Título:

TRABAJO PRÁCTICO ESPECIAL

Nivel y Área:

72.07 Protocolos de Comunicación – 2º Cuatrimestre 2018
Comisión: S - **Grupo:** 04
Fecha: 06 de noviembre de 2018

Alumnos Expositores:

- AQUILI, Alejo Ezequiel (57432)
- BASSANI, Santiago (57435)
- RITORTO, Bianca (57082)
- SANGUINETI ARENA, Francisco Javier (57565)

Índice

Protocolos y aplicaciones desarrolladas

1 - Aplicaciones

Se desarrollaron y diseñaron tres aplicaciones y un protocolo para llevar a cabo la tarea asignada. Las apliaciones implementadas fueron:

- Pop3filter - Servidor proxy POP3
- Stripmime - Censurador de "media types"
- Pop3ctl - Cliente SCTP para administración del proxy via el protocolo a nivel de aplicación diseñado.

1.1 - Pop 3filter: Servidor proxy POP3

El servidor proxy del protocolo POP3, es un servidor concurrente con entrada salida multiplexada no bloqueante. Para el desarrollo de este servidor se utilizaron, modificaron y adaptaron códigos fuentes provistos por Juan Francisco Codagnone por parte de la cátedra de Protocolos de Comunicación. Entre estos codigos fuentes se encontraba un TAD (Tipo Abstracto de Datos) para abstraer la implementación de un *multiplexor*. Específicamente, esta implementación utiliza la función **pselect()** para multiplexar filedescriptors. No se cambio la implementación manteniendo el contrato del TAD por una que use **poll()** o **epoll()** por cuestiones de tiempos, pero estas ultimas resultaban más eficientes que la función utilizada.

Sobre los diferentes *file descriptors* que pueden registrarse en el *multiplexor* se pueden configurar intereses de lectura, escritura o ningún interés. El proxy recibe como argumento la dirección del servidor POP3 origen correspondiente, la cual puede ser una dirección IPv4, IPv6 o un nombre a resolver. Por cada conexión entrante, cuando el proxy las atiende, se intenta conectar al origen, y si es necesario realizar una consulta DNS mediante **getaddrinfo()** para resolver las direcciones asociadas al nombre, se realiza la creación de un nuevo hilo, el cual es creado con el objetivo de que este se bloquee y así no bloquear el hilo principal.

El servidor recibe opciones como parámetros por línea de comando respetando el estilo del estándard POSIX. Se utilizó la función **getopt()** para ello esta tarea. (Las opciones soportadas son las definidas en el documento pop3filter.8)

También se utilizó código fuente de un motor de máquinas de estados para manejar los eventos asociados al *multiplexor*. Dentro de los estados utilizados se encuentra un estado de **CONNECTING** para efectuar la conexión de manera directa si se trataba de una dirección IP, o como resultado de antender la resolución del thread bloqueado (DNS). Luego existe un estado **HELLO** que pone en manifiesto el carácter de "orientado a conexión" del protocolo POP3, en el cual se lee el saludo inical del servidor origen. De completarse este saludo se trancisiona a un estado **CHECK_CAPABILITIES** que permite validar capacidades de un servidor POP3 de forma transparente al usuario cliente del proxy (ej. Mail User Agent). Debería mostrar que se soporta *pipelining* en cualquier caso, pero no se pudo implementar por cuestiones de tiempo. El usuario no

se percata de la validación realizada por el proxy mediante el envio del comando **CAPA**. La capacidad de interés a validar es una capacidad que como define el RFC-2449 (Extensión de POP3) no cambia su respuesta si es consultada en la etapa de autentificación o en la etapa de transacción del protocolo en cuestión. El proxy realiza este chequeo para poder ofrecerle a sus clientes la capacidad de *pipelining* sin importar si el servidor origen la implementa, y en caso de implementarla, hacer uso adecuado de la misma. Se deberia informar siempre al cliente cuando consulte el comando CAPA que cuenta con la capacidad de pipeling, el proxy simplemente copia la respuesta de ejecutar CAPA en el origen , pero deberia tener en cuenta que si no cuenta con PIPELINING el origen, imprimir en la respuesta multiliniea una nueva linea que incluya la capacidad en cuestion.

Existe un estado COPY global que se encarga de la transferencia de bytes del cliente al proxy del proxy al servidor origen del proxy al filtro de la apliación de filtro al proxy y todos sus subestados que basan su funcionamiento en información de si se dispone o no de PIPELINING o de datos obtenidos del filtro, entre otros.

