# Laboratory 2 ARP; Zebra routing daemon

# Part1. Introduction

## ARP

Address Resolution Protocol, ARP, is used by a system, which wants to send data an IP address on the local network, and it doesn't know the destination MAC address. Systems keep an ARP look-up table where they store information about the association between the IP and MAC addresses. If the MAC address is not in the ARP table, then ARP protocol is used it knowing the destination IP addresss.

ARP operation for communications inside the local network:

- System checks its ARP table for the MAC address associated with the IP address.
- If the MAC address is not in the ARP table, an ARP request is broadcasted in the local network, requesting the MAC address for the specified IP address.
- The machine with the requested IP address will reply with an ARP packet containing its MAC address.
- Thepacket is sent to the learned MAC address.

ARP operation for communication between hosts located in different networks

- System determines that the IP address does not belong to the local network and decides to send the packet to the gateway. It has to determine the MAC address of the gateway.
- It broadcast an ARP request asking for the MAC address of the IP address belonging to the gateway. It knows the gateway's IP address from the static route specifying the ***default gateway***.
- The gateway will reply with its MAC address.
- The packet is sent to the gateway.
- The gateway will be in charge with sending the packet to the next hop towards the destination.

## Zebra routing software

A router device decides where tos end an incoming packet based on the routing table. The routing table can be populated statically, using static routes for each destination, or it can be populated dynamically by a routing protocol. Zebra is a routing software package that provides TCP/IP based routing services with routing protocols support such as RIP, OSPF , IS-IS, and BGP. Zebra development stoped and now it is replaced by Quagga: "a fork of GNU Zebra [that] aims to build a more involved community around Quagga than the current centralised model of GNU Zebra".

# Part 2: Experimental part

Basic network topologies and network nodes configuration for a better understanding of the ARP protocol and zebra routing daemon.

***Remark***:

Do not forget to set the Netkit environment variables and check your configuration:

- `export NETKIT_HOME=~/netkit`
- `export MANPATH=:$NETKIT_HOME/man`
- `export PATH=$NETKIT_HOME/bin:$PATH`

Check your configuration:

- `./check_configuration.sh`

## Study of ARP protocol

In this activity two local area networks will be connected with path consisting of three routers as in Figure 1. `Ping` and tcpdump commands will be used to trigger the ARP protocol and analyse its operation.

### Network topology

The network topology is presented in Figure 3.

