

```
pdc@itba$ Protocolos de comunicación
pdc@itba$ Trabajo Práctico Especial
pdc@itba$ Grupo 01
pdc@itba$ Juan Grethe          -1    57370
pdc@itba$ Juan Manuel Alonso   -1    56080
pdc@itba$ Martin Capparelli    -1    55010
pdc@itba$ Martin Grabina       -1    57360
pdc@itba$ 1C-2018
pdc@itba$ Instituto Tecnológico de Buenos Aires
```

## Abstracto

Este documento describe el Trabajo Especial de la materia Protocolos de Comunicación del Instituto Tecnológico de Buenos Aires para la cursada del primer cuatrimestre de 2018 del grupo 1 conformado por los integrantes descritos en la carátula.

Se asume que los lectores de este documento manejan los términos básicos de protocolos de comunicación.

# Índice

<b>Abstracto</b>	<b>1</b>
<b>Índice</b>	<b>2</b>
<b>Desarrollo</b>	<b>4</b>
Proxy	4
Protocolo J2M2	4
Introducción	4
Estructura de datagrama	5
Comandos	5
Códigos de estado	6
Requests	7
Responses	7
Dinámica de conversación	7
Observaciones a tener en cuenta para desarrollar su propio cliente de J2M2	8
<b>Problemas</b>	<b>9</b>
<b>Limitaciones</b>	<b>12</b>
<b>Posibles extensiones</b>	<b>13</b>
<b>Conclusiones</b>	<b>14</b>
Aprendizaje	14
Escalabilidad	14
Conceptos Técnicos	14
<b>Ejemplos de prueba</b>	<b>15</b>
Testeos básicos propuestos por la cátedra	15
Testeos adicionales	15
Posteo de caracteres chinos	15
Metodo SEARCH (No implementado, aplica para cualquier otro método HTTP no implementado)	15
<b>Performance</b>	<b>16</b>
50 threads - GET de 10Mb, completado satisfactoriamente.	16
50 threads - POST de 10Mb, completado satisfactoriamente.	16
<b>Guia de instalacion</b>	<b>17</b>
Repositorio	17
Compilación	17
Ejecución	17
Proxy Server	17
Cliente SCTP	17
<b>Instrucciones de configuración</b>	<b>19</b>

Valores Default	19
Proxy	19
Cliente HTTP	19
Cliente SCTP	19
<b>Ejemplos de uso</b>	<b>20</b>
Proxy	20
Mediante cURL	20
Mediante User Agent	20
Cliente SCTP	20
<b>Documento de diseño de proyecto</b>	<b>21</b>
Estructura del repositorio y archivos destacados	21
Makefile	21
Estadísticas del repositorio	22
<b>Bibliografía y librerías utilizadas</b>	<b>23</b>
<b>Anexo I</b>	<b>24</b>
Manual httpd.8	24
<b>Anexo II</b>	<b>26</b>
Testeos provistos por la cátedra con sus respectivos puntajes	26
Para información más detallada de cada test, consultar la documentación oficial en material didáctico del curso.	26

# Desarrollo

## Proxy

Tal como indica la consigna se ha desarrollado un proxy HTTP transparente. El proxy..

En cuanto al manejo de las conexiones, nos ajustamos a lo indicado por el RFC 7230:

*A server that does not support persistent connections MUST send the "close" connection option in every response message that does not have a 1xx (Informational) status code.*

Vale la pena destacar también, que por la forma en que se ha desarrollado el mismo, no se generan problemas al transferir caracteres de ningún tipo (véase test personalizado de posteo de caracteres chinos en la sección pruebas).

## Protocolo J2M2

### Introducción

El protocolo J2M2 (Juan x2 & Martin x2) fue diseñado especialmente para la administración del proxy y la obtención de métricas del mismo. Además teniendo en cuenta las funcionalidades que posee existe la posibilidad de seteo de nuevas configuraciones. El protocolo es un protocolo binario, es decir está orientado a la lectura entre máquinas. Consideramos que un protocolo binario es adecuado dado que la potencia que nos ofrece es conveniente a nuestra necesidad.

El mismo está diseñado para correr sobre SCTP (Stream Control Transmission Protocol), utiliza autenticación mediante usuario y contraseña almacenado en el servidor J2M2 de manera volátil (ver: futuras extensiones), asume los caracteres en formato ASCII básico (1 Byte) y asegura una única respuesta por cada pedido del cliente:

Client -----> Server

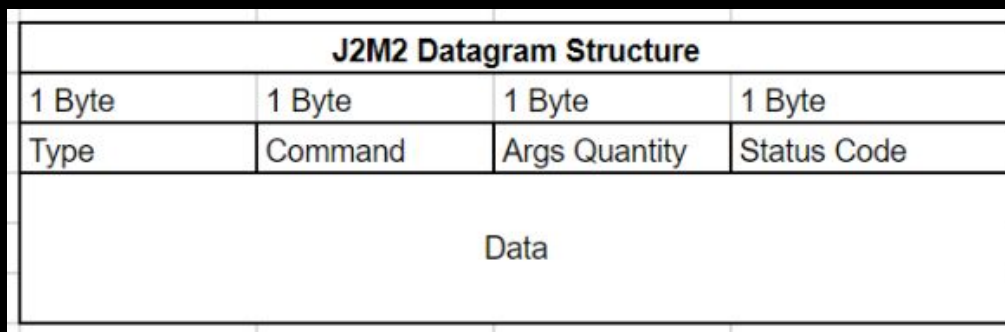
Client <----- Server

Es decir, una request tiene una response asociada, siempre. Lo que permite además cerrar las conexiones de un modo claro y prolijo .

