



**72.07 - Protocolos de Comunicación
2^{do} Cuatrimestre 2021
Trabajo Práctico Especial 2**

Grupo 1

Patrick Malcolm Dey (59290)
Matías Federico Lombardi (60527)
Matías Enrique Pavan (58296)
Santos Matías Rosati (60052)

Índice

Protocolos y aplicaciones desarrolladas	3
1 - Aplicaciones	3
1.1 - pop3filter: Servidor proxy POP3	3
1.2 - pop3ctl: Cliente PMP sobre UDP para administrar el proxy	4
1.3 - Protocolo PMP (Proxy Management Protocol)	5
Problemas durante el diseño e implementación	6
1 - Retry para la conexión con el origin	6
2 - Pipelining	6
3 - Escuchar conexiones IPv4 e Ipv6 simultáneamente	7
4 - Union	7
5 - Transform	7
6 - Multilíneas	7
7 - Liberación de recursos	7
Limitaciones de la aplicación	8
1 - CAPA	8
2 - Disponibilidad y concurrencia	8
3 - Tamaño de los buffers	8
Posibles extensiones	8
1 - Protocolo de transporte del management	8
2 - Comandos POP3	8
3 - Pipelining en PMP	9
4 - Verificación de errores en PMP	9
5 - Logueo de la aplicación	9
Conclusiones	9
Ejemplos de prueba	9
Pruebas de Estrés	13
Guía de instalación	14
Instrucciones para la configuración	14
Ejemplos de configuración y monitoreo	15
Análisis de código	16
Documento de diseño del proyecto	16

Protocolos y aplicaciones desarrolladas

1 - Aplicaciones

Se diseñaron dos aplicaciones (ambas documentadas en los archivos mencionados en el **README**) y un protocolo. Las aplicaciones implementadas fueron:

- ❖ pop3filter - Servidor proxy **POP3**
- ❖ pop3ctl - Cliente **UDP** para administrar el proxy **POP3**

1.1 - pop3filter: Servidor proxy POP3

Es un servidor concurrente con entrada/salida multiplexada no bloqueante. Para el desarrollo del servidor se utilizó código fuente provisto por Juan Francisco Codagnone para abstraer la implementación de un multiplexor y un motor de máquinas de estado para manejar los eventos asociados al multiplexor.

Mediante el multiplexor se pueden registrar los distintos *fds* para diversos intereses (escritura, lectura o ninguno). Nuestro *proxy* recibe por argumento la dirección del servidor **POP3** al que se va a conectar (puede ser en forma **IPv4**, **IPv6** o un **FQDN**). Si lo que se pasó por argumento fue un **FQDN**, el *proxy* intentará resolverlo antes de conectarse mediante una llamada a *getaddrinfo(3)*. Como esta función es bloqueante (ya que debe realizar una consulta **DNS** y esperar el resultado) se optó por crear un hilo que realice el llamado de esta función, de esta manera, el hilo principal de la aplicación no se bloqueará.

El servidor también recibe diversos parámetros por línea de comando bajo el estilo del estándar **POSIX**. Para ello se utilizó la función *getopt(3)*.

El primer estado al que se llega es al de **RESOLVE_ORIGIN** el cual analiza si la dirección del servidor es una IP o un **FQDN** y en base a ello se intenta conectar directamente o intenta resolver el **FQDN** utilizando otro hilo bloqueante. En ambos casos se pasa a un estado **CONNECT** el cual se encarga de conectarse al *origin* mediante una única función registrada para escritura (ya que al ser no bloqueante la llamada a *connect(2)*, en caso de no haber terminado el **Handshake**, retornará **EINPROGRESS**). En el caso de un **FQDN** se itera por **todas** las direcciones obtenidas en el llamado a *getaddrinfo(3)*, ya que si una falla, se debe intentar con la siguiente. Si la conexión es exitosa se pasa a un estado **HELLO** que lee el saludo inicial del *origin*.

En el estado **HELLO**, es el servidor el que habla primero. Se implementa una estrategia por turnos, en la que se lee el saludo completo del *origin* (al manejar un tamaño de buffer de 1024, nos garantizamos que podremos leer todo debido a que la máxima longitud del saludo inicial es de 512 bytes, según el rfc 2449 “*The maximum length of the first line of a command response (including the initial greeting) is unchanged at 512 octets (including the terminating CRLF)*” y recién al terminar se empieza a copiar al cliente).

Al completarse este saludo se transiciona al estado de **REQUEST** el cual lee los pedidos del cliente y los escribe en el origen. Puede transicionar a dos estados: **CAPA** o **RESPONSE**. Va a transicionar a **CAPA** cuando el usuario haga un pedido de las capacidades del origen y en este estado nos

encargamos de anexar la capacidad de **PIPELINING** en el caso de que el origen no la soporte para informarle al cliente. En el caso de que el cliente realice otro pedido que no sea el de las capacidades del origen se pasará al estado de **RESPONSE** encendiendo un flag para informarle al mismo si el pedido que solicitó el cliente puede ser multilínea. Al leer los pedidos del cliente también se utiliza una estrategia por turnos, en la que leemos lo máximo posible del buffer. Nos garantizamos que al menos un comando completo será parseado (nuevamente, según el rfc 2449 “*The maximum length of a command is increased from 47 characters (4 character command, single space, 40 character argument, CRLF) to 255 octets, including the terminating CRLF.*”). Sin importar si el *origin* soporta **PIPELINING** o no, utilizamos una *queue* donde encolamos los comandos y se envían de a uno por vez (es decir, al tener en la *queue* al menos un comando, se envía y espera una respuesta antes de mandar el siguiente).

En el estado de **RESPONSE** dependiendo del comando actual que fue enviado al *origin* tomamos distintos caminos de acción. Si lo que se envió fue un **RETR <número>** y la **transformación está activada**, se crea un proceso hijo mediante las llamadas a *fork(2)* y *execle(3)* y se reemplazan los *fds STDIN* y *STDOUT* (mediante la función *dup2(2)*) del hijo por las puntas de un *pipe(2)* con el que se comunica el proceso padre, de esta manera, se irán flusheando los bytes que leemos del origen hacia el transform (eliminando los puntos que se agregan debido a la naturaleza del protocolo) hasta la llegada del indicador de fin de multilínea y se irá enviando la información al cliente (recuperando el formato eliminando los puntos adicionales). En el caso de que la transformación no esté activada se procederá a enviarle la respuesta del origen al cliente normalmente. En caso de que la transformación falle en su creación, se *flushea* toda la respuesta (no se envía al cliente) y se devuelve un mensaje de error indicando que la misma falló.

Por otro lado, se encuentran los estados **FAILURE**, **FAILURE_WITH_MESSAGE** y **DONE** como estados terminales para casos erróneos o exitosos. En el caso del **FAILURE_WITH_MESSAGE**, envía un mensaje de error al cliente antes de pasar al estado **FAILURE**.

