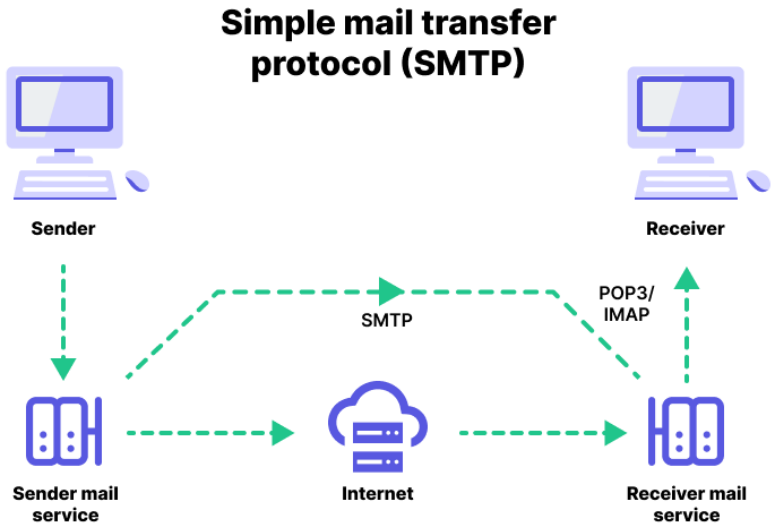
Protocolo SMTP

SMTP (Simple Mail Transfer Protocol) es un protocolo de comunicación utilizado para enviar correos electrónicos entre servidores. SMTP se encarga del envío de mensajes desde el cliente emisor al servidor del destinatario. Es un protocolo basado en texto y opera a nivel de la capa de aplicación del modelo OSI, usando el puerto 25 por defecto (o el puerto 587 para comunicaciones seguras mediante TLS/STARTTLS).

Flujo del Protocolo SMTP

El flujo de trabajo de SMTP sigue una serie de pasos estructurados que permiten el envío de correos electrónicos:

- **Conexión:** El cliente SMTP (generalmente un servidor de correo del remitente) establece una conexión TCP con el servidor SMTP del destinatario en el puerto 25 o 587.
- **Intercambio de saludos (HELO/EHLO):** El cliente SMTP inicia la comunicación enviando un comando HELO (o EHLO para indicar que soporta extensiones) seguido por su nombre de dominio. El servidor SMTP responde con un código de estado (como "250 OK").
- **Comando MAIL FROM:** El cliente envía el comando MAIL FROM para indicar la dirección de correo del remitente.
- **Comando RCPT TO:** El cliente envía el comando RCPT TO con la dirección de correo del destinatario. El servidor responde si la dirección es válida o no.
- **Comando DATA:** El cliente envía el comando DATA, lo que indica que el cuerpo del mensaje va a ser transmitido. El servidor SMTP responde con un código para indicar que está listo para recibir los datos.
- **Envío del mensaje:** El cliente envía el mensaje, incluyendo encabezados (From, To, Subject, etc.) y el cuerpo del correo. El mensaje termina con una línea que contiene un solo punto (.) para indicar el final.
- **Comando QUIT:** Tras recibir el mensaje, el cliente envía el comando QUIT, cerrando la conexión SMTP.



SMTP es fundamental para el envío de correos electrónicos y tiene múltiples aplicaciones en varios servicios y sistemas:

- **Correo Electrónico:** SMTP es el protocolo estándar para el envío de correos electrónicos a través de Internet, tanto en clientes de correo (Outlook, Thunderbird) como en servidores de correo.
- **Servicios de Notificaciones:** Aplicaciones que envían notificaciones a los usuarios (como sistemas de tickets, plataformas de marketing por correo, etc.) usan SMTP para enviar correos automatizados.
- **Transferencia entre Servidores:** SMTP es utilizado por servidores de correo para transferir mensajes entre diferentes dominios hasta que llegan al servidor de destino final.
- **Integración con Aplicaciones Web:** Las aplicaciones web usan SMTP para enviar correos de confirmación, recuperación de contraseñas, y otros tipos de notificaciones automatizadas.

