TP INTEGRADOR TELEINFORMATICA Y REDES

Mail: tobiasavila04@gmail.com

Alumno: Tobias Avila

Legajo: 195327

DNI: 46639788

Fecha: 24/06/25

Curso 2025

En el laboratorio

1. Descargue el labo base2 e implemente en el laboratorio de Kathara las redes de acuerdo al plano de topología de la Figura fig:topologia.

2. Configure las interfaces, ruteadores/rutas y resolvers. El servidor DNS maneja la zona tpfinal-tyr.com (ya se encuetra configurado). Revise qué servicios debe iniciar y los puertos en los que operan los servidores web y el proxy. El equipo de usuario (Usuario) accede al exterior a través del servidor proxy (revise la configuración para reconocer cómo es en este caso)

Servidor datos:

```
chown www-data:www-data /usr/lib/cgi-bin/datos.pl
chmod 750 /usr/lib/cgi-bin/datos.pl
sed -i 's/\KeepAlive On/KeepAlive Off/' /etc/apache2/apache2.conf
a2enmod cgid
ip a add 12.12.0.12/24 dev eth0
ip link set dev eth0 up
ip route add default via 12.12.0.1
service apache2 start
echo "nameserver 8.8.8.8" >> /etc/resolv.conf
```

DNS:

```
chmod 755 /etc
chmod 755 /etc/bind
ip a add 8.8.8.8/28 dev eth0
ip link set dev eth0 up
ip route add default via 8.8.8.1
service bind start
```

pcusuario:

```
source /root/.bashrc
ip a add 10.10.0.10/24 dev eth0
ip link set dev eth0 up
ip route add default via 10.10.0.1
echo "nameserver 8.8.8.8" >> /etc/resolv.conf
```

Proxy:

```
echo "visible_hostname proxy.tpfinal-tyr.com" >> /etc/squid/squid.conf
sed -i 's/\#http_access allow localnet/http_access allow all/' /etc/squid/squid.conf
#rm -r /var/spool/squid/*
#squid -z
ip a add 10.10.0.1/24 dev eth0
ip link set dev eth0 up
ip route add default via 10.10.0.100
echo "nameserver 8.8.8.8" >> /etc/resolv.conf
service squid start
```

WWW:

```
sed -i 's/\KeepAlive On/KeepAlive Off/' /etc/apache2/apache2.conf
ip a add 12.12.0.11/24 dev eth0
ip link set dev eth0 up
ip route add default via 12.12.0.1
```

service apache2 start
echo "nameserver 8.8.8.8" >> /etc/resolv.conf

Router 1:

ip a add 190.7.231.40/24 dev eth0
ip link set dev eth0 up
ip a add 10.10.0.100/24 dev eth1
ip link set dev eth1 up
ip route add 10.10.0.0/24 dev eth1
ip route add 8.8.8.0/28 via 190.7.231.30
ip route add 12.12.0.0/24 via 190.7.231.20

Router 2:

ip a add 190.7.231.30/24 dev eth0
ip link set dev eth0 up
ip a add 8.8.8.1/28 dev eth1
ip link set dev eth1 up
ip route add 8.8.8.0/28 dev eth1
ip route add 10.10.0.0/24 via 190.7.231.40
ip route add 12.12.0.0/24 via 190.7.231.20

Router 3:

ip a add 190.7.231.20/24 dev eth0
ip link set dev eth0 up
ip a add 12.12.0.1/24 dev eth1
ip link set dev eth1 up
ip route add 12.12.0.0/24 dev eth1
ip route add 10.10.0.0/24 via 190.7.231.40
ip rpute add 8.8.8.0/28 via 190.7.231.30

1. Descarte los mensajes que no pertenecen específicamente a esta transferencia. Ordene las PDUs por su marca de tiempo y numérelas (no use el número de secuencia asignado durante la captura). Agregue en su interface de Wireshark las direcciones de capa 2 para facilitar la identificación real de origen/destino de los mensajes.

En la captura se observaron diversos mensajes, de los cuales me parece que los mensajes de solicitudes y respuestas ICMP entre el proxy (10.10.0.1) y el servidor web (12.12.0.11) no pertenecen específicamente a la transferencia de la página HTML solicitado por el host usuario, lo cual me pareció que podrían ser descartados ya que son mensajes de red para probar conectividad, no para entregar o recibir contenido de web, lo que no descarta que puede ser parte de la configuración donde el proxy quiere confirmar que el servidor esta “vivo” antes de reenviar la solicitud HTTP. Estos mensajes ICMP son los siguientes:

```
"32","0.026258888","10.10.0.1","12.12.0.11","ICMP","78","Echo (ping)
request  id=0x5600, seq=256/1, ttl=63 (reply in
33)","f6:c7:16:9a:24:98","ee:d2:54:85:0e:55"

"33","0.026392281","12.12.0.11","10.10.0.1","ICMP","78","Echo (ping)
reply    id=0x5600, seq=256/1, ttl=63 (request in
32)","ee:d2:54:85:0e:55","f6:c7:16:9a:24:98"
```

```
"59","0.025650853","10.10.0.1","12.12.0.11","ICMP","78","Echo (ping)
request  id=0x5600, seq=256/1, ttl=64 (reply in
60)","52:a3:9f:43:07:f1","2e:59:45:5f:46:19"

"60","0.026343087","12.12.0.11","10.10.0.1","ICMP","78","Echo (ping)
reply    id=0x5600, seq=256/1, ttl=62 (request in
59)","2e:59:45:5f:46:19","52:a3:9f:43:07:f1"

"80","0.026179096","10.10.0.1","12.12.0.11","ICMP","78","Echo (ping)
request  id=0x5600, seq=256/1, ttl=62 (reply in
81)","e6:21:1e:d0:fe:d9","e2:2f:fa:78:b0:0a"

"81","0.026228129","12.12.0.11","10.10.0.1","ICMP","78","Echo (ping)
reply    id=0x5600, seq=256/1, ttl=64 (request in
80)","e2:2f:fa:78:b0:0a","e6:21:1e:d0:fe:d9"
```

Otros mensajes que no pertenecen específicamente a esta transferencia son los mensajes ARP posterior a la finalización de la transferencia principal de datos del análisis, y no aportan al cálculo del overhead ni al volumen de datos útiles transmitidos. Solo se utilizan para resolución de direcciones MAC y no forman parte de una PDU de aplicación. Estos mensajes ARP son los siguientes:

```
224  5.229254040    d6:12:4d:4b:a9:1a   82:92:89:43:39:f5   ARP  60
     Who has 12.12.0.1? Tell 12.12.0.11 d6:12:4d:4b:a9:1a
     82:92:89:43:39:f5

225  5.229260171    2a:d4:bf:1c:5a:51   82:92:89:43:39:f5   ARP  60
     Who has 12.12.0.1? Tell 12.12.0.12 2a:d4:bf:1c:5a:51
     82:92:89:43:39:f5

222  5.229343790    3a:6a:7b:76:b1:e6   8e:79:92:29:01:45   ARP  60
     Who has 8.8.8.1? Tell 8.8.8.8 3a:6a:7b:76:b1:e6
     8e:79:92:29:01:45

226  5.229448158    82:92:89:43:39:f5   d6:12:4d:4b:a9:1a   ARP  60
     12.12.0.1 is at 82:92:89:43:39:f5 82:92:89:43:39:f5
     d6:12:4d:4b:a9:1a

227  5.229451765    82:92:89:43:39:f5   2a:d4:bf:1c:5a:51   ARP  60
     12.12.0.1 is at 82:92:89:43:39:f5 82:92:89:43:39:f5
     2a:d4:bf:1c:5a:51

223  5.229507170    8e:79:92:29:01:45   3a:6a:7b:76:b1:e6   ARP  60
     8.8.8.1 is at 8e:79:92:29:01:45   8e:79:92:29:01:45
     3a:6a:7b:76:b1:e6

214  5.229637077    6a:c1:8a:1f:a2:79   3e:bc:65:7b:ff:6b   ARP  60
     Who has 190.7.231.40? Tell 190.7.231.30 6a:c1:8a:1f:a2:79
     3e:bc:65:7b:ff:6b

215  5.229641756    1e:85:ce:2d:bf:cb   3e:bc:65:7b:ff:6b   ARP  60
     Who has 190.7.231.40? Tell 190.7.231.20 1e:85:ce:2d:bf:cb
     3e:bc:65:7b:ff:6b

216  5.229737137    3e:bc:65:7b:ff:6b   6a:c1:8a:1f:a2:79   ARP  60
     190.7.231.40 is at 3e:bc:65:7b:ff:6b   3e:bc:65:7b:ff:6b
     6a:c1:8a:1f:a2:79
```

```
217  5.229740703    3e:bc:65:7b:ff:6b   1e:85:ce:2d:bf:cb   ARP  60
     190.7.231.40 is at 3e:bc:65:7b:ff:6b    3e:bc:65:7b:ff:6b
     1e:85:ce:2d:bf:cb

218  5.229814113    f2:a7:0b:0e:6e:e7   36:7a:fe:7e:ca:de   ARP  60
     Who has 10.10.0.1? Tell 10.10.0.100     f2:a7:0b:0e:6e:e7
     36:7a:fe:7e:ca:de

219  5.229880148    36:7a:fe:7e:ca:de   e2:eb:f0:ca:69:3a   ARP  60
     Who has 10.10.0.10? Tell 10.10.0.1 36:7a:fe:7e:ca:de
     e2:eb:f0:ca:69:3a

220  5.229940272    e2:eb:f0:ca:69:3a   36:7a:fe:7e:ca:de   ARP  60
     10.10.0.10 is at e2:eb:f0:ca:69:3a e2:eb:f0:ca:69:3a
     36:7a:fe:7e:ca:de

221  5.229998493    36:7a:fe:7e:ca:de   f2:a7:0b:0e:6e:e7   ARP  60
     10.10.0.1 is at 36:7a:fe:7e:ca:de  36:7a:fe:7e:ca:de
     f2:a7:0b:0e:6e:e7
```

En la planilla Excel adjuntada, en la hoja “PDUs por tiempo” contiene la captura ordenada por tiempo, ¡también se adjunta la captura en el envio del trabajo!

