Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación
ICS2333 – Sistemas Operativos y Redes
Segundo Semestre de 2017

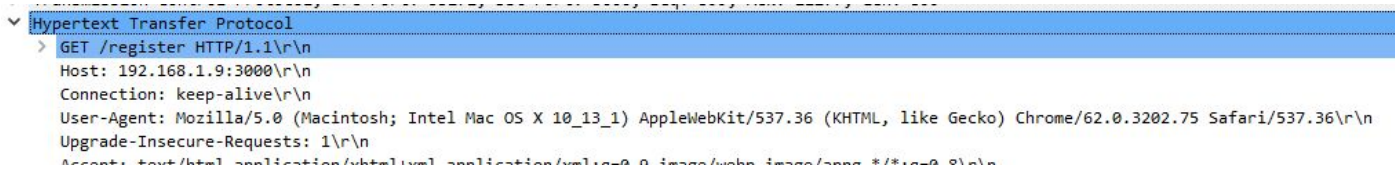
Informe Tarea 5

Manuel Silva (13637207)
Ivan Wolf (13634194)

Santiago de Chile,
13 de noviembre de 2017

Parte A

1. El browser que hace la solicitud es específicamente google chrome, esto se puede obtener desde las vistas con filtro http, con source la ip que teníamos asignada para la actividad, que corresponde a 192.168.1.136, en la captura revisamos el dato correspondiente a User-Agent como se muestra a continuación

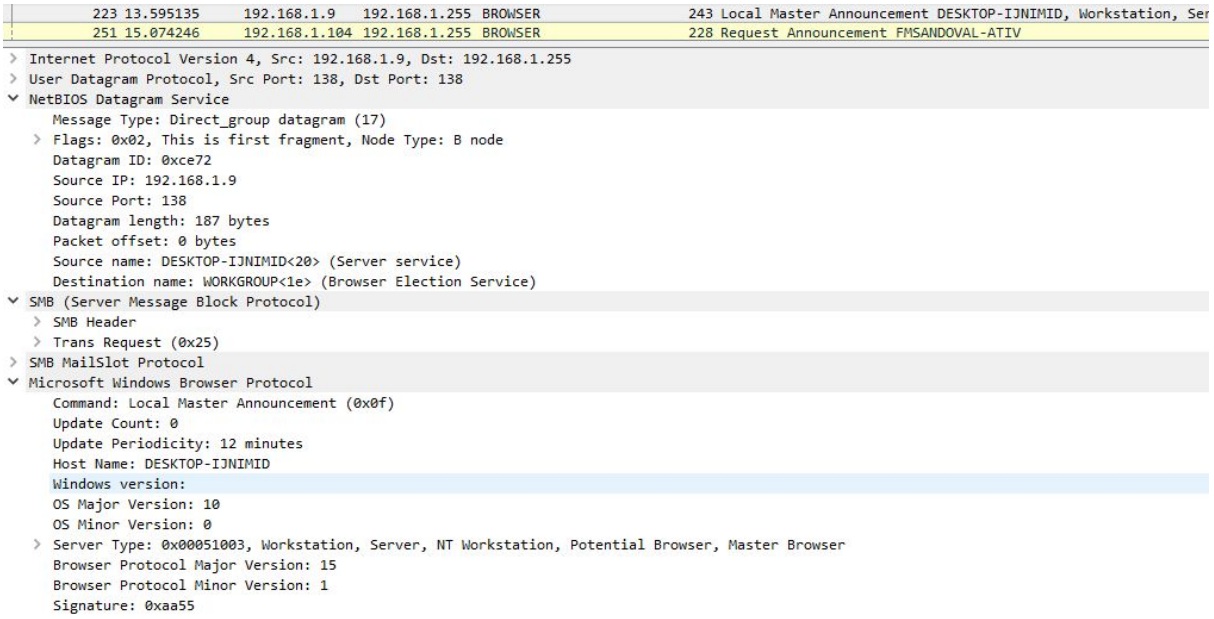


Si miramos la nomenclatura existente para el protocolo http, google chrome posee esta estructura. *"The Chrome (or Chromium/blink-based engines) user agent string is similar to the Firefox format. For compatibility, it adds strings like "KHTML, like Gecko" and "Safari" "*. Lo cual calza con nuestra situación, luego en específico se trata de google chrome el browser que realiza requests.

Examples

```
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537
```

2. Para detectar el sistema operativo del servidor, se revisó mediante un filtro display de *browser* con el cual se busca algún mensaje que tenga de source name la ip 192.168.1.9, que corresponde a la ip del servidor en la actividad.



Se ve cómo se obtiene de este protocolo características del host, que realiza broadcast hacia la ip 192.168.1.255, entre estas caracteristicas encontramos, el nombre del origen "*Desktop-IJNIMID*", que vemos posee un sistema operativo de Windows, con versión version OS Major Version: 10.

Ahora para el web server utilizado, se tiene uno del tipo 0x00051003, Workstation, Server, NT Workstation, versión 15, que corresponde a los servidores que ocupan principalmente Windows y mediante el cual se realizó de host para la actividad.

Este servidor es Hewlett Packard, con un número MAC de dc:4a:3e:e9:a5.

3. Los distintos tipos de datos que existen para cada caso se pueden ver a continuación en la tabla adjunta.

URL	**Tipo de Dato en Response**
http://192.168.1.9:3000/register	text/html
Secuencia de GET post registro para cargar assets y aplicación	text/css, application/javascript
http://192.168.1.9:3000/	text/html
http://192.168.1.9:3000/ (x2)	Code 304 Not Modified, no se entregan datos (X-Content-Type-Options: nosniff)
http://192.168.1.9/big.txt	text/plain
http://192.168.1.9/meme (GET /meme)	text/html
GET /meme_os.jpg	JPEG JFIF image
http://192.168.1.9/power	text/html

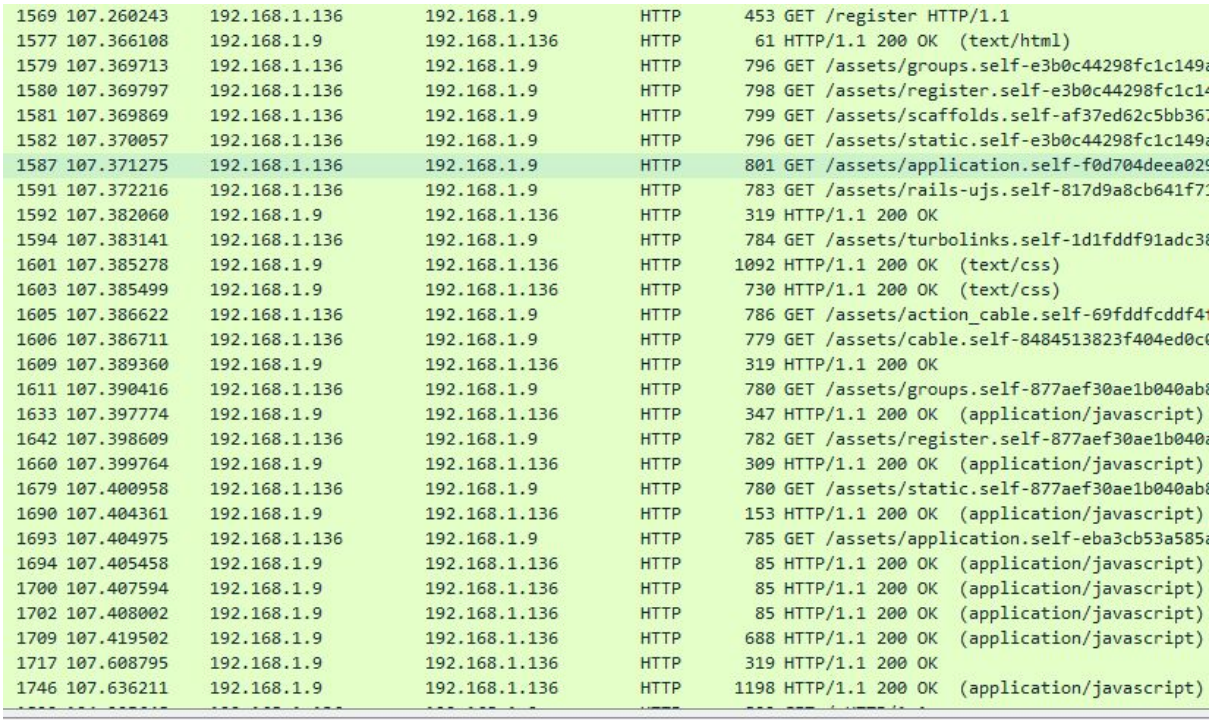
Se puede apreciar cómo dependiendo de las distintas situaciones, el formato de los datos va cambiando, por ejemplo, cuando se hace el registro por primera vez a la aplicación, se procede como respuesta un html con texto que indica cómo proceder con la recopilación de datos. Luego se hacen una serie de GET para inicializar la aplicación, donde se acceden principalmente a los assets/ de la aplicación para recibir todo lo necesario para visualizarla, recibiendo así una seguidilla de responses con formato text/css y application/javascript que es lo que ocupa principalmente RoR y lo que se está llamando con los **href** que se entregan en la respuesta a la url /register.

A continuación se ve como la segunda vez que se ingresa al root de la página, no se entrega ningún tipo de dato, solo un codigo (304) que indica que no hubieron cambios. Cuando se solicita visualizar el texto (big.txt), la respuesta consiste en un text gigante que viene en formato text/plain, que es un formato especial para tamaños grandes de strings para su lectura.

Finalmente, para la visualización de /meme, se entrega en primera instancia el html embed con el que se visualizara la ruta junto con la imagen en el formato que irá esta, luego a continuación se entrega, en formato JPEG, la imagen respectiva que contiene la información necesaria para visualizar la imagen, información que se tiene que enviar en un response distinto al del html concerniente a la página.

4. Para la siguiente sección, se dejará un cuadro que represente los códigos HTTP de respuesta obtenidos de la experiencia, y a continuación se presentan los códigos esperados, junto a una explicación cuando sea el caso de diferencias.

