Claudia Cea Tassel
Juan Riera Gómez
Pareja 5
Grupo 1301

MEMORIA PRÁCTICA 3: MONITORIZACIÓN

Introducción:

Nos hemos valido de las técnicas de shell scripting para el análisis de una traza de tráfico, simulando así la situación de análisis de tráfico de una red real. Nuestro script practica3_analisis.sh ejecuta todos los análisis del enunciado automáticamente y genera las gráficas y demás salidas que aparecen en este documento. Hemo incluido un archivo "leeme.txt" que es importante leer antes de ejecutar el script, ya que incluye más características sobre su funcionamiento y unas breves instrucciones de uso que pueden ser útiles en términos de eficiencia. Hemos procurado mantener un estilo modularizado y limpio en los scripts para facilitar su entendimiento. También hemos añadido comentarios y cabeceras a cada uno de ellos con el mismo fin.

Ejercicios:

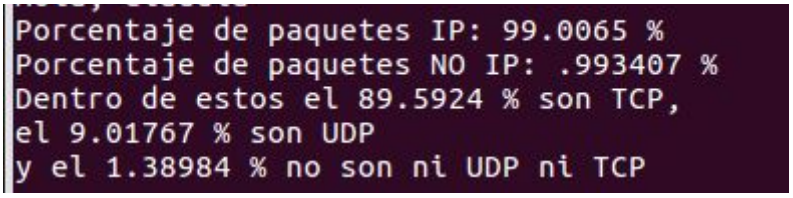
Ejercicio 1

En primer lugar se nos pide hallar el porcentaje de paquetes IP de la traza generada, y dentro de los paquetes IP el porcentaje de paquetes que sean TCP y UDP. Para ello usaremos la herramienta tshark, con la que filtraremos distintos paquetes para obtener sólo aquellos que nos interesan.

Comenzamos filtrando los paquetes que sean de tipo Ethernet. Para ello utilizamos el siguiente comando: *tshark -r traza.pcap -T fields -e eth.type > allfile*
Para sacar los paquetes de tipo IP utilizamos el comando: *tshark -r traza.pcap -T fields -e ip.proto -e ip.dst -e ip.src -e tcp.dstport -e tcp.srcport -e udp.dstport -e udp.srcport -e frame.len -Y 'eth.type == 0x0800 or (eth.type == 0x8100 and vlan.etype == 0x0800)' > ipfile*
pues como se especifica en el enunciado de la práctica entendemos como IP todo paquete cuyo tipo Ethernet es IP (0x0800), o bien el tipo Ethernet es VLAN (0x8100) y el tipo VLAN es IP (0x0800).
A continuación queremos obtener los paquetes IP que sean de tipo TCP y para ello utilizamos el siguiente comando de tshark: *tshark -r traza.pcap -T fields -e tcp.srcport -e frame.len -Y '(eth.type == 0x0800 or (eth.type == 0x8100 and vlan.etype == 0x0800)) and ip.proto == 0x06' > tcpsrcfile.tmp*
Finalmente para hallar los paquetes IP de tipo UDP utilizamos el siguiente comando: *tshark -r traza.pcap -T fields -e udp.srcport -e frame.len -Y '(eth.type == 0x0800 or (eth.type == 0x8100 and vlan.etype == 0x0800)) and ip.proto == 0x11' > udpsrcfile.tmp*

Una vez tenemos estos ficheros en ej1.sh contamos el tamaño total de la traza y el número de tramas Ethernet, contando el número de líneas de cada uno de los archivos que hemos generado previamente con tshark. Una vez tenemos estos datos simplemente tenemos que operar para hallar dichos porcentajes. El porcentaje de paquetes de tipo IP lo sacamos comparando el número de paquetes de tipo IP con el tamaño total de la traza que hemos obtenido gracias al fichero *allfile*. Por otro lado, el porcentaje de paquetes TCP y UDP lo sacamos comparando el número de paquetes de cada tipo con el número de paquetes que son de tipo IP y que hemos obtenido gracias al fichero *ipfile.*

Estos son los resultados obtenidos:



Los resultados tienen sentido, pues IP es el protocolo principal de la capa de red. Dentro de los paquetes IP hay mayor número de paquetes TCP. Estos sirven para crear conexiones entre redes de datos a través de la cuales pueden enviarse información, mientras que los paquetes de tipo UDP tienen un flujo unidireccional.

Ejercicio 2

En el ejercicio 2, se nos pide hallar el top 10 de direcciones activas y el top 10 de puertos. Ambos deben ordenarse según su tamaño en bytes y según el número de paquetes, distinguiendo el sentido.

Para ordenar los paquetes según el número de veces que aparecen en la traza usaremos el comando *uniq -c* que cuenta el número de veces que aparece la dirección o puerto en el fichero. A continuación, ordenamos estos datos con el comando *sort -n* y utilizando *head -n 10* obtenemos los 10 que han aparecido más veces.
Por otro lado, para ordenar los paquetes según su tamaño debemos ir sumando el tamaño del paquete cada vez que su dirección o puerto sea el mismo.

Para realizar este ejercicio se ejecuta el fichero ej2.sh, el cual recibe tres argumentos: el nombre del archivo de donde tiene que sacar el top 10, el nombre del campo calculado y si se trata de una dirección o de un puerto. En este ejercicio comenzamos a trabajar con *awk*, que nos sirve para imprimir por terminal los top 10 y en el caso de ordenación según tamaño para ir sumando el tamaño de los paquetes.

Estos son los resultados que hemos obtenido:

Top 10 direcciones IP origen por numero de paquetes

Direccion	Numero de paquetes
11.80.183.30	15454
86.54.153.150	11463
4.16.104.211	5805
86.72.129.177	4657
67.183.116.50	2906
8.19.191.52	2188
39.248.199.189	2161
106.241.122.57	2048
46.69.107.96	1883
31.188.139.240	1652

Top 10 direcciones IP origen por bytes transmitidos

Direccion	Bytes
11.80.183.30	23098523
86.72.129.177	6918040
67.183.116.50	4344112
8.19.191.52	3245100
39.248.199.189	3193577
106.241.122.57	3009353
46.69.107.96	2730262
31.188.139.240	2473818
4.16.104.211	1970587
86.54.153.150	1025537

Top 10 direcciones IP destino por numero de paquetes

Direccion	Numero de paquetes
86.54.153.150	34986
11.80.183.30	3881
98.107.105.10	3785
4.16.104.211	2857
86.72.129.177	1273
46.69.107.96	1046
67.183.116.50	983
39.248.199.189	666
70.139.214.253	664
31.188.139.240	619

Top 10 direcciones IP destino por bytes transmitidos

Direccion	Bytes
86.54.153.150	50345203
4.16.104.211	2853122
98.107.105.10	1816645
11.80.183.30	249160
70.139.214.253	115206
86.72.129.177	79229
110.134.240.216	76301
46.69.107.96	70017
67.183.116.50	59576
39.248.199.189	47886

Top 10 puerto TCP origen por numero de paquetes

Puerto	Numero de paquetes
80	36640
55934	1423
55860	1096
54615	1046
55865	617
43585	607
33896	603
55173	471
55848	418
33903	380

