Trabajo Práctico Especial

Protocolos de Comunicación - 1er Cuatrimestre 2020

Grupo 5

Gonzalo Hirsch - 59089 (ghirsch@itba.edu.ar)
Ignacio Ribas - 59442 (iribas@itba.edu.ar)
Augusto Henestrosa - 59189 (ahenestrosa@itba.edu.ar)
Francisco Choi - 59285 (fchoi@itba.edu.ar)

Abstracto

Este documento describe el desarrollo del Trabajo Práctico Especial para la materia Protocolos de Comunicación durante el primer cuatrimestre del año 2020. La aplicación desarrollada es un servidor proxy que implementa el protocolo SOCKSv5[RFC1928] y sus funcionalidades asociadas, junto con la implementación y creación de un protocolo propio del grupo que funciona sobre SCTP para la administración del servidor mismo.

Índice

Introducción

El objetivo de este documento es explicar la funcionalidad y el proceso de desarrollo para el servidor proxy socksv5, siguiendo el RFC 1928 requerido por la consigna del trabajo práctico, junto con las funcionalidades adicionales requeridas.

Estas funcionalidades incluyen el desarrollo de un protocolo propio que corra sobre SCTP, y un cliente que implemente este protocolo, el soporte de autenticación con usuario y contraseña siguiendo el RFC 1929, monitoreo del tráfico y resoluciones DNS sobre HTTP siguiendo el RFC 8484, entre otras funcionalidades explicadas con más profundidad en las siguientes secciones del documento.

El desarrollo del servidor fue realizado con un enfoque paralelo respecto de la realización de los componentes claves para poder maximizar el tiempo de desarrollo. El éxito al implementar este enfoque de trabajo implica que el servidor es un sistema altamente modularizado al que se le pueden agregar módulos con facilidad para poder extender su funcionalidad.

Desarrollo

Proxy

Se desarrolló un proxy socksv5 transparente que contiene varios módulos individuales requeridos por la consigna del trabajo práctico. Estos módulos son los siguientes:

Autenticación

Autenticación mediante usuario y contraseña siguiendo el RFC 1929. El usuario del proxy debe estar autenticado, y para hacer esto, al conectarse al servidor entra en una negociación con el mismo para definir el método de autenticación utilizado. El servidor en sí soporta el uso sin usuario y contraseña, pero por propósitos de la consigna, el único método habilitado de autenticación hace uso del usuario y la contraseña.

DNS sobre HTTP

Resolución DNS sobre HTTP siguiendo el RFC 8484. Cuando se utiliza el nombre de un dominio para realizar una conexión con el servidor, el mismo utiliza un servidor DNS, provisto por al momento de iniciar el servidor mediante la línea de comandos, que realice resoluciones sobre HTTP. Para realizar estas resoluciones se hacen requests GET siguiendo los estándares respecto de codificación establecidos en el RFC mencionado. Estas resoluciones están apuntadas a obtener IPs versión 4 y 6, para utilizar alguno de estos al realizar la conexión y poder tener más opciones en términos de direcciones para intentar.

Para optimizar las conexiones al hacer las consultas DNS para obtener IPv4 e IPv6, se implementó *pipelining* en la conexión con el servidor DNS. De esta forma podemos optimizar significativamente el tiempo de espera de respuesta ante la consulta DNS, ya que al mandarse las dos consultas seguidas, se evita tener que repetir el ciclo de envio y recepcion.

Para parsear el formato de tanto de las request como de la respuesta del protocolo se utilizaron estructuras que asimilan byte a byte cada campo que indica el RFC. Por otra parte, aunque para el trabajo práctico no fue necesario, se preparó el cliente para poder recibir diferentes tipos de respuestas por parte del servidor para ser más robustos.

Para todos los dominios se hacen consultas DNS para obtener nos IPs correspondientes IPv4 e IPv6, y luego se intenta con cada uno de ellos establecer la conexión hasta que alguno de una conexión exitosa.

Métricas de Uso

Recolección de métricas de uso. Se implementó una interfaz de recolección de métricas volátiles de uso del servidor, se cuentan la cantidad de bytes transferidos, junto con la cantidad de conexiones históricas y la cantidad de conexiones concurrentes. Estas métricas pueden ser accedidas a través del protocolo de administración y un cliente que lo implemente. Es importante notar que estas métricas son volátiles para simplificar el proceso de logueo y no generar operaciones bloqueantes ni demasiadas aperturas de archivos.

Registro de Acceso

Registro de acceso. Se implementó un registro de acceso para el tráfico del servidor, para indicar los destinos a los que se intentan conectar los usuarios, este registro distingue entre los distintos usuarios mediante el nombre del usuario que se utiliza para conectarse.

Monitoreo de Tráfico y Registro de Credenciales

Monitoreo de tráfico y registro de credenciales. Se implementó una serie de disectores de contraseñas para poder extraer las mismas para los protocolos HTTP y POP3. La heurística utilizada para la extracción de usuarios y contraseñas es la siguiente, se ve el puerto al que se quiere conectar al usuario, el 80 en el caso de HTTP y el 110 en el caso de POP3. A partir de esa información define qué parser usar para analizar la información que envía el usuario, esa es la primer verificación que se hace. En el caso de HTTP se busca el header *Authorization* del tipo basic, para poder decodificar el identificador en base64, que contiene el usuario y la contraseña en texto plano, ignorando el cuerpo de la request y otros headers. En el caso de POP3 se busca el texto “USER” y “PASS” que se envía en la request de POP3, ignorando luego el resto de la request.

Monitoreo y Configuración Remota

Mecanismos de monitoreo y configuración remota del servidor. Se creó un protocolo basado sobre SCTP que permita a un usuario conectarse remotamente para obtener métricas y cambios de configuraciones remotas sin necesidad de frenar la ejecución del servidor. A continuación se encuentra una descripción más detallada del mismo.

Protocolo IGAF

El protocolo IGAF (Ignacio - Gonzalo - Augusto - Francisco) fue desarrollado con el propósito de administración del proxy y configuración del mismo. El mismo es un protocolo binario, y esta decisión fue en torno a hacer que los paquetes de requests sean livianos y fáciles de analizar.

Este protocolo corre sobre SCTP (Stream Control Transmission Protocol) definido en el RFC 4960, y usa autenticación mediante usuario y contraseña definida en el RFC 1929. La

autenticación para este protocolo hace uso de credenciales persistidas sobre un archivo de texto dentro de una carpeta utilizada para guardar datos del servidor. Si bien estas credenciales se persisten, lo usuarios creados mediante este protocolo son volátiles. Esta decisión fue tomada por el simple hecho de hacer que el proceso sea más simple y evitar operaciones bloqueantes sobre archivos de texto.

Este protocolo corre bajo el principio una request y una response correspondiente, es decir que cada request del cliente tiene una response asociada.

En la posterior definición todos los números que hacen referencia a los campos de un paquete hacen referencia a la longitud del campo en bytes.

