Redes de Computadores

Tema 1: Introducción a las Redes de Computadores e Internet.

Componentes esenciales de una red:

- Hosts o sistemas terminales: número creciente y diversidad de dispositivos (televisores, impresoras, teléfonos, etc).
- Enlaces – Medio físico: medio físico que conecta al emisor y receptor. Es necesario especificar la codificación de bits (bit: unidad de información que se propaga entre emisor y receptor).
    - ✓ Medios guiados (solidos).
    - ✓ Medios no guiados (inalámbricos como la radio).
    - ✓ Ejemplos: pares trenzados, cable coaxial, fibra óptica, radio, wifi.
- La subred: dispositivos de conmutación que facilitan la comunicación. Ejemplo: repetidores, switchers, puntos de acceso, routers.

La internet: es una red de redes vagamente jerárquica.

- Identificación de Host mediante las direcciones IP.
- IPv4 -> 32 bits encapsulados para identificar todos los elementos.
- IPv6 -> compuesta por 128 bits y una notación hexadecimal de 32 dígitos.
- Proveedor de servicios de Internet (ISP). Ejemplo: Rediris.

Protocolos: un protocolo define el formato y el orden de los mensajes intercambiados entre dos o más entidades que se comunican, así como las acciones tomadas al producirse la transmisión y/o recepción de un mensaje y otro suceso.

- Protocolos TCP/IP
    - ✓ Nivel de Aplicación
    - ✓ Nivel de Transporte
    - ✓ Nivel de Red
    - ✓ Nivel de Enlace

Acceso a Internet desde una vivienda:

- Mediante DSL -> ISP (compañía telefónica):
    - ✓ 3 bandas diferenciadas: subida, bajada y telefónica.
    - ✓ Normalmente velocidad asimétrica
    - ✓ Distancias “cortas” entre la vivienda y el proveedor
    - ✓ DSL conectado con DSLAM
- Mediante cable -> utiliza la infraestructura de TV por cable:
    - ✓ Dos canales para carga y descarga.
    - ✓ Normalmente asimétrico.
    - ✓ Medio de difusión compartido.
- Mediante FTTH -> Fiber To The Home:
    - ✓ Mayores velocidades.
    - ✓ Redes activas o pasivas.

- Vía satélite:
  - ✓ Usado en lugares de difícil acceso (zonas rurales)
  - ✓ Velocidades más lentas.

Redes domésticas y corporativas: conexión a un punto de acceso creando una Red de Área local (LAN) mediante cables ethernet o Wi-Fi.

Acceso mediante red móvil (36, 46, 56): usan el espectro de radio.

Conmutación: métodos para transportar datos en una red.

- Conmutación de paquetes:
  - ✓ Datos convertidos en paquetes más pequeños y transmitidos a través de conmutadores de paquetes.
  - ✓ Transmisión de almacenamiento y envío: el conmutador espera a recibir todo el paquete para poder enviarlo.
    En una ruta con N enlaces (N-1 conmutadores) de velocidad R y suponiendo un tamaño del paquete L: $d_{extremo-extremo} = N\frac{L}{R}$
  - ✓ Retardos de cola o perdida de paquetes: los paquetes entrantes esperan en una cola (buffer) de salida para ser enviados mientras que el conmutador envía otro paquete. Si el buffer esta lleno, se puede perder algún paquete entrante o de la cola.
  - ✓ Tablas de reenvío y protocolos de enrutamiento: en la cabecera de los paquetes se incluye la dirección IP del host de destino y cada conmutador selecciona el enlace apropiado para mandar los datos según la misma, aplicando protocolos de enrutamiento.
- Conmutación de circuitos:
  - ✓ Reserva de una ruta de enlace entre hosts y de velocidad constante antes de iniciar la transmisión.
  - ✓ Multiplexación por división de frecuencia (FDM) o multiplexación por división del tiempo (TDM).
  - ✓ Con FDM cada circuito obtiene de forma continua una fracción del ancho de banda, mientras que con TDM, cada circuito dispone de todo el ancho de banda durante breves intervalos de tiempo periódicamente.
  - ✓ Esto puede llevar a la infrautilización de recursos.

La conmutación de paquetes es poco adecuada para los servicios en tiempo real por la variabilidad de sus retardos de extremo a extremo; pero ofrece una mejor compartición de la capacidad de transmisión, es más sencilla, eficiente y barata respecto a la conmutación de circuitos.

Una red de redes: conectar entre si los distintos ISP de acceso.

- Estructura de Red S: la internet (red de redes) es compleja y esta compuesta por una docena de ISP de nivel 1 y cientos de miles de ISP de nivel inferior. La cobertura geográfica de las ISP es muy variable, ya que algunas abarcan continentes y océanos y otros solo pequeñas regiones.
  Los ISP de nivel inferior se conectan con los de nivel superior, y estos entre sí. Los usuarios y proveedores de contenido son ISP de nivel inferior y estos son clientes de los ISP superiores.

En los últimos años, los principales proveedores de contenidos han creado también sus propias redes y se conectan directamente, siempre que pueden, a los ISP de nivel inferior.

Tipos de retardo:

- Conmutación de paquetes:
  - ✓ Retardo de procesamiento: examinar las cabeceras del paquete y procesar a donde enviarlo, además de comprobar errores.
  - ✓ Retardo de espera en cola: tiempo de espera en la cola. Depende del nivel de tráfico.
- Conmutación de comunicación:
  - ✓ Retardo de transmisión: longitud del paquete entre la velocidad de transmisión. $T_{trans} = \frac{L}{V_{trans}}$
  - ✓ Retardo de propagación: distancia del enlace entre la velocidad de propagación. $V_{prop} = 2 * 10^8, 3$
- Nodal: suma de todos los anteriores. $d_{nodal} = d_{proc} + d_{cola} + d_{trans} + d_{prop}$

Retardo de cola y pérdida de paquetes:

- Usualmente se emplean medidas estadísticas. Siendo L la longitud del paquete, "a" el numero de paquetes por segundo y R la velocidad de transmisión.

$$Intensidad\ de\ tráfico = \frac{L * a}{R}$$

>1 -> INDESEABLE Entre 0 y 1 -> MANEJABLE

Retardo de extremo a extremo: generalizado y haciendo varias suposiciones de uniformidad.

$$d_{extremo-extremo} = N\ (d_{proc} + d_{trans} + d_{prop})$$

Tasa de transferencia en las redes de computadores: velocidad a la que el receptor recibe la información (bps). Es el mínimo de las velocidades de transmisión de la ruta.

