Tema 4. Correo electrónico

Ingeniería de Servicios

2023-2024

T4.1 Introducción

4.1 Introducción

Servicio a proporcionar:
Envío de mensajes de texto (posteriormente cualquier tipo de fichero) a otro usuario en otra máquina, de forma asíncrona.

Las soluciones iniciales

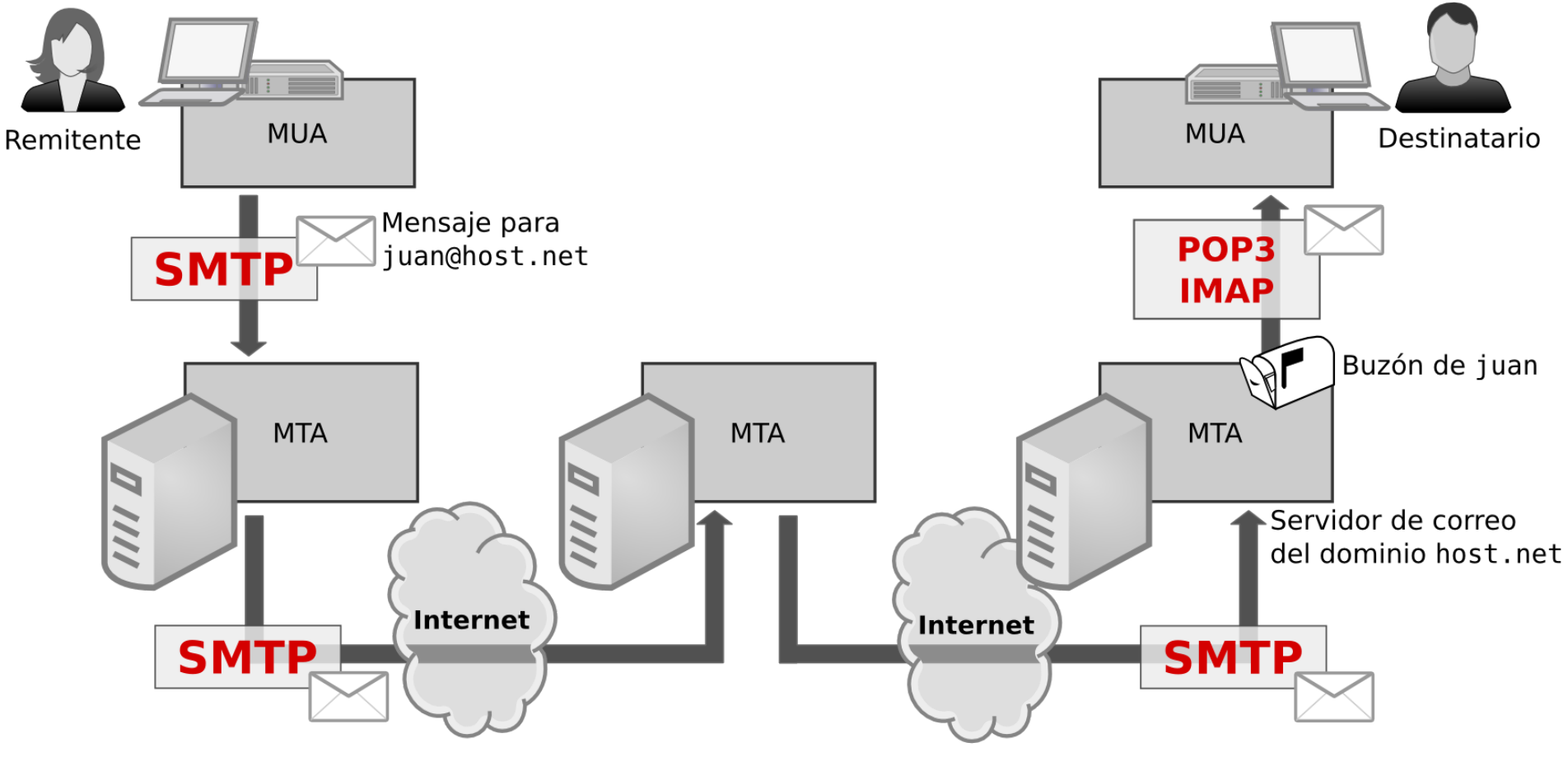
- En la misma máquina. Copia a un archivo específico: el *buzón*.
- En otra máquina. Protocolo para transmitir el mensaje a la otra máquina y almacenarlo en el buzón remoto: SMTP.

Entidades en este proceso

- **El remitente**. Usuario que escribe y envía el mensaje.
- **El destinatario**. Usuario que recibe el mensaje.
- **MUA** (*Message User Agent*). El programa que usa el remitente para enviar, y el destinatario para recibir.
- **MTA** (*Message Transfer Agent*). El programa que recibe el mensaje del MUA del remitente, y lo transmite a otro MTA en una máquina a la que el destinatario tiene acceso.
- **Buzón**. Archivo o carpeta en donde se almacenan los mensajes para ser leidos.