Top 10 puerto TCP origen por bytes transmitidos

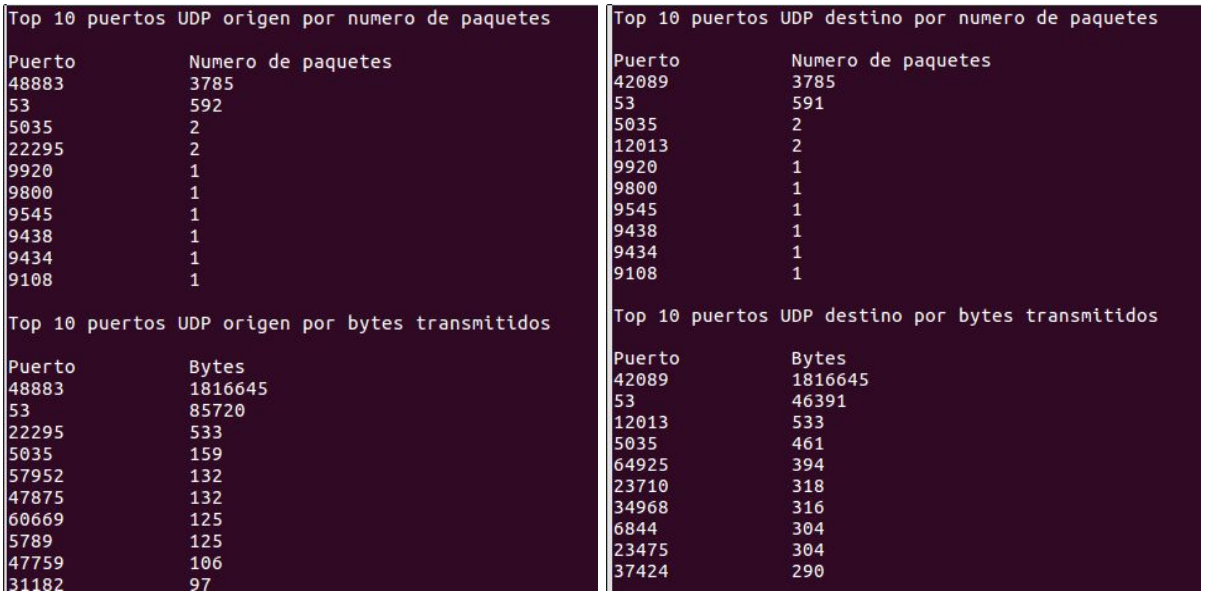
Puerto	Bytes
80	52857665
443	217800
55934	88065
54615	70017
55860	67367
55865	40574
43585	36512
33896	35533
55173	28338
46832	26382

Top 10 puertos TCP destino por numero de paquetes

Puerto	Numero de paquetes
80	12342
55934	5486
55860	4313
55865	3204
43585	2188
54615	1883
33896	1813
55173	1717
55848	1396
46371	1174

Top 10 puertos TCP destino por bytes transmitidos

Puerto	Bytes
55934	8236507
55860	6437994
55865	4808618
43585	3245100
54615	2730262
33896	2707440
55173	2566453
55848	2072650
46371	1756652
57063	1690967



```
Top 10 puertos UDP origen por numero de paquetes

Puerto          Numero de paquetes
48883           3785
53              592
5035            2
22295           2
9920            1
9800            1
9545            1
9438            1
9434            1
9108            1

Top 10 puertos UDP origen por bytes transmitidos

Puerto          Bytes
48883           1816645
53              85720
22295           533
5035            159
57952           132
47875           132
60669           125
5789            125
47759           106
31182           97
```

```
Top 10 puertos UDP destino por numero de paquetes

Puerto          Numero de paquetes
42089           3785
53              591
5035            2
12013           2
9920            1
9800            1
9545            1
9438            1
9434            1
9108            1

Top 10 puertos UDP destino por bytes transmitidos

Puerto          Bytes
42089           1816645
53              46391
12013           533
5035            461
64925           394
23710           318
34968           316
6844            304
23475           304
37424           290
```

Vemos que, con un cierto margen de error, los top 10 por paquetes y por bytes transmitidos se asemejan entre sí, indicando una cierta uniformidad en los tamaños de los paquetes. Además podemos apreciar que tanto TCP como UDP tienen un top 1 bastante diferenciado del resto, esto podría deberse a que esos son los puertos que utilizan esos protocolos.

Ejercicio 3

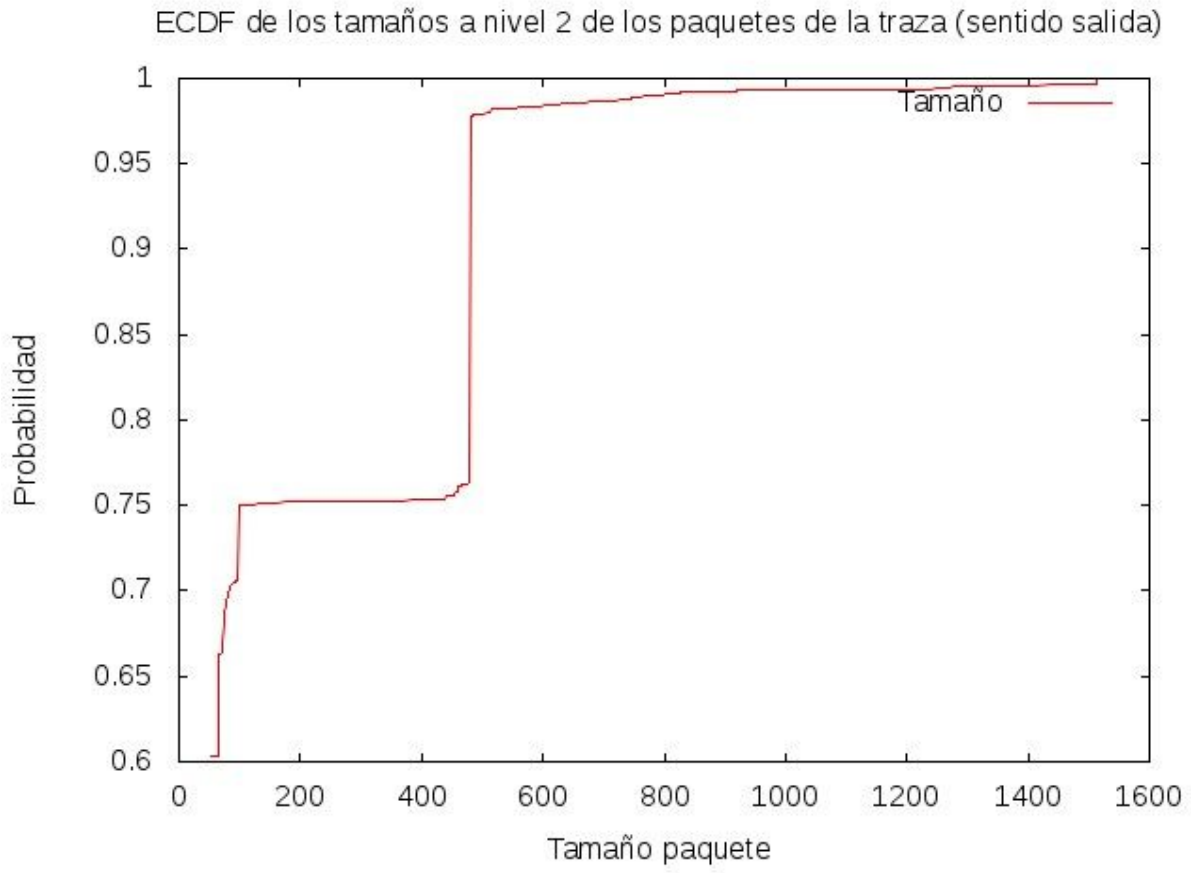
En este ejercicio se nos pide almacenar en un ECDF los tamaños de distintos tipos de paquetes y de volcar los resultados en gráficas. Para realizarlo se ejecuta ej3.sh que se encarga de filtrar los paquetes, y a su vez llama a los scripts ecdf.sh y grafica.sh.

