow \text{reconocimientos duplicados} \end{cases}$

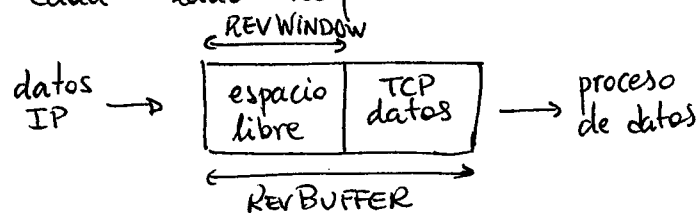
- Aplicación envía datos: TCP crea un segmento y transmite el segmento. Arranca un temporizador si es que no estuviera activado.
- Se cumple el temporizador: se retransmite el segmento del timeout y se reinicia el temporizador
- Segmento fuera de secuencia: el extremo receptor devuelve información del último segmento confirmado y, si el n° de secuencia es mayor al esperado, guarda el segmento.
- Se recibe ACK: se confirman los segmentos previos. Si quedan segmentos por confirmar se arranca el temporizador.

Concepto de RETRANSMISIÓN RÁPIDA:

- Los temporizadores pueden resultar largos: hay que esperar al temporizador y después reenviar.
- Por ejemplo: uso de ACKs duplicados \rightarrow si el receptor recibe 3 ACKs duplicados asume que el siguiente se ha perdido \Rightarrow
 \Rightarrow RETRANSMISIÓN RÁPIDA: envía el siguiente ignorando el temporizador

CONTROL DE FLUJO

Cada lado receptor de TCP tiene un buffer de recepción.



Control de flujo: el emisor no debe sobrepasar el buffer del receptor al transmitir demasiado rápido

El proceso de aplicación puede ser lento en leer del buffer (segm. ACK'ed)

\Rightarrow Servicio de adaptación de la velocidad: adaptar la tasa de envío a la tasa de lectura de la aplicación.

¿Cómo funciona?

El receptor advierte del espacio libre incluyendo el valor de la ventana de recepción (RevWindow).

El emisor limita el volumen de datos enviados pero no confirmados a este tamaño de ventana.

\rightarrow Se garantiza que el buffer no se desborde

\rightarrow Si la ventana es 0 es posible enviar paquetes igualmente

\rightarrow ventana tonta.

GESTIÓN DE LA CONEXIÓN

Recuerda: el emisor debe establecer la conexión antes de intercambiar segmentos con datos

Importante: inicializar n° de secuencia, buffers, variables de control de flujo

Acuerdo en 3 fases:

- PASO 1: El cliente envía un segmento TCP SYN al receptor, especificando n° seq. inicial (Sin datos).
- PASO 2: El servidor recibe el SYN, responde con un segmento SYNACK. Además reserva los buffers y especifica el n° seq. inicial en este extremo.
- PASO 3: El cliente recibe SYNACK y responde con un segmento ACK, que puede ya contener datos.

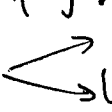
Protocolo de cierre de conexión:

- PASO 1: El cliente envía un segmento TCP con la bandera FIN.
- PASO 2: El servidor recibe FIN, responde con un flag ACK. Cierra conexión mandando antes un FIN.
- PASO 3: El cliente recibe FIN, responde con ACK.
 - ↳ Se entra en un temporizador donde se responderá con ACK's a los posibles FIN's.
- PASO 4: El servidor recibe el ACK. Conexión cerrada

Nota: pequeñas modificaciones para FIN's simultáneos y casos de error.

PRINCIPIOS DE CONTROL DE CONGESTIÓN

Congestión: Informalmente demasiadas fuentes enviando demasiados datos/demasiado rápido para que la red pueda manejarlos.

⊛ Es diferente al control de flujo!
¿Cómo nos damos cuenta? 

CAUSAS/COSTO DE LA CONGESTIÓN

Ahora analicemos unos escenarios descriptivos de la congestión.