Se definen también en general los códigos de errores posibles, siendo los valores los siguientes:

- X'00' - Éxito, sin error
- X'01' - Error general del servidor
- X'02' - Campo TYPE inválido
- X'03' - Campo CMD inválido
- X'04' - Datos enviados inválidos
- X'05' - Datos numéricos enviados inválidos

Autenticación

Lo primero que debe hacer un cliente al conectarse a un servidor que utilice este protocolo es autenticarse. Luego de estar autenticado puede comenzar a intercambiar requests y responses con el servidor para poder hacer uso de todas sus funcionalidades

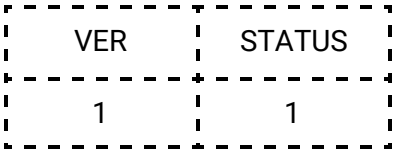
Como fue mencionado antes, la autenticación se realiza con usuario y contraseña siguiendo el RFC 1929, esto significa que el paquete de autenticación es el siguiente:

VER	ULEN	UNAME	PLEN	PASSWD
1	1	1 to 255	1	1 to 255

Los campos de este paquete son:

- VER - Versión de la negociación, X'01'
- ULEN - Longitud del nombre del usuario
- UNAME - Nombre del usuario
- PLEN - Longitud de la Contraseña
- PASSWD - contraseña

A esta request el servidor responde con el siguiente paquete:



Los campos de este paquete son:

- VER - Versión de la negociación, X'01'
- STATUS - Campo de estado definido anteriormente

Uso

Para el uso general del protocolo, se definieron 2 campos que están presentes en todas las requests, el TYPE y el CMD. El TYPE indica el dominio de la request mientras que el CMD indica el comando a realizar, esto permite que el protocolo sea más escalable y más flexible. Se aclara que en esta versión del protocolo, no todas las combinaciones de TYPE y CMD resultan en requests exitosas, no están definidas todas esas combinaciones.

En la versión 1 del protocolo se definieron 3 valores distintos de TYPE:

- USUARIOS - X'01'
- METRICAS - X'02'
- CONFIGURACIONES - X'03

También se definieron 3 valores distintos de CMD:

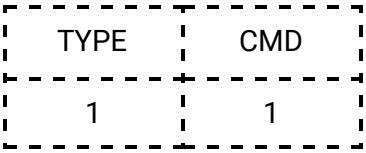
- LISTAR - X'01'
- CREAR - X'02'
- EDITAR - X'03'

Usuarios

Dentro del dominio de usuarios se definieron dos comandos aceptados, LISTAR y CREAR, las requests correspondientes y sus responses asociadas son las siguientes.

Listado de Usuarios

Se definió el listado de usuarios de forma que sea lo más intuitiva y fácil posible la request, el paquete es el siguiente:



Los campos de este paquete son:

- TYPE - Dominio de la request, X'01' (USUARIOS)
- CMD - Comando a realizar, X'01' (LISTAR)

La respuesta del servidor para esta request es la siguiente:

TYPE	CMD	STATUS	ULEN	DATA
1	1	1	4	Variable

Los campos de este paquete son:

- TYPE - Dominio de la request, X'01' (USUARIOS)
- CMD - Comando a realizar, X'01' (LISTAR)
- STATUS - Campo de estado definido anteriormente
- ULEN - Cantidad de usuarios a devolver (BigEndian)
- DATA - Nombres de los usuarios, separados por un byte en 0 (X'00')

Creación de Usuarios

Se definió la creación de usuarios de una forma muy similar a la autenticación del RFC 1929, el paquete es el siguiente:

TYPE	CMD	VER	ULEN	UNAME	PLEN	PASSWD
1	1	1	1	1 to 255	1	1 to 255

Los campos de este paquete son:

- TYPE - Dominio de la request, X'01' (USUARIOS)
- CMD - Comando a realizar, X'02' (CREAR)
- VER - Versión de la negociación, X'01'
- ULEN - Longitud del nombre del usuario
- UNAME - Nombre del usuario
- PLEN - Longitud de la Contraseña
- PASSWD - contraseña

La respuesta del servidor para esta request es la siguiente:

TYPE	CMD	VER	STATUS
1	1	1	1

Los campos de este paquete son:

- TYPE - Dominio de la request, X'01' (USUARIOS)
- CMD - Comando a realizar, X'02' (CREAR)
- VER - Versión de la negociación, X'01'
- STATUS - Campo de estado definido anteriormente

Métricas

Dentro del dominio de las métricas se definió un solo comando aceptado, LISTAR, la request correspondiente y su response asociada es la siguiente.

Listado de Métricas

Se definió el listado de métricas de forma que sea lo más intuitiva y fácil posible la request, el paquete es el siguiente:

TYPE	CMD
1	1

Los campos de este paquete son:

- TYPE - Dominio de la request, X'02' (METRICAS)
- CMD - Comando a realizar, X'01' (LISTAR)

La respuesta del servidor para esta request es la siguiente:

TYPE	CMD	STATUS	MLEN	TBYTES	HCONN	CCCONN
1	1	1	1	8	4	4

Los campos de este paquete son:

- TYPE - Dominio de la request, X'02' (METRICAS)
- CMD - Comando a realizar, X'01' (LISTAR)
- STATUS - Campo de estado definido anteriormente
- MLEN - Cantidad de métricas a devolver
- TBYTES -> Cantidad de bytes transferidos históricamente (Big Endian)
- HCONN -> Cantidad de conexiones históricas (Big Endian)
- CCONN -> Cantidad de conexiones concurrentes (Big Endian)

Configuraciones

Para el dominio de configuraciones remotas se definieron dos comandos aceptados, LISTAR y EDITAR, las requests correspondientes y las responses asociadas son las siguientes.

Listado de Configuraciones

Se definió el listado de configuraciones de forma que sea lo más intuitiva y fácil posible la request, el paquete es el siguiente:

TYPE	CMD
1	1

Los campos de este paquete son:

- TYPE - Dominio de la request, X’03’ (CONFIGURACIONES)
- CMD - Comando a realizar, X’01’ (LISTAR)

La respuesta del servidor para esta request es la siguiente:

TYPE	CMD	STATUS	CLEN	S5BSIZE	SBSIZE	DISCT
1	1	1	1	2	2	1

Los campos de este paquete son:

- TYPE - Dominio de la request, X’03’ (CONFIGURACIONES)
- CMD - Comando a realizar, X’01’ (LISTAR)
- STATUS - Campo de estado definido anteriormente
- CLEN - Cantidad de métricas a devolver
- S5BSIZE - Tamaño del buffer para el proxy socks5 (Big Endian)
- SBSIZE - Tamaño del buffer de la conexión SCTP (Big Endian)
- DISCT -> Estado de los disectores
    - Activos - X’01’
    - Inactivos - X’00’

Edición de Configuraciones

Para la edición de configuraciones se definió el siguiente formato de request:

TYPE	CMD	CONF	VALUE
1	1	1	Variable

Los campos de este paquete son:

- TYPE - Dominio de la request, X’03’ (CONFIGURACIONES)
- CMD - Comando a realizar, X’02’ (EDITAR)
- CONF - Número de configuración a editar
    - S5BSIZE - Tamaño del buffer para el proxy socks5 - X’01’
    - SBSIZE - Tamaño del buffer de la conexión SCTP - X’02’
    - DISCT -> Estado de los disectores - X’03’
- VALUE - Valor de la nueva configuración, tamaño depende del campo
    - S5BSIZE -> 2 bytes (BigEndian) - Unidad: Bytes

- SBSIZE -> 2 bytes (BigEndian) - Unidad: Bytes
- DISCT -> 1 byte - X'00'(desactivar) o X'01'(activar)

La respuesta del servidor para esta request es la siguiente:

TYPE	CMD	STATUS	CONF
1	1	1	1

Los campos de este paquete son:

- TYPE - Dominio de la request, X'03' (CONFIGURACIONES)
- CMD - Comando a realizar, X'02' (EDITAR)
- STATUS - Campo de estado definido anteriormente
- CONF - Número de configuración editada
  - S5BSIZE - Tamaño del buffer para el proxy socks5 - X'01'
  - SBSIZE - Tamaño del buffer de la conexión SCTP - X'02'
  - DISCT -> Estado de los disectores - X'03'

Errores

Si bien cada paquete de respuesta asociado a su request correspondiente informa de errores, en el caso de que se reciba un TYPE o CMD que no sea apropiado o sea una combinación inválida, se devuelve un paquete con el siguiente formato.

