



## Tarea 03

Grupo N°5: Juan Campos - Gustavo Prudencio

31 de octubre de 2018

### 1. Diagrama de la Red

A continuación se muestra el diagrama de la red, el computador con fondo azul representa el servidor montado en el computador del ayudante, el cual estuvo conectado al repetidor central y este a su vez estuvo conectado a otros 2 repetidores. Cada grupo se conecto via cable ethernet a uno de estos repetidores o via wifi a los modem wifi conectados a estos repetidores.

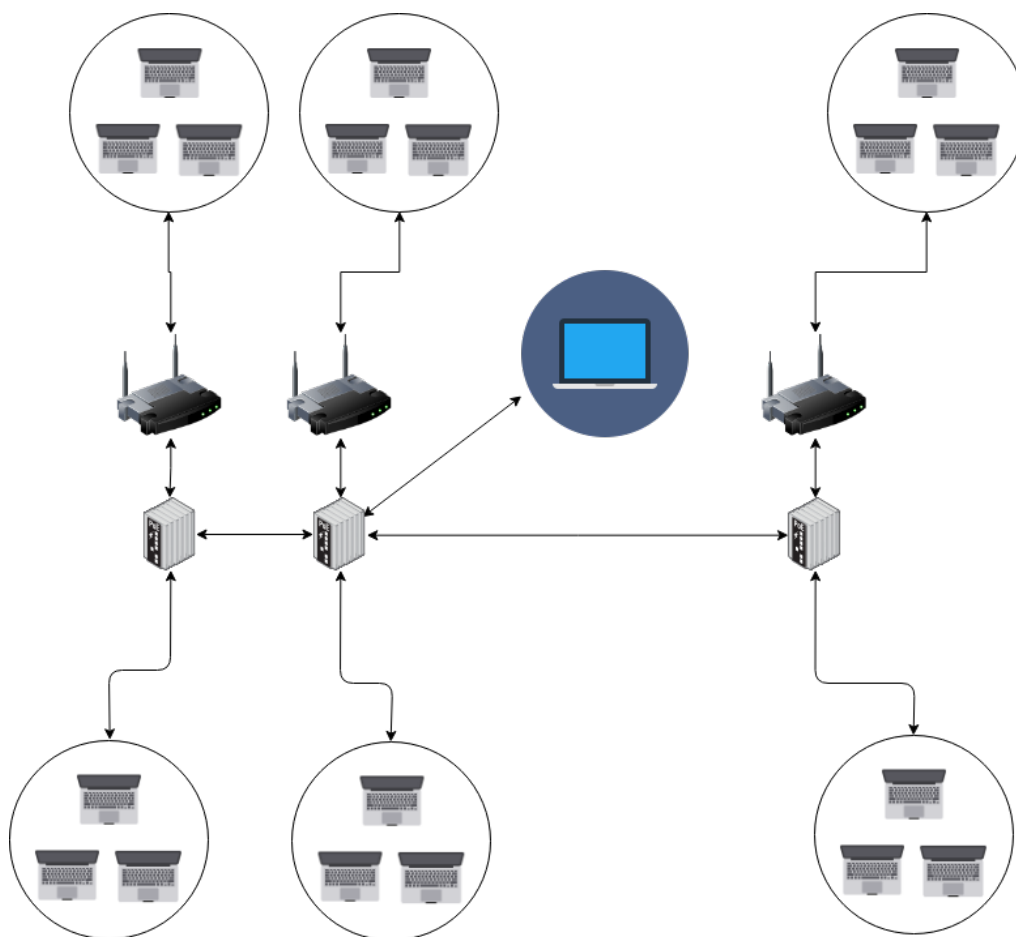


Figura 1: Diagrama

## 2. Parte a)

### 2.1. ¿Que browser hace la solicitud?

Google Chrome

### 2.2. ¿Que browser hace la solicitud?

TCP

### 2.3. Para cada acceso, ¿cual es el código HTTP de respuesta?

200 Ok

### 2.4. ¿Cual es la dirección obtenida de la imagen de la primera vista?

/comando/ultra/secreto/logouc\_top.png :

Esta imagen carga al momento de realizar una request get al sitio <http://192.168.1.9:3000/>.