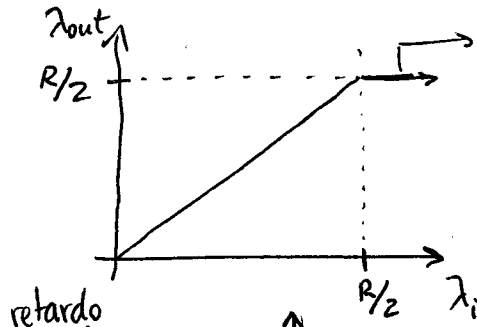
SCENARIO 1: Dos emisores, un router con buffers de capacidad ilimitada

Dos hosts (A y B) envían datos a una velocidad λ_{in} al otro.

λ_{out} = velocidad de recepción de datos en el otro extremo.

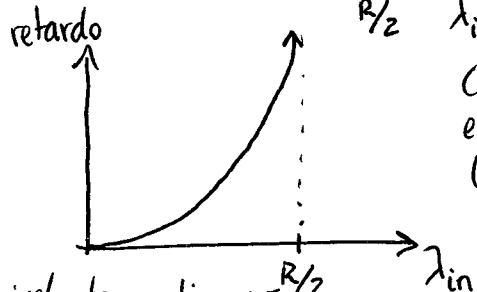
R = capacidad de enlace de salida del router

La tasa de transferencia por conexión, es decir, n° de bytes por segundo en el receptor se puede ver en la siguiente gráfica



el router no puede enviar a más de R , como $A \rightarrow B$ con velocidad λ_{in} y $B \rightarrow A$ con λ_{in} λ_{out} se estanca cuando $\lambda_{in} \geq R/2$.

Esto tiene consecuencias en el retardo, ya que el buffer se llena más de lo que se vacía



Cuando la tasa de llegada en un router se aproxima a la capacidad de enlace se producen grandes retardos \Rightarrow congestión.

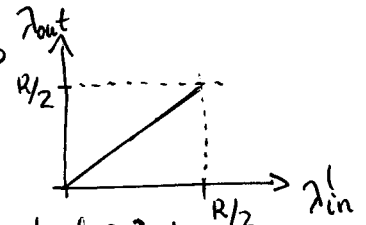
Obs: $\lambda_{in} = \text{goodput}$ = velocidad a nivel de aplicación

ESCENARIO 2: Dos emisores, un router con buffer finitos.

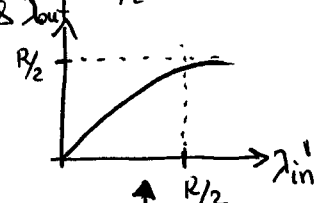
Como los buffers son finitos puede haber retransmisiones, por lo que habrá que hacer retransmisiones: $\lambda_{in}' = \text{goodput}(\lambda_{in}) + \text{tasa bytes retransmitidos}$. ($\lambda_{in}' = \text{CARGA OFRECIDA}$)

2.1: CASO IDEAL: En este caso el emisor solo envía cuando el buffer del router tiene disponibilidad.

Como pérdidas = 0 $\Rightarrow \lambda_{in}' = \lambda_{in}$



2.2: Reenvíos por buffer lleno: los paquetes pueden ser descartados en el router por tener los buffers llenos \Rightarrow reenvíos \Rightarrow \Rightarrow datos originales = $\lambda_{in} < \lambda_{in}' = \text{datos originales} + \text{datos retransm.}$

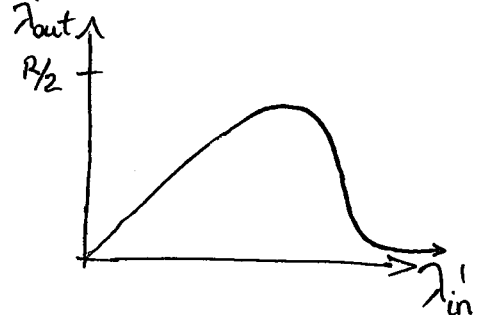


2.3: Duplicados: el emisor puede reenviar dos o más copias del mismo paquete innecesariamente $\Rightarrow \lambda_{in}' > \lambda_{in}$

gráfica igual \uparrow

ESCENARIO 3: Cuatro emisores, routers con buffers finitos y multisaltos

Al aumentar λ_{in} llega un momento en que se desbordan los buffers. Además, otro coste de la congestión cuando se descarta un paquete, toda la capacidad usada por ese paquete con anterioridad fue inútil.