TYPE	CMD	STATUS
1	1	1

Los campos de este paquete son:

- TYPE - Dominio de la request, reservado en X'00'
- CMD - Comando a realizar, reservado en X'00'
- STATUS - Campo de estado definido anteriormente

Cliente

La implementación del servidor proxy incluye una implementación de un cliente para el protocolo IGAF. A este cliente se le puede especificar el puerto e IP en el que se encuentra el servidor, para simplificar su conexión al mismo. Este cliente fue desarrollado en modo interactivo, de forma que se pueden realizar distintas pruebas de configuración sin necesidad de cortar la ejecución del cliente para volver a poner otras opciones. Al no ser un protocolo de texto, la decisión de hacerlo interactivo vino de la mano de hacer que sea interesante realizar

pruebas y tener la conexión al cliente de administración al mismo tiempo, para poder ver cambios en tiempo real en las métricas o cambios en las respuestas del servidor.

Las instrucciones para compilar y ejecutar al mismo se encuentran en el *README* en el repositorio de GIT.

Problemas

Durante todo el tiempo de desarrollo surgieron diversos problemas, algunos no tan severos como otros, pero en la mayoría de los casos estos problemas llevan a tener que reescribir grandes segmentos de código y bastante tiempo de pruebas. Si bien fueron problemas que hubiera sido conveniente evitar, la verdad es que nos hicieron entender aún más cómo funcionaba el servidor.

Durante el Desarrollo

Inicio
Uno de los principales problemas fue como comenzar con el desarrollo, nos fueron provistas distintas porciones de código, muy útiles aclaro, que no sabíamos cómo funcionaban. Tampoco sabíamos por dónde empezar, comenzamos con los parsers, pero luego nos dimos cuenta que no íbamos a llegar muy lejos si primero no teníamos un servidor andando. Con esto comenzó la ardua tarea de tratar de entender todo el código provisto por la cátedra, para poder comenzar a armar nuestro servidor y poder comenzar a integrar nuestras distintas partes. De esta forma pudimos comenzar a construir el servidor desde el principio e ir agregando funcionalidades, y de esta forma se agilizó mucho más el desarrollo.

DoH
Al comenzar las pruebas con el DoH, no sabíamos que teníamos que configurar un servidor NGINX para poder realizar las pruebas. No estábamos al tanto de que los servidores DNS no atienden consultas por HTTP. Esto nos forzó a investigar sobre esto para poder realizar todas las pruebas necesarias. El servidor NGINX simplemente hacia un forward de la request al DNS de Cloudflare para poder realizar las pruebas de resolución.

SCTP
Cuando intentamos comenzar a implementar el protocolo sobre SCTP y el cliente SCTP, vimos que hacían falta librerías y que con las estándar no era suficiente. Esto generó problemas para los integrantes del grupo que estaban programando en MACs con OSx, porque no encontramos librerías equivalentes que no implicaran cambios en todos los archivos header para poder correrlo. Esto nos llevó a siempre tener que compilar en máquinas virtuales, o en el sistema operativo nativo en el caso de alguien que usara Linux puro.

Manejo de Memoria
Ya con el desarrollo encaminado, decidimos encarar el *-fsanitize=address* y *Valgrind,* para poder resolver cualquier problema de manejo de memoria que haya. Pero nos encontramos con que no se podía compilar con el flag *-fsanitize=address* y al mismo tiempo correr Valgrind para verificar que no hubieran leaks de memoria. Esto llevó a tener que realizar todas las pruebas de

funcionamiento dos veces, para ver que no generen errores con la sanitización de direcciones ni con Valgrind.

Modularización
Algo que decidimos que era crítico encarar fue la modularización del código, ya que se volvía difícil de mantener y entender el código de los distintos componentes cuando un archivo tenía más de 1000 líneas. Por eso en distintas etapas del desarrollo tuvimos que frenar un momento para poder tomarnos el tiempo de modularizar el código para poder seguir trabajando al mismo ritmo. Esto eventualmente nos llevó a primero ver todo lo necesario en el código, probarlo para que funcione y luego modularizar para que sea más cómodo encarar algún error.

Durante las Pruebas

JMeter
Durante las pruebas surgieron algunos problemas, uno de ellos fue no poder ejecutar JMeter para poder hacer load testing sobre el servidor. Al ser un servidor proxy socks con autenticación, hubo muchos problemas para lograr que JMeter acepte que nuestro servidor proxy iba a estar corriendo por detrás y que debería primero pasar por nuestro proxy para después pasar por el servidor proxy HTTP que levanta para hacer las pruebas. Una vez que se logró configurar al servidor proxy, no se pudieron configurar las credenciales del servidor ya que tomaba las credenciales del usuario de ubuntu.

Por esta razón decidimos armar una versión de prueba del servidor en la que no se necesite autenticación, ya que era el factor que nos limitaba las pruebas al no poder configurarlo en JMeter. Utilizando esta versión de prueba creada en una rama de testing en el repositorio se pudieron hacer distintas pruebas de stress.

También hubieron problemas al probar con requests más pesadas como la descarga de un archivo, así que se optó por hacer una HTTP request GET estándar para hacer todas las mediciones.

IPv6
Como ningún integrante tenía un IPv6 público otorgado por el ISP, no se pudieron hacer pruebas de conexión IPv6 a servidores externos, todas las pruebas se realizaron de forma local para las conexiones IPv6.

Limitaciones

Proxy

Cantidad de Conexiones
El proxy tiene un límite de 1024 conexiones, aunque en realidad depende de la cantidad de descriptores ocupan los sockets pasivos que se tienen en el caso de usar IPv4 o IPv6, pero el límite del pselect que corre por detrás es 1024, así que como máximo se podrían tener 1024 conexiones concurrentes. En el caso de querer poder soportar más, se debería pasar a un esquema de fork para poder atender a nuevas conexiones.

Disectores
Los disectores de contraseñas solo pueden tomar contraseñas que en el caso de HTTP vengan como Authorization Basic y contraseñas que vengan plain text para POP3.

DoH
El DoH no implementa caching de las resoluciones, así que aunque se hagan varias requests a para un mismo dominio, siempre se va a tener el overhead de tener que resolver el nombre.

También sufre de que no tiene un timeout para definir si una request al servidor DNS hizo timeout, necesita que el protocolo le avise, y por eso dependemos del timeout del protocolo subyacente en ese caso.

Buffer Size
El buffer utilizado en DoH es el mismo que se usa en el proxy, por lo tanto no es configurable los tamaños de los dos buffers por separado.

Administración mediante SCTP

Pool de Objetos
Como el módulo de administración mediante SCTP comparte el mismo selector que el proxy, el límite de conexiones está compartido.

Extensiones

Tenemos algunas posibles extensiones que resultan interesantes de explorar para agregar funcionalidad al servidor.

Proxy

Pool de Objetos
Una mejora en términos de eficiencia sería el uso de pool de objetos para las instancias de los structs de socks5 utilizadas para la entrada no bloqueante. Esto evitaría que se generen demasiadas instancias del struct y ahorraría memoria y haría que sea un poco más eficiente.

Cache Interno de Resoluciones DNS
Una funcionalidad que mejoraría la eficiencia del servidor podría ser el caching de las resoluciones DNS, esto es algo que consume tiempo y recursos, así que poder hacer un caching de las resoluciones junto con un *Time To Live* para determinar si sigue siendo válido sería muy beneficioso.

Credenciales y Logging Persistente
Algo que no llegamos a implementar para no hacer que se degrade la performance fue la persistencia de credenciales y logs de acceso. Esto haría que no se tenga que pasar siempre por línea de comandos los valores de los usuarios y sus contraseñas, haciendo que sea más simple la ejecución.

