Assignment 2

Computer Networks (CS 456)

Reliable Data Transfer: The Go-Back-N Protocol

Due Date: Monday, March 4th 2019, at midnight (11:59 PM)

Work on this assignment is to be completed individually

Assignment Objective

The goal of this assignment is to implement the **Go-Back-N** protocol, which could be used to transfer a text file from one host to another across an unreliable network. The protocol should be able to handle network errors such as packet loss and duplicate packets. For simplicity, your protocol is unidirectional, i.e., data will flow in one direction (from the sender to the receiver) and the acknowledgements (ACKs) in the opposite direction. To implement this protocol, you will write two programs: a sender and a receiver, with the specifications given below. You will test your implementation using an emulated network link (which will be provided to you) as shown in the diagram below:

When the sender needs to send packets to the receiver, it sends them to the network emulator instead of sending them directly to the receiver. The network emulator then forwards the received packets to the receiver. However, it may randomly discard and/or delay received packets. The same scenario happens when the receiver sends ACKs to the sender.

Note: The assignment description (data structure and program names) assumes an implementation in Java.

Packet Format

All packets exchanged between the sender and the receiver should have the following structure (consult `packet.java` provided with the assignment):

```
public class packet {
      … … …
      private int type;       // 0: ACK, 1: Data, 2: EOT
      private int seqnum;     // Modulo 32
      private int length;     // Length of the String variable 'data'
      private String data;    // String with Max Length 500
      … … …
}
```

The `type` field indicates the type of the packet. It is set to 0 if it is an ACK, 1 if it is a data packet, 2 if it is an end-of-transmission (EOT) packet (see the definition and use of an end-of-transmission packet below). For data packets, *`seqnum` is the modulo 32 sequence number of the packet*. The sequence number of the first packet should be zero. For ACK packets, `seqnum` is the sequence number of the *packet being acknowledged*. The `length` field specifies the number of characters carried in the data field. It should be in *the range of 0 to 500*. For ACK packets, `length` should be set to zero.

Sender Program (`sender`)

You should implement a sender program, named `sender`, on a UNIX system. Its command line input includes the following: `<host address of the network emulator>`, `<UDP port number used by the emulator to receive data from the sender>`, `<UDP port number used by the sender to receive ACKs from the emulator>`, and `<name of the file to be transferred>` in the given order.

Upon execution, the sender program should be able to read data from the specified file and send it using the Go-Back-N protocol to the receiver via the network emulator. *The window size should be set to N=10*. After all contents of the file have been transmitted successfully to the receiver (and corresponding ACKs have been received), the sender should send an EOT packet to the receiver. The EOT packet is in the same format as a regular data packet, except that its *type* field is set to 2 and its *length* is set to zero. The sender can close its connection and exit only after it has received ACKs for all data packets it has sent and an EOT from the receiver. To keep the project simple, *you can assume that the end-of-transmission packet never gets lost in the network*.

In order to ensure reliable transmission, your program should implement the Go-Back-N protocol as follows:
If the sender has a packet to send, it first checks to see if the window is full, that is, whether there are N outstanding, unacknowledged packets. If the window is not full, the packet is sent and the appropriate variables are updated and a timer is started if not done before. The sender will use only a single timer that will be set for the oldest transmitted-but-not-yet-acknowledged packet. If the window is full, the sender will try sending the packet later. When the sender receives an acknowledgement packet with sequence number n, the ACK will be taken to be a cumulative acknowledgement, indicating that all packets with a sequence number up to and including n have been correctly received at the receiver. If a timeout occurs, the sender resends all packets that have been previously sent but that have not yet been acknowledged. If an ACK is received but there are still additional transmitted-but-yet-to-be-acknowledged packets, the timer is restarted. If there are no outstanding packets, the timer is stopped. Further description of the GBN sender and receiver can be found in slides 47-50 of Module 3 Lecture Notes.