2. Confeccione cuadros con la distribución de mensajes por protocolo por capa (Indique claramente qué protocolo/s ubica en cada capa y por qué).

Como aclaración acá se tuvieron en cuenta todos los mensajes, los mensajes ICMPv4 tambien porque más allá de que no me parecieron sumamente importantes para la transferencia del archivo si esta bueno ver una descripción del protocolo, de la cantidad de PDUs que influyeron en la captura y formarlo parte del análisis

	TOTAL DE PDUS: 227 TOTAL DE BYTES 28360						
	CAPA	PROTOCOLO	CANTIDAD DE PDUS	BYTES	DISTRIBUCION PDUS (CANTIDAD DE PDUS / TOTAL DE PDUs	DISTRIBUCION BYTES (BYTES / TOTAL DE BYTES)	DESCRIPCION
Su función es lograr una comunicación confiable entre equipos adyacentes, gestiona el intercambio entre dos dispositivos que están interconectados por algún medio. Transforma el nivel físico en un enlace fiable libre de errores, realizando control de errores, de secuencia y de flujo.	ENLACE	Ethernet	227	28360	100%	100%	protocolo de comunicación de red que opera en la capa de enlace de datos . Su función principal es permitir la transmisión fiable de datos en redes locales. Ethernet define el formato de las tramas, las direcciones físicas (direcciones MAC) usadas para identificar cada dispositivo en la red, y los métodos de acceso al medio físico compartido, como el método CSMA/CD (Carrier Sense Multiple Access with Collision Detection)
		ARP	30	1380	13.2%	4.9%	ARP es un protocolo que se encarga de mapear una dirección lógica a una dirección física a partir de una dirección IP y opera dentro de la red local (enlace). ARP es fundamental para que un datagrama IP sea encapsulado en una trama de enlace y atravesar la red física subyacente
Se encarga de conectar equipos que están en redes diferentes, permitiendo que los datos atraviesen distintas redes interconectadas (ruteo de paquetes) desde un origen hasta un destino, sin importar qué caminos ni qué tecnologías haya en el medio. Liberando a las capas superiores de la necesidad de tener conocimiento sobre la transmisión de datos subyacentes y las tecnologías de conmutación utilizadas para conectar los sistemas.	RED	IPv4	197	3940	86.8%	13.9%	Éste protocolo parece estar implícito en la captura pero su función principal es conectar equipos que está en redes diferentes, permitiendo que los datos atraviesen distintas redes interconectadas (ruteo de paquetes) desde un origen hasta el destino, sin importar que caminos ni tecnologías haya en el medio
		ICMP4 (NO LAS TUVE EN CUENTA YA QUE NO ME PARECIA SUMAMENTE ESPECIAL PARA LA RECUPERACION DEL ARCHIVO)	6	264	2.6%	0.9%	es un protocolo de control y diagnóstico utilizado por dispositivos de red, como routers y hosts, para enviar mensajes de error, advertencia o diagnóstico relacionados con el procesamiento de paquetes IP. No se utiliza para transportar datos de aplicaciones, sino para apoyar la funcionalidad del protocolo IP.
función es lograr una comunicación confiable entre sistemas finales (extremo a extremo), asegurando que los datos lleguen en el mismo orden en que han sido enviados, y sin errores.	TRANSPORTE	TCP	131	15983	57.7%	56.4%	TCP es un protocolo de la capa de transporte que brinda un servicio orientado a conexión, confiable y basado en un flujo continuo de bytes. Orientado a conexión significa que las aplicaciones que desean comunicarse mediante TCP deben primero establecer una conexión, mediante un mecanismo de establecimiento de conexión conocido como three-way handshake. Una vez establecida, TCP implementa mecanismos de control de flujo, detección de
		UDP	60	480	26.4%	1.7%	UDP es un protocolo de transporte no orientado a conexión, sin control de errores ni de flujo. Es más simple y rápido que TCP, pero no garantiza entrega ni orden de los mensajes. Se utiliza en aplicaciones donde la velocidad es prioritaria o donde la aplicación maneja la confiabilidad.
El nivel de aplicación brinda los servicios de red a las aplicaciones. Proporciona las interfaces de usuario y el soporte para servicios, Proporciona una comunicación entre procesos o aplicaciones en computadoras distintas.	APLICACIÓN	DNS	60	3135	26.4%	11.1%	protocolo de la capa de aplicación utilizado para la resolución de nombres de dominio en direcciones IP. DNS opera principalmente mediante mensajes de consulta y respuesta, usualmente sobre el protocolo UDP y el puerto 53, aunque puede usar TCP para transferencias de zona o respuestas grandes. La consulta puede ser recursiva o iterativa, y en cada caso puede involucrar múltiples servidores DNS jerárquicos, desde el servidor raíz hasta los servidores autoritativos del dominio buscado.
		HTTP	12	5855	5.3%	20.6%	diseñado para la transferencia de información entre clientes y servidores web. Opera bajo un modelo de solicitud-respuesta, donde un cliente (como un navegador o un proxy) envía una petición HTTP y el servidor responde con los datos solicitados, generalmente documentos HTML, imágenes u otros recursos web. HTTP es un protocolo sin estado, lo que significa que no conserva información sobre solicitudes anteriores entre una misma sesión, aunque pueden implementarse mecanismos como cookies para mantener cierta persistencia. Las solicitudes incluyen métodos como GET, POST, PUT o DELETE, y pueden contener cabeceras que aportan información adicional sobre el recurso, el cliente o el tipo de contenido esperado.

3. Identifique todas las conexiones TCP. Por cada una, indique: finalidad, sockets de cliente y servidor, segmentos de apertura y cierre. Muestre los parámetros intercambiados durante la fase de apertura

CONEXIÓN TCP "1" ENTRE EL USUARIO Y EL PROXY: durante la captura se identificó una primera conexión establecida entre el cliente (10.10.0.10) y el proxy (10.10.0.1), cuya finalidad es permitir establecer la comunicación del host usuario con el servidor Proxy, para que usuario le pueda realizar una solicitud HTTP hacia un servidor web externo. Como el cliente no tiene acceso directo al exterior, todo el tráfico HTTP debe pasar por el proxy quien actúa de intermediario. Esta conexión representa la base para que el usuario solicite una página web y el proxy retransmita dicha solicitud al servidor web real.

SOCKETS:

- Usuario: 10.10.0.10, puerto efímero 44658
- Servidor Proxy: 10.10.0.1, puerto 3128

Para establecer la conexión entre usuario y proxy se realiza una apertura de la conexión la cual consta de los siguientes segmentos:

1. El usuario (10.10.0.10) inicia la conexión enviando un segmento SYN al proxy, indicando que desea establecer una conexión. En este segmento también se incluye un tamaño máximo de segmento (MSS) de 1460 bytes, habilita el uso de SACK (selective acknowlendgment) para optimizar la gestión de pequetes perdidos, incluye un timestamp con TSval = 3365087325 y TSecr = 0 para sincronización y también se puede usar mecanismo para detectar retransmisiones espurias (aquellas retransmisión que se hacen sin haber perdida real de paquetes, y si no se detecta, el emisor puede bajar el rendimiento al enviar datos debido a que piensa que hay congestión en la red), y por último se propone un escala de ventana (WS) de 128 lo que esta opción de TCP puede permitir incrementar efectivamente la capacidad del campo de anuncio de ventana de TCP de 16 bits, especialmente útil en redes con gran producto retardo-ancho de banda (por ejemplo, enlaces de larga distancia y alta capacidad), donde una venta pequeña limitaría el rendimiento.
2. El proxy responde con un segmento SYN-ACK aceptando la conexión. En esta respuesta confirma los parámetros recibidos y presenta sus propios valores: un MSS de 1460 bytes también, SACK permitido, su timestamp con TSval=33576411314 y TSecr = 3365087325 y la misma escala de venta de 128
3. Finalmente, el cliente envía un segmento ACK confirmando la recepción del SYN-ACK del proxy y con este intercambio queda establecida la conexión TCP y comienza él envió de los datos de la aplicación que en este caso es la solicitud HTTP.