Capas de protocolos y sus modelos de servicio: cada capa ofrece un modelo de servicio a las capas anterior y posterior mediante protocolos. Los protocolos de las distintas capas tomadas en conjunto constituyen la pila de protocolos, formada por cinco capas:

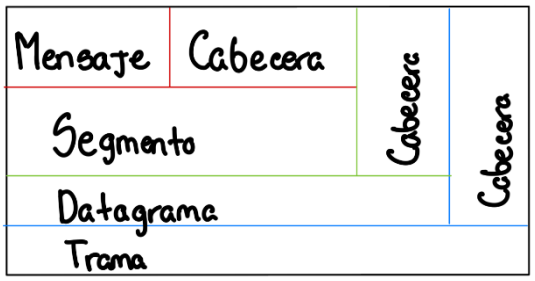
- Aplicación: aplicaciones de red y protocolos de nivel de aplicación (HTTP, SMTP, FTP, DNS…). Los protocolos de aplicación se distribuyen entre varios sistemas terminales y los paquetes que intercambian se denominan mensajes.
- Transporte: transporta los mensajes de la capa de aplicación entre los puntos terminales de la aplicación (TCP y UDP). Sus paquetes se denominan segmentos.
- Red: transporta los paquetes de la capa de red de un host a otro. Recibe un segmento y una dirección de destino, y lo transmite aplicando protocolos de enrutamiento y el protocolo IP.
- Enlace: encamina un datagrama a través de una serie de routers entre el origen y el destino. En cada nodo recibe un datagrama y lo transmite, mediante protocolos variables. Sus paquetes serán denominados como Ttrans.
- Física: Mueve de un nodo al siguiente los bits individuales que forman la trama. Sus protocolos dependen del medio físico de transmisión del enlace.

Modelo OSI: 7 capas, aplicación, **presentación, sesión**, transporte, red, enlace y física.

- Presentación: proporciona servicios para que las aplicaciones puedan interpretar el significado de los datos intercambiados.
- Sesión: sincroniza y recupera el flujo de datos.

Encapsulamiento en TCP/IP: conforme un mensaje pasa a la capa de transporte, este añade una cabecera creando un segmento que encapsula el mensaje anterior y lo transmite a la capa de red. Esta hace lo mismo y crea un datagrama para encapsular el segmento y por ultimo el enlace encapsula al datagrama creando una trama.

Un mensaje largo puede dividirse en varios segmentos de la capa de transporte (los cuales a su vez pueden dividirse en varios datagramas en la capa de red). En el extremo receptor, cada segmento deberá entonces ser reconstruido a partir de sus datagramas constituyentes.



Tema 2: Aplicaciones en Red.

Principios de aplicaciones en red: la comunicación de una aplicación de red tiene lugar entre sistemas terminales en la capa de aplicación.

Arquitectura de las aplicaciones de red: se destaca la arquitectura cliente-servidor y la P2P.

- Cliente-servidor:
  - ✓ Servidor: un host siempre activo con una IP fija que da servicios a las solicitudes de muchos otros hosts (clientes).
  - ✓ Clientes: inician la conversación, se conectan intermitentemente, pueden tener una IP dinámica y no se comunican con otros clientes.
- P2P: comunicación directamente entre sí, y las parejas de host poseen una IP dinámica. Alta escalabilidad.
- Modelo híbrido: los pares se registran previamente en un servidor, los pares buscan otros pares y una vez localizados se sigue el modelo P2P.

Comunicación entre procesos: se comunican entre si enviando mensajes a través de la red de computadores.

Proceso cliente-servidor: sesión de comunicación entre una pareja de procesos:

- Cliente: proceso que inicia la comunicación.
- Servidor: proceso que espera a ser contactado.

Interfaz entre el proceso y la red de computadoras: socket -> interfaz que conecta la capa de aplicación con la capa de transporte (API). El programador solo puede fijar el protocolo de transporte y quizás algunos elementos de la capa de transporte.

Direccionamiento de procesos: se necesita la dirección IP del host y un identificador que especifique el proceso de recepción (socket) en el host de destino. Para esto se utiliza un numero de puerto de destino (port 80 HTTP o port 25 SMTP).

Servicio de transporte disponible para las aplicaciones:

- Fiabilidad: algunas aplicaciones toleran parcialmente la perdida de datos.
- Retardos: algunas aplicaciones requieren bajos retardos.
- Productividad: algunas aplicaciones requieren una determinada tasa de productividad.
- Seguridad: confidencialidad, integridad y autenticación.

Servicio de transporte en Internet: necesidad de elegir entre los protocolos TCP y UDP.

- TCP: servicio orientado a la conexión previa y full-dúplex, y un servicio de transferencia de datos fiables, sin errores y en el orden correcto. Incluye un mecanismo de control de gestión. Se complementa en la seguridad mediante el socket SSL.
- UDP: servicio de transferencia no fiable, ofrece servicios mínimos.
- Servicios no proporcionados: TCP y UDP no proporcionan servicios relativos a la tasa de transferencia (retardos) o la temporización (productividad), sin embargo, algunas aplicaciones de Internet las integran por su cuenta.

Protocolos de la capa de aplicación: definen los tipos de mensajes que se intercambian, sintaxis de mensajes, semántica de los mensajes y reglas de cuando se envían los mensajes y como se procesan.

HTML: Lenguaje de HiperTexto Etiquetado -> las etiquetas delimitan elementos de un documento como cabeceras, párrafos, imágenes, etc.

- Modelo cliente-servidor.
- Cliente > navegador que pide, recibe y visualiza objetos de web.
- Servidor -> servidor web que envía objetos en respuestas a las peticiones

HTTP: protocolo sin memoria de estado que establece una conexión cliente-servidor, dando acceso a las paginas web. Una web consta de un archivo base HTML y diversos objetos, referenciados mediante una URL única. Conecta mediante TCP.

HTTP -> HyperText Transfer Protocol.

