



## 72.07 Protocolos de Comunicación

### **Informe del Trabajo Práctico Especial**

Link de BitBucket

<https://bitbucket.org/itba/pc-2019-02/src/master/>

#### Integrantes del Grupo n°2:

- |                          |       |
|--------------------------|-------|
| • Emilio Basualdo Cibils | 58172 |
| • Jimena Lozano          | 58095 |
| • Maite Herrán           | 57609 |
| • Fermín Gómez           | 58111 |

<b>1. Descripción detallada de los protocolos y aplicaciones desarrolladas</b>	<b>3</b>
1.1 Protocolo HPCP	3
1.1.1 Funcionamiento general	3
1.1.2 Estructuras de los mensajes	4
1.1.2.1 Estructura del request	4
1.1.3 Estructura del response	4
1.1.4 Comandos	5
1.1.4.0 HELLO (0x00)	5
1.1.4.1 AUTH (0x01)	6
1.1.4.2 CLOSE (0x02)	6
1.1.4.3 GET (0x03)	7
1.1.4.3.1 GET CONFIGURATIONS (0x03, 0x00)	8
1.1.4.3.1.1 GET TRANSFORMATION PROGRAM (0x03, 0x00, 0x00)	9
1.1.4.3.1.2 GET TRANSFORMATION PROGRAM STATUS (0x03, 0x00, 0x01)	9
1.1.4.3.1.3 GET MEDIA TYPES (0x03, 0x00, 0x02)	10
1.1.4.3.2 GET METRICS (0x03, 0x01)	11
1.1.4.3.2.1 GET CONEXIONES CONCURRENTES (0x03, 0x01, 0x00), GET ACCESOS HISTORICOS (0x03, 0x01, 0x01), GET BYTES TRANSFERIDOS (0x03, 0x01, 0x02)	12
1.1.4.4 SET (0x04)	13
1.1.4.4.1 SET CONFIGURATIONS (0x04, 0x00)	13
1.1.4.4.1.1 SET TRANSFORMATION PROGRAM (0x04, 0x00, 0x00)	14
1.1.4.4.1.2 SET TRANSFORMATION PROGRAM STATUS (0x04, 0x00, 0x01)	15
1.1.4.4.1.3 SET MEDIA TYPES (0x04, 0x00, 0x02)	16
1.2 Comentario sobre el protocolo	16
1.3 Servidor HPCP	17
1.4 Servidor proxy HTTP	17
1.5 Cliente HPCP	19
<b>2. Problemas encontrados durante el diseño y la implementación</b>	<b>21</b>
2.1 Request - response entre el cliente y el origin server	21
2.2 Qué hacer con el header accept de una request que pasa por el proxy	21
2.3 Timeout del Proxy Server y Cliente HPCP	21
2.4 Diseño del protocolo HPCP	21
<b>3. Limitaciones de la aplicación</b>	<b>23</b>
3.1 Conexiones persistentes	23
3.2 Encriptación	23
3.3 Usuarios y contraseñas hardcodeados en el código	23

3.4 Validación de función de transformación	23
3.4 Máxima cantidad de conexiones simultáneas	23
<b>4. Posibles extensiones</b>	<b>24</b>
4.1 Cantidad de conexiones al origin server	24
4.2 Loggings en archivos	24
4.3 Persistencia en los datos de las métricas y configuraciones	24
4.4 Mejora en las métricas	24
4.5 Implementación de un mecanismo de encriptación	24
4.6 Reducir cantidad de mensajes enviados	24
<b>5. Conclusiones</b>	<b>26</b>
<b>6. Ejemplos de prueba</b>	<b>27</b>
6.1 Podemos ver la inmutabilidad de los datos al pasar por el proxy con el siguiente ejemplo:	27
6.2 Ejemplos de tiempos con una conexión y con más de una conexión concurrente	27
<b>7. Guía de compilación y ejecución</b>	<b>28</b>
7.1 Ejecución del cliente HPCP	28
7.2 Ejecución del servidor	28
<b>8. Instrucciones para la configuración</b>	<b>30</b>
<b>9. Ejemplos de configuración y monitoreo</b>	<b>31</b>
9.1 Autenticación exitosa	31
9.2 Autenticación con error	31
9.3 Get transformation program status	32
9.4 Set transformation program status	33
9.5 Get concurrent connections	33
9.6 Logs del servidor	35
<b>10. Documento del diseño del proyecto</b>	<b>36</b>

# 1. Descripción detallada de los protocolos y aplicaciones desarrolladas

## 1.1 Protocolo HPCP

El HPCP es un protocolo binario con el cual se busca estandarizar la comunicación entre un usuario con una implementación de un servidor proxy para el protocolo HTTP versión 1.1. Con este protocolo, el usuario al que llamamos *administrador* podrá configurar al servidor y obtener sus métricas utilizando el mecanismo request-response.

### 1.1.1 Funcionamiento general

El servidor HPCP comienza su servicio escuchando en un puerto para quedarse a la espera de conexiones SCTP. El cliente crea una conexión SCTP en el puerto correspondiente para así poder comunicarse con el servidor. Luego de establecerse la conexión, se pasa al estado de SALUDO para negociar la versión a utilizar. Una vez que ambas partes están de acuerdo en que manejan la misma versión, se pasa al estado de AUTENTIFICACIÓN: el cliente envía usuario y contraseña y se queda a la espera de la verificación por parte del servidor. En caso de no recibir un mensaje de éxito, se le ofrece la posibilidad de autenticarse de nuevo. Caso contrario, se procede al estado de INTERCAMBIO. En este estado el cliente puede enviar requests para modificar configuraciones y recibir métricas del servidor. Si el cliente desea cerrar la conexión, se lo avisa al servidor mediante un comando.

### 1.1.2 Estructuras de los mensajes

#### 1.1.2.1 Estructura del request

	CMD	N-ARGS	ARGLEN-1	ARG-1	...	ARGLEN-N	ARG-N
# de bytes asignados	1	1	1	Variable	...	1	Variable

Dónde:

- CMD = número identificador del comando
  - HELLO 0x00
  - AUTH 0x01
  - CLOSE 0x02
  - GET 0x03
  - SET 0x04
- N-ARGS = cantidad de argumentos ( $0 \leq N-ARGS \leq 255$ )
- ARGLEN- $i$  = longitud en bytes del argumento  $i$  ( $0 \leq \text{ARGLEN-}i \leq 255, 1 \leq i \leq N-ARGS$ )
- ARG- $i$  = argumento  $i$  ( $1 \leq i \leq N-ARGS$ )

#### 1.1.3 Estructura del response

	STATUS-CODE	N-RESPTS	DATALEN-1	DATA-1	...	DATALEN-N	DATA-N
# de bytes asignados	1	1	1	Variable	...	1	Variable

Dónde:

- STATUS-CODE = código de respuesta
  - 0x00: OK
  - 0x01: ERROR sin especificación
  - 0x02: Comando ingresado inexistente
  - 0x03: Argumentos inválidos para el comando del request
  - 0x04: Credenciales incorrectas
  - 0x05: Programa de transformación ingresado inexistente
  - 0x06: Error de versión
- N-RESPTS = indica la cantidad de fragmentos de datos de respuesta ( $0 \leq N-RESPTS \leq 255$ ).
- DATALEN- $i$  = indica la longitud en bytes del fragmento de datos  $i$  ( $0 \leq \text{DATALEN-}i \leq 255, 1 \leq i \leq N$ )

- DATA- $i$  = fragmento de datos  $i$  ( $1 \leq i \leq N\text{-RESPTS}$ ).