Los mensajes involucrados en esta fase de establecimiento para la primera conexión TCP son los siguientes:

```
63   0.008056139    10.10.0.10     10.10.0.1 TCP  74   44658 → 3128
[SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3365087325
TSecr=0 WS=128 e2:eb:f0:ca:69:3a   36:7a:fe:7e:ca:de

64   0.008131572    10.10.0.1 10.10.0.10     TCP  74   3128 → 44658
[SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM
TSval=3576411314 TSecr=3365087325 WS=128    36:7a:fe:7e:ca:de
     e2:eb:f0:ca:69:3

65   0.009588077    10.10.0.10     10.10.0.1 TCP  66   44658 → 3128
[ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3365087325 TSecr=3576411314
     e2:eb:f0:ca:69:3a   36:7a:fe:7e:ca:de
```

Antes de pasar con el cierre de la conexión entre el usuario y proxy, es importante destacar que cuando se terminó de establecer la conexión entre usuario-proxy. El usuario envía su solicitud GET http.//www.tpfinal-tyr.com/ al proxy, el proxy al recibir esta solicitud envía un ACK respondiendo a la petición GET del usuario y luego abre una nueva conexión TCP al servidor web para cumplir con la petición del usuario. Por lo que, ambas conexiones (Usuario-Proxy y Proxy-Web) permanecen activas en paralelo durante el intercambio de datos, cuando el proxy reciba los datos y se los reenvié al usuario, y ya la página fue completamente entregada al usuario se cierran ambas conexiones: primero la del proxy con el servidor web y luego la del

proxy con el usuario. Ahora con lo aclarado, pasamos a la finalización del servidor proxy con el usuario, sabiendo lo que pasa antes del cierre.

En el cierre de la conexión, el proxy (10.10.0.1) envía un FIN-ACK con seg= 634, ack=246 al usuario para indicar el cirre de la conexión, este segmento también incluye una ventana de 65024 bytes y opciones de tiempo TSval=3547963083, TSecr=3336638996). El usuario responde con su propio FIN-ACK con seq=246, ack=635 confirmando el FIN enviado por el proxy y enviando su propio FIN para cerrar su lado de la conexión. Finalmente, el proxy responde con un ACK final con seq = 635 y ack=247, confirmando el cierre y la conexión TCP queda completamente cerrada.

Es importante destacar que, según la especificación del protocolo, el usuario entra en estado TIME_WAIT durante un periodo 2MSL (Maximum Segment Lifetime) para que segmentos retrasados en la red no interfieran en nuevas conexiónes, lo cual es raro porque se tendría que usar el mismo efímero pero las posibilidades nunca son cero, también este tiempo sirve en caso de que se pierda el ACK y el usuario pueda volver a enviar su FIN en dicho caso. Los mensajes involucrados en este cierre son los siguientes:

```
88   0.054674655    10.10.0.1 10.10.0.10     TCP  66   3128 → 44658
[FIN, ACK] Seq=634 Ack=246 Win=65024 Len=0 TSval=3576411349
TSecr=3365087328    36:7a:fe:7e:ca:de   e2:eb:f0:ca:69:3

91   0.068931930    10.10.0.10     10.10.0.1 TCP  66   44658 → 3128
[FIN, ACK] Seq=246 Ack=635 Win=64128 Len=0 TSval=3365087373
TSecr=3576411358    e2:eb:f0:ca:69:3a   36:7a:fe:7e:ca:de

92   0.069014046    10.10.0.1 10.10.0.10     TCP  66   3128 → 44658
[ACK] Seq=635 Ack=247 Win=65024 Len=0 TSval=3576411375
TSecr=3365087373    36:7a:fe:7e:ca:de   e2:eb:f0:ca:69:3a
```

CONEXION TCP “2” ENTRE EL PROXY Y EL SERVIDOR WEB: esta conexión tiene como finalidad permitir al proxy contactar directamente al servidor web (12.12.0.11) para reenviar una solicitud HTTP. EL proxy actúa como cliente en esta conexión y retransmite la petición GET del usuario estableciendo una conexión TCP con el servidor web para obtener la respuesta y luego reenviarla al usuario.

SOCKETS:

- Proxy: 10.10.0.1, puerto efímero 51626
- Servidor web: 12.12.0.11, puerto 80

Para establecer la conexión entre el proxy y el servidor web se realiza una apertura de la conexión la cual consta de los siguientes segmentos:

1. El proxy (10.10.0.1) envía un segmento SYN al servidor web indicando que quiere establecer una conexión. En este segmento se propone un tamaño máximo de segmento (MSS) de 1460 bytes indicando el tamaño máximo de carga útil en cada segmento, excluyendo cabeceras IP/TCP, habilita el uso de SACK (selective acknowlendgment) para optimizar la gestión de pequetes perdidos, incluye un timestamp con TSval = 1201228506 y TSecr = 0 para sincronización y tambien se puede usar mecanismo para detectar retransmisiones espurias (aquellas retransmisión que se hacen sin haber perdida real de paquetes, y si no se detecta, el emisor puede bajar el

rendimiento al enviar datos debido a que piensa que hay congestion en la red), y por último se propone un escala de ventana (WS) de 128 lo que esta opción de TCP puede permitir incrementar efectivamente la capacidad del campo de anuncio de ventana de TCP de 16 bits, especialmente útil en redes con gran producto retardo-ancho de banda (por ejemplo, enlaces de larga distancia y alta capacidad), donde una venta pequeña limitaría el rendimiento.

2. El servidor web (12.12.0.11) responde con un segmento SYN-ACK, aceptando la conexión y devolviendo los parámetros recibidos. Informa su MSS de 1460 bytes, SACK habilitado, envía su propio timestamp con TSval=827018020 y TSecr =1201228506, y coincide en la escala de ventana de 128.
3. Finalmente, el proxy envia un ACK confirmando la apertura de la conexión, lo que habilita el inicio de intercambio de datos correspondiente a la solicitud y posterior respuesta HTTP

Los mensajes involucrados en la fase de establecimiento de esta conexión TCP son los siguientes:

```
174  0.023429121    10.10.0.1 12.12.0.11     TCP  74   [TCP
Retransmission] 51626 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
SACK_PERM TSval=1201228506 TSecr=0 WS=128   82:92:89:43:39:f5
     d6:12:4d:4b:a9:1ª

14   0.023520004    12.12.0.11     10.10.0.1 TCP  74   80 → 51626 [SYN,
ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=827018020
TSecr=1201228506 WS=128  1e:85:ce:2d:bf:cb   3e:bc:65:7b:ff:6b

175  0.023569558    12.12.0.11     10.10.0.1 TCP  74   [TCP
Retransmission] 80 → 51626 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
MSS=1460 SACK_PERM TSval=827018020 TSecr=1201228506 WS=128
     d6:12:4d:4b:a9:1a   82:92:89:43:39:f5

72   0.023611768    10.10.0.1 12.12.0.11     TCP  74   [TCP
Retransmission] 51626 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
SACK_PERM TSval=1201228506 TSecr=0 WS=128   36:7a:fe:7e:ca:de
     f2:a7:0b:0e:6e:e7

73   0.023683123    12.12.0.11     10.10.0.1 TCP  74   [TCP
Retransmission] 80 → 51626 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
MSS=1460 SACK_PERM TSval=827018020 TSecr=1201228506 WS=128
     f2:a7:0b:0e:6e:e7   36:7a:fe:7e:ca:de

15   0.023778484    10.10.0.1 12.12.0.11     TCP  66   51626 → 80 [ACK]
Seq=1 Ack=1 Win=64256 Len=0 TSval=1201228507 TSecr=827018020
     3e:bc:65:7b:ff:6b   1e:85:ce:2d:bf:cb
```

Es importante destacar que los segmentos marcados como [TCP Retransmisiones] no son retransmisiones por perdida reales de paquetes o errores de la red, esta situación se debe a la metodología de captura utilizada, el tráfico fue capturado en diferentes interfaces del router, lo que puede llevar a provocar que wireshark detecte el mismo segmento más de una vez y lo interprete como retransmisión, cuando en realidad se trata de un mismo paquete observado en distintos puntos de la red. Si prestamos atención puede observarse las direcciones MAC y comprobar que las “retransmisiones” no son por perdida real sino porque wireshark las detecta en diferentes interfaces y las interpreta como retransmisiones, observando los segmentos con el mismo contenido presentan MAC diferentes evidencias que los paquetes fueron capturados en diferentes interfaces del router.

Tras completar el establecimiento de la conexión, y retransmitir al servidor web la solicitud HTTP del usuario, el proxy recibe como respuesta un mensaje HTTP 200 ok acompañado del contenido solicitado lo que indica que la petición fue exitosa y el proxy le envía un ACK al servidor web indicando que recibió correctamente el contenido solicitado. Esta interacción ocurre dentro del flujo de datos TCP, donde los segmentos transportan los mensajes de aplicación. Una vez finalizada la transferencia se inicia el cierre de la conexión TCP entre el proxy y el servidor web.

