Trabajo Práctico Especial - Protocolos de Comunicación

Servidor Proxy SOCKSv5

Materia: 72.07 - Protocolos de Comunicación
Cuatrimestre: 2C 2025
Fecha de Entrega: 11 Diciembre 2025

Integrantes del Grupo

Nombre	Legajo	Email
Octavio Zacagnino	64255	ozacagnino@itba.edu.ar
Facundo Lasserre	62165	flasserre@itba.edu.ar

Índice

---

1. Descripción de Protocolos y Aplicaciones

1.1 Protocolo SOCKSv5

El servidor implementa el protocolo SOCKS versión 5 según lo especificado en RFC 1928. SOCKS (Socket Secure) es un protocolo de Internet que permite a los clientes enrutar paquetes de red a través de un servidor intermediario (proxy).

1.1.1 Fases del Protocolo

Fase 1: Negociación de Método de Autenticación (Handshake)

El cliente inicia la conexión enviando un mensaje de saludo:

```
 +----+----------+----------+
|VER | NMETHODS | METHODS  |
+----+----------+----------+
| 1  |    1     | 1 to 255 |
+----+----------+----------+
```

Donde:

- `VER` : Versión del protocolo (0x05 para SOCKSv5)
- `NMETHODS` : Número de métodos de autenticación soportados
- `METHODS` : Lista de métodos de autenticación

El servidor responde con:

```
+----+--------+
|VER | METHOD |
+----+--------+
| 1  |   1    |
+----+--------+
```

Nuestra implementación soporta:

- `0x00` : Sin autenticación (NO SOPORTADO - rechazado)
- `0x02` : Autenticación Usuario/Contraseña (OBLIGATORIO)
- `0xFF` : No hay método aceptable

Fase 2: Autenticación (RFC 1929)

Una vez negociado el método 0x02, se procede a la subnegociación de autenticación.

Fase 3: Solicitud de Conexión (Request)

```
+----+-----+-------+------+----------+----------+
|VER | CMD |  RSV  | ATYP | DST.ADDR | DST.PORT |
+----+-----+-------+------+----------+----------+
| 1  |  1  | X'00' |  1   | Variable |    2     |
+----+-----+-------+------+----------+----------+
```

Donde:

- `CMD` : Comando (0x01 = CONNECT, único soportado)
- `ATYP` : Tipo de dirección
  - 0x01: IPv4 (4 bytes)
  - 0x03: FQDN (1 byte longitud + dominio)
  - 0x04: IPv6 (16 bytes)
- `DST.ADDR` : Dirección de destino
- `DST.PORT` : Puerto de destino (big-endian)

Fase 4: Respuesta del Servidor

```
+----+-----+-------+------+----------+----------+
|VER | REP |  RSV  | ATYP | BND.ADDR | BND.PORT |
+----+-----+-------+------+----------+----------+
| 1  |  1  | X'00' |  1   | Variable |    2     |
+----+-----+-------+------+----------+----------+
```

Códigos de respuesta (REP) implementados:

- `0x00` : Éxito
- `0x01` : Fallo general del servidor SOCKS
- `0x03` : Red no alcanzable
- `0x04` : Host no alcanzable
- `0x05` : Conexión rechazada
- `0x07` : Comando no soportado
- `0x08` : Tipo de dirección no soportado

Fase 5: Transferencia de Datos

Una vez establecida la conexión, el proxy actúa como un relay bidireccional transparente entre el cliente y el servidor de origen.

1.2 Protocolo de Autenticación RFC 1929

La autenticación Username/Password sigue el estándar RFC 1929.

Solicitud del Cliente:

```
+----+------+----------+------+----------+
|VER | ULEN |  UNAME   | PLEN |  PASSWD  |
+----+------+----------+------+----------+
| 1  |  1   | 1 to 255 |  1   | 1 to 255 |
+----+------+----------+------+----------+
```

Donde:

- `VER` : Versión de subnegociación (0x01)
- `ULEN` : Longitud del nombre de usuario
- `UNAME` : Nombre de usuario
- `PLEN` : Longitud de la contraseña

- `PASSWD` : Contraseña

Respuesta del Servidor:

```
+----+--------+
|VER | STATUS |
+----+--------+
| 1  |   1    |
+----+--------+
```

Donde:

- `VER` : 0x01
- `STATUS` : 0x00 = éxito, cualquier otro valor = fallo

1.3 Protocolo de Gestión y Monitoreo

Se diseñó un protocolo de texto simple inspirado en protocolos como SMTP y POP3, optimizado para facilidad de uso y depuración.