¿Cómo accede el destinatario al buzón? ¿Dónde está alojado?

Arquitectura general simplificada



Flujo del correo en el diagrama anterior

1. El remitente escribe el mensaje en su MUA
2. Cuando "envía", el MUA se pone en contacto con un MTA (`SMTP`)
3. Según su configuración, éste puede ponerse en contacto directamente con el MTA destino, o usar otros MTA intermedios (*relays*)
4. El mensaje llega al MTA destino (`SMTP`)
5. Se comprueba que el usuario existe y una copia del mensaje se deja en su buzón
6. El destinatario usa su MUA para descargar mensajes del buzón a su dispositivo (`POP3` o `IMAP`)
7. El destinatario usa su MUA para leer el mensaje.

Cada transmisión del mensaje puede encontrar problemas, retrasos, etc. SMTP mantiene colas y hace reintentos.

Ejemplos de agentes

- **MUA**
  - `mail`, `mailx`, `pine`: de línea de comandos para Unix, accedían directamente al buzón pues se ejecutaban en el MTA destino.
  - Microsoft Outlook (windows)
  - Mozilla Thunderbird (multiplataforma)
  - Evolution (Linux, Gnome, GUI)
  - Lectores de correo para móviles (Gmail, Apple Mail)
  - Interfaces Web (Gmail, Yahoo, iCloud)
- **MTA**
  - `sendmail` el más antiguo, para unix
  - `Postfix`, `qmail` muy populares hoy día
  - Microsoft Exchange, para windows

Los MTA *Relay*

En muchas intranets:

- Los mensajes entre máquinas de la intranet usan MTAs que se comunican directamente entre sí
- Pero los mensajes dirigidos al exterior se envían a un MTA *relay*

Esto simplifica la configuración, y permite que el *relay* haga funciones de *proxy* (filtrado de correos, eliminación de *spam* o virus, etc.)

También hace más fácil para el MTA destino "confiar" en el MTA origen. Más sobre este tema más adelante.

El mensaje

- El contenido del mensaje ha de ser ASCII puro de 7 bits.
- Su estructura es similar a la de HTTP:
  - Comienza con varias líneas de **cabecera**
  - Cada una de la forma `Campo: valor\r\n`
  - Una línea en blanco termina el encabezado (`\r\n\r\n`)
  - Cualquier número de líneas adicionales es el **cuerpo**

4.2 Cabeceras, mensajes, buzones

4.2 Cabeceras (RFC 4021)

Algunas cabeceras las pone el cliente de correo (MUA). *Todas ellas son opcionales*.

- `From:` remitente
- `To:` destino
- `Reply-To:` quién debe recibir la respuesta
- `Cc:` *carbon copy*
- `Bcc:` *blind* carbon copy
- `Subject:` asunto
- `Date:` fecha y hora de la salida del mensaje
- `Message-Id:` identificador único

Cabeceras (RFC 4021)

Otras son añadidas por los MTA por donde pasa el mensaje:

- `Date`, `From`, `To`, `Message-Id`
  Suelen ser añadidas por el primer MTA si no venían ya fijadas desde el MUA.
- `Received:`
  Es añadida por cada MTA atravesado por el mensaje, y registra qué MTA lo recibe, de dónde procede y la fecha y hora.
- `Return-path:`
  Es añadida por el último MTA (destino). Contiene cómo hacer llegar este mensaje de nuevo a su origen (si no se puede entregar, por ejemplo)

Ejemplo de cabeceras reales

```
Received: from AM2PRD0411HT005.eurprd04.prod.outlook.com
 (10.255.163.40) by AMSPRD0410HT003.eurprd04.prod.outlook.com
 (10.255.41.38) with Microsoft SMTP Server (TLS)
 id 14.16.371.2; Wed, 23 Oct 2013 16:58:25 +0000

[... más cabeceras Received omitidas ...]
Received: from CO1EHSMHS030.bigfish.com (unknown [10.243.78.252])
 by mail150-co1.bigfish.com (Postfix) with ESMTP
 id 9AE9B400040; Wed, 23 Oct 2013 16:58:19 +0000 (UTC)
Received: from micorreo.uniovi.es (156.35.11.134)
 by CO1EHSMHS030.bigfish.com (10.243.66.40) with Microsoft SMTP
 Server (TLS) id 14.16.227.3; Wed, 23 Oct  2013 16:58:19 +0000
Received: from [156.35.173.92] (172.22.11.124) by micorreo.uniovi.es
 (172.22.11.143) with Microsoft SMTP Server (TLS)
 id 14.3.146.0; Wed, 23 Oct 2013 18:58:09 +0200
Message-ID: <52680018.50309@uniovi.es>
Date: Wed, 23 Oct 2013 18:58:00 +0200
From: calidad.epigijon@uniovi.es
Reply-To: calidad.epigijon@uniovi.es
User-Agent: Mozilla/5.0 (Windows;U;Windows NT 6.0;es-ES;rv:1.9.2.28)
 Gecko/20120306 Thunderbird/3.1.20
To: destinatarios-no-revelados:;
Subject: Recomendaciones importantes
Content-Type: multipart/alternative;
    boundary="------------050408000104060809020001"
Return-Path: calidad.epigijon@uniovi.es
```

