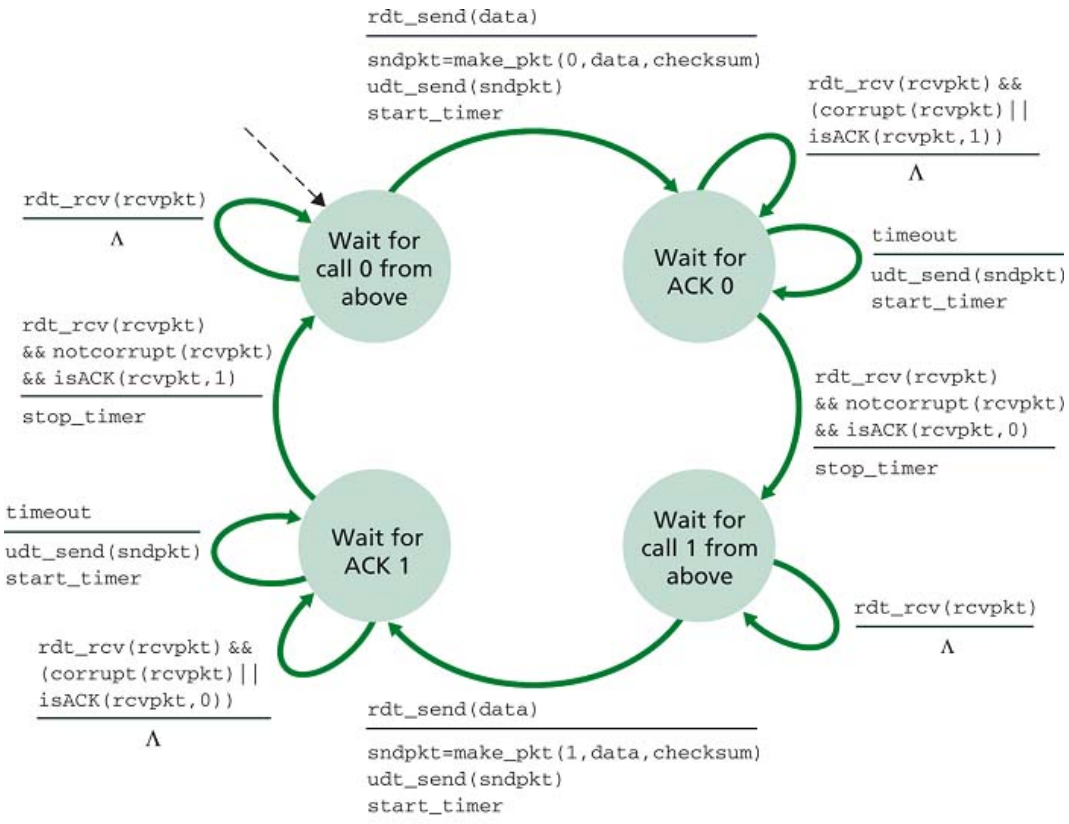
EECE 4830-5830: Network Design, Dr. Vinod Vokkarane

Programming Project Phase 4:
Implement RDT 3.0 over an unreliable UDP channel with bit-errors and loss

Deadline: Sunday, March 26th (Midnight) - Blackboard

Project description: The TCP/IP stack has five layers, namely application, transport, network, link, and physical. In Phase 1, each group implemented the standard user datagram protocol (UDP) sockets. The intention was to transfer a file (say JPEG) between a UDP client process and a UDP server process. In Phase 2 and Phase 3, we provided reliable data transfer service over the unreliable bit-error prone UDP connection. In Phase 4, you will have to enhance your sender code to handle two additional scenarios: 1. Data packet loss and 2. ACK packet loss. The FSM of the sender and receiver and shown below (Note: RDT 3.0 Receiver is same as RDT 2.2 Receiver):