1.3.1 Especificación del Protocolo

Características:

- Protocolo basado en texto (ASCII)
- Orientado a líneas (delimitador: CRLF o LF)
- Case-insensitive para comandos
- Puerto por defecto: 8080
- Conexión TCP persistente

Formato de Respuestas:

```
+OK <mensaje>     # Operación exitosa
-ERR <mensaje>    # Error en la operación
```

Banner de Conexión:

Al conectarse, el servidor envía:

```
+OK SOCKS5 Management Server v1.0
+OK Use AUTH <user> <pass> to authenticate
```

1.3.2 Comandos Disponibles

Comando	Sintaxis	Descripción	Requiere Auth
AUTH	`AUTH <user> <pass>`	Autenticación del administrador	No
STATS	`STATS`	Muestra métricas del servidor	Sí
USERS	`USERS`	Lista usuarios registrados	Sí
ADDUSER	`ADDUSER <user> <pass>`	Agrega usuario en runtime	Sí
DELUSER	`DELUSER <user>`	Elimina usuario en runtime	Sí
HELP	`HELP`	Muestra ayuda de comandos	Sí
QUIT	`QUIT`	Cierra la conexión	No

1.3.3 Detalle de Comandos

AUTH - Autenticación

```
C: AUTH admin secretpass
S: +OK Authenticated successfully. Type HELP for commands.
```

STATS - Métricas del Sistema

```
C: STATS
S: +OK Statistics:
S: +OK   Total connections:    150
S: +OK   Current connections:  5
S: +OK   Bytes transferred:    1048576
S: +OK   Bytes sent:           524288
S: +OK   Bytes received:       524288
S: +OK   Successful:           145
S: +OK   Failed:               5
```

USERS - Listar Usuarios

```
C: USERS
S: +OK Users (3 total):
S: +OK   USER admin
S: +OK   USER user1
S: +OK   USER user2
```

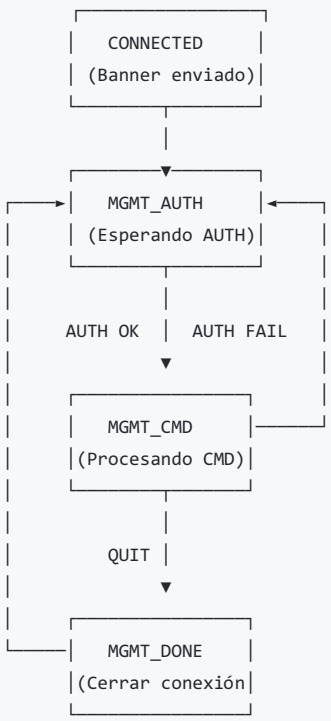
ADDUSER - Agregar Usuario en Runtime

```
C: ADDUSER newuser newpass123
S: +OK User 'newuser' added successfully
```

DELUSER - Eliminar Usuario en Runtime

```
C: DELUSER olduser
S: +OK User 'olduser' removed successfully
```

1.3.4 Diagrama de Estados del Protocolo



1.4 Aplicación Servidor (socks5d)

El servidor `socks5d` es la aplicación principal que implementa:

- Proxy SOCKSv5 (RFC 1928)
- Autenticación Usuario/Contraseña (RFC 1929)
- Servidor de Gestión y Monitoreo

Características Técnicas:

- Arquitectura single-threaded con I/O no bloqueante

- Multiplexación de conexiones mediante `select()`
- Máquinas de estado finitas para manejo de protocolos
- Soporte para 500+ conexiones simultáneas
- DNS asíncrono mediante threads auxiliares

1.5 Aplicación Cliente de Gestión (client)

El cliente `client` proporciona una interfaz interactiva para conectarse al servidor de gestión.

Características:

- Interfaz de línea de comandos interactiva
- Conexión TCP al puerto de gestión
- Soporte para todos los comandos del protocolo

---

2. Problemas Encontrados

2.1 Resolución DNS No Bloqueante

Problema: La función `getaddrinfo()` de POSIX es bloqueante y no existe una versión asíncrona estándar portable.

Solución: Implementamos un sistema de resolución DNS asíncrona utilizando:

1. Un thread auxiliar (`resolve_thread`) que ejecuta `getaddrinfo()`
2. El mecanismo `selector_notify_block()` del framework provisto para notificar al event loop principal cuando la resolución termina

```
static void *resolve_thread(void *arg) {
    struct resolve_args *args = arg;
    // ... getaddrinfo bloqueante en thread separado ...
    selector_notify_block(args->selector, args->client_fd);
    return NULL;
}
```

2.2 Conexión a Múltiples Direcciones IP

Problema: Cuando un FQDN resuelve a múltiples direcciones IP y la primera no está disponible, el cliente no debería fallar inmediatamente.