El cuerpo del mensaje

El cuerpo ha de ser una secuencia de líneas compuestas por caracteres ASCII de 7 bits

- ¿Cómo incluir caracteres no ASCII?
- ¿Cómo incluir tipos de letra, negrita, etc?
- ¿Cómo incluir imágenes?
- ¿Cómo incluir adjuntos?

Gracias a MIME, que veremos después, que codifica todo lo anterior como secuencias de líneas ASCII de 7 bits.

Los buzones

> Su formato no está estandarizado.

Básicamente, cada implementación de MTA (y los MUA que almacenan copia local del mensaje) tienen su propio formato.

Los más frecuentes:

- Cada mensaje es un fichero, todos en una carpeta (formato *Maildir*)
- Todos los mensajes van concatenados en un solo fichero (formato *mbox*)
- Formato propietario (ej: `.pst` de Outlook)

4.3. MIME

MIME: *Multipurpose Internet Mail Extensions*

- Los primeros correos eran solo texto (y ASCII)
- El estándar MIME permite transmitir cualquier otro contenido binario en el *cuerpo* del mensaje.
- MIME también permite varias *partes* en un mismo mensaje (adjuntos, diferentes versiones del mensaje, como texto plano o HTML)
- Otras extensiones resolvieron el uso de caracteres no-ASCII en cabeceras (ej, en el *Subject*)

Cabeceras relacionadas con MIME

- `MIME-version:` Especifica la versión de MIME usada para codificar el mensaje
- `Content-type:` Especifica el tipo de contenido. Casos habituales:
  - `text/plain` Texto sin formato
  - `text/html` Texto formateado como HTML
  - `multipart/mixed` o `multipart/alternative`, el mensaje contiene varias partes (cada una con su propio `Content-type`)
- `Content-Transfer-Encoding:` Especifica cómo van codificados los contenidos. Casos habituales:
  - `7bit` ASCII puro
  - `quoted-printable` caracteres de 7 bits tal cual, los de 8 bits codificados de forma especial (veremos después)
  - `base64` todo el contenido se considera binario y se codifica en base64

Multiparte

Un correo puede estar formado de varias partes o componentes:

- Alternativas: el MUA elegirá una de ellas para mostrar al usuario, según sus capacidades. P.ej, una parte es HTML y la otra es texto plano.
- Mezcladas: las partes del mensaje se muestran todas a la vez. Ejemplo, los adjuntos o imágenes incrustadas en el HTML que viajen con el mensaje.

Cómo se separan las partes:

- Mediante una cadena de texto arbitraria
- Que se declara como parte del `Content-Type`
- Y que después aparece en el cuerpo precedida de `--`
- Cada parte tiene la estructura y sintaxis de otro email completo (con cabeceras opcionales)
- Tras la última parte la cadena aparece otra vez con `--` al principio y al final

Ejemplo de mensaje multiparte

```
...[otras cabeceras omitidas...]
To: usuario@correo.com
Subject: Welcome to FooBar
Mime-Version: 1.0
Content-Type: multipart/alternative;
 boundary="==_mimepart_523c348923bd_769632994811581d";
 charset=UTF-8
Content-Transfer-Encoding: 7bit

_mimepart_523c348923bd_769632994811581d
Mime-Version: 1.0
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 7bit

Hi usuario
Welcome to Foobar!

                                                    [...sigue...]
```

...sigue el ejemplo

```
_mimepart_523c348923bd_769632994811581d
Mime-Version: 1.0
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: 7bit

<html>
<head>
<title></title>
</head>
<body style="font-size: 12px; font-family: 'Arial; color: #333333;">
Hi usuario, <br> Welcome to Foobar!<br>
</body>
</html>

_mimepart_523c348923bd_769632994811581d--
```

Codificación *quoted-printable*

- En esta codificación los caracteres ASCII imprimibles pasan sin modificación
- Y los no-ascii o no imprimibles se sustituyen por `=XX` siendo `XX` el código (hexadecimal) del byte.
- El espacio puede representarse como `=20` o por el propio espacio
  - Salvo si aparece al final de la línea, que debe ser `=20`
- El signo igual se representa como `=3D`
- Las líneas se limitan al 76 caracteres
  - Para preservar líneas más largas se puede añadir `=` al final

