****

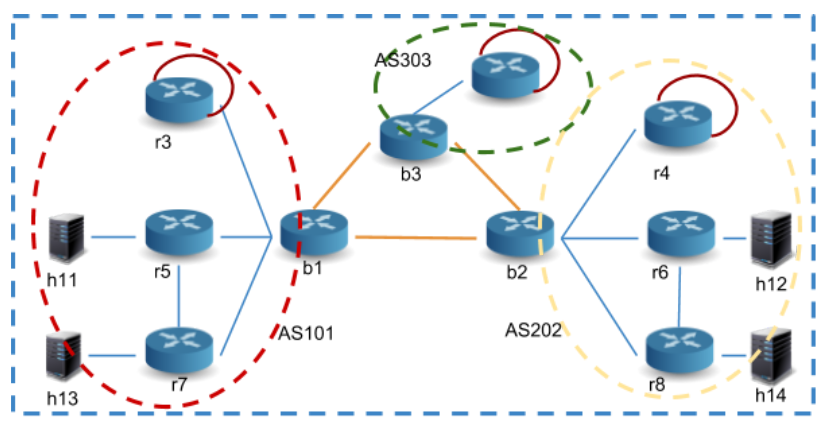
**Redes de Computadoras**

**Práctico 5: Ruteo internet.**

**Alumnos:**

* **Heredia, Marco.**
* **Yepez Hinostroza, Franz.**

### Ejercicio 1: Ruteo Internet.

****

Podemos ver la asignación de IPs en la siguiente tabla:

|  |  |  |
| --- | --- | --- |
| Nodo | Interfaz compartida con | Dirección IP de la red |
| b1 | b2 | 2001:a:1::/64 |
| b1 | b3 | 2001:a:2::/64 |
| b2 | b3 | 2001:a:3::/64 |
| b1 | r3 | 2001:a:aaaa:1::/64 |
| b1 | r5 | 2001:a:aaaa:2::/64 |
| b1 | r7 | 2001:a:aaaa:3::/64 |
| r5 | r7 | 2001:a:aaaa:4::/64 |
| r5 | h11 | 2001:a:aaaa:5::/64 |
| r7 | h13 | 2001:a:aaaa:6::/64 |
| r3 | r3 | 2001:a:aaaa:7::/64 |
| b2 | r4 | 2001:a:bbbb:1::/64 |
| b2 | r6 | 2001:a:bbbb:2::/64 |
| b2 | r8 | 2001:a:bbbb:3::/64 |
| r6 | r8 | 2001:a:bbbb:4::/64 |
| r6 | h12 | 2001:a:bbbb:5::/64 |
| r8 | h14 | 2001:a:bbbb:6::/64 |
| r4 | r4 | 2001:a:bbbb:7::/64 |
| b3 | r9 | 2001:a:cccc:1::/64 |
| r9 | r9 | 2001:a:cccc:2::/64 |

Por empezar, instalamos Docker CE y Docker Compose (herramientas que nos permiten crear y administrar contenedores) y quagga el cual es un software que nos brinda características de enrutamiento, de los cuales para este trabajo se utilizó OSPFv3.

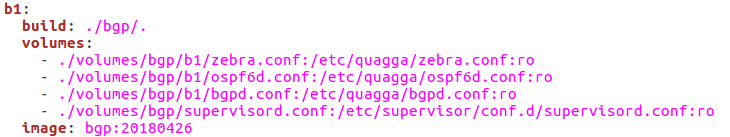
Utilizando los archivos preconfigurados en el repositorio de git:[*https://github.com/maticue/docker\_quagga.git*](https://github.com/maticue/docker_quagga.git)*.*

Configuramos docker para que funcione con IPv6 como indica en el Readme del repositorio, pusimos en marcha la red y luego la fuimos ampliando de acuerdo a lo que pedía la consigna.

