

Trabajo Práctico Especial

72.07 - Protocolos de Comunicación



2021 1Q, Grupo 10
Fecha de entrega: 15/06/2021

BERNAD, Francisco (59.538)
MANFREDI, Ignacio (58.513)
LEGAMMARE, Joaquín (59.376)
RAMPOLDI, Nicolas (59.066)

Índice

Protocolo y Aplicación Desarrollada	3
Problemas encontrados durante el diseño y la implementación	4
Conexiones IPv4, IPv6 y resolución FQDN	4
Manejo de los diferentes estados	4
Elección del método HTTP	4
Bloqueo	4
Parsing	5
Cierre de conexiones	5
Cierre de la aplicación	5
Limitaciones de la aplicación	6
Máxima cantidad de conexiones	6
Requests DoH	6
Posibles extensiones	7
DNS over HTTP	7
Cache	7
PERCY	7
Conclusiones	8
Ejemplos de prueba	9
Transferencia de un archivo pesado	9
Transferencia de archivo pesado con concurrencia	10
Utilización del proxy en un browser	11
Funcionamiento del Garbage Collector	13
Instrucciones para la configuración de monitoreo	14
Ejemplos de configuración y monitoreo	15
Número de conexiones históricas	15
Número de bytes enviados	15
Número de bytes recibidos	15
Número de bytes transferidos	16
Solicitud y cambio de tamaño de los buffers	17
Solicitud y cambio del timeout del selector	18
Cambio del valor de los disectores	19
Gráfico de requests	20
Guía de instalación	21
Instrucciones para la configuración	22

Documento de diseño del proyecto	24
Carpetas	24
connections_manager	24
logger	24
management	24
management_client	24
metrics	24
parsers	24
state_machine	25
states	25
utils	25

1. Protocolo y Aplicación Desarrollada

Se llevó a cabo la implementación de un servidor proxy para el protocolo HTTP/1.1 siguiendo los estándares establecidos en el RFC 7230. El servidor tiene la capacidad de atender múltiples clientes en forma concurrente y simultánea, soportar los métodos tradicionales descritos en el RFC 7231 (Sec. 4.3) y ejecutar conexiones salientes a servicios TCP a direcciones IPv4 (RFC 791), IPv6 (RFC 2460) o Fully Qualified Domain Names.

Para cumplir el objetivo de atender múltiples clientes en forma concurrente y simultánea, el servidor utiliza un multiplexor de entrada/salida que emplea la syscall pselect.

Si se requiere realizar una resolución de FQDN, la misma se efectúa mediante DoH (DNS over HTTP), siguiendo lo determinado en el RFC 8484. Este procedimiento otorga tanto registros A como registros AAAA.

En primer lugar se realiza una consulta al servidor DOH preguntando por el registro A y se intenta la conexión con la primera IP obtenida en la respuesta. Si no se puede establecer la conexión, se intenta con la siguiente IP disponible. En caso de no poder conectarse con ninguna IPv4, se realiza una consulta al servidor por los registros AAAA y nuevamente se itera por las IPv6 obtenidas en la respuesta intentando establecer la conexión con alguna de ellas. En el supuesto escenario que el proxy no se pueda conectar a ninguna de ellas, se devuelve al cliente un error con un status code de la familia del 500.

Además, se mantiene un registro de las conexiones históricas, las conexiones concurrentes, la cantidad de bytes enviados y recibidos y el total de conexiones fallidas.

Por otro lado, se implementó el monitoreo del tráfico y un registro de credenciales de acceso con usuarios y contraseñas para los protocolos HTTP y POP3. Las credenciales son enviadas a salida estándar.

Para poder acceder a las estadísticas mencionadas previamente, se tuvo que crear un protocolo denominado PERCY. Dicho protocolo le permite al cliente, mediante una passphrase que lo autoriza, acceder a las métricas, setear si se desea sniffear las credenciales de los usuarios, y cambiar el tamaño de los I/O buffers y del timeout del pselect .

2. Problemas encontrados durante el diseño y la implementación

Conexiones IPv4, IPv6 y resolución FQDN

A la hora de realizar las conexiones, el hecho de analizar si se trata de una IPv4, IPv6 o un FQDN y resolverlo adecuadamente fue un gran obstáculo. Por un lado, asegurarse que al parsear el argumento recibido, cumpla con los estándares soportados. Además, hacer el intento de conexión y analizar si fue exitosa o no.

Luego, si se trata de un FQDN, realizar la resolución DNS sobre HTTP, lo cual trae consigo la dificultad de que una vez hecha la conexión con el servidor DoH, se recuperan las IPs correspondientes y el acto de intentar una por una hasta que se efectúe la conexión exitosamente.

Manejo de los diferentes estados

En cuanto al manejo de los diferentes estados, la complicación vino por parte del planeamiento previo. Se diagramaron las diferentes posibilidades para poder tener un análisis de antemano y no caer en estados redundantes o que repitieran funcionalidad.

Una vez alcanzado un estado, se ejecuta un handler que se encarga de la iniciación de los parámetros correspondientes. Además, hay un handler de escritura y uno de lectura. De esta forma, se modulariza apropiadamente cada estado y se facilita el mantenimiento.