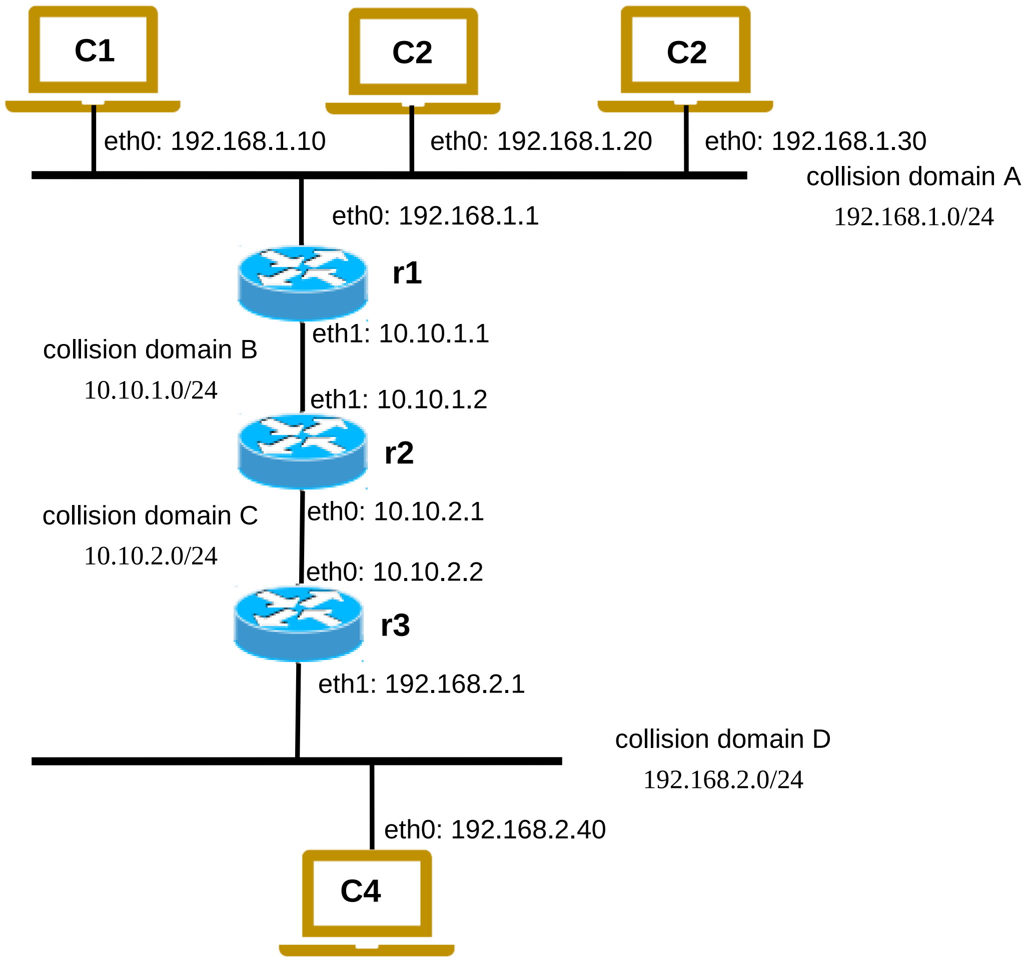



Figure 1: Network topology

## 1. Creating the topology lab

Inside the `netkit` folder create a new folder `lab2`. Inside the `lab2` folder create the structure of subdirectories and files associated with the topology.

- Create an empty folder for each device in the topology:C1 folder, C2 folder, …
- Create the `lab.conf` file where the topology is described:
  - `c1[0]=A`
  - …
  - `r1[0]=A`
  - `r1[1]=B`
  - …
- Create the `.startup` files for each device were the initial configuration is specified:
  - `c1.startup`:
    - `ifconfig eth0 192.168.1.10 netmask 255.255.255.0 up`
    - `route add default gw 192.168.1.1`
  - `c2.startup`
    - …
  - `c3.startup`
    - …
  - `r1.startup`
    - `ifconfig eth0 192.168.1.10 netmask 255.255.255.0 up`
    - `ifconfig eth1 10.10.1.1 netmask 255.255.255.0 up`
    - `route add -net 192.168.2.0 netmask 255.255.255.0 gw 10.10.1.2 dev eth1`
    - `route add -net 10.10.2.0 netmask 255.255.255.0 gw 10.10.1.2 dev eth1`
  - `r2.startup`
    - …
  - `r3.startup`
    - …
  - `c4.startup`
    - …

Start the lab with `lstart` command:

- **~/netkit$** `lstart –d ~/netkit/lab2`

## 2. Inspect the ARP tables

To avoid flooding the network with ARP messages, the network devices store the MAC addresses learned from previous ARP querries to a local table/cache. With `arp` command one can inspect the content of the ARP cache (Figure 2).

- On `c2` machine inspect the ARP table with `arp` command
  - `c2:~# arp`
- `Ping` from `c2` the `c1` machine
  - `c2:~# ping 192.168.1.10`
- Inspect the content of the `c2` and `c1` ARP tables with `arp` command

Q1: Which MAC addresses are learned by `c1` and `c2`?

Repeat the experiment by pinging `c4` machine from `c1`. Inspect the ARP cache on `c1, r1, r2, r3, c4`.

Q2: Which MAC addresses are learned by each device (`c1, r1, r2, r3, c4`)?

