CS 421 – Computer Networks
Programming Assignment II

Implementing a Reliable Transfer Protocol over UDP

Due: *December 17, 2023 at 11:59 PM*

This assignment should be submitted according to the submission rules defined at the bottom of this document.

Introduction

In this programming assignment, you are asked to implement a **Java or Python** program for simulating the operations of a sender which uses **Go-Back-N (GBN)** over a lossy network layer and write an application layer program using **User Datagram Protocol (UDP) as the underlying transport layer protocol.** Your program is required to transfer a specified image. **GBN** receiver is provided to you as a Python program (tested for Python 3.7.7) to your convenience. You will need to use threads to implement **GBN.**

Go-Back-N Receiver Program

The receiver implementation is provided to you as a Python3 program. You can run the receiver as follows:

```
python3 receiver.py <IP> <Port > <packet_loss_probability> <max_packet_delay >
```

The receiver binds to the IP **(IP)** address at the specified port **(Port**) and starts listening to incoming data in accordance with the GBN protocol. IP address should be entered as "127.0.0.1", which is the local host. It also contains the implementation of the receiver side of the file transfer application. Since you will be running both the sender and the receiver on the same machine, there will not be significant packet losses or delays in reality. Hence, packet losses are emulated by the receiver code that is provided to you by intentionally not sending the ACK segments with some specified probability p **(packet_loss_probability)**. Similarly, the network delay is emulated such that the receiver sends each ACK segment after a delay randomly chosen from the interval [0, **max_packet_delay**] in milliseconds starting from the reception of the data packet.

Over the course of the transmission, the program will continuously write the accepted packages to the file **"received.png**". Once the receiver receives the termination signal, it will first display how long the transmission took. You can use that info while preparing your plots for the report. You should make sure

that the content and size of this image is identical to the original image to see if your program works correctly.

Go-Back-N Sender Program

This is the program you're asked to implement. In the application layer, you will be responsible to send an image (a total of 4.266.854 bytes), which is provided to you inside this assignment folder (image.png). Your program should be written such that it accepts the path of this file as a command line argument and should send it to the receiver program using the GBN protocol.

For reliable data transfer, you should implement GBN sender over UDP. The details of the GBN sender algorithm can be found in Chapter 3 of your textbook. You should implement every aspect of the GBN sender including the countdown retransmission timer. Your program must accept the **window size (N)** in segments, and the **timeout interval** in milliseconds as command-line parameters. Your sender must divide the file to be transferred into UDP segments each containing a maximum of **1 KB (1024 bytes)**, including **2 bytes of header** indicating the sequence number of the segment and **a maximum of 1022 bytes of data**. Please note that the number of data bytes in the last segment may be smaller than 1022, since the file size is not necessarily an integer multiple of 1022 bytes.

The sequence number of each UDP segment is expected to be the first **2 bytes** of the payload of the UDP segment in **Big-Endian** form (see the API for implementation). In big endian, you store the most significant byte in the smallest address. Similarly, acknowledgment numbers received from the receiver will be the first (and only) **2 bytes** of the payload of the acknowledgments carried as UDP segments. The receiver assumes sequence numbers to start from **1** and to be incremented by 1 for each segment sent.

Since the receiver is not aware of the size of the file being sent, the sender should signal the end of transmission to the receiver by sending an empty header of all zeros, meaning **two bytes of zeros** only without data. Note that no artificial delay/drop is applied to this special segment, for your convenience. Therefore, you should terminate the sender program after sending this signal without waiting for an acknowledgement.

To summarize the command-line arguments to your program, the usage of the sender program should be as follows:

```
java Sender <file_path> <receiver_port> <window_size_N> <retransmission_timeout>
```

or

```
python3 Sender <file_path> <receiver_port> <window_size_N> <retransmission_timeout>
```

Note that this command should be able to run properly when your terminal points to the assignment folder (use 'cd <folder>' consecutively to move to the assignment folder). You can compile your Sender source code from the terminal using:

```
javac Sender.java
```

Threading

In GBN, a retransmission timer for the entire window must be kept. Your program must retransmit all previously transmitted segments in the window once the timer expires while listening for incoming acknowledgements at the same time. Handling such a task requires **concurrency**. From a simplified perspective, concurrency is the concept of handling multiple tasks "almost" at the same time.

Although there are lots of ways, design patterns and primitives to implement concurrency, we can achieve the required amount of concurrency in this assignment by using threads in Java. By running multiple threads which are assigned different tasks (such as retransmission of different segments if timeout occurs), we can run those different tasks at the same time.

