Servicios Telemáticos

Jose Antonio Lorencio Abril

2019/2020

Contents

1 Introducción a los SSTT. Servicios de información y aplicaciones de red

1.1 ¿Qué es Internet?

Una descripción esencial

Internet es una red de ordenadores que interconecta centenares de millones de dispositivos en el mundo. Estos dispositivos se llaman **hosts o sistemas finales**. Estos están conectados entre sí mediante una red de **enlaces de comunicación** y **encaminadores de paquetes** (packet switch).

Diferentes enlaces pueden transmitir datos a diferentes tasas, midiendose la **tasa de transmisión** de un enlace en bits/segundo.

Cuando un sistema final tiene datos que enviar a otro sistema final, el que envía segmenta los datos y añade bytes de encabezamiento a cada segmento. Los paquetes de información resultantes, conocidos como **paquetes**, se envían a través de la red hacia el sistema final destino, donde se reensamblan para contener la misma información que antes de ser dividido.

Un encaminador toma paquetes que llegan a uno de sus enlaces de entrada y los envía a través de uno de sus enlaces de salida. Los dos tipos de encaminador más importantes en internet son los routers y los switch de la capa de enlace. Ambos tipos reenvían paquetes hacia su destino final.

Los **switch de la capa de enlace** son usados en redes de acceso.

Los **routers** son usados en el centro (core) de la red.

La secuencia de enlaces de comunicación y encaminadores atravesados por un paquete desde el host que envía hasta el que recibe se conoce como **ruta** o **camino**.

Los sistemas finales acceden la internet a través de **ISPs (Internet Service Providers)**, que pueden ser residenciales, corporativos, universitarios y los ISP que proporcionan acceso WiFi en espacios públicos. Cada ISP es, en sí mismo, una red de encaminadores de paquetes y hosts. Los ISPs proporcionan variedad de tipos de acceso a redes a los sitemas finales, incluyendo acceso residencial con banda ancha como modem por cáble o DSL. También proporcionan acceso a internet a proveedores de contenido, conectando sitios web directamente a la internet.

Los ISPs que proporcionan acceso a sistemas finales tienen que estar tambibén interconectados. Estos ISPs de los niveles bajos están interconectados a través de otros ISPs de niveles superiores, nacionales o internacionales, como Level 3 Communications o AT&T.

Un IPS de nivel superior consiste en routers de alta velocidad interconectados con enlaces de fibra óptica de alta velocidad. Cada red ISP, ya sea inferior o superior, se controla independientemente, usa los protocolos IP y se ajusta a ciertas convenciones de nombres y direcciones.

Los sistemas finales, los switches y otras partes de la internet usan **protocolos** que controlan el envío y recepción de información a través de internet.

El **protocolo IP** especifica el formato de los paquetes que se envían y reciben por routers y sistemas finales. Los principales protocolos de internet se conocen conjuntamente como **TCP/IP**.

Los **estándars de internet** se desarrollaron por el Internet Engineering Task Force. Los documentos de estandarización de IETF se conocen como **requests for comments (RFCs)**, y suelen ser bastante técnicos y detallados. Se usan para definir protocolos.

Una descripción de servicios

Internet puede definirse como "una infraestructura que proporciona sercicios a distintas aplicaciones".

Las aplicaciones se dice que son **aplicaciones distribuidas**, ya que involucran diferentes sistemas finales que itnercambian datos entre ellos. Las aplicaciones de internet se ejecutan en los hosts y no en los encaminadores. Aunque los encaminadores facilitan el intercambio de datos entre hosts, ellos no tienen nada que ver con la aplicación que genera o recibe los datos.

Los hosts conectados a intrnet proporcionan una **Application Programming Interface (API)**, que especifica cómo un programa ejecutándose en un hosts pide a la infraestructura de internet que lleve datos a un programa destino específico, que se ejecuta en otro host. Esta API de internet es un conjunto de reglas que el programa que envía tiene que seguir para que internet pueda entregar los datos al programa destino.

¿Qué es un protocolo?

Un protocolo define el formato y el orden de los mensajes intercambiados entre dos o más entidades de comunicación, así como las acciones llevadas a cabo en la transmisión y/o recepción de un mensaje o algún otro evento.

Cualquier actividad en Internet que involucre dos o más entidades de comunicación remotas debe usar un protocolo.

1.2 El extremo de la red

Los hosts se pueden dividir en dos categorías:

- **Clientes:** suelen ser ordenadores portátiles y de escritorio, smartphones,...
- **Servidores:** son máquinas más potentes que almacenan y distribuyen páginas web, video streaming, retransmiten emails,... La mayoría de los servidores de los que recibimos información están localizados en grandes **centros de datos**.

Redes de acceso

Una red de acceso es una red que conecta físicamente un host al primer router (**edge router**) en un camino desde el host hasta cualquier otro host distante.

Acceso de hogar: DSL, Cable, FTTH, línea conmutada, y satélite

- **Digital Subscriber Line (DSL):** una residencia normalmente obtiene conexión DSL a internet de la misma compañia de teléfono local que proporciona su acceso local inalámbrico telefónico. Cuando se usa DSL, estas compañías actúan como ISP.

  Cada módem DSL de un consumidor usa la línea de teléfono existente para intercambiar datos con un multiplexor de acceso de línea de abonado digital (DSLAM) localizado en la oficina central local de la compañía (CO). El módem DSL del hogar toma datos digitales y los traduce a ondas de alta frecuencia para ser transmitidos por los cables telefónicos hasta el CO; las señales analógicas de muchos hogares se vuelven a traducir de forma inversa en formato digital en el DSLAM.

  La línea residencial telefónica lleva tanto datos como la señal telefónica tradicional simultáneamente, por lo que deben codificarse en distintas frecuencias:

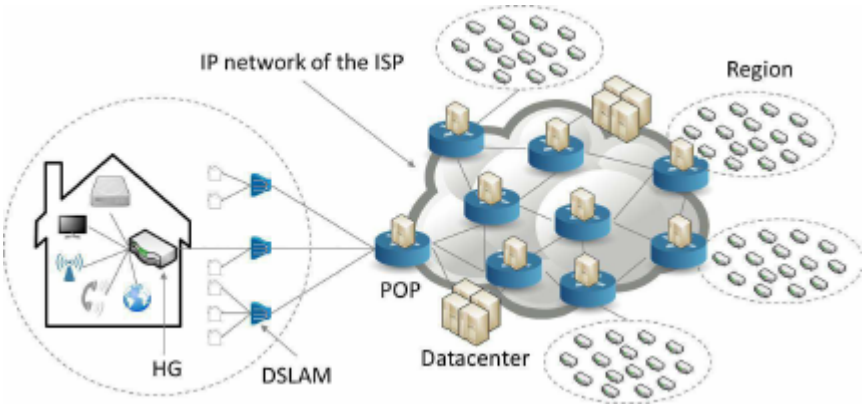
  - Canal de bajada de alta velocidad: 50kHz-1MHz
  - Canal de subida de velocidad media: 4-50kHz
  - Canal telefónico de doble sentido: 0-4kHz

  Esto hace que un único enlace DSL funcione como si albergase tres enlaces distintos, de forma que una llamada telefónica y una conexión a internet pueden compartir el enlace DSL.

  En el lado del cliente, un separador (splitter) divide los datos y las señales telefónicas que llegan al hogar y reenvía la señal de datos al módem DSL.

En el lado de la compañía, en la CO, el DSLAM separa los datos y las señales telefónicas y envía los datos hacia internet.

Como las tasas de subida (1.8Mbps ó 2.5Mbps) y bajada (12Mbps ó 24Mbps) son distintas, el acceso se dice que es **asimétrico**.



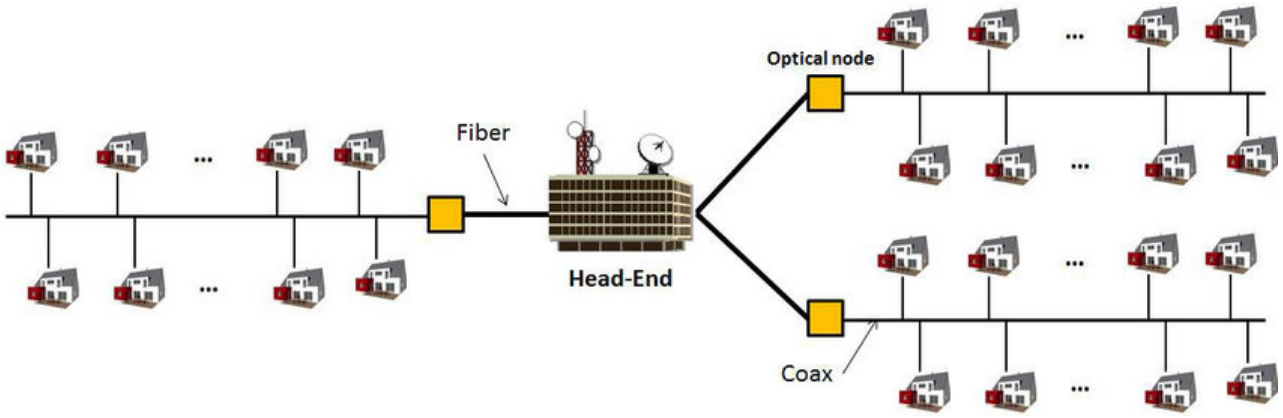
- **Acceso a internet por cable:** hace uso del cable preexistente de las compañías de televisión. Una residencia obtiene acceso a internet por cable de la misma compañía que le proporciona televisión por cable. La **fibra óptica** conecta la cabeza del cable a una intersección a nivel de vecindario, desde donde el cable coaxial tradicional se usa para alcanzar los hogares individuales. Como en este sistema se usan tanto fibra óptica como cable coaxial, a veces se le llama **hybrid fiber coax (HFC)**.

  El acceso por cable a internet requiere de módems especiales, llamados módems de cable. Como los módem DSL, los de cable normalmente son un dispositivo externo y conectan con el PC del hogar a través de un puerto ethernet.

  En la cabeza del cable, el sistema terminal del módem de cable (CMTS) hace una función similar a la del DSLAM en las redes DSL. O sea, los módems de cable dividen la red HFC en dos canales, uno de bajada y otro de subida. El acceso es, como antes, asimétrico.

  Otra característica importante del acceso por cable a internet es que es un medio de broadcast compartido, esto quiere decir que todo paquete envíado por la cabeza viaja por el camino de bajada por todos los enlaces de todos los hogares conectados y todo paquete envíado por un hogar viaja por el camino de subida hasta la cabeza. Por este motivo, si varios usuarios están descargando un video por el canal de bajada, la tasa a la que cada usuario recibe el vídeo será significativamente menor que la tasa de bajada agregada del cable. Si hay pocos usuarios sus tasas de bajada serán mayores.

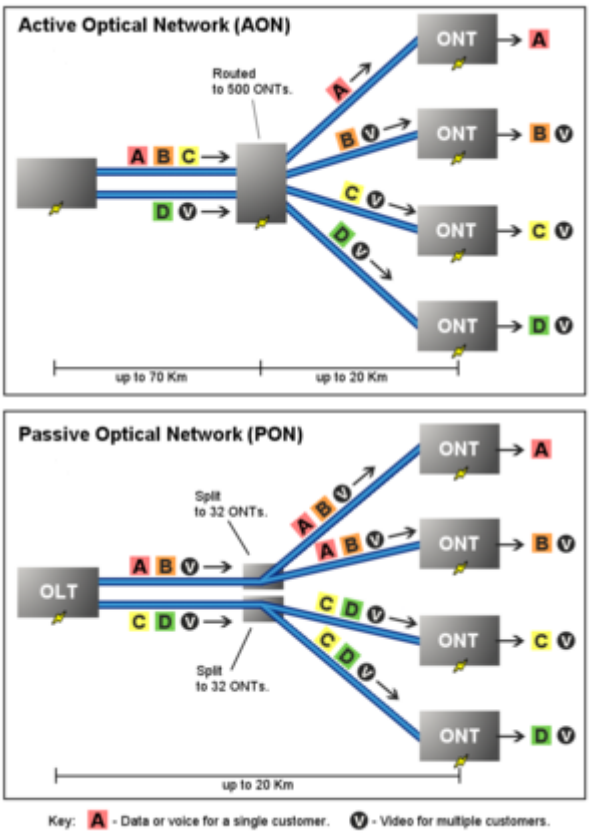
  Como el canal de subida también es compartido, se hace necesario un protocolo de acceso múltiple y distribuido para coordinar transmisiones y evitar colisiones.



- **FTTH (Fiber To The Home,** fibra hasta el hogar**):** es un concepto simple, proporcionar un camino de fibra óptica desde el CO directamente al hogar.

  Hay varias tecnologías compitiendo en este nicho. La red de distribución de fibra más simple es la **fibra directa**, con una fibra saliendo de la CO para cada hogar. Más comúnmente, cada fibra que sale de la CO se comparte por varios hogares, y no es hasta que la fibra está relativamente cerca de los hogares cuando se separa en fibras individuales para cada consumidor. Para esta segunda opción hay dos principales tecnologías competidoras:

  - **Active optical networks (AON):** esencialmente ethernet conmutado
  - **Passive optical networks (PON):** cada hogar tiene un terminal de red óptica (ONT), que se conecta por una fibra óptica dedicada a un separador por vecindario. El separador combina una cantidad de hogares en una única y compartida fibra óptica, que conecta a un terminal de línea óptica (OLT) en la CO de la compañía. El OLT proporciona la conversión entre señales ópticas y electricas, conecta a internet por un router de la compañía. En el hogar, los usuarios conectan un router de hogar (normalmente inalámbrico) al ONT y acceden internet vía este router del hogar. En la arquitectura PON, todos los paquetes envíados desde el OLT al separador son duplicados en el separador.



FTTH potencialmente puede proporcionar tasas de acceso a internet del orden de los gigabits por segundo.

- Otras dos tecnologías de acceso a la red son usadas para proporcionar conexión a internet desde el hogar.
  - **Enlace por satélite:** puede ser usado para conectar una residencia al internet a velocidades superiores a 1Mbps
  - **Enlace por línea conmutada:** se apoya sobre las líneas telefónicas tradicionales y se basa en la misma idea que DSL. Un módem de hogar conecta por una línea telefónica a un módem en el ISP.

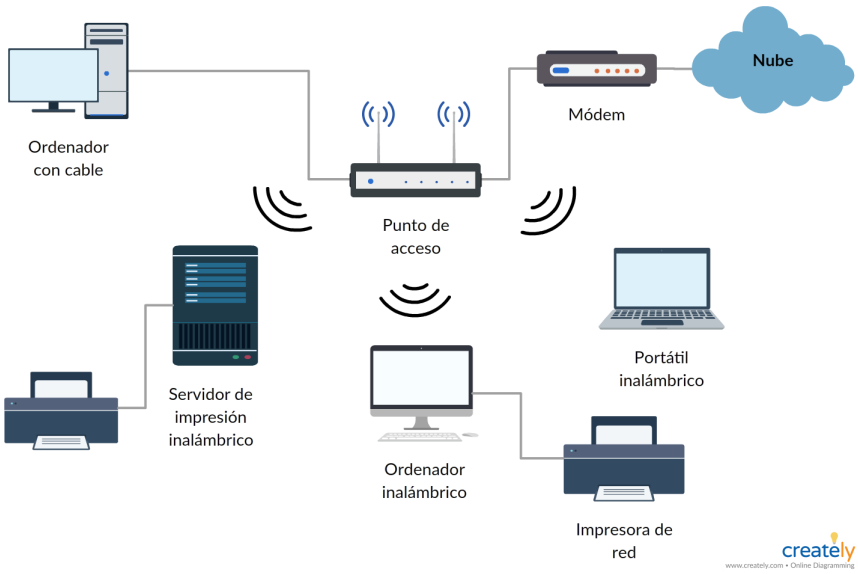
Acceso en la empresa (y el hogar): Ethernet y WiFi

En las empresas y los campus de las universidades, y, cada vez más, en los hogares, es normal usar una **red de área local (LAN)** para conectar un sistema final al router frontera. Aunque hay diversos tipos de tecnologías LAN, **Ethernet** es la más extendida.

Los usuarios de Ethernet usan cable de cobre de par trenzado para conectarse a un switch Ethernet. Este, o una red de switches interconectados, entonces se conecta a la internet. Con el acceso Ethernet, los usuarios normalmente tienen 100Mbps de acceso al switch Ethernet.

La gente, cada vez más, está accediendo la internet inalámbricamente, desde portátiles, móviles,... En una **configuración LAN inalámbrica** los usuarios envían/reciben paquetes a/desde un punto de acceso que está conectado a la red de la empresa o el hogar, que estará conectada a la internet cableada. Los usuarios de una LAN inalámbrica deben estar a unas pocas decenas de metros del punto de acceso si estamos hablando de tecnología IEEE 802.11 o **WiFi**.

Aunque las redes de acceso por Ethernet o WiFi inicialmente se desplegaron en la empresa, cada vez es más normal verlas en los hogares. Muchos hogares combinan acceso residencial de banda ancha con estas tecnologías LAN inalámbricas, que son baratas, para crear potentes redes en el hogar.



Acceso Inalámbrico de Larga Distancia: 3G y LTE

Las compañías de telecomunicaciones han hecho enormes inversiones en las tecnologías llamadas **inalámbricas de tercera generación (3G)**, que proporcionan acceso inalámbrico a internet en una gran área y por conmutación de paquetes, a unas velocidades de 1Mbps.

LTE (Long Term Evolution) se basa en las tecnologías 3G y puede, potencialmente, alcanzar tasas de 10Mbps.

1.2.1 Medios físicos

Los medios físicos pueden ser de varios tipos e, incluso, en una ruta de internet pueden haber diferentes medios físicos involucrados. Podemos dividirlos en dos categorías:

- **Medios guiados:** las ondas son guiadas a través de un medio sólido
- **Medios no-guiados:** las ondas se propagan en la atmósfera o en el espacio

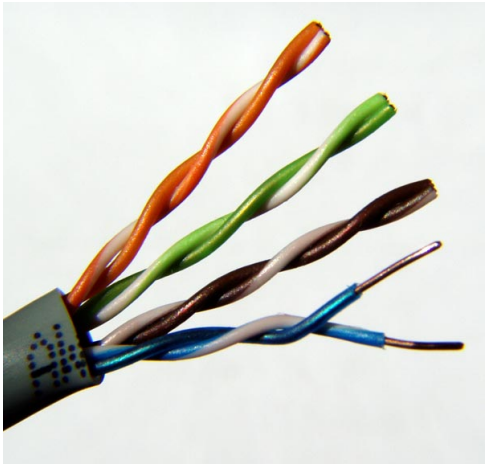
El coste actual de los enlaces físicos suele ser relativamente pequeño comparado con otros costes de la red. Es particularmente cara la instalación de los enlaces físicos, mucho más que el propio enlace.

Cable par trenzado de cobre

Es el menos caro y el más utilizado de los medios guiados.

Consiste en dos cables de cobre aislados, cada uno de 1mm de ancho, posicionados en un patrón regular en espiral. Los cables se enrollan juntos para reducir la interferencia eléctrica de otros pares similares cercanos.

Normalmente, una cantidad de pares se agrupan en un cable envolviéndolos en un escudo protector.



Un par cableado constituye un único enlace de comunicación.

El **par trenzado sin escudo (UTP)** se usa en redes de ordenadores dentro de un edificio, con tasas entre 10Mbps y 10Gbps.

La tecnología moderna de par trenzado puede alcanzar tasas de 10Gbps en distancias de hasta cientos de metros. Así, el apr trenzado se ha posicionado como la solución dominante para las redes LAN de alta velocidad.

Cable coaxial

Consiste en dos conductores de cobre, pero en este caso son concéntricos. Con esta construcción y esta separación y protección, el cable coaxial puede alcanzar grandes tasas de transmisión. Es muy usado en sistemas de televisión por cable.

En televisión por cable y acceso a internet por cable, el transmisor cambia la señal digital original a una frecuencia específica, y la señal analógica resultante es enviada desde el transmisor a uno o más receptores. El cable coaxial puede usarse como un medio guiado compartido.

Fibra óptica

Es un medio fino y flexible que conduce pulsos de luz, con cada pulso representando un bit. Una única fibra óptica puede transmitir a altas tasas, hasta las decenas o centenares de Gbps. Son inmunes a las interferencias electromagnéticas y tienen muy poca atenuación de la señal hasta los 100Km, además son muy difíciles de interceptar por agentes indeseados. Estas características hacen de la fibra óptica el medio guiado preferido para recorridos largos, particularmente para enlaces transoceánicos.

La fibra óptica es predominante también en el backbone de internet.

Sin embargo, el alto coste de los dispositivos ópticos ha obstaculizado su desplegamiento para transporte de corta distancia.

Canales de radio terrestres

Los canales de radio transmiten señales del espectro electromagnético. Son medios atractivos porque no requieren cables que instalar, pueden atravesar paredes, dar conexión a usuarios móviles y, potencialmente, transmitir señales en largas distancias.

Las características de un canal de radio depende mucho del entorno de propagación y de la distancia que la señal debe atravesar. Las condiciones del entorno pueden determinar la aparición de pérdidas de información o disipación de las ondas, interferencias de una señal consigo misma debido a rebotes e interferencias con otras señales.

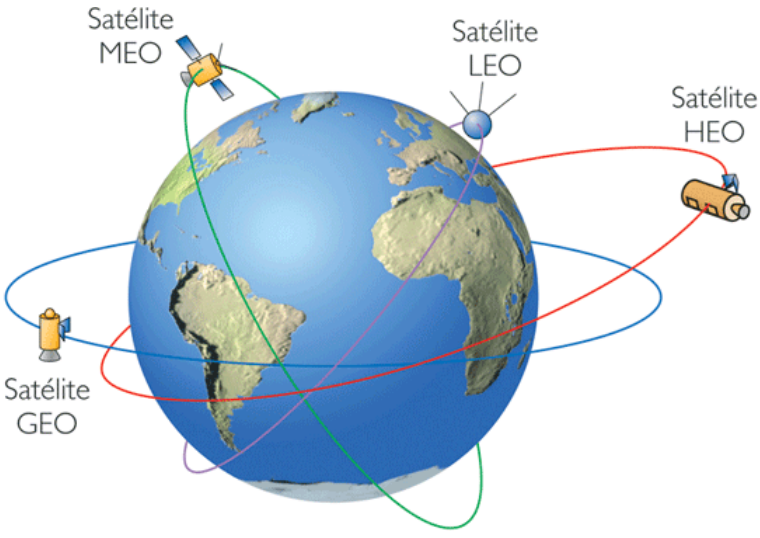
Pueden ser clasificados en tres grupos: los que operan en distancias muy cortas; los que operan en áreas locales (10 a algunos cientos de metros); y los que operan en grandes áreas (decenas de kilómetros).

Canales de radio por satélite

Un satélite de comunicación enlaza dos o más transmisores/receptores de microondas localizados en la tierra, conocidos como estaciones terrestres. El satélite recibe transmisiones en una frecuencia, regenera la señal usando un repetidor y la trasmite en otra frecuencia.

En comunicaciones se usan dos tipos de satélites:

- **Satélites geoestacionarios:** permanecen orbitando sobre el mismo punto de la tierra, a 36000Km de altura. El retardo de propagación es de 280 ms
- **Satélites LEO (órbita terrestre baja):** están mucho más cerca de la tierra y no están siempre sobre el mismo punto. Giran alrededor de la tierra y pueden comunicarse unos con otros. Para proporcionar cobertura continua en un área, muchos satélites tienen que ser lanzados a órbita.



1.3 El núcleo de la red

1.3.1 Conmutación de paquetes

En una aplicación de red, los hosts intercambian **mensajes** entre ellos. Estos pueden contener lo que sea que el diseñador de la aplicación desee. Los mensajes pueden desarrollar una función de control o pueden contener datos. Para enviar un mensaje desde un emisor a un destino, el emisor parte los mensajes largos en bloques más pequeños de datos, conocidos como **paquetes**. Entre la fuente y el

destino, cada paquete atraviesa enlaces de comunicación y encamincadores. Los paquetes se transmiten por cada enlace de comunicación a una tasa igual a la tasa total de transmisión del link.

$$t_{tr} = \frac{L}{R}$$

O sea, el tiempo de transmisión es el tamaño del paquete, L, dividido entre la tasa de transferencia del enlace, R.

Transmisión por almacenamiento y reenvío (store-and-forward)

Los encaminadores deben recibir el paquete entero antes de poder transmitir el primer bit del paquete por el enlace de salida. Un router normalmente tendrá varios enlaces conectados, ya que su trabajo es es conmutar un paquete entrante a uno saliente. Veamos el proceso de envío de un paquete:

- El emisor empieza a transmitir en el instante 0.
- En el instante $\frac{L}{R}$ s, el emisor ha transmitido el paquete entero, y este ha sido recibido y almacenado en el router.
- En este mismo instante, como el router ya tiene el paquete entero, puede comenzar a transmitirlo por un enlace de salida hacia el destino.
- En el instante $\frac{2L}{R}$ s, el router ha transmitido el paquete entero, y este ha sido recibido por el destinatario.

Por tanto, el retardo total es de $\frac{2L}{R}$ s. Si el router transmitiese tan pronto como le llegan los bits, entonces el retraso total sería $\frac{L}{R}$ s, ya que los bits no esperarían en el router. Pero esto no es posible, los routers necesitan recibir, almacenar y procesar el paquete completo antes de poder enviarlo.

Ahora vamos a ver qué sucede al enviar tres paquetes:

- Igual que antes, en el instante $\frac{L}{R}$, el router comienza a enviar hacia el destino el primer paquete. Pero también en este instante el emisor comienza a enviar el segundo paquete, ya que ha terminado de enviar el primero.
- Así, en el instante $\frac{2L}{R}$, el destino ha recibido el primer paquete y el router el segundo.
- En $\frac{3L}{R}$, el destino ha recibido el segundo paquete y el router el tercero.
- Finalmente, en $\frac{4L}{R}$, el destino recibe los tres paquetes completos.

Generalizando, el retraso experimentado al enviar $N-1$ paquetes, es

$$t_{tr}(N-1) = N \cdot \frac{L}{R}$$

Retrasos en colas y pérdida de paquetes

Cada encaminador tiene múltiples enlaces conectados. Y para cada uno de estos enlaces, tienen un **buffer de salida (output buffer)** que almacena paquetes que el router está apunto de enviar por ese enlace.

Si un paquete entrante necesita ser transmitido por un enlace pero este está ocupado con la transmisión de otro paquete, el paquete entrante debe esperar en el buffer de salida. Así, además de los retardos por almacenamiento y reenvío, los paquetes sufren **retardos por colas** en los buffers de salida.

Además, como el espacio de un buffer es limitado, un paquete entrante podría encontrarse el buffer lleno con otros paquetes esperando ser transmitidos. En este caso incurriremos en **pérdida de paquetes**, pues el paquete entrante no puede ser almacenado y debe descartarse.

Tablas de rutas y protocolos de enrutamiento

En la internet, cualquier host tiene una dirección llamada **dirección IP**. Cuando un emisor quiere enviar un paquete a un destino, el emisor incluye la IP del destinatario en la cabecera del paquete.

Esta dirección tiene una estructura jerárquica. Cuando un paquete llega a un router en la red, este examina una porción de la dirección del paquete de destino y lo reenvía a un router adyacente. Concretamente, cada router tiene una **tabla de rutas** que mapea direcciones de destino con enlaces de salida. Cuando un paquete llega al router, este examina la dirección y busca en su tabla de rutas, usando la dirección de destino, para encontrar el enlace de salida correcto. Entonces, el router reenvía el paquete por este enlace de salida.

Los **protocolos de enrutamiento** son usados para rellenar automáticamente estas tablas de rutas.

1.3.2 Conmutación de circuitos

Hay dos formas principales de encaminar datos a través de una red de enlaces y encaminadores: conmutación de paquetes (como hemos visto antes) o conmutación de circuitos.

En las redes de circuitos conmutados, los recursos necesarios a lo largo de un camino para proporcionar la comunicación entre los hosts se reserva por la duración de la sesión de comunicación entre los hosts.

Por el contrario, en las redes de paquetes conmutados, estos recursos no se reservan, un mensaje de una sesión usa los recursos en demanda y, como consecuencia, deberá esperar para acceder al enlace.

Antes de que el emisor pueda enviar información, la red debe establecer una conexión entre el emisor y el receptor. Esto es una **conexión de buena fe** para la que los switches en el camino entre los dos hosts mantienen un estado de conexión para esta conexión. Esto se llama un **circuito**. Cuando la red establece el circuito, también reserva una tasa constante de transmisión en los enlaces de la red para la duración de la conexión.

Multiplexión en redes de circuitos conmutados

Un circuito en un enlace se implementa de dos formas:

- **Multiplexión por división de frecuencia (FDM):** el espectro de frecuencias de un enlace se divide para las conexiones establecidas a través del enlace. Más concretamente, el enlace dedica una horquilla de frecuencias a cada conexión durante lo que dure la sesión. El ancho de esta horquilla se denomina **ancho de banda**.
- **Multiplexión por división de tiempo (TDM):** el tiempo se divide en marcos de duración fija, y cada marco se divide en una cantidad fija de espacios. Cuando la red establece una conexión a lo largo de un enlace, la red dedica un espcio de cada marco a esta conexión. Estos espacios están dedicados para el único uso de esa conexión, con un espacio de tiempo disponible por uso para transmitir los datos de la conexión.

Los defensores de la conmutación de paquetes han argumentado que la conmutación de circuitos es ineficiente porque los circuitos dedicados permanecen ociosos durante períodos de silencio. También apuntan que establecer los circuitos extremo a extremo y reservar la capacidad de transmisión es complicado y requiere software de señalización complejo para coordinar la operación de los switches a lo largo del camino extremo a extremo.

Conmutación de paquetes VS Conmutación de circuitos

Los críticos de la conmutación de paquetes argumentan que esta no es apropiada para servicios en tiempo real debido a sus retrasos impredecibles y variables. Los defensores de los paquetes conmutados argumentan que:

1. Ofrece mejor compartición de la capacidad de transmisión que los circuitos conmutados
2. Es más simple, más eficiente y menos costoso de implementar que los circuitos conmutados

Los circuitos conmutados preasignan el uso del enlace de transmisión independientemente de la demanda, con enlaces asignados pero innecesarios que quedan ociosos.

Por otro lado, los paquetes conmutados usan el enlace en demanda. La capacidad de transmisión del enlace será compartida en una dinámica paquete a paquete solo entre aquellos usuarios que tienen paquetes que necesitan transmitir a través del enlace.

1.3.3 Una red de redes

Los ISPs de acceso deben estar interconectados entre sí. Esto se hace creando una red de redes.

A lo largo de los años, la red de redes que conforma internet ha evolucionado a una estructura muy compleja. Gran parte de esta evolución se debe a asuntos económicos y políticas nacionales, más que a consideraciones sobre la eficiencia.

Vamos a proceder a una construcción de más sencillez a más complejidad de la red:

- **Estructura de Red 1:** interconecta todos los ISPs de acceso a través de un único ISP global de tránsito. Nuestro ISP global de tránsito es una red de routers y enlaces de comunicación que no solo abarca el mundo, pero también tiene al menos un router cerca de cada uno de los ISPs de acceso. Sería muy costoso para el ISP global construir una red tan extensa. Para ser rentable, debería cobrar a cada ISP de acceso por la conectividad, con el precio reflejando la cantidad de tráfico que un ISP intercambia con el ISP global. Como los ISP de acceso pagan al ISP global de tránsito, el ISP de acceso se dice que es un **cliente** y el ISP global se dice que es un **proveedor**.

- **Estructura de Red 2:** consiste en cientos de miles de ISPs de acceso y múltiples ISPs de tránsito globales. Los ISPs de acceso prefieren la estructura de red 2 antes que la 1, ya que ahora pueden elegir entre los ISPs de tránsito competidores en función del precio y los servicios ofrecidos. Nótese, sin embargo, que los ISPs de tránsito globales deben interconectarse también entre ellos.

  Así, esta segunda estructura es una jerarquía de dos niveles con proveedores de tránsito globales en el nivel superior y ISPs de acceso en el inferior. Aquí asumimos no solo que los ISPs de tránsito son capaces de dar servicios cerca de los ISP de acceso, sino que lo encuentran económicamente deseable.