### 2.5. ¿Que tipo de paquetes se registraron cuando abrieron la dirección reemplazando el nombre de la imagen por win98.mp3? ¿Por que existen múltiples paquetes de este tipo luego de acceder? ¿Cuántos *bytes* se transmitieron del archivo?

Una vez hecha la solicitud HTTP por *win98.mp3* se registraron multiples paquetes TCP. Esto sucede producto que se empieza a descargar el audio en nuestro browser y este se envia por partes a travez de paquetes TCP, los cuales envian señales de cuan largo es el fragmento y asi sabe el browser en que byte deberia partir el proximo paquete, este por su parte envia una señal TCP indicando que recibio el paquete bien.

Del archivo *win98.mp3* se transmitieron 129361 bytes contenidos en 16 paquetes TCP.

```
✓ [16 Reassembled TCP Segments (129361 bytes): #18275(166), #18277(8192), #18279(2896), #18281(5296), #18283(8192), #18285(1448), #18287(6744), #18289(8192), #18291(1448), #18292(1448), #18294(11584), #18296(8688), #18298(15928), #18300(15928), #18302(20272), #18304(12939)]
[Frame: 18275, payload: 0-165 (166 bytes)]
[Frame: 18277, payload: 166-8357 (8192 bytes)]
[Frame: 18279, payload: 8358-11253 (2896 bytes)]
[Frame: 18281, payload: 11254-16549 (5296 bytes)]
[Frame: 18283, payload: 16550-24741 (8192 bytes)]
[Frame: 18285, payload: 24742-26189 (1448 bytes)]
[Frame: 18287, payload: 26190-32933 (6744 bytes)]
[Frame: 18289, payload: 32934-41125 (8192 bytes)]
[Frame: 18291, payload: 41126-42573 (1448 bytes)]
[Frame: 18292, payload: 42574-44021 (1448 bytes)]
[Frame: 18294, payload: 44022-55605 (11584 bytes)]
[Frame: 18296, payload: 55606-64293 (8688 bytes)]
[Frame: 18298, payload: 64294-80221 (15928 bytes)]
[Frame: 18300, payload: 80222-96149 (15928 bytes)]
[Frame: 18302, payload: 96150-116421 (20272 bytes)]
[Frame: 18304, payload: 116422-129360 (12939 bytes)]
[Segment count: 16]
[Reassembled TCP length: 129361]
[Reassembled TCP Data: 485454502f312e3120323036205061727469616c204366e...]
```

Figura 2: Diagrama

## 2.6. Para cada acceso de la captura, ¿cuantos *byte* retorna el *browser* en cada acceso?

El browser retorna 66 bytes en protocolo TCP, en ellos esta contenido el ACK de el TCP anterior.

## 2.7. Para cada acceso, ¿cuantos GET se efectúan en cada caso y por qué?

Para `http://192.168.1.9:3000/register` fueron 15 GET *request*, por que despues del primer GET el server le dice al browser que otras solicitudes son necesarias para cargar la vista y logica de la pagina, en este caso estas van de cargar las vistas en html, el estilo en css, turbolinks y la logica en javascript, como tambien los scaffolds hechos por rails. Cada uno de los archivos requeridos para renderizar la pagina se obtiene de una llamada GET diferente.

Para `http://192.168.1.9:3000/` fueron 16 GET *request*, estos 16 llamadas GETs fueron assets como imagenes, gif y la vista junto sus turbo links.

Al volver a cargar `http://192.168.1.9:3000/` fueron 2 GET *request*, esto es producto que la mayoria de los assets fueron cargados previamente por el browser en el primer GET y al ver de que ya estan en cache y que lo unico que volvio a pedir fue el `favicon.ico`.

Para `http://192.168.1.9:3000/big.txt` fueron 1 GET *request*, esto se debe a que al acceder a un recurso en el servidor (archivo .txt) el cual no es necesario renderizar asi que se envia al cliente directamente.