Elección del método HTTP

Un dilema que hubo que resolver fue determinar qué método HTTP utilizar a la hora de crear el cliente DoH. Hay dos opciones, GET o POST, ambas válidas. Se optó por la opción de elegir POST debido a que no es necesario encodear la URL en base64url y las requests son generalmente más pequeñas que su equivalente con GET (mencionado en el RFC 8484).

Bloqueo

Ya que tanto la utilización de sockets como las operaciones de entrada/salida debían ser no bloqueantes, había que ser cautelosos en la implementación, lo cual requiere de más planeamiento e investigación. Se utilizó la implementación del selector de la cátedra que permite solucionar el bloqueo de cada conexión. Como fue mencionado anteriormente, la máquina de estados permite predeterminar mediante la utilización de un handler, cómo va a actuar cada estado al momento de ser ejecutado por el selector.

Parsing

Otro inconveniente encontrado fue el hecho de cómo llevar a cabo el parseo de la distinta información a lo largo de toda la aplicación. Si bien el hecho de parsear en sí no es altamente complejo para los casos que se debían contemplar, si fue un desafío evitar al máximo posible la utilización excesiva de almacenamiento. Aun así, en la mayoría de los casos, no quedó más opción que guardar los parsers junto a las entidades que hacían uso de los mismos. Por ejemplo, `doh_connection_t` contiene un `headers_parser`. De esta forma, se facilita mantener el estado de cada parseo, lo cual de lo contrario sería imposible.

Cierre de conexiones

Se encontró una dificultad para realizar el correcto cierre de conexiones y limpieza de recursos. Cuando un socket decide cerrar la conexión o no escribir más en el socket, hubo que asegurarse de que los bytes presentes en los buffers lleguen a destino antes de cerrar la conexión. Además hubo que tener cuidado al suscribir a los sockets en este momento, ya que un mal manejo de esta situación provocaba que quede suscripto en el selector y que no se libere el recurso.

Cierre de la aplicación

Por otro lado, al momento de finalizar la aplicación mediante señales (CTRL+C o kill) hubo que manejar estas mismas mediante handlers que se ocuparan de la correcta finalización y liberación de los recursos alocados.

3. Limitaciones de la aplicación

Máxima cantidad de conexiones

Existe una máxima cantidad de conexiones simultáneas que está limitada por la implementación del selector que permite monitorear 1024 sockets a la vez. Tanto el proxy http como el servidor de configuración y de obtención de métricas utilizan 2 sockets cada uno para poder establecer conexiones ipv4 e ipv6 Además se utilizan los file descriptors correspondientes a salida estándar y de errores que consumen un file descriptor cada uno. Por lo tanto, el número de conexiones concurrentes se limita a 509, ya que cada una de ellas utiliza 2 file descriptors (uno para conectarse con el cliente y otro para comunicarse con el origin).

Requests DoH

Al hacer la request DoH, como fue mencionado anteriormente, primero se pregunta por direcciones de tipo IPv4 y luego IPv6. Esto trae como consecuencia tiempo muerto que podría solucionarse si se hicieran las requests en paralelo.

4. *Possibles extensiones*

DNS over HTTP

En cuanto a DoH, las request se podrían mejorar utilizando el protocolo HTTP 1.1, el cual cuenta con conexiones persistentes. De esta forma, se podría tener una serie de conexiones en ejecución y cuando un cliente requiera de la resolución de un FQDN, tomaría una de esas conexiones, lo cual sería más eficiente que lo implementado. Además, no sería necesario establecer una conexión por cada request, por ende disminuiría la cantidad de file descriptors utilizados.

Cache

Se podría realizar un cacheo de las direcciones resueltas por el servidor DoH para no tener que realizar nuevamente consultas por dominios resueltos con anterioridad, teniendo siempre en cuenta el valor del TTL de la respuesta.

PERCY

Tanto la consulta como la respuesta del protocolo PERCY cuentan con un byte reservado para permitir la expansión del protocolo a futuro. Ejemplos de funcionalidades que podrían ser agregadas son: agrandar el tamaño del valor de las respuestas o la cantidad de métodos de consulta.

5. Conclusiones

Se llegó al consenso de que si bien la implementación del servidor proxy HTTP con los cimientos basados en el multiplexor de entrada/salida, es un trabajo complejo y requiere de mucho estudio e investigación, ayuda a comprender de manera integra los fundamentos del protocolo y la materia.

A su vez, el hecho de contar con una máquina de estado y estados definidos para cada momento clave de la ejecución, permite abstraer al encargado de utilizarlos y brinda un programa mucho más extensible, fácil de comprender y mantenible. Esto se cree de vital importancia para conseguir llevar a cabo una aplicación de este tipo.

Por último, la indagación en los distintos RFC ayudó no solo a entender los estándares a seguir y cómo lograr los objetivos, sino también a afianzar y terminar de comprender el funcionamiento de conceptos vistos a lo largo de la materia.

6. Ejemplos de prueba

Para permitir la resolución de nombres en el entorno de prueba se utilizó una [imagen de docker](#) que provee servidor de DNS over HTTP. Para simular la transferencia de tráfico pesado, se utilizó el siguiente archivo:

```
~/PROTOS/http-latest/webapp > ls -lh 1.iso && sha256sum 1.iso
-rw-r--r-- 1 frbernad frbernad 1.2G Jun 13 16:35 1.iso
593deded36a768af99f26689ae930583e0d3b65f855f9a54f7482c5ba50f54cc  1.iso
```

Figura 1. Archivo pesado utilizado para transferencia. Se puede observar su hash.

