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

7. Übung zur Protokolldesign WS 08/09

Aufgabe 1: (40 Punkte) Mehrere Verbindungen pro RDT-Instanz

Erweitere Dein Programm um die Fähigkeit, mit mehreren eingehenden Verbindungen umgehen zu können.

Jede Verbindung muß nun ihre eigenen Statemachines und ihre eigenen Puffer haben. Außerdem müßen innerhalb von `rdt_select()` folgende Erweiterungen eingebaut werden:

- `send_all()` ist für alle bestehenden Verbindungen aufzurufen statt nur für eine.
- Nach Empfang eines Paketes muß zuerst anhand des Paket-Absenders eine passende Verbindung gesucht und gegebenenfalls neu angelegt werden.
- Alle anderen Funktionen innerhalb von `rdt_select()` (nämlich `statemachine_send()` und `statemachine_recv()`) müssen gegebenenfalls zusätzlich mit der gefundenen/neuen Verbindung parametrisiert werden.
- `rdt_select()` muß die Verbindung auf der Daten eingegangen sind zurückgeben, sofern der Empfangspuffer dieser Verbindung nicht leer ist.

Folgende Funktionen müssen ebenfalls angepaßt werden:

- `rdt_send()` und `rdt_recv()` benötigen nun gegebenenfalls auch eine Verbindung als zusätzlichen Parameter. Die zu benutzenden Puffer müssen die der übergebenen Verbindung zugeordneten Puffer sein.

Aufgabe 2: (60 Punkte) RDT 3.0 Funktionalität

Erweitere Dein Programm um die Funktionalität von RDT 3.0 (verläßliche Übertragung bei Paketverlusten).

In RDT3.0 darf jetzt nur noch begrenzt lange auf ein ACK-Paket gewartet werden. Sollte innerhalb dieser Zeit (Timeout) kein passendes ACK empfangen worden sein, dann muß das letzte gesendete Paket erneut versendet werden.

Dazu sind folgende Dinge zu tun:

- `send_all()` Der Zustand der Statemachine muß um einen Timer-Wert ergänzt werden. Dieser Timer-Wert ist ein Zähler und wird beim Senden von Daten mit 6 vorbelegt. Er representiert den Zeitpunkt, bis zu dem ein ACK *spätestens* empfangen worden sein muß. Wir wollen höchstens 1000 Millisekunden (+ Jitter) lang auf ein passendes ACK warten.
- `rdt_select()` Hier muß der Methode `can_read()` ein Timeout-Wert übergeben werden. `can_read()` kehrt im Falle eines Timeouts mit einer leeren Liste zurück (`perldoc IO::Select`).

  Der Timeout-Wert ist so zu wählen, das `can_read()` *mindestens* regelmäßig alle 200ms zurückkehrt und zwar unabhängig davon, ob `can_read()` aufgrund von empfangenen Paketen zurückkehrt. Der Timeoutwert für `can_read()` muß also vor jedem Aufruf neu berechnet werden, und zwar abhängig vom letzten Rückkehr-Zeitpunkt (siehe dazu `perldoc Time::HiRes`).

  Sollte `can_read()` wegen eines Timeouts zurückkehren, dann sind *alle* bestehenden Verbindungen dahingehend zu überprüfen, ob der Timeout für die gegebene Verbindung relevant ist. Dazu werden die Timeout-Zähler aller Verbindungen um 1 erniedrigt und dann geprüft, ob der Wert

auf 0 gesunken ist. In diesem Fall liegt dann ein Timeout vor. Dieser Test ist innerhalb von `statemachine_send()` auszuführen.

Bei einem Timeout ist das letzte verschickte Paket erneut zu versenden und der Zähler wieder mit 6 zu initialisieren ($6 * 200\text{ms} > 1\text{sec} \geq 5 * 200\text{ms}$).

- `statemachine_send()` In dieser Funktion ist jetzt die Funktionalität der RDT3.0 Sender-Statemachine zu implementieren.

  Hier darf jetzt *nicht* mehr auf falsche ACKs und falsche Prüfsummen mit einem ACK mit falscher Sequenznummer reagiert werden! Stattdessen ist auf einen Timeout zu prüfen, und zwar anhand des zusätzlichen Timerwertes. Bei Timeout muß dann ein Retransmit erfolgen.

Abzugeben sind:

- Der Quelltext Deines Programmes

—Ende der Aufgabenstellung für das 7. Aufgabenblatt.—

Abgabe: Mittwoch, 10.12.2008, 23:59 s.t.