|  |
| --- |
| Protocolos de comunicación |
| Informe pre-entrega |
|  |

* Besteiro, Florencia 51117
* Buireo, Juan Martín 51061
* Menzella, Facundo 51533
* Purita, Martín 51187

23-10-2013

Índice

Introducción ………………………………………………………………………………………………… 2

Descripción y análisis de los RFC ………………………………………………………………………………… 2

RFC 2616 ………………………………………………………………………………………………………… 2

RFC 3117 ………………………………………………………………………………………………………… 4

Descripción del protocolo de monitoreo …………………………………………………………………… 4

Potenciales problemas detectados en el RFC ……………………………………………………………… 6

Establecimiento de aplicaciones ………………………………………………………………………………… 6

Casos de prueba a realizarse ……………………………………………………………………………………… 7

Introducción

El objetivo del trabajo práctico es realizar un proxy HTTP server que funcione tanto de manera transparente como con una configuración previa.

Para la implementación del proxy server se tuvieron en cuenta los RFC 2616 (HTTP) y 3117 (diseño de protocolos).

Previo al análisis de los RFC fue necesario comprender algunas palabras claves explicadas en el RFC 2119 para comprender aquellos requisitos obligatorios de aquellos deseables. Dichas palabras son las siguientes:

1. MUST (DEBE): esta palabra significa que la definición es un requisito absoluto de la aplicación.
2. MUST NOT (NO DEBE): esta palabra significa que la definición debe estar prohibida en la aplicación.
3. SHOULD (DEBERÍA): esta palabra significa que la definición puede ser ignorada en ciertas circunstancias en la aplicación.
4. SHOULD NOT (NO DEBERÍA): esta palabra significa que la definición no estar ignorada en ciertas circunstancias en la aplicación.
5. MAY (PUEDE): esta palabra significa que la definición es opcional en la aplicación.

1.MUST (DEBE): esta palabra o los términos “requerido” o “deberán” significa que la definición es un requisito absoluto de la aplicación.  
2. MUST NOT (NO DEBE): esta palabra o el término “no deberán”, significa que la definición debe estar prohibida en la aplicación.  
3. SHOULD

This word, or the adjective "RECOMMENDED", mean that there  
  may exist valid reasons in particular circumstances to ignore a  
  particular item, but the full implications must be understood and  
  carefully weighed before choosing a different course.

4. SHOULD NOT

Descripción y análisis de los RFC

RFC 2616

El RFC 2616 fue de vital importancia para comprender las especificaciones deseables de un proxy HTTP.

Una vez leído dicho RFC llegamos a la conclusión que hay que ser muy poco restrictivo en lo que leemos y muy restrictivos en lo que enviamos.

El formato de un request es el siguiente:

Request = Request-Line                
                 \*(( general-header         
                   | request-header           
                   | entity-header ) CRLF)   
                   CRLF  
                   [ message-body ]

Hay que tener en cuenta ciertas características de HTTP. No se deben aceptar requests cuya versión sea distinta a 1.1 (el RFC propone enviar un código de error o bien bajar la versión).

Es aconsejable que se retorne un error de tipo 414 (request-uri demasiado larga) si la dirección es más larga que lo que el servidor proxy puede manejar.

Las fechas/timestamps deben estar en formato GMT (Greenwich Mean Time) sin excepción.

Al recibir un request con cuerpo, es condición necesaria que se encuentre el header Content-length o bien Transfer-Encoding (para enviar de a partes). De esta manera se puede saber hasta dónde hay que leer el cuerpo del request. Un request con cuerpo sin dichos headers debe ser inválido (ya que no se sabe en qué momento dejar de leer).

El formato del response es el siguiente:

Response = Status-Line

                     \*(( general-header        
                      | response-header      
                      | entity-header ) CRLF)  
                      CRLF  
                      [ message-body ]

Los responses al método HEAD no deben incluir nunca un cuerpo (de lo contrario sería un response inválido). Los códigos de respuesta 1xx, 204 y 304 tampoco deben incluir un cuerpo. El resto de las respuestas pueden incluir un cuerpo aunque la longitud del mismo pueda ser 0.

Un dato importante que se menciona acerca de los proxy´s transparentes es que nunca se debe reescribir la sección “abs\_path” del Request-URI recibido cuando se realiza un forwarding al servidor siguiente, a excepción del reemplazo de un “abs\_path” NULL con “/”.

RFC 3117

A la hora de diseñar un protocolo de aplicación hay 4 formas de proceder:

1. Encontrar un protocolo de intercambio existente que más o menos haga lo que se pretende.
2. Definir un modelo de intercambio por sobre la capa de transporte que más o menos haga lo que se pretende.
3. Definir un modelo de intercambio por sobre la estructura de correo electrónico que más o menos haga lo que se pretende.
4. Definir un nuevo protocolo desde sus raíces.

Analizando esto, evidentemente la solución más acertada es; a nuestro entender;  la número 1, montando un protocolo de monitoreo sobre HTTP. (HTTP define las reglas para intercambio de datos, y nuestro protocolo, define una sintaxis y una semántica para ese intercambio).