Request-Responses caso http://192.168.1.9:3000/register



```
1569 107.260243   192.168.1.136   192.168.1.9     HTTP    453 GET /register HTTP/1.1
1577 107.366108   192.168.1.9     192.168.1.136   HTTP     61 HTTP/1.1 200 OK  (text/html)
1579 107.369713   192.168.1.136   192.168.1.9     HTTP    796 GET /assets/groups.self-e3b0c44298fc1c149
1580 107.369797   192.168.1.136   192.168.1.9     HTTP    798 GET /assets/register.self-e3b0c44298fc1c1
1581 107.369869   192.168.1.136   192.168.1.9     HTTP    799 GET /assets/scaffolds.self-af37ed62c5bb36
1582 107.370057   192.168.1.136   192.168.1.9     HTTP    796 GET /assets/static.self-e3b0c44298fc1c149
1587 107.371275   192.168.1.136   192.168.1.9     HTTP    801 GET /assets/application.self-f0d704deea02
1591 107.372216   192.168.1.136   192.168.1.9     HTTP    783 GET /assets/rails-ujs.self-817d9a8cb641f7
1592 107.382060   192.168.1.9     192.168.1.136   HTTP    319 HTTP/1.1 200 OK
1594 107.383141   192.168.1.136   192.168.1.9     HTTP    784 GET /assets/turbolinks.self-1d1fddf91adc3
1601 107.385278   192.168.1.9     192.168.1.136   HTTP   1092 HTTP/1.1 200 OK  (text/css)
1603 107.385499   192.168.1.9     192.168.1.136   HTTP    730 HTTP/1.1 200 OK  (text/css)
1605 107.386622   192.168.1.136   192.168.1.9     HTTP    786 GET /assets/action_cable.self-69fddfcddf4
1606 107.386711   192.168.1.136   192.168.1.9     HTTP    779 GET /assets/cable.self-8484513823f404ed0c
1609 107.389360   192.168.1.9     192.168.1.136   HTTP    319 HTTP/1.1 200 OK
1611 107.390416   192.168.1.136   192.168.1.9     HTTP    780 GET /assets/groups.self-877aef30ae1b040ab
1633 107.397774   192.168.1.9     192.168.1.136   HTTP    347 HTTP/1.1 200 OK  (application/javascript)
1642 107.398609   192.168.1.136   192.168.1.9     HTTP    782 GET /assets/register.self-877aef30ae1b040
1660 107.399764   192.168.1.9     192.168.1.136   HTTP    309 HTTP/1.1 200 OK  (application/javascript)
1679 107.400958   192.168.1.136   192.168.1.9     HTTP    780 GET /assets/static.self-877aef30ae1b040ab
1690 107.404361   192.168.1.9     192.168.1.136   HTTP    153 HTTP/1.1 200 OK  (application/javascript)
1693 107.404975   192.168.1.136   192.168.1.9     HTTP    785 GET /assets/application.self-eba3cb53a585
1694 107.405458   192.168.1.9     192.168.1.136   HTTP     85 HTTP/1.1 200 OK  (application/javascript)
1700 107.407594   192.168.1.9     192.168.1.136   HTTP     85 HTTP/1.1 200 OK  (application/javascript)
1702 107.408002   192.168.1.9     192.168.1.136   HTTP     85 HTTP/1.1 200 OK  (application/javascript)
1709 107.419502   192.168.1.9     192.168.1.136   HTTP    688 HTTP/1.1 200 OK  (application/javascript)
1717 107.608795   192.168.1.9     192.168.1.136   HTTP    319 HTTP/1.1 200 OK
1746 107.636211   192.168.1.9     192.168.1.136   HTTP   1198 HTTP/1.1 200 OK  (application/javascript)
```

Se puede apreciar como el request de /register resulta con un código 200 OK, seguido de una serie de request para poder visualizar la aplicación web, donde se solicita acceso a los /assets de la aplicación mediante los href de la respuesta del primer GET, resultando cada uno de estos con un código 200 OK de igual forma.

A continuación se presentan los códigos concernientes al resto de los casos.

URL	**Codigo Response Obtenido**	**Codigo Response Esperado**
http://192.168.1.9:3000/ (x2)	Code 304 Not Modified	-
http://192.168.1.9/big.txt	Code 500 Internal Server Error	200 OK

http://192.168.1.9/meme (GET /meme)	Code 500 internal Server Error 200 OK (text/html)	200 OK
GET /meme_os.jpg	200 OK (JPEG JFIF image)	-
http://192.168.1.9/power	Code 500 Internal Server Error 200 OK (text/html)	200 OK

Vemos como la segunda vez que accedemos al root de la aplicación, al no existir cambios en esta, se retorna un código 304, el cual hace referencia a que no hubieron cambios en la aplicación, por lo tanto se mantiene lo almacenado en la caché de la página.

Para el caso de /power y /meme, vemos como se tiene un acceso errado en primera instancia con un error interno del servidor (500), lo cual no permite la visualización de la aplicación correctamente, pero luego de reintentar el acceso, se consigue el resultado esperable de 200 OK. Esto se puede deber en primera instancia al error existente en la actividad con /power donde no se mostraba lo necesario y botó la url por unos minutos, tiempo en el cual nosotros hicimos el acceso, para el caso de /meme, quizás no esperamos el tiempo suficiente para cargar la imagen o hubo un error de otra índole.

Finalmente para big.txt, se dio el caso que se visualizó el texto en la página y se logró lo que se pretendía al acceder a la pagina, pero luego al revisar el tráfico de datos nos dimos cuenta de un error interno del servidor, cosa que no visualizamos en el momento, esto debido a que quizás se levantó un error en la aplicación una vez cargado el texto en pantalla.

5. En la presente sección, se tabularon los resultados obtenidos para la cantidad de Bytes retornados en cada caso. Se hizo el supuesto de que al primer acceso *http://192.168.1.9:3000/register*, como se tiene que cargar todo el contenido de los href correspondientes a la respuesta del primer GET, se dividirá este acceso en 2, lo concerniente al acceso propiamente tal y la suma de los retornos relacionados con la inicialización de los html mediante los href correspondientes.

También considerando el hecho de que /big.txt, retorno un error 500, no se tiene información de la devolución en bytes, por lo tanto se ocupo informacion externa para poder determinar cuánto era el ideal que debiera retornar este archivo.

URL	**File Data Response (bytes)**	**Packet Length (bytes)**

http://192.168.1.9:3000/register	2480	61
Secuencia de GET post registro para cargar assets y aplicación	0 + 2498 + 676 + 0 + 293 + 35295 + 20539 + 31 + 31+ 31 + 634 + 0 + 25964 = **85992**	319 + 1092 + 730 + 319 +347+309 + 153 + 85+ 85+ 85 + 688 + 319 + 1198 = **5729**
http://192.168.1.9:3000/	-	680
http://192.168.1.9:3000/ (x2)	-	680
http://192.168.1.9/big.txt	6488666	656
http://192.168.1.9/meme (GET /meme)	2413	61
GET /meme_os.jpg	78999	1349
http://192.168.1.9/power	5236	61

Los datos anteriores, suponen por un lado, el tamaño total en bytes del cuerpo del mensaje que se estaba retornando, por ejemplo el contenido en bytes del texto grande o el tamaño en bytes de los pixeles del meme.

Pero también se encuentran los tamaños de los paquetes que se entregan al destinatario, que son los que se reciben, que son en su mayoría más reducidos, ya que estos contienen información del paquete que se está entregando, conteniendo los headers del protocolo http, ips de destino y origen, información del tamaño de la información en conjunto con otra información, pero que en empaquetación significa en tamaños más reducidos que la información del cuerpo del mensaje.

6. Para la siguiente sección se mostraran la cantidad de métodos GET hechos para cada caso en la tabla adjunta.

URL	Número de GET
http://192.168.1.9:3000/register	14 (1+ 13)
http://192.168.1.9:3000	1
http://192.168.1.9:3000 (x2)	1

http://192.168.1.9:3000/big.txt	1
http://192.168.1.9:3000/meme	2
http://192.168.1.9:3000/power	1

De lo anterior se pueden extraer varias cosas. Primero que todo, cuando se inicializa la página en la dirección del registro, suceden dos cosas, 1) se solicita el GET para la url de la página y 2) en el retorno del primer get, se encuentra un html en el cual se realizan referencias mediante href, los cuales necesitan ser cargados para visualizar lo que se está pidiendo mediante el html. Es por esta razón que sucede en el primer ingreso posee 1 GET para la url en sí, pero también suceden 13 GET adicionales que son hacia inicializadores de la aplicación web que utiliza el html retornado en el primer GET, llamando a los /assets necesarios y la funcionalidades de /application/javascript de la aplicación web.

Vemos que todo el resto de los accesos solo necesitan 1 GET, ya que se llama a rutas especificar de la aplicación y solo se pide que se haga un “render” de la ruta, lo cual se hace de manera directa a la aplicación y no es necesario cargar información adicional como material gráfico o información externa como apis, mediante más GET.

Sin embargo, para el caso de “/meme” sucede que primero se hace 1 GET para llamar al html correspondiente a la ruta de la página y enseguida se realiza otro GET, para poder obtener la asset o imagen que irá embed en el html que se solicitó anteriormente, haciendo este paso en 2 GET, uno para el html y otro para la información en bytes de la imagen que se quiere mostrar.

7. ¿Que metodo (de HTTP) se usa en el caso de la request http://192.168.1.9/power y por que? ¿Qué inconvenientes podrıa provocar el no usar ese método?

Para el request solicitado, se realiza el método GET, la razón de esto es que, se solicita la visualización del html relacionado con la ruta http://192.168.1.9:3000/power como solo lectura de lo que existe en tal ruta.

En general el protocolo http posee distintos métodos como GET, POST, DELETE, PATCH, entre otros (usando la lógica de Ruby on Rails de CRUD -- create, read, update, delete), luego como no se quiere realizar ningún cambio sobre la ruta o ingresar información adicional a la aplicación, el método que se tiene utilizar es el de GET, para leer los datos de la ruta especificada y no otro, ya que estos se utilizan con distintos fines.

Los inconvenientes que podrían ocurrir al tratar de ocupar otro método que no sea GET, pueden ser, primero, que se no se puede obtener la información necesaria para visualizar la información de la aplicación, ya que los otros métodos están destinados para editar el contenido, pero ninguno para leerlo. Dependiendo del lenguaje en que se programó la aplicación, por ejemplo RoR, la url puede estar asociado sólo al tipo READ, luego si se intenta realizar un POST para realizar, por ejemplo un CREATE, UPDATE o DELETE, se tendrá un error al tratar de realizar una acción indebida en algo que solo se dispone para

lectura. Siguiendo la lógica del punto anterior, si no se tiene seguridad en la página, si se tratara de realizar un método de edición en algo que es solo de lectura, mediante GET, se puede tener un problema al editar información que se supone no está disponible para ser editada.
Finalmente, el protocolo http es una convención para poder realizar y revisar contenido de las páginas web, luego no seguir este protocolo con sus métodos, garantiza que existan errores al ir en contra de lo que se tiene pre establecido para el uso, a nivel de capa aplicación.

Parte B

1. ¿Cuántos segmentos TCP se transmiten en cada caso? ´

Sitio	**Cantidad**	**Esperado**
http://192.168.1.9:3000/register	145	-
http://192.168.1.9:3000/	1+ 1 (304 Not Modified)	-
http://192.168.1.9:3000/	1 + 1 (Idem anterior)	-
http://192.168.1.9:3000/big.txt	4243	5850 aprox (referencia datos externos de transferencia completa)
http://192.168.1.9:3000/meme (GET /meme)	6 + 1 (200 OK)	-
(GET /meme_os.jpg)	95 + 1 (200 OK)	-
http://192.168.1.9:3000/power	9 + 1 (200 OK) + 1 (winows update)	-

Vemos que en el caso de /big.txt nosotros tuvimos un problema donde finalmente tuvimos un error tipo 500 Internal Server Error, donde se terminó de inesperadamente la transferencia de datos, debido a segmentos de TCP con la categoría Bad TCP, los cuales se verán más adelante y los paquetes involucrados.

