EECE 4830-5830 Network Design, Spring 2017, Dr. Vinod Vokkarane

Programming Project Phase 5:
Implement Go-Back-N protocol over an unreliable UDP channel

Deadline: Sunday, April 16th (Midnight) - Blackboard

Project description: 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, Phase 3, and Phase 4, we provided reliable data transfer service over the unreliable UDP connection. In Phase 5, you will have to enhance your code to handle pipelined data transfer. The sender has to maintain a buffer as shown below.

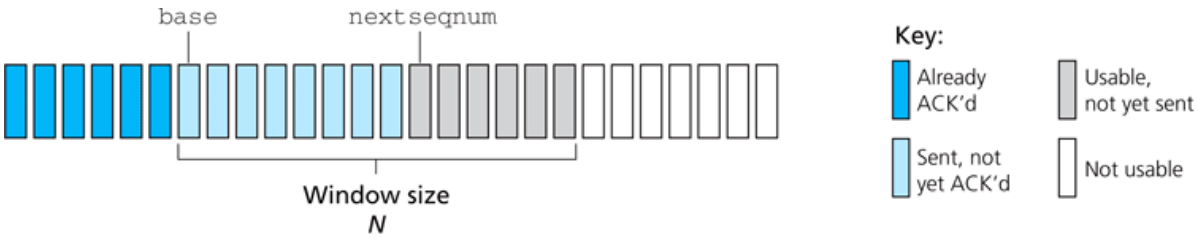

Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N

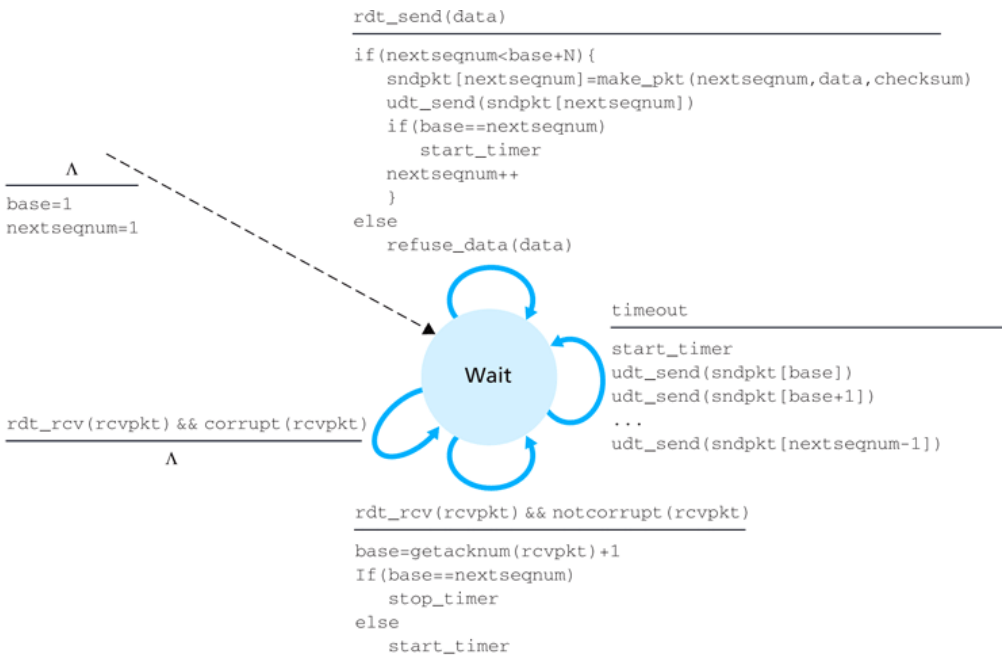

Figure 3.20 ♦ Extended FSM description of GBN sender

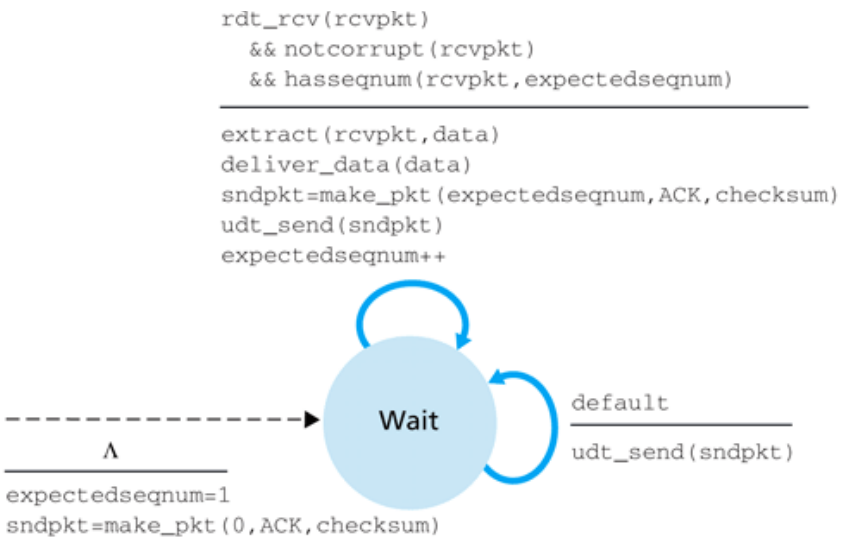

Figure 3.21 ♦ Extended FSM description of GBN receiver

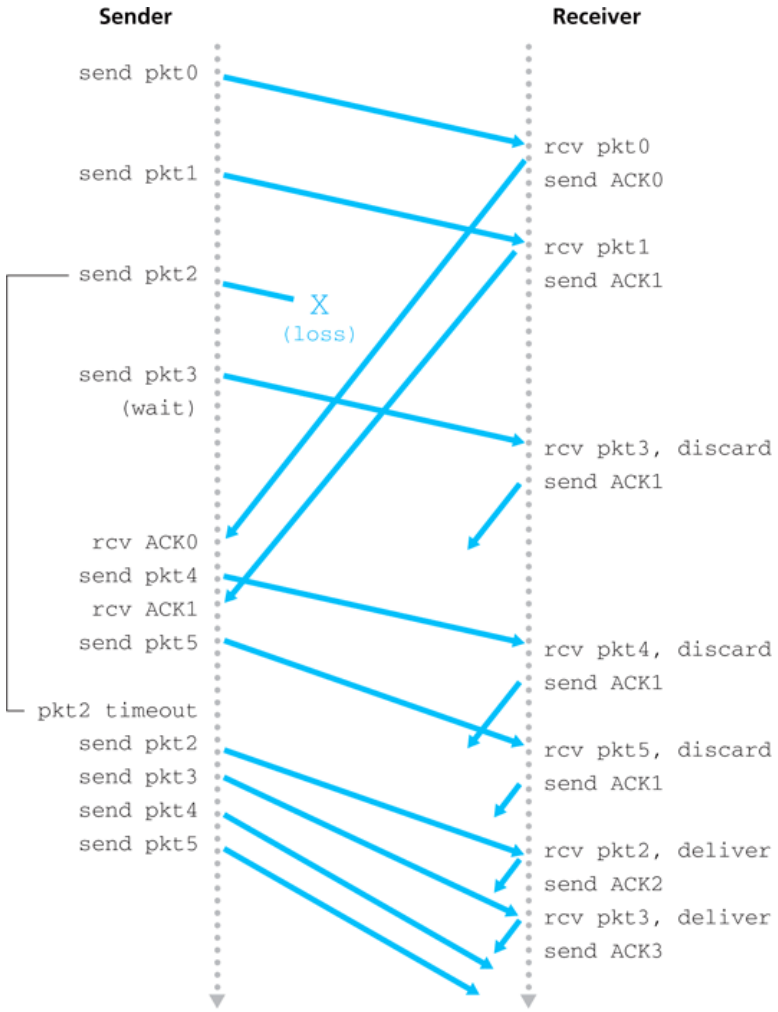

Figure 3.22 ♦ Go-Back-N in operation

The following are the basic implementation steps:

1. Pick a transfer file - BMP image file (recommended), easier identifying 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. Set a fixed window size (N).

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

5. Implement: Sequence Numbers (to identify duplicates), Checksum (implement your own, similar to UDP), ACKs, and a countdown timer (to handle packet loss).

6. The GBN 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.

IMPORTANT: You should be able to demo Phase 5 with and without loss recovery.

Performance Graphs

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.

Plot charts for completion time for all your 5 options for transferring the same file at 0% loss/error to 60% loss/error in increments of 5%.

Chart 1: Phase 5 performance