Timeout
Una extensión que se podría hacer es agregar un timeout a los descriptores para evitar que se creen conexiones ociosas que ocupen la totalidad de los descriptores disponibles. Por una cuestión de tiempo esto no fue implementado en la sección de configuraciones remotas.

Otras funcionalidades SOCKSv5
Se pueden implementar las otras opciones de conexión que provee el protocolo SOCKSv5, que serían el BIND y UDP ASSOCIATE, para poder terminar de implementar el protocolo en todo su potencial. Y al mismo tiempo ofrecer mucha más funcionalidad a los usuarios.

Mejor Extracción de Credenciales
Se podría implementar una mejor estrategia de extracción de credenciales de acceso e información importante para los disectores, ya que al ser una heurística relativamente simple actualmente (pero que cumple su cometido en términos de los estándares de autenticación mediante HTTP y POP3) no puede extraer credenciales que no se pasen en un formato de Authorization Basic en HTTP y el USER y PASS explícito de POP3. Se podría implementar una

heurística que busque dentro de la request HTTP para ver si hay algún campo de usuario o contraseña o ver dentro de la información transferida por el protocolo POP3 en busca de información importante.

Más Métricas Significativas
Otra extensión podría ser el agregado de obtención de más métricas significativas como la cantidad de conexiones exitosas versus la cantidad de conexiones que devuelven un error.

Blacklisting de IPs
Una última extensión que puede resultar muy interesante podría ser el blacklisting de ciertos IPs/dominios y contenido, por ejemplo, no permitir que se generen conexiones a ciertos dominios pornográficos, simulando un proxy que podría utilizarse en una empresa por ejemplo. Esta extensión vendría de la mano del agregado de nuevas métricas que indiquen la cantidad de requests denegadas por este motivo, junto a una extensión del protocolo definido sobre SCTP para permitir la configuración remota de estos dominios IPs.

Administración mediante SCTP

Pool de Objetos
En el caso de la administración mediante SCTP, se hace un comentario similar a la mejora mediante la utilización del pool de objetos para el proxy. El componente de administración mediante SCTP está construido de una forma similar y también podría implementar esta lógica del pool de objetos para una administración más eficiente de la memoria.

Protocolo IGAF

Uso de Streams
Una extensión interesante para el protocolo podría ser aprovechar la funcionalidad de los streams que provee SCTP para hacer un mejor manejo de la información enviada y para poder hacer un envío más eficiente.

Conclusiones

Durante el desarrollo de este trabajo aprendimos muchas cosas, cada problema ayudó a afianzar mejores prácticas y a hacer que aprendamos los conceptos como se debe. Si bien esos problemas nos retrasaron en varios momentos, al final fueron los que definieron el producto final que tenemos, ya que ayudaron a mejorarlo.

Aprendimos lo que implica el desarrollo de un servidor y un protocolo, y también aprendimos que el desarrollo de estas aplicaciones es muy escalable. Una vez que conseguimos tener un core de servidor funcionando, resultaba muy simple (en la mayoría de los casos), integrar y modularizar nuevos componentes. También entendimos la importancia de la buena definición de los protocolos de comunicación, ya que hacen que el desarrollo sea más dinámico y las aplicaciones a las que les brindan servicios, más seguras.

Por último, creemos que el trabajo nos forzó a pasar por la experiencia de buscar la documentación oficial de las librerías, ver los RFCs para poder implementar los protocolos y entender bien cuál es el flujo de desarrollo en estos casos. Ayudó a entender y aplicar todos los conceptos vistos en clase.

Pruebas

Todos los ejecutables generados fueron probados con **Valgrind** utilizando el siguiente comando:

```
valgrind --leak-check=full --trace-children=yes --tool=memcheck
--read-var-info=yes --show-reachable=yes -v --track-fds=yes
--log-file="valgrind-output.txt" <EJECUTABLE>
```

Este comando ejecuta Valgrind sobre el ejecutable que se use, y se utilizaron todos los flags que se muestran ahí para evitar los leaks de memoria, operaciones incorrectas sobre la memoria y mal uso de descriptores.

También se realizaron pruebas con el flag **-fsanitize=address** para evitar el mal uso de direcciones de memoria y corregir errores de ese estilo. Este flag no fue incorporado en la versión entregada del trabajo, ya que tiene un impacto en el uso de memoria y el tiempo de CPU [1].

Máxima Cantidad de Conexiones

Para realizar estas pruebas de stress se utilizó JMeter y se dividió en 2 grupos de threads para las conexiones:

- Grupo A: Grupo de medición, 10 conexiones concurrentes que ciclan durante 50 iteraciones realizando HTTP requests a twitter.com
- Grupo B: Grupo masivo para analizar el cambio del grupo A. Se prueba con 0, 100 y 500 conexiones concurrentes realizando 50 iteraciones de request HTTP a twitter.com para analizar el impacto en la performance del grupo A.

Caso de prueba 1 (Minimo Stress): Grupo A: 10 conexiones, Grupo B: 0 conexiones

Como se puede apreciar en la tabla de resultados, se obtiene un Throughput de aproximadamente 14 requests/ segundo. El tiempo medio y la mediana de respuesta se estabiliza en torno a los 600 ms. Se obtiene un porcentaje de error del 0%.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
HTTP Request	500	607	460	1202	111.80	0.00%	14.1/sec	608.80	44195.0
TOTAL	500	607	460	1202	111.80	0.00%	14.1/sec	608.80	44195.0

Figura 1: Resumen de conexiones con para la prueba 1 (10 conexiones concurrentes)

[1] https://www.usenix.org/system/files/conference/atc12/atc12-final39.pdf

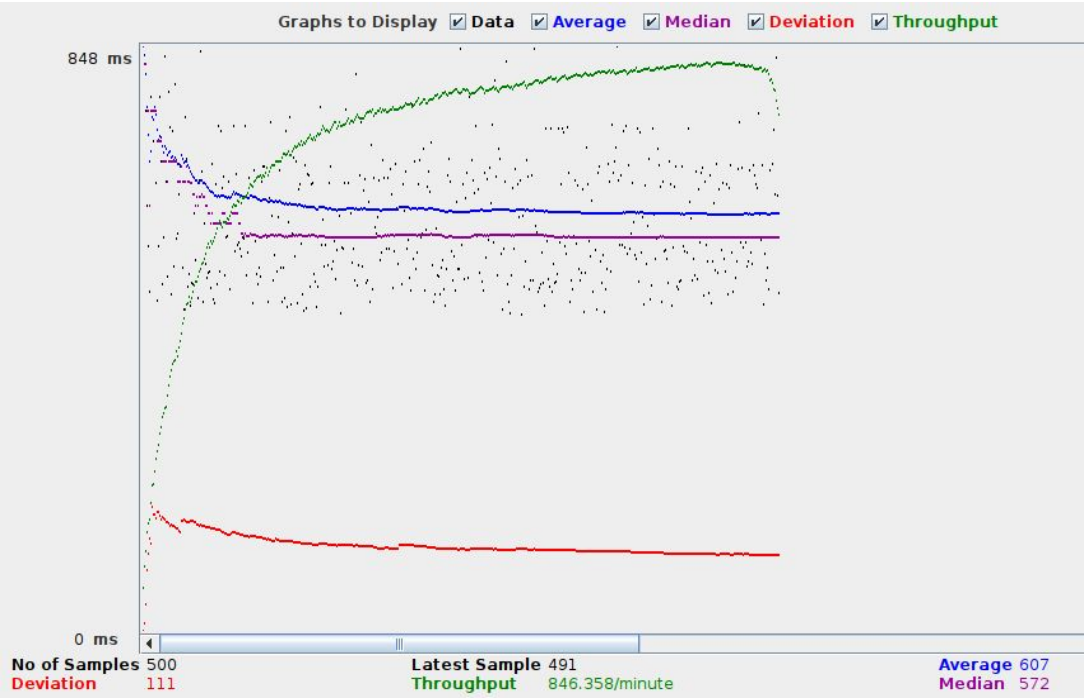


Figura 2: Gráfico de throughput, tiempo medio de respuesta y mediana para la prueba 1 (10 conexiones concurrentes)

