

FR-pero-bien-COMPLETO.pdf



Anónimo



Fundamentos de Redes



3º Grado en Ingeniería Informática



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

Aprende Inglés
Con nuestros cursos **GRATUITOS**
para desempleados

Pincha aquí e
inscríbete ya

Fórmate con nuestros cursos de Inglés para las titulaciones
A1, A2, B1 y B2 100€ subvencionados para desempleados



GOBIERNO
DE ESPAÑA

MINISTERIO
DE TRABAJO
Y ECONOMÍA SOCIAL

SERVICIO PÚBLICO
DE EMPLEO ESTATAL
SEPE



Estudiar sin publi es posible.



Compra Wuolah Coins y que nada te distraiga durante el estudio



Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



Aprende en 1 hora lo que no has aprendido en 4 meses de FR

Tema 1. Introducción.

La intención de este pdf es intentar explicar FR con muchos ejemplos y simplificando las cosas porque explicarle algo a alguien que ya lo sabe pues tiene poca gracia. No es un pdf para memorizar, si no para leer y comprender el tema 1.

¿Por qué las redes son importantes?

Internet es algo que todos usamos a día de hoy. En esta asignatura vamos a intentar aprender cómo funciona. Internet es la red de redes, así que vamos a explicar lo que es una red.

- Sistema de comunicación: todo lo involucrado en el intercambio de información, persona que manda mensaje, persona que lo recibe, canal de comunicación...
- Información: lo que mandas, un mensaje, una foto...
- Red: conjunto formado por dispositivos (ordenador, móvil, consola) que son capaces de enviar y recibir mensajes de manera **autónoma** (que lo hace el chisme solo, no tienes que hacer nada para que traduzca tus datos a bits o para pasar de bits a datos) y facilitan el intercambio **eficaz** (que funciona bien) y **transparente** (la información que traduce el dispositivo es la que se te muestra) de información.

¿Por qué queremos usar las redes? Porque se pueden compartir recursos, es escalable (se pueden añadir muchos dispositivos a una red), es fiable y robusta. Una ve hemos visto que las redes son la leche, vamos a clasificarlas.



WUOLAH

Clasificación de redes

Por escala:

- LAN (Local Area Network): una red de ordenadores (pc, móvil, consola, etc) que ocupa un área, como tu casa o una oficina.
- WAN (Wide Area Network): una red que conecta varias LAN, puede que estas LAN no estén en la misma ubicación física. Estas redes se usan en empresas o por proveedores de Internet (toman todas las LAN a las que dan servicio y las juntan en una red mayor).

Por tecnología de transmisión:

- Difusión o canal compartido (broadcast): hay un único canal de comunicación que comparten todos los dispositivos de la red. Los mensajes que se envían pueden llegar al resto de ordenadores de ese canal o si se especifica, a uno en concreto. Ejemplos de esto podría ser WiFi, los datos móviles y Bluetooth.
- Punto a punto: se hace una comunicación directa entre un par de ordenadores.

Estructura de una red

Ya hemos visto que hay un montón de redes, pero ¿qué tiene una red? Una red está formada principalmente por 2 cosas, hosts y una subred.

- Hosts: los que envían y reciben datos en la red (ordenadores, móviles, ...)
- Subred: infraestructura por la que se mandan los mensajes. Esta infraestructura está formada por 2 elementos:
 - ~ Nodos o elementos de commutación: routers/switches, la información va pasando por estos puntos hasta llegar a su receptor.
 - ~ Líneas de transmisión: los cables que conectan los routers/switches entre ellos.

Bueno, una vez hemos ya todo lo que se puede ver de una red, que si su clasificación, sus elementos, blablabla. ¿Cómo es posible que pueda enviarle un mensaje desde mi móvil a una persona y que le llegue? Porque mucho cable y mierda pero como le digo tkm a mi novia y le llega eso por un cable.

Estandarización de redes

A principio de los años 80, cuando empezó el boom de las redes, cada empresa enviaba los mensajes por la red como le daba la gana porque como se comunicaban únicamente los ordenadores de su red pues funcionaba. El problema llegaba cuando querían comunicarse dos empresas pues como cada una se comunicaba como le daba la gana, a la hora de comunicarse entre ellas, seguían unas instrucciones distintas y no funcionaba.

Entonces, llegó la ISO (International Organization for Standardization, gente que se dedica a decir cómo tienen que ser las cosas, como tu madre pero en grande), y dijo, esto de que cada uno manda los mensajes como le dé la gana se ha acabado. Propuso lo que se conoce como el modelo OSI (Open System Interconnection).

El modelo OSI

Aplicación
Presentación
Sesión
Transporte
Red
Enlace
Física

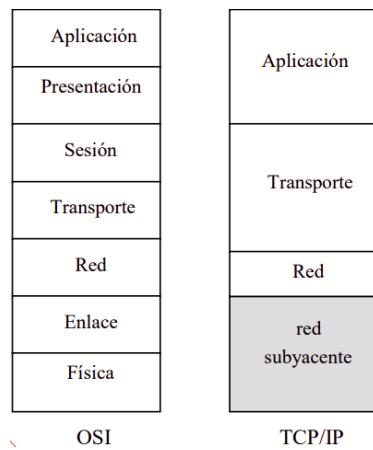
El modelo OSI se divide en distintas capas, cada una de estas capas va a realizar una función independiente de las demás capas que va a ser necesaria para la comunicación entre dos equipos. Las capas son las siguientes:

Para que un mensaje se transfiera de un dispositivo a otro, va a tener que pasar por estas capas para que vayan refinando el mensaje como tal para ser transmisible. Cada capa hace determinadas cosas, que explicaré más tarde. Para memorizar las capas se puede usar la regla mnemotécnica "FERTSPA" (pronunciación de primer spa en inglés, que es donde nos gustaría estar).

Pero el modelo OSI en realidad es un cuento, porque nadie lo utiliza, porque no era eficiente. Es un **modelo de referencia**, es decir, los modelos que funcionan de verdad se basan en este modelo pero este modelo como tal no se usa.

Entonces si este modelo no se usa, ¿cuál es el que usa en realidad? El modelo que se usa es el TCP/IP, está formado por dos protocolos (protocolo: serie de reglas que hay que seguir para que funcione), el TCP Y el IP. Este modelo agrupa en una capa, varias capas del modelo OSI.

Aquí vemos las agrupaciones que se hacen.



Función de cada capa del modelo OSI

Es un poco rollo pero al menos te enteras de qué hace cada capa.

Capa de aplicación (Nivel 7)

Es el nivel más cercano a las personas, las aplicaciones que usamos, interactúan con esta capa. Es decir, que el usuario no usa esta capa, el usuario lo que hace es usar un programa que interactúa con la capa de aplicación. Por ejemplo, yo utilizo un navegador y ese navegador no está en la capa de aplicación, pero los protocolos que usa para conseguir las páginas web (HTTP), sí es de la capa de aplicación.

Vamos a enumerar una serie de protocolos que pertenecen a esta capa. **Esto es importante porque luego en el examen preguntan a qué capa pertenece cierto protocolo.**

- HTTP (HyperText Transfer Protocol) el protocolo que se usa en www.
- FTP (File Transfer Protocol) para transferir ficheros.
- SMTP (Simple Mail Transfer Protocol) envío y distribución de correo electrónico.
- POP (Post Office Protocol)/IMAP: reparto de correo al usuario final.
- SSH (Secure SHell) para usar un terminal remoto
- Telnet es otro terminal remoto pero no se usa porque es una mierda.
- SNMP (Simple Network Management Protocol)
- DNS (Domain Name System)

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



En los temas próximos se van explicando estos protocolos así que don't worry.

Capa de presentación (Nivel 6)

Se encarga de presentar la información de forma que aunque los ordenadores tengan representaciones internas distintas de caracteres (ASCII, Unicode), números (little-endian, big-endian), sonido ... los datos lleguen de una manera reconocible para todos. Vamos, que realiza las conversiones de datos para interpretarlos correctamente. En esta capa también se puede cifrar y comprimir datos.

Capa de sesión (Nivel 5)

Establece, gestiona y termina las conexiones entre usuarios finales (las aplicaciones). LLeva a cabo una serie de servicios importantes:

- Control de la sesión entre el emisor y el receptor (quién envía datos, quién recibe datos, ...).
- Control de concurrencia (que dos comunicaciones críticas no se realicen a la vez).
- Mantener puntos de verificación (checkpoints): si se interrumpe la transmisión, que empiece desde este punto y no desde el principio, vamos como en los videojuegos.

Los firewalls funcionan sobre esta capa, aunque normalmente muchos servicios de esta capa son prescindibles.

Capa de transporte (Nivel 4)

Esta capa es la tocha, el tema 3 es entero sobre esto así que de momento vamos a dar una descripción breve.

Su función es aceptar los datos enviados por las capas superiores, dividirlos en partes más pequeñas (paquetes) y pasarlo a la capa de red. Debe aislar a las capas superiores de las distintas implementaciones de tecnologías de red con las inferiores, por lo que es muy importante.



En esta capa se dan servicios de conexión a las capas superiores que serán utilizados por las aplicaciones (en realidad por los protocolos que usan esas aplicaciones) de la red para enviar y recibir paquetes.

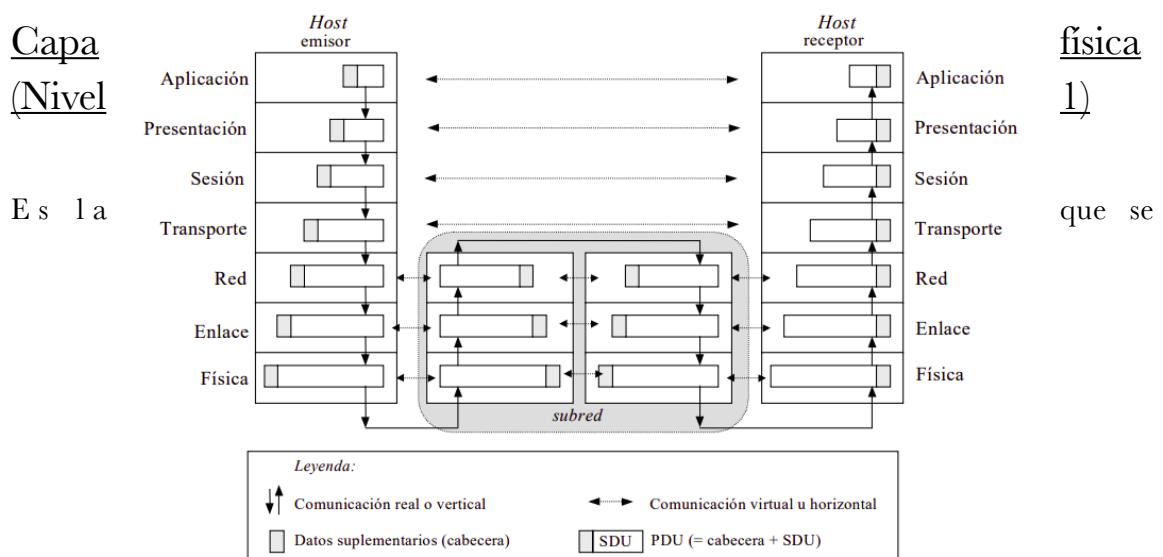
Capa de red (Nivel 3)

Esta capa se encarga de que los datos lleguen desde el origen al destino, incluso si no están conectados directamente entre ellos. Además, controla la congestión de red (cuando hay muchas peticiones en una red, no se pueden servir todas, como un atasco en la carretera).

Los routers trabajan en esta capa aunque si trabajan como switches trabajan en el nivel 2, también los firewalls actúan sobre esta capa. Por último, en esta capa se determina la ruta que deben seguir los datos para llegar a su destino y quién será su receptor (IP).

Capa de enlace de datos (Nivel 2)

Debe reconocer los límites de los paquetes y resolver los problemas del deterioro, pérdida y duplicados. Esta capa se encarga del direccionamiento físico, de la topología de red (la forma en la que están conectados los equipos), de la notificación de errores, distribuir de manera ordenadas los paquetes y controlar el flujo. Los switches funcionan aquí.



encarga de las conexiones físicas del ordenador hacia la red (cable, infrarrojo, etc). Se encarga de transmitir los bits de información (de los paquetes) a través del medio utilizado para la transmisión (mira si es full duplex y to eso).

Es un tocho gordo, pero básicamente esta asignatura se dedica a hablar de las distintas capas así que es un resumen general de la asignatura.

Comunicación real y virtual modelo OSI

La comunicación funciona de la siguiente manera. El emisor envía unos datos desde la capa de aplicación, cada capa va añadiendo una serie de cabeceras que llevan cierta información sobre los datos transportados hasta llegar a la capa física. Estos datos se envían junto a todas las cabeceras por una subred y llega a la capa física del receptor. A partir de aquí, cada capa va quitando las cabeceras añadidas por su similar en el envío. La capa física del receptor quita la cabecera de la capa física del emisor, etc. Este procedimiento se llama encapsulado y desencapsulado.

La comunicación real es la que hay entre las capas que se envían mensajes entre ellas, (en el dibujo las flechas que son verticales) y la comunicación virtual es para quien va el mensaje que envía cada capa (su símil en el receptor).

Retrasos en la comunicación

Como cualquier cosa en este mundo, no es instantáneo, hay una serie de retardos. En la comunicación entre equipos, son los siguientes:

- 1.- Transmisión: Tiempo de transmisión = Longitud a enviar (bits) / Velocidad transmisión (bits por segundo)
- 2.- Propagación: Tiempo de propagación = Distancia (metros) / Velocidad propagación (metros/segundos)
- 3.- Procesamiento (interno) + retardo en la cola + acceso al medio

Tipos de servicios

Vamos a distinguir una serie de servicios, por una lado tenemos:

- Orientado a conexión (SOC): se debe establecer una conexión antes de transferir datos.
- No orientado a conexión (SNOC): un mensaje puede ser enviado desde un punto a otro sin un previo acuerdo.

Y por otro lado tenemos:

- Confirmado (fiable): envía una confirmación (ACK o acknowledge) para confirmar que ha recibido el mensaje.
- No confirmado (no fiable): no confirma que haya recibido el mensaje.

¿Cómo funciona Internet?

Internet, es una serie de ordenadores conectados mediante cables, fin. Las empresas tienen almacenes de servidores repartidos por el mundo, en estos servidores se guardan datos que están en Internet, una página web, vídeos, etc. Estas empresas se llaman ISP (Internet Server Providers), y dependiendo de lo grandes que sean y cómo estén distribuidas a lo largo del mundo pueden ser Tier 1, Tier 2, Tier 3. Las Tier 1 son las reinas del mambo y las que ganan millones pero bien, las Tier 2 están bien y las Tier 3 son unas desgraciadas. Más adelante explico el porqué.

Estas empresas, en un principio, es evidente que solo va a tener conectados sus propios servidores. Por ejemplo, Movistar tiene una serie de servidores repartidos por todo el mundo, y en estos servidores están alojadas las páginas web de universidades de toda Europa (ejemplo). Pues si mi ISP es Movistar podré acceder a todas las páginas webs de universidades de Europa, pero ¿y si quiero acceder a una página web de una universidad de Estados Unidos, que no está en los servidores de Movistar? Uno puede acceder a cualquier página de Internet así que algo falla.

Hay acuerdos entre estas empresas (ISP) para poder acceder a los servidores de la otra. Estas conexiones a servidores de otra empresa se realizan en un lugar físico llamado punto neutro o IXP (Internet Exchange Point) donde se enchufan los cables de unas a otras. Pero claro, yo no voy a dejar que te enganches a mis servidores así por tu cara bonita, hay una serie de acuerdos que dependiendo del Tier que sean las empresas se harán de una forma o de otra.

