Question 1: (100 points) Reliable Data Transfer: RDT 2.2 over UDP:

Write a sender, which enables bit-error resistant communication over the UDP protocol. To achieve this, implement the RDT-2.2-protocol from the book *Kurose and Ross: Computer Networking*.¹

Your program should read input from the keyboard and send it to a RDT receiver, which is supplied by us.

You can find the receiver in the 
/afs/net.t-labs.tu-berlin.de/home/praktikum/daten/11.uebung/ directory. It can be started by you with the following commandline:

/afs/net.t-labs.tu-berlin.de/home/praktikum/daten/11.uebung/empfaenger rdt22. The receiver will enter the state `wait for 0`. The receiver will automatically corrupt some incoming and outgoing packets, but it does not lose any packets. To find out about additional commandline parameters (e.g., to select a different UDP port) use

/afs/net.t-labs.tu-berlin.de/home/praktikum/daten/11.uebung/empfaenger -help.

Recall that your program should read its input from the keyboard and submit this data to our receiver using RDT-2.2 over UDP. Our receiver will output the data it received. You may discard keyboard input while waiting for pending ACK packets.

Your program should print for each sent and received packet:

- whether it was sent or received
- (for sent packets:) why it was sent (e.g., new data, retransmit)
- (for received packets:) what it signifies and your planned reaction
- (for received packets:) whether it is corrupted
- which sequence / ACK number it carries.
- how many bytes of payload it carries (i.e., without UDP and RDT 2.2 header information)
- the payload (if any)

Packets shall be sent and received over a UDP socket and they have the following format:

- 2 byte sequence number (for packets you sent) or ACK number for packets you received
- 2 byte checksum
- 2 byte length of the packet’s payload in byte
- payload (can be empty)

The header fields are in network byte order (first Hi-byte, then Lo-byte). ACK-packets you get from the receiver do not have any payload. The checksum function is pretty easy: a ‘good’ packet always has a ‘checksum’ of 0x4f4b, independently of its contents. Any other value signals a corrupt packet². :-)

¹available online at [http://www.net.t-labs.tu-berlin.de/](http://www.net.t-labs.tu-berlin.de/) — Login: student; Password: quarter24

²Of course, this is not a ‘real’ checksum function. But the goal of this exercise is to implement a transport protocol, not a checksum algorithm
Deliverables:

- Your program (as source code and compiled in case you used C, C++ or Java)
- The output of your program in verbose ('chatty') mode for a short example session. The output should show that your program can indeed handle corrupted packets.
- The output of the receiver for the same session, again in verbose mode\(^3\).
- A description of things that should work but don’t really work in your program (if applicable)...

Submission details: look at the FAQ (on http://www.net.t-labs.tu-berlin.de/teaching/ws0708/PD_labcourse/)

Due Date: Tuesday, January 29, 2008, 11:59 h s.t.

---

\(^3\)You can enable the verbose mode on the receiver with

/afs/net.t-labs.tu-berlin.de/home/praktikum.daten/11.uebung/empfaenger --verbose rdt2.2