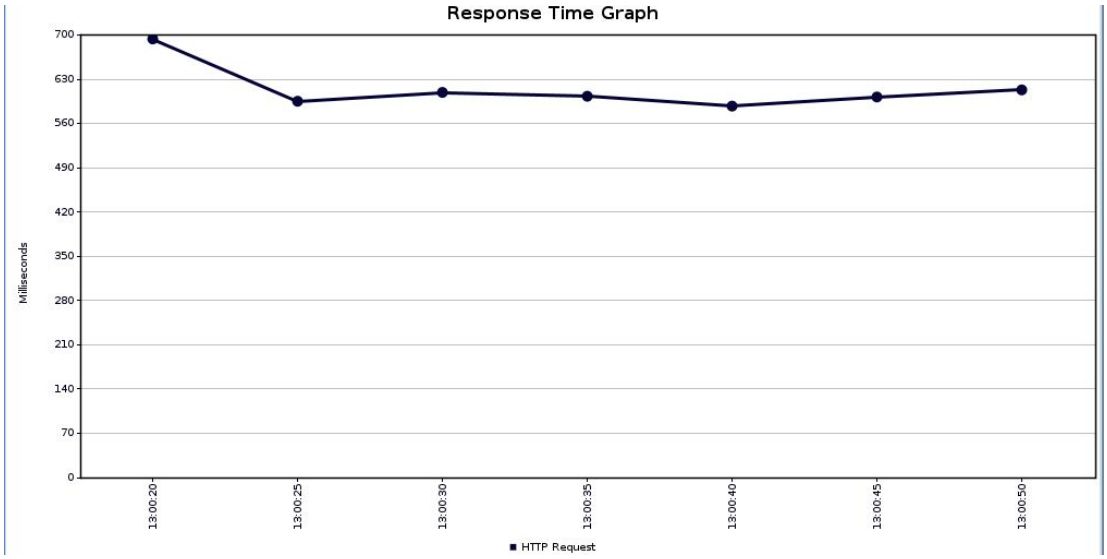


Figura 3: Tiempo de respuesta (en ms) en función del tiempo para la prueba 1 (10 conexiones concurrentes)

Caso de prueba 2: Grupo A: 10 conexiones, Grupo B: 100 conexiones

En este caso, el Throughput se reduce a aproximadamente 4.7 requests/ segundo. El tiempo medio de respuesta aumenta a 1843 ms, y la mediana a 1576 ms. El error se mantiene en 0%. El comporamiento sigue siendo estable.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
HTTP Request	500	1843	525	9481	1033.23	0.00%	4.7/sec	206.66	44762.9
TOTAL	500	1843	525	9481	1033.23	0.00%	4.7/sec	206.66	44762.9

Figura 4: Resumen de conexiones con para la prueba 2(110 conexiones concurrentes)

Figura 5: Gráfico de throughput, tiempo medio de respuesta y mediana para la prueba 2 (110 conexiones concurrentes)

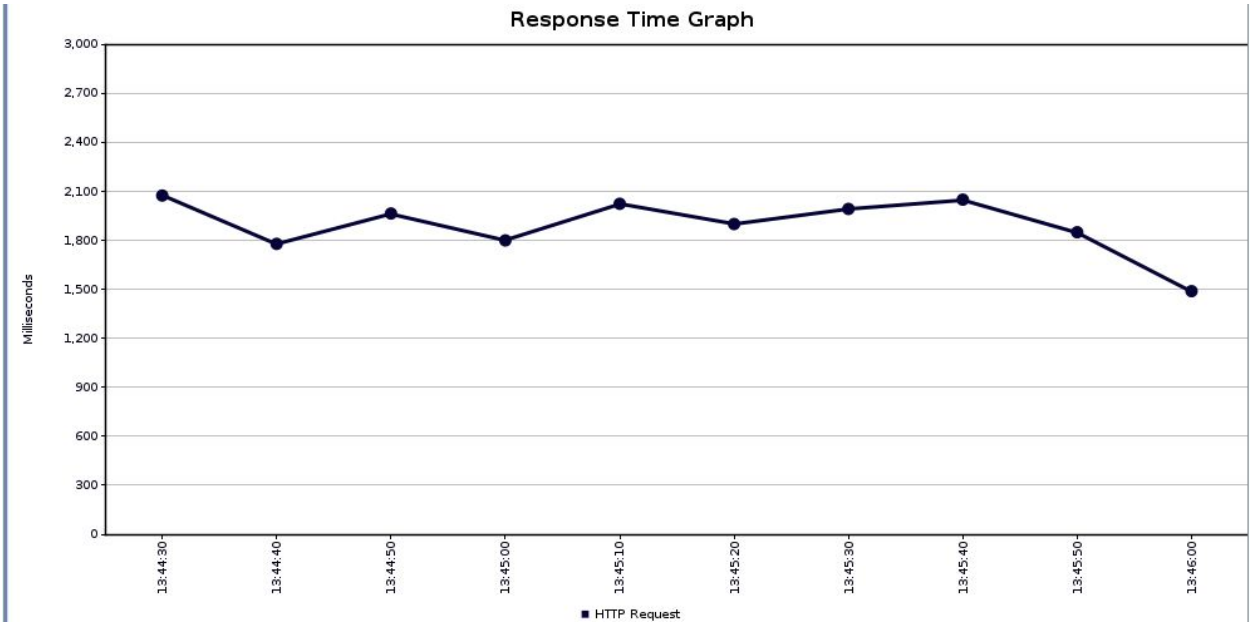


Figura 6: Tiempo de respuesta (en ms) en función del tiempo para la prueba 2 (110 conexiones concurrentes)

Caso de prueba 3 (Maximo Stress): Grupo A: 10 conexiones, Grupo B: 500 conexiones
En este caso se puede apreciar un comportamiento mucho más errático, propio de llevar el servidor al límite de sus capacidades. El throughput se reduce considerablemente a 55.8/ minuto. El tiempo medio de respuesta se estabiliza en torno al 4000 ms, para luego aumentar al final de la prueba hasta 36000 ms, resultando en una promedio de 4849 ms. La media se estabiliza en torno a los 2400 ms.. Se aprecia un error del 0,2%, lo cual es esperable debido a la cantidad de conexiones

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
HTTP Request	500	4849	514	54831	7319.97	0.20%	55.8/min	40.60	44678.0
TOTAL	500	4849	514	54831	7319.97	0.20%	55.8/min	40.60	44678.0

Figura 7: Resumen de conexiones concurrentes para la prueba 3 (510 conexiones concurrentes)