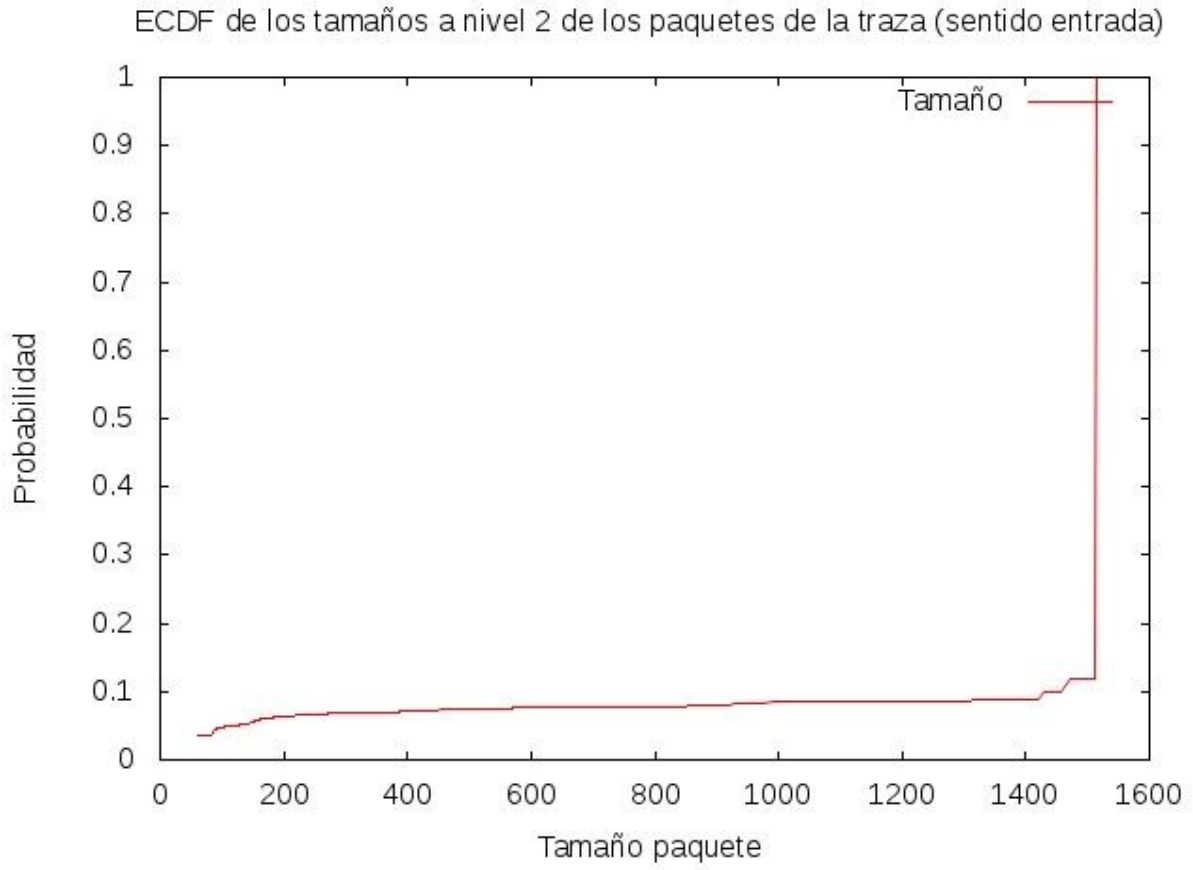
En ecdf.sh nuestro objetivo es generar un fichero que contenga la función de distribución de la lista de paquetes que le metemos como primer y único argumento. Para ello ordenamos el archivo con *sort -n*, agrupamos los paquetes con el mismo tamaño con *uniq -c* y finalmente con *awk* hallamos la probabilidad de que el tamaño de un paquete sea menor o igual que el tamaño del paquete en el que estamos situados.

Una vez tenemos el ECDF en un archivo hacemos una gráfica con esos valores mediante grafica.sh. Este script recibe como argumentos el título de la gráfica, el nombre del eje X, el nombre del eje Y, el nombre del archivo donde se encuentran los datos de los cuales queremos hacer una gráfica, el nombre del archivo de salida donde queremos guardar la gráfica (en este caso generamos siempre un fichero de tipo jpeg) y lo que estamos midiendo en la gráfica, que en este caso serán tamaños.

Empezamos con los paquetes a nivel 2 de la traza. Para ello filtramos con la dirección MAC proporcionada por el generador (00:11:88:CC:33:21) con los siguientes comandos: *tshark -r traza.pcap -T fields -e frame.len -Y 'eth.src == 00:11:88:cc:33:21' > macsrcfile.tmp* para obtener los paquetes que tienen la dirección MAC como origen y *tshark -r traza.pcap -T fields -e frame.len -Y 'eth.dst == 00:11:88:cc:33:21' > macdstfile.tmp* para obtener los paquetes que tienen la dirección MAC como destino.

Con los datos obtenidos de *macsrcfile.tmp* y *macdstfile.tmp* obtenemos los siguientes ECDFs:

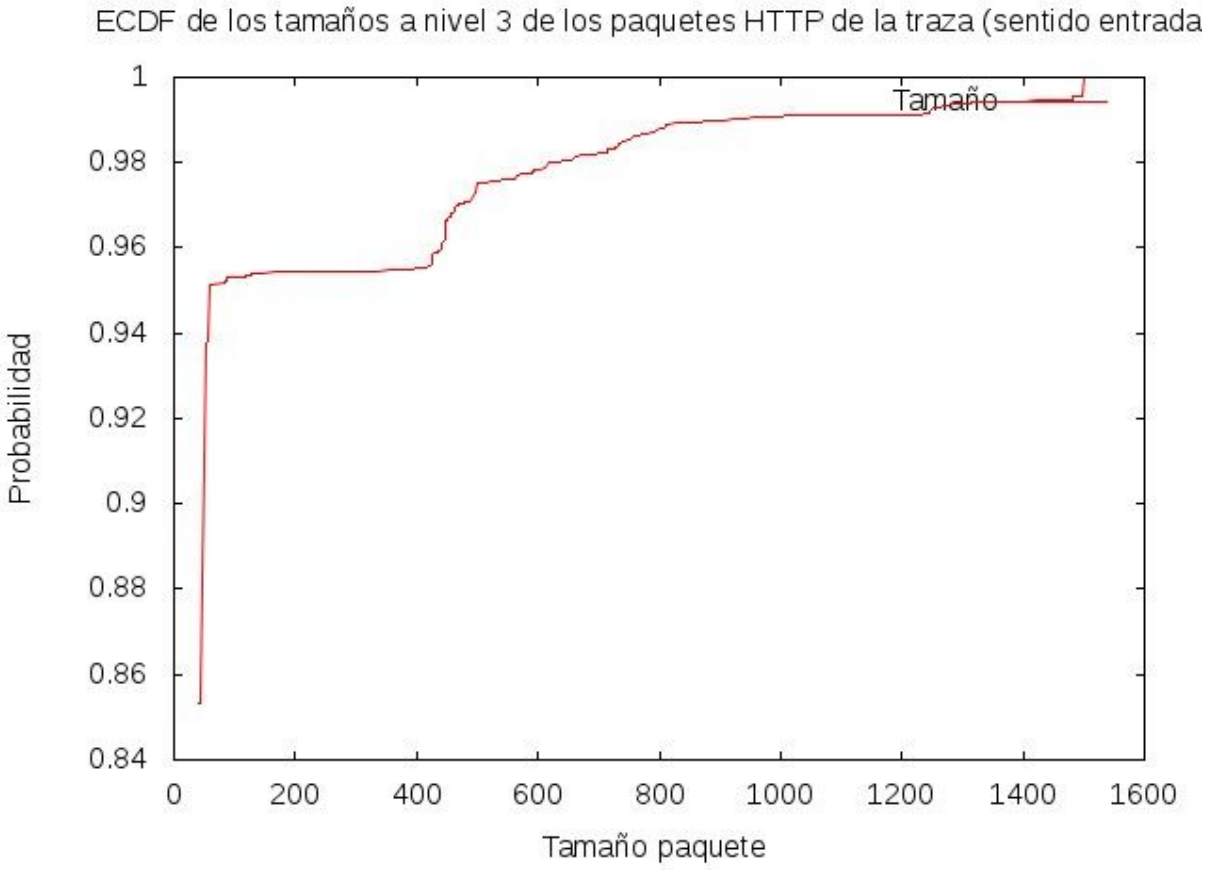


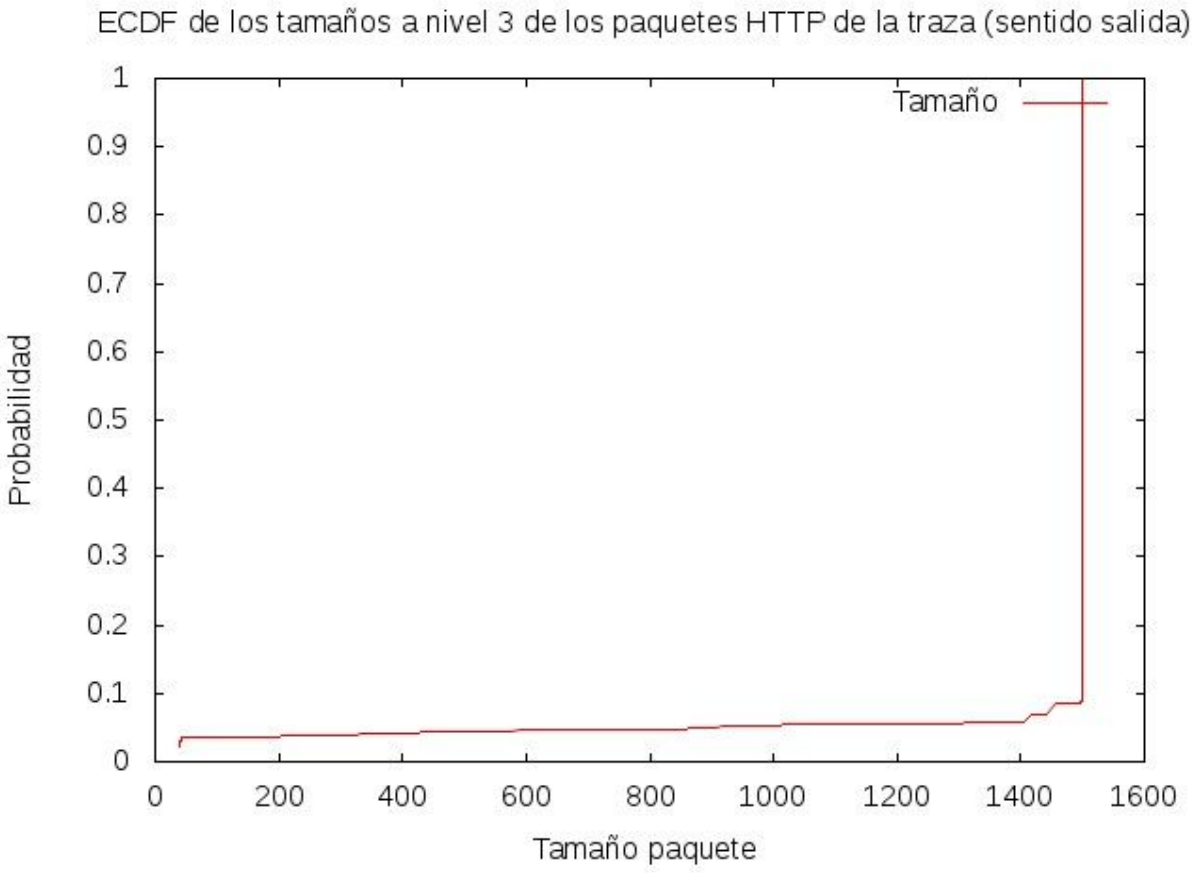


En la primera gráfica podemos observar como los paquetes que tienen la dirección MAC 00:11:88:CC:33:21 como origen suelen tener un tamaño de alrededor de 100 bytes o de alrededor de 500 bytes, pues son las zonas de la gráfica donde la probabilidad aumenta de manera repentina. En cambio en la segunda gráfica podemos observar como el tamaño de la mayoría de los paquetes está en torno a los 1500 bytes. Estas diferencias de tamaño se producen porque cuando utilizamos la dirección MAC como origen normalmente es porque estamos realizando diversas peticiones a un servidor, mientras que al utilizar la dirección MAC como destino el servidor nos envía las respuestas a dichas peticiones, es decir, nos proporciona el contenido solicitado. Es por ello que el tamaño de estos paquetes es mucho mayor.

A continuación queremos generar los ECDFs de los tamaños a nivel 3 de los paquetes HTTP. Consideramos que un paquete es HTTP cuando este usa el puerto 80 de TCP en origen o destino. Para obtener estos paquetes utilizamos los filtros: *tshark -r traza.pcap -T fields -e ip.len -Y 'tcp.srcport == 80' > httpsrcfile.tmp* para obtener los paquetes que usan el puerto 80 de TCP como origen y *tshark -r traza.pcap -T fields -e ip.len -Y 'tcp.dstport == 80' > httpdstfile.tmp* para obtener los paquetes que usan el puerto 80 de TCP como destino.

Con los datos obtenidos de *httpsrcfile.tmp* y *httpdstfile.tmp* obtenemos los siguientes ECDFs:

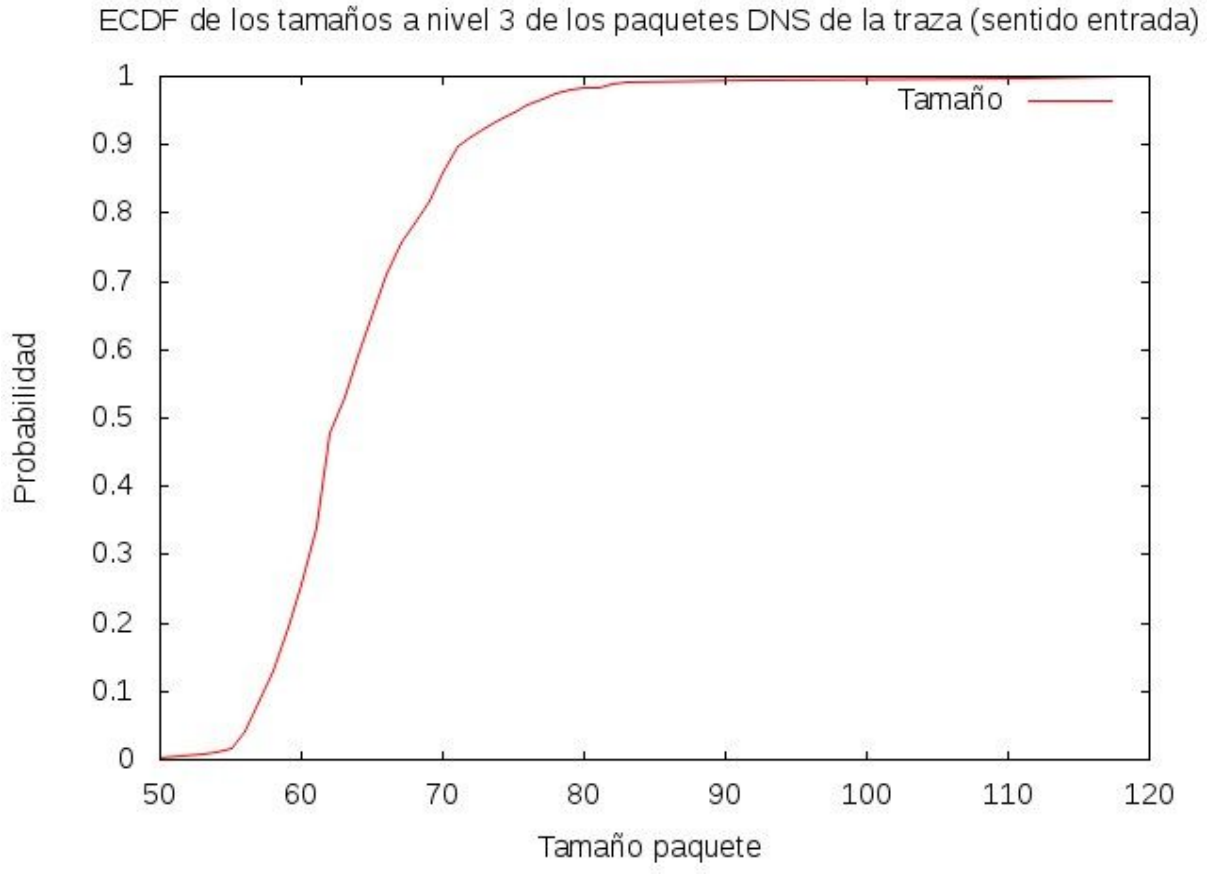

ECDF de los tamaños a nivel 3 de los paquetes HTTP de la traza (sentido entrada
Tamaño
Probabilidad
Tamaño paquete
1
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0.84
0
200
400
600
800
1000
1200
1400
1600

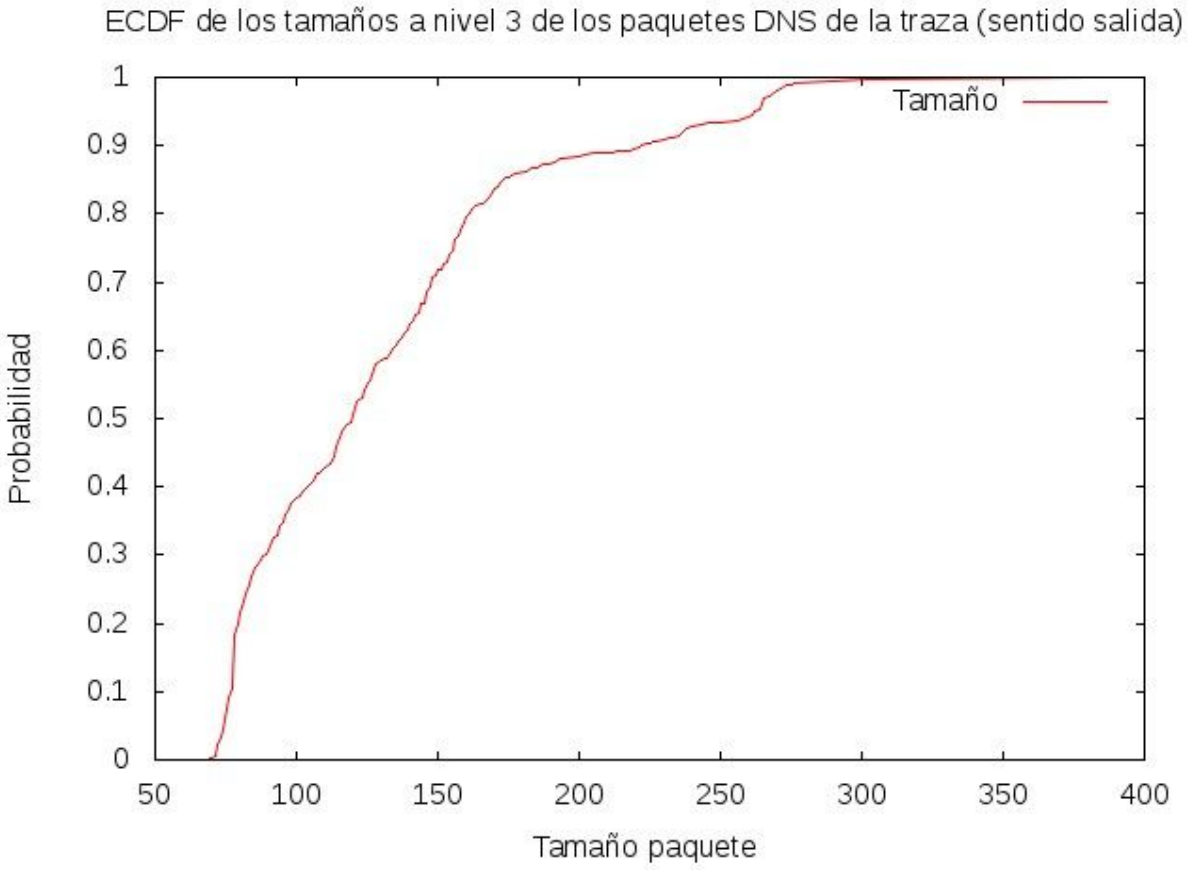

ECDF de los tamaños a nivel 3 de los paquetes HTTP de la traza (sentido salida)
Tamaño
Probabilidad
Tamaño paquete
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
200
400
600
800
1000
1200
1400
1600

En la primera gráfica podemos observar como los paquetes que tienen el puerto 80 de TCP como destino suelen tener en su mayoría un tamaño de alrededor de 100 bytes, pues es la zona de la gráfica donde la probabilidad aumenta de manera repentina. En cambio, en la segunda gráfica donde utilizamos el puerto 80 de TCP como origen podemos observar como el tamaño de la mayoría de los paquetes está en torno a los 1500 bytes. Estas diferencias de tamaño se producen porque cuando utilizamos el puerto 80 de TCP como destino estamos realizando una petición a un servidor. Estas peticiones siguen un formato en el que se suelen realizar muchas peticiones de muy poco tamaño, por eso el tamaño de los paquetes no es demasiado grande. Por otro lado, al utilizar el puerto 80 de TCP como origen el servidor está enviando la respuesta a esas peticiones, es decir, el recurso solicitado, que tiene un tamaño mucho mayor.

Finalmente queremos generar los ECDFs de los tamaños a nivel 3 de los paquetes DNS. Consideramos que un paquete es DNS cuando este usa el puerto 53 de UDP en origen o destino. Para obtener estos paquetes utilizamos los filtros: *tshark -r traza.pcap -T fields -e ip.len -Y 'udp.srcport == 53' > dnssrcfile.tmp* para obtener los paquetes que usan el puerto 53 de UDP como origen y *tshark -r traza.pcap -T fields -e ip.len -Y 'udp.dstport == 53' > dnsdstfile.tmp* para obtener los paquetes que usan el puerto 53 de UDP como destino.

Con los datos obtenidos de *dnssrcfile.tmp* y *dnsdstfile.tmp* obtenemos los siguientes ECDFs:





En la primera gráfica podemos observar como los paquetes que tienen el puerto 53 de UDP como destino suelen tener en su mayoría un tamaño pequeño de entre 60 o 80 bytes, pues es la zona de la gráfica donde la probabilidad aumenta más. En cambio, en la segunda gráfica podemos observar como el tamaño de la mayoría de los paquetes es mayor y más variado con paquetes cuyo tamaño está principalmente entre los 75 y los 250 bytes. Estas diferencias de tamaño se producen porque cuando utilizamos el puerto 53 de UDP como destino estamos enviando un dominio a un servidor, mientras que al utilizar el puerto 53 de UDP como origen el servidor está enviando ese mismo dominio junto con su dirección IP asociada. Por este motivo, es claro que el tamaño de los paquetes DNS en sentido salida sea más grande que aquellos en sentido entrada.