Por otro lado, se encuentran los estados de ERROR y DONE como estados terminales para succesos o errores.

Otras características desarrolladas por el servidor son el manejo de *timeouts* para desconectar clientes inactivos para lo cual se definió un valor de timeout (como puede ser 20 segundos) y se decidió, post consultas con la cátedra, realizar chequeos sobre los *file descriptors* del **pselect()** cada cierta fracción del timeout (como pueden ser 0.25 o 0.5) para no generar *overhead* inecesario sobre las iteraciones del *multiplexor*. Para aprovechar la potencia del protocolo POP3 se implementó un sistema que permite escribir (sin bloquearse) en el cliente un error con indicador negativo **-ERR** seguido de algún mensaje personalizado que permita avisar al cliente sobre fallas inesperadas como no poder conectarse al servidor origen indicado, entre otras.

Los buffers utilizados en el proxy son un TAD que cuenta con tres punteros diferentes: uno de lectura, otro de escritura y finalmente incorpora un puntero de procesamiento. Esta idea sobre la estructura fue una idea sugerida por el ayudante de cátedra Fernán Oviedo, la cual nos resultó altamente práctica dado que con esta estructura se pueden analizar los bytes enviados del cliente al servidor o del origen al servidor así pudiendo extraer información e involucrar el procesamiento por la consumición de octetos en los distintos parsers utilizados, entre otros beneficios.

Teniendo esta estructura innovadora de buffer solo requeríamos guardar información mínima de los comandos y las respuestas de POP3 del cliente a proxy y del origen al proxy. Esta información está almacenada en una estructura de datos FIFO con una cola que almancena esos “esqueletos” de comandos y respuestas asociadas. Estos esqueletos pueden almacenar un argumento o parámetro de un comando POP3 asociad para limitar esta información sin privarnos de poder almacenarla. Esto lo hacemos dado que para el caso de los comandos USER y APOP con respuestas cuyos indicadores fueron positivas (en el caso de USER además lo acompaña un comando PASS con respuesta positiv)a nos interesa guradar el nombre de usuario POP3 logeado en dicha sesión, para poder enviar mediante el uso de variables de entorno esta información y muchas más a la aplicacion de

filtro para que ella lo utilice a su gusto y necesidad. Así fue como siguiendo el RFC - 1939 (POP3 Definition) se decidió que el tamaño máximo permitido para los argumentos de pop 3 es de 40 bytes.

Se utilizaron en total 3 buffers, uno de lectura, uno para la application filtro, y uno para escritura. Los primeros dos, producto de las pruebas de stress realizadas, pueden operar con un tamaño minimo de 1 byte, mientras que el tercero requiere un mínimo de 3 bytes para poder operar dado que debe poder renocer la secuencia de octetos de terminación de las respuestas multilineas POP3 (".\r\n").

En lo que respecta a eficiencia en el uso de memoria, por un lado se utilizaron activamente allocamientos dinámicos para sustentar las estructuras de datos y TAD subyacentes. Pero por otra parte en lo que respecta a las estructuras utilizadas por el proxy para mantener el contextos y datos por los diferentes estados y las diferentes trancisiones, se armó una *pool* para mejorar el manejo del alocamiento tan costoso de estructuras como estas que almacenan parsers, información, entre otros.

1.2 - Stripmime -Programa para filtrar tipos MIME

Ésta aplicacion se desarrolló con el objetivo de que el proxy pueda ceder el control sobre el contenido de los mails que maneja sobre POP3 a aplicaciones externas que, como es el caso de esta, permitan, por ejemplo, censurar correos electrónicos. En particular, censurar *media types* o tipos **MIME** donde, especificando un media range o lista de media types (compatible con la lista enviada en el header ACCEPT del protocolo HTTP, seccion 5.3.2 RFC-7231) puede copiar un mensaje de remplazo sobre estas partes.

En el diseño y desarrollo de ésta aplicación se contempló configurar el *header* "Content-Transfer-Encoding" para poder realizar reemplazos con caracteres donde la parte más significativa de los caracteres ASCII pueda estar en estado alto. Para ello se utilizaría el valor *quoted-printable*. No se pudo realizar la función que dado un string lo codifique en *quoted-printable* por cuestiones de tiempos.

