

Based on
Mastering Networks - An Internet Lab Manual
by Jörg Liebeherr and Magda Al Zarki

Adapted for
'Labo Computernetwerken'
by Johan Bergs, Nicolas Letor, Michael Voorhaen and Kurt Smolderen

Completed by
Stan Draulans Sam Mylle Federico Quin Group 3

March 12, 2017

Lab 2

Single-Segment IP Networks

What you will learn in this lab:

- How to capture and filter network traffic
- How to configure a network interface for IP networking
- How to access IP statistics and settings with the netstat command
- How ARP works
- How hackers snoop passwords from the network

Prelab 2

New commands

Read the manual pages of the following commands at <http://manpages.ubuntu.com/> for the operating system version “trusty 14.04 LTS”:

- arp
- ifconfig
- netstat
- tcpdump

IP Addresses

Read the article “Understanding IP Addressing: Everything You Ever Wanted To Know” by Chuck Semeria, at <http://holdenweb.com/static/docs/3comip.pdf>

Wireshark capture and display filters

Go to the Wireshark website and read about capture filters and display filters for Wireshark:

- <http://wiki.wireshark.org/CaptureFilters>
- http://www.wireshark.org/docs/wsug_html_chunked/ChWorkBuildDisplayFilterSection.html.

Prelab Questions

Question 1)

Write the syntax for an `ifconfig` command that sets the IP address of the interface `eth0` to 128.143.2.3/16 with broadcast address 128.143.255.255.

```
| ifconfig eth0 128.143.2.3 netmask 255.255.0.0 broadcast 128.143.255.255
```

Question 2)

Write the syntax of a `tcpdump` command that captures packets containing IP datagrams with source or destination IP address equal to 10.0.1.12.

```
| tcpdump -n "(src host 10.0.1.12 or dst host 10.0.1.12) and ip"
```

Question 3)

Write the syntax of a `tcpdump` command that captures packets containing ICMP messages with source or destination IP address equal to 10.0.1.12.

```
| tcpdump -n "(src host 10.0.1.12 or dst host 10.0.1.12) and icmp"
```

Question 4)

Write the syntax of a `tcpdump` command that captures packets containing IP datagrams between two hosts with IP addresses 10.0.1.11 and 10.0.1.12, both on interface `eth1`.

```
| sudo tcpdump -i eth1 -n host 10.0.1.11 and host 10.0.1.12 and ip
```

Question 5)

Write a `tcpdump` filter expression that captures packets containing TCP segments with source or destination IP address equal to 10.0.1.12.

```
| sudo tcpdump -n host 10.0.1.12 and tcp
```

Question 6)

Write a `tcpdump` filter expression that, in addition to the constraints in Question 5, only captures packets using port number 23.

```
| tcpdump -n host 10.0.1.12 and tcp and port 23
```

Question 7)

Write the syntax for a `wireshark` command with capture filter so that all IP datagrams with source or destination IP address equal to 10.0.1.12 are recorded.

```
| ip && (ip dst 10.0.1.12 || ip src 10.0.1.12)
```

Question 8)

Write the syntax for a Wireshark display filter that shows IP datagrams with destination IP address equal to 10.0.1.50 and frame sizes greater than 400 bytes.

```
| ip && ip.dst == 10.0.1.50 && ip.len > 400
```

Question 9)

Write the syntax for a Wireshark display filter that shows packets containing ICMP messages with source or destination IP address equal to 10.0.1.12 and frame numbers between 15 and 30.

```
| icmp && (ip.dst == 10.0.1.12 || ip.src == 10.0.1.12 ) && icmp.seq > 15 && icmp.seq < 30
```

Question 10)

Write the syntax for a Wireshark display filter that shows packets containing TCP segments with source or destination IP address equal to 10.0.1.12 and using port number 23.

```
| tcp && (ip.dst == 10.0.1.12 || ip.src == 10.0.1.12 ) && tcp.port == 10
```

Question 11)

Write a Wireshark capture filter expression for Question 10.

```
| tcp && (ip dst 10.0.1.12 || ip src 10.0.1.12 ) && tcp port 10
```

Lab 2

In Lab 2 you become acquainted with IP configuration issues on a single Ethernet segment. The lab also exposes you to advanced use of `tcpdump` and Wireshark.

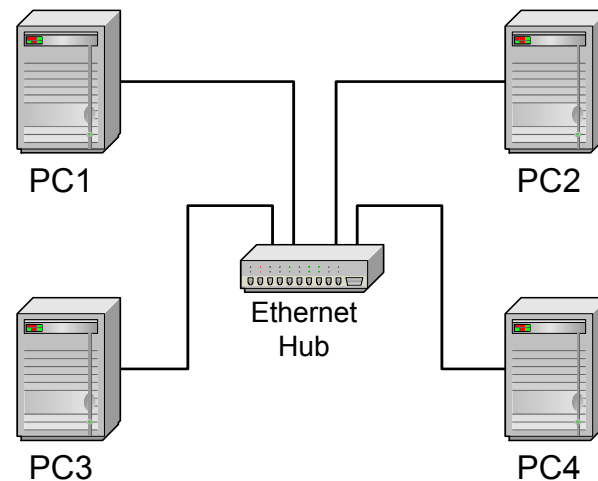


Figure 2.1: Network configuration for Lab 2.