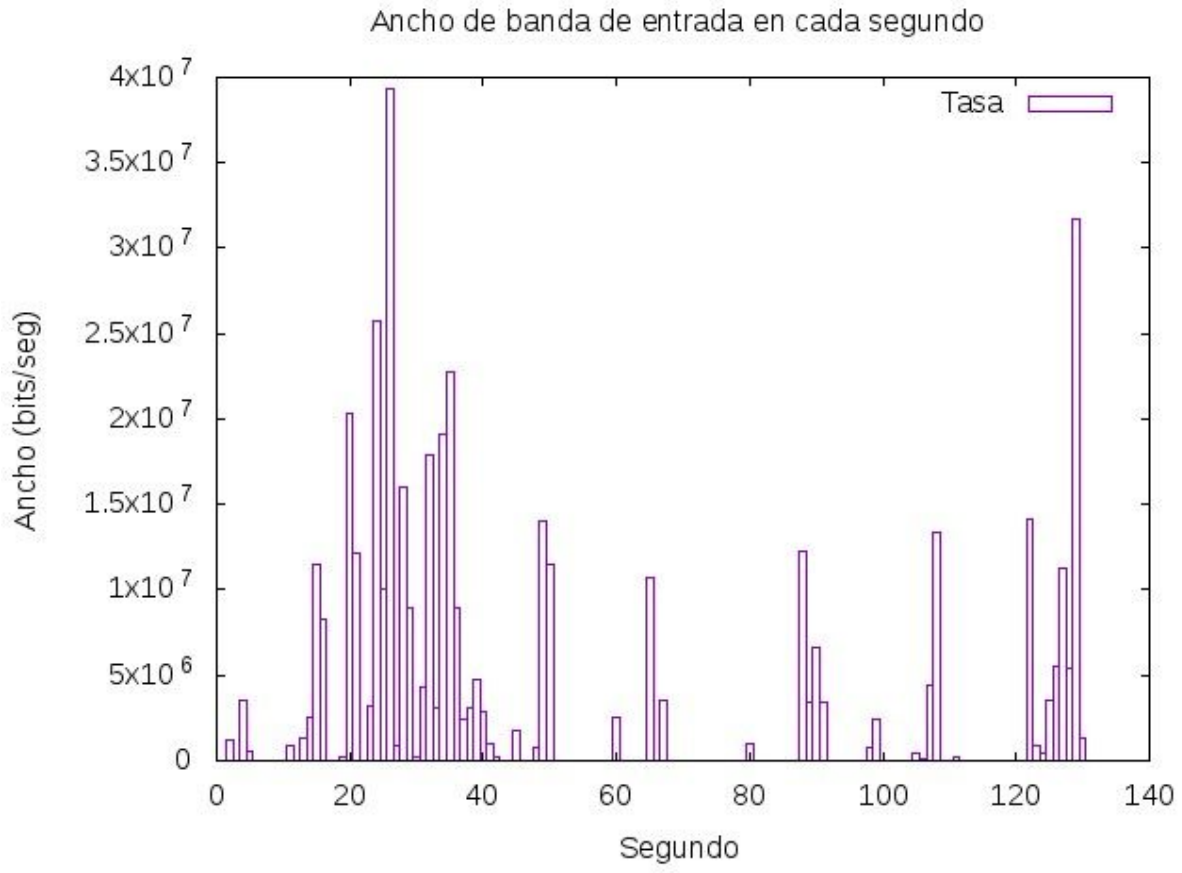
Ejercicio 4

En este ejercicio se nos pedía generar unas figuras que mostraran el ancho de banda a nivel 2 en bits por segundo (b/s) y por sentido. Los segundos sin tráfico deben representarse a cero.
Para generar el archivo con las tasas de entrada hemos usado el siguiente comando de tshark

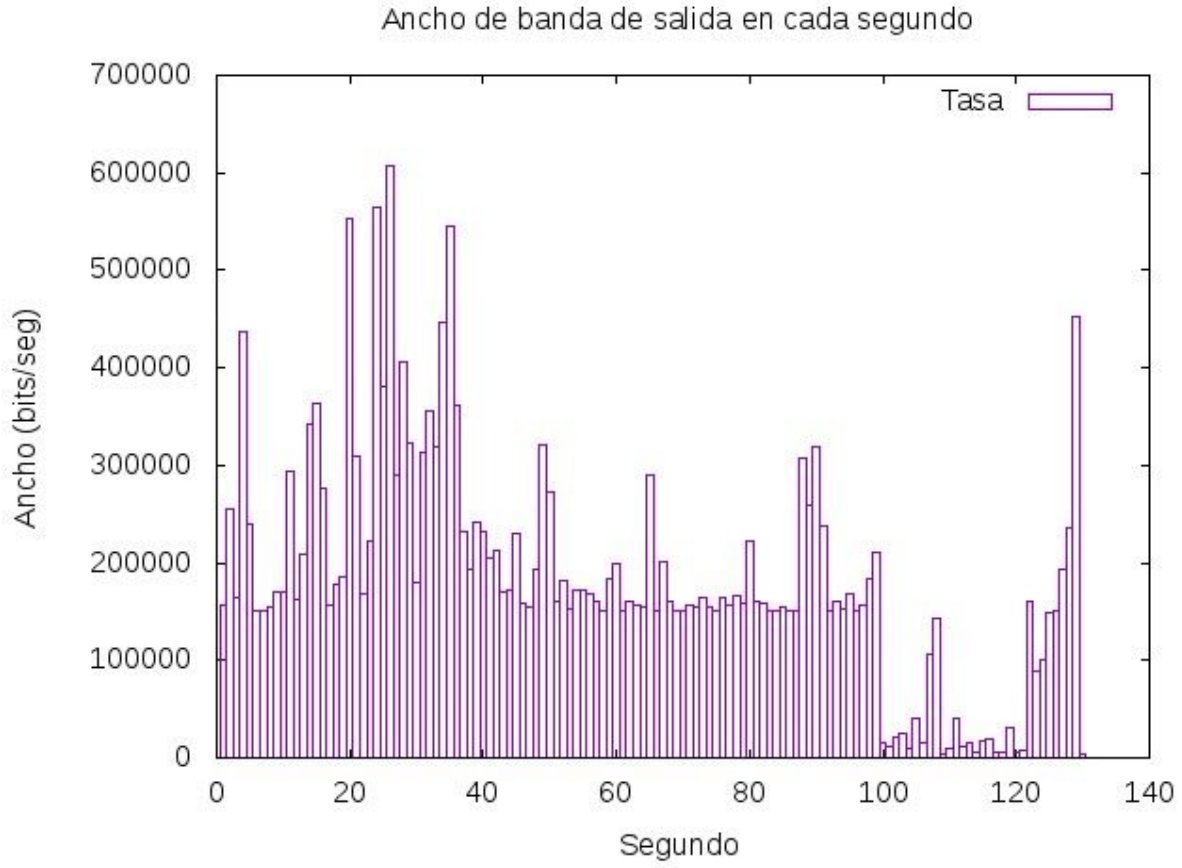
tshark -r $1 -qz io,stat,1,"SUM(frame.len)frame.len&&eth.dst==00:11:88:CC:33:21" > tasasin.tmp

Que genera un archivo de estadísticas sobre el tamaño de los paquetes en cada segundo, es decir, la cantidad de bytes transmitidos en cada segundo, es decir, el ancho de banda. Para el otro sentido el comando era el mismo, sustituyendo eth.dst por eth.src.