Solución: Implementamos un algoritmo de fallback que:

1. Obtiene todas las direcciones IP resueltas para un FQDN
2. Intenta conectar a cada una secuencialmente
3. Solo reporta error si ninguna dirección está disponible

```
while (current != NULL) {
    int origin_fd = socket(current->ai_family, SOCK_STREAM, IPPROTO_TCP);
    // ... intentar conexión ...
    if (connect(origin_fd, current->ai_addr, current->ai_addrlen) < 0) {
        if (errno != EINPROGRESS) {
            current = current->ai_next;  // Intentar siguiente
            continue;
        }
    }
    // Éxito o EINPROGRESS
    break;
}
```

2.3 Lecturas y Escrituras Parciales

Problema: Las operaciones de I/O en sockets no bloqueantes pueden completarse parcialmente.

Solución: Utilizamos el módulo `buffer.c` provisto por la cátedra que maneja:

- Punteros de lectura y escritura separados
- Compactación automática del buffer
- Seguimiento de bytes disponibles para lectura/escritura

2.4 Manejo de Conexiones Concurrentes

Problema: Gestionar el ciclo de vida de conexiones que comparten recursos.

Solución: Implementamos un sistema de conteo de referencias:

```
struct socks5 {
    int references;  // client_fd + origin_fd
    // ...
};
```

La conexión solo se libera cuando `references` llega a 0.

2.5 Compatibilidad MacOS / Linux

Problema: `MSG_NOSIGNAL` no existe en MacOS.

Solución: Definimos un fallback:

```
#ifndef MSG_NOSIGNAL
#define MSG_NOSIGNAL 0
#endif
```

Y en MacOS, el manejo de SIGPIPE se hace a nivel de proceso.

---

3. Limitaciones de la Aplicación

3.1 Limitaciones Funcionales

Limitación	Descripción	RFC
Solo CONNECT	No se soporta BIND ni UDP ASSOCIATE	RFC 1928
Sin GSSAPI	Solo autenticación username/password	RFC 1929
Sin cifrado	El tráfico no está cifrado (usar con túnel TLS)	-
Usuarios en memoria	Los usuarios agregados en runtime se pierden al reiniciar	-
Máx. 10 usuarios CLI	Límite de usuarios en línea de comandos	-

3.2 Limitaciones Técnicas

Limitación	Valor	Causa
Max conexiones	~1000	`FD_SETSIZE` de `select()`
Buffer por conexión	8KB	Definido en `BUFFER_SIZE`
Longitud FQDN	255 bytes	RFC 1928
Usuario/Pass	255 bytes cada uno	RFC 1929

3.3 Limitaciones de Seguridad

- **Sin rate limiting:** Un cliente malicioso podría saturar el servidor
- **Sin blacklist de IPs:** No hay mecanismo para bloquear IPs
- **Logs en memoria:** El registro de acceso se pierde al reiniciar
- **Admin password en CLI:** Visible en `ps` y en historial del shell

---

4. Posibles Extensiones

4.1 Extensiones de Corto Plazo

1. **Persistencia de usuarios:** Guardar usuarios en archivo de configuración
2. **Logs persistentes:** Escribir logs de acceso a archivo con rotación
3. **Rate limiting:** Limitar conexiones por IP/usuario
4. **Blacklist/Whitelist:** Control de acceso por IP o dominio

4.2 Extensiones de Mediano Plazo

1. **Soporte epoll/kqueue:** Reemplazar `select()` para mayor escalabilidad
2. **Pool de threads para DNS:** Múltiples resoluciones DNS concurrentes
3. **Métricas Prometheus:** Endpoint para scraping de métricas
4. **TLS para gestión:** Cifrar el protocolo de gestión

4.3 Extensiones de Largo Plazo

1. **Clustering:** Múltiples instancias con estado compartido
2. **BIND y UDP ASSOCIATE:** Comandos adicionales de SOCKS5
3. **Autenticación GSSAPI:** Para entornos empresariales
4. **Interfaz Web:** Dashboard para monitoreo y gestión

---

5. Conclusiones

5.1 Objetivos Cumplidos

El proyecto cumple satisfactoriamente todos los requerimientos funcionales de la primera entrega:

#	Requerimiento	Estado
1	500+ conexiones concurrentes	✅ Probado con 1000
2	Autenticación RFC 1929	✅ Implementado
3	IPv4, IPv6, FQDN	✅ Soportados
4	Robustez múltiples IPs	✅ Implementado
5	Códigos de error SOCKS	✅ Completos
6	Métricas	✅ Conexiones y bytes
7	Gestión en runtime	✅ ADDUSER/DELUSER
8	Registro de acceso	✅ Logs por conexión

5.2 Aspectos Técnicos Destacables

1. **Arquitectura no bloqueante correcta:** Todo el I/O utiliza el selector provisto por la cátedra, sin llamadas bloqueantes en el event loop principal.
2. **DNS asíncrono:** La resolución de nombres se ejecuta en un thread separado y notifica al selector mediante `selector_notify_block()`.
3. **Máquinas de estado claras:** Tanto SOCKS5 como el protocolo de gestión utilizan FSM bien definidas con transiciones explícitas.
4. **Código modular:** Separación clara entre capas (buffer, selector, stm, protocolo, aplicación).