Se utilizaron en total 5 buffers, uno del cliente hacia el origen, uno del origen al cliente, otros dos de lectura y escritura hacia y desde la transformación y un último de escritura desde el proxy hacia el cliente. Si la transformación no está activada, sus respectivos buffers tampoco lo estarán, y además si no se necesita transformar la respuesta del origen, el buffer de *origin* a cliente será el mismo que el del *proxy* al cliente. Se utilizaron asignamientos dinámicos para las estructuras de datos pero en lo que respecta a las estructuras utilizadas por el *proxy*, para mantener el contexto y los datos de los diferentes estados se armó una pool. Esto permitió mejorar el manejo de asignación costosa de la estructura pop3 que almacena parsers, buffers, *queue*, entre otros.

1.2 - pop3ctl: Cliente PMP sobre UDP para administrar el proxy

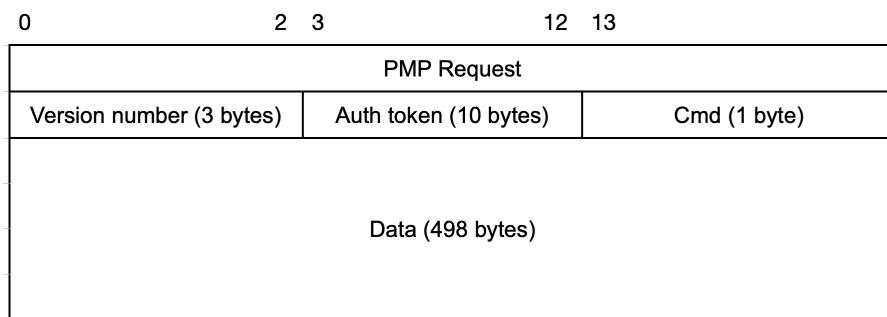
La aplicación cliente permitirá al usuario administrar la configuración del *proxy pop3filter*. Este se comunicará con el servidor mediante **UDP** enviando en el datagrama de request la información solicitada por el protocolo: la versión, el token de autorización, el comando deseado y, si lo requiere, data que el servidor necesitará para ejecutar el comando enviado. El administrador deberá ingresar por entrada estándar el comando deseado con los parámetros necesarios y la aplicación se encargará de enviar la solicitud al servidor en el formato correcto. El servidor responderá un datagrama con la versión, el status code correspondiente y opcionalmente un mensaje de respuesta a la acción realizada.

Al igual que para *pop3filter*, el cliente recibe diversos parámetros por línea de comando bajo el estilo del estándar **POSIX**. Para ello se utilizó la función *getopt(3)*. Es importante mencionar que el cliente aceptará como argumento el servidor al que quiere conectarse en formato **IPv6**, **IPv4** o **FQDN**.

1.3 - Protocolo PMP (Proxy Management Protocol)

Es un protocolo de aplicación que utiliza **UDP** como protocolo de transporte con paquetes de un tamaño máximo de 512 bytes. Es un protocolo diseñado para obtener distintas métricas del *proxy* **POP3** y además realizar ciertos cambios de configuración. El paquete que envía el cliente está modelado en la siguiente estructura: cuenta con 3 bytes para la versión del protocolo, 10 bytes para el token de identificación, 1 byte para indicar el comando que se quiere ejecutar y el resto para los datos necesarios para ejecutar el comando del lado del servidor, si es que los hay. Los comandos deben finalizar con **CRLF**. Entre los comandos disponibles, ofrece:

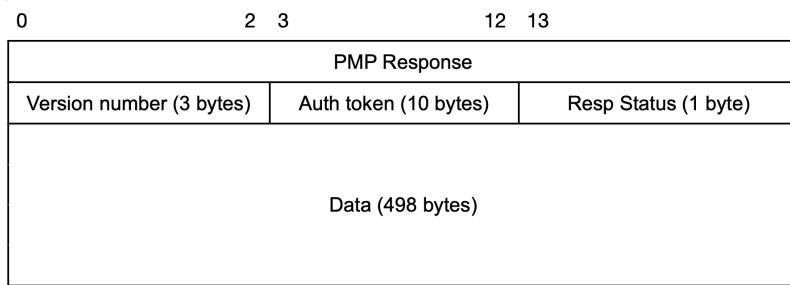
- **STATS**: que devuelve algunas métricas del *proxy*.
- **GET_TIMEOUT**: devuelve el máximo tiempo de espera del servidor.
- **SET_TIMEOUT [timeout]**: cambia el máximo tiempo de espera del servidor.
- **GET_FILTER_CMD**: devuelve el comando de filtro ejecutado por el servidor.
- **SET_FILTER_CMD [command]**: cambia el comando de filtro ejecutado por el servidor.
- **GET_ERROR_FILE**: cambia el archivo al que se loguean los errores del transform.
- **SET_ERROR_FILE [error file]**: cambia el archivo al que se loguean los errores del transform.



Por otro lado, el servidor enviará paquetes del mismo tamaño modelados en la estructura *t_admin_resp*. Esta cuenta también con 3 bytes para la versión, 1 byte para indicar el código de estado de la respuesta y el resto para, si es necesario, enviar un mensaje al cliente. El datagrama de respuesta también estará siempre finalizado por un **CRLF**.

Los status de respuesta posibles son:

- **OK**: El comando se ejecutó con éxito.
- **UNSUPPORTED_COMMAND**: El comando enviado es invalido.
- **UNSUPPORTED_VERSION**: La versión enviada es inválida.
- **INVALID_ARGS**: Longitud de paquete invalida.
- **UNAUTHORIZED**: El token enviado no es válido.
- **INTERNAL_ERROR**: Ocurrió un error interno del lado del servidor.



```

typedef struct t_admin_req {
    uint8_t      version[VERSION_SIZE];
    uint8_t      token[TOKEN_SIZE];
    uint8_t      command;
    uint8_t      data[DATA_SIZE];
} t_admin_req;

typedef struct t_admin_resp {
    uint8_t      version[VERSION_SIZE];
    uint8_t      status;
    uint8_t      data[DGRAM_SIZE - VERSION_SIZE - 1];
} t_admin_resp;

```

Donde **VERSION_SIZE** es 3, **TOKEN_SIZE** es 10, **DATA_SIZE** es 498 y **DGRAM_SIZE** es 512

Problemas durante el diseño e implementación

1 - Retry para la conexión con el *origin*

Al estar utilizando un método por turnos (en este caso, por intereses de los *fd*) se presentó un problema cuando realizamos una conexión a una posible address del origen. Cuando el método connect retornaba **EINPROGRESS** y nos suscribimos al *fd* del *origin* para escritura, una vez dentro del handler de escritura para este *fd*, era posible que la conexión haya tenido un problema por lo que deberíamos volver al estado anterior para intentar conectarnos con otro address (cosa que era imposible porque, en ese momento, los intereses de los *fd* no están seteados para poder volver al estado de conexión). Se pudo solucionar invocando una función fuera de la máquina de estados para hacer un retry de la conexión.