The setup for this lab is identical as in Lab 1. All Linux PCs are connected to the same Ethernet segment by an Ethernet hub as shown in Figure 2.1. The IP addresses for the Linux PCs are configured as shown in Table 2.1 below. Configure `eth0` for each of the PCs. e.g. for PC1 use the following command.

```
PC1% ifconfig eth0 10.0.1.11 netmask 255.255.255.0 broadcast 10.0.255 up
```

As an alternative, you can use the `ip` command:

```
PC1% ip addr add 10.0.1.11/24 dev eth0
PC1% ip link set dev eth0 up
```

Linux PC	IP Addresses of Ethernet Interface eth0
PC1	10.0.1.11/24
PC2	10.0.1.12/24
PC3	10.0.1.13/24
PC3	10.0.1.14/24

Table 2.1: IPv4 addresses for Lab 2

Using filters in tcpdump

In the first part of the lab, you explore `tcpdump` in more detail. In particular, you learn how to write filter expressions so that `tcpdump` monitors only selected traffic flows on the network. See 2.1 for more details on the use of filters in `tcpdump`.

Exercise 1. Writing filter expressions for tcpdump

In this exercise, you explore the use of simple filter expressions with the `tcpdump` command. Save the output for your lab report.

1. On PC1, execute a `tcpdump` command with a filter that prints all packets with PC2 as source or destination. This command is the answer to Question 2 from the Prelab. Save the output of this `tcpdump` session to a file using the `tee` or `tail` commands discussed in Lab 1.



As in Lab 1, always use the `-n` option (i.e. `tcpdump -n`) to avoid that `tcpdump` tries to resolve hostnames.

2. In another terminal, issue a ping command to PC2 by typing `ping -c 5 10.0.1.12` on PC1 and observe the output. Recall that the ping command to a host triggers the transmission of an ICMP Echo Request. The destination host responds with an ICMP Echo Reply message.
3. Repeat steps 1 - 2 above. In addition to the existing filter, set the filter so that only ICMP messages are captured. This command is the answer to Question 3 from the Prelab.



Make sure to include the saved data in your lab report as they are part of your evaluation!

Using filters in Wireshark

In this part of the lab, you experiment with filter expressions using the `wireshark` command. Recall that Wireshark has two types of filters: capture filters and display filters.



There are several command line options that can be assigned when starting the `wireshark` command:

- **Capture Filters:** A capture filter specifies the traffic to be captured by the Wireshark tool. A capture filter expression can be specified from the command line using the `-f` option or using the Wireshark GUI, under the “Capture:Start” menu. The syntax for specifying the filter expression is the same syntax as used by `tcpdump`.
- **Display Filters:** By default, Wireshark displays all captured packets. With a display filter, just those packets, which meet the requirements of the filter, are displayed. The display filter cannot be set from the command line. It must be entered in the “Filter” window at the bottom of the GUI. The syntax for setting the display filter is different from the syntax for setting a capture filter.
- **Setting an interface:** When you run Wireshark on a host with multiple network interfaces, you may specify the interface with the `-i` argument. For example, to start Wireshark to capture traffic on interface `eth1`, type

```
| wireshark -i eth1
```

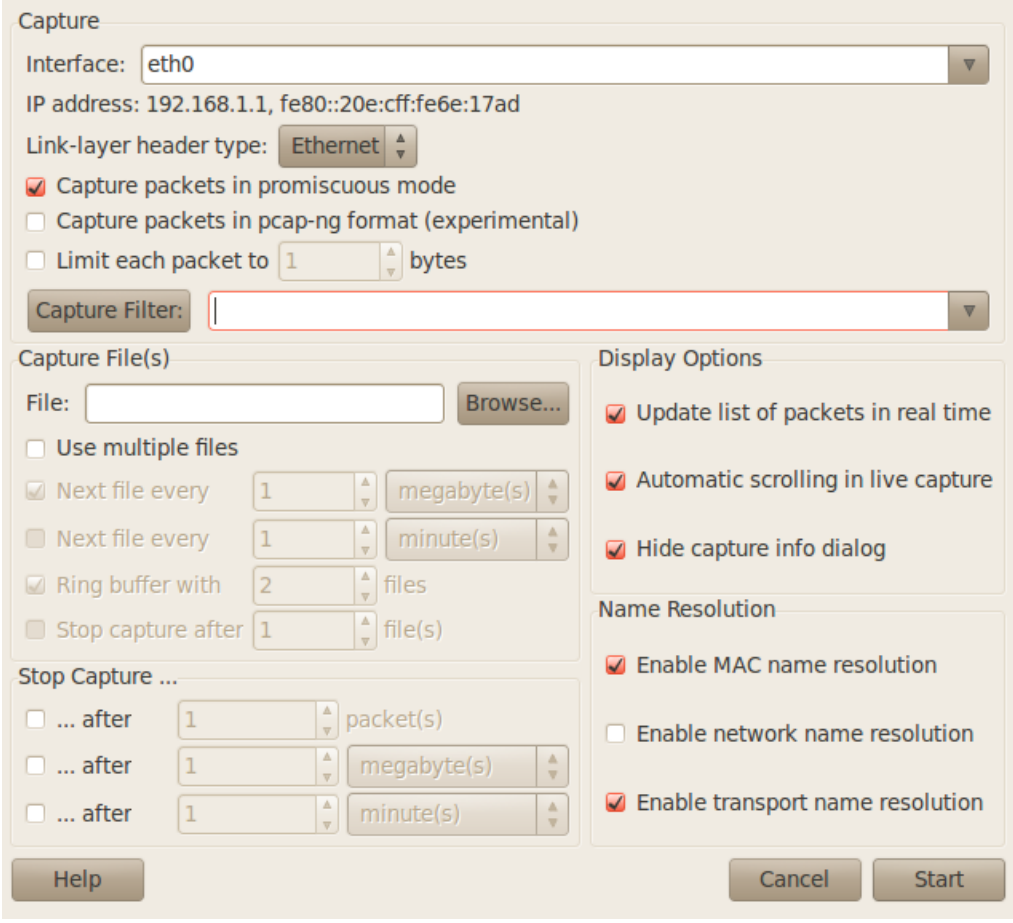
If you do not specify an interface, the default is `eth0`. Alternatively, you can change the interface using the Wireshark GUI, under the “Capture:Start” menu.

