Workshop 2 (3 weeks)

Table of Contents

Objectives

- Use FRR routers to emulate a network in GNS3.
- Learn and observe the basics of routing protocols.
- Program a basic routing protocol (Dijkstra's algorithm).
- Learn the basics of socket programming using Python socket libraries.

Common objectives of all workshops:

- Gaining hands-on experience and learning by doing!
- Understanding how theoretical knowledge gained in lectures relates to practice.
- Observe how networks and systems operate in practice and develop curiosity for gaining further theoretical knowledge.

Overview

In this workshop, you will first use GNS3 to emulate a network with multiple subnets and routers. This will let you gain hands-on experience with routing protocols and observe how they work in practice. Secondly, you will learn how to represent simple graphs in Python using the *networkx* library and program a basic routing algorithm (Dijkstra's algorithm) and find the shortest paths on simple networks (graphs). Finally, you will learn the basics of network programming using sockets in Python and implement a simple client-server messaging program. Thanks to its "batteries included" philosophy, the powerful libraries of Python make these tasks more straightforward than in any other language.

As before, we have discussed a basic overview of these in the lectures and will continue to do so in the upcoming lectures. ***Note** that the learning order does not always have to be first lecture, then workshop. It can be a lecture (overview), workshop, and lecture (details).*

Workshop Preparation: (before you arrive at the lab)

We recommend that you prepare before coming to workshops and learn much more in workshops!

We will give you a lot of time to finish the tasks but those arethe bare minimums. Just like in the lectures, the topics we cover in the workshops are pretty deep and we can only do somuch in two hours. There is much more to learn and being prepared for the workshop is one of the best ways to gain more knowledge! For example, there are a few questions in each workshop which you can answer beforehand.

Self-learning is one of the most important skills that you should acquire as a student. Today, self-learning is much easier than it used to be thanks to a plethora of online resources.

For this workshop, start by exploring the resource mentioned in the preparation steps below.

Workshop Preparation Steps:

1. *Common steps for all workshops*: read the Workshop Manual beforehand!
2. Review relevant lecture slides on layering, protocol stack, and encapsulation.
3. Read the relevant online documentation of the software we use (Wireshark, FRR, GNS3, Networkx etc.)

Did you know?

- FRRouting (FRR) is a free and open-source Internet routing protocol suite for Linux and Unix platforms. Users of FRR include ISPs, web businesses, and Fortune 500 private clouds!
- Networkx is a very well-known Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

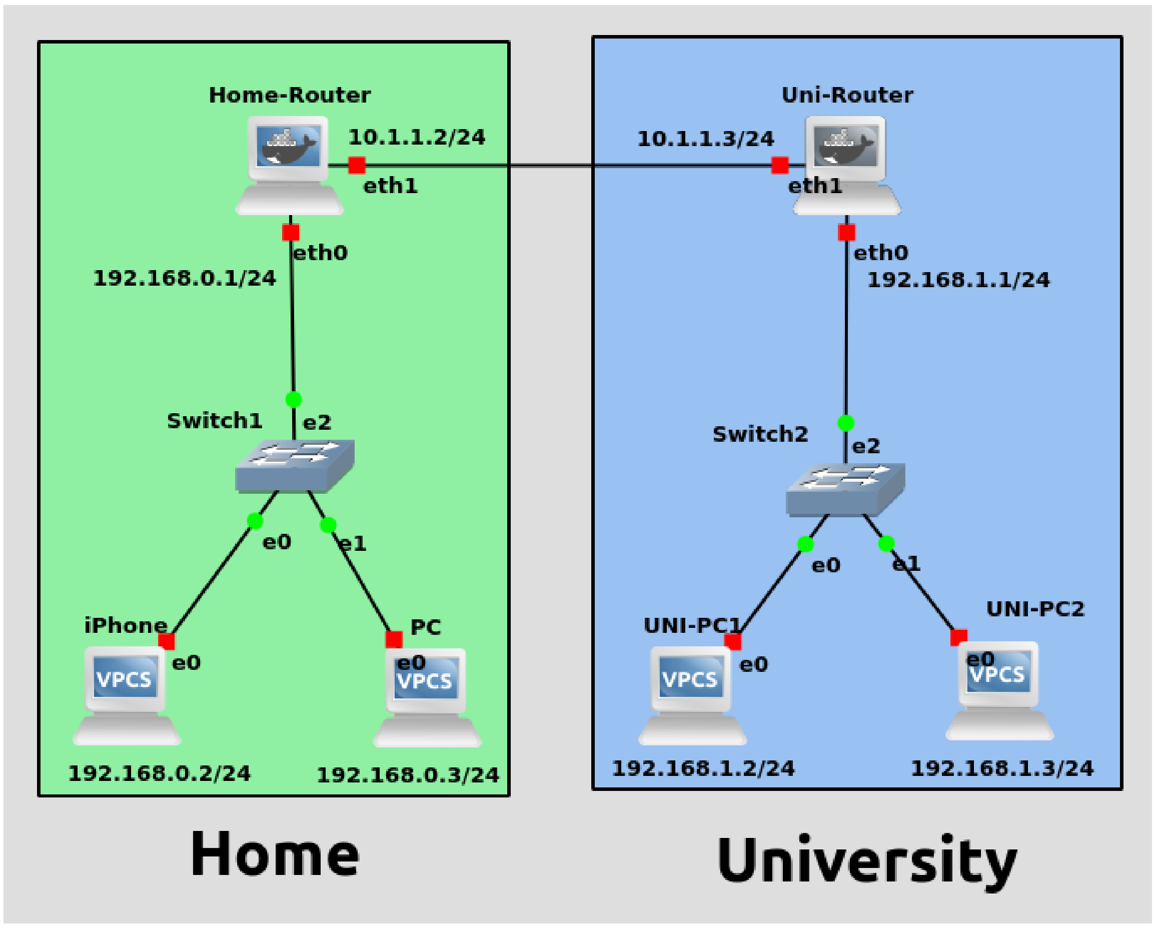
Tasks and Questions:

Follow the procedures described below and answer the workshop questions for your Workshop Report and oral quiz which you will prepare according to the instructions. Keep your answers shortand legible!

The goal is to learn, NOT blindly follow the procedures in the fastest possible way!

Part 1 – Routing using FRR in GNS3.

This networking task builds upon the LAN you emulated in the previous workshop. This time, we have two LANs (Home and University) that are connected to each other with two FRR routers (Home-Router and Uni-Router) as shown below.



You already know how to set up an Ethernet Switch and connect VPCSs to a Switch from the previous workshop. Therefore, let's focus on how to achieve the setup shown above starting from the routers.

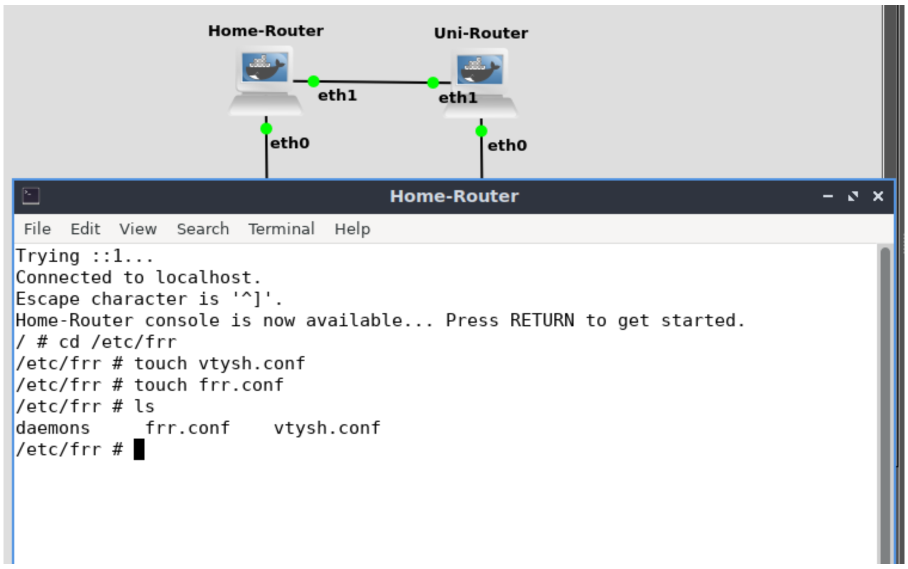
Step 1 – Configure and connect two FRR Routers in GNS3.