- ISP Tier 1: participa a través de acuerdos de interconexión libre, llamado **peering** libre. Básicamente, tú te puedes meter a todos mis servidores y yo a los tuyos y ninguno de los dos pagamos.

Básicamente, esto. Hay 17 ISP Tier 1, que son los que mandan, porque cobran a sus clientes por el servicio prestado, pero estas empresas no le tienen que pagar a otra para acceder a sus servidores. De España hay 1, Telxius que es de Telefónica, ole ahí mi Españita.



Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



- ISP Tier 2: cubren una región específica y pueden pasar 2 cosas, que llegue a un acuerdo de peering con algunas empresas o que le tenga que pagar una cantidad a un Tier 1 para acceder a sus servidores. Entonces, están bien porque no tienen que pagar por todo pero ya las Tier 1 se benefician de estas pues le tienen que pagar por acceder a algunos servidores. Esto de pagar se llama acuerdo de tránsito.
- ISP Tier 3: son las pobreticas de las empresas que tienen que pagar si o si para tener un acceso al resto de servidores. Estos ISP solo tienen acuerdos de tránsito, ni peering ni mierdas, a pagar.



Es por eso que Movistar, Orange,..., nos cobran una tarifa. Si son Tier 1, es puro beneficio, pero de otra forma, con parte del dinero que le pagamos pues le pagan a empresas Tier 1 para acceder a otros servidores. El negocio está en cobrarnos menos de lo que les cuesta a ellos sus acuerdos de tránsito con otras empresas.

Las redes con las que se conectan estas empresas con clientes se llaman redes de acceso y hay de distinto tipo: xDSL, RDSI, FTTH. Las redes con las que se conectan los grandes operadores se llaman redes troncales y hay distintas: ATM, SDH, SONET, MPLS. Esto tampoco es muy importante pero por si pregunta el profe.

Y el tema 1 ya se ha acabado, espero que se os haya hecho entretenido y que entendáis algo un poco mejor esto. Cuando haga, si es que lo hago, el tema 2 lo subiré.



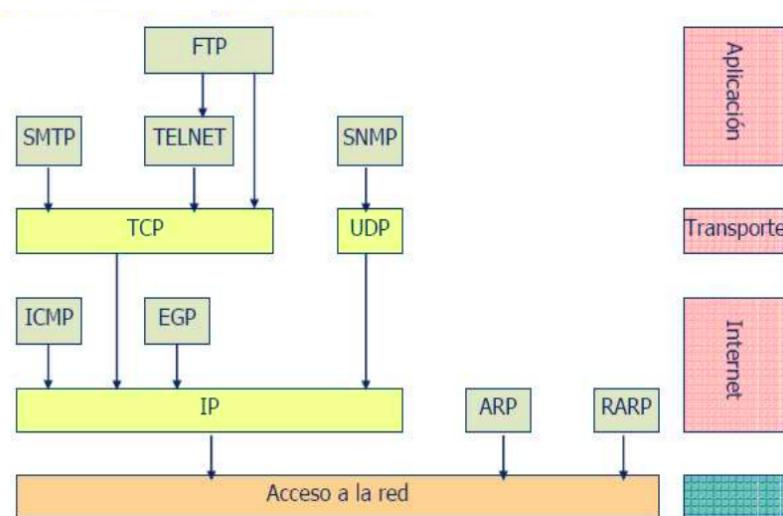
Aprende en 1 hora lo que no has aprendido en 4 meses de FR

Tema 2. Servicios Y Protocolos De Aplicación En Internet.

La intención de este pdf es intentar explicar FR con muchos ejemplos y simplificando las cosas porque explicarle algo a alguien que ya lo sabe pues tiene poca gracia. No es un pdf para memorizar, si no para leer y comprender el tema 2.

Las aplicaciones de red

En este tema se va a hablar sobre la capa de aplicación vista en el tema anterior. La capa 7 en el modelo OSI y la 4 en TCP/IP. Es importante recordar que la capa de aplicación no es usada por aplicaciones como podría ser Google Chrome, estas aplicaciones hacen uso de una serie de protocolos que sí forman parte de esta capa como podría ser HTTP, SMTP, SNMP, etc. Aquí podemos ver algunos protocolos perteneciente a cada capa.



La IP

Para empezar, vamos a decir qué es una IP. Es un conjunto de números que identifica una interfaz de red. Importante no confundir el numerito de IP con el protocolo IP de TCP/IP. Con este numerito podemos identificar de manera única una red. Hay dos tipos de IP, IPv4 e IPv6.

La IPv4 identifica a una red con 32 bits en 4 octetos separados por puntos. Es decir, va desde 0.0.0.0 hasta 255.255.255.255, lo que da un total de 4 mil millones de direcciones IP públicas. Esto en su día, pues se ve que les pareció bien a los señores que diseñaron esto y que Internet no iba a ser tan popular como para agotar estas direcciones (recordamos que son únicas, no hay otra como esa en el mundo). Entonces, pasaron los años e Internet dijo:



Ahora casi están agotadas pues cualquier router y servidor necesita una IP pública. Para arreglar la cagada que hicieron pues surge lo que es la IPv6, que se trata de una dirección hexadecimal de 128 bits, divididas en 8 secciones. Entonces IPv6 va desde 0:0:0:0:0:0:0:0 hasta ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff. Esto da lugar a 2^{128} direcciones distintas, unas 340 sextillones. Al menos, ahora si que fueron previsores.

Su función es sustituir a IPv4, pero actualmente se utiliza tanto IPv4 como IPv6 porque no es tan fácil cambiar todas las direcciones asignadas de IPv4. Vamos que porque la cagaron pues ahora tenemos que usar IPv4 e IPv6. Conclusión: mejor que sobre a que falte.

Todo esto que hemos hablado ha sido sobre IPs públicas, es decir, la que identifica a nuestro router desde el exterior. Pero también existen las IPs privadas, que son las direcciones que asigna el router a cada uno de los dispositivos conectados a él. Las IPs privadas las podemos cambiar si queremos. Por tanto, para resumir, nuestro router tiene una IP pública que lo identifica y por tanto, a todos los dispositivos que estén conectados a él. Estos dispositivos

tienen una IP privada que los identifica en una red privada. Una vez contado todo este lío, podemos seguir.

Interacción cliente/servidor

Un servidor es básicamente un ordenador que da servicios a otro ordenador, de ahí la palabra servidor. Los servidores tienen ciertas características:

- Siempre en funcionamiento, si falla un servidor no nos podrá devolver los datos en su interior y a nadie le gusta que fallen las cosas.
- IP permanente y pública: aquí hay 2 cosas a explicar, primero pública, pues si tenemos un servidor es para que se pueda acceder a él y por tanto su identificador (IP) es público. Lo segundo es que sea permanente, esto se hace para poder identificar fácilmente el servidor al que queremos acceder. Por ejemplo, Google tiene una IP 73.67.23.45 (Invención total pero nos imaginamos que es esa), pues esa IP será siempre la de Google para acceder de forma fácil.
- Agrupados en granjas: ponen muchos servidores en una nave industrial porque es rentable.

Un cliente es el ordenador al que da servicios un servidor, es decir, nosotros. Y tiene las siguientes características:

- Funcionando intermitentemente: el ordenador se enciende y se apaga.
- Tienen una IP dinámica y privada: lo de privada lo hemos visto arriba, pero, ¿qué significa dinámica? Estamos hablando de la IP pública, pues nuestro router, al contrario que un servidor, no tiene siempre la misma IP pública (no somos tan importantes como para tener siempre la misma :(). En realidad, esto pasa porque nuestro operador, por ejemplo Movistar, compra una serie de IPs para darnos una y poder entrar a Internet. Si tiene 1 millón de clientes, es evidente que no va a comprar 1 millón de IPs porque primero, no es que sobren las IPs y eso cuesta su dinero (la empresa quiere ganar dinero), y segundo, su millón de clientes no van a entrar a Internet a la vez. Entonces lo que hace es comprar por ejemplo 700 000 IPs y las va asignando de manera que cuando alguien necesite entrar en Internet, se la asigne. Si quieras ver tu IP pública puedes meterte a alguna web y ver como va cambiando a lo largo del tiempo.
- Los clientes se comunican con el servidor pero no entre ellos.

La comunicación básica es que un cliente inicia la comunicación con el servidor y el servidor espera que a ser contactado. (Resumen, el servidor hace la estrella de mar). Para poder comunicarse, tanto el cliente como el servidor envían/reciben mensajes desde su **socket**, y

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



para recibir mensajes debes tener un identificador (IP (el numerito que identifica la red) + puerto (un número que identifica la aplicación que se está usando)).

La interfaz socket

Un socket es un descriptor de una transmisión con el que se pueden enviar/recibir información hacia otra aplicación. En resumen, es una "tabla" con información para poder comunicarte. El socket es una puerta que une la capa de aplicación con la de transporte. (Según el protocolo TCP/IP).

En la práctica, es un puntero a una estructura. Vamos, que es un puntero a distinta información que hace falta para comunicarse.

¿Qué es y qué define un protocolo?

Importante que luego preguntan en el examen sobre esto. Es un poco rollo pero bueno.

Un protocolo es un conjunto de reglas e instrucciones a tener en cuenta para que funcione un determinado aspecto. Un protocolo está definido por lo siguiente:

- **Tipo de servicio:** (Explicado en el tema 1)
 - ~ Orientando o no orientado a conexión
 - ~ Confirmado o no confirmado
- **Tipo de mensaje:**
 - ~ Por ejemplo, si es una **solicitud** o una **respuesta**.
- **Sintaxis:** Define y estructura los **campos** que va a tener un mensaje.
 - ~ Protocolos orientados a texto (**HTTP**)
 - ~ Protocolos no orientados a texto (**DNS**)
 - ~ Como tendencia se usa el formato Type-Length-Value (TLV), con esto podemos representar información de forma que haya presencia opcional y longitud variable. Un inconveniente es que su formato es difícil entenderlo por personas. Tiene bastantes ventajas:
 1. Es fácil de interpretar.
 2. Se pueden añadir etiquetas sin hacer incompatible lo anterior.
 3. Menor redundancia de datos que XML.
 4. Un formato TLV puede ser a su vez otro TLV.



- **Semántica:**
 - ~ Es el **significado de los campos** definidos por la sintaxis.
- **Reglas:**
 - ~ Especifican **cuándo y para qué** se envían o responden mensajes.

Tipos de protocolos

Existen distintas clasificaciones para los protocolos, vamos a verlas:

- Protocolos de **dominio público** (HTTP, SMTP) son protocolos disponibles para todo el mundo vs **protocolos propietarios** (Skype, IGRP) que son de empresas privadas.
- Protocolos **in-band**, son aquellos en el **control de datos está regulado** y pasa el control de los datos en la misma conexión que los datos principales (HTTP y SMTP) vs **out-of-band** que no realizan esto (FTP).
- Protocolos **stateless** que trata cada petición como independiente y no tiene relación con la solicitud anterior, no tiene que tener en cuenta el estado de los que se mensajean (HTTP e IP) vs **state-full** que sí lo tiene en cuenta.
- Protocolos **persistentes** en los que la conexión permanece abierta por un periodo y puede ser reutilizada vs **no-persistentes**.

Tipos de protocolos

Las aplicaciones tienen una serie de requisitos distintos entre ellas. Algunos de estos son:

- **Tolerancia a pérdidas de datos:** algunas apps pueden tolerar pérdidas de datos (si una canción se corta 1 segundo no pasa nada), pero hay otras que necesitan que la transferencia sea 100% fiable (FTP, Telnet, HTTP).
- **Exigencia de requisitos temporales:** algunas apps necesitan que lleguen rápido los datos (si estoy hablando por Discord quiero que me llegue el audio lo antes posible para que parezca una conversación), estas apps son inelásticas, otras no necesitan tanta latencia.
- **Demandas de ancho de banda (tasa de transmisión o throughput):** hay apps que requieren envío de datos a una tasa determinada (codec de vídeo).
- **Niveles de seguridad:** los requisitos de cada app dependen (encriptación, autenticación, no repudio, integridad...)

Aquí pongo una tabla de los requerimientos de algunas aplicaciones, que luego esto cae en el examen y te deja ko.

Application	Data loss	Throughput	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's ms
stored audio/video	loss-tolerant	same as above	yes, few s
interactive games	loss-tolerant	few kbps up	yes, 100's ms
instant messaging	no loss	elastic	yes and no

Protocolos de red

Voy a hablar un poco sobre las características sobre TCP y UDP, que son protocolos de transporte. No obstante, en el tema 3 se habla mucho de esto. Aquí solo hay que entender las características que tienen porque esto 100% cae en el examen.

TCP

- Orientado a conexión: tanto el emisor como el receptor deben conectarse. Es como si fuera una llamada telefónica.
- Transporte fiable con control de errores: si pierde algún paquete por el camino, lo recupera por lo que llega toda la información que se envía.
- Control de flujo: es necesario para que el emisor de la información no sature el buffer del receptor. (Si yo tengo que leer algo y tú me escribes más rápido de lo que yo leo pues me saturo).
- Control de congestión: Intenta que no haya un colapso por una cantidad muy grande de paquetes. (Si tengo que leer lo que me mandas, si me mandas un mensaje de 1 línea pues lo leo al momento, pero si me envías tu drama en 50 mensajes del tirón pues voy a tardar un rato en leerlos, así que manda menos).

Vamos que TCP es la leche porque todo lo que mandas le llega al otro, y controla que se haga todo de la manera más fluida posible. Problema: es lento, para mandar un correo pues está bien, pero si yo quiero ver un vídeo a 4k 60fps y cada vez que pierde un pixel tiene que reenviar el paquete, capaz me veo el vídeo en un año. Para esto existe el siguiente servicio, UDP.

UDP

- No orientado a conexión: no tienen que conectarse el emisor y el receptor. Es como si te mando un mensaje, no hace falta que estés ahí para obtener esa información.

- Transporte no fiable: se la suda si pierda algo, queremos que UDP vaya rápido no que vaya bien. No obstante, tiene una política de Best-Effort, es decir, que intenta hacerlo lo mejor que puede.
- Sin control de flujo: mucho lío si hay que ver cuando se satura el receptor, si no hay control de flujo tarda menos.
- Control de congestión: lo mismo que arriba.

Tanto TCP como UDP al ser usuarios del protocolo IP no garantizan Calidad de Servicio (QoS), esto quiere decir:

- El retardo no está acotado: aunque uno vaya más rápido que otro, no hay un máximo de tiempo que puedan tardar en hacer lo suyo.
- Las fluctuaciones en el retardo no están acotadas: nunca sé si me va a llegar 1 mensaje o me van a llegar 50 de golpe.
- No hay una velocidad de transmisión mínima garantizada: si tienen que tardar más por lo que sea pues va a tardar lo que necesite, no agobies a los protocolos hombre.
- No hay una probabilidad de pérdidas acotada: nunca sabemos cuantos paquetes se van a perder, independientemente si luego se vuelven a reenviar.
- No hay garantías de seguridad.

Una vez contado todo esto, vamos a estudiar una serie de aplicaciones de red. Esta tabla es importante porque cae en el examen. Nos dice el protocolo que siguen las distintas aplicaciones.

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



Servicio de Nombres de Dominio (DNS)

Con lo que sabemos, para buscar en Internet, tendríamos que saber la IP de cada servidor para recibir los recursos que tiene almacenados. Pero a los seres humanos pues no se nos da bien memorizar números. Entonces a estas IPs les vamos a asignar un nombre, por ejemplo, para acceder a Google, en vez de poner 73.46.25.98 (invención total) pues ponemos www.google.com y tiene el mismo efecto.