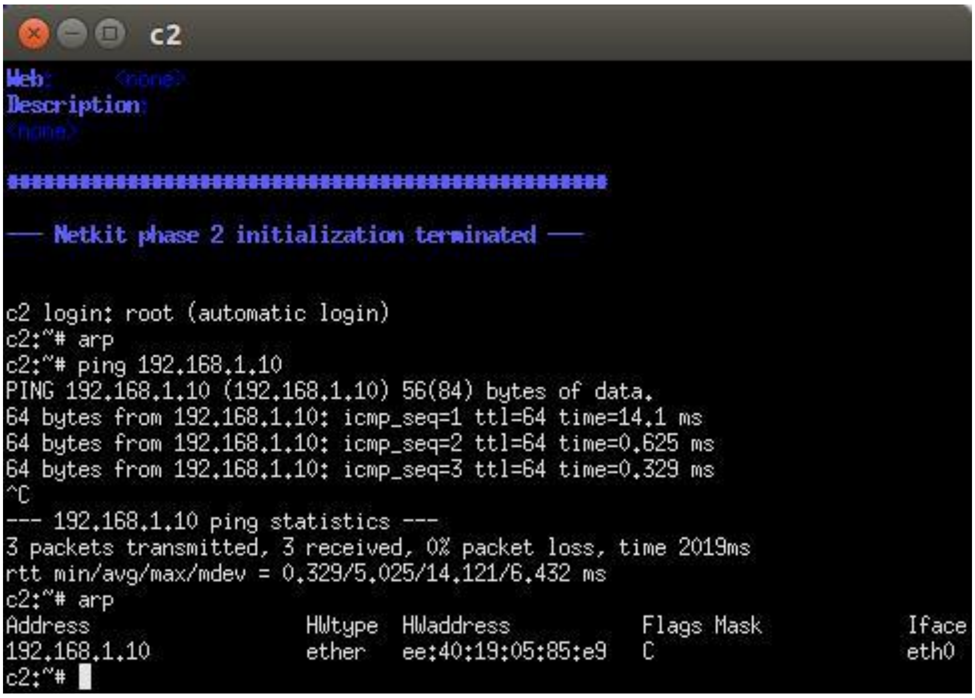


Figure 2: Inspecting the ARP cache on machine `c2`

### 3. Analyzing the ARP traffic

Restart the lab in ordet to clear the ARP caches.

- `$ lcrash -d /path/to/lab/directory/`
- `$ lstart -d /path/to/lab/directory/`

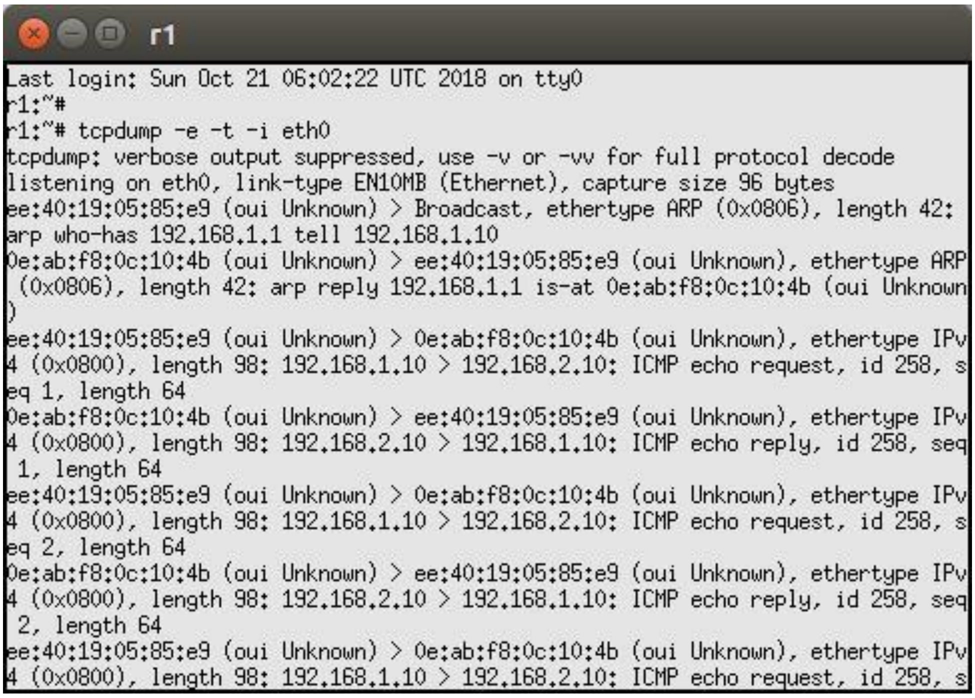
Start capturing the traffic on `r1-eth0, r2-eth1, r3-eth0, c4`, with `tcpdump` command.

