



Protocolos de Comunicación - 72.07

Informe del desarrollo

- **Barthelemy**, Augusto - 64.502
- **Garfi**, Valentín - 64.486
- **Searles**, Ignacio - 64.536
- **Salama**, Nicole - 64.488





1. Índice

2. Descripción detallada de los protocolos y aplicaciones desarrolladas.....	3
2.1 Aplicación Principal: Servidor SOCKS5.....	3
2.1.1 Arquitectura del Servidor SOCKS5.....	3
2.1.2 Funcionalidades Implementadas.....	4
2.1 Protocolo PCTP (Proxy Configuration and Tracking Protocol).....	4
2.2.1 Diseño del Protocolo PCTP.....	4
2.2.2 Sistema de Estadísticas y Logging.....	5
2.3 Cliente PCTP.....	6
3. Problemas encontrados durante el diseño y la implementación.....	6
3.1 Problemas de Arquitectura.....	6
3.1.1 Modularización del protocolo PCTP y SOCKS5.....	6
3.2 Problemas de Rendimiento.....	7
3.2.1 Optimización del estado COPY en SOCKS5.....	7
4. Limitaciones de la aplicación.....	7
4.1 Limitaciones del Protocolo SOCKS5.....	8
4.1.1 Comandos Soportados.....	8
4.1.2 Métodos de Autenticación.....	8
4.2 Limitaciones de Escalabilidad.....	8
4.2.1 Modelo de Concurrencia.....	8
4.3 Limitaciones de Administración.....	10
4.3.1 Persistencia de Configuración.....	10
4.3.2 Logs de Conexión.....	10
5. Posibles extensiones.....	10



5.1 Extensiones del Protocolo SOCKS5.....	11
5.1.1 Soporte UDP y BIND.....	11
5.2 Extensiones de Escalabilidad.....	11
5.2.1 Modelo Multi-thread.....	11
5.2.2 Cambio de select(2) por poll(2).....	11
5.3 Extensiones de Monitoreo.....	12
5.3.1 Métricas Avanzadas.....	12
5.4.1 Filtrado de Destinos.....	12
6. Conclusiones.....	12
6.1 Logros Técnicos.....	12
6.2 Valor Técnico del Proyecto.....	13
6.3 Aplicabilidad Real.....	13
7. Ejemplos de prueba.....	14
7.1 Pruebas Básicas de Conectividad.....	14
7.2 Múltiples conexiones.....	14
7.4 Pruebas avanzadas de rendimiento.....	15
8. Guía de instalación detallada y precisa.....	17
9. Instrucciones para la configuración.....	17
9.1 Comandos de autenticación con PCTP.....	18
9.2 Comandos principales de PCTP.....	18
10. Ejemplos de configuración y monitoreo.....	18
11. Documento de diseño del proyecto.....	20



2. Descripción detallada de los protocolos y aplicaciones desarrolladas

Para el desarrollo del Trabajo Práctico Especial de la materia Protocolos de Comunicación, hemos desarrollado un servidor proxy para el protocolo **SOCKS5** y un protocolo para su gestión y configuración (el Proxy Configuration and Tracking Protocol, o por sus siglas, **PCTP**). A continuación, se detallan los aspectos fundamentales de cada parte del trabajo práctico.

2.1 Aplicación Principal: Servidor SOCKS5

El proyecto implementa un servidor **proxy SOCKS5** completo que cumple con la especificación RFC 1928 (*SOCKS Protocol Version 5*). La aplicación está diseñada como un sistema de alta concurrencia utilizando multiplexación de I/O asíncrono, permitiendo manejar múltiples conexiones simultáneas de manera eficiente.

2.1.1 Arquitectura del Servidor SOCKS5

El servidor implementa una máquina de estados finitos (STM) que gestiona el **ciclo de vida completo** de cada conexión SOCKS5

Estados principales:

- **HELLO_READ/WRITE**: Negociación inicial del protocolo SOCKS5
- **AUTH_READ/WRITE**: Autenticación usuario/contraseña (RFC 1929)
- **REQUEST_READ/WRITE**: Procesamiento de solicitudes de conexión
- **CONNECTING/AWAITING_CONNECTION/CONNECTING_RESPONSE**:
Establecimiento de conexión al servidor destino
- **COPY**: Relay bidireccional de datos entre cliente y servidor destino
- **DONE/ERROR**: Estados terminales



2.1.2 Funcionalidades Implementadas

Se ha implementado el protocolo SOCKS5 de manera completa. Esto significa que el presente desarrollo posee negociación de métodos de **autenticación, login** con usuario/contraseña, soporte para comandos **CONNECT, IPv4, IPv6 y nombres de dominio** y el **relay bidireccional de datos** con túnel persistente una vez establecida la conexión. Además, se mantuvo un manejo de errores mediante los códigos adecuados de acuerdo a lo establecido según el RFC 1928, para que nuestro servidor respete las normas y estándares establecidos.

Logramos desarrollar un servidor SOCKS5 que gestiona conexiones TCP de manera no bloqueante, con resolución de nombres asíncrona y una gestión eficiente de buffers de lectura/escritura, haciendo uso de estructuras y archivos provistos por la cátedra como punto de partida.