- **Estructura de Red 3:** en muchas regiones, podría haber **ISP regionales** a los que los ISPs de acceso en esa región se conectarían. Cada ISP regional conectaría entonces con algún **ISP de nivel 1**, que son similares a nuestros ISPs de tránsito globales de la estructura 2, pero los ISPs de nivel 1, que sí existen en la realidad, no tienen presencia en todas las ciudades del mundo.

  Además, no solo hay múltiples ISPs de nivel 1 competidores, también puede haber múltiples ISPs regionales competidores en una región. En una jerarquía como esta, cada ISP de acceso paga el ISP regional al que se conecta, y cada ISP regional paga al ISP de nivel 1 al que se conecta. Por tanto, hay una relación cliente-proveedor en cada nivel de la jerarquía.

  En algunas regiones, podría haber un ISP regional muy grande al que otros ISPs regionales más pequeños se conectarían. Después, el ISP regional grande se conectaría a un ISP de nivel 1.

Para acercarnos más aún a la red que conforma internet, debemos añadir los **puntos de acceso (PoPs)**, que existen en todos los niveles de la jerarquía excepto en el menor de todos. Un PoP simplemente es un grupo de uno o más routers en la red del proveedor donde los ISPs clientes pueden conectarse al ISP proveedor.

Una red cliente, para conectarse a un Pop de un proveedor, puede alquilar un enlace de alta velocidad a un tercer proveedor de telecomunicaciones, para conectar directamente uno de sus routers a un router del PoP.

Cualquier ISP (menos los de primer nivel) podría elegir hacer **multi-home**, o sea, conectar dos o más ISPs proveedores. Cuando un ISP multihomea, puede continuar enviando y recibiendo paquetes de internet incluso si uno de los dos proveedores tiene un fallo.

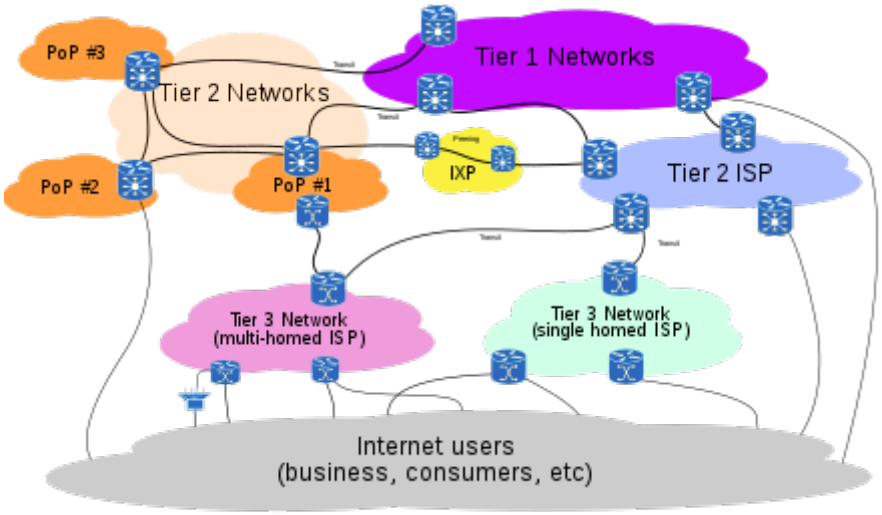
Ahora bien, la cantidad que un ISP cliente paga a un ISP proveedor refleja la cantidad de tráfico que intercambia con este. Para reducir estos costes, un par de ISPs cercanos del mismo nivel podrían aliarse (**Peer**), conectando sus redes directamente y dejando que el tráfico pase a través de ellos en lugar de subir a ISPs de niveles superiores. Normalmente sib acuerdos igualitarios (settlement-free), es decir, que ninguno de ellos paga nada al otro.

También podría darse el caso de que una compañía de terceros crease un **Punto de Intercambio de Internet (IXP)**, que es un lugar donde múltiples ISP pueden hacer peering entre ellos. Todo este exosistema final conjunta la **Estructura de Red 4**.

La **Estructura de Red 5** describe la internet de 2012. Se construye sobre la cuarta, añadiendo **redes proveedoras de contenido**, que, creando su propia red, no solo reducen los pagos a ISPs de niveles superiores, sino que tienen mayor control sobre cómo sus servicios son entregados, en última instancia, a los usuarios finales.

Así, la internet de hoy en día es compleja, consiste en una docena o más de ISPs de nivel uno y cientos de miles de ISPs de niveles inferiores. Los ISPs son variados por su cobertura, con algunos abarcando varios continentes y oceános y otros limitados a regiones geográficas muy pequeñas. Los ISPs inferiores conectan a los superiores, y los superiores están interconectados entre sí. Los usuarios y los proveedores de contenido son clientes de ISPs inferiores, y estos los son de los superiores.

En los últimos años, los proveedores de contenido más grandes también han creado sus propias redes y conectan directamente con ISPs inferiores allí donde es posible.



1.4 Capas de protocolos y sus modelos de servicio

1.4.1 Arquitectura por capas

Una arquitectura por capas nos permite explicar cada parte específica y bien definida de un gran y complejo sistema. Esta simplificación es, en sí misma, de un valor considerable, ya que proporciona modularidad, haciendo mucho más fácil cambiar la implementación del servicio ofrecido por una capa. Mientras que la capa proporcione el mismo servicio a la capa superior, y use los mismos servicios de la capa inferior, el resto del sistema permanece inalterado cuando la implementación de una capa cambia.

Organización en capas de los protocolos

Para proporcionar estructura al diseño de los protocolos de red, los diseñadores de redes organizan los protocolos en capas. Cada protocolo pertenece a una capa.

Nosotros estamos interesados en los **servicios** que una capa ofrece a la inmediatamente superior, el llamado **modelo de servicio de una capa**. Cada capa proporciona su servicio:

1. realizando ciertas acciones dentro de la capa
2. usando los servicios de la capa inmediatamente inferior

Una capa de protocolos puede ser implementada en software, hardware o en una combinación de ambos.

Los **protocolos de la capa de aplicación** son casi siempre implementados en software en los hosts, al igual que los **protocolos de la capa de transporte**.

Como la **capa física** y la **capa de enlace de datos** son responsables de manejar la comunicación sobre un enlace en específico, normalmente se implementan en una tarjeta de red en la interfaz correspondiente al enlace.

La **capa de red** suele ser una implementación mixta de hardware y software. Esta capa es una capa de protocolos distribuidos por los hosts, encaminadores y otros componentes de las redes.

La organización en capas de los protocolos tiene ventajas conceptuales y estructurales. Esta organización proporciona una forma estructurada de discutir los componentes del sistema y la modularidad facilita la actualización de estos. Algunos investigadores e ingenieros de redes se oponen a este tipo de organización en capas. Una potencial desventaja es que una capa podría duplicar funcionalidad de una capa inferior; otra es que la funcionalidad de una capa podría requerir información de otra capa, lo que viola el objetivo de la organización en capas.

Cuando los juntamos, los protocolos de las distintas capas formas la **pila de protocolos (protocol stack)**, que consiste en cinco capas: física, enlace, red, transporte y aplicación.

Capa de aplicación

Es donde las aplicaciones de red y sus protocolos de la capa de aplicación residen. Incluye muchos protocolos, como HTTP o FTP. Ciertas funciones de red, como la traducción DNS, que traduce direcciones legibles por humanos a su correspondiente dirección de red. Un protocolo de la capa de aplicación es distribuido por múltiples hosts, con la aplicación en un host usando el protocolo para intercambiar paquetes de información con la aplicación en otro host. Nos referiremos a este paquete de información en la capa de apicación como **mensaje**.

Capa de transporte

Transporta mensajes de la capa de aplicación entre extremos de la aplicación. Hay dos protocolos de transporte:

- **Transmission Control Protocol (TCP):** proporciona a las aplicaciones un servicio orientado a conexión. Incluye entrega garantizada de mensajes y control de flujo. Además divide los mensajes en segmentos más pequeños y proporciona mecanismos de control de la congestión, de forma que el emisor reduzca su tasa de envío cuando la red está sobrecargada.
- **User Datagram Protocol (UDP):** proporciona un servicio no orientado a conexión. Es un servicio simple que no ofrece fiabilidad, ni control de flujo o congestión.

A los paquetes de la capa de transporte los llamaremos **segmentos**.

Capa de red

Es la responsable de mover los los paquetes de la capa de red, conocidos como **datagramas**, de un host a otro. El protocolo de la capa de transporte en un emisor pasa un semento de la capa de transporte y la dirección del destino a la capa de red, esta proporciona el servicio de entrega del segmento a la capa de transporte del host de destino.

Incluye el **protocolo IP**, que define los campos de un datagrama, así como cómo los hosts y routers actúan sobre estos campos. Solo hay un protocolo IP, y todos los componentes de internet que tienen una capa de transporte deben usarlo. La capa de red también contiene protocolos de enrutamiento que determinan las rutas que sigue un datagrama entre dos hosts. Hay muchos protocolos de enrutamiento distintos.

Capa de enlace

Para mover los paquetes de un nodo al siguiente en la ruta, la capa de red se apoya en los servicios de la capa de enlace. En cada nodo, la capa de red pasa el datagrama a la capa de enlace, que entrega el datagrama al siguiente nodo por la ruta. En el siguiente nodo, la capa de enlace pasa el datagrama a la capa de red.

Los servicios proporcionados por la capa de enlace dependen de los protocolos específicos de la capa de enlace que se emplean en el enlace.

Como los datagramas normalmente tienen que atravesar varios enlaces para ir de una fuente a un destino, un datagrama puede ser manipulado por distintos protocolos de la capa de enlace en diferentes enlace a lo largo de la ruta.

Los paquetes en la capa de enlace se denominan **marcos (frames)**.

Capa física

El trabajo de la capa física es mover los bits individuales de un marco de un nodo a otro. Los protocolos en esta capa son, de nuevo, dependientes del enlace utilizado y dependen más aún del medio de transmisión del enlace.

1.4.2 El modelo OSI

La pila de protocolos de internet TCP/IP no es la única pila de protocolos que existe. El modelo **OSI, Open Systems Interconnection**, está organizado en siete capas: aplicación, presentación, sesión, transporte, red, enlace y física. La funcionalidad de las cinco capas con el mismo nombre en la pila TCP/IP es la misma.

La **capa de presentación** proporciona servicios que permiten a aplicaciones en comunicación interpretar el significado de los datos intercambiados. Estos servicios incluyen compresión de datos y su encriptación y desencriptación.

La **capa de sesión** proporciona servicios de delimitación y sincronización de intercambio de datos, incluyendo la construcción de puntos de control y esquemas de recuperación.

1.4.3 Encapsulación

De forma similar a los hosts, los routers y switches de la capa de enlace organizan su hardware y software de red en capas. Pero los router y switches no implementan todas las capas de la pila de protocolos, normalmente solo implementan las capas inferiores.

La encapsulación consiste en lo siguiente:

- En el emisor, un mensaje de la capa de aplicación se pasa a la capa de transporte
- La capa de transporte coge el mensaje y le añade información adicional, que será usada por la capa de transporte del receptor. El mensaje de la capa de aplicación, junto con la cabecera de la capa de transporte se denomina **segmento de la capa de transporte**. Este, por tanto, encapsula el mensaje de la capa de aplicación. La información añadida puede incluir información que permita a la capa de transporte del receptor entregar el mensaje a la aplicación correcta y saber si el mensaje ha sufrido cambios en la ruta mediante bits de detección de errores. La capa de transporte entonces pasa el segmento a la capa de red.

- La capa de red añade una cabecera de la capa de red, con información como las direcciones de emisor y destino, creando un **datagrama de la capa de red**. Este es pasado a la capa de enlace.
- La capa de enlace, de nuevo, añadirá una cabecera con información útil y conforma el **marco de la capa de enlace**.
- Las cabeceras se van eliminando cuando los mensajes se pasan a las capas superiores

En general, vemos como un mensaje tiene dos tipos de campos: campos de cabecera y campos de carga útil (payload).

1.5 Principios de las aplicaciones de red

El **desarrollo de aplicaciones de red** consiste en escribir programas que corren en diferentes hosts y se comunican entre ellos a través de la red. El software no tiene que programarse para funcionar en routers ni switches, únicamente en hosts.

1.5.1 Arquitecturas de las aplicaciones de red

Desde el punto de vista del desarrollador, la architectura de red es fija y proporciona un conjunto específico de funciones a sus aplicaciones. La **arquitectura de la aplicación** es diseñada por el desarrollador y dicta cómo se estructura la aplicación sobre los distintos hosts. Hay varios tipos:

- **Cliente-servidor:** hay un host que permanece siempre encendido, el **servidor**, que analiza peticiones de otros muchos hosts, llamados **clientes**. El servidor tiene una dirección IP fija y conocida, de forma que los clientes pueden conectarse en cualquier momento enviando un paquete a esa dirección IP.

  En muchas ocasiones, un único servidor es incapaz de procesar todas las peticiones que le llegan, para esto están los **centros de datos**, que albergan gran cantidad de hosts y pueden usarse para crear un servidor virtual muy potente. Pueden tener cientos de miles de servidores, que deben ser suministrados de electricidad y mantenidos. Además, los proveedores de servicio deben pagar periódicamente la conexión y el ancho de banda para enviar datos desde sus centros de datos.

- **Peer to peer (P2P):** hay una dependencia mínima o nula en servidores dedicados de centros de datos. En lugar de esto, la aplicación explota la comunicación directa entre pares de hosts conectados de forma intermitente, llamados **peers**. Los peers no son únicamente del proveedor de servicios, también pueden ser dispositivos controlados por usuarios. Una de las características más atractivas de la arquitectura P2P es que es auto-escalable, es decir, que cuantos más usuarios hay usando la aplicación, más potencia de procesamiento hay. Sin embargo, las aplicaciones P2P venideras tienen que plantar cara a tres desafíos principalmente:
  - **ISP friendly:** la mayoría de ISPs residenciales han optado por el ancho de banda asimétrico, proporcionando mucho más ancho de banda de bajada que de subida. Pero las aplicaciones P2P de streaming de vídeo o de distribución de archivos cambian el tráfico saliente desde servidores a ISPs residenciales, poniendo bastante carga en estos.
  - **Seguridad:** devido a su naturaleza distribuida y abierta, las aplicaciones P2P pueden ser difífiles de mantener seguras.
  - **Incentivos:** el éxito de aplicaciones P2P futuras depende de convencer a los usuarios de prestar voluntariamente ancho de banda, almacenamiento y recursos de computación a las aplicaciones, esto es el desafío del diseño con incentivos.

- **Arquitecturas híbridas:** algunas aplicaciones combinan los dos modelos vistos anteriormente.

1.5.2 Comunicación de procesos

Un **proceso** es un programa que está ejecutándose en un host. Cuando los procesos están ejecutándose en un mismo host, pueden comunicarse entre ellos con comunicación entre procesos.

Los procesos que se ejecutan en dos hosts diferentes se comunican entre ellos mediante intercambio de mensajes a través de la red. Un proceso emisor crea y envía mensajes por la red; un proceso receptor recibe esos mensajes y posiblemente responde enviando mensajes de vuelta.

Procesos del cliente y el servidor

Una aplicación de red consiste en pares de procesos que envían mensajes el uno al otro a través de la red. Para cada par de procesos en comunicación, normalmente uno será el cliente y otro será el servidor. En algunas aplicaciones un proceso puede ser tanto cliente como servidor.

El proceso que inicia la comunicación se etiqueta como **cliente** y el que es contactado para comenzar la conexión como **servidor**.

La interfaz entre el proceso y la red

Un proceso envía y recibe mensajes de la red a través de una interfaz software llamada **socket**.

Un socket es la interfaz entre la capa de aplicación y la de transporte en un host. También se puede llamar **Application Programming Interface (API)** entre la aplicación y la red, ya que el socket es la interfaz de programación con la que las aplicaciones de red se construyen. El desarrollador tiene control sobre todo en el lado de la capa de aplicación del socket pero tiene muy poco control en el lado de la capa de transporte del socket. El único control que tiene aquí es:

- La elección del protocolo de transporte
- Quizás la habilidad de fijar alguno parámetros de la capa de transporte como el tamaño del buffer y el tamaño máximo de los segmentos

Direccionando procesos

Para que un proceso ejecutándose en un host que envía paquetes a otro proceso ejecutándose en otro host, el receptor necesita tener una dirección. Para identificar al proceso receptor, hay que identificar dos partes de información:

- La dirección del host
- Un identificador que especifique el proceso receptor dentro del host receptor

En la internet los hosts se identifican por su **dirección IP**, un número de 32 bits que identifica unívocamente un host.

El proceso emisor también tiene que identificar al proceso receptor. Esta información es necesaria porque en general un host puede estar ejecutando varias aplicaciones de red. Los **números de puerto** sirven para esto. Las aplicaciones más populares tienen asignados números de puerto específicos.

1.5.3 Servicios de transporte disponibles para las aplicaciones

Muchas redes proporcionan más de un protocolo de transporte. Cuando desarrollas una aplicación, debes elegir uno de ellos. Estos pueden clasificarse de forma simple atendiendo cuatro aspectos:

1. **Transferencia de datos garantizada:** en muchas aplicaciones la pérdida de datos puede tener consecuencias desastrosas. Para dar soporte a estas aplicaciones, algo debe hacerse para garantizar que los datos enviados llegan al destino completamente y correctamente.

   Cuando un protocolo de transporte proporciona este servicio, el emisor puede pasar sus datos al socket y sabrá con completa seguridad que los datos llegarán al receptor.

   Cuando un protocolo de la capa de transporte no proporciona transferencia de datos garantizada, algunos datos enviados por el emisor pueden no llegar nunca al receptor, esto puede ser aceptable en aplicaciones con tolerancia a pérdidas.

2. **Throughput:** es la tasa a la que el emisor puede enviar bits al receptor.

   Como otras sesiones compartirán el ancho de banda a lo largo del camino de red y estas sesiones van y vienen, el throughput disponible fluctúa con el tiempo. En ocasiones es útil garantizar un throughput disponible a una tasa específica. La aplicación podría pedir un throughput de r bits/s, y el protocolo de la capa de transporte tendría que asegurar que el throughput disponible siempre es de, al menos, r bits/s.

   Las aplicaciones que tienen requirimientos mínimos de throughput se dice que son aplicaciones sensibles al ancho de banda. Por otro lado, las aplicaciones elásticas pueden hacer uso de más o menos throughput, dependiendo de la disponibilidad.

3. **Tiempo:** puede ser interesante proporcionar garantías de tiempo de entrega.

4. **Seguridad:** un protocolo de transporte puede proporcionar a la aplicación con uno o más servicios de seguridad. Estos servicios proporcionarían confidencialidad entre los dos procesos, incluso si los datos son de alguna forma observados entre el emisor y el receptor. También puede proporcionar otros servicios como integridad de los datos o autenticación extremo a extremo.

1.5.4 Servicios de transporte proporcionados en internet

Internet pone dos protocolos de transporte a disposición de las aplicaciones, UDP y TCP. Cuando creas una aplicación para internet, una de las primeras decisiones que debes hacer es si usarás UDP o TCP.

Servicios de TCP

- **Servicio orientado a conexión:** TCP hace que el cliente y el servidor intercambien información de control de transporte entre ellos antes de que pueda comenzar el flujo de mensajes a nivel de aplicación. Esto se denomina **procedimiento de handshaking (saludo)**, e informa a cliente y servidor de que deben prepararse para un intercambio de paquetes. Después del saludo, se establece una **conexión TCP** entre los sockets de los dos procesos. Es una conexión full-dúplex en la que dos procesos pueden enviar mensajes uno al otro sobre la conexión al mismo tiempo. Cuando la aplicación termina de enviar mensajes, debe cerrar la conexión.

- **Servicio de entrega de datos garantizada:** los procesos en comunicación pueden confiar en TCP para entregar todos los datos enviados win errores y en orden.

- **Mecanismo de control de la congestión:** es un servicio para el bienestar general de internet, más que para el beneficio directo de los procesos en comunicación. Avisa al emisor cuando la red está congestionada entre él y el receptor e intenta limitar cada conexión TCP para un reparto justo de ancho de banda en la red.

Servicios de UDP

UDP es un protocolo ligero y sin lujos, que proporciona los servicios mínimos. Es **no orientado a conexión**, así que no hay handshaking antes de que los dos procesos comiencen la comunicación. La entrega de datos **no es garantizada**: los mensajes pueden llegar desordenados o no llegar. UDP tampoco incluye un mecanismo de control de la congestion: el emisor puede pasar datos a la capa de transporte a la tasa que desee.

Servicios no proporcionados por los protocolos de transporte de internet

No ofrecen garantías de tiempo ni de throughput. Internet ha estado manteniendo aplicaciones sensibles al tiempo durante muchos años. Estas aplicaciones normalmente funcionan bastante bien porque han sido diseñadas para hacer frente a esta falta de garantías.

La internet de hoy en día puede proporcionar un servicio aceptable a las aplicaciones sensibles al tiempo, pero no puede dar garantías de tiempo ni de throughput.

1.5.5 Protocolos de la capa de aplicación

Definen cómo los procesos de una aplicación, ejecutándose en distintos hosts, intercambian mensajes. Define:

- Los tipos de mensajes intercambiados
- La sintaxis de estos
- La semántica de los campos
- Reglas para determinar cuándo y cómo un proceso envía y responde mensajes

Es importante distinguir entre aplicaciones de red y protocolos de la capa de aplicación. Un protocolo solo es una parte de la aplicación de red.

2 Web HTTP

2.1 La Web y HTTP

2.1.1 Visión de conjunto de HTTP

El **Hypertext Transfer Protocol (HTTP)**, el protocolo de la capa de aplicación de la Web, está en el corazón de la Web. Se implementa en dos programas: un programa cliente y un programa servidor.

Los programas cliente y servidor, ejecutándose en diferentes hosts, hablan entre ellos intercambiando mensajes HTTP. HTTP define la estructura de estos mensajes y cómo el cliente y el servidor los intercambian.

Una **página Web** está formada por objetos.

Un **objeto** es un fichero direccionable por una única URL.

La mayoría de páginas Web consisten de un **fichero base HTML** y varios objetos referenciados. El fichero base HTML referencia los otros objetos de la página con sus URLs. Cada URL tiene dos componentes: el hostname del servidor que almacena el objeto y la ruta del objeto dentro del servidor.

Como los **navegadores Web** implementan la parte del cliente de HTTP, en el contexto de la Web, no distinguiremos entre los términos navegador y cliente.

Los **servidores Web**, que implementan la parte del servidor de HTTP, almacenan objetos Web, cada uno direccionable por una URL. Los servidores Web populares incluyen Apache y Microsoft Internet Information Server.

HTTP define cómo los clientes Web piden páginas Web a los servidores Web y cómo los servidores transfieren las páginas a los clientes.

Cuando un usuario pide una página Web, el navegador envía mensajes **HTTP requests** para pedir los objetos de la página al servidor. El servidor recibe las peticiones y responde con mensajes **HTTP response** que contienen cada objeto.

HTTP usa TCP como protocolo subyacente de transporte. El cliente HTTP inicia una conexión TCP con el servidor. Una vez que la conexión queda establecida, los procesos navegador y servidor acceden a esta a través de las interfaces de su socket. En el lado del cliente la interfaz del socket es la puerta entre el proceso cliente y la conexión TCP; en el lado del servidor es la puerta entre el proceso servidor y la conexión TCP.

El cliente envía mensajes HTTP request por la interfaz de su socket y recibe mensajes HTTP response de esta misma interfaz. En el servidor pasa lo propio, pero al contrario.

Es importante notar que el servidor envía los archivos pedidos a los clientes sin almacenar ningún tipo de información de estado sobre el cliente. Como un servidor HTTP no mantiene información sobre los clientes, se dice que HTTP es un **protocolo sin estado (stateless)**.

La Web usa, como venimos explicando, la arquitectura de aplicación cliente-servidor. Siempre debe haber un servidor Web encendido, con una dirección IP fija, y este procesa peticiones de, potencialmente, millones de navegadores distintos.

2.1.2 Conexiones persistentes y no-persistentes

Cuando esta interacción cliente-servidor está teniendo lugar sobre TCP, el desarrollador de la aplicación necesita hacer una decisión importante, ¿debería cada par petición/respuesta ser enviado en una conexión TCP distinta o deberían todos los pares de este tipo ser enviados sobre una misma conexión TCP? El primer caso se denomina **conexión no-persistente** y el segundo **conexión persistente**.

HTTP con conexión no-persistente

El funcionamiento, grosso modo, sería:

1. El cliente inicia conexión TCP con el servidor

2. El cliente envía un mensaje HTTP request al servidor por su socket
3. El servidor recibe la petición por su socket, lo procesa, busca el objeto pedido, lo encapsula en un mensaje HTTP response y envía este mensaje de respuesta al cliente por su socket
4. El servidor le dice a TCP que cierre la conexión TCP
5. El cliente recibe el mensaje de respuesta. Ahora es cuando termina la conexión TCP. El cliente procesa el mensaje de respuesta y busca todas las referencias a objetos que tenga
6. Para cada objeto referenciado, se repite el proceso

Cuando el navegador recibe la página Web, muestra la página al usuario. Dos navegadores distintos pueden interpretar una misma página de formas diferentes, HTTP no tiene nada que ver con cómo una página Web es interpretada por el cliente.

Como vemos, en el modelo no-persistente, cada conexión TCP transporta exactamente un mensaje de petición y uno de respuesta.

Para medir cuánto tiempo tarda el cliente en recibir completamente la página que desea usamos el concepto de **Round Trip Time (RTT)**, el tiempo que tarda un paquete pequeño en viajar desde el cliente al servidor y volver. El RTT incluye los retardos de propagación, encolamiento, paso por routers y switches intermedios y de procesamiento.

Pensemos, entonces, en qué sucede al clickar en un hiperenlace: el navegador inicia una conexión TCP con el navegador, esto requiere de un establecimiento de conexión (3-way handshake).

Las dos primeras partes del establecimiento de la conexión consumen un RTT.

Tras esto, el cliente envía el mensaje HTTP request junto con la tercera parte del establecimiento de la conexión. Cuando esto llega al servidor, este responde con la página pedida. Este proceso consume otro RTT.

Es decir, para enviar un objeto, se consumen 2 RTTs.

Si queremos obtener una página que contiene $N-1$ objetos, entonces, se requerirán

$$2 \cdot N \ RTTs$$

dos RTTs por cada objeto pedido. Hay que tener en cuenta que la página es también un objeto.

HTTP con conexión persistente

La opción no-persistente presenta varias desventajas:

- Una nueva conexión debe establecerse y mantenerse para cada objeto pedido. En cada conexión los buffers TCP deben asignarse, así como las variables de TCP, tanto en cliente como en servidor. Esto puede suponer una carga importante para el servidor, que puede procesar respuestas de cientos de clientes distintos simultáneamente
- Cada objeto sufre un retardo de 2 RTTs

Con las conexiones persistentes, el servidor deja la conexión TCP abierta tras enviar la respuesta. Posteriores peticiones y respuestas entre los mismos cliente y servidor pueden ser enviados por la misma conexión.

Estas peticiones podrían incluso hacerse en **pipeline**, sin esperar cada respuesta.

Normalmente, el servidor cerrará la conexión cuando transcurre cierto tiempo sin recibir peticiones. Cuando el servidor recibe las peticiones en pipeline, envía también las respuestas en pipeline.

El retardo sin pipeline será 1RTT del establecimiento de la conexión, 1RTT por cada objeto requerido. Si queremos una página con $N-1$ objetos, será

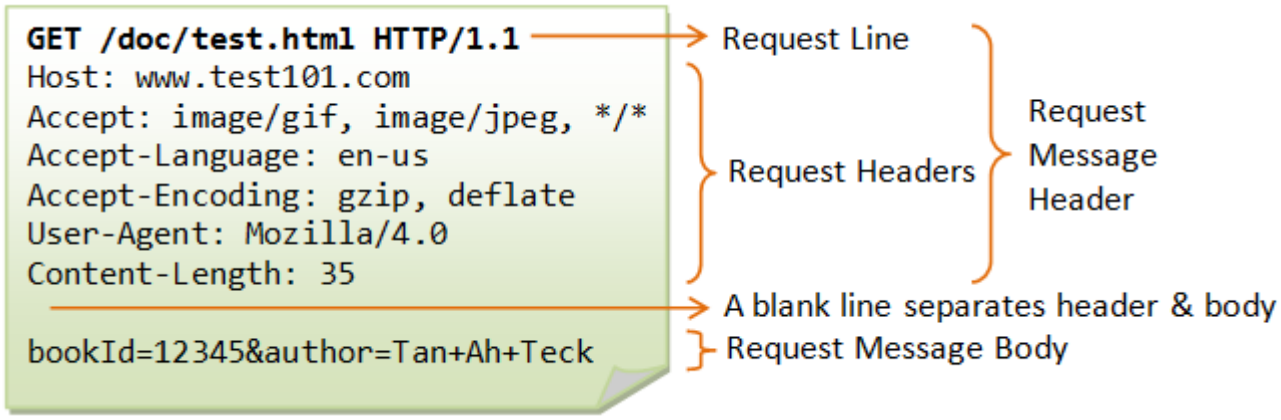
$$N+1 \ RTTs$$

El retardo con pipeline depende del tamaño del pipeline (cuántos mensajes pueden haber en vuelo) y de posibles retardos de encolado. Si suponemos el pipeline admite M mensajes (y suponemos que se envían simultáneamente), entonces consumiremos 1RTT para el establecimiento de la conexión y 1RTT (siendo realistas sería más) por cada M pares petición/respuesta. Si queremos una página con $N-1$ objetos, el retardo será

$$\left\lceil \frac{N}{M} \right\rceil + 1 \; RTTs$$

2.1.3 Formato de los mensajes HTTP

Mensaje HTTP Request



El mensaje se escribe en texto ASCII ordinario y consiste de una cantidad de líneas, cada una seguida de un retorno de carro y otro de line-feed (\r\n). La última línea lleva un \r\n adicional, para indicar que termina la cabecera.

La primera línea se denomina **request line** y las siguientes **header lines**.

La request line tiene tres campos:

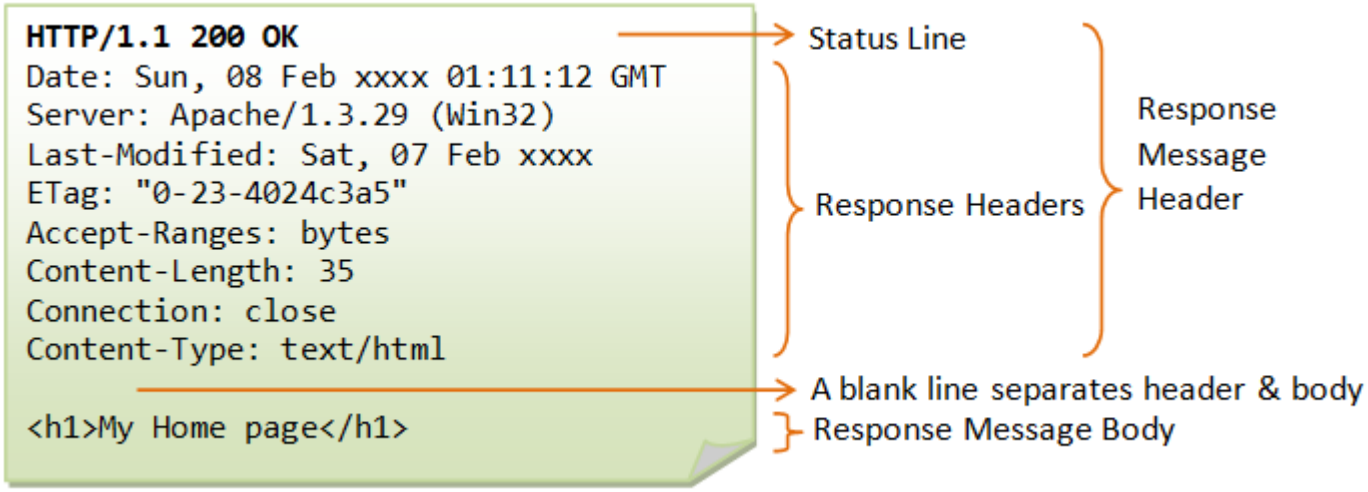
- **Método:** indica la acción requerida por la petición. Puede ser GET, POST, HEAD, PUT y DELETE
  - GET: el método más usado. Se usa para pedir un objeto, con el objeto requerido identificado en el campo URL.
  - POST: usado cuando el usuario rellena un formulario. Se usa también para pedir una página, pero los contenidos de esta dependen de lo que el usuario ha introducido en los campos del formulario.
  - HEAD: parecido a GET. Cuando un servidor recibe una petición con el método HEAD, responde con un mensaje HTTP pero no envía el objeto requerido.
  - PUT: suele usarse junto con herramientas de publicación Web. Permite al usuario subir un objeto a un directorio específico de un servidor Web.
  - DELETE: permite a un usuario eliminar objetos de un servidor Web.
- **URL:** del objeto requerido
- **Versión HTTP**

Tras la request line, vemos las header lines. Las más comunes son:

- User-Agent: informa del tipo de navegador que emplea el cliente
- Referer: URL del docuento desde el que se accedió al actual
- If-Modified-Since: solo se quiere el documento si se ha modificado tras la fecha indicada
- Accept: muestra los formatos que acepta el cliente
- Accept-Language: lenguajes aceptados por el navegador
- Host: servidor al que el cliente envía la petición
- Date: fecha actual
- Connection: se emplea para mantener la conexión
- Cookie: devuelve una cookie al servidor

Después de las header lines, nos encontramos con el **cuerpo**. Este está vacío en el método GET, pero se usa con el método POST.