5.3 Lecciones Aprendidas

1. **I/O no bloqueante es complejo:** Manejar lecturas/escrituras parciales y estados intermedios requiere un diseño cuidadoso.
2. **Los RFCs son la verdad:** Seguir estrictamente las especificaciones evita problemas de interoperabilidad.
3. **Testing es fundamental:** Las pruebas de estrés revelaron problemas que no eran evidentes en uso normal.
4. **La simplicidad es valiosa:** El protocolo de gestión basado en texto facilita enormemente la depuración.

---

6. Ejemplos de Prueba

6.1 Prueba de Handshake SOCKS5

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('127.0.0.1', 1080))

# HELLO: VER=5, NMETHODS=1, METHOD=0x02 (user/pass)
s.send(b'\x05\x01\x02')
response = s.recv(2)
print(f"Response: {response.hex()}")  # Esperado: 0502
s.close()
```

6.2 Prueba de Autenticación

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('127.0.0.1', 1080))

# HELLO
s.send(b'\x05\x01\x02')
s.recv(2)

# AUTH: VER=1, USER=user1, PASS=pass1
user, pwd = b'user1', b'pass1'
auth = bytes([0x01, len(user)]) + user + bytes([len(pwd)]) + pwd
s.send(auth)
response = s.recv(2)
print(f"Auth status: {response[1]}")  # 0 = éxito
s.close()
```

6.3 Prueba de Conexión Completa con curl

```
# Conexión HTTPS a través del proxy
curl -x socks5h://user1:pass1@127.0.0.1:1080 https://www.google.com

# Verificar IP pública
curl -x socks5h://user1:pass1@127.0.0.1:1080 https://api.ipify.org
```

6.4 Prueba del Protocolo de Gestión

```
# Conectar con netcat
nc 127.0.0.1 8080

# Secuencia de comandos:
# > (Banner automático)
# AUTH admin admin123
# STATS
# USERS
# ADDUSER newuser newpass
# USERS
# DELUSER newuser
# QUIT
```

6.5 Prueba de Estrés

```
import socket
import threading
import time

def socks5_handshake():
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(5)
        s.connect(('127.0.0.1', 1080))
        s.send(b'\x05\x01\x02')
        s.recv(2)
        s.close()
        return True
    except:
        return False

# 500 conexiones concurrentes
threads = [threading.Thread(target=socks5_handshake) for _ in range(500)]
start = time.time()
for t in threads: t.start()
for t in threads: t.join()
print(f"500 conexiones en {time.time() - start:.2f}s")
```

6.6 Resultados de Pruebas de Estrés

Conexiones	Exitosas	Tiempo	Throughput
100	100/100	0.01s	8,678/s
500	500/500	0.07s	7,590/s
1000	1000/1000	0.12s	8,135/s

Límite observado: ~1,018 conexiones (límite de `FD_SETSIZE`)

---

7. Guía de Instalación

7.1 Requisitos Previos

Sistema Operativo:

- Linux (probado en Ubuntu 20.04+)
- macOS (probado en Ventura+)

Dependencias:

- GCC con soporte C11
- GNU Make
- pthread library (incluida en sistemas POSIX)

Verificar dependencias:

```
gcc --version    # Requiere GCC 4.9+ o Clang 3.3+
make --version   # GNU Make 3.81+
```

7.2 Compilación

```
# Clonar el repositorio (o extraer el tarball)
git clone https://github.com/ozacagnino/tpe-protos.git
cd tpe-protos

# Compilar
make

# Verificar binarios generados
ls -la socks5d client
```

Binarios generados:

- `socks5d` : Servidor proxy SOCKS5 (~100KB)
- `client` : Cliente de gestión (~60KB)

7.3 Verificación de Instalación

```
# Verificar que el servidor muestra ayuda
./socks5d -h

# Verificar versión
./socks5d -v
```

7.4 Limpieza

```
# Eliminar archivos compilados
make clean
```

---

8. Instrucciones de Configuración

8.1 Argumentos de Línea de Comandos

Servidor (socks5d)

Argumento	Descripción	Default
`-h`	Muestra ayuda y termina	-
`-v`	Muestra versión y termina	-
`-l <addr>`	Dirección para SOCKS	0.0.0.0
`-p <port>`	Puerto para SOCKS	1080
`-L <addr>`	Dirección para gestión	127.0.0.1
`-P <port>`	Puerto para gestión	8080
`-u <user:pass>`	Agregar usuario (hasta 10)	-
`-N`	Desactivar disectores	-

Cliente (client)