Observa que un carácter como 'ñ' debe pasarse a bytes según alguna codificación (utf8, latin1) antes de pasar a `quoted-printable`

Ejemplo de codificación *quoted-printable*

El siguiente texto (en una sola línea):

> Si en python tenemos la asignación x=[1,2,3] entonces x es una lista y len(x) valdrá 3.

asumiendo codificación UTF-8, lo que implica que `ó` se codificará con la pareja de bytes `0xc3`, `0xb3` y `á` con la pareja `0xc3`, `0xa1`, entonces la cadena anterior codificada como `quoted-printable` sería:

```
Si en python tenemos la asignaci=C3=B3n x=3D[1,2,3] entonces x es una lista=
 y len(x) valdr=C3=A1 3.
```

Codificación *base64*

- El contenido original se agrupa en grupos de 3 bytes
  - Cada grupo tiene por tanto 24 bits
  - Y se divide en trozos de 6 bits (codifican valores entre 0 y 63)
  - Cada trozo de 6 bits se convierte en un símbolo, según la correspondencia:
    - 0-25 → "A"-"Z"
    - 26-51 → "a"-"z"
    - 52-61 → "0"-"9"
    - 62 → "+"
    - 63 → "/"
- Si el último grupo tiene sólo 1 byte, se termina por `==` y si sólo dos, por `=`
- La secuencia de caracteres resultante se divide en líneas de 76

Ejemplo *base64*

Texto a codificar: "Eñe"

1. En UTF-8 será la secuencia de bytes: `0x45`, `0xc3`, `0xb1`, `0x65`
2. El primer grupo de 3 bytes es por tanto: `0x45c3b1`
3. En binario es `010001011100001110110001`
4. Produce cuatro trozos de 6 bits: `010001`, `011100`, `001110`, `110001`
5. Que numéricamente son: 17, 28, 14 y 49
6. Según la conversión antes vista son los símbolos: "R", "c", "O" y "x"
7. Análogo para el segundo grupo, que sólo tiene un byte (0x65)

   Resultado: `RcOxZQ==` (observar el `==` de "relleno")

4.3 SMTP

SMTP (*Simple Mail Transfer Protocol*)

- Descrito en RFC 821 (extensiones posteriores en RFC 5321, 2008)
- Va sobre TCP
- El cliente puede air autenticado o no
- La conexión puede ir cifrada (TLS) o no
- Dos usos diferentes:
  1. Entre un MUA y un MTA (normalmente requiere autenticación)
  2. Entre dos MTA (normalmente no va cifrado)

Características del protocolo

- Orientado a mensajes de texto (tipo HTTP)
  - Petición: comando + parámetros
  - Respuesta: código + descripción textual
  - Terminador de línea: CRLF (`\r\n`)
- Algunas peticiones pueden requerir más líneas después de la que lleva el comando
- Algunas respuestas también pueden ser varias líneas

Comandos básicos (enviados por el cliente)

Comando	Parámetros	Significado
`HELO`	`nombre_nodo_origen`	Saludo del cliente
`MAIL`	`FROM:` remitente	Solicita envío de un correo, desde el remite dado
`RCPT`	`TO:` destino(s)	Indica a quién enviar el mensaje
`DATA`	--	Tras este comando va el mensaje
`QUIT`	--	Finaliza la conexión SMTP

El mensaje que sigue a `DATA` contiene las cabeceras (terminadas por `\r\n`) y el cuerpo. Se considera finalizado al recibir una línea que contiene tan solo un punto (el terminador es por tanto `\r\n.\r\n`)

Respuestas habituales (enviadas por el servidor)

Código	Significado
220	OK (servidor listo)
250	OK (servicio completado, mail aceptado)
354	OK, esperando el contenido de `DATA`
421	No puedo realizar el servicio. Cierro la conexión
4xx	No pudo completarse el comando
5xx	Error en el comando, o en el protocolo, o en el acceso al buzón, etc.

Diálogo típico:

El cliente es el ordenador `pc23.ejemplo.com`. El servidor SMTP es el ordenador `smtp.servidor.com`.

Tras establecer conexión TCP (puerto 25):

```
Servidor: 220 smtp.servidor.com service ready
Cliente > HELO pc23.ejemplo.com
Servidor: 250 OK
Cliente > MAIL FROM:<usuario@ejemplo.com>
Servidor: 250 OK
Cliente > RCPT TO:<destino@dominio>
Servidor: 250 OK
Cliente > RCPT TO:<otro.destino@dominio>
Servidor: 250 OK
Cliente > DATA
Servidor: 354 Start mail input; end with <CRLF>.<CRLF>
Cliente > Blah blah (cabeceras)
Cliente > Blah blah (más cabeceras)
Cliente >   (línea en blanco, fin de cabeceras)
Cliente > Blah blah blah (cuerpo del mensaje)
Cliente > etc. etc.
Cliente > .  (una línea que contiene sólo un punto)
Servidor: 250 OK
Cliente > QUIT
Servidor: 221 servidor.ejemplo.com closing transmission channel
```

El "sobre" (*envelope*) SMTP

Para transportar el mensaje a su destino (y para poder devolverlo a su remitente en caso de problemas), el protocolo SMTP debe conocer:

- Quién es el destino (lo obtiene del comando `RCPT TO:`)
- Quién es el origen (lo obtiene del comando `MAIL FROM:`)

Esta información **no forma parte del mensaje** (tampoco son las cabeceras), pero viaja con él. Es el llamado ***envelope***.