Mensaje HTTP Response



También tiene tres partes: una **línea de estado**, una cantidad de **header lines** y el **cuerpo**.
El cuerpo contiene la parte importante del mennsaje, lleva el objeto requerido.
La línea de estado tiene tres campos: la versión HTTP, un código de estado y el mensaje de estado correspondiente.
Los header lines más comunes son:

- Location: nueva localización del documento solicitado
- Server: nombre y versión del software servidor
- MIME-version: versión del protocolo MIME usada por el servidor
- Content-Length: longitud en bytes del cuerpo de la respuesta
- Content-Type: tipo MIME que identifica el tipo de dato de la respuesta

- Last-Modified: fecha y hora en la que se modificó por última vez
- Set-Cookie: el servidor solicita al cliente que almacene una cookie

Y los códigos, junto con sus mensajes de estado, más frecuentes son:

- 200 OK: la petición ha sido atendida con éxito y la respuesta se envía en la respuesta
- 301 Moved Permanently: el objeto requerido ha sido movido de forma permamente. La nueva URL se especifica en 'Location:'. El cliente pedirá esta nueva URL
- 400 Bad Request: es un código de error genérico que indica que la petición no pudo ser entendida por el servidor
- 404 Not Found: el documento pedido no existe en este servidor
- 505 HTTP Version Not Supported: el protocolo HTTP de la petición no es soportado por el servidor.

2.1.4 Interacción usuario-servidor: Cookies

Es común que los sitios Web encuentren útil identificar usuarios, ya sea porque el servidor quiera restringir el acceso a algunos usuarios o porque quiera ofrecer contenido en función de la identidad del usuario. Para estos propósitos, HTTP utiliza **cookies**, que permiten a las páginas seguirle la pista a los usuarios. La mayoría de Web comerciales usan cookies hoy en día.

La tecnología de las cookies tiene cuatro componentes:

1. Una header line cookie en el mensaje HTTP response.
2. Un header line cookie en el mensaje HTTP request.
3. Un archivo cookie en el equipo del cliente, manejado por el navegador.
4. Una base de datos back-end en el servidor.

Las cookies se pueden usar para identificar a un usuario. La primera vez que un usuario visita una página, este puede proporcionar una identificación de usuario. Durante las siguientes sesiones, el navegador pasa una cabecera cookie al servidor, identificando al usuario en el servidor. Por tanto, las cookies pueden usarse para crear una capa de sesión de usuario sobre HTTP sin estado.

Son polémicas, ya que pueden ser consideradas como invasión de la privacidad. Usando una combinación de cookies y de información proporcionada por el usuario un sitio Web puede aprender muchas cosas sobre el usuario y potencialmente vender esta información a terceros.

Formato de las Cookies HTTP

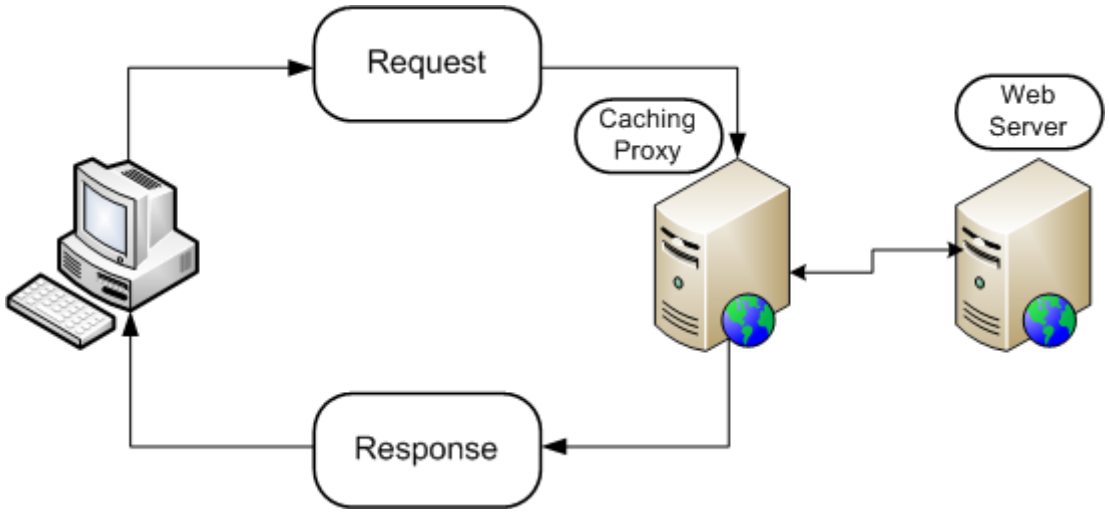
Las cookies son una sola línea de valores separados por ';', los valores más importantes son:

- nombre=valor : asocia a una propiedad un valor específico
- expires=fecha : fecha de caducidad de la cookie
- domain=ámbito : especifica los hosts permitidos para recibir la cookie
- path=camino : indica una ruta URL que debe existir en la URL solicitada para enviar el header
- secure : solo se envía cuando se emplea el protocolo HTTPS

2.1.5 Caching Web

Una **caché Web (Proxy server)** es una entidad de red que satisface peticiones HTTP en lugar del servidor Web original. La caché Web tiene su propio disco de almacenamiento y guarda copias de objetos recientementes solicitados. Un navegador Web puede ser configurado para que todas las peticiones HTTP del usuario sean primero dirigidas a la caché Web. El proceso es así:

1. El navegador establece una conexión TCP con la caché Web y envía una petición HTTP para el objeto a la caché Web
2. La caché Web comprueba si tiene una copia del objeto almacenada localmente. Si lo tiene, devuelve el objeto en una respuesta HTTP al navegador cliente
3. Si la caché Web no tiene el objeto, abre una conexión TCP con el servidor original. La caché Web entonces envía una petición HTTP para el objeto en el servidor. Tras recibir esta petición, el servidor original envía el objeto en una respuesta HTTP a la caché Web
4. Cuando la caché Web recibe el objeto, guarda una copia en su almacenamiento local y envía una copia, en una respuesta HTTP, al navegador cliente



Nótese que una caché es tanto cliente como servidor al mismo tiempo.

Normalmente una caché Web es comprada e instalada por un ISP.
El caching Web ha sido bastante usado en Internet por dos razones:

- Una caché Web reduce sustancialmente el tiempo de respuesta para una petición, sobre todo si el cuello de botella de ancho de banda entre el cliente y el servidor original es mucho menor que el cuello de botella entre el cliente y la caché
- Las cachés Web pueden reducir significativamente el tráfico en un enlace de acceso institucional a Internet. Reduciendo el tráfico, la institución no tiene que aumentar el ancho de banda tan rápido, reduciendo costes.

Mediante el uso de **Content Distribution Networks (CDNs)**, red superpuesta de computadoras que contienen copias de datos, colocados en varios puntos de una red con el fin de maximizar el ancho de banda para el acceso a los datos de clientes por la red, las cachés Web han ido incrementalmente siendo más importantes en Internet. Una compañía de CDN instala muchas cachés distribuidas geográficamente a lo largo de internet. Hay CDNs compartidas y dedicadas.

2.1.6 El GET condicional

Aunque el caching puede reducir los tiempos de respuesta percibidos por el usuario, introduce un nuevo problema: la copia del objeto en la caché puede estar desfasada, pues puede haber sido en el servidor desde que fue cacheado.

HTTP tiene un mecanismo que permite a la caché verificar si los objetos están actualizados. Este mecanismo es el GET condicional. Un mensaje HTTP request es un GET condicional si:

1. Usa el método GET
2. Include una header line 'If-Modified-Since:'

Si el objeto ha sido modificado después de la fecha indicada en este último campo, entonces el servidor lo enviará a la Caché. Si no, enviará un mensaje con código de estado 304 Not Modified. En cualquiera de los casos la caché se asegura de tener el objeto actualizado.

2.2 Common Gateway Interface

Con HTML se pueden crear formularios cuyos datos se tienen que procesar en el servidor. Los métodos GET y POST permiten enviar datos del cliente al servidor, pero los servidores web no pueden procesar cualquier tipo de consulta.

El interfaz CGI define la comunicación estándar entre un servidor web y una aplicación externa.
El proceso es:

1. El usuario del navegador pulsa en el botón SUBMIT
2. El navegador codifica los datos antes de enviarlos al servidor
3. El navegador emplea GET o POST para enviar los datos al servidor
4. El servidor reconoce que están solicitando la ejecución de un programa gateway, e invoca dicho programa en un nuevo proceso
5. El gateway recibe los datos enviados desde el navegador a través de variables de entorno, parámetros de la línea de comando o la entrada estándar
6. Gateway procesa la solicitud y genera la respuesta en forma de página web a través de la salida estándar
7. El servidor web envía al cliente los resultados que emite el programa gateway. Normalmente completará las cabeceras de respuesta

8. El cliente visualiza la página que contiene los resultados

Variables de entorno relativas al servidor que puede consultar CGI:

- DOCUMENT_ROOT: directorio que conteine las páginas HTML del servidor
- SERVER_NAME: nombre del servidor que gestione el script CGI
- SERVER_PORT: puerto del servidor

Variables de entorno relativas a la conexión cliente-servidor:

- HTTP_ACCEPT: tipos MIME soportados por el cliente
- HTTP_COOKIE: cookies asociadas por el cliente al recurso solicitado
- REMOTE_ADDR: dirección IP del equipo desde el que se efectúa la petición

Variables relativas a la petición:

- CONTENT_LENGTH: tamaño, en bytes, del contenido de la información
- CONTENT_TYPE: tipo MIME de la petición
- QUERY_STRING: cadena de caracteres, en formato URL, que aparece en la petición GET

El método GET

La entrada de datos se hace mediante la variable QUERY_STRING.

La ventaja es que el usuario puede conservar consultas en forma de una URL.

El inconveniente es que la cadena de caracteres que define la codificación URL puede ser larga y sobrepasar el tamaño máximo tolerado por el servidor.

El método POST

La entrada de datos se hace mediante la entrada estándar.

La ventaja es que el método no se ve restringido por el tamaño de los datos producidos por un formulario.

El inconveniente es que el usuario no puede conservar las consultas realizadas.

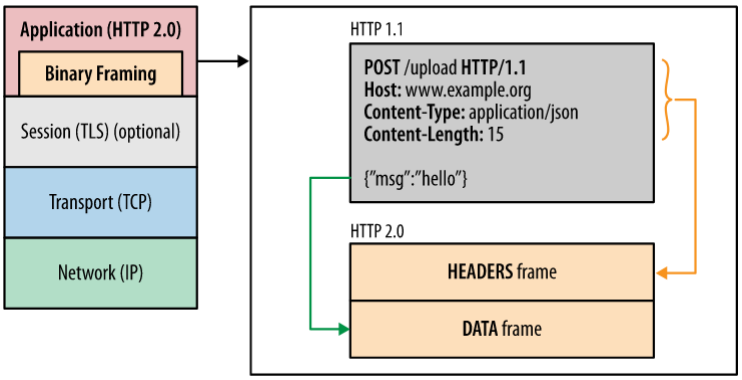
2.3 HTTP/2

Tiene como objetivos la reducción de la latencia mediante la multiplexación completa de solicitudes y respuesta, minimizar la sobrecarga del protocolo mediante la compresión y compatibilizar con la priorización de solicitudes y push del servidor.

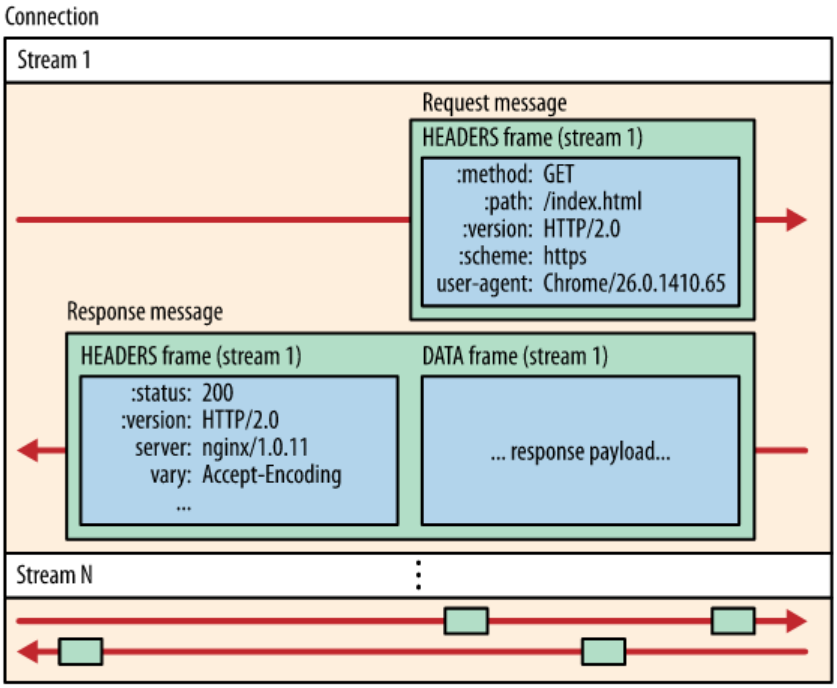
Mantiene la semántica de HTTP.

Capa de enmarcado binario

Determina la forma en que los mensajes de HTTP se encapsulan y se transfieren entre el cliente y el servidor.



A diferencia del protocolo HTTP/1.x de texto plano delimitado por línea nueva, toda la comunicación de HTTP/2 se divide en mensajes y marcos más pequeños, cada uno de los cuales está codificado en formato binario.

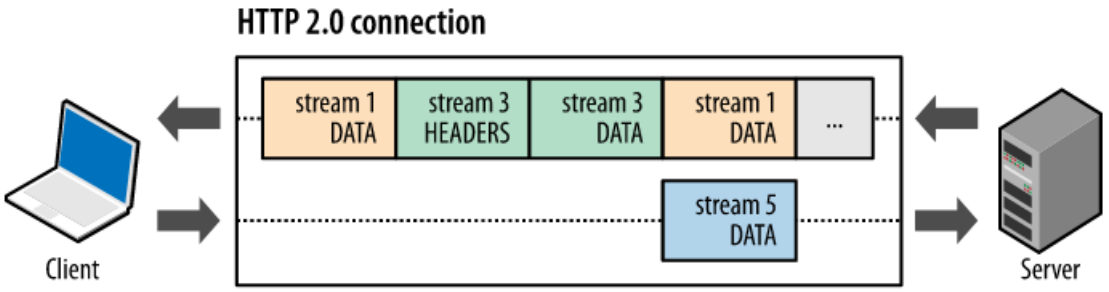


HTTP/2 desglosa la comunicación del protocolo HTTP en un intercambio de tramas con codificación binaria, que luego se asignan a los mensajes que pertenecen a una transmisión específica, todo está multiplexado dentro de una única conexión de TCP. Esta es la base que habilita al resto de las funciones y las optimizaciones de rendimiento ofrecidas por el protocolo HTTP/2.

Multiplexación de peticiones y respuestas

Con HTTP/1.x, si el cliente desea realizar múltiples solicitudes paralelas para mejorar el rendimiento, se deben usar múltiples conexiones de TCP. Este comportamiento es una consecuencia directa del modelo de entrega de HTTP/1.x, que asegura que solo una respuesta pueda entregarse por vez (cola de respuestas) por conexión. Peor aún, esto también desencadena el bloqueo de cabeza de línea y el uso ineficiente de la conexión de TCP subyacente.

La nueva capa de entramado binario de HTTP/2 elimina estas limitaciones y habilita una multiplexación total de solicitudes y respuestas, al permitir que el cliente y el servidor desglosen un mensaje de HTTP en tramas diferentes, intercalarlas y luego reensamblarlas en el otro extremo.



La capacidad de desglosar un mensaje de HTTP en marcos independientes, intercalarlos y luego reensamblarlos en el otro extremo es la mejora más importante de HTTP/2. De hecho, produce un efecto dominó con numerosos beneficios de rendimiento en toda la pila de tecnologías web, lo que nos permite:

- Intercalar múltiples solicitudes en paralelo sin bloquear ninguna.
- Intercalar múltiples respuestas en paralelo sin bloquear ninguna.
- Usar una única conexión para entregar múltiples solicitudes y respuestas en paralelo.
- Eliminar métodos alternativos de HTTP/1.x innecesarios, como archivos concatenados, image sprites y fragmentos de dominio.
- Proporcionar tiempos de carga de páginas inferiores mediante la eliminación de la latencia innecesaria y la mejora de la utilización de la capacidad de red disponible.

Priorización de transmisiones

Una vez que un mensaje de HTTP puede dividirse en muchos marcos individuales y permitimos que los marcos de múltiples transmisiones estén multiplexados, el orden en el cual el cliente y el servidor intercalan y entregan los marcos se convierte en una consideración de rendimiento crítica. Para facilitar esto, el estándar HTTP/2 permite que cada transmisión tenga un peso y una dependencia asociados:

- A cada transmisión se le puede asignar un peso entero de entre 1 y 256.
- Cada transmisión puede recibir una dependencia explícita de otra transmisión.

La combinación de dependencias y pesos de transmisión permite que el cliente construya y comunique un "árbol de priorización" que exprese cómo preferiría recibir las respuestas. A su vez, el servidor puede usar esta información para priorizar el procesamiento de transmisión al controlar la asignación de CPU, memoria y otros recursos y, una vez que los datos de la respuesta están disponibles, la asignación de ancho de banda para asegurar la entrega óptima de respuestas de prioridad alta al cliente.

Una conexión por origen

Con el nuevo mecanismo de enmarcado binario en funcionamiento, HTTP/2 no necesita más de múltiples conexiones de TCP para multiplexar transmisiones en paralelo: cada transmisión se divide en muchos marcos, que pueden intercalarse y priorizarse. Como resultado, todas las conexiones de HTTP/2 son persistentes y solo se requiere una conexión por origen, lo cual ofrece numerosos beneficios de rendimiento.

Control de flujo

HTTP/2 proporciona un conjunto de elementos básicos simples que permiten que el cliente y el servidor implementen su propio control de flujo en el nivel de transmisión y de conexión:

- El control de flujo es direccional. Cada receptor puede optar por configurar cualquier tamaño de ventana que desee para cada transmisión y para toda la conexión.
- El control de flujo se basa en el crédito. Cada receptor indica su conexión inicial y la ventana de control de flujo de transmisión (en bytes), que se reduce cuando el emisor emite un marco DATA y se incrementa mediante un marco WINDOW_UPDATE enviado por el receptor.
- El control de flujo no se puede inhabilitar. Cuando se establece la conexión de HTTP/2, el cliente y el servidor intercambian marcos SETTINGS, que configuran los tamaños de la ventana de control de flujo en ambas direcciones. El valor predeterminado de la ventana de control de flujo se configura en 65,535 bytes, pero el receptor puede configurar un mayor tamaño de ventana máximo (2^31-1 bytes como tope) y mantenerlo mediante el envío de un marco WINDOW_UPDATE cada vez que se reciben datos.
- El control de flujo es de salto a salto, no de extremo a extremo. Es decir, un intermediario puede usarlo para controlar el uso de recursos e implementar mecanismos de asignación de recursos basados en criterios y heurística propios.

HTTP/2 no especifica ningún algoritmo en especial para implementar el control de flujo. En cambio, proporciona elementos básicos simples y concede la implementación al cliente y el servidor, que lo pueden usar para implementar estrategias personalizadas a fin de regular el uso y la asignación de recursos, además de implementar nuevas capacidades de entrega que ayuden a mejorar tanto el rendimiento real como el percibido de nuestras aplicaciones web.

Servidor Push

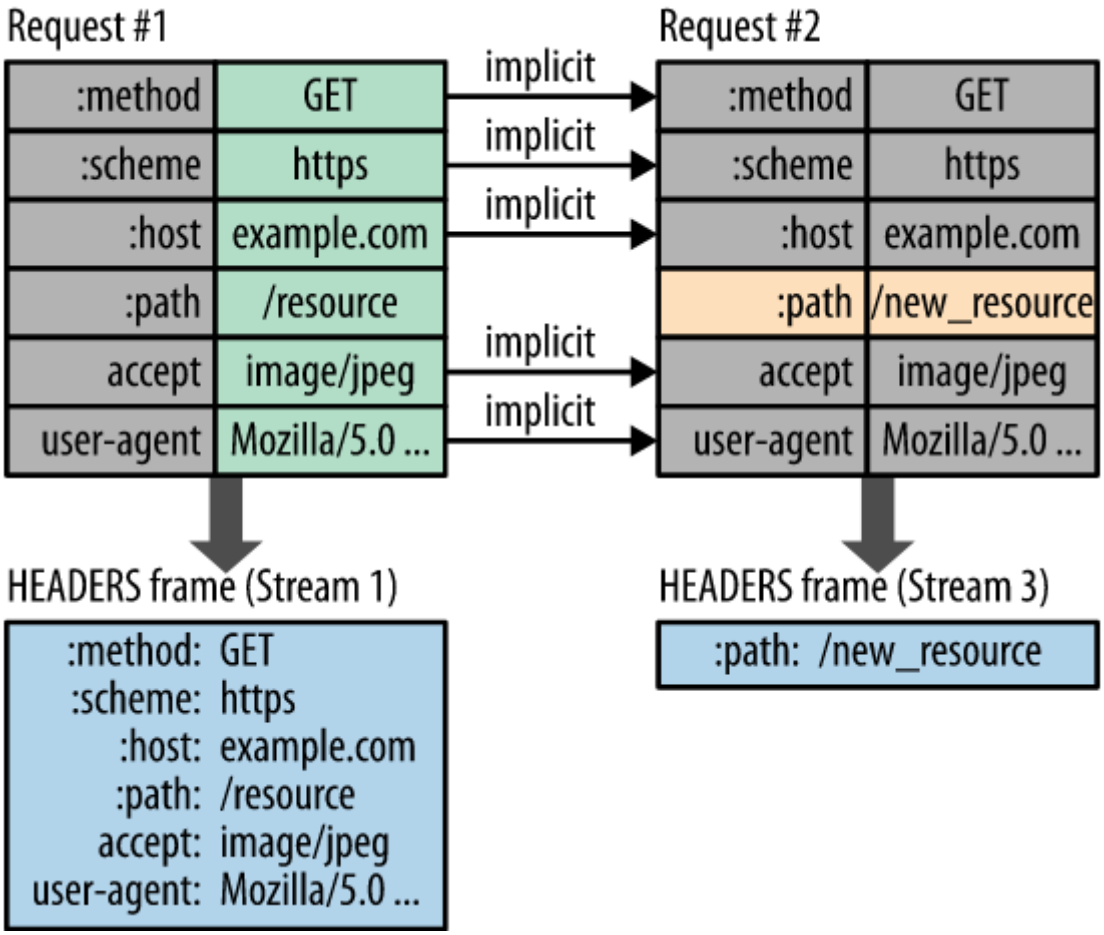
Otra nueva función poderosa de HTTP/2 es la capacidad del servidor de enviar respuestas múltiples para una única solicitud del cliente. Es decir, además de la respuesta a la solicitud original, el servidor puede insertar recursos adicionales en el cliente sin necesidad de que este los solicite de manera explícita.

Compresión de cabeceras

Cada transferencia de HTTP lleva un conjunto de encabezados que describen al recurso transferido y sus propiedades. En HTTP/1.x, estos metadatos siempre se envían como texto sin formato y agregan entre 500 y 800 bytes de sobrecarga por transferencia o a veces incluso kilobytes, si se usan cookies de HTTP. Para reducir esta sobrecarga y mejorar el rendimiento, HTTP/2 comprime los metadatos del encabezado de solicitud y respuesta mediante el formato de compresión HPACK, que usa dos técnicas simples pero poderosas:

Permite que los campos del encabezado transmitido se codifiquen a través de un código Huffman estático, que reduce su tamaño de transferencia individual. Requiere que tanto el cliente como el servidor mantengan y actualicen una lista indexada de campos de encabezado previamente vistos (en otras palabras, establece un contexto de compresión compartido), que luego se usa como referencia para codificar de manera eficaz los valores transmitidos con anterioridad.

La codificación Huffman permite que los valores individuales se compriman cuando se transfieren y la lista indexada de los valores transferidos anteriormente nos permite codificar valores duplicados transfiriendo valores indexados que pueden usarse para buscar y reconstruir todas las claves y los valores del encabezado de manera eficaz.



Como una optimización adicional, el contexto de la compresión de HPACK consiste en una tabla estática y una tabla dinámica: la tabla estática está definida en la especificación y proporciona una lista de campos de encabezados de HTTP comunes que todas las conexiones tienen más posibilidades de usar (p. ej., nombres de encabezado válidos); la tabla dinámica inicialmente está vacía y se actualiza en base a los valores intercambiados dentro de una conexión específica. Como resultado, el tamaño de cada solicitud se reduce al usar codificación Huffman estática para los valores nunca antes vistos y una sustitución de índices por valores que ya están presentes en las tablas estáticas o dinámicas a cada lado.

3 Protocolos para el correo electrónico

3.1 Correo electrónico en Internet

El correo electrónico es un medio de comunicación asíncrono que, en comparación con el correo postal, es rápido, de fácil de distribución y barato.

Los componentes del correo electrónico son:

- **User agent:** permite a los usuarios leer, responder, reenviar, guardar y redactar mensajes
- **Servidor mail:** forman la base de la infraestructura del e-mail. Cada receptor tiene un **mailbox** localizado en uno de los servidores mail. Un mensaje típico comienza su viaje en el user agent del emisor, viaja al mail server del emisor, y va al mail server del receptor, donde se deposita en el mailbox del receptor. Si el servidor del emisor no puede entregar el mensaje al del receptor, lo almacena en una **cola de mensajes** y lo intenta reenviar más tarde.
- **Simple Mail Transfer Protocol (SMTP):** es el principal protocolo de la capa de aplicación para el correo electrónico. Usa TCP para transferir correos del servidor mail del emisor al derl receptor. Tiene dos caras:
  - Cara cliente: ejecuta el servidor mail del emisor
  - Cara servidor: ejecuta el servidor mail del receptor

  Ambas caras de SMTP se ejecutan en todo servidor mail. Cuando un servidor envía mensajes a otros servidores actúa como cliente SMTP. Cuando recibe correos de otros servidores actúa como servidor SMTP.

3.1.1 SMTP

Transfiere mensajes de servidores mail de emisores a servidores mail de receptores.

Aunque SMTP tiene bastantes buenas propiedades, como evidencia su presencia en Internet, es, sin embargo, una tecnología antigua que tiene algunas características arcaicas.

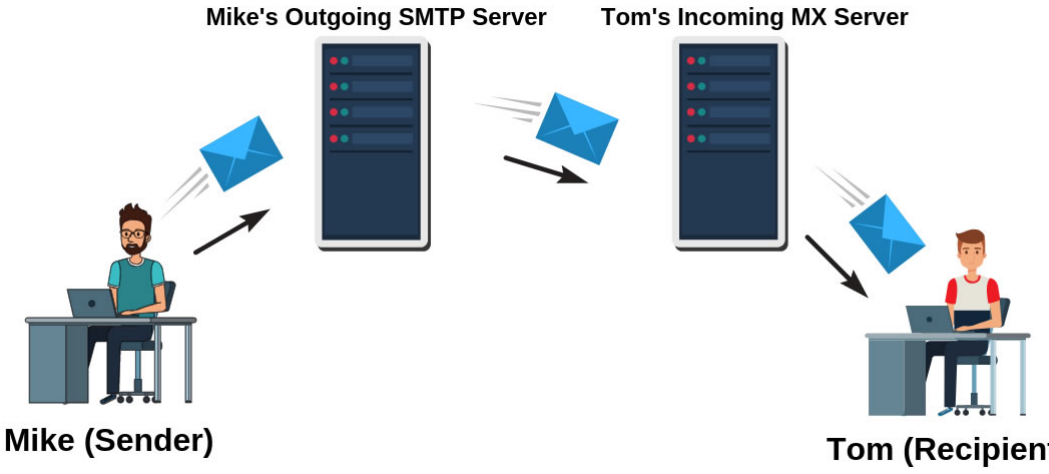
El cuerpo está restringido a usar caracteres 7-bit ASCII. Esta restricción tenía sentido en los 90 cuando la capacidad de transmisión era escasa y nadie enviaba correos con grandes archivos adjuntos. Pero, ahora, esta restricción dificulta mucho la implementación: obliga a codificar la información binaria a ASCII antes de ser enviada sobre SMTP, y, por supuesto, estos datos codificados, deben ser decodificados en recepción.

Para ver el funcionamiento de SMTP, supongamos que A quiere enviar un mensaje ASCII a B.

1. A inicia su user agent, le proporciona la dirección e-mail de B, escribe el mensaje y le dice al user agent que lo envíe.
2. El user agent de A envía el mensaje a su servidor mail, donde se almacena en una cola de mensajes.
3. La parte cliente de SMTP, ejecutándose en el servidor mail de A, ve el mensaje en la cola. Abre una conexión TCP con un servidor SMTP que se ejecuta en el servidor mail de B.
4. Tras el saludo inicial STMP, el el cliente SMTP envía el mensaje de A por la conexión TCP.

5. En el servidor mail de B, la parte servidor de SMTP recibe el mensaje. El servidor mail de B lo coloca en el mailbox de B.

6. B ejecuta su user agent para leer el mensaje cuando lo vea conveniente.



SMTP normalmente no usa servidores mail intermedios para enviar correos, ni siquiera cuando los servidores emisor y receptor están muy alejados.

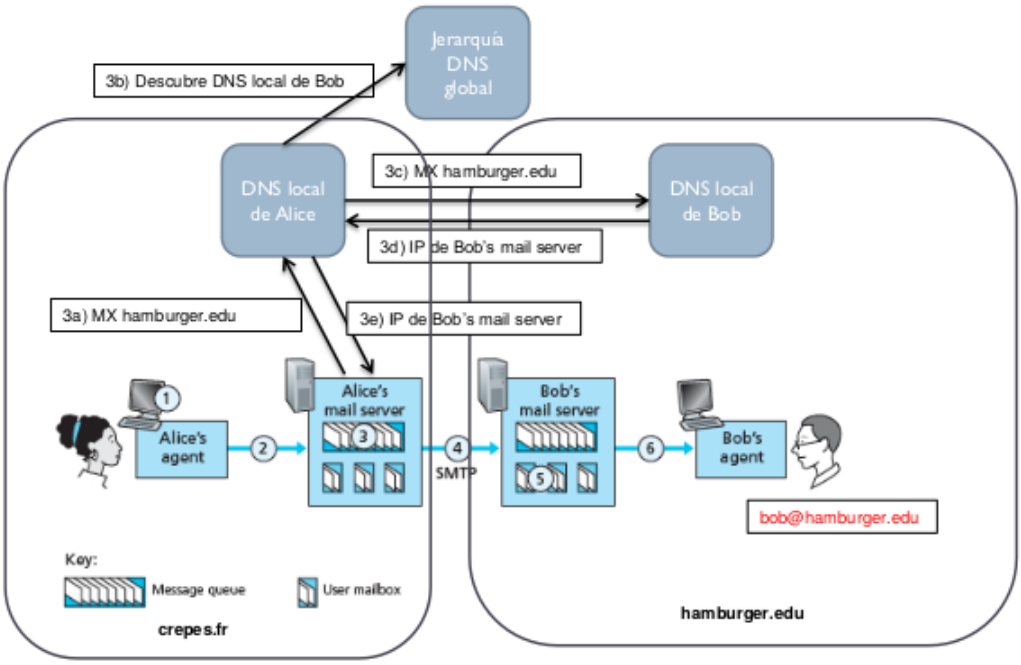
Ahora vamos a ver algunos de estos mismos pasos en más detalle:

3. El cliente SMTP establece una conexión por el puerto 25 en el servidor SMTP del receptor. Si el servidor está caído, el cliente lo intenta de nuevo más tarde.