El Domain Name System (DNS) es el encargado de realizar una traducción de nombres de dominio (nombres de dominio = palabra que le damos a la IP) a direcciones IP:

www.google.com <--> 73.46.25.98

Esto es como cuando llega un extranjero con un nombre impronunciable y lo llamas Paco.

El espacio de nombres de dominio tiene una estructura jerárquica en forma de árbol, en el que su dominio raíz (root) es ". ". Estos dominios tienen la forma:

parteLocal.dominioN.dominio2.dominio1

A los dominios de nivel 1 se les llama Top Level Domains (TLD) (estos son los tochos que conoce todo el mundo) (.com, .es, .edu, etc). Importante, el dominio raíz y un TLD no son lo mismo.

Bueno, todo esto está muy bonito que si la traducción y tal pero [¿cómo funciona el DNS?](#)

Cuando preguntamos por www.google.com lo que hace nuestro ordenador es lo siguiente, mira si tiene su IP en la caché del ordenador (porque si lo has buscado hace poco, pues se guarda un tiempo ahí para no tener que hacer todo el proceso y así entras antes a la página web y todos queremos eso).

Si no está en la caché del ordenador, le pregunta a un servidor local, que es un servidor de una compañía como por ejemplo Movistar, a un nivel un poco más grande como podría ser una ciudad. Si alguna persona del área que cubre este servidor ha buscado hace poco esa página, pues estará en el servidor local y no tendrás que esperar mucho para encontrar la página.

Y ya si nadie de tu ciudad ha buscado eso anteriormente, se hace lo que se llama **resolución**. Pues el servidor local de Movistar, Orange, etc, sabe a quién preguntarle si no tiene esa IP. Primero le pregunta a los servidores raíz, (el que era ". ", hay 13 servidores raíz en todo el mundo, nombrados de la A a la M), y estos servidores conocen la información de los TLD (.com, .es, .edu, etc. Una vez estamos en el TLD busca la IP que le hemos pedido. Si es



un sitio famoso como google, pues lo va a encontrar y nos lo va a devolver, pero puede ser que www.ugr.es no lo encuentre. En el TLD está .es pero no .ugr, lo que se hace es que el TLD nos va a mandar a otro servidor, llamado servidor autorizado donde tendrá una serie de dominios loquesea.es. Y así se hace sucesivamente hasta encontrarla. Una vez encontrada nos mandan la IP asociada a ese nombre y se guardará en la caché de nuestro ordenador por si nos queremos meter otra vez dentro de poco.

Hay 3 tipos de resoluciones: iterativa, recursiva y mixta (lo pongo por si lo preguntan). Es un poco largo esto pero bueno fuerza, al menos, ya sabemos como obtenemos la IP dado su nombre. Ahora nos falta especificar un poco cómo son estos servidores y eso.

Base de datos DNS

Los servidores DNS son bases de datos principalmente.

- En la base de datos se guardan ficheros de texto (.txt) y se les llama **zone files**.
- Cada zone file tiene unos registros que se llaman **Resource Record (RR)** (información sobre el dominio y la IP).
- Pero claro, puede ser que los servidores cambien de IP por lo que sea, y si en la base de datos está guardada una dirección IP no válida pues la has liado porque no te lleva a ningún sitio. Para esto existe el llamado **Time To Live (TTL)**, que es el tiempo de validez que tienen los RR, cada cierto tiempo se actualizan los datos.
- Si os interesa la información que guarda los RR, es esta:

Nombre del dominio: nombre del dominio al que se refiere el RR.

Clase: en Internet siempre IN.

Tipo: Tipo de registro.

SOA	Registro (S tart O f A uthority) con la autoridad de la zona.
NS	Registro que contiene un servidor de nombres.
A	Registro que define una dirección IPv4.
MX	Registro que define un servidor de correo electrónico.
CNAME	Registro que define el nombre canónico de un nombre de dominio.
HINFO	Información del tipo de máquina y sistema operativo.
TXT	Información del dominio.
PTR	Registro que contiene un nombre de dominio (para resoluciones inversas)

Valor: Contenido que depende del campo tipo

- También hay una base de datos que hace una resolución inversa, es decir, traduce las direcciones IP a nombres de dominio.
- DNS se ofrece en el puerto (puerto = numerito que identifica el protocolo o la aplicación que se usa en el ordenador) 53 mediante UDP o TCP (para respuestas > 512 bytes).

Pues ya hemos acabado con el DNS, ya falta menos.

La navegación Web

Una página web es un fichero formado por objetos, como podría ser contenido en HTML, imágenes, vídeos... Cada objeto se puede direccionar con una URL, por ejemplo, pagina/index.html va al html, pero por ejemplo pagina/foto3.jpg va a una foto.

Las páginas webs usan el protocolo **HTTP** (Hyper Text Transfer Protocol) (importante que esto luego cae) en el que se adopta un **modelo cliente-servidor**.

- **Cliente:** buscador (Google Chrome, Mozilla, Opera,...) que solicita, recibe y muestra objetos web.
- **Servidor:** el que le manda los objetos web al cliente y donde están almacenadas las páginas webs y eso.

Hay páginas webs que pueden ser **estáticas** (que no cambia su contenido como podría ser un pdf) o **dinámicas** (que cambia su contenido, como YouTube que cada vez que lo abres es distinto). Para ser dinámicas se pueden usar lenguajes de scripting en el cliente (JavaScript) o scripting en el servidor (Perl, PHP).

Vamos a ver las características del protocolo HTTP, esto luego preguntan cosas modo si es in-band, y cosas así.

Características del protocolo HTTP

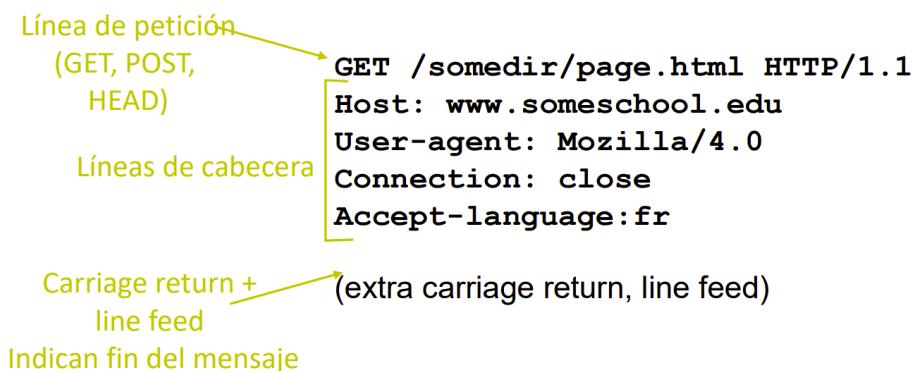
- **HTTP se basa en TCP** (el que no falla pero va lento, y es orientado a conexión) en el **puerto 80** (puerto = numerito que identifica a la aplicación dentro de un ordenador).
- **HTTP es "stateless"** (no tiene en cuenta las peticiones anteriores), pero usa **Cookies** para simplificar la interacción. El servidor no mantiene información sobre las peticiones de los clientes y así se ahorra recursos (a nadie le gusta gastar).
- **HTTP es un protocolo in-band** (se va haciendo un control de los datos en la misma conexión que por donde va la información principal), **orientado a texto** (forma que tienen los paquetes que manda, cabeceras, información , etc).
- Existen dos tipos de servidores HTTP:
 - ~ **No persistente:** se envía solo un objeto en cada conexión TCP.
 - ~ **Persistente:** se pueden enviar varios objetos en cada conexión TCP.

Cómo funciona HTTP?

- 1.- El cliente HTTP (Google Chrome, ...) pide un objeto web identificado por la URL (por ejemplo www.ugr.es/pages/Universidad, por defecto es index.html).
- 2.- Le pedimos al DNS la IP de www.ugr.es (como hemos visto arriba, hace todo ese proceso tocho).
- 3.- El DNS nos devuelve la IP que le hemos pedido, en este caso 150.214.204.231.
- 4.- El cliente (Google Chrome, ...) abre una conexión TCP (recordamos que es orientado a conexión y por tanto tiene que estar conectado) al puerto 80 de 150.214.204.231 (3 bandas).
- 5.- Una vez se establece la conexión entre el cliente y el servidor (donde están almacenadas las páginas webs y todo eso), el cliente le hace una petición GET /pages/universidad/
- 6.- El servidor le envía lo que le pide en la misma conexión TCP.
- 7.- Como es TCP que era la polla pues es una comunicación transparente y fiable.
- 8.- Si HTTP es del tipo persistente, puede hacer más peticiones GET.
- 9.- Se cierra la conexión TCP y se liberan los recursos.
- 10.- Ya puedes ver la maravillosa página de la UGR en tu portátil.

Mensajes HTTP

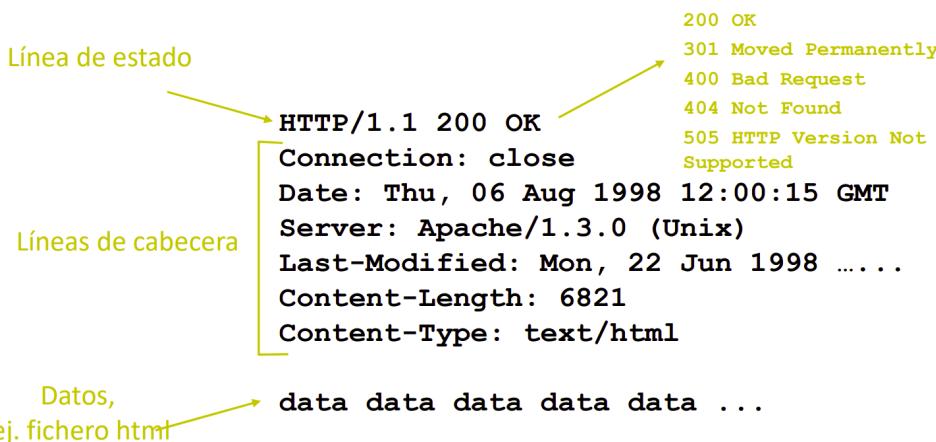
Existen dos tipos de mensajes en HTTP, request (para pedir) y response (para dar).



- HTTP request message son solicitudes que le hace el cliente al servidor, y tiene esta forma:

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



- HTTP response message son respuestas que le da el servidor al cliente, y tienen esta forma:

Evidentemente, los clientes pueden hacer algo más que GET, hay una serie de acciones distintas llamadas **métodos**. Aquí algunos:

- **OPTIONS**: solicitud de información sobre las opciones disponibles
- **GET**: solicitud de un recurso (puede ser condicional)
- **HEAD**: igual que GET pero el servidor no devuelve el "cuerpo" sólo cabeceras
- **POST**: solicitud al servidor para que acepte y subordine a la URI especificada, los datos incluidos en la solicitud
- **PUT**: solicitud de sustituir la URI especificada con los datos incluidos en la solicitud.
- **DELETE**: solicitud de borrar la URI especificada.

Para los códigos de respuesta del servidor al cliente también hay distintos valores. En el ejemplo de antes era 200 que significa OK. Estos se llaman **códigos de respuesta**, aquí algunos:

- **1xx** indican mensajes exclusivamente informativos
- **2xx** indican algún tipo de éxito
- **3xx** redireccionan al cliente a otra URL
- **4xx** indican un error
- **5xx** indican un error



También hay una serie de cabeceras que se mandan en los mensajes entre cliente y servidor, pero eso si te apetece verlo te lo ves en las diapositivas, que son muchas y rezaremos para que no pregunte nada de eso ;).

Web caché

¿Os acordáis que el ordenador tenía una caché DNS por si tenía que acceder a una dirección web dentro de poco y así no tener que hacer todo el proceso de buscarla? Pues con las páginas web lo mismo. Hay una caché en el ordenador para que no tenga que cargar la misma imagen cada vez que te metes porque se tarda mucho.

El usuario configura su navegador para todas las solicitudes cursen por proxy/caché. La caché puede residir dentro del ordenador y el navegador enviará todos los requerimientos HTTP a la caché, si el objeto está lo devuelve, si no está lo pide al servidor, actualiza la caché con el objeto y le da el objeto al cliente. Evidentemente, las cosas de la caché tienen un TTL (tiempo de vida) y se van actualizando. Un ejemplo de respuesta de un servidor a la caché es el siguiente:

HTTP/1.1 200 OK
Date: Fri, 30 Oct 1998 13:19:41 GMT

Server: Apache/1.3.3 (Unix)

Cache-Control: max-age=3600

Expires: Fri, 30 Oct 1998 14:19:41 GMT

Last-Modified: Mon, 29 Jun 1998 02:28:12 GMT

ETag: "3e86-410-3596fbcc"

Content-Length: 1040

Content-Type: text/html

Las cabeceras se asocian al fichero en la caché local. Si quieres ver más información sobre las cookies mira las diapositivas, que viene bien explicado (raro es).

Para acabar, hay que añadir que HTTP no es seguro, pero incluye cabeceras (authenticate y authorization) para restringir el acceso a recursos. Es vulnerable a ataques por repetición.

El correo electrónico

El correo electrónico está formado principalmente por dos cosas, el **agente de usuario** (MUA) y el **servidor de correo** (MTA).

- El agente de usuario (MUA): es la aplicación que te permite recibir y enviar correos, es con lo que interactúas (Outlook, Thunderbird). Hay que diferenciar una aplicación como tal que podría ser Outlook (un programa dedicado exclusivamente al correo electrónico), con un servicio de correo web (Gmail, Yahoo, etc) que son páginas webs y no aplicaciones como tal. Ambas son MUAs, pero Gmail es una página web y Outlook es un programa.
- Servidor de correo (MTA): es el encargado de enviar y recibir correos a otros ordenadores.

El protocolo (reglas que se siguen) de envío de correos es **Simple Mail Transfer Protocol (SMTP)**. El protocolo de descarga o lectura de correos puede ser **POP3, IMAP, HTTP**. No hay que confundir los protocolos con MUA y MTA.

Características de SMTP

Importante que luego preguntas mierdas de estas. SMTP (es un protocolo) está implementado mediante dos programas (incluidos en ambos servidores de correo, tanto el que envía el correo como el que lo recibe).

- Cliente SMTP: se ejecuta en el servidor de correo (MTA) que está enviando el correo.
- Servidor SMTP: se ejecuta en el servidor de correo (MTA) que está recibiendo el correo.
- **SMTP usa TCP en el puerto 25** (numerito que identifica el protocolo que se usa en el ordenador). Es un protocolo **orientado a texto**.
- SMTP es un protocolo **orientado a conexión** (SoC, que tienen que estar conectados clientes y servidor), es **in-band** (se va haciendo un control de los datos en la misma conexión que por donde va la información principal) y es **state-full** (tiene en cuenta con la solicitud anterior). Como es orientado a conexión, tiene 3 fases:
 - ~ Handshaking (saludo): que empieza la conexión.
 - ~ Transferencia de mensajes: se mandan lo que se tengan que mandar.
 - ~ Cierre: se cierra la conexión.
- La interacción entre cliente SMTP y servidor SMTP se realiza mediante comandos y respuestas:
 - ~ Comandos: texto ASCII.
 - ~ Respuestas: código de estado y frases explicativas.
- Al principio, los mensajes estaban codificados en ASCII de 7 bits pero con la definición de las extensiones MIME se puede enviar ASCII de 8 bits.