El cierre comienza con el servidor web (12.12.0.11) enviando un segmento con las banderas FIN y ACK activadas. Este segmento tiene como puerto de origen el 80 (HTTP) y como puerto de destino el 51626, que es el puerto efímero utilizado por el proxy. La secuencia relativa del segmento es 494, y el ACK es 348, lo que indica que el servidor ha recibido todos los datos del proxy hasta ese punto. Este segmento informa un tamaño de la ventana de 64896 bytes, y se incluyen marcas de tiempo con valores TSval=798569773 y TSecr=1172780258. En respuesta, el proxy envía su propio segmento FIN-ACK, también con puertos 34122 → 80, confirmando el cierre por parte del servidor. La ventana de recepción indicada es de 64128 bytes y también incluye marcas de tiempo TSval=827018039 y TSecr=1201228526. Finalmente, el servidor web responde con un segmento ACK, lo que confirma la recepción del FIN del proxy y completa la secuencia de cierre. Se mantienen los mismos valores de ventana y timestamps (TSval=798569774, TSecr=1172780260). Los mensajes involucrados en este cierre son:

```
21   0.037446160    12.12.0.11     10.10.0.1 TCP  66   80 → 51626 [FIN,
ACK] Seq=494 Ack=348 Win=64896 Len=0 TSval=827018033 TSecr=1201228508
     1e:85:ce:2d:bf:cb   3e:bc:65:7b:ff:6b

182  0.037496305    12.12.0.11     10.10.0.1 TCP  66   [TCP
Retransmission] 80 → 51626 [FIN, ACK] Seq=494 Ack=348 Win=64896 Len=0
TSval=827018033 TSecr=1201228508  d6:12:4d:4b:a9:1a
     82:92:89:43:39:f5

80   0.037750237    12.12.0.11     10.10.0.1 TCP  66   [TCP
Retransmission] 80 → 51626 [FIN, ACK] Seq=494 Ack=348 Win=64896 Len=0
TSval=827018033 TSecr=1201228508  f2:a7:0b:0e:6e:e7
     36:7a:fe:7e:ca:de

184  0.042961319    10.10.0.1 12.12.0.11     TCP  66   [TCP
Retransmission] 51626 → 80 [FIN, ACK] Seq=348 Ack=495 Win=64128 Len=0
TSval=1201228526 TSecr=827018033  82:92:89:43:39:f5
     d6:12:4d:4b:a9:1ª

23   0.042807027    10.10.0.1 12.12.0.11     TCP  66   51626 → 80 [FIN,
ACK] Seq=348 Ack=495 Win=64128 Len=0 TSval=1201228526 TSecr=827018033
     3e:bc:65:7b:ff:6b   1e:85:ce:2d:bf:cb

82   0.043156600    10.10.0.1 12.12.0.11     TCP  66   [TCP
Retransmission] 51626 → 80 [FIN, ACK] Seq=348 Ack=495 Win=64128 Len=0
TSval=1201228526 TSecr=827018033  36:7a:fe:7e:ca:de
     f2:a7:0b:0e:6e:e7

83   0.043227735    12.12.0.11     10.10.0.1 TCP  66   80 → 51626 [ACK]
Seq=495 Ack=349 Win=64896 Len=0 TSval=827018039 TSecr=1201228526
     f2:a7:0b:0e:6e:e7   36:7a:fe:7e:ca:de
```

CONEXIÓN TCP “3” ENTRE EL USUARIO Y EL PROXY: se identificó una nueva conexión TCP establecida entre el cliente (10.10.0.10) y el proxy (10.10.0.1), con la finalidad de realizar

una segunda solicitud HTTP a través del proxy. En este caso, el usuario busca acceder al recurso http://datos.tpfinal-tyr.com/idex.html por lo que inicia una nueva conexión TCP con el proxy para poder transmitir esta solicitud.

SOCKETS:

- Usuario: 10.10.0.10, puerto efímero 44664
- Servidor Proxy: 10.10.0.1, puerto 3128

Al igual que la primera conexión se realiza una apertura de la conexión que tiene en común los parámetros con la primera conexión con un único cambio en el timestamp, el establecimiento consta de los siguientes segmentos:

1. El usuario (10.10.0.10) inicia la conexión enviando un segmento SYN al proxy, indicando que desea establecer una conexión. En este segmento también se incluye un tamaño máximo de segmento (MSS) de 1460 bytes, habilita el uso de SACK (selective acknowlendgment) para optimizar la gestión de pequetes perdidos, incluye un timestamp con TSval = 3365087389 y TSecr = 0 para sincronización y también se puede usar mecanismo para detectar retransmisiones espurias (aquellas retransmisión que se hacen sin haber perdida real de paquetes, y si no se detecta, el emisor puede bajar el rendimiento al enviar datos debido a que piensa que hay congestión en la red), y por último se propone un escala de ventana (WS) de 128 lo que esta opción de TCP puede permitir incrementar efectivamente la capacidad del campo de anuncio de ventana de TCP de 16 bits, especialmente útil en redes con gran producto retardo-ancho de banda (por ejemplo, enlaces de larga distancia y alta capacidad), donde una venta pequeña limitaría el rendimiento.
2. El proxy (10.10.0.1) responde con un segmento SYN-ACK aceptando la conexión. En esta respuesta confirma los parámetros recibidos y presenta sus propios valores: un MSS de 1460 bytes también, SACK permitido, su timestamp con TSval=3576411379 y TSecr=3365087389 reflejando correctamente el timestamp enviado por el cliente, y se indica la misma escala de venta de 128
3. Finalmente, el cliente envía un segmento ACK confirmando la recepción del SYN-ACK del proxy y con este intercambio queda establecida la conexión TCP y comienza el envió de los datos de la aplicación que en este caso es la solicitud HTTP.

Inmediatamente después del establecimiento, el usuario transmite su solicitud HTTP completa mediante un segmento de 363 bytes que contiene el GET correspondiente a la URL deseada. El proxy recibe dicha solicitud y responde con un ACK confirmando la recepción de petición enviada por el usuario

Los mensajes involucrados en esta fase de establecimiento para la primera conexión TCP son los siguientes:

```
99   0.072239349    10.10.0.10     10.10.0.1 TCP  74   44664 → 3128
[SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3365087389
TSecr=0 WS=128 e2:eb:f0:ca:69:3a   36:7a:fe:7e:ca:de

100  0.072310544    10.10.0.1 10.10.0.10     TCP  74   3128 → 44664
[SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM
TSval=3576411379 TSecr=3365087389 WS=128    36:7a:fe:7e:ca:de
     e2:eb:f0:ca:69:3ª
```

```
101  0.072527355    10.10.0.10     10.10.0.1 TCP  66   44664 → 3128
[ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3365087390 TSecr=3576411379
     e2:eb:f0:ca:69:3a   36:7a:fe:7e:ca:de

102  0.072996296    10.10.0.10     10.10.0.1 HTTP 363  GET
http://datos.tpfinal-tyr.com/index.html HTTP/1.0 e2:eb:f0:ca:69:3a
     36:7a:fe:7e:ca:de

103  0.073059946    10.10.0.1 10.10.0.10     TCP  66   3128 → 44664
[ACK] Seq=1 Ack=298 Win=64896 Len=0 TSval=3576411379 TSecr=3365087390
     36:7a:fe:7e:ca:de   e2:eb:f0:ca:69:3ª
```

Se puede observar que después del establecimiento en el sgmento 155 el usuario realiza una petición HTTP GET pero al proxy: GET http://datos.tpfinal-tyr.com/index.html HTTP/1.0. A diferencia de la primera conexión (que solicitaba http://www.tpfinal-tyr.com/), aquí se solicita explícitamente otro recurso (la ruta /index.html) ubicado en un subdominio diferente (datos.tpfinal-tyr.com)

Luego de cumplida la transferencia de datos y enviada la respuesta HTTP correspondiente, se inicia la fase de cierre de la conexión. El proxy envía un segmento con las banderas FIN-ACK activas indicando que quiere cerrar la conexión y confirmando datos anteriores. Este segmento incluye las opciones de timestamp manteniendo la coherencia con la conexión y no incluye carga de datos ya que len = 0. El usuario responde enviando su propio FIN-ACK indicando que finaliza su lado de la conexión y confirmando el FIN del proxy, con este cambio ambas partes han indicado que no tienen más datos que enviar, y finalmente, el proxy responde con un ACK confirmando la recepción del FIN del cliente y cerrando la conexión TCP de forma ordenada, liberando los recursos asociados.

Los mensajes involucrados en la fase de cierre:

```
120  0.086298017    10.10.0.1 10.10.0.10     TCP  66   3128 → 44664
[FIN, ACK] Seq=985 Ack=298 Win=64896 Len=0 TSval=3576411387
TSecr=3365087390    36:7a:fe:7e:ca:de   e2:eb:f0:ca:69:3ª

122  0.086873058    10.10.0.10     10.10.0.1 TCP  66   44664 → 3128
[FIN, ACK] Seq=298 Ack=986 Win=64128 Len=0 TSval=3365087404
TSecr=3576411387    e2:eb:f0:ca:69:3a   36:7a:fe:7e:ca:de

123  0.086941749    10.10.0.1 10.10.0.10     TCP  66   3128 → 44664
[ACK] Seq=986 Ack=299 Win=64896 Len=0 TSval=3576411393
TSecr=3365087404    36:7a:fe:7e:ca:de   e2:eb:f0:ca:69:3a
```

CONEXIÓN TCP “4” ENTRE PROXY Y EL SERVIDOR DE DATOS: esta conexión se establece cuando el **proxy** necesita recuperar el recurso solicitado por el usuario (la página index.html del subdominio datos.tpfinal-tyr.com). Como ese recurso reside en otro servidor diferente al de la primera conexión, el proxy inicia una nueva conexión TCP con la IP **12.12.0.12** a través del puerto **80** (puerto estándar HTTP), usando un puerto efímero propio

SOCKETS:

- Proxy: 10.10.0.1, puerto efímero 44564
- Servidor datos: 12.12.0.12, puerto bien conocido 80

El establecimiento consta de los siguientes segmentos:

1. El proxy (10.10.0.1) inicia la conexión enviando un segmento SYN al servidor de datos (12.12.0.12), indicando que desea establecer una conexión. En este segmento se especifica un tamaño máximo de segmento (MSS) de 1460 bytes, habilita el uso de SACK (selective acknowlendgment) para optimizar la gestión de pequetes perdidos, incluye un timestamp con TSval = 4156560722 y TSecr = 0 para sincronización y también se puede usar mecanismo para detectar retransmisiones espurias (aquellas retransmisión que se hacen sin haber perdida real de paquetes, y si no se detecta, el emisor puede bajar el rendimiento al enviar datos debido a que piensa que hay congestión en la red), y por último se propone un escala de ventana (WS) de 128 que permitirá manejar eficientemente buffers más grande si fuera necesario
2. El servidor de datos con un segmento SYN-ACK aceptando la conexión. En esta respuesta confirma los parámetros recibidos y presenta sus propios valores: un MSS de 1460 bytes también, SACK permitido, su timestamp con TSval=1185731650 TSecr=4156560722 y la misma escala de venta de 128
3. Finalmente, el proxy envía un segmento ACK confirmando la recepción del SYN-ACK del servidor de datos y con este intercambio queda establecida la conexión TCP y comienza el envió de los datos de la aplicación que en este caso es la solicitud HTTP.

