Network lab #4
Transport Protocols : UDP / TCP

Ensimag – Phelma – Master CSE

1 Foreword

During this lab course you will have to go through a series of configurations and observations that will help you understand network principles.

0 – This icon indicates that you must configure the network. Below is the first configuration to do before going further! Before starting any configuration, you must isolate your workstations from the university network they are connected to through interface `bge0` using the following commands:

```
# ifconfig bge0 down
# ifconfig bge0 delete
```

0 – This icon indicates that you have questions to answer. There is a blank space on this sheet to write down your answers.

Making experiments in networking requires great attention and thoroughness. Be sure to stop the commands that are running once you have finished your observations. Try to use several terminals and start only one command in each of them so that you really control what is going on. Have `tcpdump` running to check what is going on in the network, and for example that all the commands generating traffic were stopped correctly before proceeding to the next experiment.

2 Introduction

During this lab, the four stations must belong to the same network.

3

1 – Use two stations, execute `socklab` and say that you want to work with UDP:

```
# socklab udp
```

Create an UDP socket on each station.

```
socklab-udp> sock
```

These two sockets will be used to exchange data between two stations.

This is equivalent to: `sock udp`, creation of a UDP *socket* and `bind`, assignment of an IP address and a port.

Write down this port number: it identifies the socket in your machine in a unique manner.

PC3: port = 30620 **PC4: port = 561328**

2 – On one of these two stations, ask for the emission of a data packet to the other station. Of course, you must specify the name and port number of the destination:
`socklab-udp> sendto <Socket Id> <hostname> <port #>`
Specify the string that must be transmitted to the destination.

3 – What is the goal of the socket identifier?

On a local machine, the access to a socket is done through a unique file. The socket identifier is the file descriptor of the socket. It allows the OS to identify the fils.

4 – On the second station, ask to receive a packet on the socket you created, and specify the number of bytes you want to read:
`socklab-udp> recvfrom <Socket Id> <byte #>`
Using `ethereal`, analyze the packet(s) generated by this emission.
TIP: In the `capture` menu select `options` and tick the capture in real-time for a live analysis of the network. Moreover, to avoid `ethereal` filtering information go into `Preferences` → `protocols` → `TCP` and uncheck everything.

5 – Specify the role of each field in the UDP header. What are the information transmitted by UDP to IP (data + service parameters)?
A UDP packet has the following structure :

0	16 31
Source port (16)	Destination port (16)
Message length (16)	Checksum (16)

- Source port: Port number where the message has been sent on the source machine.
- Destination port: Port number where the message is received on the destination machine.
- Message length: Size (in bytes) of the message (header + payload)
- Checksum: Check the partial correctness of the message (and header). Is computed using the sum of the first complement of the bytes + 1 if odd number of bytes.

6 – Retry the previous operations (emission and reception) by swapping the role of the two stations (one socket can be used for both reception and emission). You can try the following variants:
- ask to receive the data before the emission;
- send several packets before trying to read them;
- ask for one emission from both sides: Execute the emissions of both extremities, and try to receive them;
- observe what is happening when you try to read less/more data than what is emitted;
- send a data packet to a station disconnected from the network;
- Send several large packets so that the reception buffer is full: e.g., send 5 packets of 9000 bytes, and Try to receive them (with parameter `#9000` to send packets of 9000 bytes).

7 – Try to sum up the behavior of UDP.

The UDP protocol has a time efficient behaviour:
- Transmission may be successfully done from both sides at the same time.
- A receiver may read more bytes than the buffer size.

But this results in a bad memory coherence:
- UDP protocol does not check that a packet has been received.
- If the receiver's buffer is full, the arriving packets are lost.
- The corrupted packets are not corrected neither detected (only detect byte removal)
- If the received message is bigger than the expected size, the extra packets are omitted (erased).

4 TCP

Reminder : TCP is a connection oriented protocol, and guaranteed *byte flows* without errors.

TCP headers are :

0	4	10	16 31
Source port (16)			Destination port (16)
Sequence number (32)			
Acknowledgement number (32)			
Hlen (4)	Reserved (6)	Control bits (6)	Window (16)
Checksum (16)			Urgent pointer (16)
Options ($8.n$)		Padding	

The field **control bits** is divided in 6 *flags* : URG, ACK, PSH, RST, SYN, FIN.

Fields signification:

source port and **destination port** have a similar meaning as in UDP;

sequence number and **ack number** are used for the sequencing and error control (the flag ACK indicates if the field **ack number** contains a valid value);

hlen specifies the header length in words of 4 bytes (the TCP header length could vary : one or several optional fields of 4 bytes can be present);

checksum is the same as in UDP;

urgent pointer is used to carry urgent data (the flag URG specifies if the field **urgent pointer** is a valid value);

the flags SYN and FIN are used to open and close virtual connections ; RST is used to reinitialize the virtual connections in uncertain state (SYN duplicated or failed);

window is used for flow control.