4. Una vez esta conexión qeuda establecida, el servidor y el cliente realizan un saludo inicial a nivel de aplicación. Durante esta fase de saludo, el cliente SMTP indica la dirección email del receptor y la del emisor. Tras esta fase, la parte del cliente envía el mensaje, contando con la entrega asegurada de TCP para ello.

   Si el cliente tiene más mensajes que enviar al servidor, repite este proceso sobre la misma conexión TCP. En otro caso, cierra la conexión TCP.

- ¿Cómo el cliente SMTP sabe quién es el servidor SMTP al que debe enviar el mensaje?



Como parte del diálogo, el cliente puede utilizar cinco comandos:

- HELO
- FROM
- RCPT TO
- DATA
- QUIT

También puede indicar una línea con un solo punto, lo que indica el final del mensaje al servidor.

Para cada comando, el servidor responde con un código de respuesta y posiblemente alguna explicación.

Si el servidor mail del emisor tiene varios mensajes para enviar al mismo servidor receptor, puede enviarlos todos sobre la misma conexión TCP. Para cada mensaje, el cliente comienza el proceso con un nuevo 'MAIL FROM:', indica el el final del mensaje con un único punto y solo usa 'QUIT' cuando ha enviado todos los mensajes.

3.1.2 Comparación con HTTP

Similitudes:

- Ambos protocolos se usan para transferir ficheros de un host a otro: HTTP transfiere ficheros (objetos) desde un servidor Web a un cliente Web; SMTP transfiere ficheros (emails) desde un servidor mail a otro servidor mail.
- Ambos usan conexiones persistentes.

Diferencias:

- HTTP es un **pull protocol** (protocolo de pedir ('tirar')), alguien carga información en un servidor Web y los usuarios usan HTTP para pedir la información desde el servidor en conveniencia. Concretamente, la conexión TCP la inicia la máquina que desea recibir el fichero. Por otro lado, SMTP es un **push protocol** (protocolo de dar ('empujar')), el servidor mail emisor envía el fichero al servidor receptor. En particular, la conexión TCP la comienza la máquina que quiere enviar el fichero.
- SMTP presenta la restricción de la codificación en 7-bit ASCII, pero HTTP no.
- HTTP encapsula cada objeto en su propio mensaje de respuesta HTTP, mientras que SMTP coloca todos los objetos del mensaje en un mensaje

3.1.3 Formato de los mensajes de correo electrónico

Consiste en:

- Una cabecera que contiene información general. Esta está contenida en una serie de líneas de cabecera. Las líneas de cabecera y el cuerpo del mensaje se separan por una línea en blanco.

  - Al igual que HTTP, en cada línea aparece una cabecera, formada por "nombre: valor"

Cabecera	Significado
To:	Direcciones email de los destinatarios primarios
Cc:	Direcciones email de los destinatarios secundarios
Bcc:	Direcciones email para copias "ciegas"
From:	quién escribió el mensaje
Sender:	quién envió el mensaje
Received:	MTA por el que ha pasado el mensaje
Return-Path:	El último MTA puede indicar la trayectoria de regreso (a partir de las cabeceras Received)

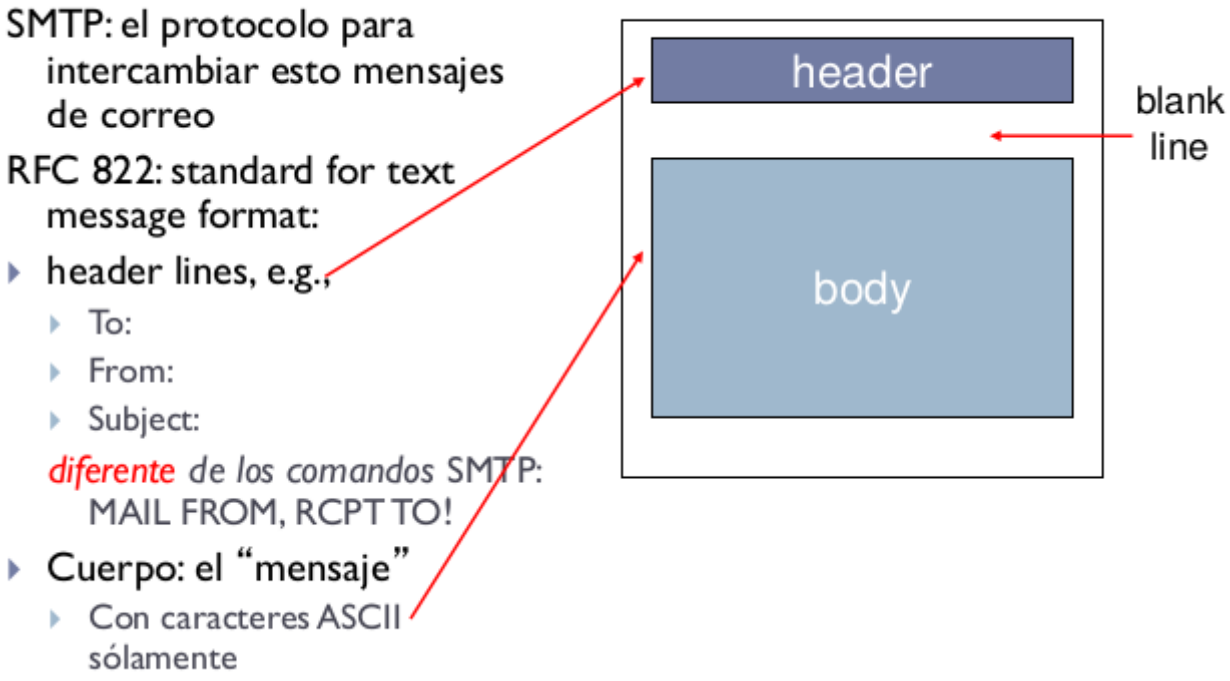
  - Cabeceras del encabezado de los mensajes SMTP

Cabecera	Significado
Date:	Fecha y hora de envío del mensaje
Reply-To:	Dirección de email a la que deben enviarse las contestaciones
Message-Id:	Número único de referencia del mensaje
In-Reply-To:	Identificador del mensaje al que este responde
Subject:	Resumen corto del mensaje
X-*loquesea*	Cabecera inventada por el usuario o el MUA Ej: X-Mailer indica el software MUA

  Como en HTTP, cada línea de cabecera contiene texto leíble, consistente en una palabra clave seguida de ':' seguida de un valor. Algunas keyword son obligatorias y otras son opcionales.

  Toda cabecera debe contener las líneas de cabecera 'From:' y 'To:'.

- Tras la cabecera, una línea en blanco y después el cuerpo del mensaje (en ASCII).



3.1.4 Protocolos de acceso al email

El acceso al email utiliza una arquitectura cliente-servidor. El usuario típico lee emails con un cliente que se ejecuta en el sistema final del usuario. Ejecutando un cliente mail en un ordenador local, los usuarios disfrutan de muchas características, como la habilidad de ver mensajes con contenido multimedia y archivos adjuntos.

Si suponemos que B (el receptor) ejecuta su user agent en su ordenador local, es natural considerar colocar un servidor mail en su ordenador local también. Con esta enfoque, el servidor mail de A (el emisor) dialogaría directamente con el ordenador de B. El problema está en que, en este caso, el ordenador de B debería quedar permanentemente encendido y conectado a internet, para poder recibir nuevos correos, pues estos pueden llegar en cualquier momento. Esto no es práctico para la mayoría de usuarios.

En lugar de esto, los usuarios suelen ejecutar un user agent en ordenador local pero acceden a su mailbox almacenado en un servidor mail compartido y siempre en línea. Este servidor se comparte con otros usuarios y normalmente lo mantiene el ISP del usuario.

Ahora bien, normalmente el user agent de A no dialoga directamente con el servidor mail de B, sino que usa SMTP para dar el mensaje email a su servidor mail, entonces el servidor mail de A usa SMTP (como cliente) para entregar el mensaje email al servidor mail de B.

¿Por qué un procedimiento en dos pasos?

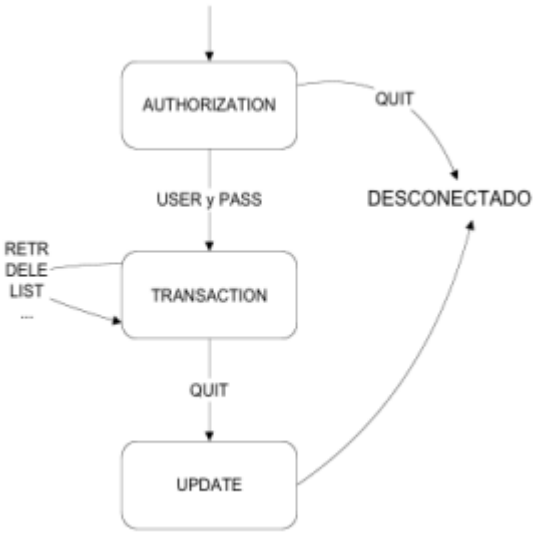
Porque sin retransmitir por el servidor mail de A, el user agent de A no tiene ninguna salida a un servidor mail destino inalcanzable. Haciendo que A primero deposite su email en su servidor mail, este puede intentar reenviar el mensaje repetidamente al servidor mail de B, hasta que este esté operativo.

Ahora bien, falta ver cómo B, ejecutando un user agent en su ordenador local, obtiene sus mensajes, que están en un servidor mail del ISP de B. Nótese que el user agent de B no puede usar SMTP para obtener los mensajes porque obtener los mensajes es una operación pull, mientras que SMTP es un protocolo push. Para este propósito se introduce un porotoclo especial de correo que transfiere mensajes del servidor mail de B a su ordenador local.

- **POP3:** es un procotolo de acceso al correo extremadamente simple, por lo que su funcionalidad es limitada.

  POP3 coienza cuando el user agent (el cliente) inicia una conexión TCP con el servidor mail (el servidor) en el puerto 110. Con la conexión TCP establecida, POP3 progresa por tres fases: authorization, transaction y update.

  - **Authorization:** el user agent envía un nombre de usaurio y contraseña para autenticar al usuario
  - **Transaction:** el user agent obtiene los mensajes. También durante esta fase el user agent puede marcar mensajes para su borrado, quitar marcas de borrado y obtener estadísticas
  - **Update:** cuando el cliente ejecuta el comando 'quit' acaba la sesión POP3. En este momento, el servidor mail elimina los mensajes que fueron marcados para borrado.



  En una transación POP3, el user agent emite comandos y el servidor responde a cada comando. Hay dos posibles respuestas:

  - **+OK:** a veces seguido de datos del servidor para el cliente. Lo usa el servidor para indicar que el comando anterior estaba bien
  - **-ERR:** usado por el servidor para indicar que hubo algún error con el comando anterior.

  La fase de autorización tiene dos comandos principales: 'user <username>' y 'pass <password>'

  Si escribes mal un comando, el servidor POP3 responderá con mensaje -ERR.

  Un user agent que usa POP3 puede ser configurado de dos modos:

  - **download-and-delete mode:** el user agent ejecuta los comandos 'list', 'retr' y 'dele'. Es decir, el user agent primero pide al servidor mail que liste el tamaño de cada uno de los mensajes almacenados. Entocnes el user agent obtiene y elimina cada mensaje del servidor.

Tras procesar el comando 'quit', el servidor POP3 entra en la fase update y elimina los mensajes del mailbox. Un problema de este modo es que B podría ser nómada y quere acceso a sus emails desde distintas máquinas. En este modo, una vez que lea un mensaje en una máquina, se eliminará del servidor y no podrá acceder a él desde otras máquinas.

- **download-and-keep mode:** el user agente deja los mensajes en el servidor después de descargarlos

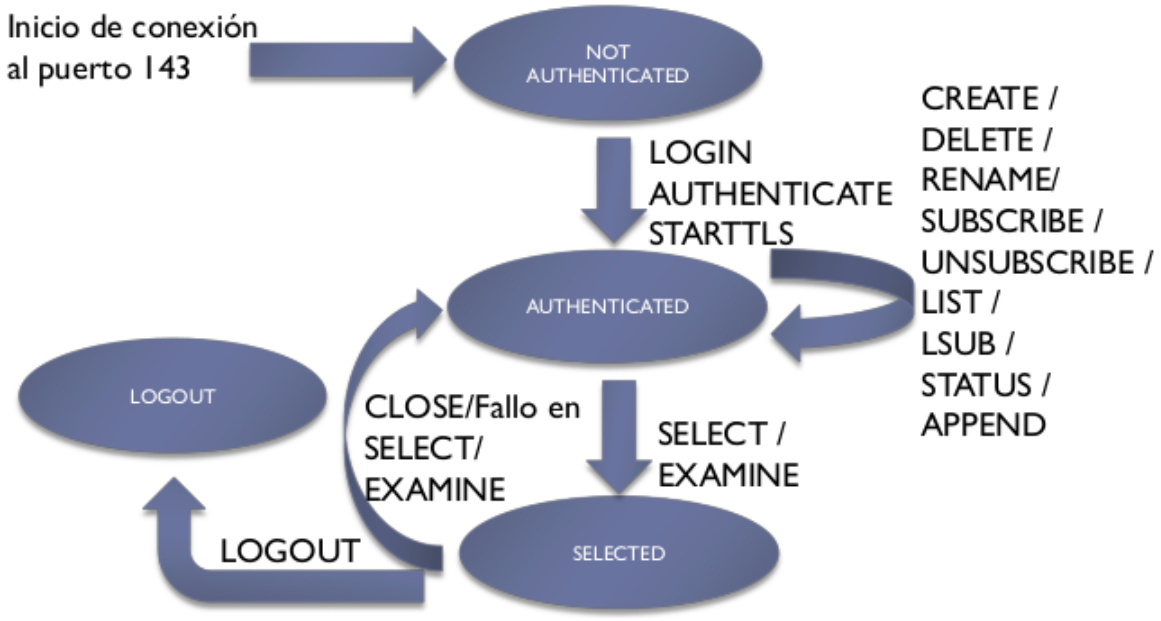
Durante una sesión POP3 entre un user agent y un servidor mail, el servidor POP3 mantiene alguna información de estado. Hace un seguimiento de qué mensajes del usuario han sido marcados para borrado. Sin embargo, el servidor POP3 no mantiene información de estado entre diferentes sesiones. Esta falta de información de estado entre distintas sesiones simplifica mucho la implementación de un servidor POP3.

Con el acceso POP3, una vez que B ha descargado sus mensajes a su máquina local, puede crear carpetas de correo y mover los mensajes descargados a estas. B puede entonces eliminar mensajes, moverlos de carpeta y buscar mensajes. Pero este paradigma (**carpetas y mensajes en la máquina local**) posee un problema para el usuario nómada, que prefiere mantener una jerarquía de correos en un servidor remoto que puede ser accedido desde cualquier orndeador. Esto no es posible con POP3. Para solucionar este y otros problemas, se inventa el protocolo IMAP.

- **IMAP:** es un procotolo de acceso a correo. Tiene muchas más caracterísitcas que POP3, pero es también mucho más complejo.

  Un servidor IMAP asocia cada mensaje con una carpeta. Cuando un mensaje llega por primera vez al servidor, se asocia a la carpeta INBOX del receptor. El receptor puede entonces mover el mensaje a una nueva carpeta creada por él, leer el mensaje, eliminarlo,...

  Luego, todas las comunicaciones toman la forma de comandos enviados por el cliente y respuestas retornadas por el servidor (como POP3). Es posible enviar varios comandos a la vez y posee cuatro estados: NOT AUTHENTICATED, AUTHENTICATED, SELECTED, LOGOUT.



IMAP proporciona comandos para permitir a los usuarios crear carpetas y mover mensajes entre carpetas. IMAP también proporciona comandos que permiten a los usuarios buscar en carpetas remotas mensajes que verifiquen ciertos criterios.

Nótese que para esto IMAP debe mantener información de estado del usuario entre sesiones IMAP diferentes.

También tiene comando que permiten a un user agent obtener componentes de mensajes

IMAP funciona sobre TCP en el puerto 143, en un modelo cliente-servidor. Primero se establece la conexión IMAP.

Pero presenta una desventaja: si no hay conexión IP no se puede acceder a este servicio y, por tanto, el correo no está disponible.

- **Web Mail:** Con este servicio, el user agent es un navegador Web normal, y el usuario se comunica con su mailbox remoto via HTTP. Cuando un receptor quiere acceder a un mensaje en su mailbox, el email es enviado desde el servidor mail del receptor al navegadir del receptor usando el protocolo HTTP en lugar de POP3 o IMAP. Cuando un emisor quiere enviar un mensaje, el email es enviado desde su navegador a su servidor mail sobre HTTP en lugar de sobre SMTP.

3.2 MIME

Multipurpose Internet Mail Extension (MIME) es una extensión en el marco del RFC 5322 que intenta solventar algunos de los problemas y limitaciones del uso de SMTP.

Las limitaciones del esquema SMTP/5322 son:

- SMTP no puede transmitir ficheros ejecutables u otros objetos binarios
- SMTP no puede transmitir datos que incluyan caracteres nacionales de algunos lenguajes por la representación en códigos de 8 bits
- Los servidores SMTP pueden rechazar emails que sobrepasen cierto tamaño
- Los mecanismos SMTP que traducen entre ASCII y el código EBCDIC no usan un conjunto consistente de mapeado, resultando en problemas de traducción
- Las interfaces SMTP de redes email X.400 no pueden manejar datos no textuales incluidos en los mensajes X.400
- Algunas implementaciones SMTP no se ajustan totalmente al estándar SMTP

MIME intenta resolver estos problemas de una forma que sea compatible con el estándar 5322. La especificación de MIME incluye los elementos siguientes:

1. Se definen 5 nuevos campos de cabecera, que proporcionan información sobre el cuerpo del mensaje
2. Se define una cantidad de formatos de contenido, lo que estandariza las representaciones que permiten el email multimedia
3. Las codificaciones se definen de forma que permitan la conversión de cualquier formato de contenido a una forma que quede protegida de cualquier alteración por el sistema de mensajes

Los **campos de cabecera** definidos en MIME son:

- **MIME-version:** debe tener el valor 1.0. Este campo indica que el mensaje se adecúa a los RFCs 2045 y 2046
- **Content-type:** describe los datos contenidos en el cuerpo con suficiente detalle que el user agent receptos puede elegir el mecanismo correcto de representación de datos para el usuario o tratar esos datos de la forma adecuada
- **Content-Transfer-Encoding:** indica el tipo de transformación que ha sido usada para representar el cuerpo del mensaje de una forma que sea aceptable para el transporte
- **Content-ID:** usado para identificar entidades MIME de forma unívoca en múltiples contextos
- **Content-Description:** una descripción de texto del objeto con el cuerpo. Esto es útil cuando el objeto no es leíble.

MIME content types

El bulto de la especificación MIME se preocupa de la definición de una variedad de tipos de contenido. Esto refleja la encesidad de proporcionar maneras estandarizadas de tratar con una amplia variedad de representaciones de la información en un entorno multimedia.

Para el **text type** del cuerpo, no se requiere ningún software especial para conseguir el total significado del texto aparte del soporte para el conjunto de caracteres (character set) indicado.

El campo **multipart type** indica que el cuerpo contiene partes múltiples e independientes. El campo de cabecera Content-Type incluye un parámetro, llamado boundary, que define el delimitador entre partes del cuerpo. Este boundary no debe aparecer en ninguna parte del mensaje. Cada boundary comienza en una nueva línea y consiste en dos guiones seguidos del valor de boundary. El boundary final, que indica el final de la última parte, también tiene un sufijo con dos guiones. En cada parte, puede haber un encabezado opcional MIME.

```
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: Formatted text mail
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary = boundary42

   —boundary42
Content-Type: text/plain; charset = us-ascii
   ...plain text version of message goes here....
   —boundary42
Content-Type: text/enriched
   ...RFC 1896 text/enriched version of same message
goes here...
   —boundary42—
```

Hay 4 subtipos del multipart type, y todos tienen la misma sintaxis general.

- El **multipart/mixed subtype** se usa cuando hay múltiples partes independientes en el cuerpo que necesitan ser tratadas en un orden en particular.
- Para el **multipar/parallel subtype** el orden de las partes no es significativo.

- El **multipart/alternative subtype** indica que las distintas partes son diferentes representaciones de la misma información. Las partes se ordenan en términos de preferencia creciente
- El **multipart/digest subtype** se usa cuando cada parte del cuerpo se interpreta como un mensaje RFC 5322 con cabeceras. Este subtipo permite la construcción de un mensaje cuyas partes son mensajes individuales

El **message type** proporciona una cantidad importante de capacidades en MIME:

- **message/rfc822 subtype** indica que el cuerpo es un mensaje completo, incluyendo cabecera y cuerpo. A pesar el nombre del subtipo, el mensaje encapsulado podría no ser solo un mensaje RFC 5322 sino cualquier mensaje MIME
- **message/partial subtype** permite la fragmentación de un mensaje grande en partes, que deben ser reunidas en destino. Se especifican tres parámetros en el Content-Type:
  - Message/Partial: un id común a todos los fragmentos de todo el mensaje
  - Un número de secuencia único para cada fragmento
  - El número total de fragmentos
- **message/external-body subtype** indica que los datos transmitidos en este mensaje no están contenidos en el cuerpo. En su lugar, el cuerpo contiene la información necesaria para acceder los datos. Tiene una cabecera externa y un mensaje encapsulado con su propia cabecera. El único campo necesario en la cabecera exterior es el Content-Type, que indicará que es un message/external-body subtype. La cabecera itnerior es la cabecera del mensaje para el mensaje encapsulado. El campo Content-Type en la cabecera exterior debe incluir un parámetro access-type, que indica el método de acceso, como FTP
- **application type** refiere a otros tipos de datos, tipicamente o datos binarios sin interpretar o información para ser procesada por una aplicación basada en mail

MIME transfer encodings

El otro componente principal de la especificación MIME es la definición de las codificaciones de transferencia para los cuerpos de los mensajes. El objetivo es proporcionar una entrega asegurada a través del mayor rango posible de entornos.

El estandar MIME define dos métodos de codificación de datos. El campo **Content.Transfer-Encoding** puede tomar 6 valores. Sin embargo, tres de estos valores (7bit, 8bit y binario) indican que ninguna codificación se ha efectuado, solo proporciona información sobre el futuro de los datos.

Otro valor es **x-token**, que indica que algún otro esquema de codificación se ha utilizado pero su nombre debe ser proporcionado. Esto puede ser un esquema específico de una aplicación concreta.

La codificación **quoted-printable** es útil cuando los datos consisten mayoritariamente en octetos que se corresponden con caracteres ASCII printables. Representa caracteres inseguros para la representación hexadecimal de su código e introduce line breaks reversibles para limitar las líneas del mensaje a 76 caracteres.

La codificación **base64** es común para codificar datos binarios de tal forma que sea invulnerable al procesamiento por los programas de transporte mail

Forma Canónica

Es el formato apropiado para el content type, que se estandariza para el uso entre sistemas. Esto está en ocntraste con la forma nativa, que es el formato que puede ser peculiar para un sistema concreto.

4 Protocolos de acceso a la red: DNS

El identificador de un host es su **hostname**, que son mnemotécnicos y por tanto entendibles por los humanos. Sin embargo, proporcionan muy poca o ninguna información sobre la localización en Internet del host. Además, como los hostnames consisten en cadenas de texto de longitud variable, serían muy difíciles de procesar por los routers. Por estas razones, los hosts también se identifican por las llamadas **direcciones IP,** que consisten en 4 bytes y tienen una estructura jerárquica rígida. Es jerárquica porque conforme procesamos la dirección de izquierda a derecha, obtenemos más y más información específica sobre dónde está el host en internet.

4.1 Servicios proporcionados por el DNS

El **domain name system (DNS)** es (1) una base de datos distribuida implementada en una jerarquía de servidores DNS y (2) un protocol de la capa de aplicación que permite a los host hacer peticiones a la base de datos distribuida.

El protocolo DNS se ejecuta sobre UDP y usa el puerto 53.

DNS es comúnment empleado por otros protocolos del nivel de aplicación para traducir los hostnames proporcionados por los usuarios a direcciones IP. Esto se hace de la siguiente forma:

1. La máquina del usuario ejecuta la parte del cliente de la aplicación DNS
2. El navegador extrae el hostname de la URL y pasa el hostname a la parte del cliente de la aplicación DNS
3. El cliente DNS envía una petición que contiene el hostname a un servidor DNS
4. El cliente DNS recibirá una respuesta, que incluirá la dirección IP para ese hostname
5. Una vez el navegador recibe la dirección IP desde el DNS, puede iniciar una conexión TCP con el servidor HTTP localizado en el puerto 80 de esa dirección IP

Vemos como el uso del DNS añade un delay adicional a la aplicaciones que lo usan.

Además de traducir hostnames a direcciones IP, DNS proporciona otros servicios importantes:

- **Host aliasing:** un host con un hostname complejo puede tener uno o más alias. Uno de ellos será el **hostname canónico**, los demás son los **hostname alias**, que cuando están presentes son normalmente más fáciles de recordar que los canónicos. Así, DNS puede ser invocado por una aplicación para obtener el hostname canónico para un alias proporcionado, así como la dirección IP del host.
- **Mail server aliasing:** es altamente deseable que las direcciones email sean fáciles de recordar. DNS puede ser invocado por una aplicación mail para obtener el hostname canónico para un alias proporcionado, así como la dirección IP del host. De hecho, el **registro MX** permite al servidor mail y web de una compañía tener hostnames idénticos
- **Distribución de carga:** DNS se usa también para dsitribuir la carga entre servidores replicados. Sitios muy visitados se replican en distintos servidores, con cada servidor ejecutándose en diferentes sistemas finales y cada uno con una dirección IP distinta. Para servidores web

replicados, un conjunto de direcciones IP se asocia con el hostname canónico. La base de datos DNS contiene este conjunto de direcciones IP. Cuando los clientes hacen una petición DNS por un nombre mapeado a un conjunto de direcciones, el servidor responde con el conjunto completo de direcciones IP, pero rota el orden de las direcciones con cada respuesta. Como un cliente normalmente envía sus peticiones HTTP a la dirección IP listada en primera posición, la rotación que hace DNS distribuye el tráfico entre los servidores replicados

4.2 Visión general del funcionamiento de DNS

Supongamos que alguna aplicación que está ejecutándose en la máquina de un usuario necesita traducir un hostname a una dirección IP. La aplicación invocará la parte de cliente del DNS, especificando el hostname que quiere traducir. Ahora DNS toma el control, enviando una petición a la red. Todos los mensajes de petición y respuesta DNS se envían mediante datagramas UDP al puerto 53. Tras un delay (desde ms a s), el DNS en la máquina del usuario recibe un mensaje de respuesta DNS que proporciona la traducción deseada. Esta traducción se pasa entonces a la aplicación que la requirió.

Desde la perspectiva de la aplicación en el ordenador del usuario, el DNS es una caja negra que le da un servicio de traducción simple y directa. Pero, de hecho, la caja negra que implementa el servicio es compleja y consiste de un gran número de servidores DNS distribuidos por el mundo, así como un protocolo en la capa de aplicación que especifica cómo los servidores y clientes DNS se comunican.

Un diseño simple para DNS sería tener un solo servidor DNS que contuviera todas las traducciones. En este **diseño centralizado** los clientes envían las peticiones directamente al único servidor DNS, que responde directamente a los clientes solicitantes. Aunque la simplicidad del diseño es atractiva, no es apropiada para el Internet actual. Presenta los siguientes problemas:

- Tiene un único punto de fallo: si el servidor DNS falla, cae todo Internet
- Volumen de tráfico: un único servidor DNS debería manejar todas las peticiones
- Base de datos centralizada y distante: un único servidor DNS no puede estar cerca de todos los clientes solicitantes
- Mantenimiento: el único servidor DNS debería mantener la información de todos los hosts de Internet. No solo debería ser enorme, sino que debería de actualizarse constantemente para tener en cuenta a todos los nuevos hosts

Por tanto, una base de datos centralizada en un único servidor DNS no escala. Consecuentemente, DNS debe ser distribuido.

Una base de datos jerárquica y distribuida

Para tratar el problema de la escalabilidad, DNS usa un gran número de servidores, organizados de forma jerárquica y distribuidos por el mundo. Ningún servidor tiene por sí solo todas las traducciones para todos los hosts de Internet.

En una primera aproximación, hay tres grandes clase de servidores DNS, organizados jerárquicamente.

El cliente primero contacta uno de los **servidores raíz** (root servers), que devuelven la dirección de los **servidores TLD**

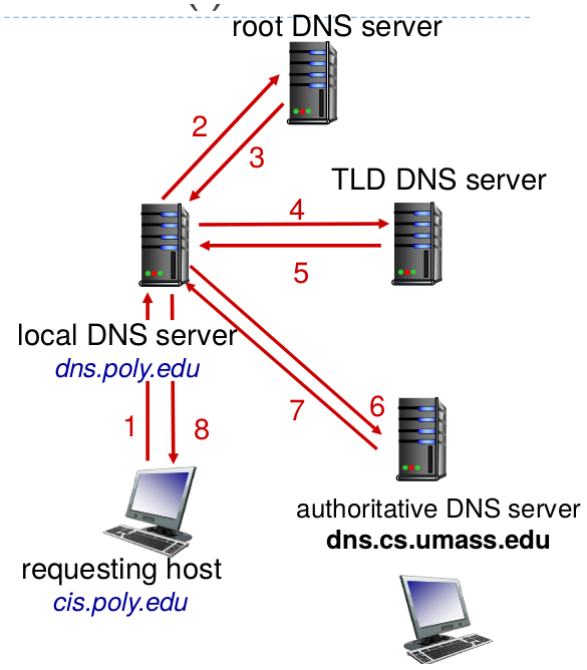
(top level domain). El cliente entonces contacta uno de estos servidores TLD, que le devuelve la dirección de un **servidor autorizado** (authoritative). Finalemente, el cliente contacta uno de estos servidores autorizados, que le devuelve la dirección IP del hostname buscado.

- **Servidores DNS raíz:** en Internet hay 13 servidores DNS raíz, la mayoría de los cuales están en Norteamérica. Aunque nos hayamos referido a cada uno de los 13 servidores DNS raíz como si fueran un único servidor, cada servidor es en realidad una red de servidores replicados, para proporcionar seguridad y fiabilidad. En total hay 247 servidores raíz.
- **Servidores TLD:** estos servidores son responsables de los dominios de alto nivel como com, org, net, edu o gov, además de los dominios de alto nivel de cada país.
- **Servidores DNS autorizados:** toda organización con hosts públicamente accesibles en Internet deben proporcionar registros DNS accesibles públicamente que mapeen los nombres de esos hosts con direcciones IP. Un DNS autorizado de una organización almacena estos registros DNS. Una organización puede elegir implementar su propio servidor DNS autorizado o pagar por tener estos registros almacenados en un servidor DNS autorizado de algún proveedor de servicios.