  - X-axis: Intentional loss probability (0% -50% range)
  - Y-axis: File Transfer Completion Time

Chart 2: Phase 5 performance with a fixed 10% loss probability
  - X-axis: Window Size (1, 2, 5, 10, 20, and 50)
  - Y-axis: File Transfer Completion Time

Chart 3: Performance comparison of the different phases at a fixed loss probability (say 10%) and a fixed bit-error probability (say 10%) and using the same transfer image
  - X-axis: Phase 3, Phase 4, Phase 5, (Selective Repeat and TCP - optional)
  - Y-axis: File Transfer Completion Time

Observe and conclude if there is an optimal window size and an optimal timeout value for your project.

Extra Credit:

1. **Implement an applet/GUI to show the data transfer (25%).**
2. **Implement an applet/GUI to show the (sender and receiver) FSM (25%).**
3. **Implement Selective Repeat protocol (75%) - Details refer the text book (Section 3.4.4).**
4. **Implement TCP- RDT, connection setup, connection teardown, dynamic window size, checksum, flow control, (Skip the final and get an A grade!) - Details refer the textbook (Section 3.5, pg. 230-258).**

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

Programming language: C, C++, 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 Image File** (Bitmaps preferred): sample file used to test the functionality of your program. (File is Optional)
5. Performance charts with detailed analysis of the results.
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_Phase5.zip**. Both team members should submit Item 6 (contribution rating). All submission will be inspected using the **Plagiarism detection software.**

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).