2.1 Protocolo PCTP (Proxy Configuration and Tracking Protocol)

Nuestro sistema incluye un protocolo personalizado, PCTP, que permite **administración y monitoreo** en tiempo real del servidor SOCKS5. Además, PCTP brinda a su vez la posibilidad de modificar la **configuración** de diversos aspectos propios del server, permitiendo alterar valores del mismo sin la necesidad de tener que reiniciarlo.

2.2.1 Diseño del Protocolo PCTP

El protocolo desarrollado por el equipo es de **tipo texto**, y consta de diversos comandos que permiten la operación de cualquier usuario con el servidor. Como características principales, cuenta con autenticación obligatoria de nivel administrativo y la capacidad de agregar usuarios con múltiples roles de acceso (ADMIN/BASIC), una máquina de estados para la gestión de sesiones y persistencia de conexión durante la sesión. También le permiten a un usuario conocer métricas sobre el estado del servidor y modificar valores pre-establecidos.



PCTP cuenta con el siguiente listado de comandos aceptados:

```
USER [username]      # Inicio de autenticación
PASS [password]      # Finalización de autenticación
STATS                # Estadísticas del servidor
LOGS [N]              # Últimos N logs de conexión al proxy
CONFIG IO=[BYTES]    # Asigna tamaño a los buffers de IO
ADD ADMIN             # Agregar usuario administrador
ADD BASIC             # Agregar usuario básico
DEL [username]        # Eliminar usuario
LIST                  # Listar usuarios registrados
EXIT                  # Terminar conexión
```

2.2.2 Sistema de Estadísticas y Logging

El servidor implementa un sistema de **estadísticas** que rastrea

Métricas en tiempo real	Información por conexión
Conexiones activas actuales	Timestamp de conexión
Total de conexiones históricas	IP/puerto del cliente
Bytes transferidos (actual y total)	IP/puerto destino
Logs detallados por conexión	Usuario autenticado
	Bytes transferidos
	Estado de autenticación
	Código de respuesta final

Esta información es accesible a través del protocolo PCTP.



2.3 Cliente PCTP

El proyecto incluye un **cliente interactivo** para administración remota del servidor a través del protocolo PCTP.

Este cliente se ejecuta desde la línea de comandos y permite establecer una conexión con el servidor PCTP, especificando la dirección (con -h) y el puerto (vía -p). Una vez conectado, se pueden enviar los comandos previamente descritos por el protocolo PCTP, tales como autenticación o consultas de estadísticas en tiempo real. Dichas funcionalidades son propias del protocolo PCTP, y mediante el cliente desarrollado se facilita su ejecución.

3. Problemas encontrados durante el diseño y la implementación

A lo largo del desarrollo del presente proyecto, hemos encontrado ciertas dificultades que supimos sortear mediante planificación y decisiones grupales.

3.1 Problemas de Arquitectura

3.1.1 Modularización del protocolo PCTP y SOCKS5

En un inicio, el código PCTP estaba concentrado en un solo **archivo monolítico**, dificultando el mantenimiento y la extensibilidad. Caso similar ocurría con lo desarrollado para el servidor SOCKS5, donde en un único archivo se mantenían todas las funcionalidades. Por esto, se refactorizaron por completo ambas secciones, dividiendo la funcionalidad en **módulos especializados** (pctp_auth, pctp_commands, pctp_users, pctp_protocol, pctp_stm para PCTP, y en socks5_auth, socks5_copy, socks5_hello, socks5_protocol y socks5_request para el servidor SOCKS5,), mejorando la organización, legibilidad y mantenibilidad del código.



3.2 Problemas de Rendimiento

3.2.1 Optimización del estado COPY en SOCKS5

Durante las revisiones del código, se detectó que el intercambio constante entre modos de lectura y escritura en el selector podría llegar a generar **overhead**, aumentando potencialmente la latencia en conexiones con alto tráfico bidireccional. Para resolver este problema, evaluamos diferentes estrategias de optimización, como por ejemplo mantener ambos intereses (OP_READ | OP_WRITE) activos simultáneamente, implementar forwarding inmediato de datos sin buffering innecesario o reducir las llamadas a `selector_set_interest` para minimizar overhead.

Finalmente, se optó por una implementación híbrida que mantiene la claridad del código mediante funciones separadas (`copy_read`, `copy_write`, `origin_read`, `origin_write`) pero incorporando optimizaciones específicas (como el uso de OP_READ | OP_WRITE en puntos críticos del flujo de datos). De este modo, la implementación final balancea **rendimiento y mantenibilidad**, siguiendo las mejores prácticas de programación asíncrona en C mientras, a su vez, se cumple completamente con el RFC 1928 del protocolo SOCKS5.

4. Limitaciones de la aplicación

El sistema presenta limitaciones en cuanto a la cantidad de conexiones en simultáneo que puede tener, las funcionalidades y su administración. Esto se debe principalmente al alcance que el trabajo naturalmente posee, y creemos que muchas de estas restricciones resultan aspectos que se podrían considerar e incluso implementar en caso de expandir el desarrollo aún más.



