Protocolos de Comunicación

Trabajo Práctico Especial:

Grupo:
Tobias Ves Losada (63342)
Matias Mutz (63590)
Ben Deyheralde (63559)

Profesores:
Marcelo Fabio Garberoglio
Juan F. Codagnone
Sebastian Kulesz

1. Protocolos y aplicaciones desarrolladas..... 2
SMTP..... 2
Mensajes de respuesta..... 3
Direcciones válidas..... 3
Máquina de estados..... 3
Monitoreo..... 4
Datagrama request..... 4
Datagrama response..... 5
2. Problemas encontrados durante el diseño y la implementación..... 5
3. Limitaciones de la aplicación..... 6
4. Posibles extensiones..... 6
5. Conclusiones..... 6
6. Ejemplos de prueba..... 7
7. Guía de instalación..... 8
8. Ejemplos de configuración y monitoreo..... 9
9. Documento de diseño del proyecto..... 10

1. Protocolos y aplicaciones desarrolladas

SMTP

Lo primero que se desarrolló fue un protocolo SMTP que sigue los lineamientos del RFC 5321. Este protocolo atiende conexiones tanto IPv4 como IPv6 de manera no bloqueante. El mismo acepta los dominios @mydomain.com y se guardan los mails siguiendo el formato "Maildir", es decir, por cada usuario se crea un directorio con su nombre de usuario que dentro tiene los subdirectorios "new" (mails sin leer), "cur" (mails leídos) y tmp (mails en proceso de envío). Las etapas del protocolo fueron diseñadas mediante una máquina de estados y se dividen de la siguiente manera:

EHLO - lo primero que espera el servidor es un saludo del cliente mediante un "EHLO" o "HELO" (observación: tanto este comando como el resto son case insensitive, por ende "ehlo" también se considera como un saludo).

MAIL FROM - luego se espera un "MAIL FROM:" que indique el remitente del mensaje. El remitente debe ser una dirección válida. El servidor responderá con un "250 OK" en caso de éxito. Para más información respecto a direcciones válidas y mensajes de respuesta, ver las secciones "Mensajes de respuesta" y "Direcciones válidas" .

RCPT TO - el cliente deberá indicar el o los destinatarios mediante "RCPT TO:" y el protocolo soporta hasta 100 destinatarios que deben ser indicados mediante una seguidilla de "RCPT TO:"

DATA - luego el cuerpo del mensaje el cliente lo debe insertar después de que el servidor le responda con un "354" a su "DATA". El cuerpo del mensaje debe finalizar con un "<CRLF>.<CRLF>".

Una vez finalizado estos pasos, se puede repetir desde el "MAIL FROM" hasta el "DATA" para enviar un mensaje diferente.

Finalmente, cuando el cliente quiera cerrar la conexión, lo podrá hacer con "**QUIT**".

Para una mejor visualización de cómo se transiciona entre estados, ver la sección "Máquina de estados".