Argumento	Descripción	Default
`-h`	Muestra ayuda	-
`-a <addr>`	Dirección del servidor	127.0.0.1

Argumento	Descripción	Default
`-p <port>`	Puerto de gestión	8080

8.2 Ejemplos de Configuración

Configuración mínima (un usuario):

```
./socks5d -u admin:secretpass
```

Configuración con múltiples usuarios:

```
./socks5d -u admin:adminpass -u user1:pass1 -u user2:pass2
```

Configuración en puertos personalizados:

```
./socks5d -p 9050 -P 9051 -u admin:pass
```

Solo aceptar gestión desde localhost:

```
./socks5d -L 127.0.0.1 -u admin:pass
```

Aceptar SOCKS desde cualquier interfaz:

```
./socks5d -l 0.0.0.0 -u admin:pass
```

8.3 Ejecución como Servicio

Ejemplo con systemd (Linux):

```
 # /etc/systemd/system/socks5d.service
[Unit]
Description=SOCKS5 Proxy Server
After=network.target

[Service]
Type=simple
ExecStart=/opt/socks5d/socks5d -u admin:secretpass
Restart=always

[Install]
WantedBy=multi-user.target
```

```
 sudo systemctl enable socks5d
sudo systemctl start socks5d
```

---

9. Ejemplos de Configuración y Monitoreo

9.1 Iniciar el Servidor

```
 # Iniciar con configuración básica
./socks5d -u admin:admin123 -u user1:pass1

# Salida esperada:
# [INFO] User added: admin
# [INFO] User added: user1
# [INFO] SOCKS5 server listening on 0.0.0.0:1080
# [INFO] Management server listening on 127.0.0.1:8080
# [INFO] Server started successfully. Waiting for connections...
```

9.2 Conectar con Cliente de Gestión

```
./client -a 127.0.0.1 -p 8080
```

9.3 Sesión de Monitoreo Completa

```
 $ nc 127.0.0.1 8080
+OK SOCKS5 Management Server v1.0
+OK Use AUTH <user> <pass> to authenticate

AUTH admin admin123
+OK Authenticated successfully. Type HELP for commands.

HELP
+OK Available commands:
+OK   STATS   - Show server statistics
+OK   USERS   - List registered users
+OK   ADDUSER <user> <pass> - Add a new user
+OK   DELUSER <user> - Remove a user
+OK   HELP    - Show this help
+OK   QUIT    - Close connection

STATS
+OK Statistics:
+OK   Total connections:    25
+OK   Current connections:  3
+OK   Bytes transferred:    152847
+OK   Bytes sent:           76423
+OK   Bytes received:       76424
+OK   Successful:           23
+OK   Failed:               2

USERS
+OK Users (2 total):
+OK   USER admin
+OK   USER user1

ADDUSER guest guest123
+OK User 'guest' added successfully

USERS
+OK Users (3 total):
+OK   USER admin
+OK   USER user1
+OK   USER guest

DELUSER guest
+OK User 'guest' removed successfully

QUIT
+OK Bye
```

9.4 Monitoreo de Logs de Acceso

Los logs de acceso se imprimen en stdout con el formato:

```
[timestamp] ACCESS <usuario>@<ip_cliente>:<puerto> -> <destino>:<puerto> <estado> <bytes_recv>/<bytes_sent>
```

Ejemplo:

```
[2025-12-10 15:30:45] ACCESS user1@192.168.1.100:54321 -> www.google.com:443 OK 15234/892
[2025-12-10 15:30:50] ACCESS user2@192.168.1.101:54322 -> api.example.com:80 OK 4521/156
[2025-12-10 15:31:02] ACCESS baduser@192.168.1.102:54323 -> -:0 ERROR 0/0
```