Transferencia de un archivo pesado

Se evaluó y comparó la transferencia de un archivo de 1,20 GB mediante el comando curl con el proxy y sin el proxy. Puede observarse que los tiempos de transferencia promedio son similares. Más aún, puede observarse que no hubo pérdida de información comparando el hash del archivo transferido con el original.

```
/mnt/d/Escritorio/PROGRAMMING/ITBA/PROTOS/TP/HTTPD-PROXY master *8 > time curl -s http://bar.leak.com.ar:2020/1.iso | pv | sha256sum
1.20GiB 0:00:07 [ 173MiB/s] [ =>
593deded36a768af99f26689ae930583e0d3b65f855f9a54f7482c5ba50f54cc -
curl -s http://bar.leak.com.ar:2020/1.iso 0.51s user 1.50s system 28% cpu 7.064 total
pv 0.12s user 1.72s system 26% cpu 7.064 total
sha256sum 5.51s user 0.68s system 87% cpu 7.064 total
```

Figura 2. Tiempo de transferencia del archivo sin el uso del proxy.

```
/mnt/d/Escritorio/PROGRAMMING/ITBA/PROTOS/TP/HTTPD-PROXY master *8 > export ALL_PROXY="http://localhost:8080"
/mnt/d/Escritorio/PROGRAMMING/ITBA/PROTOS/TP/HTTPD-PROXY master *8 > time curl -s http://bar.leak.com.ar:2020/1.iso | pv | sha256sum
1.20GiB 0:00:06 [ 180MiB/s] [ =>
593deded36a768af99f26689ae930583e0d3b65f855f9a54f7482c5ba50f54cc -
curl -s http://bar.leak.com.ar:2020/1.iso 0.37s user 1.71s system 30% cpu 6.807 total
pv 0.21s user 1.66s system 27% cpu 6.806 total
sha256sum 5.33s user 0.91s system 91% cpu 6.806 total
```

Figura 3. Tiempo de transferencia del archivo con el uso del proxy.

```
/mnt/d/Escritorio/PROGRAMMING/ITBA/PROTOS/TP/HTTPD-PROXY master *8 > ./httpd
2021-06-13T16:38:48Z      A      127.0.0.1      46594      GET      http://bar.leak.com.ar:2020/1.iso      200
```

Figura 4. Log de acceso que muestra el GET realizado al proxy.

Transferencia de archivo pesado con concurrencia

Se evalúo la transferencia de un archivo de 1,20 GB mediante el comando curl utilizando el proxy con cuatro clientes a la vez. Puede observarse que los tiempos de transferencia son similares y que la performance sigue siendo óptima. Más aún, puede observarse que no hubo pérdida de información comparando el hash del archivo transferido con el original en cada cliente.

```
/mnt/d/Escritorio/PROGRAMMING/ITBA/PROTOS/TP/HTTPD-PROXY master *8 > ./httpd
2021-06-13T16:40:25Z      A      127.0.0.1      46608    GET      http://bar.leak.com.ar:2020/1.iso      200
2021-06-13T16:40:25Z      A      127.0.0.1      46618    GET      http://bar.leak.com.ar:2020/1.iso      200
2021-06-13T16:40:25Z      A      127.0.0.1      46626    GET      http://bar.leak.com.ar:2020/1.iso      200
2021-06-13T16:40:26Z      A      127.0.0.1      46638    GET      http://bar.leak.com.ar:2020/1.iso      200
```

Figura 5. Log de acceso que muestra cuatro métodos GET realizados al proxy.

```
/mnt/d/Es/P/I/PR/TP/HTTPD-PROXY master *8 > export ALL_PROXY="http://localhost:8080/"
/mnt/d/Es/P/I/PR/TP/HTTPD-PROXY master *8 > time curl -s http://bar.leak.co
m.ar:2020/1.iso | pv | sha256sum
1.20GiB 0:00:08 [ 139MiB/s ] [ => ] 593deded36a768af99f26689ae930583e0d3b65f855f9a54f7482c5ba50f54cc -
curl -s http://bar.leak.com.ar:2020/1.iso 0.40s user 1.80s system 24% cpu 8
.801 total
pv 0.13s user 1.79s system 21% cpu 8.801 total
sha256sum 7.57s user 0.97s system 97% cpu 8.801 total
/mnt/d/Es/P/I/PR/TP/HTTPD-PROXY master *8 >                                                 9s
```



```
/mnt/d/Es/P/I/PR/TP/HTTPD-PROXY master *8 > export ALL_PROXY="http://localhost:8080/"
/mnt/d/Es/P/I/PR/TP/HTTPD-PROXY master *8 > time curl -s http://bar.leak.co
m.ar:2020/1.iso | pv | sha256sum
1.20GiB 0:00:08 [ 137MiB/s ] [ => ] 593deded36a768af99f26689ae930583e0d3b65f855f9a54f7482c5ba50f54cc -
curl -s http://bar.leak.com.ar:2020/1.iso 0.46s user 1.71s system 24% cpu 8
.901 total
pv 0.08s user 1.87s system 21% cpu 8.900 total
sha256sum 7.46s user 1.02s system 95% cpu 8.900 total
/mnt/d/Es/P/I/PR/TP/HTTPD-PROXY master *8 >                                                 9s
```



```
/mnt/d/Es/P/I/PR/TP/HTTPD-PROXY master *8 > export ALL_PROXY="http://localhost:8080/"
/mnt/d/Es/P/I/PR/TP/HTTPD-PROXY master *8 > time curl -s http://bar.leak.co
m.ar:2020/1.iso | pv | sha256sum
1.20GiB 0:00:09 [ 134MiB/s ] [ => ] 593deded36a768af99f26689ae930583e0d3b65f855f9a54f7482c5ba50f54cc -
curl -s http://bar.leak.com.ar:2020/1.iso 0.37s user 1.79s system 23% cpu 9
.117 total
pv 0.23s user 1.67s system 20% cpu 9.117 total
sha256sum 7.42s user 1.01s system 92% cpu 9.117 total
/mnt/d/Es/P/I/PR/TP/HTTPD-PROXY master *8 > |                                                 9s
```