Task 1. Drag two FRRs and two Ethernet Switches to the GNS3 workspace and connect them as shown. Rename the FRRs as Home-Router and Uni-Router (using right-click, change hostname). For simplicity, match the interfaces to what is shown above. For example, Home-Router eth0 interface is connected to Switch 1 eth2 (e2) and Home-Router and Uni-Router are connected to each other via eth1. Note that, you can use View/Show interface labels from GNS3 main menu to see labels of network interfaces.

Task 2. First, we need to do a minor bit of configuration on each router. Start all devices and then first double-click Home-Router to open its (Linux) Terminal (or choose Console from its right-click menu). Write the following commands in the Terminal of the Home-Router:

```
cd /etc/frr
touch vtysh.conf
touch frr.conf
```

Check using the ls command that both of the empty files (vtysh.conf and frr.conf) are created. (These files will be used in the background when you save your router configuration. They should have been there in the Docker image we use, but they are not – something to update in the future!)



```
Trying ::1...
Connected to localhost.
Escape character is '^]'.
Home-Router console is now available... Press RETURN to get started.
/ # cd /etc/frr
/etc/frr # touch vtysh.conf
/etc/frr # touch frr.conf
/etc/frr # ls
daemons     frr.conf    vtysh.conf
/etc/frr #
```

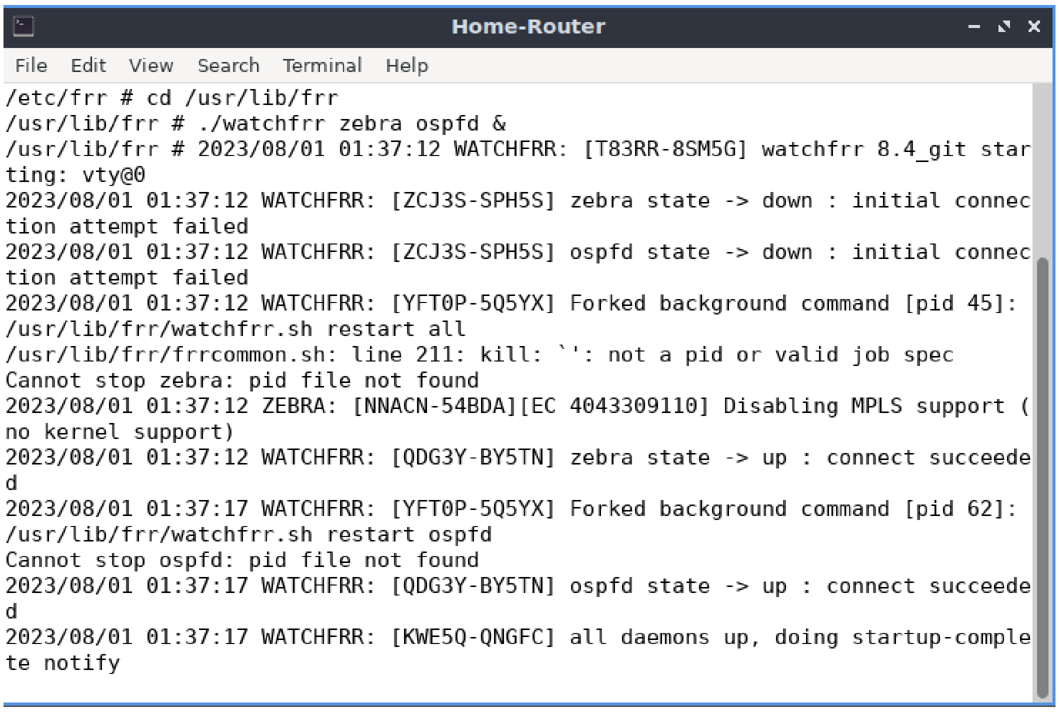
Next, start the router and start a couple of background processes:

cd /usr/lib/frr
./watchfrr zebra ospfd &

Some info about what is going on here:

- watchfrr is a daemon (fancy name for any Unix background process) that handles failed daemon processes and intelligently restarts them as needed.
- zebra is an IP routing manager. It provides kernel routing table updates, interface lookups, and redistribution of routes between different routing protocols.
- ospfd (ospf daemon) starts OSPF version 2 is a routing protocol which is described in RFC 2328. OSPF is an IGP. Compared with RIP, OSPF can provide scalable network support and faster convergence times. OSPF is widely used in large networks such as ISP backbone and enterprise networks.
- There are other (better) ways of configuring this FRR router, but this is sufficient for now and our purposes.

You should see an output like this.



```
/etc/frr # cd /usr/lib/frr
/usr/lib/frr # ./watchfrr zebra ospfd &
/usr/lib/frr # 2023/08/01 01:37:12 WATCHFRR: [T83RR-8SM5G] watchfrr 8.4_git star
ting: vty@0
2023/08/01 01:37:12 WATCHFRR: [ZCJ3S-SPH5S] zebra state -> down : initial connec
tion attempt failed
2023/08/01 01:37:12 WATCHFRR: [ZCJ3S-SPH5S] ospfd state -> down : initial connec
tion attempt failed
2023/08/01 01:37:12 WATCHFRR: [YFT0P-5Q5YX] Forked background command [pid 45]:
/usr/lib/frr/watchfrr.sh restart all
/usr/lib/frr/frrcommon.sh: line 211: kill: `': not a pid or valid job spec
Cannot stop zebra: pid file not found
2023/08/01 01:37:12 ZEBRA: [NNACN-54BDA][EC 4043309110] Disabling MPLS support (
no kernel support)
2023/08/01 01:37:12 WATCHFRR: [QDG3Y-BY5TN] zebra state -> up : connect succeede
d
2023/08/01 01:37:17 WATCHFRR: [YFT0P-5Q5YX] Forked background command [pid 62]:
/usr/lib/frr/watchfrr.sh restart ospfd
Cannot stop ospfd: pid file not found
2023/08/01 01:37:17 WATCHFRR: [QDG3Y-BY5TN] ospfd state -> up : connect succeede
d
2023/08/01 01:37:17 WATCHFRR: [KWE5Q-QNGFC] all daemons up, doing startup-comple
te notify
```

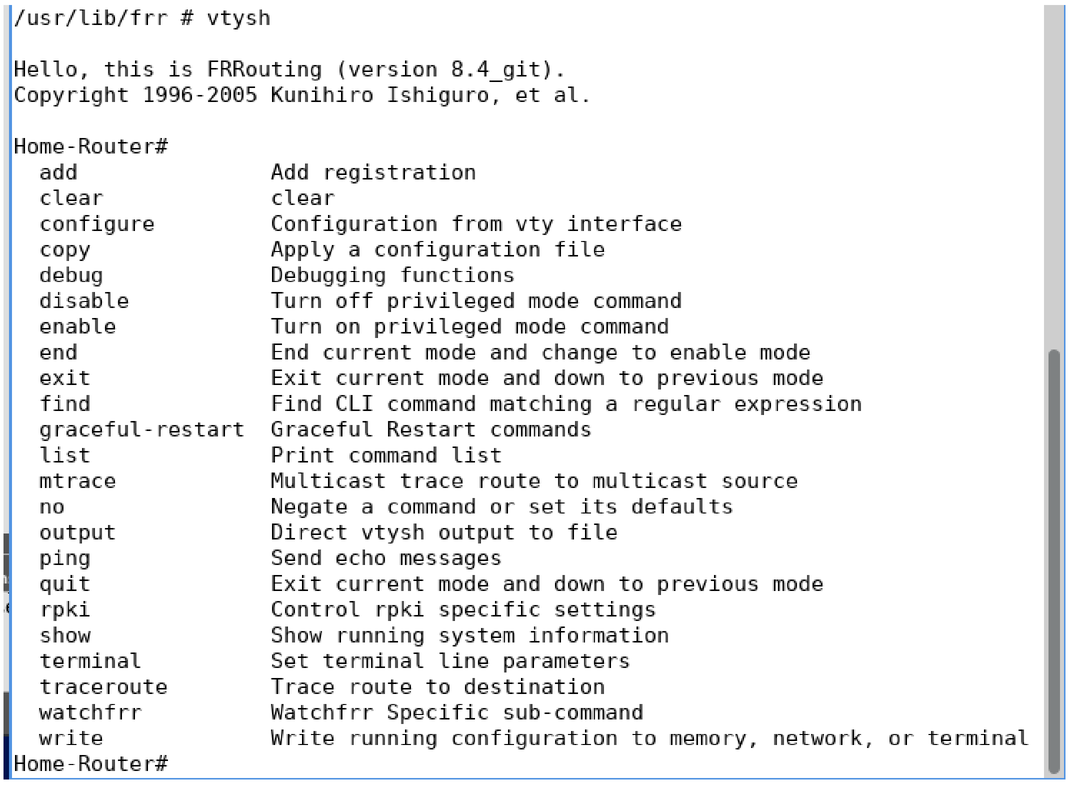
Next, press *enter* to see the prompt in the terminal.