Para `http://192.168.1.9:3000/comando/ultra/secreto/win98.mp3` fueron 3 GET *request*, debido a que el audio se puede considerar un recurso grande en tamaño, este es separado en tres partes y el browser necesita hacer 3 GET para tenerlo completamente.

Para `http://192.168.1.9:3000/meme` fueron 3 GET *request*, la primera llamada devuelve por parte del servidor el texto plano del html y las otras dos llamadas son para solicitar los assets de los memes en formato jpg.

Para `http://192.168.1.9:3000/power` fueron 1 GET *request*, este request devuelve un texto plano en html por lo tanto no es considerado un recurso muy grande y es por ello que se logra conseguir toda la información en un solo GET

## 2.8. ¿Podría indicar si las imágenes de `http://192.168.1.9/meme` se descargan en forma serial o si esta operación se realiza en paralelo?

, la llamada se hace en forma serial ya que recién cuando se recibe el mensaje OK 200 se realiza la llamada a la siguiente imagen.

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

El metodo utilizado es GET y permite hacer una solicitud al servidor en modo read-only. todos los otros metodos (como POST por ejemplo) envian contenido al servidor y generalmente estan asociados a modificar datos del servidor. esto podria generar un comportamiento no deseado.

### 3. Parte b)

#### 3.1. ¿Cuántos segmentos TCP se transmiten en cada acceso?

Para `http://192.168.1.9:3000/` fueron 8 TCP *request*,

Al volver a cargar `http://192.168.1.9:3000/` fueron solo 5 TCP *request*

Para `http://192.168.1.9:3000/big.txt` fueron 983 tcp *request*, esta gran cantidad se debe a la comunicacion intensa que ocurre entre el cliente y el servidor para transmitir un archivo pesado, la cantidad de llamadas tcp esta directamente relacionada con el tamaño del archivo que se envia o recibe.

Para `http://192.168.1.9:3000/comando/ultra/secreto/win98.mp3` fueron 1958 tcp *request*, este caso es identico al interior la justificacion.

Para `http://192.168.1.9:3000/meme` fueron 2521 tcp *request*, los cuales corresponden a el texto html + las dos imagenes que se llaman posteriormente para incluir en la vista.

Para `http://192.168.1.9:3000/power` fueron 11070 tcp *request*, la mayoría de las conexiones son de Keep-Alive que su unico rol es mantener la conexion entre el servidor y el cliente.

#### 3.2. ¿Cuáles son los rangos de segmentos TCP que corresponden a cada mensaje HTTP?

#### 3.3. ¿Hubo paquetes perdidos, dañados, o duplicados? Indique cuántos hubo de cada caso y cómo los identificó

#### 3.4. Identifique una secuencia de handshake. Indique en qué paquetes se efectúa y los números de secuencia de cada lado.

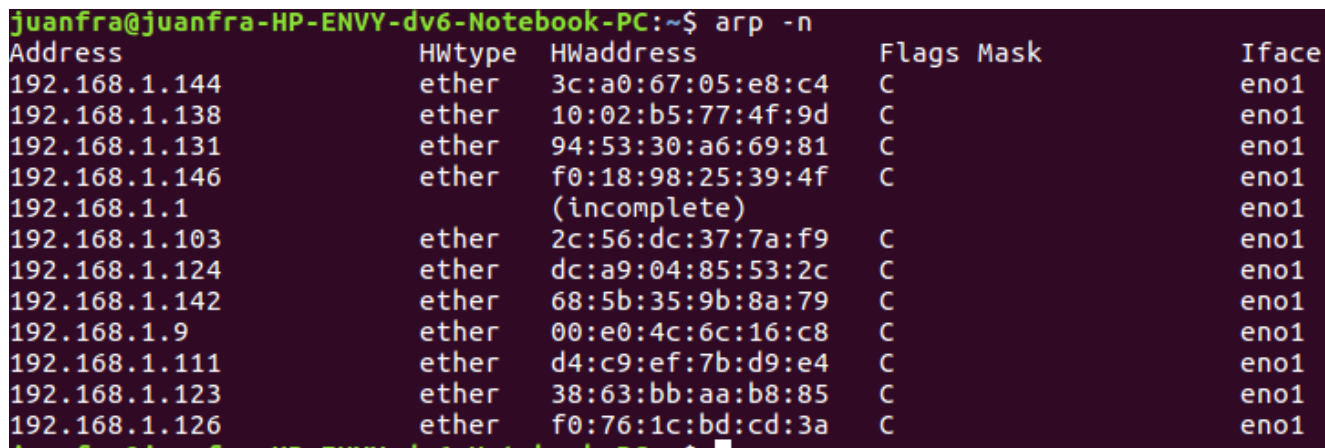
## 4. Parte c)