Exercise 2-A. Setting capture filters in Wireshark

This exercise is a review of the traffic capture capabilities of Wireshark. As a new feature, you are introduced to the notion of capture filters.

1. Start Wireshark on PC1 and set the same capture preferences as in Lab 1 and as shown again in Figure 2.2 for your convenience. You should always set these same preferences for all your experiments.

Selecting capture preferences in wireshark



- Select *eth0* in “Interface”.
- Select “Capture packets in promiscuous mode”.
- Select “Update list of packets in real time”.
- Select “Automatic scrolling in live capture”.
- Unselect “Enable MAC name resolution”.
- Unselect “Enable network name resolution”.
- Unselect “Enable transport name resolution”.

Figure 2.2: General capture settings for Wireshark

2. Setting a capture filter: In the window “Capture Preferences”, set a filter so that all packets that contain the IP address of PC2 are recorded. The filter is set in the “Filter” box under “Capture Preferences” (see Figure 2.2). The required filter expression is the answer to Question 7 from the Prelab.
3. Start the capture by clicking “OK” in the “Capture Preferences” window.
4. In another terminal window of PC1, issue a `ping` command to PC2

```
| PC1% ping -c 2 10.0.1.12
```

5. Stop the capture process of Wireshark.
6. Save the results of the capture. This is done by selecting “Print” in the “File” menu as described in Lab 1. (As instructed in Lab 1, unless asked to save the details of captured frames, selecting the summary option is usually sufficient.)



Make sure to include the saved data in your lab report as they are part of your evaluation!

Exercise 2-B. Working with display filters

Next you set display filters, which allow you to select a subset of the captured data for display in the main window of Wireshark.

1. In the Wireshark main window on PC1 from Exercise 2-A, set the display options as listed below. You can find the display options in the “Capture Options” window (select “Start” in the “Capture” menu)(see Figure 2.2):
 - Select “Update list of packets in real time”.
 - Select “Automatic Scrolling in live capture”.
 - Unselect “Enable MAC name resolution”.
 - Unselect “Enable network name resolution”.
 - Unselect “Enable transport name resolution”.
2. Setting a display filter: Type the desired display filter in the field next to the “Filter” box, which is located at the bottom of the Wireshark main window, as shown in Figure 2.3. Click the “Reset” button next to the “Filter” box to clear any existing filter:



Figure 2.3: Filter box for setting display filters.

Enter a display filter so that all IP datagrams with destination IP address 10.0.1.12 are shown. Refer to Question 8 from the Prelab.

3. Observe the changes in the display panel of Wireshark. Only packets with 10.0.1.12 in the IP destination address field are now being displayed.
4. Save the displayed data, by selecting “File:Print”. Note that the “Print” command only saves packets that are currently being displayed. If a display filter is used, the saved data is limited to the packets that match the display filter.
5. Repeat the above exercise with a display filter that lists only IP datagrams with source IP address equal to 10.0.1.12. Save the results.



Make sure to include the saved data in your lab report as they are part of your evaluation!

Exercise 2-C. More complex capture and display filters

In this exercise, you learn how to use more sophisticated filters to restrict the packets being captured and displayed.

1. Start Wireshark on PC1 and start to capture traffic using the same settings as in Exercise 2-A. Do not set any capture or display filters!

2. From a new terminal on PC1, execute the `ping` command for PC2

```
| PC1% ping -c 5 10.0.1.12
```

3. At the same time, start a Telnet session from PC1 to PC2 in another terminal by typing

```
| PC1% telnet 10.0.1.12
```

and log in as *telecomlabo*. After you logged in successfully to PC2, logout with the command `exit`.

4. Stop the traffic capture of wireshark.
5. Apply a set of display filters to the captured traffic and save the output to a text file. Select the option "Print summary" in the "Print" window.
 - Display packets that contain ICMP messages with the IP address of PC2 either in the IP destination address or IP source address. Refer to Question 9 from the Prelab. Save the output.
 - Display packets that contain TCP traffic with the IP address of PC2 either in the IP destination address or IP source address. Refer to Question 10 from the Prelab. Save the output.
 - Display packets that, in addition to the constraints in the previous filter expression, use the port number 23. Refer to Question 10 from the Prelab. Save the output.



Make sure to include the saved data in your lab report as they are part of your evaluation!

ARP - Address Resolution Protocol

This part of the lab explores the operation of the Address Resolution Protocol (ARP) which resolves a MAC address for a given IP address. The lab exercises use the Linux command `arp` for displaying and manipulating the contents of the ARP cache. The ARP cache is a table that holds entries of the form <IP address, MAC address>.



The most common uses of the `arp` command are as follows:

`arp -a`

Displays the content of the ARP cache.

`arp -d <IPAddress>`

Deletes the entry with IP address `IPAddress`.