Output

For both testing and grading purposes, your *sender* program should be able to *generate two log files*, named as *seqnum.log* and *ack.log*. Whenever a packet is sent, its sequence number should be recorded in *seqnum.log*. The file *ack.log* should record the sequence numbers of all the ACK packets that the sender receives during the entire period of transmission. *The format for these two log files is one number per line. You must follow this format to avoid losing marks.*

Receiver Program (`receiver`)

You should implement the receiver program, named as `receiver`, on a UNIX system. Its command line input includes the following: `<hostname for the network emulator>`, `<UDP port number used by the link emulator to receive ACKs from the receiver>`, `<UDP port number used by the`

`receiver to receive data from the emulator>`, and `<name of the file into which the received data is written>` in the given order.

When receiving packets sent by the sender via the network emulator, it should execute the following:

- check the sequence number of the packet;
- if the sequence number is the one that it is expecting, it should send an ACK packet back to the sender with the sequence number equal to the sequence number of the received packet;
- in all other cases, it should discard the received packet and resends an ACK packet for the most recently received in-order packet;

After the receiver has received all data packets and an EOT from the sender, it should send an EOT packet then exit.

Output

The receiver program is also required to generate a log file, named as ***arrival.log***. The file *arrival.log* should record the sequence numbers of all the data packets that the receiver receives during the entire period of transmission. ***The format for the log file is one number per line. You must follow the format to avoid losing marks.***

Network Emulator (`nEmulator`)

You will be given **the executable code** for the network emulator. To run it, you need to supply the following command line parameters in the given order:

- `<emulator's receiving UDP port number in the forward (sender) direction>`,
- `<receiver's network address>`,
- `<receiver's receiving UDP port number>`,
- `<emulator's receiving UDP port number in the backward (receiver) direction>`,
- `<sender's network address>`,
- `<sender's receiving UDP port number>`,
- `<maximum delay of the link in units of millisecond>`,
- `<packet discard probability>`,
- `<verbose-mode>` (Boolean: Set to 1, the network emulator will output its internal processing).

When the link emulator receives a packet from the sender, it will discard it with the specified probability. Otherwise, it stores the packet in its buffer, and later forwards the packet to the receiver with a random amount of delay (less than the specified maximum delay).

Hints

- Use the `packet` class given in `packet.java` containing necessary declarations, definitions, and helper methods
- ***All the packets must be sent and received as `byte` arrays*** instead of as Java `packet` objects. Since the network emulator is written in C/C++, it cannot read Java objects. Necessary code to convert the `packet` class into `byte` array and vice versa is provided in the `packet.java` file.

- *You must run the programs in the CS Undergrad Environment* in order to allow `nEmulator` to work.
- Experiment with network delay values and sender time-out to understand the performance of the protocol.
- Run `nEmulator`, `receiver`, and `sender` on three different machines *in this order* to obtain meaningful results.

Example Execution

1. On the host **host1**: `nEmulator 9991 host2 9994 9993 host3 9992 1 0.2 0`
2. On the host **host2**: `java receiver host1 9993 9994 <output File>`
3. On the host **host3**: `java sender host1 9991 9992 <input file>`

Procedures

Due Date

The assignment is due on Monday, March 4th 2019, at midnight (11:59 PM)
Late submission policy: 10% penalty every late day, up to 3 late days. Submissions not accepted beyond 3 late days.

Hand in Instructions

Submit all your files in a single compressed file (.zip, .tar etc.) using LEARN.

You must hand in the following files / documents:

- *Source code* files.
- *Makefile*: your code **must** compile and link cleanly by typing "*make*" or "*gmake*".
- *README* file: this file ***must*** contain instructions on how to run your program, which undergrad machines your program was built and tested on, and what version of *make* and *compilers* you are using.

Your implementation will be tested on the machines available in the **undergrad environment**.

Documentation

Since there is no external documentation required for this assignment, you are expected to have a reasonable amount of internal code documentation (to help the markers read your code).

You **will** lose marks if your code is unreadable, sloppy, inefficient, or not modular.