- `r1:~# tcpdump -e  -t  -i eth0`
- `r2:~# tcpdump -e  -t  -i eth1`
- …

From `c1` machine start `ping` the `c2` machine. Send only two ICMP packets with the `ping` command (**-c 2** parameter instruct `ping` to send only two packets):

- `c1:~# ping -c 2 192.168.2.40`

Analyse the packets captures with tcpdump on `r1, r2, r3, c4` (Figure 3).



```
Last login: Sun Oct 21 06:02:22 UTC 2018 on tty0
r1:~#
r1:~# tcpdump -e -t -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
ee:40:19:05:85:e9 (oui Unknown) > Broadcast, ethertype ARP (0x0806), length 42:
arp who-has 192.168.1.1 tell 192.168.1.10
0e:ab:f8:0c:10:4b (oui Unknown) > ee:40:19:05:85:e9 (oui Unknown), ethertype ARP
 (0x0806), length 42: arp reply 192.168.1.1 is-at 0e:ab:f8:0c:10:4b (oui Unknown
)
ee:40:19:05:85:e9 (oui Unknown) > 0e:ab:f8:0c:10:4b (oui Unknown), ethertype IPv
4 (0x0800), length 98: 192.168.1.10 > 192.168.2.10: ICMP echo request, id 258, s
eq 1, length 64
0e:ab:f8:0c:10:4b (oui Unknown) > ee:40:19:05:85:e9 (oui Unknown), ethertype IPv
4 (0x0800), length 98: 192.168.2.10 > 192.168.1.10: ICMP echo reply, id 258, seq
 1, length 64
ee:40:19:05:85:e9 (oui Unknown) > 0e:ab:f8:0c:10:4b (oui Unknown), ethertype IPv
4 (0x0800), length 98: 192.168.1.10 > 192.168.2.10: ICMP echo request, id 258, s
eq 2, length 64
0e:ab:f8:0c:10:4b (oui Unknown) > ee:40:19:05:85:e9 (oui Unknown), ethertype IPv
4 (0x0800), length 98: 192.168.2.10 > 192.168.1.10: ICMP echo reply, id 258, seq
 2, length 64
ee:40:19:05:85:e9 (oui Unknown) > 0e:ab:f8:0c:10:4b (oui Unknown), ethertype IPv
4 (0x0800), length 98: 192.168.1.10 > 192.168.2.10: ICMP echo request, id 258, s
```

Figure 3: Packets captured with `tcpdump` on `r1`

Q3:Draw the message sequence chart for the messages exchanged in the network until the first ICMP echo reply returns to `c1`.

Stop tcpdump on `r1, r2, r3, c4` (use `Ctrl C` to stop `tcpdump`)

Start capturing the traffic on `r1-eth0, r2-eth1, r3-eth0, c4`, with `tcpdump` command in verbose mode (the verbose option instructs tcpdump to print more output data on the screen).

- `r1:~# tcpdump –e -v -t  -i eth0`
- `r2:~# tcpdump –e -v -t  -i eth1`
- …

From `c1` machine start `traceroute` command to find out the path to the `c2` machine.:

- `c1:~# traceroute 192.168.2.40`

Analyse the packets captures with tcpdump on `r1, r2, r3, c4`. Pay attention at TTL parameter and at the ICMP response provided by the routers along the path based on TTL value.

Q4: Explain how `traceroute` works based on the packets captured with `tcpdump`.

## Study of Zebra routing agent

The configuration files for zebra software are located in the folder `/etc/zebra`.There is a configuration file for each routing protocol supported by zebra (RIP, OSPF, IS-IS, and BGP). The daemons file is used to configure which routing Zebra is a routing software package that provides TCP/IP based routing services with routing protocols support such as RIP, OSPF and BGP.

```
r1
r1:~#
r1:~#
r1:~#
r1:~# cd /etc/zebra
r1:/etc/zebra# ls
bgpd.conf          isisd.conf           ospfd.conf          ripngd.conf.sample
bgpd.conf.sample   isisd.conf.sample    ospfd.conf.sample   vtysh.conf.sample
bgpd.conf.sample2  ldpd.conf            ripd.conf           zebra.conf
daemons            ospf6d.conf          ripd.conf.sample    zebra.conf.sample
debian.conf        ospf6d.conf.sample   ripngd.conf
r1:/etc/zebra# less daemons
r1:/etc/zebra# less daemons
r1:/etc/zebra# tail daemons
#
zebra=yes
bgpd=no
ospfd=no
ospf6d=no
ripd=no
ripngd=no
isisd=no
ldpd=no

r1:/etc/zebra#
```