Si bien el protocolo es binario, frente al cliente parece ser textual, y la forma de utilización es mediante línea de comandos (descritos debajo).

Las ventajas de trabajar sobre SCTP es que los mensajes se envían en datagramas, por lo tanto el protocolo lee por paquetes para facilitar la implementación. Además existe una limitación que los comandos y argumentos deben ser contenibles en el datagrama.

## Estructura de datagrama



El datagrama cuenta con 4 headers de 1 byte cada uno. El header type indica la categoría al cual el siguiente byte debe corresponder, al momento de ser parseado. Finalmente los últimos dos bytes hacen referencia la cantidad de argumentos y el estado del datagrama.

## Categorías de comandos

Existen 4 categorías de comandos, por una cuestión de simplicidad son volátiles, detallados a continuación:

1. Auth: Corresponde a la categoría de autorización, la misma se envía una única vez al comienzo de la conexión para habilitar al usuario utilizar los restantes comandos disponibles.
2. Metric: Nos permite acceder a las métricas que el protocolo ofrece, currconn, histacc, trabytes y connsucc.
3. Config: Permite poder activar o desactivar transformaciones, establecer media types y un comando para ejecución.
4. Sys: tiene solo dos comandos, exit y help. Exit cierra la conexión de manera adecuada y help muestra los comandos disponibles y la forma de utilización.

Type	
Auth	0
Metric	1
Config	2
Sys	3

## Comandos

Commands					
Auth	0				
Metric	currcon	0	Config	transform	0
	histacc	1		mediatypes	1
	trabytes	2		command	2
	connsucc	3			
Sys	help	0			
	exit	2			

*Antes de comenzar debemos aclarar que los números asociados a cada comando son los correspondientes a cada uno y que deberían ser seteados en el header del datagrama.*

**Auth** no es propiamente un comando, el comando es login, para que al cliente le resulte más sencillo comprender la forma de utilizarlo. Login recibe 2 parametros un usuario y una contraseña, ambos separados por un espacio.

*login user SP pass*

**Metric** permite acceder a los siguientes comandos, ninguno de ellos recibe parámetros:

1. currcon: Retorna la cantidad de conexiones establecidas en ese momento.
2. histacc: Retorna la cantidad de accesos históricos al proxy.
3. trabytes: Retorna la cantidad de datos transferidos. Cabe aclarar que los datos transferidos se calculan únicamente cuando el proxy escribe, esta decisión fue tomada dado que de esta forma podemos calcular cuánto se transfiere pese esté activada o desactivada la transformación.
4. consucc: Retorna la cantidad de conexiones que fueron establecidas y cerradas de manera correcta, es decir que no quedaron truncas/tuvieron error.

**Config** permite realizar modificaciones al servidor proxy.

1. transform: Este comando no recibe parámetros, simplemente funciona como un switch entre activar o desactivar las transformaciones.
2. mediatypes: Setea una lista de media-types en el servidor proxy. Es importante aclarar que para el envío de este comando el header argsq debe ser 1, lo que no indica que la cantidad de media types sea 1, sino que el argumento es una lista con uno o más media types.
3. command: Setea un comando para la transformación del proxy. Misma aclaración que comando anterior.

**Sys** en esta categoría se encuentran los comandos que permiten cerrar la conexión y recibir información.

1. exit: cierra la conexión de manera adecuada.
2. help: muestra en pantalla los comandos disponibles. Es importante aclarar que este comando debe estar implementado únicamente del lado del cliente, esto se debe a que nosotros proponemos nombres de comandos que podrían no ser respetados por la implementación del cliente. Mientras los datos se encuentren en el datagrama de manera correcta y los headers también, el servidor debería ser capaz de procesar la request.

## Códigos de estado

Status Codes	
0	Request
1	OK
2	Bad Credentials
3	Bad Request
4	Invalid Arguments
5	Unauthorized
6	
7	

\* Se dejan espacios en blanco para futuras implementaciones

## Requests

Distribucion de data - Request							
				type	command	args	status code
Auth	->	Request	->	0	0	2	0
username	" "	pass	/0	Max	128 Bytes		
Metric	->	Request	->	1	0,1,2,3	0,1	0
				Data			
Config	->	Request	->	2	0,1,2,	0,1	0
				Without Param			
Value	/0			With Param			

\*Donde aparecen enumeraciones con comas ',' debe interpretarse que puede aparecer solo uno de ellos.