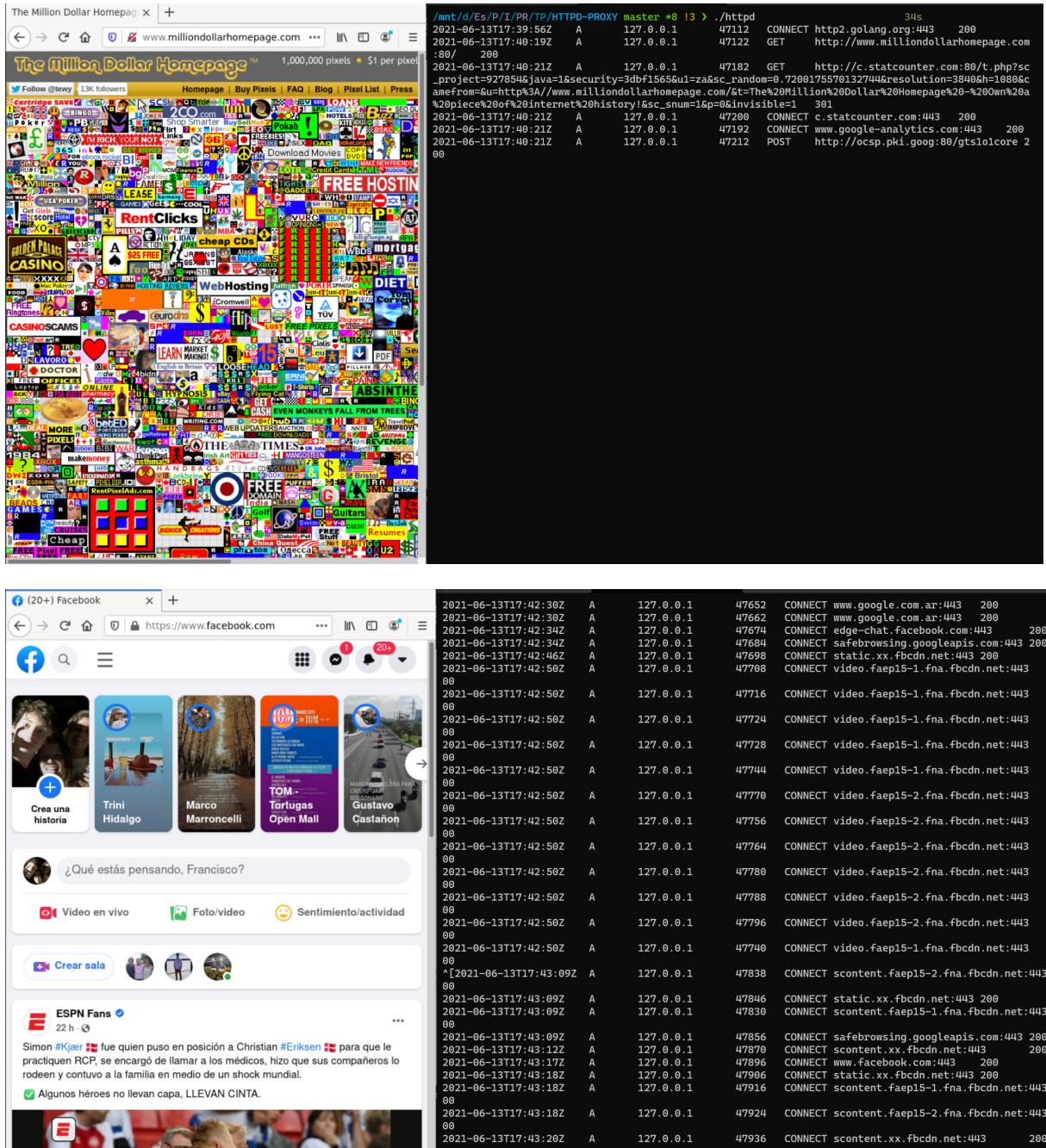


```
/mnt/d/Es/P/I/PR/TP/HTTPD-PROXY master *8 > export ALL_PROXY="http://localhost:8080/"
/mnt/d/Es/P/I/PR/TP/HTTPD-PROXY master *8 > time curl -s http://bar.leak.co
m.ar:2020/1.iso | pv | sha256sum
1.20GiB 0:00:08 [ 140MiB/s ] [ => ] 593deded36a768af99f26689ae930583e0d3b65f855f9a54f7482c5ba50f54cc -
curl -s http://bar.leak.com.ar:2020/1.iso 0.42s user 1.76s system 25% cpu 8
.714 total
pv 0.13s user 1.77s system 21% cpu 8.714 total
sha256sum 7.58s user 1.01s system 98% cpu 8.713 total
/mnt/d/Es/P/I/PR/TP/HTTPD-PROXY master *8 >                                                 9s
```