Ésta implementación utiliza códigos fuente de la cátedra, tambien adaptados, para parsers y utilidades de parsers. Por otro lado, utiliza un container de *media types* que parsea el *media range* con las funciones reentrantes de **strtok()** y realizando un listado de tipos y sus subtipos asociados para cada *media type* en cuestión.

Previo a la implementación se analizaron los RFC-822, RFC-5322, RFC-1341 y RFC-2045. De aqui y otras fuentes se recolecto información necesaria para la implementación y colección de datos de diseño como el maximo de los argumentos de headers de mensajes del tipo mime de 70 bytes.

Por, útilimo se utiliza un stack para llevar el seguimiento de los delimitadores "boundaries" para el control de los tipos *multipart* y también los tipos message-RFC822 y así poder censurar las partes discretas asociadas al tipo compuesto en cuestión.

1.3 - Pop3ctl - Cliente SCTP para administración del proxy

El cliente administrador del servidor es un intérprete de comandos básico que adeás soporta las opciones de argumentos por líneas de comando -o para elegir el puerto a conectarse y -l para setear la dirección a la cual conectarse (IPv4 , IPv6 o nombre). Estas opciones son también compatibles con el estilo del estandar POSIX.

2- Protocolo implementado (RAP: Remote Admin Protocol)

El protocolo de configuración del proxy se comunica a través de SCTP. Requiere de autenticación mediante una contraseña configurable desde el mismo administrador. Al iniciar la conexión se pide la contraseña y luego se envía un mensaje de bienvenida acompañado con la lista de comandos disponibles, sus argumentos, y las descripciones.

El protocolo es orientado a bytes, por dos razones. La primera porque es más sencillo el traslado de la información por el socket que utilizamos. La segunda, porque minimiza el *overhead* del datagrama.

Al trabajar sobre SCTP se tiene la ventaja de que los datos se transportan por mensajes. Además, se limita el tamaño del comando y de la respuesta ya que tiene que entrar en un paquete del protocolo.

Al iniciar la conexion el servidor envia un mensaje de bienvenida el cual indica que esta disponible para atender al cliente con el siguiente datagrama:

respCode:	200
etag:	0
encoding	TEXT_TYPE
dataLength	3
data:	“OK”

En caso de respuesta de error el servidor debe responder con el etag actual del campo corresponsiente.

Al hacer cualquier request de tipo SET o UPDATE deben pasarse los etags correspondientes al valor mas actual que se posea. En caso de que este difiera del servidor, el mismo respondera con el error correspondiente y el etag actual. Todos los request de tipo GET obviaran el etag. Pero el servidor enviara el etag actual.

2.1 - Comandos

Hay 19 comandos disponibles. Cada uno es case insensitive, y corresponde con una sola palabra. Los argumentos (si son necesarios) se separan del comando con un espacio.

2.1.0: Comando “help”

- Imprime los comandos disponibles
- Argumentos: NULL

- Respuestas: *Imprime los comandos*

2.1.1: Comando "setCredential"

- Cambia la contraseña del administrador
- Argumentos: La nueva contraseña
- Respuestas:
    - "[SUCCESS] Credentials updated!"
    - "[FAILURE] There was a problem updating the credentials"

2.1.2: Comando "setFilterProgram"

- Determina el programa para usar para los filtros
- Argumentos: El nombre del programa
- Respuestas:
    - "[SUCCESS] Filter updated!"
    - "FAILURE] There was a problem updating the filtering program"

2.1.3: Comando "viewMetrics"

- Muestra las estadísticas del proxy
- Argumentos: NULL
- Respuestas:
    - "[SUCCESS] Metrics: Number of Bytes: x Buffer Size: y Active Users: z"
    - "[FAILURE] There was an error fetching the metrics"

2.1.4: Comando "getNBytes"

- Muestra la cantidad de bytes transferidos
- Argumentos: NULL
- Respuestas:
    - "[SUCCESS] The number of transfered bytes is: x"
    - "[FAILURE] There was an error fetching the transfered bytes"

2.1.5: Comando "getBufferSize"

- Muestra el tamaño del *buffer*
- Argumentos: NULL
- Respuestas:
    - "[SUCCESS] The buffer size is: x bytes"
    - "[FAILURE] There was an error fecthing the buffer size"

2.1.6: Comando "getFilter"

- Muestra si los filtros están activados o no
- Argumentos: NULL
- Respuestas:

- "[SUCCESS] The filter is: ON"
- "[SUCCESS] The filter is: OFF"
- "[FAILURE] There was an error fecthing the filter state"

2.1.7: Comando "setFilter"

- Activa o desactiva los filtros
- Argumentos: "Y" o "N"
- Respuestas:
  - "[SUCCESS] Filter updated!"
  - "[FAILURE] There was a problem updating the filter"

2.1.8: Comando "getFilterProgram"

- Muestra el programa que se estña utilizando para los filtros
- Argumentos: NULL
- Respuestas:
  - "[SUCCESS] The filter command is: *string*"
  - "[FAILURE] There was an error fecthing the filter command"

2.1.9 Comando "setBufferSize"

- Modifica el tamaño del buffer del proxy
- Argumentos: El nuevo tamaño del buffer
- Respuestas:
  - "[SUCCESS] Buffer size updated!"
  - "[FAILURE] There was a problem updating the buffer size"

2.1.10 Comando "getNConnections"

- Muestra el número de conexiones concurrentes en ese momento
- Argumentos: NULL
- Respuestas:
  - "[SUCCESS] The number of total connections is: x"
  - "[FAILURE] There was an error fetching the number of connections"

2.1.11 Comando "getNUsers"

- Muestra el número de usuarios concurrentes en ese momento
- Argumentos: NULL
- Respuestas:
  - "[SUCCESS] The number of active connected users is: x"
  - "[FAILURE] There was an error fetching the number of active connected users"

2.1.12 Comando "setMediaRange"

- Determina el array de *media types* a ser filtradas por el parser de MIME

- Argumentos: Array de media types
- Respuestas:
    - "[SUCCESS] Media range updated!"
    - "[FAILURE] There was a problem updating the media range"

2.1.13 Comando "getMediaRange"

- Muestra los *media types* que están siendo filtrados por el parser de MIME
- Argumentos: NULL
- Respuestas:
    - "[SUCCESS] Media range is: *string*"
    - "[FAILURE] There was a problem getting the media range"

2.1.14 Comando "setReplaceMessage"

- Reemplaza el *replace message* en el server
- Argumentos: String del nuevo replace message
- Respuestas:
    - "[SUCCESS] Replace message updated!"
    - "[FAILURE] There was a problem setting the replace message"

2.1.15 Comando "getReplaceMessage"

- Muestra el *replace message* del server
- Argumentos: NULL
- Respuestas:
    - "[SUCCESS] Replace message is: *string*"
    - "[FAILURE] There was a problem getting the replace message"

2.1.16 Comando "addReplaceMessage"

- Appendea el mensaje al *replace message* del server
- Argumentos: String del mensaje a appendear
- Respuestas:
    - "[SUCCESS] New replace message added!"
    - "[FAILURE] There was a problem adding the replace message"

2.1.17 Comando "setErrorPathFile"

- Muestra el número de usuarios concurrentes en ese momento
- Argumentos: NULL
- Respuestas:
    - "[SUCCESS] Error path file updated!"
    - "[FAILURE] There was a problem replacing the error path file"

2.1.18 Comando "getErrorPathFile"

- Muestra el número de usuarios concurrentes en ese momento
- Argumentos: NULL
- Respuestas:
    - "[SUCCESS] The error path is: *string*"
    - "[FAILURE] There was a problem getting the error path"

2.2 - Notación ABNF del protocolo

```
request         = op-code e-tag encoding data-length data
response        = resp-code e-tag encoding data-length data

op-code         = %x00000001-00000015
resp-code       = "0" / "101" / "200" / "601" / "702" / "801"
                / "1001" / "1701" / "1901" / "2001" / "2101"
e-tag           = %x00000000-0000000A
encoding        = %x00000001-00000003
data-length     = %x00000000-000010000
data            = kOCTET
```

Comentario: la "k" de la variable *data* se corresponde con un int representado por la variable *data-length***.**

2.3 - Descripción de los campos

op-code: El código del comando elegido por el administrador. Int de 4 bytes. Valores del 1 al 21.