En la imagen superior podemos ver la forma en que los datos se organizan en la sección data y header del datagrama. Es importante comprender que se aceptan las posibles combinaciones siempre y cuando se respete la cantidad de argumentos, el estado acorde a cada tipo de comando.

## Responses

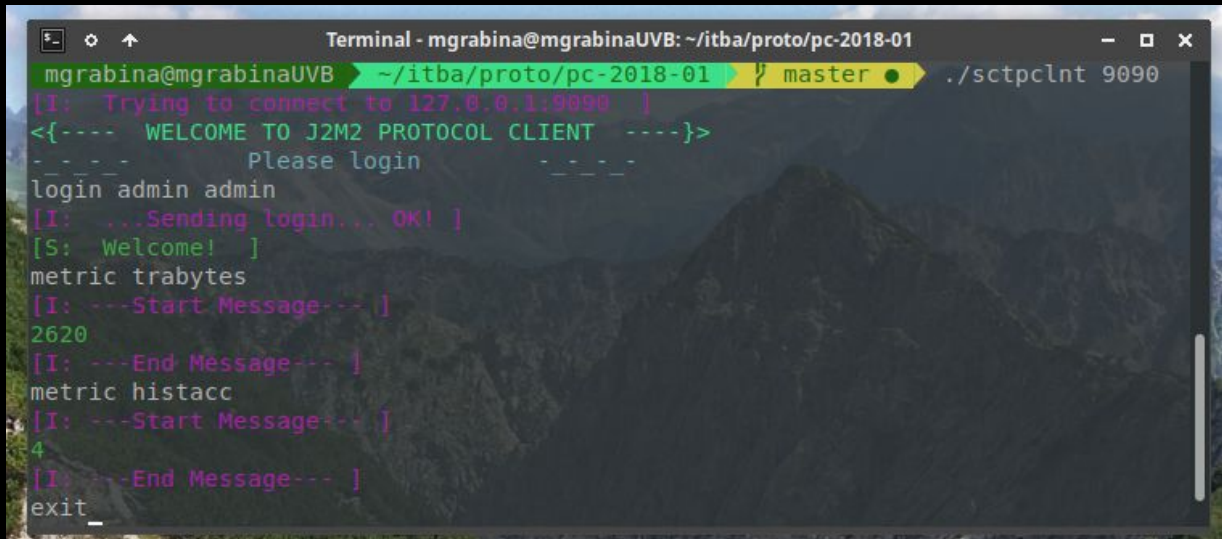
Distribucion de data - Response							
				type	command	args	status code
Auth	->	Response	->	0	0	0	1,2,3,5,
Message	/0			Max Length 128 Bytes			
Metric	->	Response	->	1	0,1,2,3	0	1,3,4,5,
Message	/0			Max Length 128 Bytes			
Config	->	Response	->	2	0,1,2,	0,1	1,3,4,5,
Message	/0			Max Length 128 Bytes			

\*Donde aparecen enumeraciones con comas ',' debe interpretarse que puede aparecer solo uno de ellos.



En la imagen superior podemos ver la forma en que los datos se organizan en la sección data y header del datagrama. Es importante comprender que se aceptan las posibles combinaciones siempre y cuando se respete la cantidad de argumentos, el estado acorde a cada tipo de comando.me

## Dinámica de conversación



```
Terminal - mgrabina@mgrabinaUVB: ~/itba/proto/pc-2018-01
mgrabina@mgrabinaUVB ~/itba/proto/pc-2018-01 master ./sctpcInt 9090
[I: Trying to connect to 127.0.0.1:9090 ]
<{---- WELCOME TO J2M2 PROTOCOL CLIENT ----}>
- - - - - Please login - - - - -
login admin admin
[I: ...Sending login... OK! ]
[S: Welcome! ]
metric trabytes
[I: ---Start Message--- ]
2620
[I: ---End Message--- ]
metric histacc
[I: ---Start Message--- ]
4
[I: ---End Message--- ]
exit_
```

\* El ejemplo está basado en la planificación plana del protocolo y luego se implementaron mejoras de UX que no están incluidas en el ejemplo, ya que tiene por objetivo demostrar la dinámica de la conversación y no tendría sentido incluir textos que no aporten al mismo.

## Observaciones a tener en cuenta para desarrollar su propio cliente de J2M2

- Los 4 Bytes de header (type, command, argsq y code) deben estar siempre presentes.
- No es necesario modificar los headers/body en el response si no serán utilizados. Dado que ambas partes deberían únicamente utilizar los headers necesarios para asegurar el correcto funcionamiento. De todas formas se recomienda devolver el mismo header con los datos actualizados.
- El protocolo se maneja por bytes. En particular la correcta forma de utilizarlo los datos es serializar y deserializar en tipo de datos uint8\_t.
- Debe enviar un '\0' luego de cada conjunto de caracteres en mensaje (data).
- Para utilizar comandos metric y config debe estar logueado.
- La primera versión del protocolo solo soporta un usuario admin.

# Problemas

Durante todo el desarrollo nos encontramos con diversos inconvenientes, que por diferentes motivos nos llevaron a deshacer lo hecho y comenzar de cero reiteradas veces, dando como resultado una gran pérdida de tiempo pero también un gran aprendizaje sobre estos problemas encontrados.