9.5 Configuración de Navegador (Firefox)

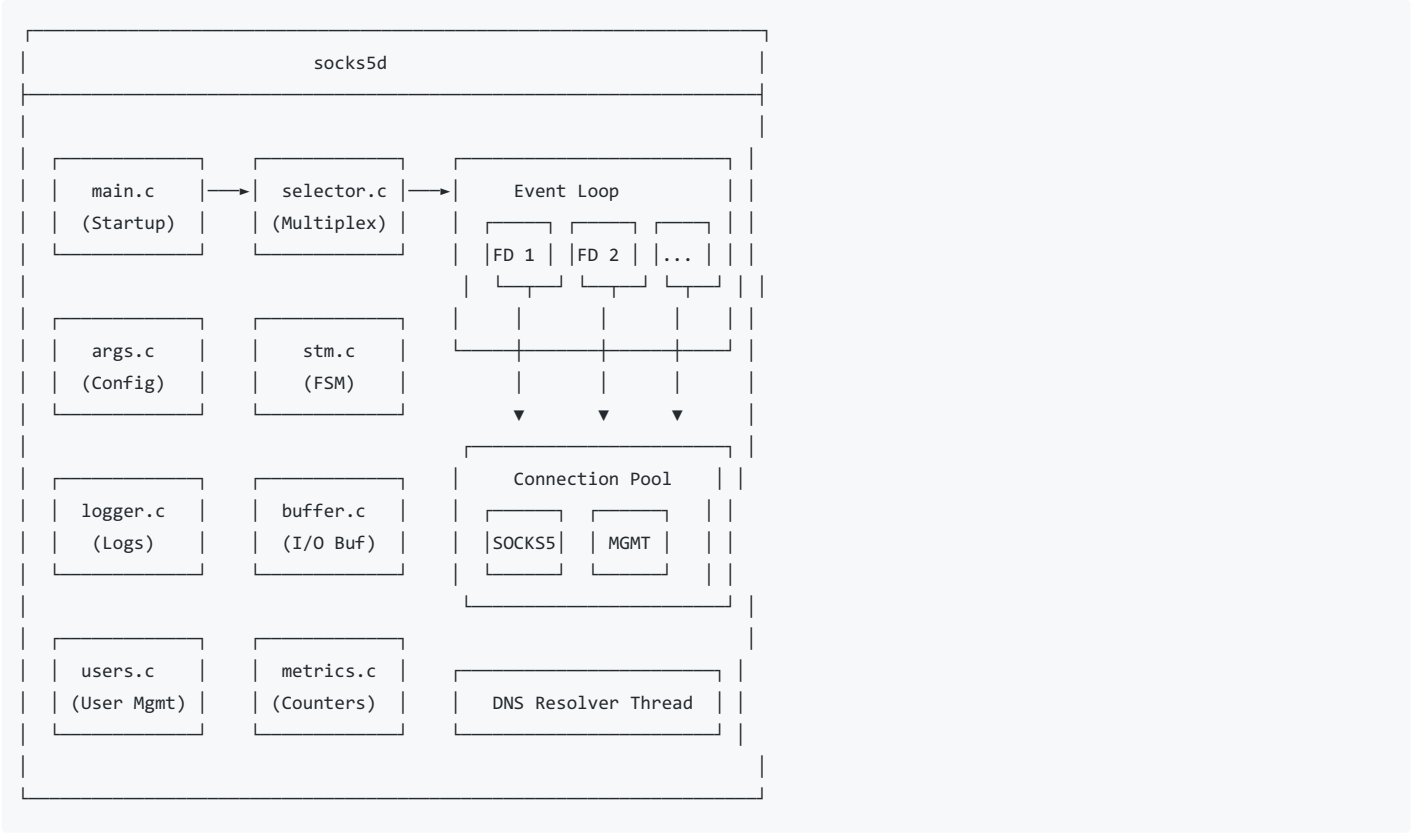
1. Ir a `about:preferences`
2. Buscar "proxy" en el buscador
3. Click en "Settings..."
4. Seleccionar "Manual proxy configuration"
5. En "SOCKS Host": `127.0.0.1`
6. En "Port": `1080`
7. Seleccionar "SOCKS v5"
8. Marcar "Proxy DNS when using SOCKS v5"
9. Click "OK"

Al navegar, Firefox solicitará las credenciales de usuario/contraseña.