Task 3. We will now set up the router interfaces. For this, we will use a special program (shell) called vtysh, which provides a combined frontend to all FRR daemons in a single combined session.

Type *vtysh* into the terminal and press *enter* to enter the vtysh shell.

vtysh

If you press "?" or write "list", it will show a list of commands in the top level of *vtysh*.

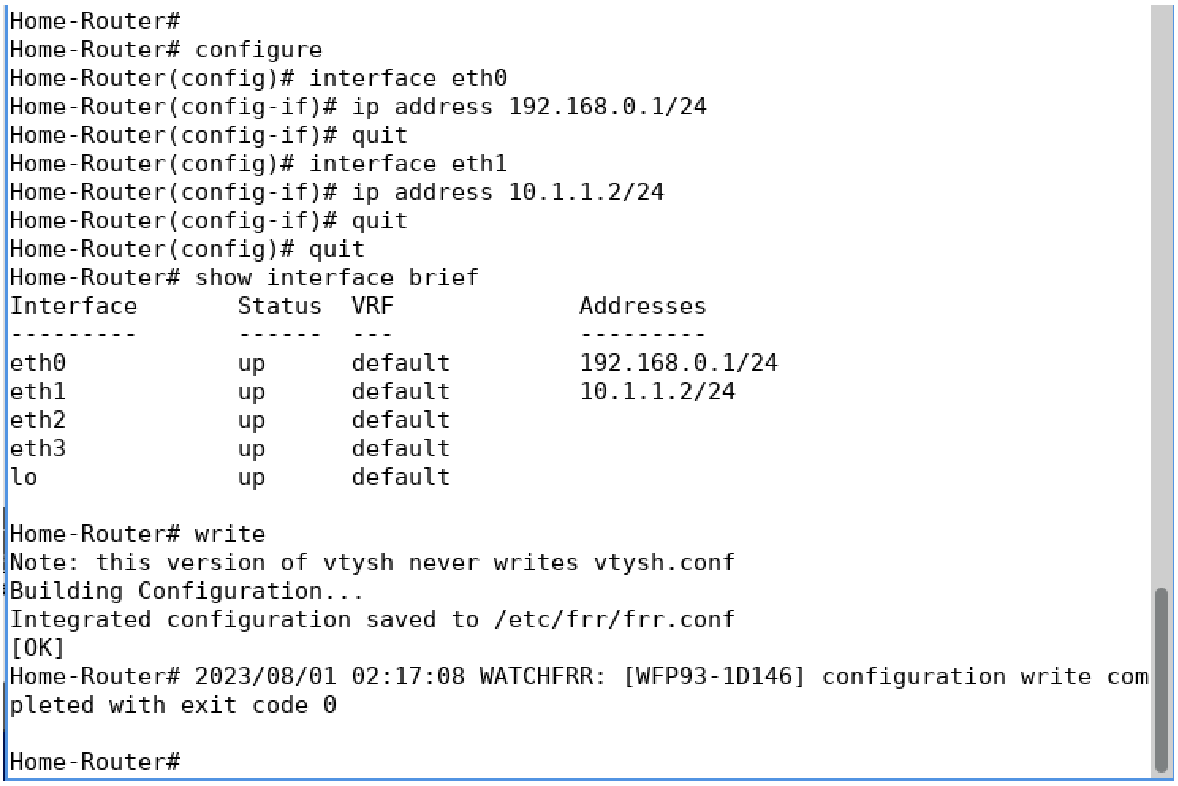


```
/usr/lib/frr # vtysh

Hello, this is FRRouting (version 8.4_git).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

Home-Router#
  add                 Add registration
  clear               clear
  configure           Configuration from vty interface
  copy                Apply a configuration file
  debug               Debugging functions
  disable             Turn off privileged mode command
  enable              Turn on privileged mode command
  end                 End current mode and change to enable mode
  exit                Exit current mode and down to previous mode
  find                Find CLI command matching a regular expression
  graceful-restart    Graceful Restart commands
  list                Print command list
  mtrace              Multicast trace route to multicast source
  no                  Negate a command or set its defaults
  output              Direct vtysh output to file
  ping                Send echo messages
  quit                Exit current mode and down to previous mode
  rpki                Control rpki specific settings
  show                Show running system information
  terminal            Set terminal line parameters
  traceroute          Trace route to destination
  watchfrr            Watchfrr Specific sub-command
  write               Write running configuration to memory, network, or terminal
Home-Router#
```

Next, we need to configure the interfaces of our router. This is achieved through the following commands. Ignore the comments marked with # below.

configure   #or config
interface eth0
ip address 192.168.0.1/24
quit   # or q to exit the interface
interface eth1
ip address 10.1.1.2/24
quit # exit interface
quit # to exit configure
show interface brief #shows the interface configuration in a brief way

write # saves the configuration



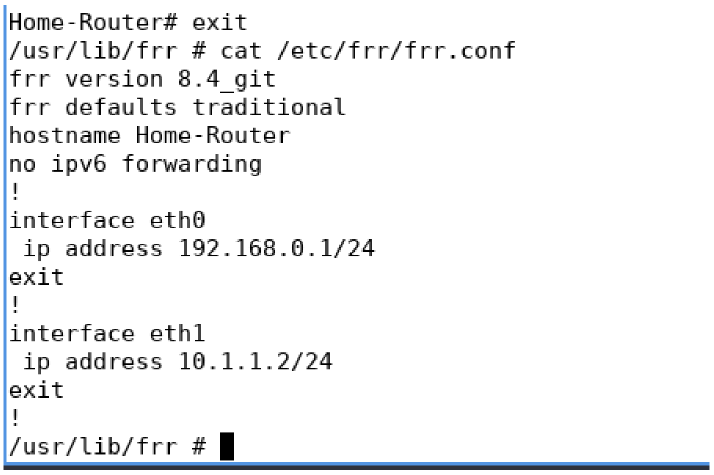
```
Home-Router#
Home-Router# configure
Home-Router(config)# interface eth0
Home-Router(config-if)# ip address 192.168.0.1/24
Home-Router(config-if)# quit
Home-Router(config)# interface eth1
Home-Router(config-if)# ip address 10.1.1.2/24
Home-Router(config-if)# quit
Home-Router(config)# quit
Home-Router# show interface brief
Interface       Status  VRF             Addresses
---------       ------  ---             ---------
eth0            up      default         192.168.0.1/24
eth1            up      default         10.1.1.2/24
eth2            up      default
eth3            up      default
lo              up      default

Home-Router# write
Note: this version of vtysh never writes vtysh.conf
Building Configuration...
Integrated configuration saved to /etc/frr/frr.conf
[OK]
Home-Router# 2023/08/01 02:17:08 WATCHFRR: [WFP93-1D146] configuration write com
pleted with exit code 0

Home-Router#
```

If you make a mistake with the configuration, there is an easy way to correct it. Use the *no ip address* command as shown below (and as mentioned in the help above) to remove IP addresses from interfaces.
For example,

no ip address 192.168.0.1/24

removes that IP address from the respective interface when used instead of the “ip address” command.