Entre los mayores desafíos estuvieron:

## El puntapié inicial

Desde un comienzo logramos visualizar cuál era el objetivo de un proxy http transparente, pero pese a esto la dificultad mayor radica en entender correctamente cuáles eran los pasos a seguir, que había que hacer y por donde empezar. Una vez enfocados en el camino correcto, las primeras implementaciones resultaron muy vagas, poco estables y con problemas de concurrencia y bloqueo. Para poder comenzar a entender íntegramente lo que se solicitaba en la consigna y como solucionar dichos problemas, decidimos hacer varias mesas redondas, muy extensas, donde cada uno de los integrantes del grupo previo al encuentro debía entender perfectamente una sección de las tareas que debíamos realizar y luego en orden, avanzamos explicandonos mutuamente hasta lograr comprender el flujo completo de una request-response http y el papel que los sockets cumplan en dicho ciclo.

Comenzamos por varias semanas integrando nuestros trabajos introductorios. En el transcurso notamos decisiones de diseño que nos iban a dificultar el desarrollo del proxy. Algunos ejemplos de estas son el parseo de request y response a línea, que más tarde fue penosamente descartado por lo difícil que se tornó mantener estados internos del procesamiento y reemplazado por parsers a byte, y el programa inicial de las transformaciones, que funcionaba correctamente como tal, pero a la hora de integrarlo al proxy, en concreto, al selector y su máquina de estados global, dejaba de hacerlo ya que el control granular de las condiciones de *seteo* de descriptores se perdía. Si bien se logró reeimplementar lo último, fue un sacrificio importante para el equipo asimilar estas decisiones y el tiempo trabajado.

## Sockets

Si bien contábamos con la interface desarrollada por la universidad de Berkeley, a simple vista nos resultó muy sencillo comprender el flujo que debían seguir los sockets, desde que se crean hasta que comienzan a escuchar, sin embargo al momento de utilizarlos nos encontramos con la dificultad de manejar sockets que debían ser no bloqueantes y la modalidad de utilización de los mismos para lograr conexiones concurrentes. En este caso lograr una solución parcial nos fue más sencillo dado que contamos con implementaciones realizadas en clase, y con el código documentado. Luego de realizar distintas pruebas y modificar el código, decidimos utilizar `select()` para evitar la concurrencia, lo cual resultó exitoso.

## Readfds - Writefds

En un principio sabíamos que `select()` era el camino para poder lograr concurrencia, a excepción de un integrante los demás no comprendemos cómo operaba `select` y que `roll` cumplieran los descriptors `readfds` y `writefds`. Para solucionarlo, tuvimos como disertante a aquel que sabe utilizar correctamente `select()`, a medida que íbamos comprendiendo cómo se utilizaba íbamos cotejando con la documentación los puntos más débiles/complejos.

Al momento de comenzar la implementación la utilización de `select` era muy sencilla. De todas formas, existen muchas variables que debíamos contemplar para hacer un uso correcto, que no hicimos. Entre ellas y la más importante, nuestro programa reaccionaba de maneras extrañas dado que teníamos una actualización errónea de los `fds` tanto de escritura como lectura, y luego de varios días de investigación y soporte por parte de los profesores de la cátedra dimos con la solución correcta. Con la utilización de *strace* logramos ver en qué momento perdimos control de los descriptors y se actualizaban/desactualizaban de manera correcta/incorrecta.

## El selector

Una de estas elecciones más importantes sin duda fue el selector, luego de tener la implementación más simple en funcionamiento, nos encontramos que al acoplar la solución con nuestra implementación de proxy, nos dimos cuenta que la forma en que habíamos modularizado las tareas no era la óptima, aún así decidimos seguir adelante dado el estadio avanzado en el que nos encontrábamos.

El problema principal se presentó cuando logramos realizar 2 conexiones concurrentes y parsear dos respuestas http diferentes. Estaba claro que el parser hacía uso correcto de las peticiones pero las mismas nunca se concretaban dado que el parser no tenía un estado, y mezclaba ambos request. En un principio lo que decidimos hacer fue tener un mapa que asociaba un `fds` a un estado del parser pero nos encontramos con un sinfín de casos que debían ser tenidos en cuenta, lo que nos resultaba imposible terminar de implementar en tiempo y forma nuestro proxy. Para ello decidimos utilizar la implementación del selector provista por la cátedra la cual nos ofrece la funcionalidad que nosotros necesitamos de manera gratuita.