Sin embargo, el mensaje, en sus cabeceras también tiene un campo `From:` y otro `To:`

¿Por qué esta redundancia? ¿Y si son diferentes las cabeceras del *envelope*?

Pista: Piensa en el campo `Bcc:`

Problemas de seguridad

En el diálogo anterior hay muchos detalles conflictivos:

- ¿Puede el cliente desde el dominio `ejemplo.com` usar el servidor SMTP de otro dominio (en el ejemplo: `servidor.com`)?
  - Sí puede. Si el servidor lo admite el protocolo no dice nada en contra.
- ¿Puede mentir el cliente en el `HELO` y ponerse un nombre falso?
  - Puede, pero el servidor de todas formas conoce su IP a través del socket
- ¿Puede el cliente poner un `FROM` diferente en el comando `MAIL` del que luego va en el campo `From:` del mensaje?
  - Puede. El destinatario final verá las del mensaje (y si sabe cómo, puede examinar la cabecera `Return-path` donde figurará el remite dado en `MAIL FROM`)

Sigue

- ¿Puede el cliente poner un `FROM` de un usuario que no existe en `ejemplo.com`?
  - Puede. El servidor no conoce usuarios. Cuando el remitente intente responder al correo lo hará a una dirección inexistente.
- ¿Puede el cliente poner un `FROM` completamente falso, de otro usuario desde otro dominio?
  - Puede. El protocolo SMTP no lo impide. Esto permite enviar correo "en nombre de" otra persona, lo que podría ser legítimo en algunos casos.

 ¿No es todo esto demasiado "abierto"?

Sí, y es un campo abonado para *spammers*

Algunas soluciones

- SPF. Es un mecanismo que el servidor SMTP puede usar para decidir si admite el remite proporcionado en `MAIL FROM` o lo rechaza. Aceptará el mensaje si:
  - El dominio del remite tiene una entrada en el DNS
  - Esa entrada tiene un campo `TXT` que lista una serie de IPs permitidas
  - La IP del cliente está en su lista de IPs permitidas

Por ejemplo, un servidor SMTP para `uniovi.es` sólo admite remites que terminen en `uniovi.es` y que provengan de IPs de dentro de la Universidad.

- SMTP-AUTH. Es una extensión de SMTP que permite autenticar al remitente con un nombre y clave, u otros métodos.

Extensiones de SMTP

El protocolo básico se ha extendido desde su origen para soportar autenticación, cifrado y otras características.

Un cliente puede consultar qué extensiones tiene un servidor si usa `EHLO` en lugar de `HELO`

Algunas extensiones importantes:

- **Transporte cifrado**. Si el servidor lo soporta, el cliente puede enviar el comando `STARTTLS`, y "envolver" el socket de comunicación con una biblioteca SSL.

  El resto de comandos y respuestas irán sobre el transporte cifrado.

Extensiones de SMTP

Algunas extensiones importantes:

- **Autenticación del cliente**. El cliente puede usar el comando `AUTH <metodo>` para autenticarse usando el método en cuestión. Posibles métodos:
  - `LOGIN`. Las dos líneas siguientes envían nombre y clave respectivamente, ambas codificadas en Base64.
  - `PLAIN`. En la línea siguiente va nombre y clave, ambas sin codificar y terminadas con un ASCII nulo.
  - Otros: CRAM-MD5, OAuth, OAuth2, etc.

Ejemplo de diálogo entre cliente y servidor extendido

```
Cliente conecta al puerto 25 o el 587 (si va a usar STARTTLS)

Servidor: 220 mx.google.com ESMTP r48sm20049827eev.14 - gsmtp
Cliente > EHLO mi.maquina.mi.dominio
Servidor: 250-mx.google.com at your service, [93.156.102.17]
Servidor: 250-SIZE 35882577           <-- Esto es la lista de
Servidor: 250-8BITMIME                <-- extensiones que
Servidor: 250-STARTTLS                <-- soporta el servidor
Servidor: 250-ENHANCEDSTATUSCODES
Servidor: 250 CHUNKING
Cliente > STARTTLS
Servidor: 220 2.0.0 Ready to start TLS
```

A partir de este punto, el resto del diálogo iría cifrado sobre SSL.