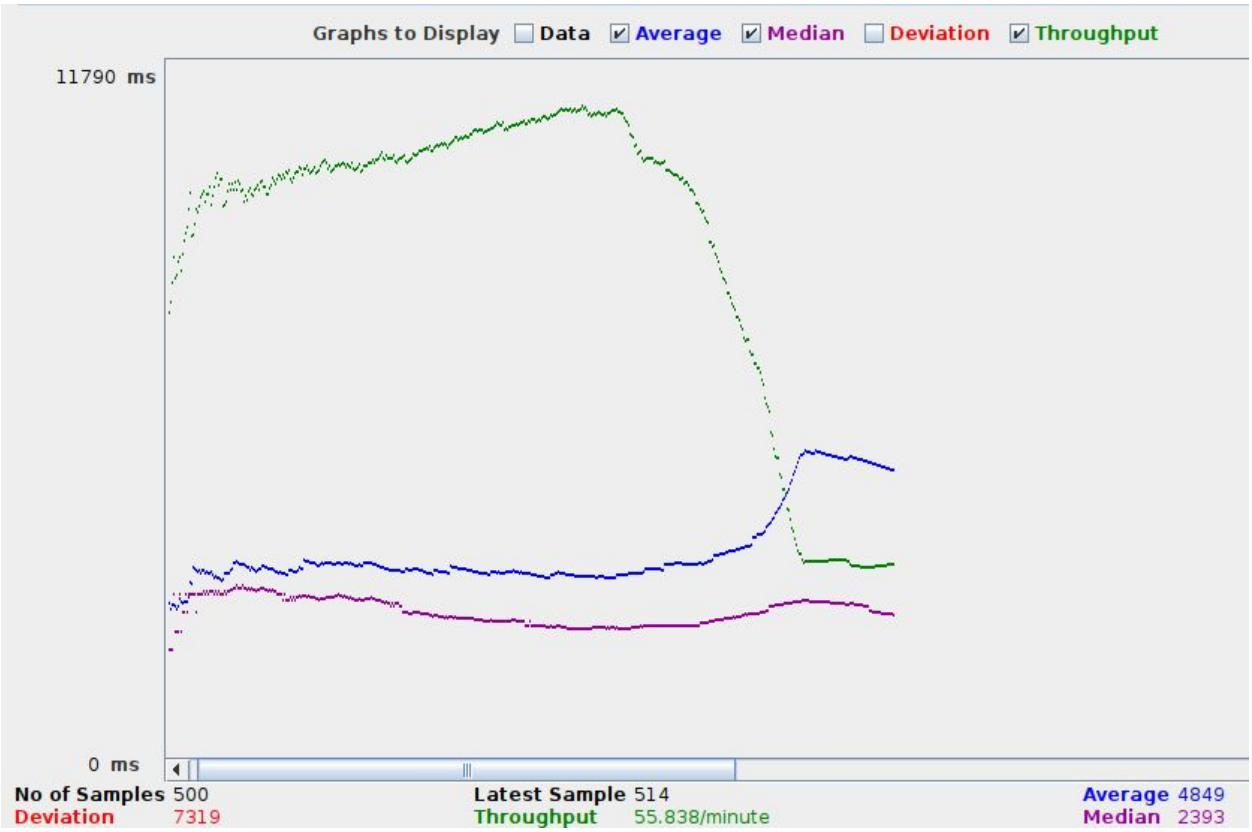


Figura 8: Gráfico de throughput, tiempo medio de respuesta y mediana para la prueba 3 (510 conexiones concurrentes)

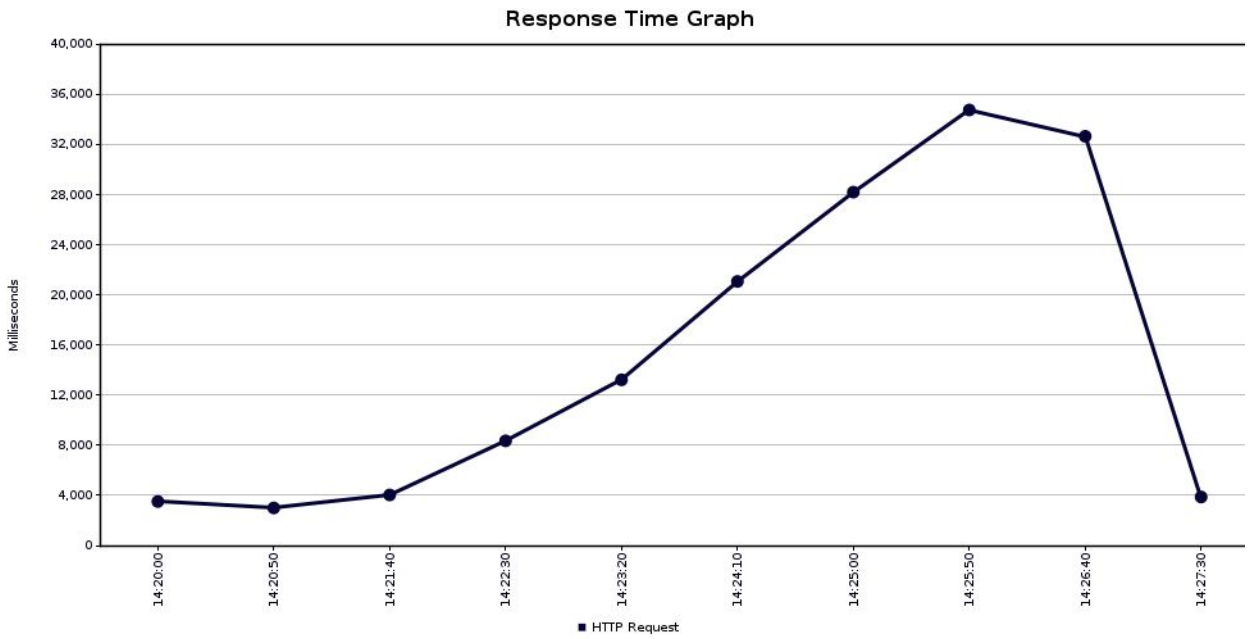


Figura 9: Tiempo de respuesta (en ms) en función del tiempo para la prueba 3 (510 conexiones concurrentes)

Conclusiones

Al pasar de las capacidades mínimas del servidor hasta el máximo (512 conexiones concurrentes), se evidencia la caída en el throughput y el aumento de la media y el promedio de tiempo de respuesta. El servidor se mantiene estable pero la diferencia se hace evidente a los usuarios. Por otra parte, es interesante analizar el 0,2% de error para la prueba de máxima cantidad de conexiones: es altamente probable que esto se deba a que Jmeter debe incializarlas en 1 segundo, por lo que si la respuesta no llega del servidor se corta la conexion y el resultado de la prueba da incorrecto.

Degradación de Throughput Vs Tamaño de Buffer

Para realizar estas pruebas de stress se utilizó JMeter, se generaron pruebas con 250 clientes concurrentes con 0 segundos de ramp up, y cada uno ejecutando 10 requests para asegurarnos la concurrencia, y con diferentes tamaños de buffer, 4k, 8k, 16k y 32k. El objetivo de estas pruebas era comparar el throughput con estas diferentes configuraciones. Todas las pruebas fueron hechas contra el feed de twitter (twitter.com/explore). Los resultados son los siguientes:

Buffer de 4k

Como se puede ver en la Figura 10, el throughput es de 4122 requests por minuto aproximadamente, y el tiempo medio de respuesta ronda los 1270-1557 ms.