#### 1.1.4 Comandos

Luego de establecerse una conexión SCTP entre el cliente y el servidor, el cliente debe enviar un saludo con el comando hello explicado a continuación:

##### 1.1.4.0 HELLO (0x00)

Mediante el comando 0x00, el cliente saluda al servidor y le envía la versión con la que puede trabajar. Este comando puede usarse sólo en la etapa de saludo inmediatamente luego de iniciarse la conexión.

##### Estructura del request

CMD	N-ARGS	ARGLEN-1	ARG-1
0x00	1	2	número de versión

Dónde:

- CMD = 0x00
- N-ARGS = 1
- ARGLEN-1 = 2 (2 bytes para identificar a la versión)
- ARG-1 = *número de versión*

##### Estructura del response

Si el servidor soporta la versión:

STATUS-CODE	N-RESPTS
0x00	0

Si el servidor no soporta la versión:

STATUS-CODE	N-RESPTS
0x06	0

Si el servidor no soporta la versión, le avisa al cliente y el programa termina.

Si el servidor soporta la versión, se procede a la etapa de autenticación en donde el cliente debe proveer un usuario y contraseña válidos. Esto lo hace con el comando auth explicado a continuación:

**1.1.4.1 AUTH (0x01)**

Mediante el comando 0x01 identificado como AUTH, el usuario se autentica para luego poder realizar cambios en el servidor u obtener métricas de él. Este comando puede usarse sólo en la etapa de autenticación que ocurre luego de atravesar satisfactoriamente la etapa de SALUDO en la que se negocia la versión.

Estructura del request

CMD	N-ARGS	ARGLEN-1	ARG-1	ARGLEN-2	ARG 2
0x01	2	11	'myusername\0'	7	'mypass\0'

Dónde:

- CMD = 0x01
- N-ARGS = 2
- ARGLEN-1 = longitud en bytes del ARG-1
- ARG-1 = representación ASCII del nombre de usuario terminado con un carácter nulo ('\0' o ASCII 0)
- ARGLEN-2 = longitud en bytes del ARG-2
- ARG-2 = representación ASCII de la contraseña del usuario terminado con un carácter nulo ('\0' o ASCII 0)

Estructura del response

Si las credenciales son correctas:

STATUS-CODE	N-RESPS
0x00	0

Si las credenciales son incorrectas:

STATUS-CODE	N-RESPS
0x04	0

En este caso, se le informa al cliente del error y el programa termina.

Si las credenciales son válidas, se procede al estado de intercambio en donde el cliente puede utilizar los comandos get y set que están explicados en las secciones 1.1.4.3 y 1.1.4.4.

**1.1.4.2 CLOSE (0x02)**

Mediante el comando 0x02 identificado como CLOSE, el cliente termina la conexión con el servidor. Este comando puede usarse sólo en la etapa de intercambio, luego de haber atravesado la etapa de saludo y de autenticación exitosamente.

Estructura del request

CMD	N-ARGS
0x02	0

Dónde:

- CMD = 0x02
- N-ARGS = 0

Estructura del response

Sin Error:

STATUS-CODE	N-RESPS
0x00	0

**1.1.4.3 GET (0x03)**

Mediante el comando 0x03 identificado como GET, el cliente solicita recursos. Este comando puede usarse sólo en la etapa de intercambio, luego de haber atravesado la etapa de saludo y de autenticación exitosamente.

Estructura del request

CMD	N-ARGS	ARGLEN-1	ARG-1	ARGLEN-2	ARG-2	...	ARGLEN N	ARG-N
0x03	N	1	número identifi- catorio del recurso que se quiere obtener	arglen-2	arg-2	...	arglen-n	arg-n

Dónde:

- CMD = 0x03
- N-ARGS = cantidad de argumentos
- ARGLEN-1 = 1
- ARG-1 = posibles recursos
  - Configuraciones 0x00
  - Métricas 0x01
- ARGLEN-*i* = longitud en bytes del argumento *i*  
(0 ≤ ARGLEN-*i* ≤ 255, 2 ≤ *i* ≤ N)
- ARG-*i* = argumento *i* (2 ≤ *i* ≤ N)



**1.1.4.3.1 GET CONFIGURATIONS (0x03, 0x00)**

Para poder obtener las configuraciones del proxy, se debe pasar como primer argumento del comando get al número 0x00, y como segundo argumento al número identificador de la configuración solicitada.

Estructura del request:

CMD	N-ARGS	ARGLEN 1	ARG 1	ARGLEN 2	ARG 2
0x03	2	1	0x00	1	número identificatori o de las configurations solicitadas

Dónde

- COMMAND = 0x03
- N-ARGS = 2
- ARGLEN 1 = 1
- ARG 1 = 0x00
- ARGLEN 2 = 1
- ARG 2 = posibles configuraciones
  - Transformation program 0x00
  - Transformation program status 0x01
  - Media types 0x02

Estructura del response

Sin error

STATUS-CODE	N-RESP	DATALEN-1	DATA-1	...	DATALEN-N	DATA-N
0x00	N	datalen-1	data-1	...	datalen-2	data-n

Dónde:

- N-RESP = cantidad de fragmentos de datos de respuesta ( $0 \leq N-RESPS \leq 255$ ).
- DATALEN- $i$  = longitud en bytes del fragmento de data  $i$  ( $1 \leq i \leq N$ ).
- DATA- $i$  = fragmento de data  $i$  ( $1 \leq i \leq N$ ).

Con error (por ejemplo si se piden configuraciones inexistentes)

STATUS-CODE	N-RESP
0x01	0

**1.1.4.3.1.1 GET TRANSFORMATION PROGRAM (0x03, 0x00, 0x00)**Estructura del request

CMD	N-ARGS	ARGLEN 1	ARG 1	ARGLEN 2	ARG 2
0x03	2	1	0x00	1	0x00

Estructura del response

Sin Error y con un programa de transformación seteado:

STATUS-CODE	N-RESP	DATALEN-1	DATA-1
0x00	1	4	'cat\0'

Sin Error y sin un programa de transformación seteado:

STATUS-CODE	N-RESP
0x00	0

Con error:

STATUS-CODE	N-RESP
0x01	0

**1.1.4.3.1.2 GET TRANSFORMATION PROGRAM STATUS (0x03, 0x00, 0x01)**Estructura del request

CMD	N-ARGS	ARGLEN 1	ARG 1	ARGLEN 2	ARG 2
0x03	2	1	0x00	1	0x01

Estructura del response

Sin Error y con el programa de transformación inactivo:

STATUS-CODE	N-RESP	DATALEN-1	DATA-1
0x00	1	1	0x00

Sin Error y con el programa de transformación activo:

STATUS-CODE	N-RESP	DATALEN-1	DATA-1
0x00	1	1	0x01

Con error:

STATUS-CODE	N-RESP
0x01	0

Observación: si no existe un programa de transformación seteado, el estado del programa de transformación se encuentra en inactivo. Una vez que se setea un programa de transformación, su estado pasa a ser activo por defecto. Si se quiere desactivar, se deberá de hacer manualmente.