MÉTODOS PARA CONTROLAR LA CONGESTIÓN

Estos son los dos métodos más comunes de control de congestión:

CONTROL DE CONGESTIÓN TERMINAL A TERMINAL

- La capa de red no proporciona soporte explícito a la capa de transporte.
- La congestión es inferida analizando el comportamiento de la red en los sistemas terminales (pérdidas, retardos...).
- Método usado en la práctica por TCP.

CONTROL DE CONGESTIÓN ASISTIDO POR LA RED

- Los routers proporcionan realimentación explícita a los sistemas finales, por ejemplo, un bit en el paquete que indique congestión. Puede incluso informar de cuánto está un router capacitado para ofrecer

CONTROL DE CONGESTIÓN EN TCP

Se usa una nueva variable: ventana de congestión (cwnd)

La cantidad de datos/paquetes no reconocidos (sin ACK) es igual al tamaño de la ventana de congestión. Además, no puede exceder el mínimo entre la ventana de recepción del receptor y congestión (local). El valor de cwnd es modificado dinámicamente según la congestión.

¿Cómo el emisor percibe la congestión?

Por las pérdidas: timeout o 3 ACK's duplicados. En este caso el emisor reduce la ventana de congestión después de estos eventos.

¿Cómo? 3 mecanismos

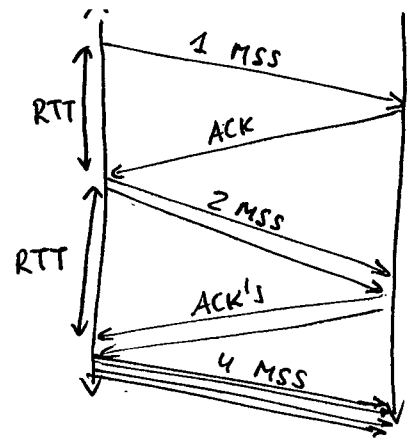
- arranque lento
- evitación de la congestión
- recuperación rápida (TCP Reno)

TCP ARRANQUE LENTO

Cuando la conexión empieza, se incrementa la tasa exponencialmente hasta el primer evento de pérdida.

Inicialmente $cwnd = 1 \text{ MSS}$

Se dobla $cwnd$ cada RTT (modelo simplificado)



RECUPERACIÓN RÁPIDA (TCP Reno) y EVITACIÓN DE LA CONGESTIÓN

Después de un timeout:

→ Arranque lento ($cwnd = 1$)

→ Umbral ($ssthresh$) colocado a la mitad de $cwnd$ anterior. Cuando se alcance de nuevo este umbral significará intuitivamente cerca de una posible congestión por lo que se entrará en la fase de evitación de la congestión.

→ Evitación de la congestión: La ventana de congestión crece linealmente 1 MSS por RTT (modelo simplificado).

Después de 3 ACK's duplicados:

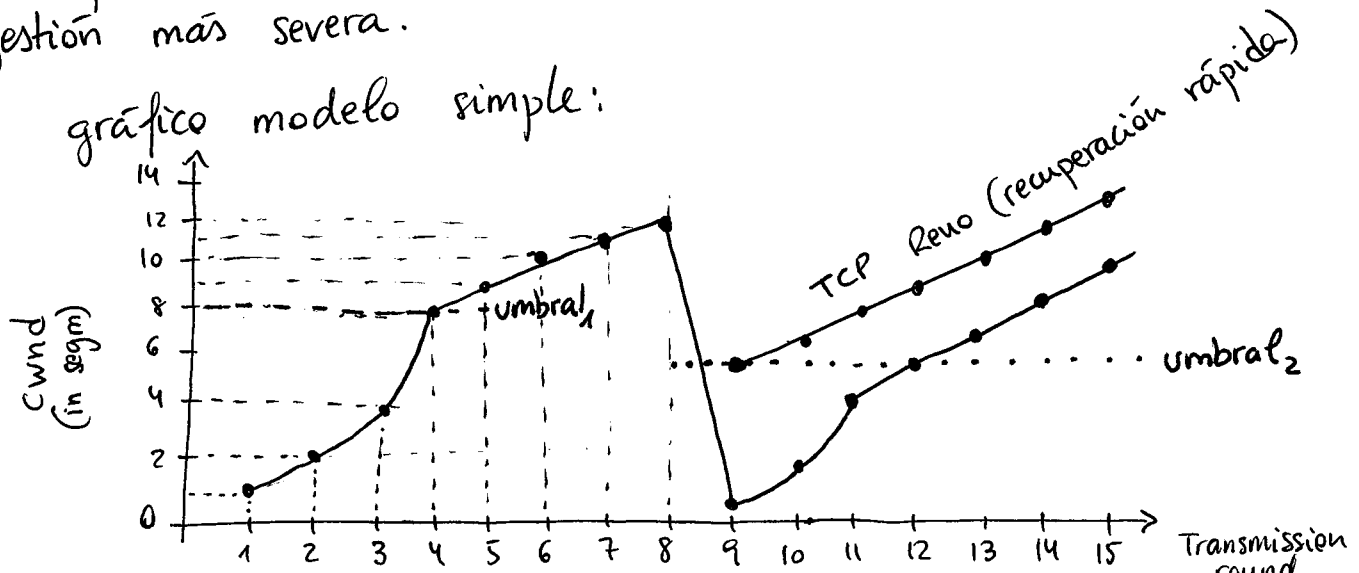
→ En TCP Tahoe igual que un timeout.

