# Trabajo práctico Especial
# Protocolos de Comunicación
# 2022

Integrantes:
- Squillari, Bruno Omar
- Zimbimbakis, Facundo
- Budó Berra, Gaspar

Fecha de entrega: 21/06/22

# Contenido

# Descripción de protocolo y aplicaciones desarrolladas:

Para este proyecto se implementó un servidor proxy SOCKSv5 [RFC1928]. Este mismo soporta autenticación mediante usuario y contraseña [RFC1929]. El servidor es capaz de atender más de un cliente de forma simultánea y concurrente. También, soporta conexiones salientes que necesiten resolución de nombres FQDN, IPv6 o IPv4. En cuanto a la conexión con el servidor siendo cliente, es posible utilizar direcciones IPv4 o IPv6. El servidor también cuenta con password dissector para interceptar las contraseñas que sean enviadas con la estructura método de autenticación USER/PASS del protocolo POP3 a través del mismo. Todas las conexiones al servidor son no-bloqueantes, logrando que más de un solo cliente pueda conectarse al mismo y atenderlos de manera simultánea.

Para el desarrollo del trabajo práctico se debió implementar un protocolo propio, el cual puede leerse en el archivo server-management-protocol-05.pdf. Este protocolo funciona sobre TCP, es binario y orientado a sesión. Este mismo permite a un cliente autenticarse, manejar el estado del servidor y obtener métricas sobre las conexiones realizadas por el mismo. Las características que permite modificar son algunas configuraciones típicas del servidor

- agregar y modificar usuarios
- activar o desactivar la autenticación
- activar o desactivar el password Dissector

Además, las métricas que permite obtener son:

- lista de usuarios activos
- cantidad histórica de conexiones al server
- conexiones concurrentes
- cantidad máxima de conexiones concurrentes
- cantidad histórica de intentos de autenticación
- cantidad histórica de intentos de conexión
- cantidad de bytes enviados por el server
- promedio de bytes enviados por cada read/write.

Para poder probarlo correctamente se desarrolló un cliente que pueda realizar cada uno de los métodos descritos anteriormente. Esta aplicación permite cambiar las configuraciones y obtener las métricas que se deseen, dentro de las soportadas por el protocolo.

# Problemas encontrados durante el diseño y la implementación

## Sesiones persistentes

Durante las pruebas que se realizaron con un browser, haciendo que el tráfico pase por nuestro proxy, descubrimos que ciertas conexiones quedaban abiertas mucho tiempo después de que la conexión pareciera que se había terminado.

## Flujo de conexiones con manejo de estados

El manejo del flujo del protocolo ameritaba de una buena compresión de las distintas etapas que atravesaba el mismo y lo que ocurría en cada una de ellas. En consecuencia, fue necesario aprender a plasmar este procedimiento que sigue el protocolo lo cual no fue fácil. Se tuvo que empezar de cero varias veces para poder entender bien el funcionamiento. Sin embargo, con la utilización de la máquina de estados pudimos cumplir el objetivo de reflejar de buena forma el flujo del protocolo.

## Manejo de errores

Un concepto que repetidas veces tuvimos que corregir fue el manejo de errores utilizando toda la potencia del protocolo para informar al usuario. El principal problema fue que desde un principio el código no se construyó para informar de manera debida al usuario los errores, lo cual rompía con el curso del protocolo, por lo cual tuvimos que arreglarlo lo más minuciosamente posible.

## Resolución de nombres

A la hora de realizar resolución de nombres mediante DNS, tuvimos problemas para encontrar el flujo correcto debido a las características propias de la función getaddrinfo, la cual es bloqueante. Por eso mismo, debimos realizar la resolución en un hilo aparte para no bloquear al servidor. Luego, cuando la resolución haya sido completada, el selector y la máquina de estados se encargarán de que el flujo continúe su rumbo dirigiéndose a la etapa de conexión con el origen. Si la primera resolución de la lista falla se sigue intentando con las siguientes. En caso de que ninguna de las direcciones sea válida o haya concretado una conexión exitosa, se le avisará al cliente en cada respectivo caso.

# Limitaciones de la aplicación

## Conexiones persistentes:

Como mencionamos anteriormente, uno de los problemas que nos encontramos fue la posibilidad de que una conexión no se cierre. Esto perjudica a la performance del servidor ya que se sigue preguntando si esas conexiones tienen información para leer/escribir.

## Límite de conexiones simultáneas

Para el manejo de los distintos File Descriptor se utiliza un selector que responde cuál de los sockets está listo para lectura/escritura. Uno de los problemas de esta implementación es que solo permite "escuchar" en 1024 file descriptors como máximo. Además, por cada conexión, necesitamos 2 sockets para el cliente y otro el origen. De esta forma, nos quedan solo 512 posibles conexiones simultáneas. Por último, tenemos 4 socket pasivos (IPv4, IPv6 para SOCKS e IPv4, IPv6 para M16) que debemos escuchar siempre, por lo tanto, la cantidad final de conexiones simultáneas es menor a 509.

## Límite de usuarios

Decidimos que la cantidad máxima de usuarios que pueden conectarse al servidor sea 255, debido a que nuestra máxima cantidad de conexiones concurrentes es 500 y podemos asumir que cada usuario hará 2 conexiones. Además, el máximo de caracteres que puede tener un nombre de usuario o una contraseña es 255, el cual nos parece un valor bastante aceptable.

## Límite de buffer

El tamaño máximo de buffer que utilizamos es de 2048. Para determinar un valor adecuado, se realizaron pruebas similares a las de performance que se mostrarán en la sección 7 y analizando los distintos resultados obtenidos, determinamos que este era un valor apropiado.

# Posibles extensiones

## Timeout de conexiones

Para solucionar el problema de las conexiones persistentes decidimos que sería adecuado agregar un tiempo límite en el cual una conexión puede estar sin enviar información. Por cuestiones de tiempo no lo llegamos a implementar, pero consideramos que podríamos agregar un timestamp a cada conexión y preguntar si el tiempo máximo para cada conexión se ha superado, para así cerrar la conexión.

## Protocolo m16

Con más tiempo disponible nos hubiera gustado tener un protocolo donde se pueda realizar más de un request sin tener que cerrar la conexión entre el cliente y el servidor. Además, se podría considerar hacer uso de la opción de dar una descripción en los errores en los campos de la respuesta.

# Conclusiones

A lo largo de este TPE pudimos integrar los conocimientos que se vieron a lo largo de toda la cursada de la materia. Pudimos levantar servidores, consultar direcciones IP, utilizar proxys y usar túneles SSH para comprobar el correcto funcionamiento de nuestro servidor. Todos los contenidos que podían haber sido evaluados en la práctica formaron parte de alguna de las etapas del proyecto. Pudimos trabajar a fondo con un protocolo de comunicación fuertemente establecido como lo es SOCKS5 y por otro lado diseñar e implementar uno propio, basándonos en los distintos protocolos que vimos durante la cursada. Aprendimos como se conforma a grandes rasgos la estructura de cualquier protocolo y, además, como implementarla lo cual abre un abanico enorme de posibilidades para desarrollar.