Figura 6. Tiempo de transferencia del archivo con el uso del proxy con cuatro conexiones concurrentes.

Utilización del proxy en un browser

Las siguientes imágenes demuestran el correcto funcionamiento del proxy en un browser, en este caso Firefox. Se pueden observar request HTTP y la utilización del método CONNECT para acceder a páginas que requieren del uso de HTTPS.



Figuras 7 y 8. Sitios web accedidos mediante el proxy con demanda de recursos y HTTPS.

Mozilla Firefox

http://http2.golang.org/gopherfile

A grid of 180 tiled images is below. Compare:

[HTTP/2, 0s latency] [HTTP/1, 0s latency]
[HTTP/2, 30ms latency] [HTTP/1, 30ms latency]
[HTTP/2, 200ms latency] [HTTP/1, 200ms latency]
[HTTP/2, 1s latency] [HTTP/1, 1s latency]



Times from connection start:
DOM loaded: 1217ms
DOM complete (images loaded): 1762ms

<< Back to Go HTTP/2 demo server

./ROTO/VEGETTA

./httpd

./P/HTTPD-PROXY

```
/mnt/d/E/S/P/I/PR/TP/HTTPD-PROXY master *:8 !3 > ./httpd
2021-06-13T17:39:56Z     A      127.0.0.1    47112   CONNECT http2.golang.org:443      34s
2021-06-13T17:39:56Z     A      127.0.0.1    47112   CONNECT http2.golang.org:443      200
```

HTTP/2: the Future of the Internet | Akamai - Mozilla Firefox

HTTP/2: the Future of the Web, and it is here!

Your browser supports HTTP/2!

This is a demo of HTTP/2's impact on your download of many small tiles making up the Akamai Spinning Globe.

HTTP/1.1

Latency: ms Load time: 5.43s



Demo concept inspired by Golang's Gopherites

HTTP/2

Latency: ms Load time: 2.66s



Demo concept inspired by Golang's Gopherites

./httpd

./P/HTTPD-PROXY

```
2021-06-13T17:37:10Z   A   127.0.0.1    46826   GET   http://detectportal.firefox.com:88/success.txt 200
2021-06-13T17:37:10Z   A   127.0.0.1    46834   CONNECT content-signature=2.cdn.mozilla.net:443 200
2021-06-13T17:37:10Z   A   127.0.0.1    46844   GET   http://detectportal.firefox.com:88/success.txt?ipv4= 200
2021-06-13T17:37:10Z   A   127.0.0.1    46858   CONNECT firefox.settings.services.mozilla.com:443 200
2021-06-13T17:37:11Z   A   127.0.0.1    46868   CONNECT firefox.settings.services.mozilla.com:443 200
2021-06-13T17:37:11Z   A   127.0.0.1    46882   CONNECT safebrowsing.googleapis.com:443 200
2021-06-13T17:37:11Z   A   127.0.0.1    46894   CONNECT snippets.cdn.mozilla.net:443 200
2021-06-13T17:37:11Z   A   127.0.0.1    46904   POST  http://ocsp.pki.goog:88/gts10core 200
2021-06-13T17:37:11Z   A   127.0.0.1    46918   CONNECT snippets.cdn.mozilla.net:443 200
2021-06-13T17:37:18Z   A   127.0.0.1    46938   CONNECT http2.akamai.com:443 200
2021-06-13T17:37:21Z   A   127.0.0.1    46944   CONNECT http1.akamai.com:443 200
2021-06-13T17:37:23Z   A   127.0.0.1    46952   CONNECT http1.akamai.com:443 200
2021-06-13T17:37:23Z   A   127.0.0.1    46953   CONNECT http1.akamai.com:443 200
2021-06-13T17:37:23Z   A   127.0.0.1    46988   CONNECT http1.akamai.com:443 200
2021-06-13T17:37:23Z   A   127.0.0.1    46988   CONNECT http1.akamai.com:443 200
2021-06-13T17:37:23Z   A   127.0.0.1    46992   CONNECT http1.akamai.com:443 200
```

Figuras 9 y 10. Sitios web accedidos mediante el proxy con demanda de recursos y HTTPS.