- 1: Login
- 2: Getnbytes
- 3: GetNusers
- 4: GetNconnections
- 5: GetBufferSize
- 6: UpdateBufferSize
- 7: UpdateCredential
- 8: SetFilter
- 9: GetFilter
- 10: SetFilterCommand
- 11: GetStatistics
- 12: GetFilterCommand
- 13: GetAverageBufferSize
- 14: GetActiveConnections
- 15: SetMediaRange
- 16: GetMediaRange
- 17: SetReplaceMsg
- 18: GetReplaceMsg

- 19: AddReplaceMsg
- 20: SetErroFilePath
- 21: GetErroFilePath

resp-code: Código de respuesta del server. Int de 4 bytes. 11 opciones:

- 0: Server error
- 101: Login error (credenciales erróneas)
- 200: OK
- 601: Buffer size modification error (encoding o etag inválidos)
- 702: Credentials modification error (encoding o etag inválidos)
- 801: Filter status modification error (encoding o etag inválidos)
- 1001: Filter program modification error (encoding o etag inválidos)
- 1701: Replace message modification error (encoding o etag inválidos)
- 1901: Replace message appending error (encoding o etag inválidos)
- 2001: Error file path modification error (encoding o etag inválidos)
- 2102: Error file path obtention error (encoding o etag inválidos)

e-tag: Representa el id de un recurso, si éste se ve modificado, el e-tag cambia también. Int de 4 bytes.

encoding: El tipo de datos de la sección *data*. Int de 4 bytes, (credential_type = 1, text_type = 2, int_type = 3)

data-length: Representa la cantidad de bytes en *data*. Int de 4 bytes.

data: Vector con los valores que se pasaron a través de los argumentos. Cada uno es null-separated. Máximo de 4096 bytes.

3 - Testing y debbuging:

Se intentó seguir la filosofía de XP (extreme programming: programación de a pares, entre otras prácticas) y llevar a cabo TDD (Test Driven Development), realizando tests unitarios para el desarrollo guiado por testing, implementándose test unitarios de los distintos componentes del sistema. Para ello, se utilizó el Framework CUTest (Copyright 2003 Asim Jalis.) (https://github.com/aiobofh/cutest.git).

Para el debugging dinámico y estático se utilizaron las herramientas de **valgrind** y **cppcheck**

Se utilizó el comando:

```
valgrind --leak-check=full --show-leak- kinds=all -v <archivoEjecutable>
```

para revisar los leaks de memoria.

Problemas durante el diseño e implementación

1 - Pipelining

- Dificultades al poder mapear la respuesta de un comando envíado, ya que se enviaban varios al mismo tiempo, y se debían corresponder las respuestas con ellos. Se tuvo que almacenar un "esqueleto" de comando, para poder llenarlo con la respuesta cuando se obtuviese desde el origen.
- Cuando no soporta pipelining el origen, fue difícil coordinar el envío del comando, la espera de la respuesta, y luego recién ahí el siguiente comando.

2- Multilíneas

Identificar cuánto leer de la respeusta del comando. En síntesis, se encontraron dificultades con saber cuándo la respuesta es multilínea y cuándo no.

3 - Concurrencia de comunicación

No comunicarse con los distintos *endpoints* (cliente, origen, filter) por turnos, hacerlo al "mismo tiempo" y según la necesidad del momento.

4- Escuchar en IPv4 e IPv6 al mismo tiempo

A partir de las pruebas del guión de pruebas surgió la incertidumbre de cómo lograr escuchar en todas las interfaces y tanto para IPv4 como IPv6. Finalmente, se optó por escuchar en "0.0.0.0" para el defecto de todas las interfaces (si se esuchara tanto en IPv4 como IPv6 sería algo completamente fuera de nuestro control y plataforma-dependiete) y en "127.0.0.1" para los casos de *loopback-only* (si esto resulta en escuchar tanto en "127.0.0.1" como en "::1" también es algo ajeno a nosotros y plataforma-dependiente).

5- Escapar puntos

Como se postergó hasta último momento la incorporación y soporte del "byte-sutffing" de los comandos multilínea POP3, se quiso integrar un parser para "desescapar" los puntos escapados para no ser confundidos con un *termination octet* antes de enviar el cuerpo de un mail como respuesta multilínea POP3. Esta respuesta sería resultado de un comando **retr** al programa de filtrado y se tuvo que procesar en otro parser para volver a escapar los puntos correspondientes antes de enviar el contenido transformado al cliente. Para resolver este problema se contemplaron varias alternativas con los docentes y ayudante de la cátedra. Finalmente se optó por la opción que menos modificaba la lógica vigente y que además involucraba principalmente manejo de mecanismos de IPC, pero que resultó más prometedora para futuras expansiones. Esta alternativa se basa en crear un proceso *wrapper* que lee por STDIN y maneja el *byte stuffing* de POP3 para desescapar los puntos y escribir sobre la entrada de otro proceso (producto de otro par **fork()** y **execl()**, pero este segundo siendo un **execl()** compatible con **system()** para el programa filtro) y luego el *wrapper* lee del pipe redirigido a la salida del filtro, vuelve a procesar el *byte-stuffing* y finalmente envía de regreso al proxy.