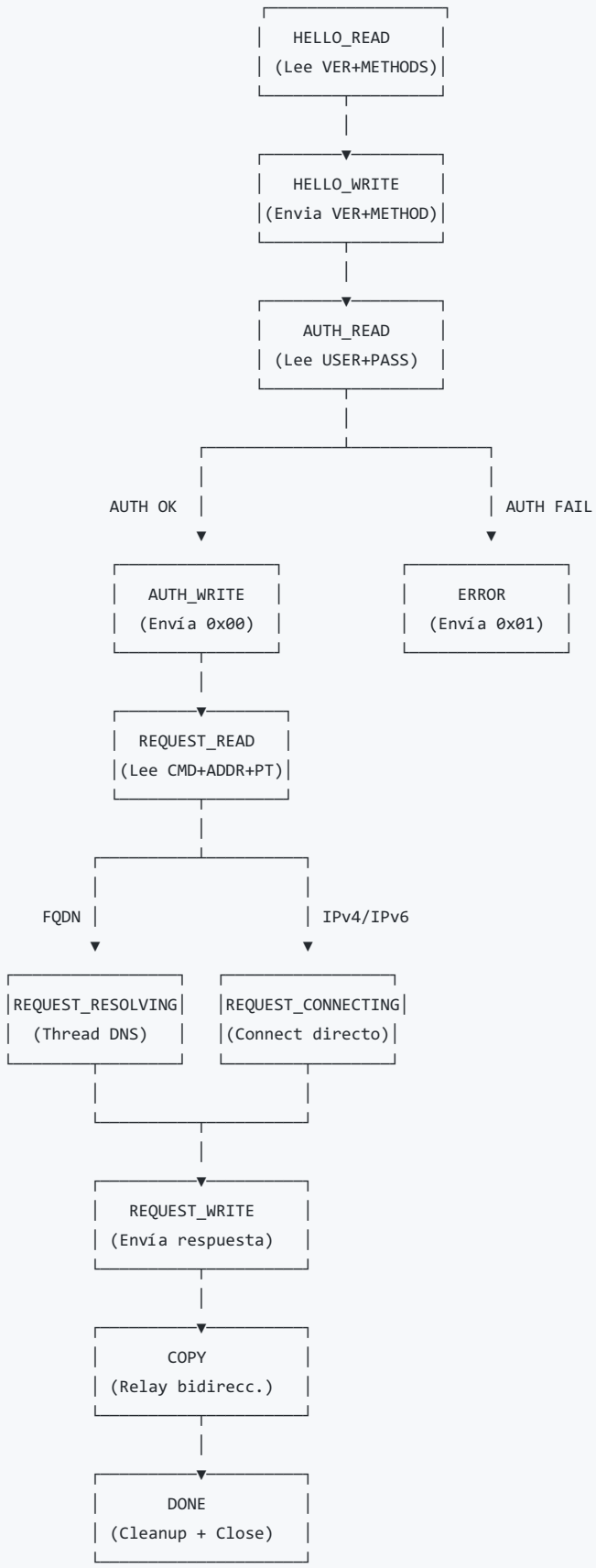
---

10. Documento de Diseño

10.1 Arquitectura General



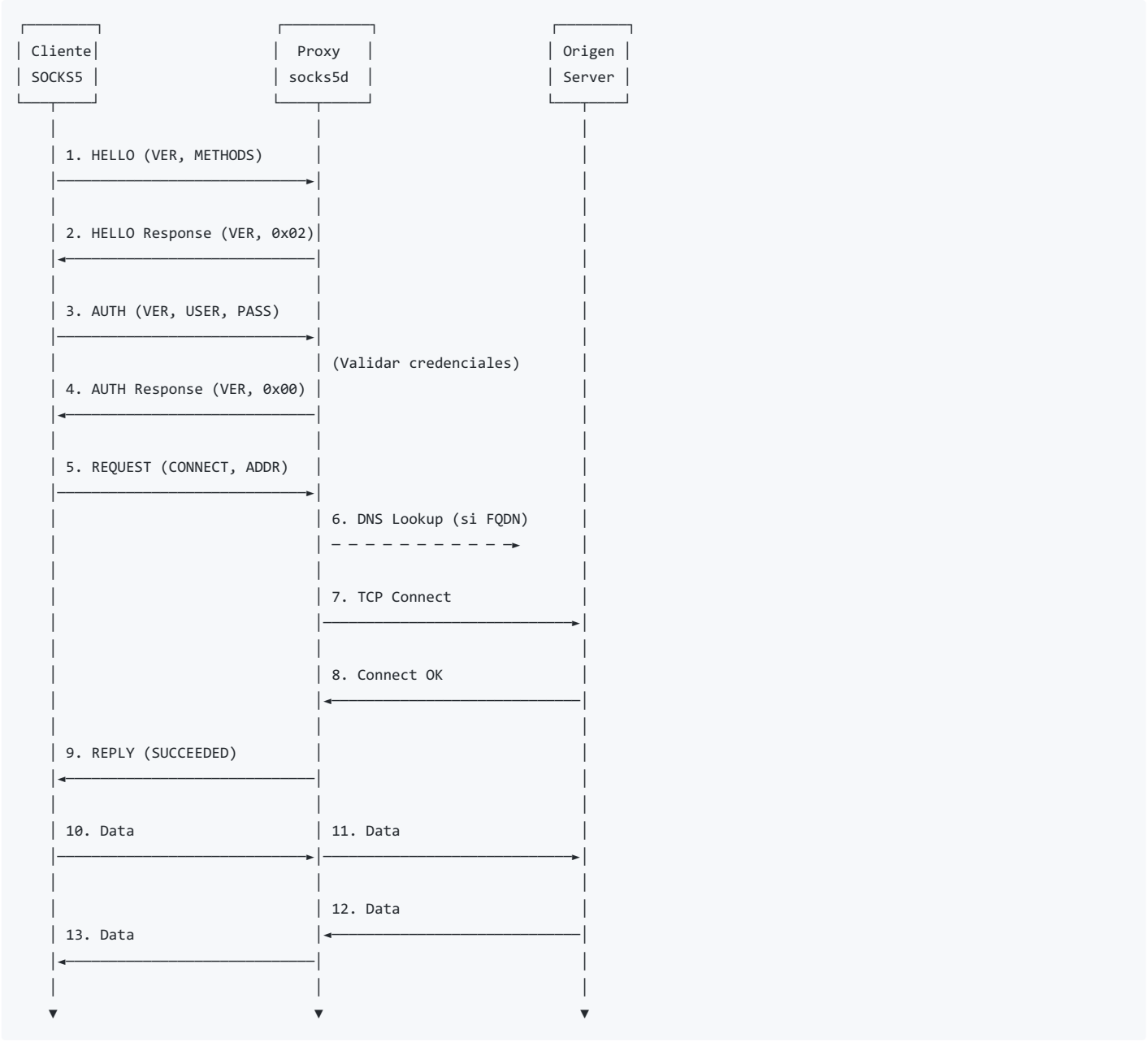
10.2 Diagrama de Estados SOCKS5



10.3 Estructura de Datos Principal

```
struct socks5 {
    // File descriptors
    int client_fd;          // Socket del cliente SOCKS
    int origin_fd;          // Socket al servidor de origen

    // Conteo de referencias
    int references;         // Cuántos FDs referencian esta estructura

    // Máquina de estados
    struct state_machine stm;

    // Buffers de I/O
    uint8_t raw_buff_read[BUFFER_SIZE];
    uint8_t raw_buff_write[BUFFER_SIZE];
    buffer read_buffer;
    buffer write_buffer;

    // Estado de cada dirección (cliente y origen)
    struct {
        struct hello_st hello;      // Estado HELLO
        struct auth_st auth;        // Estado AUTH
        struct request_st request;  // Estado REQUEST
        struct copy_st copy;        // Estado COPY
    } client;

    // Resolución DNS
    struct addrinfo *origin_resolution;
    struct addrinfo *origin_resolution_current;

    // Metadatos para logging
    char username[256];
    char target_host[256];
    uint16_t target_port;
    uint64_t bytes_sent;
    uint64_t bytes_received;

    // Referencia al selector
    fd_selector selector;
};
```

10.4 Flujo de Datos



10.5 Organización de Módulos

```
tpe-protos/
├── include/                    # Headers públicos
│   ├── args.h                  # Argumentos de línea de comandos
│   ├── buffer.h                # Buffer de I/O (cátedra)
│   ├── logger.h                # Sistema de logging
│   ├── metrics.h               # Métricas del servidor
│   ├── mgmt.h                  # Protocolo de gestión
│   ├── netutils.h              # Utilidades de red (cátedra)
│   ├── parser.h                # Parser genérico (cátedra)
│   ├── selector.h              # Multiplexor I/O (cátedra)
│   ├── socks5nio.h             # Protocolo SOCKS5