Fucionamiento del Garbage Collector

En la siguiente imagen se puede observar como el garbage collector finaliza las conexiones inactivas luego de un determinado tiempo.

```
pselect6([16, [0 3 4 5 6 7 8 9 10 11 12 13 14 15], [], NULL, {tv_sec=8, tv_nsec=0}, {[[], 8]} = 1 (in [1
5], left {tv_sec=7, tv_nsec=960384300})
read(15, "\27\3\3\0M8\233\243\4\2\213\254\300\340\317;?\33\35\316*6HQ\367|\;251\373\262*\265"..., 8192
) = 152
pselect6([16, [0 3 4 5 6 7 8 9 10 11 12 13 14 15], [14], NULL, {tv_sec=8, tv_nsec=0}, {[[], 8]} = 1 (out
[14], left {tv_sec=7, tv_nsec=999992900})
sendto(14, "\27\3\3\0M8\233\243\4\2\213\254\300\340\317;?\33\35\316*6HQ\367|\;251\373\262*\265"..., 15
2, MSG_NOSIGNAL, NULL, 0) = 152
pselect6([16, [0 3 4 5 6 7 8 9 10 11 12 13 14 15], [], NULL, {tv_sec=8, tv_nsec=0}, {[[], 8]} = 1 (in [1
4], left {tv_sec=7, tv_nsec=999908400})
read(14, "\27\3\3\0\"324\304\345/2A,Zt\262\337b\265\16Jc\235!\374.\32\262\2\307\334\240\264"..., 8192
) = 39
pselect6([16, [0 3 4 5 6 7 8 9 10 11 12 13 14 15], [15], NULL, {tv_sec=8, tv_nsec=0}, {[[], 8]} = 1 (out
[15], left {tv_sec=7, tv_nsec=999993400})
sendto(15, "\27\3\3\0\"324\304\345/2A,Zt\262\337b\265\16Jc\235!\374.\32\262\2\307\334\240\264"..., 39
, MSG_NOSIGNAL, NULL, 0) = 39
pselect6([16, [0 3 4 5 6 7 8 9 10 11 12 13 14 15], [], NULL, {tv_sec=8, tv_nsec=0}, {[[], 8]} = 0 (Timeo
ut)
pselect6([16, [0 3 4 5 6 7 8 9 10 11 12 13 14 15], [], NULL, {tv_sec=8, tv_nsec=0}, {[[], 8]} = 0 (Timeo
ut)
close(7) = 0
close(6) = 0
close(11) = 0
close(9) = 0
close(8) = 0
close(10) = 0
close(13) = 0
close(12) = 0
close(15) = 0
close(14) = 0
pselect6(6, [0 3 4 5], [], NULL, {tv_sec=8, tv_nsec=0}, {[[], 8]}
```

Figuras 11. Garbage Collector del proxy.

7. Instrucciones para la configuración de monitoreo

El proyecto cuenta con la implementación de un cliente que permite configurar y alterar el funcionamiento del proxy en tiempo real. El cliente recibe una dirección IP y un puerto mediante el cual se comunicará con el servicio de monitoreo y configuración del proxy. Por default corre en loopback en el puerto 9090.

Una vez ejecutado el cliente, se brindan una serie de opciones las cuales pueden ser ejecutadas para modificar, configurar y recuperar métricas y opciones del proxy en tiempo real. La configuración del servidor de monitoreo y modificación están explicadas de manera más detallada en el README.

```
/mnt/d/Es/P/ITBA/PROTOS/TP/HTTPD-PROXY master *8 > ./httpdctl
Select an option

Request methods:

-1 Request the number of historical connections.
-2 Request the number of concurrent connections.
-3 Request the number of bytes sent.
-4 Request the number of bytes received.
-5 Request the number of total bytes transferred.
-6 Request I/O buffer sizes
-7 Request selector timeout.
-8 Request the number of failed connections.

Modification methods:

-9 Enable or disable sniffer mode.
-10 Set I/O buffer size.
-11 Set selector timeout.

Option: |
```

Figura 12. Se observan los métodos de request y modificación disponibles para el cliente.

8. Ejemplos de configuración y monitoreo

A continuación se ejemplifican algunos casos de la configuración y el monitoreo del proxy. El estado inicial será el siguiente:

```
/mnt/d/Escritorio/PROGRAMMING/ITBA/PROTOS/TP/HTTPD-PROXY master *8 > ./httpd
2021-06-13T16:43:26Z      A      127.0.0.1      46654      GET      http://bar.leak.com.ar:2020/1.iso      200
```

Figura 13. Se observa el estado inicial del servidor proxy HTTP.

Número de conexiones históricas

```
RESPONSE:
Version 1
Status  0
Resv    0
Value   1

press enter to continue
```

Figura 14. Se observa que al pedir la cantidad de conexiones históricas se obtiene el valor 1.

Número de bytes enviados

Se puede observar que al pedir el número de bytes enviados, los mismos corresponden a los bytes enviados en la request del archivo 1.iso.

```
RESPONSE:
Version 1
Status  0
Resv    0
Value   291
```

Figura 15. Se observa que al pedir la cantidad de bytes enviados se obtiene el valor 291.

Número de bytes recibidos

Se puede observar que al pedir el número de bytes recibidos, los mismos corresponden al tamaño del archivo 1.iso solicitado.

```
RESPONSE:  
  
Version 1  
Status  0  
Resv    0  
Value   1287000187
```

Figuras 16. Se observa que al pedir la cantidad de bytes recibidos se obtiene el valor 1287000187.