5) Analizamos el archivo docker-compose.yml, el cual tiene distintas partes que se analizarán a continuación:

\*Sección *services*: se declaran todos los contenedores que van a crearse (para este caso los routers y hosts). Contiene algunas subsecciones, como **build**, que crea una imagen de dicho contenedor que se va a poder instanciar las veces que sean necesarias. **volumes**, que contiene 3 parámetros: el primero indica la ruta donde se encuentra el archivo de configuración, el segundo indica la ruta del archivo en el cual se guardara la configuracion en el contenedor y el tercero indica los privilegios de lectura/escritura que se tendrán sobre el archivo, los 3 parametros estan separados por ‘:’ . Para todos los casos de este trabajo, sólo se tendrá permisos de lectura solamente.

Esta cantidad de entradas varía entre 3 y 4, dependiendo si es requerido o no bgp.



**image** hace referencia a qué imagen se usará para crear el contenedor. Una imagen contiene todo lo necesario para crear el contenedor particular.

**ports**, sirve para comunicarse entre procesos. Cuando el contenedor se inicia se le asigna un número de puerto, el cual va a escuchar el puerto del proceso zebra (2601). Si el usuario quiere acceder a las configuraciones de zebra, mediante telnet deberá accederse al puerto asignado y docker hará la conexión con el puerto 2601. Lo mismo se puede hacer si se quiere acceder al proceso de ospf para IPv6. Una vez hecha la conexión, se podrá configurar el router casi con los mismos comandos que usan los de CISCO.

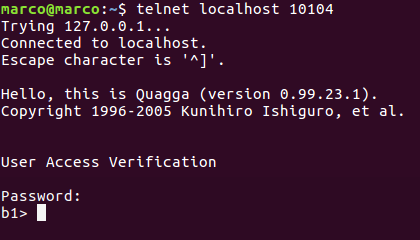
Cabe aclarar que los puertos asignados para el lado del contenedor se pueden repetir entre distintos contenedores, pero no así el puerto del lado del usuario.

En **networks**, se especifican las redes a las que va a pertenecer un determinado contenedor o router, y las ip (versión 4 y 6) de sus interfaces.

\*Sección *networks*: crea las redes aisladas, independientes de los contenedores, disponibles para que estos las usen.

5.1) El servicio bgp escucha en el puerto 2605, y para conectarse (desde b1 teniendo en cuenta la configuración por defecto del repositorio), se hace con el comando:

$ *telnet localhost 10104*

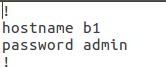
**

El número de puerto al cual hacer conexión telnet se obtiene del archivo docker-compose.yml. Para el caso del router b1 es:

**

6) A cada contenedor se les asocia 3 archivos de configuración, estos son: bgp (bgpd.conf) ospfv3(ospf6d.conf) y zebra (zebra.conf). El primero, configura las opciones de bgp para los routers de borde en los cuales se define la red interna de cada sistema autónomo y la conexión con sistemas autónomos directamente conectados. El segundo, habilita y configura el protocolo ospfv3, estableciendo costos, área, etc, y el tercero, habilita los enrutamientos para IPv4/6.

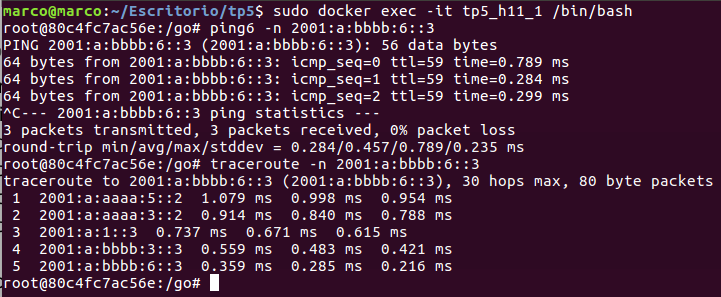
6.1) En los 3 archivos nombrados anteriormente se encuentra el password para conectarse por telnet, de acuerdo a cuál es el servicio al que nos queremos conectar. En todos los casos la contraseña está configurada como admin:



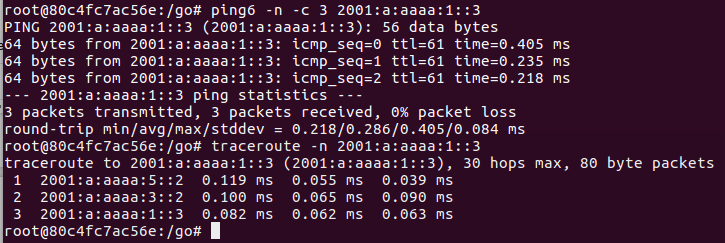
7) Editamos el archivo docker-compose.yml para armar la red propuesta, para la cual tuvimos en cuenta las IPs de cada router según la tabla descrita al comienzo de este informe.