2. ¿Cuales son los rangos de segmentos TCP que corresponden a cada mensaje HTTP?

Para los mensajes involucrados en el primer acceso de http://192.168.1.9:3000/register tenemos las siguientes estadisticas, cabe notar que estos estan en un rango entre [54, 1514] siendo la mayoría de estos paquetes de tamaño igual a los extremos, siendo alguno

simples paquetes con información pequeña y otros de mayor tamaño concerniente a /assets, referentes a los href de la parte de registro una vez que se tiene el resultado del primer GET

Packet Lengths:

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	145	658.93	54	1514	0.4356	100%	1.1300	107.345
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	76	55.58	54	78	0.2283	52.41%	0.6200	107.345
80-159	0	-	-	-	0.0000	0.00%	-	-
160-319	3	319.00	319	319	0.0090	2.07%	0.0300	107.402
320-639	7	321.29	320	322	0.0210	4.83%	0.0600	107.384
640-1279	2	908.50	738	1079	0.0060	1.38%	0.0200	107.345
1280-2559	57	1514.00	1514	1514	0.1712	39.31%	0.4000	107.346
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Para los acceso o reingresos a la url http://192.168.1.9:3000/ se tiene solo un paquete TCP que corresponde a la información desde el cliente hacia el servidor para solicitar el get de la URL anterior, como esta no ha cambiado, solo se responde con un código 304 Not Modified, luego solo se tiene este único paquete de transferencia TCP desde 192.168.1.137 -> 192.168.1.9 por un tamaño de 54 bytes. Se repite para el próximo ingreso a la misma URL con el mismo resultado.

Para request/response de http://192.168.1.9:3000/big.txt en nuestro caso obtuvimos las siguientes estadísticas:

Packet Lengths:

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	4243	1100.49	54	1514	0.3057	100%	12.5600	217.690
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	1060	54.06	54	60	0.0764	24.98%	4.3100	217.690
80-159	0	-	-	-	0.0000	0.00%	-	-
160-319	1	170.00	170	170	0.0001	0.02%	0.0100	217.691
320-639	2	378.00	378	378	0.0001	0.05%	0.0200	217.744
640-1279	357	944.30	690	946	0.0257	8.41%	1.4500	217.820
1280-2559	2823	1514.00	1514	1514	0.2034	66.53%	7.4800	217.695
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Los cuales podemos contrastar con los resultados esperados de una transferencia completa con código 200 OK al final del request.

Packet Lengths:

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	5848	1163.57	54	1514	10.6385	100%	12.6600	149.005
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	1116	54.01	54	60	2.0302	19.08%	4.2700	149.001
80-159	0	-	-	-	0.0000	0.00%	-	-
160-319	1	170.00	170	170	0.0018	0.02%	0.0100	149.002
320-639	0	-	-	-	0.0000	0.00%	-	-
640-1279	737	946.05	656	1270	1.3407	12.60%	1.4500	149.060
1280-2559	3994	1514.00	1502	1514	7.2657	68.30%	7.5500	149.005
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Vemos como por el problema que tuvimos en la transferencia de paquetes del protocolo TCP concernientes al texto gigante, falto la transferencia de un gran número de datos, generados por el error interno del servidor al tratar de concretar el request solicitado.

Para el acceso a http://192.168.1.9:3000/meme en que se solicita primero un GET para el html conformado por esta pagina y luego un GET para el asset correspondiente que se necesita se registraron las siguientes estadísticas para los paquetes TCP.

Packet Lengths:

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	103	839.69	54	1514	0.0433	100%	0.9600	233.862
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	45	54.13	54	60	0.0189	43.69%	0.4100	233.862
80-159	0	-	-	-	0.0000	0.00%	-	-
160-319	1	168.00	168	168	0.0004	0.97%	0.0100	233.862
320-639	0	-	-	-	0.0000	0.00%	-	-
640-1279	4	910.50	738	1012	0.0017	3.88%	0.0200	231.546
1280-2559	53	1514.00	1514	1514	0.0223	51.46%	0.5200	233.863
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Vemos como en este caso, se tiene un rango de valores para los paquetes entre [54,1514] igual que en los casos anteriores, solo que unda cantidad distintas de paquetes en el total del request/response.

Finalmente para el acceso a la url http://192.168.1.9:3000/power se obtuvieron las siguientes estadisticas:

Packet Lengths:

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	11	593.27	54	1514	0.5602	100%	0.1100	845.581
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	6	55.00	54	60	0.3055	54.55%	0.0600	845.581
80-159	0	-	-	-	0.0000	0.00%	-	-
160-319	0	-	-	-	0.0000	0.00%	-	-
320-639	0	-	-	-	0.0000	0.00%	-	-
640-1279	2	827.00	738	916	0.1018	18.18%	0.0200	845.597
1280-2559	3	1514.00	1514	1514	0.1528	27.27%	0.0300	845.598
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

De este último caso, vemos que se da una similitud en los rangos en que se mueven los distintos paquetes entre todas las solicitudes.

Se puede concluir que protocolo de transferencia de capa de red TCP, tiene paquetes muy estándares vistas empíricamente en la actividad, donde se pueden paquetes que son netamente de comunicación cliente-servidor y otros paquetes que son netamente información sobre lo que le retorna el servidor al cliente para visualizar en el browser, esta da la diferencia grande que existe entre los rangos de los paquetes que en general son de 54, 66 o 1514 bytes, que dejan en claro por si solos a qué tipo de paquete corresponde cada uno.

Ahora dentro de cada paquete vemos que lo que los diferencia principalmente es la cantidad en bytes que posee cada uno, ya que por ejemplo los paquetes correspondientes a la información de la url para conseguir big.txt es mucho mayor que la información en bytes que contendrán los paquetes de la dirección http://192.168.1.9:3000/power, ya que solo contendrán informacion correspondiente al html, lo cual en cantidad de bytes de información resulta ser mucho menor.

3. Primero que todo, para obtener el número de paquetes que eran duplicados, se aplicó el filtro correspondiente a "***tcp.analysis.duplicate_ack***". En nuestro caso, no encontramos paquetes duplicados

Ahora, para obtener aquellos paquetes que se encuentran perdidos, existe un filtro especializado: "***tcp.analysis.lost_segment***". En nuestro caso no se encontraron paquetes perdidos.

Finalmente, para los paquetes con señales (posibles daños, no necesariamente dañados), ocupamos el filtro de display "***tcp.analysis.flags***". En nuestro caso vemos que poseemos distintos segmentos con distintos mensajes entre ellos, Keep-Alive, Retransmission, Windows Full, ZeroWindowProbeAck, siendo estos ultimos los vistos principalmente en el problema con big.txt. Esto debido a que se saturo la cantidad de información en el buffer disponible por parte del cliente, esto debido a que a veces suceden este tipo de errores cuando se hace un "live scrolling" de algo que se está cargando, en este caso del texto gigante que no había terminado de cargar y nosotros presionaremos para ver su contenido

sin finalizar su carga. Obtuvimos las siguientes estadísticas para todos estos tipos de paquetes

Packet Lengths:

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	254	68.43	54	946	0.0003	100%	0.7300	217.711
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	250	54.38	54	60	0.0003	98.43%	0.7300	217.711
80-159	0	-	-	-	0.0000	0.00%	-	-
160-319	0	-	-	-	0.0000	0.00%	-	-
320-639	0	-	-	-	0.0000	0.00%	-	-
640-1279	4	946.00	946	946	0.0000	1.57%	0.0100	217.929
1280-2559	0	-	-	-	0.0000	0.00%	-	-
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

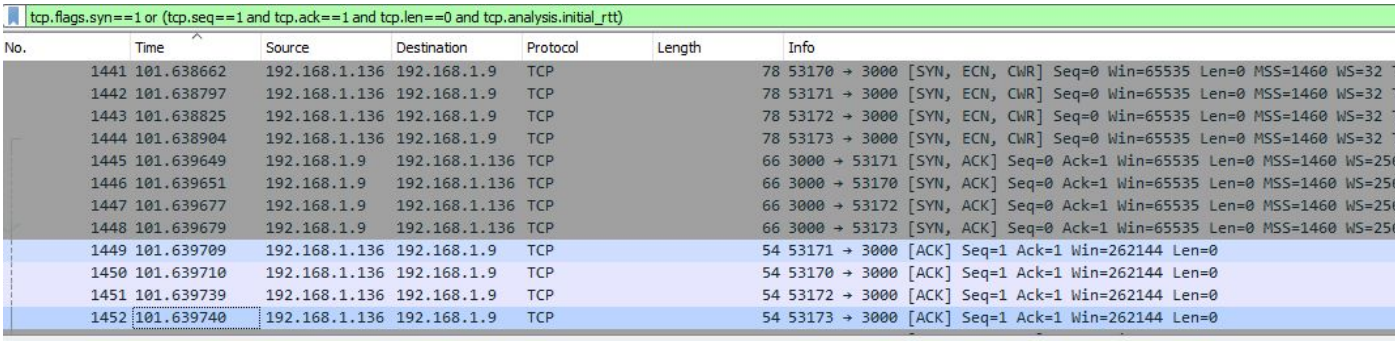
Muchos de estos paquetes, indican por ejemplo que el buffer se llenó, lo que produce que se detenga la transmisión como son el caso de los ZeroWindowProbe y ACK. También están los que se presentan en caso de desconexión (ocurrido durante la actividad con /power) que se representan principalmente por el identificador Retransmission entre otros, siendo estos últimos los candidatos mas favorables de presentar paquetes perdidos/dañados.

4. Para identificar la secuencia de handshake se utilizó el siguiente filtro: ***tcp.flags.syn==1 or (tcp.seq==1 and tcp.ack==1 and tcp.len==0 and tcp.analysis.initial_rtt)*** (Fuente: https://blog.packet-foo.com/2015/03/advanced-display-filtering/)

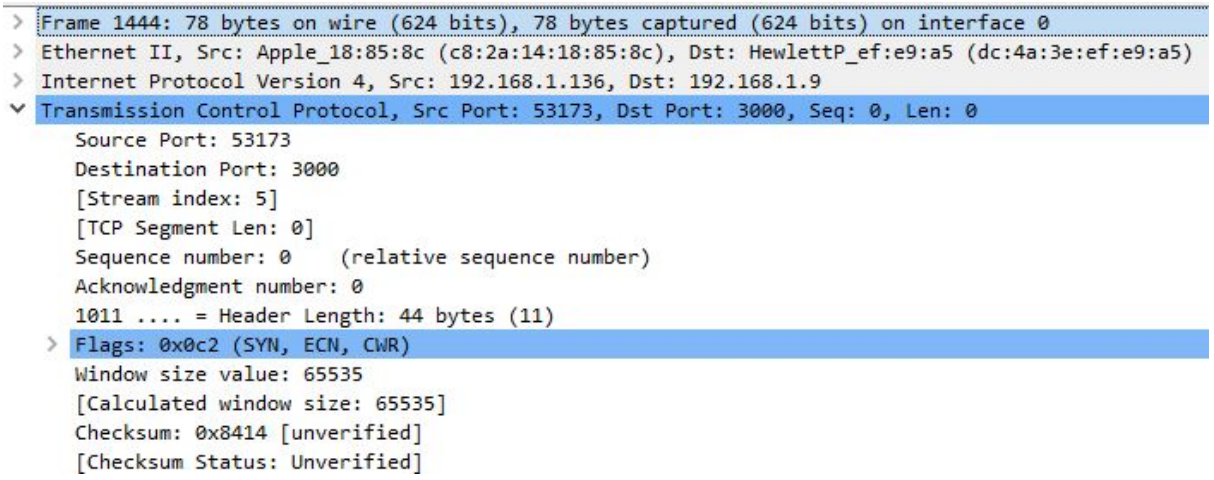
La explicación de este filtro es primero, sabemos que cada SYN tiene un relative sequence number igual a 0, por lo tanto el tercer paquete del handshake tendra relative sequence number de 1, por lo tanto obtenemos los filtros "**tcp.seq==1**". Cómo debemos recibir el "acknowledge" el SYN/ACK desde el servidor sabemos que el número ACK también es 1, luego "**tcp.ack==1**".