rdt3.0 sender

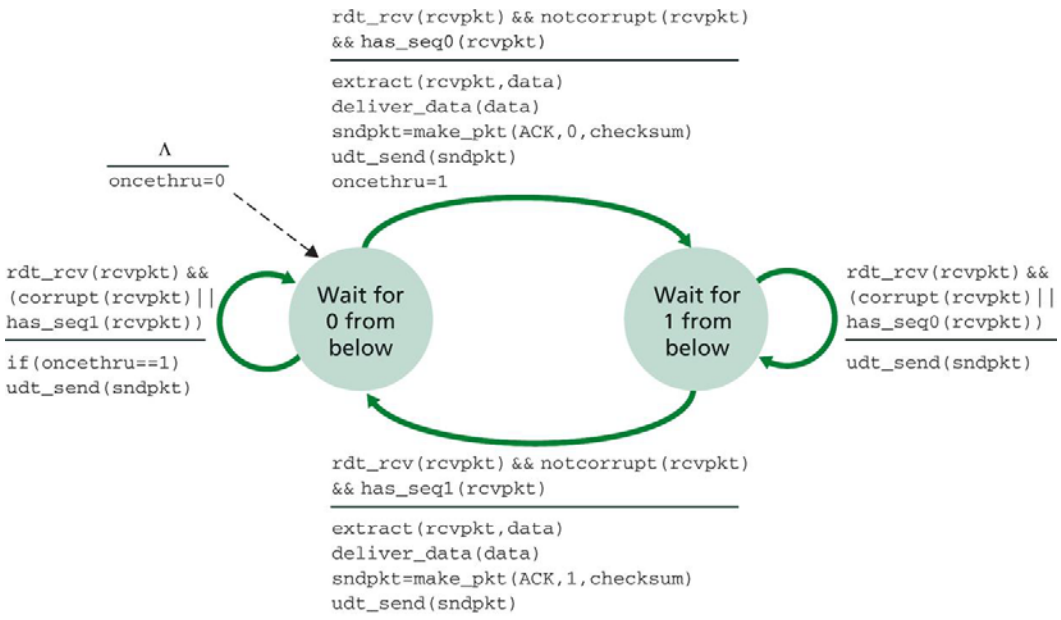


Figure 3.14 ♦ rdt2.2 receiver

The following are the basic implementation steps:

1. Pick a transfer file - JPEG image file (recommended), easier to identify loss of packets in an image file (lost pixels).

2. Make_packet function - parses the image file and breaks it down to several packets (set a fixed packet size, say 1024 bytes).

3. Use the UDP sockets (Phase 1) to send and receive RDT2.2 packets (Phase 3). UDP sockets are unreliable, so they simulate the unreliable Internet IP (network) layer.

4. Implement: Sequence Numbers (to identify duplicates), Checksum (implement your own, similar to UDP), and ACKs (remember, RDT 2.2 is a NAK-free protocol).

New Features for Phase 4:

5. Enhance sender to handle packet loss (RDT 3.0) by using a countdown timer.

6. The RDT3.0 receiver (similar to RDT2.2 receiver) should assemble packets in order and deliver the entire transfer file to the application.

In your implementation, use binary variables for representing bits (instead of strings).

In this phase, you will need to implement different data transfer scenarios

Option 1 - No loss/bit-errors.

Option 2 - ACK packet bit-error: you need to intentionally change the data bits of the received ACK packet at the sender (some error %) and implement suitable recovery mechanism.

Option 3 - Data packet bit-error: you need to intentionally change the data bits of the received DATA packet at the receiver (some error %) and implement suitable recovery mechanism.

Option 4 - ACK packet loss: you need to intentionally drop the received ACK packet at the sender (some loss %) and implement suitable recovery mechanism.

Option 5 - Data packet loss: you need to intentionally drop the received DATA packet at the receiver (some loss %) and implement suitable recovery mechanism.

Note:

- Use at least a 500KB file for transmission.
- Don't set your timeout to be too high (30-50ms should be enough).
- Each point on the plot can be an average of multiple runs at a given loss/error rate.
- For accuracy remove print commands and debug statements to the output console statements while measuring the completion time for the chart.

Extra Credit (50%): Implement an applet/GUI to show the data transfer and the (sender and receiver) FSM.

Expectations: In this phase of the project, you will learn about the basic principles used to provide non-pipelined reliable data transfer over a data channel with bit errors and packet losses.

Programming language: C, C++, C-sharp, Python or Java (your choice).

Deliverables:

1. **ReadMe.txt**: Name of the team members, list the names of files submitted and their purpose, and explain steps required to set up and execute your program. Also, how to check for the different scenarios (1 - no loss/error; 2 - ACK bit error; 3 - Data packet error, 4 – ACK packet loss, and 5 – Data packet loss).

2. **Updated Design Documents** (yourlastname.doc): updated detailed design document with possibly screen-shots of a sample scenario.
3. **Sender and Receiver source files** (yourlastname.tar.gz/.zip): well documented source code; mention ALL references for reuse of source code (if any).
4. **Transfer File** (No .EXE files): sample file used to test the functionality of your program. (File is Optional)
5. Plot charts for completion time for all the 5 data transfer options for the same file starting from 0% loss/error to 60% loss/error in increments of 5%.
6. **Individual Contribution (contribution.txt/doc):**
   a. Bulleted list of tasks implemented by each team member and
   b. Teammate rating between 1 (not good) – 5 (excellent) on
      - project time commitment,
      - design contribution,
      - coding contribution,
      - debugging contribution, and
      - report preparation (confidential submission for each member).

One team member can submit all your documents (except item 6) in a single compressed file with name **Student1LastName_Student2LastName_Phase4.zip**. Both team members should submit Item 6 (contribution rating).

References:

1. Socket Programming
   - JAVA Socket Programming
   - Linux Gazette's Socket Programming
   - http://www.programminglogic.com/sockets-programming-in-c-using-udp-datagrams/
   - Socket Programming by JAVA World and O'REILLY JAVA Network Programming
   - Python Socket Programming
2. UDP: RFC768
3. Principles of Reliable Data Transfer, Section 3.4.1, Page 206 -207, Kurose/Ross (6th Ed).