8) Creamos los archivos para cada uno de los contenedores en volumes, y allí configuramos los routers de borde en la carpeta bgp y los routers de cada sistema autónomo y sus hosts en la carpeta ospf.

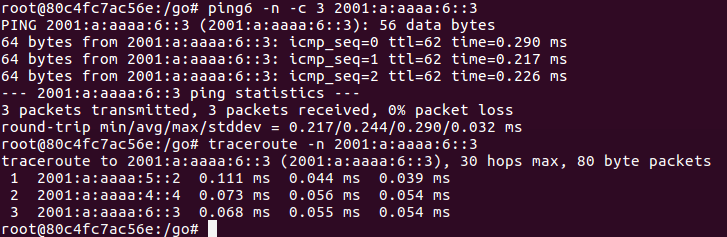
9) Se verifica conectividad y el trayecto que realizan los paquetes desde el host 11 del sistema autónomo 101 a un host del sistema autónomo 202:



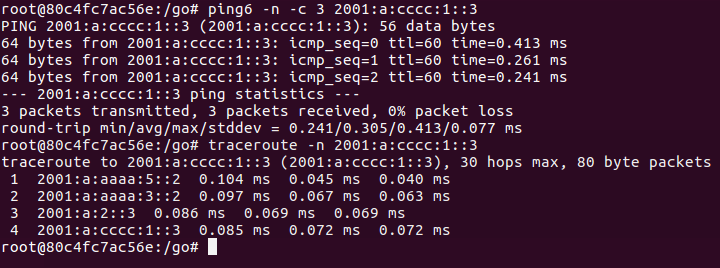
Se verifica conectividad y el trayecto que realizan los paquetes desde el host 11 del sistema autónomo 101 a al router 3 dentro del mismo área:



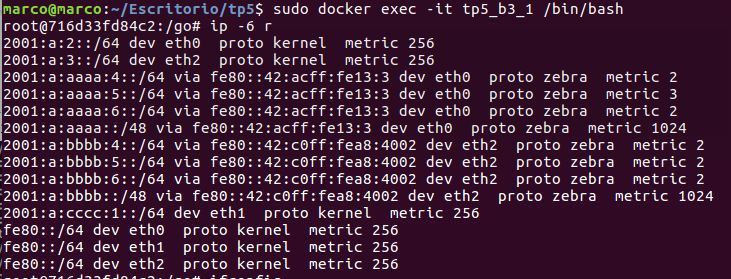
Se verifica conectividad y el trayecto que realizan los paquetes desde el host 11 del sistema autónomo 101 al host 13, dentro del mismo área:



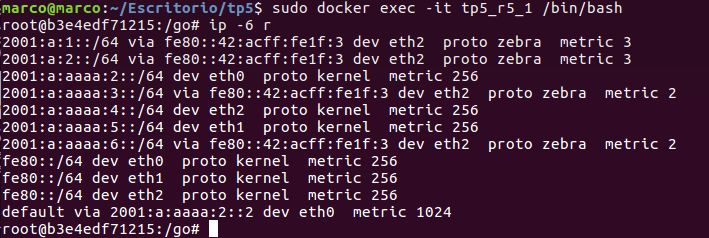
Por último, se verifica conectividad y el trayecto que realizan los paquetes desde el host 11 del sistema autónomo 101, al router 9 del sistema autónomo 303:

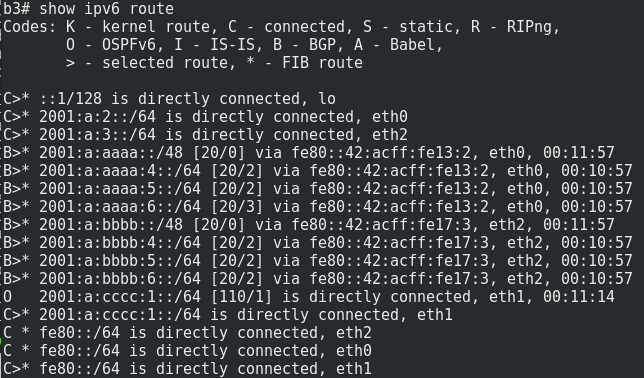


Podemos ver la tabla de routeo del router de frontera b3, en la que vemos que por ser un router de frontera conoce como llegar a todas las redes:

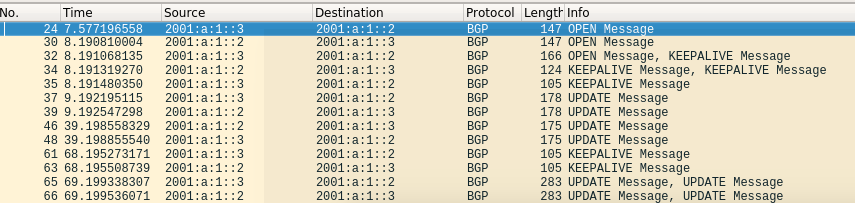


Ahora si vemos la tabla de routeo del router r5, el cual no es un router de frontera, sino que está dentro del sistema autónomo 101, solo conoce cómo llegar a los nodos dentro de su misma área:



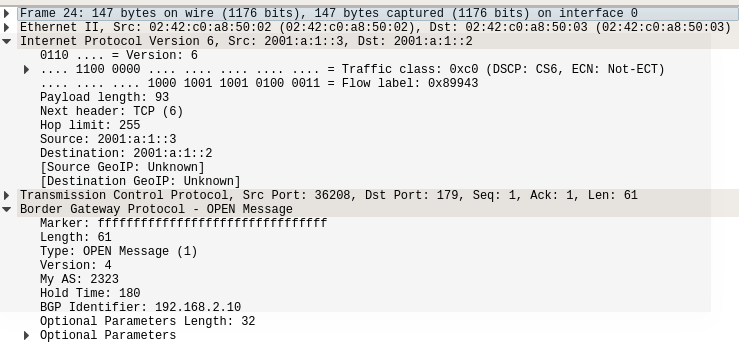
Desde la conexión telnet al router de borde b3 se obtuvo la siguiente información sobre la tabla de ruteo y cómo fueron aprendidas las rutas.  


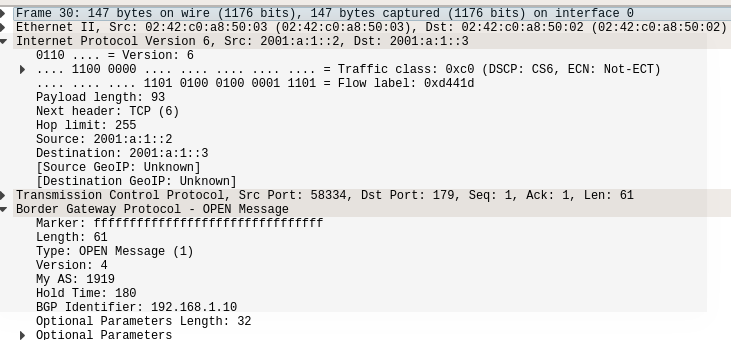
10) Al analizar los paquetes bgp con wireshark, se pudieron observar 3 tipos de paquetes distintos. Al iniciar los dockers, se ven que los routers de borde se envían entre sí un mensaje de OPEN.



El cual contiene un identificador BGP, el cual es el mismo que está configurado en el archivo bgpd.conf, de esta forma cada uno de los routers bgp guarda información de los otros routers a los que está conectado.