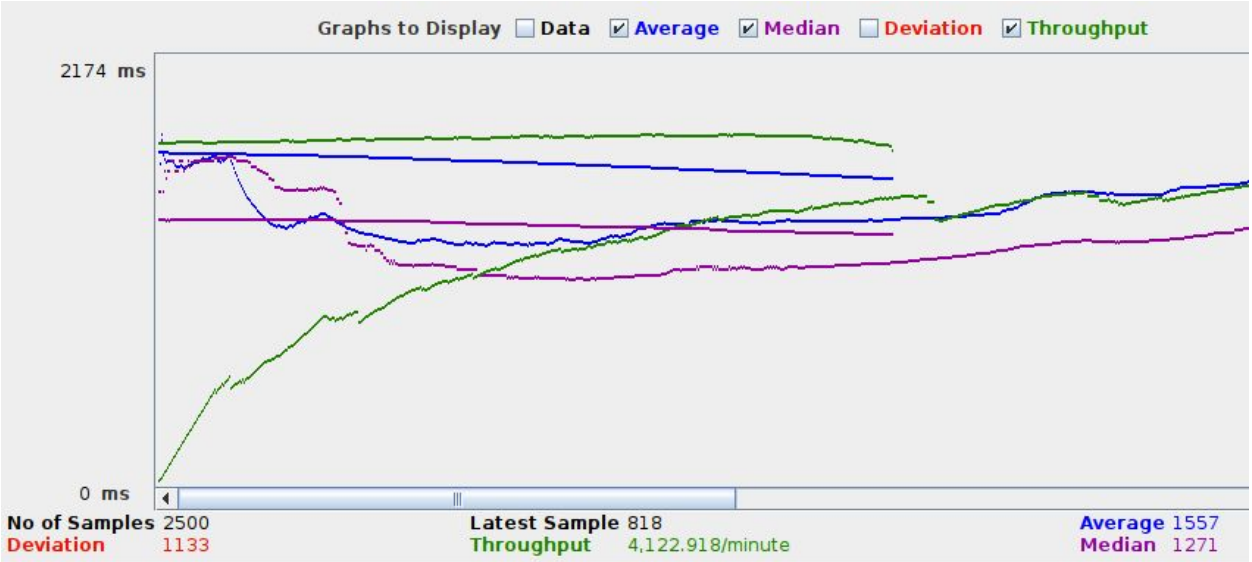


Figura 10: Gráfico de throughput, tiempo medio de respuesta y mediana con un buffer de 4k con 250 clientes concurrentes

Buffer de 8k

Como se puede ver en la Figura 11, el throughput es de 4093 requests por minuto aproximadamente, y el tiempo medio de respuesta ronda los 1518-1861 ms.

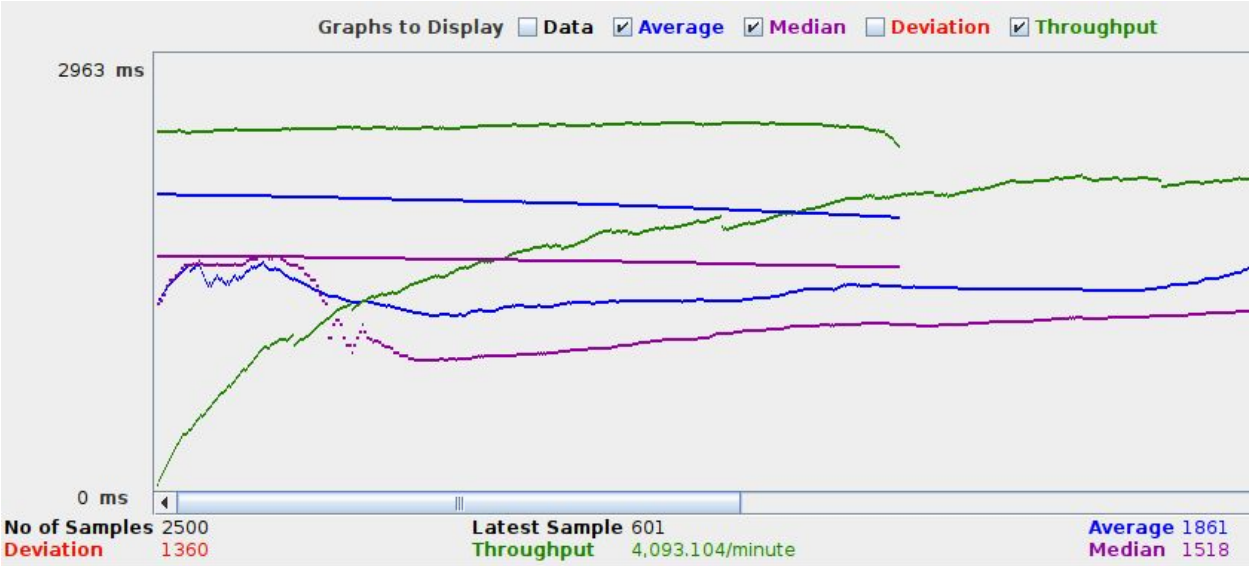


Figura 11: Gráfico de throughput, tiempo medio de respuesta y mediana con un buffer de 8k con 250 clientes concurrentes

Buffer de 16k

Como se puede ver en la Figura 12, el throughput es de 4332 requests por minuto aproximadamente, y el tiempo medio de respuesta ronda los 910-1105 ms.

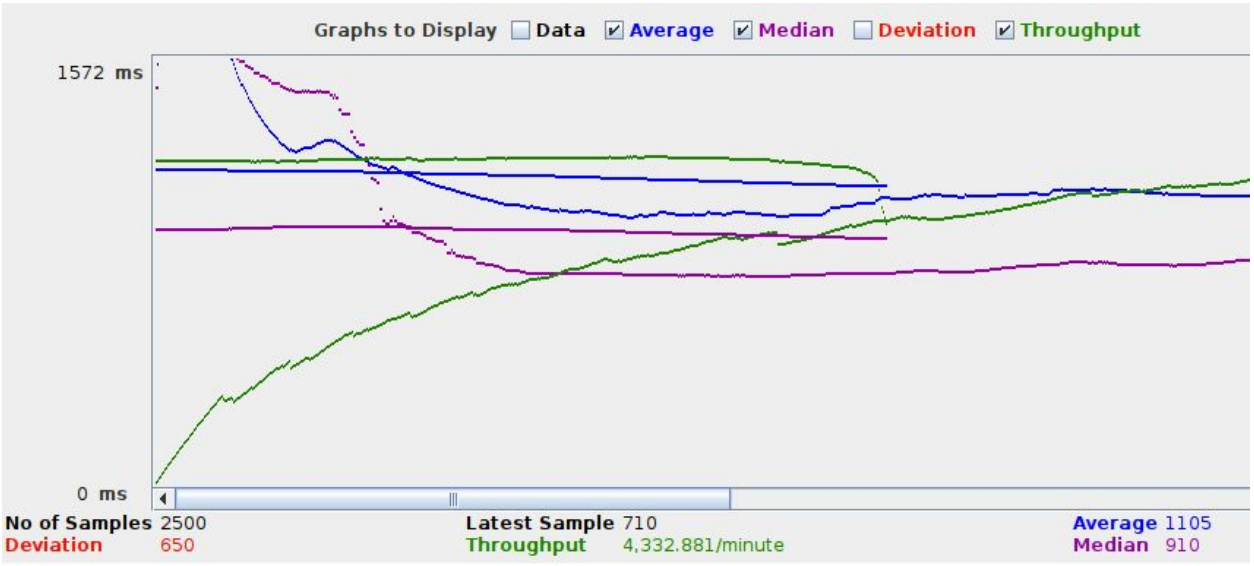


Figura 12: Gráfico de throughput, tiempo medio de respuesta y mediana con un buffer de 16k con 250 clientes concurrentes

Buffer de 32k

Como se puede ver en la Figura 13, el throughput es de 3983 requests por minuto aproximadamente, y el tiempo medio de respuesta ronda los 1575-2125 ms.