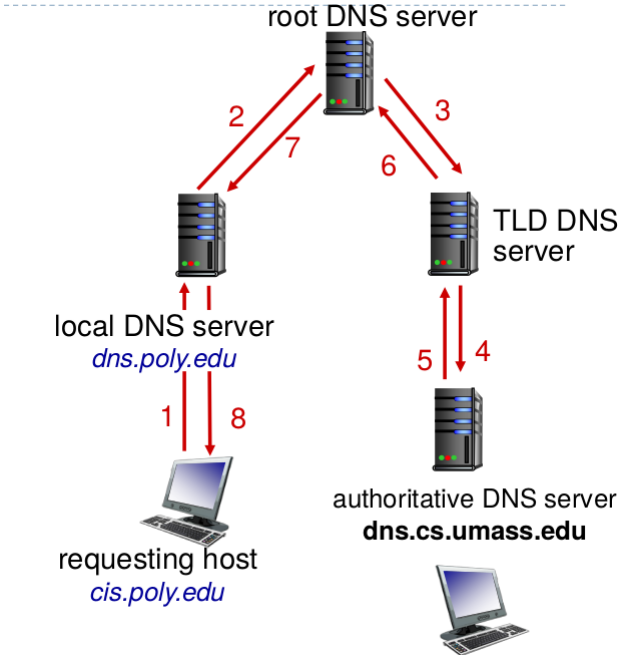
Hay otro tipo importante de servidor DNS llamado **servidor DNS local**, que no pertenece estrictamente a la jerarquía de servidores pero es importante en la arquitectura de DNS. Cada ISP tiene un servidor DNS local. Cuando un host conecta con un ISP, este proporciona al host la IP de uno o más de sus servidores DNS locales. Cuando un host hace una petición DNS, la petición es enviada al servidor DNS local, que actúa como un proxy, reenviando la petición a la jerarquía de servidores DNS.

La solicitud de **búsqueda iterativa** realiza una búsqueda en la base de datos de la dirección IP relacionada con el nombre de dominio, si no la obtiene pregunta al dominio donde realizar la próxima búsqueda. Si la consulta es **recursiva**, la resolución prevé la dirección de nuevo sin necesidad de realizar ningún otra consulta. Si es iterativa, el servidor DNS devuelve una dirección donde puede estar ubicada la dirección y si no, se conserva en la misma dirección. La resolución depende de la posibilidad de contactar la dirección o si se debe ubicar una ruta a través de su propia lista de servidores DNS.

Iterativa:



Recursiva:



Lo más común es que la petición del cliente al DNS local sea recursiva, pero las siguientes sean iterativas.

DNS Caching

DNS explota el DNS caching para mejorar los delays y reducir el número de mensajes DNS circulando por la red. En una cadena de peticiones, cuando un servidor DNS recibe una respuesta DNS, puede cachear la traducción en su memoria local.

Si un par hostname/IP es cacheado en un servidor DNS y una nueva petición llega al servidor preguntando por el mismo hostname, ahora puede dar la traducción directamente, aunque no sea un

servidor autorizado para el hostname. Como los hosts y las traducciones entre hostnames y direcciones IP no son permanentes, los servidores DNS descartan la información cacheada tras un período de tiempo.

4.3 Registros y mensajes DNS

Los servidores DNS implementan la base de datos distribuida DNS y almacenan **resource records (RRs)**. Cada mensaje de respuesta DNS lleva uno o más RRs. Un RR es una cuádrupla que contiene los campos:

(Name, Value, Type, TTL)

TTL es el tiempo de vida del RR, determina cuándo un RR debería ser eliminado de la caché. El significado de **name** y **value** depende de **type:**

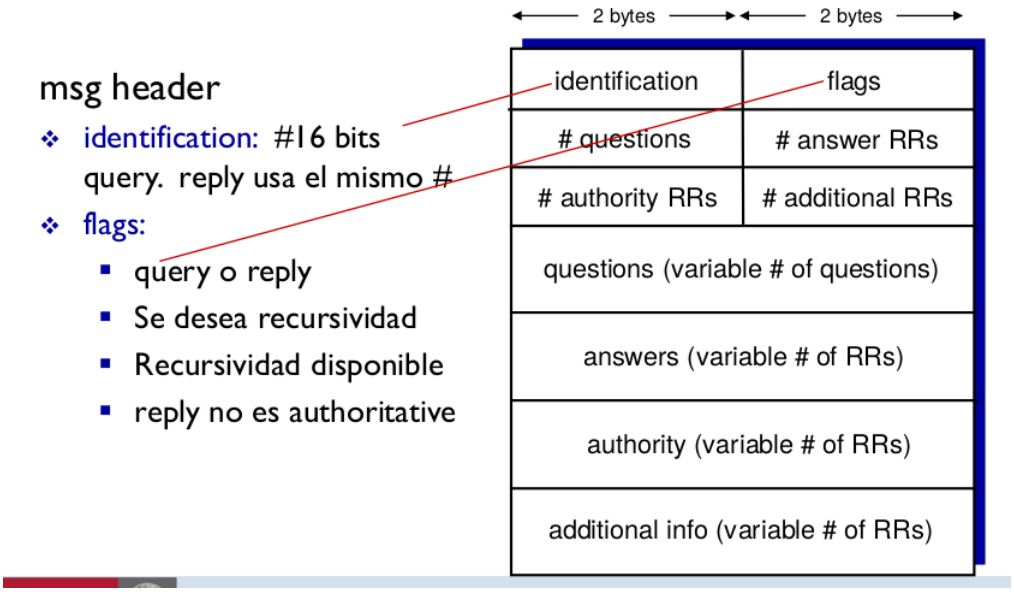
- **type=A**: entonces Name es el hostname y Value es la dirección IP asociada al hostname
- **type=NS:** entonces Name es un domiio y Value es el hostname de un servidor DNS autorizado que conoce cómo obtener las direcciones IP de hosts en ese dominio
- **type=CNAME:** entonces Value es el hostname canónico del hostname alias Name
- **type=MX:** entonces Value es el hostname canónico de un servidor mail que tiene un hostname alias Name

Si un servidor DNS es autorizado para un hostname particular, entonces el DNS contendrá un registro Type A para el hostname.

Si un servidor no es autorizado para un hostname, entonces el servidor contendrá un registro Type NS para el dominio que incluye el hostname. También contendrá un registro Type A que proporcionará la dirección IP del servidor DNS en el campo Value del registro NS.

Mensajes DNS

Los mensajes de petición y respuesta tienen el mismo formato.



- Los primeros 12 bytes son la **sección de cabecera**. El primer campo es un número de 16 bits que identifica la petición. Este identificador se copia en el mensaje de respuesta a una petición, permitiendo al cliente asociar respuestas recibidas con peticiones enviadas. Hay una cantidad de flags en el **campo de flags**. Un flag query/reply de 1 bit que indica si el mensaje es una petición (0) o una respuesta (1). Un flag autorizado de 1 bit que vale 1 cuando el servidor DNS es autorizado para un nombre requerido. Un flag de recursion-desired de 1 bit que se pone a 1 cuando el cliente desea que el servidor DNS efectúe recursión cuando no encuentre el registro. Un flag de recursion-available de 1 bit que se pone a 1 en una respuesta cuando el servidor DNS soporta recursión. En la cabecera, también hay cuatro campos de números, que indican la cantidad de ocurrencias de los cuatro tipos de secciones de datos que siguen a la cabecera.

- **Sección de pregunta:** contiene información sobre la petición que se está haciendo. Incluye (1) un campo de nombre que contiene el nombre por el que se pregunta y (2) y campo de tipo que indica el tipo de pregunta que se está haciendo acerca del nombre

- **Sección de respuesta:** contiene los RRs para el nombre por el que se preguntó. Una pregunta puede retornar varios RRs en la respuesta, porque un hostname puede tener múltiples direcciones IP

- **Sección de autoridad:** contiene registros de otros servidores autorizados

- **Sección adicional:** contiene otros registros útiles

Insertando Registros en la base de datos DNS

Un **registrador (registrar)** es una entidad comercial que verifica la unicidad de un nombre de dominio, introduce el nombre de dominio en la base de datos DNS y recibe una tasa por estos servicios.

Cuando registras el nombre de dominio networkutopia.com con algún registrador, necesitas proporcionar al registrador los nombres y direcciones IP de tus servidores DNS autorizados primario y secundario. Para cada uno de los servidores DNS autorizados, el registrador debe asegurar que los registros Type NS y Type A se introducen en los servidores TLD com. También debes asegurarte de que los RR Type A de tu servidor web y el RR Type MX de tu servidor de correo están introducidos en tus servidores DNS autorizados.

4.4 DNS primario y secundario

Existen dos tipos de servidores DNS:

- **Primario o Maestro:** almacenan en el mismo ordenador la información referente a las zonas que gestiona

- **Secundario o Esclavo:** obtiene información sobre una zona copiando la que tiene el servidor primario que la administra

Los dos están autorizados a responder sobre una zona, y se encuentran en direcciones IP distintas.

4.5 Vulnerabilidades del DNS

- **Ataque DDoS de inundación de ancho de banda:** un atacante podría intentar enviar a cada servidor raíz un diluvio de paquetes, de forma que la mayor parte de peticiones DNS legítimas no puedan ser respondidas. Un ataque DDoS a tan gran escala tuvo lugar. Los atacantes usaron una botnet para enviar muchísimos mensajes ping ICMP a cada uno de los 13 servidores raíz. Sin embargo, el daño fue mínimo. Los atacantes consiguieron enviar un diluvio de paquetes a

los servidores raíz, pero muchos de estos servidores estaban protegidos con filtros de paquetes, configurados para bloquear siempre los ping ICMP dirigidos a los servidores raíz. Estos servidores protegidos estaban funcionando de forma normal.

Un ataque DDoS potencialmente más peligroso sería enviar un aluvión de peticiones DNS a un servidor TLD. Sería más complicado filtrar las peticiones DNS dirigidas a los servidores DNS, y los servidores TLD se acceden más comúnmente que los servidores raíz. De todas formas, la severidad de estos ataques sería parcialmente mitigada por el caching en los servidores DNS locales

- **Ataque Man-in-the-middle:** el atacante intercepta peticiones desde los host y devuelve respuestas falsas.
- **Ataque DNS poisoning:** el atacante envía respuestas falsas a un servidor DNS, engañando al servidor aceptando registros falsos en su caché.

  Cualquiera de estos ataques podría ser utilizado para redirigir a un usuario despistado a la web del atacante. Sin embargo, la implementación de estos ataques es compleja.
- **Ataque DDoS a un host objetivo:** el atacante envía peticiones DNS a muchos servidores DNS autorizados, cada petición conteniendo como dirección de origen la del host objetivo. El servidor DNS entonces envía sus respuestas directamente al host objetivo. Si las peticiones pueden hacerse de tal forma que la respuesta sea mucho más larga que la petición, entonces el atacante puede, potencialmente, abrumar al objetivo sin generar mucho de su propio tráfico. Pero este ataque ha tenido muy poco impacto hasta el momento.

Así, DNS ha demostrado ser sorprendentemente robusto contra ataques.

5 Protocolos de acceso a la red: DHCP

Un host normalmente tiene un único enlace a la red. Cuando IP en el host quiere enviar un datagrama, lo hace por ese enlace. La frontera entre el host y el enlace fñisico se denomina **interfaz**.

Un router tiene múltiples interfaces, una para cada enlace. Como cada host y router es capaz de enviar y recibir datagramas IP, IP requiere que cada host y cada interfaz de un router tenfa su propia dirección IP. Por tanto, una dirección IP técnicamente está asociada con una interfaz, y no al equipo que tiene la interfaz.

Cada dirección IP tiene 32 bits, por lo que hay 2^{32} posibles direcciones IP. Estas direcciones normalmente se escriben en **notación decimal con puntos (dotted-decimal)**, en la que cada byte de la dirección se escribe en su forma decimal y se separa por un punto de los otros bytes de la dirección (e.g. 192.32.217.9).

Cada interfaz en todo host y router en el Internet global debe tener una dirección IP que sea globalmente única (excepto para NATs). Estas direcciones no pueden ser elegidas al azar. Una porción de la dirección IP de la interfaz será determinada por la subred a la que está conectado.

Para determinar las subredes, desconecta cada interfaz de su host o router, creando islas de redes aisladas, con interfaces marcando el final de las redes aisladas. Cada una de estas redes aisladas se denomina **subred**.

Una organización con múltiples segmentos Ehternet y enlaces punto a punto tendrá varias subredes, con todos los dispositivos en una misma subred teniendo la misma dirección de subnet. En principio, diferentes subredes podrían tener bastante distinta dirección de subred. En la práctica las direcciones de subred suelen tener bastante en común.

La estrategia de asignación de direcciones en Internet se conoce como **Classless Interdomain Routing (CIDR)** y generaliza la noción del direccionamiento de subredes. Como en estas, la dirección de 32 bits se divide en dos partes y ahora tiene la forma en decimal punteado $a.b.c.d/x$, donde x indica los números de bits que son la primera parte de la dirección, que constituye la porción de red de la dirección IP, y se suele denominar **prefijo**.

A una organización normalmente se le asigna un bloque de direcciones contiguas (con prefijo común).

El prefijo es considerado por los routers fuera de la red de la organización.

Los restantes $32 - x$ bits de la dirección pueden pensarse como un distintivo entre los dispositivos dentro de la organización, todos con el mismo prefijo. Estos son los bits que se considerarán cuando se reenvíen paquetes en los routers dentro de la organización.

Antes de CIDR, las porciones de una dirección IP estaban restringidas a ser los primeros 8,16 o 24 primeros bits. Este esquema de direccionamiento se conoce como **Classful Addressing**, y las clases se denominaban A, B o C, respectivamente.

Hay un tipo especial de dirección IP, la **dirección de broadcast 255.255.255.255**. Cuando un host envia un datagrama con esta dirección como destino, el mensaje se entregará a todos los hosts de la misma subnet. Opcionalmente los routers pueden derivar el mensaje a las subredes vecinas.

5.1 Obteniendo un bloque de direcciones

Para obtener un bloque de direcciones IP para usar en una subred de una organización, un administrador de red debe contactar con su ISP, que le proporcionará direcciones de un bloque mayor de direcciones que ha sido asignado al ISP.

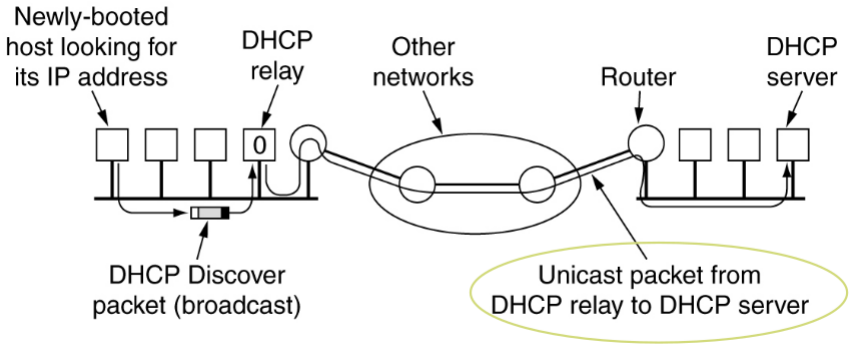
5.2 Obteniendo una dirección de host: el DHCP (Dynamic Host Configuration Protocol)

Una vez que una organización ha obtenido un bloque de direcciones, puede asignar direcciones IP individuales a los hosts y las interfaces de los routers en su organización. Un administrador del sistema normalmente configurará manualmente las direcciones IP en el router.

Las direcciones IP de los host pueden ser configuradas manualmente, pero lo más común es usar DHCP, que permite a un host obtener una dirección IP automáticamente. Un administrador de red puede configurar DHCP de forma que un determinado host reciba siempre la misma dirección IP cuando se conecta a la red, o un host puede ser asignado con una **dirección IP temporal**, que sera distinta cada vez que el host se conecte a la red. Además de la asignación de direcciones IP a los hosts, DHCP también permite a un host aprender información adicional, como la máscara de subred, la dirección de su router más cercano y la dirección de su servidor DNS local.

Suele decirse que es un **protocolo plug-and-play**.

DHCP es un protocolo cliente-servidor. El cliente normalmente es un nuevo host que quiere obtener la información de configuración de la red, incluyendo una dirección IP para sí. En el caso más simple, cada subred tedrá un servidor DHCP. Si no hay un servidor en la subred, habrá un **agente DHCP relay**, que conoce la dirección de un servidor DCHP.



El protocolo DHCP tiene cuatro pasos:

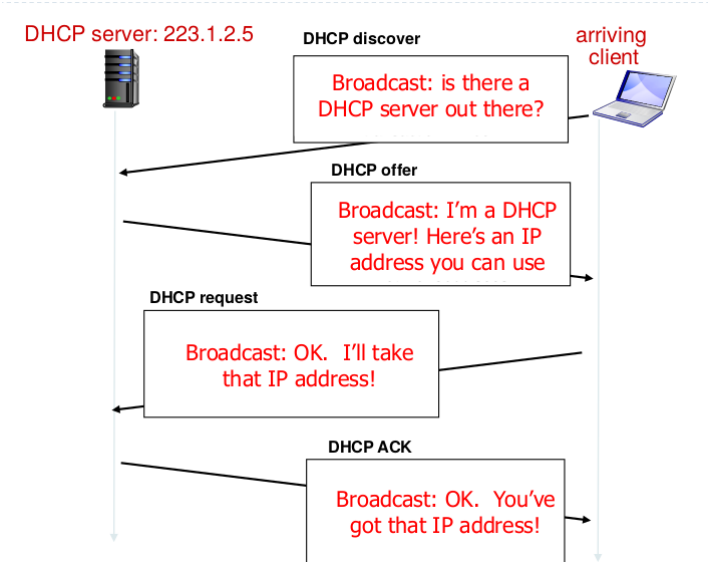
1. **DHCP server discovery:** la primera tarea de un nuevo host es encontrar un servidor DHCP con el que itneractuar. Esto se hace usando un mensaje DHCP discover, que un cliente manda en un paquete UDP por el puerto 67. Este se encapsula en un datagrama IP. El host no conoce ni siquiera la dirección IP de la red en la que opera. El cliente crea un datagrama IP que contiene el mensaje DHCP discover y dirección de destino la dirección de broadcast y dirección de origen 0.0.0.0. El cliente DHCP pasa el datagrama IP a la capa de enlace, que entonces retransmite la trama a todos los nodos de la subred

2. **DHCP server offer(s):** un servidor DHCP que recibe un mensaje DHCP discover responde al cliente con un mensaje DHCP offer que se retransmite a todos los nodos de la subred, usando de nuevo la dirección de broadcast. Como varios servidores DHCP pueden estar presentes en una subred, el cliente puede encontrarse en la posición de tener que elegir entre varias ofertas. Cada mensaje de oferta contendrá la ID de transacción del mensaje DHCP discover recibido, la dirección IP propuesta al cliente, la máscara de red, y el tiempo de préstamo de la dirección IP (lease time), o sea la cantidad de tiempo por la que esa dirección IP será válida.

3. **DCHP request:** el nuevo cliente elegirá entre una o más ofertas de los servidores y responderá a la que elija con un mensaje DHCP request, devolviendo los parámetros de configuración.

4. **DHCP ACK:** el servidor responde al mensaje DHCP request con un mensaje DHCP ACK, confirmando los parámetros requeridos.

Otros mensajes DCHP son DHCP Release, enviado por el cliente para liberar su dirección DHCP; DHCP Nak, que indica que un préstamo ha expirado o que la solicitud del cliente no pudo ser llevada a cabo; y DHCP Inform, que es enviado por el cliente para solicitar más información de la que el servidor le ha enviado con el DHCP ACK.



Una vez que el cliente recibe el DHCP ACK, la interacción termina y el cliente puede utilizar la dirección IP asignada mediante DHCP durante el tiempo de préstamo. Para ello:

- Envía un DHCP Request solicitando la misma IP que ya posee
- Se se renueva el servidor envía un DHCP ACK
- Si no se puede renovar la IP, el servidor envía un DHCP Nak. En este caso se inicia de nuevo el proceso general.

DHCP también proporciona un mecanismo que permite al cliente renovar el préstamo de una dirección IP.

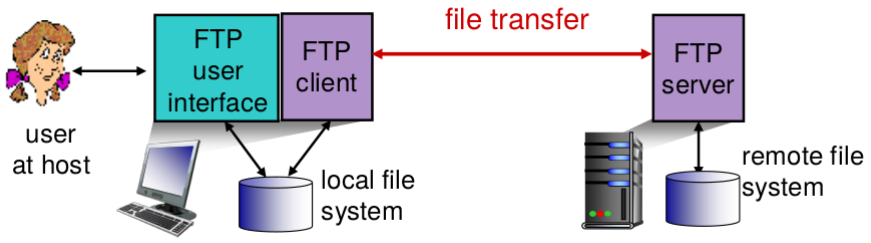
El valor de la característica plug-and-play queda claro considerando el hecho de que la alternativa es configurar manualmente la dirección IP de cada host (pensemos, por ejemplo, qué ocurriría en una biblioteca pública).

Desde el aspecto de la movilidad, sin embargo, DHCP tiene defectos. Como una nueva dirección IP es obtenida con DHCP cada vez que un nodo se conecta a una red, una conexión TCP a una aplicación remota no puede ser mantenida mientras un nodo móvil se desplaza entre redes.

6 Protocolos de aplicación: File Transfer Protocol (FTP)

En una sesión FTP típica, el usuario quiere transferir ficheros a/desde un host desde/a un host remoto. Para que el usuario pueda acceder la cuenta remota, el usuario debe proporcionar una identificación de usuario y una contraseña. Tras esto, el usuario puede transferir archivos desde el sistema de ficheros local al sistema remoto y viceversa.

El usuario interactúa con FTP a través de un **FTP user agente**. El usuario primero proporciona el hostname del host remoto, haciendo que el proceso cliente FTP en el host local establezca una conexión TCP con el proceso servidor FTP en el host remoto. El usuario entonces proporciona el ID de usuario y la contraseña, qu se envían sobre una conexión TCP como parte de comandos FTP. Una vez el servidor ha autorizado al usuario, el usuario copia uno o más archivos almacenados en el sistema de ficheros local al remoto.



HTTP y FTP son ambos protocolos de transferencia y tienen muchas características en común. Sin embargo, los dos protocolos del nivel de aplicación tienen importantes diferencias. La más importante es que FTP usa dos conexiones TCP paralelas para transmitir un fichero. La **conexión de control** se usa para mandar información de control entre los dos hosts. La **conexión de datos** se usa para enviar los ficheros.

Por este motivo, se dice que FTP envía la información de control **out-of-band (fuera de banda)**. HTTP se dice que envía la información de control **in-band**.

Cuando un usuario comienza una sesión FTP con un host remoto, la parte cliente de FTP primero inicia una conexión TCP de control con la parte servidor en el puerto 21 del servidor. La parte cliente de FTP envía ID+contraseña por esta conexión de control. La parte cliente de FTP envía también comandos para cambiar el directorio remoto. Cuando la parte del servidor recibe un comando requiriendo una transferencia de un archivo el servidor inicia una conexión TCP de datos con la parte del cliente. Si, durante la misma sesión, el usuario quiere transferir otro fichero, FTP abrirá otra conexión de datos. Por tanto, la conexión de control permanece abierta durante la duración de la sesión completa, pero las conexiones de datos se crean para cada fichero transferido en una sesión.

A lo largo de una sesión, el servidor FTP debe mantener el estado del usuario. Debe asociar la conexión de control con una cuenta de usuario específica, y debe hacer un seguimiento del directorio actual del usuario mientras el usuario se mueve por el árbol de directorios remoto.

6.1 FTP: modos de operación

6.1.1 Modo activo

1. El cliente inicia la conexión usando

$$IPOrigen = cliente,\ PuertoOrigen = x$$

$$IPDestino = servidor,\ PuertoDestino = 21$$

2. El cliente envía el comando PORT, indicando el número de puerto en el cliente que debe usar el servidor para el envío de datos (Y)

3. El servidor crea nueva conexión TCP usando:

$$IPOrigen = servidor,\ PuertoOrigen = 20$$

$$IPDestino = cliente,\ PuertoDestino = Y$$

6.1.2 Modo pasivo

1. (Igual) El cliente inicia la conexión usando

$$IPOrigen = cliente,\ PuertoOrigen = x$$

$$IPDestino = servidor,\ PuertoDestino = 21$$

2. El cliente envía el comando PASV para activar el modo pasivo, el servidor devuelve un número de puerto libre (Z)

3. El cliente crea nueva conexión TCP usando:

$$IPOrigen = cliente,\ PuertoOrigen = Y$$

$$IPDestino = servidor,\ PuertoDestino = Z$$

6.2 Comandos y respuestas FTP

Los comandos, del cliente al servidor, y las respuestas, del servidor al cliente, se envían a través de la conexión de control en formato ASCII de 7 bits. Así, como los comandos HTTP, son entendibles por los humanos. Un retorno de carro y un line feed terminan cada comando.

- **USER username:** usado para enviar la identificación del usuario al servidor
- **PASS password:** usado para enviar la contraseña al servidor
- **LIST:** usado para pedir al servidor que envíe una lista de todos los ficheros del directorio remoto actual. La lista de ficheros se envñia sobre una conexión de datos nueva
- **RETR filename:** usado para transferir un fichero desde el directorio remoto actual
- **STOR filename:** usado para almacenar un fichero en el directorio remoto actual

Normalmente hay una correspondencia uno-a-uno entre los comandos del usuario y los comandos FTP enviados a lo largo de la conexión de control. Cada comando va seguido de una respuesta, enviada desde el servidor al cliente. Las respuestas son números de tres dígitos, con un mensaje opcional a continuación.

- **331 Username OK, password required**
- **125 Data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

7 Introducción a la seguridad en redes

7.1 ¿Qué es la seguridad en redes?

Cuando Alicia envía un mensaje a Bob, es probable que quiere que Bob sea el único que pueda leer el mensaje que ella le envía, aunque se estén comunicando sobre un medio inseguro, en el que un intruso, al que vamos a llamar Truso, podría interceptar el mensaje. Bob también querrá estar seguro de que lo que recibe efectivamente lo ha enviado Alicia, así como Alicia quiere asegurarse de que se está comunicando con Bob. Alicia y Bob también quieren asegurarse de que el contenido de los mensajes no se ha alterado en el camino. Así, podemos identificar las siguientes propiedades deseables en una **comunicación segura**:

- **Confidencialidad:** solo el emisor y el receptor requerido deben ser capaces de entender el contenido del mensaje transmitido. Como un espía puede interceptar el mensaje, esto requiere que el mensaje, de alguna forma, sea **encriptado** de forma que un mensaje interceptado no pueda ser entendido por el interceptor.
- **Integridad:** Alicia y Bob quiere asegurarse de que el contenido de su comunicación no se altera en tránsito.
- **Autenticación end-point:** el emisor y el receptor deben ser capaces de confirmar la identidad del otro.
- **Seguridad operacional:** casi todas las organizaciones tienen redes y están conectadas a Internet. Estas redes, potencialmente, pueden quedar comprometidas. Los atacantes pueden intentar depositar worms en los hosts de la red, obtener secretos corporativos,...

Por otro lado, in intruso podría:

- **Espiar (eavesdrop)**
- **Modificar, insertar o eliminar mensajes o parte de estos**

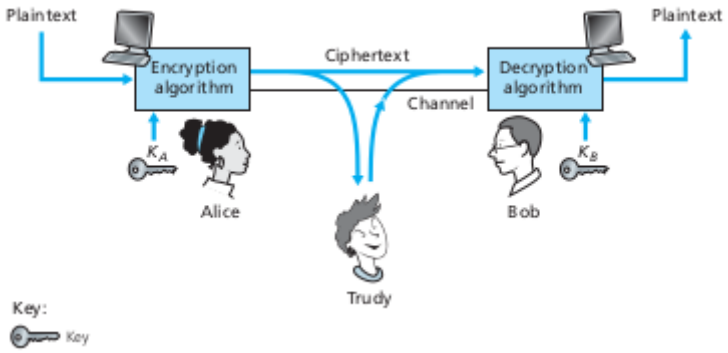
Mientras no se tomen las contramedidas apropiadas, estas capacidades habilitan a un intruso para llevar a cabo una gran variedad de ataques a la seguridad: **fisgonear** la comunicación, **impersonar** a una entidad, **secuestrar** una sesión de comunicación, **denegar el servicio** a una red legítima de usuarios sobrecargando el sistema,...

La utilidad de la criptografía para proporcionar confidencialidad es evidente, pero también es central en proporcionar autenticación end-point e integridad en los mensajes.

7.2 Principios de la criptografía

Las técnicas crptográficas permiten a un emisor disfrazar los datos, de forma que un intruso no pueda obtener información de ellos aunque los intercepte. El receptor debe ser capaz de recuperar los datos originales desde los datos camuflados.

El mensaje de Alicia en su forma original se conoce como **plaintext (texto plano)**. Alicia encripta su mensaje en texto plano usando un **algoritmo de encriptación** de forma que el mensaje encriptado, llamado **ciphertext (tecto cifrado)**, sea ininteligible para cualquier intruso.



Si todos saben el método para encriptar los datos, entonces debe haber alguna información secreta que prevenga a un intruso de desencriptar los datos transmitidos. Aquí es donde entran las keys.

En la figura anterior, Alicia proporciona una **clave (key)**, K_A, que no es más que una cadena de números o caracteres, al algoritmo de encriptación. Este toma la key y el mensaje en texto plano, m,

como input y produce como output el texto cifrado, $K_A(m)$. De forma similar, Bob proporcionará una key, K_B, al **algoritmo de decriptación**, que toma el texto cifrado y la key de Bob como input y produce el texto plano original como output. Es decir, K_B es tal que $K_B(K_A(m)) = m$.

En los **sistemas de clave simétrica**, las keys de Alicia y Bob son idénticas y secretas.

En los **sistemas de clave pública**, se usa un par de keys. Una de ellas es conocida tanto por Alicia como por Bob. La otra solo es conocida por uno de ellos.

7.2.1 Criptografía de clave simétrica

Cifrado César

Toma cada letra del abecedario en el texto plano y la sustituye por una letra que está k letras después en el alfabeto. La clave es el valor de k. Nótese que no llevaría mucho tiempo descifrar el código, ya que solo hay tantas claves posibles como letras tiene el abecedario.

Cifrado monoalfabético

Es una mejora del cifrado César. También sustituye una letra del alfabeto por otra. Sin embargo, ahora no se desplazan todas las letras una distancia k, sino que cualquier letra puede ser sustituida por cualquier otra, siempre y cuando dos letras distintas no se sustituyan por la misma y una misma letra siempre se sustituya por la misma letra. Como se puede observar, esto son las permutaciones de las letras del alfabeto, por lo que habrá 26! posibilidades para la clave (en alfabeto inglés). Por fuerza bruta es infactible probar todas las combinaciones. Pero hay técnicas estadísticas y conocimientos lingüísticos que pueden facilitar mucho la decodificación del mensaje.

Cuando consideramos cómo de fácil sería para Truso romper el esquema de codificación de Alicia y Bob, podemos distinguir 3 escenarios, dependiendo de la información que posee Truso:

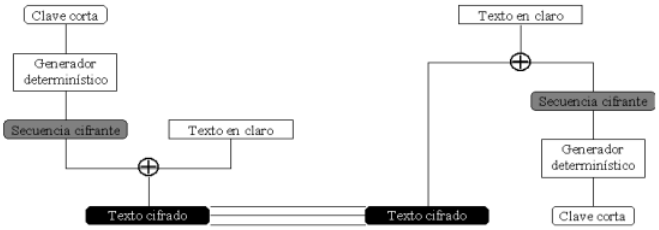
- **Ataque solo al texto cifrado:** el intruso podría tener acceso solo al texto cifrado interceptado, sin información sobre le contenido del mensaje original
- **Ataque conociendo el texto plano:** cuando un intruso conoce algunos de los emparejameintos (*plaintext*, *ciphertext*)
- **Ataque con texto plano seleccionado:** el intruso puede elegir el texto plano y obtener el correspondiente texto cifrado

Cifrado polialfabético

La idea es usar varios cifrados monoalfabéticos, con un cifrado monoalfabñetico específico para codificar una letra en una determinada posición del mensaje en texto plano. Así, la misma letra, en distintas posiciones del mensaje en texto plano, podría estar codificada de forma distinta.

Cifrado de Flujo

Se utiliza una clave con el mismo número de bits que el mensaje a cifrar. Se utiliza un generador de números pseudo-aleatorios (PRNG) para generar la clave.



Cifrado de Bloque

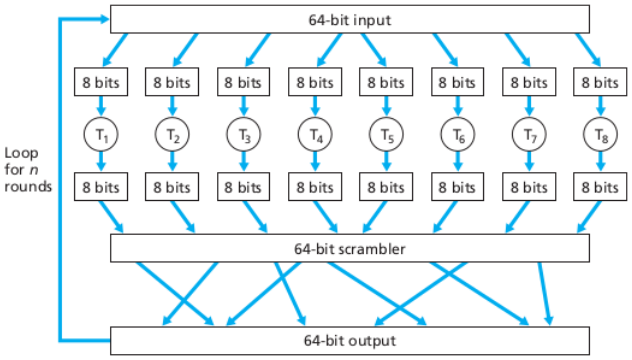
El mensaje a encriptar se procesa en bloques de k bits. Para codificar un bloque, el cifrador usa una asociación uno-a-uno para asociar el bloque de k-bits de plaintext con un bloque de k-bits de ciphertext.