# Ejemplos de prueba

## Consideraciones

- Todos los tests se pueden dividir en 3 etapas


- ./bin/server se refiere al binario con el servidor SOCKS5.
- ./bin/client se refiere al binario con el cliente usado para obtener métricas y configurar el servidor en ejecución.
- Algunos de los ejemplos mostrados a continuación fueron ejecutados en el servidor pampero.itba.edu.ar. Debido a una disponibilidad de puertos, es posible que esos ejemplos utilicen puertos diferentes a los estándares del servidor. Esto solo fue aclarado para el test de modificación de puertos (B) y se considera que se sobreentiende el funcionamiento para todos los otros tests. Puede haber diferencias entre el comando escrito en el informe con el mostrado en las imágenes.
- A partir del test C incluido y en adelante, se considera siempre como Precondición que los puertos estén disponibles para la correcta ejecución del servidor.

A. Binding en puertos default correct
- ✓ Precondiciones:
  - o Tener libre los puertos 1080 y 8080.
- ✓ Ejecución:
  - o TermA: $ ./bin/server -f credentials.txt
  - o TermB: $ netstat -nlp | grep server

```
[gbudoberra@pampero PR]$ netstat -nlp | grep /server
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:1080            0.0.0.0:*               LISTEN      159743/./bin/server
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN      159743/./bin/server
tcp6       0      0 ::1:8080                :::*                    LISTEN      159743/./bin/server
tcp6       0      0 :::1080                 :::*                    LISTEN      159743/./bin/server
[gbudoberra@pampero PR]$
```

- ✓ Validación:
  - o Puerto TCP 1080 escucha en todas las interfaces (IPv4, IPv6) para conexiones entrantes.
  - o Puerto TCP 8080 escucha en loopback (IPv4, IPv6) para el manejo de la configuración.

B. Cambio de puertos
- ✓ Precondiciones:
  - o Tener libre los puertos deseados para cambiar.
- ✓ Ejecución:
  - o TermA: $ ./bin/server -f credentials.txt -p 6060 -P 6061

- TermB: $ netstat -nlp | grep server

```
[gbudoberra@pampero PR]$ netstat -nlp | grep /server
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 127.0.0.1:6061          0.0.0.0:*               LISTEN      159593/./bin/server
tcp        0      0 0.0.0.0:6060            0.0.0.0:*               LISTEN      159593/./bin/server
tcp6       0      0 ::1:6061                :::*                    LISTEN      159593/./bin/server
tcp6       0      0 :::6060                 :::*                    LISTEN      159593/./bin/server
[gbudoberra@pampero PR]$
```

✓ Validación:

- Los puertos fueron modificados correctamente.

C. Conexión a dirección IPv4

✓ Precondiciones:

- Tener una dirección IPv4 que responda. Elegimos la IP de google.com.ar que nos devolvió el comando dig, pero esta misma puede variar.

✓ Ejecución:

- TermA: $ ./bin/server -f credentials.txt –u user:pass
- TermB: $ curl –x socks5h://user:pass@127.0.0.1:1080 '216.58.202.99'

```
[gbudoberra@pampero ~]$ curl -x socks5h://user:pass@127.0.0.1:6000 '216.58.202.99'
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
[gbudoberra@pampero ~]$ curl '216.58.202.99'
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
```

✓ Validación:

- Volvió una respuesta con su HTML correspondiente. En este caso un 301.
- La respuesta es la misma usando el proxy o no.

D. Conexión a dirección IPv6

✓ Precondiciones:

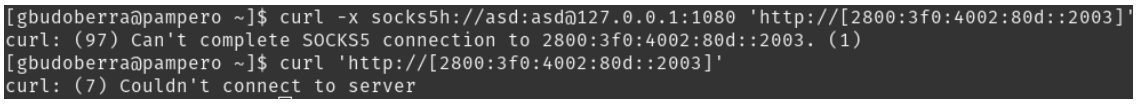
- Tener una dirección IPv6 que responda. Elegimos la IP de google.com.ar que nos devolvió el comando dig, pero esta misma puede variar.

✓ Ejecución:

- TermA: $ ./bin/server -f credentials.txt -u user:pass

- TermB: $ curl -x socks5h://user:pass@127.0.0.1:1080 'http://[2800:3f0:4002:810::2003]'

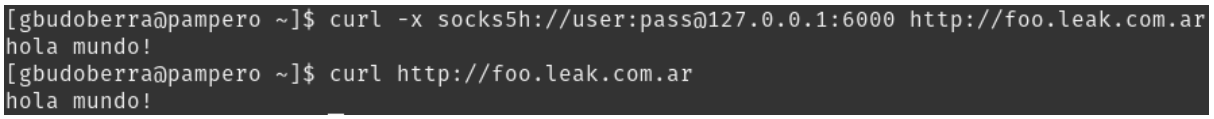

✓ Validación:
- Misma respuesta pasando y sin pasar por el proxy
(Nota: esta prueba se corrió de manera local debido a que en pampero no se logra completar la conexión a esta dirección inclusive sin utilizar el proxy)




E. Resolución por nombre

✓ Precondiciones:
- Dirección que pueda ser resuelta por una consulta DNS. Importante: usar -x socks5**h** para la resolución de nombres.

✓ Ejecución:
- TermA: $ ./bin/server -f credentials.txt -u user:pass
- TermB: $ curl -x socks5h://user:pass@127.0.0.1:1080 http://foo.leak.com.ar
- TermC: curl http://foo.leak.com.ar




✓ Validación:
- Respuesta textual correcta, pasando y sin pasar por el proxy.

F. DNS ipv6

✓ Precondiciones:
- Dirección que solo se resuelva por ipv6. Importante: usar -x socks5**h** para la resolución de nombres.

✓ Ejecución:
- TermA: $ ./bin/server -f credentials.txt -u user:pass

- TermB: $ curl -x socks5h://user:pass@127.0.0.1:1080 http://ipv6.google.com

```
gbudoberra@pop-os:~$ curl -x socks5h://asd:asd@127.0.0.1:1080 http://ipv6.google.com
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="es-419"><head><me
ta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta content="/images/brandin
g/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><title>Google</title><script n
once="tVvaYZhCp8Ayv4ifFWHgWg">(function(){window.google={kEI:'VfexYsvSBfmj1sQPju6zmA0',kEXPI:'
0,1302536,56873,6059,206,4804,2316,383,246,5,1354,4013,1123753,1197723,678,380089,16115,28684,
17572,4858,1362,9291,3021,4752,12835,4020,978,13228,3847,6885,3737,7432,15309,5966,708,1279,27
42,149,1103,840,1983,4314,3514,606,2023,1777,520,6345,8325,3227,2845,7,4811,1,12638,8102,3522
```

✓ Validación:

- Obtener una respuesta coherente al link utilizado.

G. Performance con archivos "grandes"

✓ Precondiciones:

- Consideramos adecuado el uso de un archivo de tamaño superior a 1Gb para comprobar el funcionamiento. En este caso se utilizó una imagen antigua de linux.