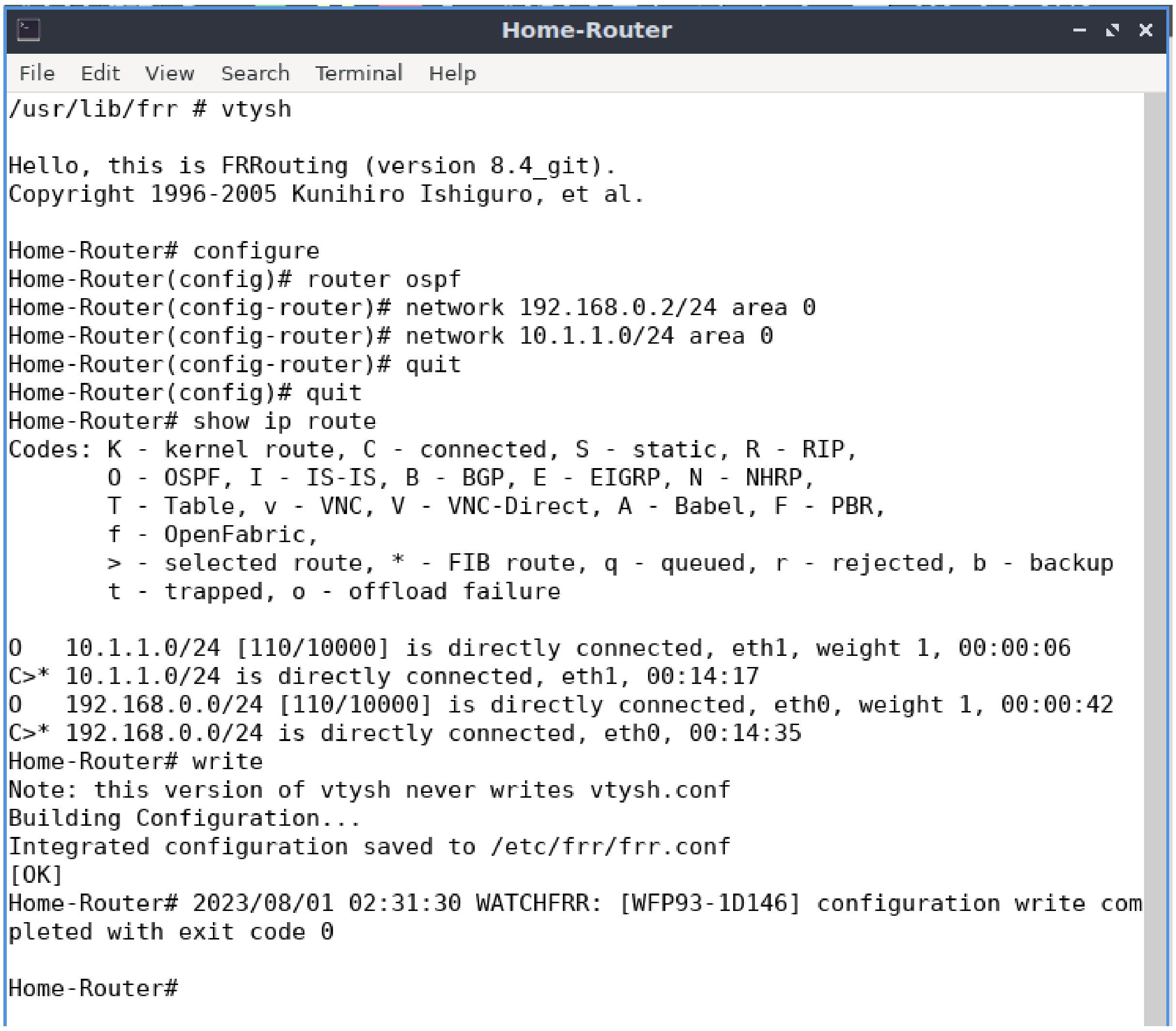
The configuration is saved to /etc/frr/frr.conf as you can see below:



```
Home-Router# exit
/usr/lib/frr # cat /etc/frr/frr.conf
frr version 8.4_git
frr defaults traditional
hostname Home-Router
no ipv6 forwarding
!
interface eth0
 ip address 192.168.0.1/24
exit
!
interface eth1
 ip address 10.1.1.2/24
exit
!
/usr/lib/frr #
```

Task 4. We will now set up the OSPF routing protocol at our Home-Router. The commands are:

Config
router ospf
network 192.168.0.2/24 area 0
network 10.1.1.0/24 area 0
quit
quit
show ip route
write



```
/usr/lib/frr # vtysh

Hello, this is FRRouting (version 8.4_git).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

Home-Router# configure
Home-Router(config)# router ospf
Home-Router(config-router)# network 192.168.0.2/24 area 0
Home-Router(config-router)# network 10.1.1.0/24 area 0
Home-Router(config-router)# quit
Home-Router(config)# quit
Home-Router# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

O   10.1.1.0/24 [110/10000] is directly connected, eth1, weight 1, 00:00:06
C>* 10.1.1.0/24 is directly connected, eth1, 00:14:17
O   192.168.0.0/24 [110/10000] is directly connected, eth0, weight 1, 00:00:42
C>* 192.168.0.0/24 is directly connected, eth0, 00:14:35
Home-Router# write
Note: this version of vtysh never writes vtysh.conf
Building Configuration...
Integrated configuration saved to /etc/frr/frr.conf
[OK]
Home-Router# 2023/08/01 02:31:30 WATCHFRR: [WFP93-1D146] configuration write com
pleted with exit code 0

Home-Router#
```

Task 5. Set up the router, its interfaces and OSPF routing protocol at the Uni-Router in a similar way. Here are the commands for your convenience:

cd /etc/frr
touch vtysh.conf
touch frr.conf

cd /usr/lib/frr
./watchfrr zebra ospfd &

```
config
int eth0
ip address 192.168.1.1/24
int eth1
ip address 10.1.1.3/24
q
q
show interface brief
write

config
router ospf
network 192.168.1.0/24 area 0
network 10.1.1.0/24 area 0
q
q
write
show ip route
```

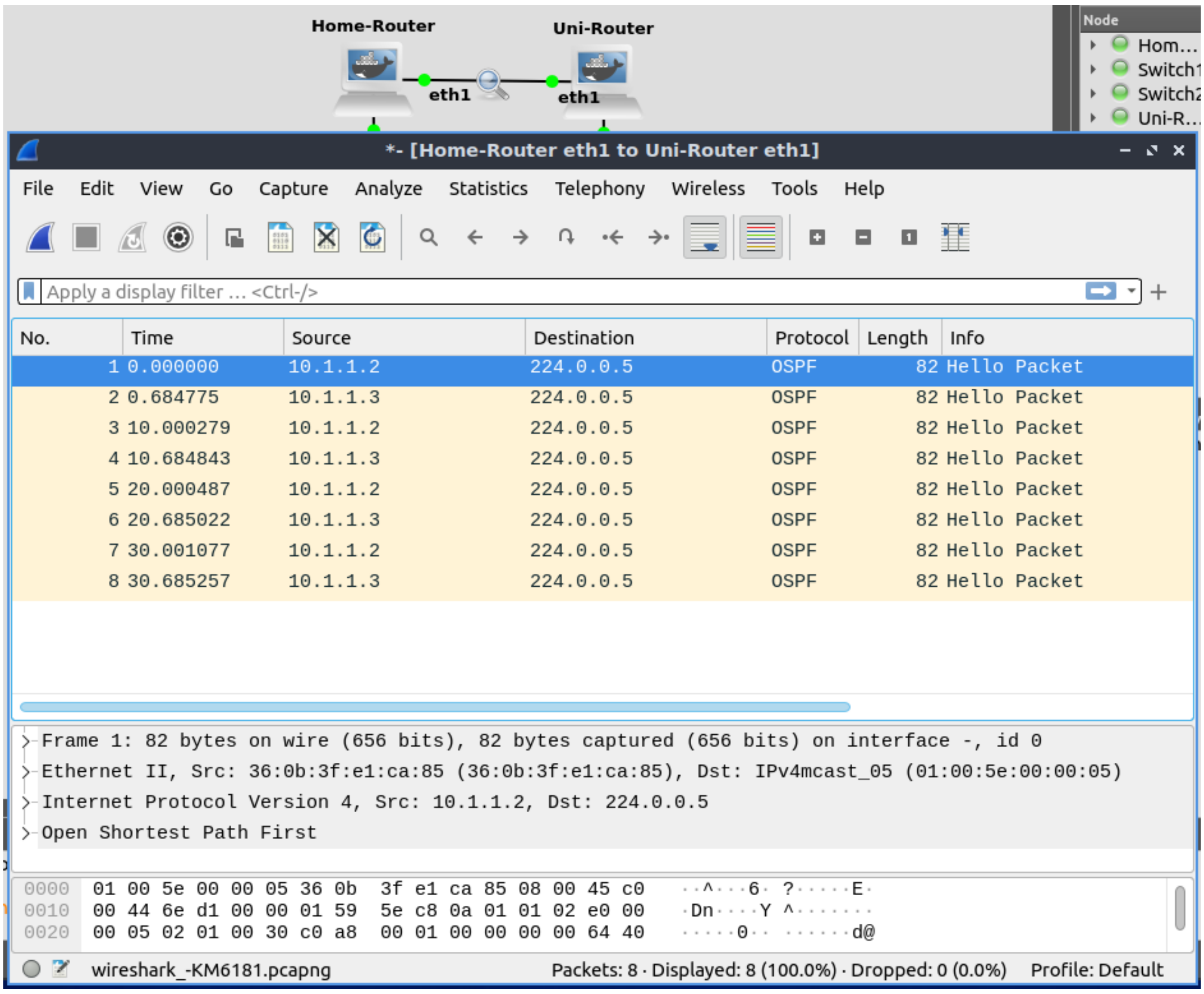
Reflect on the questions below and write your brief answers to the report, maybe along with a couple of screenshots, to illustrate what you have done. You are expected to answer such questions during the oral examination at the end of each workshop and show your working system.