4.1 Limitaciones del Protocolo SOCKS5

4.1.1 Comandos Soportados

El servidor actualmente sólo implementa el comando **CONNECT** del protocolo SOCKS5, lo que habilita conexiones salientes TCP hacia un destino. Los comandos **BIND** y **UDP ASSOCIATE** no están contemplados en esta versión, lo que significa que no es posible establecer conexiones entrantes ni realizar proxy de tráfico UDP, limitando así ciertos usos como transferencia de archivos FTP o aplicaciones que requieren túneles UDP.

4.1.2 Métodos de Autenticación

La autenticación está limitada al método usuario y contraseña conforme al RFC 1929. Otros métodos como **GSSAPI** u **opciones más avanzadas** no están implementados en este TPE, lo que restringe la autenticación a escenarios con credenciales simples.

4.2 Limitaciones de Escalabilidad

4.2.1 Modelo de Concurrencia

La arquitectura se basa en un único hilo de ejecución que utiliza un selector de eventos. Esta decisión implica que el servidor está limitado por el **número máximo de descriptores** de archivo permitidos por el sistema operativo, que en configuraciones comunes ronda las 1000 conexiones, y que tareas costosas en CPU pueden bloquear la atención de otras conexiones si no se controlan cuidadosamente.

Para evaluar la capacidad de concurrencia de nuestro servidor, diseñamos un test de carga (prueba de stress) que simula múltiples clientes accediendo al proxy en paralelo:

```
# Test de carga
```

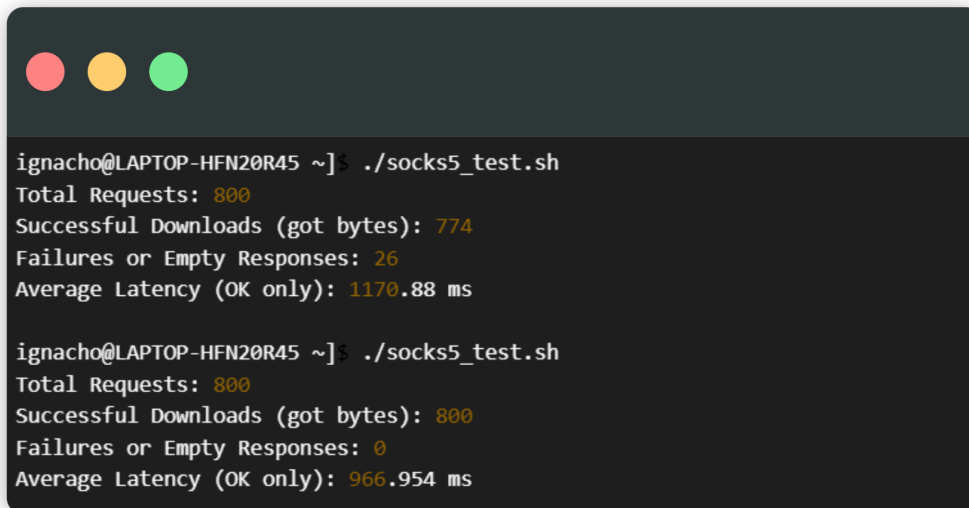


```
for i in {1..TOTAL}; do
    curl -s -o /dev/null \
        --write-out "%{size_download}" \
        -x socks5h://user:pass@server:port \
        --max-time 10 "$URL" &
done

wait
```

Este script lanza múltiples instancias de curl en segundo plano, cada una configurada para usar nuestro servidor SOCKS5 como proxy, y descargar contenido desde un servidor destino (elegido aleatoriamente). El objetivo principal era verificar que la conexión se estableciera correctamente y que se transferían datos desde el destino hasta el cliente, lo cual sería un indicio de un funcionamiento exitoso del proxy.

Durante estas pruebas logramos confirmar que el servidor logró manejar satisfactoriamente el requisito de más de **500 conexiones concurrentes**, e incluso alcanzar las **800 conexiones simultáneas** en algunos casos.



```
ignacho@LAPTOP-HFN20R45 ~]$ ./socks5_test.sh
Total Requests: 800
Successful Downloads (got bytes): 774
Failures or Empty Responses: 26
Average Latency (OK only): 1170.88 ms

ignacho@LAPTOP-HFN20R45 ~]$ ./socks5_test.sh
Total Requests: 800
Successful Downloads (got bytes): 800
Failures or Empty Responses: 0
Average Latency (OK only): 966.954 ms
```

Una de las principales restricciones que encontramos está relacionada con el uso de `select(2)`, que únicamente permite manejar descriptores de archivo con valores



menores a 1024. Dado que por cada cliente se necesitan dos sockets (uno para la conexión entrante y otro para la conexión saliente al servidor destino), este límite impone una barrera práctica.