Limitaciones de la aplicación

1 - Concurrencia y disponibilidad:

La implementación utiliza la función *pselect* que puede verificar hasta 1024 *file descriptors* al mismo tiempo. Se utilizan dos de ellos para aceptar las conexiones entrantes (clientes regulares o administradores), por lo que quedan 1022 *file descriptors* libres para aceptar clientes, ya sean administradores o regulares , servidores y pipes.

Cada cliente puede ocupar como máximo 6 *file descriptors* y como mínimo 1 en la última etapa de la conexión. Cuando se inicia la conexión se necesitan 4 *file descriptors* para lograr la conexión entre cliente y origen. En conclusión, se pueden tener 1018 + 1 cliente (en el caso de que sean todos clientes regulares).

Posibles extensiones

1 - Pipelining sobre los comandos del administrador

Así como para el cliente regular se soporta pipelining, se podría llegar a implementar la misma función pero para los clientes que se conectan al proxy que quieran administrarlo.

2 - Encriptación de las credenciales del administrador

En la implementación actual, las credenciales de acceso al proxy del administrador se envían en texto plano, y por lo tanto son fácilmente obtenidas por algún ataque malicioso. Se podría implementar un sistema de encriptación como por ejemplo el SHA 256 SUM.

3 - Datos específicos para el administrador

Se podría mejorar la cantidad y especificidad de los datos a los que el administrador puede acceder. Por ejemplo, se podría emplear un comando que deje ver la cantidad de datos intercambiados con el proxy de un cliente en específico.

4- Mejorar el programa de filtros

Dado que se le dedicó poco tiempo al filtro strip MIME por no tenerlo como una prioridad desde el principio del proyecto, se podría arreglar y modificar en abundancia el transformador. La implementación actual dista de ser la versión ideal a implementar en el sistema.