Question 1.1. Why do we need a router to connect subnetworks? Based on the lectures, reflect on which layers and protocols play a role on which devices in the network configuration shown in the beginning.

Question 1.2. Is OSPF protocol the only available option in an FRR router? What are the other available options? How do they differ from each other? Explain at a high-level based on your knowledge from the lectures.

Step 2 – Use Wireshark to observe the network.

Right-click on the link between the two routers and choose “start capture” to start Wireshark. You should observe an output similar to the one below:



*- [Home-Router eth1 to Uni-Router eth1]

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.1.1.2	224.0.0.5	OSPF	82	Hello Packet
2	0.684775	10.1.1.3	224.0.0.5	OSPF	82	Hello Packet
3	10.000279	10.1.1.2	224.0.0.5	OSPF	82	Hello Packet
4	10.684843	10.1.1.3	224.0.0.5	OSPF	82	Hello Packet
5	20.000487	10.1.1.2	224.0.0.5	OSPF	82	Hello Packet
6	20.685022	10.1.1.3	224.0.0.5	OSPF	82	Hello Packet
7	30.001077	10.1.1.2	224.0.0.5	OSPF	82	Hello Packet
8	30.685257	10.1.1.3	224.0.0.5	OSPF	82	Hello Packet

```
> Frame 1: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface -, id 0
> Ethernet II, Src: 36:0b:3f:e1:ca:85 (36:0b:3f:e1:ca:85), Dst: IPv4mcast_05 (01:00:5e:00:00:05)
> Internet Protocol Version 4, Src: 10.1.1.2, Dst: 224.0.0.5
> Open Shortest Path First
```

```
0000  01 00 5e 00 00 05 36 0b  3f e1 ca 85 08 00 45 c0   ··^···6· ?·····E·
0010  00 44 6e d1 00 00 01 59  5e c8 0a 01 01 02 e0 00   ·Dn····Y ^·······
0020  00 05 02 01 00 30 c0 a8  00 01 00 00 00 00 64 40   ·····0·· ······d@
```

wireshark_-KM6181.pcapng   Packets: 8 · Displayed: 8 (100.0%) · Dropped: 0 (0.0%)   Profile: Default

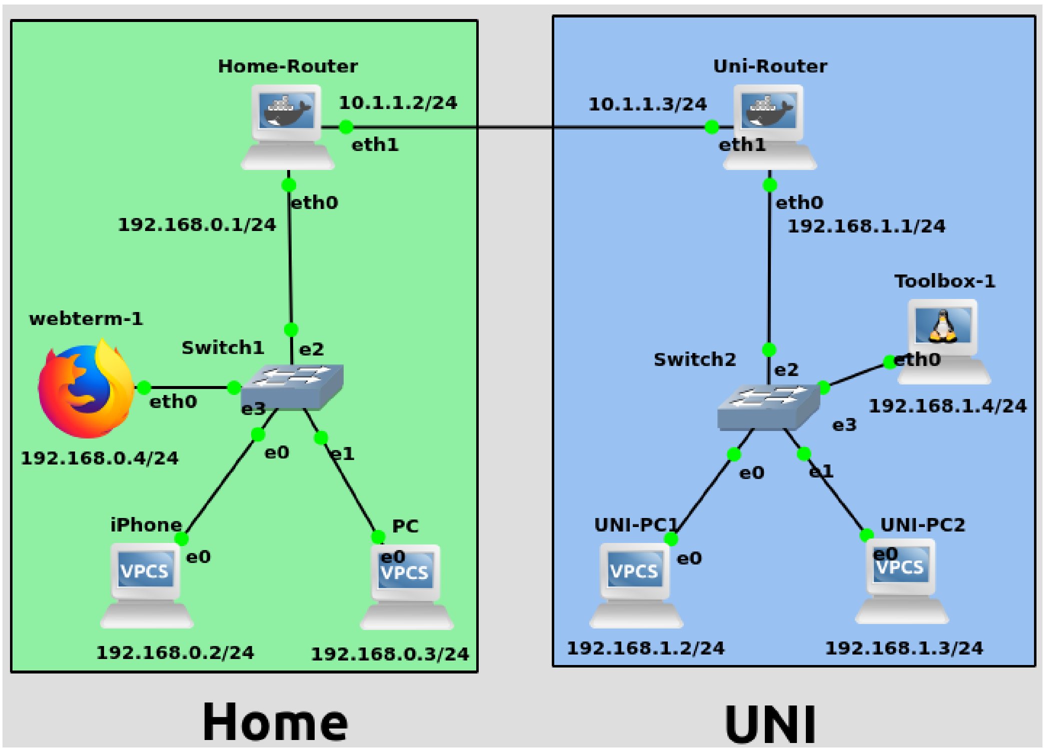
When you look at the Wireshark window, there is a wealth of information that was hidden behind the scenes as before. Carefully study the panels of Wireshark to learn more!

Reflect on the question below and write your brief answers to the report.

Question 1.3. What is the purpose of the “Hello” packets that you observe? Investigate it online in the context of the OSPF protocol. What is the destination address 224.0.0.5? What is it used for?

Task 1. Complete the original network shown in the beginning by adding 4 VPCSs. Send messages between the VPCSs. Observe the ping packets between the two LANs using Wireshark.

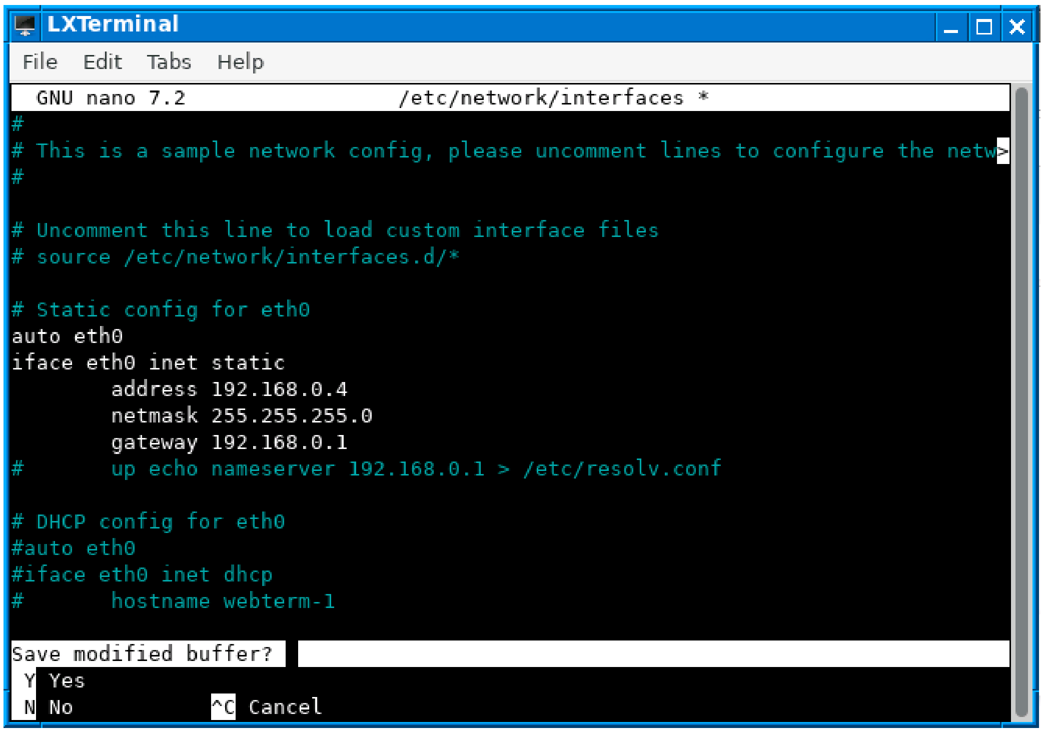
Task 2. Add a ***webterm*** to home LAN and a Toolbox to Uni LAN as shown below. The **Toolbox** appliance contains server-side software for secondary management of network devices: - www (nginx) - ftp (vsftpd) - tftp (tftpd) - syslog (rsyslog) - dhcp (isc-dhcpd) - snmp server (snmpd + snmptrapd).