Pasos para enviar/recibir un correo

- 1.- El emisor del mensaje mediante su Agente de Usuario (MUA, puede ser Gmail, Outlook...) escribe un mensaje dirigido a la dirección de correo electrónico del receptor.
- 2.- Se envía con SMTP (o HTTP) el mensaje al servidor de correo (MTA, el que va a enviar el mensaje al servidor del receptor) del emisor, y pone el correo en una cola de mensajes salientes.
- 3.- El cliente SMTP abre una conexión con TCP con el MTA (el servidor del receptor, lo obtiene mediante DNS) del receptor.
- 4.- El cliente SMTP envía el mensaje en conexión TCP.
- 5.- El servidor de correo del receptor (MTA), ubica el mensaje en el una caja de correos del usuario destino.
- 6.- El receptor entra en su Agente de Usuario (MUA que puede ser Gmail, Outlook...), y para leer el mensaje se usa POP3, IMAP o HTTP.

Una representación es la siguiente:



Hay una serie de comandos que pueden enviar el cliente y el servidor en función de lo que quieran hacer, pero eso viene en los apuntes una pedazo lista de comandos.

Multipurpose Internet Mail Protocol Extensions (MIME)

Es la manera estándar de mandar contenido a través de la red. Los tipos MIME especifican el tipo de datos que se envían, como texto, imagen, audio... MIME adjunta a cada archivo una cabecera donde se indica el tipo de contenido que es, de esta manera, luego se puede presentar de manera correcta.



Hay una serie de cabeceras en las diapositivas pero son muchas y no se que pretenden poniéndolas porque se las va a estudiar su prima. Pero vamos que son una pila de diapositivas así que algo preguntarán de ahí.

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



IMAP, POP3 y Web Mail

- POP3 descarga los mensajes eliminándolos del servidor. Los mensajes de correo electrónico ya no se encuentran disponibles por correo web o un programa de correo. Vamos que los mensajes se pierden.
- IMAP permite organizar en carpetas en el lado del servidor (MTA), vamos que se ordena. Para ordenarlos mantiene información entre sesiones (con flags que ponen en la cabecera de los mensajes), permite descargar partes de los mensajes. Y permite acceder con varios clientes a un mismo buzón (POP también pero solo en modo guardar y cerrar).
- Web Mail tiene una organización total en el servidor, y se puede acceder desde cualquier cliente con HTTP (así que podemos acceder al correo desde cualquier lado). Y tiene un punto de seguridad mediante HTTPS.

Una lista de los puertos (numerito que identifica el protocolo que se usa dentro del ordenador) relaciones con el email es esta, por si lo preguntan vamos:

- POP3, puerto 110.
- IMAP, puerto 143.
- SMTP, puerto 25.
- HTTP, puerto 80.
- Secure SMTP (SSMTP), puerto 465.
- Secure IMAP (SIMAP), puerto 585.
- IMAP4 con SSL (IMAPS), puerto 993.
- Secure POP3 (SSL-POP3), puerto 995.

Ya hemos acabado con el email, ahora vamos a ver un poco de seguridad.

Introducción a los protocolos seguros

Una red es segura cuando se garantizan todos los aspectos, spoiler: no hay protocolos ni redes 100% seguros. La seguridad cubre múltiples aspectos, entre ellos:

- **Confidencialidad/privacidad:** el contenido de la información es comprensible sólo por entidades autorizadas.
- **Autenticación:** las entidades son quien dicen ser.



- **Control de accesos:** los servicios están accesibles sólo a entidades autorizadas.
- **No repudio o irrenunciabilidad:** el sistema impide la renuncia de la autoría de una determinada acción. (Que no puedas hacer algo y luego digas yo no he sido).
- **Integridad:** el sistema detecta todas las alteraciones de la información.
- **Disponibilidad:** el sistema mantiene las prestaciones de los servicios con independencia de la demanda.

La seguridad se debe situar en **TODAS** las capas, el grado de seguridad lo fija el punto más débil. Un ataque de seguridad es cualquier acción intencionada o no que perjudica cualquiera de los aspectos dichos arriba. Existen muchos tipos de ataques informáticos, entre ellos:

- ~ Sniffing: vulneración a la confidencialidad, escuchar, husmear.
- ~ Spoofing (phising): suplantación de la identidad.
- ~ Man in the middle: una persona en mitad de la conexión (intercepta paquetes).
- ~ Distributed Denial of Service (DDoS): denegación de un servicio distribuido, por ejemplo Flooding.
- ~ Malware: troyanos, gusanos, spyware, nackdoors...

Para todos estos ataques se han diseñado una serie de mecanismos de seguridad que veremos a continuación, entre ellos:

- Cifrado (simétrico y asimétrico).
- Autenticación con clave secreta (reto-respuesta).
- Intercambio Diffie-Hellman (establecimiento de clave secreta).
- Funciones Hash. Hash Message Authentication Code (HMAC).
- Firma digital.
- Certificados digitales.

Don't worry que esto es solo nombrarlos, luego se ve como funcionan. Que tema más largo me cago en dios.

Cifrado de datos

Con este procedimiento se garantiza la **confidencialidad**. Nos ponemos en situación de tener un texto y querer que solo lo lea el receptor, porque le vas a decir que te gusta y te mueres de vergüenza si se entera alguien más. Entonces teniendo el texto plano P, lo vamos a cifrar de tal manera que se transforme en un texto cifrado C.

Entonces, para cifrar nuestro texto, por ejemplo, podríamos aplicar un algoritmo en el que la A->B, B->C y así sucesivamente (obviamente esto es muy básico y se enteraría todo el mundo). Lo importante es que va a existir un algoritmo que va a cifrar nuestro mensaje de una manera que no lo entienda nadie. Este algoritmo de cifrado/criptación lo llamaremos **E_k()**, donde k será la clave con la que vamos a cifrar (clave = manera con la que lo vamos a cifrar). Cuando lo queramos descifrar pues aplicamos otro algoritmo de descifrado llamado **D_{k'}()**. k' será la clave con la que vamos a descifrar. La gracia de esto es que la clave de cifrado k y descifrado k' sean desconocidas, de esta manera estaremos seguros.

De aquí surgen dos formas de cifrar, el cifrado simétrico y el asimétrico.

Cifrado simétrico, algoritmos de clave secreta

El cifrado asimétrico se basa en la existencia de una única clave para cifrar y descifrar ($k = k'$). Existen una serie de algoritmos para encriptar nuestro mensaje, entre ellos:

- **DES** (Data Encryption Standard): es un algoritmo de sustitución monoalfabético, y se le aplica el llamado encadenamiento DES para evitar que sea un algoritmo de sustitución. Si te interesa saber cómo se aplica el algoritmo te metes en Internet y buscas lo que hace. Para mejorar su robustez se usa DES doble y 3DES. (Lo importante, que DES es un algoritmo de cifrado simétrico fin).
- **IDEA** (International Data Encryption Algorithm): un algoritmo de cifrado simétrico, que trabaja con claves de 128 bits, (DES es de 56 bits) y además opera en tiempo real.

El problema de esto es que tendría que tener una clave distinta para cifrar con cada persona, y si lo hago con 1 persona pues está bien pero con 500 no tantos. De aquí surge el cifrado asimétrico.

Cifrado asimétrico, algoritmos de clave pública/secretaria

- Cada usuario (A) va a tener dos claves, una pública llamada **KpubA** y una privada, **KpriA** (luego se explica cómo se calculan).

- La clave pública la puede saber todo el mundo, la puede poner en un cartel publicitario con luces de neón y fuego artificiales, pero la clave privada sólo la puedes saber tú, esa la proteges como si tu vida te fuera en ello.
- Vamos a suponer que soy el usuario A y quiero enviarle un mensaje al usuario B. Entonces para cifrar un mensaje (recordamos que cifrar es $E_k()$ y descifrar $D_k()$) hacemos lo siguiente. Como yo sé la clave pública de B porque la ha puesto en un cartel luminoso con neones, tengo que, texto cifrado = $E_{k\text{pub}}(B(\text{mensaje}))$ y cuando B quiera leerlo, como sólo él tiene la clave privada porque la ha defendido con su vida, será el único que lo puede descifrar y hará mensaje = $E_{k\text{priv}}(B(\text{texto cifrado}))$.
- Todo esto está muy bonito, pero si todo el mundo tiene la clave pública de B, entonces pueden interceptar el paquete y no podrán saber lo que pone porque no tienen la clave privada, pero podrán escribir en él. ¿Y cómo hago yo para mandarle un mensaje de A a B para que sepa que soy yo el que se lo envía? Pues yo puedo hacer texto cifrado = $E_{k\text{priv}}(A(\text{mensaje}))$. Como yo soy el único que tengo mi clave privada, voy a ser el único que pueda escribir en ese mensaje. Luego llega B y como sabe mi clave pública porque la he puesto en un cartel luminoso, solo tendrá que hacer mensaje = $D_{k\text{pub}}(A(\text{texto cifrado}))$ para saber que he sido yo quien se lo manda. Esto provee **AUTENTICACIÓN**. El problema de esto es que intercepten el paquete, y no podrán escribir en él pero si lo podrán leer porque tienen mi clave pública. Vamos que hagas lo que hagas estás jodido por un lado o por otro.

Todo esto está muy bonito, pero ¿cómo calculo las claves públicas y privadas? Se aplica un algoritmo llamado **RSA** (Rivest, Shamir y Adleman). Que si quieres ver cómo es, te vas a las diapositivas que lo explican muy bien.

Autenticación y cifrado de clave respuesta

Estos protocolos son una familia de protocolos que permiten la **autenticación** de entidades de la siguiente manera:

- 1.- Una parte (verificador) presenta una cuestión (desafío).
- 2.- La parte que se quiere autenticar recibe el desafío y elabora una respuesta que envía al verificador.
- 3.- El verificador recibe la respuesta y evalúa si la respuesta es correcta y por tanto, la entidad que lo respondió queda autenticada.

Muchas de estos desafíos tienen que ver con la dirección MAC (un número que identifica al ordenador).

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



Intercambio Diffie-Hellman

Con este protocolo podemos establecer una clave secreta entre dos entidades a través de un canal no seguro. De esta manera, dos personas podrán comunicarse de manera segura aunque haya un man in the middle escuchando. Si quieras ver como funciona, mírate este vídeo porque lo explica muy bien y es cortito.

<https://www.youtube.com/watch?v=TWhax2wQOrU>

Funciones Hash (compendios)

- Son funciones unidireccionales (irreversible) de cálculo sencillo, como por ejemplo, multiplicar dos números primos grandes es bastante fácil, pero averiguar qué dos números primos multiplicados es computacionalmente imposible. Tendría que pasar mucho tiempo hasta encontrarlos.
- Le podemos pasar el texto de entrada que queramos a la función Hash que la salida va a ser de longitud fija. (256 o 512 bits). Lo mismo da pasarte el Quijote que la palabra "hola", que su resultado tras aplicarle la función Hash va a ser igual de larga. Aplicar la función Hash se representa como $H(\text{sobre un texto})$.
- Es imposible obtener M a partir de $H(M)$.
- Son invulnerables a ataques de colisión, dado M es imposible encontrar $M' = M$ y $H(M) = H(M')$
- Estas funciones Hash se usan para garantizar **integridad y autenticación**.
- Hash Message Authentication Code (HMAC) se utiliza para calcular un código de autenticación de un mensaje que implica una función Hash. Es $M + H(K \mid M)$ pero para evitar ataques de extensión se usa $M + H(K \mid H(K \mid M))$. Vamos un lio que te cagas.
- Algunos ejemplos de estas funciones HASH son: MD5, SHA-1, SHA-512, que si os interesa ver cómo funcionan aparece en las diapositivas.



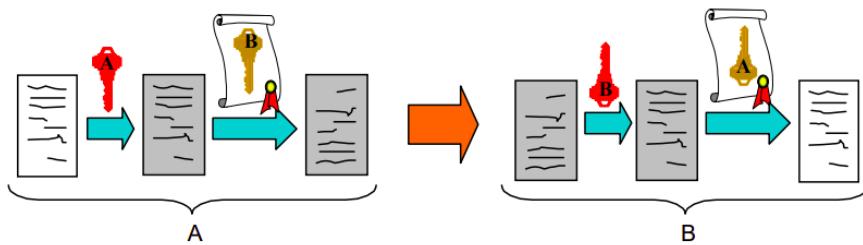
Firma digital

Los objetivos de la firma digital son:

- El receptor puede **autenticar al emisor**.
- **No haya repudio**.
- El emisor tenga garantías de no falsificación (**integridad**).

Vamos que esto es la leche porque hace pila de cosas. Existen 2 tipos, con clave simétrica y asimétrica.

- Firma digital con clave secreta: se llama Big Brother y es una mierda así que ni caso, con



saber que usa una sola clave y que se llama así pues perfecto.

- Firma digital con clave asimétrica. Doble cifrado (se parece a lo de cifrado asimétrico explicado antes):
 - ~ Primero, se hace un cifrado con KpriA, lo que proporciona una autenticación de A.
 - ~ Después, se hace otro para proporcionar privacidad, con KpubB.
 - ~ Para cifrar un mensaje de A hacia B, se envía KpubB(KpriA(texto)).
 - ~ Para que B descifre el mensaje, se hace KpubA(KpriB(KpubB(KpriA(texto)))) = texto

El esquema sería el siguiente:

Hay una debilidad con esto, para garantizar el no repudio, se necesita garantizar la asociación indisoluble de la identidad A con su clave pública KpubA. Porque claro, yo puedo hacer algo con una clave pública y luego decir que esa no era mi clave pública y por tanto no soy yo. Para remediar esto existe lo que se llama certificado digital.

Existen una serie de empresas, llamadas autoridades de certificación en la que le podemos solicitar un certificado digital. Luego, debemos ir a recogerlo con nuestro DNI a una oficina, esto hace que todo lo que se realice con esa claves públicas y privadas, sea hecho bajo nuestro nombre. Así que interesa no perderlas porque si no las pierdes.

Ya hemos visto muchos protocolos seguros, así que ahora vamos a hablar sobre su implementación.

Seguridad

Existen dos tipos de seguridad, que son las siguientes: (importante que luego preguntas de aquí)

- Seguridad perimetral: firewalls + sistemas de detección de intrusiones (IDS) y de respuesta (IRS).
- Seguridad en protocolos: los protocolos que se usan en cada capa para la seguridad. Algunos son:
 - ~ Capa de aplicación: Pretty Good Privacy (**PGP**) para el correo electrónico seguro y Secure Shell (SSH) para un acceso remoto seguro.
 - ~ Capa de transporte: Transport Secure Layer (**TSL**) (antes SSL): entre ellos HTTPS, IMAPS, SSL-POP, VPN. TLS asegura **confidencialidad, autenticación e integridad**.
 - ~ Capa de red: IPSec (VPN).
 - ~ Capas inferiores: PAP, CHAP, MS_CHEAP, EAP.

Vamos a hablar un poco de TSL porque luego cae.

Transport Secure Layer (TSL) (SSL)

Como hemos explicado antes, a TSL pertenecen HTTPS, IMAPS, SSL-POP Y VPN.

- SSL Record Protocol encapsula los protocolos y ofrece un canal seguro con **confidencialidad, autenticación e integridad**.
- SSL Handshake Protocol: negocia el algoritmo de cifrado, negocia la función Hash y autentica al servidor con X.509. El cliente genera unas claves mediante Diffie-Hellman o aleatorias cifradas.
- SSL Alert Protocol: informa sobre errores en la sesión
- Change Cipher Spec Protocol: para notificar cambios de estado.

Por último, de seguridad ya queda muy poco, aguanta lo poco que queda lo voy a resumir mucho y dejar lo principal. Queda hablar sobre IPSec, que es la seguridad en la capa de red.