Los mensajes involucrados en esta fase de establecimiento para la primera conexión TCP son los siguientes:

```
31   0.077050482    10.10.0.1 12.12.0.12     TCP  74   44564 → 80 [SYN]
Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=4156560722 TSecr=0
WS=128    3e:bc:65:7b:ff:6b   1e:85:ce:2d:bf:cb

188  0.077288273    10.10.0.1 12.12.0.12     TCP  74   [TCP
Retransmission] 44564 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
SACK_PERM TSval=4156560722 TSecr=0 WS=128    82:92:89:43:39:f5
     2a:d4:bf:1c:5a:51

32   0.077402189    12.12.0.12     10.10.0.1 TCP  74   80 → 44564 [SYN,
ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1185731650
TSecr=4156560722 WS=128  1e:85:ce:2d:bf:cb   3e:bc:65:7b:ff:6b

189  0.077425313    12.12.0.12     10.10.0.1 TCP  74   [TCP
Retransmission] 80 → 44564 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
MSS=1460 SACK_PERM TSval=1185731650 TSecr=4156560722 WS=128
     2a:d4:bf:1c:5a:51   82:92:89:43:39:f5

106  0.077468365    10.10.0.1 12.12.0.12     TCP  74   [TCP
Retransmission] 44564 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
SACK_PERM TSval=4156560722 TSecr=0 WS=128    36:7a:fe:7e:ca:de
     f2:a7:0b:0e:6e:e7

107  0.077734601    12.12.0.12     10.10.0.1 TCP  74   [TCP
Retransmission] 80 → 44564 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
MSS=1460 SACK_PERM TSval=1185731650 TSecr=4156560722 WS=128
     f2:a7:0b:0e:6e:e7   36:7a:fe:7e:ca:de

33   0.077856953    10.10.0.1 12.12.0.12     TCP  66   44564 → 80 [ACK]
Seq=1 Ack=1 Win=64256 Len=0 TSval=4156560726 TSecr=1185731650
     3e:bc:65:7b:ff:6b   1e:85:ce:2d:bf:cb
```

Una vez establecida la conexión, el proxy procede a enviar la solicitud HTTP tipo GET /index.html HTTP/1.1 conteniendo 397 bytes de carga útil de aplicación. El servidor responde con un ACK que confirma la recepción de la solicitud y luego envía la respuesta HTTP esperada con un encabezado HTTP/1.1 200 ok y contenido tipo text/html ocupando un total de 844 bytes.

Una vez finalizado el envió de respuesta, el servidor inicia el cierre de la conexión con un segmento FIN-ACK informando que no tiene más datos que enviar con seq=845 y ack=398 reconociendo los bytes enviados por el proxy. El proxy envía su propio FIN-ACK indicando que también desea cerra la conexión de su lado con seq=398, ack=845 (ultimo byte recibido del servidor), con timestamp sincronizados. Finalmente, el servidor de datos responde con un ACK al FIN enviado por el proxy concluyendo el cierre ordenado de esta conexión.

Mensajes involucrados en la fase de cierre:

```
37   0.079930259    12.12.0.12     10.10.0.1 TCP  66   80 → 44564 [FIN,
ACK] Seq=845 Ack=398 Win=64768 Len=0 TSval=1185731652 TSecr=4156560726
     1e:85:ce:2d:bf:cb   3e:bc:65:7b:ff:6b

194  0.079964123    12.12.0.12     10.10.0.1 TCP  66   [TCP
Retransmission] 80 → 44564 [FIN, ACK] Seq=845 Ack=398 Win=64768 Len=0
TSval=1185731652 TSecr=4156560726  2a:d4:bf:1c:5a:51
     82:92:89:43:39:f5

112  0.080461056    12.12.0.12     10.10.0.1 TCP  66   [TCP
Retransmission] 80 → 44564 [FIN, ACK] Seq=845 Ack=398 Win=64768 Len=0
TSval=1185731652 TSecr=4156560726  f2:a7:0b:0e:6e:e7
     36:7a:fe:7e:ca:de

38   0.080579000    10.10.0.1 12.12.0.12     TCP  66   44564 → 80 [ACK]
Seq=398 Ack=845 Win=64128 Len=0 TSval=4156560728 TSecr=1185731652
     3e:bc:65:7b:ff:6b   1e:85:ce:2d:bf:cb

195  0.080617664    10.10.0.1 12.12.0.12     TCP  66   [TCP Keep-Alive]
44564 → 80 [ACK] Seq=398 Ack=845 Win=64128 Len=0 TSval=4156560728
TSecr=1185731652    82:92:89:43:39:f5   2a:d4:bf:1c:5a:51

113  0.080658631    10.10.0.1 12.12.0.12     TCP  66   [TCP Keep-Alive]
44564 → 80 [ACK] Seq=398 Ack=845 Win=64128 Len=0 TSval=4156560728
TSecr=1185731652    36:7a:fe:7e:ca:de   f2:a7:0b:0e:6e:e7

39   0.080774501    10.10.0.1 12.12.0.12     TCP  66   44564 → 80 [FIN,
ACK] Seq=398 Ack=845 Win=64128 Len=0 TSval=4156560729 TSecr=1185731652
     3e:bc:65:7b:ff:6b   1e:85:ce:2d:bf:cb

196  0.080871796    10.10.0.1 12.12.0.12     TCP  66   [TCP
Retransmission] 44564 → 80 [FIN, ACK] Seq=398 Ack=845 Win=64128 Len=0
TSval=4156560729 TSecr=1185731652  82:92:89:43:39:f5
     2a:d4:bf:1c:5a:51

114  0.081050515    10.10.0.1 12.12.0.12     TCP  66   [TCP
Retransmission] 44564 → 80 [FIN, ACK] Seq=398 Ack=845 Win=64128 Len=0
TSval=4156560729 TSecr=1185731652  36:7a:fe:7e:ca:de
     f2:a7:0b:0e:6e:e7
```

CONEXIÓN TCP “5” USUARIO Y PROXY: se identificó otra conexión TCP establecida entre el usuario (10.10.0.10) y el proxy (10.10.0.1), con la finalidad de realizar una tercera solicitud HTTP a través del proxy. En este caso, el usuario busca acceder al recurso http://datos.tpfinal-tyr.com/cgi-bin/datos.pl, utilizando el protocolo HTTP/1.0 por lo que inicia una nueva conexión TCP con el proxy para poder transmitir esta solicitud.

1. El proceso de establecimiento comienza con el usuario (10.10.0.10) enviando un segmento SYN al proxy, indicando que desea establecer una conexión. En este segmento también se incluye un tamaño máximo de segmento (MSS) de 1460 bytes, habilita el uso de SACK (selective acknowlendgment) para optimizar la gestión de pequetes perdidos, incluye un timestamp con TSval = 3365087389 y TSecr = 0 para sincronización y también se puede usar mecanismo para detectar retransmisiones espurias (aquellas retransmisión que se hacen sin haber perdida real de paquetes, y si no se detecta, el emisor puede bajar el rendimiento al enviar datos debido a que piensa que hay congestión en la red), y por último se propone un escala de ventana (WS) de 128 lo que esta opción de TCP puede permitir incrementar efectivamente la capacidad del campo de anuncio de ventana de TCP de 16 bits, especialmente útil en redes con gran producto retardo-ancho de banda (por ejemplo, enlaces de larga distancia y alta capacidad), donde una venta pequeña limitaría el rendimiento.
2. El proxy (10.10.0.1) responde con un segmento SYN-ACK aceptando la conexión y reflejando las mismas opciones TCP (MSS=1460, SACK, WS=128). También responde con su propio timestamp (TSval=3576411405 y TSecr=3365087416), y se indica la misma escala de venta de 128
3. Finalmente, el usuario envía un segmento ACK confirmando la recepción del SYN-ACK del proxy y con este intercambio queda establecida la conexión TCP y comienza el envió de los datos de la aplicación que en este caso es la solicitud HTTP.