## Modularización

Sin lugar a duda modularizar un proyecto de estas características no es sencillo y más aún si es la primera vez en que se realiza. Decidimos desde un comienzo formar grupos de dos integrantes y comenzar a trabajar de a pares, lo que resultó muy útil porque nos permitió organizar nuestros esquemas horarios más acordes a otra persona en lugar de un grupo de cuatro integrantes, no obstante cada 3 días se informaba del estatus en cual nos encontrábamos y si necesitábamos apoyo extra en determinada tarea, por parte de otros integrantes. De todas formas coincidimos de forma unánime que la distribución de tareas no fue la óptima, dado que la realización de un proxy y un protocolo, consta de muchas áreas grises que si bien la mayoría están determinadas por un RFC (en el caso de HTTP) hay otras donde es decisión del programador tomar cierto camino por sobre otro (más aún en nuestro protocolo). Esas decisiones que parecían correctas desde un punto de vista, podrían no serlo. Allí comenzaron los conflictos de inconsistencia entre los desarrollos de a pares. Para solucionarlas nos pusimos de acuerdo que al momento de tomar una decisión que podría poner en riesgo la funcionalidad del programa, se anunciará y previera las

operaciones que podrían resultar afectadas con el fin de reducir el trabajo a aquel que se encuentra trabajando sobre dicha funcionalidad.

### **Parseo bytes**

Complicaciones referentes al lenguaje C no propias de los conceptos de protocolos de comunicación.

### **Envío de datos por byte en vez de por texto**

Nos resultó muy complicado poder abstraernos de HTTP para poder desarrollar nuestro protocolo. El problema fue que nos “atamos” a un protocolo textual, que si bien su comprensión es más sencilla, en realidad al momento de parsear request/responses no era tan sencillo, y requiere de un poder declarativo por parte del protocolo más amplio, y que contemple más casos. Luego de debatirlo con la cátedra llegamos a la conclusión de que optar por un protocolo binario, podría bajar el nivel de dificultad en cuanto al parseo y podríamos ofrecer la misma funcionalidad. Una vez determinado esto, al basarse en SCTP nuestro protocolo facilita el manejo de datos, dado que el envío se realiza mediante datagramas. Ya definido el protocolo, el problema surge al momento de establecer un canal en el cual ambas partes (tanto cliente como servidor) puedan levantar esa información de manera consistente. Para ello optamos por la utilización de bytes que en C se representan como `uint8_t`.

### **Ultima porcion a transformar con 'sed'**

El comando `sed` tiene una opción `'--unbuffered'`, esta setea en `sed` la configuración de no esperar un buffer de bytes para realizar la transformación y evolución de los datos.

En nuestro caso, cuando solicitamos transformar la última sección (que de por sí tiene un tamaño menor a los anteriores en caso de que no sea múltiplo de la cantidad especificada), surgió el problema de que cuando la opción comentada no esta activa, se espera más bytes para poder transformar y devolvernos los datos requeridos.

# Limitaciones

## **Transformación byte por byte**

Por como está implementado el sistema de transformación, que espera enviar la misma cantidad que recibió, en esta primera versión del desarrollo no se permiten transformaciones que devuelvan mas bytes de los transformados.

## **Transformaciones pequeñas**

No está diseñado para realizar transformaciones de pocos caracteres, debido al problema del buffer de sed explicado en la sección problemas.

## **Múltiples request en la misma línea**

No está implementado, por lo cual, se admite de a una request por vez.

## **SCTP en Mac**

Este sistema requiere de librerías que contemplen las funciones necesarias para manejar protocolo SCTP, sin embargo descubrimos que Mac no las posee por default. Es importante que esto sea tenido en cuenta a la hora de querer ejecutarlo.

# Posibles extensiones

## Almacenamiento de usuarios y contraseñas

Hoy en día en nuestra implementación contamos con un único usuario y contraseña que nos permiten acceder al protocolo J2M2 para obtener métricas. Esta implementación fue provista únicamente para evitar que cualquiera pueda acceder a las métricas del proxy, de todas formas en un futuro podría implementarse un sistema de usuarios con roles para moderar el acceso a los datos.

## Implementación de tokens

Hoy en día una vez que el usuario se encuentra logueado puede acceder a los datos sin restricciones temporales, ni con ningún otro chequeo más que usuario y password. Para aumentar los niveles de seguridad podríamos intentar enviar en cada pedido un token con el objetivo de evitar violaciones de seguridad en el server.

## Nuevas métricas

Hoy en día las métricas toman un papel muy importante en diversos aspectos de un producto, en especial cuando un producto es nuevo, como en nuestro caso. Las métricas que podrían implementarse podrían ser referentes a los tiempos de respuesta de las response/request esto daría la posibilidad de cuantificar la eficiencia temporal de nuestro programa. La cantidad de clientes distintos con que el proxy establece conexión dando un panorama de cuál es el uso que el cliente le está dando al proxy. El número de veces que el proxy debió rechazar conexiones por superar su máximo, esto permitiría apreciar si las características son adecuadas o no frente al uso.

## Nuevas configuraciones

Como teníamos previsto en la presentación oral, entendemos que casi todas las variables involucradas pueden ser modificadas (ya que trabajamos en un proceso donde select elige qué operaciones realizar), por lo cual, es una de las posibilidades a tener en cuenta para extender el producto.

## Nuevos comandos HTTP

Implementar PUT, DELETE y SEARCH, entre otros.

