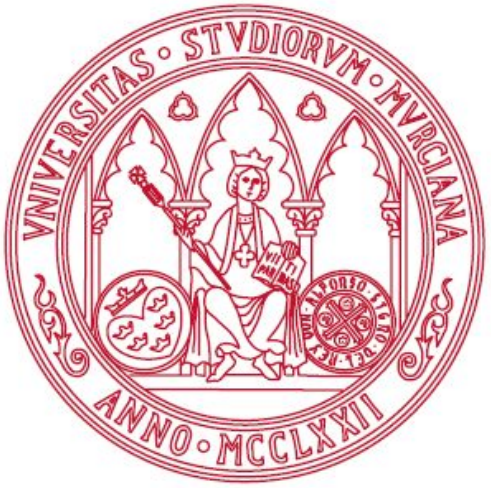


Universidad de Murcia

Grado en Ingeniería Informática
Curso 2020/2021

Arquitectura de Redes Avanzadas

Práctica 2: Balanceo de Carga y Alta Disponibilidad

Profesor: Rafael Marín López

Ignacio Pagán Molera - 49275872M - ignacio.paganm@um.es
Jaime Ortiz Aragón - 49196689B - jaime.ortiza@um.es

Escenario(1 pto)

1.Añada un cliente, un balanceador de carga y dos servidores más, al escenario que se entrega con el enunciado de prácticas.

Para extender el escenario de prácticas y adaptarlo, hemos decidido que la subred de los clientes sea la **178.9.2.0/24** y la de los servidores la **187.2.9.0/24** a partir de las indicaciones presentadas en el boletín de prácticas. Para la subred de los clientes (17X.Y.Z.0), los valores X e Y son las dos últimas cifras del DNI de Jaime y la Z se corresponde con la última cifra del DNI de Ignacio. Para los servidores lo contrario, la X y la Y se corresponden con el DNI de Ignacio y la Z con la última cifra del DNI de Jaime.

Siguiendo esto, el fichero *docker-compose.yml* queda como se muestra a continuación.

```
version: '3'
services:
  lb1:
    build: base/haproxy
    volumes:
      - ./base/haproxy:/tmp
    hostname: lb1
    cap_add:
      - SYS_ADMIN
      - NET_ADMIN
    networks:
        client_net:
            ipv4_address: 178.9.2.200
        server_net:
            ipv4_address: 187.2.9.100

  lb2:
    build: base/haproxy
    volumes:
      - ./base/haproxy:/tmp
    hostname: lb2
    cap_add:
      - SYS_ADMIN
      - NET_ADMIN
    networks:
        client_net:
            ipv4_address: 178.9.2.210
        server_net:
            ipv4_address: 187.2.9.110

  h1:
    build: base/cliente
    hostname: h1
    cap_add:
      - SYS_ADMIN
      - NET_ADMIN
    depends_on:
      - "lb1"
    networks:
        client_net:
            ipv4_address: 178.9.2.204

  h2:
    build: base/cliente
    hostname: h2
    cap_add:
      - SYS_ADMIN
      - NET_ADMIN
    depends_on:
      - "lb1"
    networks:
        client_net:
```

Modificamos las direcciones físicas de los balanceadores. El resultado de este cambio se va a explicar en los posteriores ejercicios en los que mostramos cómo quedan las subredes y los ficheros *routes.sh* tanto de cliente como de servidor.

En un principio hemos puesto que tanto clientes como servidores dependen del primer balanceador, ya que es este el que actúa de maestro y tiene la prioridad más alta. Estos parámetros se configuran en los ficheros de keepalive, pero no son trascendentes para esta primera prueba de conectividad. Otro parámetro que también cambiaremos más adelante para la configuración es montar ambos balanceadores sobre distintos volúmenes, teniendo sus ficheros de haproxy y de keepalived separados. Estos se describirán más adelante.

```
s1:
  hostname: s1
  build: base/servidor
  ports:
        - "8080:80"
  cap_add:
    - SYS_ADMIN
    - NET_ADMIN
  depends_on:
    - "lb1"
  networks:
    server_net:
      ipv4_address: 187.2.9.101

s2:
  hostname: s2
  build: base/servidor
  ports:
        - "8081:80"
  cap_add:
    - SYS_ADMIN
    - NET_ADMIN
  depends_on:
    - "lb1"
  networks:
      server_net:
          ipv4_address: 187.2.9.102

s3:
  hostname: s3
  build: base/servidor
  ports:
        - "8082:80"
  cap_add:
    - SYS_ADMIN
    - NET_ADMIN
  depends_on:
    - "lb1"
  networks:
      server_net:
          ipv4_address: 187.2.9.103
```

Como podemos observar, los puertos de cada servidor son distintos. De hecho el puerto que no puede ser igual es el que abren para escuchar las peticiones del cliente, por eso como vemos en la captura de la derecha, hemos asignado un puerto distinto a cada servidor. En cuanto a las IPs de clientes y servidores, se han configurado conforme al escenario descrito

en el boletín y adaptándose conforme a la descripción que se ha dado al principio del presente ejercicio.

2.Configure las diferentes subredes, IPs, etc. en el nuevo escenario siguiendo la convención del enunciado de prácticas.

En el fichero *docker-compose.yml* hemos definido las dos subredes en las que van a estar clientes, servidores y las IPs físicas y virtuales de los balanceadores. La subred de los clientes será la **178.9.2./24** y la de los servidores **187.2.9.0/24,** de acuerdo con la asignación de XYZ que hemos mencionado en el ejercicio anterior.

```
networks:
  client_net:
    driver: bridge
    ipam:
     config:
       - subnet: 178.9.2.0/24
    driver_opts:
     com.docker.network.bridge.name: client_net
  server_net:
    driver: bridge
    ipam:
     config:
       - subnet: 187.2.9.0/24
    driver_opts:
```

Otro cambio importante a considerar es el de los ficheros *routes.sh* tanto de cliente como de servidor. En ellos se indica cómo llegar a la subred de los otros dispositivos. En nuestro caso, llegarán por las IPs físicas del primer balanceador, como mostramos a continuación.

```
ip route add 178.9.2.0/24 via 187.2.9.100
```

```
ip route add 187.2.9.0/24 via 178.9.2.200
tail -f /dev/null
```

Esto nos sirve para realizar la primera prueba de conectividad entre clientes y servidores, sin tener en cuenta la operativa de los balanceadores de carga. Cuando se realice una petición desde los clientes hacia cualquier equipo que se encuentre en la red **187.2.9.0/24**, la salida hacia esa subred será la dirección **178.9.2.200**, que es la primera IP física del balanceador 1.