Excluimos todos los paquetes que contienen un payload de TCP, porque el tercero está vacío, agregando la restricción "**tcp.len==0**". Finalmente, mediante la búsqueda de información respecto a filtros de display, se utiliza ***tcp.analysis.initial_rtt*** para poder filtrar lo que necesitamos.

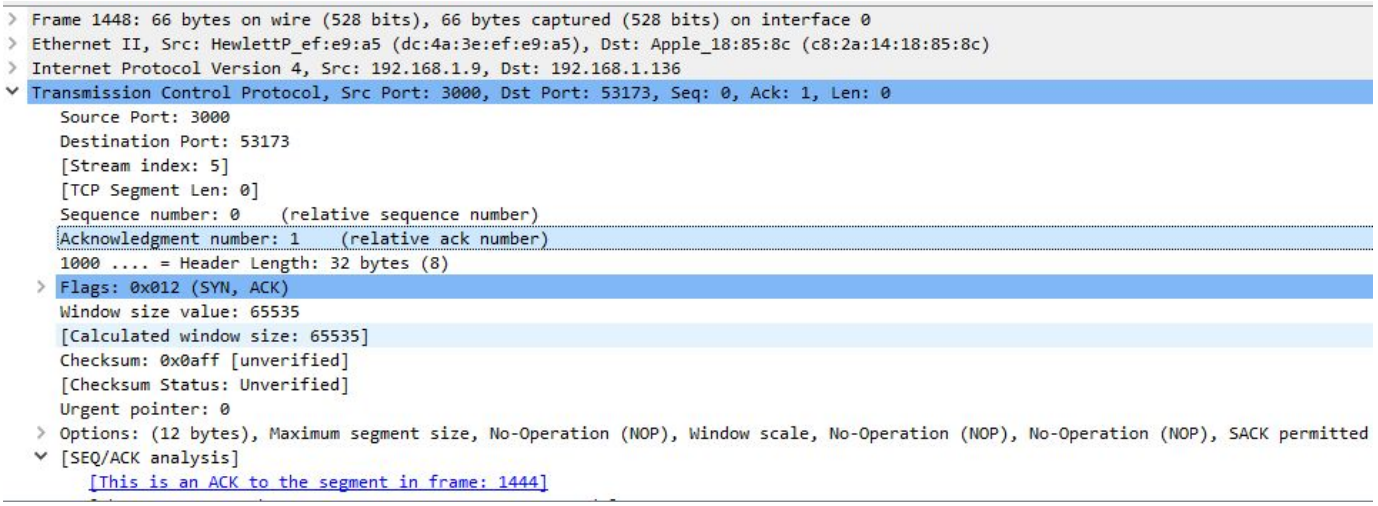
Un ejemplo de handshakes que fueron detectados en la actividad se muestra en la imagen adjunta.



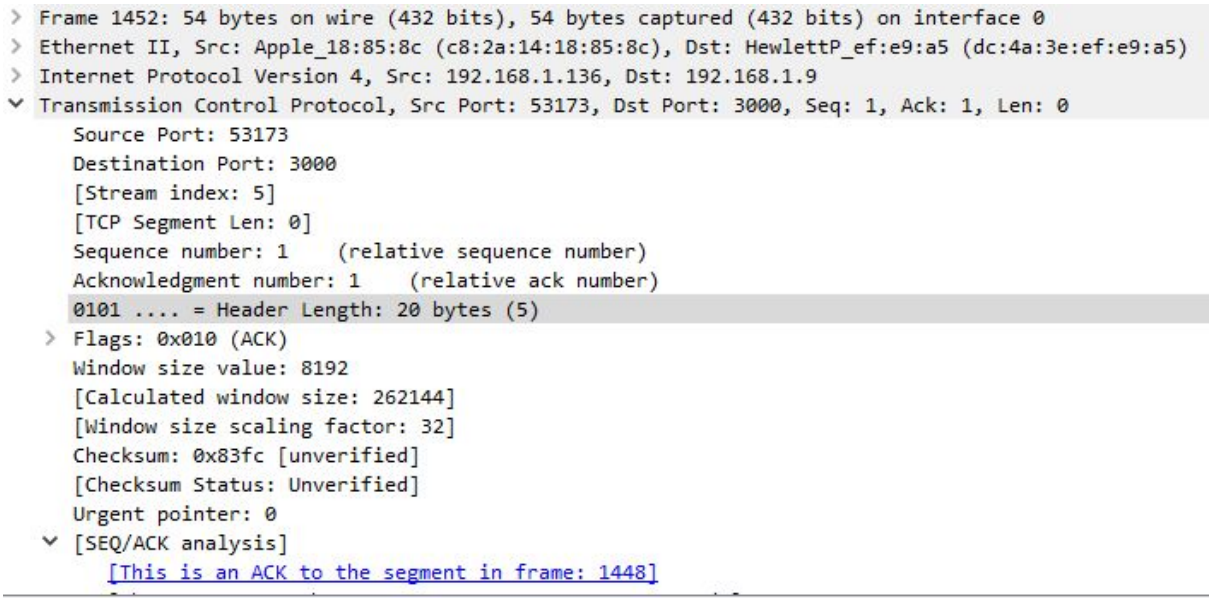
Vemos como se da el handshake triple en esta situación, primero los paquetes entre [1441,1444] son el primer paso, poseen un Sequence Number de 0 y un Acknowledgment number de 0 de igual formar, al ser estos la primera parte del triple handshake, desde nosotros (192.168.1.136) hacia el servidor (192.168.1.9). Donde se hace un SYN (Synchronize), con el cual se pide sincronizar el cliente con el servidor para la transferencia.



Ahora tenemos la respuesta, desde el servidor (192.168.1.9) hacia nosotros (192.168.1.136), con los paquetes con números de frame entre [1445, 1448], que contienen Sequence Number igual a 0 y Acknowledgment Number igual a 1, esto debido a que surgen como respuesta a los paquetes de request anteriores. En palabras simples, el servidor le da a entender al cliente que ya sabe que se está tratando de sincronizar con el (Synchronize + acknowledgment)



Finalmente, vemos la respuesta como parte final del handshake, que involucra los paquetes con frames [1449, 1452], con origen nuestra IP (192.168.1.136) al servidor (192.168.1.9). Estos contienen un Sequence Number igual a 1 y un Acknowledgment number igual a 1, esto debido a que surgen como la respuesta final del trío que conforma el handshake, son la respuesta al ACK por parte de servidor, con request de nuestra parte, como se ve a continuación. Se entiende de manera simple, como que se termina el handshake cliente - servidor mediante el sequence number distinto a los anteriores (puede ser un número aleatorio distinto de 1 o 0)



.

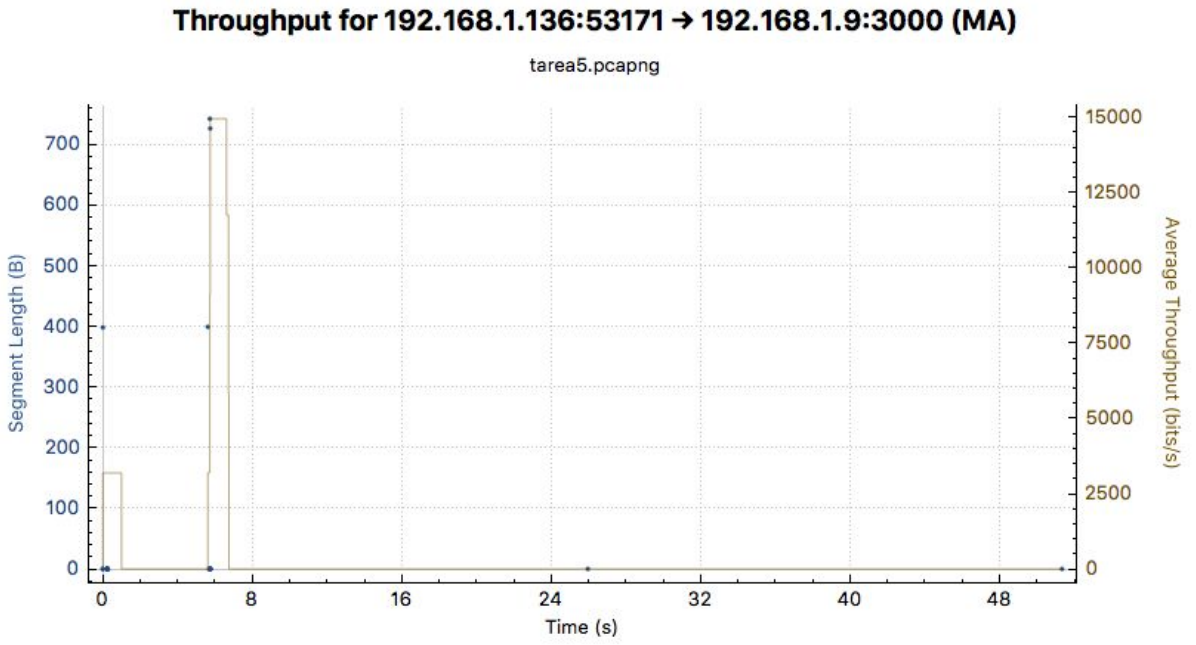
5.