│   ├── stm.h                   # Máquina de estados (cátedra)
│   └── users.h                 # Gestión de usuarios
│
├── src/
│   ├── lib/                    # Framework de la cátedra
│   │   ├── buffer.c
│   │   ├── netutils.c
│   │   ├── parser.c
│   │   ├── selector.c
│   │   └── stm.c
│   │
│   ├── server/                 # Servidor SOCKS5
│   │   ├── main.c              # Punto de entrada
│   │   ├── args.c              # Parsing de argumentos
│   │   ├── logger.c            # Logging
│   │   ├── metrics.c           # Contadores atómicos
│   │   ├── mgmt.c              # Servidor de gestión
│   │   ├── socks5nio.c         # Implementación SOCKS5
│   │   └── users.c             # Gestión de usuarios
│   │
│   └── client/                 # Cliente de gestión
│       └── main.c
│
├── docs/                       # Documentación
│   ├── consigna.txt            # Enunciado del TP
│   ├── informe.md              # Este documento
│   ├── stress_report.md        # Informe de pruebas de estrés
│   └── stress_test.py          # Script de pruebas
│
├── patches/                    # Framework original (patches)
│
├── Makefile                    # Sistema de build
├── README.md                   # Documentación rápida
└── .gitignore                  # Archivos ignorados
```

10.6 Decisiones de Diseño

Aspecto	Decisión	Justificación
Multiplexor	`select()`	Provisto por la cátedra, portable
Threading	Single-thread + DNS auxiliar	Evita race conditions
Buffers	8KB por dirección	Balance memoria/rendimiento
Protocolo gestión	Texto plano	Facilidad de depuración
Logs	stdout	Simplicidad, permite redirección
Métricas	Atómicas	Thread-safe para DNS thread

Referencias

- RFC 1928 - SOCKS Protocol Version 5
- RFC 1929 - Username/Password Authentication for SOCKS V5
- RFC 2119 - Key words for use in RFCs
- IEEE Std 1003.1-2008 - POSIX Base Specifications
- ISO/IEC 9899:2011 - C11 Standard