4.1 Connection establishment

☞ **8** – In two stations, execute `socklab` with TCP :
`# socklab tcp`
On the first machine, create a *passive* socket. Write down the port number and put it in listen mode.
`socklab-tcp> passive`

It is equivalent to : `sock tcp`, socket creation, `bind`, IP address and port number assignment, `listen`, waits on this socket.

☞ **9** – Put the socket in a mode where it waits to accept a connection:
`socklab-tcp> accept`
On the second machine, create an *active* socket and connect it to the first machine (with the *passive* socket). You can use :
`socklab-tcp> connect <hostname> <port #>`

It is equivalent to : `sock tcp`, socket creation, `bind`, IP address and port number assignment and `connect`, connection request.

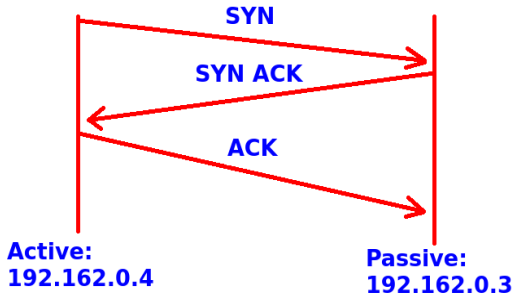
10 – Why the first socket is called *passive*, and the second one *active* ? What are the role of these two sockets in the connection opening?

- The first socket sends no message. Its role is to wait for remote requests, and acknowledge them. Thus it is passive.

- The second socket is the one that initiates and seeks for the transmission. Thus it is active

11 – Analyze the packets for this connection.

12 – Extract the steps of this connection, how they are organized, messages exchanged, with a temporal diagram



13 – Explain what happens when the socket is in the *accept* mode. Try to do it before and after the *connect*. Take a look on the socket list (status command).

By analysing the package transmission, we have noticed that:
- Before the "connect" operation is performed, the passive machine has sent an ACK. The connection is established on the active machine.
- After the "connect" is performed, the connection is also established on the passive machine.

14 – What information is exchanged ? What is the goal of the flag SYN ?

- The exchanged informations are: source port, destination port, sequence number, header length, flags, window size, checksum, synchronization info (initialized sequence number and message id on both sides).
- The SYN flag is used to initialize the connection protocol (hand shake) and send the passive machine informations.

15 – Open several connections to the same port destination.

16 – What is the real identifier of one connection, how TCP associated the received messages to one connection?

- The real identifier of a connection is the quadruple
  * Source machine IP * Source machine port
  * Destination machine IP * Destination machine port

- Each exchanged TCP package contains the four previous informations. Thus TCP is able to identify a unique connection mapped to each packet.

4.2 Connection termination

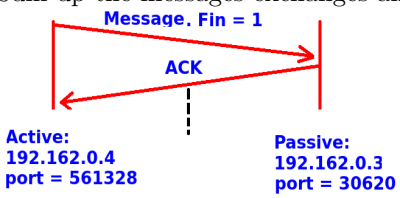
17 – After opening a connection between two hosts, close it:

```
socklab-tcp> close
```

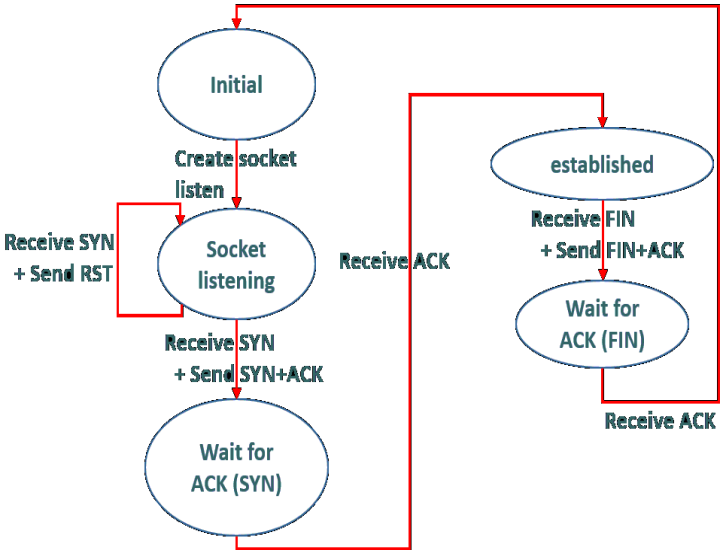
✎ **18** – Analyze the packets and their direction. Find the steps of this closing. What is the goal of the flag FIN ?

The goal of the FIN flag is to inform the remote machine about the end of the connection.