Mensajes de respuesta

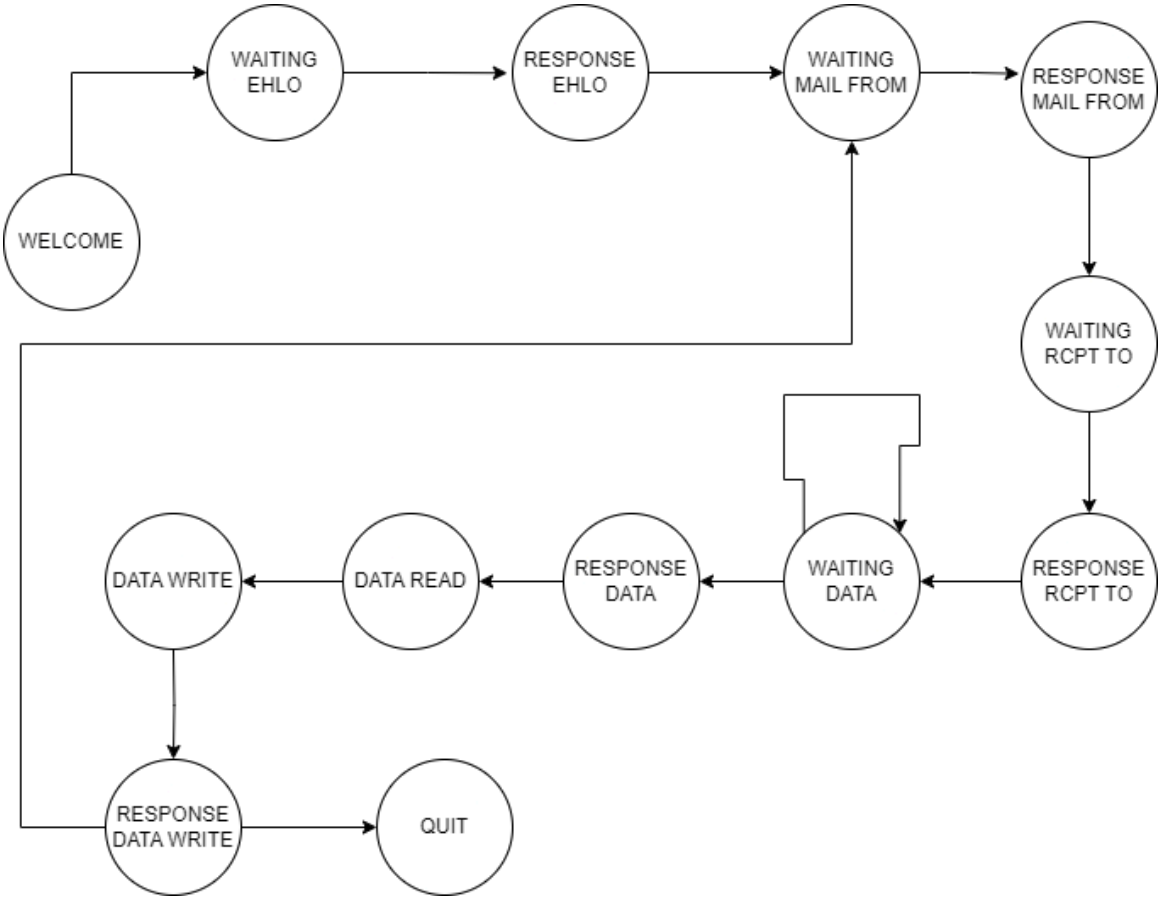
El protocolo implementado responderá:

- 220 Service Ready: en el momento que empieza la conexión.
- 250 server at your service: luego de recibir el saludo “HELO” o “EHLO”.
- 250 OK ante casos de éxito, por ejemplo al recibir un “RCPT TO”, o un “MAIL FROM:” válido.
- 354: cuando está listo para recibir el cuerpo del mensaje luego de un “DATA”.
- 221 Bye: al cerrar la conexión.
- 555 Syntax error: al enviar un remitente o destinatario invalido.
- 501 Syntax error: al enviar “DATA” con un argumento.

Direcciones válidas

- No pueden comenzar con “.”.
- No pueden tener dos “.” consecutivos.
- No pueden tener más de una “@”.
- No pueden tener espacios.

Máquina de estados



Como se mencionó anteriormente, en el estado WAITING DATA, se puede seguir recibiendo “MAIL FROM:” sin cambiar de estado. En el momento que se recibe un “DATA” ya se transiciona a “RESPONSE DATA”.

Los estados “WAITING… ” son los estados en que el servidor espera una acción por parte del cliente. Los estados “RESPONSE” son los estados en que el servidor responde con un mensaje informativo o de éxito/error.

Los estados “DATA READ” y “DATA WRITE” son los encargados de leer el cuerpo del mensaje y enviarlo al archivo creado en la carpeta “tmp”.