`arp -s <IPAddress><MAC_Address>`

Adds a static entry to the ARP cache which is never overwritten by network events. The MAC address is entered as a 6 hexadecimal bytes separated by colons.

Example: `arp -s 00:02:2D:0D:68:C1`



Time-outs in the ARP cache:

The entries in an ARP cache have a limited lifetime. Entries are deleted unless they are refreshed. The typical lifetime of an ARP entry is 2 minutes, but much longer lifetimes (up to 20 minutes) have been observed. You may want to verify when your Linux system does remove ARP entries automatically after a certain amount of time.



Refreshing the ARP cache:

In Linux, you will observe that occasionally, a host sends out ARP requests to interfaces that are already in the ARP cache. Example: Suppose that a host with IP address 10.0.1.12 has an ARP cache entry: "<10.0.1.11>is-at <00:02:83:39:2C:42>". Then, this host occasionally sends a unicast ARP Request to MAC address 00:02:83:39:2C:42 of the form "Who has 10.0.1.11? Tell 10.0.1.12" to verify that the IP address 10.0.1.11 is still present before deleting the entry from the ARP cache.

Exercise 3-A. A simple experiment with ARP

1. On PC1, view the ARP cache with `arp -a` and delete all entries with the `-d` option.
2. Start Wireshark on PC1 with a capture filter set to the IP address of PC2.
3. Issue a ping command from PC1 to PC2:

```
| PC1% ping -c 2 10.0.1.12
```

Observe the ARP packets in the Wireshark window. Explore the MAC addresses in the Ethernet headers of the captured packets. Direct your attention to the following fields:

- The destination MAC address of the ARP Request packets.
 - The Type field in the Ethernet headers of ARP packets and ICMP messages.
4. View the ARP cache again with the command `arp -a`. Note that ARP cache entries get refreshed/ deleted fairly quickly (approx. 2 minutes).
 5. Save the results of Wireshark to a pcap dump file.

Use the saved data to answer to the following questions:

Question 3.A.1)

What is the destination MAC address of an ARP Request packet?

The destination MAC-address for an ARP request packet is ff:ff:ff:ff:ff:ff (the broadcast address).

Question 3.A.2)

What are the different values of the Type field in the Ethernet headers that you observed?

The observed types are 0x0800 (IPv4) and 0x0806 (ARP).

Question 3.A.3)

Use the captured data to discuss the process in which ARP acquires the MAC address for IP address 10.0.1.12.

ARP sends a broadcast message, requesting the MAC Address of the device which has 10.0.1.12 as IP address (packet 1). The device whose IP matches the queried one responds with its MAC-address (packet 2).

1	No.	Time	Source	Destination	Protocol	Length
	Info					
	1	0.000000000	68:05:ca:36:33:a0	ff:ff:ff:ff:ff:ff	ARP	42
		Who has 10.0.1.12? Tell 10.0.1.11				
3	Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0					
5	Ethernet II, Src: 68:05:ca:36:33:a0, Dst: ff:ff:ff:ff:ff:ff					
	Address Resolution Protocol (request)					
7	No.	Time	Source	Destination	Protocol	Length
	Info					
9	2	0.000320047	68:05:ca:36:31:f0	68:05:ca:36:33:a0	ARP	60
		10.0.1.12 is at 68:05:ca:36:31:f0				
11	Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0					
	Ethernet II, Src: 68:05:ca:36:31:f0, Dst: 68:05:ca:36:33:a0					
13	Address Resolution Protocol (reply)					

All packets can be found at "traces/3-A.pcapng"

Exercise 3-B. Matching IP addresses and MAC addresses

Identify the MAC addresses of all interfaces connected to the network, and enter them in Table 2.2. You can obtain the MAC addresses from the ARP cache of each PC. You can fill up the ARP cache at a host, by issuing a ping command from that host to every other host on the network. Alternatively, you can obtain the MAC addresses from the output of the `ifconfig -a` command explained in Part 5.

Question 3.B.1)

Include the completed Table 2.2 in your lab report.

Linux PC	IP Address of eth0	MAC Address of eth0
PC1	10.0.1.11/24	68:05:ca:36:33:a0
PC2	10.0.1.12/24	68:05:ca:36:31:f0
PC3	10.0.1.13/24	68:05:ca:36:39:c7
PC4	10.0.1.14/24	68:05:ca:39:e1:2f

Linux PC	IP Address of eth0	MAC Address of eth0
PC1	10.0.1.11/24	
PC2	10.0.1.12/24	
PC3	10.0.1.13/24	
PC3	10.0.1.14/24	

Table 2.2: IP and MAC addresses.

Exercise 3-C. ARP requests for a non-existing address

Observe what happens when an ARP Request is issued for an IP address that does not exist.

1. On PC1, start Wireshark with a capture filter set to capture packets that contain the IP address of PC1:

```
| PC1% wireshark -f 'host 10.0.1.11'
```

2. Establish a Telnet session from PC1 to 10.0.1.10 (Note that this address does not exist on this network)

```
| PC1% telnet 10.0.1.10
```

Observe the time interval and the frequency with which PC1 transmits ARP Request packets. Repeat the experiment a number of times to discover the pattern.

3. Save the captured output.

Question 3.C.1)

Using the saved output, describe the time interval between each ARP Request packet issued by PC1. Describe the method used by ARP to determine the time between retransmissions of an unsuccessful ARP Request. Include relevant data to support your answer.

ARP stops after 5 retransmissions (so 6 messages in total). The time between two consecutive ARP messages is 1 second. The packets can be seen at the file "traces/3-C.pcapng".