Para los paquetes que envíen los servidores hacia los equipos que se encuentren en la red **178.9.2.0/24**, la salida hacia esa subred será la dirección **187.2.9.100**, que es la segunda IP física del balanceador 1.

Cuando posteriormente se tengan en cuenta, deberemos cambiar estos valores por las **direcciones virtuales** y volver a crear las tablas de rutas.

3.Compruebe la conectividad y el funcionamiento del escenario antes de poner en marcha las herramientas de balanceo de carga y alta disponibilidad.

La primera prueba de conectividad que hemos realizado consiste en un ping desde el cliente 1, que se construye en el contenedor **enunciado_h1_1** hacia el servidor 1, **enunciado_s1_1**. Podemos ver tanto en la terminal como en wireshark que los paquetes ICMP se intercambian correctamente, tanto peticiones como respuestas.

```
root@h1:/# ping 187.2.9.101
PING 187.2.9.101 (187.2.9.101) 56(84) bytes of data.
64 bytes from 187.2.9.101: icmp_seq=1 ttl=63 time=0.510 ms
64 bytes from 187.2.9.101: icmp_seq=2 ttl=63 time=0.192 ms
64 bytes from 187.2.9.101: icmp_seq=3 ttl=63 time=0.186 ms
64 bytes from 187.2.9.101: icmp_seq=4 ttl=63 time=0.187 ms
64 bytes from 187.2.9.101: icmp_seq=5 ttl=63 time=0.188 ms
```

1	0.000000000	178.9.2.204	187.2.9.101	ICMP	98	Echo (ping) request id=0x0013, seq=1/256, ttl=64 (reply in 2)
2	0.000146939	187.2.9.101	178.9.2.204	ICMP	98	Echo (ping) reply id=0x0013, seq=1/256, ttl=63 (request in…
3	1.022187164	178.9.2.204	187.2.9.101	ICMP	98	Echo (ping) request id=0x0013, seq=2/512, ttl=64 (reply in 4)
4	1.022318049	187.2.9.101	178.9.2.204	ICMP	98	Echo (ping) reply id=0x0013, seq=2/512, ttl=63 (request in…
5	2.046310048	178.9.2.204	187.2.9.101	ICMP	98	Echo (ping) request id=0x0013, seq=3/768, ttl=64 (reply in 6)
6	2.046439452	187.2.9.101	178.9.2.204	ICMP	98	Echo (ping) reply id=0x0013, seq=3/768, ttl=63 (request in…

A continuación, hemos realizado otra prueba, que consiste en el ping que realiza el servidor 1, esta vez hacia el cliente 2, que se encuentra en **enunciado_h2_1**. Podemos comprobar que efectivamente hay conexión.

```
root@s1:/# ping 178.9.2.205
PING 178.9.2.205 (178.9.2.205) 56(84) bytes of data.
64 bytes from 178.9.2.205: icmp_seq=1 ttl=63 time=0.196 ms
64 bytes from 178.9.2.205: icmp_seq=2 ttl=63 time=0.212 ms
64 bytes from 178.9.2.205: icmp_seq=3 ttl=63 time=0.207 ms
64 bytes from 178.9.2.205: icmp_seq=4 ttl=63 time=0.175 ms
64 bytes from 178.9.2.205: icmp_seq=5 ttl=63 time=0.186 ms
```

1	0.000000000	187.2.9.101	178.9.2.205	ICMP	98	Echo (ping) request id=0x0051, seq=1/256, ttl=64 (reply in 2)
2	0.000133557	178.9.2.205	187.2.9.101	ICMP	98	Echo (ping) reply id=0x0051, seq=1/256, ttl=63 (request in…
3	1.032179198	187.2.9.101	178.9.2.205	ICMP	98	Echo (ping) request id=0x0051, seq=2/512, ttl=64 (reply in 4)
4	1.032300191	178.9.2.205	187.2.9.101	ICMP	98	Echo (ping) reply id=0x0051, seq=2/512, ttl=63 (request in…
5	2.056123106	187.2.9.101	178.9.2.205	ICMP	98	Echo (ping) request id=0x0051, seq=3/768, ttl=64 (reply in 6)
6	2.056258781	178.9.2.205	187.2.9.101	ICMP	98	Echo (ping) reply id=0x0051, seq=3/768, ttl=63 (request in…
7	3.080130703	187.2.9.101	178.9.2.205	ICMP	98	Echo (ping) request id=0x0051, seq=4/1024, ttl=64 (reply in …
8	3.080251869	178.9.2.205	187.2.9.101	ICMP	98	Echo (ping) reply id=0x0051, seq=4/1024, ttl=63 (request i…

Balanceo de Carga(3 ptos)

4.Explique los parámetros principales del fichero haproxy.cfg proporcionado (0,5 ptos)

HAProxy es un software simple pero potente que nos permite probar el balanceo de cargas y la alta disponibilidad de un sistema mediante peticiones TCP. El fichero de configuración que vamos a describir se encuentra en el directorio *haproxy*. Lo primero que encontramos son los parámetros globales. Estos suelen ser estáticos y en nuestro caso encontramos dos tipos: **log** y **stats**.

La primera cláusula tiene la forma **log <address><facility>.** Estos son quienes reciben las peticiones de inicio y de salida del sistema. En nuestro caso encontramos dos logs, con dirección **localhost**. El segundo parámetro es **local0** en el primero y **local1** en el segundo. Estos son dos de los estándares asignados para este parámetro.

A continuación, encontramos las secciones de los proxies. La primera de ellas se corresponde con la cláusula **default.** En ella encontramos ciertos parámetros que funcionarán por defecto en el resto de secciones conforme a cómo se han declarado en esta. De esta forma encontramos parámetros como el tipo de peticiones (http), los timeouts para cada dispositivo, etc.

