72.07 Protocolos de comunicación
1er Cuatrimestre 2018
Trabajo Práctico Especial

Grupo 5

Cifuentes, Ignacio Imanol	54311
Manganaro Bello, Santiago	56289
Soracco, Tomas	56002
Vazquez, Agustin Ignacio	55354

1. Índice

1. Introducción

El objetivo del trabajo es implementar un servidor proxy que soporte el protocolo HTTP version 1.1 y a su vez pueda ser utilizado por distintos User-Agents para navegar por Internet.
El servidor debe poder utilizarse de manera transparente y con previa configuración de los navegadores. Es necesario que se soporte la concurrencia de usuarios en simultáneo.
Adicionalmente, el servidor implementa funciones de transformación, soporte de métricas y datos de uso. Estos datos se pueden obtener por consola en tiempo de ejecución, para los cuales se diseñó un protocolo "admin" que funciona sobre SCTP.

2. Descripción de los protocolos y aplicaciones

2.1. Aplicación servidor proxy

Se desarrolló un servidor proxy para las versiones 1.0 y 1.1 del protocolo HTTP.

2.1.1. Funcionamiento del servidor

El flujo es el siguiente : se recibe un request por parte del cliente En caso de estar mal formado (ej: método no soportado, no tener header Host, request mal formado, etc) se envía un mensaje de error al cliente y se cierra la conexión. De lo contrario, el mensaje de request es parseado para poder establecer los parámetros de conexión y procesamiento interno de nuestro servidor. Una vez realizado esto, se resuelve el dominio del origin server y se intenta establecer la conexión con el mismo. En caso de no poder realizar la conexión, se envía un mensaje de error al cliente. Una vez conectados, se envía al origen el mensaje con sus headers correspondientes y, de ser necesario, el body.
Posteriormente, el origin server dará un mensaje de response, el cual se parsea en el servidor. En caso de tener errores de malformación de response (ej: no incluir código de respuesta, malformación de response, versión no soportada de HTTP) se cierra la conexión. Cuando el cliente recibe la respuesta correctamente formada, la conexión se cierra.

2.1.2. Transformaciones

Para las transformaciones, tenemos una estructura global de la aplicación que almacena los parámetros compartidos por las varias partes de la misma. Desde el admin se pueden ajustar las configuraciones según se desee y esto se aplicará a los mensajes del proxy.
Al realizar un cambio en la configuración, el admin va a realizar un display de las configuraciones vigentes, para ofrecer mayor legibilidad.

2.1.3 Parsers

Los parsers son implementados con una máquina de estados tradicional. Leen byte a byte desde un buffer dado, cambiando su estado de manera acorde. Estos fueron implementados de manera tal que la cantidad de bytes que deban procesar en cada pasada no influyera en su funcionamiento. Esto permite que las request y responses fueran procesadas a medida que se lee sin necesidad de tenerlas guardadas enteras.
Sin embargo, optamos por guardar los headers en memoria. Estos se imprimen en conjunto debido a que normalmente el tamaño de los mismos suele ser acotado. Se realizó de esta manera por la sencillez a la hora de implementarlos.

A la hora de recibir el request, decidimos parsear los siguientes headers:

Header ABC-Proxy : Este header es agregado por el servidor proxy y es interpretado para evitar conexiones recursivas a nuestro proxy.

Host: Este header es obligatorio de parsear, pues, según el RFC, un request puede tener en su primer línea o no la uri. En caso de que no esté, este header es el encargado de proporcionar el host (puerto default 80).

A la hora de recibir el response, decidimos parsear los siguientes headers:

Content-Type : Este header se parsea para poder realizar la comparación con la lista establecida por el administrador en la configuración. Se interpretan por default un media-type junto a tres parámetros.

Content-Encoding : Este header es utilizado para poder determinar si el contenido recibido está comprimido o no (El default -identity- indica que no). Considerando que son cuatro las posibles opciones según el rfc (gzip, compressed, deflate, identity) y las primeras tres de ellas son compresiones de algún tipo, decidimos lo siguiente: en caso de que se reciba cualquier otra opción que no sea identity, ya sea válida o no, no se realiza la transformación.

Transfer-Encoding : Este header es utilizado a la hora de transformación. En caso de que su valor sea chunked, la manera de manejar la transformación es diferente, pues se recibe en el cuerpo de la respuesta, valores que refieren a la forma de transferir el paquete y no al contenido del mismo. Como servidor proxy, interpretamos esto y a la hora de enviarselo al cliente, se vuelve a chunkear el contenido dependiendo como se vaya leyendo.

