Cpr E 489: Computer Networking and Data Communications
Lab Experiment #5 – Error Detection and Go-Back-N ARQ Protocol
(Total Points: 100)

Objective

To write code to implement the sender part of the Go-Back-N ARQ protocol.

Pre-Lab

Read over the provided source code that has already been implemented (`secondary.c`, `utilities.h/.c`, `ccitt16.h`, `introduceerror.h/.c`, and the skeleton code in `primary.c`). Develop state machines and/or pseudo code to help implement your solution.

Lab Expectations

Work through the lab and let the TA know if you have any questions. After the lab, write up a lab report. Please make sure to:

1) **Attend** the lab. **(5 points)**
2) **Summarize** what you learned in a few paragraphs. **(25 points)**
3) **Submit** your well-commented code in a .zip file with your report. **(30 points)**
4) **Demo** your code to the TA. **(25 points)**
5) **Include** your answer to the exercise. **(15 points)**

Submit your **well-commented code** to the TA with your lab report for grading. Submit your report as a PDF file, and your code (`primary.c`) in a .c file.

Demonstration Policy

This lab will require you to **demo** your code to the TA. The lab may be demonstrated during:

- **this lab section, or**
- **the TA office hours, or**
- **the first hour of the next lab** (everyone will have a chance to demo once)

Problem Description

In this lab experiment, you are required to design and develop a sender function (`primary.c`) to implement the Go-Back-N ARQ protocol, with a Primary function (which will act as a client or sender) while the Secondary function (which will act as a server or receiver) is given to you. This Go-Back-N ARQ protocol is a revised version from the one we learned in class. One difference is that it uses **both ACK and NAK packets**; this is to simplify the implementation and avoid having to implement the timeout mechanism.

For CRC generation and error detection, you should use the provided object file (`ccitt16.o`). See the CRC notes below for more information.

You will be provided with the client (sender) and server (receiver) .o files. The sender establishes a TCP connection and hands control to your "primary" function, while the receiver accepts the TCP connection and hands control to your "secondary" function. The provided implementation already contains an example of two-way communication with comments explaining the different parameters necessary to send and receive data.

Primary Function "primary.c" (to be completed) (30 points)

- Send the alphabet, ABCDEFGHIJKLMNOPQRSTUVWXYZ, to the Secondary in a total of 13 packets (two characters per packet). NOTE: the total number of packets sent may be higher since the packets may be corrupted by `introduceError()` and thus need to be redelivered.

- Use function "`buildPacket`" that found in "`utilities.c`" to build a packet of the following format:

Packet Type	Packet Number	Data	CRC
1 byte	1 byte	2 bytes	2 bytes

  - **Packet Type**
    - 1: Data Packet (sent from Primary to Secondary)
    - 2: Acknowledgement Packet (ACK) (sent from Secondary to Primary)
    - 3: Negative Acknowledgment Packet (NAK) (sent from Secondary to Primary)
  - **Packet Number**
    - Starts from 0 and increments sequentially to 12
  - **Data**
    - Two alphabet characters (sent from Primary to Secondary)
    - No data is sent from Secondary to Primary
  - **CRC**
    - CRC generated for this entire packet, including Packet Type, Packet Number, and Data fields (see the section on CRC)

- Display the packet (`printPacket()`).
- Apply the "`introduceError.c`" routine to the entire packet. Pass the BER value to the program as an argument in the command line.
- Implement the Go-Back-N ARQ protocol with a send window of size N = 3:
  - The Primary sends the packet to the Secondary and keeps it in a buffer of unACKed packets until an ACK is received for this packet. The Primary displays an indication of sending the packet, together with its sequence number.
  - When the Primary receives an ACK from the Secondary, it adjusts the send window and removes the packet from its buffer. It also displays an indication of which packet was positively acknowledged.
  - When the Primary receives a NAK from the Secondary, it adjusts the send window and retransmits all the packets in the send window. This is to emulate the timeout mechanism. It also displays an indication of which packets were negatively acknowledged and were retransmitted.
  - There are no requirements for the purposes of this lab when there are less than N = 3 packets to send except that all packets must be acknowledged (ACK) before the sending cycle ends.
  - Notes: the following two optional functions are found in "`utilities.c`":
    `shiftWindow()`: used to shift the send window frame.
    `shiftBuf()`: used to shift the packet buffer frame.

Secondary Function "secondary.c" (already completed)

Note: The secondary function has already been completed and you don't need to make any modifications, but it's a good example to help you build the primary function.

- Accept data packets from the Primary.
- Run the CRC:
  - If the packet is received error free and in sequence, it displays the packet content and sequence number, then sends back an ACK.

- If the packet is received error free but out of sequence, it does not display the packet content but displays the sequence number, then sends back an ACK.
- If the packet is received in error, it does not display the packet content but displays the sequence number, if possible, then sends back a NAK.
- ACK and NAK packets are not corrupted.

Procedure

- Complete the Primary function as described above.
- Correctly transmit packets from the Primary (sender) to the Secondary (receiver) and observe the sequence of transmissions.
- Submit your (well-commented) code to the TA with your lab report for grading.

Exercise (15 points)

Run your program 6 times with each of the following BER values: 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, and graph the average number of transmission attempts per data packet for each BER value. Summarize your observations.

CRC Generation and Checking

An object file, `ccitt16.o`, is provided that will generate and check CRC for you. The function provided by `ccitt16.o` has the following prototype:

```
short int calculate_CCITT16(unsigned char cData[], unsigned int iLen,
      unsigned int iAction);
```

`iAction` is defined as either `GENERATE_CRC` or `CHECK_CRC` in the `ccitt16.h` header file:

`#define GENERATE_CRC 1`

- Returns the checksum of `cData[]` with length `iLen` as a short `int`

`#define CHECK_CRC 2`

- Uses the last two bytes of `cData[]` as CRC check bits to check `cData[]`; returns either 0 or 1:
  - `#define CRC_CHECK_SUCCESSFUL 0`
  - `#define CRC_CHECK_FAILURE 1`

Compiling and Running "sender" and "receiver" Programs

A makefile is provided, which allows you to compile and link all programs together with one command:

```
make
```

Then, run `receiver`, and have it listen on some port number (the port number does not have to be 50404).

```
./receiver 50404
```

Finally, run `sender`, and connect the sender program to the receiver program on a given IP and port number (that should match the receiver's). You must also pass the BER value that will be handed down to your primary function.

```
./sender 127.0.0.1 50404 0.001
```