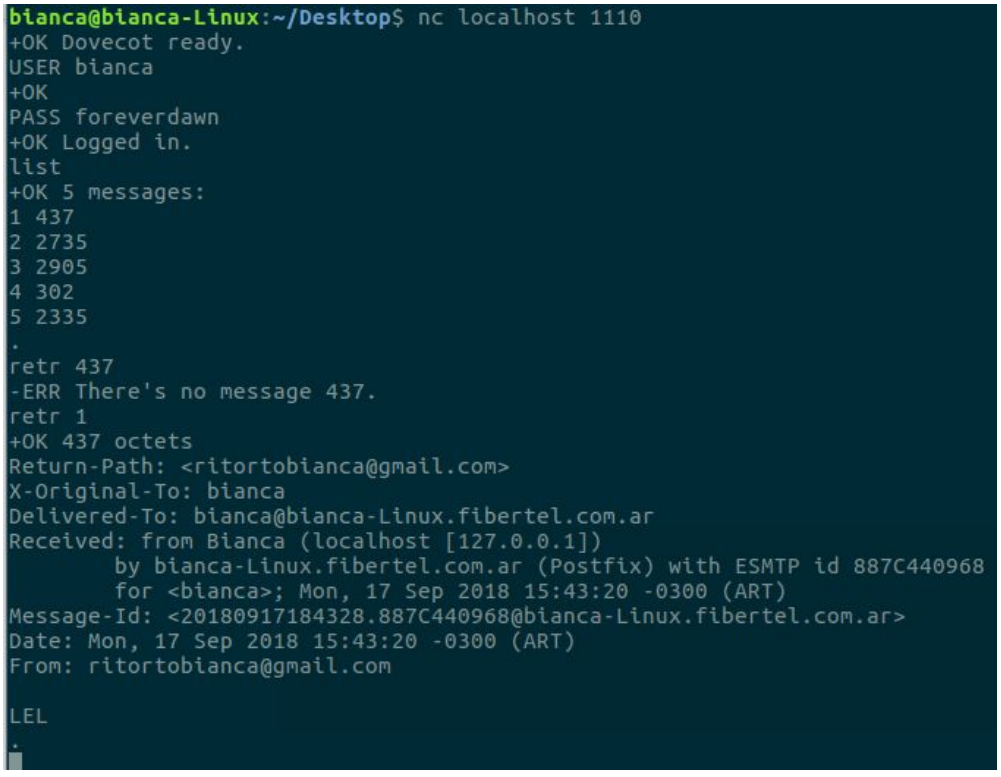
Conclusiones

Durante el diseño, desarrollo e implementación de este trabajo práctico especial no solo reforzamos conceptos de asignaturas previa como Arquitectura de Computadoras o Sistemas Operativos, sino que se lograron introducir conceptos nuevos y fascinantes del mundo de los protocolos de comunicación. Aprender sobre el funcionamiento de algo tan habitual como Internet, o enviar correos electrónicos, nos permitió no solo mejorar nuestros conceptos sobre el funcionamiento de estos sistemas sino que ponerlo en práctica, con el motor del trabajo en equipo.

Tuvimos que aprender a separar las tareas, ya que era un proyecto tan abarcativo y que requería de conocimientos de más de un sector de la materia. Por ejemplo, se distinguió entre el desarrollo del protocolo del administrador y de la implementación del proxyPOP3. Luego, se tuvo que integrar el trabajo, que presentó nuevas dificultades que se tuvieron que superar como equipo.

Ejemplos de prueba

Utilización de dovecot a través del proxy:



Guía de instalación

Para poder correr el proyecto, uno se debe hacer make dentro de la carpeta /pc-2018b-04/src y se compilaran y linkeditaran los tres ejecutables, seguido con la ejecución de los tests y del proxy invocado con los siguientes argumentos:

```
./pop3filter/pop3filter.out -t cat 127.0.0.1
```

Los ejecutables se encuentran en:

- ./pc-2018b-04/src/pop3filter/pop3filter.out
- ./pc-2018b-04/src/pop3ctl/pop3ctl.out
- ./pc-2018b-04/src/stripmime/stripmime.out

Instrucciones para la configuración

Para correr el cliente **pop3ctl** se debe correr el archivo ejecutable arrojado por la ejecución de make que se encuentra en la carpeta pop3ctl y se pueden utilizar las opciones antes mencionadas -o (puerto) y -L (dirección).

Ejemplos de configuración y monitoreo

Actualización del tamaño del buffer (y consulta del tamaño actual del buffer):

```
>> getbuffersize
Getting buffer size...
[SUCCESS] The buffer size is: 4000 bytes
>> setbuffersize 3999
setting buffer size..
[SUCCESS] Buffer size updated!
>> getbuffersize
Getting buffer size...
[SUCCESS] The buffer size is: 3999 bytes
>>
```

Actualización de credenciales de administrador:

```
>> setcredential admin2
New credentials were sent..
[SUCCESS] Credentials updated!
```

```
PASS: admin2
Sending credentials...
Waiting response...

**Successfull login**
```

Actualización de filtros:

```
>> getfilter
Trying to fetch filter...
[SUCCESS] The filter is: ON
>>
```

```
>> setfilter N
Trying to set filter...
[SUCCESS] Filter updated!
>> GETFILTER
Trying to fetch filter...
[SUCCESS] The filter is: OFF
```

Ver métricas del proxy:

```
>> viewmetrics
[SUCCESS] Metrics:
        Number of Bytes: 0
        Buffer Size:  4000
        Avr Buffer Size: 0
        Total Users: 0
```