2 - Pipelining

La principal dificultad estuvo en que existe la posibilidad de que no le llegue al *proxy* un paquete completo (ej: “CAPA\r\nUS”), debido a que al utilizar **TCP** como protocolo de transporte, es un flujo de bytes, lo cual causaría que se pierda/parsee incorrectamente todo mensaje que llegue “cortado”.

Para solucionar esto se implementó una queue, en la que se encolan comandos una vez que se *parsean*, de esta manera se desencola un comando por vez y se aguarda la respuesta del *origin server*.

3 - Escuchar conexiones IPv4 e Ipv6 simultáneamente

Para solucionar esta dificultad se optó por utilizar dos sockets, uno **IPv4** y otro **IPv6**, en este último se habilita la opción **IPV6_V6ONLY** (mediante *setsockopt(2)*).

4 - Union

Citando la definición de union: “*A union is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time.*”. El problema radicó en que se tenían dos parsers dentro de un union (uno era el parser unilínea “\r\n” y otro el de multilínea “\r\n.\r\n”) pero en runtime toda respuesta multilínea estaba siendo procesada con el parser unilínea porque el union tenía el valor de ese parser en ese momento.

5 - Transform

Inicialmente esperábamos a leer toda la respuesta (es decir, hasta un “\r\n.\r\n”) de un **RETR** del origen para enviarselo al transform. Nos dimos cuenta rápidamente que esto era un error ya que era posible que la respuesta del *origin* supere el tamaño de nuestro buffer por lo que se perderían todos los bytes iniciales que no entran en nuestro buffer. Esto se soluciona enviando al transform la información del mail cuando se llene nuestro buffer (de cierta manera lo “flusheamos”) y repetimos esta acción hasta llegar al el indicador de final de una respuesta multilínea. Luego otra dificultad que se presentó fue que no se contemplaba el caso en el que el buffer del Sistema Operativo se llenara, ya que sólo arrancamos a leer la respuesta procesada de la aplicación externa una vez que habíamos escrito todo. La solución propuesta fue llegar al estado **TRANSFORM** con los dos intereses (escritura y lectura del *pipe*) activos.

6 - Multilíneas

Como no es trivial poder saber si una respuesta es multilínea o no dinámicamente optamos por proveer información al *proxy* para que este sepa cuáles respuestas son (o podrían ser, dependiendo el código de respuesta del origen) multilínea y cuáles no.

7 - Liberación de recursos

Se presentó un problema donde no se estaban liberando los recursos adecuadamente cuando un cliente se desconectaba forzosamente (ej: **CTRL+C**) o enviaba el comando “**QUIT**”. Esta problemática también surgió al no estar familiarizados con el uso del atributo *references* del selector.

Limitaciones de la aplicación

1 - CAPA

Asumimos que en la respuesta de **CAPA** la primera ocurrencia de “\r\n.” es cuando el mismo ya listó todas sus capacidades (es decir, no aparece un “\r\n.” a menos que no sea al final, por lo que no puede haber líneas que empiecen con un punto).

2 - Disponibilidad y concurrencia

La implementación de selectores utiliza *pselect(2)* que solo puede verificar hasta 1024 *fds* al mismo tiempo. Utilizamos dos para aceptar conexiones entrantes y dos para el management y a su vez cerramos **STDIN**, lo que nos deja con 1021 *fds* libres (a menos que la dirección que utilice el *proxy/admin* sea sólo **IPv4/IPv6**, en cuyo caso contamos con más *fds*) para aceptar clientes. Como en el peor de los casos, cada cliente tiene como máximo 4 *fds* (1 para el cliente, 1 para el *origin*, y 2 para los *pipes*), esto nos garantiza una disponibilidad máxima de alrededor de 250 clientes en simultáneo que ocupen toda la funcionalidad que provee el *proxy*.

3 - Tamaño de los buffers

En caso de tener que ejecutar una transformación externa, asumimos que el tamaño de los buffers del sistema operativo serán mayores que los que manejamos dentro de la aplicación, ya que de esta manera nos garantizamos que podremos escribir el buffer completo.

Posibles extensiones

1 - Protocolo de transporte del management

- Cambiar **UDP** por **SCTP** o **TCP** para el manejo del management ya que presenta grandes vulnerabilidades de seguridad, sobre todo si se extiende la funcionalidad del mismo para realizar operaciones más críticas.

2 - Comandos POP3

- Una posible extensión sería que el *proxy* sea consciente de qué comandos son correctos antes de enviarlos al *origin server*, por ejemplo si no tienen la cantidad correcta de parámetros. De esta manera, podríamos responder inmediatamente en caso de un comando erróneo (la implementación actual es un pasamanos, excepto en los casos en los que se ejecuta el comando **RETR** y hay una transformación activa).

3 - Pipelining en PMP

- En la implementación actual del protocolo solo se permite enviar un comando por *datagrama*, una futura extensión podría permitir varios comandos dentro del mismo paquete.

4 - Verificación de errores en PMP

- En la implementación actual se asume que para los comandos en los que se cambia la configuración del servidor (el archivo de errores y la transformación) los datos son correctos (es decir, no llegan solo espacios por ejemplo). Una futura extensión podría validar estos casos.

5 - Logueo de la aplicación

- Se podría setear un archivo en el cuál se tendrán todos los registros para un administrador (de manera similar al archivo de errores para el proceso *transform*). En la implementación actual se loguea a salida estándar y de errores.