El siguiente apartado consiste en los equipos que van a funcionar de **frontend**. Estos equipos recibirán las peticiones de los clientes y serán los encargados de reenviarlas a los servidores para tratar las cookies. En este caso, podemos observar que quienes actuarán como tales serán los que se definan en **web_servers**, ya que se indican en la cláusula default_backend. En la propia sección **backend** se especifica el tipo de balanceo que realizarán los servidores. En nuestro caso utilizan el algoritmo **roundrobin**, cuyo funcionamiento describiremos en el ejercicio 6.

Encontramos algunos parámetros como la cookie que usará la aplicación. La opción que acompaña a la cookie es **insert**, por lo que para conseguir persistencia, la cookie se insertará si el cliente no la tenía ya registrada. Este modo de cookie activo para conseguirla persistencia se hace totalmente transparente entre cliente y servidor con el uso de la opción **indirect**. En este caso, cuando el cliente hace una petición al servidor, no la hace con la lista de todas las cookies que ya tiene. Dado que esto supone un mecanismo de persistencia a nivel de aplicación, se hará uso del **delayed binding**.

El principal parámetro que hemos modificado en este fichero se corresponde con qué servidores actuarán de backend y dónde se encuentran. Con el escenario inicial sólo estaba el único servidor activo : **server webserver0 187.2.9.101:80 cookie webserver0 check**, pero tras hacer la ampliación hemos puesto dos más: **server webserver1 187.2.9.102:80 cookie webserver1 check** y **server webserver2 187.2.9.103:80 cookie webserver2 check**. El formato incluye el nombre del servidor, la IP donde se encuentra (en nuestro caso en cada uno de los contenedores), el puerto en el que escuchan (80 al ser peticiones HTTP) y el nombre de la cookie que almacenan.

El último proxy que se define es el de las estadísticas, con el cual comprobaremos el correcto funcionamiento de los algoritmos, el cual estará corriendo en el puerto 9000.

5.Pruebe y estudie con una herramienta analizadora de paquetes de red (p.ej. Wireshark) el comportamiento del sistema cuando un cliente accede al servicio web a través de la IP física de cada uno de los balanceadores. (1 pto)

Esta prueba consiste en hacer una solicitud mediante *lynx* por medio de comandos. Es un browser muy sencillo al que le indicamos simplemente la IP a la que queremos acceder y lleva a cabo la solicitud. En nuestro caso, hemos realizado una petición desde cliente 1. Si realizamos esta petición sin tener activado el demonio **haproxy**, se obtiene el mensaje de *Unable to connect*, por lo que no se puede procesar nuestra petición. Cuando lo activamos mediante el comando **haproxy -D -f /tmp/haproxy.cfg**, haremos que las conexiones entrantes de los clientes sean enviadas a los servidores que las tratarán.

En este primer ejemplo, realizamos desde el cliente 1, una consulta a la dirección física del primer balanceador **178.9.2.200**. Esta consulta será balanceada al servidor que toque en función de los paquetes por la IP física del primer balanceador que lo conecta con la subred de los servidores. **187.2.9.100** como podemos observar en Wireshark.

17	0.667191505	187.2.9.100	187.2.9.101	HTTP	85	HEAD / HTTP/1.0
19	0.667375912	187.2.9.101	187.2.9.100	HTTP	340	HTTP/1.1 200 OK
29	1.332034206	187.2.9.100	187.2.9.102	HTTP	85	HEAD / HTTP/1.0
31	1.332264991	187.2.9.102	187.2.9.100	HTTP	340	HTTP/1.1 200 OK
40	2.001227954	187.2.9.100	187.2.9.103	HTTP	85	HEAD / HTTP/1.0
42	2.001538795	187.2.9.103	187.2.9.100	HTTP	340	HTTP/1.1 200 OK

Capturando el tráfico en la red de los servidores, podemos comprobar que todos los paquetes que salen hacia los servidores lo hacen por la IP física .100. El hecho de que se vea un intercambio tan grande, hay por medio trazas de conexión TCP, se debe al mencionado anteriormente **delayed binding**. Dado que estamos haciendo uso de persistencia a nivel de aplicación, necesitamos un método para el SLB capaz de mirar datos a este nivel 5. Para poder mirar en este nivel, necesitamos tener establecida una conexión con el servidor, de aquí que se produzca un intercambio TCP SYN como mostramos a continuación, para cada servidor.

14	0.667133212	187.2.9.100	187.2.9.101	TCP	74	56136 → 80 [SYN] Seq=
15	0.667166706	187.2.9.101	187.2.9.100	TCP	74	80 → 56136 [SYN, ACK]
16	0.667178043	187.2.9.100	187.2.9.101	TCP	66	56136 → 80 [ACK] Seq=
17	0.667191505	187.2.9.100	187.2.9.101	HTTP	85	HEAD / HTTP/1.0
18	0.667200669	187.2.9.101	187.2.9.100	TCP	66	80 → 56136 [ACK] Seq=
19	0.667375912	187.2.9.101	187.2.9.100	HTTP	340	HTTP/1.1 200 OK
20	0.667384357	187.2.9.100	187.2.9.101	TCP	66	56136 → 80 [ACK] Seq=
21	0.667405047	187.2.9.100	187.2.9.101	TCP	66	56136 → 80 [FIN, ACK]
22	0.667420789	187.2.9.101	187.2.9.100	TCP	66	80 → 56136 [FIN, ACK]
23	0.667424294	187.2.9.100	187.2.9.101	TCP	66	56136 → 80 [RST, ACK]
24	0.667428599	187.2.9.100	187.2.9.101	TCP	54	56136 → 80 [RST] Seq=

En esta captura sólo mostramos la segunda parte del proceso, es decir, entre balanceador y servidor, pero este proceso lo inicia realmente el cliente, que es quien realiza la petición de los datos.

6.Pruebe y estudie con el comportamiento del sistema cuando el algoritmo de balanceo de carga es roundrobin.(1 pto)

El algoritmo roundrobin trata las peticiones en función del número de servidores, sin tener en cuenta las particularidades de cada uno. Todos los servidores para el algoritmo roundrobin básico tienen el mismo peso y se les balancea la carga de forma cíclica. Realmente lo que balancea este algoritmo son las peticiones, pero no sabríamos si la carga está realmente bien balanceada. En nuestro sistema en el cual siempre se van a realizar las mismas peticiones siempre, es un algoritmo que funciona bastante bien.

