TRABAJO PRÁCTICO ESPECIAL

PROTOCOLOS DE COMUNICACIÓN

PROXY HTTP

CAVALLIN, FLORENCIA
FARIÑA, SEGUNDO
PAGANINI, NICOLÁS A.
VICTORY, MARTÍN

ÍNDICE

Introducción

El objetivo del trabajo práctico especial fue implementar un servidor proxy para el protocolo HTTP versión 1.1 que pudiese ser usado por User Agents tal como curl, Mozilla Firefox, Internet Explorer y Google Chrome para navegar por Internet. Para lograr este objetivo se debió programar las aplicaciones de cliente/servidor con sockets, teniendo en cuenta los estándares de la industria y diseñando nuestro propio protocolo de aplicación.

El servidor proxy puede ser utilizado desde los User Agents con configuración previa de los mismos o como proxy transparente. En el primer caso el User Agent es consciente de que se está comunicando con un proxy. Debido a esto, es necesario que el User Agent envíe la URL absoluta correspondiente a la request (esto está especificado en el RFC 7231). De encontrarse en el segundo caso, el User Agent no es consciente que se está comunicando con un proxy y para obtener la URL, debe obtener el host a través del header Host en el caso de que no se envíe la URL.

El servidor proxy puede soportar múltiples clientes de forma concurrente y simultánea. El mismo reporta los fallos a los User-Agents usando los códigos de error del protocolo HTTP.

Diseño

El proxy http corre en un único thread multiplexado, su implementación está basada en el selector provisto por la cátedra. Al iniciar el proxy se registra el file-descriptor en el cual va a estar escuchando el proxy. Cuando se

Parsers

Los parsers fueron implementados como máquinas de estados. Todos ellos leen byte a byte de un buffer. De ésta manera, a medida que se va parseando la información, el parser va cambiando su estado interno y en algunos casos, los bytes se van derivando a otros parsers internos.

Como los parsers fueron implementados de manera tal de ir procesando byte a byte, los mismos funcionan de igual manera sin importar la cantidad de bytes que deben procesar en cada request o response. De ésta manera, las requests y responses se procesan a medida que se leen, sin necesidad de tener que guardarlas en su totalidad en el buffer.

Muchos de los parsers implementados están compuestos por otros parsers más especificos internos. De ésta manera mejoramos el entendimiento y prolijidad del código, logrando mayor modularización.

Cada parser, genera una estructura donde guarda la información relevante que extrajo de lo que procesó. De esta manera el proxy logra conocer propiedades específicas del request o response en cuestión.

Handlers

Cuando se crea el servidor proxy, se crea un handler pasivo para el file descriptor de lectura del servidor. Este handler especifica el procedimiento que se debe realizar a la hora de leer del file descriptor en cuestión. De igual forma se crea un handler pasivo para el file descriptor de lectura del administrador.

El servidor proxy luego, crea un socket pasivo para el servidor proxy y otro para el administrador. El puerto del proxy es por default 9090, mientras que el del administrador es 8080. Ambos puertos pueden ser cambiados por parámetro.

El servidor agrega ambos sockets a un selector implementado y comienza a escuchar en ambos. Cuando se recibe una conexión en el socket del servidor proxy, se crea un socket activo para manejarla y se la agrega al selector. En caso de recibir una conexión en el socket del administrador, también se crea un socket activo con su respectivo handler y se lo agrega al selector de igual forma.

Cliente

El cliente o administrador se conecta por consola al servidor proxy e interactúa con él en tiempo real. Puede haber más de un administrador conectado. Los comandos posibles son:

settransf <t>	setea la transformación t pasada por parámetro. En caso de no ser una transformación válida, no la setea.
rmtransf	elimina la transformación actual. En caso de no haber transformación seteada, no hace nada.
addmediatype <mt>	agrega el media type mt pasado por parámetro. En caso de no ser un media type válido, no lo agrega.
addmediatypes <mt1>, <mt2>, <mt3>	agrega todos los media types pasados por parámetro. En caso de que alguno de los media types no sea válido, no lo agrega. Solo agrega los válidos.
rmmediatype <mt>	elimina el media type mt pasado por parámetro. En caso de que el media type no sea válido, no hace nada.
rmmediatypes <mt1>, <mt2>, <mt3>	elimina todos los media types pasados por parámetro. En caso de que alguno

