Assignment 2 – Recovering from Packet Loss and Internet Delay

CS456/656 Computer Networks

Spring 2020

1) Objective

The goal of this assignment is to implement the **Go-Back-N (GBN)** protocol, which could be used to reliably transfer a text file from one host to another across an unreliable network. The GBN sliding window protocol is used to perform reliable data transfer and can handle network errors, such as packet loss, duplicate packets and delayed packets. The network emulator provided in the assignment will emulate the lossy network, which will randomly drop or delay packets. Both sender and receiver will communicate via the network emulator, so data and, or acknowledgements can be delayed or dropped.

2) Background

You will find detailed description of the GBN protocol for the sender and receiver in Chapter 3, Section 3.4.3 Go-Back-N (GBN) of the textbook.

You will also be provided with a network emulator 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>
```

When the link emulator receives a packet from the sender, it will discard it with the specified `<packet discard probability> probability`. Otherwise, it stores the packet in its buffer, and later forwards the packet to the receiver with a random amount of delay, which is less than the specified `<maximum delay of the link in units of millisecond>` maximum delay. The minimum value you can set for maximum delay is 1 (do not try 0). The `verbose-mode` can be set to 1 to output the internal processing of the network emulator.

3) Specifications

All packets exchanged between the sender and the receiver should have the following structure (consult `packet.java` provided with the assignment). In this assignment description, the data structure and program names are implemented in in Java, however, you are free to implement your assignment in your preferred programming language.

```
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
}
```

a) The `type` field indicates the type of the packet.
   i) 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
b) `seqnum` field is the sequence number of a packet
   i) For a data packet, `seqnum` is a modulo 32 sequence number and the sequence number of the first packet should be **zero**.
   ii) For ACK packets, `seqnum` is the sequence number of the packet being acknowledged.
c) The `length` field specifies the number of characters carried in the data field.
   i) Data packets length should be in the range of 0 to 500, inclusive.
   ii) ACK packets and `EOT` packet `length` should be set to zero.
d) The sender
   i) should read data from the specified file and send it to the receiver using GBN via the network emulator.
   ii) The window size for GBN should be set to 10, that is, `N=10`.
   iii) 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.
   iv) 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.
   v) To keep this simple, you can assume that the `EOT` packet never gets lost in the network.
e) The receiver should send acknowledgements for every data packet received from the sender via the network emulator. The acknowledgement packet should set the `seqnum` field to indicate the sequence number of the packet the receiver is acknowledging.
   i) If the sequence number in the data packet received from the sender, is the one that the receiver is expecting, it should send an ACK packet back to the sender with the `seqnum` equal to the sequence number of the received packet
   ii) In all other cases, it should discard the received packet and resend an ACK packet with the `seqnum` equal to the sequence number of the most recently received in-order packet

iii) Once the receiver has received all data packets and an `EOT` from the sender, it should send an `EOT` packet to the sender and exit.

4) Deliverables

a) A Sender Program (sender)

Implement a sender program, named sender, on a UNIX system. Its takes four parameters, in order,
`<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>`,
`<name of the file to be transferred>`

Upon execution, the sender reads data from `<name of the file to be transferred>` and sends it using the GBN protocol, with window size `N=10`, to the receiver via the network emulator. To send data packets, the sender first checks to see if the window is full, that is, whether there are `N` outstanding, unacknowledged packets. If the window is full, the sender will try sending the packet later.

If the window is not full, the packet is sent and the appropriate variables (from the GBN protocol) 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.
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 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.

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 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.

b) Sender Program Log Files

For both testing and grading purposes, your *sender* program should be able to generate two log files, named `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 both of these two log files is one number per line. You must follow this format to avoid losing marks.

c) A Receiver Program (receiver)

Implement a receiver program, named as receiver, on a UNIX system. It takes the following parameters in the order listed