IPSec

Su objetivo es garantizar **autenticación, integridad y (opcionalmente) privacidad** a nivel de IP. Para ello sigue 3 procedimientos:

- 1.- Establecimiento de una asociación de seguridad: el objetivo es establecer una clave secreta mediante Diffie Hellman. Incluye una autenticación previa con certificados digitales para evitar un Man In The Middle. Es simplex (la asociación de seguridad tiene un único sentido) y se identifica con la IP origen + Security Parameter Index que es de 32 bits. Vulnera el carácter NO orientado a conexión de IP. Vamos que después de hacer un puñao de cosas, asocia la seguridad.
- 2.- Garantizar autenticación e integridad de los datos: para ello usa cabeceras de autenticación.
- 3.- (Opcional) Garantizar autenticación, integridad y privacidad de los datos: usa un protocolo de encapsulado de seguridad de la carga.

IPSec tiene 2 modos de operación, que funcionan así:

- Modo Transporte: la asociación se hace extremo a extremo entre host origen y host destino.
- Modo túnel: la asociación se hace entre dos routers intermedios.

Y fin del tema de seguridad, ahora viene algo más sencillo. Así que descansito ole ahí.

Aplicaciones multimedia

Hay que dar una serie de definiciones, pero son fáciles.

- Aplicaciones multimedia: audio, vídeo, juegos, real-time. Cosas que nos divierten vamos.
- Calidad de servicio (QoS): tener una capacidad para ofrecer un rendimiento requerido para una aplicación.
- Por ejemplo, el protocolo IP es de Mejor esfuerzo (best effort), vamos, que lo va a intentar hacer lo mejor que pueda pero no ofrece una QoS.

Existen muchos tipos de aplicaciones, entre ellas: audio y vídeo almacenado (YouTube), audio y vídeo en directo (la TV) y audio y vídeo interactivo (Skype). Las características fundamentales que cumplen estas aplicaciones son:

- Elevado ancho de banda: necesitan bastantes datos.
- Tolerantes relativamente a pérdida de datos: si no llega un pixel en un momento de la videoconferencia no pasa nada.
- Exigen Delay (retardo) acotado: no pueden esperar mucho tiempo para mostrar los datos.

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



Reservados todos los derechos. Queda permitida la impresión en su totalidad.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.



WUOLAH

Aprende en 1 hora lo que no has aprendido en 4 meses de FR

Tema 3. Capa De Transporte En Internet.

La intención de este pdf es intentar explicar FR con muchos ejemplos y simplificando las cosas porque explicarle algo a alguien que ya lo sabe pues tiene poca gracia. No es un pdf para memorizar, si no para leer y comprender el tema 3.

Introducción

En esta tema vamos a hablar sobre la capa de transporte del protocolo TCP/IP que hemos visto en el tema 1. Esta capa ofrece una comunicación extremo a extremo (end to end) y se encarga de realizar la multiplexación/demultiplexación de aplicaciones mediante puertos (más adelante se explicará lo que es un puerto).

Esta capa está formada por 2 protocolos, TCP y UDP. (No confundir TCP con TCP/IP, TCP es parte de TCP/IP). Ambos realizan multiplexación/demultiplexación de aplicaciones. Multiplexar significa tomar los datos de la capa de aplicación, etiquetarlos con un número de puerto que identifica la aplicación emisora y enviar dicho paquete a la capa de red. Vamos que esta capa se dedica a poner cabeceras y vas a memorizar cabeceras como un desgraciado. Demultiplexar es el proceso contrario.

Una introducción a los protocolos UDP y TCP. (Importante, que luego preguntan).

- UDP: **no orientado a conexión** (un paquete puede ser enviado desde un punto a otro sin una conexión previa) y **no fiable** (no tiene recuperación de errores, si se pierde uno por el camino pues que se joda, se ha perdido y punto).
- TCP: **orientado a conexión** (se debe establecer una conexión antes de transferir datos), y **fiable** ya que incluye control de errores y de flujo, control de la conexión y control de la congestión.

Una vez hecha esta introducción, podemos pasar a hablar de los puertos.

Puertos

Los puertos son **números enteros de 2 Bytes**, es decir, va desde 0 hasta $2^{16} - 1 = 65535$. Estos números identifican al proceso origen y al proceso destino. Por ejemplo, sabemos que tengo una IP pública (que es la del router y por tanto, todos los dispositivos conectados a él tienen la misma) y luego cada dispositivo tiene una IP privada (asignada mediante DHCP y todo eso del tema 2). Pues si yo pido un paquete de algo, ese paquete podría llegar hasta mi ordenador porque tenemos la IP pública y la IP privada, pero ¿qué aplicación del ordenador necesita los datos que han llegado? Si me llega una página web, para qué iba a querer el correo electrónico esa información? Para esto existen los **puertos, para identificar la aplicación que envía y recibe datos.**

Existen una serie de puertos preasignados a aplicaciones/servicios que son comunes. Estos puertos van del 0 al 1024. Y entre ellos se encuentran los siguientes:

Puerto	Aplicación/Servicio	Descripción
53	DNS	Servicio de nombres de dominio
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

De 1024 hasta 65535 están a libre disposición del desarrollador. **¿Para qué iba a querer alguien tener puertos libres si las aplicaciones que todos usamos ya están preasignadas?** Pues, por ejemplo, se da el caso que yo puedo tener 2 pestañas abiertas de un navegador cualquiera. Y ahora, si yo pido un paquete y solo tuviera el puerto preasignado de la aplicación, **¿a qué pestaña de las 2 hay que servirle los datos?**

Si te quieres informar bien sobre los puertos, mira el siguiente vídeo en el que viene bien explicado: <https://www.youtube.com/watch?v=hmGmeGDRUAU> es un poco largo, pero mejor perder 20 minutos ahora que 4 meses de FR otra vez.

Ya que sabemos lo que es un puerto pues podemos empezar a hablar de UDP.

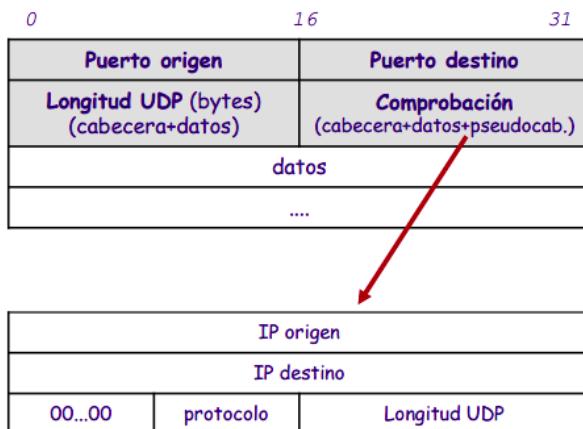
User Datagram Protocol (UDP)

Vamos a decir sus características principales una vez más. (Es importante porque luego se pregunta sobre las características en el examen).

- Ofrece una funcionalidad **best-effort**: intenta hacer sus cosas lo mejor posible pero tampoco asegura una calidad de servicio. Vamos que lo intenta pero no le pidas más.
- Es un servicio **no fiable**: hay pérdidas, si se pierde un paquete por el camino, no hace el esfuerzo de recuperarlo. Si se pierde que se joda, no es tan importante como para que UDP se dé media vuelta.
- **No hay garantías de entrega ordenada**: los paquetes no llegan en un orden determinado.
- **No hay control de congestión ni control de flujo**: se intenta hacer una entrega tan rápida como se pueda, los controles ralentizan mucho.
- Realiza la **multiplexación/demultiplexación**: transporta las TPDU (unidad de datos de protocolo, paquetitos que se envían).

Bueno pero si hemos visto que todas las características que tienen son una puta mierda, quién va a querer usar este protocolo. Pues lo bueno que tiene este protocolo es la velocidad que tiene, y hay ciertas aplicaciones que necesitan velocidad. Son aplicaciones multimedia (un video, una canción), son tolerantes a fallos (si 1 segundo de una canción no se escucha perfecto pues tampoco pasa nada) y muy sensibles a los retardos (eso sí, la canción la quiero escuchar medio decente en un periodo corto, no quiero esperar 3 días para escuchar una canción a la perfección de Bad Bunny).

Cada segmento de UDP se encapsula en un datagrama IP (lo veremos en el tema 4), el datagrama es el siguiente:



Ya hemos visto UDP, ahora a ver TCP que es más largo pero es lo que toca.

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



User Datagram Protocol (TCP)

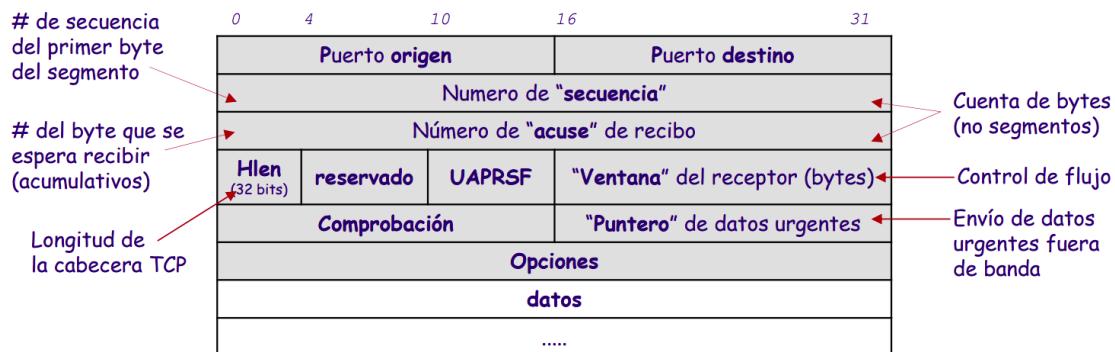
Vamos a listar todas sus características, aunque muchas ya las hemos dicho.

- Ofrece un servicio **punto a punto**: no sirve para comunicaciones multicast (de uno a muchos).
- Es un servicio **orientado a conexión**: exige un estado común entre el emisor y el receptor, llamado **handshaking**. Tiene 3 fases la conexión, establecimiento, intercambio de datos y cierre.
- **Garantiza** la **entrega ordenada** de las secuencias **de bytes generadas** por una aplicación.
- Opera en **transmisión full-duplex** (ISE vibes): esto significa que **puede tanto enviar como recibir mensajes de forma simultánea**.
- Incluye **mecanismos de detección y recuperación de errores** (ARQ, esto no es como UDP que se la sudaba, aquí hay que tener un control para que llegue todo) con **confirmaciones positivas ACKs** (acumulativas, siempre son positivas y se dedican a dar el visto bueno de que ha llegado un paquete) y **"timeouts"** adaptables. (Time out, **tiempo** que tienes para enviar un paquete, si en ese tiempo no le confirman que ha llegado pues tendrá que reenviarlo).
- Usa la técnica de **incorporación de confirmaciones** ("piggybacking"). En vez de enviar **una confirmación (ACK)** sola, pues se envía con el siguiente paquete.
- Ofrece un **servicio fiable**: control de congestión y control de flujo con ventanas deslizantes para que las cosas funcionen bien, ya se explicará más adelante.
- Para mejorar su eficacia, **TCP se adapta a las condiciones de red dinámicamente**.

Te dejo esto para animarte porque lo que viene es largo.



Las TPDUs (unidad de datos de protocolo, datos que mandamos) de TCP se llaman segmentos TCP y tienen la siguiente forma (en este enlace te enteras perfectamente http://cvuoc.edu/UOC/a/moduls/90/90_329/web/main/m2/v4_2_2.html):



Cada segmento TCP (datos que mandamos) se encapsula en un paquete IP.

TCP tiene una serie de funcionalidades que tiene para cumplir sus características. Así que vamos a explicar cada una de ellas. Las funciones son:

- Multiplexación/demultiplexación de aplicaciones.
- Control de la conexión (establecimiento y cierre).
- Control de errores y de flujo.
- Control de congestión.

Multiplexación/demultiplexación TCP

El objetivo es transportar las TPDUs (segmentos TCP, datos a enviar) al proceso correcto. Para esto se usan los puertos, que hemos explicado arriba. (Numeritos que identifican el

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 dominio
80	HTTP	Acceso hipertexto (web)
110	POP3	Descarga de correo

proceso a donde van los datos). Hay una serie de puertos preasignados, antes mostramos los de UDP y ahora vamos a ver los de TCP.

Cada conexión TCP se identifica por IP origen, puerto origen, IP destino, puerto destino.

Control de la conexión TCP

Como bien sabemos, TCP es un servicio orientado a conexión por lo que tanto el emisor como el receptor tienen que estar conectados para intercambiar información. Es intercambio de información tiene tres fases:

- Establecer la conexión.
- Intercambiar datos (full duplex, que envía y recibe de manera simultánea).
- Cerrar la conexión (liberar recursos).

NO es posible garantizar un establecimiento/cierre fiable de la conexión sobre un servicio (IP) no fiable.

Vamos a hablar del **establecimiento de la conexión**: (three-way handshake). Esta conexión consta de tres pasos.

1.- El cliente se quiere conectar al servidor para enviar los datos. Para esto, le manda un segmento TCP (datos que se envían con una serie de cabeceras) con SYN (Synchronize Sequence Number, bit que va en la cabecera indicando que quiere establecer una conexión, luego se explicará), esto informa al servidor que hay alguien que quiere iniciar la comunicación.

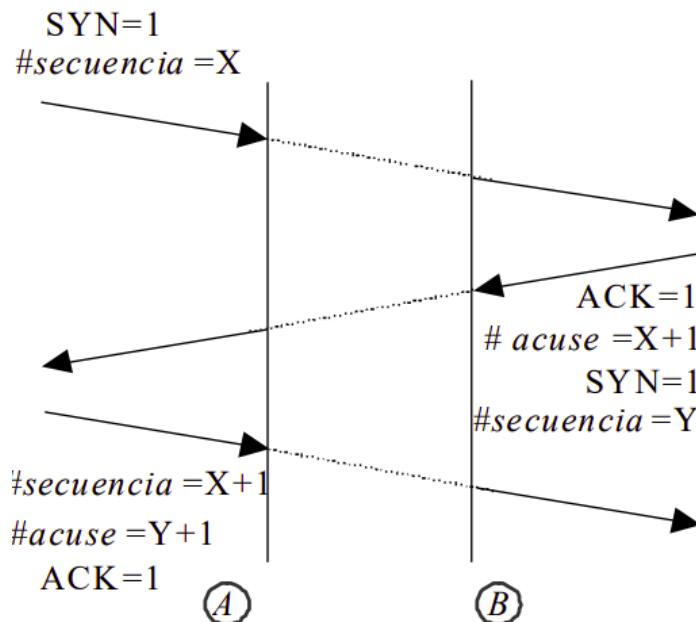
En resumen, el cliente es un tío en una fiesta intentando roncarle a una muchacha (servidor) y para iniciar la comunicación le tira un hielo a la cabeza (SYN). Entonces la muchacha ya sabe que hay alguien que quiere iniciar una comunicación con ella.

2.- El servidor responde al cliente con las cabeceras del segmento SYN (booleano de la cabecera que indica que se quiere sincronizar) y ACK (acknowledge, un booleano que le confirma al cliente que le ha llegado su solicitud al servidor).

Esto es como si a la muchacha a la que le han tirado el hielo, mira al tío que se lo ha tirado (ACK, confirma que le ha llegado el pepinazo del hielo) y coge y ella le tira otro para que el tío espabile y se dé cuenta de que ya pueden iniciar una conversación (SYN).

3.- Por último, el cliente recibe el ACK (la confirmación) del servidor, así que el cliente le manda otro ACK para establecer una conexión y empezar la transferencia de datos.