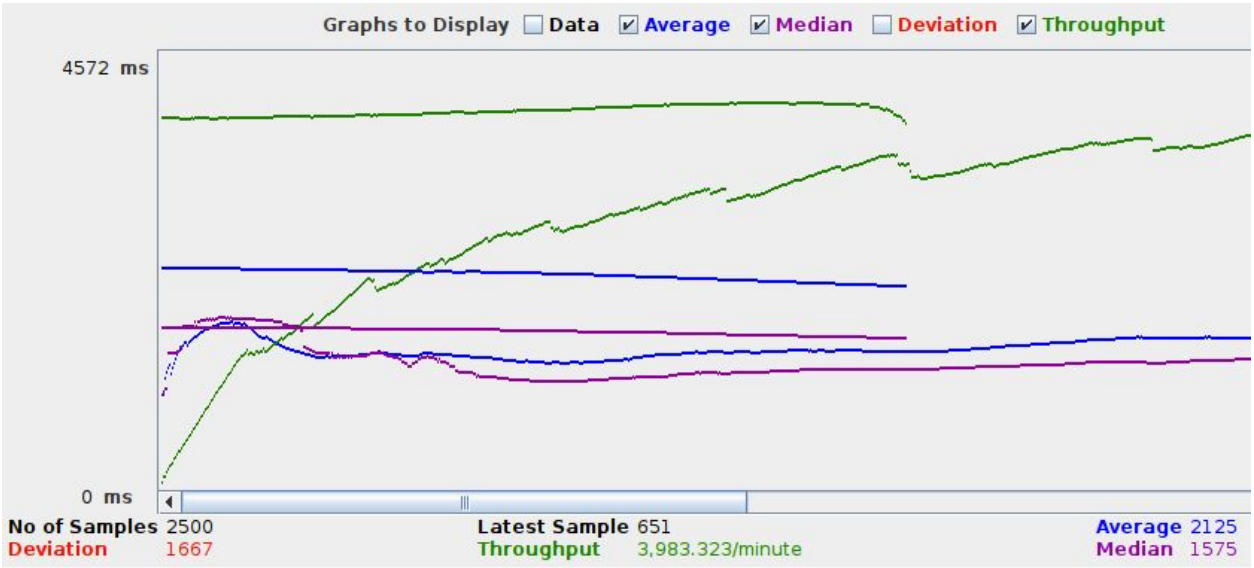


Figura 13: Gráfico de throughput, tiempo medio de respuesta y mediana con un buffer de 32k con 250 clientes concurrentes

Conclusiones

Se puede observar que el throughput aumenta entre los buffers de 4k, 8k y 16k, pero que comienza a degradarse si se incrementa a 32k. En términos de los tiempos de respuesta, en el

caso del buffer de 16k es el mejor, pero varía bastante dependiendo del tamaño que se esté probando.

El tamaño de buffer que se elige a partir de estas pruebas es de 16k, parece ser el tamaño de buffer que genera resultados más consistentes y un menor tiempo de respuesta. Es importante notar que estas pruebas se realizaron con una request que no es grande, por ende esta decisión podría cambiar si se pudieran hacer pruebas con requests más pesadas como una descarga de un archivo.

Guía de Instalación

Para poder instalar y ejecutar todos los entregables, se sugiere seguir la guía que se encuentra en el *README* en el repositorio GIT.

Para construir los ejecutables se ejecuta el comando:

```
$ make clean all
```

Esto genera los binarios del servidor y el cliente SCTP para poder ejecutarlos.

Para poder ejecutar el servidor se corre el siguiente comando:

```
$ ./server <argumentos>
```

Donde los argumentos son los definidos en el *man* del servidor otorgado por la cátedra, y que se encuentra en la carpeta de *Documentación* del repositorio GIT.

Para ejecutar el cliente se corre el siguiente comando:

```
$ ./sctp_client <argumentos>
```

Donde los argumentos son los definidos en el *man* del cliente sctp creado por el grupo, y que se encuentra en la carpeta *Documentación* del repositorio GIT.

Configuración

El servidor y el cliente SCTP desarrollado por nosotros permiten la configuración al momento de inicializar las distintas aplicaciones. La configuración del servidor y del cliente se encuentran especificados en los manpages otorgados por la cátedra y generados por nosotros, los mismos se encuentran en la carpeta de *Documentación* en el repositorio GIT.

Proxy

Para el proxy son configurables desde la línea de comandos:

- IP del servidor DoH - El IP del servidor DNS utilizado
- Puerto del servidor DoH - El puerto del servidor DNS utilizado
- Host del servidor DoH - El nombre del host del servidor DNS utilizado
- Path al servidor DoH - El path para la request DNS
- Query para el servidor DoH - El query para realizar las consultas DNS
- Dirección del servidor proxy - IP donde sirve el servidor proxy
- Puerto del servidor proxy - Puerto donde sirve el servidor proxy
- Dirección del servidor de administración - IP donde sirve el servidor de administración
- Puerto del servidor de administración - Puerto donde sirve el servidor de administración
- Estado de los disectores - Activa o desactiva los disectores de contraseñas
- Usuarios habilitados - Agrega usuarios habilitados para ser usados

Para el proxy también son configurables desde el protocolo de administración otras configuraciones:

- Tamaño del buffer del proxy - Tamaño del buffer usado en el proxy
- Tamaño del buffer de administración - Tamaño del buffer usado para el protocolo de administración
- Estado de los disectores - Activa o desactiva los disectores de contraseñas

Cliente SCTP

Para el proxy son configurables desde la línea de comandos:

- Dirección del servidor de administración - IP donde sirve el servidor de administración
- Puerto del servidor de administración - Puerto donde sirve el servidor de administración

A través del cliente SCTP se pueden realizar las configuraciones especificadas en la sección de proxy y configuraciones mediante el protocolo de administración.

Todas estas configuraciones se encuentran explicadas en el *man* de socks5d y sctp_clientd. Estos pueden ser vistos de la siguiente forma:

```
$ man <path al archivo .d>
```

Monitoreo y Configuración

Al poseer un cliente interactivo para el protocolo IGAF, se pueden hacer pruebas más interesantes sobre la configuración y monitoreo. Para todos los casos de prueba del cliente se usarán las credenciales con usuario **admin** y contraseña **admin**, aunque el servidor posee credenciales default para los administradores del mismo, estas credenciales se encuentran en el archivo serverdata/admin_user_pass.txt en texto plano en formato "usuario(espacio)contraseña". Para todos los casos de prueba del proxy se usarán las credenciales con usuario **proxy** y contraseña **admin**, se pueden agregar nuevas credenciales en el archivo serverdata/user_pass.txt en texto plano en formato "usuario(espacio)contraseña".

Para todas las pruebas se necesita ejecutar el servidor y posteriormente el cliente, y luego logueandose como el admin con las credenciales especificadas en el párrafo anterior.

Prueba 1 - Creación de Administradores

1. Seleccionar la opción de listar usuarios, aquí se pueden ver todos los usuarios administradores que hay.
2. Seleccionar la opción de crear usuario e ingresar un nuevo nombre de usuario y una nueva contraseña.
3. Seleccionar la opción de listar usuarios, aquí aparecerá el nuevo usuario recién creado.
4. Si se ejecuta una nueva instancia del cliente en otra terminal, podrá loguearse con las nuevas credenciales.

Prueba 2 - Obtención de Métricas

1. Seleccionar la opción de mostrar métricas y notar las métricas obtenidas.
2. Ejecutar chrome o firefox haciendo uso del proxy con las credenciales del proxy especificadas arriba y navegar
3. Ir seleccionando cada una cierta cantidad de segundos la opción de mostrar métricas y se puede ver el cambio en tiempo real de las métricas debido a que el servidor se encuentra atendiendo conexiones actualmente.

Prueba 3 - Configuraciones Remotas

1. Realizar una prueba de velocidad de descarga a través del proxy.
2. Con el cliente SCTP elegir la opción de modificar configuraciones y seleccionar la opción de configurar tamaño del buffer del proxy.
3. Volver a realizar la misma prueba de velocidad de descarga a través del proxy para observar cambios.

Documento de Diseño

Este documento de diseño muestra los distintos estados por los que pasa la aplicación de servidor cuando recibe una request. Hacemos énfasis en que tiene dos posibles flujos principales, el flujo en que recibe una request de un usuario para cumplir el rol de proxy socks, y el flujo en que recibe una request de un usuario para poder acceder a las configuraciones remotas.