# Conclusiones

## Aprendizaje

**"No fracasé, sólo descubrí 999 maneras de cómo no hacer una bombilla." Thomas Edison**

Creemos que la prueba y error nos ayudó mucho en nuestra formación, cada problema que nos encontramos nos dejó algo nuevo, y si bien tomó mucho tiempo ineficientemente, logramos solucionarlos para conseguir un producto final completo.

## Escalabilidad

En este trabajo, logramos entre otras cosas, diseñar un protocolo en corto tiempo, orientado a manejar nuestro proxy y documentado de tal forma que cualquier compañero o profesor pueda utilizarlo sin nuestra ayuda y a su vez, desarrollamos un proxy que parsea la información recibida.

Entendemos entonces, el potencial que tiene esto en el ámbito laboral, ya que se puede aplicar en cualquier industria como por ejemplo la bancaria, donde es clave formar un protocolo seguro entre ATM o un proxy en una universidad para evitar que los alumnos entren en sitios no adecuados.

## Conceptos Técnicos

Logramos, cada uno desde sus tareas y como grupo tratando de solucionar problemas en conjunto, absorber conocimientos técnicos sobre protocolos, programación en sockets, conexiones concurrentes no bloqueantes, utilización de librerías estándares, RFCs, mejoramos nuestras habilidades con el lenguaje C incluyendo herramientas como Valgrind, Strace o otras como Jmeter para evaluar nuestros desarrollos.

## Ejemplos de prueba

Creamos un archivo **tests.sh** que contiene todos los tests básicos propuestos por la cátedra y algunos adicionales propuestos por el grupo, que se ejecutan continuamente ahorrando así tiempo en testing, separando los resultados de los mismos para verificar el correcto funcionamiento del sistema.

## Testeos básicos propuestos por la cátedra

Se probó el sistema con todos los tests básicos propuestos por la cátedra (adjuntos en anexo) de los cuales se concluyó:

Pasaron todos los tests exceptuando:

1. Response Byte a Byte
2. Múltiples pedidos

En el repositorio se puede encontrar el archivo tests.sh que simula gran parte del guión de pruebas que se puede correr iterativamente.

## Testeos adicionales

### Posteo de caracteres chinos

```
curl -x localhost:8080 --data-binary '卡拉科特' -X POST  
http://localhost:9090/$
```

### Metodo SEARCH (No implementado, aplica para cualquier otro método HTTP no implementado)

```
curl -x localhost:8080 --data-binary '卡拉科特' -X POST  
http://localhost:9090/$
```

Para testeos de performance vease seccion Performance.



# Performance

Hemos utilizado JMeter para realizar pruebas de stress y performance, aquí se describen los resultados obtenidos.

## 50 threads - GET de 10Mb, completado satisfactoriamente.



Se corrió el 'GET' 1051 veces, concluyendo un Throughput cumpliendo un gráfico cuasi-logarítmico, esto tiene sentido, ya que al inicio es cuando recién se empiezan a conectar los clientes y no se demuestra todo el potencial, luego por su puesto se estabiliza hasta nuevos eventos.

## 50 threads - POST de 10Mb, completado satisfactoriamente.



En esta prueba se puede ver un resultado parecido (lógicamente) al GET, también demuestra una curva logarítmica por el mismo motivo. Esta vez corrido desde un disco ssd para notar la diferencia, teniendo en cuenta además que el GET requiere más procesamiento de parsing que el POST, lo cual de por sí ya implica mayor tiempo.

# Guia de instalacion

Esta guía provee los pasos a seguir para una instalación predeterminada, en caso de querer configurar algunos elementos como puerto y host, se debe entender este punto del documento y pasar al siguiente para comprender las distintas configuraciones posibles. Además, en el repositorio puede encontrar un archivo README.md con la información necesaria para ejecutar el sistema de forma resumida.

## Repositorio

En primer lugar se debe descargar o clonar el repositorio provisto por la universidad. Habiéndose situado en la carpeta donde se quiere depositar el proyecto:

```
git clone git@bitbucket.org:itba/pc-2018-01.git
```

## Compilación

En segundo lugar se debe compilar todo el código en cuestión, para ello, hemos desarrollado un Makefile que facilita la tarea e incluye flags pertinentes para obtener errores y alertas más puntillosas.

Habiéndose situado en la carpeta previamente descargada o clonada:

```
make clean all
```

## Ejecución

En último lugar se debe ejecutar los archivos ejecutables producto de la compilación previa.

Habiéndose situado en la carpeta que previamente se descargo o clono:

## Proxy Server

En primer lugar se debe ejecutar el servidor para poder aceptar peticiones correspondientes de otros aplicativos.

```
./httpd
```

## Cliente SCTP

Si se desea ejecutar el cliente SCTP, esto se debe hacer posterior a la ejecución del servidor.

```
./sctpclnt
```



# Instrucciones de configuración

## Valores Default

Proxy Port	8080
SCTP Port	9090

## Proxy

Una vez instalado y habiéndose situado en la carpeta del mismo:

```
./httpd [parámetros]
```

Donde los parámetros respetan el modelo provisto por la cátedra que se adjunta en anexos de este documento: Manual **httpd.8**.