En la siguiente captura podemos observar cómo se va balanceando cada petición a uno de nuestros servidores empezando por el **187.2.9.101**, después el **187.2.9.102** y finalmente el **187.2.9.103**. Cuando se cumple un ciclo, vuelve a empezar de la misma forma, enviando una petición al primer servidor.

```
web_servers

Queue Session rate Sessions Bytes Denied Errors Warnings Ser
Cur Max Limit Cur Max Limit Cur Max Limit Total LbTot Last I
webserver0 0 0 - 0 1 0 1 - 1
   Cum. sessions:     1
Cum. HTTP responses:  1
- HTTP 1xx responses: 0 (0%)
- HTTP 2xx responses: 1 (100%)
- HTTP 3xx responses: 0 (0%)
- HTTP 4xx responses: 0 (0%)
- HTTP 5xx responses: 0 (0%)
 - other responses:   0 (0%)
Avg over last 1024 success. conn.
    - Queue time:     0 ms
   - Connect time:    0 ms
  - Response time:    1 ms
    - Total time:     1 ms
1 1m15s 257 3440 0 0 0
Connection resets during transfers: 0 client, 0 server
0 0 1m23s UP L7OK/200 in 0ms
Layer7 check passed: OK
1 Y - 0
Failed Health Checks
0 0s -
webserver1 0 0 - 0 1 0 1 - 1
   Cum. sessions:     1
Cum. HTTP responses:  1
- HTTP 1xx responses: 0 (0%)
- HTTP 2xx responses: 1 (100%)
- HTTP 3xx responses: 0 (0%)
- HTTP 4xx responses: 0 (0%)
- HTTP 5xx responses: 0 (0%)
 - other responses:   0 (0%)
Avg over last 1024 success. conn.
    - Queue time:     0 ms
   - Connect time:    0 ms
  - Response time:    1 ms
    - Total time:     1 ms
1 54s 257 3440 0 0 0
Connection resets during transfers: 0 client, 0 server
0 0 1m23s UP L7OK/200 in 0ms
Layer7 check passed: OK
1 Y - 0
Failed Health Checks
0 0s -
webserver2 0 0 - 0 1 0 1 - 1
   Cum. sessions:     1
Cum. HTTP responses:  1
```

En este caso hemos realizado 3 peticiones y observamos cómo se ha balanceado cada petición a un servidor.

A continuación, nos disponemos a realizar otra nueva petición y observaremos cómo el número de peticiones al primer servidor aumenta.

Cabe destacar, que para realizar esta prueba no debemos guardar las cookies porque esto causaría que todas las peticiones fueran para el servidor que la ha almacenado.

```
webserver0 0 0 - 0 1 0 1 - 2
   Cum. sessions:     2
Cum. HTTP responses:  2
- HTTP 1xx responses: 0 (0%)
- HTTP 2xx responses: 2 (100%)
- HTTP 3xx responses: 0 (0%)
- HTTP 4xx responses: 0 (0%)
- HTTP 5xx responses: 0 (0%)
 - other responses:   0 (0%)
Avg over last 1024 success. conn.
    - Queue time:     0 ms
   - Connect time:    0 ms
  - Response time:    0 ms
    - Total time:     0 ms
```

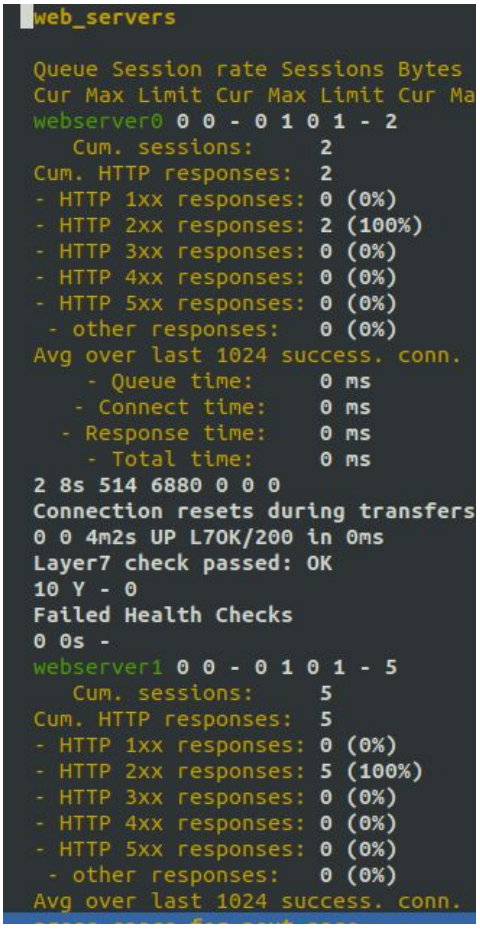
7.Escoja un algoritmo de balanceo diferente y comente sobre el escenario en qué se diferencia. El grupo puede escoger el algoritmo que desee.(0,5 ptos)

En cuanto al algoritmo que hemos elegido, ha sido el roundrobin pero con pesos. El motivo es que para nuestro escenario algunos algoritmos como **leastconnnections** o **first** iban a funcionar como el roundrobin clásico ya que las conexiones se iban a ir balanceando en función de su índice, ya que en nuestro caso al ser peticiones cortas y tener sólo 2 clientes el funcionamiento iba a ser cíclico.

Para configurar esto, simplemente tenemos que asignar los pesos en el fichero de configuración del haproxy.