Here, we outline a sample design pattern that you can use in your programs. According to this design, a separate thread is initialized in the main thread for the sender window. A draft for the run() method of this thread is provided below:

```
public void run() {
	try {
		while(true) {
			// Send packets in window
			for packet in packets:
				socket.send(packet);

			// Wait for main thread notification or timeout
			Thread.sleep(timeout);
		}
	}

	// Stop if main thread interrupts this thread
	catch (InterruptedException e) {
		return;
	}
}
```

A thread with a run() method like this will first send its "packet" through the DatagramSocket "socket", then wait for "timeout" milliseconds until retransmission. This happens in an infinite loop, which will only be broken by an external intervention from the main thread. In this design pattern, the main thread calls Thread.interrupt() method of the corresponding thread when it receives the ACK for all segments sent within that window. We recommend you create two separate threads: one for sending the data and one for listening to the acknowledgements. This way, you can update your window when a new ACK is received and also send the data when needed.

For Python, you may use threading and multiprocessing modules to achieve similar results.

Report

In addition to all the codes that you have written for this programming assignment, you need to submit a report. In your report, you need to provide the following 2 plots and include a discussion about how your results regarding these plots relate with what you have learned in class:

- *Average Throughput (in bps) vs. loss rate (p) for p ∈ {0, 0.1, 0.2, 0.3, 0.4, 0.5}.*
  Use maximum delay = 100 ms, timeout = 120 ms, N = 20.
- *Average Throughput (in bps) vs. window size (N) for N ∈ {20, 40, 60, 80, 100}.*
  Use maximum delay = 100 ms, timeout = 120 ms, p = 0.1.
- *Average Throughput (in bps) vs timeout for timeout ∈ {40, 80, 120, 160, 200, 240}*

The average throughput, in bps, during the entire transfer must be calculated by dividing the size of the file, in bits, by the total file transfer time.

Final Remarks

- For UDP communication, you should use DatagramSocket and DatagramPacket classes in Java and socket module for Python.
- You should run the receiver program first, then run your sender program.
- DO NOT print anything to the screen until the file transmission is complete in the final version of your code since this might seriously affect the performance.
- DO NOT use ImageIO while reading the input image. Your code should be working with ANY kind of input file, without paying attention to its content. It should treat the input as a series of bytes. The reason you are given an image is to make it easier to compare the received file and the input visually. You can use FileInputStream to read any files as byte arrays.

- Note that we will be testing your code with different files. Therefore, you might want to check your program with different input files before you submit.
- You can modify the source code of the receiver for experimental purposes. However, do not forget that your projects will be evaluated based on the version we provide.
- We have tested that these programs work with the discussed Java-Python combination.
- You might receive some socket exceptions if your program fails to close sockets from its previous instance. In that case, you can manually shut down those ports by waiting them to timeout, restarting the machine, etc.
- No third-party packages/sources are allowed.
- It is recommended that you start your implementation with a smaller image of your own choice for faster progression and use the image we provided to prepare your plots.
- Please contact your assistant if you have any doubt about the assignment.

Submission rules

You are required to comply with the following rules in your submission. You will lose points otherwise or if your program does not run as described in the assignment above.

- Your programming assignment must be submitted through corresponding link in the course Moodle page.
- All the files must be submitted in a zip file (not a .rar file, or any other compressed file) whose name must include your name and student ID. For example, the file name must be "AliVelioglu20141222" if your name and ID are "Ali Velioglu" and "20141222", respectively.
- Each submission should include a README file in the zip archive that provides the name(s) of the authors and a brief summary of contents and structure of the source code(s) written.
- Grouping with another student from the same section is allowed.
    - If you are submitting as a group of two students, the file name must include the names and IDs of both group members. For example, the file name must be "AliVelioglu20141222AyseFatmaoglu20255666" if group members are "Ali Velioglu" and "Ayse Fatmaoglu" with IDs "20141222" and "20255666", respectively.
    - For group submissions ONLY one member must make the submission. The other member must NOT make a submission.

- All the files must be in the root of the zip file; directory structures are not allowed. The archive should not contain any file other than the source code(s) with .java or .py extension and the README file. The archive should not contain these:
    - Any class files or other executables,
    - Any third-party library archives (i.e., jar files),
    - Project files used by IDEs (e.g., JCreator, JBuilder, SunOne, Eclipse, Idea or NetBeans, etc.). You may, and are encouraged to, use these programs while developing, but the end result must be a clean, IDE-independent program.

The standard rules for plagiarism and academic honesty apply; if in doubt refer to http://ascu.bilkent.edu.tr/Academic_Honesty.pdf.