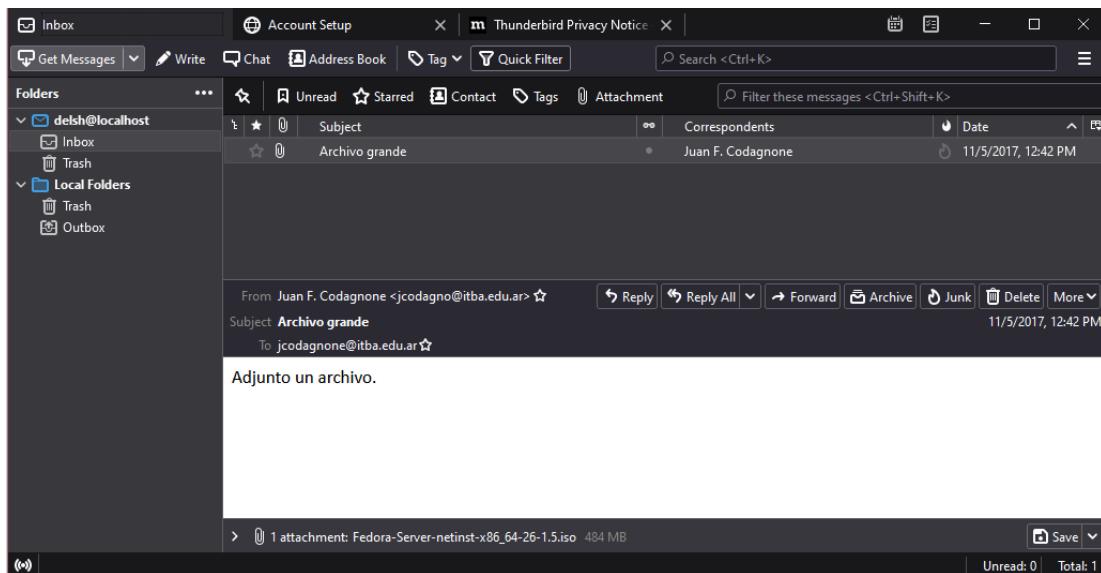
Conclusiones

El primer **TPE** de la materia nos permitió familiarizarnos con la programación de *sockets* en C, además del manejo de buffers que se utilicen para lectura y escritura y parsers para validar los comandos entrantes. Junto con este conocimiento y la adición de un selector (que implementa de forma transparente la manera en la que lo hace, puede utilizar *select(2)*, *pselect(2)* o cualquier implementación deseada) pudimos completar este **TPE** en el que diseñamos un protocolo y desarrollamos un *proxy POP3* y una aplicación cliente que implemente el protocolo mencionado. El mayor obstáculo fue la cantidad de investigación que tuvimos que realizar, desde funcionamiento básico de sockets, distintas implementaciones en un entorno **UNIX**, hasta los diferentes **RFC** pertinentes al **TPE (POP3, Internet Message, etc)**.

Ejemplos de prueba

Nota: todos los casos de prueba posibles que no involucren permisos o la instalación de nuevos programas fueron realizadas en pampero bajo un túnel remoto:

(ssh -R58296:localhost:110 user@pampero.itba.edu.ar)

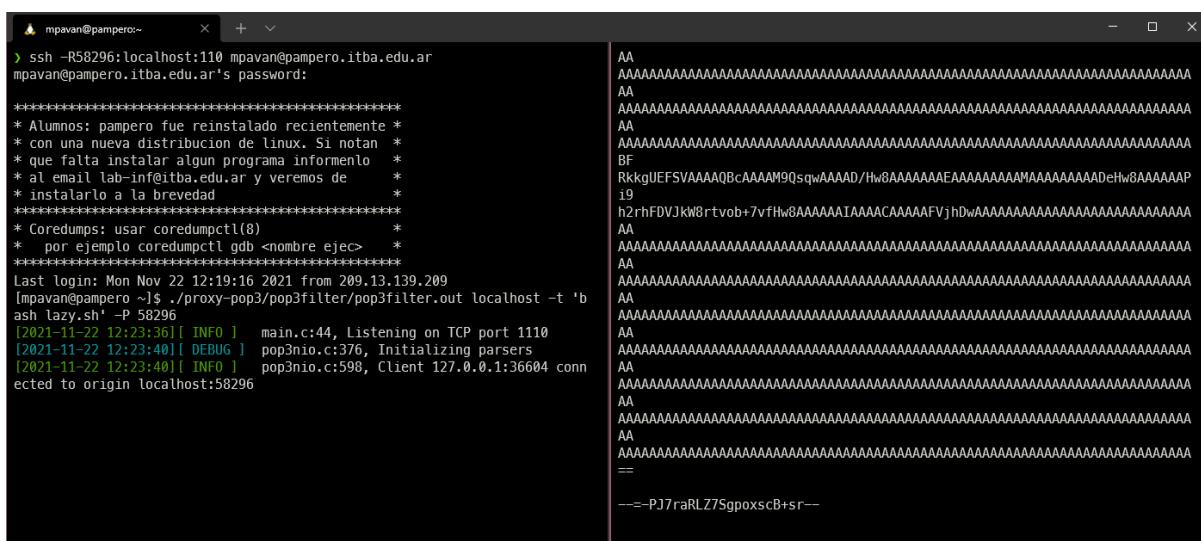


Prueba 1 - MUA vía proxy para el retrieve de un mail grande

```
> ssh -R58296:localhost:110 mpavan@pampero.itba.edu.ar
mpavan@pampero.itba.edu.ar's password:
*****
* Alumnos: pampero fue reinstalado recientemente *
* con una nueva distribucion de linux. Si notan   *
* que falta instalar algun programa informenlo   *
* al email lab-inf@itba.edu.ar y veremos de      *
* instalarlo a la brevedad                      *
*****
* Core dumps: usar coredumpctl(8)                 *
* por ejemplo coredumpctl gdb <nombre ejec>      *
*****
Last login: Mon Nov 22 12:19:16 2021 from 209.13.139.209
[mpavan@pampero ~]$ ./proxy-pop3/pop3filter/pop3filter.out localhost -t 'bash lazy.sh' -P 58296
[2021-11-22 12:23:36] [INFO ] main.c:44, Listening on TCP port 1110
[2021-11-22 12:23:40] [DEBUG ] pop3nio.c:376, Initializing parsers
[2021-11-22 12:23:40] [INFO ] pop3nio.c:598, Client 127.0.0.1:36604 connected to origin localhost:58296
```

```
[mpavan@pampero ~]$ nc -C localhost 1110
+OK Dovecot (Debian) ready.
USER delsh
+OK
PASS asd123
+OK Logged in.
list
+OK 1 messages:
1 694489434
.
retr 1
```

Prueba 2.1 - Invocación Lazy Response sobre mail grande en pampero vía túnel remoto



Prueba 2.2 - Terminación Lazy Response sobre mail grande en pampero vía túnel remoto

```
nc -l 1110
> ./pop3filter.out 127.0.0.1 -t 'bash stuf.sh'
[2021-11-22 12:36:00] [ INFO ]  main.c:44, Listening on TCP port 1110
[2021-11-22 12:36:12] [ DEBUG ]  pop3nio.c:376, Initializing parsers
[2021-11-22 12:36:12] [ INFO ]  pop3nio.c:600, Client 127.0.0.1:48336 connected to origin 127.0.0.1:110
> nc -C localhost 1110
+OK Dovecot (Debian) ready.
USER delsh
+OK
PASS asd123
+OK Logged in.
list
+OK 1 messages:
1 694489434
.
retr 1
+OK
X-Header: true

..hola
..
.
```

Prueba 3 - Byte stuffed sobre mail grande

```
nc -l 1110
> ./pop3filter.out -t 'bash envs.sh' 127.0.0.1
[2021-11-22 12:40:18] [ INFO ]  main.c:44, Listening on TCP port 1110
[2021-11-22 12:40:24] [ DEBUG ]  pop3nio.c:376, Initializing parsers
[2021-11-22 12:40:24] [ INFO ]  pop3nio.c:600, Client 127.0.0.1:48340 connected to origin 127.0.0.1:110
> nc -C localhost 1110
+OK Dovecot (Debian) ready.
user delsh
+OK
pass asd123
+OK Logged in.
list
+OK 1 messages:
1 694489434
.
retr 1
+OK
X-Version: 0.0.0
X-User: delsh
X-Origin: 127.0.0.1

Body.
.
```