- HTTP 1.0:
  - ✓ Conexiones no persistentes.
  - ✓ Cada par solicitud/respuesta se envía a través de conexiones TCP separadas.
  - ✓ La conexión se cierra después de que el servidor envíe el objetivo.
  - ✓ RTT -> Tiempo de ida y vuelta de un paquete (cliente -> servidor -> cliente).
- HTTP 1.1:
  - ✓ Conexiones persistentes.
  - ✓ Única conexión TCP para todos los pares solicitud/respuesta que permanece abierta luego de que el servidor envíe una respuesta.

Formato de mensajes HTTP:

- Dos tipos de mensajes: petición (cliente) y respuesta (servidor).
- Estructura:
  - ✓ Línea de petición. (termina con \r\n)
  - ✓ Línea de cabeceras. (termina con \r\n)
  - ✓ Línea en blanco. (termina con \r\n)
  - ✓ Cuerpo del mensaje. (opcional)

Mensajes de solicitud HTTP:

- Línea de solicitud: primera línea del mensaje con tres campos:
  - ✓ Método: GET, POST, HEAD, PUT, DELETE.
  - ✓ URL: identifica el objeto solicitado.
  - ✓ Versión HTTP utilizada.
- Línea de cabecera: información sobre el host donde reside el objeto, la conexión, el navegador cliente o la preferencia del idioma.
- Cuerpo de entidad: información adicional relevante según que método.

Mensajes de respuesta HTTP:

- Línea de estado: primera línea del mensaje con tres campos:
  - ✓ Versión del protocolo HTTP.
  - ✓ Código de estado: 100 (información), 200 (satisfactorio), 300 (redirección), 400 (error del cliente) y 500 (error de servidor).
  - ✓ Mensaje explicativo del estado: OK, Bad Request, Not Found, HTTP version not supported.
- Línea de cabecera: información de la conexión, fecha y hora de envío, ultima modificación, servidor que genera el mensaje, tamaño de bytes y tipo de objeto.

- Cuerpo de entidad: contiene el objeto solicitado.

Interacción usuario-servidor: Cookies.

- Permiten a los sitios web seguir la pista a los usuarios.
- Componentes:
  - ✓ Línea de cabecera de la cookie en el mensaje de respuesta HTTP.
  - ✓ Línea de cabecera de la cookie en el mensaje de solicitud HTTP.
  - ✓ El archivo de las cookies almacenado en el cliente y gestionado por el navegador.
  - ✓ Base de datos en el servidor.

Almacenamiento en caché web (Servidor Proxy): elemento que actúa de intermediario entre servidor y cliente y que actúa como ambos. Guarda una copia de las paginas visitadas por el cliente. Ante una solicitud, la caché busca en su memoria y, si la encuentra la envía directamente al cliente; si no, la solicitud al servidor, se guarda una copia y se envía al cliente.

Reduce los tiempos de respuesta al cliente y el tráfico externo de las instituciones.

GET condicional: permite verificar a la cache si sus objetos están actualizados. Si hay una nueva solicitud, la cache envía un get condicional al servidor, y este envía un mensaje sin el objeto en caso de que este no haya sido modificado, o con el sí lo ha sido.

Correo electrónico en Internet:

Medio de comunicación asíncrono, barato y con posibilidad de inclusión de adjuntos, formado por tres componentes principales: agentes de usuario, servidores de correo y el Protocolo simple de transferencia de correo (SMTP).

Los agentes de usuario transfieren los mensajes salientes a su servidor de correo, desde donde son mandamos mediante SMTP al servidor de correo del agente de usuario del receptor y se almacenan en su buzón personal, donde son recuperados.

SMTP:

Transfiere mensajes mediante conexión TCP directa desde los servidores de correo de los emisores a los servidores de correo de los destinatarios.

El cliente invoca a su agente de usuario para enviar el correo, que lo envía a su propio servidor, colocándolo en una cola de mensajes. Este cliente abre una conexión TCP con un servidor SMTP ejecutada en el servidor de correo del receptor y después de la negociación inicial, el cliente envía el mensaje que el servidor recibe y coloca en el buzón del usuario receptor. Finalmente, el receptor invoca a su agente de usuario para leer el mensaje cuando le apetezca. (Puerto 25)

En un dialogo SMTP el cliente ejecuta cinco comandos con distintas funcionalidades: HELO, MAIL FROM, RCPT TO, DATA y QUIT. La conexión TCP permanecerá abierta hasta ejecutar el comando QUIT:

Tiene muchas características comunes con HTTP, pero también algunas diferencias. HTTP es un protocolo de extracción (pull) y SMTP de inserción (push); SMTP restringe el cuerpo a código ASCII de 7 bits y HTTP no. HTTP encapsula cada objeto en su propio mensaje mientras que SMTP incluye todos los objetos en el mismo mensaje. Ambos establecen conexiones TCP persistentes cliente-servidor.

Formato de respuestas del SMTP:

- 2XX, operación solicitada mediante el comando anterior concluida con éxito.
- 3XX, orden aceptada, pero el servidor espera que el cliente mande nuevos datos para terminar la operación.
- 4XX, respuesta de error, pero se espera que se repita la instrucción.
- 5XX, error permanente, no se debe repetir la instrucción.

Protocolos de acceso para correo electrónico: SMTP se emplea para transmitir los mensajes del emisor a su servidor de correo y desde éste al servidor de correo del destinatario. Pero para que el destinatario extraiga sus mensajes de entrada desde su servidor de correo se emplean protocolos de acceso como POP3 o IMAP.

- POP3 (Post Office Protocol): protocolo extremadamente simple y de funcionalidad bastante limitada definido en [RFC 1939]. Tres fases:
  - ✓ Autorización: autentificación del usuario.
  - ✓ Transacción: recuperación de mensajes y más operaciones.
  - ✓ Actualización: tras ejecutar el comando "QUIT".
- IMAP: más funcionalidades que POP3 pero es significativamente más complejo. Permite mantener una jerarquía de carpetas en un servidor remoto, accesible desde cualquier computadora, ya que mantiene información acerca del estado a los largo de las sesiones.

Correo electrónico en WEB: el agente de usuario es un navegador web corriente y el usuario se comunica con su buzón remoto mediante HTTP.

Servicio de Directorio de Nombres (DNS): para identificar un host utilizamos su nombre, compuesto por mnemónicos entendidos por las personas pero que ofrecen poca información sobre su ubicación en internet. Para ello utilizamos direcciones IP compuestas por 4 bytes separados por puntos que ofrecen información cada vez más especificas acerca de su situación.