```
<hostname for the network emulator>,
<UDP port number used by the 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>
```

The receiver should receive packets from the sender via the network emulator, and 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. Otherwise, it should discard the received packet and resend 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 also send an `EOT` packet to the sender and exit.

d) Receiver Program Log File

The receiver program is also required to generate a log file, named as `arrival.log`. The file log file 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.

e) Experiment Report

After implementing the GBN protocol, measure the impact of packet loss and delay on transmission time. The table below delineates cases and the parameter settings for the network emulator. For each case, where there is a combination of parameters, execute your program three times and record the average transmission time.

Table 1 Parameter settings of network emulator to investigate impact of packet loss and delay on transmission time

Case	Maximum Delay (ms)	Packet Discard Probability
Baseline	0	0
No Delay 1	0	0.1
No Delay 2	0	0.2

Case	Maximum Delay (ms)	Packet Discard Probability
No Delay 3	0	0.3
No Delay 4	0	0.4
No Delay 5	0	0.5
No Discard 1	10	0
No Discard 2	20	0
No Discard 3	30	0
No Discard 4	40	0
No Discard 5	50	0
Delay and Discard 1	20	0.1
Delay and Discard 2	20	0.2
Delay and Discard 3	20	0.3
Delay and Discard 4	40	0.1
Delay and Discard 5	40	0.2
Delay and Discard 6	40	0.3

Repeat these experiments for the provided small, medium, and large input files. Plot the results. Produce a report that illustrates your plots and discusses the results. Consider the impact of packet delay, packet discard probability, and file size (i.e. number of packets) on transmission time.

5) Hints

a) Use the packet class given in `packet.java, packet.cpp, or packet.py` containing necessary declarations, definitions, and helper methods.

b) Do not log `EOT` packets in `ack.log`. You can either log `EOT` packets in **both** `seqnum.log` and `arrival.log` or do not log it in any of them.

c) 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.

d) You should not bind UDP sockets you use for sending UDP packets.

e) Before the receiver receives packet #0, when receiving packets other than #0, either ACK -1 (i.e. #31) or do not ACK.

f) Open source file in '`r`' mode and the destination file in '`w`' mode. Treat `source_file` and `destination_file` arguments as absolute paths.

g) You must run the programs in the CS Undergrad Environment in order to allow `nEmulator` to work.

h) Experiment with network delay values and sender time-out to understand the performance of the protocol.

i) Run `nEmulator`, receiver, and sender on three different machines in this order to obtain meaningful results.

6) 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>

7) Instructions for Submission

Submit all your files in a single compressed file (.zip, .tar etc.) using Assignment 2 drop box in LEARN. Remember, you cannot share your program or work with any other student or in groups. This programming assignment is to be completely individually. You can use any programming language to design and implement the and server programs. You are expected to have a reasonable amount of code documentation, to help the graders read your code. You **will** lose marks if your code is not readable, or documented.

You must hand in the following files / documents:

- Source code files.
- Makefile: your code **must** compile and link 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 https://cs.uwaterloo.ca/cscf/internal/infrastructure/inventory/CS-TEACHING/hosts.
Please compile and test your code on these machines prior to submission. You should include scripts named sender.sh and receiver.sh that run your code with given arguments. Any code that fails the automated grading process is subject to a penalty.

The late policy is a 10% penalty for each day (24hrs) late up to a maximum of 3 days. Submissions will not be accepted beyond 3 late days.

8) Grading Rubric

No	Scenario Description	Expected Response	Failure Penalty
Automated Grading (70%)			
1	Test 1 a file less than 32 packets, no delay, no loss.		15%
2	Test 2 a file larger than 32 packets, no delay, no loss		10%

No	Scenario Description	Expected Response	Failure Penalty
3	Test 3 a file less than 32 packets, high loss, no delay.		10%
5	Test 4 a file less than 32 packets, medium loss, no delay.		5%
6	Test 5 a file less than 32 packets, low loss, high delay.		5%
7	Test 6 a file less than 32 packets, low loss, low delay		5%
8	Test 7 a file less than 32 packets, high loss, high delay.		10%
9	Test 8 a file larger than 32 packets, low loss, high delay.		10%
Experiment (20%)			
10	Record the time of sending the file in `time.log`.		2.5%
11	A report containing the result of the experiments	Plot(s) results are consistent with expectations when drop rate, delay, or file size increases (10%) Plot(s) are appropriately labelled (2.5%) Clear description of the experiments that are run and a plot showing the results (5%)	17.5%
Readme/Makefile/Shell Scripts (5%)			
12	Appropriate scripts Scripts exist for running the sender and receiver (1%), are correctly named (1%), and correctly run the code with given arguments (1%)		3%
13	Readme (1%) and Makefile for all compilations and dependency installations if any (no need for a Makefile for Python implementation) (1%)		2%
Style of Coding (Comment/Code)			5%

9) **Additional Notes**

a) **Grading for each automated tests**

- 50% for correct transmission of the file, should be an exact match byte by byte checked by `diff `source_file destination_file`

- 50% for correct logging in `seqnum.log`, `ack.log`, `arrival.log` (dynamic check, partial grade possible).

b) Test1 – Test8 are graded automatically and if your code doesn't run as instructed, you won't receive their marks.
c) Your code should finish running in at most 30 seconds in all tests (use a reasonable timer). Most implementations finish in much less time. We use relatively small files for grading (< 20 KB for high drop cases and < 32 KB for low drop cases).
d) During the tests, `sender` and `receiver` may reside on the same or different machines.

Penalties. Any changes to the code after the submission deadline is subject to a penalty.

- **[5%]** Small changes to fix permissions on the source files, naming log files incorrectly, writing the destination file in the incorrect path.
- **[10%]** Formatting changes such changing the `id` of first packet from 1 to 0, fixing the logging of `EOT` packets, fixing missing or extra newline/NULL characters in the destination file. Technical bugs unrelated to GBN such as fixing the binding of the UDP socket in the sender side, threads not exiting after finishing the transmission.
- **[15%]** Small hot fixes in the sending/receiving (GBN) logic.
- **[>= 30%]** Significant changes in the GBN logic of sending/receiving data and ACK packets.