La cabecera del mensaje OPEN es el siguiente:

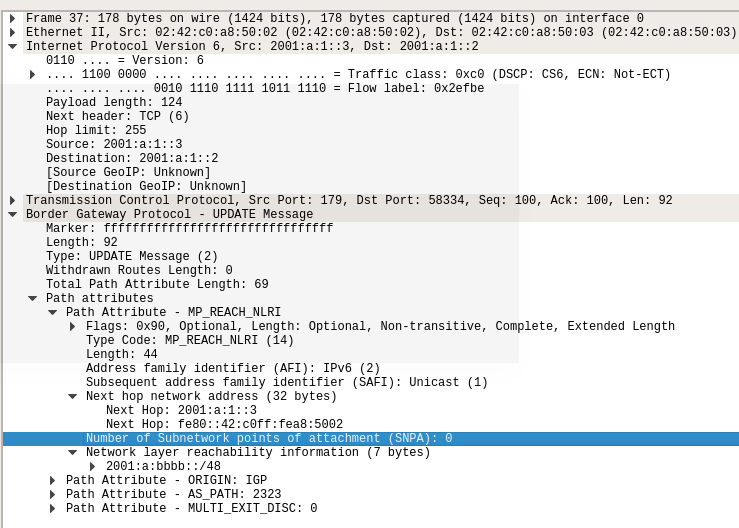




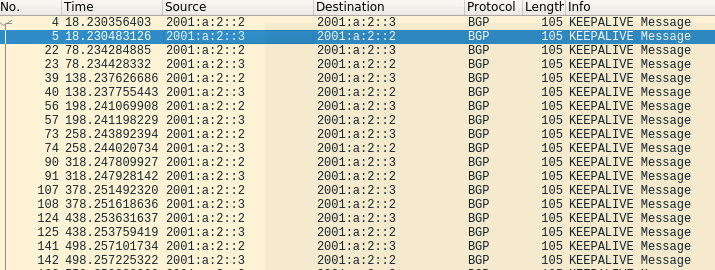
Una vez que un router de borde tiene información sobre su propio sistema autónomo, envía mensajes de UPDATE a los demás routers de borde a los que está conectado, para que ellos sepan cuáles son las redes de esos sistemas autónomos. En estos mensajes se puede observar que se indica la dirección y máscara de red del sistema autónomo.

Los mensajes update se envían siempre que haya información nuevo proveniente del sistema autónomo.

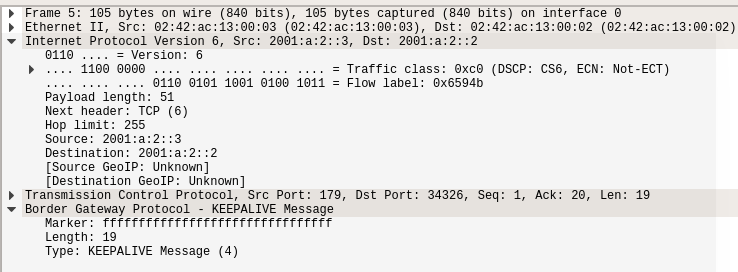
La cabecera del mensaje UPDATE es el siguiente:



Periódicamente (cada 60 segundos) se envía un mensaje KEEPALIVE, para comunicarle al resto de los routers de borde que el sistema autónomo sigue disponible.



La cabecera del mensaje KEEPALIVE es el siguiente:



No se observaron mensajes de NOTIFICATION que se envía al cerrar una sesión BGP (generalmente por un error que obliga a cerrar la conexión).

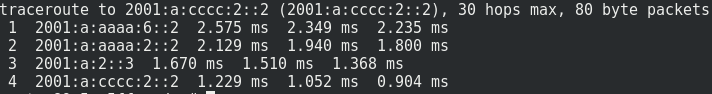
11)

11.1) La política impuesta desde el router de borde B2 estableció no compartir las redes internas conocidas por el con el otro router de borde B1, aunque sí con el router B2.

Por lo tanto,el router B1 sólo conoce cómo llegar a las rutas internas de B3 por medio de B2.

Esto se puede aplicar modificando el archivo bgpd.conf, aceptando o no al router B1 como vecino.

Informando las rutas internas propias a B1:



Luego de modificar la política de B3, no informando al router B1 de sus rutas internas:

