TEMA 3 FUNDAMENTOS DE REDES.pdf

pikopakoi

Fundamentos de Redes

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa

TEMA 3: CAPA DE TRANSPORTE EN INTERNET.

1. Introducción.

Funciones y servicios de la capa de transporte:

- Comunicación extremo a extremo (end-to-end).
- Multiplexación/demultiplexación de aplicaciones -> puerto.

Protocolo UPD:

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

Protocolo TCP:

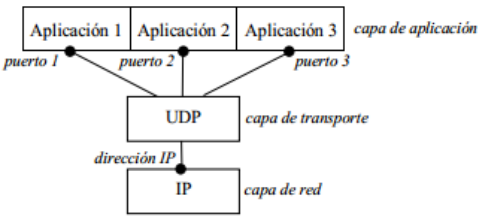
- Multiplexación/demultiplexación de aplicaciones.
- Servicio orientado a conexión, fiable: Control de errores y de flujo. Control de la conexión. Control de la congestión.

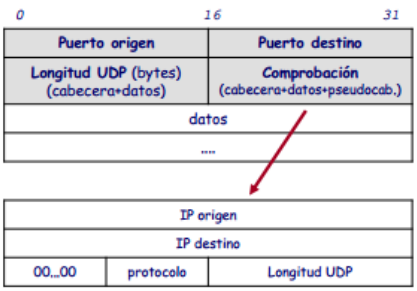
2. Protocolo de datagrama de usuario (UPD).

"User Datagram Protocol": RFC 768.

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.





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

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

Cada segmento UPD se encapsula en un datagrama IP.

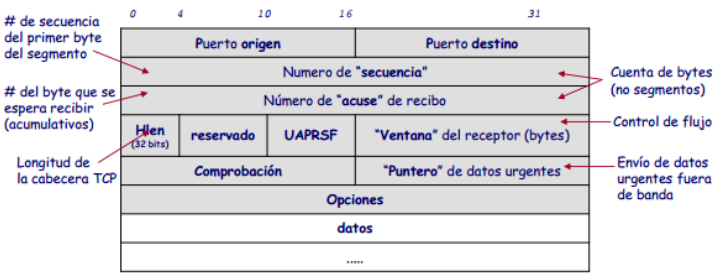
3. Protocolo de control de transmisión (TCP).

Multiplexación/demultiplexación lo mismo que en UPD.

Características del "Transmission Control Protocol": RFC 793 (1122,1323,2018,2581).

- Servicio orientado a conexión ("hand-shaking").
- Entrega ordenada.
- Full-duplex.
- Mecanismo de control de flujo de detección y recuperación de errores (ARQ: Automatic Repeat reQuest (se encarga del reenvío de paquetes que no llegan al destino o llegan mal)).
    - Confirmaciones positivas (ACKs) y acumulativas.
    - Incorporación de confirmaciones ("piggybacking").
    - "timeouts" adaptables.
    - Ventanas adaptables.

- Mecanismo de control de congestión.
- Servicio fiable -> control de congestión y control de flujo.



Cada segmento TCP se encapsula en un datagrama IP.

- ➔ **Numero de secuencia**: cuando un paquete llega al destino, el destino sabe el elemento al que le corresponde.
- ➔ **Acuse**: Va en el otro sentido, es el AFK positivo, le decimos al destino el último paquete que he recibido por él.
- ➔ **UAPRSF:**
- U – Urgente.
- A – Si A = 1, Es el AFK y el 1 es para que consulte.
- S – Abre conexión.
- F – Cierra conexión.
- ➔ **Ventana:** me dice el tamaño de la ventana. Se comprime la ventana o agranda mapea la velocidad dependiendo del entorno.
- ➔ **Puntero a datos urgentes:** Unido al flag U, El puntero separa de lo que es urgente de lo que no, los urgentes están al principio y el puntero indica cuando dejan de ser urgentes.
- ➔ **Datos:** string[1,2,3,4,5] de paquetes. Si yo mando un paquete 1 y espero confirmación es perder mucho tiempo que yo estoy parado (lento para el usuario). Se ha visto que entonces envío 1,2,3 (enviar muchos paquetes se llama ventana) y espero confirmación de uno. Cuando mando paquetes a la vez, a veces llegan desordenados, por tanto, se le añade un numerito para que el destino sepa reordenarlos (Numero de secuencia).

Multiplexación/demultiplexación de aplicaciones:

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

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

- FTP: Recibimos fichero, abrir conexión parar recibirlo y después cerramos conexión.
- SSH (telnet): Conexión remota, la terminal recibe los datos como quiero, menor delay posible.
- SMTP: No queremos que el correo llegue mal.
- DNS: Subconjunto que se basa en TCP (por eso en rojo).

Control de la conexión:

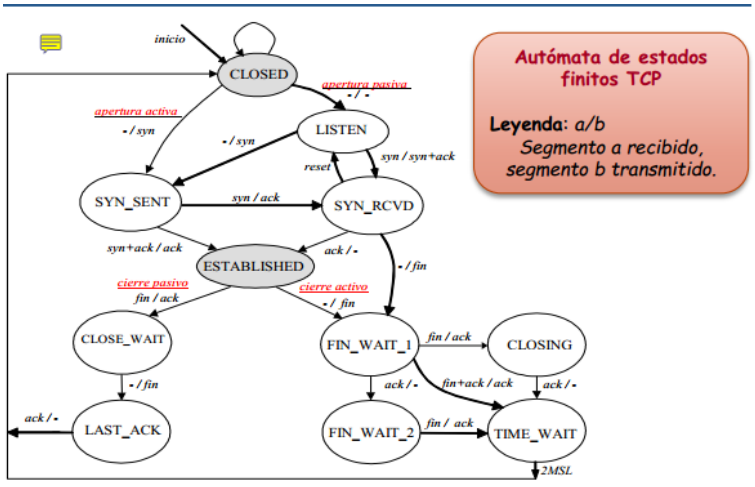
- El intercambio de información tiene tres fases: three-way handsake.

  I. Establecimiento de la conexión (sincronizar # de secuencia y reservar recursos).
  II. Intercambio de datos (full – dúplex).
  III. Cierre de la conexión (liberar recursos).

- Se inicia con el trhee-way handshake: Primero se inicia el número de secuencia, se puede hacer siempre al mismo número (0 o 1) pero puede ser inseguro. Si el número de secuencia empieza por el mismo número y las dos conexiones empiezan a la vez puede haber confusiones. Así que se inicia el número de secuencia de forma aleatoria, para evitar que un proceso acepte paquetes que no van para él.

- Números de secuencia:
  - Campo de 32 bits -> 2^32 valores.
  - 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 -> Ciclo al cabo de 4h 46min.
    - Protege de coincidencias, pero no de sabotajes.

  - Incremento (#secuencia = #secuencia + Nbytes).
    - Se incrementa según los bytes de carga útil (payload).
    - Los flags SYB y FIN incrementan en 1 el número de secuencia.



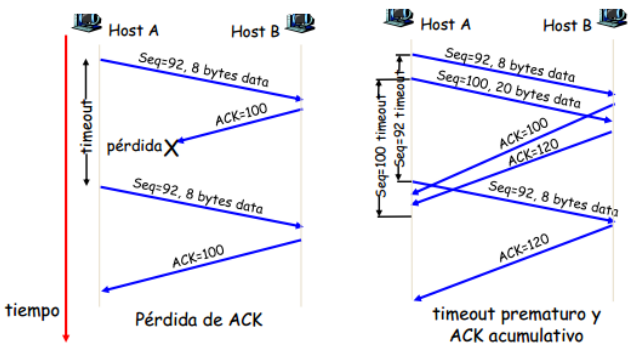
- Autómata de cómo debería funcionar TCP.
  - Listen: servidor "escucha" de peticiones.
  - Paso 1 de cliente: closed a syn_Sent, que se queda en ESTABLISHED después de enviar el ACK. Cuando este activo a LISTEN, después syn+ack a SYN_RCVD, SYN_RCVD pasa a ESTABLISHED cuando recibe ack. Al estar los dos en ESTABLISHED se pueden comunicar.
  - Cierre activo -> desconexión.
  - De FIN_WAIT_1 a TIME_WAIT: Si recibe fin+ack, se va dónde está el temporizador, al expirarse se hace se hace dos veces MSL y se va a CLOSED.