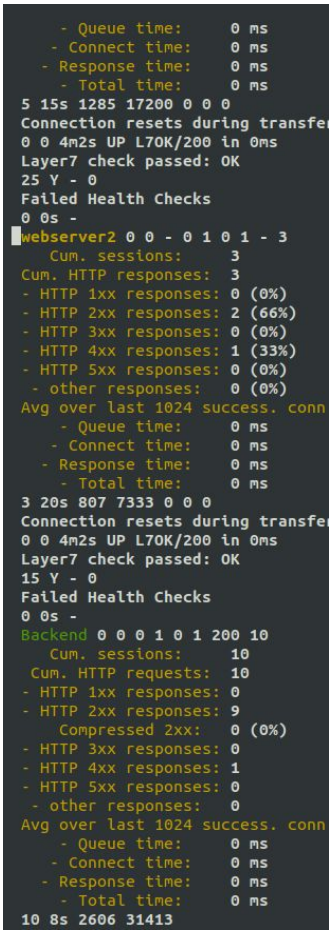
```
server webserver0 187.2.9.101:80 cookie webserver0 check weight 10
server webserver1 187.2.9.102:80 cookie webserver1 check weight 25
server webserver2 187.2.9.103:80 cookie webserver2 check weight 15
```

Para mostrar las estadísticas de los balanceos hemos realizado una petición a la VIP de los SLBs pero al puerto 9000, que es en el que definimos en el fichero *haproxy*, que es donde se iban a almacenar las estadísticas. Podemos ver que de 10 peticiones : 2 han ido a **webserver0**, 5 a **webserver1** y 3 a **webserver2**.



```
web_servers

Queue Session rate Sessions Bytes
Cur Max Limit Cur Max Limit Cur Ma
webserver0 0 0 - 0 1 0 1 - 2
   Cum. sessions:     2
Cum. HTTP responses:  2
- HTTP 1xx responses: 0 (0%)
- HTTP 2xx responses: 2 (100%)
- HTTP 3xx responses: 0 (0%)
- HTTP 4xx responses: 0 (0%)
- HTTP 5xx responses: 0 (0%)
 - other responses:   0 (0%)
Avg over last 1024 success. conn.
    - Queue time:     0 ms
   - Connect time:    0 ms
  - Response time:    0 ms
    - Total time:     0 ms
2 8s 514 6880 0 0 0
Connection resets during transfers
0 0 4m2s UP L7OK/200 in 0ms
Layer7 check passed: OK
10 Y - 0
Failed Health Checks
0 0s -
webserver1 0 0 - 0 1 0 1 - 5
   Cum. sessions:     5
Cum. HTTP responses:  5
- HTTP 1xx responses: 0 (0%)
- HTTP 2xx responses: 5 (100%)
- HTTP 3xx responses: 0 (0%)
- HTTP 4xx responses: 0 (0%)
- HTTP 5xx responses: 0 (0%)
 - other responses:   0 (0%)
Avg over last 1024 success. conn.
```



```
    - Queue time:     0 ms
   - Connect time:    0 ms
  - Response time:    0 ms
    - Total time:     0 ms
5 15s 1285 17200 0 0 0
Connection resets during transfe
0 0 4m2s UP L7OK/200 in 0ms
Layer7 check passed: OK
25 Y - 0
Failed Health Checks
0 0s -
webserver2 0 0 - 0 1 0 1 - 3
   Cum. sessions:     3
Cum. HTTP responses:  3
- HTTP 1xx responses: 0 (0%)
- HTTP 2xx responses: 2 (66%)
- HTTP 3xx responses: 0 (0%)
- HTTP 4xx responses: 1 (33%)
- HTTP 5xx responses: 0 (0%)
 - other responses:   0 (0%)
Avg over last 1024 success. conn
    - Queue time:     0 ms
   - Connect time:    0 ms
  - Response time:    0 ms
    - Total time:     0 ms
3 20s 807 7333 0 0 0
Connection resets during transfe
0 0 4m2s UP L7OK/200 in 0ms
Layer7 check passed: OK
15 Y - 0
Failed Health Checks
0 0s -
Backend 0 0 0 1 0 1 200 10
   Cum. sessions:     10
 Cum. HTTP requests:  10
- HTTP 1xx responses: 0
- HTTP 2xx responses: 9
    Compressed 2xx:   0 (0%)
- HTTP 3xx responses: 0
- HTTP 4xx responses: 1
- HTTP 5xx responses: 0
 - other responses:   0
Avg over last 1024 success. conn
    - Queue time:     0 ms
   - Connect time:    0 ms
  - Response time:    0 ms
    - Total time:     0 ms
10 8s 2606 31413
```

Alta Disponibilidad(3 ptos)

8.Explique los parámetros principales del fichero keepalived-lb1.conf y keepalived-lb2.conf proporcionados(0,75 ptos)

Estos ficheros configuran el comportamiento de los balanceadores de carga. En nuestro caso, al haber dos SLB, cada uno de los ficheros de configuración estará en el volumen haproxy correspondiente.

La primera sección que encontramos hace referencia a los parámetros globales . En nuestro caso no prestamos mucha atención dado que estas definiciones son referentes al envío de notificaciones mediante correo electrónico, y esto no está contemplado en nuestro despliegue. La primera parte realmente relevante para nosotros hace referencia a lo que se conoce como grupo de sincronización. Al conectar clientes y servidores ambas interfaces deben trabajar en consonancia para que las conexiones lleguen a los servidores que están actuando de backend.

Estas dos instancias que colaboran en el grupo son **RH_EXT** y **RH_INT**. La primera hace referencia a la interfaz eth0 que conecta a los balanceadores con los clientes y la segunda instancia hace referencia a la interfaz que los conecta con los servidores. Hasta este punto, tanto el fichero *keepalived_lb1* como *keepalived_lb2* son idénticos.

Los cambios vienen en los parámetros de estas instancias , ya que el protocolo que se utiliza para balancear la carga es **Virtual Router Redundancy Protocol (VRRP).** La forma en la que este funciona consiste en que ambos routers (balanceadores), adquieren roles distintos. El primer SLB será el maestro mientras que el segundo será el esclavo. Esto se indica en las instancias con los parámetros **MASTER** y **BACKUP**.

A continuación se indica la interfaz por la que trabajan y el ID de la dirección virtual que tratan. El significado de esta dirección virtual lo explicaremos más adelante cuando comprobemos cómo funciona el sistema.

Otro parámetro fundamental es la prioridad con la que se anuncian los routers. El router maestro debe tener la prioridad más alta para que siempre que esté activo sea quien trate las peticiones. En nuestro caso, hemos decidido mantener en este primer balanceador la prioridad que se nos establece, que es 100. Para el caso del segundo SLB, hemos de tener en cuenta que según la documentación, la diferencia de prioridades para que realmente esté actuando de backup y no están teniendo el mismo rol, debe ser mayor o igual que 50. Por este motivo, hemos establecido la prioridad del segundo SLB a 49.