Una vez realizado el establecimiento el usuario emite una solicitud HTTP GET al proxy con destino http://datos.tpfinal-tyr.com/cgi-bin/datos.pl utilizando el protocolo HTTP/1.0, el tamaño del segmento es de 369 bytes. El proxy responde con un ACK que confirma la recepción del segmento GET lo que indica que ha recibido correctamente la solicitud de la solicitud HTTP. A diferencia de la tercera conexión que se solicito el recurso (http://datos.tpfinal-tyr.com/index.html HTTP/1.0) y primera conexión que solicitaba (http://www.tpfinal-tyr.com/)

Mensajes involucrados en la fase de establecimiento:

```
130  0.098826390    10.10.0.10     10.10.0.1 TCP  74   44668 → 3128
[SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3365087416
TSecr=0 WS=128 e2:eb:f0:ca:69:3a   36:7a:fe:7e:ca:de

131  0.098901402    10.10.0.1 10.10.0.10     TCP  74   3128 → 44668
[SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM
TSval=3576411405 TSecr=3365087416 WS=128    36:7a:fe:7e:ca:de
     e2:eb:f0:ca:69:3ª

132  0.099092986    10.10.0.10     10.10.0.1 TCP  66   44668 → 3128
[ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3365087416 TSecr=3576411405
     e2:eb:f0:ca:69:3a   36:7a:fe:7e:ca:de

133  0.099418404    10.10.0.10     10.10.0.1 HTTP 369  GET
http://datos.tpfinal-tyr.com/cgi-bin/datos.pl HTTP/1.0
     e2:eb:f0:ca:69:3a   36:7a:fe:7e:ca:de
```

```
134  0.099482295    10.10.0.1 10.10.0.10     TCP  66   3128 → 44668
[ACK] Seq=1 Ack=304 Win=64896 Len=0 TSval=3576411405 TSecr=3365087416
     36:7a:fe:7e:ca:de   e2:eb:f0:ca:69:3ª
```

Una vez finalizada la transferencia de datos el proxy inicia el cierre de la conexión enviando un segmento FIN-ACK con seq=528 y ack=304 reconociendo el último byte recibido del usuario. El usuario responde con su propio FIN-ACK confirmando el FIN enviado por el proxy y enviando su propio FIN para cerrar su lado de la conexión. Finalmente, el proxy responde con una ACK final confirmando el FIN del usuario y queda completa el cierre de la conexión TCP de forma ordenada.

Mensajes involucrados en la fase de cierre:

```
147  0.134288218    10.10.0.1 10.10.0.10     TCP  66   3128 → 44668
[FIN, ACK] Seq=528 Ack=304 Win=64896 Len=0 TSval=3576411437
TSecr=3365087416    36:7a:fe:7e:ca:de   e2:eb:f0:ca:69:3ª

148  0.134537882    10.10.0.10     10.10.0.1 TCP  66   44668 → 3128
[FIN, ACK] Seq=304 Ack=529 Win=64128 Len=0 TSval=3365087452
TSecr=3576411437    e2:eb:f0:ca:69:3a   36:7a:fe:7e:ca:de

149  0.134602925    10.10.0.1 10.10.0.10     TCP  66   3128 → 44668
[ACK] Seq=529 Ack=305 Win=64896 Len=0 TSval=3576411441
TSecr=3365087452    36:7a:fe:7e:ca:de   e2:eb:f0:ca:69:3a
```

CONEXIÓN TCP “6” PROXY Y SERVIDOR DE DATOS: esta conexión parece ser la misma que la última conexión entre el proxy y el servidor. En la anterior luego de que el proxy haya solicitado la petición y obtuvo la respuesta del servidor de datos se cerró esa conexión, y después se abrió esta nueva conexión, la cual puede parecer raro porque si miramos el encabezado de la solicitud get /index.html que envió el proxy en la anterior conexion se ve que se usa keep-alive (Conecction: keep-alive) lo que indica la intención del cliente (proxy) de mantener la conexión TCP abierta para posibles solicitudes adicionales. Sin embargo, en la respuesta del servidor de datos incluye en la cabecera “Conecction: close” lo que implica que el servidor decide cerrar la conexión una vez finalizada la transferencia del recurso solicitado. Debido a esta indicación la conexión anterior entre el proxy y el servidor de datos se cierra tras completar la transferencia del archivo index.html y cuando el proxy necesita realizar una nueva solicitud HTTP, como lo es en este caso para el recurso /cgi-bin/datos.pl se establece esta nueva conexión TCP realizando nuevamente el proceso de establecimiento de la conexión.

1. En esta nueva conexión en primer lugar el proxy (10.10.0.1) envia un segmento SYN desde el puerto 44572 al puerto 80 del servidor de datos (12.12.0.12), en el cual incluye parámetros TCP de optimización como tamaño máximo de segmento (MSS) de 1460 bytes indicando lo máximo que puede recibir el emisor, se permite el uso de selective acknowledgment para una recuperación mas eficiente ante perdidas, se incluyen marcas de tiempo timestamp con TSval = 4156560748 y TSecr = 0, utilizado para sincronización y también se puede usar mecanismo para detectar retransmisiones espurias (aquellas retransmisión que se hacen sin haber perdida real de paquetes, y si no se detecta, el emisor puede bajar el rendimiento al enviar datos debido a que piensa que hay congestión en la red), y por último, se propone una escala de ventana (WS) de 128 permitiendo ampliar el tamaño de la ventana de recepción para mejorar el rendimiento en conexiones de alta latencia.
2. El servidor de datos responde con un segmento con los flags SYN y ACK activos aceptando la conexión y devolviendo sus propios parámetros de configuración: también

indica MSS de 1460, habilita SACK, un escalado de ventana de 128 y valores de timestamp Tsval=1185731672 y TSecr=4156560748.

3. Finalmente, el proxy responde con un ACK confirmando el segmento enviado por el servidor de datos y completando asi el establecimiento de la conexión, quedando ambos listos para el intercambio de datos

Mensajes involucrados en la fase de establecimiento:

```
46   0.099923162    10.10.0.1 12.12.0.12     TCP  74   44572 → 80 [SYN]
Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=4156560748 TSecr=0
WS=128    3e:bc:65:7b:ff:6b   1e:85:ce:2d:bf:cb

199  0.100086572    10.10.0.1 12.12.0.12     TCP  74   [TCP
Retransmission] 44572 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
SACK_PERM TSval=4156560748 TSecr=0 WS=128    82:92:89:43:39:f5
     2a:d4:bf:1c:5a:51

200  0.100141176    12.12.0.12     10.10.0.1 TCP  74   [TCP
Retransmission] 80 → 44572 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
MSS=1460 SACK_PERM TSval=1185731672 TSecr=4156560748 WS=128
     2a:d4:bf:1c:5a:51   82:92:89:43:39:f5

47   0.100219154    12.12.0.12     10.10.0.1 TCP  74   80 → 44572 [SYN,
ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1185731672
TSecr=4156560748 WS=128  1e:85:ce:2d:bf:cb   3e:bc:65:7b:ff:6b

135  0.100234403    10.10.0.1 12.12.0.12     TCP  74   [TCP
Retransmission] 44572 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
SACK_PERM TSval=4156560748 TSecr=0 WS=128    36:7a:fe:7e:ca:de
     f2:a7:0b:0e:6e:e7

136  0.100407121    12.12.0.12     10.10.0.1 TCP  74   [TCP
Retransmission] 80 → 44572 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
MSS=1460 SACK_PERM TSval=1185731672 TSecr=4156560748 WS=128
     f2:a7:0b:0e:6e:e7   36:7a:fe:7e:ca:de

137  0.100631286    10.10.0.1 12.12.0.12     TCP  66   44572 → 80 [ACK]
Seq=1 Ack=1 Win=64256 Len=0 TSval=4156560749 TSecr=1185731672
     36:7a:fe:7e:ca:de   f2:a7:0b:0e:6e:e7
```

Una vez que la transferencia del recurso /cgi-bin/datos.pl ha finalizado, el proxy inicia el proceso de cierre de la conexión enviando un segmento FIN, ACK con seq = 399 y ack = 388. El servidor responde con su propio FIN, ACK, confirmando el cierre de su lado, con números de secuencia y acuse invertidos (Seq=388 Ack=400). El proxy finalmente responde con un ACK confirmando el segmento del servidor y quedando asi definitivamente cerrada la conexión de forma ordenada.

Mensajes involucrados en la fase de cierre:

```
211  0.130432759    10.10.0.1 12.12.0.12     TCP  66   [TCP
Retransmission] 44572 → 80 [FIN, ACK] Seq=399 Ack=388 Win=64128 Len=0
TSval=4156560779 TSecr=1185731702  3e:bc:65:7b:ff:6b
     1e:85:ce:2d:bf:cb
```

```
206  0.130499926    10.10.0.1 12.12.0.12     TCP  66   [TCP
Retransmission] 44572 → 80 [FIN, ACK] Seq=399 Ack=388 Win=64128 Len=0
TSval=4156560779 TSecr=1185731702  82:92:89:43:39:f5
     2a:d4:bf:1c:5a:51

142  0.130509775    10.10.0.1 12.12.0.12     TCP  66   44572 → 80 [FIN,
ACK] Seq=399 Ack=388 Win=64128 Len=0 TSval=4156560779 TSecr=1185731702
     36:7a:fe:7e:ca:de   f2:a7:0b:0e:6e:e7

207  0.146583123    12.12.0.12     10.10.0.1 TCP  66   [TCP
Retransmission] 80 → 44572 [FIN, ACK] Seq=388 Ack=400 Win=64768 Len=0
TSval=1185731719 TSecr=4156560779  2a:d4:bf:1c:5a:51
     82:92:89:43:39:f5

212  0.146687170    12.12.0.12     10.10.0.1 TCP  66   [TCP
Retransmission] 80 → 44572 [FIN, ACK] Seq=388 Ack=400 Win=64768 Len=0
TSval=1185731719 TSecr=4156560779  1e:85:ce:2d:bf:cb
     3e:bc:65:7b:ff:6b

150  0.146780367    12.12.0.12     10.10.0.1 TCP  66   80 → 44572 [FIN,
ACK] Seq=388 Ack=400 Win=64768 Len=0 TSval=1185731719 TSecr=4156560779
     f2:a7:0b:0e:6e:e7   36:7a:fe:7e:ca:de
```

4. Para la conexión TCP utilizada para recuperar el archivo .html entre usuario y proxy indique la finalidad de cada PDU intercambiada a nivel de transporte y aplicación.