*Content-Length : Este header se parsea tanto en request como en response, sin embargo, por decisiones de implementación, su uso es casi nulo.

2.1.4 Decisiones de implementación

- Al utilizarse el proxy como origin server, este lo detecta y devuelve un error de la familia de los 500. Esto se lleva a cabo marcando los mensajes que pasan por el proxy con un header. Tomamos esta decisión por que creemos que está es la manera más práctica de manejar la situación.
- Para cualquier error de respuesta que no sea una malformacion, el proxy actuara de manera transparente, redirigiendo el mensaje de manera inalterada al cliente.

2.1.5 Logging

Nuestra implementación de logging escribe a un archivo.

LOG_DEBUG: Utilizado para debuggear la aplicación en el desarrollo. Flag para activar y desactivar.

LOG_ERROR: Utilizado para llevar una lista para control de los errores ocurridos en el servidor.

LOG_INFO: Utilizado para llevar una lista para control de los usuarios que se conectan. Se guardan tanto la ip del usuario como la uri o el host al que se conectan.

LOG_PRIORITY: Utilizado para registrar mensajes importantes. Ejemplos: Comienzo de la ejecución, comienzo de los tests, sigterm, exit, etc.

LOG_RECOVER: Utilizado para recuperar los últimos logs.

NOTA: Dado que la implementación es bloqueante, decidimos remover está funcionalidad.

2.1.6 Manejo de errores

Los errores que se manejan son los siguientes:

400 - Bad Request : Ocurre cuando el usuario hace un pedido inválido.

500 - Internal Proxy Error : Ocurre cuando el proxy tiene un error interno.

503 - Service Unavailable : Ocurre cuando hay un error en el servicio de conexión provisto por el proxy cuya responsabilidad es del servidor. Ejemplo: el servidor no existe.

405 - Method not supported : Ocurre cuando el usuario intenta hacer un pedido utilizando un método no soportado por el proxy.

2.1.7 Multithreading

Con el fin de aprovechar el procesamiento en aquellas computadoras con más de un núcleo, se hace uso de threads a la hora de establecer una coneccion entre el cliente y un servidor, aumentando la performance.

2.2. Cliente

Consiste en un programa simple que se conecta a la ip y puerto enviando un comando de protocolo a ejecutar. Toda esta información se recibe por parámetro. El objetivo del mismo es meramente demostrativo, pues no se realizan validaciones complejas. En caso de que algo falle, se envía un mensaje de error.

En lo que refiere a las características de implementación del mismo, cabe destacar que solo acepta puertos y no dominios. Por otro lado, solo acepta ip, no resuelve dns.

2.3. Protocolo

El protocolo de admin diseñado funciona sobre SCTP y es el que nos permite setear la configuración del servidor proxy. El mismo es orientado a texto.

2.2.1. Comandos

El administrador se conecta al servidor y envía una versión identificador, un código de acceso secreto, el método a usar y un campo de datos.

```
                  +----+-----------------+----------+--------------+
*                 |VER | SECRET_PASSCODE | METHOD   |    DATA      |
*                 +----+-----------------+----------+--------------+
*                 | 1  |     string      | string   |   string     |
*                 +----+-----------------+----------+--------------+
```

El campo VER se establece en X'01 para esta versión del protocolo.
El campo SECRET_PASSCODE contiene una cadena utilizada para autenticar con

el código de acceso del administrador (codificado en código).
El campo de método contiene una cadena utilizada para cambiar entre todos los comando.
El cliente debe enviar una cadena en el campo de datos para algunos de los comandos soportados (como buffer_change_size ())

2.2.1.1 Comando METRICS
Argumentos : -
Descripción: Devuelve estadísticas sobre el proxy. Se informa al admin la cantidad de usuarios conectados en el momento, la cantidad total de usuarios desde su inicio y la cantidad de bytes transferidos

2.2.1.2 Comando LOGS
Argumentos : -
Descripción : Devuelve los últimos logs hechos por el proxy (1000 bytes).

2.2.1.3 Comando ENABLE_TRANSFORMER
Argumentos : -
Descripción: Permite habilitar la transformación del proxy.

2.2.1.4 Comando DISABLE_TRANSFORMER
Argumentos : -
Descripción: Permite habilitar la transformación del proxy.

2.2.1.5 Comando COMMAND_TRANSFORMER
Argumentos : String que represente el llamado a una función de transformación. Ejemplo: sed -u -e ‘s/o/0/g’
Descripción: Permite setear la función con la cual se va a transformar el contenido.