Número de bytes transferidos

Se puede observar que al pedir el número de bytes transferidos, los mismos corresponden a la suma de los bytes enviados y los recibidos.

```
RESPONSE:  
  
Version 1  
Status  0  
Resv    0  
Value   1287000478
```

Figuras 17. Se observa que al pedir la cantidad de bytes transferidos se obtiene el valor 1287000478.

Solicitud y cambio de tamaño de los buffers

Por defecto el tamaño de los buffers utilizados por el proxy es de 8192, y se puede observar que al pedir dicho valor, el mismo coincide. Una vez modificado el valor y pedido de nuevo se puede observar que los cambios se efectuaron.

```
RESPONSE:  
  
Version 1  
Status 0  
Resv 0  
Value 8192
```

Figuras 18. Se observa que al pedir los tamaños de los buffers se obtiene el valor 8192.

```
Buffer I/O size must be between 1024 and 8192  
  
Value: 4024|
```

Figuras 19. Se solicita que ahora los tamaños de los buffers pasen a tener el valor 4024.

```
RESPONSE:  
  
Version 1  
Status 0  
Resv 0  
Value 4024  
  
press enter to continue
```

Figuras 20. Al solicitar nuevamente los tamaños de los buffers se puede ver que el tamaño ha sido modificado correctamente.

Solicitud y cambio del timeout del selector

A continuación se puede observar que el valor del timeout cambia una vez hecho el pedido.

```
pselect6(6, [0 3 4 5], [], NULL, {tv_sec=8, tv_nsec=0}, {[], 8}) = 0 (Timeout)
```

Figura 21. Se observa el timeout actual es 8.

```
RESPONSE:  
  
Version 1  
Status  0  
Resv    0  
Value   8  
  
press enter to continue
```

Figura 22. Se puede obtener también el valor del timeout del selector mediante el cliente.

```
Selector timeout must be between 4 and 12  
  
Value: 10
```

Figura 23. Se modifica el timeout del selector.

```
pselect6(6, [0 3 4 5], [], NULL, {tv_sec=10, tv_nsec=0}, {[], 8}) = 0 (Timeout)
```

Figura 24. Se observa que el timeout actual ahora es 10.

Cambio del valor de los disectores

Por defecto los disectores de credenciales están activados, al correr el siguiente comando se puede observar que las credenciales son obtenidas y que una vez deshabilitado los disectores y repitiendo la request la credencial no es obtenida.

```
~/PROTOS/VEGETTA > curl www.google.com -H"authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l" --proxy localhost:8080 > /dev/null
```

Figura 25. Cliente realizando un curl con el header authorization

```
2021-06-13T17:01:31Z P HTTP http://www.google.com:80/ 80 aladdin opensesame  
2021-06-13T17:01:31Z A 127.0.0.1 46710 GET http://www.google.com:80/ 200
```

Figura 26. Log de acceso y de credenciales. Se puede visualizar que el usuario aladdin con contraseña opensame realizó un GET a <http://www.google.com:80/>

```
1 to enable or 0 to disable  
  
Value: 0
```

Figura 27. Se desactiva el disector.

```
2021-06-13T17:02:15Z DEBUG Modification method: 0 with entry: 0  
2021-06-13T17:02:30Z A 127.0.0.1 46722 GET http://www.google.com:80/ 200
```

Figura 28. En este caso al ser modificado el disector, y haberse desactivado el sniffer, puede verse que ante el mismo request, no se obtienen credenciales

Figura 29. Connect al servidor POP3

```
2021-06-13T17:05:37Z A 127.0.0.1 46742 CONNECT 2.0.0.110:110 200  
2021-06-13T17:05:53Z P POP3 2.0.0.110:110 110 frbernad [REDACTED] 5
```

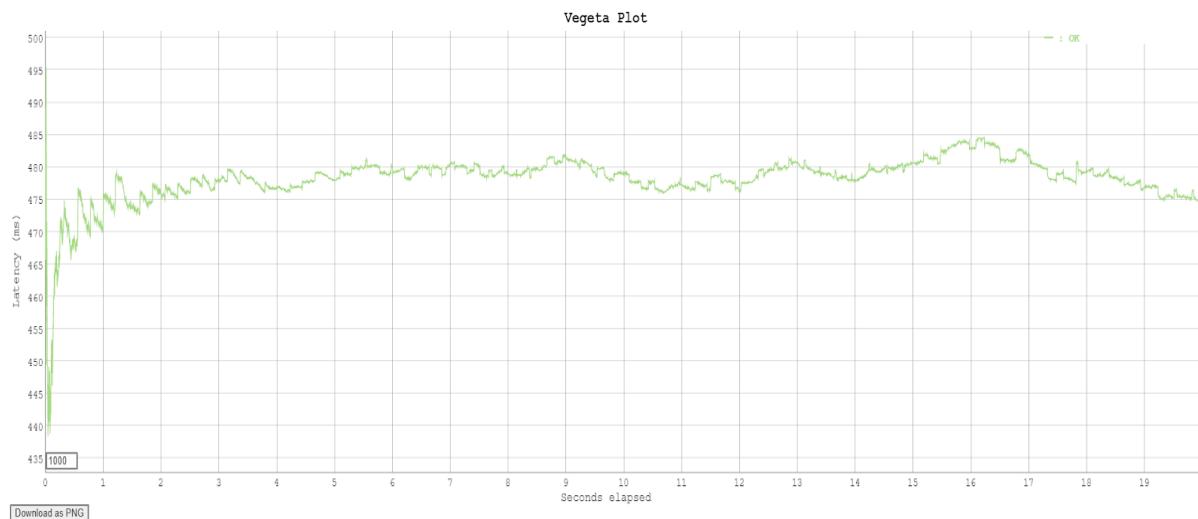
Figura 30. Log de acceso y de credenciales. Se puede visualizar que el usuario frbernad se loguea con el protocolo POP3.