Podemos ver cada una de estas asociaciones como una clave.

El ataque por fuerza bruta para este cifrado es intentar desencriptar el ciphertext usando todas las asociaciones. Pero nótese que el número de posibles asociaciones para un k-bloque en general es 2^k!

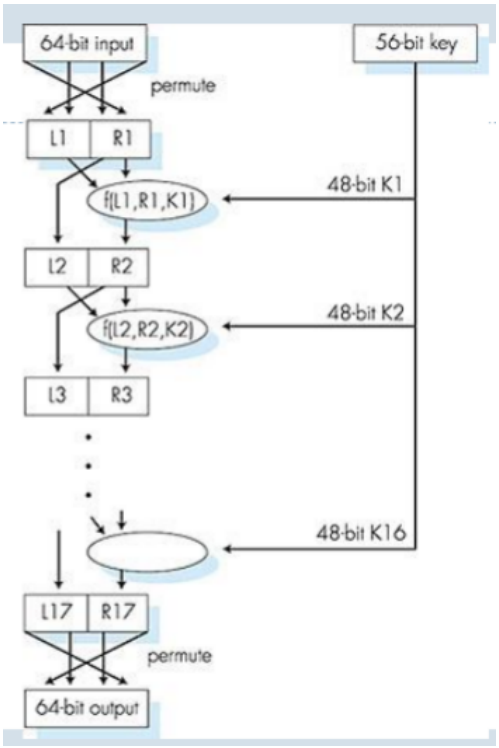
Aunque cifrados de bloque que usan una tabla de asociaciones, para valores moderados de k pueden producir esquemas de encriptación de clave simétrica muy robustos, son difíciles de implementar. Para $k = 64$ y una asociación dada, Alicia y Bob deberían mantener una tabla con 2^{64} valores de entrada, lo que es una tarea infactible. Además, si Alicia y Bob quisieran intercambiar claves, deberían, cada uno, regenerar la tabla.

En su lugar, los cifrados de bloque normalmente usan funciones que simulan tablas permutadas de forma aleatoria.



Los cifrados de bloque más populares son:

- **DES (Data Encryption Standard):** es el estándar de cifrado US encryption standard. Usa clave simétrica de 56-bits, con entrada plaintext de 64-bits. Es un cifrado de bloque con cipher block chaining.



Una frase cifrada con una clave de 56 bits se descifra en menos de un día. Pero no se le concoe un buen ataque analítico.

- **DES3:** surge para hacer DES más seguro. Consiste en cifrar 3 veces con 3 claves distintas:

$$E_{K3}\left(D_{K2}\left(E_{K1}\left(m\right)\right)\right)$$

- **AES (Advanced Encryption Standard):** sustituye a DES como estándar. Procesa los datos en bloques de 128 bits. Puede trabajar con claves de 128, 192 y 256 bits. Una máquina que pudiera descifrar DES en un segundo, tardaría 149 billones de años en descifrar AES.

Cipher Block Chaining

Si aplicamos cifrado de bloque como hemos visto, simplemente cortando el mensaje en trozos de k-bits y encriptando cada bloque independientemente, un problema sutil pero importante ocurre. Observemos que dos o más bloques del texto pueden ser idénticos. Para estos bloques idénticos, el cifrado en bloque produciría el mismo ciphertext. Un atacante podría potencialmente adivinar el plaintext cuando ve bloques de ciphertext idénticos y podría conseguir desencriptar el mensaje indentificando estos bloques y usando conocimiento específico de la estructura del protocolo.

Para solucionar este problema, podemos añadir un poco de aleatoriedad al ciphertext de forma que bloques de plaintext idénticos produzcan bloques de ciphertext distintos.

Sea $m\left(i\right)$ el bloque i del plaintext, $c\left(i\right)$ el bloque i del ciphertext, y $a \oplus b$ la operación XOR entre dos cadenas a y b. Denotemos, también, el algoritmo de encriptación de cifrado en bloque con clave S como K_S.

La idea básica es la siguiente:

1. El emisor crea un número de k-bits aleatorio $r(i)$ para el bloque i y calcula $c(i) = K_S(m(i) \oplus r(i))$
2. El emisor entonces envía $c(1), r(1), c(2), r(2), ...$ Como el receptor recibe $c(i), r(i)$ entonces puede recuperar clada bloque del plaintext calculando $m(i) = K_S(c(i)) \oplus r(i)$

Aunque $r(i)$ es enviado en claro y puede ser atrapado por Truso, no puede obtener el plaintext $m(i)$ porque no conoce la clave K_S.

Pero introducir esta aleatoriedad soluciona un problema, pero añade uno nuevo: ahora Alicia debe transmitir el doble de bits que antes. Aquí entra la técnica llamada **Cipher Block Chaining (CBC)**. La idea es enviar solo un valor random con el primer mensaje, y después usar los bloques anteriormente calculados en lugar de un nuevo número aleatorio. Queda, entonces:

1. Antes de encriptar el mensaje, el emisor genera una cadena random de $k-$bits, llamada **Vector de Inicialización (IV)**. A este lo denotamos $c(0)$. El emisor envía al receptor el IV en texto plano.
2. Para el primer bloque, el emisor calcula $c(1) = K_S(m(1) \oplus c(0))$ y envía $c(1)$ al receptor.
3. Para el i-ésimo bloque, el emisor genera el i-ésimo bloque de ciphertext haciendo

$$c(i) = K_S(m(i) \oplus c(i-1))$$

Esta técnica tiene las siguientes consecuencias:

- El receptor debe seguir siendo capaz de obtener el mensaje original. Y efectivamente es así, porque cuando recibe $c(i)$, ya ha calculado $c(i-1)$, por lo que solo tiene que hacer $m(i) = K_S(c(i)) \oplus c(i-1)$
- Si dos bloques de plaintext son idénticos, es casi seguro que sus correspondientes bloques de ciphertext no lo serán
- Aunque el emisor envía el IV en claro, el intruso no conoce la clave secreta
- El emisor solo envía un mensaje adicional a los bloques, el IV, por lo que el ancho de banda necesario para la comunicación no aumenta

7.2.2 Encriptación de Clave Pública

La comunicación encriptada con clave simétrica requiere que las dos partes en comunicación compartan un secreto común. Una dificultad de este enfoque es que las dos partes deben, de alguna forma, ponerse de acuerdo en la clave compartida, pero esto tambiénr equiere comunicación segura. Quizás las partes podrían reunirse primero y hacer este acuerdo en persona, y tras esto comunicarse con encriptación. En un mundo en red, sin embargo, las partes en comunicación puede que nunca se reunan y que nunca conversen aparte de sobre la red.

Ahora, Alicia y Bob, en lugar de compartir una clave secreta compartida, tienen dos claves cada uno.

Una **clave pública** ,K_B^+, que está disponible para cualquiera y una **clave privada** ,K_B^-, que solo la conoce el poseedor. La B es porque estamos suponiendo que es la de Bob.

Para comunicarse con Bob, Alicia busca la clave pública de Bob y encripta el mensaje, m, con esta clave y un algoritmo de encriptación conocido, o sea, Alicia calcula $K_B^+(m)$.

Bob recibe el mensaje encriptado y calcula $K_B^- \left(K_B^+ (m) \right)$.

Hay algoritmos de encriptación y desencriptación para elegir las claves públicas y privadas de tal forma que se obtenga $K_B^- \left(K_B^+ (m) \right) = K_B^+ \left(K_B^- (m) \right) = m$.

Una **primera consideración** es que aunque un intruso que intercepta el mensaje de Alicia no puede entenderlo, el intruso conoce tanto la clave (la pública de Bob), como el algoritmo de encriptación. Truso puede entonces montar un ataque de plaintext seleccionado, usando los algoritmos de encriptación estandarizados y la clave pública de Bob para codificar cualqueir mensaje que quiera. Entonces podría codificar mensajes o partes de mensajes que sospeche que Alicia podría enviar a Bob. Por tanto, para que la criptografía de clave pública funcione, la elección de claves debe ser hecha de forma que sea imposible para un intruso determinar la clave privada de Bob o desencriptar el mensaje de Alicia para Bob.

Una **segunda consideración** es que como la clave de encriptación de Bob es pública, cualquiera puede enviar un mensaje a Bob, incluyendo alguien que se hace pasar por Alicia. En el caso de una única clave secreta compartida, el hecho de que el emisor conoce la clave secreta implícitamente lo identifica. En el caso de la criptorgrafía con clave pública, sin embargo, esto no funciona. Una firma digital es necesario para asociar un emisor a un mensaje.

RSA

Hace uso de la aritmética modular. Se verifican las propiedades:

$$[a \mod n + b \mod n] \mod n = (a + b) \mod n$$

$$[a \mod n - b \mod n] \mod n = (a - b) \mod n$$

$$[a \mod n \cdot b \mod n] \mod n = (a \cdot b) \mod n$$

$$(a \mod n)^k \mod n = a^k \mod n$$

Como un mensaje no es más que un patrón de bits, todo mensaje puede ser representado por un número entero.

Entonces, encriptar un mensaje con RSA es equivalente a encriptar el único entero que representa el mensaje.

Hay dos componentes interrelacionados de RSA:

- La elección de las claves públicas y privadas
- El algoritmo de encriptación y desencriptación

Para **generar las claves públicas y privadas RSA**, Bob hace los siguientes pasos:

1. Elige dos primos grandes, p, q
2. Calcula $n = pq$ y $z = (p-1)(q-1)$
3. Elige un número, $e < n$, coprimo con z.
4. Encuentra un número, d, tal que $ed - 1$ es múltiplo de z
5. La clave pública de Bob es el par (n, e) y la clave privada es (n, d).

El proceso de encriptación por Alicia y desencriptación por Bob es como sigue:

Supongamos que Alicia quiere enviar el mensaje que, expresado como número, es $m < n$. Para obtener el mensaje codificado, hace

$$c = m^e \mod n$$

Y envía el mensaje codificado, c, a Bob.

Para desencriptar el mensaje, Bob calcula

$$m = c^d \mod n$$

¿Por qué funciona RSA?

Si p, q son primos, $n = pq$ y $z = (p-1)(q-1)$ entonces

$$x^y \mod n = x^{(y \mod z)} \mod n$$

Y, por tanto

$$c^d \mod n = (m^e \mod n)^d \mod n = m^{ed} \mod n \overset{\text{mod } z=1}{=} m$$

La seguridad de RSA se apoya en el hecho de que no hay algoritmos conocidos para factorizar rápidamente un número. En nuestro caso el número es n y sus factores p y q. Si alguien conociera p, q entonces dada la clave pública uno poría encontrar fácilmente la clave privada. Aunque, dado que no hay conocimiento sobre si tal algoritmo de factorización puede existir o no, la seguridad de RSA no puede estar totalmente garantizada.

Claves de sesión

La exponenciación que hay que realizar en RSA es un proceso que consume mucho tiempo. DES es al menos 100 veces más rápido en software y entre 1000-10000 veces más rápido en hardware.

RSA es normalmente usado en combinación con criptografía de clave simétrica. Si Alicia quiere enviar a Bob una gran cantidad de datos encriptados, ella podría hacer lo siguiente:

1. Alicia elige una clave que usará para codificar los datos. Esta clave se conoce como **clave de sesión**, K_S.
2. Alicia debe informar a Bob de la clave de sesión, que será una clave simétrica compartida. Por lo que encripta la clave usando la clave pública de Bob.
3. Bob recibe la clave de sesión encriptada y la desencripta usando su clave privada.
4. Ahora Bob también tiene la clave de sesión compartida, transmitida de forma segura y la comunicación puede comenzar usando la clave simétrica.

Diffie y Hellman (el de la mayonesa no, otro) crearon un algoritmo que soluciona esta limitación.

Intercambio de Claves Diffie-Hellman

El propósito del algoritmo es permitir a dos usuarios intercambiar de forma segura una clave que pueda después ser usada para encriptación simétrica de los mensajes. El algoritmo se limita al intercambio de valores secretos compartidos.

El algoritmo es efectivo por la dificultad de calcular logaritmos discretos.

Se denomina **raíz primitiva** de un primo p es un número cuyas potencias módulo p generan todos los enteros desde 1 hasta $p-1$. O sea, si a es raíz primitiva de p, entonces

$$a \mod p,\ a^2 \mod p, ..., a^{p-1} \mod p$$

don distintos y son los enteros del 1 al $p-1$ en alguna permutación.

Para cualquier entero b y una raíz primitiva a de un primo p, podemos definir un único exponente i tal que

$$b \equiv a^i \mod p$$

El exponente i es el **logaritmo discreto** de b con base a módulo p, o sea $i = dlog_{a,p}(b)$.

El algoritmo

Hay dos números conocidos públicamente: un primo q y un entero a que es raíz primitiva de q. Supongamos que Alicia y Bob quieren crear un secreto compartido.

Alicia escoge un entero aleatorio $X_A < q$ y calcula $Y_A = a^{X_A} \mod q$ y Bob hace lo propio $X_B < q$ y $Y_B = a^{X_B} \mod q$.

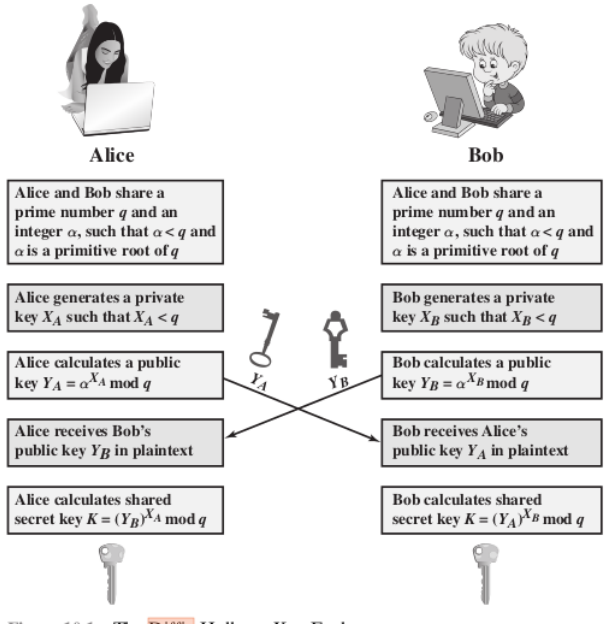
Cada parte guarda el valor de X y hace el valor de Y público. Así, X_A es la clave privada de Alicia y Y_A su clave pública, y de forma similar con Bob.

Alicia entonces calcula el secreto como $K = (Y_B)^{X_A} \mod q$ y Bob hace lo propio $K = (Y_A)^{X_B}$.

Estas dos claves resultan ser iguales:

$$K = (Y_B)^{X_A} \mod q = \left(a^{X_B} \mod q\right)^{X_A} \mod q = a^{X_B X_A} \mod q = \left(a^{X_A} \mod q\right)^{X_B} \mod q = (Y_A)^{X_B} \quad \text{r}$$

Este sereto compartido se puede usar como clave secreta simétrica compartida.



Ataque Man in the Middle

Supongamos que Alicia y Bob quieren intercambiar las claves y Truso quiere infiltrarse. El ataque procede así:

1. Truso se prepara generando dos claves privadas aleatorias X_{T1} y X_{T2} y calcula Y_{T1} e Y_{T2}
2. Alicia transmite Y_A a Bob
3. Truso intercepta Y_A y transmite a Bob Y_{D1}. Truso calcula $K2 = (Y_A)^{X_{T2}} \mod q$
4. Bob recibe Y_{D1} y calcula $K1 = (Y_{D1})^{X_B} \mod q$
5. Bob transmite Y_B a Alicia
6. Truso intercepta Y_B y trnasmite Y_{D2} a Alicia. Truso calcula $K1 = (Y_B)^{X_{T1}} \mod q$
7. Alicia recibe Y_{D2} y calcula $K2 = (Y_{D2})^{X_A} \mod q$

En este punto, Bob y Alicia creen que comparten una clave secreta, pero en su lugar Bob y Truso comparten el secreto $K1$ y Alicia y Truso comparten el secreto $K2$.

Todos los mensajes posteriores están comprometidos.

Vemos como el protocolo de itnercambio de claves es vulnerable a este ataque. Pero esta vulnerabilidad se solucionar con el uso de firmas digitales y certificados de clave pública.

7.3 Integridad de los mensajes y firmas digitales

Supongamos que Bob recibe un mensaje y cree que ha sido enviado por Alicia. Para autenticar el mensaje, Bob necesita verificar que (1) el mensaje es originario de Alicia y (2) no ha sido modificado en el camino.

7.3.1 Funciones Hash criptográficas

Una función hash toma un input, m, y obtiene una cadena de longitud fija $H(m)$ conocida como hash.

Una **función hash criptográfica** debe tener la propiedad adicional de que sea computacionalmente infactible encontrar dos mensajes diferentes x, y tales que $H(x) = H(y)$.

Si $(m, H(m))$ son el mensaje y el hash del mensaje creados por el emisor, entonces un intruso no puede forjar otro mensaje, y, de forma que tenga el mismo valor que el mensaje original.

Algoritmo MD5

Se usa ampliamente hoy en día. Calcula un hash de 128-bits en un proceso de 4 pasos consistente en:

1. Paso de relleno: añadir un 1 seguido de tantos 0s como sea necesario para que la longitud del mensaje satisfaga las condiciones requeridas
2. Paso de añadir: añadir una representación de 64-bit de la longitud del mensaje antes de rellenar
3. Inicialización de un acumulador
4. Paso final de bucle: los bloques de 16 palabras del mensaje son procesados en cuatro rondas

Secure Hash Algorithm (SHA-1)

Este algoritmo se basa en unos principios similares a los usados en el diseño de MD4, el predecesor de MD5. SHA-1, que es un estándar federal de EEUU, se requiere siempre que un algoritmo de hash cirptográfico se requiere en aplicaciones federales. Produce un mensaje de 160-bits.

7.3.2 Message Authentication Code

Veamos cómo podríamos llevar a cabo integridad de los mensajes:

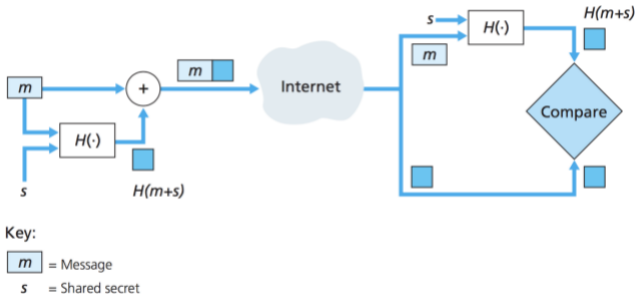
1. Alicia crea un mensaje m y calcula su hash $H(m)$
2. Alicia añade $H(m)$ al final de m, creando un mensaje extendido $(m, H(m))$ y lo envía a Bob
3. Bob recibe el mensaje extendido (m, h) y calcula $H(m)$. Si $H(m) = h \implies$ todo ha ido bien.

Este enfoque es obvio que tiene fallos. Truso podría crear un mensaje trampa m' en el que dice que es Alicia, calcula $H(m')$ y envía a Bob $(m', H(m'))$. La comprobación del paso 3 no informará a Bob de que no es Alicia quien le ha enviado el mensaje.

Además de usar la función hash criptográfica, Alicia y Bob necesitan un secreto compartido s. Este secreto compartido es la **clave de autenticación**. Usando esta, la integridad de los mensajes se realiza como sigue:

1. Alicia crea el mensaje m, concatena s con m para crear $m + s$ y calcula el hash $H(m + s)$. A este hash se le denomina **message authentication code (MAC)**
2. Alicia envía a Bob el mensaje extendido $(m, H(m + s))$
3. Bob recibe (m, h) y calcula $H(m + s)$. Si $H(m + s) = h \implies$ Todo ha ido bien

Una buena propiedad de una MAC es que no requiere un algoritmo de encriptación. Usando una MAC, las endidades pueden autenticar los mensajes que se envían entre ellas sin tener que integrar algoritmos complejos de encriptación en el proceso de integridad. El estándar más popular hoy en día es HMAC, que puede ser usado con MD5 o SHA-1.



7.3.3 Firmas digitales

Una **firma digital** es una técnica criptográfica que hace las veces de una firma en el mundo digital.

Debe ser posible probar que un documento firmado por un individuo fue realmente firmado por ese individuo y solo por ese individuo.

Supongamos que Bob quiere firmar digitalmente un documento m. Para ello, Bob simplemente usa su clave privada, K_B^- para calcular $K_B^-(m)$. En un principio, puede parecer extraño que Bob use su propia clave privada para firmar un documento. Pero recordemos que la encriptación y desencriptación no son más que operaciones matemáticas y que el objetivo de Bob es no revolver el contenido del documento, pero firmar el documento de forma que sea verificable y no modificable. La firma digital de Bob del documento es $K_B^-(m)$.

¿Es esta clave verificable y no modificable?

Supongamos que Alicia tiene m y $K_B^-(m)$. Quiere probar que Bob firmó el documento y que es la única persona que pudo haberlo hecho. Alicia calcula $K_B^+\left(K_B^-(m)\right)$ y obtiene m. Esto significa que quien firmó el documento tiene la clave privada de Bob. Por tanto debe ser Bob y hemos comprobado la verificabilidad.

Si el documento original se modificase y pasase a ser m', entonces la firma que Bob hizo para m no servirá para m' y por tanto se tiene la inalterabilidad.

Una consideración respecto a firmar datos por encriptación es que encriptación y desencriptación son operaciones costosas. Una forma más eficiente de hacerlo es introducir funciones Hash en la firma digital. Usando una función hash, Bob firma el hash de un mensaje en lugar del mensaje en sí mismo. O sea, Bob hace $K_B^-(H(m))$. Como $H(m)$ suele ser mucho más pequeño que el mensaje original, el esfuerzo computacional es mucho menor.

7.3.4 Comparación de MAC y firma digital

Ambos comienzan con un mensaje.

Para crear un MAC de este mensaje, añadimos una clave de autenticación al mensaje, y tomamos el hash del resultado.

Para crear una firma digital, primero tomamos el hash del mensaje y después encriptamos el mensaje con la clave privada.

Por tanto, la firma digital es una técnica más pesada.

7.3.5 Certificación de clave pública

Para que la criptografía de clave pública sea útil, debemos poder verificar que tenemos la clave pública de la entidad con la que nos queremos comunicar.

La asociación de una clave pública con una entidad particular la hacen las **Certification Authority (CA)**, cuyo trabajo es validar entidades y emitir certificados. Una CA tiene los siguientes roles:

- Verificar que una entidad es quien dice ser. No hay procesos estandar para esto. Cuando tratas con una CA, debes confiar en que la CA ha hecho una verificación rigurosa de la identidad
- Una vez que la CA verifica la identidad de la entidad, la CA crea un certificado que asocia la clave pública de la entidad con su identidad. El certitificado contiene la clave púbñica e información única de identificación sobre el poseedor de la clave pública. El certificado es digitalmente firmado por la CA

8 Certificados de identidad X.509

Un **directorio** es un servidor o conjunto de servidores distribuidos que mantiene una base de datos de información sobre usuarios.

X.509 define un marco de referencia para la provisión de servicios de autenticación por el directorio X.500 a sus usuarios. El directorio puede servir como repositorio de certificados de clave públic. Cada certificado contiene la clave pública de un usuario y está firmado con la clave privada de una CA de confianza. Además, X.509 define protocolos de autenticación alternativos basados en el uso de certificados de clave pública.

Se basa en el uso de criptografía de clave pública y firmas digitales. El estandar no dicta el uso de un algoritmo concreto pero recomienda RSA. El esquema de firma digital se asume que requiere el uso de una función hash. El estándar tampoco dicta el uso de un algoritmo hash específico.

8.1 Certificados

El núcleo del esquema X.509 es el certificado de clave pública asociado con cada usuario. Estos certificados de usuario se asumen creados por alguna CA de confianza y que fueron puestos en el directorio por la CA o por el usuario. El directorio no es responsable de la creación de claves públicas o de la función de certificación. Solamente proporciona un lugar de fácil acceso para que los usuarios obtengan certificados.

Un certificado tiene los siguientes campos:

- **Version**
- **Serial number:** un entero único dentro del CA emisor
- **Signature algorithm identifier:** el algoritmo usado para firmar el certificado junto con cualquier parámetro asociado
- **Issuer name:** nombre X.500 de la CA que creó y firmó el certificado
- **Period of validity:** consiste en dos fechas: la primera y la última en las que el certificado es válido
- **Subject name:** nombre del usuario al que refiere el certificado
- **Subject's public-key information:** la clave pública del usuario, más un identificador del algoritmo para el que esta clave debe usarse, junto con cualquier parámetro asociado
- **Issuer unique identifier:** un parámetro opcional usado para identificar al CA emisor en el evento
- **Subject unique identifier:** parámetro opcional que identifica al usuario en el evento
- **Signature:** cubre todos los otros campos del certificado: contiene el código hash de los otros campos encriptados con la clave privada del CA. Incluye el identificador del algoritmo de firma

$Y\ \langle\langle A\rangle\rangle$ denota que el certificado del usuario A ha sido emitido por la CA Y

La CA firma el certificado con su clave privada. Si la correspondiente clave pública es conocida por un usario, entocnes el usuario puede verificar que un certificado firmado por la CA es válido.

8.2 Obteniendo el certificado de un usuario

Los certificados generados por un CA tienen las siguientes características:

- Cualquier usuario con acceso a la clave pública del CA puede verificar la clave pública que fue certificada
- Nadie aparte del CA emisor puede modificar el certificado sin ser detectado

Como los certificados no son modificables, pueden ponerse en un directorio sin necesidad de protegerlos especialmente.

Si todos los usuarios se suscribiesen al mismo CA, entonces habría una confianza generalizada en ese CA. Todos los certificados podrían ser puestos en el directorio para acceso de todos los usuarios. Además, un usuario puede transmitir su certificado directamente a otros usuarios. En cualquier caso, una vez que B tiene el certficado de A, B tiene la confianza de que los mensajes que encripte con la clave pública de A estarán seguros del espionaje y que los mensajes firmados con la clave privada de A no son modificables.

Si hay una gran comunidad de usuarios, puede no ser práctico que todos se suscriban al mismo CA. Como el CA firma certificados, cada participante debe tener una copia de la clave pública del CA para verificar firmas. Esta clave pública debe ser transmitida a cada usuario con total seguridad para que el usuario tenga confianza en los certificados asociados. Por tanto, con muchos usuarios, podría ser más práctico que hubiera varios CAs, cada uno proviendo su clave pública de forma segura a una porción de los usuarios.

Supongamos ahora que Alicia ha obtenido un certificado de una CA X y Bob lo ha obtenido de una CA Y. Si Alicia no conoce la clave pública de Y de forma segura, entonces el certificado de Bob es inútil para Alicia. Para solucionar esto, basta que las dos CAs intercambien de forma segura sus claves públicas:

1. Alicia obtiene del directorio el certificado de Y firmado por X. Como Alicia conoce de forma segura la clave pública de X, puede obtener la clave pública de Y de su certificado y verificarlo
2. Alicia entonces vuelve al directorio y obtiene el certificado de Bob firmado por Y. Como Alicia ya tiene una copia segura de la clave pública de Y, puede verificar la firma y obtener la clave pública de Bob de forma segura.

O sea, hace

$$X \langle\langle Y \rangle\rangle \, Y \langle\langle B \rangle\rangle$$

Todos estos certificados de CAs por CAs necesitan estar en el directorio, y el usuario debe saber cómo están relacionadas para encontrar el camino a la clave pública de otro usuario. X.509 sugiere que las CAs se ordenen de forma jerárquica de forma que la navegación sea obvia.

La entrada del directorio para cada CA incluye dos tipos de certificados:

- **certificados directos:** certificados de X generados por otras CAs
- **certificados inversos:** certificados generados por X que son certificados de otras CAs

8.3 Revocación de certificados

Recordemos que cada certificado incluye un período de validez. Un nuevo certificado suele ser emitido justo antes de la expiración del antiguo. Además, puede ser deseable revocar certificados antes de que expiren, por alguna de las razones siguientes:

- Se cree que la clave privada del usuario ha sido comprometida

- El usaurio ya no está certificado por la CA
- El certificado de la CA se cree que está comprometido

Cada CA debe mantener una lista consistente en todos los certificados revocados pero no expirados emitidos por esa CA, incluyendo tanto los emitidos para usuarios como para otras CAs. Estas listas deben estar en el directorio.

Estas listas se denominan **certificate revocation list (CRL)** y son firmadas por el emisor. Incluyen el nombre del emisor, la fecha en que se creó la lista, la fecha en la que la siguiente lista está programada para ser emitida, y una entrada por cada certificado revocado.

Cuando un usuario recibe un certificado en un mensaje, debe determinar si el certificado ha sido revocado. El usuario podría comprobar el directorio cada vez que recibe un certificado. Pero para ahorrar delays es probable que el usuario quiera mantener una caché local de certificados y de CRLs.

8.4 Public Key Infraestructure (PKI)

Es el conjunto de hardware, software, gente, políticas y procedimientos necesarios para crear, administrar, almacenar, distribuir y revocar certificados digitales basados en criptografía asimétrica.

El objetivo es permitir la adquisición segura, conveniente y eficiente de claves públicas.

El grupo de trabajo PKIX ha dirigido la fuerza detrás de establecer un modelo formal basado en X.509 que sea adecuado para desplegar una arquitectura de certificación en Internet.

Los elmentos clave de PKIX son:

- **Entidad final:** término genérico usado para denotar usuarios finales, dispositivos o cualquier otra entidad que pueda ser identificada como sujeto de un certificado de clave pública
- **Autoridad de certificación (CA):** el emisor de certificados y de CRLs. Puede dar soporte a diversas tareas administrativas, aunque estas pueden ser delegadas a una o más autoridades de registro
- **Autoridad de registro (RA):** componente opcional que puede asumir funciones administrativas del CA. El RA se suele asociar con el proceso de registro de una entidad final pero puede asistir en otras áreas.
- **Emisor CRL:** componente opcional en el que la CA puede delegar la emisión de CRLs
- **Repositorio:** término genérico usado para denotar cualquier método para almacenar certificados y CRLs para que puedan ser recuperados por las entidades finales

9 Protocolos para comunicaciones seguras: SSL/TLS

9.1 Secure Socket Layer (SSL)

Dos de los servicios de seguridad más extendidos son SSL y su sucesor TLS.

SSL es un servicio de propósito general implementado como un conjunto de protocolos que se apoya en TCP. Hay dos opciones de implementación.

SSL puede ser provisto como parte de la suite de protocolos inferior y ser transparente a las aplicaciones o puede ser embebido en paquetes específicos.

9.1.1 Arquitectura de SSL

SSL está diseñado para hacer uso de TCP para proporcionar un servicio punto a punto confiable. SSL no es un protocolo único,sino dos capas de protocolos.

Dos conceptos importantes son:

- **Conexión:** es un transporte que proporciona un tipo apropiado de servicio. Para SSL, las conexiones son relaciones P2P. Las conexiones son transitorias. Toda coenxión está asociada a una sesión.
- **Sesión:** es una asociación entre un cliente y un servidor. Las sesiones se crean por el Hanshake Protocol. Definen un conjunto de parámetros criptográficos de seguridad que pueden ser compartidos en múltiples conexiones. Las sesiones se usan para evitar la negociación cara de nuevos parámetros de seguridad para cada conexión.

Entre todo par de partes puede haber múltiples conexiones seguras. En teoría podría haber también múltiples sesiones simultáneas entre partes, pero esta característica no se usa en la práctica.