Sin embargo, en la práctica logramos superar esta cifra gracias a la naturaleza asincrónica del tráfico: no todas las conexiones están activas al mismo tiempo. Algunas terminan antes de que otras se inicien, puesto que el *getaddrinfo(3)* y la conexión inicial con el cliente tienen un tiempo de resolución variable, lo que permite reutilizar descriptores y aumentar el throughput total del servidor.

4.3 Limitaciones de Administración

4.3.1 Persistencia de Configuración

En este desarrollo, la configuración de usuarios y demás parámetros no persiste entre **reinicios del servidor**. Esto implica que toda la configuración debe cargarse de nuevo cada vez que se reinicia la aplicación.

4.3.2 Logs de Conexión

El sistema utiliza un buffer circular para almacenar logs de conexiones con un tamaño fijo (4096 entradas). Si bien esto evita desbordamientos, también implica que en situaciones de alta carga se **sobrescriben eventos recientes**, ya que al no persistir los logs se pierde historial.

5. Posibles extensiones

Si bien lo desarrollado para la entrega final de la materia cumple con todas las funcionalidades requeridas y es completamente utilizable, consideramos que existen características adicionales que se podrían agregar para sumarle completitud y expandir aún mas las posibilidades de nuestro servidor.



5.1 Extensiones del Protocolo SOCKS5

5.1.1 Soporte UDP y BIND

Se podría agregar soporte para el comando **UDP ASSOCIATE**, permitiendo gestionar tráfico UDP de forma transparente. Esta extensión haría posible utilizar el servidor como proxy para aplicaciones que dependen de UDP, como consultas DNS o videojuegos en línea. También se podría sumar la implementación del comando **BIND**, ampliando la máquina de estados finitos (STM) para aceptar conexiones entrantes y notificar al cliente cuando estén listas. Esta funcionalidad sería especialmente útil en casos como conexiones de datos FTP o aplicaciones peer-to-peer.

5.2 Extensiones de Escalabilidad

5.2.1 Modelo Multi-thread

Una mejora posible es adoptar un modelo multi-thread, donde se distribuyan las conexiones entre **diversos hilos de trabajo**. De esta forma, se lograría una escalabilidad horizontal que permita manejar miles de conexiones simultáneas sin sobrecargar un solo hilo.

5.2.2 Cambio de select(2) por poll(2)

Como se ha mencionado anteriormente, select(2) solo permite manejar file-descriptors menores a 1024. Esto limita significativamente la cantidad de clientes que se pueden manejar con el modelo single-threaded actual. En cambio poll(2) permite **manejar muchos más clientes**. La interfaz utilizada permite cambiar la implementación del selector sin mayores complicaciones.



5.3 Extensiones de Monitoreo

5.3.1 Métricas Avanzadas

Creemos que se podrían agregar métricas avanzadas de monitoreo, como la **latencia promedio** por conexión, el **throughput individual** por cliente, y hasta features más complejas como la **distribución geográfica** de los usuarios y mecanismos para detectar **patrones de tráfico anómalos**.

5.4 Extensiones de Seguridad

5.4.1 Filtrado de Destinos

Finalmente, creemos que otra extensión valiosa sería incorporar **reglas de filtrado de destinos**, de forma que se pueda permitir o bloquear tráfico hacia dominios o rangos de red específicos según políticas definidas por administradores. De este modo, podríamos construir servidores que se comporten como hemos visto que suelen hacer los proxies analizados en clases teóricas.

6. Conclusiones

Consideramos que lo desarrollado para el presente trabajo práctico nos permitió plasmar todo lo aprendido durante la cursada de la materia, y estamos sumamente satisfechos con el resultado al que hemos logrado llegar.

6.1 Logros Técnicos

Nuestro proyecto ha logrado implementar un servidor SOCKS5 completamente compatible con el RFC 1928, capaz de manejar múltiples conexiones de manera **robusta** utilizando multiplexación. Además, cuenta con un protocolo de administración propio (PCTP) que incorpora autenticación y autorización, con una arquitectura basada en



eventos la cual asegura una **gestión eficiente de recursos y escalabilidad**. También se implementó un sistema de estadísticas en tiempo real y un buffer circular de logs, lo que facilita el monitoreo de actividad del servidor durante ejecución. El desarrollo fue validado con múltiples pruebas y *scripts* de testeo, asegurando un comportamiento estable bajo cargas razonables y esperables.

6.2 Valor Técnico del Proyecto

En cuanto a la arquitectura de software, mediante este trabajo logramos evidenciar dominio en el **diseño de protocolos de red, programación asíncrona** y orientada a **eventos**, como también de manejo de **estados** complejos mediante máquinas de estados finitos y parsing robusto. Asimismo, el uso de multiplexación de conexiones en un único hilo evidencia conocimiento profundo de técnicas de concurrencia sin hilos. La implementación en C, junto con una gestión cuidadosa de memoria, validaciones estrictas y logs detallados, contribuyó a desarrollar un sistema **estable, eficiente y fácilmente extensible**.

6.3 Aplicabilidad Real

El servidor desarrollado está listo para ser utilizado como un proxy de datos TCP. Además, sirve como base para desarrollar extensiones más complejas en el futuro, como las que hemos mencionado en el apartado **5. Posibles extensiones**.