2.2.1.6 Comando TYPE_TRANSFORMER
Argumentos : String. Cada media type se separa con ‘,’ y sus argumentos con ‘;’. Se permite el carácter ‘*’ para representar todo. Ejemplo: text/plain;charset=UTF-8,img/*,text/html;charset=*
Descripción: Permite definir la lista de media types que se van a transformar.

2.2.1.7 Comando BUFFER_TRANSFORMER
Argumentos : Número entero positivo
Descripción: Permite setear el tamaño del buffer principal utilizado por el proxy. String que represente el llamado a una función de transformación.

3. Problemas encontrados

A lo largo del desarrollo del servidor, nos encontramos con los siguientes problemas:

- Entendimiento y buen manejo del código provisto por la cátedra.
- Adaptación del código de las entregas opcionales a nuestra implementación (se debe principalmente a no haber contemplado todos los casos).
- Discusiones del grupo a la hora de manejar errores.
- Poco conocimiento de casos específicos documentados en los RFC, que llevaron a lecturas rápidas con poco tiempo.
- Poco tiempo para entender y desarrollar la implementación en relación a la cantidad de trabajo que implicaba.
- Amplia discusión en las mejores prácticas a la hora de realizar parseos y cómo integrarlos con el resto de la implementación.
- Falta de modularización de código para reducir la repetición.
- A la hora de hacer el logging, en vez de mantener el archivo abierto y escribir directamente en él, se abre y cierra el archivo. Esto se acredita a la falta de experiencia en el manejo de archivos para estos casos y una tardía detección del problema que causaba.
- Dificultades a la hora de settear los tamaños de los buffers. Ya que los tamaños elegidos para estos impactan drásticamente en la performance del servidor.
- Mantener la claridad del código resultó dificultoso debido a los problemas que surgen y los distintos estilos de coding de cada uno. Para atacar este problema, nos pusimos de acuerdo en un estilo particular y al finalizar la implementación intentamos aplicarlo a la totalidad del código, comentando todo lo posible.
- Por cuestiones de tiempo, no pudimos realizar testeos fiables de carga y velocidad. Sabemos que esto es muy importante a la hora de hacer correcciones de performance y throughput y nos gustaría haber podido realizar estos tests con mayor profundidad.
- En cuanto a testing de funcionamiento, el testing fue bastante monótono y cercano a las guías provistas por la cátedra.
- Lo que respecta a testing modular, a pesar de ser en profundidad encontramos dos problemas. El primero es que fueron realizados por una o dos personas, lo cual reduce la variedad de tests. El segundo es que algunos quedaron deprecados por cambios posteriores en el funcionamiento de los módulos.
- La implementación de los logs es bloqueante, no seria dificil pasarlo a no bloqueante, pero por cuestiones de tiempo no se llevó a cabo.

4. Limitaciones de la aplicación

4.1. Reducción de performance con transformaciones

Debido a la implementación, al activarse las transformaciones se realizan muchas operaciones (de copia, por ejemplo). Esto afecta la velocidad de respuesta del servidor en gran medida.

4.2 Concurrencia de usuarios

La aplicación tiene como máximo teórico definida la concurrencia de 50 usuarios.

4.3 Limitaciones de cantidad de headers

Por tener un tamaño definido para la cantidad de headers que vamos a aceptar, si se envían muchos de estos en un mensaje, algunos no serán transmitidos.

4.4 Conexiones persistentes

No se soportan conexiones persistentes.

4.5 IPv6

Por cuestiones de tiempo, se soportan IPv6 para el request en el URI pero no en el header host.

4.6 Métodos soportados

La implementación solo soporta los métodos GET, POST y HEAD.

4.7 Transformaciones

No se soportan transformaciones con contenido comprimido.

4.8 Recursión

Al utilizar el proxy como origin server, este lo detecta y devuelve un error de la familia de los 500. Esto se lleva a cabo marcando los mensajes que pasan por el proxy con un header. Tomamos esta decisión por que creemos que está es la manera más práctica de manejar la situación.

4.9 Sed

Para poder utilizar la función sed de linux, nos vimos obligados a agregar el flag “-u” para lograr que funcionara.

5. Posibles extensiones

Algunas de las extensiones posibles serían evitar los errores que tuvimos, realizar mayor testing para mejorar la performance del servidor. Adicionalmente se podrían agregar los siguientes features:

5.1 Métodos

Agregar los métodos PUT y DELETE.