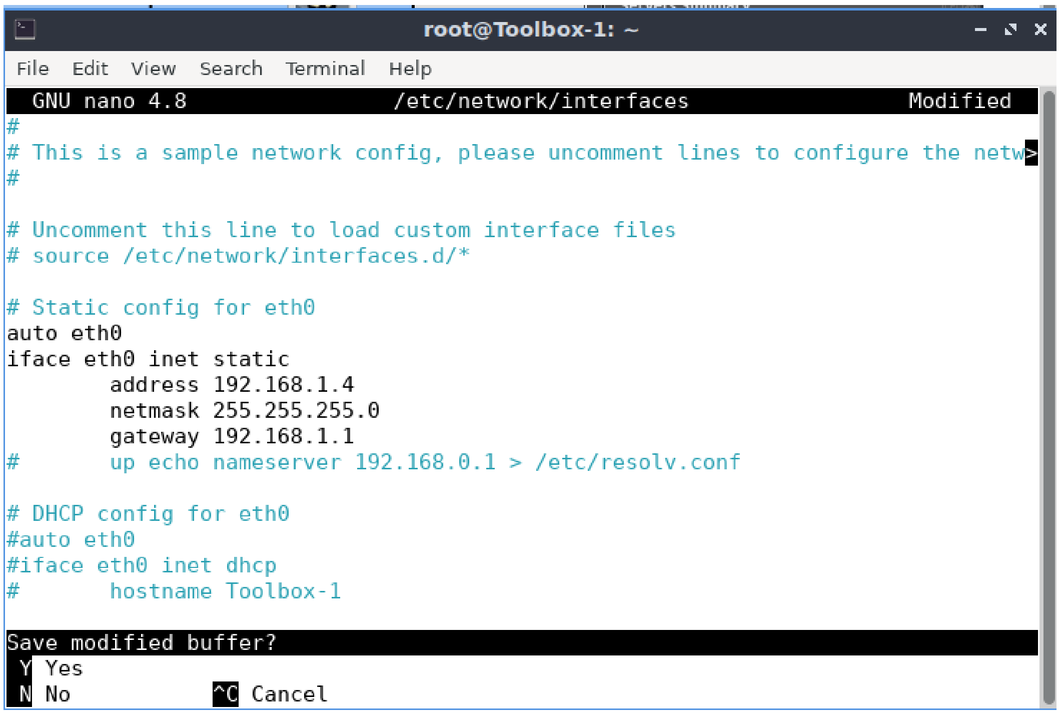
For both devices, go to their Terminal/Console and edit */etc/network/interfaces* file using nano editor.

nano /etc/network/interfaces

Restart the devices and check their IP addresses using the *ifconfig* command from their terminals.

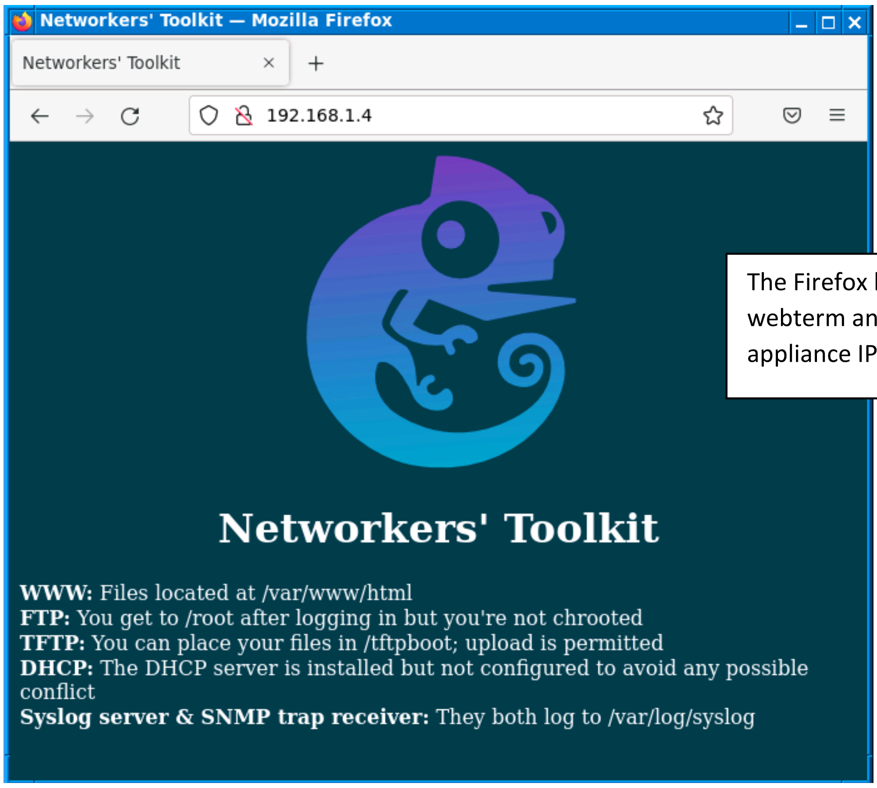


```
  GNU nano 7.2                    /etc/network/interfaces *
#
# This is a sample network config, please uncomment lines to configure the netw>
#

# Uncomment this line to load custom interface files
# source /etc/network/interfaces.d/*

# Static config for eth0
auto eth0
iface eth0 inet static
        address 192.168.0.4
        netmask 255.255.255.0
        gateway 192.168.0.1
#       up echo nameserver 192.168.0.1 > /etc/resolv.conf

# DHCP config for eth0
#auto eth0
#iface eth0 inet dhcp
#       hostname webterm-1

Save modified buffer?
 Y Yes
 N No           ^C Cancel
```



```
  GNU nano 4.8                /etc/network/interfaces                   Modified
#
# This is a sample network config, please uncomment lines to configure the netw>
#

# Uncomment this line to load custom interface files
# source /etc/network/interfaces.d/*

# Static config for eth0
auto eth0
iface eth0 inet static
        address 192.168.1.4
        netmask 255.255.255.0
        gateway 192.168.1.1
#       up echo nameserver 192.168.0.1 > /etc/resolv.conf

# DHCP config for eth0
#auto eth0
#iface eth0 inet dhcp
#       hostname Toolbox-1

Save modified buffer?
 Y Yes
 N No           ^C Cancel
```

Task 3. Start capturing packets using Wireshark between the two routers, then go to webterm and connect to the default webserver of the Toolbox as shown below. Then, observe the packets captured using Wireshark between the webterm and toolbox.