Servicios proporcionados por DNS: DNS es (1) una base de datos distribuida implementada en una jerarquía de servidores DNS y (2) un protocolo de la capa de aplicación que permite a los hosts consultar una base de datos distribuida. Su principal función es traducir los nombres de los hosts en direcciones IP y el protocolo se ejecuta sobre UDP utilizando el puerto 53.

DNS también proporciona otros servicios como los alias de un host, los alias de un servidor de correo, o un sistema de distribución de carga mediante rotación de direcciones IP en servidores replicados. No es ejecutado directamente por el usuario.

Como funciona DNS: Los mensajes DNS se envían dentro de datagramas UDP al puerto 53, utilizando un gran número de servidores DNS distribuidos y un protocolo de la capa de aplicación que especifica como los servidores DNS y los hosts que realizan las consultas se comunican. Usamos un diseño distribuido en vez de centralizado para evitar problemas como: tener un único punto de fallo, no soportar el volumen de tráfico, difícil accesibilidad por la distancia y un mantenimiento complicado.

- **Una base de datos jerárquica y distribuida:** ningún servidor DNS dispone de todas las correspondencias de todos los hosts de Internet, por ello, se implementa una jerarquía entre los distintos tipos de servidores DNS:

- ✓ Servidores DNS raíz: unos 400 servidores distribuidos por todo el mundo que proporcionan las direcciones IP de los servidores TLD.
- ✓ Servidores DNS de nivel superior (TLD): servidor o agrupación de servidores para cada dominio de nivel superior (com, org, net, …) que proporcionan las direcciones IP para los servidores DNS autoritarios.
- ✓ Servidores DNS autoritarios: proporcionan los registros DNS accesibles públicamente que establezcan la correspondencia entre los nombres de dichos hosts y sus direcciones IP.

También existen servidores DNS locales, cercanos a los hosts y que actúan como proxys, o servidores autoritarios intermedios. Las consultas utilizadas entre los distintos elementos pueden ser tanto recursivas como iterativas, no siendo estas dos opciones excluyentes.

- **Almacenamiento en caché DNS:** funcionalidad explotada por DNS para reducir los retardos que le permite "saltarse" servidores en la jerarquía. Puesto que las correspondencias no son permanentes, pasado un cierto tiempo (unos dos días), la información almacenada en caché es descartada.

Registros y mensajes DNS: los registros de recursos (RR) están formados por 4 campos (nombre, valor, tipo, TTL). TTL es el tiempo de vida del registro de recurso y el significado de nombre y valor, dependerá del tipo.

- Tipo = A: Nombre es el host y Valor la dirección IP. Correspondencia estándar en servidores autoritativos.
- Tipo = NS: Nombre es un dominio y Valor es el nombre de host de un servidor de dominio autoritativo.
- Tipo = CNAME. Nombre es un alias y Valor es el nombre de host canónico.
- Tipo =MX. Nombre es un alias y Valor el correspondiente nombre canónico de un servidor de correo.

Si un servidor no es autoritativo, contendrá un registro de tipo NS y otro de tipo A.

Mensajes DNS: tanto los mensajes de consulta como los de respuesta utilizan el mismo formato con la siguiente semántica:

- Sección de cabecera: compuesta por 12 bytes y contiene una serie de campos. El primero es un número de 16 bits que identifica la consulta. Una serie de indicadores de un bit: consulta/respuesta, autoritativo/no, recusión/no. Cuatro campos "número de", que indica el número de apariciones de los cuatro tipos de secciones de datos que siguen a la cabecera.
- Sección cuestiones: incluye un campo de nombre y un campo de tipo.
- Sección respuestas: contiene al registro o los registros del recurso para el nombre consultado originalmente.
- Sección autoridad: contiene registros de otros servidores autoritativos.
- Sección información adicional: contiene otros registros útiles.

Inserción de registros en la base de datos DNS: mediante un registrador, que es una entidad que verifica la unicidad del nombre de dominio, la añade a la base de datos DNS y percibe unas pequeñas tasas por sus servicios, podemos registrar nuestro nombre de dominio.

Para ello, tenemos que proporcionarle a la entidad registradora los nombres y direcciones IP de sus servidores DNS autoritativos principal y secundario, que se asegurará de que se introduzca un registro de tipo NS y otro de tipo A en los servidores TLD correspondientes, además de asegurarse de que el registro de recurso tipo A y el de tipo MX se han introducido en los servidores DNS autoritativos.

Tema 4: Introducción a las Redes de Computadores e Internet.

3.1. La capa de transporte y sus servicios:

Sus protocolos proporcionan una comunicación lógica entre procesos de aplicación. Estudiaremos los protocolos de la capa de transporte para Internet TCP y UDP.

3.1.1. Relaciones entre las capas de transporte y de red:

Mientras que un protocolo de la capa de transporte proporciona una comunicación lógica entre procesos, un protocolo de la capa de red proporciona una comunicación lógica entre host. Los protocolos de la capa de transporte actúan solo en los sistemas terminales llevando los mensajes desde los procesos de aplicación a la frontera de la red, ofreciendo cada uno de ellos un modelo de servicio distinto a las aplicaciones. Estos servicios están restringidos por el protocolo de la capa de red subyacente, aunque se puede ofrecer ciertos servicios pese a que el protocolo de red no los proporcione.

3.1.2. La capa de transporte en Internet:

Hemos de entender que el protocolo IP de la capa de red no es fiable para garantizar la entrega pese a que se realiza el mejor esfuerzo para ello. UDP y TCP tienen la responsabilidad de ampliar el servicio de entrega de IP entre hosts a procesos, lo que se denomina multiplexación y demultiplexación de la capa de transporte, entre algunos otros servicios. Además, TCP proporciona una transferencia de datos fiable y mecanismos de control de congestión.

3.2. Multiplexación y demultiplexación:

- Multiplexación: reúne los fragmentos de los sockets emisores, los encapsula y les añade cabeceras necesarias para crear los segmentos que serán enviados a la capa de red.
- Demultiplexación: consiste en la decodificación de los segmentos para que los fragmentos sean entregados a los sockets correctos.

Para ello, cada segmento contiene campos especiales que indican el socket al que tiene que ser entregado:

- Campo nº puerto origen.
- Campo nº puerto destino.