Genere un diagrama en draw.io indicando cada PDU intercambiada a nivel de transporte y aplicación entre el usuario y el proxy, en el mismo diagrama se explica la finalidad de cada PDU asi como también se refleja el tiempo estando cada PDU una debajo de otra. El diagrama esta en el github con el nombre “PDUs_intercambiadas_punto4”

5. Genere un diagrama de intercambios en el tiempo que muestre cómo sucedieron los mensajes, incluyendo TODOS los dispositivos involucrados. Por cada mensaje, identifique los principales parámetros que hacen a su función. En su gráfico, utilice diferentes colores para las diferentes conexiones (incluya las referencias correspondientes)3.

Se adjunta diagrama de intercambio que incluye todos los dispositivos involucrados en el github con el nombre “Diagrama_de_intercambios”

6. Compare los headers HTTP del requerimiento del cliente al proxy con respecto del realizado por el proxy al servidor web. Muestre y explique las diferencias.

Se identificaron 3 peticiones realizadas por el usuario que luego fueron reenviadas por el proxy al servidor correspondiente, se va a comparar cada uno de estas solicitudes. Empecemos con la primera solicitud que realiza el usuario (10.10.0.10) al proxy (10.10.0.1) GET http://www.tpfinal-tyr.com/ HTTP/1.0, y luego el proxy que actúa como intermediario las reenvía la solicitud al servidor web (12.12.0.11):

Header enviado por el usuario al proxy:

```
Transmission Control Protocol, Src Port: 44658, Dst Port: 3128, Seq: 1, Ack: 1, Len: 245
Hypertext Transfer Protocol
```

```
GET http://www.tpfinal-tyr.com/ HTTP/1.0\r\n
Expert Info (Chat/Sequence): GET http://www.tpfinal-tyr.com/ HTTP/1.0\r\n]
```

```
    Request Method: GET
    Request URI: http://www.tpfinal-tyr.com/
    Request Version: HTTP/1.0
```

```
User-Agent: w3m/0.5.3+git20230121\r\n
Accept: text/html, text/*;q=0.5, image/*, application/*\r\n
Accept-Encoding: gzip, compress, bzip, bzip2, deflate\r\n
Accept-Language: en;q=1.0\r\n
Host: www.tpfinal-tyr.com\r\n
```

Header enviado por el proxy al servidor web:

```
Internet Protocol Version 4, Src: 10.10.0.1, Dst: 12.12.0.11
Transmission Control Protocol, Src Port: 51626, Dst Port: 80, Seq: 1, Ack: 1, Len: 347
Hypertext Transfer Protocol
GET / HTTP/1.1\r\n
[Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
```

```
    [GET / HTTP/1.1\r\n]
    [Severity level: Chat]
    [Group: Sequence]
```

```
Request Method: GET
Request URI: /
Request Version: HTTP/1.1
User-Agent: w3m/0.5.3+git20230121\r\n
Accept: text/html, text/*;q=0.5, image/*, application/*\r\n
Accept-Encoding: gzip, compress, bzip, bzip2, deflate\r\n
Accept-Language: en;q=1.0\r\n
Host: www.tpfinal-tyr.com\r\n
Via: 1.0 proxy.tpfinal-tyr.com (squid/5.7)\r\n
X-Forwarded-For: 10.10.0.10\r\n
Cache-Control: max-age=259200\r\n
Connection: keep-alive\r\n
```

Se observa que el usuario realiza su solicitud al proxy utilizando la versión HTTP/1.1n mientras que el proxy traduce esta solicitud para el servidor web usando HTTP/1.1, esto se debe a que el proxy puede aprovechar las ventajas de HTTP/1.1 como el uso de conexiones persistentes (indicadas por el header Conecction: keep-alive) que permiten optimizar la comunicación con el servidor web y reducir el número de conexiones. También se puede notar que el usuario envía la URL completa “GET http.//www.tpfinal-tyr.com/ HTTP/1.0” porque el proxy necesita saber el destino final. En cambio, el proxy al reenviar la solicitud al servidor web envía solo el path “GET / HTTP/1.1” ya que el servidor web determina el host gracias al campo Host. Otra diferencia son los campos agregados por el proxy:

- Via: 1-0 proxy.tpfinal-tyr.com (squid/5.7): para indicar que la solicitud fue procesada por el proxy y permitir la trazabilidad del recorrido de la petición
- X-Forwarded-For: 10.10.0.10: informa al servidor web cual es la IP original del cliente que inicio la solicitud

- Cache-Control: max-age=259200: establece directivas relacionadas con el almacenamiento en cache para optimizar las respuestas si el contenido no cambia con frecuencia
- Connection: keep-alive: mantiene la conexión abierta para enviar más de una petición sobre la misma conexión permitiendo una mejor latencia que en caso de tener que hacer un establecimiento y cierre por conexión.

Los campos User-Agent, Accept, Accept-Enconding, Accept-Languaje y Host se mantienen sin modificaciones ya que son las características del cliente original.

Ahora se compara la segunda peticion realizada por el usuario al proxy, en este caso “GET http://datos.tpfinal-tyr.com/index.html” y leugo el proxy envia esta peticion al servidor de datos

Header enviado por el usuario al proxy:

```
Frame 102: 363 bytes on wire (2904 bits), 363 bytes captured (2904 bits) on interface eth0, id 0
Transmission Control Protocol, Src Port: 44664, Dst Port: 3128, Seq: 1, Ack: 1, Len: 297
Hypertext Transfer Protocol

GET http://datos.tpfinal-tyr.com/index.html HTTP/1.0\r\n
[Expert Info (Chat/Sequence): GET http://datos.tpfinal-tyr.com/index.html HTTP/1.0\r\n]
```

```
    Request Method: GET
    Request URI: http://datos.tpfinal-tyr.com/index.html
    Request Version: HTTP/1.0
```

```
User-Agent: w3m/0.5.3+git20230121\r\n
Accept: text/html, text/*;q=0.5, image/*, application/*\r\n
Accept-Encoding: gzip, compress, bzip, bzip2, deflate\r\n
Accept-Language: en;q=1.0\r\n
Host: datos.tpfinal-tyr.com\r\n
Referer: http://www.tpfinal-tyr.com/\r\n
```

Header enviado por el proxy a servidor de datos:

```
Frame 34: 463 bytes on wire (3704 bits), 463 bytes captured (3704 bits) on interface eth1, id 1
Transmission Control Protocol, Src Port: 44564, Dst Port: 80, Seq: 1, Ack: 1, Len: 397
Hypertext Transfer Protocol

GET /index.html HTTP/1.1\r\n
[Expert Info (Chat/Sequence): GET /index.html HTTP/1.1\r\n]
```

```
    Request Method: GET
    Request URI: /index.html
    Request Version: HTTP/1.1
```

```
User-Agent: w3m/0.5.3+git20230121\r\n
Accept: text/html, text/*;q=0.5, image/*, application/*\r\n
Accept-Encoding: gzip, compress, bzip, bzip2, deflate\r\n
Accept-Language: en;q=1.0\r\n
Referer: http://www.tpfinal-tyr.com/\r\n
Host: datos.tpfinal-tyr.com\r\n
Via: 1.0 proxy.tpfinal-tyr.com (squid/5.7)\r\n
X-Forwarded-For: 10.10.0.10\r\n
```

```
Cache-Control: max-age=259200\r\n
Connection: keep-alive\r\n
```

El Usuario envia un GET con la versiopn HTTP/1.0 al proxy, incluyendo la URL completa: GET http://datos.tpfinal-tyr.com/index.html HTTP/1.0, esto es necesario porque el proxy debe conocer el destino final del recurso solicitado. El proxy luego reenvía la solicitud al servidor de datos, usando un GET /index.html HTTP/1.1 usando solo el path del recurso porque el servidor de datos ya conoce el host gracias al campo Host. Ademas de cambiar la versión del protocolo a HTTP/1.1, el proxy agrega varios campos adicionales que no estaban en la solicitud original:

- Via: 1-0 proxy.tpfinal-tyr.com (squid/5.7): para indicar que la petición fue procesada por el proxy quien actuó como intermediario
- X-Forwarded-For: 10.10.0.10: informa al servidor web cual es la IP original del cliente que inicio la solicitud
- Cache-Control: max-age=259200: establece políticas relacionadas con el almacenamiento en cache para optimizar las respuestas si el contenido no cambia con frecuencia. El proxy sugiere al servidor que el contenido pueda ser almacenado en cache por hasta 259200 segundos (3 días). Esto puede servir para optimizar el tráfico y reducir solicitudes repetidas
- Connection: keep-alive: mantiene la conexión abierta para enviar más de un petición sobre la misma conexión permitiendo una mejor latencia que en caso de tener que hacer un establecimiento y cierre por conexión.