El tío le tira otro hielo a la muchacha ya para avisarle que va para donde está ella a comunicarse. Con los pasos 1 y 2, la conexión se establece en un sentido (cliente a servidor), con los pasos 2 y 3, la conexión se establece en el otro sentido (servidor a cliente). Por tanto, es full-duplex. Vamos a ver el diagrama de lo que hemos explicado, en el que A es el cliente y B es el servidor. Van a aparecer una serie de valores, unos los hemos visto antes (ACK y SYN que son bits == booleano que indican cierta acción), pero va a haber otros que aún no hemos visto (secuencia y acuse) que van a ser explicados.



A manda un paquete con SYN activo (indicando que quiere sincronizarse) y un número de secuencia (luego se explica). Entonces B manda un paquete con ACK activo (quiere decir que ha recibido su solicitud), un número acuse (que contiene el valor del siguiente número de secuencia que A espera). Y como se quiera sincronizar B con A, envía también un SYN y su número de secuencia. Por último, A le comunica con el ACK que ha llegado su solicitud y le envía su número de secuencia y acuse. Mucho lío verdad, vamos a intentar aclarar una cosa.

¿Para qué mierda vale el número de secuencia, cómo se calcula y para qué queremos el número acuse?

- El **número de secuencia** es un campo de 32 bits (cabecera de un segmento TCP = datos que se envían) que cuenta con **bytes en módulo 2^{32}** (el contador vuelve a 0 cuando se llega al valor máximo). Muy bien, ya sabemos que el nº de secuencia es un número de 32 bits que va en la cabecera de un paquete. Ahora, ¿cómo se calcula?

Estudiar sin publi es posible.

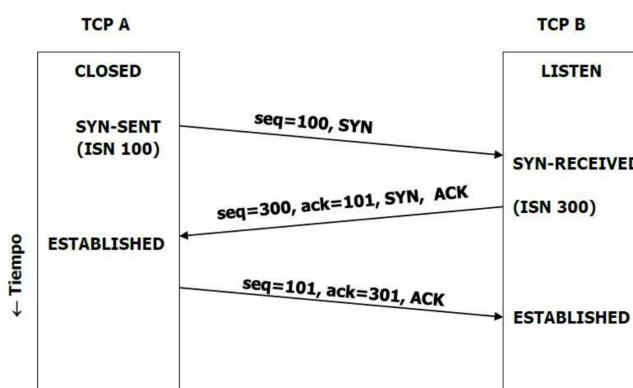
Compra Wuolah Coins y que nada te distraiga durante el estudio.



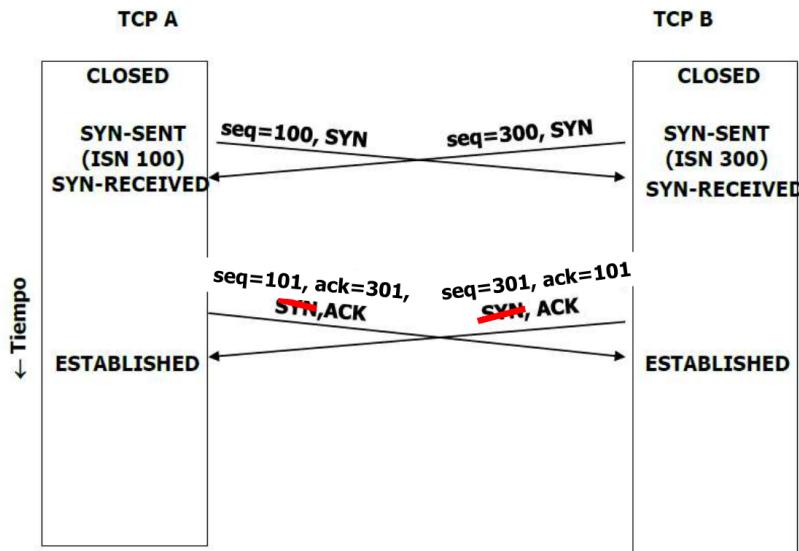
- El número de secuencia no suele empezar en 0, es un valor denominado **ISN** (Initial Sequence Number) elegido teóricamente al azar, para evitar confusiones con solicitudes que se hayan hecho antes.
- El ISN (el valor inicial del número de secuencia) es elegido por el sistema (ya sea cliente o servidor). El estándar sugiere utilizar un contador que se incremente en 1 cada 4 µs. Entonces, el ISN volvería a 0 después de 4 horas 46 minutos.
- Este mecanismo de selección de ISN es muy útil para que no haya coincidencias, pero no protege frente a sabotajes y ataques. Es muy fácil averiguar el ISN.
- TCP incrementa el número de secuencia de cada segmento según los bytes que tuviera el segmento anterior, con una excepción: los flags SYN y FIN incrementan en 1 el número de secuencia.
- Los segmentos ACK no aumentan el número de secuencia.
- Todo muy bonito, ¿pero para qué vale este número? En TCP los segmentos (paquetes que se envían con datos) no se numeran. Es decir, no es algo modo, he enviado el paquete 1, el paquete 2... En TCP se enumeran los bytes enviados. Por tanto, el número de secuencia identifica el primer byte de los datos que se envía en el segmento. EJEMPLO: tenemos un número de secuencia inicial (ISN) que vale 100, entonces cuando enviamos un único byte, como por ejemplo al hacer SYN, el número de secuencia pasa a valer 101. Si luego envío un paquete de 500 bytes, el número de secuencia será el 601. De esta manera se hace un seguimiento de lo que hemos enviado. ESTARÍA BIEN QUE HUBIERAN EXPLICADO ESTO EN CLASE PORQUE AHORA UNO YA SABE POR QUÉ CUANDO HACEMOS UN SYN SE LE SUMA 1 AL NÚMERO DE SECUENCIA.
- Por último, el número acuse es una ayuda, contiene el valor del siguiente número de secuencia que el emisor del segmento espera. Por esto, enviar ACKs no aumentan el número de secuencia, ya que el número de acuse siempre es parte de la cabecera.

Ahora, te recomiendo que mires el diagrama que había puesto arriba y entiendas por qué es así. Porque ya sabemos lo que es el número de secuencia y el número acuse. Voy a poner un par de ejemplos más, en estos ejemplos, número de secuencia es seq y acuse es ack en minúscula.

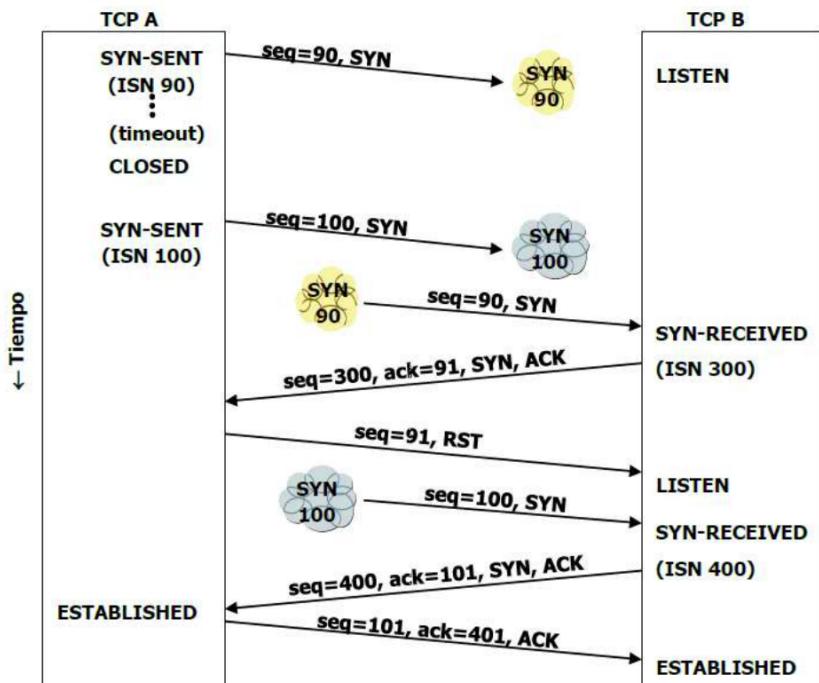
Conexión sin problemas



Conexión simultánea



Conexión con retardos y time outs (don't worry que ya se explicará)



Vale, estamos bien, ya sabemos cómo se abre una conexión y los números de secuencia. Te dejo esto para relajar un poco el asunto.



Bueno, ya sabemos cómo abrir una conexión, así que ahora toca **cierre de la conexión:** Esto se hace para liberar los recursos ocupados durante la conexión, pero si no lo hacemos de manera ordenada es posible que haya pérdidas de información.

El proceso sería el siguiente:

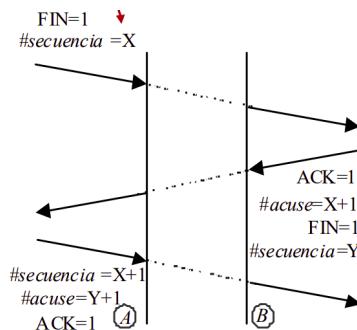
1.- El cliente ya ha mandado todos los datos que tenía que mandar, tiene su número de secuencia X y quiere cerrar la conexión. Para ello, en vez de activar el bit de SYN (el que activábamos para comenzar una conexión) pues vamos a activar el bit de FIN (que le indica al otro que en vez de sincronizarnos pues queremos terminar de hablar). Entonces, le mandamos un segmento con el bit de FIN activo y nuestro número de secuencia.

2.- El servidor recibe el paquete y como siempre pues va a mandar un ACK indicando que le ha llegado su solicitud, le manda el número de secuencia del servidor, el acuse (como siempre) y como también se quiere desconectar el servidor del cliente (full duplex) pues le manda el bit de FIN.

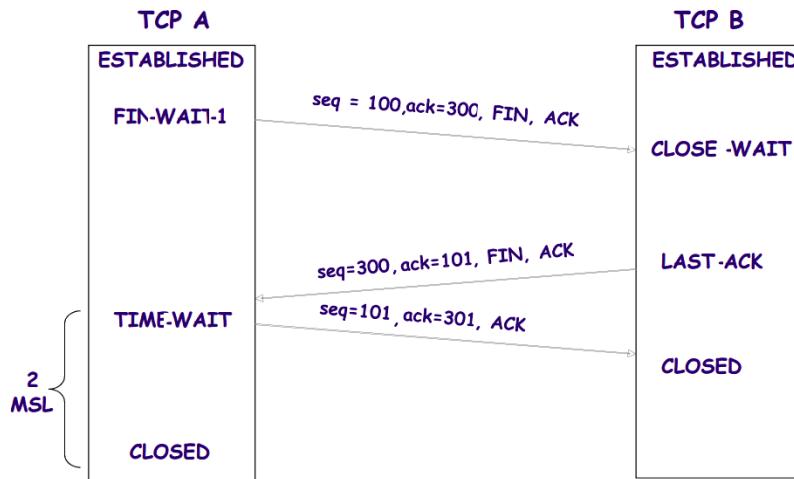
3.- Como el segmento le llega al cliente pues le va a enviar un ACK de que lo ha recibido, junto con la secuencia y el acuse al servidor. Y ya la conexión estaría cerrada (más o menos).

Para que no se pierdan datos por bloqueos o pérdidas, el cliente se espera un tiempo llamado timeout. Esto se hace porque puede haber paquetes que se hayan perdido por el camino y se estén recuperando, esta recuperación va a durar un tiempo llamado Maximum Segment Lifetime = 2 min. En 2 minutos ya no van a aparecer paquetes anteriores perdidos, y por tanto, ya se puede cerrar la conexión sin perder nada.

Vamos a ver unos ejemplitos del cierre de conexión, es casi igual que el comienzo de la conexión.



Aquí se puede ver que se cierra el cliente una vez han pasado los 2 minutos.



MSL: Maximum Segment Lifetime (normalmente 2 minutos)

Ya hemos terminado con el control de la conexión, sabemos abrir y cerrar una conexión pero aún falta lo importante, ¿cómo se mandan los datos?

Control de errores y de flujo

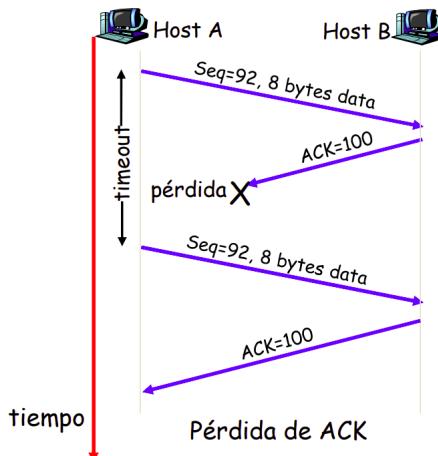
Todo lo que vamos a hablar ahora es sobre **CONTROL DE ERRORES**. Vamos a hacer una pequeña introducción:

- El control de errores es un **esquema ARQ** (Automatic Repeat Request, un protocolo que se usa) que garantiza la integridad de los datos.
- En el control de errores **hay una serie de campos** (cabeceras que vienen junto a los datos en el segmento que se envía) **involucrados**. Muchos ya lo hemos visto:
 - ~ **Campo de secuencia**: contiene el número de secuencia (número de bytes enviados).
 - ~ **Campo acuse**: contiene el número de acuse (número de byte que espera el receptor).
 - ~ **Bit A(ACK)**: para confirmar que te ha llegado algo.
 - ~ **Campo de comprobación**: **checksum del segmento**, es un campo que se utiliza para detectar errores. Es de 16 bits.

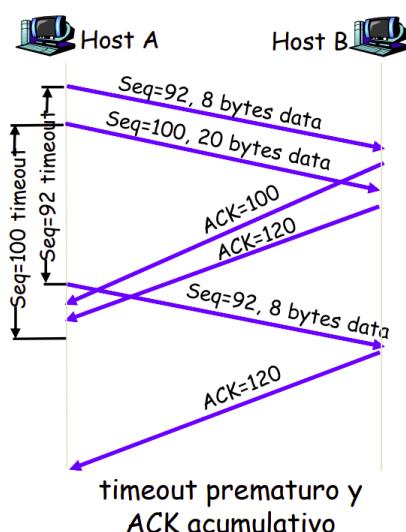
Vamos a ver un par de errores que pueden ocurrir durante TCP.

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



A le envía un segmento a B, entonces B pues le tendría que enviar un ACK para confirmarle que le ha llegado lo que le ha enviado. Pues B le manda este ACK, pero este ACK se pierde por el camino porque las cosas se pierden. Entonces pasa un tiempo (timeout) y A dice, el gilipollas este no me dice si le ha llegado, supondré que se ha perdido mi paquete y se lo vuelve a enviar. Entonces esta vez, el ACK de B no se pierde y confirma que le ha llegado ese segmento. Así puede continuar la comunicación.



A le manda primero un segmento, y luego dice, bueno pues si le mando uno más, me confirma 2 veces que le ha llegado y todos felices, así tarda menos. Entonces A manda sus dos segmentos, y B pues manda 2 ACKs, pero por un problema, la confirmación de que le ha llegado el primer segmento de A tarda mucho en enviarse. Entonces, pasa le timeout de A y se mosquea y dice, no ves el retraso que le envío un segmento y no me confirma si le ha llegado, y se lo vuelve a enviar. Pero resulta que, no es que se hubiera perdido la confirmación de antes, es que llegaba más tarde. No obstante, sí que le ha confirmado que el segundo paquete le ha llegado. Igualmente, al servidor le vuelve a llegar el primer segmento, y dice, que sí que me han llegado los 2, y le envía el ACK del segundo diciéndole así que puede continuar con la comunicación.

