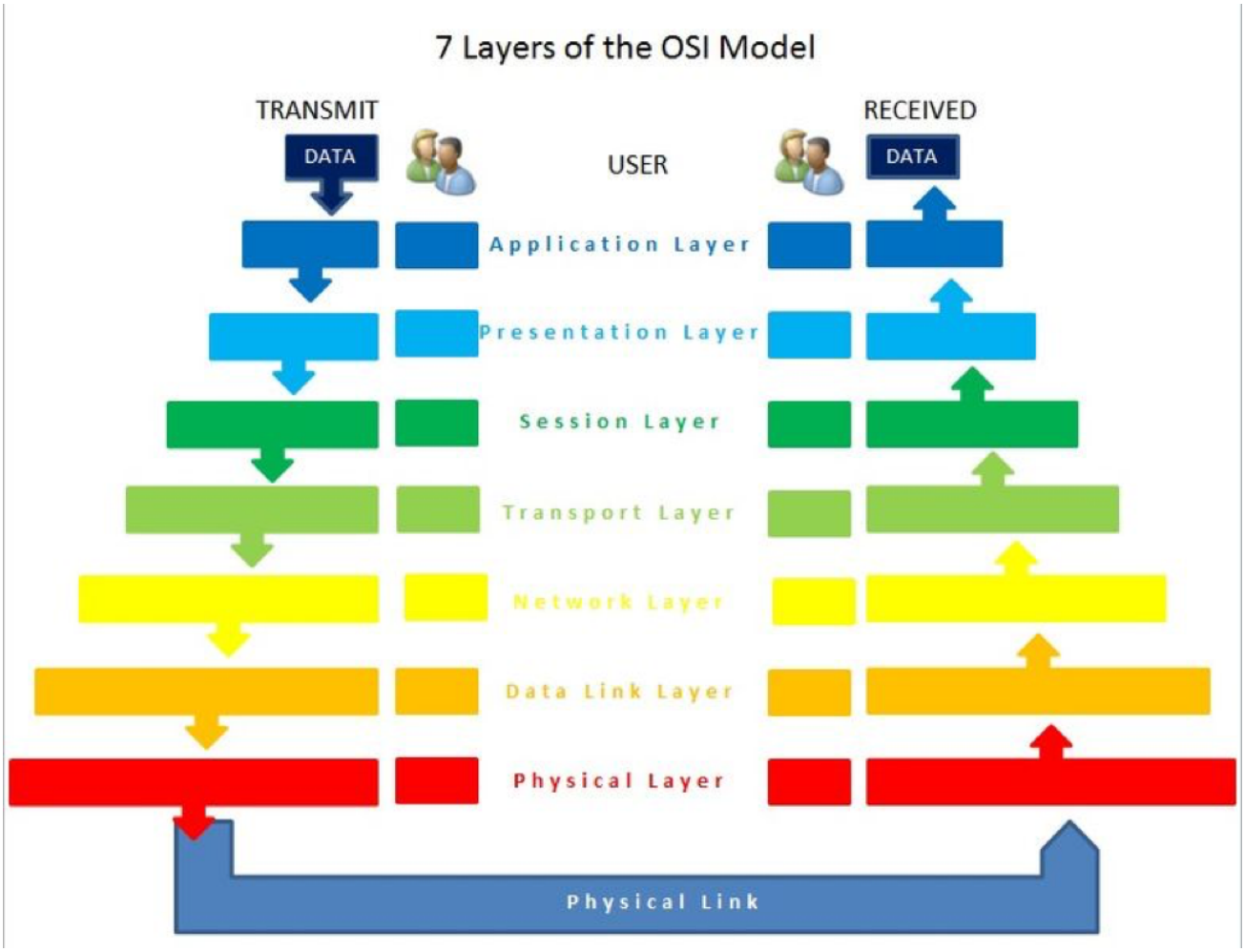
2. Tutorial

Task 1

1. Get informed about the ISO/OSI communication model.
- Explain the tasks of the transport layer. What is the difference between TCP and UDP?
- What is the addressing scheme of TCP/UDP?
- How does error detection in TCP work?
2. Get familiar with a sniffing tool of your choice[1]. Record the traffic, which emerges if you request http://www.tu-chemnitz.de from your Web browser. Which transport protocols are used by HTTP and DNS?