Figure 4:zebra files; daemons file content

```
r1
! -*- zebra -*-
!
! zebra sample configuration file
!
! $Id: zebra.conf.sample,v 1.1.1.1 2002/12/13 20:15:30 paul Exp $
!
hostname Router
password zebra
!enable password zebra
!
! Interface's description.
!
!interface lo
! description test of desc.
!
!interface sit0
! multicast

!
! Static default route sample.
!
!ip route 0.0.0.0/0 203.181.89.241
!
zebra.conf
```

Figure 5: zebra.conf file

In Figures 4 and 6 there are presented the configuration files of the zebra software. The `daemons` file is used to configure which routing daemon is started. The `zebra.conf` file stores the configuration of the router.

One can use `telnet` command to connect to the main zebra daemon or the other routing daemons.The command will work only if the selected daemon is started.

## 1. Inspecting the zebra configuration shell

This section helps to familiarize with the zebra configuration console and with the basic commands.

To start the zebra daemon the following command will be used:

- `/etc/init.d/zebra start`

To connect to the main zebra daemon:

- `r1:~# telnet localhost zebra`

As a result of the `telnet` command the zebra management shell is opened. The configuration of zebra using the zebra shell is similar with the configuration of a CISCO router using the CLI interface (Figure 6). The are three operation modes in the management shell: unprivileged user mode, privileged user mode and configurator mode.