✓ Ejecución:

- TermA: $ ./bin/server -f credentials.txt -u user:pass
- TermB: $ time curl -x socks5h://user:pass@127.0.0.1:1080 https://old-releases.ubuntu.com/releases/16.10/ubuntu-16.10-desktop-amd64.iso | md5sum
- TermC: time curl https://old-releases.ubuntu.com/releases/16.10/ubuntu-16.10-desktop-amd64.iso | md5sum

```
[gbudoberra@pampero PR]$ time curl -x socks5h://asd:asd@127.0.0.1:6000 https://old-releases.ub
untu.com/releases/16.10/ubuntu-16.10-desktop-amd64.iso| md5sum
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1520M  100 1520M    0     0  12.5M      0  0:02:01  0:02:01 --:--:-- 12.7M
3f50877c05121f7fd8544bef2d722824  -

real    2m1.374s
user    0m4.296s
sys     0m2.727s
[gbudoberra@pampero PR]$ time curl https://old-releases.ubuntu.com/releases/16.10/ubuntu-16.10
-desktop-amd64.iso| md5sum
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1520M  100 1520M    0     0  10.7M      0  0:02:21  0:02:21 --:--:-- 10.5M
3f50877c05121f7fd8544bef2d722824  -

real    2m21.586s
user    0m8.826s
sys     0m8.651s
```

✓ Validación:

- Mismo md5, por lo tanto, los archivos descargados fueron iguales
- Time similar (incluso menor a través del proxy)
- Nota: habremos más comentarios sobre la performance en otro apartado.

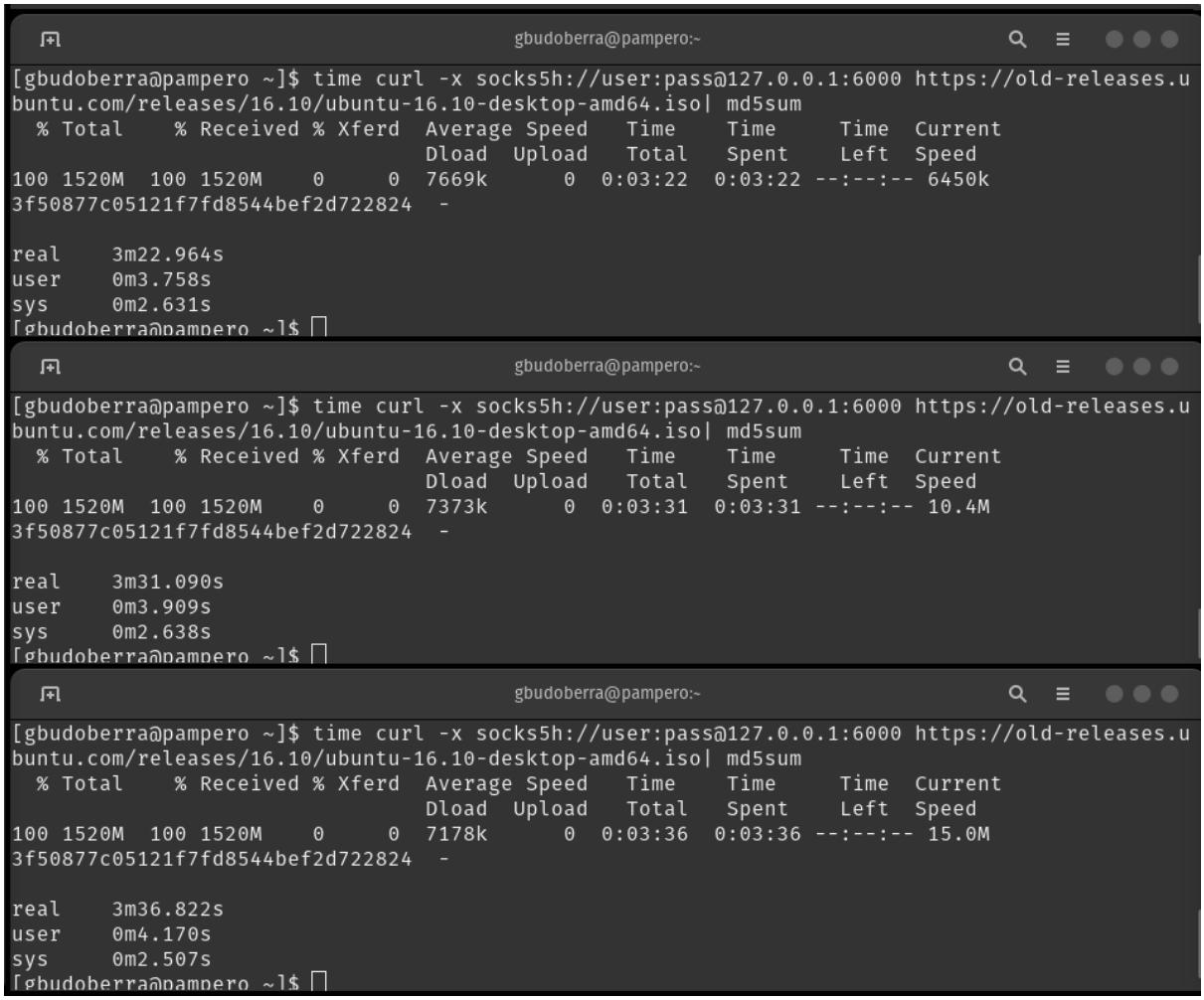
H. Concurrencia

✓ Precondiciones:

- Iguales al punto anterior.

✓ Ejecución:

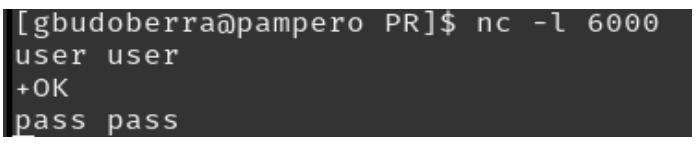
- En 3 terminales diferentes se corrió el mismo comando que en el ítem anterior.

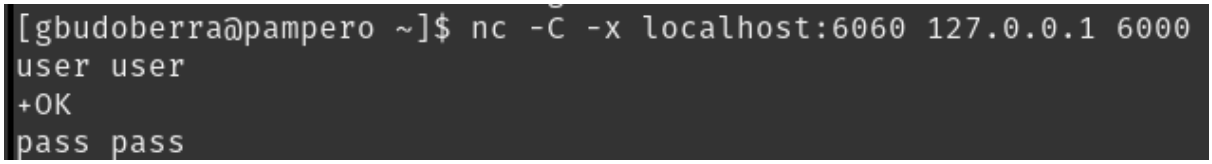



- ✓ Validación:
  - o Mismo md5 para todas las ejecuciones.

I. Password Dissector

- ✓ Precondiciones:
  - o Deshabilitar el authentication del server ./bin/client -u admin:admin luego elegir indice '12' y la opción 'off'
- ✓ Ejecución:
  - o TermA: ./bin/server -u user:pass
  - o TermB: nc -l 6000
  - o TermC: nc -C -x localhost:6060 127.0.0.1 6000



```
[gbudoberra@pampero PR]$ ./bin/server -u user:pass -f credentials.txt -p 6060 -P
 6061
Inicializando server...
2022-06-21T04:39:12Z    user     to: 127.0.0.1:6000      from: 127.0.0.1:44582
 status: 0
2022-06-21T04:39:23Z    Active user: user       Register: P     Procolo: POP3
Client address: 127.0.0.1:44582 Origin address: 127.0.0.1:6000        Username:
user  Password: pass
```

- ✓ Validación:
  - o Se muestran el nombre de usuario y la contraseña ingresados a través del netcat. Nota: aparece en varias líneas para facilitar la lectura del informe.

J. Conexión a IPv4 invalida

- ✓ Precondiciones:
  - o Dirección IPv4 invalida.
- ✓ Ejecución:
  - o TermA: ./bin/server -f credentials.txt -u user:pass
  - o TermB: curl -x socks5://user:pass@127.0.0.1:6000 'http://127.0.0.1:3333'

```
DEBUG, CONNECTING WRITE: Código: 0. Mensaje: Connection refused -> REQUEST_WRITE to reply error to client, 19
user     to: 127.0.0.1:3333      from: 127.0.0.1:44156   Mon Jun 20 22:45:34 2022
         status: 5
```

```
[gbudoberra@pampero ~]$ curl -x socks5://user:pass@127.0.0.1:6000 'http://127.0.0.1:3333'
curl: (97) Can't complete SOCKS5 connection to 127.0.0.1. (5)
```

- ✓ Validación:
  - o La conexión se rechaza.
  - o Queda registrado el intento de conexión con el status de error correspondiente.

K. Conexión a IPv6 invalida

- ✓ Precondiciones:
  - o Dirección IPv6 invalida.
- ✓ Ejecución:
  - o TermA: ./bin/server -f credentials.txt -u user:pass
  - o TermB: curl -x socks5://user:pass@127.0.0.1:6000 'http://[::1]:3333'

```
[gbudoberra@pampero ~]$ curl -x socks5://user:pass@127.0.0.1:6000 'http://[::1]:3333'
curl: (97) Can't complete SOCKS5 connection to ::1. (5)
```

```
DEBUG, CONNECTING WRITE: Código: 0. Mensaje: Connection refused -> REQUEST_WRITE to reply error to client, 19
user     to: ::1:3333    from: 127.0.0.1:38262   Mon Jun 20 22:47:12 2022
         status: 5
```

- ✓ Validación:
  - o La conexión se rechaza.
  - o Queda registrado el intento de conexión con el status de error correspondiente.

L. Conexión DNS invalida

- ✓ Precondiciones:

- nombre no resoluble por consulta DNS

✓ Ejecución:
- TermA: $ /bin/server -f credentials.txt -u user:pass
- TermB: $ curl -x socks5h://user:pass@127.0.0.1:6000 'http://xxxxxxxxxx/'

```
[gbudoberra@pampero ~]$ curl -x socks5://user:pass@127.0.0.1:6000 'http://xxxxxxxxx/'
curl: (97) Could not resolve host: xxxxxxxxx
[gbudoberra@pampero ~]$ curl -x socks5h://user:pass@127.0.0.1:6000 'http://xxxxxxxxx/'
curl: (97) Can't complete SOCKS5 connection to xxxxxxxxx. (1)
```

```
DEBUG, REQUEST RESOLV DONE: Código: 0. Mensaje: Error FQDN resolution came back empty -> REQUE
ST_WRITE to reply error to client, 6
DEBUG, MAIN: Código: 0. Mensaje: New selector cycle, 0
DEBUG, REQUEST WRITE: Código: 0. Mensaje: Starting stage, 6
DEBUG, REQUEST WRITE: Código: 10. Mensaje: Writing to client, 6
DEBUG, REQUEST WRITE: Código: 10. Mensaje: Wrote to client, 6
DEBUG, REQUEST WRITE: Código: 9. Mensaje: Finished stage, 6
DEBUG, MAIN: Código: 0. Mensaje: New selector cycle, 0
DEBUG, MAIN: Código: 0. Mensaje: New selector cycle, 0
```

✓ Validación:
- Podemos ver el error que tira curl, relacionado con la conexión del SOCKS5.
- Si debugueamos el server vemos que la resolución FQDN volvió vacía.

M. DNS IPv6 invalida

✓ Precondiciones:
- Dirección que al resolver por nombre devuelve una dirección IPv6.

✓ Ejecución:
- TermA: $ /bin/server -f credentials.txt -u user:pass
- TermB: $ curl -x socks5h://user:pass@127.0.0.1:6000 'http://ipv6.leak.com.ar/'

```
[gbudoberra@pampero PR]$ curl -x socks5h://user:pass@127.0.0.1:6060 'http://ipv6.leak.com.ar/'
curl: (97) Can't complete SOCKS5 connection to ipv6.leak.com.ar. (1)
```

✓ Validación:
- Podemos ver el error que tira curl, relacionado con la conexión del SOCKS5. Esto es porque la dirección que devuelve ese nombre es ::1.

N. Agnóstico del protocolo

✓ Precondiciones:
- Como stty no está instalado en pampero, hacemos 2 túneles para conectar el tráfico de nuestra terminal al proxy corriendo en pampero, y para redirigir el localhost 9090 de pampero.
- Además, debe apagarse la autenticación mediante el cliente para que el netcat no sea rechazado por el proxy.

✓ Ejecución:
- TermA: $ ssh -R 9090:localhost:9090 pampero.itba.edu.ar
- TermB: $ ssh -L 1080:localhost:6000 pampero.itba.edu.ar

- TermC (pampero): $ ./bin/server -f credentials.txt -p 6000 -P 6001 -u user:pass
- TermD: $ stty -icanon && nc -l 9090
- TermE: $ stty -icanon && nc -x localhost:1080 localhost 9090

```
gbudoberra@pop-os:~$ stty -icanon && nc -l 9090
hola :)
```

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 11 | 16.037928929 | ::1 | ::1 | Socks | 87 | Unknown |
| 12 | 16.037942129 | ::1 | ::1 | TCP | 86 | 38308 → 1080 [ACK] Seq=1 Ack=2 Win=512 Len=0 TSval=1461724990 TSecr=1461724990 |
| 15 | 17.270251800 | ::1 | ::1 | Socks | 87 | Unknown |
| 16 | 17.270263378 | ::1 | ::1 | TCP | 86 | 38308 → 1080 [ACK] Seq=1 Ack=3 Win=512 Len=0 TSval=1461726223 TSecr=1461726223 |
| 24 | 42.486947412 | ::1 | ::1 | Socks | 87 | Unknown |
| 25 | 42.486957323 | ::1 | ::1 | TCP | 86 | 38308 → 1080 [ACK] Seq=1 Ack=4 Win=512 Len=0 TSval=1461751439 TSecr=1461751439 |
| 30 | 42.860620123 | ::1 | ::1 | Socks | 87 | Unknown |
| 31 | 42.860627198 | ::1 | ::1 | TCP | 86 | 38308 → 1080 [ACK] Seq=1 Ack=5 Win=512 Len=0 TSval=1461751813 TSecr=1461751813 |
| 32 | 42.866367543 | ::1 | ::1 | Socks | 87 | Unknown |
| 33 | 42.866373318 | ::1 | ::1 | TCP | 86 | 38308 → 1080 [ACK] Seq=1 Ack=6 Win=512 Len=0 TSval=1461751819 TSecr=1461751819 |
| 36 | 42.913123267 | ::1 | ::1 | Socks | 87 | Unknown |
| 37 | 42.913138444 | ::1 | ::1 | TCP | 86 | 38308 → 1080 [ACK] Seq=1 Ack=7 Win=512 Len=0 TSval=1461751866 TSecr=1461751866 |
| 40 | 43.054840838 | ::1 | ::1 | Socks | 87 | Unknown |
| 41 | 43.054851288 | ::1 | ::1 | TCP | 86 | 38308 → 1080 [ACK] Seq=1 Ack=8 Win=512 Len=0 TSval=1461752007 TSecr=1461752007 |

✓ Validación:
- Lo que se escribe en una terminal se ve inmediatamente en la otra.
- Se envían los paquetes socks correctamente.

# Análisis de performance

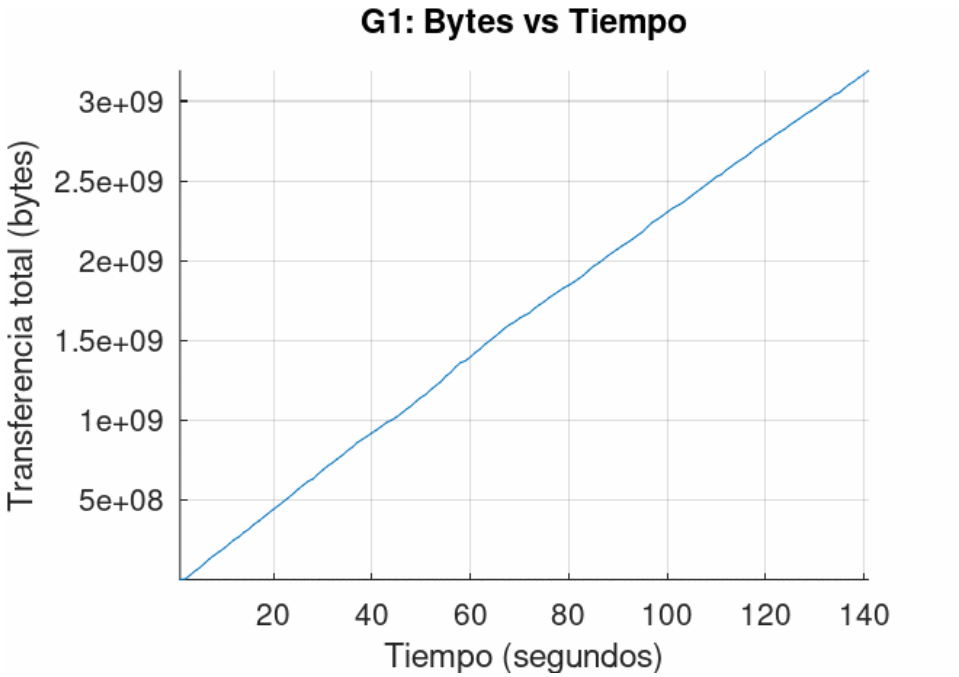
En la siguiente sección se grafican ciertas estadísticas obtenidas a partir del protocolo m16 para ser analizadas. Cabe aclarar que las caídas repentinas en los gráficos se deben a que se alcanza el número máximo que almacena el servidor por lo que se reinicia el conteo.

## Variación en la cantidad de conexiones que realizan descargas "grandes" en conjunto y su efecto en la velocidad de transferencia

En este caso, vamos a analizar cómo cambia la velocidad de transferencia dependiendo de la cantidad de conexiones al proxy. Para ello, se correrán en conjunto distintas cantidades de comandos curl que descargan un archivo "grande". El mismo que se utilizó en el ejemplo de prueba J.

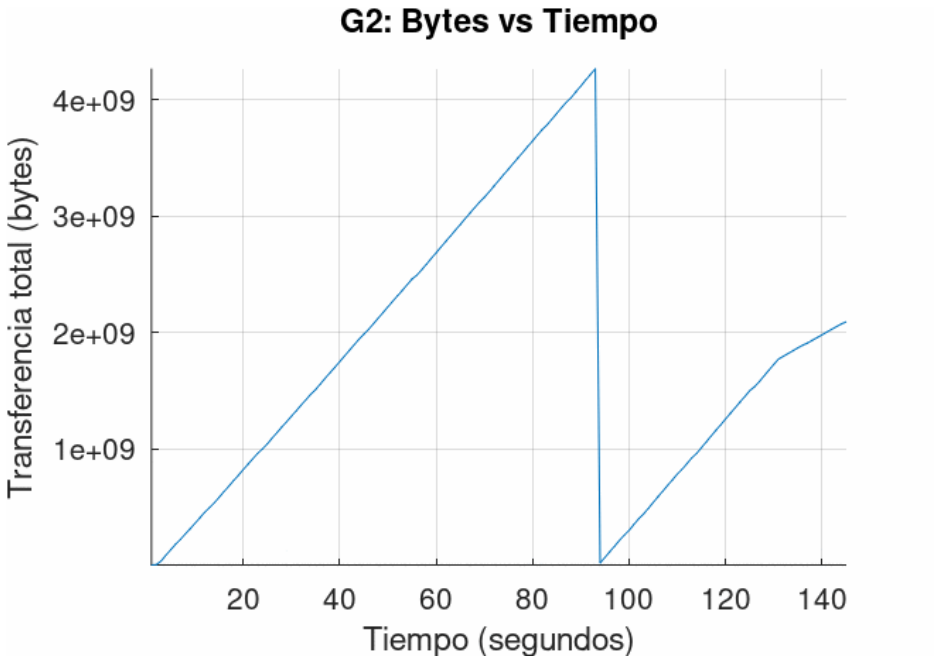
Comando:
*time curl -x socks5h://bruno:123@127.0.0.1:3000 https://old-releases.ubuntu.com/releases/16.10/ubuntu-16.10-desktop-amd64.iso | md5sum &*

Caso 1, un solo curl



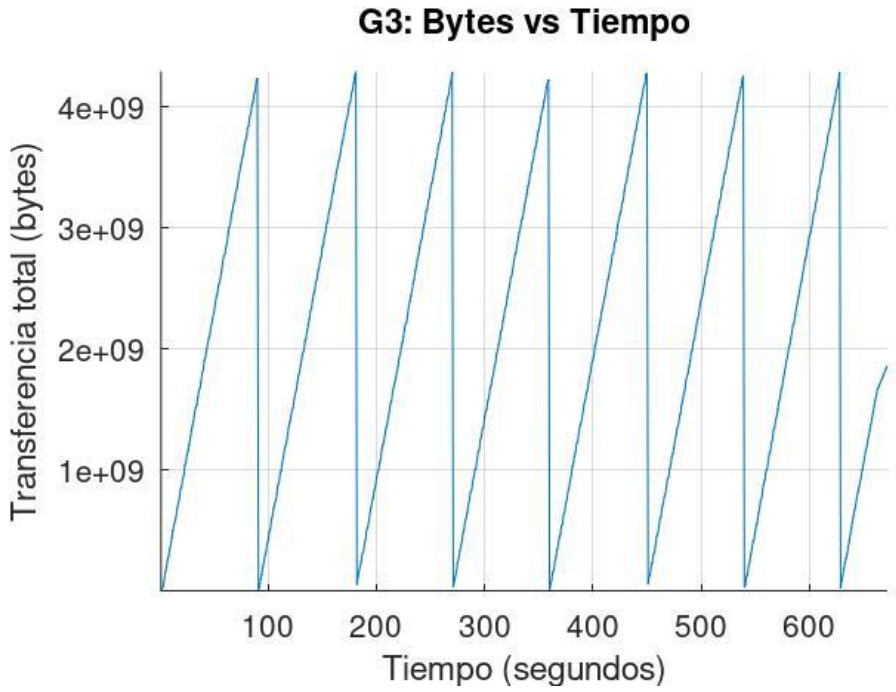

En el caso base, un solo curl, se obtuvo el gráfico G1. Podemos apreciar que resulta bastante lineal. La pendiente de la recta es de 2.2799e+07. Alcanzando 3.1920e+09 bytes transferidos en 140 segundos.

Caso 2, dos curls



En la siguiente prueba, dos curls, no se aleja mucho del primer caso. En total obtenemos 6.3527e+09 transferidos en 145 segundos con una pendiente de 4.3812e+07 bytes transferidos por segundo.
Vemos que al agregar un solo curl la velocidad de transferencia aumenta.

Caso 3, diez curls



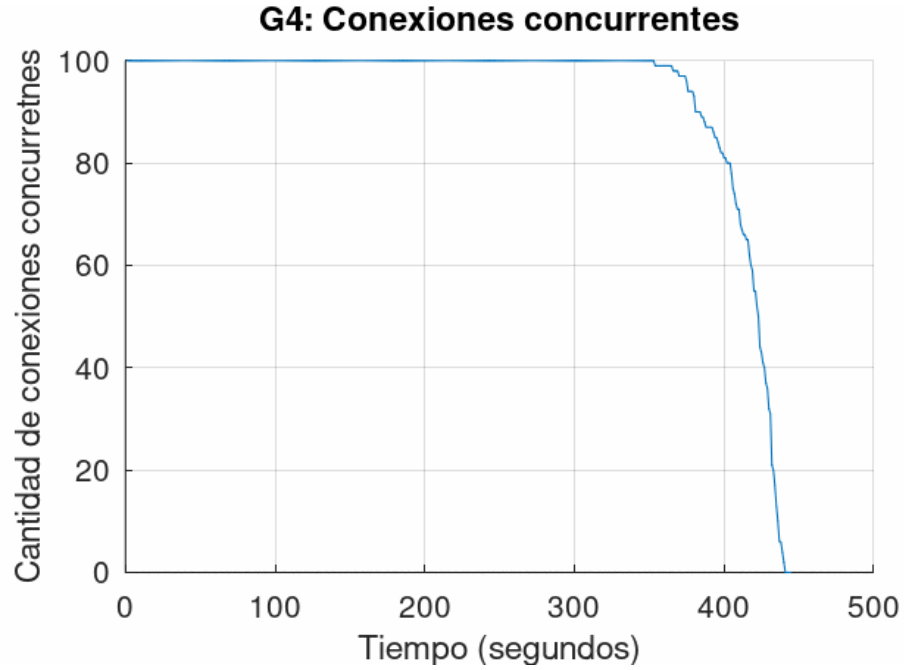

Finalmente, corriendo 10 curls obtenemos el gráfico G3. En total se transfirieron 2.9868e+10 en 672 con una velocidad de transferencia 4.4446e+07.

Si comparamos las velocidades de los tres casos podemos observar que 10 conexiones, sin importar que las descargas realizadas sean de gran tamaño, no afectaron la velocidad de transferencia.

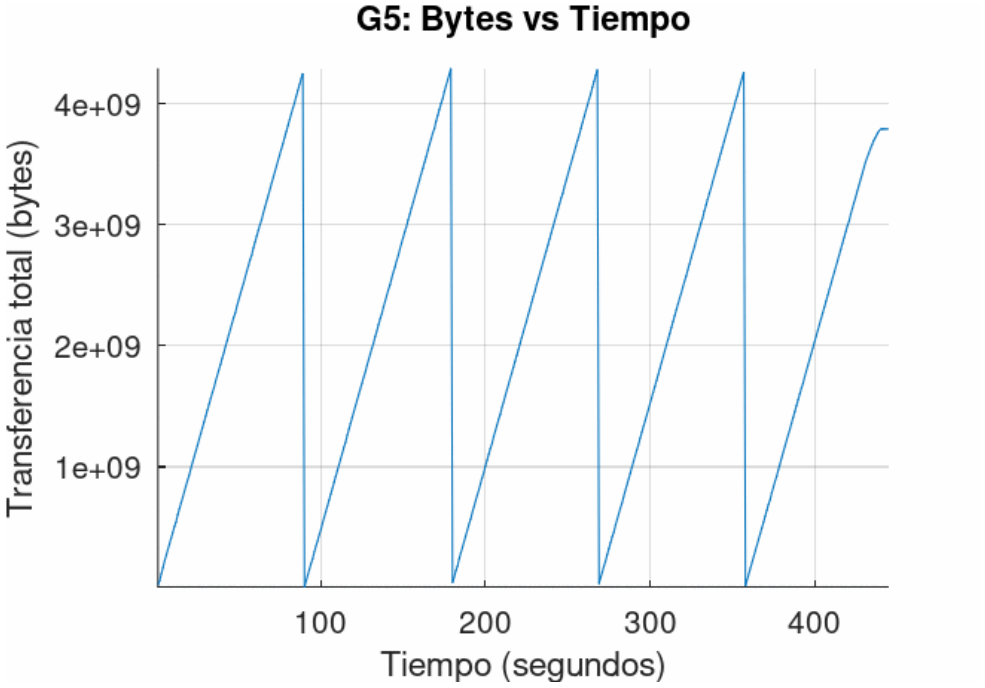
## Gran variación en la cantidad de conexiones que realizan descargas en conjunto y su efecto en la velocidad de transferencia

Este caso comparte el mismo objetivo que el caso uno. La diferencia radica en que esta vez la variación de cantidad de conexiones será mayor aprovechando que las descargas no son grandes.

Caso 1, 100 curls de 100 MB:

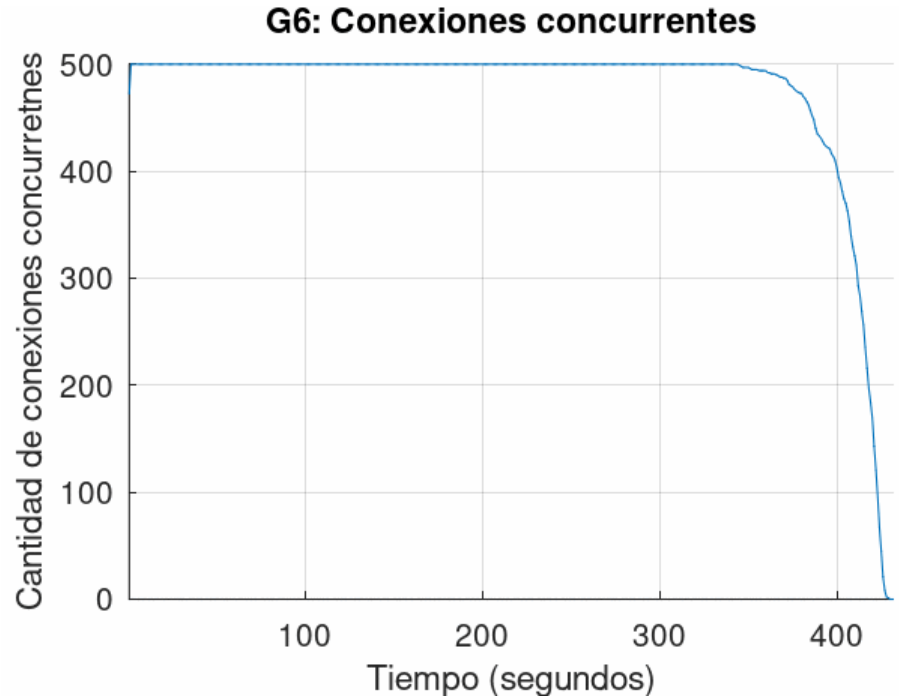



El gráfico G4 muestra el momento en que las conexiones comienzan a caer debido a la finalización de las descargas. Podemos ver que todas las conexiones fueron administradas en forma concurrente lo cual evita la espera entre ellas, pero tiene un impacto en la terminación de las tareas ya que consumen recursos en conjunto.

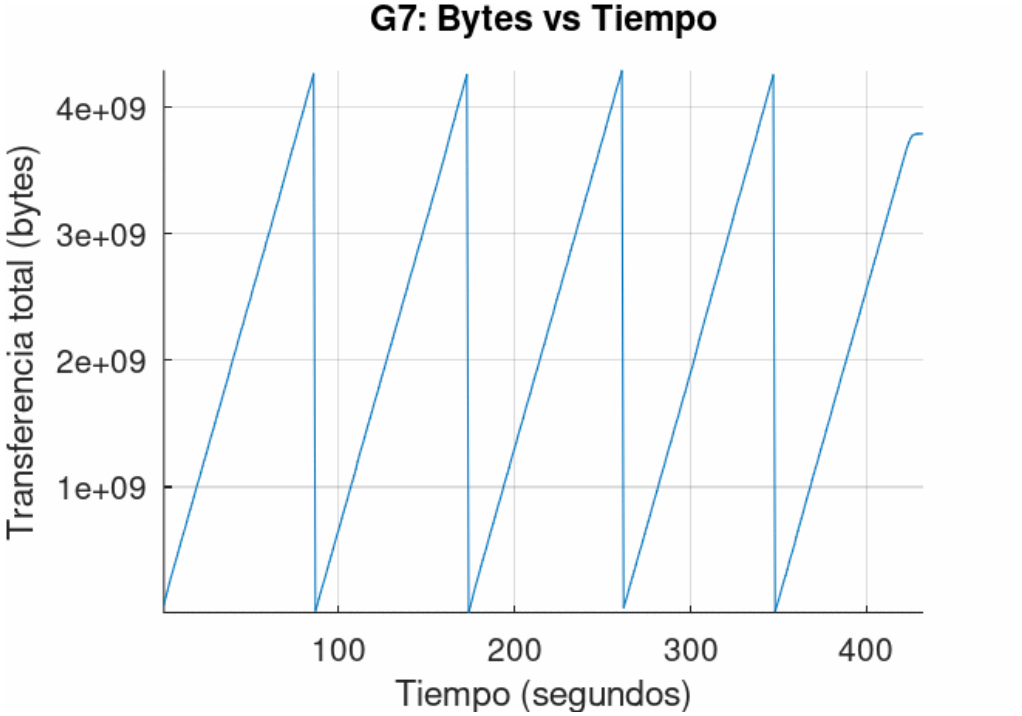



Luego, el gráfico G5 presenta la transferencia de bytes contra el tiempo. Se transfirieron alrededor de 2.0880e+10 bytes en 445 segundos. Lo que nos deja una velocidad de transferencia de alrededor de 4.6922e+07 por segundo.

Caso 2, 500 curls de 20 MB




Vemos otra vez la cantidad de conexiones concurrentes desde que se corren los comandos hasta que terminan.




Esta vez tenemos 2.0868e+10 bytes en 432 segundos con una velocidad de transferencia de 4.8305e+07.

En conclusión, podemos ver que las velocidades de transferencia tienen pequeñas variaciones a medida que se cambia la cantidad de conexiones en el proxy, sin embargo, no podemos ver una tendencia clara ya que las modificaciones son leves. Por lo que se podría concluir que la cantidad de conexiones concurrentes no afecta de manera considerable la velocidad de transferencia de bytes. Por otro lado, podría considerarse que el rendimiento individual si se ve afectado ya que, si miramos un único proceso, el mismo tarda más tiempo en terminar, pero la transferencia de bytes total no se ve afectada en gran medida.

# Guía de instalación

Para la compilación del proyecto se utilizó la herramienta Make, utilizando el archivo Makefile para especificar las instrucciones de compilación. De esta forma podemos utilizar distintos compiladores (cc, gcc, Clang) para la generación de los archivos objeto. Para generar cada uno de los ejecutables necesarios para el proyecto se puede ejecutar 'make server' o 'make client'. Luego, los ejecutables se encontrarán en la carpeta bin dentro del directorio principal.

## Instrucciones de configuración

### socks

Para poder entender completamente la configuración del servidor y la sintaxis de los argumentos recomendamos leer el archivo sock5d.8. Los parámetros principales son:
-h: Imprime la ayuda y termina.
-l dirección-socks: Establece la dirección donde servirá el proxy SOCKS. Por defecto escucha en todas las interfaces.
-N: Deshabilita los passwords dissectors.
-L dirección-de-management: Establece la dirección donde servirá el servicio de management. Por defecto escucha únicamente en loopback.
-p puerto-local: Puerto TCP donde escuchará por conexiones entrantes SOCKS. Por defecto el valor es 1080.
-P puerto-conf: Puerto TCP donde escuchará por conexiones entrante del protocolo de configuración. Por defecto el valor es 8080.
-u user:pass: Declara un usuario del proxy con su contraseña.
-v: Imprime información sobre la versión y termina.
-f path-credenciales: Especifica el archivo donde se obtendrán las credenciales para acceder al servicio de manejo y estadísticas que se ofrece mediante el protocolo m16. El repositorio tiene un archivo credentials.txt de prueba con el ejemplo usuario:admin contraseña:admin.