Prueba 4 - Seteo de variables de entorno sobre mail grande

```
> ./pop3filter.out -t 'touch /tmp/1 && cat' 127.0.0.1
[2021-11-22 12:47:53] [ INFO ]  main.c:39, Listening on TCP port 1110
[2021-11-22 12:48:05] [ DEBUG ]  pop3nio.c:376, Initializing parsers
[2021-11-22 12:48:05] [ INFO ]  pop3nio.c:600, Client 127.0.0.1:48356 connected to origin 127.0.0.1:110
> nc -l 1110 > out
> ls /tmp
1 fire_emu_1589.sock fire_emu_898.sock
bar fire_emu_1845.sock foo
core-js-banners fire_emu_1876.sock hidden
fire_emu_1054.sock fire_emu_1987.sock hsperrfdata.delsh
fire_emu_1065.sock fire_emu_1998.sock npm-185-668f2574
fire_emu_1096.sock fire_emu_2021.sock npm-185-668f2574
fire_emu_1126.sock fire_emu_2073.sock npm-2639-c5258a9f
fire_emu_1157.sock fire_emu_2104.sock npm-3599-4ec3f48d
fire_emu_1188.sock fire_emu_2135.sock out
fire_emu_1219.sock fire_emu_2167.sock remote-wsl-loc.txt
fire_emu_1243.sock fire_emu_2198.sock runtime-delsh
fire_emu_1250.sock fire_emu_328.sock runtime-root
fire_emu_1281.sock fire_emu_359.sock ssh-VmfolHwDiV1N
fire_emu_1368.sock fire_emu_571.sock v8-compile-cache-1000
fire_emu_1395.sock fire_emu_582.sock 'vgdb-pipe-from-vgdb-to-1390-by-de
fire_emu_1465.sock fire_emu_589.sock 'vgdb-pipe-shared-mem-vgdb-1390-by-de
fire_emu_1496.sock fire_emu_628.sock 'vgdb-pipe-to-vgdb-from-1390-by-de
fire_emu_1515.sock fire_emu_659.sock
fire_emu_1558.sock fire_emu_819.sock
> diff out ~/Maildir/cur/delsh:2,S
1,5d0
< +OK Dovecot (Debian) ready.
< +OK
< +OK Logged in,
< +OK
< +
8983737d8903731
< .
```

Prueba 5 - Transform utilizando cat sobre mail grande

```

./proxy-pop3
[2021-11-22 17:07:03] [ INFO ] main.c:40, Listening on TCP port 1110
[2021-11-22 17:07:25] [ DEBUG ] pop3nio.c:378, Initializing parsers
[2021-11-22 17:07:25] [ INFO ] pop3nio.c:601, Client 127.0.0.1:39664 connected to origin 127.0.0.1:110
[2021-11-22 17:07:25] [ DEBUG ] pop3nio.c:1506, Client disconnected

> printf 'USER delsh\nPASS asd123\nLIST\nTOP 1 1\nQUIT\n' | nc -C 127.0.0.1 1110
+OK Dovecot (Debian) ready.
+OK
+OK Logged in.
+OK 1 messages:
1 694489434
.
+OK
Message-ID: <1509896531.21636.1.camel@itba.edu.ar>
Subject: Archivo grande
From: "Juan F. Codagnone" <jcodagno@itba.edu.ar>
To: jcodagno@itba.edu.ar
Content-Type: multipart/mixed; boundary="--PJ7raRLZ7SgpoxscB+sr"
Date: Sun, 05 Nov 2017 12:42:11 -0300
Mime-Version: 1.0
--=PJ7raRLZ7SgpoxscB+sr
.
+OK Logging out.

```

Prueba 6 - Pipelining cliente

```

./proxy-pop3
[2021-11-22 17:39:54] [ INFO ] main.c:40, Listening on TCP port 1110
[2021-11-22 17:41:13] [ DEBUG ] pop3nio.c:378, Initializing parsers
[2021-11-22 17:41:13] [ INFO ] pop3nio.c:601, Client 127.0.0.1:39672 connected to origin 127.0.0.1:110
[2021-11-22 17:41:13] [ DEBUG ] pop3nio.c:1506, Client disconnected

> printf 'USER delsh\nPASS asd123\nLIST\nTOP 1 1\nQUIT\n' | nc -C 127.0.0.1 1110
+OK Dovecot (Debian) ready.
+OK
+OK Logged in.
+OK 1 messages:
1 694489434
.
+OK
Message-ID: <1509896531.21636.1.camel@itba.edu.ar>
Subject: Archivo grande
From: "Juan F. Codagnone" <jcodagno@itba.edu.ar>
To: jcodagno@itba.edu.ar
Content-Type: multipart/mixed; boundary="--PJ7raRLZ7SgpoxscB+sr"
Date: Sun, 05 Nov 2017 12:42:11 -0300
Mime-Version: 1.0
--=PJ7raRLZ7SgpoxscB+sr
.
+OK Logging out.

```

Prueba 7 - Trailing spaces con pipelining

```

time printf 'USER delsh\nPASS asd123\nRETR 1\n' | nc -C 127.0.0.1 110 | pv | tail -n +5 | head -n -1 | sha256sum
662MiB 0:00:02 [ 256MiB/s ] =>
5e95fc3477fb33b6f9574bad93bbe4c4a55593f049a3de644546fceaa3916 -
printf 'USER delsh\nPASS asd123\nRETR 1\n' 0.00s user 0.00s system 16% cpu 0.001 total
nc -C 127.0.0.1 110 0.23s user 0.53s system 29% cpu 2.586 total
pv 0.00s user 0.74s system 28% cpu 2.585 total
tail -n +5 0.08s user 0.79s system 33% cpu 2.585 total
head -n -1 0.20s user 0.73s system 36% cpu 2.586 total
sha256sum 2.28s user 0.29s system 99% cpu 2.586 total

time printf 'USER delsh\nPASS asd123\nRETR 1\n' | nc -C localhost 1110 | pv | tail -n +5 | head -n -1 | sha256sum
662MiB 0:00:29 [ 22.5MiB/s ] =>
5e95fc3477fb33b6f9574bad93bbe4c4a55593f049a3de644546fceaa3916 -
printf 'USER delsh\nPASS asd123\nRETR 1\n' 0.00s user 0.00s system 13% cpu 0.001 total
nc -C localhost 1110 0.22s user 0.23s system 4% cpu 29.413 total
pv 0.07s user 1.01s system 3% cpu 29.412 total
tail -n +5 0.11s user 0.83s system 3% cpu 29.412 total
head -n -1 0.17s user 0.65s system 2% cpu 29.412 total
sha256sum 2.83s user 0.11s system 9% cpu 29.412 total

```

Prueba 8 - Performance e inmutabilidad de bytes sobre mail grande (3s vs 29s)

```

time printf 'USER delsh\nPASS asd123\nRETR 1\n' | nc -C localhost 1110 | pv | tail -n +5 | head -n -1 | sha256sum
662MiB 0:01:50 [ 5.90MiB/s ] =>
5e95fc3477fb33b6f9574bad93bbe4c4a55593f049a3de644546fceaa3916 -
printf 'USER delsh\nPASS asd123\nRETR 1\n' 0.00s user 0.00s system 10% cpu 0.001 total
nc -C localhost 1110 0.46s user 3.64s system 3% cpu 1:50.84 total
pv 0.29s user 2.64s system 2% cpu 1:50.84 total
tail -n +5 0.34s user 2.31s system 2% cpu 1:50.84 total
head -n -1 0.36s user 1.74s system 1% cpu 1:50.84 total
sha256sum 3.19s user 0.40s system 3% cpu 1:50.84 total

time printf 'USER delsh\nPASS asd123\nRETR 1\n' | nc -C localhost 1110 | pv | tail -n +5 | head -n -1 | sha256sum
662MiB 0:01:52 [ 5.89MiB/s ] =>
5e95fc3477fb33b6f9574bad93bbe4c4a55593f049a3de644546fceaa3916 -
printf 'USER delsh\nPASS asd123\nRETR 1\n' 0.00s user 0.00s system 11% cpu 0.001 total
nc -C localhost 1110 0.22s user 3.74s system 4% cpu 1:52.45 total
pv 0.31s user 2.68s system 2% cpu 1:52.45 total
tail -n +5 0.22s user 2.55s system 2% cpu 1:52.45 total
head -n -1 0.30s user 1.86s system 1% cpu 1:52.45 total
sha256sum 3.34s user 0.20s system 3% cpu 1:52.45 total

time printf 'USER delsh\nPASS asd123\nRETR 1\n' | nc -C localhost 1110 | pv | tail -n +5 | head -n -1 | sha256sum
662MiB 0:01:51 [ 5.93MiB/s ] =>
5e95fc3477fb33b6f9574bad93bbe4c4a55593f049a3de644546fceaa3916 -
printf 'USER delsh\nPASS asd123\nRETR 1\n' 0.00s user 0.00s system 13% cpu 0.001 total
nc -C localhost 1110 0.58s user 3.29s system 3% cpu 1:51.77 total
pv 0.26s user 2.76s system 2% cpu 1:51.77 total
tail -n +5 0.22s user 2.49s system 2% cpu 1:51.77 total
head -n -1 0.54s user 1.57s system 1% cpu 1:51.77 total
sha256sum 3.32s user 0.27s system 3% cpu 1:51.77 total

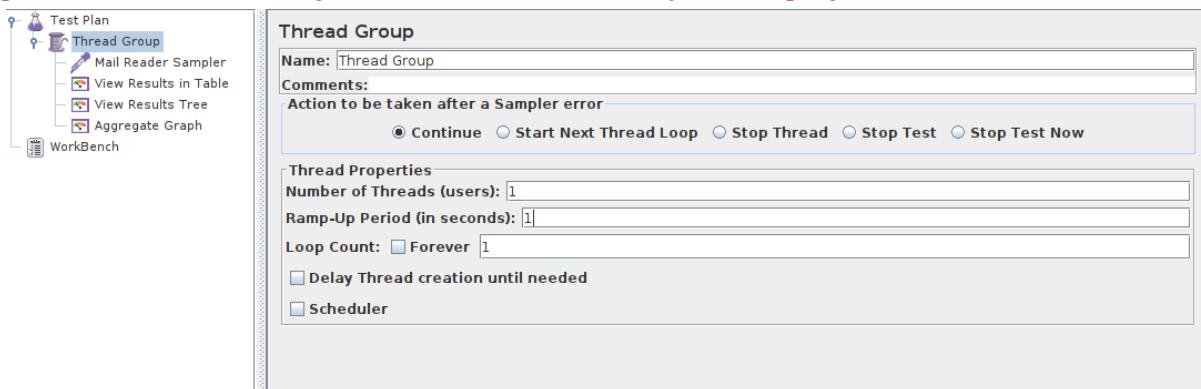
```

Prueba 9 - Concurrencia (Se incrementó x4 el tiempo de transferencia, de 29s a 110s)

Pruebas de Estrés

Para la realización de las pruebas de estrés, para ver cómo reaccionaba el *proxy POP3* implementado ante la conexión de varios clientes, se utilizó la aplicación dada por la cátedra, [JMeter](#). Se decidió realizar dos casos de prueba con una casilla de 13 mails. Entre ellos se encuentra un mail con la distribución de Fedora y un mail de 12 MB.

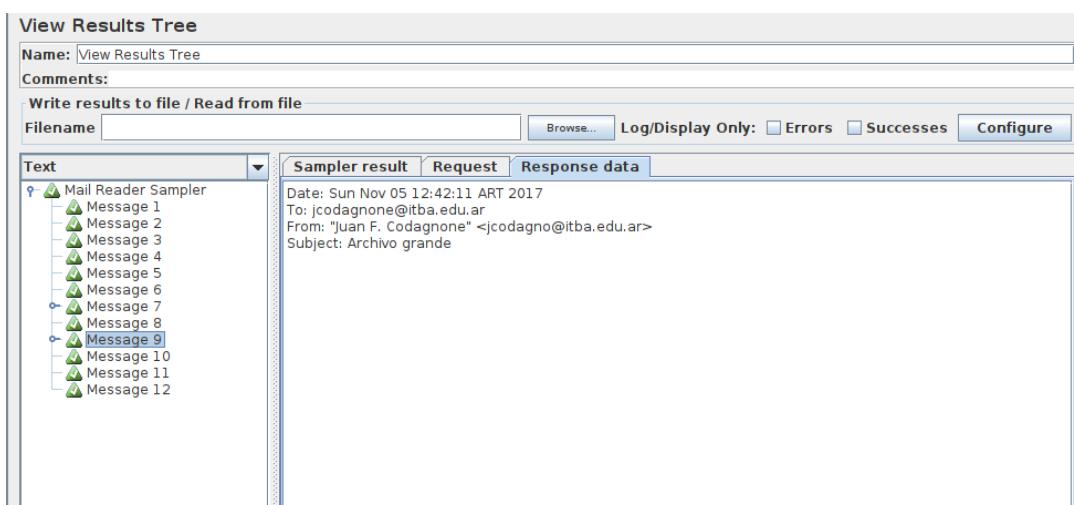
En el primer caso se realizó una prueba con un único cliente conectado. En la siguiente imagen podemos observar la configuración de **JMeter** antes de ejecutar el programa.