## Cliente HTTP

Para utilizar el proxy HTTP, no se desarrolló un nuevo cliente, teniendo en cuenta el enunciado del trabajo y la amplia oferta de estos en el mercado (como CURL o Chrome). Ejemplo:

```
nc [host] [port]
```

## Cliente SCTP

Una vez instalado el cliente y habiéndose situado en la carpeta del mismo:

```
./sctpc1nt [host] [port]
```

Donde [host] y [port] son parámetros opcionales que pueden ser utilizados de la siguiente manera:

- Host y Port: Si el servidor se encuentra ejecutando en un host específico con un puerto específico
- Port: Si el servidor se encuentra ejecutando en el server host por default pero en un puerto específico
- Sin parámetros: El servidor se encuentra ejecutando con los valores por defecto

# Ejemplos de uso

Una vez el proxy instalado, configurado y ejecutado.

## Proxy

### Mediante cURL

GET basico a **google.com**

```
curl -x [proxy_host]:[proxy_port] google.com
```

### Mediante User Agent

En primer lugar se debe configurar el proxy en las configuraciones del user agent.

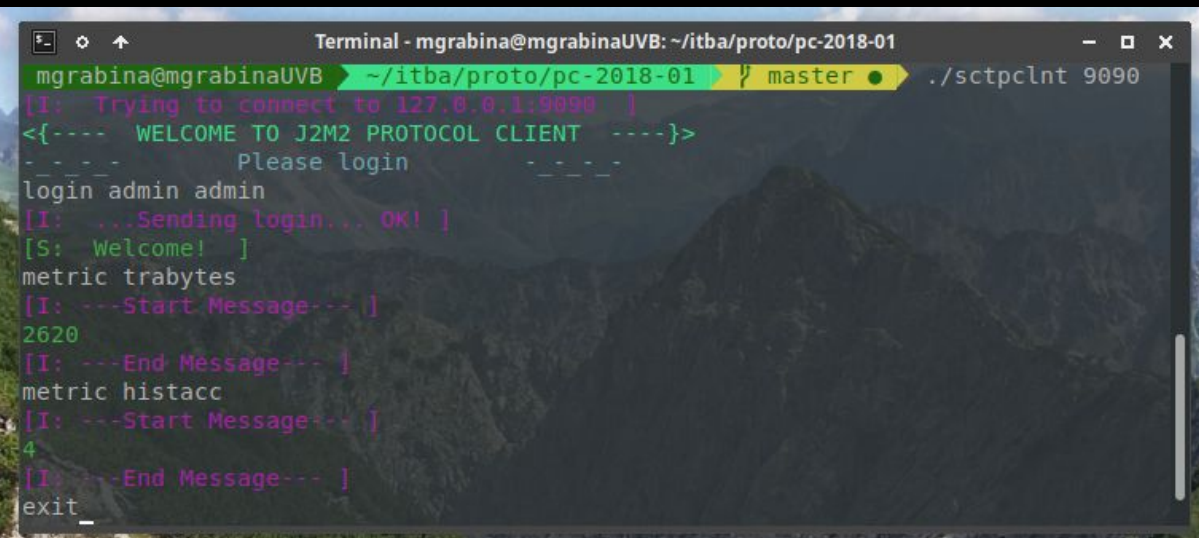
En Google Chrome: Configuraciones->Configuraciones Avanzadas -> Configuraciones de proxy -> Seguir los pasos propuestos para ingresar los datos del proxy (host, puerto, etc.)  
Luego simplemente navegar.

```
google.com
```

## Cliente SCTP

```
./sctpcInt [sctp_host] [sctp_port]
```

Luego comienza la "conversación":



```
Terminal - mgrabina@mgrabinaUVB: ~/itba/proto/pc-2018-01
mgrabina@mgrabinaUVB ~/itba/proto/pc-2018-01 master ➤ ./sctpcInt 9090
[I: Trying to connect to 127.0.0.1:9090 ]
<{---- WELCOME TO J2M2 PROTOCOL CLIENT ----}>
-_-_- Please login -_-_-
login admin admin
[I: ...Sending login... OK! ]
[S: Welcome! ]
metric trabytes
[I: ---Start Message--- ]
2620
[I: ---End Message--- ]
metric histacc
[I: ---Start Message--- ]
4
[I: ---End Message--- ]
exit
```

# Documento de diseño de proyecto

## Estructura del repositorio y archivos destacados

- ❑ pc-2018-5
  - ❑ **main.c**  
Encargado de crear los sockets e inicializar el selector para poner en funcionamiento el sistema.
  - ❑ sctp
    - ❑ **sctp client and server files**  
sctpintegration.c  
sctpclnt.c
  - ❑ **http proxy**  
httpnio.c  
request.c  
response.c  
media\_types.c  
selector.c  
http.c  
parameters.c

Se crearon librerías propias para modularizar el sistema y mejorar la UX del mismo, como por ejemplo:

### I. **Styles.h**

Provee strings definidos con formatos de colores y encabezados para la implementación del cliente sctp de manera más “amigable” con el usuario.