Monitoreo

En segundo lugar, se diseñó un protocolo de manejo y monitoreo basado en UDP que permite obtener datos históricos: cantidad de conexiones, bytes transferidos, entre otros.

Para esto se diseñó una aplicación cliente que se comunica con la aplicación manager mediante udp. Teniendo en cuenta esto, como en el caso de udp en este protocolo no hay un concepto de sesión, sino que el cliente se comunica con el servidor con un request de algún comando y posteriormente termina la comunicación con la respuesta del servidor.

Datagrama request

Para el diseño del protocolo nos basamos en lo discutido en clase, con un datagrama “request” de 14 bytes conformado de la siguiente manera:

- Firma del protocolo (2 bytes)
- Versión del protocolo (1 byte)
- Identificador del “request” (2 bytes)
- Autenticación (8 bytes)
- Comando - 1 byte
    - 0x00 - cantidad de conexiones históricas
    - 0x01 - cantidad de conexiones concurrentes
    - 0x02 - cantidad de bytes enviados
    - 0x03 - cantidad de bytes recibidos
    - 0x04 - cantidad de bytes transferidos

Datagrama response

Análogamente, diseñamos un datagrama de respuesta de 14 bytes conformado de la siguiente manera:

- Firma del protocolo (2 bytes)
- Versión del protocolo (1 byte)
- Identificador del “request” (2 bytes)
- Estado - 1 byte
    - Éxito - 0x00
    - Errores:
        - Autenticación fallida - 0x01
        - Versión inválida - 0x02
        - Comando inválido - 0x03
        - Longitud del “request” inválido - 0x04
        - Error genérico - 0x05
- Respuesta:
    - Dato solicitado (4 bytes)
    - Booleano (1 byte)

2. Problemas encontrados durante el diseño y la implementación

Uno de los problemas enfrentados fue el diseño de la máquina de estados. Aunque en las clases prácticas del trabajo práctico se explicó, nunca habíamos implementado una en código y al principio tuvimos problemas para identificar qué momentos debían modelarse como un estado. Al principio tuvimos el problema de querer simplificar la máquina de estados juntando varios momentos en un mismo estado pero luego nos dimos cuenta de que teníamos que separarlo en más estados.

Otro de los problemas fue durante el desarrollo de la aplicación de monitoreo cuando en un principio se armaban mal los datagramas por un incorrecto pasaje al orden de bytes de la red. Luego de imprimir requests y responses se terminó solucionando mediante el uso de las funciones htons(); y ntohs(); que ordenan los bytes de acuerdo al orden de la red y el host.

3. Limitaciones de la aplicación

Debido a los tiempos ajustados, no hemos llegado a implementar las transformaciones de mensajes requeridas por el enunciado.

Además, el protocolo SMTP no soporta autenticación, que en un principio era un requisito y luego se quitó.

4. Posibles extensiones

Para mejorar la funcionalidad y seguridad del servidor SMTP y la aplicación de monitoreo, se podría implementar un sistema de creación de usuarios y gestión de contraseñas. Esto incluiría una interfaz de administración que permita a los administradores gestionar usuarios. También se incorporaría un sistema seguro de recuperación y restablecimiento de contraseñas basado en tokens, y se definirían roles y permisos para diferenciar los niveles de acceso entre usuarios, proporcionando un control granular sobre las acciones y el acceso dentro del sistema.

5. Conclusiones

El proyecto fue un desafío importante que probó nuestra capacidad para implementar protocolos de comunicación, específicamente en la creación de un servidor SMTP y una aplicación de monitoreo basada en UDP. Valoramos la experiencia práctica que nos proporcionó al realizar pruebas reales dentro de un entorno controlado, lo cual fue crucial para entender la aplicación y los protocolos diseñados. Durante el desarrollo del módulo de monitoreo, destacamos la importancia de definir claramente los protocolos para facilitar un desarrollo efectivo y entender bien los problemas que surgían. En resumen, este proyecto fue una oportunidad valiosa para aplicar teorías en un contexto práctico, fortaleciendo nuestra comprensión y habilidades en redes y comunicaciones.