Los números de puerto son de 16 bits (entre 0 y 65535) y algunos (0 al 1023) restringidos.

Multiplexación y demultiplexación sin conexión: Los segmentos enviados incluyen los datos, nº puerto origen y destino; la capa de red lo encapsula en un datagrama IP. Luego los sockets UDP son dirigidos (demultiplexados) al socket apropiado, que queda perfectamente identificado por una dirección IP y un nº de puerto. Los nº de puerto de origen actúan como direcciones de retorno en caso de respuesta.

Multiplexación y demultiplexación orientados a la conexión: Los sockets TCP identifican por una tupla de 4 elementos direcciones IP de origen y destino, y números de puerto de origen y destino. Dos segmentos TCP con dirección IP o numero de puerto de origen distintos serán dirigidos a dos sockets distintos. Esto permite a los servidores gestionar y mantener multiples

conexiones TCP abiertas conectadas a un mismo puerto destino, ya que las diferencia por sus direcciones IP y/o número de puerto origen.

Servidores web y TCP: En servidores web actuales de altas prestaciones, a menudo se utiliza un solo proceso que crea nuevos sockets para cada nueva conexión. Con conexiones persistentes se usará el mismo socket para cada mensaje; con conexiones no persistentes se creará una nueva conexión TCP para cada mensaje y por tanto, un nuevo socket.

3.3. Transporte sin conexión: UDP.

UDP hace casi lo mínimo que un protocolo de transporte debe hacer: la función de multiplexación o demultiplexación y algún mecanismo de detección de errores. Es un protocolo sin conexión porque no realiza un establecimiento previo de la misma antes de enviar los mensajes. El servicio DNS es un ejemplo de utilización de UDP.

Algunas aplicaciones están mejor adaptadas a UDP que a TCP porque ofrece: mejor control en el nivel de aplicación sobre qué datos se envían y cuando, sin establecimiento de conexión, sin información del estado de la conexión, y poca sobrecarga debida a la cabecera de los paquetes (8 bytes frente a los 20 de TCP).

UDP suele ir bien para aplicaciones multimedia que soportan cierta perdida de paquetes; no obstante TCP se está convirtiendo en un protocolo cada vez más atractivo para ellas. La falta de mecanismos de control de congestión de UDP puede llevar a una alta tasa de perdida de paquetes UDP y al estrangulamiento de sesiones TCP. No obstante, es posible (aunque costoso) implementar un servicio de transmisión fiable en la capa de aplicación que luego se ejecute sobre UDP.

3.3.1. Estructura de segmentos UDP:

Los segmentos UDP están formados por un campo de datos (los datos del mensaje y una cabecera con cuatro campos de dos bytes:

- Nº puerto origen.
- Nº puerto destino.
- Longitud -> Nº bytes del segmento UDP.
- Suma de comprobación.

3.3.2. Suma de comprobación de UDP:

Utilizada para detectar errores terminal-terminal. Contiene el Ca(1) (complemento a 1) de la suma de todas las palabras de 16 bits del mensaje; en el receptor, se suma a la suma anterior y se comprueba si todo son unos (sin errores) o si hay algún cero (algún error). No obstante, UDP no repara los errores; o bien desecha los paquetes, o bien los envía con alguna advertencia.

3.4. Principios de un servicio de transferencia de datos fiable:

La abstracción del servicio consiste en un canal fiable para transmitir datos sin que ningún bit se corrompa ni se pierda, y que se entreguen manteniendo el orden en que fueron enviados. Un protocolo de transferencia de datos fiable debe implementar esta abstracción, aunque la capa inferior sea no fiable.

Trabajaremos con un modelo de protocolo “rdt” (reliable data transfer) y considerando una transferencia de datos unidireccional. Emisor y receptor enviaran paquetes de datos y de control mediante udt_enviar(); invocaremos el lado emisor mediante rdt_enviar(); rdt_recibir()

será llamado desde el lado receptor cuando llegue un paquete; y el protocolo suministrará datos a la capa superior mediante entregar_datos().

3.4.1. Construcción de un protocolo de transferencia de datos fiable:

Iremos viendo protocolos de complejidad creciente, hasta llegar a uno sin defectos.

Transferencia de datos fiables sobre un canal totalmente fiable rdt1.0: En este caso el protocolo resulta trivial de forma que el protocolo se limita a las acciones antes mencionadas sin funcionalidades extra. Para representar el funcionamiento de emisor y receptor emplearemos máquinas de estados finitos (FSM) separadas, donde las flechas indican la transición del protocolo de un estado a otro, mostrando en su etiqueta el suceso que provoca la transición (arriba) y las acciones a realizar (abajo). En este caso, tendremos FSMs de un único estado, con una transición desde este a sí mismo.

Transferencia de datos fiable sobre un canal de errores de bit rdt2.0: Trataremos con protocolos que empleen reconocimientos, tanto positivos como negativos en la recepción de mensajes, conocidos como protocolos ARQ (solicitud automática de repetición). Necesitaremos tres capacidades extra:

- Detección de errores: Requiere el envío de bits adicionales para el calculo de la suma de comprobación.
- Realimentación del receptor: Paquetes de un bit con valor 0 para el reconocimiento negativo (NAK) y 1 para el reconocimiento positivo (ACK).
- Retransmisión: De un paquete recibido con errores.

Ahora el lado emisor estará formado por dos estados. El primero recibe los datos de la aplicación y los envía, transicionando al segundo. El segundo espera el reconocimiento; si es negativo reenvía los datos y vuelve a esperar, si es positivo transiciona al primer estado. Así, solo se enviarán nuevos datos si el reconocimiento es positivo. Por su parte el receptor, de un único estado, recibe los datos y comprueba su integridad. Si están corruptos envía un paquete NAK; si no, los extrae, entrega a la aplicación y envía un paquete ACK.

No obstante, los paquetes de reconocimiento también pueden verse corrompidos. Para solucionarlo, el emisor añadirá un nuevo campo “numero de secuencia” al paquete de datos (0 o 1) y reenviará los paquetes cuando se haya corrompido el reconocimiento. El receptor analizará este campo para saber si se trata de un nuevo paquete o de una retransmisión. Ahora los FDMs de emisor y receptor contienen el doble de estados, en relación a si se espera un 1 o un 0.