El resultado lo hemos convertido a ECDF mediante nuestro script ecdf.sh y lo hemos graficado con graficaBars.sh. Hemos usado este en lugar de grafica.sh porque se nos pedia que los segundos sin caudal aparecieran en cero, cosa que con las líneas no quedaba tan bien representada.



En la gráfica podemos apreciar picos considerables alrededor de los segundos 25, 33 y 129. Por otro lado hay grandes valles en los intervalos (5-15), (30-60), (66-88),(91,107), (110, 120). Estos picos indican un mayor ancho de banda, es decir una mayor actividad del enlace en sentido de entrada, y los valles una menor actividad del mismo.



En el ancho de banda de salida obtenemos observaciones que van en paralelo a las de entrada, los picos se hallan aproximadamente en los mismos momentos, sin embargo, los valles no bajan de los 100000 bits hasta los 100 segundos. Esto indica un mayor nivel de carga de datos constante en ese periodo de tiempo, al que además se suma el de los picos ya mencionados.

Ejercicio 5

En este ejercicio se nos pedía generar una ECDF de los tiempos entre llegadas del flujo TCP, UDP indicado por el generador de la traza, y una gráfica a partir de esta para cada sentido.

Para la realización de este ejercicio hemos utilizado los siguientes comandos tshark para generar los análisis de la traza:

tshark -r traza.pcap -T fields -e frame.time_delta_displayed -Y 'tcp&&ip.dst==46.69.107.96' > timeipin.tmp

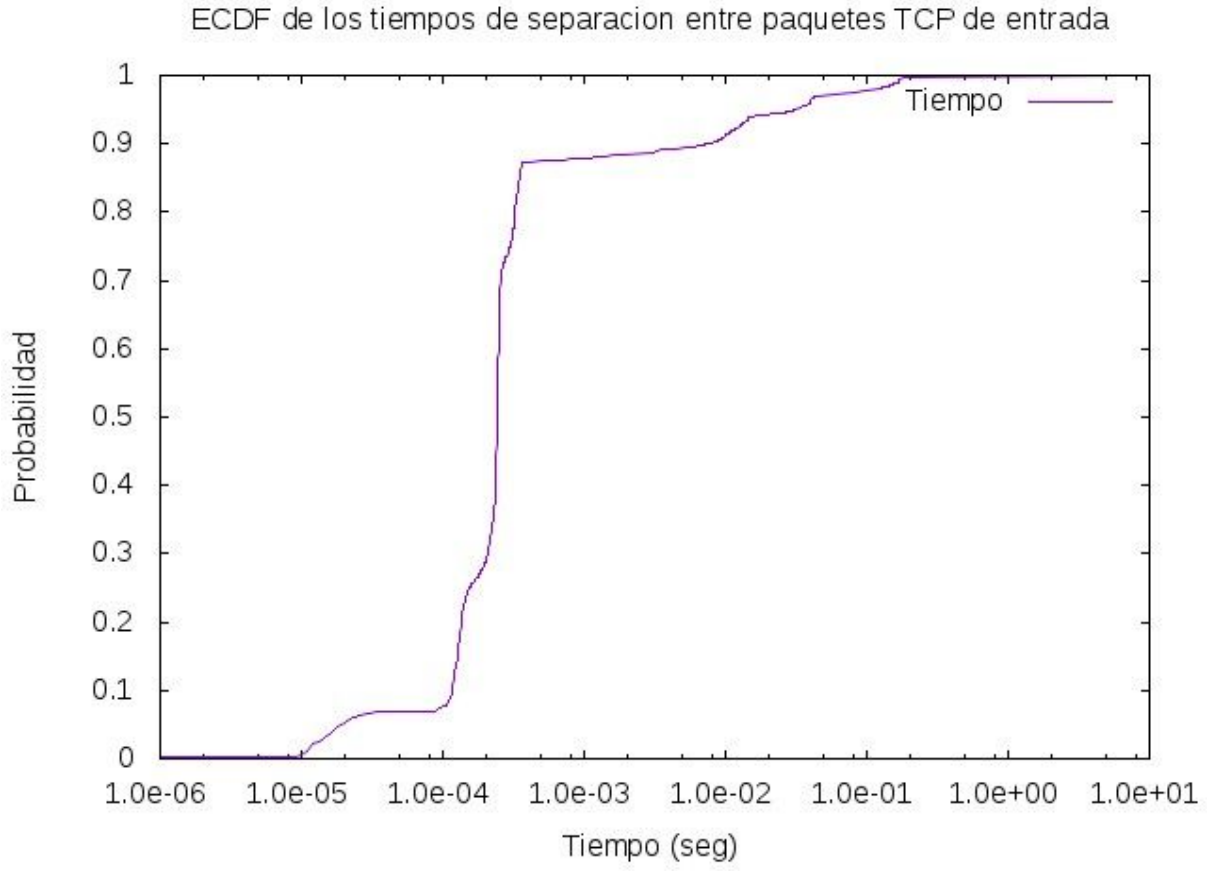
tshark -r traza.pcap -T fields -e frame.time_delta_displayed -Y 'tcp&&ip.src==46.69.107.96' > timeipout.tmp

tshark -r traza.pcap -T fields -e frame.time_delta_displayed -Y 'udp.dstport==42089' > timeudpin.tmp

tshark -r traza.pcap -T fields -e frame.time_delta_displayed -Y 'udp.srcport==42089' > timeudpout.tmp

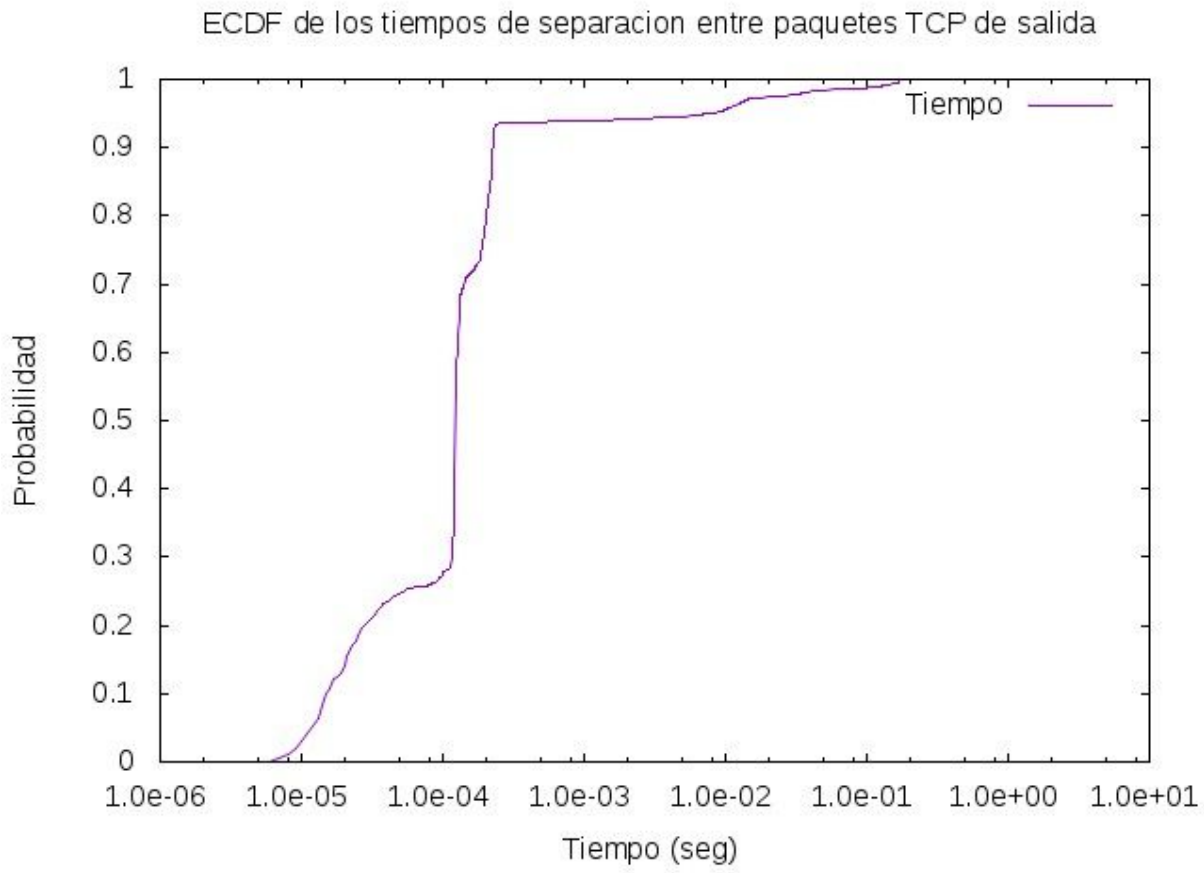
Donde obtenemos en todos el campo frame.time_delta, es decir, el tiempo de separación entre paquetes, y los filtros tcp y udp para obtener los paquetes tcp y udp por separado, y por último udp.srcport udp.dstport para obtener los paquetes asociados al puerto del flujo por cada sentido udp, y los filtros ip.src y ip.dst para obtener los paquetes asociados al flujo por cada sentido tcp.