#### 1.1.4.3.1.3 GET MEDIA TYPES (0x03, 0x00, 0x02)

Estructura del request

CMD	N-ARGS	ARGLEN 1	ARG 1	ARGLEN 2	ARG 2
0x03	2	1	0x00	1	0x02

Estructura del response

Sin error:

STATUS-CODE	N-RESP	DATALEN-1	DATA-1
0x00	1	<i>datalen de data-1</i>	<i>'media type -1, media type -2, ..., media type -k\0'</i>

'Dónde:

- N-RESP = 1
- DATALEN-1 = longitud en bytes del nombre del string que contiene a los media types separados por comas.
- DATA-1 = nombres de los media  $k$  types separados por comas.

Con error:

STATUS-CODE	N-RESP
0x01	0

**1.1.4.3.2 GET METRICS (0x03, 0x01)**

Para poder obtener las métricas del proxy, se debe pasar como primer argumento del comando get al número 0x01, y como segundo argumento al número identificador de la métrica solicitada.

Estructura del request

CMD	N-ARGS	ARGLEN-1	ARG-1	ARGLEN-2	ARG-2
0x03	2	1	0x01	1	número identificador de las métricas solicitadas

Dónde:

- COMMAND = 0x03
- N-ARGS = 2
- ARGLEN 1 = 1
- ARG 1 = 0x01
- ARGLEN 2 = 1
- ARG 2 = posibles métricas
  - Conexiones concurrentes = 0x00
  - Cantidad de accesos históricos = 0x01
  - Bytes transferidos = 0x02

Estructura del response

Sin error

STATUS-CODE	N-RESP	DATALEN-1		...	DATALEN-N	DATA-N
0x00	N	datalen-1		...	datalen-n	data-n

Dónde:

- N-RESP = cantidad de fragmentos de datos de respuesta ( $0 \leq N\text{-RESPS} \leq 255$ ).
- DATALEN- $i$  = longitud en bytes del fragmento de data  $i$  ( $1 \leq i \leq N$ ).
- DATA- $i$  = fragmento de data  $i$  ( $1 \leq i \leq N$ ).

Con error (por ejemplo si se piden métricas inexistentes)

STATUS-CODE	DATALEN
0x01	0

**1.1.4.3.2.1 GET CONEXIONES CONCURRENTES (0x03, 0x01, 0x00), GET ACCESOS HISTORICOS (0x03, 0x01, 0x01), GET BYTES TRANSFERIDOS (0x03, 0x01, 0x02)**Estructura del request

CMD	N-ARGS	ARGLEN-1	ARG-1	ARGLEN-2	ARG-2
0x03	2	1	0x01	1	0x00 o 0x01 o 0x02

Estructura del response

Sin Error y si la cantidad de conexiones concurrentes o accesos históricos o bytes transferidos eran, por ejemplo, 11:

STATUS-CODE	N-RESP	DATALEN-1	DATA-1
0x00	1	1	11

Dónde:

- DATALEN-1: especifica la longitud del número que está presente en el campo DATA
- DATA-1: cantidad de conexiones concurrentes/cantidad de accesos históricos/ cantidad de bytes transferidos

Con error:

STATUS-CODE	N-RESP
0x01	0

**1.1.4.4 SET (0x04)**

Mediante el comando 0x04 identificado como SET, el cliente puede setear recursos. Este comando puede usarse sólo en la etapa de intercambio, luego de haber atravesado la etapa de saludo y de autenticación exitosamente.

Estructura del request

CMD	N-ARGS	ARGLEN-1	ARG-1	ARGLEN-2	ARG-2	...	ARGLEN- N	ARG-N
0x04	N	1	número identific atorio del tipo de recurso que se quiere setear	arglen-2	arg-2	...	arglen-n	arg-n

Dónde:

- COMMAND = 0x04
- N-ARGS = cantidad de argumentos
- ARGLEN-1 = 1
- ARG-1
  - Configurations 0x00
- ARGLEN-*i* = longitud en bytes del argumento *i*  
(0 ≤ ARGLEN-*i* ≤ 255, 2 ≤ *i* ≤ N).
- ARG-*i* = argumento *i* (2 ≤ *i* ≤ N).

**1.1.4.4.1 SET CONFIGURATIONS (0x04, 0x00)**

Para poder setear las configuraciones del proxy, se debe pasar como primer argumento del comando set al número 0x00, y como segundo argumento al número identificador de la configuración a setear.

Estructura del request:

CMD	N-ARGS	ARGLE N-1	ARG- 1	ARGLEN -2	ARG 2	...	ARGLEN - N	ARG-N
0x04	N	1	0x00	1	número identificat orio de la configuraci ón a setear	...	arglen -n	arg-n

Dónde:

- COMMAND = 0x04
- N-ARGS = cantidad de argumentos

- ARGLEN-1 = 1
- ARG-1 = 0x00
- ARGLEN-2 = 1
- ARG-2
  - Transformation program 0x00
  - Transformation program status 0x01
  - Media types 0x02
- ARGLEN- $i$  = longitud del argumento  $i$  ( $0 \leq \text{ARGLEN-}i \leq 255$ ,  
 $3 \leq i \leq N$ ).
- ARG- $i$  = argumento  $i$  ( $3 \leq i \leq N$ ).

#### 1.1.4.4.1.1 SET TRANSFORMATION PROGRAM (0x04, 0x00, 0x00)

Con este comando, el usuario seteará en el servidor el programa de transformación a utilizar. El comando de transformación seteado quedará en estado activo por defecto. Si se desea cambiar el estado, usar el comando set transformation program status.

##### Estructura del request

CMD	N-ARG S	ARGLEN -1	ARG-1	ARGLEN-2	ARG-2	ARGLEN-3	ARG-3
0x04	2	1	0x00	1	0x00	4	'cat\0'

##### Estructura del response

Sin Error:

STATUS-CODE	N-RESP
0x00	0

Con error general:

STATUS-CODE	N-RESP
0x01	0

**1.1.4.4.1.2 SET TRANSFORMATION PROGRAM STATUS (0x04, 0x00, 0x01)**

Con este comando el cliente podrá activar y desactivar el programa transformador.

Estructura del request

CMD	N-ARG S	ARGLEN -1	ARG-1	ARGLEN-2	ARG-2	ARGLEN-3	ARG-3
0x04	2	1	0x00	1	0x01	1	0x00 o 0x01

Dónde:

- ARG-3 = estado del programa
  - desactivado 0x00
  - activado 0x01

Estructura del response:

Sin Error:

STATUS-CODE	N-RESP
0x00	0

Con error general:

STATUS-CODE	N-RESP
0x01	0



**1.1.4.4.1.3 SET MEDIA TYPES (0x04, 0x00, 0x02)**Estructura del request

CMD	N-ARGS	ARGL EN-1	ARG-1	ARGLE N-2	ARG-2	ARGLEN-3	ARG-3
0x04	3	1	0x00	1	0x02	arglen del arg 3	'mediaType1, mediaType2..., mediaTypek\0''

Dónde:

- N-ARGS = 3
- En los campos arg3 van, en su representación ASCII, los k media types separados por comas.