Source: MrsValdry, http://commons.wikimedia.org/wiki/File:OSIModel.jpg

Transport Layer:

- Reliable data exchange between nodes in a network
- Tasks
  - Error recovery
  - Flow control

    - Segmentation and reassembly
    - Multiplexing and demultiplexing of data streams
- Examples: TCP, UDP, SCTP

TCP vs. UDP

	TCP	UDP
Multiplexing *Several applications communicate „in parallel“*	x	x
Error detection *Hash Values (Checksum), for TCP – sequence numbers and acknowledgements*	x	x
Speed	Slower than UDP	Higher than TCP
Connection-Orientation	x	-
Data type *stream means, that applications don‘t see how the data was splitted and transferred. In case of UDP applications should be aware that one incoming message is exactly one data item*	Byte stream	Message
Keeps orders of packets	x	-
Error recovery *repeated sending if acknowledgement is missing or acknowledges a lower sequence number than expected*	x	-
Flow control *send no more packets than receiver can process*	x	-
Congestion control *decrease send rate if no acknowledgement returns, raise slowly*	x	-

Addressing Scheme: Ports

- Numbers up to 65535
- Port numbers till 255 are reserved for well known services (e.g., 21 for FTP, 23 for Telnet, 80 for HTTP)

Task 2

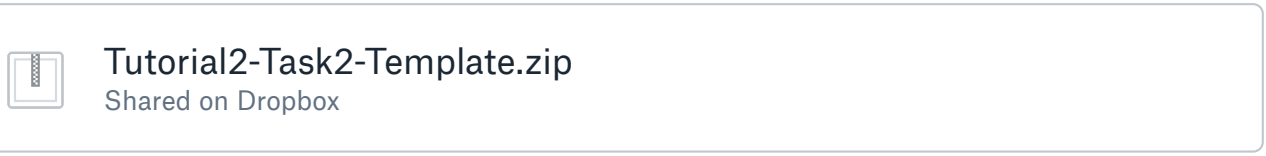
Get informed about Sockets and Berkley Sockets. Extend the provided template with following functionalities:

1. Develop a simple TCP client that can exchange text messages. It should be able to:
   a. Establish a connection
   b. Send a string to a server
   c. Wait for and receive an answer
   d. Close active connections

   Test your client by starting the *TcpClient* subproject – a response from the remote server should be printed to the console.

2. Develop a simple TCP server that is able to receive text messages and process them line by line. Furthermore, it should be able to:
   a. Accept incoming connections
   b. Pass each line of the incoming message to a method with processing logic
   c. Send responses
   d. Close active connections

   Test your server by 1) changing the main function of TcpClient to request a local server and 2) by starting both subprojects in parallel (SolutionàPropertiesàStartup Projects). The TcpClient should print the server answer to the console.



Sockets

- Data structure for reading from/writing to a network connection
- 5-Tupple: Source IP, Source Port, Destination IP, Destination Port, Protocol

Berkley Sockets

- API for communication using sockets
- Available for many programming languages and operating systems
  - **Socket()** – create new data structure
  - **Bind()** – bind to an address (register socket by OS)
  - **Listen()** – set state to LISTENING (let OS accept and queue incoming connections)
  - **Accept()** – put application into sleeping state until the OS connection queue becomes not empty
  - **Connect()** –connection request from the client-side

- **Send()** – send data
- **Recv()** – receive data

Task 3: Hometask

Extend implementation of the server so that it can process incoming connections in parallel. Use the given examples of Thread-programming in C#.

---

[1] For example Wireshark, http://www.wireshark.org/