TECHNISCHE UNIVERSITÄT BERLIN
Fakultät IV – Elektrotechnik und Informatik
Fachgebiet Intelligente Netze und Management verteilter Systeme
Prof. Anja Feldmann, Ph. D.
Harald Schiöberg, Bernhard Ager, Vlad Manilici

5th Assignment Protocol Design WS 08/09

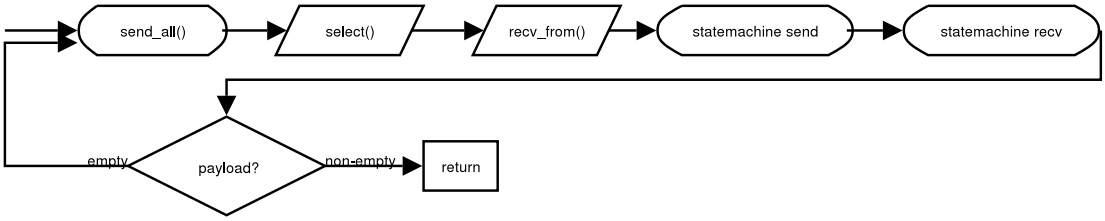
Note

This is the first of a series of assignments that build on each other. Thus you should carefully decide which programming language you choose. No matter what language you decide for, you will be asked to implement a framework consisting of a number of given functions.
The task summary for this assignment: implement a sender and a receiver for *RDT 1.0*. Change the server and client from the 2nd assignment such that it uses *RDT 1.0* as a transport protocol.
The principal goal of this assignment is that you create a well-structured framework for the following assignments. That's why we give detailed directions on *how* to implement certain things. Please follow these directions, even when simpler solutions exist. This will significantly help when working on the subsequent assignments.

Question 1: (70 points) Functions

Implement the following functions. Additional helper functions are allowed but you may not avoid implementing any of the functions described below:

- `rdt_connect()`
  This function sets all the parameters required for connecting to a server. Moreover, it opens a UDP socket.
- `rdt_listen()`
  This function is the server-side counterpart of `rdt_connect()`. It opens a UDP socket for incoming connections and sets all the required parameters.
- `rdt_send()`
  This function takes an arbirary amount of data as input and writes it into a send buffer. It does *not* send anything to the network.
- `rdt_recv()`
  This function returns the whole receive buffer. It does *not* really read data from the network, but only from the receive buffer.
- `rdt_select()`
  This function does the whole work of moving data over the network. It behaves similar to the system call `select()`, i.e. it blocks until there is some data in the receive buffer. The received data can be subsequently read with `rdt_recv()`. A flow diagram of the functionality of this function follows:



  - `send_all()`
    This function sends all the data in the send buffer over the network to the receiver. If more than 500 bytes are present in the send buffer, more packets must be sent. The *MSS* (Maximum Segment Size) of an RDT packet is 534 bytes.

- `select()`
  This is the well-known system call that waits until the receive buffer contains data.
- `recv_from()`
  This is the well-known system all `recv_from()` that reads data from the network.
- `statemachine_send()`
  This is a function that processes a newly received packet. In *RDT 1.0* nothing is done here, becuase the sender does not expect any ACK packets from the receiver.
- `statemachine_recv()`
  This function processes incoming packets. In *RDT 1.0* the header is removed and the payload copied to the receive buffer.
- *Payload*
  `rdt_select()` returns at this point if there's any data in the receive buffer. If not, it starts from the beginning.

Question 2: (30 points) File Server and Client

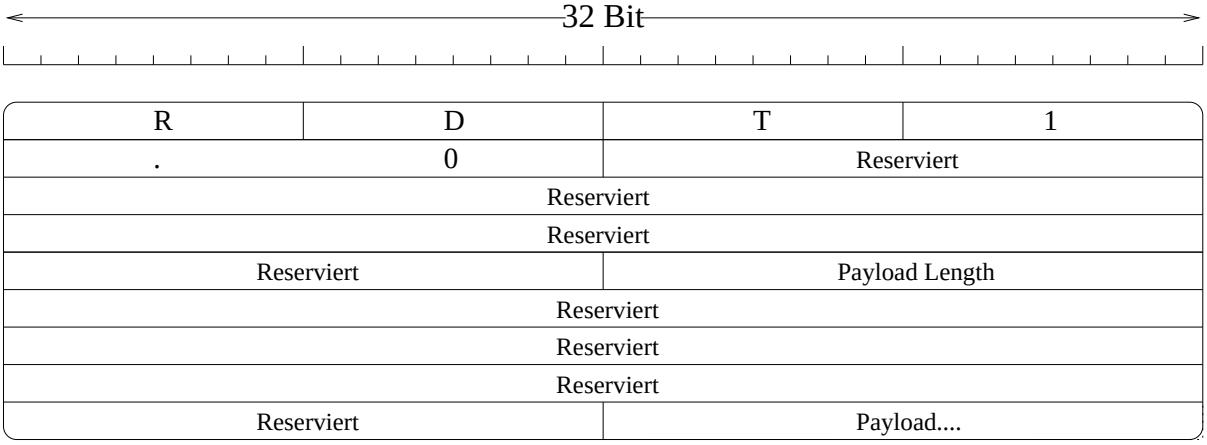
Change the file server and the client from *Assignment 2* such that they use RDT functions instead of system calls.

The Structure of an RDT Packet

The RDT header is built as follows:

- 6 bytes of RDT *magic*, filled with the value 0;
- 12 reserved bytes, filled with nulls;
- 2 bytes payload length. They contain the length of the payload in this packet, in network byte order;
- 14 reserved bytes, filled with nulls.

The total length of the header is 34 bytes. A header is followed by at most 500 bytes payload.



Details on how to submit your assignments:
see FAQ: `http://www.net.t-labs.tu-berlin.de/teaching/ws0809/PD_labcourse/faq.shtml`
Due date: Wednesday, 26.11.2008 at 23:59 s.t.