Estructura del response

Sin Error:

STATUS-CODE	N-RESP
0x00	0

Con error general:

STATUS-CODE	N-RESP
0x01	0

**1.2 Comentario sobre el protocolo**

Se decidió utilizar las estructuras detalladas en los puntos 1.1.2.1 y 1.1.2.1 por su naturaleza genérica. Como se puede ver, la estructura del request permite que haya hasta 255 comandos diferentes con hasta 255 argumentos de longitud de hasta 255 bytes cada uno. Esto permite un abanico de posibilidades a la hora de extender el protocolo. Lo mismo ocurre para las respuestas.

### 1.3 Servidor HPCP

Se implementó un servidor capaz de atender las peticiones de una aplicación cliente y devolverle una respuesta acorde utilizando, para la transmisión de datos, el protocolo HPCP a nivel aplicación. Este servidor es el que hace posible que el cliente pueda tanto modificar configuraciones del proxy como obtener métricas de él. El servidor está basado en códigos de Juan F. Codagnone, profesor a cargo de la parte práctica de la materia Protocolos de Comunicación de la carrera Ingeniería en Informática en el Itba.

El servidor HPCP es un servidor concurrente escrito en el lenguaje de programación C que utiliza sockets no bloqueantes y pthreads. Este, luego de recibir e interpretar argumentos por línea de comandos, arma un socket pasivo a la espera de conexiones entrantes para luego generar sockets activos y así establecer conexiones entre el cliente y el servidor. El socket pasivo escucha en el puerto especificado en los argumentos o en el 9090 por defecto.

### 1.4 Servidor proxy HTTP

Se implementó un servidor proxy para el protocolo HTTP versión 1.1 que hace de intermediario en las peticiones de recursos que hace una aplicación cliente a otro servidor al cual llamaremos origin server. El proxy implementa mecanismos que permiten transformar el cuerpo de ciertas respuestas HTTP utilizando aplicaciones externas a las que llamaremos programas transformadores. El servidor está basado en códigos de Juan F. Codagnone, profesor a cargo de la parte práctica de la materia Protocolos de Comunicación de la carrera Ingeniería en Informática en el Itba.

El servidor proxy HTTP es un servidor concurrente escrito en el lenguaje de programación C que utiliza sockets no bloqueantes y pthreads. Este, luego de recibir e interpretar argumentos por línea de comandos, arma un socket pasivo a la espera de conexiones entrantes para luego generar sockets activos y así establecer conexiones entre los clientes y él. El socket pasivo escucha en el puerto especificado en los argumentos o en el 8080 por defecto.

Cuando al proxy le llega un request por parte de un cliente, va guardando en un buffer con tamaño definido los datos que recibió. Si en esos datos no se encuentra un host (en un header host o en la url), el proxy se quedará a la espera de nuevos requests y seguirá llenando el buffer hasta que lo encuentre. Si el buffer se llena y no encontró un host, se le retorna un error de tipo 400 y se cierra la conexión. Si por el

contrario, el proxy encuentra un host, intentará establecer una conexión con el origin server utilizando al puerto 80 por defecto o el especificado por el cliente.

Para conectarse con el host, primero se hace una resolución de nombres. Como la resolución de nombres es bloqueante, se realiza en un thread aparte. Se itera por todas las ips obtenidas hasta que con alguna la conexión sea satisfactoria. En caso de que ninguna conexión sea satisfactoria, se devuelve un 503. Si se puede establecer una conexión con el origin server, el proxy proseguirá a enviarle lo que tenía guardado en el buffer y lo que siga recibiendo del cliente. Se le avisa al origin server que una vez que termine de enviar la respuesta al request, el proxy cerrará la conexión. Esto se hace poniendo connection close en el header connection.

Cuando el origin server responda, el proxy va a parsear la respuesta y se va a fijar en los headers. Si se encuentra un header connection en connection keep alive, el proxy lo cambiará a connection close para avisarle así al cliente que una vez que se termina la respuesta, se cerrará la conexión. No hay conexiones persistentes. El header de la codificación se usará cuando el programa de transformación esté activado. Si la transformación está desactivada, el proxy funciona casi como un pasamanos con lo que no le interesa este header. En cambio, si la transformación está activada, el proxy se va a fijar en qué codificación se encuentra el body y, si es una soportada por este, lo decodificará y se lo pasará al programa transformador. Por ejemplo si la respuesta del origin server está chunkeada, primero nos vamos a encontrar con la cantidad de bytes que se están mandando, luego \r\n y luego todos los bytes. Entonces el proxy le enviará al programa transformador el body sin los \r\n ni la cantidad de bytes ni los header trailers y luego le enviará el resultado al cliente de forma chunkeada. Si el proxy no soporta la codificación, le enviará la respuesta al cliente sin haber pasado el body por el programa transformador.

Entonces, el proxy escucha en dos bocas (cliente y origin server) poniendo connection close para ambos lados y hace uso de un timeout para asegurarse de que las conexiones no sean persistentes: si por 90 segundos tanto el cliente como el origin server no hacen ningún read ni write, automáticamente el proxy cierra la conexión con ambos lados.

Se logean los accesos al servidor, logs de debug propias de debug, se logean los errores y warnings.

### 1.5 Cliente HPCP

El propósito de este programa es proveer de una plataforma donde un usuario pueda interactuar con el manager del servidor proxy HTTP y, dada las correctas credenciales, hacer cambios en la configuración del proxy o pedirle información sobre la configuración o métricas.

En primer lugar, el usuario debe ingresar en la aplicación cliente la dirección del manager del servidor y el puerto donde está escuchando el manager. Habiendo creado y establecido la comunicación con el administrador, este último debe negociar la versión del protocolo a utilizar, que fue previamente ingresada por el usuario al iniciar la aplicación, con el comando HELLO.

Luego, el comando AUTH es ejecutado, donde se le va a pedir al usuario sus credenciales de administrador. Si son incorrectas, el programa termina con el mensaje de error apropiado. Si son correctas, el usuario podrá elegir desde el siguiente menú (figura 1.4.1) el próximo comando a ejecutar:

```
Select from the following commands:
1 --> Get configurations
2 --> Get metrics
3 --> Set configurations
4 --> Help
5 --> Quit
```

*Figura 1.4.1: comandos disponibles*

El cliente podrá entonces pedir las configuraciones disponibles, cambiarlas, o pedir las métricas disponibles. Las configuraciones disponibles para el cliente son las siguientes: programa transformador, estado (activo/inactivo) del programa transformador, y los tipos de media aceptados en las respuestas del proxy.