Otra opción consiste en enviar un paquete ACK para el ultimo paquete recibido correctamente en lugar de un NAK para el siguiente. Para ello se debe incluir un numero de secuencia en el paquete ACK y, si el emisor recibe ACKs duplicados sabrá que el siguiente paquete no ha sido recibido correctamente.

Transferencia de datos fiable sobre un canal con perdidas y errores de bit rdt3.0: Para detectar la perdida de paquetes por el canal subyacente, ya sean los datos enviados o el reconocimiento, estableceremos un temporizador de espera al reconocimiento, tras cuya finalización es muy probable que se haya perdido un paquete o haya ocurrido algún error. Si esto ocurre, se reenvía el paquete. El emisor necesitará (1) iniciar el temporizador cada vez que envíe un paquete, (2) responder a una interrupción del temporizador y (3) detener el temporizador.

Con las funcionalidades vistas hasta ahora, hemos implementado un protocolo de transferencia de datos fiable de bit alternante. A continuación, trabajaremos con ello.

3.4.2. Protocolo de transferencia de datos fiable con procesamiento en cadena:

Mediante protocolos de parada y espera como el visto con anterioridad, se infrautilizan los recursos obteniendo un rendimiento del enlace extremadamente pobre. Para solucionarlo, el emisor podría enviar varios paquetes sin esperar sus respectivos reconocimientos, técnica conocida como pipeling o procesamiento en cadena, con las siguientes consecuencias:

- Necesidad de aumentar el rango de los números de secuencia.
- Implementación de buffers de almacenamiento de paquetes en los lados emisor (transmitidos, pero no reconocidos) y receptor (recibidos correctamente).
- Recuperación de errores mediante dos métodos: retroceder N y la repetición selectiva.

3.4.3. Retroceder N (GBN):

En este caso el numero de paquetes no reconocidos en el canal esta restringido a un máximo N, debido a mecanismos de control de flujo. Definimos la base como el numero de secuencia del paquete no reconocido más antiguo y signumsec como el número de secuencia más pequeño no utilizado; podemos identificar cuatro intervalos en el rango de los números de secuencia:

- [0, base - 1] -> Paquetes transmitidos y reconocidos.
- [0, SigNumSec - 1] -> Paquetes enviados, pero no reconocidos.
- [SigNumSec, base + N - 1] -> Paquetes que pueden ser enviados.
- [(base + N)…] -> Paquetes no utilizables.

(Protocolo de ventana deslizante con tamaño de ventana N).

El número de secuencia se incluye en un campo de longitud K en la cabecera del paquete, siendo el rango [0, 2^k - 1] y todas las operaciones aritméticas que las impliquen utilizarán aritmética en modulo 2^k (el número de secuencia 2^k – 1 va seguido del 0).

El emisor del protocolo GNB ha de responder a tres tipos de sucesos:

- Invocación desde la capa superior: Comprobando si la ventana está llena y devolviendo los datos en caso de que lo esté, o enviándolos en caso de que no lo esté.
- Recepción de un mensaje de reconocimiento ACK: Con reconocimiento acumulativo de los paquetes anteriores.
- Un suceso de fin de temporización. Reenviando todos los paquetes ya transmitidos, pero no reconocidos.

Si el receptor, por su parte, recibe un paquete en orden y sin errores, envía el paquete ACK correspondiente y, en cualquier otro caso lo descarta a la espera de un reenvío. Solo ha de mantener la variable NumSecEsperado. Este método simplifica el almacenamiento en el buffer del receptor, pero aumenta la necesidad de retransmisiones.

3.4.4. Repetición selectiva (SR):

Estos protocolos evitan las retransmisiones innecesarias haciendo que el emisor únicamente retransmitida aquellos paquetes sospechosos de llegar al receptor con error. El receptor confirmará individualmente que paquetes ha recibido correctamente.

Las acciones realizadas por el emisor serán:

- Datos recibidos de la capa superior: Comprobación de la ventana y, si procede, envío de los paquetes.
- Fin de temporización. Individualizados para cada paquete.
- ACK recibido. Marcando el paquete y gestionando la ventana.

Por su parte el receptor confirmará cualquier paquete correcto y almacenará en un buffer los desordenados hasta completar la secuencia y enviarlos a la aplicación conjuntamente. Si un paquete pertenece al intervalo [base recepción – N, base recepción - 1] se genera un ACK para el mismo, aunque ya haya sido reconocido anteriormente, para evitar bloqueos. Si el paquete no pertenece al intervalo anterior ni al intervalo [base recepción, base recepción + N - 1] se ignora.

En estos protocolos las ventanas de emisor y receptor no siempre coinciden. Por este motivo, y para evitar confusiones con respecto a si un paquete es una retransmisión o contiene datos nuevos, el tamaño de la ventana deberá ser menor o igual que la mitad del tamaño del espacio de números de secuencia.

Por último, para hacer frente a una posible reordenación de paquetes, es necesario asegurarse de que no se reutiliza un numero de secuencia “x” hasta asegurarse de que los paquetes enviados con anterioridad con dicho número ya no se encuentran en la red. Esto se hace suponiendo un tiempo de vida máximo que en las ampliaciones de TCP para redes de alta velocidad se ha fijado en tres minutos.

3.5. Transporte orientado a la conexión TCP:

Protocolo de la capa de transporte de Internet, fiable y orientado a la conexión, que implementa muchos de los principios básicos vistos anteriormente.

3.5.1. La conexión TCP:

Las conexiones TCP son lógicas, con un estado común que reside solo en los niveles TCP de los hosts que se comunican, quienes, como parte de este proceso, iniciarán muchas variables de estado TCP. Una conexión TCP proporciona un servicio full-duplex y casi siempre punto a punto, donde los routers intermedios son inconscientes de las mismas.

Para establecerla se emplea un acuerdo en tres fases: (1) el cliente envía un segmento TCP especial, (2) el servidor responde con otro segmento TCP especial y (3) el cliente responde con un nuevo segmento especial; solo el ultimo puede transportar carga útil. Ahora los dos procesos pueden enviarse datos entre sí, cada una con sus respectivos buffers, variables y socket de conexión.