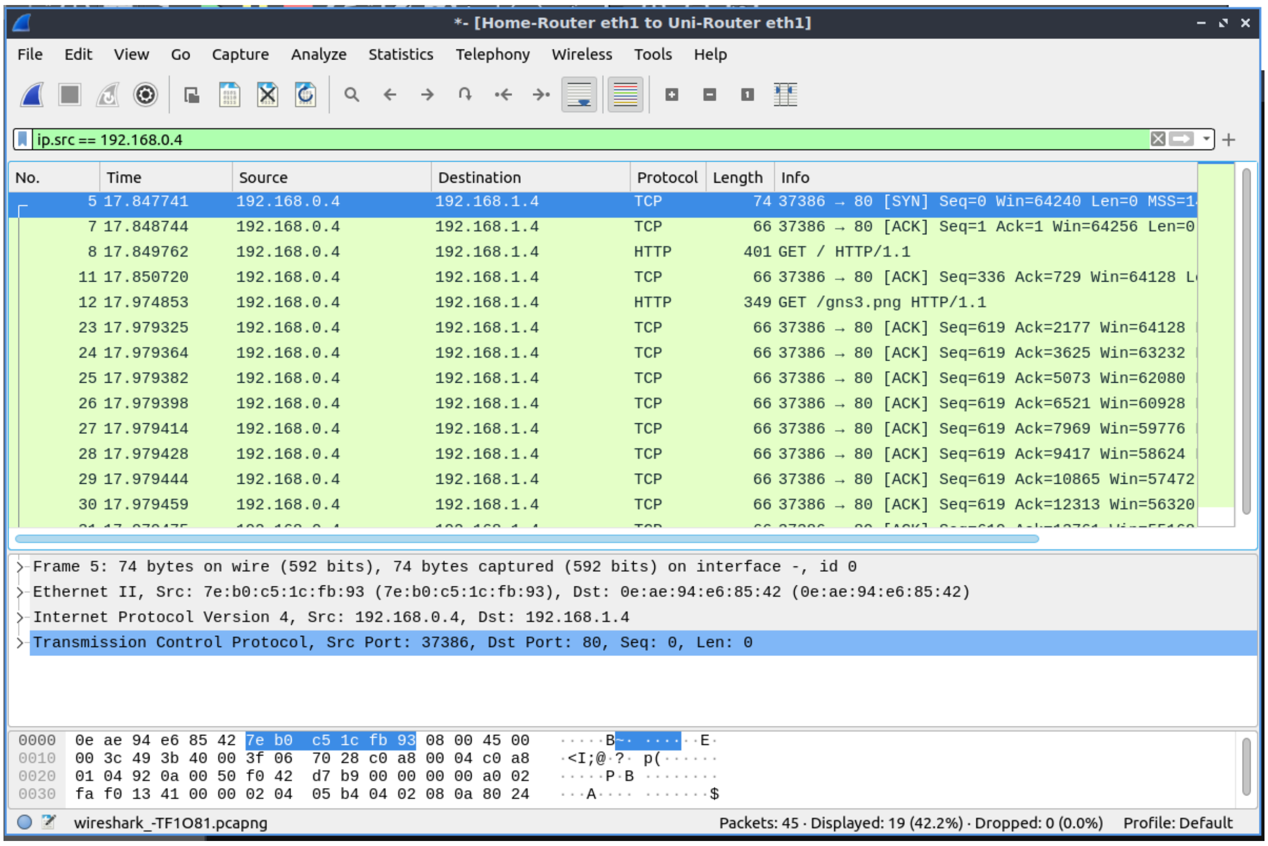


Networkers' Toolkit

WWW: Files located at /var/www/html
FTP: You get to /root after logging in but you're not chrooted
TFTP: You can place your files in /tftpboot; upload is permitted
DHCP: The DHCP server is installed but not configured to avoid any possible conflict
Syslog server & SNMP trap receiver: They both log to /var/log/syslog

The Firefox browser is running on webterm and connects to toolbox appliance IP address as shown.

Now you can observe multiple protocols in action and investigate their properties in Wireshark, see below.


*- [Home-Router eth1 to Uni-Router eth1]

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.src == 192.168.0.4

```
No.   Time       Source       Destination  Protocol Length Info
    5 17.847741  192.168.0.4  192.168.1.4  TCP         74 37386 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1
    7 17.848744  192.168.0.4  192.168.1.4  TCP         66 37386 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
    8 17.849762  192.168.0.4  192.168.1.4  HTTP       401 GET / HTTP/1.1
   11 17.850720  192.168.0.4  192.168.1.4  TCP         66 37386 → 80 [ACK] Seq=336 Ack=729 Win=64128 L
   12 17.974853  192.168.0.4  192.168.1.4  HTTP       349 GET /gns3.png HTTP/1.1
   23 17.979325  192.168.0.4  192.168.1.4  TCP         66 37386 → 80 [ACK] Seq=619 Ack=2177 Win=64128
   24 17.979364  192.168.0.4  192.168.1.4  TCP         66 37386 → 80 [ACK] Seq=619 Ack=3625 Win=63232
   25 17.979382  192.168.0.4  192.168.1.4  TCP         66 37386 → 80 [ACK] Seq=619 Ack=5073 Win=62080
   26 17.979398  192.168.0.4  192.168.1.4  TCP         66 37386 → 80 [ACK] Seq=619 Ack=6521 Win=60928
   27 17.979414  192.168.0.4  192.168.1.4  TCP         66 37386 → 80 [ACK] Seq=619 Ack=7969 Win=59776
   28 17.979428  192.168.0.4  192.168.1.4  TCP         66 37386 → 80 [ACK] Seq=619 Ack=9417 Win=58624
   29 17.979444  192.168.0.4  192.168.1.4  TCP         66 37386 → 80 [ACK] Seq=619 Ack=10865 Win=57472
   30 17.979459  192.168.0.4  192.168.1.4  TCP         66 37386 → 80 [ACK] Seq=619 Ack=12313 Win=56320
```

```
> Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface -, id 0
> Ethernet II, Src: 7e:b0:c5:1c:fb:93 (7e:b0:c5:1c:fb:93), Dst: 0e:ae:94:e6:85:42 (0e:ae:94:e6:85:42)
> Internet Protocol Version 4, Src: 192.168.0.4, Dst: 192.168.1.4
> Transmission Control Protocol, Src Port: 37386, Dst Port: 80, Seq: 0, Len: 0
```

```
0000  0e ae 94 e6 85 42 7e b0  c5 1c fb 93 08 00 45 00   ·····B~· ······E·
0010  00 3c 49 3b 40 00 3f 06  70 28 c0 a8 00 04 c0 a8   ·<I;@·?· p(······
0020  01 04 92 0a 00 50 f0 42  d7 b9 00 00 00 00 a0 02   ·····P·B ········
0030  fa f0 13 41 00 00 02 04  05 b4 04 02 08 0a 80 24   ···A···· ·······$
```

wireshark_-TF1O81.pcapng   Packets: 45 · Displayed: 19 (42.2%) · Dropped: 0 (0.0%)   Profile: Default

Reflect on the questions below and write your brief answers to the report.

Question 1.4. Which protocols do you observe in Wireshark during the communication between the web browser and server? Study the properties of the protocols and build connections between what you have learned in lectures and what you observe here.

Part 2 – Network Routing Algorithms in Python.

At this point, we switch gears a bit and will use the Python libraries to investigate network routing. For this task, you will use the Anaconda installation on your own (or lab) computer (not the virtual machine). The Jupyter Notebook you are given contains all the tasks and you can write your answers directly there. Your report will be a simple printout of that notebook (which you can do from your browser's print function).

Step 1 – Complete all the tasks in the provided Jupyter notebook.

Follow the instructions in the Jupyter Notebook. Now you have a better idea of how addressing is done on the Internet.

Part 3 – Sockets and Client-Server Communication

Step 1 – Socket Programming using Python.

First, let us have an overview of **sockets** using the Python implementation as an example. Other languages have similar but often more involved implementations of the same basics. Check the suggested readings (and internet resources) to answer the questions below.

- Nice tutorials
    - https://realpython.com/python-sockets/
    - https://www.binarytides.com/python-socket-programming-tutorial/
- Official Python socket documentation https://docs.python.org/3/library/socket.html
- A relevant book https://github.com/PacktPublishing/Learning-Python-Networking-Second-Edition

Reflect on the questions below and write your brief answers to the report.

Question 3.1. Briefly discuss socket family and type within the context of the socket() function. What are the differences between the SOCK_STREAM and SOCK_DGRAM socket types? In which context would you use one over the other?

Question 3.2. What is the purpose of the *socket.bind()* function in the Python socket module? What is the format of the address parameter in the AF_INET case? Would you use this function when implementing a server, a client or both? Justify your answer.

Question 3.3. Which functions of the socket API would you need to implement a simple server using a connectionless mode of communication? Which functions of the socket API would you need to implement a simple server using a connection-oriented mode of communication?

Question 3.4. Explain the difference between a blocking and non-blocking socket. Which one is used most in practice?

Step 2 – Simple Socket Server and Client.

For this task, you will use your favourite IDE (VSCode or Spyder) and a couple of terminals (within the IDE or separately).

You will write a simple client-server using Python sockets. First, write the client which is one the simplest programs in this workshop, and then the server.

The **Client** should:

- Identify the IPv4 address and port (e.g., 50007)
- Create the appropriate TCP socket.
- Connect to the Server
- Take a brief text message from the User.
- Send the message to the Server.
- Receive the response from the server and display it to the user.
- Wait for the next message unless the user enters “exit” as a message which should terminate the program.

The **Server** should:

- Create and bind the appropriate socket.
- Listen for a message from the user.
- Accept the connection.
- Receive the message and display it on the server screen.

- Send an acknowledgement response to the user with a timestamp.
- Wait for the next message.