De esta manera, si el usuario quiere pedirle al administrador alguna configuración, ingresando el número 1 contará con el siguiente menú (figura 1.4.2) para elegirla:

```
Select from the following configurations:
1 --> Transformation program
2 --> Transformation program status
3 --> Media types
```

*Figura 1.4.2: configuraciones disponibles*

Al mismo tiempo, si el usuario quiere modificar aquellas configuraciones, ingresando el número 3 contará con el mismo menú de opciones (figura 1.4.2) para luego ingresar la respectiva configuración. Si ocurre algún error por el contenido ingresado por el cliente como configuración, se

informará del error y el menú principal será mostrado nuevamente (figura 1.4.1).

Además, el usuario cuenta con la posibilidad de pedir las métricas del proxy. Estas métricas incluyen: conexiones concurrentes, accesos históricos y bytes transferidos. Por lo tanto, ingresando el número 2 luego del menú principal (figura 1.4.1), se desplegará el próximo menú:

*Figura 1.4.3: métricas disponibles*

```
Select from the following metrics:
1 --> Concurrent connections
2 --> Historical accesses
3 --> Transferred bytes
```

Cuando el cliente quiera cerrar la aplicación, con el comando CLOSE, ingresando el número 5, se le informará al administrador que el cliente quiere cerrar la comunicación. Si el administrador le indica en la respuesta (con el número 0) que puede continuar, la conexión se cierra, se liberan los recursos empleados y la aplicación termina.

## **2. Problemas encontrados durante el diseño y la implementación**

### **2.1 Request - response entre el cliente y el origin server**

Uno de los puntos de discusión durante el diseño del proxy HTTP fue si era necesario o no leer un request del cliente para enviar el mismo request al origin server, análogamente con el response. Ya que es muy complejo parsear de forma entera cada request/response, se optó por dejar las bocas del cliente y el origin abiertas. De esta manera no seguimos con el patrón leer un request - mandar un request, y recibir un response - mandar un response, resultando en un diseño más simple y fácil de implementar.

### **2.2 Qué hacer con el header accept de una request que pasa por el proxy**

Si el servidor proxy pisa el header accept de la request que le envió el cliente y pone únicamente los que él acepta, nos aseguramos de que siempre que haya un programa transformador válido, el proxy podrá procesar la respuesta del origin server y enviarla al programa transformador. Sin embargo, decidimos no hacer esto por un tema de simpleza para no estar parseando todos los headers de la request HTTP hasta encontrar un header accept y, si está pisarlo y sino agregarlo.

### **2.3 Timeout del Proxy Server y Cliente HPCP**

Optamos por setear 90 segundos como timeout para cerrar la comunicación después de inactividad. Ahora bien, si alguien desea realizar una ataque podría enviar un paquete cada 89 segundos para mantener las conexiones abiertas y así bloquear la conexión indefinidamente.

### **2.4 Diseño del protocolo HPCP**

Al momento de elegir cómo pasar los parámetros al administrador del proxy, tuvimos que decidir entre pasar strings con el tamaño del string antes de que comience el primer caracter o escribir el string hasta un caracter centinela. Optamos por hacer una combinación de ambas. Al comienzo de un parámetro que está en formato string le indicamos la

cantidad de caracteres que contiene y lo hacemos null terminated, por lo cual un '\0' indica que el string terminó. Aunque el hecho de que el string sea null terminated es suficiente para leerlo correctamente, optamos por mantener la convención de indicar el tamaño del argumento previo al argumento, como en todos los demás casos.

Por otra parte, también tuvimos que elegir entre indicar la cantidad de argumentos previamente a listarlos en el request/response, o leer cada argumento hasta que un caracter centinela indique que el datagrama terminó. Optamos por lo primero, pero ambas soluciones hubieran sido correctas en nuestra implementación.

### **3. Limitaciones de la aplicación**

#### **3.1 Conexiones persistentes**

El proxy no cuenta con conexiones persistentes. Esto significa una pérdida de tiempo en abrir y cerrar conexiones de un mismo cliente.

#### **3.2 Encriptación**

Actualmente los paquetes de nuestra aplicación no viajan encriptados en la red. Con lo que, por ejemplo, se pueden interceptar paquetes y obtener el usuario y la contraseña del administrador.

#### **3.3 Usuarios y contraseñas hardcodeados en el código**

Esto deja expuesto a los programadores datos privados de los usuarios además de que, si se quieren agregar usuarios nuevos, se debe recompilar el servidor. Podrían haber sido variables de entorno o levantado de una base de datos.

#### **3.4 Validación de función de transformación**

El servidor HPCP no valida la función de transformación configurada por el usuario cliente. Por ende se podría correr cualquier comando que exista en el computador donde se corra el servidor.

#### **3.4 Máxima cantidad de conexiones simultáneas**

El selector del profesor Juan F. Codagnone permite un máximo de `FD_SETSIZE` file descriptors abiertos en simultáneo. Este valor es determinado por el sistema operativo del computador donde serán corridos ambos servicios de Proxy y Admin.

Por otro lado, el proxy del profesor permite un máximo de 50 conexiones simultáneas para la conexión.



## 4. Posibles extensiones

### 4.1 Cantidad de conexiones al origin server

Para futuras implementaciones, se podría aprovechar una conexión a un origin server para enviar requests de distintos clientes.

### 4.2 Loggings en archivos

Actualmente el servidor loggea en la consola. Es deseable que los loggings se hagan en archivos para que se tenga un registro de eventos del servidor y no se pierdan al reiniciar el programa.

### 4.3 Persistencia en los datos de las métricas y configuraciones

Como las métricas y las configuraciones del servidor no se guardan en un archivo externo, una vez que se reinicia el servidor, estas se pueden perder. Este comportamiento no es recomendable aunque, a efectos de este trabajo práctico, no lo pusimos como prioridad a implementar.

### 4.4 Mejora en las métricas

Se podrían implementar más métricas además de dar más detalles sobre ellas.

### 4.5 Implementación de un mecanismo de encriptación

De esta forma, se preservará la seguridad de los datos de los clientes sin revelar usuarios ni contraseñas.

### 4.6 Reducir cantidad de mensajes enviados

El protocolo dicta que para cada GET o SET de métricas o configuración se debe enviar un mensaje por separado. Por ejemplo, si se quisiera consultar varias métricas, se podría reducir la cantidad de mensajes enviados a uno solo simplemente haciendo un | (binary or) entre las métricas necesitadas.

Ej: Si las métricas estuvieran definidas de la siguiente forma:

- posibles métricas
  - Conexiones concurrentes = 0x01
  - Cantidad de accesos históricos = 0x02
  - Bytes transferidos = 0x04

y quisiéramos hacer un GET sobre las métricas de conexiones concurrentes y bytes transferido enviaríamos algo por el estilo:

CMD	N-ARGS	ARGLEN-1	ARG-1	ARGLEN-2	ARG-2
0x03	2	1	0x01	1	0x5

Pues  $0x5 = 00000101$  es equivalente a hacer  $0x01 \mid 0x04$ .

Al mismo tiempo, el servidor HPCP respondería en un único mensaje todos los resultados.