5.2 Caching

Una evidente mejora que se le puede incluir al proxy, guardando en memoria o en disco (se pueden contemplar cualquiera de las dos opciones dependiendo del comportamiento buscado) los recursos que piden los clientes, una característica común en este tipo de aplicaciones.

5.3 Blacklisting

Otra característica útil en un proxy es la posibilidad de bloquear el acceso a ciertos clientes o a ciertos servidores, creando "listas negras" para cada caso. Para configurar dicha funcionalidad, se puede aprovechar la forma actual de comunicarse con el proxy mediante el protocolo diseñado, incluyendo un nuevo método y pasando por parámetros los IPs o los dominios de los hosts que se quieren prohibir.

5.4 HTTPS

Dado que la mayoría de las páginas web más visitadas requieren HTTPS, implementar esta funcionalidad ampliará significativamente los casos de uso del proxy.

5.5 Transformación

Con un parser de HTML apropiado, podría realizarse la transformación en el contenido textual de las páginas cargadas y no en sus tags, permitiendo el display en un User-Agent.

5.6 Seguridad

Si bien existe una autenticación con password, creemos que se podría extender este aspecto, otorgando mayor seguridad a nuestro servidor y evitando el mal manejo de los datos.

6. Conclusiones

A pesar de que estamos conformes con los resultados logrados, y que pudimos implementar la mayoría de las decisiones que tomamos de manera efectiva, creemos que podríamos haber realizado un trabajo muchísimo mejor en caso de haber podido dedicar más tiempo al trabajo. Esto no se debe a cuestiones de tiempo en sí, sino a otras responsabilidades que afectan a cada integrante en particular.
La diferencia que logramos reuniéndonos y trabajando en equipo al no haber tareas más urgentes fue crucial para la finalización del trabajo.
En una reflexión un tanto más profunda, estamos de acuerdo en que podríamos haber realizado más tests y agregado algunas funcionalidades que habrían tenido un impacto muy positivo en el funcionamiento y nos permitirían tener código mucho más claro y extensible.

7. Ejemplos de prueba

Para probar el funcionamiento del servidor se pueden ejecutar los siguientes comandos utilizando la herramienta “netcat” :

- proxy
> ./proxy
- actuar como cliente (desde otra terminal)
> nc localhost 1080 -C
- actuar como origin (desde una tercera terminal)
> nc -l 2000 -C

Al ejecutar estos comandos, desde la consola cliente podremos enviar un request, cuyo procesamiento veremos desde la terminal de proxy y obtendremos la respuesta al cliente. Desde la terminal origin, podremos enviar la respuesta, nuevamente veremos el procesamiento de está en la terminal de proxy y la correspondiente respuesta en la terminal origin.

8. Guía de instalación

Para obtener el binario ejecutable se necesita la herramienta “make”.
se deben ejecutar los siguientes comandos desde el directorio “pc-2018-05”:

```
$ cd src
$ make
```

Esto generará el binario “proxy” en el path “pc-2018-05/src”.
En el directorio “pc-2018-05” se encuentra un archivo “top-secret.txt” que contiene la password para modificar la configuración del proxy.

9. Instrucciones para la configuración

Para la configuración se siguieron los lineamientos propuestos por la cátedra en el man page de linux subido con el nombre httpd.8.
Se carga por defecto la la configuración predeterminada

-e archivo-de-error Especifica el archivo donde se redirecciona stderr de las ejecuciones de los filtros. Por defecto el archivo es /dev/null.

-h Imprime la ayuda y termina.

-T Activa la transformación.

-l dirección-http Establece la dirección donde servirá el proxy HTTP. Por defecto escucha en todas las interfaces.

-L dirección-de-management Establece la dirección donde servirá el servicio de management. Por defecto escucha únicamente en loopback.

-M media-types-transformables Lista de media types transformables. La sintaxis de la lista sigue las reglas del header Accept de HTTP (sección 5.3.2 del RFC7231). Por defecto la lista se encuentra vacía. Por ejemplo: text/plain,image/* transformará todas las respuestas declaradas como text/plain o de tipo imagen como ser image/png.

-o puerto-de-management Puerto STCP donde se encuentra el servidor de management. Por defecto el valor es 9090.

-p puerto-local Puerto TCP donde escuchará por conexiones entrantes HTTP. Por defecto el valor es 8080.