El siguiente parámetro que nos encontramos es **advert_int**, que indica la frecuencia con la que se envían anuncios por parte de los routers. En cuanto a la autenticación, se utiliza un método de simple plano, indicado por la opción **PASS** y se establece la contraseña pass123 para ambas instancias.

Otro parámetro fundamental que hemos modificado es la VIP. Esta dirección es la que representa a los balanceadores indistintamente de cuál esté activo. Siguiendo con lo

propuesto en el enunciado, hemos establecido en la interfaz cliente la VIP **178.9.2.220** y en la interfaz de los servidores la **187.2.9.120**. Estas direcciones también deben ser cambiadas en los ficheros de las tablas de rutas de clientes y servidores para que ahora las peticiones no se realicen a una dirección fija física, sino a esta dirección virtual. En función del router que esté activo con más prioridad en ese momento, se atenderá a la petición por un balanceador o el otro.

9.Pruebe y estudie el comportamiento del sistema cuando la consulta http desde un cliente se realiza estando el sistema de alta disponibilidad (keepalived) en marcha. Es decir, a la IP virtual que keepalived gestiona. Analice cómo funciona la sincronización del servicio de alta disponibilidad a través del protocolo VRRP.(1,25ptos)

Para este ejercicio tanto los demonios del **haproxy** como los de **keepalive** deben estar activos. El demonio del haproxy ya lo hemos mencionado anteriormente y su utilidad es la misma que para atender a las peticiones que se realizaban a las IPs físicas de los routers.

En este caso, al contar con IPs virtuales, que son a las que van a llegar las peticiones de los clientes como hemos indicado en los ficheros *routes.sh* necesitamos que los balanceadores estén activos. Para lanzarlos utilizaremos en el caso del primer SLB **keepalived -l -D -f /tmp/keepalived-lb1.conf** y **keepalived -l -D -f /tmp/keepalived-lb2.conf** para el caso del segundo balanceador. Una vez lanzados, los podemos ver en funcionamiento con el comando **ps aux**

```
root@lb1:/# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0   4632   812 ?        Ss   22:42   0:00 /bin/sh -c tail -f
root           6  0.0  0.0   4572   812 ?        S    22:42   0:00 tail -f /dev/null
root           7  0.0  0.0  18508  3388 pts/0    Ss   22:42   0:00 /bin/bash
root          25  0.0  0.0  91812  2920 ?        Ss   22:44   0:00 keepalived -l -D -f
root          26  0.0  0.0  93932  5252 ?        S    22:44   0:00 keepalived -l -D -f
root          27  0.0  0.0  94076  5372 ?        S    22:44   0:00 keepalived -l -D -f
root          28  0.0  0.0  34408  2768 pts/0    R+   22:44   0:00 ps aux
```

Como hemos mencionado anteriormente, en este caso las peticiones HTTP no se realizan a la dirección física de los SLBs sino a la VIP. Esta dirección se presenta como "una caja", que representa a ambos balanceadores, teniendo en cuenta que se va a intentar que el que actúa como maestro sea quien asuma los procesos de poner en contacto las peticiones de frontend con los tratamientos de backend que realizarán los servidores.

Para esta primera prueba sólo lanzamos el servicio en el primer router, por lo que vamos a observar cómo se producen los anuncios periódicos del protocolo VRRP. En este caso, al haber solo un router, cuando lancemos una petición del cliente será este balanceador quien la trate independientemente de si es maestro o esclavo. En nuestro caso el router lanzado es el maestro, por lo que podemos ver los anuncios enviados con prioridad **100**. En concreto se muestra el envío que se hace por la interfaz de los servidores, donde se puede observar la

prioridad. Esta es la misma en la subred de los clientes aunque no se muestre, de acuerdo con lo definido en la instancia **RH_EXT** que definimos en el fichero *keepalive-lb1.conf*.

```
    27 0.526048867   187.2.9.100          224.0.0.18           VRRP          56 Announcement (v2)
    28 0.526088784   187.2.9.100          224.0.0.18           VRRP          56 Announcement (v2)
    29 0.526097507   187.2.9.100          224.0.0.18           VRRP          56 Announcement (v2)
    30 0.526105123   187.2.9.100          224.0.0.18           VRRP          56 Announcement (v2)
    31 0.526112465   187.2.9.100          224.0.0.18           VRRP          56 Announcement (v2)
    32 0.526048867   187.2.9.100          224.0.0.18           VRRP          56 Announcement (v2)
    33 0.526184007   178.9.2.200          224.0.0.18           VRRP          56 Announcement (v2)
    34 0.526197570   178.9.2.200          224.0.0.18           VRRP          56 Announcement (v2)
    35 0.526204976   178.9.2.200          224.0.0.18           VRRP          56 Announcement (v2)
    36 0.526212025   178.9.2.200          224.0.0.18           VRRP          56 Announcement (v2)
    37 0.526184007   178.9.2.200          224.0.0.18           VRRP          56 Announcement (v2)

▸ Frame 27: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface any, id 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 187.2.9.100, Dst: 224.0.0.18
▾ Virtual Router Redundancy Protocol
  ▸ Version 2, Packet type 1 (Advertisement)
    Virtual Rtr ID: 2
    Priority: 100 (Default priority for a backup VRRP router)
    Addr Count: 1
    Auth Type: Simple Text Authentication [RFC 2338] / Reserved [RFC 3768] (1)
    Adver Int: 1
    Checksum: 0x2847 [correct]
    [Checksum Status: Good]
    IP Address: 187.2.9.120
    Authentication String: passw123
```