Bueno, hemos visto un par de cosas que pueden pasar, pero pueden pasar más y podemos verlo en esta tabla. (Es importante entender esto porque luego los ejercicios sobre TCP van sobre esto).

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.



Con esto surge un problema, ¿el timeout cuánto debe durar? Cómo de impaciente tienen que ser? Pasan 2 cosas. Es evidente, que hay que esperar como mínimo el tiempo que tarda el mensaje en llegar (de forma física) y la confirmación en volver. (RTT, Round Trip Time) Ejemplo: si envío una carta y sé que tarda como mínimo 2 días en llegar y 2 días en llegar una respuesta, pues como mínimo tendría que esperar 4 días, pero si cuánto tiempo tiene que pasar para darme cuenta de que esa carta se ha perdido o es que tarda mucho.

- Si el tiempo de espera es muy pequeño, voy a pensar que si en 4 días no me ha llegado una respuesta me la han perdido, y voy a tener que escribir la misma carta otra vez y enviarla. Pero capaz, envías de nuevo la carta y al día siguiente te llega la respuesta de la anterior. Has escrito la misma carta para nada (Retransmisiones innecesarias).
- Si el tiempo es demasiado grande: capaz pasan 3 meses y tú sigues esperando una respuesta, pues así la comunicación no es que sea muy eficaz.

Pues tenemos un problema, porque ni podemos esperar mucho ni poco. Entonces la mejor solución posible es adaptarse dinámicamente.

Hubo una primera solución con un algoritmo de Kurose & Ross, que es calcular el timeout de esta forma:

Kurose & Ross

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

$$RTT_{nuevo} = (1-\alpha) \times RTT_{viejo} + \alpha \times RTT_{medido}, \quad \alpha \in [0,1]$$

$$Desviacion_{nueva} = (1-\beta) \times Desviacion_{vieja} + \beta \times | RTT_{medido} - RTT_{nuevo} |$$

$$Timeout = RTT_{nuevo} + 4 * Desviacion$$

El problema de esto es cuando te llegan ACKs repetidos porque se ha perdido un paquete o algo así. Entonces el RTT medido se va a tomar por culo y el timeout con él. Hay un ambigüedad en la interpretación. Para solucionar este problema, aparece el **algoritmo de Karn**, que únicamente actualiza el RTT para cuando no hay ambigüedad, pero si un segmento tiene que volverse a enviar, se duplica el timeout. $timeoutNuevo = A * timeoutViejo$, donde $A = 2$.

Una explicación básica, el timeout va a ser el RTT de un segmento que se envía y recibe su ACK a la primera. Si por alguna razón, el segmento tiene que ser reenviado, entonces el timeout = RTT de antes * 2. Si queréis informaros más, que luego preguntan sobre esto,

mirad esta página: <https://w3.ual.es/~vruiz/Docencia/Apuntes/Networking/Protocols/Level-4/05-TCP/index.html>

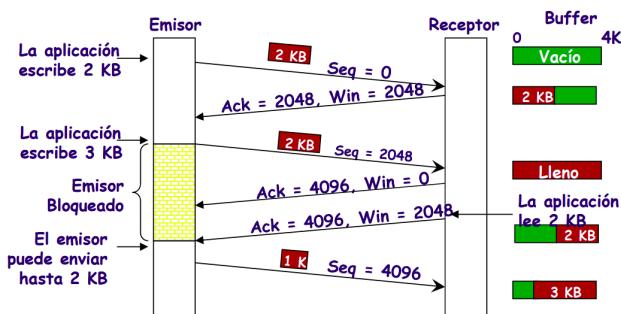
Y ya hemos acabado con el control de errores, a partir de ahora se va a hablar sobre **CONTROL DE FLUJO**. Vamos ahí ostia que ya queda poco.

- El control de flujo es un procedimiento que se usa para evitar que el emisor sature al receptor con el envío de demasiada información y/o demasiado rápido. Vamos que sirve para decirle al emisor, no seas tan pesao hermano, un poquito más lento.
- Es un esquema crediticio, esto significa que el receptor informa al emisor sobre los bytes autorizados a emitir sin enviar respuesta. El receptor le dice al emisor, mira a tí que te gusta ser un pesado, en vez de mandarme 1 mensaje, que yo te confirme y así, que se hace pesado. Te dejo que me envíes 8 mensajes del tirón y hasta que yo no te confirme que me han llegado te callas la boca.
- Esta cantidad de mensajes que se pueden enviar se llama ventana:

Ventana útil emisor = ventana ofertada receptor - bytes en tránsito. Para entendernos:

Mensajes que puedo enviar = mensajes que me deja enviarle en total - mensajes que he enviado.

Vamos a ver un ejemplo:



Ack es acuse, no el flag ACK, y Win indica la ventana que tiene libre (cuántos mensajes me puedes mandar).

El receptor tiene una ventana de 4k, es decir, que 5k del tirón no se los podemos mandar. Vamos con la secuencia:

- 1.- El emisor le envía 2k y como la ventana es de 4k pues se los envía enteros.
- 2.- El receptor le envía una confirmación de que ha recibido el paquete flag ACK (aunque no aparezca ahí), le dice en ack (acuse, el que puso ack es subnormal) que tiene que enviarle los bytes a partir del 2048 porque los anteriores ya le han llegado. Y en win le indica cuánto espacio le queda para que le mande mensajes, en este caso aún tiene 2k libres.
- 3.- La aplicación le quiere enviar 3KB, pero el receptor aún no ha procesado los datos que le ha enviado antes, así que no tiene los 4k, sólo tiene 2k. Pues el emisor le envía lo máximo posible de 3k que es 2k.

4.- Al receptor le llega este paquete y le devuelve un paquete diciendo, perfecto, me ha llegado (flag ACK que no sale en el diagrama), cuando tengas que seguir, tienes que seguir por el byte 4096 (ack que es igual al acuse) pero ahora no tengo espacio libre (win = 0). Así que hasta que no te diga que tengo espacio libre te estás quieto.

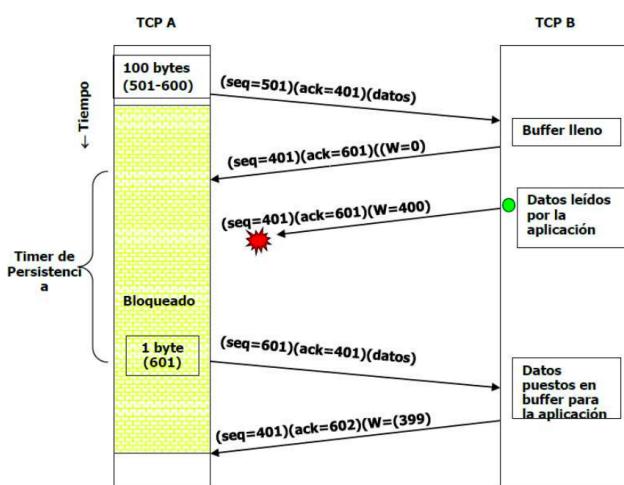
5.- El receptor procesa los 2KB primeros que le envío, y le vuelve a mandar un mensaje al emisor, exactamente igual que el de antes, excepto que ahora le dice que tiene 2KB libres para enviarme.

6.- Como le faltaba 1KB por mandar de antes, pues lo manda y aún le sobra 1KB para mandar.

Y así sucesivamente. Espero que quede claro.

Todo esto está muy bonito, pero qué pasa si el mensaje 5 se pierde y no le dice al emisor que ya tiene hueco para que le envíe más mensajes. Para esto existe una cosa que se llama **temporizador de persistencia**.

Este temporizador lo que hace es lo siguiente: después de un tiempo sin que la ventana se abra y le diga al emisor que le puede enviar más datos, el emisor le pregunta periódicamente si la ventana se ha actualizado. Vamos, como un niño pesado preguntando cuánto queda para llegar en un viaje cada 5 minutos. El emisor hace esta pregunta a través de una cosa que se llaman windows probes. Son segmentos (los datos que se envían) de 1 byte que se dedican a preguntar si está ya libre la ventana. Aquí podemos ver un ejemplo:



Existe un problema llamado **síndrome de la ventana tonta**. Cuando el receptor tiene por ejemplo 1KB libres, el emisor decide enviar 2 paquetes de 512B. Llega el primero al receptor y le dice al emisor que tiene 512B libres. Entonces A hace lo mismo, le quiere enviar 2 paquetes de 256B, pero cuando llega el primero, el receptor le dice que tiene 256B libres. Y

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



así se sigue repitiendo el proceso, entonces el emisor acaba mandando mensajes que son una puta mierda de cortos de 1B, para enviar eso no envíes nada. Pero claro si hay 1B libre en la ventana pues dice, se lo mando.

Este problema se soluciona con la **ventana optimista**: le va indicando al emisor un mínimo de tamaño para que le envíe mensajes.

Con TCP es posible enviar segmentos desordenados, pero deberán indicarlo en su cabecera, activando el bit U (de URG, casi como la UGR, vota Pilar Aranda). Y también se puede solicitar una entrega inmediata a la aplicación activando el bit P (PSH).

Pues ya están explicados el control de errores de flujo. Vamos a lo último de TCP, el control de congestión.

Control de congestión

- El control de congestión es un problema debido a la **falta de recursos** (la capacidad y velocidad de transmisión de las líneas y el buffer en los routers y hosts no son infinitos. Las cosas tienen un límite).
- Es un problema **distinto al control de flujo**: el control de congestión se usa para proteger a la red debido a sus limitaciones, mientras que el control de flujo se usa para enviar los mensajes de la manera más eficiente posible.
- Puede tener naturaleza **adelante-atrás**, aunque en el protocolo IP no.
- Los episodios de congestión provocan retrasos en las ACKs y **pérdida de segmentos**, dependiendo del nivel de seriedad que tenga este episodio.
- La solución propuesta extremo a extremo es **limitar el tráfico generado en el emisor** para evitar pérdidas, pero siendo **eficaz**.
- Esta limitación se consigue limitando el tamaño de la ventana (ventana = cantidad de mensajes que puedo enviar) de emisión.
- Es muy importante el **BandWith-Delay Product**, que básicamente nos dice cuántos bits se pueden enviar sabiendo los bits/segundo de velocidad de emisión y los segundos que tardan en ir y volver (RTT).

Vamos a ver cómo es esto de las ventanas y el mínimo de datos que tiene que enviar el emisor. En el emisor se utilizan 2 ventanas y un umbral.



- Bytes permitidos para enviar = $\min\{\text{Ventana de Congestión}, \text{Ventana del Receptor}\}$.
- La ventana del receptor es la del control de flujo, que hemos visto antes, el espacio que tenía libre el buffer para leer.
- La ventana de congestión inicialmente vale $\text{ventanaCongestion} = 1 * \text{MMS}$;

Hacemos un inicio lento para que no se congestione. El inicio lento funciona así:

Si $\text{ventanaCongestion} < \text{umbral}$, por cada ACK que recibamos

$\text{VentanaCongestion} += \text{MMS}$ (crece de una manera exponencial).

Y para prevenir la congestión, se hace lo siguiente:

Si $\text{ventanaCongestion} > \text{umbral}$, cada vez que recibe todos los ACKs pendientes.

$\text{ventanaCongestion} += \text{MMS}$ (crecimiento lineal)

Y cuando hay un timeout se hace:

$\text{umbral} = \text{ventanaCongestion} / 2$

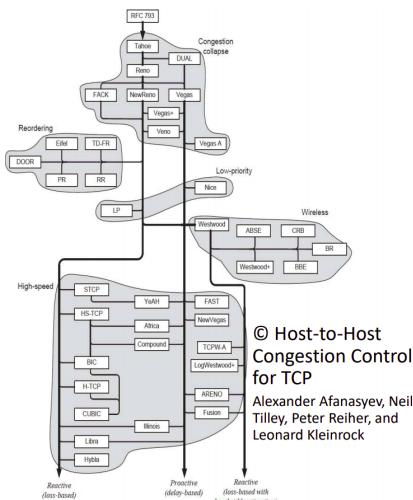
$\text{ventanaCongestion} = \text{MMS}$

Ya vamos a acabar, lo que queda es muy simple así que nada, somos los mejores y punto.

Versiones TCP

- TCP tiene muchas variantes denominadas "sabores".
- Que sean de distintos sabores no afecta a la interoperabilidad entre los extremos.
- Cualquier versión de Linux con kernel superior a la 2.16.19 usa TPC CuBIC.

Aquí dejo el esquema de los sabores TCP:



Hay una adaptación de TCP a las redes actuales, algunos de los datos que saben son: (pero vamos que esto lo pongo por si preguntan pero no importa mucho, si os interesa, en Internet explica qué hace cada cosa).

- Ventana escalada: TCP en segmentos SYN, hasta $2^{14} * 2^{16}$ bytes = 2^{30} bytes = 1 GB autorizados.
- Estimación RTT (tiempo que se tarda en enviar un mensaje y recibir su confirmación): opción TCP sellado de tiempo, en todos los segmentos.
- PAWS ("Protect Against Wrapped Sequence Numbers"): uso de sello de tiempo y rechazo de segmentos duplicados.
- SACKS ("ACKs selectivos").

Y ya se ha acabado el tema, sólo falta el 4.

Aprende en 1 hora lo que no has aprendido en 4 meses de FR

Tema 4. Redes Conmutadas E Internet.

La intención de este pdf es intentar explicar FR con muchos ejemplos y simplificando las cosas porque explicarle algo a alguien que ya lo sabe pues tiene poca gracia. No es un pdf para memorizar, si no para leer y comprender el tema 4.

Introducción

En este tema vamos a estudiar la capa de red. El tema 2 fue sobre la capa de aplicación, el tema 3 sobre la capa de transporte y ahora toca la de red. El objetivo de la capa de red en Internet es interconectar redes, independientemente de la tecnología subyacente. Es decir, la capa de red conecta las redes, independientemente si estás conectados desde un ordenador, un móvil o una patata.

Voy a dar un par de definiciones que son necesarias.

- Comutación: acción de cursar tráfico entre nodos de la red. Comutar es la acción de establecer un camino de extremo a extremo entre dos puntos, un emisor y un receptor, a través de nodos y equipos de transmisión. Esto permite entregar una señal desde un origen hasta el destino. (Luego se habla más de esto así que don't worry).
- Encaminamiento: encontrar la mejor ruta hasta el destino. De todos los caminos que realizan una comutación (conectar dos puntos distintos), encontrar el camino más corto.

El control de congestión que vimos en el tema anterior, que se hacía en la capa de transporte en el protocolo TCP/IP, en el modelo OSI se hace en esta capa.

Hay distintos protocolos de red, entre ellos X.25 e IP.

Una vez hecha esta introducción, vamos a hablar sobre la comutación.

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.



Conmutación

La conmutación es la acción de cursar tráfico para establecer o determinar un camino que permita transmitir información extremo a extremo. Es transmitir información de un punto a otro pasando por nodos intermedios. Existen diferentes tecnologías de conmutación, entre ellas:

- Conmutación de circuitos:

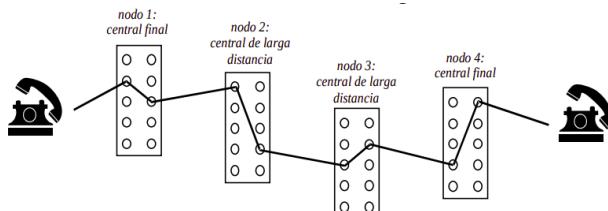
Un ejemplo es el teléfono, la línea telefónica. Es un servicio orientado a conexión (como TCP) y por tanto, tiene 3 pasos: conexión, transmisión y desconexión. Tiene una serie de ventajas y de desventajas. Vamos a ver primero las ventajas:

- ~ La transmisión se realiza en tiempo real: esto está muy bien para las llamadas porque parece que estás teniendo una conversación real.
- ~ Usa permanente los recursos, por tanto, el circuito se mantiene en uso durante toda la sesión.
- ~ No hay contención, es decir, no tienes que esperar para acceder al medio pues eres el único que lo usa en ese momento.
- ~ El circuito es fijo, no tienes que tomar decisiones de encaminamiento una vez está establecido el camino.
- ~ Es más simple en la gestión de los nodos intermedios.