7. Ejemplos de prueba

7.1 Pruebas Básicas de Conectividad

Se pueden realizar pruebas básicas ejecutando las siguientes secuencias de comandos. Es recomendable realizarlo en dos terminales separadas para poder visualizar de manera más cómoda los outputs en cada una

```
# Terminal 1: Iniciar servidor
./bin/socks5d -u admin:password -d DEBUG

# Terminal 2: Test con curl
curl --socks5-hostname 127.0.0.1:1080 \
  --proxy-user admin:password \
  http://httpbin.org/ip
```

Cuyo resultado esperado es

```
{
  "origin": "YOUR_PUBLIC_IP"
}
```

7.2 Múltiples conexiones

Se puede realizar una prueba básica de múltiples conexiones mediante el siguiente *script*

```
# Test de carga básica
for i in {1..10}; do
  curl --socks5-hostname 127.0.0.1:1080 \
    --proxy-user admin:password \
```



```
http://httpbin.org/delay/1 &  
done  
wait
```

7.4 Pruebas avanzadas de rendimiento

Para realizar una prueba de transferencia descargando archivos grandes

```
# Terminal 1: Servidor con logs detallados  
./bin/socks5d -u admin:password -p 1081 -P 9090 -l 127.0.0.1  
  
# Terminal 2: Descarga de archivo grande para probar buffering  
curl --socks5-hostname 127.0.0.1:1080 \  
    --proxy-user admin:password \  
    -o /tmp/test_file.zip \  
    http://httpbin.org/bytes/10485760 # 10MB  
  
# Monitorear estadísticas en tiempo real  
./bin/client -h localhost -p 9090  
USER admin  
+OK Please send password  
PASS password  
+OK Successfully logged in  
STATS  
+OK Sending stats...  
current_connections: 1  
total_connections: 2  
current_bytes_proxied: 0  
total_bytes_proxied: 102730
```

Y también un testeo de conexiones concurrentes intensivas

```
# Corremos el ciclo y luego analizamos los valores via el cliente  
for i in {1..400}; do  
    curl --socks5-hostname 127.0.0.1:1080 \  
        --proxy-user admin:password \  
        --max-time 30 \  
        http://httpbin.org/delay/$(($RANDOM % 5 + 1)) \  
done
```




```
-s -o /dev/null &  
done  
  
./bin/client -h localhost -p 9090  
USER admin  
+OK Please send password  
PASS password  
+OK Successfully logged in  
STATS  
+OK Sending stats...  
current_connections: 0  
total_connections: 400  
current_bytes_proxied: 0  
total_bytes_proxied: 246542
```



8. Guía de instalación detallada y precisa

Son requisitos previos para la instalación los comandos make y gcc. Una vez descomprimido el proyecto ejecutar los siguientes comandos:

```
# Compilación mediante Makefile
```

```
make all
```

```
# Compilación únicamente del servidor
```

```
make server
```

```
# Compilación únicamente del cliente
```

```
make client
```

```
# Ejecución del servidor
```

```
./bin/socks5d [opciones]
```

Opciones disponibles:

Opción	Descripción
-----	-----
-h	Muestra ayuda y termina
-l <addr>	Dirección donde se servirá el proxy SOCKS
-L <addr>	Dirección donde se servirá el protocolo PCTP
-p <port>	Puerto SOCKS (por defecto: 1080)
-P <port>	Puerto PCTP (por defecto: 8080)
-u <user:pass>	Agrega un usuario ADMIN inicial (máximo 10)
-v	Muestra la versión y termina
-d <log_level>	Nivel de log: DEBUG, INFO, WARN, ERROR, NONE

```
# Ejecución del cliente
```

```
./bin/client -h <addr> -p <port> [-d <log_level>]
```

9. Instrucciones para la configuración

El servidor SOCKS5 se puede configurar mediante el protocolo desarrollado. El mismo permite la configuración de usuarios y la configuración de los buffers de entrada y salida de las conexiones del servidor.



9.1 Comandos de autenticación con PCTP

```
USER [username]      # Inicio de autenticación
PASS [password]      # Finalización de autenticación
```

9.2 Comandos principales de PCTP

```
STATS                # Estadísticas del servidor
LOGS [N]              # Últimos N logs de conexión al proxy
CONFIG IO=[BYTES]     # Asigna tamaño a los buffers de IO
ADD ADMIN             # Agregar usuario administrador
ADD BASIC             # Agregar usuario básico
DEL [username]        # Eliminar usuario
LIST                  # Listar usuarios registrados
EXIT                  # Terminar conexión
```

10. Ejemplos de configuración y monitoreo

Vía el protocolo desarrollado por el equipo es posible realizar configuraciones en nuestro servidor y un monitoreo de estadísticas del mismo. Si se desea testear las funcionalidades principales de nuestro protocolo PCTP, el siguiente fragmento de código ejemplifica un flujo habitual que incluye autenticación, retrieving de estadísticas, configuración, agregado de usuarios y finalización.

```
# Conectar al servidor PCTP via el cliente (en DEBUG)
.bin/client -h localhost -p 9090 -d DEBUG

# Secuencia de comandos
USER admin
PASS password
STATS
LOGS 5
CONFIG IO=5000
ADD BASIC
USER testuser
PASS testpass
LIST
```