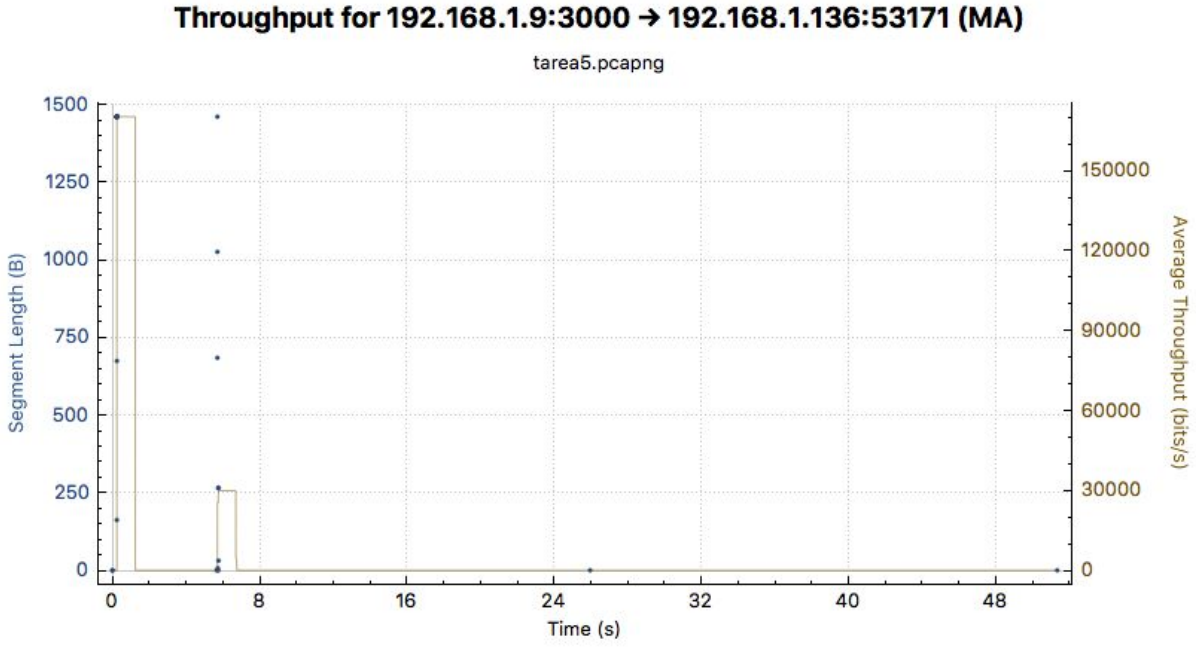
Throughput es una medida de cuántas unidades de información un sistema puede procesar en una cierta cantidad de tiempo. Teóricamente para calcular el throughput debemos calcular:

$$Transmission\ time = \frac{Size\ of\ file}{Bandwidth}$$
$$Throughput = \frac{Size\ of\ file}{Transmission\ time}$$

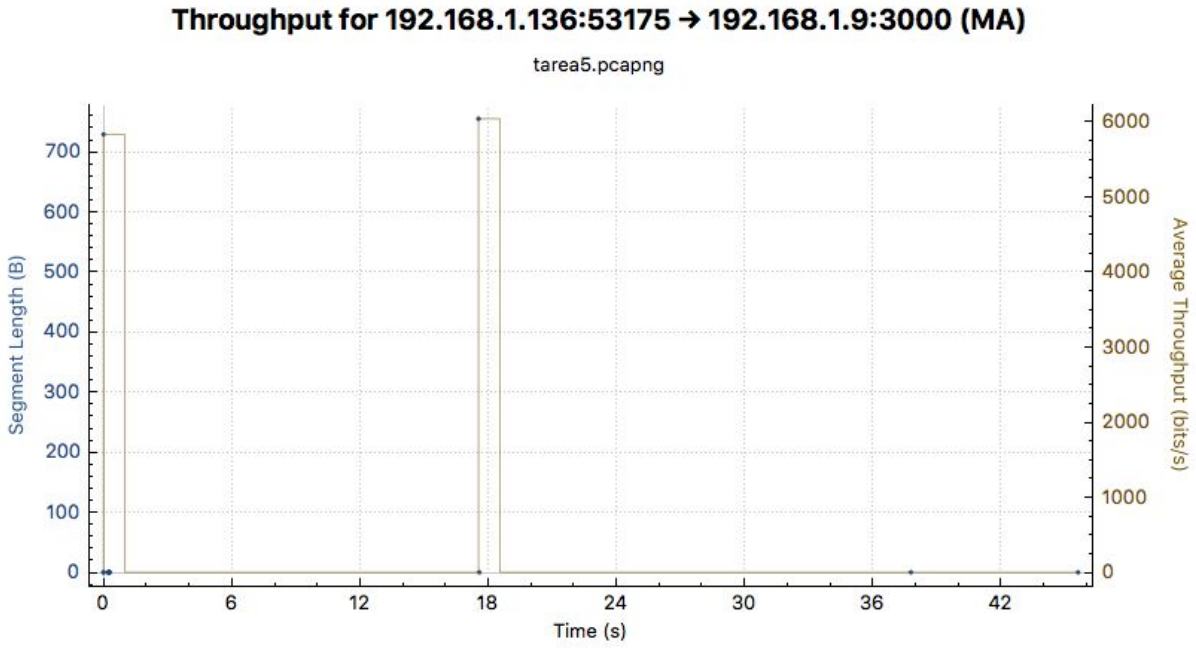
A continuación revisaremos los throughputs para cada acceso:

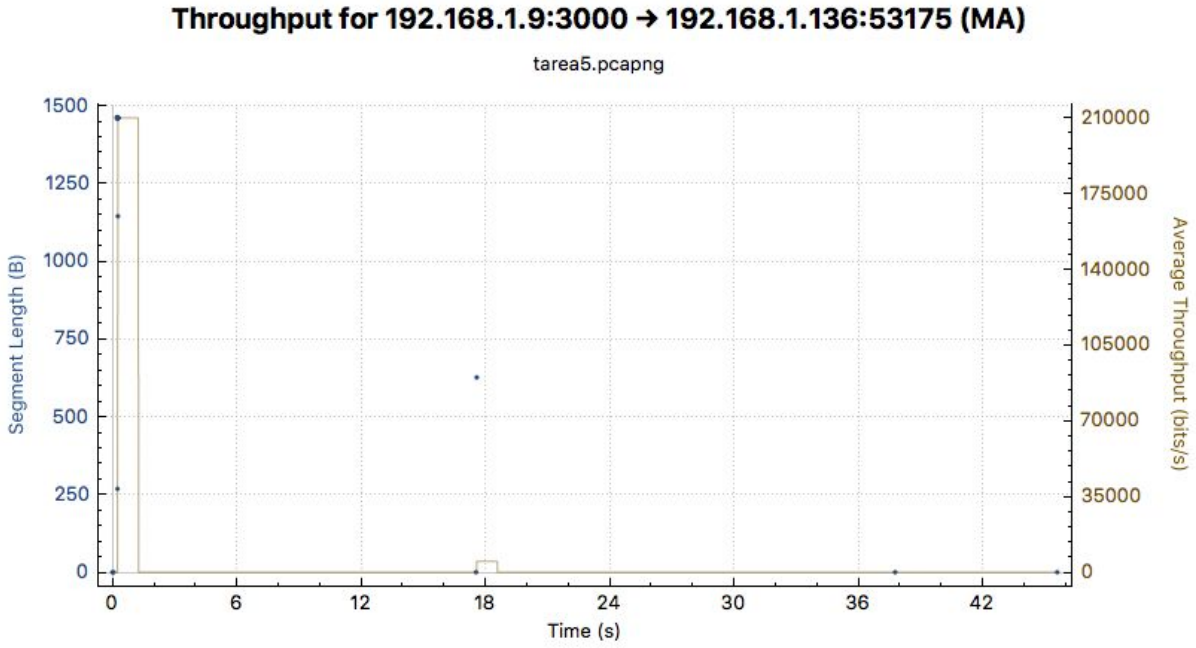
1. /register un peak de casi 15000 (bits/s)



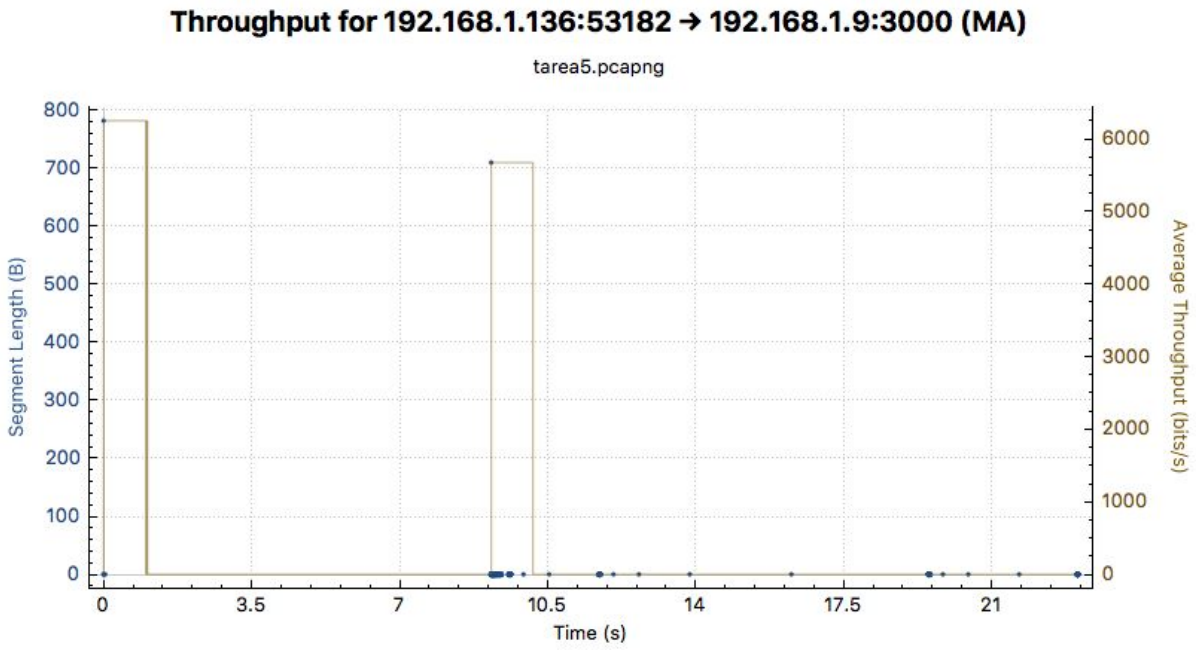


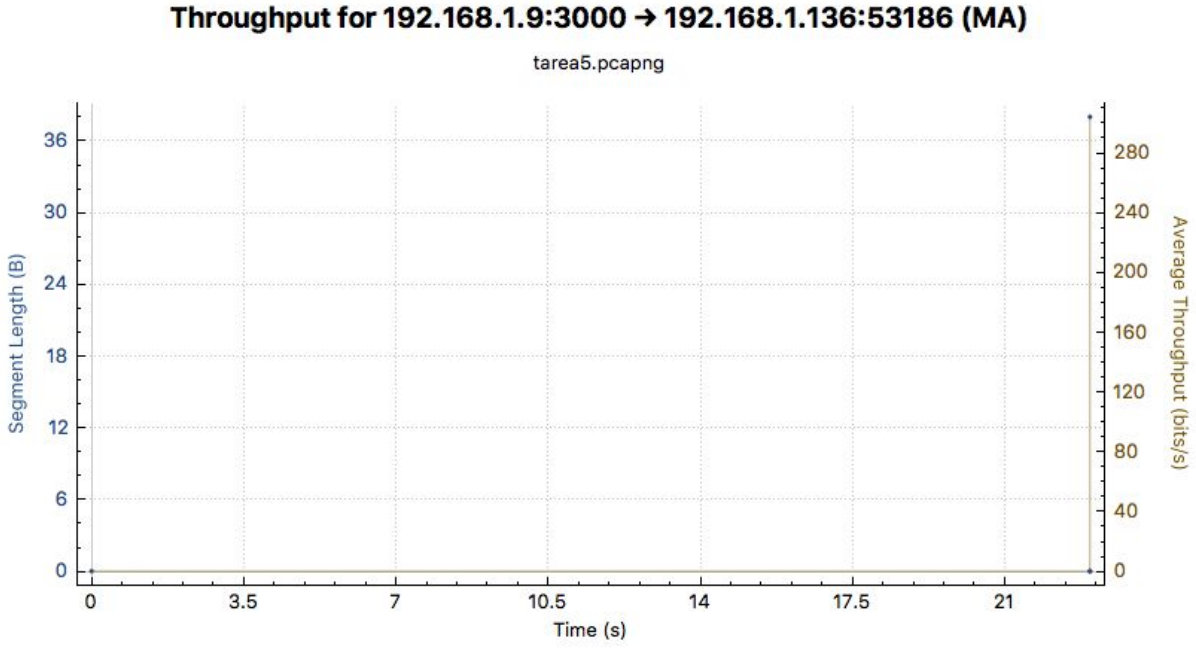
2. http://192.168.1.9:3000/ con un peak de 6000 bits/seg