La diapositiva siguiente muestra cómo continuaría la conversación

Lo siguiente iría cifrado. Mostramos aquí su versión descifrada

```
Cliente > EHLO mi.maquina.mi.dominio  <-- Repite saludo
Servidor: 250-SIZE 35882577
Servidor: 250-8BITMIME
Servidor: 250-AUTH LOGIN PLAIN XOAUTH XOAUTH2 PLAIN-CLIENTTOKEN
Servidor: 250-ENHANCEDSTATUSCODES
Servidor: 250 CHUNKING
Cliente > AUTH LOGIN
Servidor: 334 VXNlcm5hbWU6                <-- Esto es "Username:"
Cliente > amxkaWF6QHVuaW92aS5lcw==         <-- Nombre de usuario
Servidor: 334 UGFzc3dvcmQ6                 <-- Esto es "Password:"
Cliente > ZXN0YW5vZXNtaWNsYXZl             <-- Contraseña
Servidor: 235 2.7.0 Accepted
Cliente > MAIL FROM: <usuario@gmail.com>
Servidor: 250 2.1.0 OK i1sm20257012eeg.0 - gsmtp
Cliente > RCPT TO: <jldiaz@uniovi.es>
Servidor: 250 2.1.5 OK i1sm20257012eeg.0 - gsmtp
Cliente > DATA
Servidor: 354  Go ahead i1sm20257012eeg.0 - gsmtp
Cliente > (cabeceras)
Cliente >
Cliente > (mensaje)
Cliente > .
Servidor: 250 2.0.0 OK 1382720990 i1sm20257012eeg.0 - gsmtp
```

4.4 POP3

POP3 (*Post Office Protocol*)

Funcionalidad básica:

- Conectarse al MTA que tiene el buzón de mensajes
- Consultar cuántos mensajes hay en el buzón
- Descargar mensajes seleccionados
- Borrar mensajes del buzón

Es un protocolo de tipo comando/respuesta, similar a SMTP en la sintaxis.
Puerto estándar: 110.

El protocolo pasa por los estados:

- Autorización
- Transacción
- Actualización

POP3: Estado de Autorización

- Inicialmente sólo contaba con la autorización basada en los comandos `USER` y `PASS`
  - El cliente emitía ambos en secuencia
  - El nombre de usuario y la clave van "en claro"
- Posteriormente se añade el comando `APOP`
  - El servidor, incluye en el saludo inicial un *timestamp*
  - El cliente concatena ese *timestamp* con la clave de usuario
  - Y calcula el MD5 de la clave
  - Después envía el comando `APOP username md5_obtenido`
  - El servidor hace la misma operación que el cliente y comprueba que le sale el mismo md5 (esto implica que el servidor ha de conocer la clave del usuario)

Ejemplo de APOP

```
Servidor: Espera (en el puerto 110)
Cliente > Conecta con puerto 110 del servidor
Servidor: +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
Cliente > APOP mrose c4c9334bac560ecc979e58001b3e22fb
Servidor: +OK mrose's maildrop has 2 messages (320 octets)
```

- El *timestamp* del servidor es la cadena `<1896.697170952@dbc.mtview.ca.us>`
- El nombre de usuario en este ejemplo es `mrose`
- Su clave es `tanstaaf`

Posteriormente se ampliaría POP3 para soportar TLS.

POP3: Estado de transacción

En este estado hay varios comandos disponibles:

Comando	Descripción
`STAT`	Obtiene cuántos mensajes hay en el buzón y cuántos bytes ocupa el buzón
`LIST` [m]	Muestra cuántos bytes ocupa el mensaje `m`, o cada mensaje del buzón si no se especifica `m`
`RETR` m	Descarga el mensaje `m`
`DELE` m	Marca para borrado el mensaje `m`
`TOP` m n	Descarga las cabeceras y las n primeras líneas del mensaje `m`
`QUIT`	Desconexión

POP3: Estado de actualización

- Ocurre cuando el cliente envía el comando `QUIT`
- El servidor entonces borra realmente los mensajes marcados para borrar con `DELE`
- Si no se marcó ninguno, esta fase no tiene lugar.