Hay cantidad de estados asociados con una sesión. El **estado de una sesión** se define por los parámetros:

- **Session identifier**
- **Peer certificate:** un certificado X509v3 del peer
- **Compression method:** el algoritmo usado para comprimir datos antes de encriptar
- **Cipher spec:** especifica el algoritmo de encriptación y el algoritmo de hash usado para el cálculo de MAC. También define atributos criptográficos.
- **Master secret:** secreto de 48 bits compartido entre el cliente y el servidor
- **Is resumable:** un flag que indica si la sesión se puede usar para iniciar nuevas conexiones

El **estado de una conexión** se define por los parámetros:

- **Server and client random:** secuencias de bytes elegidas por el servidor y el cliente para cada conexión
- **Server write MAC secret:** la clave secreta usada en las operaciones MAC en los datos enviados por el servidor
- **Client write MAC secret:** la clave secreta usada en las operaciones MAC en los datos enviados por el cliente
- **Server write key:** clave de encriptación secreta para los datos encriptados por el servidor y desencriptados por el cliente
- **Client write key:** clave de encriptación secreta para los datos encriptados por el servidor y desencriptados por el cliente
- **Initialization vectors:** cuando un cifrado de bloque en modo CBC se usa, se mantiene un IV para cada clave
- **Sequence numbers:** cada parte mantiene números de secuencia distintos para los mensajes transmitidos y recibidos para cada conexión

9.1.2 SSL Record Protocol

Proporciona dos servicios para las conexiones SSL:

- **Confidencialidad:** el Handshake Protocol define una clave secreta compartida que se usa para encriptación convencional de carga SSL
- **Message integrity:** el Handshake Protocol define una clave secreta compartida que se usa para formar un MAC

El Record Protocol toma un mensaje de aplicación para ser transmitido, lo fragmenta en bloques manejables (**fragmentación**), opcionalmente comprime los datos (**compresión**), aplica una MAC, encripta, añade una cabecera, y transmite la unidad resultante en un segmento TCP.

Los datos recibidos se desencriptan, verifican, descomprimen y se ensamblan antes de ser entregados a los usuarios de niveles superiores.

La cabecera añadida tiene los siguientes campos:

- **Content type**
- **Major version**
- **Minor version**
- **Compressed Length**

9.1.3 Change Cipher Spec Protocol

Es uno de los 3 protocoloos específicos de SSL que utiliza el SSL Record Protocol y es el más simple.

Consiste en un único mensaje que contiene un único byte con el valor 1. El propósito de este mensaje es causar que el estado pendiente se copie al estado actual, que actualiza la suite de cifrado que será usada en la conexión.

9.1.4 Alert Protocol

Se usa para transmitir alertas relacionadas con SSL al peer. Como con otras aplicaciones que usan SSL, los mensajes de alerta son comprimidos y encriptados, como se especifique en el estado actual. Cada mensaje de este protocolo consiste en dos bytes:

1. Toma los valores de warning (1) o de fatal (2) para transmitir la severidad de la alerta. Si el nivel es fatal, SSL termina la conexión. Otras sesiones abiertas pueden continuar, pero no se establecerán sesiones nuevas.
2. Contiene un código que indicala alerta en específico. Las alertas fatales son:
   - unexpected_message
   - bad_record_mac
   - decompression_failure
   - handshake_failure
   - illegal_parameter

   Las alertas de warning son:

- close_notify
- no_certificate
- bad_certificate
- unsupported_certificate
- certificate_revoked
- certificate_expired
- certificate_unknown

9.1.5 Handshake Protocol

Este protocolo permite al servidor y al cliente autenticar al otro y negociar algoritmos de encriptación y MAC y claves criptográficas a usar para proteger los datos enviados por el record SSL.

Este protocolo se usa antes de que los datos de aplicación se transmitan y conssite en una serie de mensajes intercambiados por cliente y servidor. Cada mensaje tiene tres campos:

- **Type:** indica uno de 10 tipos de mensajes
- **Length:** la longitud del mensaje en bytes
- **Content:** los parámetros asociados al mensaje

Fase 1: Estableciendo las Prestaciones de Seguridad

Esa fase se usa para comenzar una conexión lógica y para establecer las prestaciones de seguridad que serán asociadas con ella. El intercambio es iniciado por el cliente, que envía un mensaje **client_hello** con los siguientes parámetros:

- **Version:** la version SSL más alta entendida por el cliente
- **Random:** una estructura random generada por el cliente consistente en un timestamp de 32 bits y 28 bytes generados por un generador de números aleatorios seguro. Estos valores sirven como nonce (number that can be only used once) y se usan durante el intercambio de claves para prevenir ataques de replicación
- **Session ID:** un identificar de sesión de longitud variable. Un valor distinto de 0 indica que el cliente quiere actualizar los parámetros de una conexión existente o crear una nueva conexión en esta sesión
- **CipherSuite:** lista que contiene las combinaciones de algoritmos criptográficos soportados por el cliente, el orden decreciente de preferencia. Cada elemento de la lista define un algoritmo de intercambio de claves y un algoritmo de cifrado
- **Compression Method:** lista de los métodos de compresión que soporta el cliente

Ahora el cliente espera el mensaje **server_hello** que contiene los mismos parámetros que el client_hello.

El primer elemento del parámetro CipherSuite es el método de intercambio de claves. Se soporta:

- **RSA**

- **Fixed Diffie-Hellman:** es un DH en el que el certificado de blace pública contiene los parámetros de la clave pública de DH. Este método resulta en una clave secreta fija entre dos peers basada en DH usando claves públicas fijas.
- **Ephemeral DH:** se usa para crear claves secretas temporales. Las claves públicas de DH se intercambian firmadas usando la clave privada RSA o DSS del emisor. El receptor puede usar la correspondiente clave pública para verificar la firma. Los certificados se usan para autenticar las claves públicas.
- **Anonymous DH:** DH sin autenticación, como vimos este proceso es vulnerable al ataque man in the middle.
- **Fortezza**

El segundo parámetro es el CipherSpec, que tiene los siguientes campos:

- **CipherAlgorithm** (DES, 3DES,...)
- **MACAlgorithm** (MD5 o SHA-1)
- **CypherType** (de serie o de bloque)
- **IsExportable** (true o false)
- **HashSize**
- **Key Material** (secuencia de bytes que contiene datos usados para generar las claves de escritura)
- **IV Size** (tamaño del IV)

Fase 2: Autenticación del servidor e intercambio de claves

El servidor comienza esta fase enviando su certificado si nevesita ser autenticado. El mensaje contiene un certificado x509 o una cadena de estos.

El **certificate message** se requiere para cualquier método de intercambio de claves acordado excepto anonymous DH.

Después, un mensaje **server_key_exchange** se envía si es requerido. No se requiere si: (1) el servidor ha enviado un certificado con fixed DH o (2) se va a utilizar intercambio de claves RSA. Se necesita en los demás casos:

- **Anonymous DH:** el contenido del mensaje consiste en los dos valores globales de DH y la clave pública DH del servidor
- **Ephemeral DH:** contiene los tres mismos àrámetros anteriores y una firma en esos parámetros
- **Intercambio de vlaces RSA** (cuando el servidor usa RSA pero tiene una clave RSA de solo firma): el cleinte no puede simplemente enviar una clave secreta encriptada con la clave pública del servidor. Tiene que crear un par temporal RSA de claves pública y privada y usar el mensaje server_key_exchange para enviar la clave pública. El contenido del mensaje incluye los dos parámetros de la clave pública RSA temporal y una firma de esos parámetros
- **Fortezza**

La firma se crea tomando el hash de un mensaje y encriptándolo con la clave privada del emisor.

El hash cubre no solo los parámetros DH o RSA sino también los dos nonces de los mensajes hello iniciales. Esto asegura ante ataques de replicación.

Después, un servidor no anónimo puede requerir un certificado del cliente. El mensaje **certificate_request** incluye dos parámetros: **certificate_type** y **certificate_authorities**. El certificate type indica el algoritmo de clave pública y su uso:

- RSA, solo firma
- DSS, solo firma
- RSA para fixed DH: la firma solo se usa para autenticación, enviando un certificado firmado con RSA
- DSS para fixed DH: igual que el anterior pero con DSS
- RSA para ephemeral DH
- DSS para ephemeral DH
- Fortezza

El segundo parámetro es una lista de los nombres de CAs aceptables.

El mensaje final de la fase dos es el mensaje **server_done**, enviado por el servidor para indicar el fin del hello y los mensajes asociados. Después de este mensaje, el servidor esperará a que el cliente responda. Este mensaje no tiene parámetros.

Fase 3: Autenticación del cliente e intercambio de claves

Tras recibir el mensaje server_done, el cliente debe verificar que el servidor ha proporcionado un certificado válido y comprobar que los parámetros del server_hello son aceptables. Si todo es correcto, el cliente envía uno o más mensajes para responder al servidor.

Si el servidor requiere un certificado, el cliente comienza esta fase enviando un **certificate message**. Si no hay certificados apropiados disponibles, el cliente envia una alerta no_certificate.

Ahora viene el mensaje **client_key_exchange**, que debe ser enviado ahora. El contenido de este mensaje depende del tipo de intercambio de claves:

- **RSA:** el cliente genera un secreto pre-master de 48 bytes y lo encripta con la clave pública del certificado del servidor o una clave RSA temporal del mensaje server_key_Exchange. Se usa más tarde para calcular un secreto master.
- **Ephemeral y anonymous DH:** los parámetros públicos de DH se envían
- **Fixed DH:** los parámetros públicos de DH se envían en un certificate message, por lo que el contenido del mensaje es nulo
- **Fortezza**

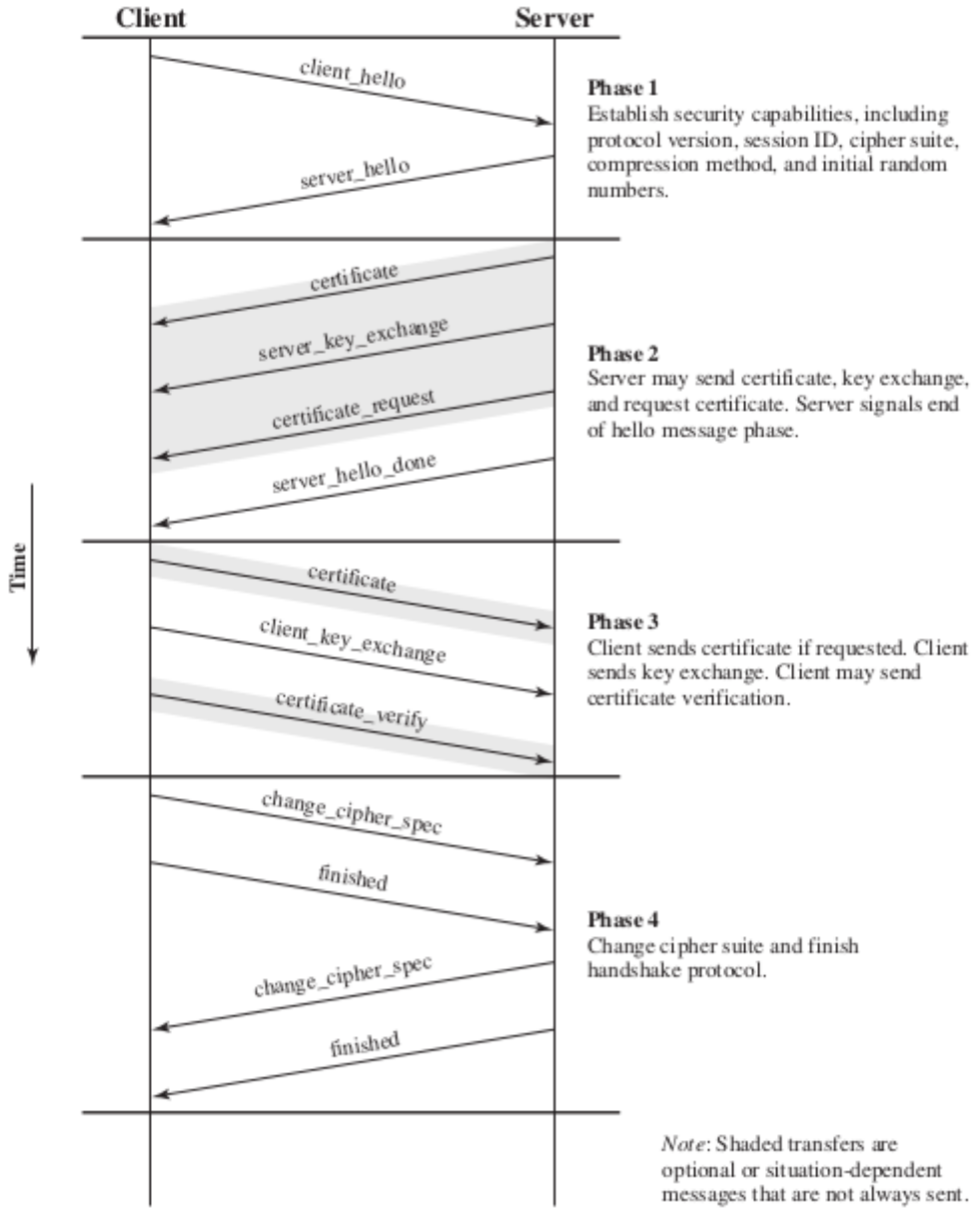
Finalmente, en esta fase el cliente podría enviar un mensaje **certificate_verify** para proporcionar una verificacióne xplícita de un certificado de cliente. Este mensaje solo se envía tras algún certificado del cliente que tenga capacidad de firma. Este mensaje firma un código hash basado en los mensajes anteriores.

Fase 4: Final

Esta fase completa el establecimiento de una conexión segura. El cliente envía un mensaje **change_cipher_spec** y copia el CipherSpec pendiente al CipherSpec actual. Este mensaje no se considera parte del Handshake Protocol sino del Change Cipher Spec Protocol. El cliente entonces envía el mensaje **finished** bajo los nuevos algoritmos, claves y secretos. El mensaje finished verifica que el intercambio de claves y los procesos de autenticación fueron exitosos.

En respuesta a estos dos mensajes, el servidor envía su propio mensaje change_cipher_spec, transfiere el CipherSpec pendiente al actual y envía su mensaje finished.

En este momento, el hanshake es completado y cliente y servidor pueden comenzar a intercambiar datos del nivel de aplicación.



Creación del secreto master

El secreto master compartido es un valor de 48 bytes de un único uso generado para esta sesión para un intercambio de claves seguro. La creación se hace en dos estapas:

1. Se intercambia la pre_master_secret
2. El master_secret se calcula por ambas pates.

Para el intercambio del pre_master_secret hay dos posibilidades:

- **RSA:** el cliente genera el pre_master_secret de 48 bytes y lo encripta con la clave pública RSA del servidor, y se lo envía. El servidor lo desencripta usando su clave privada.
- **DH:** cliente y servidor generan una clave pública DH. Cuando estas son intercambiadas, cada lado hace un cálculo DH para crear la pre_master_secret compartida.

Para generar el secreto master usan alguna función hash sobre la pre_master_secret y algunos más valores.

9.2 Transport Layer Security (TLS)

Tiene como objetivo producir una versión estándar, no dependiente de un solo fabricante, de SSL. TLS se define como un estándar propuesto de Internet y es muy similar a SSLv3. Las principales diferencias son:

- **Número de versión:** versión mayor 3, menor 1
- **MAC**: TLS tiene un algoritmo de firma basado en HMAC estándar
- **Función pseudoaleatoria:** usa una función pseudoaleatoria conocida como PRF para expandir valores secretos en bloques de datos útiles para generación de claves o validación. El objetivo es utilizar un valor secreto compartido relativamente pequeño, pero generar bloques de datos más grandes de forma segura contra los ataques a las funciones hash y a los MAC
- **Códigos de alerta:** TLS admite todos los códigos de alerta de SSLv3 con la excepción de la alerta no_certificate y define nuevos códigos de alerta
- **Padding variable:** siempre múltiplo de la longitud del bloque de cifrado
- **Suite de algoritmos de cifrado disponible:** no admite nulo ni Fortezza
- **Define más tipos de certificados de cliente**
- **Mensajes certificate_verify y finished:** en el mensaje certificate_verify se tiene campos extra. Los cálculos del mensaje finished son ligeramente diferentes. En ambos mensajes lo que cambia es el cáclulo del hash.

TLS 1.1: protege contra atauqes CBC

TLS 1.2: reemplaza MD5-SHA-1 por SHA-1 en PRF, mensaje finished y en el elemento firmado en el handshake (1 solo hash). Mejora en la especificación de los algoritmos soportados. Algoritmos de cifrado autenticado y suites AES. Extensiones TLS.

TLS 1.3: ha sido propuesto como Internet Standard. Mejora el rendimiento de als comunicaciones web. Principales mejoras respecto a TLS 1.2:

- Separación de los algoritmos de intercambio de claves y de autenticación en las suites

- Modo 0-RTT
- Todos los mensajes después del Server_hello están cifrados
- Rediseño de las funciones de derivación de clave: HDKF
- Curvas elípticas en la especificación base y nuevos algoritmos
- Resumen de sesión basado en PSK

10 Protocolos para comunicaciones seguras: IPsec

10.1 Conceptos básicos de IPSec

IPSec es una extensión de IP para proteger comunicaciones, diseñada para funcionar modo transparente en redes existentes.

Proporciona la capacidad de asegurar las comunicaciones a través de una LAN, de una WAN privada y pública y de Internet. Se usa, por rjemplo, para establecer una conexión segura entre oficinas sucursales a través de Internet, acceso remoto seguro a través de Internet, o mejorar la seguridad en el comercio electrónico.

La característica principal que tiene es que permite dar soporte a esta variedad de aplicaciones y puede cifrar y/o autenticar todo el tráfico en el nivel IP usando criptografóa. Por tanto, pueden asegurarse todas las aplicaciones distribuidas, incluyendo conexión remota, cliente/servidor, correo electrónico, transferencia de ficheros,...

Es independiente de los algoritmos de cifrado y firma y es aplicable para proteger IPv4 e IPv6

10.1.1 Beneficios de IPSec

- **IPSec puede ser transparente a usuarios finales**
- **IPSec puede proporcionar seguridad a usuarios individuales si es necesario**
- **IPSec protege la entrada a través de un firewall a una red:** cuando IPSec se implementa en un cortafuegos o un router, proporciona una gran seguridad que se puede aplicar a todo el tráfico que lo cruza.

10.1.2 Servicios IPSec

Proporciona servicios de seguridad en la capa IP, permitiendo que un sistema elija los protocolos de seguridad necesarios, determine los algoritmos que va a usar para el servicio o servicios y ubique las claves criptográficas necesarias para proporcionar los servicios solicitados. Se usan dos protocolos para proporcionar seguridad: un protocolo de autenticación designado por la cabecera del protocolo, AH, y un protocolo combinado de cifrado/autenticación designado por el formato del paquete para ese protocolo, ESP.

Los servicios son los siguientes:

- Control de acceso
- Integridad por paquete

- Autenticación de los datos origen
- Rechazo de paquetes reenviados
- Condifencialidad de los datos
- Confidencialidad limitada en el flujo de tráfico

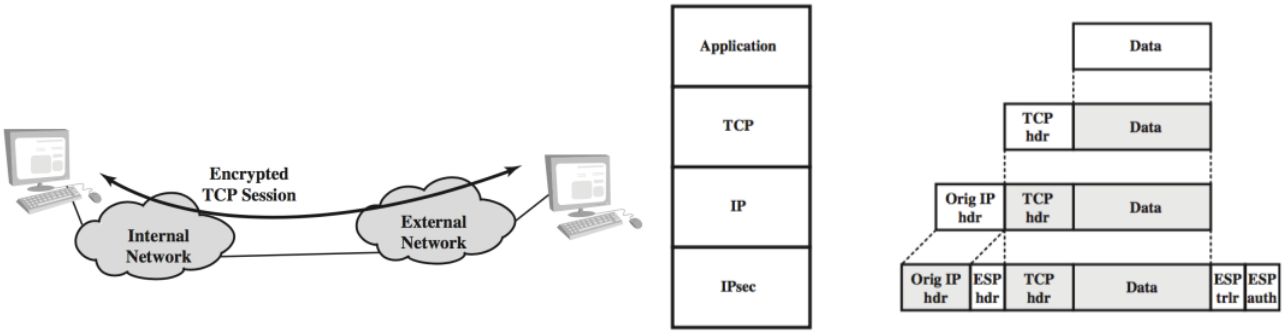
10.1.3 Modos de operación

- **Modo transporte:** proporciona protección a los protocolos de capas superiors. O sea, la protección del modo transporte se extiende a la carga útil del paquete IP. Normalmente el modo transporte se usa para la comunicación extremo a extremo entre dos hosts.

  Con AH autentica la carga útil y algunas porciones de la cabecera IP.

  Con ESP encripta la carga útil y cualquier cabecera de extensión de IPv6 que vaya tras la cabecera ESP.

  Con ESP con autenticación encripta la carga útil y las cabeceras de extensión de IPv6 que vayan tras la cabecera ESP y autentica la carga útil aunque no la cabecera IP.

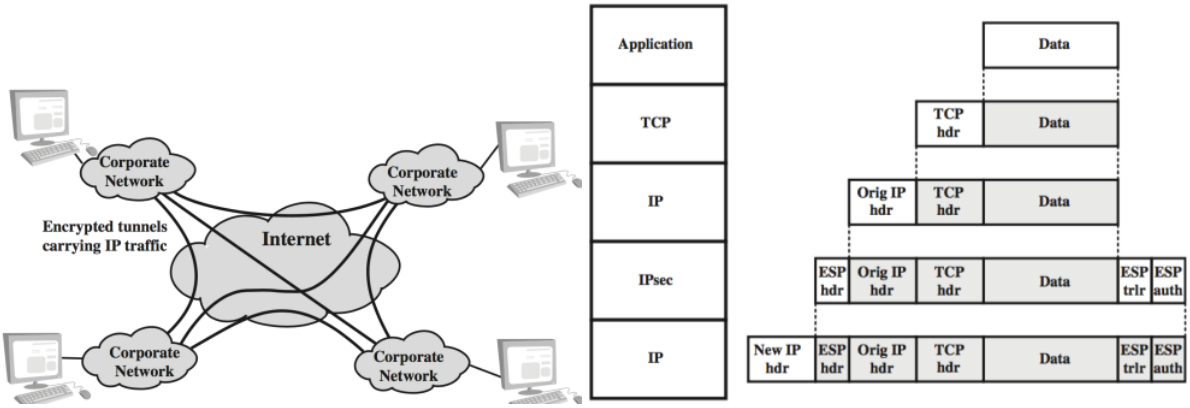


- **Modo túnel:** proporciona protección al paquete PI completo. Para conseguirlo después de que se han añadido los campos AH y ESP al paquete IP, el paquete completo más los campos de seguridad se tratan como carga útil de un paquete IP exterior nuevo con una nueva cabecera IP exterior. El paquete IP original entero viaja a través de un túnel desde un punto de la red IP a otro; ningún router a lo largo del camino puede examinar la cabecera IP interior. Como el paquete original está encapsulado, el nuevo paquete, que es mayor, puede tener direcciones de origen y destino totalmente diferentes, añadiendo seguridad. El modo túnel se usa cuando uno de los extremos implementa IPSec.

  AH autentica el paquete interior IP completo más porciones seleccionadas de la cabecera IP exterior.

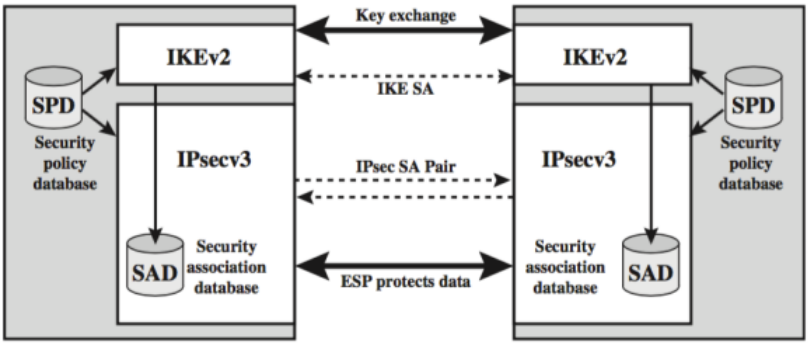
  ESP encripta el paquete IP interior completo.

  ESP con autenticación encripta el paquete IP interior completo y lo autentica.



10.1.4 Arquitectura general

Es importante la política de seguridad aplicada a cada paquete IP que transita desde origen hasta su destino. La política de IPSec está determinada principalmente por la interacción de dos bases de datos, la base de datos de asociación de seguridad (SAD) y la base de datos de política de seguridad.



Asociación de seguridad (SA)

Una SA es una relación unidireccional entre un emisor y un receptor que ofrece servicios de seguridad al tráfico que se transporta. Si se necesita una relación que haga posible un itnercambio bidireccional seguro, entonces se requieren dos asociaciones de seguridad, una en cada sentido. Los servicios de seguridad se suministran a una SA para que se use AH o ESP, pero no los dos.

Una AS se identifica por tres parámetros:

- **Índice de parámetros de seguridad (SPI)**
- **Dirección IP de destino**
- **Identificador del protocolo de seguridad**

Tiene asociados otros parámetros como un contador de número de secuencia, algoritmos criptográficos negociados, tiempo de vida,...

SAD

En cada implementación de IPSec hay una base de datos cuya funcionalidad debe estar presente en cada implementación de IPSec, pero la forma en que se proporciona dicha funcionalidad es decisión del implementador, de asociaciones de seguridad que define los parámetros asociados con cada SA. Una asociación de seguridad se define, normalmente, por los siguientes parámetros:

- **SPI:** número de 32 bits seleccionado por el receptor de un SA y que identifica el SA
- **AH information:** algoritmos de autenticación, claves, tiempo de vida y parámetros relacionados con AH
- **ESP information:** algoritmos de cifrado, de autenticación, vectores de inicialización, claves de autenticación y cifrado, tiempo de vida de las claves,...
- **Tiempo de vida de la SA**
- **IPSec Protocol Mode:** túnel o transporte
- **Path MTU:** cualquier MTU (maximum transmission unit) descubierto
- **Sequence number counter, sequence counter overflow,...**

SPD

El medio por el que el tráfico IP se relaciona con SAs específicas es la base de datos de políticas de seguridad. En su forma más simple una SPD contiene entradas, cada una de las cuales define un subconjunto de tráfico IP y señala una SA para ese tráfico.

Cada entrada de la SPD se define por un conjunto de valores de campos del protocolo IP y de protocolos de capas superiores, llamados **selectores**. Estos selectores se usan para filtrar tráfico saliente y establecer la correspondencia con una SA en particular. El procesamiento de tráfico saliente obedece a los siguientes pasos:

1. Comparar los valores de los campos adecuados del paquete con la SPD para encontrar una entrada coincidente, que señalará 0 o más SAs
2. Determinar la SA, si la hubiera, para este paquete y su SPI asociado
3. Llevar a cabo el procesamineto IPSec necesario

Los siguientes selectores determinan una entrada de la SPD:

- **Remote IP address**
- **Local IP address**
- **Next layer protocol:** el protocolo qiue transporte IP sobre el cual se quiere realizar la acción IPSec
- **Name:** identificador para el sistema operativo
- **Local and Remote ports**

10.1.5 Procesamiento de tráfico

IPSec se ejecuta paquete por paquete. Cuando se implementa IPSec cada paquete IP es procesado por IPSec antes de ser transmitido y cada paquete entrante también es procesado por IPSec después de su recepción y antes de su entrega a la capa superior.

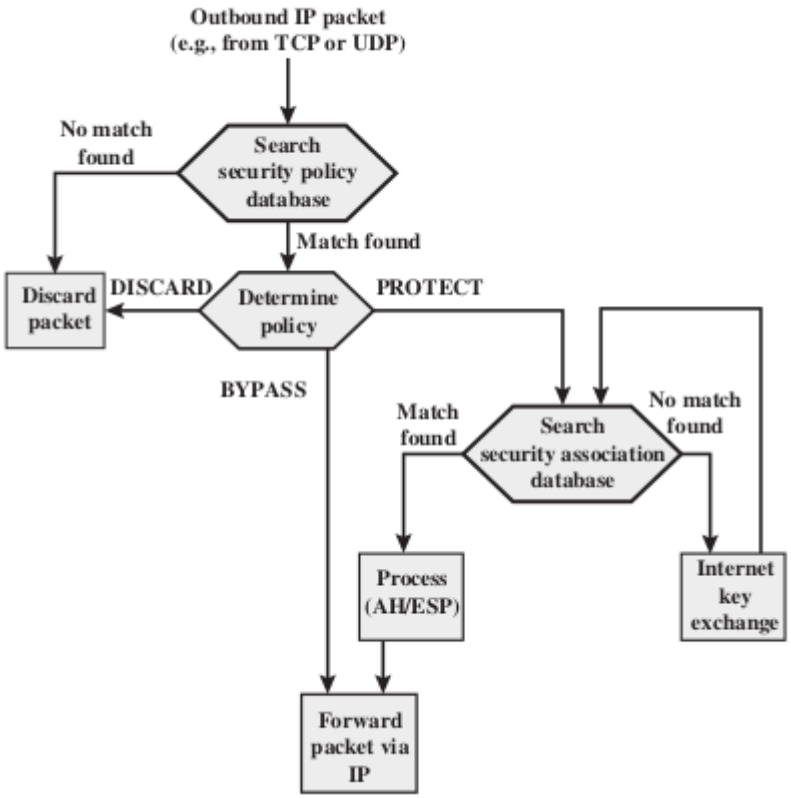
Paquetes salientes

Un bloque de datos de una capa superior, como TCP, se pasa a la capa de IP y se forma un paquete IP que consiste en una cabecera IP y un cuerpo IP. Ocurre lo siguiente:

1. IPSec busca en el SPD una coincidencia con este paquete
2. Si no encuentra ninguna, el paquete se descarta y se genera un mensaje de error
3. Si se encuentra una coincidencia, el procesamiento adicional se determina por la primera entrada coincidente en el SPD.

   Si la política para este paquete es DESECHAR, entonces el paquete se descarta.

   Si la política es BYPASS, entonces no hay más procesamiento IPSec. El paquete se reenvía a la red para su transmisión.
4. Si la política es PROTEGER, entonces se realiza una búsqueda de SAD para una entrada coincidente. Si no se encuentra ninguna entrada, se invoca IKE para crear una SA con las claves apropiadas y se crea una entrada en la SAD.
5. La entrada coincidente en el SAD determina el procesamiento de este paquete. Se puede realizar cifrado, autenticaicón o ambos, y se puede utilizar modo transporte o túnel.



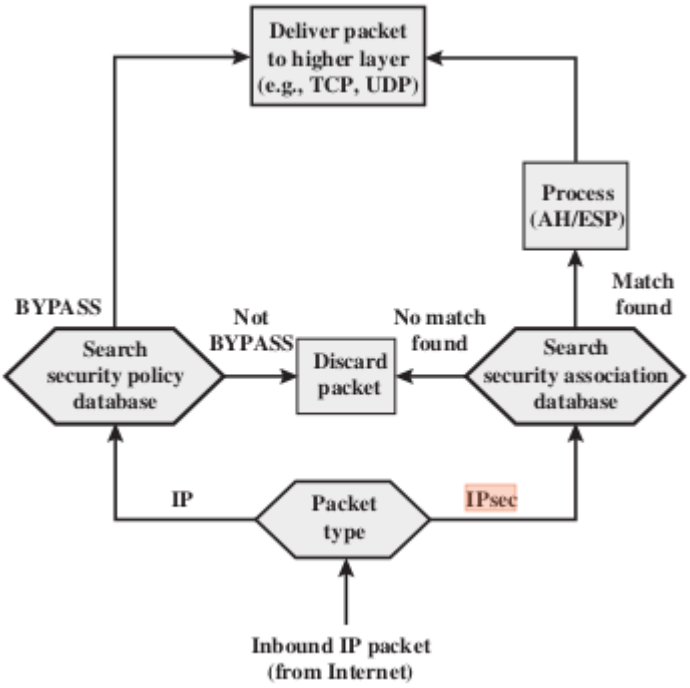
Paquetes entrantes

1. IPSec determina si se trata de un paquete IP no seguro o uno que tiene encabezados ESP o AH.

2. Si el paquete no es seguro, IPSec busca en el SPD una coincidencia con este paquete. Si la primera entrada coincidente tiene política de BYPASS, el encabezado IP se procesa y elimina y el cuerpo del paquete se entrega a la capa superior.

   Si la primera entrada coincidente tiene política PROTEGER o DESECHAR, o si no hay una entrada coincidente, el paquete se descarta.

3. Para un paquete seguro, IPSec busca el SAD. Si no se encuentra ninguna coincidencia, paquete se descarta. De lo contrario, IPSec aplica el procesamiento ESP o AH apropiado. Luego, el encabezado IP se procesa y se elimina y el cuerpo del paquete se encuentra a la siguiente capa.