Gráfico de requests

Gráfico de [Vegeta](#), una herramienta versátil para testear servicios HTTP con una cantidad constante de requests. El gráfico representa la latencia de los requests por segundo (en este caso 20s).

Se ejecutó el siguiente comando, estableciendo en una duración de 20 segundos, 500 requests por segundo. La opción -redirects -1 configura que no se sigan los redirects.

```
echo "GET http://www.google.com/" | ./vegeta attack -duration=20s -redirects -1 -rate 500 | ./vegeta plot > plot.html
```



Figuras 31. Gráfico de latencia correspondiente a cada request durante 20 segundos.

9. Guía de instalación

Se utilizó Make de GNU para la compilación del proyecto. Este obtiene su información de compilación del archivo *makefile*. Para la generación del proxy daemon (*httpd*) y el cliente (*httpdctl*), simplemente se debe ejecutar el comando *make all* en el directorio principal. El resto de las opciones están especificadas en el README del proyecto.

10. Instrucciones para la configuración

El servidor se configura mediante parámetros en la ejecución del deamon, los mismos se encuentran especificados en el manual (httpd.8). Para configurar el servidor DoH con el cual trabaja el deamon tenemos los siguientes parámetros:

- Para establecer la dirección del servidor DoH:

 --doh-ip <dirección-doh>

 (Si no se especifica toma como valor por defecto 127.0.0.1)

- Para establecer el puerto del servidor DoH:

 --doh-port <port>

 (Si no se especifica toma como valor por defecto 8053)

- Para establecer el valor del header host:

 --doh-host <hostname>

 (Si no se especifica toma como valor por defecto localhost)

- Para establecer el path del request DoH:

 --doh-path <path>

 (Si no se especifica toma como valor por defecto /getnsrecord)

- Para establecer el query string si el request DoH utiliza el método por defecto ?dns=:

 --doh-query <query>

Además también se puede configurar otras funcionales particulares por parámetro como las siguientes:

- Para establecer la dirección donde servirá el servicio de management:

 -l <dirección-http>

 (Si no se especifica se escucha en todas las interfaces)

- Para establecer la dirección donde servirá el servicio de management:

 -L <dirección-de-management>

 (Por defecto escucha únicamente en loopback)

- Para especificar el puerto donde se encuentra el servidor de management:

 -o <puerto-de-management>

(Si no se especifica el servidor de managment toma 9090 como puerto por defecto)

- Para especificar el puerto TCP donde se escuchara por conexiones entrantes HTTP:

-p <puerto-local>

(Si no se especifica se escucharán conexiones entrantes HTTP en el puerto 8080 por defecto).

- Para deshabilitar los passwords dissectors:

-N

11. Documento de diseño del proyecto

Carpetas

- connections_manager
 - connections_manager.c: Dentro de este archivo se encuentran las funciones encargadas de crear, iniciar, aceptar, y cerrar las nuevas conexiones. Además, están las responsables de la lectura/escritura en tanto cliente como servidor.
 - connections_manager_def.h: Dentro del archivo se encuentran definidos los diferentes estados de la conexión, los tipos de errores y la estructura que posee la información de cada conexión.
- logger
 - logger.c: Dentro de este archivo se encuentran las funciones encargadas de los logs no bloqueantes.
- management
 - management.c: Dentro de este archivo se encuentra la función de inicialización de la estructura dedicada al protocolo de nuestra autoría.
- management_client
 - httpdctl.c: Dentro de este archivo se encuentran las funciones de la aplicación del cliente, el cual se conecta con el servidor y se autentica con una passphrase.
- metrics
 - metrics.c: Dentro de este archivo se encuentran las funciones encargadas del manejo de métricas (cantidad de bytes recibidos y enviados, conexiones históricas, conexiones concurrentes, conexiones fallidas).
- parsers
 - Dentro de esta carpeta se encuentran los parsers que son utilizados a lo largo de todo el proyecto: doh_parser (para la response), headers_parser, percy_request_parser, request_line_parser, sniffer_parser y status_line_parser.

- state_machine
 - stm.c: Dentro de este archivo se encuentran las funciones encargadas de inicializar la máquina de estados y manejar los eventos del selector.
- states
 - Dentro de esta carpeta se encuentran los archivos que definen cada estado con su funcionamiento correspondiente.
- utils
 - Dentro de estas carpetas se encuentran las utilidades de varias secciones del proyecto, además, está presente una implementación del decoder/encoder de base64 con licencia BSD
[\(http://web.mit.edu/freebsd/head/contrib/wpa/src/utils/base64.c\).](http://web.mit.edu/freebsd/head/contrib/wpa/src/utils/base64.c)