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

7th Assignment Protocol Design WS 08/09

Question 1: (40 points) Support for several connections for each RDT instance

Extend your program with the capability of simmultaneously processing several incoming connections.

Each connection must have its own state machine and buffers. The following changes must be implemented within the `rdt_select()` function:

- `send_all()` must be called for all existing connections in stead of just one.
- the first thing to be done after receiving a packet is to find the corresponding connection based on the sender address. If none exists, a new connection must be created.
- all other functions within `rdt_select()` (that is, `statemachine_send()` and `statemachine_recv()`) must also be extended with a new parameter for the found (or new) connection.
- `rdt_select()` must return the connection for which data was received, as long as the receive buffer of the connection is not empty.

The following functions must also be extended:

- `rdt_send()` and `rdt_recv()` also require a connection as an additional parameter. The functions must use the buffers that belong to the connections given as parameters.

Question 2: *(60 points) RDT 3.0* Functionality

Extend your program with the functionality of *RDT 3.0* (reliable transmission over a medium with packet losses).

RDT 3.0 only waits for an ACK packet for a limited period of time. The last packet must be retransmitted if no ACK arrives before the timeout.

Implement the following changes in order to support *RDT 3.0*:

- `send_all()` The state machine must be extended with a timer value. When the timer expires, the expected ACK reply is considered lost. The timer will be initialized with the value 6 and decremented every 200 milliseconds. We generally wait at most 1000 milliseconds ($\pm$ jitter) for an ACK.
- `rdt_select()` The `can_read()` receives an extra parameter for the timeout. In the case the operation times out, the function returns an empty list (see: `perldoc IO::Select`).

  The timeout value must be chosen in such a way that `can_read()` returns *at least* every 200 milliseconds. This should happen no matter if it returns because packets were received or it has timed out. The timeout value for `can_read()` must be computed before every call, depending on the moment it last returned (see `perldoc Time::HiRes`).

  If `can_read()` returns because it times out, then you must check *all* existing connections if the timeout is relevant for them. In order to achieve this, decrement all the timers and check if they reach 0. A timer value of 0 means a timeout. The test described here must be implemented within `statemachine_send()`.

  In the case of a timeout, the last sent packet must be *resent* and the timer value must be initialized to 6 ($6 * 200\text{ms} > 1\text{sec} \geq 5 * 200\text{ms}$).

- Implement in `statemachine_send()` the functionality of sender statemachine for *RDT 3.0*. The function may *not* react to ACKs with the wrong sequence number or with bad checksums. In stead, the last packet must be retransmitted after a timeout period.

Please submit the source code of your program.

—End of the 7th Assignment.—

Due Date: Wednesday, 10.12.2008 at 23:59 s.t.