Question 3.C.2)

Why are ARP Request packets not transmitted (i.e. not encapsulated) as IP packets? Explain your answer.

Because IP is an end-to-end protocol. ARP is a node-to-node protocol. They both work on different network layers.

The netstat command

The Linux command `netstat` displays information on the network configuration and activity of a Linux system, including network connections, routing tables, interface statistics, masquerade connections, and multicast memberships. The following exercise explores how to use the `netstat` command to extract different types of information about the network configuration of a host.



The most common uses of the `netstat` command are as follows:

`netstat -i`

Displays a table with statistics of the currently configured network interfaces.

`netstat -rn`

Displays the kernel routing table. The `-n` option forces `netstat` to print the IP addresses. Without this option, `netstat` attempts to display the hostnames.

`netstat -an; netstat -tan; netstat -uan`

Displays the active network connections. The `-a` option displays all active network connections, the `-ta` option displays only information on TCP connections, and the `-ua` option displays only information on UDP traffic. Omitting the `-n` options prints hostnames and names of servers, instead of IP addresses and ports numbers.

`netstat -s`

Displays summary statistics for each protocol that is currently running on the host.

Exercise 4. Basic usage of the netstat command

On PC1, try the different variations of the `netstat` command listed above and save the output to a file.

1. Display information on the network interfaces by typing

```
| PC1% netstat -in
```

2. Display the content of the IP routing table by typing

```
| PC1% netstat -rn
```

3. Display information on TCP and UDP ports that are currently in use by typing

```
| PC1% netstat -a
```

4. Display the statistics of various networking protocols by typing

```
| PC1% netstat -s
```



The values of the statistics displayed by some of the `netstat` commands are reset each time a host is rebooted.

Question 4.1)

Attach the saved output to your report. Using the saved output, answer the following questions.

Kernel	Interface	table										
lface	MTU	Met	RX-OK	RX-ERR	RX-DRP	RX-OVR		TX-OK	TX-ERR	TX-DRP	TX-OVR	Flg
eth0	1500	0	986	0	0	0		1209	0	0	0	BMRU
lo	65536	0	63	0	0	0		63	0	0	0	LRU

traces/4.1.txt

Question 4.1.a)

What are the network interfaces of PC1 and what are the MTU (Maximum Transmission Unit) values of the interfaces?

PC1 has 2 network interfaces: eth0 and lo.

The Maximum Transmission Unit of the interfaces is 1500 and 65536 respectively.

Question 4.1.b)

How many IP datagrams, ICMP messages, UDP datagrams, and TCP segments has PC1 transmitted and received since it was last rebooted?

The PC has:

- sent 1175 and received 1105 IP datagrams
- sent and received 59 ICMP messages
- sent 134 and received 196 UDP datagrams
- sent 992 and received 846 TCP segments

Question 4.2)

Explain the role of interface lo, the loopback interface. In the output of "netstat -in", why are the values of RX-OK (packets received) and TX-OK (packets transmitted) different for interface "eth0" but identical for interface lo?

eth0 sends packets on the network to some other device. Not every packet needs a reply, so there's no guarantee that the pc receives as many packets as it sends. The lo interface sends packets to the device itself, meaning it receives the same amount of packets as it sends.

Configuring IP interfaces in Linux

The `ifconfig` command is used to configure parameters of network interfaces on a Linux system, such as enabling and disabling of interfaces and setting the IP address. The `ifconfig` command is usually run when a system boots up. In this case, the parameters of the commands are read from a file. Once the Linux system is running, the `ifconfig` command can be used to modify the network configuration parameters.



The most common uses of the `ifconfig` command to query the status of network interfaces are as follows:

`ifconfig`

Displays the configuration parameters of all active interfaces.

`ifconfig -a`

Displays the configuration parameters of all network interfaces, including the inactive interfaces.

`ifconfig <interface>`

Displays the configuration parameters of a single interface. For example, `ifconfig eth0` displays information on interface `eth0`.



There are numerous options for configuring a network interface with `ifconfig`. The following example shows how to enable and disable an interface and how to change the IP configuration.

`ifconfig eth0 down`

Disables the `eth0` interface. No traffic is sent or received on a disabled interface.

`ifconfig eth0 up`

Enables the `eth0` interface.

`ifconfig eth0 10.0.1.8 netmask 255.255.255.0 broadcast 10.0.1.255`

Assigns interface `eth0` the IP address 10.0.1.8/24 and a broadcast address of 10.0.1.255. The interface should be disabled before a new IP address is assigned, and should be enabled after the IP address has been modified.

`ifconfig eth0 down 10.0.1.8 netmask 255.255.255.0 broadcast 10.0.1.255 up`

Performs all three commands above in sequence. Interface `eth0` is disabled, an IP address and a broadcast address are assigned, and the interface is enabled.

`ifconfig eth0 mtu 500`

Sets the MTU of interface `eth0` to 500 bytes.

Exercise 5. Changing the IP address of an interface

Use the `ifconfig` command to modify the IP address of the `eth0` interface of PC4.

1. On PC4, run `ifconfig -a` and save the output.
2. Change the IP address of interface `eth0` of PC4 to 10.0.1.11/24.
3. Run `ifconfig -a` again and save the output.

Question 5.1)

Attach the saved files to your report and explain the fields of the `ifconfig` output.