Una vez obtenidos los datos los almacenamos en archivos temporales y generamos las ecdf con nuestro script ecdf.sh. Finalmente utilizamos grafica.sh para obtener la grafica asociada a la ecdf de los datos anteriores.

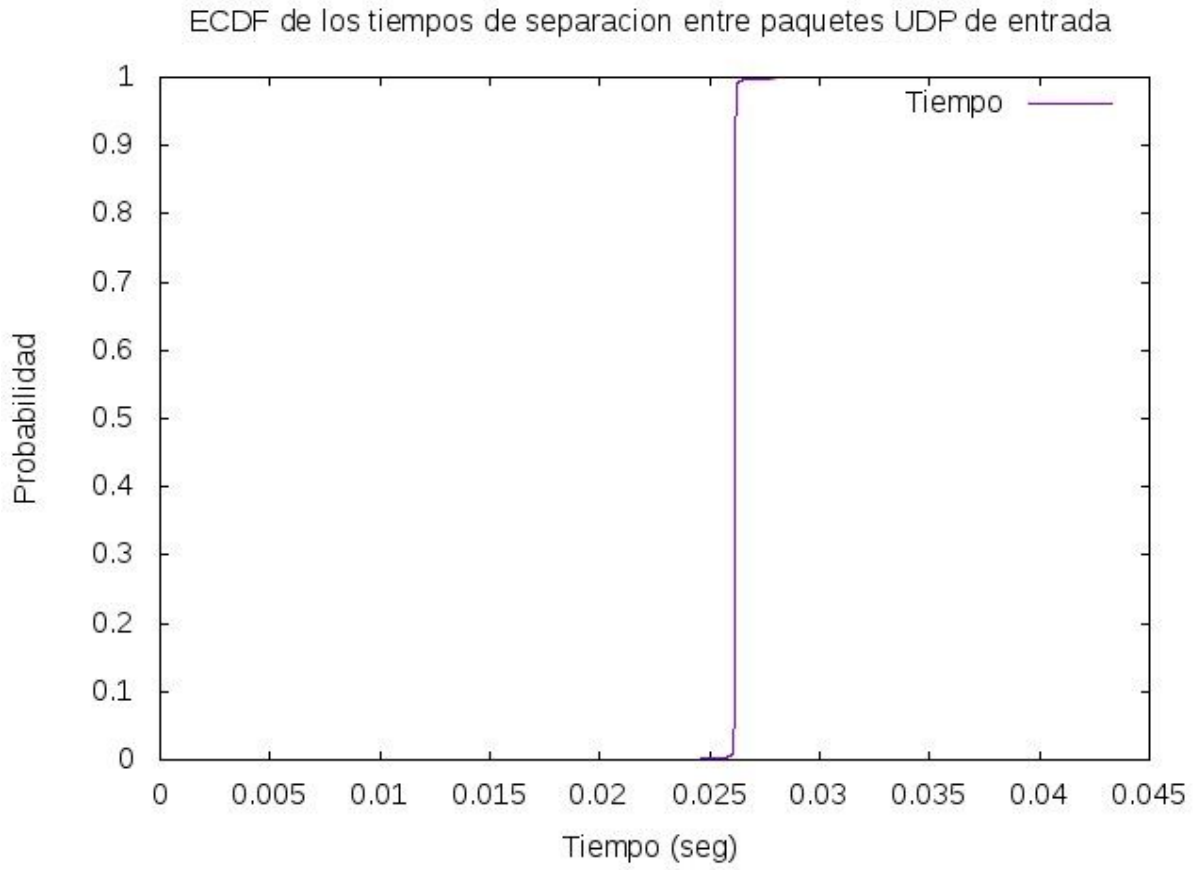


En esta gráfica podemos apreciar como una grandísima mayoría de paquetes TCP de entrada tienen como tiempo de separación entre sí un valor inferior a 0.0001 segundos. Sin embargo, hay unos pocos distribuidos de forma más o menos uniforme desde el 0.01 hasta valores más altos. Esto es bastante esperable, ya que en general los tiempos de separación entre paquetes son bajos si el flujo de los mismos es constante, pero hay pausas (no muy

abundantes) en la transmisión porque no hay nada que transmitir en ese momento, y por esa razón hay algunas pausas.



De esta gráfica sacamos conclusiones muy similares a las que obtuvimos de la gráfica anterior. Casi todas las separaciones entre paquetes son inferiores a 0.001s, pero hay algunas pausas que generan unas pocas separaciones mucho más largas.



Vemos que la aplicación que genera los paquetes UDP debía de tener un período estándar de envíos de paquetes alrededor de los 0.027 segundos aproximadamente. De ahí que haya un esa subida tan abrupta en ese punto y se mantenga estable en la mayoría de los demás puntos.

Al ejecutar el script para generar la gráfica que falta, la de ECDF de los tiempos de separación entre paquetes UDP de salida, obtendremos el siguiente mensaje por terminal:

"El archivo para generar la gráfica ECDF de los tiempos de separación entre paquetes UDP de salida está vacío o no existe, no se puede generar gráfica"

Esto es esperable, ya que UDP es un protocolo que no asegura la llegada de los paquetes, y que se utiliza para enviar datos de servicios que se pueden permitir algunas pérdidas de paquetes, o paquetes desordenados, ya que no tiene control de flujo. Por tanto cuando el ordenador recibe un paquete UDP, no responde para confirmar su llegada, de ahí que no haya paquetes UDP de salida.

Conclusión:

Las herramientas de shell scripting han facilitado mucho la práctica, y hemos aprendido a utilizarlas, en conjunción con otras herramientas como awk y tshark, para analizar el tráfico de una traza, simulando un análisis real del tráfico de una red.

Por otro lado, de este análisis hemos obtenido una mejor comprensión del funcionamiento y uso común de los distintos protocolos, sobre todo los DNS, HTTP, UDP, TCP e IP que son, a su vez, algunos de los más utilizados en las redes reales.

Este proceso que hemos llevado a cabo es el conocido como monitorización, es decir, el análisis de distintos aspectos del tráfico de una red para hallar patrones que nos permitan mejorarla, y problemas que solucionar, así como detectar posibles ataques a la red y otras irregularidades. En todos estos casos el fin último es el mantenimiento de la red, que constituye una parte importante del trabajo de un ingeniero de comunicaciones.