### II. **http\_codes.h**

Provee strings definidos de respuestas a todos los códigos http, inclusive aquellos que el sistema no abarca hoy en día, con el objetivo de escalar de manera más sencilla: Todos los códigos de respuesta ya están implementados para su uso y no hace falta, por ejemplo, utilizar la opción -i en curl ya que esta implementación envía el texto del código también como parte del body de la respuesta para informar al usuario.

## Makefile

Si bien también utilizamos CMake, para compilar en CLion, nuestra **compilación por defecto es mediante el makefile**.

Esta se teado los flags a gusto y modularizado para que sea más sencillo agregar o quitar elementos.

Se tuvo en cuenta el caso de hacer dos veces seguidas *make clean* donde en la primera se eliminan los archivos ejecutables y por ende, en la segunda devuelve error, simplemente incluyendo un -f en el comando de *remove*.

Se incluyeron *hecho* que funcionan como avisos al usuario de que está sucediendo y se ocultaron los comandos de compilación y eliminación, ya que al ser muy extensos y llenos de elementos se dificulta la lectura y “ensucia” la pantalla. De esta forma se sabe el estado actual de la acción mediante títulos personalizados.

## Estadísticas del repositorio

Se adjunto al repositorio un resumen del mismo en la carpeta **git\_stats**, generado con el comando **gitstats** desarrollado por **Heikki Hokkanen** y su equipo.

Este es una página html básica con gráficos que reflejan el comportamiento del repositorio en detalle a lo largo del tiempo.

Vale la pena aclarar, que algunas métricas no deben ser consideradas al pie de la letra, ya que por ejemplo, el equipo se juntó en reiteradas veces donde no necesariamente cada integrante trabajaba en su computadora de manera individual.

## Bibliografía y librerías utilizadas

- Regex
- SCTP
- Selector de la cátedra
- Internet (osdev, stack overflow, etc.)
- Libro recomendado por la cátedra



# Anexo I

## Manual httpd.8

### NAME

httpd - proxy HTTP que permite transformar el cuerpo de las respuestas

### SINOPSIS

httpd [ POSIX style options ]

### OPCIONES

-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.

-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 <<https://tools.ietf.org/html/rfc7231#section-5.3.2>>).

Por defecto la lista se encuentra vacía.

Por ejemplo el valor 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. Compatible con system(3). La sección FILTROS describe cómo es la interacción entre httpd(8) y el comando filtro. Por defecto no se aplica ninguna transformación.

-v

Imprime información sobre la versión versión y termina.

## FILTROS

Por cada respuesta del origin server de status code 200 que contenga un body (no HEAD) y que tenga un Content-Type compatible con los del predicho, se lanza un nuevo proceso que ejecuta el comando externo. Si el intento de ejecutar el comando externo falla se debe reportar el error al administrador por los logs, y copiar la entrada en la salida (es decir no realizar ninguna transformación).

El nuevo proceso recibe por entrada estándar el contenido del body de la respuesta (libre de cualquier transfer-encoding), y retorna por la salida estándar el cuerpo procesado.

Los programas que realizan las transformaciones externas tienen a su disposición las siguientes variables de entornos:

### HTTPD\_VERSION

Versión de httpd. Por ejemplo: 0.0.0.

## EJEMPLOS

- Se desea procesar sin ninguna transformación:  
httpd
- En Linux es posible redirigir el tráfico al proxy de forma transparente con una regla que implemente destination NAT:  
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to 8080

25 de Mayo 2018

HTTPP(0.0.0)

## Anexo II

### Testeos provistos por la cátedra con sus respectivos puntajes

#### Caso Puntaje

200 - Binding en puertos default correcto 0.05  
201 - Cambio de puertos e interfaces default 0.05  
000 - Performance y inmutabilidad de bytes 0.2  
001 - Concurrencia 0.2  
002 - GET Respuestas 204 0.2  
003 - Soporte método POST básico 0.2  
004 - Soporte método POST grande 0.25  
005 - Soporte método HEAD 0.2  
006 - Soporte GET condicional 0.2  
007 - Múltiples request en la misma línea 0.2  
008 - Conexión a origin server no resoluble 0.15  
009 - Conexión a origin server no disponible 0.15  
010 - El proxy como Origin Server 0.15  
011 - Origin server deja de estar en línea durante transferencia de respuesta 0.15  
012 - Origin server deja de estar en línea durante transferencia de request 0.15  
113 - L33t con Content-Length 0.2  
114 - L33t Chunked 0.2  
115 - L33t Chunked gzipped 0.2  
300 - Lecturas parciales del request 0.5  
301 - Lecturas parciales del response 0.5  
Total 4

**Para información más detallada de cada test, consultar la documentación oficial en material didáctico del curso.**

[https://campus.itba.edu.ar/bbcswebdav/pid-104452-dt-content-rid-287962\\_1/courses/72.07-22459/gui%C3%B3n-definitivo-r0.pdf](https://campus.itba.edu.ar/bbcswebdav/pid-104452-dt-content-rid-287962_1/courses/72.07-22459/gui%C3%B3n-definitivo-r0.pdf)