Both client and server should politely close the sockets when they are done, whenever possible. Note that this is easy to achieve with the Python "with" statement, see e.g. https://builtin.com/software-engineering-perspectives/what-is-with-statement-python

Suggestions and Notes

- First write a simple echo version of the client and server. Debugging client-server programs can be tricky, so it is good to adopt a step-by-step approach.
- Once the simple version is working, you can create loops (e.g., using *while True*) to make both client and server persistent.
- You can stop the server (or in principle any Python script) using the (CTRL + C) keyboard shortcut.
- Some of the relevant objects from the Python socket library are:
  - socket.socket(), socket.bind(), socket.listen(), socket.accept(), socket.recv(), socket.sendall(), socket.connect().
- If you get stuck, the solutions are kind of in front of you. However, don't simply copy-paste and don't trust ChatGPT that much 😊
- If you wish to draw fancy diagrams [entirely optional and only if you have extra time], have a look at
  - https://plantuml.com/state-diagram
  - https://plantuml.com/sequence-diagram

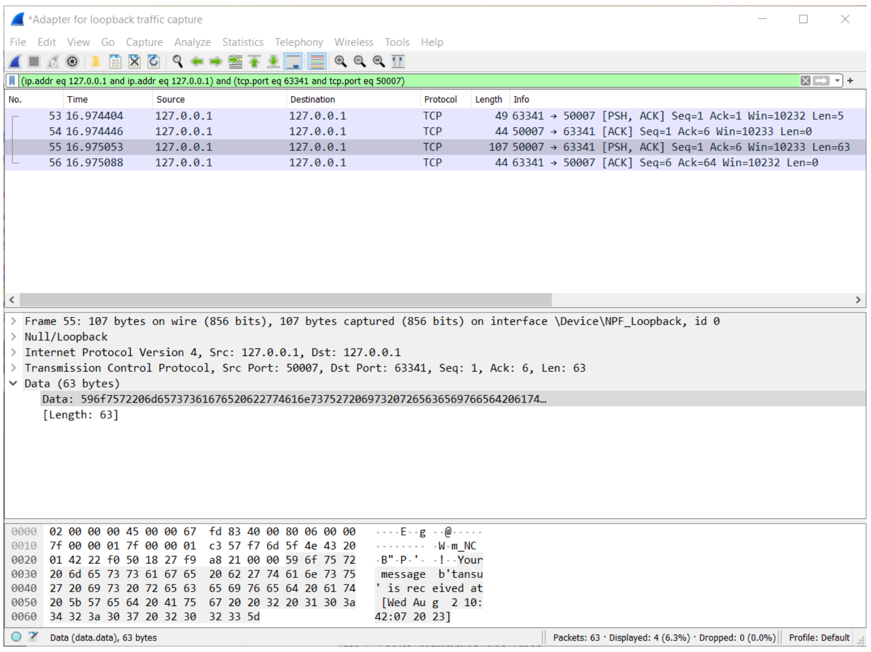
Reflect on the questions below and write your brief answers to the report.

Question 3.5. Write at least one working client and one server script. Comment them well. Reflect on how they work and put them into context of the lectures.

Question 3.6. Use Wireshark to observe the traffic between your client and server (using loopback traffic capture). You can use 127.0.0.1 (localhost) to create a filter. Which protocol do you observe? Why?

Question 3.7. Draw simple finite-state-machine (FSM) diagrams of your client and server. Handwritten diagrams are perfectly fine.

Wireshark screenshot (as an example)


*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
(ip.addr eq 127.0.0.1 and ip.addr eq 127.0.0.1) and (tcp.port eq 63341 and tcp.port eq 50007)

No.   Time        Source      Destination  Protocol  Length  Info
   53 16.974404   127.0.0.1   127.0.0.1    TCP        49 63341 → 50007 [PSH, ACK] Seq=1 Ack=1 Win=10232 Len=5
   54 16.974446   127.0.0.1   127.0.0.1    TCP        44 50007 → 63341 [ACK] Seq=1 Ack=6 Win=10233 Len=0
   55 16.975053   127.0.0.1   127.0.0.1    TCP       107 50007 → 63341 [PSH, ACK] Seq=1 Ack=6 Win=10233 Len=63
   56 16.975088   127.0.0.1   127.0.0.1    TCP        44 63341 → 50007 [ACK] Seq=6 Ack=64 Win=10232 Len=0

> Frame 55: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 50007, Dst Port: 63341, Seq: 1, Ack: 6, Len: 63
v Data (63 bytes)
    Data: 596f7572206d657373616765206227746616e7375272069732072656365697665642061 74…
    [Length: 63]

0000  02 00 00 00 45 00 00 67  fd 83 40 00 80 06 00 00   ····E··g ··@·····
0010  7f 00 00 01 7f 00 00 01  c3 57 f7 6d 5f 4e 43 20   ········ ·W·m_NC
0020  01 42 22 f0 50 18 27 f9  a8 21 00 00 59 6f 75 72   ·B"·P·'· ·!··Your
0030  20 6d 65 73 73 61 67 65  20 62 27 74 61 6e 73 75    message  b'tansu
0040  27 20 69 73 20 72 65 63  65 69 76 65 64 20 61 74   ' is rec eived at
0050  20 5b 57 65 64 20 41 75  67 20 20 32 20 31 30 3a    [Wed Au g  2 10:
0060  34 32 3a 30 37 20 32 30  32 33 5d                  42:07 20 23]

Data (data.data), 63 bytes        Packets: 63 · Displayed: 4 (6.3%) · Dropped: 0 (0.0%)    Profile: Default
```

Appendix – A Note on bytes and strings in Python

In communication and computing systems, we naturally distinguish between text and binary data. The classical character encoding standard for representing text has been the **American Standard Code for Information Interchange**, or ASCII, which supports mainly western or Latin languages. For example, the lowercase letter "**i**" is represented in the ASCII encoding by binary 1101001 = hexadecimal 69 = decimal 105. See Wikipedia for more on ASCII.

ASCII was not a problem as long as everybody using computers speaks English or German, or French. But what about the rest of the world and other languages? As time passed, the limitations of ASCII became obvious, and Unicode was born**. Unicode** is a computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems. For example, UTF-8 is a character encoding capable of encoding all 1,112,064 valid code points in Unicode using one to four 8-bit bytes.

Python 3 has finally moved to modern times and now all string objects in Python3 are encoded by default in UTF-8. This is great if you wish to write in Chinese, or Indian, or Greek! But how to convert strings to byte objects and vice versa? The answer is simple:

Use **encode()** to convert string to a byte stream and **decode()** to do the reverse!

```
>>> "tansu".encode()

b'tansu'   # this is now a byte stream. The output shows the ascii representation only for convenience!

>>> b'tansu'[0]  # in reality the byte object is nothing but a stream (but not an array!) of bytes

116

>>> b'tansu'.decode()  # decoding back to a (UTF-8) string

'tansu'
```

This is an involved topic and can be quite confusing sometimes. For more information, see
http://pythoncentral.io/encoding-and-decoding-strings-in-python-3-x/

http://www.diveintopython3.net/strings.html

or any other alternative Python 3 sources.

Finally, the relationship to sockets:

Socket implementation in Python3 sends and receives only **bytes** objects. Therefore, if you have a text message to send, you should appropriately encode/decode the strings before sending and after receiving, respectively! Try encoding/decoding in Python console and ask your demonstrator if you need help.

Workshop Report Guidelines:

You should complete the workshop tasks and answer the questions within the respective session!

It is mandatory to follow all of the submission guidelines given below:

1. All figures produced and programs written should be uploaded to the right place in LMS by the announced deadline.
2. Filenames should be “ELEN90061 Workshop **W**: StudentID1-StudentID2 of session Day-Time”, where

   W refers to the workshop number,

   StudentID1-StudentID2 are your student numbers, and
   Day-Time is your session day and time, e.g. Tue-14.
3. Submit your report in 3 separate files as follows:
   - A pdf file converted e.g. from a Word file that contains the answers to Parts 1 and 3.
   - A pdf file printed directly from the Jupyter Notebook as output of Part 2.
   - A zip file that contains all the codes (e.g. Matlab m files or Python scripts for Parts 1 and 3) **and** the Jupyter Notebook ipynb file (from Part 2).
4. You should **NOT** zip all your files into one. You **MUST** submit your reports as separate PDFs to pass the plagiarism check. Hence, we require 3 separate files.
5. Answers to questions, simulation results and diagrams should be embedded in your reports.
6. One report submission per group.
7. Please check with your demonstrator in case of any questions.