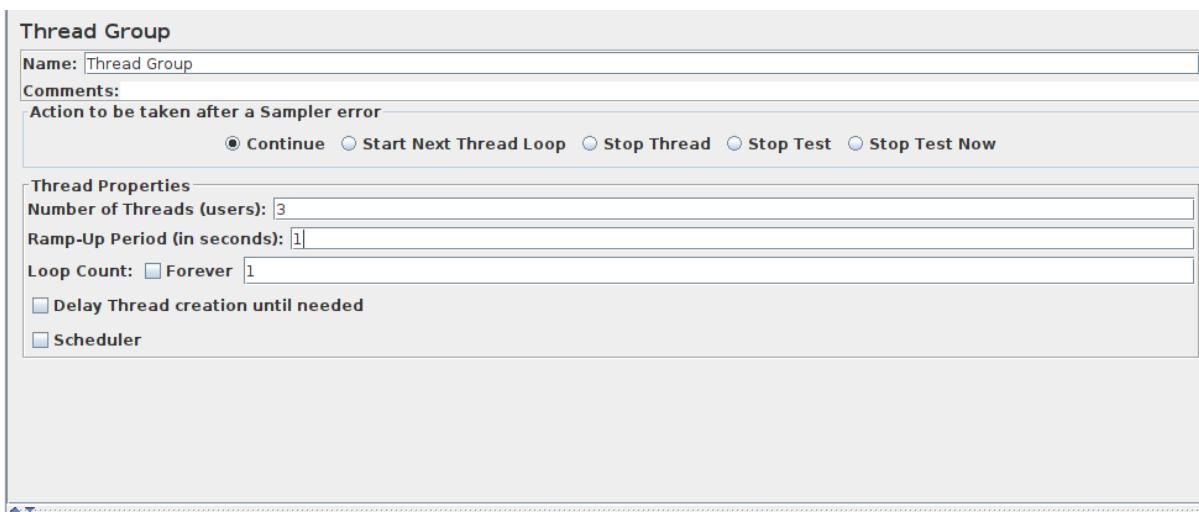
Y en la siguiente imagen podemos observar los resultados de la prueba con distintas estadísticas. Podemos resaltar la tasa de envío de bytes por segundo y el porcentaje de error.

Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
62947	62947	62947	62947	62947	62947	62947	0.00%	57.2/hour	8273.3

Podemos verificar que todos los mails se pudieron recuperar correctamente:



El segundo caso de prueba se realizó bajo las mismas condiciones pero con 3 clientes conectados. Al igual que el caso anterior, en la siguiente imagen podemos observar la configuración



Nuevamente, podemos observar los resultados en la siguiente imagen.

Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
170080	169571	170589	170589	170589	169571	170589	0.00%	42.2/hour	6103.9
170080	169571	170589	170589	170589	169571	170589	0.00%	42.2/hour	6103.9

Se intentó realizar pruebas con mayor cantidad de usuarios conectados, sin embargo, con más de 3 usuarios, **JMeter** avisa que se queda sin memoria.

Guía de instalación

Para la instalación del proyecto es necesario primero descargar el proyecto disponible en [el repositorio de git](#). Luego se debe ejecutar *make* (seteando la variable de entorno **CC** al compilador deseado, como se explica en el **README**) dentro de la carpeta principal del proyecto (*proxy-pop3*) donde se compilan y linkean los dos ejecutables, obteniéndose:

- */pop3filter/pop3filter.out*
- */pop3ctl/pop3ctl.out*

La invocación para ejecutar el proxy es la siguiente:

```
./pop3filter/pop3filter.out 127.0.0.1
```

Instrucciones para la configuración

Para ejecutar el cliente de management se debe correr el archivo ejecutable entregado por el *make* que se encuentra en la carpeta de *pop3ctl* donde se proporciona un argumento para la dirección del servidor, -P como puerto de **PMP** del servidor y -t para indicar el token:

```
./pop3ctl/pop3ctl.out 127.0.0.1 -P 1110 -t SECRETPROX
```

Ejemplos de configuración y monitoreo

The screenshot shows two terminal windows side-by-side. The left window displays a user session with Dovecot, showing commands like 'nc -C localhost 1110' and responses such as '+OK Dovecot (Ubuntu) ready.' and '+OK Logged in.'. The right window shows log output from a proxy server, with entries like 'INFO [main.c:42] Listening on TCP port 1110' and 'ERROR [server_utils.c:69] Unable to bind Cannot assign requested address'.

```

srosati — root@33d8c79e1306:/home/srosati/University/Protos/proxy-pop3 — docker exec -ti 3...
root@33d8c79e1306:/# nc -C localhost 1110
+OK Dovecot (Ubuntu) ready.
USER dove
+OK
PAS test
+OK Logged in.
RETR 1
+OK 428 octets
Return-Path: <srosati@itba.edu.ar>
X-Original-To: dove@localhost
Delivered-To: dove@localhost
Received: from localhost (localhost [127.0.0.1])
        by 33d8c79e1306 (Postfix) with SMTP id 975E710031F
        for <dove@localhost>; Fri, 12 Nov 2021 17:37:27 -0300 (-03)
Message-Id: <>20211112203744.975E710031F@33d8c79e1306>
Date: Fri, 12 Nov 2021 17:37:27 -0300 (-03)
From: srosati@itba.edu.ar

HOLA
ESTO ES UNA PRUEBA DEA
HOLA
.
ERR Disconnected for inactivity.

srosati — root@33d8c79e1306:/home/srosati/University/Protos/proxy-pop3 — docker exec -ti 3...
root@33d8c79e1306:/# ./pop3filter/pop3filter.out 127.0.0.1
[2021-11-23 11:06:09][ INFO ] main.c:42, Listening on TCP port 1110
[2021-11-23 11:06:09][ ERROR ] server_utils.c:69, Unable to bind Cannot assign requested address
[2021-11-23 11:06:19][ DEBUG ] pop3nio.c:392, Initializing parsers
[2021-11-23 11:06:19][ INFO ] pop3nio.c:625, Client 127.0.0.1:46426 connected to origin 127.0.0.1:1
10
[2021-11-23 11:06:30][ INFO ] pop3nio.c:1488, Disconnecting client for inactivity
[2021-11-23 11:06:30][ DEBUG ] pop3nio.c:1539, Client disconnected