Cuando lanzamos una petición desde un cliente, el SLB la recibe por la interfaz virtual que lo conecta a la red de los clientes y balancea a un servidor en función del algoritmo de balanceo, como hemos mencionado anteriormente.

```
    25 21.037145285  178.9.2.204          178.9.2.220          TCP             66 59862 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSv
    26 21.037346698  178.9.2.204          178.9.2.220          HTTP           323 GET / HTTP/1.0
    27 21.037462691  178.9.2.220          178.9.2.204          TCP           7306 80 → 59862 [PSH, ACK] Seq=1 Ack=258 Win=65024 Le
    28 21.037483074  178.9.2.204          178.9.2.220          TCP             66 59862 → 80 [ACK] Seq=258 Ack=7241 Win=61312 Len=
    29 21.037490897  178.9.2.220          178.9.2.204          TCP           5851 80 → 59862 [PSH, ACK] Seq=7241 Ack=258 Win=6502
    30 21.037496272  178.9.2.204          178.9.2.220          TCP             66 59862 → 80 [ACK] Seq=258 Ack=13026 Win=57856 Len
    31 21.037538794  178.9.2.220          178.9.2.204          TCP           7607 80 → 59862 [PSH, ACK] Seq=13026 Ack=258 Win=650
    32 21.037547678  178.9.2.204          178.9.2.220          TCP             66 59862 → 80 [ACK] Seq=258 Ack=20567 Win=54016 Len
    33 21.037561582  178.9.2.220          178.9.2.204          HTTP            66 HTTP/1.1 200 OK  (text/html)
    34 21.040182378  178.9.2.204          178.9.2.220          TCP             66 59862 → 80 [FIN, ACK] Seq=258 Ack=20568 Win=641
    35 21.040211525  178.9.2.220          178.9.2.204          TCP             66 80 → 59862 [ACK] Seq=20568 Ack=259 Win=65024 Len
```

*server_net

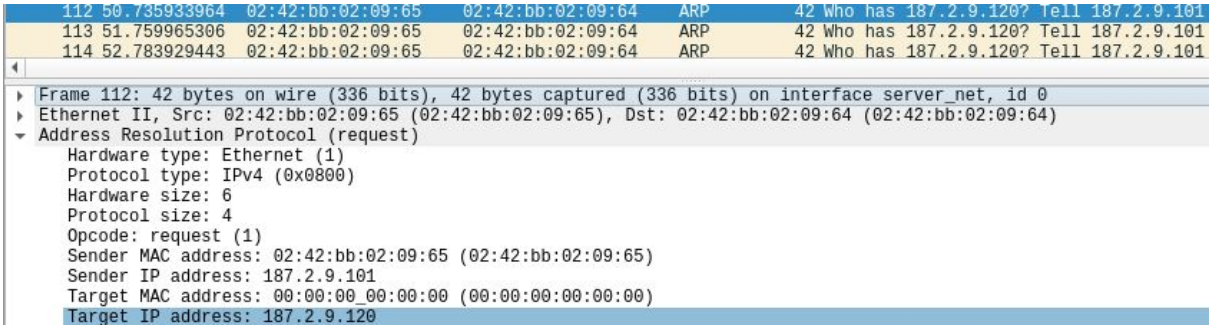
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

```
No.  Time          Source           Destination      Protocol  Length Info
    33 2.189301932  187.2.9.100      187.2.9.101      TCP           74 45058 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
    34 2.189341590  187.2.9.101      187.2.9.100      TCP           74 80 → 45058 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=
    35 2.189354699  187.2.9.100      187.2.9.101      TCP           66 45058 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSv
    36 2.189369827  187.2.9.100      187.2.9.101      HTTP          85 HEAD / HTTP/1.0
    37 2.189380859  187.2.9.101      187.2.9.100      TCP           66 80 → 45058 [ACK] Seq=1 Ack=20 Win=65152 Len=0 TS
    38 2.189575576  187.2.9.101      187.2.9.100      HTTP         340 HTTP/1.1 200 OK
    39 2.189587672  187.2.9.100      187.2.9.101      TCP           66 45058 → 80 [ACK] Seq=20 Ack=275 Win=64128 Len=0
    40 2.189646851  187.2.9.100      187.2.9.101      TCP           66 45058 → 80 [FIN, ACK] Seq=20 Ack=275 Win=64128 L
    41 2.189652478  187.2.9.101      187.2.9.100      TCP           66 80 → 45058 [FIN, ACK] Seq=275 Ack=20 Win=65152 L
    42 2.189660441  187.2.9.100      187.2.9.101      TCP           66 45058 → 80 [ACK] Seq=21 Ack=276 Win=64128 Len=0
    43 2.189660769  187.2.9.101      187.2.9.100      TCP           66 80 → 45058 [ACK] Seq=276 Ack=21 Win=65152 Len=0
```

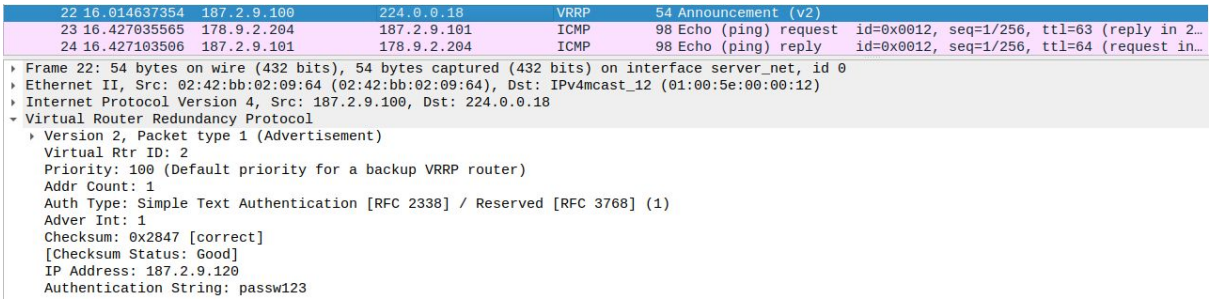
10.Analice qué pasa cuando el demonio de keepalived de lb1 deja de funcionar. ¿Cuál es la reacción a nivel ARP?(0,5 ptos)

Para realizar esta prueba, hemos lanzado el demonio del keepalive del primer balanceador con el comando **keepalived -l -D -f /tmp/keepalived_lb1.conf** y realizado un ping desde el cliente 1 al servidor 1. Hemos hecho un **ps aux** en el SLB para saber el PID del proceso del demonio y mientras se ejecutaba el ping hemos desactivado el demonio con un kill. Pasado un tiempo vemos que uno de los extremos empieza a preguntar por la VIP. Como hemos comentado en el ejercicio anterior, tanto clientes como servidores saben llegar a la otra subred por medio de la VIP de los balanceadores, cuando el demonio de estos deja de estar activo la VIP desaparece y sólo podríamos llegar a la otra subred por medio de las interfaces físicas de ambos SLBs cambiando las tablas de rutas de clientes y servidores. La forma en la que preguntan es en un mensaje ARP conteniendo la IP a la que quieren llegar, con el fin de obtener la MAC.