6. Ejemplos de prueba

Para justificar que nuestro servidor atiende a al menos 500 clientes de forma concurrente y simultánea, ejecutamos un test que crea hilos y envía correos mediante nuestro servidor. El programa simula eficazmente un escenario de carga elevada al enviar múltiples correos electrónicos simulados al servidor SMTP local. Cada hilo representa un cliente simulado que envía un correo electrónico y recibe una respuesta del servidor. Este tipo de prueba nos es útil para evaluar la capacidad de respuesta y el rendimiento del servidor SMTP bajo presión.

Obtuvimos los siguientes resultados con las pruebas del stress test:

20 threads:

real 0m1.512s
user 0m0.098s
sys 0m0.089s
throughput: 13.22

100 threads:

real 0m3.551s
user 0m0.136s
sys 0m0.127s
throughput: 28.16

250 threads:

real 0m8.706s
user 0m0.135s
sys 0m0.283s
throughput: 28.7

500 threads:

real 0m12.798s
user 0m0.191s
sys 0m0.522s
throughput: 39.06

Notamos que a medida que aumenta el número de hilos, el tiempo real necesario para completar el test también aumenta significativamente. Por ejemplo, con 20 hilos el tiempo real es de 1.512 segundos, mientras que con 500 hilos se incrementa a 12.798 segundos. Esto sugiere que el servidor SMTP local tiene un límite en la cantidad de conexiones concurrentes que puede manejar eficientemente, igualmente no es algo descabellado que tarde 12 segundos en enviar 500 correos.

7. Guía de instalación

Para compilar y ejecutar el programa es necesario tener instalado GCC y Make. Luego para ejecutar las aplicaciones se debe correr ./smtpd y ./client, cada uno con sus respectivos comandos. Para probar un test como por ejemplo el stress test, se debe ejecutar make stress_test y eso ejecutará el test de stress.

Para compilar y manejar el proyecto, se pueden utilizar los siguientes comandos **make**:

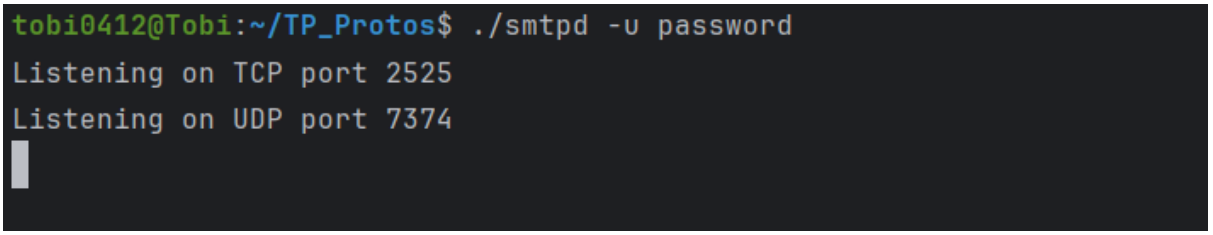
- **make clean**: Este comando elimina todos los archivos generados durante la compilación, limpiando el entorno de construcción.
- **make all**: Compila tanto el código del servidor como el del cliente. Genera los binarios necesarios en el directorio raíz del proyecto.
- **make server**: Compila exclusivamente el código del servidor, generando el binario correspondiente.
- **make client**: Compila únicamente el código del cliente, generando el binario correspondiente.
- **make stress_test**: Compila y ejecuta específicamente la prueba de estrés, útil para verificar el comportamiento del sistema bajo carga.