Cuando los datos atraviesan el socket, TCP los dirige al buffer de emisión, desde donde los encapsulará y mandará a la capa de red. Cuando recibe un segmento en el otro extremo, los datos se colocan en un buffer desde donde la aplicación puede leerlos. La cantidad de datos que pueden cogerse y colocar en un segmento está limitada por el tamaño máximo de segmento (MSS), a su vez limitada por otros aspectos.

3.5.2. Estructura del segmento TCP:

Un segmento TCP consta de un campo de datos con un fragmento de los datos de aplicación, limitado por MSS y diversos campos de cabecera (que habitualmente ocupa 20 bytes). La cabecera incluye los números de puerto de origen y destino, un campo de suma de comprobación y otros campos como:

- Numero de secuencia y numero de reconocimiento, de 32 bits cada uno.
- Ventana de recepción, de 16 bits para el control de flujo.
- Longitud de cabecera, de 4 bits y normalmente vacío.
- Opciones, opcional y de longitud variable.
- Indicador (6 bits), ACK para validar reconocimientos, RST, SYN y FIN para establecimiento y cierre de conexiones, CWR y ECE para notificaciones de congestión explícita, PSH y URG (no utilizados en la práctica).

Numero de secuencia y números de reconocimiento: parte critica del servicio de transferencia de datos fiable de TCP. El número de secuencia de un segmento es el número del primer byte del segmento dentro del flujo de bytes, mientras que el número de reconocimiento que un host incluye en su segmento es el del siguiente byte que espera del otro host, proporcionando reconocimientos acumulativos. Ante la recepción de segmentos desordenados se pueden descartar o almacenar a la espera de rellenar huecos (más común). El número de secuencia inicial es elegido aleatoriamente por ambos lados de TCP. Los reconocimientos se superponen al segmento de datos.

3.5.3 Estimación del tiempo de ida y fin de temporización:

Estimación del tiempo de ida y vuelta: primero tomaremos RTTMuestra como el tiempo transcurrido desde que se envía el segmento hasta que se recibe el correspondiente ACK, estimada a partir de uno solo de los segmentos transmitidos, pero no reconocidos (excluyendo retransmisiones). Como este valor fluctúa de un segmento a otro, TCP mantiene un valor promedio, RTTEstimado, y lo actualiza según: $RTTEstimado = (1 - \alpha) \times RTTEstimado + \alpha \times RTTMuestra$, donde α suele ser igual a 0,125 (1/8). Finalmente; para analizar la variabilidad de RTT calculamos RTTDesv como:

$$RTTDesv = (1 - \beta) \times RTTDesv + \beta \times |RTTMuestra - RTTEstimado|$$

Donde β habitualmente es 0,25 (1/4).

Definición y gestión del intervalo de fin de temporización para la retransmisión:

$$IntervaloFinTemporización = RTTEstimado + 4 \times RTTDesv$$

Se recomienda un valor inicial de 1s. Se duplica tras finalizar la temporización para evitar un fin de temporización prematuro para el segmento subsiguiente, pero tan pronto como se recibe un segmento y se actualiza RTTEstimado, se calcula de nuevo usando la formula anterior.

3.5.4. Transferencia de datos fiable:

Garantiza que el flujo de bytes extraído por el proceso receptor de su buffer es exactamente igual al flujo enviado por el emisor. De forma recomendada, estos procedimientos usan un único temporizador para controlar la retransmisión. Primero veremos una versión extremadamente simplificada donde el emisor TCP solo emplea los fines de temporización

para recuperarse de las pérdidas de segmentos, y suponiendo que los datos se envían en una única dirección. Vemos que hay 3 sucesos importantes:

- Datos recibidos desde la aplicación: encapsula los datos en un segmento y lo pasa a IP, iniciando el temporizador si no lo está ya.
- Fin de temporización: retransmite el segmento y reinicia el temporizador.
- Llegada de un ACK (y = nº reconocimiento). Si y > BaseEmision, actualiza esta última al valor de y reconociendo todos los segmentos anteriores y, si existen segmentos no reconocidos en la red inicia el temporizador.

Duplicación del intervalo de fin de temporización: Esta técnica se emplea de forma constante hasta recibir el ACK correspondiente, proporcionando una forma limitada de control de congestión.

Retransmisión rápida: Con un periodo de fin de temporización, aumenta el retardo terminal a terminal. Para evitarlo, el emisor TCP observa los ACK duplicados (reconocen segmentos ya reconocidos) y, si recibe tres ACK seguidos para los mismos datos, interpreta que el siguiente segmento se ha perdido, así que realiza una retransmisión rápida (antes de que acabe el temporizador).

Retroceder N o Repetición selectiva: Podríamos considerar TCP como un hibrido entre los protocolos GNB y SR en lo que respecta a su mecanismo de recuperación de errores. Por un lado, el empleo de reconocimientos acumulativos hace que el emisor solo tenga que mantener las variables BaseEmision y SigNumSec, pareciéndose a GNB. No obstante, el almacenamiento en el buffer de recepción le evitan un reenvío innecesario de segmentos en caso de recepciones desordenadas o paquetes perdidos. Además, una modificación propuesta en 2018 implementa el reconocimiento selectivo.

3.5.5. Control de flujo:

El proceso de aplicación del receptor leerá los datos del buffer de recepción, pero no necesariamente en el instante en que llegan, lo que puede provocar el desbordamiento del buffer. Para evitarlo, TCP proporciona un servicio de control de flujo, adaptando la velocidad de transmisión del emisor a la de lectura del receptor. Aunque son similares, no hay que confundir el mecanismo de control de flujo con el de control de congestión. Para analizarlo, supondremos que el receptor TCP descarta los segmentos que no llegan en orden.

El emisor mantendrá una variable dinámica conocida como ventana de recepción, que proporciona una idea del espacio disponible en el buffer de recepción. Supongamos una conexión entre un host A (emisor) y un host B (receptor). El host B asigna un buffer a esta conexión de tamaño BufferRecepción y define las variables UltimoByteLeido (por la aplicación) y UltimoByteRecibido (procedente de la red). Para evitar el desbordamiento:

$$\text{UltimoByteRecibido} - \text{UltimoByteLeido} \leq \text{BufferRecepcion}, \text{ por tanto:}$$

$$\text{VentanaRecepcion} = \text{BufferRecepcion} - [\ \text{UltimoByteRecibido} - \text{UltimoByteLeido}\ ]$$

El host B incluye este valor en el campo correspondiente de cada segmento que envia al host A. El host A gestionará otras dos variables UltimoByteEnviado y UltimoByteReconocido, asegurándose de que se cumpla:

$$\text{UltimoByteEnviado} - \text{UltimoByteReconocido} \leq \text{VentanaRecepcion}$$

Para evitar el bloqueo del host A sin VentRecepcion = 0 y el host B no tiene datos que enviar, TCP requiere al host A que siga enviando segmentos con un byte de datos que serán reconocidos por el receptor y contendrán un valor de VentRecepcion ≠ 0 cuando el buffer empiece a vaciarse.