	se los media types no sea válido, no hace nada. Elimina todos los media types válidos.
getmetrics	muestra en salida estándar la cantidad de veces que se utilizó el método head, post y get. También la cantidad de clientes activos y el historial de conecciones.
gettransf	muestra en salida estándar la transformación actual.
getmediatypes	muestra en salida estándar todos los media types agregados.

Logs

Decidimos loggear solo las nuevas conexiones y los errores.

Protocolo

El protocolo está especificado en el RFC adjunto.

Problemas encontrados durante diseño e implementación

Modularización

Durante el desarrollo del proyecto se dió en evidencia el tamaño que una aplicación así conlleva. Para poder manejar los tiempos más eficientemente, se decidió dividir el trabajo entre los integrantes del grupo y para esto se desarrollaron varias ramas de trabajo. Además, se llevó a cabo la separación de archivos y de carpetas del proyecto para poder encontrar el código de manera eficaz.

Protocolo

Originalmente, el protocolo se había ideado y presentado ante la clase como orientado a texto. Luego de recibir feedback de parte del profesor, decidimos cambiarlo a un protocolo orientado a byte para así aprovechar los beneficios que ofrecía SCTP.

Limitaciones de la aplicación

Métodos soportados

Los métodos soportados por este proxy son GET, HEAD y POST.

Buffers

Los tamaños de los buffers se estandarizaron a través del proyecto. Estos son estáticos y son de 2048 bytes.

Conexiones

La aplicación cuenta con un número limitado de clientes que puede atender. Al mismo tiempo puede tener activos 509 clientes, si no están habilitadas las transformaciones, y 254 clientes si están habilitadas las transformaciones.
Estos números provienen de una base de 1024 conexiones menos 5 que están reservadas.

Posibles extensiones

Modificaciones de Buffer y Timeouts

Durante las últimas etapas del desarrollo del proyecto se decidió dejar de lado las opciones para modificar el tamaño de los buffers y el tiempo de los timeouts al proxy debido a los límites de tiempo impuestos. Sin embargo, la posibilidad de extender el proyecto a incluir estas modificaciones es una extensión posible.

Conexiones persistentes

Durante el desarrollo del proyecto se tomó la decisión de no implementar éstas dado el tiempo limitado de este proyecto y la dificultad que éste representaba.

HTTPS

Dado el surgimiento y el frecuente uso de HTTPS por las páginas más visitadas, agregar esta funcionalidad al proxy lo haría útil en más ocasiones.

Conclusiones

Finalmente, se logró desarrollar una implementación de la aplicación de acorde a lo pedido.

Como se puede ver en el historial de git, la división del trabajo fue manejada muy eficientemente. Desde la separación de branches hasta la modularización y separación de código, fue de suma importancia mantener todo separado y organizado correctamente. Esto queda en evidencia en la cantidad de archivos que se encuentran en el proyecto.

Ejemplos de prueba

Los estados que devuelve el proxy están mejor detallados en la carpeta proxyStates, dentro de serverComponents.

Los métodos soportados son GET, HEAD y POST.

El proxy se adapta a lo especificado por los RFCs correspondientes respecto a los métodos soportados (por ejemplo si el GET no cuenta con la URL absoluta, el proxy busca el host debajo del header “Host” como especifica el RFC).

Guía de instalación

Muchas gracias por elegirnos! Sabemos que tiene muchas opciones de Proxy HTTP y esperamos que esté satisfecho con el proporcionado. Sin más preámbulo, a continuación se detalla la forma apropiada de instalar el proxy para ser usado.

Para facilitar esta etapa, se concluyó incluir los siguientes programas para automatizar el proceso.

Primero que nada, necesitará instalar los programas “git” y “cmake” para utilizar el proxy.