Ventajas de SMTP

- **Amplia Compatibilidad:** Es un estándar bien establecido y compatible con casi todos los servidores de correo y clientes de correo electrónico.
- **Simplicidad:** El protocolo es simple y fácil de implementar, con un flujo de comandos básico y bien documentado.
- **Robustez:** SMTP está diseñado para garantizar la entrega de mensajes de correo, haciendo múltiples intentos si el servidor de destino no está disponible temporalmente.
- **Extensiones (ESMTP):** SMTP ha sido extendido a ESMTP (Extended SMTP), lo que permite características adicionales como autenticación, encriptación (STARTTLS), y gestión de mensajes grandes.
- **Soporte para Múltiples Destinatarios:** SMTP permite enviar un correo a múltiples destinatarios en un solo envío.

Desventajas de SMTP

- **Falta de Autenticación por Defecto:** SMTP original no incluía autenticación, lo que lo hace vulnerable a abusos como el spam. Aunque ESMTP permite autenticación, no todos los sistemas la implementan correctamente.
- **Inseguridad (sin cifrado nativo):** SMTP, en su forma básica, no incluye cifrado. Esto significa que los correos pueden ser interceptados. Sin embargo, esto se ha mitigado con el uso de STARTTLS para cifrar las comunicaciones.
- **Entrega Incierta:** SMTP solo asegura la entrega del correo hasta el servidor del destinatario, pero no garantiza que el destinatario final lea o reciba el mensaje (debido a factores como filtros de spam).
- **No Adecuado para Recepción de Correos:** SMTP es principalmente para enviar correos, y no maneja la recepción de correos electrónicos. Para recibir correos, se usan otros protocolos como POP3 o IMAP.
- **Limitaciones en el Tamaño del Mensaje:** Muchos servidores SMTP imponen límites en el tamaño del correo que pueden procesar, lo que puede ser un inconveniente cuando se trata de adjuntos grandes.

Desarrollo:

Para la presente práctica se va a realizar un programa para comprobar el funcionamiento del protocolo SMTP, donde se va a tener un cliente desarrollado en lenguaje C que va a permitir enviar un correo electrónico a un servidor instalable SMTP.

Para la instalación del servidor se va a utilizar la siguiente aplicación llamada MailHog:

```
sudo apt-get -y install golang-go
go install github.com/mailhog/MailHog@latest
```

Para ejecutar el servidor de prueba SMTP se hace de la siguiente manera:

```
~/go/bin/MailHog
```

Una vez instalado el servidor SMTP se va a realizar la lógica para el cliente que va a mandar la prueba de correo electrónico al servidor, para ello creamos el archivo ClienteSMTP.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
```

La biblioteca stdio.h se utiliza para funciones de entrada y salida, como printf. stdlib.h se usa para funciones de utilidad general, como exit(). string.h proporciona funciones para manipular cadenas, como strstr(). unistd.h contiene constantes y tipos para las funciones POSIX, y arpa/inet.h es crucial para las operaciones de red, ya que define funciones de dirección de red, como inet_pton().

```
#define SERVER "127.0.0.1"  // Dirección IP del servidor SMTP (MailHog)
#define PORT 1025           // Puerto de MailHog (1025 por defecto)
#define BUFSIZE 1024
```

SERVER especifica la dirección IP del servidor SMTP (en este caso, MailHog ejecutándose localmente en la dirección 127.0.0.1). PORT establece el puerto en el que MailHog escucha las conexiones SMTP (el puerto 1025 es el puerto predeterminado de MailHog). BUFSIZE se define como 1024, que es el tamaño del buffer utilizado para almacenar los datos que se envían y reciben.

```
void send_smtp_command(int sockfd, const char *cmd, const char
*expected_response) {
    char buffer[BUFSIZE] = {0};

    // Enviar el comando SMTP
    send(sockfd, cmd, strlen(cmd), 0);
    printf("Enviado: %s", cmd);

    // Leer la respuesta del servidor
    recv(sockfd, buffer, BUFSIZE, 0);
    printf("Respuesta: %s", buffer);

    // Verificar si la respuesta es la esperada
    if (strstr(buffer, expected_response) == NULL) {
        fprintf(stderr, "Error: no se recibió la respuesta esperada.\n");
        exit(EXIT_FAILURE);
    }
}
```

send_smtp_command maneja el envío de comandos SMTP al servidor y la verificación de la respuesta. Recibe el descriptor del socket (sockfd), el comando SMTP a enviar (cmd), y la respuesta esperada del servidor (expected_response). Se inicializa un buffer para almacenar la respuesta del servidor.