STATUS-CODE	N-RESP	DATALEN-1	DATA-1	DATALEN-2	DATA-2
0x00	2	1	11	2	333123

El problema es que para esto, habría que redefinir los valores asignados a cada métrica y configuración nuevamente.

## 5. Conclusiones

El presente trabajo práctico supuso un desafío complejo pero realizable en el que se debieron utilizar conocimientos de programación en C, de Sistemas Operativos y, por supuesto, de Protocolos de Comunicación. El hecho de llevar a cabo un proyecto que requiera de conocimientos teóricos y prácticos aprendidos en clase nos permitió entender y aprender de una forma más profunda.

El trabajo nos dio la oportunidad de diseñar un protocolo a nuestro propio gusto, de programar un parser para este protocolo además de uno para un protocolo ajeno al nuestro, de programar una aplicación cliente que utilice nuestro protocolo, de programar un servidor para el protocolo, de programar un servidor proxy para el protocolo HTTP y de experimentar con su potencia, de utilizar implementaciones de soluciones a problemas como lo hicimos con el servidor SocksV5, entre otras muchas cosas. Fue un trabajo realmente abarcativo que requirió, fundamentalmente, de una organización grupal con la que se pudo explotar las facilidades de cada integrante para poder así concluirlo. Se presentaron dificultades en el camino que pudieron ser superadas y, a pesar de que sabemos que el trabajo puede ser mejorado como se puede ver en la sección 4, creemos que interiorizamos una gran cantidad de contenido fascinante que nos va a permitir seguir aprendiendo y descubriendo sobre el mundo de la computación.

## 6. Ejemplos de prueba

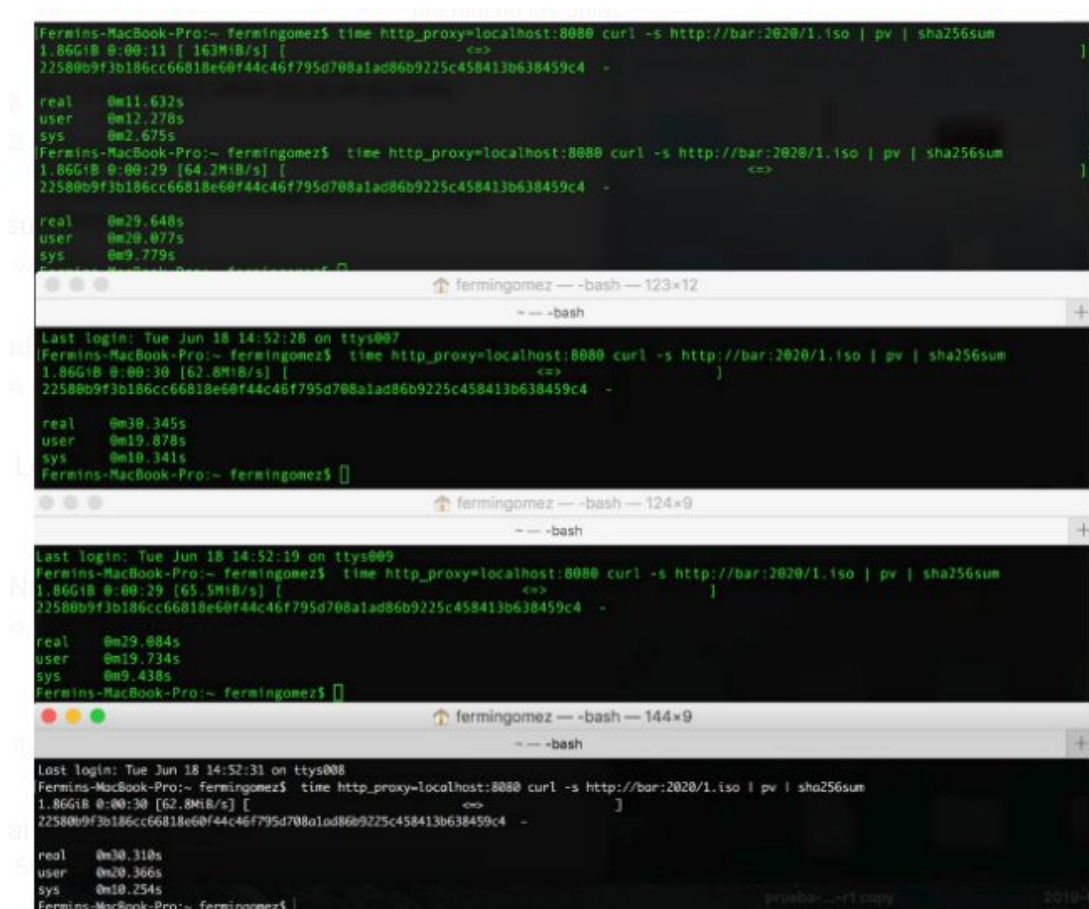
6.1 Podemos ver la inmutabilidad de los datos al pasar por el proxy con el siguiente ejemplo:

```
Fermins-MacBook-Pro:~ ferminomez$ curl -s http://bar:2020/leet.txt | sha256sum  
8a71f2e35c9683ff2b4fc59aace628b5ceae21f97e0bbb4eea576ffa8e75db5f -
```

Se seteó al servidor con los parámetros `-t "sha256sum"` `-M "text/plain"` y se ejecutó la misma línea que en la imagen anterior solo que sin redirigir la salida estándar a sha256sum.

```
Fermins-MacBook-Pro:~ ferminomez$ http_proxy=localhost:8080 curl -s http://bar:2020/leet.txt  
8a71f2e35c9683ff2b4fc59aace628b5ceae21f97e0bbb4eea576ffa8e75db5f -  
Fermins-MacBook-Pro:~ ferminomez$
```

6.2 Ejemplos de tiempos con una conexión y con más de una conexión concurrente



```
Fermins-MacBook-Pro:~ ferminomez$ time http_proxy=localhost:8080 curl -s http://bar:2020/1.iso | pv | sha256sum  
1.86GiB 0:00:11 [ 163MiB/s] [ <=> ]  
22580b9f3b186cc66818e60f44c46f795d708a1ad86b9225c458413b638459c4 -  
  
real    0m11.632s  
user    0m12.278s  
sys     0m2.675s  
Fermins-MacBook-Pro:~ ferminomez$ time http_proxy=localhost:8080 curl -s http://bar:2020/1.iso | pv | sha256sum  
1.86GiB 0:00:29 [ 64.2MiB/s] [ <=> ]  
22580b9f3b186cc66818e60f44c46f795d708a1ad86b9225c458413b638459c4 -  
  
real    0m29.648s  
user    0m29.877s  
sys     0m9.779s  
Fermins-MacBook-Pro:~ ferminomez$  
  
Last login: Tue Jun 18 14:52:28 on ttys007  
Fermins-MacBook-Pro:~ ferminomez$ time http_proxy=localhost:8080 curl -s http://bar:2020/1.iso | pv | sha256sum  
1.86GiB 0:00:30 [ 62.8MiB/s] [ <=> ]  
22580b9f3b186cc66818e60f44c46f795d708a1ad86b9225c458413b638459c4 -  
  
real    0m30.345s  
user    0m19.878s  
sys     0m10.341s  
Fermins-MacBook-Pro:~ ferminomez$  
  
Last login: Tue Jun 18 14:52:19 on ttys009  
Fermins-MacBook-Pro:~ ferminomez$ time http_proxy=localhost:8080 curl -s http://bar:2020/1.iso | pv | sha256sum  
1.86GiB 0:00:29 [ 65.5MiB/s] [ <=> ]  
22580b9f3b186cc66818e60f44c46f795d708a1ad86b9225c458413b638459c4 -  
  
real    0m29.884s  
user    0m19.734s  
sys     0m9.438s  
Fermins-MacBook-Pro:~ ferminomez$  
  
Last login: Tue Jun 18 14:52:31 on ttys008  
Fermins-MacBook-Pro:~ ferminomez$ time http_proxy=localhost:8080 curl -s http://bar:2020/1.iso | pv | sha256sum  
1.86GiB 0:00:30 [ 62.8MiB/s] [ <=> ]  
22580b9f3b186cc66818e60f44c46f795d708a1ad86b9225c458413b638459c4 -  
  
real    0m30.310s  
user    0m20.366s  
sys     0m10.254s  
Fermins-MacBook-Pro:~ ferminomez$
```