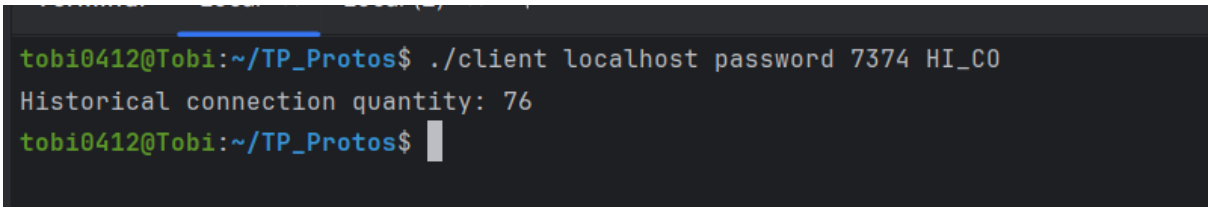
Una vez que se hayan compilado los binarios, se pueden ejecutar las aplicaciones utilizando los siguientes comandos:

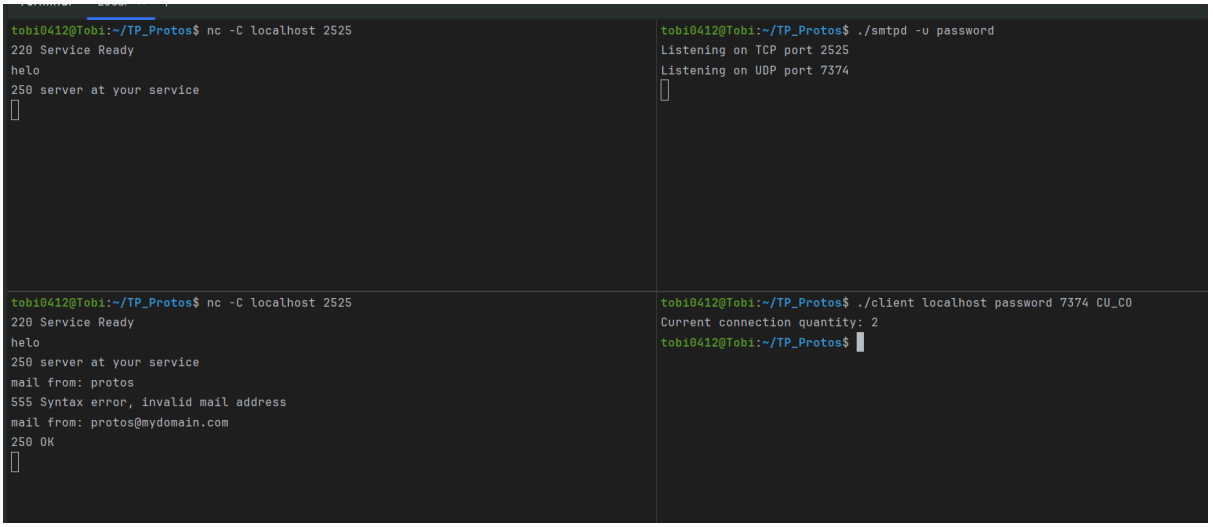
- **Para el servidor**: Ejecutar **./smtpd** en la terminal.
- **Para el cliente**: Ejecutar **./client** en la terminal.

Para ejecutar una prueba específica, como la prueba de estrés, se utiliza el comando **make stress_test**. Esto compilará y ejecutará el test de estrés, permitiendo evaluar el rendimiento del sistema bajo condiciones exigentes.