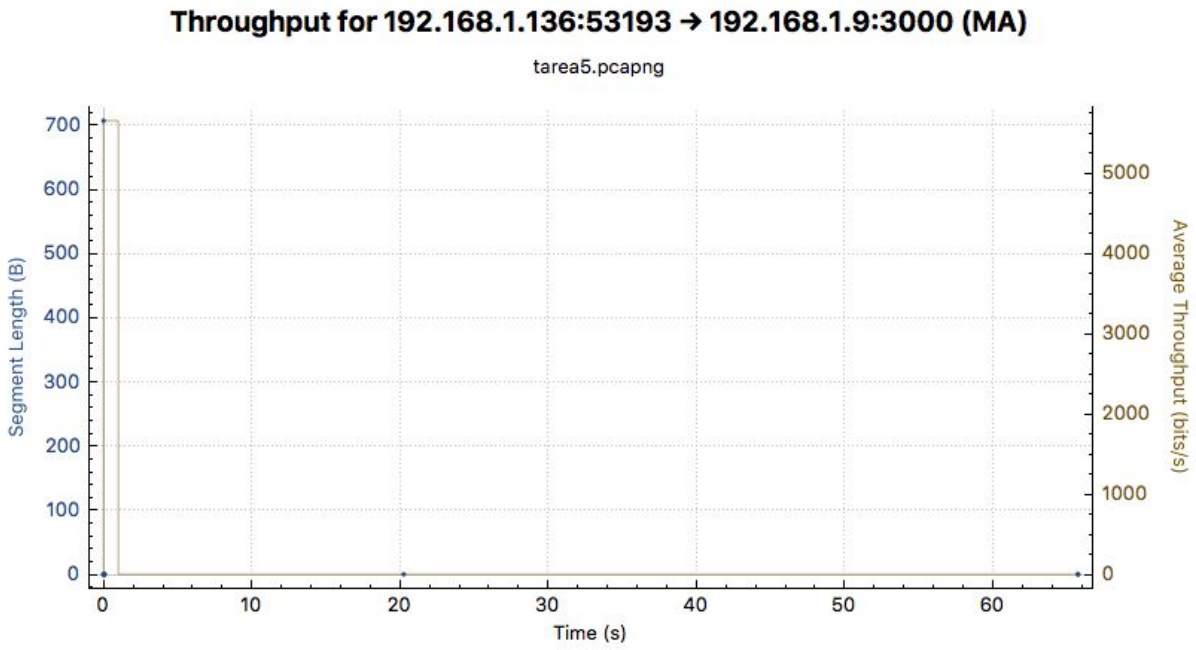


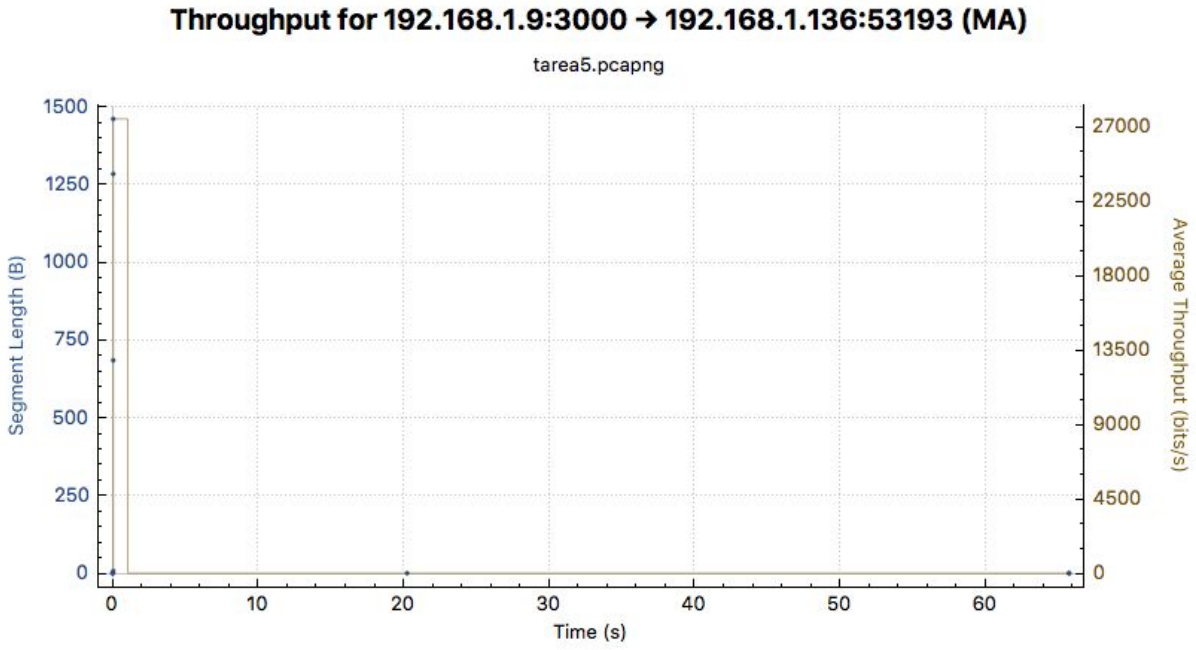
3. http://192.168.1.9:3000/big.txt un peak de 6000 bits/s de ida pero de vuelta 0. Esto puede ser debido a que la respuesta del servidor en este caso fue un 500.