Filosofía de POP3

- El protocolo básicamente sirve para eliminar los buzones del MTA y transferirlos al MUA.
  - El usuario clasifica los mensajes en carpetas en su MUA
  - Hace búsquedas en los mensajes en su MUA
  - Chequeo de mensajes nuevos mediante *polling*

Este enfoque es incómodo si el usuario maneja varios ordenadores o dispositivos

Limitaciones de POP3

- Si borramos los mensajes del servidor tras descargarlos, sólo estarán accesibles en el dispositivo que los descargó.
- Si no los borramos, los buzones pueden exceder su cuota
- POP3 no proporciona ningún mecanismo para saber qué mensajes han sido ya descargados
- En todo caso la clasificación de mensajes en carpetas deberíamos repetirla en cada dispositivo al que descarguemos
- No podemos hacer búsquedas vía POP3 en los mensajes del MTA para descargar sólo los de un cierto tema (aunque podemos usar `TOP` para descargar su comienzo y hacer búsquedas locales)

4.5 IMAP

IMAP (*Internet Message Access Protocol*)

Este protocolo es mucho más complejo que POP3, y elimina muchos de sus problemas:

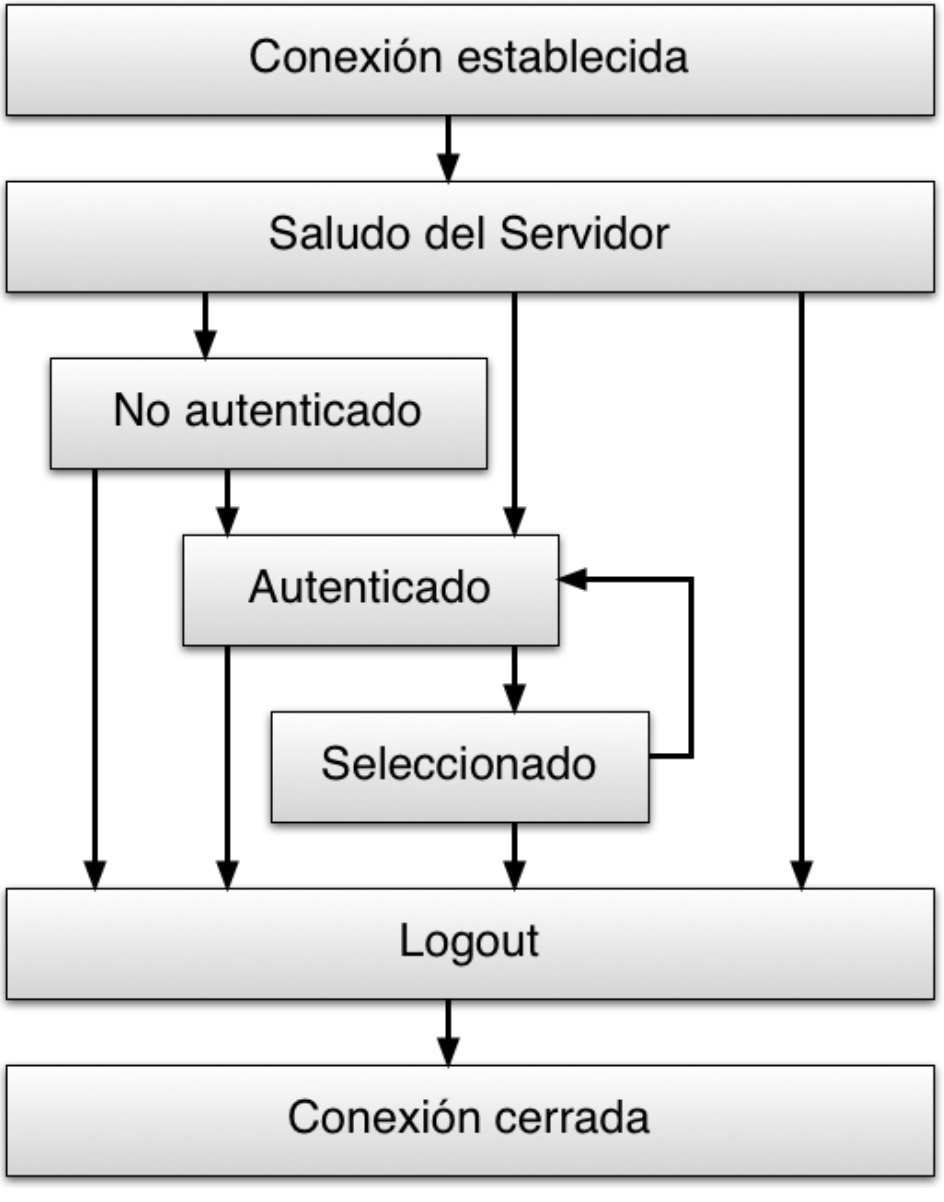
- Los mensajes se dejan en el servidor.
  - Las copias que pueda tener el cliente de los mensajes se consideran una *cache*, el original siempre está en el servidor.
  - Son accesibles por tanto desde varios clientes (MUA) incluso a la vez.
- El usuario puede clasificar mensajes en carpetas *directamente en el servidor*.
  - Por tanto esta clasificación es visible desde cualquiera de los clientes.
- Se pueden realizar búsquedas en los mensajes directamente en el servidor.