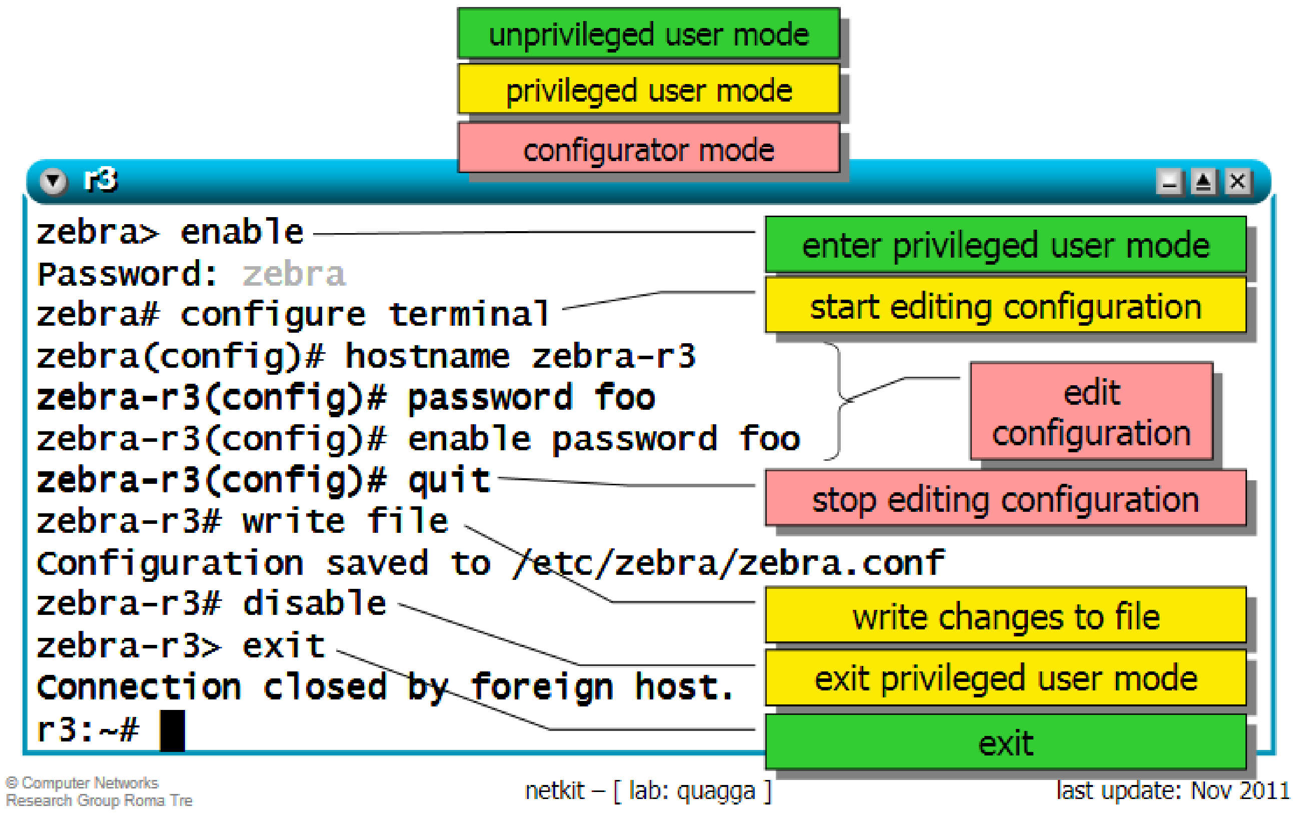



Figure 6: Configuring zebra routing daemon from the zebra shell [2]

The `list` command will show the existing command in the zebra shell, in each operation mode (unprivileged user mode, privileged user mode and configuration mode) – Figure 7.

```
r1
Router# list
  clear ip prefix-list
  clear ip prefix-list WORD
  clear ip prefix-list WORD A.B.C.D/M
  clear ipv6 prefix-list
  clear ipv6 prefix-list WORD
  clear ipv6 prefix-list WORD X:X::X:X/M
  configure terminal
  copy running-config startup-config
  debug zebra events
  debug zebra kernel
  debug zebra packet
  debug zebra packet (recv|send)
  debug zebra packet (recv|send) detail
  debug zebra rib
  debug zebra rib queue
  disable
  end
  exit
  help
  list
  logmsg (emergencies|alerts|critical|errors|warnings|notifications|informationa
l|debugging) .MESSAGE
 --More--
```

Figure 7: Listing the commands in privileged mode with `list` command

The show command is used to print the routing table, the interfaces configuration, etc. In Figure 8 is presented the output of `show ip route` and `show interface eth0` on device `r1,` which is named Router on zebra.

```
r1
Router# sh ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 10.10.1.0/24 is directly connected, eth1
K>* 10.10.2.0/24 via 10.10.1.2, eth1
C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.1.0/24 is directly connected, eth0
K>* 192.168.2.0/24 via 10.10.1.2, eth1
Router# sh ip inter
Router# sh ip interf
Router# sh int
Router# sh interface eth
Router# sh interface eth0
Interface eth0 is up, line protocol detection is disabled
  index 3 metric 1 mtu 1500
  flags: <UP,BROADCAST,RUNNING,MULTICAST>
  HWaddr: 0e:ab:f8:0c:10:4b
  inet 192.168.1.1/24 broadcast 192.168.1.255
  inet6 fe80::cab:f8ff:fe0c:104b/64
    39 input packets (0 multicast), 1764 bytes, 0 dropped
    0 input errors, 0 length, 0 overrun, 0 CRC, 0 frame
    0 fifo, 0 missed
    39 output packets, 2982 bytes, 0 dropped
    0 output errors, 0 aborted, 0 carrier, 0 fifo, 0 heartb
    0 window, 0 collisions
Router#
gnome-scr
Take Screenshot
```

Figure 8: The output of `show ip route` and `show interface eth0` commands

Instead of having to connect to each single daemon, users can interact with quagga by using a built-in shell, `vtysh`. All the commands from the single routing daemons (including zebra itself) are available in this shell.

- `r1:~# vtysh`

### 2. Start the network topology

To study the zebra routing software the topology given in Figure 1 will be used. Update the `.startup` files for routers `r1`, `r2`, and r3 by removing the static routes. The static routes will be introduced in the zebra software.

- `r1.startup`
    - `ifconfig eth0 192.168.1.10 netmask 255.255.255.0 up`
    - `ifconfig eth1 10.10.1.1 netmask 255.255.255.0 up`
- …

Restart the lab topology:

- `$ lcrash –d /path/to/lab/directory/`
- `$ lstart –d /path/to/lab/directory/`

Start the zebra routing daemon:

- `/etc/init.d/zebra start`

Connect to the zebra routing using `wtysh`:

- `r1:~# vtysh`
- …

Configure the static routes on each router inside the zebra daemon:

- Use `ip route` command on each router.
-

## References:

1. http://wiki.netkit.org/netkit-labs/netkit-labs_basic-topics/netkit-lab_arp/netkit-lab_arp.pdf
2. http://wiki.netkit.org/netkit-labs/netkit-labs_basic-topics/netkit-lab_quagga/netkit-lab_quagga.pdf