M16

Por otro lado, tenemos al cliente del protocolo creado con el fin de acceder a información del servidor en tiempo real o también modificar su configuración. El mismo se puede ejecutar con distintos flags especificados en el archivo manual clientM16. Los flags disponibles son:
-L dirección-de-management: Establece la dirección donde se conectará al servidor de management. Por defecto se conecta a loopback. Debe utilizarse en caso de que se modifique el default del servidor con el flag -L.
-P puerto-conf: Puerto TCP donde se conectará al servidor del protocolo de configuración. Por defecto el valor es 8080. Debe utilizarse en caso de que se modifique el default del servidor con el flag -P.
-u user:pass: Se loguea con usuario y contraseña al servidor de management. Se debe usar obligatoriamente ya que se está accediendo a una funcionalidad que requiere de ciertos privilegios. **Los usuarios permitidos los impone el servidor.**
-f path-archivo: Especifica un archivo donde se escribirán al final del mismo las estadísticas obtenidas separadas por un espacio. (Solo para los índices de 2 a 9)

# Ejemplos de configuración

## Cambio de puerto conexiones entrantes

```
[gbudoberra@pampero PR]$ ./bin/server -f credentials.txt -P 6061
Inicializando server...
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:1080            0.0.0.0:*               LISTEN      119078/./bin/server
tcp        0      0 0.0.0.0:6061            0.0.0.0:*               LISTEN      119078/./bin/server
tcp6       0      0 :::1080                 :::*                    LISTEN      119078/./bin/server
tcp6       0      0 :::6061                 :::*                    LISTEN      119078/./bin/server
```

## Cambio de puerto configuración

```
[gbudoberra@pampero PR]$ ./bin/server -f credentials.txt -p 6060 -P 6061
Inicializando server...
[gbudoberra@pampero ~]$ netstat -nltp | grep /server
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:6061            0.0.0.0:*               LISTEN      119082/./bin/server
tcp        0      0 0.0.0.0:6060            0.0.0.0:*               LISTEN      119082/./bin/server
```

## Add user

```
./bin/server -f credentials.txt -u user:pass
```

### Deshabilitar password dissectors

```
./bin/server -f credentials.txt -N
```

### Generar archivo de estadísticas

(solo funciona con los índices 0x02 a 0x09)

```
./bin/client -f statistic.txt
```

# Documento del diseño del proyecto

Dentro del proyecto tenemos el directorio src donde se almacenan todos los archivos necesarios para compilar ambas aplicaciones: servidor proxy socks y cliente M16.
Por otro lado, se encuentra el directorio include compuesto por todos los encabezados necesarios a la hora de linkeditar el proyecto.

## Archivos utilizados para el server

- main.c: Archivo donde se encuentra la función main. Esta función se encarga de invocar funcionalidad desarrollada en todos los demás archivos del proyecto. Primero comienza parseando los argumentos ingresados por línea de comandos. Al terminar, utilizando la información ya procesada se crean sockets pasivos necesarios para escuchar direcciones IPv6, IPv4 y el socket pasivo relacionado al cliente de nuestro protocolo implementado. Al finalizar esta acción de forma exitosa se encarga de inicializar el selector y registrar los file descriptors de los respectivos sockets creados. Por último, realizará un ciclo infinito (salvo que ocurra un error dentro del selector) esperando a que el selector avise si determinado file descriptor está disponible para leer/escribir según corresponda.
- args.c: Archivo donde se encuentran las funciones necesarias para procesar los argumentos ingresados por línea de comando. Provisto por la cátedra.
- selector.c: Módulo encargado del manejo del selector dependiendo de la necesidad. Provisto por la cátedra.
- socket_utils: Archivo encargado del manejo de sockets no bloqueantes. Provisto por la cátedra.
- stm.c: Máquina de estados. Gracias a esto el flujo del servidor se maneja de forma eficiente y ordenada. Provisto por la cátedra.
- socks5nio.c: Módulo donde se define el flujo del servidor con sus distintos estados posibles relacionados a sus respectivos handlers dependiendo del estado. Provisto por la cátedra.
- netutils: Clase utilizada principalmente para pasar el formato de una dirección IP a string para que pueda visualizarse por el usuario. Provisto por la cátedra.

- buffer.c: Archivo encargado del manejo de buffers necesarios para cada interacción servidor-cliente. Provisto por la cátedra.
- dissec_parser.c: Parser capaz de sniffear usuarios y contraseñas en formato POP3.
- address_utils.c: Clase utilizada principalmente para procesar las direcciones ip ingresadas por línea de comandos. Provisto por la cátedra.
- debug.c: Clase creada para facilitar la etapa de debugging a la hora de desarrollar.

  En esta sección vamos a describir los distintos archivos utilizados para llevar a cabo cada etapa necesaria para un servidor proxy socksv5 y un servidor para el protocolo creado m16. El protocolo creado se asemeja a socksv5 por lo cual se pudieron reutilizar funciones donde los propósitos no difieren.
- hello.c: Módulo encargado de parsear el handshake y enviarle la respuesta adecuada según corresponda. Utilizado para llevar a cabo tanto el protocolo socksv5 como nuestro protocolo.
- authentication.c: Archivo que contiene funciones necesarias para llevar a cabo la sub-negociación servidor-cliente tanto el socksv5 como en m16. Utiliza myParser.c para parsear lo enviado por el cliente.
- request.c: Clase capaz de manejar los requerimientos del cliente socksv5 y responderle según corresponda. Utiliza request_parser.c para parsear el envío del cliente. En esta etapa se decide según el request del cliente, cómo va a continuar el flujo. Si se debe resolver el nombre, crea un hilo para no bloquear al servidor. Si no, ya intenta conectarse al origen para poder enviarle su resultado al cliente.
- mng_request.c: Archivo creado para el manejo de la etapa de requests del cliente del protocolo creado. Según el requerimiento, se encarga de obtener la información solicitada para poder enviársela al cliente.
- resolv.c: Clase conformada por la función que será ejecutada en un hilo aparte (resolución del nombre) y la función llamada luego de resolver los nombres para setear las variables necesarias para la posterioridad.
- connecting.c: Módulo encargado de establecer la conexión con el origen específico. En caso de ser una dirección resuelta, se intenta por todas las direcciones posibles hasta poder conectarse de forma exitosa.

## Archivos utilizados para el cliente:

- client.c: Clase que contiene la función main y maneja todas las etapas posibles del protocolo de forma ordenada.
- clientArgs.c: Archivo similar a parse_args.c pero con menor cantidad de argumentos posibles.
- clientUtils.c: Parser de respuestas del servidor y requerimientos del cliente para cada determinada etapa especificada en nuestro protocolo.