Más ventajas sobre POP3

- La conexión TCP puede quedar abierta mucho tiempo
  - Esto permite *push mail* (notificaciones del servidor para correo nuevo)
  - Y menores tiempos de latencia
- Acceso partes del mensaje para descarga parcial (por ejemplo a los adjuntos)
- Acceso a estado del mensaje (leido, visto, contestado, borrado), que puede cambiarse desde un cliente y verse en los otros.
- Puede funcionar *offline* (descarga copia local de los buzones, permite operar sobre ellos) y después sincronizarse con el servidor cuando vuelva a haber conexión.

Resumen simplificado del protocolo

IMAP es un protocolo *con estado*.



Forma general de los mensajes del protocolo

Del cliente al servidor:

- El mensaje consta de: Etiqueta + Comando + Parámetro(s)
- La *Etiqueta* es cualquier cadena que el cliente ponga
- Pero debe ser diferente para cada comando

Ejemplo:

```
Cliente > a002 select inbox
```

Forma general de los mensajes del protocolo

Del servidor al cliente, dos tipos:

- *Respuesta sin etiquetar* (Datos o Respuesta parcial)
- *Respuesta etiquetada* (Resultado final del comando)
  - Forma: Etiqueta + código de estado
  - El código de estado puede ser: `OK`, `NO`, `BAD`

Ejemplo:

```
Cliente > a002 select inbox
Servidor: * 18 EXISTS
Servidor: * FLAGS (\Answered\Flagged\Deleted\Seen\Draft)
Servidor: ...etc...
Servidor: a002 OK [READ-WRITE] SELECT completed
```

Fases del protocolo

Cuando el cliente se conecta, el servidor le saluda y pasa al estado siguiente:

- **No autenticado** Suele ser el estado inicial, y en él sólo se permiten los comandos para autenticarse o desconectar.
- **Autenticado** Estado en que el usuario se ha autenticado pero no ha seleccionado carpeta. En este estado puede obtener información sobre las carpetas disponibles, crear nuevas, borrarlas, pero *NO* acceder a los mensajes.
- **Seleccionado** Estado al que se pasa tras seleccionar una carpeta. En este estado puede manejar los mensajes. El comando `CLOSE` o `EXPUNGE` vuelve al estado anterior.
- **LOGOUT** A este estado se pasa mediante el comando `LOGOUT` y es entonces cuando se borran los mensajes marcados para borrar. Sin `LOGOUT` los mensajes no se borran.

IMAP: Estado no autenticado

Comando	Descripción
`STARTTLS`	Inicia una negociación TLS (como vimos para SMTP)
`LOGIN`	Proporciona *Nombre* y *Clave*
`AUTHENTICATE`	Solicita otro mecanismo de autenticación (ej: Kerberos u otros)

Ejemplo

```
Cliente > (Conecta al socket del servidor)
Servidor: * OK IMAP4rev1 Service Ready
Cliente > a001 login usuario secreto
Servidor: a001 OK LOGIN completed
```

Estado autenticado

Comando	Descripción
`SELECT` buzon	Selecciona un buzón (carpeta)
`EXAMINE` buzon	Igual que `SELECT` pero en modo solo-lectura
`CREATE` buzon	Crea un buzón nuevo (carpeta)
`DELETE/RENAME`	Borra o renombra buzón (salvo `INBOX`)
`LIST` opciones	Obtener lista de buzones, según opciones
`STATUS` buzon	Obtener estado del buzón
`APPEND` buzon [msgs]	Añadir mensajes a un buzón

`SELECT` y `EXAMINE` pasan al "*estado seleccionado*".

Estado seleccionado

Comando	Descripción
`CLOSE`	Se cierra el buzón actual
`EXPUNGE`	Borra los mensajes marcados y sale del buzón
`SEARCH` criterio	Busca en el buzón, soporta criterios muy complicados, basados en fechas, remitentes, etc
`FETCH` que	Descarga un mensaje o parte (por ejemplo, solo adjunto, solo cabecera, etc)
`STORE` opciones	Modifica opciones de un mensaje (sus *flags*, tales como `\Deleted`, `\Seen`, etc.)
`COPY` msg buzon	Copia un mensaje a otro buzón

Mensajes disponibles en cualquier estado

Comando	Descripción
`LOGOUT`	Pasar a estado *Logout* (cerrar sesión)
`CAPABILITY`	Mostrar capacidades del servidor (la respuesta depende del estado)

Ejemplo

```
Cliente > aa00 CAPABILITY
Servidor: * CAPABILITY IMAP4rev1 STARTTLS AUTH=GSSAPI LOGINDISABLED
Servidor: aa00 OK CAPABILITY completed
Cliente > aa01 SARTTLS
Servidor: aa01 OK STARTTLS completed
--- A partir de aqui todo iría cifrado ---
Cliente > aa02 CAPABILITY
Servidor: * CAPABILITY IMAP4rev1 AUTH=GSSAPI AUTH=PLAIN
Servidor: aa02 OK CAPABILITY completed
```