Paso 1: Clone el repositorio encontrado en
https://bitbucket.org/itba/pc-2018-04/src/master/
Paso 2: Ingrese a la carpeta “pc-2018-04”. Esta es la carpeta raíz del proyecto.
Paso 3: Cree una carpeta “build”. Ingrese a esta.
Paso 4: Corra el siguiente comando “cmake ..”
Paso 5: Corra el siguiente comando “make proxy”
Paso 6: Corra el siguiente comando “make admin”

Listo! Su Proxy HTTP está listo para ser utilizado. Asegúrese de configurar su computadora para que los pedidos pasen por el proxy y luego ejecútelo con “./proxy”.

Si deseara correr el programa de administrador para cambiar las opciones del proxy puede correr el programa “./admin” proporcionado o crear el suyo respetando el protocolo establecido en la carpeta raíz, en su última versión.

Instrucciones para la configuración

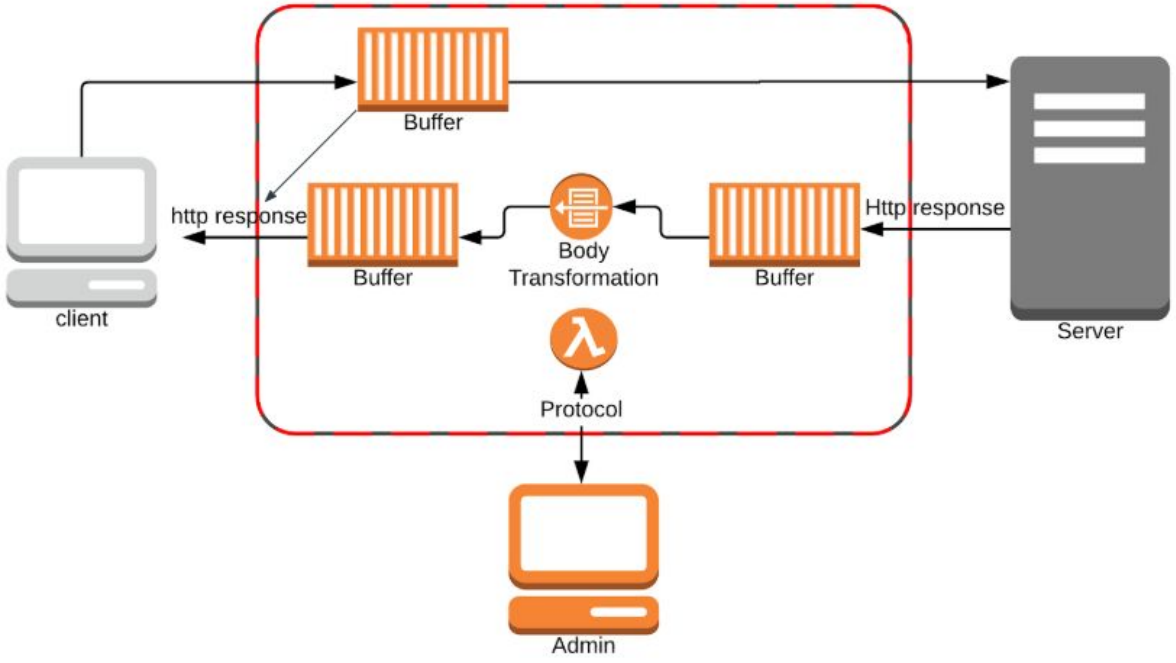
El proxy cuenta con configuraciones que pueden ser alteradas como lo desee el administrador. Estas se encuentran detalladas en la sección de “Diseño”, bajo “Cliente”. Estas incluyen la posibilidad de establecer la transformación actual y agregar media-types en runtime.

Monitoreo

La aplicación cuenta con servicios de logs de acceso como también logs de errores.

Además de éstos, se puede monitorear el estado del proxy a través de los servicios de métricas ofrecidas al administrador a través del protocolo establecido.

Foto y diseño del Proxy HTTP



El proxy http corre en un único thread multiplexado, su implementación está basada en el selector provisto por la cátedra. Las carpetas del proyecto están separadas acorde a sus distintas funciones a cumplir.