-t cmd Comando utilizado para las transformaciones externas.
Por defecto: sed -u -e 's/a/4/g' -e 's/e/3/g' -e 's/i/1/g' -e 's/o/0/g' -e's/s/5/g'.

-v Imprime información sobre la versión versión y termina.

10. Ejemplos de configuración y monitoreo

10.1 Habilitar transformaciones

```
>  ADMIN_V1 admin enable_transformer
```

10.2 Media types transformables

```
>  ADMIN_V1 admin type_transformer {param}
```

10.3 Comando de transformación

(sample values)

```
>  ADMIN_V1 admin command_transformer {param}
>  + ****PROXY TRANSFORMER****
>  + STATUS: OFF
>  + Command: cat
>  + Filter:
```

10.4 Logs

```
>  ADMIN_V1 admin logs
>  + print logs
```

10.5 Métricas

(sample values)

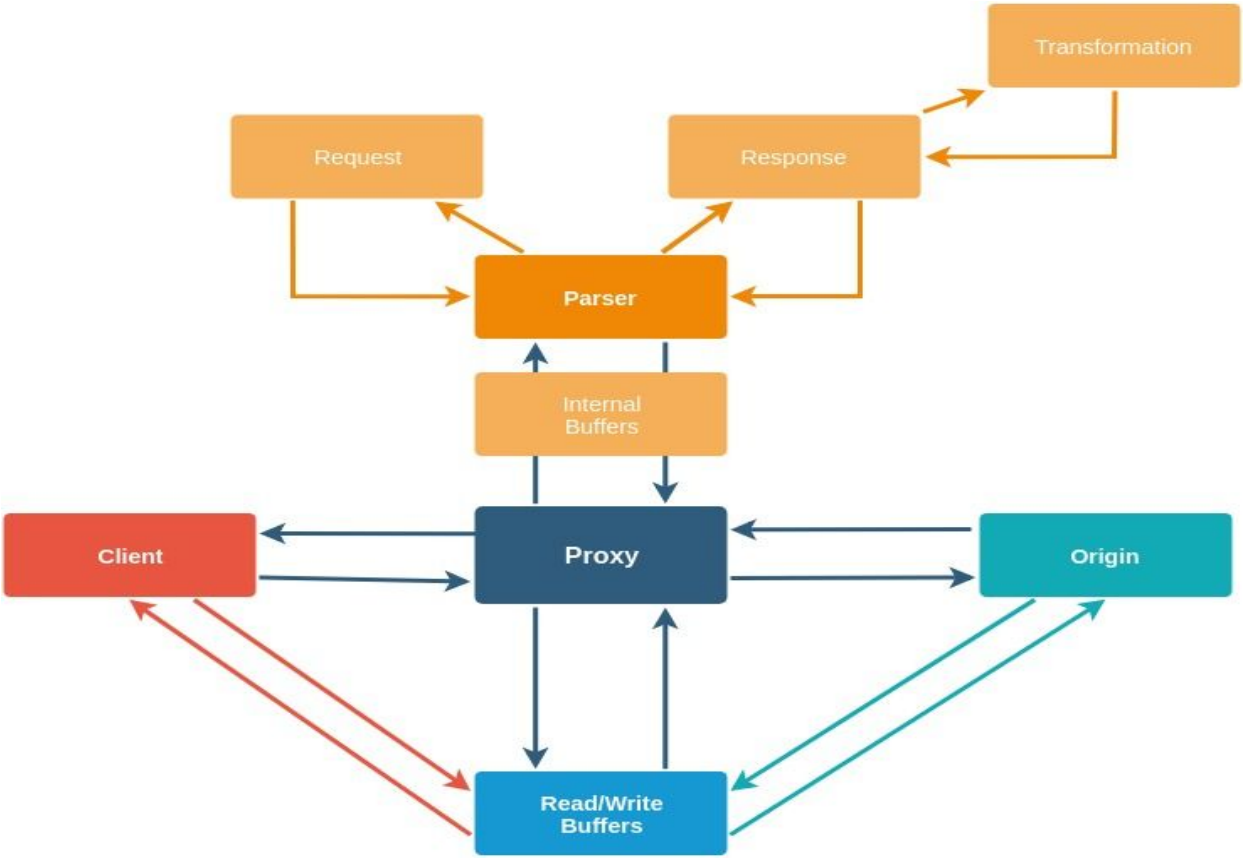
```
>  ADMIN_v1 admin metrics
>  + ****PROXY METRICS****
>  + HTTP port: 1080
>  + SCTP port: 1081
>  + Concurrent Users: 0
>  + Total Users Connected: 0
>  + Transfered bytes: 0
```