Documento de diseño del proyecto

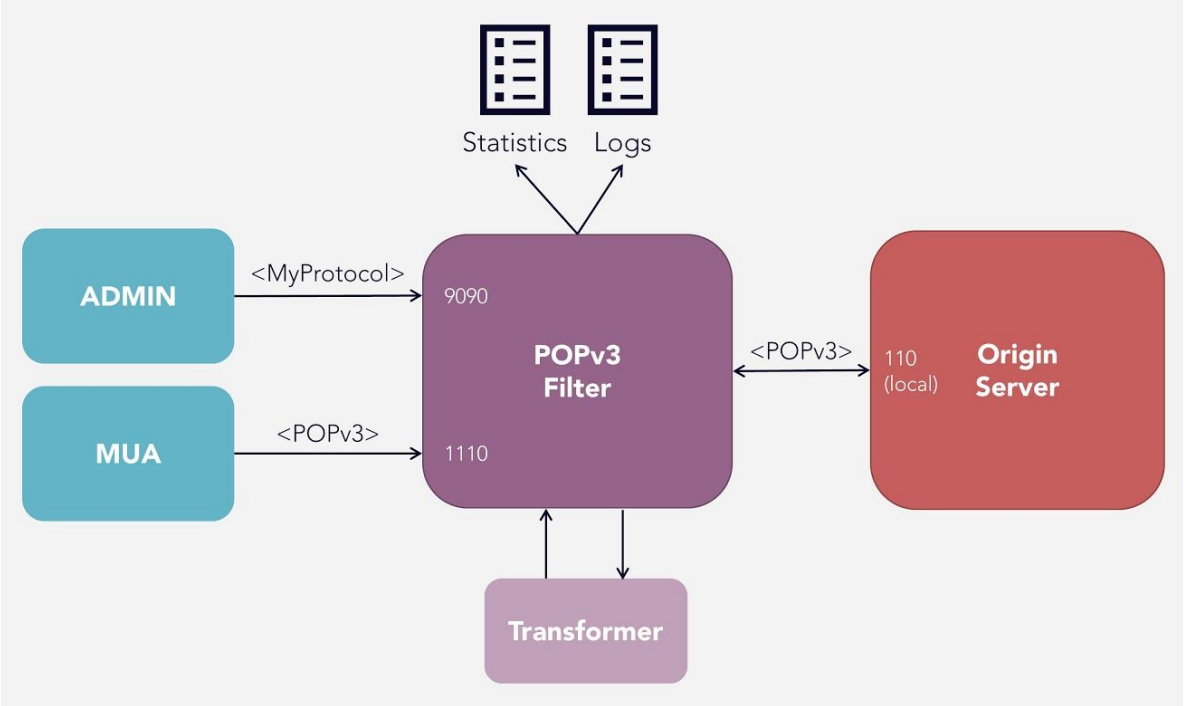


Figura 1: Diseño general del sistema

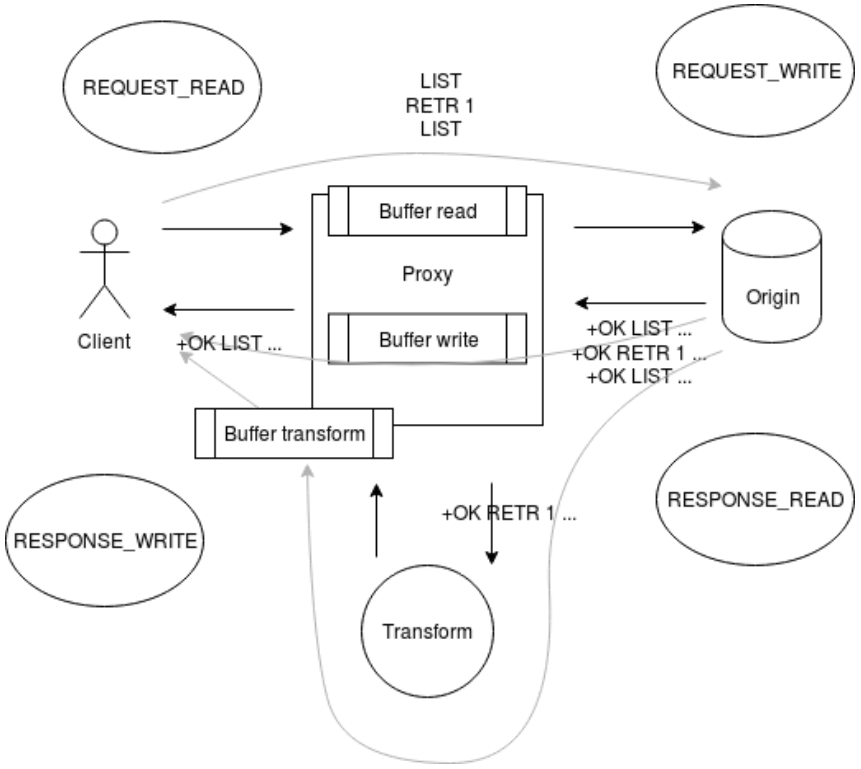


Figura 2: Diseño más detallado del servidor Proxy POP3