```
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Error al crear el socket");
        exit(EXIT_FAILURE);
    }
```

- AF_INET indica que se trata de una conexión de red basada en IPv4.
- SOCK_STREAM define que se utilizará el protocolo de transporte TCP, necesario para SMTP, ya que requiere una conexión confiable.

```
// Enviar los comandos SMTP
    send_smtp_command(sockfd, "HELO localhost\r\n", "250");
    send_smtp_command(sockfd, "MAIL FROM:<test@example.com>\r\n", "250");
    send_smtp_command(sockfd, "RCPT TO:<recipient@example.com>\r\n", "250");
    send_smtp_command(sockfd, "DATA\r\n", "354");

    // Enviar el contenido del correo
    send_smtp_command(sockfd, "Subject: Test Mail\r\n", "");
    send_smtp_command(sockfd, "From: test@example.com\r\n", "");
    send_smtp_command(sockfd, "To: recipient@example.com\r\n\r\n", "");
    send_smtp_command(sockfd, "Este es un mensaje de prueba.\r\n.\r\n", "250");
```

Se envían una serie de comandos SMTP (HELO, MAIL FROM, RCPT TO, DATA) mediante la función send_smtp_command, que ya se explicó.

HELO: Identifica el cliente al servidor.

MAIL FROM: Indica la dirección de correo del remitente.

RCPT TO: Especifica el destinatario del correo.

DATA: Comienza el envío del contenido del correo electrónico.

Para el compilado se va a realizar con el siguiente comando:

```
gcc ClienteSMTP.c -o clientesmtp
```

Una vez ejecutando el cliente smtp en la terminal tenemos la siguiente salida:

```
root@User1158-PC:/mnt/e                                                    SMTP# ./clientes
mtp
Respuesta del servidor: 220 mailhog.example ESMTP MailHog

Enviado: HELO localhost
Respuesta: 250 Hello localhost
Enviado: MAIL FROM:<test@example.com>
Respuesta: 250 Sender test@example.com ok
Enviado: RCPT TO:<recipient@example.com>
Respuesta: 250 Recipient recipient@example.com ok
Enviado: DATA
Respuesta: 354 End data with <CR><LF>.<CR><LF>
Enviado: Subject: Test Mail
```

```
[APIv1] KEEPALIVE /api/v1/events
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] Starting session
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: INVALID] Started session, switching to ESTABLISH state
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] Sent 35 bytes: '220 mailhog.example ESMTP MailHog\r\n'
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] Received 16 bytes: 'HELO localhost\r\n'
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: ESTABLISH] Processing line: HELO localhost
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: ESTABLISH] In state 1, got command 'HELO', args 'localhost'
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: ESTABLISH] In ESTABLISH state
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: ESTABLISH] Got HELO command, switching to MAIL state
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] Sent 21 bytes: '250 Hello localhost\r\n'
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] Received 30 bytes: 'MAIL FROM:<test@example.com>\r\n'
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: MAIL] Processing line: MAIL FROM:<test@example.com>
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: MAIL] In state 6, got command 'MAIL', args 'FROM:<test@example.com>'
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: MAIL] In MAIL state
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: MAIL] Got MAIL command, switching to RCPT state
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] Sent 32 bytes: '250 Sender test@example.com ok\r\n'
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] Received 33 bytes: 'RCPT TO:<recipient@example.com>\r\n'
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: RCPT] Processing line: RCPT TO:<recipient@example.com>
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: RCPT] In state 7, got command 'RCPT', args 'TO:<recipient@example.com
>'
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: RCPT] In RCPT state
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: RCPT] Got RCPT command
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] Sent 40 bytes: '250 Recipient recipient@example.com ok\r\n'
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] Received 6 bytes: 'DATA\r\n'
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: RCPT] Processing line: DATA
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: RCPT] In state 7, got command 'DATA', args ''
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: RCPT] In RCPT state
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] [PROTO: RCPT] Got DATA command, switching to DATA state
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] Sent 37 bytes: '354 End data with <CR><LF>.<CR><LF>\r\n'
2024/09/29 22:06:00 [SMTP 127.0.0.1:53040] Received 20 bytes: 'Subject: Test Mail\r\n'
[APIv1] KEEPALIVE /api/v1/events
[APIv1] KEEPALIVE /api/v1/events
[APIv1] KEEPALIVE /api/v1/events
```