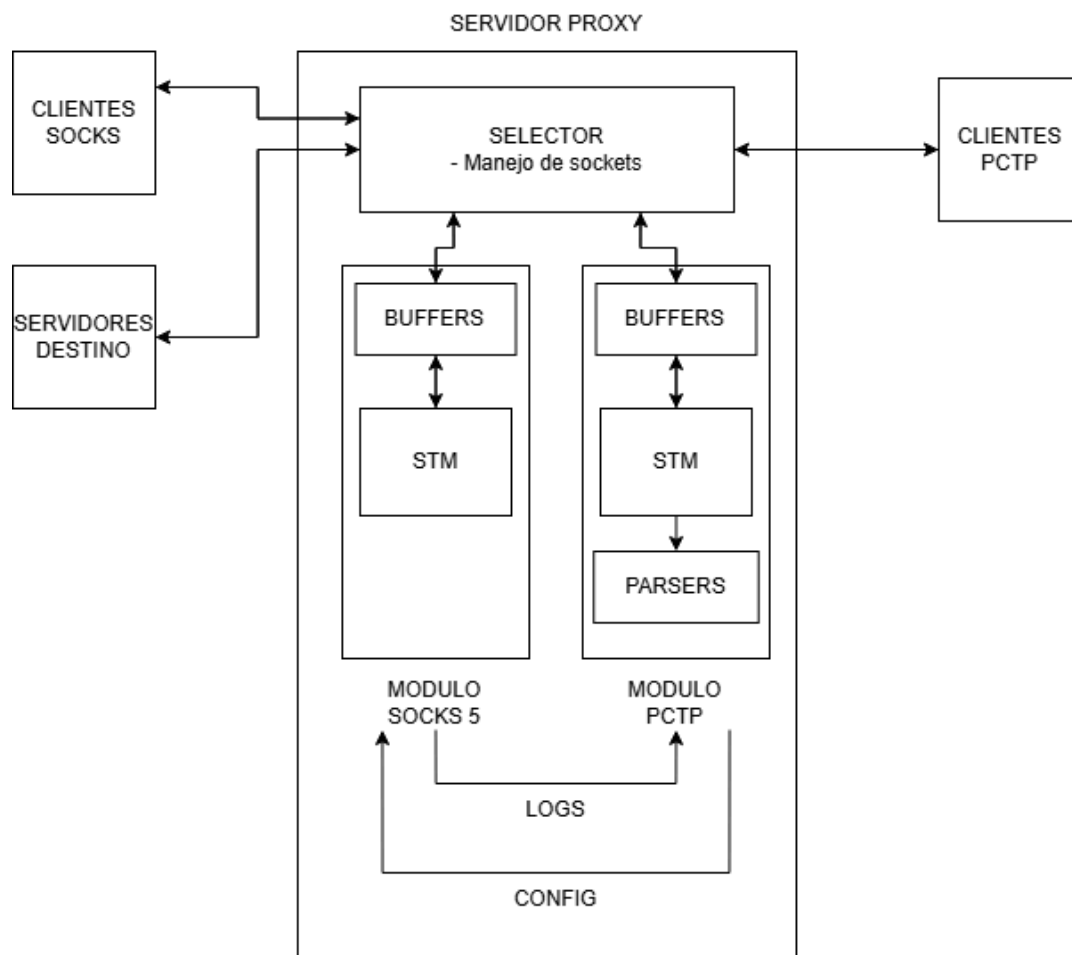
EXIT

Si se desea hacer gestión de los usuarios con sus distintos roles, se puede hacer uso de los comandos ADD (role), indicando las credenciales para el nuevo usuario a agregarse. Un ejemplo de este flujo, incluyendo las respuestas del servidor, se puede observar en el siguiente ejemplo:

```
./bin/client -h localhost -p 9090
USER admin
+OK Please send password
PASS password
+OK Successfully logged in
LIST
+OK Sending user list...
admin    Admin
ADD BASIC
+OK Please provide new user credentials
USER basic123
+OK Please send password
PASS pass123
+OK Successfully added user
LIST
+OK Sending user list...
admin    Admin
basic123    Basic
DEL basic123
+OK Successfully deleted user
LIST
+OK Sending user list...
admin    Admin
```

11. Documento de diseño del proyecto

Para un mayor entendimiento de la arquitectura de nuestro proyecto, hemos diseñado diversos diagramas en ASCII que permiten entender a simple vista los distintos estados de cada módulo del servidor y como se interconectan para lograr su objetivo



A su vez, creamos un RFC para el protocolo de monitoreo y configuración del servidor. A continuación, embebemos este documento para un mayor entendimiento de sus capacidades, comandos y funcionalidades



Trabajo Práctico Especial
Internet-Draft
Intended status: Informational
Expires: 19 December 2025

Barthelemy Augusto
Garfi Valentín
Salama Nicole
Searles Ignacio

Proxy Configuration and Tracking Protocol (PCTP) - Version 1

Abstract

Este documento describe el protocolo de configuración desarrollado para el Trabajo Especial de la materia Protocolos de Comunicación durante la cursada del primer cuatrimestre del año 2025.