10.6 Buffer size

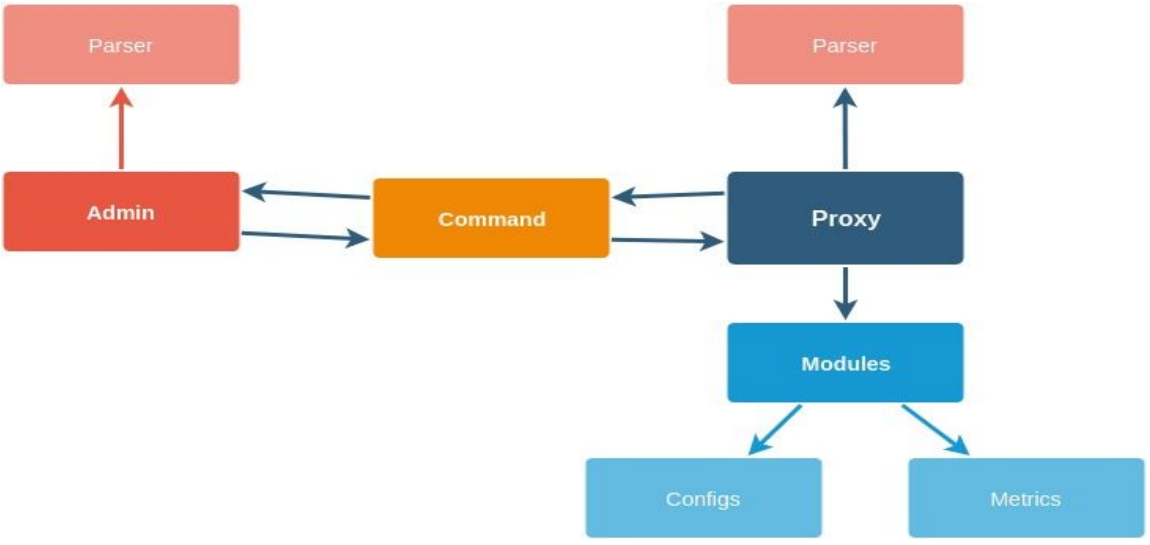
(sample values)

```
>  ADMIN_V1 admin buffer_transformer {param}
>  +****PROXY TRANSFORMER****
>  +STATUS: OFF
>  +Command: cat
>  +Filter:
>  +Buffer: 100
```

11. Documento de diseño del proyecto



Como se puede ver en el diagrama, se refleja el flujo explicado en el ítem **2.1.1** . Notar que los buffers de Read/Write son a través de los cuales el proxy recibe los mensajes y los devuelve a los respectivos client/origin servers.
Los buffers internos son aquellos a través de los cuales el servidor proxy se comunica con los parsers, para recibir información importante de los mensajes y poder parsear el body sin tener que guardarlo completamente en memoria.



La imagen anterior representa el flujo entre el servidor proxy y el protocolo admin implementado. Mediante comandos, se comunican mutuamente para poder acceder a las métricas y hacer cambios en las configuraciones (por ejemplo para habilitar las transformations).

A continuación se explican las tareas de algunos de los archivos principales:

- /src/httproxynio.c: punto de entrada del proxy, donde se lleva a cabo todo el manejo del flujo de mensajes.
- /src/admin.c: se conecta con el servidor, envía un mensaje donde se puede visualizar el estado de las configuraciones.
- /src/sctpadminnio.c: manejo de mensajes del protocolo implementado, funcionando sobre sctp.
- /src/main.c: monta un socket pasivos y maneja las conexiones entrantes.
- /src/proxy_state.c: almacena la estructura general que tiene las configuraciones de comunicación entre admin y server.
- /src/selector.c: multiplexor de entrada/salida, permite mantener la concurrencia de manera no bloqueante.
- /src/HTTPResponsev2.c : parser del response. Máquina de estados que byte a byte interpreta la respuesta y en caso que sea necesario, almacena información.
- /src/HTTPRequest.c : parser del request. Máquina de estados que byte a byte interpreta y en caso que sea necesario, almacena información.
- /src/buffer_size.c : contiene funciones para el seteo del buffer y se establecen los defaults.
- /src/logging.c : maneja los logs del servidor.
- /src/transformation_program_invoker.c : se encarga de la creación de un hijo y el manejo de pipes para ejecutar la transformación.