Control de errores y de flujo:

- Mejorar rendimiento -> ventana deslizante.
- Control de errores: esquema ARQ con confirmaciones positivas y acumulativas (receptor de confirmaciones positivas, todos los bytes recibidos bien. Sistema funciona bien cuando solo es un paquete. Emisor envía varios segmentos sin esperar confirmación del receptor, así que se sigue enviando hasta que empiece a recibir confirmaciones. Todo emisor está continuamente enviando datos. Y necesita que alguna vez, se le envíe confirmación).

- Campos involucrados:
  - Campos secuencia: offset (en bytes) dentro del mensaje (por donde voy).
  - Campo acuse: número de bytes esperado en el receptor (por donde voy confirmando).
  - Bit A (ACK) del campo de control (byte 28, todo lo anterior lo tengo hasta el byte 27, el siguiente segmento se envía a partir del 28 con más datos).
  - Campo comprobación: checksum de todo el segmento y uso de pseudo-cabecera.

- Escenarios de retransmisión:



- 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): Tiempo de viaje en redondo (tiempo que tarde en enviar un mensaje y recibir confirmación)
  - 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.
  - Si ponemos un temporizador muy largo llegamos tarde si hay muchos errores.
  - Ajustamos timeouts dependiendo del estado que el emisor vea en la red.
  - El timeout tiene que ser acorde al funcionamiento de la red ya que sino esta se deteriora mucho.

Kurose & Ross

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

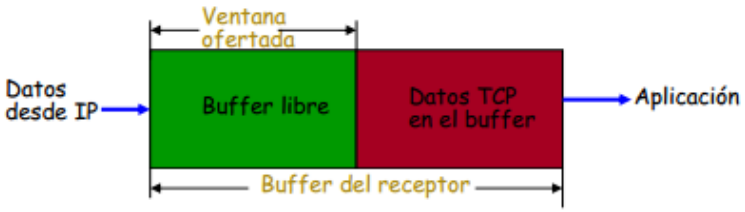
$$RTT = (1-\alpha) \times RTT + \alpha \times RTTmedido \ , \ \alpha \ \& \ \beta \in [0,1]$$

$$Desv = (1-\beta) \times Desv + \beta \times |RTTmedido - RTT|$$

$$Timeout = RTT + 4 * Desv \ (Inicial\ 1s)$$

  - Problema con segmentos re-enviados: ambigüedad en la interpretación.
  - Solución: Algoritmo de Karn: actualizar el RTT sólo para los segmentos no repetidos.
  - Si hay que repetir un segmento incrementar el timeout.

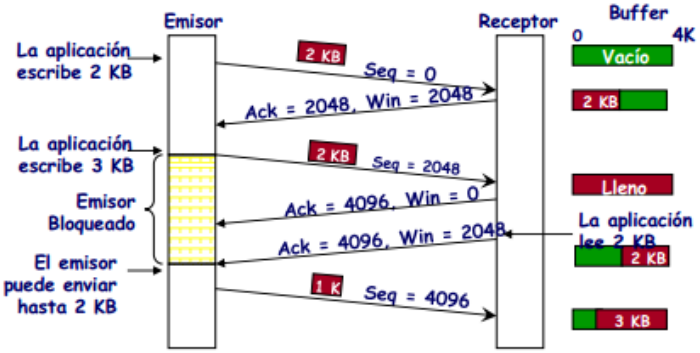
- Control de flujo:
  - Procedimiento para evitar que el emisor sature al receptor.
  - Es un esquema crediticio: el receptor avisa al emisor de lo que puede aceptar.
  - Se utiliza el campo ventana (WINDOW) en el segmento TCP para establecer la ventana ofertada.
  - En el receptor (PASO 1):



  - En el emisor (PASO2):

    Ventana útil emisor = ventana ofertada receptor = bytes en tránsito

    (VENTANA OFERTADA: receptor cada vez que manda una confirmación (ACK con Flag A Activo) también envía una porción del buffer)



  - 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).