8. Ejemplos de configuración y monitoreo



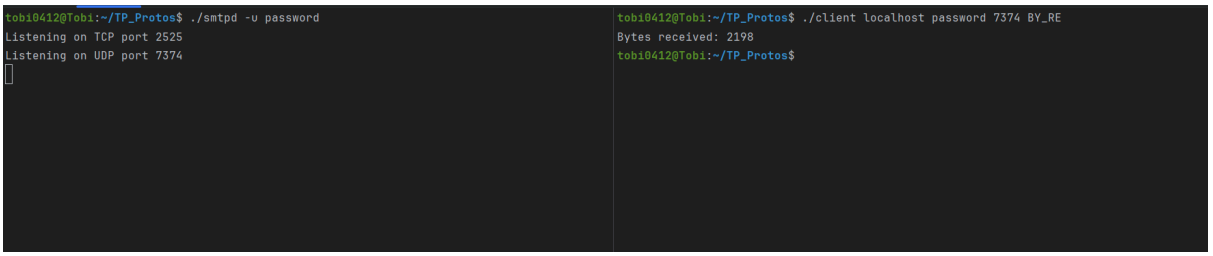




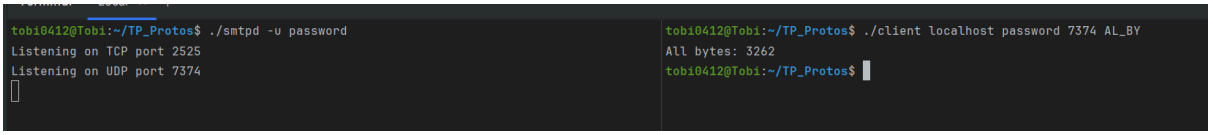
En esta imagen se puede ver tres clientes de la aplicación. Los dos de la izquierda están enviando un mail y a la derecha inferior está consultando la métrica de cuántos clientes conectados.

Esta imagen muestra el seteo de contraseña. Al correr ./smtpd seguido de -u password, del lado cliente se debe ingresar password para que se permita el acceso a las métricas.

En este caso, como las contraseñas no coinciden, se rechaza el pedido de métrica.



En esta imagen, se consulta la métrica de bytes recibidos.



En esta imagen, se consulta la métrica de total de bytes transferidos.

9. Documento de diseño del proyecto

Servidor SMTP:

1. **Inicio y Configuración:**
   - El servidor SMTP se inicia con configuración de puertos y parámetros específicos como la contraseña del manager.
2. **Preparación del Entorno:**
   - Se crean sockets para aceptar conexiones en los puertos SMTP y de métricas especificados.
3. **Proceso de Envío de Correo:**
   - Establecimiento de Conexión:
     - El cliente establece una conexión TCP con el servidor SMTP en el puerto 2525.
   - Intercambio de Comandos:
     - El cliente envía comandos como HELO/EHLO, MAIL FROM, RCPT TO, DATA.
     - El servidor responde con códigos de respuesta adecuados (250, 354, etc.).
   - Transferencia de Datos:
     - El cliente envía el cuerpo del mensaje.
     - El servidor almacena el mensaje y envía una confirmación (250 OK).
   - Cierre de Conexión:

        - El cliente envía QUIT y el servidor responde con 221 Bye, cerrando la conexión.

4. **Manejo de Errores:**
    - El servidor valida los comandos y responde con códigos de error específicos en caso de comandos inválidos (500, 501, 502, 503).
5. **Recolección de Estadísticas:**
    - Se registran métricas como cantidad de conexiones, bytes transferidos y mensajes procesados.
    - Las métricas se mantienen en memoria y se pierden en reinicios del servidor.

Protocolo de Monitoreo:

1. **Recepción de Solicitudes UDP:**
    - El servidor de métricas recibe solicitudes UDP del cliente de monitoreo.
2. **Procesamiento de Solicitudes:**
    - Se verifica la autenticación y se valida la solicitud de comando (cantidad de conexiones, bytes transferidos, etc.).
3. **Envío de Respuesta:**
    - El servidor responde con un mensaje UDP que indica el resultado de la solicitud (éxito o error).

Cliente de Monitoreo:

1. Preparación de la Solicitud:
    - El cliente prepara un datagrama UDP con la firma, versión, ID de solicitud, autenticación y comando necesarios.
2. Envío y Recepción:
    - Envía el datagrama al servidor de métricas y recibe la respuesta.
    - Valida la respuesta y muestra el resultado o error al usuario.