## 7. Guía de compilación y ejecución

Para poder compilar el proyecto, dentro de la carpeta TpProtos/build se deben correr los siguientes comandos:

```
$ make clean  
$ cmake ..  
$ make
```

### 7.1 Ejecución del cliente HPCP

Para poder ejecutar al cliente, se debe correr dentro de la carpeta TpProtos/build:

```
$ ./Client
```

seguido de los siguientes argumentos:

**-L management-address**

Sets the address where the management service is serving. By default it listens in loopback.

**-o management-port**

STCP port where the management server is located. By default port is 9090.

**-v protocol-version**

Protocol version of the configuration administrator.

Las credenciales correctas para usar en el cliente son:

username: admin

password: admin

### 7.2 Ejecución del servidor

Para poder ejecutar el servidor, se debe correr dentro de la carpeta TpProtos/build:

```
$ ./Server
```

seguido de los siguientes argumentos:

**-e file-error**

Specifies the file where stderr is redirected from the executions of the filters. By default the file is /dev/null.

**-h**

Shows options available and terminates.

**-l HTTP-address**

Sets the address where the HTTP proxy is serving. By default it listens on all interfaces.

**-L management-address**

Sets the address where the management service is serving. By default it listens in loopback.

**-M transformable-media-types**

List of transformable media types. The syntax of the list follows the rules of the HTTP Accept header (section 5.3.2 of RFC7231). By default the list is empty.

**-o management-port**

STCP port where the management server is located. By default port is 9090.

**-p local-port**

TCP port listening for incoming HTTP connections. By default port is 8080.

**-t cmd**

Command used for external transformations. Compatible with system(3).

**-v**

Shows information about the version and terminates.

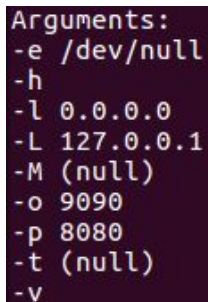
## **8. Instrucciones para la configuración**

Si se desea configurar el proxy en tiempo de ejecución, se puede hacer uso de la aplicación cliente especificada en la sección 1.4.

## 9. Ejemplos de configuración y monitoreo

En una terminal se corre al programa servidor:

```
$ ./Server
```

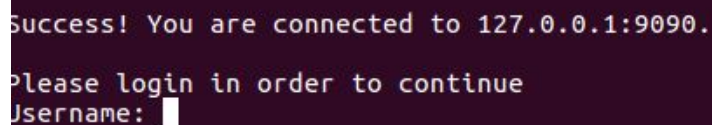
A terminal window with a dark purple background. The text is white. It shows the command './Server' and its output, which is a list of arguments for the server program.

```
Arguments:  
-e /dev/null  
-h  
-l 0.0.0.0  
-L 127.0.0.1  
-M (null)  
-o 9090  
-p 8080  
-t (null)  
-v
```

*Figura 9.1: argumentos del server*

En otra terminal se corre al programa cliente:

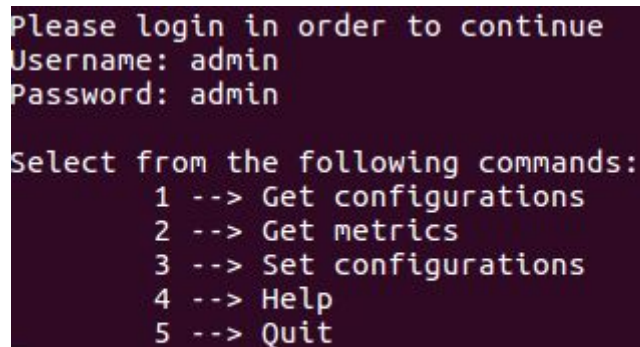
```
$ ./Client
```

A terminal window with a dark purple background. The text is white. It shows the output of the './Client' command, indicating a successful connection to the server at 127.0.0.1:9090 and prompting for a login.

```
Success! You are connected to 127.0.0.1:9090.  
  
Please login in order to continue  
Username: █
```

*Figura 9.2: resultado sin pasar parámetros al cliente*

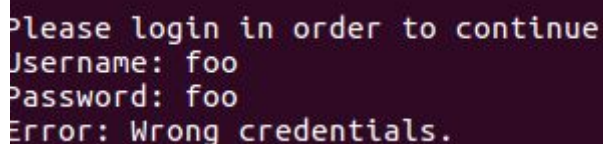
### 9.1 Autenticación exitosa

A terminal window with a dark purple background. The text is white. It shows the login process for the 'admin' user, followed by a list of available commands.

```
Please login in order to continue  
Username: admin  
Password: admin  
  
Select from the following commands:  
1 --> Get configurations  
2 --> Get metrics  
3 --> Set configurations  
4 --> Help  
5 --> Quit  
█
```

*Figura 9.1.1: resultado de una autenticación exitosa*

### 9.2 Autenticación con error

A terminal window with a dark purple background. The text is white. It shows the login attempt for the 'foo' user, which fails due to incorrect credentials.

```
Please login in order to continue  
Username: foo  
Password: foo  
Error: Wrong credentials.
```



Figura 9.2.1: resultado de una autenticación fallida

### 9.3 Get transformation program status

```
Select from the following commands:
  1 --> Get configurations
  2 --> Get metrics
  3 --> Set configurations
  4 --> Help
  5 --> Quit
1
Select from the following configurations:
1 --> Transformation program
2 --> Transformation program status
3 --> Media types
2
Transformation program: inactive
Select from the following commands:
  1 --> Get configurations
  2 --> Get metrics
  3 --> Set configurations
  4 --> Help
  5 --> Quit
█
```