Control de congestión:

- Procedimiento para evitar que el emisor sature la red:
    - Ancho de banda de las líneas.
    - Buffer en dispositivos de interconexión.

- Solución: limitar el tráfico generado (=ctrl de flujo)
- Problema: No hay "ente" que avise de lo que se puede enviar (distinto del ctrl de flujo)
- Usar medida indirecta -> pérdidas y/o retrasos en los ACKs.
- Con el control de congestión estamos controlando la velocidad de transmisión, controla los envíos del emisor sin que atosigue al receptor.
- Con el control de flujo el sistema lo tiene claro, cuando puedo enviar envío.
- Con el control de congestión estamos siendo más heurísticos que el control de flujo.
- IDEA: Es el TCP original y se empieza enviando con calma, y cada vez aumentando más y más la velocidad, si veo que el temporizador cae, bajo un poco la velocidad y después voy aumentándola de nuevo poco a poco.

- En el emisor se utilizan una ventana y un umbral.
    - Controla la velocidad en base de una ventana (como el control de flujo).
    - Limitamos nuestra velocidad dependiendo del tamaño de ventana, y eso, de nuevo, controla la velocidad del emisor. Un 70% del tiempo estoy quieto y eso hace que sea un cuarto de rápido de lo que podría si estuviese enviando todo el tiempo. Si la red encaja bien con el segmento de ventana, aumento el segmento de ventana y tengo dos, y así continuamente, de esta forma envío más rápido y envío todo el rato ya que mi ventana se va ampliando.
    - El control de congestión intenta no congestionar la red, por eso la primera ventana es pequeña.
    - Adicionalmente, se define un umbral, que determina dos operaciones de avance:
        I. Inicio lento (aceleración más rápida, crecimiento exponencial): mientras el tamaño de mi ventana esté por debajo del umbral, voy a ir aumentando la ventana de congestión 1MSS por cada ACK que recibo.
        II. Prevención de la congestión: si te confirman la ventana completa, creces 1MSS.

            Si hay timeout: El retraso de timeout percibo que mi ventana se está congestionando así que empiezo de nuevo. Se pone el umbral a la mitad.

VCongestion = MSS (maximum segment size) ➔ Actualmente, 2, 3 o 4
Umbral a un cierto valor

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 hay timeout entonces
Umbral = *VCongestion*/2
VCongestion = MSS

Combinación del Control de flujo y congestión:

- El emisor elige el mínimo de las ventanas correspondientes:

  Bytes_permitidos_enviar = min{VCongestión,VentanaOfertadaReceptor)

  Ventana útil emisor = Bytes_permitidos_enviar – bytes en tránsito

4. Extensiones TCP.

- TCP se define con múltiples “Sabores”.
- Los diferentes sabores no afectan a la interoperabilidad entre los 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 x 2^16 (=2^30bytes=1GB) autorizados.
    - Tamaño de ventana que me da el receptor es de 16B.
  - Estimación RTT:
    - Opción TCP de sello de tiempo, en todos los segmentos.
    - Envio segmento uno y he empezado en el segundo 300 y el receptor lo copia al AFK (“Segundo 300”) y lo envía al emisor. El emisor lee que se ha enviado en el segundo 300 y al emisor le ha llegado en el segundo 308 así que se sabe el tiempo exacto en el que ha tardado en llegar el ACK (8 segundos).

- PAWS ("Protect Against Wrapped Sequence numbers"):
  - Sello de tiempo y rechazo de segmentos duplicados.
  - Cuando llega un ACK con tiempo anticuado es rechazado a paquetes retrasados.

- SACK:
  - Confirmaciones selectivas.
  - Confirmar por segmentos en lugar de acumulativas.
  - Beneficio: evitas duplicar envíos ("He recibido paquete 3, me falta 1 y 2", y se envían solo esos)