3.5.6. Gestión de la conexión TCP:

Para establecer una conexión TCP primero el proceso de aplicación cliente informa al cliente TCP de ello y, a continuación, el establecimiento se ejecuta en tres pasos:

1. El lado cliente de TCP envía un segmento especial al lado del servidor, sin datos, y con el bit SYN de la cabecera a 1. Se asigna un número de secuencia inicial (cliente-nsi), se encapsula en un datagrama IP y se envía al servidor.
2. Si el datagrama llega, el servidor extrae el segmento, asigna los buffers y variables TCP a la conexión y envía un segmento de conexión concedida al cliente. El segmento no contiene datos, el bit SYN se pone a 1, el reconocimiento se hace igual a: cliente – nsi + 1 , y el servidor elige su propio número de secuencia inicial (servidor – nsi).
3. Al recibir este segmento, el cliente asigna buffers y variables a la conexión, y envía otro segmento al servidor, confirmando el segmento de conexión concedida, con bit SYN a cero y la posibilidad de transportar datos a la aplicación.

Cualquiera de los dos procesos puede dar por terminada la conexión ejecutando un comando de cierre. Esto se hace enviando un segmento especial TCP con el bit FIN a 1; en el otro lado, se envía un reconocimiento y un segmento de desconexión propio; por último, el lado inicial reconoce este último segmento, y los recursos de ambos hosts asociados a la conexión que dan liberados.

3.7. Mecanismos de control de congestión de TCP:

Para limitar la velocidad de transmisión, este gestiona otra variable, la ventana de congestión, imponiendo otra restricción sobre la cantidad de datos no reconocidos en la red de forma que:

UltimoByteEnviado – UltimoByteReconocido ≤ min (VentCongestion , VentRecepcion)

Suponiendo un escenario en el que el emisor siempre tiene datos que enviar la única limitación es la ventana de congestión, al inicio de cada RTT el emisor puede enviar bytes a través de la conexión y recibir a los ACKs correspondientes al final del RTT. Así, la velocidad de transmision será VentCongestion/RTT bytes/s y, ajustando la ventana el emisor puede ajustar la velocidad de transmisión. Sí un emisor detecta un suceso de perdida paréntesis (fin de temporización o recepción de 3 ACKs duplicados), lo interpretará cómo una indicación de que existe congestión en la ruta entre emisor y receptor. Si por el contrario recibe los ACKs, interpretara que no hay congestión y aumentará la velocidad de transmisión, en mayor o menor medida en función de la llegada de los paquetes, por lo que se dice que TCP es auto-temporizado.

Para determinar la velocidad del emisor TCP sin congestionar la red ni infrautilizar el ancho de banda, TCP se basa en los siguientes principios:

- Un segmento perdido implica congestión y, por tanto, la velocidad de transmisión del emisor TCP debe reducirse.

- La llegada de un ACK correspondiente a un segmento no reconocido todavía indica que la red entrega los segmentos del receptor, por lo que la velocidad de transmisión puede incrementarse.
- Tanteo del ancho de banda, incrementando la velocidad en respuesta a los ACKs y reduciéndola tras una pérdida.

Ahora veremos el algoritmo de control de congestión de TCP, que consta de tres componentes principales:

1. Arranque lento: en esta fase, el valor inicial de la ventana de congestión es de 1 MSS y se incrementa en un MSS por cada ACK recibido (se duplica cada RTT). Si se produce un fin de temporización, almacena en una variable de estado, umbral(A), el valor de VentCongestion/2 y se reinicia el proceso; al alcanzar de nuevo un tamaño de ventana igual a umbral(A), esta fase termina y se pasa al modo de evitación de la congestión. Si en lugar de un fin de temporización se detectan tres ACKs duplicados, TCP realiza una retransmisión rápida y entra en el estado de recuperación rápida.

2. Evitación de la congestión (congestion avoidance): ante la posibilidad de estar al borde de la congestión, en esta fase se aumenta la ventana de congestión en 1 MSS cada RTT (MSS / VentCongestion para cada ACK recibido). Ante un fin de temporización, esta fase transiciona al inicio de la de arranque lento, pero ante la recepción de tres ACKs duplicados fija umbral(A) = VentCongestion / 2 y actualiza la ventana de forma que VentCongestion = umbral(A) + 3 MSS. A continuación, se entra en el estado de recuperación rápida.

3. Recuperación rápida: en esta fase la ventana se incrementa en 1 MSS por cada ACK duplicado. Al recibir el ACK que falta, vuelve al estado de evitación de la congestión después de fijar VentCongestion = umbral(A); ante un fin de temporización, transiciona al inicio de la fase de arranque lento.

Control de congestión en TCP: retrospectiva: Ignorando la fase inicial de arranque lento, el control de congestión en TCP consiste en un incremento lineal (aditivo) de VentCongestión a razón de 1 MSS por RTT, seguido de un decrecimiento multiplicativo (división entre dos) del tamaño de la ventana cuando se reciben tres ACKs duplicados. Por esto, se le llama una forma de crecimiento aditivo y decrecimiento multiplicativo de control de gestión, que gráficamente presenta un comportamiento en forma de diente de Sierra.

3.7.2. Notificación explícita de congestión (ECN): control de congestión asistido por la red.

Recientemente, se han propuesto, implementado e implantado extensiones tanto a IP cómo a TCP que permiten a la red señalizar explícitamente la congestión a un emisor y un receptor de TCP. Para ello, en la capa de red se utilizan dos bits del campoTipo de servicio de la cabecera del datagrama IP. Los routers usan una de las posibles configuraciones para indicar que experimentan congestión. Esta indicación es transportada al host de destino, que a su vez informa al host emisor.