

Computer Networks – Practical session #4

Transport protocols: UDP and TCP

M1 MOSIG

April 2022

In this practical session, you will learn in details about layer 4 of the Internet protocol stack, namely *transport*. The two most common transport protocols are UDP and TCP, which we will study thanks to the `socklab` software. The focus will be on the programming interface that allows applications to use UDP and TCP, called the *sockets* interface.

Don't forget to send your report to your teacher at the end of the practical session! You can send one report for each group. Don't forget to mention the name of the other person in the group.

1 Introduction

In this lab, you will work in groups of two persons, each with a FreeBSD computer. The computers can simply stay attached to the ENSIMAG network (i.e. there is no need to manually connect them together with a switch).

You will need the `socklab` program to open UDP/TCP connections and send data, and Wireshark to analyse the network traffic. Both are pre-installed on your FreeBSD computers.

2 First steps with `socklab`

2.1 About `socklab` and the sockets API

The sockets API is a standard interface that allows programs to make use of a communication network. It originates from the BSD systems, and has been designed so that network communication looks like reading and writing to a file.

More precisely, programs use the sockets API to interact with the *transport layer* of a network. Two types of services are defined: *connection-oriented* (often implemented with TCP) and *datagram-oriented* (often implemented with UDP). You will see the difference in details during this practical session.

`socklab` is an interactive program that allows to manipulate this API without having to write C code. That is, using a shell-like interface, you can interactively create communication endpoints (sockets), connect them to a remote host, send and receive data, and so on.

2.2 socklab tutorial

All manipulations must be done between a pair of computers.

First, you will learn how to use `socklab` with a simple case based on UDP.


 On each computer, launch `socklab` in UDP mode:


```
# socklab udp
```

 Create an UDP socket on each computer:

```
socklab-UDP> socket
```

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


 Write down the port number returned by the socket creation: together with the IP address of your computer, it uniquely identifies the communication socket you just created.

 On one of the computers, send a packet of data to the other computer. You must specify the host name and port number of the destination computer. Here, the hostname can simply be the DNS name of the remote machine (`ensipcXXX`), but you could also specify an IP address directly.


```
socklab-UDP> sendto 3 <hostname> <port number>
```

You must then type a string that will be transmitted to the destination.

Note: all `socklab` commands like `sendto` can be used without any arguments. In this case, you will be prompted for the necessary arguments. This can be helpful if you don't remember how to use a command!

 Send another UDP message by just calling `sendto` without arguments. For the “Id. socket”, keep the default value by just pressing Enter.

You will now receive the messages on the other computer, using the `recvfrom` command.

 On the other computer, use the `help` command to understand about the usage of the `recvfrom` command.

Note: you can also use the `help` command without argument to list all available commands.

☞ Use `recvfrom` to receive data from the UDP socket.

☞ Use the `status` command (shorthand: `=`) to display the state of your sockets.

Q 1 — *What is the value of the column `RWX` for your UDP socket on each computer?*

This column indicates whether the socket is ready for **R**eading and/or **W**riting. The **X** column indicates an exception and is rarely used.

Q 2 — *What is the value of the column `RWX` after you have received all pending messages?*

3 Analysis of the UDP protocol

Reminder: UDP is a datagram-oriented protocol, without any guarantee about successful delivery and ordering of datagrams.

☞ Start a packet capture with Wireshark. You can filter on the IP address of the remote computer so that you will only see the network packets generated by `socklab`.

☞ Send a UDP message from one computer, and then read it on the other computer.

Q 3 — *When is the message actually transmitted over the network: when you call `sendto` on the transmitting computer, or when you call `recvfrom` on the receiving computer? What can you deduce about message buffering, i.e. where are messages stored if they are not read immediately?*

☞ Perform transmission and reception of messages in various situations to understand how UDP works. Note that the same socket can be used both for transmission and for reception. You can experiment with the following variants:

- try to receive data on a socket before sending a message towards it;
- transmit several messages before trying to receive them on the remote socket; also try with `mrecvfrom`;
- transmit simultaneously from both sockets to the other, and then try to receive from both sockets;
- observe what happens when you try to read less/more bytes than what was transmitted;
- send a message to a computer that is physically unplugged from the network (or whose network interface has been brought down with `ifconfig`);
- send several large messages before trying to receive them on the remote socket (for instance, send several messages of 9000 bytes each, using the syntax described below).

Note: to send large messages, you can use the following syntax for the message content: #42 (this will send a message with 42 bytes).

Q 4 — *Based on your observations, write a summary of the behavior of UDP. In particular, give details:*

1. *on which functions can “block” (i.e. not return immediately) and in which conditions;*
2. *on the reliability of message delivery;*
3. *on error reporting between sender and receiver;*
4. *on the buffering mechanism.*


4 The TCP protocol


Reminder: TCP is a connection-oriented protocol, that guarantees reliable communication as a *byte flow* between two networked programs.

4.1 TCP basics

 Run `socklab` in TCP mode on both computers:

```
# socklab tcp
```

 On the first computer, use the `passive` command to create a socket that will accept connections (“server” mode). Write down the port number.