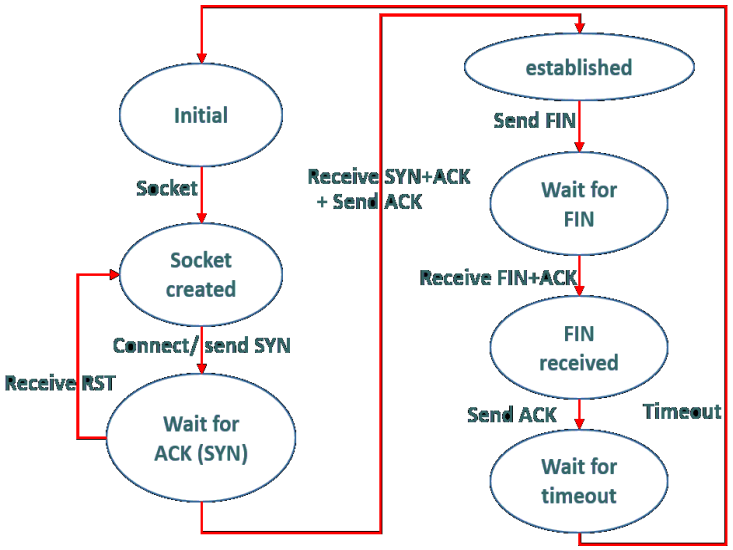
✎ **19** – Sum up the messages exchanges and specify the field values for this step.



✎ **20** – Summary: Write the state machine for TCP active and passive sockets (cf. Figure 1).



state machine for the TCP passive socket



state machine for the TCP active socket

4.3 Sequencing and error control

☞ **21** – Establish a TCP connection between the two stations and exchange some data:
`socklab-tcp> read <byte #>`
to read bytes on the new socket, and
`socklab-tcp> write`
to send messages (instead of a normal message, you can use the notation `#n` to send a message of n bytes).

22 – Analyze this exchange. Explain the goal of the fields **sequence number** and **ack number** in TCP packets.

- **Sequence number** is the number of bytes in the last sequence from which the package begins
  So sequence number is used to track what has been sent.
- **ACK number** is the number of the last byte in the last sequence that has been correctly received (acknowledged).
  For example, if we have byte stream <23, 24 (lost), 25> ACK will be equal to 23.

23 – Give an intuitive explanation of the update rules for the value of these two fields. Distinguish the case in which we receive a packet from the case in which we emit one packet.

- **Emission:** if the previous packet (or set of packets) has been acknowledged, the ACK number is incremented and sent with the new packet. Otherwise the packet is sent again with the same ACK.
- **Reception:** update the sequence number using the ACK number (sequence number := ACK)

24 – Unplug the cable between the two stations (disconnected stations) and redo the operation. What happens? Reconnect the stations, check the traffic.

25 – Explain what happens in this kind of situation (segment loss), compare with UDP.

- If the connection is lost, the sender will not receive ACK for a given packet. Thus it will carry on sending the same packet using an increasing timeout delay (until it reaches a limit).

- TCP protocol guaranties the transmission, contrary to UDP.

4.4 Flow control

26 – Open a TCP connection. Send a massive amount of data on this connection (70'000 bytes).

27 – What is happening? Why?

- Once the message has been sent, the sender receives a message with window size = 0. Then the sender carries on sending periodically a message with a window size = 0.
- This behaviour is due to the "TCP zero window" algorithm. It helps thepassive socket to know if the active socket is still reachable in order to keep the connection alive.

If the window size of receiver is bigger than 0, sender continues transmission.

28 – Execute several successive `read` at the receiver (with 1 byte then 100 and then 5'000).

29 – Analyze the last exchanged packets. Take a closer look on the field **window**. Explain how the control flow works.

- Once the receiver performs the read operation, he acknowledges the message with a window size. The receiver computes this size using the transfer time of the packet and the routers congestion informa
- The TCP control flow principle is to constantly increase the window size in order to reach the optimal one: use a maximum of the network possibilities without creating packet loss. The window size is periodically decreased when a packet loss occur or when the receiver requires it.

5 IP fragmentation

We will now take a look on the IP fragmentation mechanism.

IP fragmentation is used when an IP packet is too large to be transmitted as is through the link layer

(an Ethernet link cannot deal with packets longer than 1518 bytes : Ethernet header and CRC included). Several small fragments must be created.

When a packet is fragmented, it must be recombined at the receiver (n the IP layer). If the destination does not support re-assembly, fragmentation must be forbidden (with the flag DF, *Don't Fragment*). When this flag is set, fragmentation is forbidden. The delivery could be impossible. We will now analyze this fragmentation.

30 – Execute `ethereal` on the monitoring station. On another station, send a UDP segment of 4'600 bytes to a socket that was previously created.

31 – Analyze exchanges. In particular, study the role of **total length**, **fragment offset**, and the flag MF, *More Fragments*.

- Fragment offset is responsible of the offset alignment within an IP packet.

- NF flag indicates whether there are more fragments in the packet to be transmitted.

- Total length is the length (in bytes) of the fragment (maximum value of fragment offset).

32 – Repeat this manipulation with longer packets (if necessary). Repeat the same operations with TCP.

33 – Conclusions?

- At the IP level, this operation has been handled in the same way using TCP and UDP: the message has been fragmented in fragments of equal size (500 bytes) with different offset values.

- The previous observations made on the UDP and the TCP protocols still hold during this experience.

34 – Verify the conclusions about **sequence number** and **ack number** fields of the TCP header previously described.

Thanks to the "socklab" tool, we have observed the consecutive acknowledgements sent during a TCP connection session. This way we have confirmed that at each step, the sender updates the sequence numberusing:

new_sequence_number = old_sequence_number + previous_acknowledgement_size