→ En TCP Reno se efectúa recuperación rápida:

$cwnd$ pasa a ser la mitad ($cwnd = \text{umbral}$) y comienza ya la fase evitación de la congestión.

La filosofía es que 3 ACK's duplicados indican la capacidad de la red para transportar segmentos. En cambio, un timeout indica una congestión más severa.

Resumen gráfico modelo simple:



Es un crecimiento aditivo y decrecimiento multiplicativo.

Método: incrementar tasa de transmisión (tamaño ventana), hasta que se determinen pérdidas.

- Aditivo: incrementar cwnd hasta que se detecten pérdidas.
- Decremento multiplicativo: fijar cwnd a la mitad después de pérdidas.

Si se representa cwnd size en función del tiempo se observará una gráfica con un comportamiento de dientes de sierra.

THROUGHPUT TCP (ESTACIONARIEDAD)

Si obriamos el arranque lento y sea w el tamaño de la ventana cuando hubo pérdidas entonces:

Cuando $cwnd = w \Rightarrow \text{throughput} = w/RTT$

Cuando $cwnd = w/2$ (evento) $\Rightarrow \text{throughput} = (w/2)/RTT$

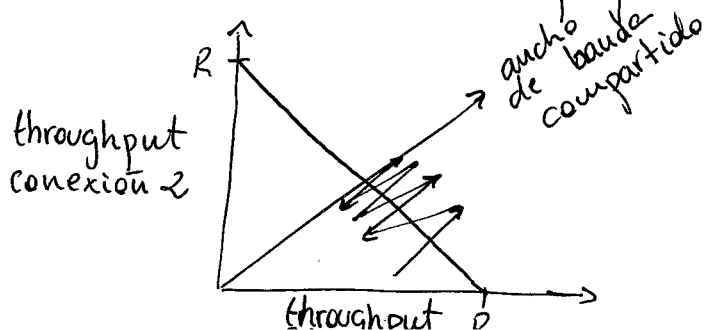
\Rightarrow Luego el throughput medio entre pérdidas de forma ideal sería $= 0.75 w/RTT$.

EQUIDAD TCP

Objetivo equidad: Si K conexiones TCP comparten un enlace cuya capacidad es R , cada una debería tener una tasa media de R/K .

¿Por qué TCP es justo? Ejemplo simple, de 2 sesiones compitiendo:

- Los incrementos tienen pendiente 1 cuando el throughput aumenta.
- Los decrementos son proporcionales



EQUIDAD Y UDP

Aplicaciones multimedia muchas veces no usan TCP porque prefieren pérdidas a reducir la tasa. Esto puede perjudicar al resto de usuarios.

EQUIDAD Y CONEXIONES TCP PARALELAS

No hay límite al nº de conexiones a abrir entre dos equipos. Los navegadores o aplicaciones de tipo FH utilizan conexiones TCP paralelas.

Ejemplo: 1 aplicación compartida con otras nueve consigue $R/10$.

1 aplicación abre 11 TCPs paralelas, comparte con otras nueve pero obtiene $R/2$!