 Try to *accept* a new connection on this socket:

```
socklab-TCP> accept 3
```

Q 5 — *What happens?*


 On the second computer, create a socket and connect it to the passive socket of the first computer. To do this:

```
socklab-TCP> connect <hostname> <port number>
```

Q 6 — *What happens on the first computer once the connection is established? How many sockets are now present on the first computer?*


Q 7 — *What is the role of the passive socket? By looking at the `RWX` flags of this passive socket, do you think you can use this socket to communicate with another computer?*

4.2 Multiple connections

 Run `accept` a second time on the first computer.

 On the second computer, open a new TCP connection towards the first computer, on the same TCP port as before.

Q 8 — *How many sockets are there on the first computer now? And on the second computer?*

 Start a packet capture.

 On the second computer, send a message on one of the sockets.

Q 9 — *On the first computer, which socket is now ready to read? Could you have guessed by just looking at the addresses and ports shown by the `status` command?*

Q 10 — *Look at the packet capture: could you have guessed the right socket by just looking at information contained in the packets? If yes, how?*

Q 11 — *Based on the previous questions, how would you precisely define a “TCP connection”? That is, what distinguishes two different TCP connections from one another?*

Q 12 — *Conclude: why would a program need to use several TCP sockets? What is the difference with UDP?*

4.3 Connection establishment

 Capture and analyse the packets used to open a new TCP connection.

Q 13 — *Draw a temporal diagram to explain the different types of message involved in opening a TCP connection. Make sure to mention the TCP flags present in each message.*

Q 14 — *What is the role of the SYN flag?*

Q 15 — *What TCP fields are exchanged between the client and the server during the connection initialisation?*

4.4 Connection termination

 Close one of the connections you have opened:

```
socklab-TCP> close
```

Q 16 — *Is the connection closed on both computers? What should you do to properly close the connection?*

☞ Capture and analyse the packets involved in a connection termination.

Q 17 — Describe the steps of the connection termination, in each direction. Make sure to describe the TCP flags present in each packet. What is the role of the FIN flag?

Q 18 — Summary: draw a simple TCP state machine for a passive socket, and for an active socket. See Figure 1 for an example.

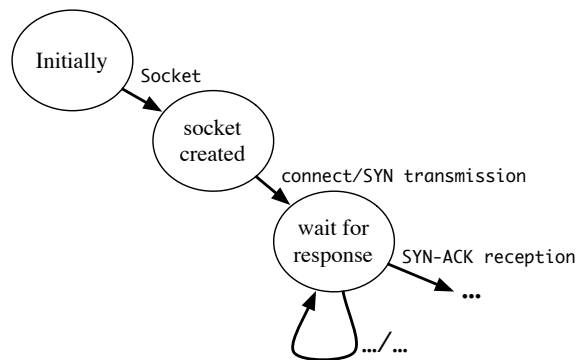


Figure 1: State machine example

4.5 Sequencing and error control

☞ Start a packet capture.

☞ Establish a TCP connection between the two computers, and exchange some data using the `read` and `write` commands.

Note: don't forget that you can use `#1337` as message to send 1337 bytes without typing them manually...

Q 19 — Do you only see packets going from the sender to the receiver, or are there packets in both directions? What is the size of each packet?

☞ Analyse the details of each packet generated by your exchange. Pay particular attention to the **sequence number** and **ack number** fields of the TCP header.

Q 20 — How does the **sequence number** changes from one TCP packet to the next? Is there a difference depending on which direction the packet is going? Explain the role of the sequence number.

Q 21 — Same question for the **ack number**.

☞ Unplug one of the computers from the network and try to send some data. Using Wireshark, observe what packets are generated in this case.

Q 22 — *What happens?*

☞ Connect the computer back to the network.

Q 23 — *Can you now **read** the data that was sent while the computer was disconnected?*

Q 24 — *Explain in details the mechanism used to ensure reliability in case of packet loss. In particular, compare with UDP.*

4.6 Flow control

☞ Open a TCP connection, and send a large amount of data through the connection (70000 bytes, or more if needed).

Q 25 — *What happens? Compare with UDP.*

☞ Perform several successive **read** at the receiver (1 byte, then 100 bytes, then 5000 bytes, and so on). Observe the packets exchanged while doing this.

Q 26 — *Based in particular on the **window** field of the TCP header, explain how flow control works.*

5 IP fragmentation

We will now take a look at 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 sent fragmented, it must be reassembled by the receiver (at the IP layer). If the destination does not support re-assembly, fragmentation must be forbidden (with the flag **DF** (*Don't Fragment*) in the IP header). When this flag is set, fragmentation is forbidden, and it could thus happen that delivery is impossible if the packet is too large.

☞ Start a packet capture, and send a UDP message of 4600 bytes on a UDP socket.

Q 27 — *Analyse the packets generated on the network. In particular, look at the IP headers and explain the role of the fields **total length**, **fragment offset**, and the flag **MF** (More Fragments).*

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

Q 28 — *What can you conclude?*