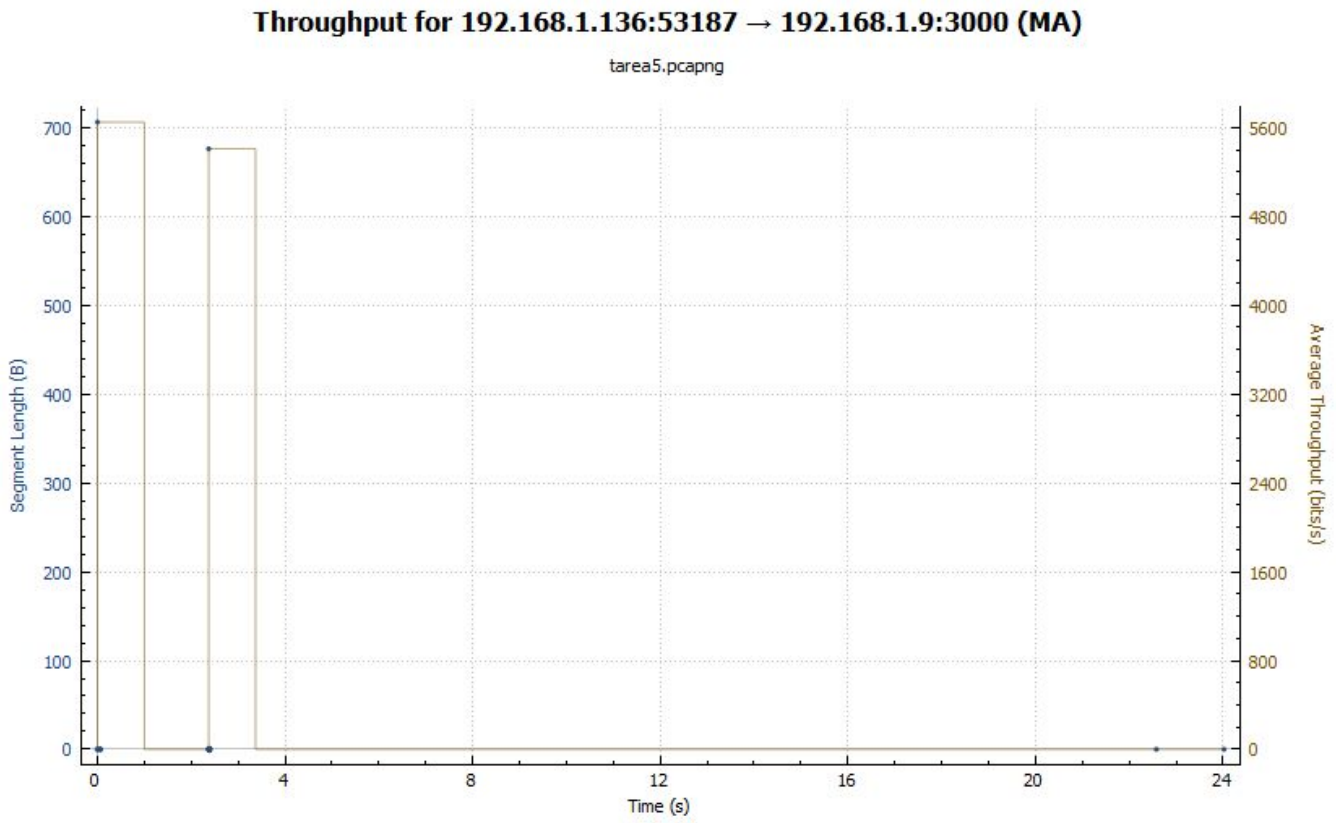
4. http://192.168.1.9:3000/power

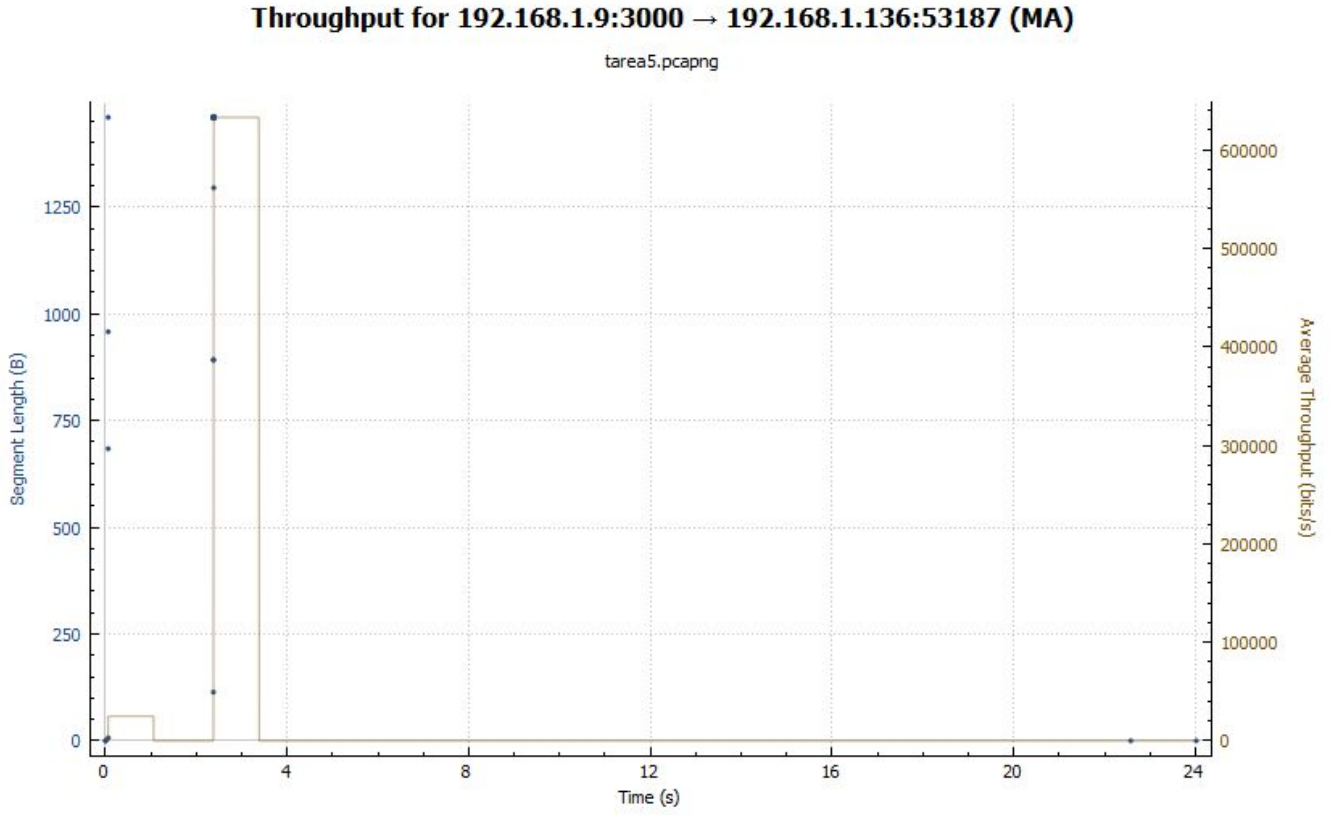




5. Throughput grafico para el acceso a http://192.168.1.9:3000/meme

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
192.168.1.136	53187	192.168.1.9	3000	103	86 k	44	2376	59	84 k	231.488713	2.3806	7984	


Throughput for 192.168.1.136:53187 → 192.168.1.9:3000 (MA)
tarea5.pcapng
Segment Length (B)
Average Throughput (bits/s)
Time (s)
0
100
200
300
400
500
600
700
800
1600
2400
3200
4000
4800
5600
4
8
12
16
20
24


Throughput for 192.168.1.9:3000 → 192.168.1.136:53187 (MA)
tarea5.pcapng
Segment Length (B)
Average Throughput (bits/s)
Time (s)
0
250
500
750
1000
1250
100000
200000
300000
400000
500000
600000
4
8
12
16
20
24

Parte C

A continuación se presentan los miembros observados en la red de acuerdo con el protocolo ARP.

Dirección MAC	Dirección IP	Fabricante
f4:0f:24:0a:45:69	192.168.1.127	Apple, Inc.
c8:2a:14:18:85:8c	192.168.1.136	Apple, Inc.
b8:e8:56:37:a7:40	169.254.109.189	Apple, Inc.
84:38:35:41:43:1a	169.254.61.66	Apple, Inc.
38:c9:86:54:a3:88	192.168.1.140	Apple, Inc.
20:c9:d0:82:aa:f7	192.168.1.142	Apple, Inc.
98:01:a7:99:25:e7	169.254.39.187	Apple, Inc.
0c:4d:e9:a4:13:18	192.168.1.105	Apple, Inc.
0c:4d:e9:b6:a6:0e	192.168.1.119	Apple, Inc.
10:dd:b1:b7:b0:76	192.168.1.128	Apple, Inc.
ac:bc:32:c6:d4:1f	192.168.1.130	Apple, Inc.
98:5a:eb:ce:13:70	192.168.1.108	Apple, Inc.
8c:29:37:f1:ec:4c	192.168.1.116	Apple, Inc.
2c:56:dc:37:7a:d0	192.168.1.103	ASUS
dc:85:de:7f:27:c5	192.168.1.131	AzureWave
9c:eb:e8:5d:67:86	192.168.1.101	BizLink
20:89:84:87:19:ef	192.168.1.113	Compal Information
54:b8:0a:68:8d:31	169.254.230.123	D-Link
98:e7:f4:53:0b:06	192.168.1.117	Hewlett Packard
dc:4a:3e:ef:e9:a5	192.168.1.9	Hewlett Packard
60:6d:c7:0b:c5:a5	169.254.160.187	Hon Hai Precision
74:29:af:f8:77:b3	192.168.1.111	Hon Hai Precision

5c:c5:d4:4b:3e:7b	192.168.1.138	Intel Corporate
34:e6:ad:81:a1:82	192.168.1.143	Intel Corporate
a0:a8:cd:cb:02:b9	169.254.115.223	Intel Corporate
c8:f7:33:d2:d8:77	192.168.1.104	Intel Corporate
b4:6d:83:d8:62:d8	192.168.1.45	Intel Corporate
50:7b:9d:d7:1a:0b	192.168.1.107	LCFC
a4:db:30:07:af:7b	169.254.220.123	Liteon Technology
30:52:cb:84:f8:a7	192.168.1.120	Liteon Technology
40:f0:2f:fe:3d:cc	192.168.1.121	Liteon Technology
00:1c:42:5a:51:78	0.0.0.0	Parallels
08:9e:01:6f:55:71	192.168.1.112	Quanta Computer
c4:54:44:b8:a8:61	192.168.1.110	Quanta Computer
00:e0:4c:68:00:37	192.168.1.106	Realtek
c0:25:e9:2e:53:8a	192.168.1.124	TP-Link

Podemos ver que la dirección mac 00:1c:42:5a:51:78 tiene asociada la IP 0.0.0.0. Esto sucede cuando el dispositivo no tiene configurada una IP de salida y se usa la IP null. O en general cuando se ocupa una maquina virtual com.

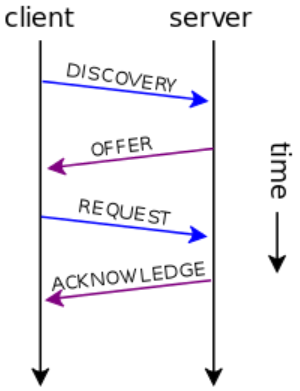
Parte D

1. DHCP (*Dynamic Host Configuration Protocol*):es un protocolo cliente/servidor que permite al servidor asignar de manera dinámica una dirección IP a un computador, de un rango de direcciones posibles, configuradas para una cierta red.

El protocolo asigna direcciones IP cuando un sistema es iniciado de manera general como sigue:

1) Un usuario enciende un computador con un cliente DHCP
2) Cliente envía un broadcast request (Discover), buscando una respuesta de DHCP del servidor para que le asigne una IP
3) El router redirige el paquete DISCOVER al servidor DHCP

4) Servidor determina la IP adecuada para el cliente (puede enviar una IP tentativo OFFER y se la reserva).
5) Cliente envía un paquete REQUEST, diciendo al servidor que planea ocupar esa IP
6) Servidor envía de vuelta un ACK (DHCPACK) que confirma al cliente el uso de esa IP por un periodo de tiempo



En resumen, se tiene los siguientes tipo de mensajes:

DHCPDiscover

DHCPOffer

DHCPRequest

DHCPAcknowledgement

DHCPInformation

DHCPRelease

Cada uno con sus estructuras internas para el header del mensajes, bajo los criterios que cada uno de estos amerita, en particular nos interesa la estructura del mensajes DHCPDiscover que son los que en su mayoría fueron capturados en la actividad y tan solo 9 fueron del tipo REQUEST.

```
> Frame 14095: 590 bytes on wire (4720 bits), 590 bytes captured (4720 bits) on interface 0
> Ethernet II, Src: D-LinkIn_68:8e:a2 (54:b8:0a:68:8e:a2), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
> User Datagram Protocol, Src Port: 68, Dst Port: 67
v Bootstrap Protocol (Discover)
    Message type: Boot Request (1)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0x556a4700
    Seconds elapsed: 0
  > Bootp flags: 0x0000 (Unicast)
    Client IP address: 0.0.0.0
    Your (client) IP address: 0.0.0.0
    Next server IP address: 0.0.0.0
    Relay agent IP address: 0.0.0.0
    Client MAC address: D-LinkIn_68:8e:a2 (54:b8:0a:68:8e:a2)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  > Option: (53) DHCP Message Type (Discover)
  > Option: (61) Client identifier
  > Option: (12) Host Name
  > Option: (60) Vendor class identifier
  > Option: (55) Parameter Request List
  > Option: (255) End
    Padding: 000000000000000000000000000000000000000000000000...
```

Este protocolo tiene la dependencia de BOOTS que es un protocolo que funciona a nivel de capa Transporte.

En la actividad, en los registros de la actividad, solo se encuentran paquete de la categoría DISCOVER, INFORM y REQUEST que poseen las siguientes estadísticas en torno al tamaño de paquete y la cantidad. En general, las direcciones de origen son 0.0.0.0 ya que estos son clientes que no tienen la conexión TCP establecida, luego se utiliza esta para cuando se solicita una IP. Mientras que el destino es 255.255.255.255 para referirse al servidor de manera “mask” al cual se le solicita y entrega las ips disponibles a los clientes que se están conectando.

Packet Lengths:

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	993	504.36	342	590	0.0008	100%	0.0300	123.214
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	0	-	-	-	0.0000	0.00%	-	-
80-159	0	-	-	-	0.0000	0.00%	-	-
160-319	0	-	-	-	0.0000	0.00%	-	-
320-639	993	504.36	342	590	0.0008	100.00%	0.0300	123.214
640-1279	0	-	-	-	0.0000	0.00%	-	-
1280-2559	0	-	-	-	0.0000	0.00%	-	-
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Vemos que existieron un total de 993 paquete con el protocolo de DHCP, con un tamaño entre 320- 639 bytes en los paquetes enviados. El promedio de los paquete enviados es de 504 bytes. Se ve que la frecuencia con la cual se hacen los registros de DHCP es de 0.0008 ms, lo cual representa que se hacen muy frecuentemente, esto debido a que continuamente se está registrando las IPs disponibles y los que están conectados. Estos paquete tienen información de la dirección MAC del destino-origen, el protocol, tamaño del mensaje y la ip que se quiere/tiene asignada y transfieren mediante el protocolo UDP de capa de transporte.