Los campos User-Agent, Accept, Accept-Enconding, Accept-Languaje y Host se mantienen sin modificaciones ya que son las características del cliente original.

Ahora se compara la tercera y última petición realizada por el usuario al proxy, en este caso “GET http://datos.tpfinal-tyr.com/cgi-bin/datos.pl” y luego el proxy envía esta petición al servidor de datos

```
Header enviado por el usuario al proxy

Internet Protocol Version 4, Src: 10.10.0.10, Dst: 10.10.0.1
Transmission Control Protocol, Src Port: 44668, Dst Port: 3128, Seq: 1, Ack: 1, Len: 303
Hypertext Transfer Protocol

GET http://datos.tpfinal-tyr.com/cgi-bin/datos.pl HTTP/1.0\r\n
[Expert Info (Chat/Sequence): GET http://datos.tpfinal-tyr.com/cgi-bin/datos.pl HTTP/1.0\r\n]
```

```
    [GET http://datos.tpfinal-tyr.com/cgi-bin/datos.pl HTTP/1.0\r\n]
    [Severity level: Chat]
    [Group: Sequence]
```

```
Request Method: GET
Request URI: http://datos.tpfinal-tyr.com/cgi-bin/datos.pl
Request Version: HTTP/1.0
User-Agent: w3m/0.5.3+git20230121\r\n
Accept: text/html, text/*;q=0.5, image/*, application/*\r\n
Accept-Encoding: gzip, compress, bzip, bzip2, deflate\r\n
Accept-Language: en;q=1.0\r\n
Host: datos.tpfinal-tyr.com\r\n
Referer: http://www.tpfinal-tyr.com/\r\n
```

Header enviado por el proxy al servidor de datos:

```
Hypertext Transfer Protocol

GET /cgi-bin/datos.pl HTTP/1.1\r\n
Expert Info (Chat/Sequence): GET /cgi-bin/datos.pl HTTP/1.1\r\n]

    Request Method: GET
    Request URI: /cgi-bin/datos.pl
    Request Version: HTTP/1.1

User-Agent: w3m/0.5.3+git20230121\r\n
Accept: text/html, text/*;q=0.5, image/*, application/*\r\n
Accept-Encoding: gzip, compress, bzip, bzip2, deflate\r\n
Accept-Language: en;q=1.0\r\n
Referer: http://www.tpfinal-tyr.com/\r\n
Host: datos.tpfinal-tyr.com\r\n
Via: 1.0 proxy.tpfinal-tyr.com (squid/5.7)\r\n
X-Forwarded-For: 10.10.0.10\r\n
Cache-Control: max-age=0\r\n
Connection: keep-alive\r\n
```

Se puede observar el cliente vuelve a realizar una peticion HTTP/1.0 al igual que las anteriores comparaciones, con un GET que incluye la URL completa “GET http://datos.tpfinal-tyr.com/cgi-bin/datos.pl HTTP/1.0”, ya que el proxy necesita la URL completa para saber a que servidor dirigirse. El proxy convierte esta solicitud al formato esperado por el servidor de datos, simplificando la URL a solo el path: “GET /cgi-bin/datos.pl HTTP/1.1” y actualizando la versión del protocolo a HTTP/1.1. Esto permite usar características adicionales como las conexiones persistentes (keep-alive). Ademas al igual que las anteriores comparaciones el proxy agregar varios campos propios:

- Via: 1-0 proxy.tpfinal-tyr.com (squid/5.7): para indicar al servidor de datos que la peticion paso por proxy e identificarlo
- X-Forwarded-For: 10.10.0.10: informa al servidor web cual es la IP original del cliente que inicio la solicitud
- Cache-Control: max-age=0: en este caso el proxy solicita explícitamente que no se use contenido cacheado.
- Connection: keep-alive: mantiene la conexión abierta para enviar más de un petición sobre la misma conexión permitiendo una mejor latencia que en caso de tener que hacer un establecimiento y cierre por conexión.

Los campos User-Agent, Accept, Accept-Enconding, Accept-Language, Host y Refere permanecen identicos entre la peticion del usuario y la del proxy conservando las características originales del usuario.

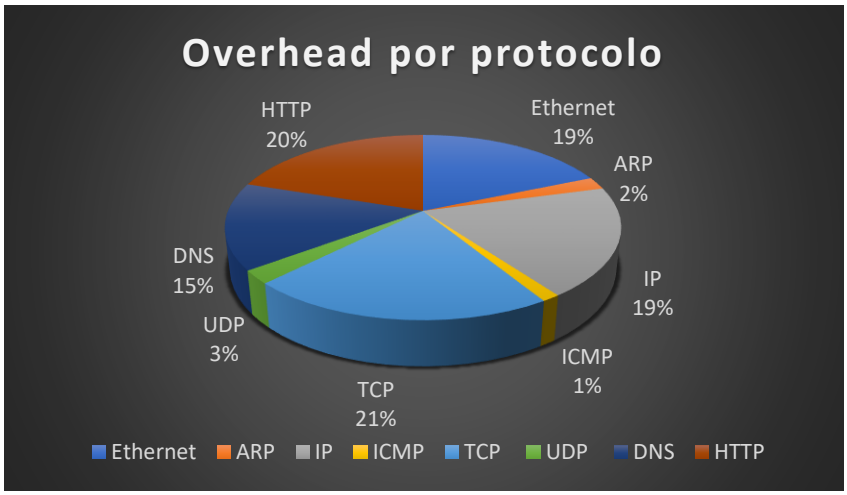
7. Confeccione una tabla con los diferentes protocolos involucrados, cantidad de PDUs, total en headers y total en datos. De allí, calcule el overhead ($Overhead = \frac{Total\ \ Datos\ \ Control}{Total\ \ Datos\ \ Tx}$) total y por protocolo generado para lograr la transferencia. Se sugiere armar una tabla en la cual cada fila Fi corresponde a una PDU y cada columna a un protocolo Pj (de diferentes capas). Luego, en cada celda i,j consigne el tamaño en bytes que corresponde a esa Fi y Pj. Genere una gráfica acorde para mostrar los resultados.

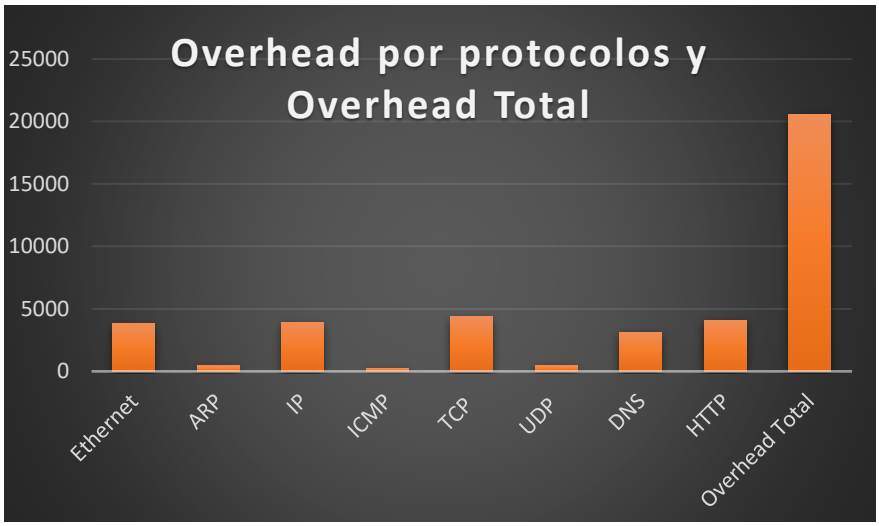
Realice en Excel una tabla donde cada fila corresponde a una PDU y cada columna a un protocolo de diferentes capas, en cada celda se encuentra la cantidad de bytes de header que corresponde a esa fila y columna, obteniéndose al final el overhead por protocolo y el

opverhead total. Del análisis realizado se observa que el overhead total representa el 74,84% del tráfico transmitido, lo cual indica que menos del 25% del total de bytes corresponde a datos útiles de aplicación, mientras que el resto se destina a cabeceras de protocolos necesarios para el control, direccionamiento y funcionamiento correcto de la comunicación. Los principales contribuyentes al overhead son los protocolos de las capas de transporte y aplicación:

- TCP (15,94%), utilizado para garantizar la entrega confiable de los datos,
- HTTP (14,92%), que incluye campos extensos en las solicitudes y respuestas,
- IP (14,32%), responsable del direccionamiento lógico de los paquetes,
- Y Ethernet (13,93%), como protocolo de enlace de datos, presente en todas las tramas.

Otros protocolos como DNS (11,39%) y ARP (1,63%) también aportan al overhead, aunque en menor proporción. Incluso ICMP (0,96%), si bien no es crítico en la transferencia del archivo, genera tráfico adicional que contribuye al total.





Se adjunta planilla de Excel con los datos recolectados, la hoja tiene el nombre “overhead”