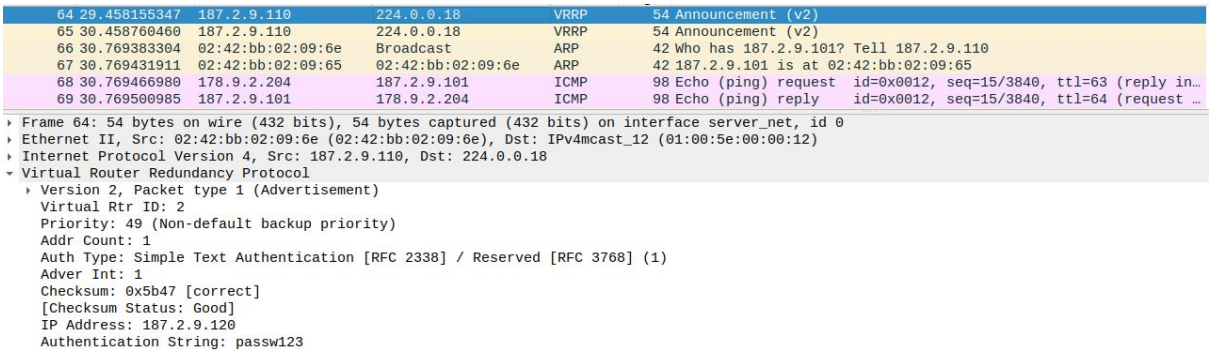


11.Con los dos equipos que actúan de balanceadores están funcionando con la herramienta keepalived, analice el comportamiento del sistema cuando se hace un ping desde un host a la IP de un servicio web (p.ej. S2). Tire uno de los balanceadores (desactivando la interfaz eth0) y compruebe qué ocurre. ¿Sigue funcionando el ping? ¿Por qué?(0,5 ptos)

Para realizar esta prueba, realizamos un ping desde el cliente con la IP **178.9.2.204** hacia el servidor y con IP **187.2.9.2**. Como se muestra, el primer router en enviar anuncios es el balanceador que actúa de maestro, con prioridad 100. Estos anuncios sabemos que son del primer SLB porque se mandan por la IP física .100, que es la referente a la de la red de los servidores. La captura que se muestra es concretamente en la subred de los servidores, pero sería equivalente en la subred de los clientes, cambiando la IP física del balanceador por la .200.



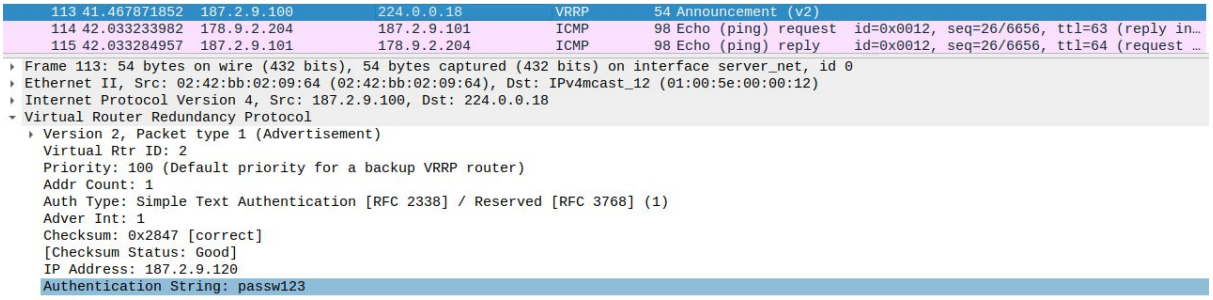
A continuación, realizamos la acción que se pide en el enunciado y comprobamos que el ping sigue funcionando. Como podemos observar, en el momento en el que ejecutamos sobre el balanceador 1 **ifconfig eth0 down**, el segundo router comienza a mandar anuncios VRRP cumpliendo su función de backup router. Estos anuncios son enviados por la IP física .110, lo que nos indica que los envía el segundo router por la subred de los servidores.



El ping sigue funcionando porque la interfaz de salida de ambas subredes son VIPs. Esto quiere decir que no hacen referencia a ninguna dirección física concreta de ningún SLB, sino a "la caja" en la que estos permanecen. Cabe decir que nuestro escenario funciona de esta forma con 2 balanceadores, pero podría ampliarse a cualquier número siempre que los roles y las prioridades estén bien asignadas y seguiríamos viendo el correcto funcionamiento del intercambio de mensajes **ICMP**.

Podemos observar a continuación la importancia de las de cada balanceador, como comentábamos en la descripción del fichero. Por cómo funciona el protocolo VRRP, cuando el SLB maestro cae, el esclavo es quien toma el control del envío de tramas Pero si el balanceador que actúa como maestro consigue volver a estar activo, este debe volver a ser el que realice las tareas. Esto se puede observar en la siguiente captura.

El router maestro comienza a enviar anuncios VRRP con su prioridad, lo que hace que vuelva a asumir su rol en el escenario.


113 41.467871852 187.2.9.100 224.0.0.18 VRRP 54 Announcement (v2)
114 42.033233982 178.9.2.204 187.2.9.101 ICMP 98 Echo (ping) request id=0x0012, seq=26/6656, ttl=63 (reply in…
115 42.033284957 187.2.9.101 178.9.2.204 ICMP 98 Echo (ping) reply id=0x0012, seq=26/6656, ttl=64 (request …
Frame 113: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface server_net, id 0
Ethernet II, Src: 02:42:bb:02:09:64 (02:42:bb:02:09:64), Dst: IPv4mcast_12 (01:00:5e:00:00:12)
Internet Protocol Version 4, Src: 187.2.9.100, Dst: 224.0.0.18
Virtual Router Redundancy Protocol
Version 2, Packet type 1 (Advertisement)
Virtual Rtr ID: 2
Priority: 100 (Default priority for a backup VRRP router)
Addr Count: 1
Auth Type: Simple Text Authentication [RFC 2338] / Reserved [RFC 3768] (1)
Adver Int: 1
Checksum: 0x2847 [correct]
[Checksum Status: Good]
IP Address: 187.2.9.120
Authentication String: passw123