La utilidad de este protocolo, es poder administrar la asignación de IP para la red de manera automática bajo ciertos criterios, pero también seguir el rastro de los conectados y la reasignación de IP en desconexiones.

2. **ICMP** (*Internet Control Message Protocol version 4*) Es un protocolo de capa de red utilizado para reportar errores comunes y eventos en los protocolos de IP, TCP y UDP, se utiliza como forma de manejar mensajes de control(ping, trace, echo).

ICMP es parte del protocolo IP y es utilizado por routers, dispositivos intermedios o hosts para comunicar errores de información o actualización a otros routers, dispositivos o hosts, se tiene la versión 6 de este protocolo conocido como ICMPv6, que se utiliza en nodos IPv6.

Los mensaje ICMP se transmiten como “datagrams” y consisten de un header IP que encapsula la información ICMP. Los paquete ICMP son paquete IP con ICMP en la porción de información de IP. Los mensajes contienen 1) *Type* que identifica el tipo de mensaje (en el caso analizado es el tipo 8 que es una solicitud de ping) 2) *Code* que contiene información adicional (en este caso de ping es 0) y 3) *Checksum* que permite detectar errores durante la transmisión (en los casos analizados corresponde a *Checksum:Good*)

Las estadísticas obtenidas durante la actividad para este protocolo, que fue principalmente en cuando se realizaban ping a las demás conexiones, se presentan a continuación:

Packet Lengths:

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	209	98.00	98	98	0.0002	100%	0.0200	362.368
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	0	-	-	-	0.0000	0.00%	-	-
80-159	209	98.00	98	98	0.0002	100.00%	0.0200	362.368
160-319	0	-	-	-	0.0000	0.00%	-	-
320-639	0	-	-	-	0.0000	0.00%	-	-
640-1279	0	-	-	-	0.0000	0.00%	-	-
1280-2559	0	-	-	-	0.0000	0.00%	-	-
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Vemos que existieron 209 paquetes en este protocolo (solo se considero ICMP no ICMPv6), que corresponden a los distintos ping que se realizaron tanto los hechos por nosotros como los que fueron efectuados hacia nosotros. Estos paquetes contienen información sobre un evento en que una IP “pregunta” a otra IP si existe y esta responde Si o No y este protocolo encapsula la comunicación para este tipo de mensajes mediante PING

Vemos que en general todos los paquetes constan de 98 bytes conteniendo lo enunciado anteriormente, más otra información propia del protocolo.

Vemos que para experiencia de ping, cuando se solicitaba un ping a una ip, esta responde con un desfase de 1 seg con la información anterior Type=8, Code=0, Checksum=Good, y consiste en casi el mismo mensaje ida y vuelta, con la excepción que se distingue cuando es “request” y “reply”

3. **UDP**(User Datagram Protocol): Es un protocolo de capa de transporte, que define un mecanismo de transferencia de mensajes, en este caso *datagrams,* a otros host en protocolo de Internet (IP), sin requerir una conexión previa para poder crear un canal de comunicación entre las partes (connection-less).

Las característica de ser *coneccion-less* son; no ser necesario establecer conexión previa, no mantener el estado de conexión y cada segmento se gestiona de manera independiente. Esto puede traer problemas en 1) el orden en que llegan los paquetes 2) se pueden perder paquetes en el envio de informacion. Pero en general es un protocolo altamente usado en servicios *Real-Time-Services,* como transmisiones en vivo.

La estructura de los datagramas del protocolo UDP, siguen el siguiente formato:

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

Source y Destination Port, corresponden a los puertos respectivos de origen y destino de la transferencia.

Length, especifica el largo en bytes del UDP header y UDP data, el tamaño teórico es de 65535 bytes dados por 8 bytes del header + 65527 bytes de data del datagram a entregar.

Checksum, se utiliza como forma de chequear errores en el header y en data del datagram, es opcional en IPv4 y obligatorio en IPv6

Para la experiencia se obtuvieron distintos resultados, donde request de distintos protocolos se basan en UDP, luego diversas acciones que suceden fuera de las interacciones que realiza el usuario, se realizan bajo este protocolo. Por ejemplo, acciones hechas bajo

protocolos como DHCP, LLMNR, MDNS, NBNS, SSDP, entre otros, se registran con paquetes hechos mediante datagrams.

4. **MDNS** (*Multicast Domain Name System*) Es un protocolo de la capa de aplicación, utilizado para transcribir dominios en direcciones IP, utilizado generalmente en redes pequeñas que no incluyen un un nombre de servidor local. Este posee el protocolo UDP para la transferencia de datos.

La función principal de Multicast DNS es establecer una correspondencia de dominios, contra direcciones IP en ausencia de un servidor DNS especializado, con solo realizar cambios mínimos al protocolo DNS tradicional. Un problema de este protocolo es que está diseñado para trabajar en el dominio *.local.* Es decir, no hay intercambio de consultas y respuestas entre varias redes IP, ya que cada dominio *.local* está constituido por una red local.

Un mDNS Ethernet frame es un multicast con formato UDP para:

- Dirección Mac
- direcciones IPv4 o IPv6
- UDP port 5353
-

Además se tiene los siguientes formatos que caracterizan a este tipo de protocolo:

Queries: Se tienen los siguientes formatos que se explican en la tabla adjunta:

Field	Description	Length bits
QNAME	Name of the node to which the query pertains	Variable
QTYPE	The type of the query, i.e. the type of RR which should be returned in responses.	16
UNICAST-RESPONSE	Boolean flag indicating whether a unicast-response is desired	1
QCLASS	Class code, 1 a.k.a. "IN" for the Internet and IP networks	15

Resources Records: Todos registros en las respuesta del servidor tienen el mismo formato conocido como Resources Records (RR), los tipos se presentan a continuación en la tabla adjunta.

Field	Description	Length bits
RRNAME	Name of the node to which the record pertains	Variable
RRTYPE	The type of the Resource Record	16
CACHE-FLUSH	Boolean flag indicating whether outdated cached records should be purged	1
RRCLASS	Class code, 1 a.k.a. "IN" for the Internet and IP networks	15
TTL	Time interval (in seconds) that the RR should be cached	32
RDLENGTH	Integer representing the length (in octets) of the RDATA field	16
RDATA	Resource data; internal structure varies by RRTYPE	Variable

Las estadísticas obtenidas de la experiencia entorno a los paquetes bajo el protocolo mdns se explican a continuación:

Packet Lengths:

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	4682	228.95	78	1066	0.0036	100%	0.1100	188.679
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	23	78.09	78	79	0.0000	0.49%	0.0200	339.349
80-159	1693	117.01	81	158	0.0013	36.16%	0.0700	668.982
160-319	1996	217.48	160	319	0.0015	42.63%	0.0600	188.696
320-639	911	431.58	320	634	0.0007	19.46%	0.0500	228.516
640-1279	59	758.73	647	1066	0.0000	1.26%	0.0400	69.056
1280-2559	0	-	-	-	0.0000	0.00%	-	-
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Se ve como existen un total de 4682 paquetes, con un promedio de largo de 228 bytes, oscilando entre valores [78, 1066], con una frecuencia de 0.0036ms.

Se ve cómo todos estos paquetes surgen en la medida en que se quiere saber informacion de la ip del dominio o cuando el servidor pregunta por alguna de las conexiones actuales y al ser un sistema multicast, todos ven estas preguntas y el que corresponda con las condiciones de la “query” responden a este request.

5. **LLMNR**(*Link-Local Multicast Name Resolution*): Es un protocolo basado en el protocolo de capa aplicación DNS (*Domain Name System*) siguiendo el formato de los paquetes bajo este protocolo, pero usando puerto distintos para su comunicación. Este protocolo permite hosts bajo el protocolo IPv4 y IPv6 realizar una resolución del nombre del host bajo escenarios bajo los que no es posible utilizar DNS, bien porque no existe ningún servidor o porque no es posible alcanzarlo.

Respecto a las *queries,* los que realizan los response a las request, utilizan el puerto UDP 5355 para escuchar, ya sea utilizando IPv4 o IPv6. También se puede dar que estos escuchen bajo el puerto TCP 5355, bajo otras condiciones.

La forma de los header de los paquetes entregados bajo este protocolo se muestran a continuación en un cuadro.

Offset	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	ID															
16	**QR**	**Opcode**				**C**	**TC**	**T**	**Z**	**Z**	**Z**	**Z**	**RCODE**			
32	QDCOUNT															
48	ANCOUNT															
64	NSCOUNT															
80	ARCOUNT															

ID: Representa el ID de la transacción

Flags:
QR(Query/Response), Opcode, Conflict (C) , Truncated (TC), Tentative (T), extras (Z), ResponseCode (RCODE)

QDCOUNT: Cuenta de preguntas en la entrada de preguntas
ANCOUNT: Cuenta de respuestas en la entrada respectiva.
NSCOUNT: Cuenta de Server Resource Records en la entrada de “Authority Records”
ARCOUNT: Cuenta de Resource Records en la entrada de “Aditional Records”

Para la experiencia se registraron las siguientes estadísticas en torno a los paquetes netamente bajo el protocolo LLMNR:

Packet Lengths:

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	3040	81.43	64	95	0.0023	100%	0.4800	1050.101
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	1434	70.99	64	75	0.0011	47.17%	0.2400	1050.101
80-159	1606	90.76	84	95	0.0012	52.83%	0.2400	954.338
160-319	0	-	-	-	0.0000	0.00%	-	-
320-639	0	-	-	-	0.0000	0.00%	-	-
640-1279	0	-	-	-	0.0000	0.00%	-	-
1280-2559	0	-	-	-	0.0000	0.00%	-	-
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Vemos que existen dos tipos principales de paquetes, entre 64-75 y 84-95, estos paquetes vemos que están en cantidades similares, su gran diferencia es para el protocolo IP para el cual funciona, vemos que para los paquetes de menor tamaño son para los protocolos con IPv4, mientras que los más pesados son para el IPv6, principalmente para la cantidad de bytes dirigidos para determinar las ips de origen y destino en el header de Ethernet correspondiente.

Vemos que estos paquetes no son generados por los clientes (nosotros), sino que por ips principalmente del host (192.168.1.9) y otras ips que no son propias de clientes sino del servidor.

La frecuencia de crearon y transferencia (mediante UDP principalmente), suceden durante toda la actividad con frecuencia de 0.0011 y 00012 ms en promedio, esto debido a que son necesarios que los servidores tengan una resolución del nombre del host, siguiendo este protocolo principalmente por la ausencia de DNS, al ser una conexión en LAN.