Before:

```

2  eth0      Link encap:Ethernet  HWaddr 68:05:ca:39:e1:2f
      inet addr:10.0.1.14  Bcast:10.0.0.255  Mask:255.255.255.0
4      inet6 addr: fe80::6a05:caff:fe39:e12f/64  Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:419  errors:0  dropped:0  overruns:0  frame:0
6      TX packets:94  errors:0  dropped:0  overruns:0  carrier:0
      collisions:0 txqueuelen:1000
8      RX bytes:55178 (55.1 KB)  TX bytes:10131 (10.1 KB)
      Interrupt:19  Memory:f06c0000-f06e0000
10
12  eth1      Link encap:Ethernet  HWaddr 68:05:ca:39:e1:32
      BROADCAST MULTICAST  MTU:1500  Metric:1
      RX packets:0  errors:0  dropped:0  overruns:0  frame:0
14     TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
      collisions:0 txqueuelen:1000
16     RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
      Interrupt:16  Memory:f05c0000-f05e0000
18
20  internet  Link encap:Ethernet  HWaddr d0:50:99:55:a9:0c
      BROADCAST MULTICAST  MTU:1500  Metric:1
      RX packets:0  errors:0  dropped:0  overruns:0  frame:0
22     TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
      collisions:0 txqueuelen:1000
24     RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
26
28  lo        Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      inet6 addr: ::1/128  Scope:Host
      UP LOOPBACK RUNNING  MTU:65536  Metric:1
30     RX packets:21  errors:0  dropped:0  overruns:0  frame:0
      TX packets:21  errors:0  dropped:0  overruns:0  carrier:0
32     collisions:0 txqueuelen:1
      RX bytes:2208 (2.2 KB)  TX bytes:2208 (2.2 KB)

```

traces/5.1.txt

After:

```

2  eth0      Link encap:Ethernet  HWaddr 68:05:ca:39:e1:2f
      inet addr:10.0.0.11  Bcast:10.0.0.255  Mask:255.255.255.0
4      inet6 addr: fe80::6a05:caff:fe39:e12f/64  Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:421  errors:0  dropped:0  overruns:0  frame:0
6      TX packets:106  errors:0  dropped:0  overruns:0  carrier:0
      collisions:0 txqueuelen:1000
8      RX bytes:55364 (55.3 KB)  TX bytes:12063 (12.0 KB)
      Interrupt:19  Memory:f06c0000-f06e0000
10
12  eth1      Link encap:Ethernet  HWaddr 68:05:ca:39:e1:32
      BROADCAST MULTICAST  MTU:1500  Metric:1
      RX packets:0  errors:0  dropped:0  overruns:0  frame:0
14     TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
      collisions:0 txqueuelen:1000
16     RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
      Interrupt:16  Memory:f05c0000-f05e0000
18
20  internet  Link encap:Ethernet  HWaddr d0:50:99:55:a9:0c
      BROADCAST MULTICAST  MTU:1500  Metric:1
      RX packets:0  errors:0  dropped:0  overruns:0  frame:0
22     TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
      collisions:0 txqueuelen:1000

```

```

24          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
26 lo          Link encap:Local Loopback
                inet addr:127.0.0.1  Mask:255.0.0.0
28                inet6 addr: ::1/128 Scope:Host
                UP LOOPBACK RUNNING  MTU:65536  Metric:1
30                RX packets:21  errors:0  dropped:0  overruns:0  frame:0
                TX packets:21  errors:0  dropped:0  overruns:0  carrier:0
32                collisions:0 txqueuelen:1
                RX bytes:2208 (2.2 KB)  TX bytes:2208 (2.2 KB)

```

traces/5.3.txt

The ifconfig command lists all the network interfaces on the pc (in our case eth0, eth1 and internet).

For each interface/network device it displays a number of things:

- Link encap: indicates what kind of device this is. In our case this is either an Ethernet device, or a Local Loopback (virtual) device.
- HWaddr: the MAC address of the interface.
- inet addr: the IPv4 address of the interface.
- Bcast: the IPv4 address on which the interface can send broadcast packets.
- Mask: the netmask associated with the interface.
- inet6 addr: the IPv6 address of the interface.
- Scope: determines the routing scope for this interface.
- MTU: the Maximum Transmission Unit.
- Metric: this value decides the priority of the interface, the higher the value the higher the priority.
- RX packets: the amount of received packets (normal packets, errored packets, ...).
- TX packets: the amount of transmitted packets (normal packets, errored packets, ...).
- collisions: the amount of collisions that have happened on the interface.
- txqueuelen: the length of the transmit queue of the interface.
- RX bytes: the amount of bytes received.
- TX bytes: the amount of bytes transmitted.
- Interrupt: the interrupt line used by this device.
- Memory: the memory addresses used for shared memory for this device.

Duplicate IP addresses

In this part of the lab, you observe what happens when two hosts have identical IP addresses.

Exercise 6. Duplicate IP addresses

1. After completing Exercise 5, the IP addresses of the Ethernet interfaces on the four PCs are as shown in the Table 2.3. Note that PC1 and P4 are assigned the same IP address.