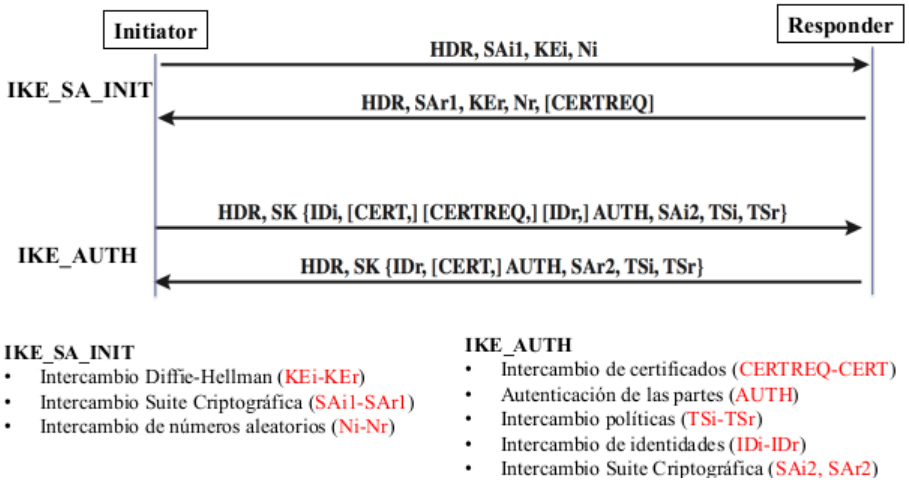


10.2 Protocolo Internet Key Exchange (IKE)

Para establecer una asociación de seguridad IPSec es necesario negociar algoritmos criptográficos, un proceso de autenticación de las partes comunicantes, distribuir el material criptográfico y seleccionar las cabeceras IPSec y el modo de operación. Para esto está el protocolo IKE.

IKEv2 implica el intercambio de mensajes en pares. Los primeros dos pares de intercambios se denominan **intercambios iniciales**. En el primer intercambio, los dos pares intercambian información sobre algoritmos criptográficos y otros parámetros de seguridad que están dispuestos a usar junto con los valores nonces y DH. El resultado de este intercambio es configurar una SA especial llamada IKE SA. Esta SA define los parámetros para un canal seguro entre los pares sobre los que tienen lugar los siguientes intercambios de mensajes. Por tanto, todos los intercambios de mensajes IKE posteriores están protegidos por cifrado y autenticación de mensajes.

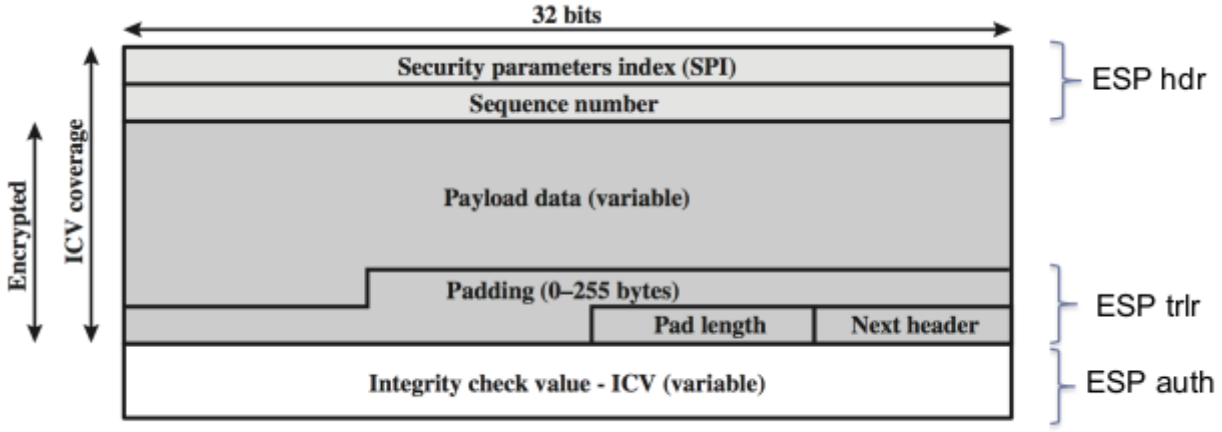
En el segundo intercambio, las dos partes se autentican entre sí y configuran un primer IPSec SA que se colocará en el SADB y se usará para proteger las comunicaciones ordinarias entre los pares. Por tanto, se necesitan cuatro mensajes para establecer la primera SA para uso general.



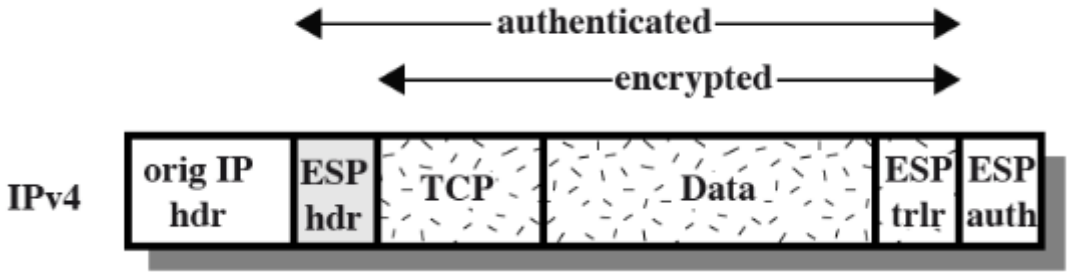
10.3 Cabeceras ESP (Encapsulating Security Payload)

El ESP se puede utilizar para proporcionar confidencialidad, autenticación de los datos de origen, integridad sin conexión, un servicio antireenvío.

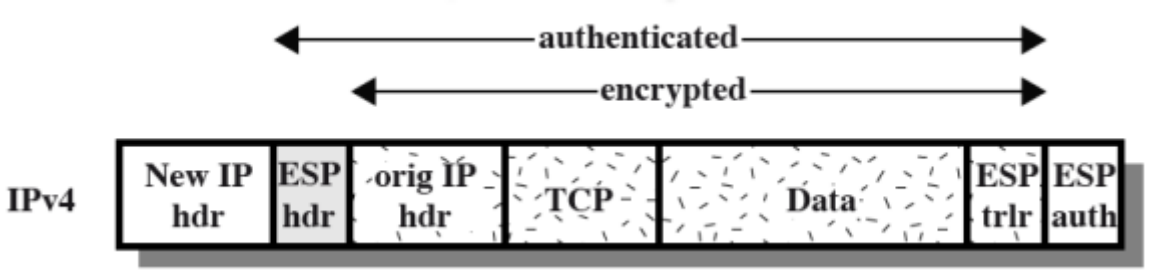
Selección opcional de servicios: solo integridad, solo cifrado o ambos.



El **modo de transporte ESP** se utiliza para cifrar y, opcionalmente, autenticar los datos transportados por IP. Para este modo, usando IPv4, el encabezado ESP se inserta en el paquete IP inmediatamente antes del encabezado de la capa de transporte y se coloca un finalizador ESP (con campos Relleno, Longitud del rello y Siguiente encabezado) después del paquete IP. Si se selecciona la autenticación, el campo Datos de autenticación ESP se agrega al final del paquete. Todo el segmento de nivel de transporte más el finalizador están cifrados. La autenticación cubre todo el texto cifrado más el encabezado ESP.



El **modo túnel ESP** se utiliza para cifrar un paquete IP completo. Para esto modo, el encabezado ESP es prefijado al paquete y se encripta todo el paquete interior más el finalizador ESP. Este método se puede utilizar para contrarrestar el análisis de tráfico.



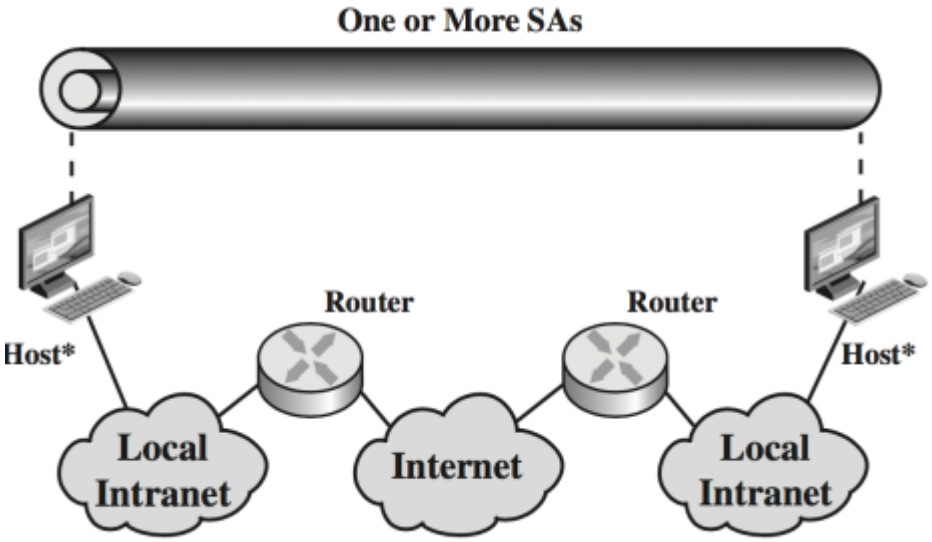
Como el encabezado IP contiene la dirección de destino y posiblemente las directivas de enrutamiento de origen y la informción de la opción salto por salto, no es posible simplmente transmitir el paquete IP cifrado prefijado por el encabezado ESP. Los routers intermedios no podrían procesar tal paquete. Por tanto, es ecesario encapsular todo el bloque con un nuevo encabezado IP que contendrá información suficiente para el enrutamiento pero no para el análisis del tráfico.

El modo transporte es adecuado para proteger las conexiones entre hosts que admiten la función ESP y el modo túnel es útil en una configuración que incluye un firewall u otro tipo de gateway de seguridad que protege una red confiable de redes externas.

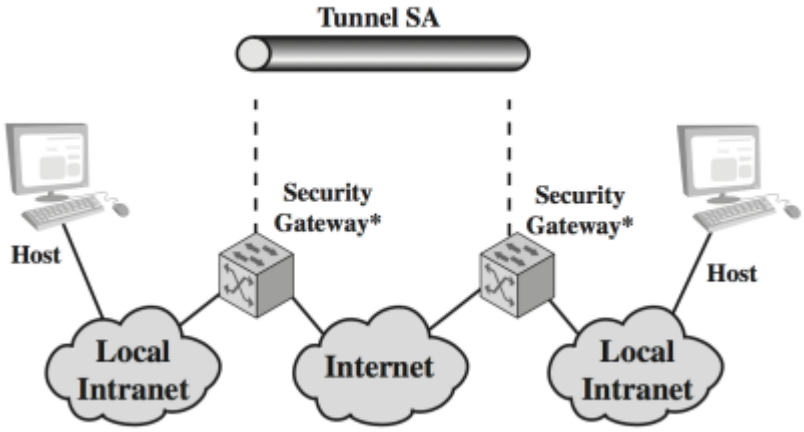
10.4 Escenarios de despliegue

1. Toda la seguridad se proporciona entre los sitemas finales que implementan IPSec. Para que los dos sistemas finales se comuniquen a través de una SA, deben compartir las claves secretas apropiadas. Entre las posibles combinaciones se encuentran:

    - AH en modo transporte
    - ESP en modo transporte
    - ESP seguido de AH en modo de transporte (un ESP SA dentro de un AH SA)
    - Cualquiera de las anteriores dentro de un AH o ESP en modo túnel



2. La seguridad se proporciona solo entre las puertas de enlace y ningún host implementa IPSec. Este caso ilustra el soporte de la red privada virtual simple. El documento de arquitectura de seguridad especifica que solo se necesita un solo túnel SA para este caso. El túnel podría admitir AH, ESP o ESP con autenticación. No se requieren túneles anidados, porque los servicios de IPSec se aplican a todo el paquete interno.



11 Protocolos para comunicaciones seguras: SSH

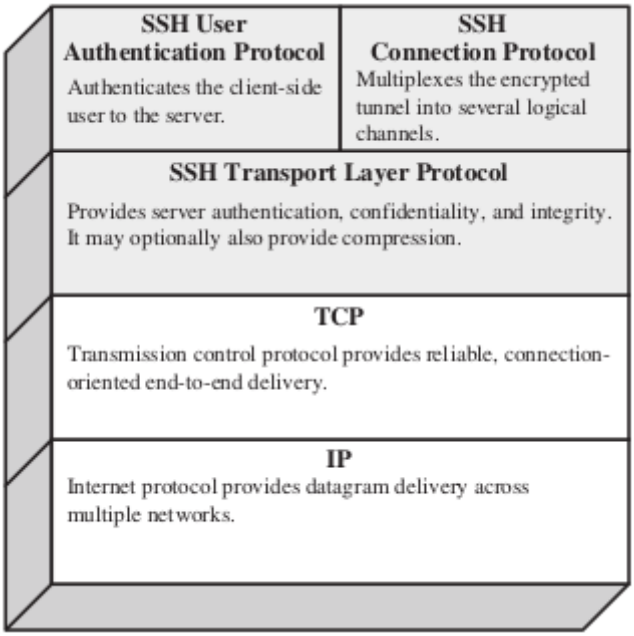
SSH es un protocolo para asegurar comunicaciones en la red diseñado para ser relativamente simple y barato de implementar. Se pensó para proporcionar una forma de login remoto regura para reemplazar TELNET y otros esquemas de login remotos que no proporcionaban sguridad. SSH también proporciona una capacidad más general cliente/servidor y puede ser usada para funciones de red como correo o transferencia de archivos.

SSH2 arregla cantidad de errores de seguridad en el esquema original.

Las aplicaciones cliente y servidor SSH están ampliamente disponiles para casi todos los sistemas operativos.

SSh se organiza en tres protocolos normalmente ejecutados sobre TCP:

- **Transport Layer Protocol:** proporciona autenticación del servidor, confidencialidad de datos e integridad de los datos con forward secrecy (esto quiere decir que si una clave se compromete ahora, no puede afectar a sesiones anteriores). Opcionalmente puede proporcionar compresión.
- **User Authentication Protocol:** autentica el usuario al servidor.
- **Connection protocol:** multiplexa varios canales de comunicación lógicos sobre una única conexión SSH subyacente



11.1 Transport Layer Protocol

11.1.1 Claves de Host

La autenticación del servidor ocurre en el transport layer, y se basa en que el servidor posea un par de claves pública/privada. Un servidor podría tener múltiples claves de host usando distintos algoritmos de encriptación asimétrica. La clave de host del servidor se usa durante el intercambio de claves para autenticar la identidad del host. El cliente debe tener conocimiento previo de la clave de host pública del servidor.

Hay dos modelos alternativos:

- El cliente tiene una base de datos local que asocia cada nombre de host con la correspondiente clave de host. El problema es que la base de datos puede ser muy costosa de mantener.
- La asociación nombre-clave del host es certificada por una CA de confianza. El cliente solo sabe la clave raíz del CA y puede verificar la validez de todas las claves de host certificadas por CAs aceptadas. Esta alternativa elimina el problema de la manutención de la base de datos. Por otro lado, cada clave de host debe estar apropiadamente certificada por una CA antes de hacer posible la autorización.

11.1.2 Intercambio de paquetes

Primero, el cliente establece una conexión TCP con el servidor. Esto se hace mediante el protocolo TCP y no es parte del Transport Layer Protocol. Una vez la conexión está establecida, el cliente y el servidor intercambian datos, llamados paquetes, en el campo de datos de los segmentos TCP. Cada paquete tiene el siguiente formato:

- **Packet length**
- **Padding length**
- **Payload:** contenido útil del paquete. Antes de la negociación del algoritmo, este campo no se comprime. Si se negocia compresión, en los siguientes paquetes sí se comprimirá.
- **Random padding:** una vez se negocia un algoritmo de encriptación, se añade este campo. Contiene una cantidad random de bytes de relleno de forma que la longitud total del paquete sea múltiplo del tamaño de un bloque de cifrado.
- **MAC:** si se negocia autenticación, este campo contiene el valor de la MAC.

Una vez que el algoritmo de encriptación se negocia, el paquete completo se encripta tras calcular el valor de la MAC.

El intercambio de paquetes consiste en una serie de pasos:

1. **Identification string exchange:** comienza con el cliente enviando un paquete con una cadena de identificación de la forma:

   $$SS - protoversion - sorftwareversion\ SP\ comments\ CR\ LF$$

   El servidor responde con su propia cadena de identificación. Estas cadenas se utilizan en el intercambio de claves DH.

2. **Negociación del algoritmo:** cada parte envía un SSH_MSG_KEXINIT que contiene la lista de algoritmos soportados en orden de preferencia del emisor. Hay una lista para cada tipo de algoritmo criptográfico. Los algoritmos incluyen el intercambio de claves, encriptación, algoritmo de MAC y algoritmo de compresión.

   Para cada categoría, el algoritmo elegido es el primer algoritmo de la lista del cliente que también es soportado por el cliente.

3. **Intercambio de claves:** solo se especifican dos versiones del intercambio de claves DH. Se siguen los siguientes pasos, donde se consideran ya negociados previamente p, q, g y el algoritmo hash:

   (a) El cliente genera un número x $(1 < x < q)$ y calcula $e = g^x \mod p$. El cliente envía e al servidor.

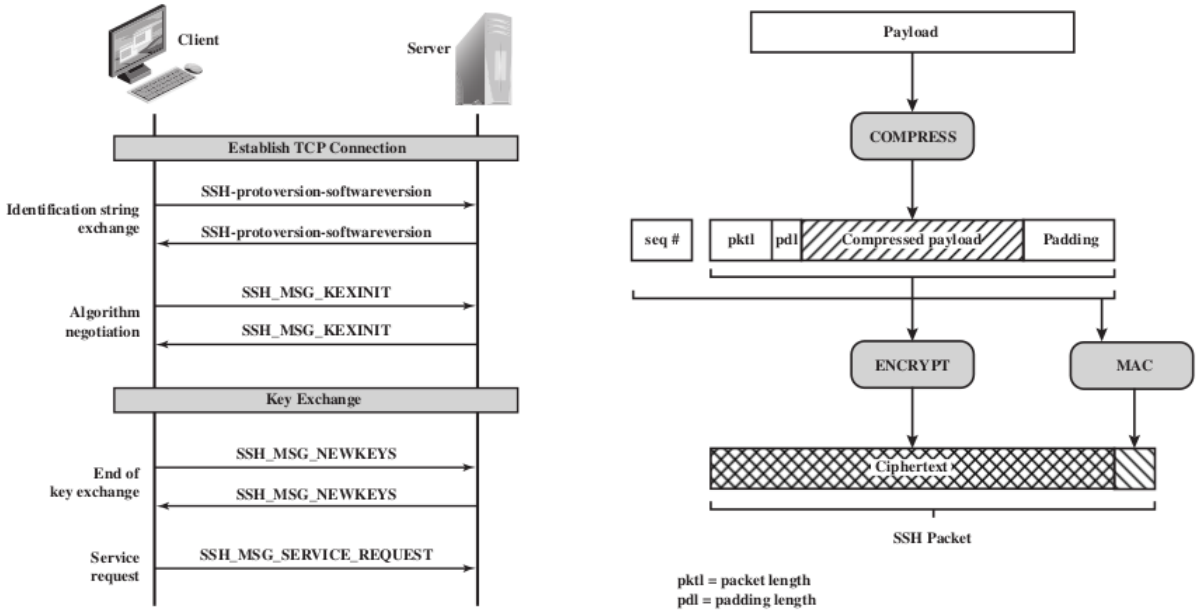
   (b) El servidor genera un número aleatorio y $(1 < y < q)$ y calcula $f = g^y \mod p$. El servidor recibe e y calcula $K = e^y \mod p$ y se calcula el hash, H, usando estos valores y algunos obtenidos anteriormente, como las cadenas de identificación. Firma H con su clave de host privada, obteniendo s. Envía al cliente su clave de host pública, f y s.

   (c) El cliente verifica que la clave de host pública es la del servidor (aunque puede seguir sin verificarlo, pero esto hace el protocolo inseguro contra ataques activos). El cliente calcula $K = f^x \mod p$, su propio hash, H, usando los mismos valores que el servidor, y verifica la firma de s en H.

Ahora las dos partes comparten una clave master K. El servidor ha sido autenticado al cliente. El valor hash H sirve como identificador de la sesión para esta conexión. Una vez calculado, el identificador de sesión no cambia, aunque se vuelva a realizar el intercambio de claves para obtener claves nuevas.

4. **Final del intercambio de claves:** se señaliza con el intercambio del mensaje SSH_MSG_NEWKEYS. En este punto, ambas partes pueden empezar a usar las claves generadas con K.

5. **Requerimiento de servicio:** el cliente envía un mensaje SSH_MSG_SERVICE_REQUEST para requerir autenticación de usuario o bien el Connection Protocol. Tras esto, todos los datos se intercambian como la carga útil de un paquete SSH Transport Layer Protocol, prtegido por encriptación y MAC.



11.1.3 Generación de claves

Las claves usadas para encriptación y MAC son generadas con la clave secreta compartida K, el valor hash obtenido en el intercambio de claves H, y el identificador de sesión (que es H a no ser que haya intercambios posteriores):

- IV inicial del cliente al servidor: $HASH\,(K \parallel H \parallel "A" \parallel session\_id)$
- IV inicial del servidor al cliente: $HASH\,(K \parallel H \parallel "B" \parallel session\_id)$
- Clave de encriptación del cliente al servidor: $HASH\,(K \parallel H \parallel "C" \parallel session\_id)$
- Clave de encriptación del servidor al cliente: $HASH\,(K \parallel H \parallel "D" \parallel session\_id)$
- Clave de integridad del cliente al servidor: $HASH\,(K \parallel H \parallel "E" \parallel session\_id)$
- Clave de integridad del servidor al cliente: $HASH\,(K \parallel H \parallel "F" \parallel session\_id)$

11.2 User Authentication Protocol

Proporciona los medios por los que el cliente se autentica al servidor.

11.2.1 Tipos de mensajes y formatos

Hay tres tipos de mensajes que siempre se usan en el UAP.

La peticiones de autenticación del cliente tienen el formato:

byte	$SSH\_MSG\_USERAUTH\_REQUEST$ (50)
string	*user name*
string	*service name*
string	*method name*
...	*method specific fields*

Si el servidor (1) rechaza la petición de autenticación o (2) la acepta pero requiere algún método de autenticación adicional, el servidor envía un mensaje con el formato:

byte	$SSH\_MSG\_USERAUTH\_FAILURE$ (51)
$name-list$	*authentications that can continue*
boolean	*partial success*

11.2.2 Intercambio de mensajes

El intercambio de mensajes tiene los siguientes pasos:

1. El cliente envía un SSH_MSG_USERAUTH_REQUEST con ningún método requerido.
2. El servidor comprueba si el nombre de usuario es válido. Si no, el servidor devuelve SSH_MSG_USERAUTH_ con un partial success a false. Si el nombre de usuario es válido, pasa al paso 3.
3. El servidor devuelve SSH_MSG_USERAUTH_FAILURE con una lista de uno o más métodos de autenticación a usar.
4. El cliente elige uno de los métodos aceptables de autenticación y envía un SSH_MSG_USERAUTH_REQUES con ese método y sus campos específicos. En este punto, puede haber una secuencia de intercambios para llevar a cabo el método.
5. Si la autenticación es exitosa y se requieren más métodos de autenticación, el servidor vuelve al paso 3, usando un partial success a true. Si la autenticación falla, el servidor vuelve al paso 3, con el partial success a false.
6. Cuando todos los métodos requeridos de autenticación han sido completados con éxito, el servidor envía un mensaje MSG_USERAUTH_SUCCESS y el UAP termina.

11.2.3 Métodos de autenticación

- **publickey:** el cliente envía un mensaje al servidor que contiene la clave pública del cliente, con el mensaje firmado con la clave privada del cliente. Cuando el servidor recibe el mensaje, comprueba si la clave suministrada es aceptable para autenticación y, si es así, comprueba que la firma es correcta.
- **password:** el cliente envía un mensaje que contiene una contraseña en texto plano, que es protegida con encriptación por el TLP.
- **hostbased:** la autenticación se hace por el host del cliente en lugar de por el propio cliente. Por tanto, un host puede dar soporte a varios cliente y proveer autenticación a todos ellos.

11.3 Connection protocol

Se ejecuta sobre el SSH TLP y asume que una conexión segura y autenticada está en curso. Esta conexión segura, llamada **túnel**, se usa por el Connection Protocol para multiplexar cantidad de canales lógicos.

11.3.1 Mecanismo de canales

Todos los tipos de comunicación que usan SSH se les da soporte usando canales separados. Cada parte puede abrir un canal. Para cada canal, cada parte asocia un número de canal único, que no tiene por qué coincidir en ambos extremos. El flujo de los canales se controla usando un mecanismo de ventana. No se envían datos a un canal hasta que se recibe un mensaje que indica que hay espacio de ventana disponible.

La vida de un canal tiene tres etapas:

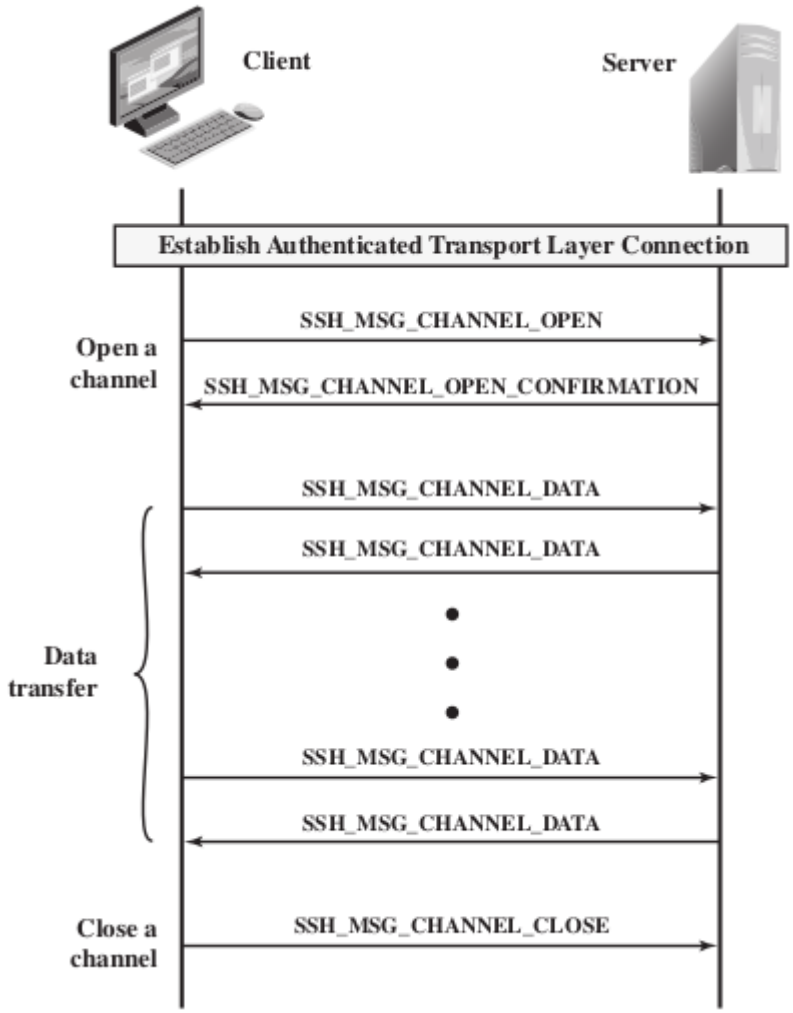
1. **Apertura del canal:** cuando alguna parte quiere abrir un nuevo canal, le asigna un número al canal y envía un mensaje de la forma:

byte	$SSH\_MSG\_CHANNEL\_OPEN$
string	*channel type*
*uint*32	*sender channel*
*uint*32	*initital window size*
*uint*32	*maximum packet size*
...	*channel type specific data follows*

   Si el lado remoto puede abrir el canal, devuelve un SSH_MSG_CHANNEL_OPEN_CONFIRMATION, que incluye el número de canal del emisor, el del receptor, y valores del tamaño la ventana y tamaño del paquete para tráfico entrante. Si no, el lado remoto envía un SSH_MSG_CHANNEL_OPEN_FA con un código que expresa la razón del fallo.

2. **Transferencia de datos:** se lleva a cabo usando un mensaje SSH_MSG_CHANNEL_DATA, que incluye el número de canal del receptor y un bloque de datos.

3. **Cierre del canal:** cuando alguna parte quiere cerrar el canal, envía un SSH_MSG_CHANNEL_CLOSE, que incluye el número de canal del receptor.



11.3.2 Tipos de canales

- **session:** para ejecutar programas de forma remota
- **x11:** se refiere al sistema X Window
- **forwarded-tcpip:** esto es port forwarding remoto
- **direct-tcpip:** para port forwarding local

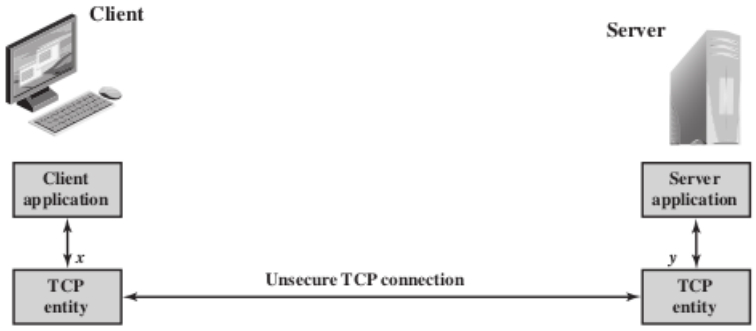
11.4 Port forwarding

Es una de las mejores utilidades de SSH. Proporciona la capacidad de convertir cualquier conexión TCP insegura en una conexión SSH segura.

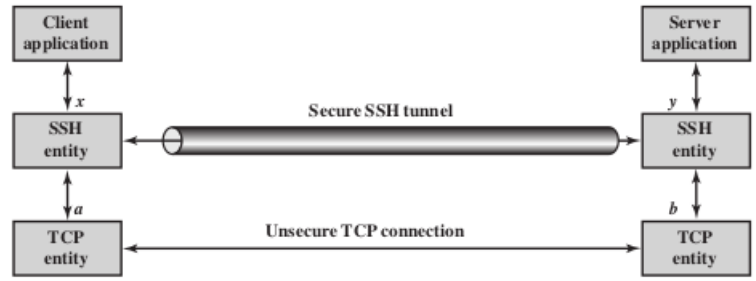
Un **puerto** es un identificador de un usuario de TCP. Así, cualquier aplicación que se ejecuta sobre TCP tiene un número de puerto asociado. El tráfico TCP entrante se entrega a la aplicación correspondiente basándose en el número de puerto.

Supongamos ahora que tenemos una aplicación cliente identificada por el número de puerto x y una aplicación de servidor identificada con el número de puerto y. En algún punto, la aplicación cliente invoca a la entidad TCP local y requiere una conexión con el servidor en el puerto y. La entidad TCP local negocia una conexión TCP con la entidad TCP remota, de tal forma que establece una conexión entre el puerto local x y el remoto y.

Para asegurar esta conexión, SSH se configura de tal forma que el SSH TLP establece una conexión TCP entre el cliente SSH y entidades de servidor, con números de puerto a y b, respectivamente. Un túnel SSH seguro se establece sobre esta conexión TCP. El tráfico del cliente en el puerto x se redirige a la entidad SSH local y viaja a través del túnel hasta que la entidad SSH entrega los datos al servidor en el puerto y. El tráfico en la otra dirección se redirige de forma similar.



(a) Connection via TCP



(b) Connection via SSH tunnel

Hay dos tipos de port forwarding:

1. **Forwarding local:** permite al cliente montar un proceso secuestrados. Este interceptará tráfico del nivel de aplicación seleccionado y lo redigirá de una conexión TCP insegura a un túnel SSH seguro. SSH se configura para escuchar en puertos seleccionados. SSH toma todo el tráfico usando un peurto seleccionado y lo envía a través de un túnel SSH. En el otro extremo, el servidor SSH envía el tráfico entrante al puerto destino dictado por la aplicación cliente.

2. **Forwarding remoto:** el cliente SSH del usuario actúa en nombre del servidor. El cliente recibe el tráfico con un determinado número de puerto destino, coloca el tráfico en el puerto correcto y lo envía al destino que el usuario desee.