Además, al utilizar HTTP, se cubren los aspectos esenciales a tener en cuenta a la hora de utilizar un protocolo de aplicación: límites(Content-lenght header),  formato de codificación (MIME), reportes de errores (3-dígitos), asincronismo (“chuncking”), autenticación (usuario/contraseña), y privacidad (TSL).

Descripción del protocolo de monitoreo

Como el trabajo práctico se basa en un proxy HTTP (versión 1.1), se decidió implementar el protocolo de monitoreo sobre HTTP debido a que ya se tiene implementado un parser con lo cual bastaría sólo con extenderlo. De esta manera nos ahorraremos trabajo y minimizaremos la probabilidad de error al contar con un solo protocolo en la aplicación.

Debido a que lo que se pide es un monitoreo, es posible que este protocolo sea request/response ya que se solicita una acción y se obtiene una respuesta de la misma con sus respectivos datos. Dado que los datos se pueden mostrar de distinta manera (ya sea que el cliente se encuentre en un navegador web o en una consola), se decidió reutilizar el header Accept para solicitar de qué manera se desean ver los datos. Esto nos permitirá mostrar los datos de manera textual, en código html o en alguna otra forma que se desee.

Todos los comandos de consulta serán ejecutados con GET.

Existe un solo comando para modificar la configuración (activar o desactivar la transformación de texto plano) por lo cual este dato será diferenciado en su forma de solicitarse (el mismo se solicitará con POST).

Ejemplo de interacción entre cliente-servidor:

El servidor del protocolo se encuentra corriendo en algún puerto (predeterminado) y un cliente se conecta a dicho puerto. Luego comienza la interacción entre cliente-servidor:

Cliente: GET /bytes PROT/1.1

Accept: text/plain

Server: PROT/1.1 200 OK

Content-Length: 32

Bytes transferidos: 2131 bytes

Cliente: POST /transformation PROT/1.1

Activate: off

Server: PROT/1.1 200 OK

Cliente: GET /histogram PROT/1.1

Accept: text/json

Server: PROT/1.1 200 OK

Content-Length: 1000

[JSON]

Cliente: GET /ajsdsasjdklsa PROT/1.1

Server: PROT/1.1 400 Bad Request

En este ejemplo se observa una interacción entre un cliente y el servidor proxy donde se puede ver cómo se reutiliza el protocolo HTTP con los métodos GET y POST (y el uso de algunos headers).

El RFC 3117 habla de 5 propiedades a tener en cuenta a la hora de diseñar un protocolo de aplicación. Estas son escalabilidad, eficiencia, simplicidad, extensibilidad y robustez.

Nuestro protocolo es eficiente, dado que utiliza solamente un header a la hora de hacer un request; y un header a la hora de enviar un response válido, seguido del contenido propiamente dicho.

Nuestro protocolo es simple, dado que existe una sola forma de realizar las cosas, y, citando al RFC: “Tareas simples, deben de ser simples de realizar”.

Nuestro protocolo es extensible, dado que ante cualquier incremento de funcionalidad, ya sea un comando nuevo, un formato nuevo, etc; sólo se deben agregar los headers pertinentes.

Potenciales problemas detectados en el RFC

Uno de los problemas que se nos presentó al momento de leer el RFC fue el header Content-Length.

Debido a que se comenzó a realizar el parser HTTP sin haber leído previamente el RFC 2616, hubo que cambiar la manera en que se leían los datos. En un principio se leía el request por líneas (tanto la primera línea como los headers y el cuerpo). Pero esto era algo incorrecto ya que no se sabía en qué momento dejar de leer. Fue por eso que se decidió alterar esta implementación de manera que ahora se leen sólo la primera línea y los headers por líneas mientras que el cuerpo del request se lee respetando el tamaño indicado por el header Content-Length. En caso que dicho header no se encuentre presente, el request no debe tener cuerpo, caso contrario será inválido.

Otro potencial problema aún no resuelto es cuando se maneje el archivo en pedazos (esto es a través de Transfer-Encoding). De alguna manera debemos frenar el envío de datos hacia el servidor destino hasta tanto no tener el request completo recibido.

Establecimiento de aplicaciones

Durante el desarrollo se utilizará el User Agent Chrome para testear la aplicación mediante una configuración previa del mismo. Como se está en una etapa temprana del desarrollo aún no se definió la forma en que se puede probar el proxy de manera transparente.

También se utilizó la herramienta netcat para enviar requests más simples (esto nos permite generar nuestros propios requests y detectar errores de manera más simple).

Casos de prueba a realizarse

Las pruebas deben basarse en el envío de datos (bien y mal formados) hacia distintos servidores de manera que pueda comprobarse que el proxy responda de manera correcta a las solicitudes (en caso que los requests estén bien formados, caso contrario debe especificar el error).

Por otro lado debido a que se están utilizando sockets no bloqueantes, una prueba deseable sería que muchos clientes envíen datos a la misma vez sin ver afectada considerablemente la performance. Esto no sucedería si se hubiese realizado con sockets bloqueantes.