Figura 9.3.1: resultado del get transformation program status

### 9.4 Set transformation program status

```

Select from the following commands:
  1 --> Get configurations
  2 --> Get metrics
  3 --> Set configurations
  4 --> Help
  5 --> Quit
3

Select from the following configurations:
  1 --> Transformation program
  2 --> Transformation program status
  3 --> Media types
2
Please enter a status (0/1): 1

Success! Transformation program status is now 1

Select from the following commands:
  1 --> Get configurations
  2 --> Get metrics
  3 --> Set configurations
  4 --> Help
  5 --> Quit
1

Select from the following configurations:
1 --> Transformation program
2 --> Transformation program status
3 --> Media types
2
Transformation program: active

Select from the following commands:
  1 --> Get configurations
  2 --> Get metrics
  3 --> Set configurations
  4 --> Help
  5 --> Quit

```

Figura 9.4.1: resultado de la modificación del transformation program status

### 9.5 Get concurrent connections

Sin ningún cliente conectado al puerto en el que está escuchando el proxy (8080 en este ejemplo):

```

Select from the following commands:
  1 --> Get configurations
  2 --> Get metrics
  3 --> Set configurations
  4 --> Help
  5 --> Quit
2

Select from the following metrics:
  1 --> Concurrent connections
  2 --> Historical accesses
  3 --> Transferred bytes
1
Concurrent connections: 0

```

*Figura 9.5.1: resultado de pedir las conexiones concurrentes*

Se ejecutó en otra terminal

\$ nc localhost 8080

y se obtuvo

```
Select from the following commands:
  1 --> Get configurations
  2 --> Get metrics
  3 --> Set configurations
  4 --> Help
  5 --> Quit
2

Select from the following metrics:
  1 --> Concurrent connections
  2 --> Historical accesses
  3 --> Transferred bytes
1
Concurrent connections: 1

Select from the following commands:
  1 --> Get configurations
  2 --> Get metrics
  3 --> Set configurations
  4 --> Help
  5 --> Quit
```

*Figura 9.5.2: nuevo resultado de la cantidad de conexiones concurrentes*

## 9.6 Logs del servidor

```

DEBUG - [Tue Jun 18 03:14:45 2019] Cliente admin conectado
DEBUG - [Tue Jun 18 03:16:24 2019] Cliente admin conectado
WARN - [Tue Jun 18 03:17:54 2019] Connection in fd: 4 has been closed for inactivity
DEBUG - [Tue Jun 18 03:19:53 2019] Cliente admin conectado
DEBUG - [Tue Jun 18 03:20:19 2019] Cliente admin conectado
DEBUG - [Tue Jun 18 03:21:55 2019] Cliente admin conectado
DEBUG - [Tue Jun 18 03:23:11 2019] Cliente admin conectado
DEBUG - [Tue Jun 18 03:23:14 2019] Cliente admin conectado
WARN - [Tue Jun 18 03:25:16 2019] Connection in fd: 4 has been closed for inactivity
DEBUG - [Tue Jun 18 03:25:23 2019] Cliente admin conectado
DEBUG - [Tue Jun 18 03:25:27 2019] Cliente admin conectado
WARN - [Tue Jun 18 03:27:10 2019] Connection in fd: 4 has been closed for inactivity
DEBUG - [Tue Jun 18 03:27:28 2019] Cliente conectado y registrado

ACCES - [Tue Jun 18 03:27:40 2019] 127.0.0.1:50618 - "" - ""
DEBUG - [Tue Jun 18 03:27:40 2019] CONEXION CERRADA
DEBUG - [Tue Jun 18 03:27:45 2019] Cliente admin conectado
DEBUG - [Tue Jun 18 03:28:04 2019] Cliente conectado y registrado

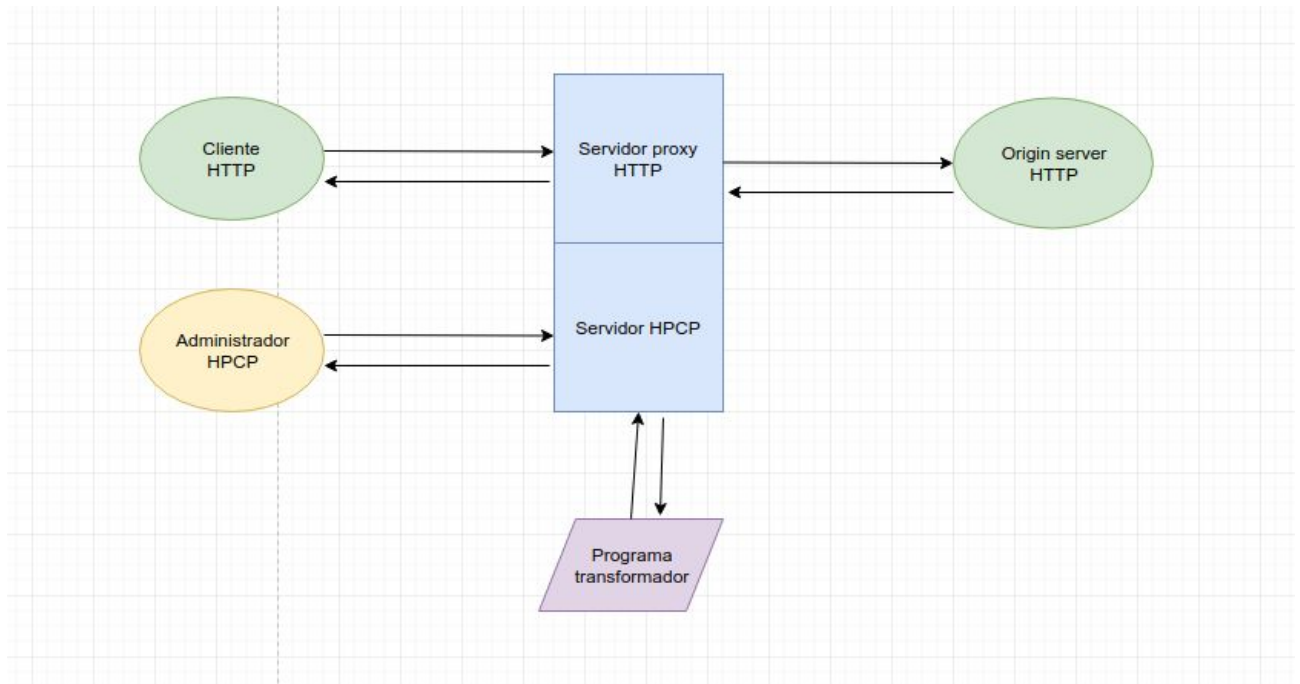
WARN - [Tue Jun 18 03:29:37 2019] Connection in fd: 4 has been closed for inactivity
WARN - [Tue Jun 18 03:29:37 2019] Connection in fd: 5 has been closed for inactivity
ACCES - [Tue Jun 18 03:29:37 2019] 127.0.0.1:50622 - "" - ""
DEBUG - [Tue Jun 18 03:29:37 2019] CONEXION CERRADA

```

*Figura 9.6.1: logs del servidor*

Se pueden ver distintos tipos de logs como cuando se conecta un cliente, cuando se cierra una conexión por inactividad, cuando el cierra una conexión, etc.

## 10. Documento del diseño del proyecto



*Figura 10.1: Diseño general*