```

Ejemplo 1 - Métricas del servidor

The screenshot shows two terminal windows side-by-side. The left window shows a user session with Dovecot, similar to the one in Example 1. The right window shows log output from a proxy server, with entries like 'INFO [main.c:42] Listening on TCP port 1110' and 'ERROR [server_utils.c:69] Unable to bind Cannot assign requested address'. It also shows a command to change the filter program.

```

srosati — root@33d8c79e1306:/home/srosati/University/Protos/proxy-pop3 — docker exec -ti 3...
root@33d8c79e1306:/# printf "USER dove\nPASS test\nRETR 1\n" | nc -C localhost 1110
+OK Dovecot (Ubuntu) ready.
+OK
+OK Logged in.
+OK 428 octets
Return-Path: <srosati@itba.edu.ar>
X-Original-To: dove@localhost
Delivered-To: dove@localhost
Received: from localhost (localhost [127.0.0.1])
        by 33d8c79e1306 (Postfix) with SMTP id 975E710031F
        for <dove@localhost>; Fri, 12 Nov 2021 17:37:27 -0300 (-03)
Message-Id: <>20211112203744.975E710031F@33d8c79e1306>
Date: Fri, 12 Nov 2021 17:37:27 -0300 (-03)
From: srosati@itba.edu.ar

HOLA
ESTO ES UNA PRUEBA DEA
HOLA
.
AC
root@33d8c79e1306:/# printf "USER dove\nPASS test\nRETR 1\n" | nc -C localhost 1110
+OK Dovecot (Ubuntu) ready.
+OK
+OK Logged in.
+OK
     14      46      429

srosati — root@33d8c79e1306:/home/srosati/University/Protos/proxy-pop3 — docker exec -ti 3...
root@33d8c79e1306:/# ./pop3ctl/pop3ctl.out -t SECRETPROX 127.0.0.1
[2021-11-23 11:15:30][ INFO ] main.c:42, Listening on TCP port 1110
[2021-11-23 11:15:30][ ERROR ] server_utils.c:69, Unable to bind Cannot assign requested address
[2021-11-23 11:15:05][ DEBUG ] pop3nio.c:392, Initializing parsers
[2021-11-23 11:16:05][ INFO ] pop3nio.c:625, Client 127.0.0.1:46450 connected to origin 127.0.0.1:1
10
[2021-11-23 11:16:36][ INFO ] pop3nio.c:724, Client 127.0.0.1:46450 disconnected
[2021-11-23 11:16:36][ DEBUG ] pop3nio.c:1538, Client disconnected
[2021-11-23 11:16:37][ INFO ] pop3nio.c:625, Client 127.0.0.1:46454 connected to origin 127.0.0.1:1
10
[2021-11-23 11:16:37][ INFO ] main.c:42, Listening on TCP port 1110
[2021-11-23 11:16:37][ ERROR ] server_utils.c:69, Unable to bind Cannot assign requested address
[2021-11-23 11:16:37][ DEBUG ] pop3nio.c:392, Initializing parsers
[2021-11-23 11:16:37][ INFO ] pop3nio.c:625, Client 127.0.0.1:46454 connected to origin 127.0.0.1:1
10
[2021-11-23 11:16:37][ INFO ] GET_FILTER_CMD
[2021-11-23 11:16:37][ DEBUG ] FILTER COMMAND: NONE
[2021-11-23 11:16:37][ DEBUG ] SET_FILTER_CMD wc
[2021-11-23 11:16:37][ DEBUG ] CHANGED FILTER PROGRAM

```

Ejemplo 2 - Cambio de programa de filtro

Análisis de código

Para realizar un análisis de la calidad de código de las aplicaciones desarrolladas, se utilizaron las herramientas *cppcheck*, *pvs-studio*, *complexity* y *scan-build*. Las últimas dos no lanzaron ningún bug y todos los errores y warnings de las primeras fueron corregidos, las notas que siguen lanzando los resultados son falsos positivos o warnings no pertinentes al proyecto de librerías brindadas por la cátedra.

Documento de diseño del proyecto

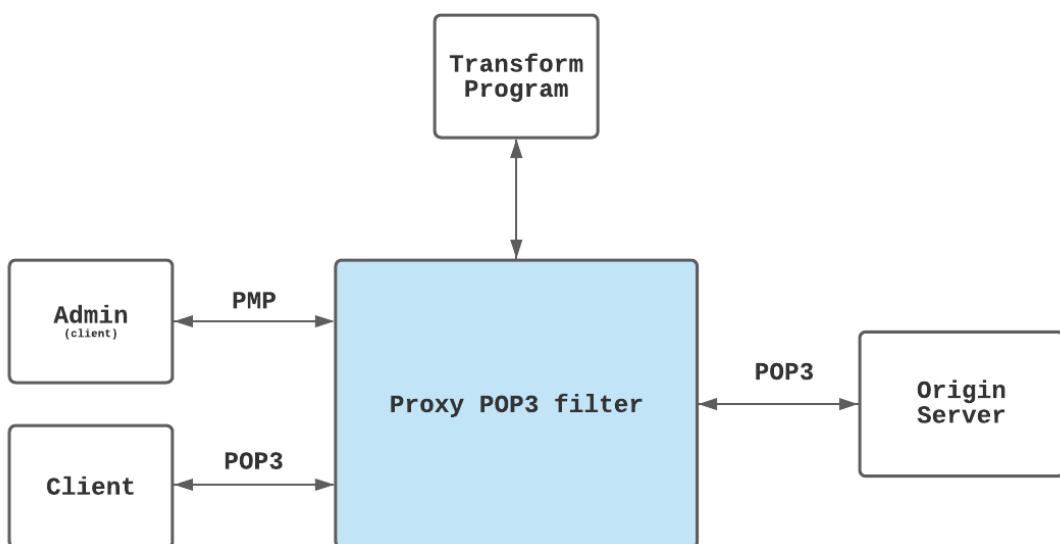


Gráfico Proxy POP3

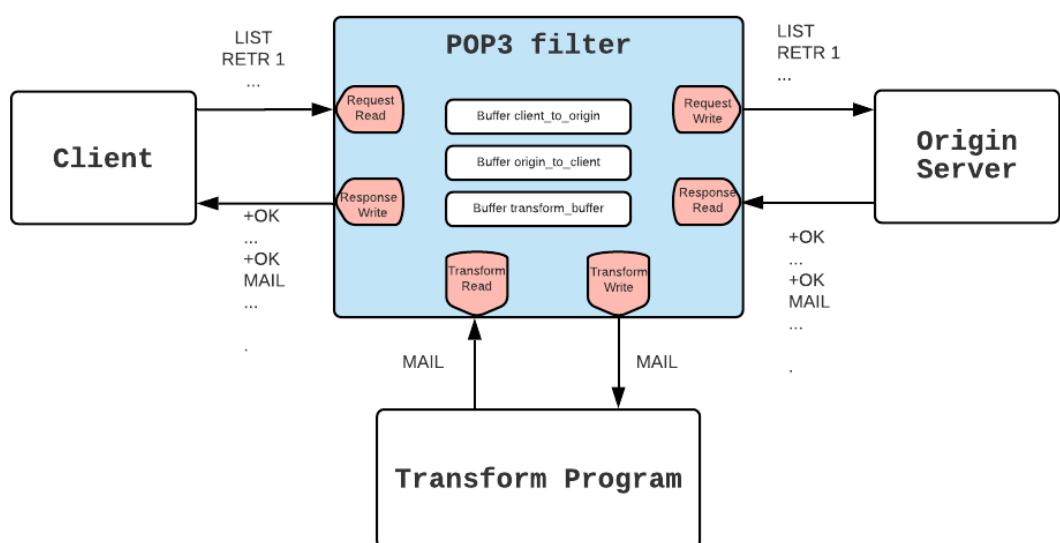


Gráfico flujo de mail transform utilizando proxy POP3