Linux PC	IP Addresses of Ethernet Interface eth0
PC1	10.0.1.11/24
PC2	10.0.1.12/24
PC3	10.0.1.13/24
PC3	10.0.1.11/24

Table 2.3: IP addresses for Part 6

2. Delete all entries in the ARP cache on all PCs.
3. Run Wireshark on PC3 and capture the network traffic to and from the duplicate IP address 10.0.1.11
4. From PC3, start a Telnet session to the duplicate IP address, 10.0.1.11, by typing


```
PC3% telnet 10.0.1.11
```

 and log in as *telecomlabo* user, using the *mvkbj1n* password.
5. Once you have logged in, determine the name of the host to which you are connected. The name of the host can be determined in several ways: (1) issue the command `hostname`, (2) inspect the ARP cache on PC3, or (3) interpret the captured Wireshark packets.
6. Stop the traffic capture in Wireshark.
7. Save all ARP packets and the first few TCP packets captured by Wireshark. Also save the ARP cache of PC3 using the `arp -a` command.
8. When you are done with the exercise, reset the IP address of PC4 to its original value as given in Table 2.1.

Question 6.1)

Explain why the Telnet session was established to one of the hosts with the duplicate address and not the other. Explain why the Telnet session was established at all, and did not result in an error message. Use the ARP cache and the captured packets to support your explanation.

ARP asked for the MAC address of the computer which has the ip address 10.0.1.11. The pc which answered first on the request will be put in the arp cache, as well as used for the telnet session. This did not result in an error, because the telnet session was established with a valid MAC address.

In the captured packets ("traces/6.packets.pcap") you can see the telnet session is established after receiving a reply to the ARP request. The arp cache is adjusted to include the reply from the ARP request.

```
? (10.0.1.14) at <incomplete> on eth0
2 ? (10.0.1.11) at 68:05:ca:36:33:a0 [ether] on eth0
```

traces/6.arpcache.txt

Changing netmasks

In this part of the lab you test the effects of changing the netmask of a network configuration. In the table below, two hosts (PC2 and PC4) have been assigned different network prefixes.



If you are having difficulties understanding the concept of netmasks, try using a subnet calculator tool. You can find one at <http://subnet-calculator.com> or most operating systems have native tools or widgets that can do the same.

Exercise 7.

1. Setup the interfaces of the hosts as shown in Table 2.4. Note that the netmasks of the hosts are different.

Linux PC	IP Addresses of eth0	Netmask
PC1	10.0.1.100/24	255.255.255.0
PC2	10.0.1.101/28	255.255.255.240
PC3	10.0.1.120/24	255.255.255.0
PC3	10.0.1.121/28	255.255.255.240

Table 2.4: IP addresses for Part 7

2. Run Wireshark on PC1 and capture the packets for the following ping commands
 - a. From PC1 to PC3:

```
| PC1% ping -c 1 10.0.1.120
```
 - b. From PC1 to PC2:

```
| PC1% ping -c 1 10.0.1.101
```
 - c. From PC1 to PC4:

```
| PC1% ping -c 1 10.0.1.121
```
 - d. From PC4 to PC1:

```
| PC1% ping -c 1 10.0.1.100
```
 - e. From PC2 to PC4:

```
| PC1% ping -c 1 10.0.1.121
```
 - f. From PC2 to PC3:

```
| PC1% ping -c 1 10.0.1.120
```
3. Save the Wireshark output, and save the output of the ping commands. Note that not all of the above scenarios are successful. Save all output, including any error messages.
4. When you are done with the exercise, reset the interfaces to their original values as given in Table 2.1.

Question 7.1)

Use your output data and ping results to explain what happened in each of the ping commands. Which ping operations were successful and which were unsuccessful? Why?



You get credits for answering the 'why?' question, not for stating the obvious.

The first 2 pings,

```

PING 10.0.1.120 (10.0.1.120) 56(84) bytes of data .
2 64 bytes from 10.0.1.120: icmp_seq=1 ttl=64 time=0.423 ms

4 — 10.0.1.120 ping statistics —
  1 packets transmitted, 1 received, 0% packet loss, time 0ms
6 rtt min/avg/max/mdev = 0.423/0.423/0.423/0.000 ms

```

traces/7.2.a.txt

and

```

PING 10.0.1.101 (10.0.1.101) 56(84) bytes of data .
2 64 bytes from 10.0.1.101: icmp_seq=1 ttl=64 time=0.450 ms

4 — 10.0.1.101 ping statistics —
  1 packets transmitted, 1 received, 0% packet loss, time 0ms
6 rtt min/avg/max/mdev = 0.450/0.450/0.450/0.000 ms

```

traces/7.2.b.txt

, are succesful. The first ping is succesful since PC1 can see PC3, according to their respective netmasks, and the same can be said the other way around.

The second ping is succesful for the same reason: PC1 can see PC2 and PC2 can see PC1 (PC2 can see addresses in range 10.0.1.97 - 10.0.1.110).

The third ping,

```

PING 10.0.1.121 (10.0.1.121) 56(84) bytes of data .
2
4 — 10.0.1.121 ping statistics —
  1 packets transmitted, 0 received, 100% packet loss, time 0ms

```

traces/7.2.c.txt

, is not succesful. The ping can be sent from PC1 to PC4, but PC4 cannot send a reply to PC1: it is not reachable with its netmask (range 10.0.1.113 – 10.0.1.126).

The other pings,

```

1 connect: Network is unreachable

```

traces/7.2.d.txt

```

1 connect: Network is unreachable

```

traces/7.2.e.txt

```

1 connect: Network is unreachable

```

traces/7.2.f.txt

, are not succesful either. This is because, in all 3 scenarios, the PC which tries to send a ping cannot reach/see the destination PC. The reason remains the same: the ip addresses are not in the range of the ip + netmask of the source PC.