Pero también tiene desventajas:

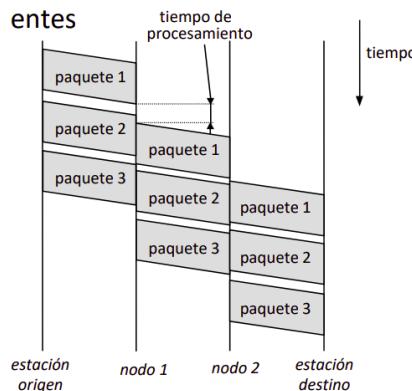
- ~ Existe un retraso en el inicio de la comunicación, cuando se pone a comunicar el teléfono hasta que empieza la comunicación hay un tiempo.
- ~ En ocasiones usa de manera ineficiente los recursos.
- ~ El circuito es fijo, ya sé que es una ventaja pero también es una desventaja el no poder cambiar la ruta de comunicación.

Esto es un sistema de conmutación de circuitos:



Es un poco mierda esta parte porque no hablamos de Internet, pero es que esto también es un sistema de comunicación. Ahora nos vamos a meter un poco más con lo nuestro. Otra tecnología de comunicación es la conmutación de paquetes.

- Comutación de paquetes (datagramas): vamos a ver una serie de características del mismo.
 - ~ Las unidades de datos (paquetes) se envían de manera independiente.
 - ~ No hay conexión.
 - ~ En cada salto se produce un almacenamiento del paquete y un re-envío.
 - ~ Cada paquete debe tener una cabecera las direcciones de origen y destino. Esto es como enviar una carta, tienes que poner la dirección de origen por si no se ha podido entregar y la dirección de destino para saber dónde tienen que llevarla.
 - ~ Los paquetes pueden seguir rutas diferentes para llegar a su destino y pueden llegar desordenados. Esto se debe a, que si hay un paquete que supuestamente tiene que llegar después pero coge un camino más corto que su anterior, llegará antes. Aquí podemos ver un ejemplo de conmutación de paquetes.



La última conmutación que vamos a ver es la de paquetes con circuitos virtuales.

Comutación de paquetes con circuitos virtuales:

- Es usada en redes ATM (Modo de transferencia asíncrona), pero esta tecnología está en desuso para las redes troncales, así que todo lo que digamos pa na sirve
- Es orientado a conexión: Tiene los 3 pasos típicos: conexión, transmisión y desconexión.
- Vemos que se parece mucho a la conmutación de circuitos pero se diferencian en una cosa, no hay una asignación de recursos.

Y ahora vamos a hablar del protocolo IP, que esto ya es más interesante.

El protocolo IP

Voy a dar una serie de características sobre IPv4, aunque en el tema 2 hubo una explicación de por qué los padres de la gente que diseñaron este protocolo eran primos hermanos. Es importante recordar que el protocolo IP no se lo mismo que la dirección IP, de momento cada vez que digamos IP va a ser referido al protocolo. Estas características son importantes porque las preguntan en el examen.

- Es un protocolo para la **interconexión de redes** (también llamadas subredes): Esto se usa para conectar redes entre ellas.
- Resuelve el **encaminamiento** en Internet: es el que se encarga de buscar una ruta más corta entre todos los routers para llegar al servidor que quieras.
- Es un protocolo **salto a salto**. Involucra a hosts y routers: es un protocolo que se aplica cada vez que hacemos un salto de un nodo a otro.
- Ofrece un servicio **no orientado a conexión** y **no fiable**: vamos que es una mierda pero cuanto peor sea más rápido va. No hay un handshake para iniciar una conexión y tampoco se hacen controles de error, de flujo ni de congestión.
- La unidad de datos (paquete) del protocolo IP se llama **datagrama** = una cabecera + unos datos.
- IP es un protocolo de **máximo esfuerzo** (best-effort) o de buena voluntad: lo intenta hacer lo mejor posible pero no asegura una calidad de servicio. Los datagramas se pueden perder, duplicar, retrasar o llegar desordenados, vamos un desastre.
- IP gestiona la **fragmentación**: adapta el tamaño de los paquetes que transporta a las diferentes MTUs (Maximum Transfer Units = lo máximo que puedes transmitir) de las subredes necesarias hasta llegar al destino.

Ya vimos con el DNS, que todos tenemos una IP pública y una privada. La IP pública es la que nos identifica de puertas para afuera. Para los usuarios comunes de una ISP (Internet Server Provider, gente que nos da acceso a Internet como Movistar, Orange,...), les asigna una dirección IP pública dinámica. Esta IP va cambiando cada cierto tiempo (ya está explicado en el tema 2) porque no le importamos a nadie y nadie va a querer acceder a nosotros. Sin embargo, las empresas tienen una IP pública estática, por ejemplo, www.google.com tiene una dirección 172.194.34.209, y quiero que sea esa siempre porque si fuera cambiando pues los DNS irían cambiando (recomendación de corazón para entender esto bien, leeros el tema 1 y el tema 2 que he subido).

Con esto llegamos a la conclusión, que todo el mundo tiene una IP pública y se identifica con ella. La de las empresas no cambian para identificarlas más fácil y la de las personas van cambiando (se explica por qué cambian en el tema 2).

Vamos a seguir viendo más características porque esto tiene un rato para hablar.

- Internet adopta un **direcccionamiento jerárquico**, esto simplifica las tablas de routing. Y te preguntarás, ¿que mierda es un direccionamiento jerárquico? Bien pues es el siguiente punto.
 - Las direcciones IP (32 bits) tienen dos partes bien diferenciadas: un **identificador de la subred** y un **identificador del dispositivo** (host) dentro de esa subred. Es decir, esta direccionamiento jerárquico es una jerarquía de 2 niveles. Los gateways (ya se explicarán) usan únicamente una parte de esta dirección IP (dirección IP = numerito que identifica una red).
 - Cada subred tiene un identificador único para direcciones privadas o para direcciones públicas.
 - Cada dispositivo tiene un identificador único en la subred.

Un pequeño inciso aclaratorio. La IP va a ser un número de 32 bits, este numerito tiene 2 partes, una que es el identificador de la subred (una parte del número grande que identifica a la red) y otra que es el identificador de dispositivo (la otra parte del número grande que identifica al dispositivo que la usa). Bueno, pero si me dan una IP, por ejemplo 200.27.4.112, ¿cómo se yo qué parte es la que identifica la red y qué identifica al dispositivo? Pues para esto surge la llamada **máscara**.

- La máscara de red es un patrón de 1s y 0s que van a determinar qué parte de la IP es la que representa a la subred. Por ejemplo,

~ Una dirección IP es 200.27.4.112 = 11001000.00011011.00000100.01110000

~ Una máscara es 255.255.255.0 = 11111111.11111111.11111111.00000000

La máscara se puede representar como 200.27.4.112/24, ya veremos el porqué del 24.

- Bueno, ya tenemos una dirección IP y una máscara, ¿cómo se qué parte es el identificador de la red? Tenemos que hacer la operación lógica &, bit a bit. En el ejemplo tendríamos.

$$11001000.00011011.00000100.01110000 = 200.27.4.112$$

$$11111111.11111111.11111111.00000000 = 255.255.255.0$$

$$11001000.00011011.00000100.00000000 = 200.27.4.0 \rightarrow \text{identificador de red}$$

Estudiar sin publi es posible.

Compra Wuolah Coins y que nada te distraiga durante el estudio.

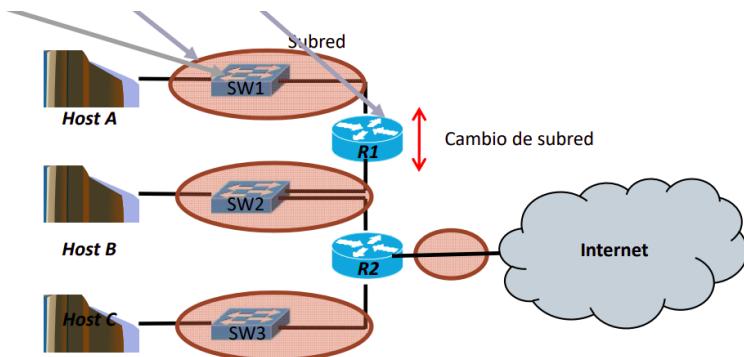


Suficiente de IP de momento, con saber que es un número de 32 bits, ese número tiene una parte que representa a la red y tiene otra que representa al dispositivo y que se pueden obtener a partir de la máscara es suficiente.

¿Qué es una subred?

Esto del numerito de la IP está muy bien, pero ¿para qué sirve? Bueno, Internet se puede considerar una red de subredes. ¿Qué es una subred entonces?

- Una subred son líneas de transmisión e infraestructuras de red que permiten una conexión **directa** de dispositivos sin intermediarios (routers).



Las subredes es lo que está en rojo, y eso de los routers y los switches están muy bonitos así que los vamos a explicar.

Switches y routers

- Un switch es un dispositivo que se encarga de interconectar los dispositivos que se encuentran en la misma subred. Esto constituye las LAN (redes de área local, explicadas en el tema 1).
- Un router es un dispositivo que se encarga de conectar subredes, que a su vez, estas subredes conectaban a los dispositivos mediante los switches.

Y dirás, pues yo en mi casa tengo un router, no tengo un switch, y lo que tengo es una subred local (LAN). Pues resulta que la tecnología avanza y ahora mismo los routers funcionan como routers y como switches, así que sí que tienes un switch en casa, porque tu router es router y switch.



Un dato importante que hay que tener en cuenta es, los hosts (dispositivos conectados en una subred, tu pc, el móvil, la consola...) y los routers tienen 1 dirección IP por cada interfaz. Pero los switches NO tienen direcciones IP.

Ea pues ya nos hemos enterado de qué son las subredes y quiénes la forman. Vamos a volver un poco a la dirección IP y las máscaras.

Máscaras

(Importante que esto lo preguntan) ¿Cómo podemos elegir la máscara que tendrá nuestra red? (Máscara, series de 1s y 0s que nos ayudaban a distinguir las 2 partes de la dirección IP. Una parte identificaba la dirección de subred y la otra a los dispositivos conectados a esa subred). Bueno, pues la máscara no tiene por qué siempre ser /24, esto dependerá del número de dispositivos que queramos conectar a nuestra subred.

Una IP podría ser 200.27.4.112 = 11001000.00011011.00000100.01110000

Al ponerle máscara 24 quiere decir que los 24 primeros bits van a identificar a la red y los 8 últimos a los dispositivos. Rojo identifican red, azul identifican dispositivos.

11001000.00011011.00000100.01110000

Genial, pues ahora es cuestión de pensar un poco. Si tengo 8 bits para representar dispositivos, ¿cuántos dispositivos se pueden identificar? Muy bien, habrá 2^8 combinaciones distintas de esos número y por tanto, 256 números distintos para identificar dispositivos. ¿Podríamos conectar 256 ordenados a nuestra subred? Pues no pero casi, esto es porque hay como 2 direcciones reservadas, la 0 y la 255 en este caso.

- 00000000 (0) es una dirección reservada para la subred.
- 11111111 (255) es una dirección reservada para la difusión.

Entonces, podremos conectar 254 ordenadores (si y no, porque normalmente hace falta un router y hemos visto que los routers necesitan 1 IP y por tanto, distinguirlos con la máscara. Pero se pueden conectar 254 dispositivos contando con ordenadores, móviles, routers...).

Pues genial, entonces si yo necesitara conectar más dispositivos a mi red local, el número de máscara debe ser menor, de esta manera puedo referenciar más máquinas. Si no necesito conectar tantos dispositivos a mi red local, entonces el número de máscara será mayor. Recordar que la máscara va desde el 0 al 31.

Direcciones públicas, privadas y NAT

Se puede entender perfectamente con estos 2 vídeos: <https://www.youtube.com/watch?v=HeZWcZmrQUY> y <https://www.youtube.com/watch?v=gVUE2IDwWA0>

Clases de IP

Bueno, esto de darle más a la máscara o menos está muy bien pero si cada uno hiciera lo que le diera la gana, tendríamos un problema. Para ello se definieron 5 clases de direcciones IP, nombradas de la A a la E. A, B y C son jerárquicas a 2 niveles como hemos explicado, tiene una parte para la red y otra para los dispositivos conectados. Cosa que no pasa con D y E. D es para multicast y E es para un uso futuro que aún no se sabe cuál es (esta gente si que es previsora no como los de IPv4). Dependiendo de si eres A, B o C tendrás más o menos hosts y redes.

Clase A	0	red (7 bits)	host (24 bits)
Clase B	1 0	red (14 bits)	host (16 bits)
Clase C	1 1 0	red (21 bits)	host (8 bits)
Clase D	1 1 10	dirección grupo <i>multicast</i> (28 bits)	
Clase E	1 1 11 0	uso futuro	

Aquí podemos ver el rango que tiene cada una, Puede que pregunten en el examen de qué tipo es una IP específica.

Clase A (/8) → 0.0.0.0–127.255.255.255	⇒	128 redes x 16.777.216 hosts
Clase B (/16) → 128.0.0.0–191.255.255.255	⇒	16.384 redes x 65.536 hosts
Clase C (/24) → 192.0.0.0–223.255.255.255	⇒	2.097.152 redes x 256 hosts
Clase D → 224.0.0.0–239.255.255.255	→	para <i>multicast</i>
Clase E → 240.0.0.0–255.255.255.255	→	usos futuros

Como hemos explicado antes, hay unas direcciones reservadas en los dispositivos, que son las que acaban en 00...0 (es una dirección origen, no se usa en dispositivos) y en 11...1 (es una dirección destino, no se usan en dispositivo, broadcast).

Si nos ponen una lista de direcciones y nos preguntan cuál sería válida para un dispositivo, tenemos que mirar que no acaben en 00...0 ni en 11...1. Teniendo en cuenta la máscara, por ejemplo, máscara 24 tiene que comprobar que los últimos 8 números no sean 00000000 o 11111111.

También, si tengo una IP pública pues tendré que asignar las IPs privadas para los dispositivos que estén en la subred. Esto depende si son de tipo A, B y C. Aquí podemos ver un ejemplo:

Reserva de direcciones privadas (RFC1918):

Clase A → 10.0.0.0 →	1 Red privada clase A
Clase B → 172.16.0.0 – 172.31.0.0 →	16 redes privadas clase B
Clase C → 192.168.0.0 – 192.168.255.0 →	256 redes privadas clase C

La gestión y asignación de las direcciones las hace la IANA e ICANN, una empresa que te asigna una dirección en función de los dispositivos que quieras conectar, etc.

NAT

Ahora vendría la NAT, pero en el vídeo que he puesto arriba lo explica perfectamente.

Asignación de redes

Mirad el vídeo que subió el profesor resolviendo el ejercicio del examen que ahí se entiende muy bien. Ya lo podría haber hecho igual antes del examen.

Lo demás aunque parezca raro, viene bien explicado en las diapositivas.

Espero haberos ayudado con explicaciones clara, ejemplos y cosas entendibles, no los 4 meses de sufrimiento de FR que hemos tenido.