El objetivo del protocolo es el de facilitar la configuración y obtención de métricas de un servidor proxy de manera remota y durante la ejecución del mismo.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 19 December 2025.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

Expires 19 December 2025

[Page 1]

Internet-Draft

PCTP

July 2025

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights



and restrictions with respect to this document.

Table of Contents

1. Operación básica	2
2. La fase de AUTORIZACIÓN	3
3. La fase PRINCIPAL	5
Dirección de los autores	7

1. Operación básica

Inicialmente, el equipo servidor inicia el servicio PCTP escuchando el puerto TCP 8080. Cuando un equipo cliente quiere hacer uso del servicio, establece una conexión TCP con el equipo servidor. Cuando la conexión se ha establecido, el servidor PCTP espera autenticación. El cliente y el servidor POP3 a partir de entonces intercambian órdenes y respuestas (respectivamente) hasta que se cierre o interrumpa la conexión.

Las órdenes en el protocolo PCTP consisten en una serie de palabras claves en mayúsculas, posiblemente seguidas de un argumento. Todos las órdenes terminan con un símbolo LF (Line Feed, Salto de Línea) o con un par CRLF (Carriage Return Line Feed, Retorno de Carro Nueva Línea). Las órdenes y sus argumentos están compuestos de caracteres ASCII imprimibles, separados por un sólo carácter ESPACIO. Las órdenes tiene tres, cuatro o cinco caracteres de longitud por palabra y pueden estar compuestas de hasta dos palabras clave. Cada argumento puede tener hasta 24 caracteres, si se proveen más de 24 caracteres, la cadena argumento será procesada hasta los primeros 24.

Las respuestas en el PCTP están formada por un indicador de estado y una palabra clave, posiblemente seguida de información adicional. Todas las respuestas están terminadas por un símbolo LF. Las respuestas pueden tener hasta 1024 caracteres de longitud, incluyendo el LF del final. Actualmente hay dos indicadores de estado: el positivo ("OK") y el negativo ("-ERR"). Los servidores DEBEN enviar "+OK" y el "-ERR" en mayúsculas.

Las respuestas a ciertas órdenes son multilínea. En esos casos, que se indicarán claramente más adelante, tras enviar la primera línea de la respuestas y un LF, pueden enviarse más líneas, cada una de ellas también terminadas por el LF. Cuando se han enviado todas las líneas de la respuesta, Se envía una línea final que consiste en un LF. De este modo, una respuesta multilínea termina con los 2 octetos "LFLF".

Una sesión PCTP pasa por varios estados durante su periodo de vida. Una vez abierta la conexión TCP, la sesión entra en la fase de AUTORIZACIÓN. En esta fase, el cliente debe identificarse ante el servidor PCTP como administrador. Una vez el cliente lo realiza con éxito, la sesión entra en la fase PRINCIPAL. En esta fase, el cliente envía órdenes al servidor PCTP. Tras el envío por el cliente de la orden EXIT, la sesión entra en la fase de AUTORIZACIÓN. el servidor PCTP libera



cualquier recurso adquirido y dice adiós cerrando la conexión TCP.

Un servidor DEBE responder a una orden no reconocida, no implementada o no válida sintácticamente, con un indicador de estado negativo. El servidor DEBE responder a una orden enviada en una fase incorrecta con un indicador de estado negativo.

2. La fase de AUTORIZACIÓN

Una vez que el cliente PCTP abre la conexión TCP, la sesión entra en la fase de AUTORIZACIÓN. En esta fase, el cliente debe autenticarse ante el servidor PCTP utilizando el siguiente mecanismo:

```
USER <nombre_de_usuario>
PASS <contraseña>
```

Ambos comandos deben ser enviados en ese orden. Primero se envía "USER", y si el nombre de usuario es válido, el servidor responde positivamente y solicita la contraseña. Luego se envía "PASS" con la contraseña correspondiente. Si las credenciales son correctas y el usuario tiene permisos administrativos, se concede el acceso y la sesión PCTP entra en la fase PRINCIPAL.

Si el servidor PCTP rechaza el nombre de usuario o la contraseña, responderá con un indicador de estado negativo. En ese caso, el usuario debe volver a enviar un usuario o contraseña válido respectivamente.

Los comandos "USER" y "PASS" solo son válidos durante esta fase y durante el proceso de añadir a un nuevo usuario mediante el comando "ADD <rol>". Si se envían en otro momento, el servidor responderá con un indicador de estado negativo.

Formato:

```
USER <nombre_usuario>
```

Argumentos:

nombre_usuario: cadena ASCII alfanumérica (acotada a 24 caracteres)

Ejemplo:

C: USER admin

S: +OK Please send password

```
PASS <contraseña>
```

Argumentos:

contraseña: cadena ASCII alfanumérica (acotada a 24 caracteres)



Ejemplo:

C: PASS admin123

S: +OK Successfully logged in

Tras devolver una respuesta positiva al comando PASS, la sesión PCTP entra en la fase PRINCIPAL.