Static mapping of IP addresses and hostnames

Since it is easier to memorize names than IP addresses, there are mechanisms to associate a symbolic name, called *hostname*, with an IP address. On the Internet, the resolution between hostnames and IP addresses is generally done by the Domain Name System (DNS), which will not be discussed in this course. This experiment illustrates another, simpler method to map IP addresses and domain names using the host file `/etc/hosts`.

Before DNS became available, the `/etc/hosts` file was the only method to resolve hostnames in the Internet. All hosts on the Internet had to occasionally synchronize with the content of other `/etc/hosts` files.

Exercise 8. Associating names with IP addresses

In this exercise, you manipulate the static mapping of hostnames and IP addresses using the `/etc/hosts` file.

1. On PC1, inspect the content of file `/etc/hosts` with a text editor.

2. On PC1, issue a `ping` command to PC2

```
| PC1% ping 10.0.1.12
```

3. Repeat Step 2, but use symbolic names instead of IP addresses (e.g., PC2 instead of 10.0.1.12). You should see that the symbolic name is unreachable at this point.
4. On PC1, edit the file `/etc/hosts` and associate hostnames with the IP addresses and save the changes. Use the names PC1, PC2, etc., as used throughout this lab to refer to the PCs.
5. Repeat Step 3. You should now be able to ping directly using the hostnames "PC2", "PC3", "PC4", as in:

```
| PC1% ping PC2
| PC1% ping PC3
| PC1% ping PC4
```

6. Reset the `/etc/hosts` file to its original state. That is, remove the changes you have made in this exercise, and save the file.

Question 8.1)

Explain why a static mapping of names and IP addresses is impractical when the number of hosts is large.

A static mapping of IP Addresses is impractical since IP Addresses are not necessarily static. When host(s) change ip address(es), it's a hassle to continuously adjust the mappings on all the different hosts to conform to the new address(es).

Question 8.2)

What will be the result of the hostname resolution when multiple IP addresses are associated with the same hostname in the `/etc/hosts` file?

It will use the first entry in the table which corresponds with the specified hostname.

Experiments with FTP and Telnet

A severe security problem with the file transfer protocol (FTP) is that the login and password information are transmitted as plain text (not encrypted). Sometimes malicious users exploit this by snooping passwords on the network.

Here you learn how easy it is to crack passwords by snooping traffic from FTP and Telnet sessions.



The use of applications that do not encrypt passwords, such as FTP and Telnet, is strongly discouraged. On the Internet, you should use protocols such as Secure Shell (SSH) tools for file transfers and remote login.

Exercise 9-A. Snoop Passwords from an FTP session

Capture traffic from an FTP session between two hosts.

The ftp server installed on the lab PC's is vsftpd, which is not started by default. Use the following command to start it on PC2:

```
|PC2% service vsftpd start
```

1. On PC1, run the Wireshark command with capture filters set to capture traffic between PC1 and PC2. The capture filter is

```
|host 10.0.1.11 and host 10.0.1.12
```

2. On PC1, initiate a FTP session to PC2 by typing

```
|PC1% ftp 10.0.1.12
```

3. Log in as *telecomlabo* user.
4. Inspect the payload of packets with FTP payload that are sent from PC1 to PC2. FTP sessions use TCP connections for data transfer.



In Wireshark, there is a simple method to view the payload sent in a TCP connection. Simply select a packet that contains a TCP segment in the main window of Wireshark, and then click on "Follow TCP Stream" in the "Tools" menu of the Wireshark window. This will create a new window that displays only the payload of the selected TCP connection.

5. Save the details of the packets, i.e., select "Print details" in the "Print" window of Wireshark, which transmit the login name and password. As a hint, you can set the display filter in Wireshark to show only the desired packet(s). Refer to Question 9 from the Prelab.

Question 9.A.1)

Using the saved output, identify the port numbers of the FTP client and the FTP server.

The source port is 49908, the destination port is 21. This can be seen in the packets in the file "traces/9-A.pcapng".

Question 9.A.2)

Identify the login name and the password, shown in plain text in the payload of the packets that you captured.

In the packets you can see a message with Request: USER student, and Request: PASS mvk bj1n (frame 5 and 11 respectively, file "traces/9-A.txt").

Exercise 9-B. Snoop Passwords from a Telnet session

Repeat the above exercise with the `telnet` command instead of `ftp`. On PC1, establish a Telnet session to PC2, and save the Wireshark output of packets used to transmit the login name and password.

Question 9.B)

Does Telnet have the same security flaws as FTP? Support your answer using the saved output.

Yes, but the username is sent one letter per packet (every letter twice for robustness). The password is also one letter per packet, but they are only sent once as opposed to the username. This can be seen in the pcap file "traces/9-B.pcapng", especially when following the TCP stream.

Exercise 9-C. Observing traffic from a Telnet session

This exercise uses the Telnet session established in the previous exercise.

1. Run Wireshark on PC1, and start to capture traffic. If the Wireshark window from the previous exercise is still open, make sure that Wireshark is capturing traffic.
2. If the Telnet session from the previous exercise is still in place, skip to the next step. Otherwise, follow the steps from the previous exercise and log in from PC1 to PC2 with the `telnet` command.
3. Once you are logged in, type a few characters. Observe the number of packets, captured by Wireshark, for each character typed. Observe that for each key you type, there are three packets transmitted. Determine why this occurs.
4. Save the Wireshark output to a text file (using the "Print Summary" option).

Question 9.C)

Attach the saved output to your report. Explain why three packets are sent in a telnet session for each character typed on the terminal.

For each character typed, 2 packets with the character are transmitted for robustness. An additional packet is sent for an acknowledgment.