- 4.1. Construya una lista que incluya los miembros observados en la red. Cada entrada de la lista debe incluir: dirección MAC, dirección IP y fabricante de tarjeta de red.

MAC	IP	Fabricante Tarjeta de Red
dc:a9:04:85:53:2c	192.168.1.124	Apple
94:53:30:a6:69:81	192.168.1.131	HonHaiPr
10:02:b5:77:4f:9d	192.168.1.138	IntelCor
68:5b:35:9b:8a:79	192.168.1.142	Apple
3c:a0:67:05:e8:c4	192.168.1.144	LiteonTe

- 4.2. Explique por que podrían existir direcciones IP sin información dentro de la tabla ARP de su interfaz de red.

Como se puede ver mi tabla ARP contiene una IP que no tiene información relacionada, esto es debido a que la tabla también puede contener la IP del repetidor por el que pasa la conexión para llegar a las otras IP.



```
juanfra@juanfra-HP-ENVY-dv6-Notebook-PC:~$ arp -n
Address          HWtype  HWaddress                     Flags Mask   Iface
192.168.1.144    ether   3c:a0:67:05:e8:c4             C         eno1
192.168.1.138    ether   10:02:b5:77:4f:9d             C         eno1
192.168.1.131    ether   94:53:30:a6:69:81             C         eno1
192.168.1.146    ether   f0:18:98:25:39:4f             C         eno1
192.168.1.1      (incomplete)                  C         eno1
192.168.1.103    ether   2c:56:dc:37:7a:f9             C         eno1
192.168.1.124    ether   dc:a9:04:85:53:2c             C         eno1
192.168.1.142    ether   68:5b:35:9b:8a:79             C         eno1
192.168.1.9      ether   00:e0:4c:6c:16:c8             C         eno1
192.168.1.111    ether   d4:c9:ef:7b:d9:e4             C         eno1
192.168.1.123    ether   38:63:bb:aa:b8:85             C         eno1
192.168.1.126    ether   f0:76:1c:bd:cd:3a             C         eno1
```

Figura 3: ARP

- 4.3. Para una de las direcciones obtenidas luego del ping, identifique los paquetes que se envían por la red y que permitan descubrir la dirección MAC de ese miembro de la red.

Después de saber el MAC address correspondiente se puede ver que se producen dos paquetes del tipo ICMP, el primero es un request de ping y el segundo es la respuesta a este, así continuamente por cada ping solicitado.

30698	1221.1791172...	Apple_85:53:2c	HewlettP_4a:a7:4f	ARP	60 192.168.1.124 is at dc:a9:04:85:53:2c
30699	1221.1791335...	192.168.1.110	192.168.1.124	ICMP	98 Echo (ping) request id=0x177f, seq=1/256, ttl=64 (reply i...
30700	1221.1841534...	fe80::1867:6dc9:6ec...	ff02::fb	MDNS	377 Standard query response 0x0000 TXT TXT, cache flush PTR Ma...
30701	1221.1889310...	192.168.1.124	224.0.0.251	MDNS	357 Standard query response 0x0000 TXT TXT, cache flush PTR Ma...
30702	1221.1889690...	192.168.1.124	192.168.1.110	ICMP	98 Echo (ping) reply id=0x177f, seq=1/256, ttl=64 (request...

Figura 4: ARP