3. La fase PRINCIPAL

Una vez que el cliente se ha identificado a sí mismo con éxito ante el servidor PCTP, la sesión PCTP se encuentra en la fase PRINCIPAL. El cliente ahora puede enviar repetidamente cualquiera de las órdenes PCTP que detallaremos a continuación. Tras recibir cada orden, el servidor PCTP envía una respuesta. Eventualmente, el cliente puede enviar la orden EXIT y cerrar la sesión PCTP.

Aquí se muestran las órdenes PCTP válidas en la fase de PRINCIPAL.

STATS

Argumentos: ninguno

Restricciones:

sólo puede utilizarse en la fase PRINCIPAL.

Comentario:

El servidor PCTP enviará una respuesta positiva seguida de múltiples líneas que contendrán información del servidor proxy.

Para simplificar el análisis, todos los servidores PCTP deben usar un formato determinado para los listados de datos del servidor. Cada línea debe estar formada por un par clave valor, en ese orden, separado por un carácter ':' y un ESPACIO, finalizando el ítem con un LF.

Ejemplos:

C: STATS

S: +OK Sending stats...

S: current_connections: 12

S: total_connections: 148

S: current_bytes_proxied: 1234

S: total_bytes_proxied: 409600

S: (LF)

LOGS [N]

Argumentos:

Un número de logs (opcional) para imprimir.



Restricciones:

solo puede darse en la fase PRINCIPAL

Comentario:

Si se envía el argumento, el servidor PCTP devuelve una respuesta positiva seguida por N líneas conteniendo información de los últimos N logs. Si N supera la cantidad de logs disponibles, se devuelven todos los logs disponibles.

Si no se da ningún argumento, el servidor PCTP devuelve una respuesta positiva seguida por 100 líneas con los últimos 100 logs disponibles.

Cada línea de log tiene el siguiente formato:

```
YYYY-MM-DDTHH:MM:SSZ <usuario> <tipo de registro> <IP cliente> <puerto
cliente>
<IP destino> <puerto destino> <código de respuesta del proxy> <bytes
transferidos> <estado de autenticación>
```

Ejemplo:

```
C: LOGS 3
S: +OK Sending logs...
S: 2025-07-13T13:55:03Z    username    A    127.0.0.1    50514
34.145.113.24  443    0    15750  Auth
S: 2025-07-13T13:54:08Z    username    A    127.0.0.1    33112
34.145.113.24  443    0    15750  Auth
S: 2025-07-13T13:54:05Z    username    A    127.0.0.1    33096
34.145.113.24  443    0    15750  Auth
S: (LF)
```

CONFIG IO=<valor>

Argumentos:

Una configuración específica, actualmente solo se acepta "IO=<valor>".

Restricciones:

solo puede utilizarse en la fase PRINCIPAL.

Comentario:

Esta orden permite modificar la configuración del servidor en tiempo de ejecución. Actualmente, la única opción válida es IO, que define el tamaño del buffer de entrada/salida utilizado por las conexiones proxy, el valor ingresado es acotado inferiormente por un mínimo de 4096 bytes antes de ser asignado para el correcto funcionamiento del servidor.

Ejemplo:



C: CONFIG IO=8000
S: +OK Setting IO buffers size

ADD <rol>

Argumentos:
Un rol: ADMIN o BASIC

Restricciones:
solo puede utilizarse en la fase PRINCIPAL.

Comentario:
Permite agregar un nuevo usuario con el rol indicado. Luego de esta orden, el servidor solicitará las credenciales para el nuevo usuario usando las órdenes USER y PASS.

Ejemplo:
C: ADD BASIC
S: +OK Please provide new user credentials
C: USER newUser
S: +OK Please send password
C: PASS newUserPass
S: +OK Successfully added user

LIST

Argumentos: ninguno

Restricciones:
solo puede utilizarse en la fase PRINCIPAL.

Comentario:
Muestra una lista de todos los usuarios existentes y su respectivo rol.

La respuesta será una lista multilínea, con el nombre de usuario seguido de un TAB y su rol.

Ejemplo:
C: LIST
S: +OK Sending user list...
S: username Admin
S: newUser Basic
S: newAdmin Admin
S: (LF)

EXIT

Argumentos: ninguno



Restricciones:

puede utilizarse solamente en la fase PRINCIPAL.

Comentario:

Termina la sesión PCTP. El servidor libera cualquier recurso asociado y cierra la conexión TCP.

Ejemplo:

C: EXIT
S: +OK Done

Dirección de los autores

Barthelemy Augusto
Email: abarthelemysola@itba.edu.ar

Garfi Valentin
Email: vgarfi@itba.edu.ar

Salama Nicole
Email: nsalama@itba.edu.ar

Searles Ignacio
Email: isearles@ejemplo.edu.ar

Expires 19 December 2025

[Page 7]