CS 421 – Computer Networks Programming Assignment II

Implementing a Reliable Transfer Protocol over UDP

Due: December 20, 2024 at 11:59 PM

Introduction

In this programming assignment, you are asked to write the Java or Python code for simulating the operation of a Selective Repeat sender over a lossy network layer and the code for transferring a specified file as an application-layer program. Selective Repeat receiver entity is supplied to you as a **Python** program (tested in Python 3.7) for your convenience. You will use threads to implement Selective Repeat protocol over the User Datagram Protocol (UDP), which on its own does not provide a reliable service to applications.

Selective Repeat Sender

This is the part you are asked to implement. Basically, at the application layer, you will send the image.png file (4,266,854 Bytes), which is given as a part of this assignment. Your program must accept the path of the file to be sent as a command-line argument and it must send this file to the receiver program using the Selective Repeat protocol.

For reliable data transfer, you should implement Selective Repeat sender over UDP. The details of the Selective Repeat sender entity can be found in course slides. You should implement every aspect of the Selective Repeat sender including the countdown retransmission timer for each segment. Your program must accept the **window size**, N, and the **timeout interval** in milliseconds as command-line parameters. Your sender must divide the file to be transferred into chunks (segments) each containing a maximum of **1 KB (1024 bytes)**, including **2 bytes** of header indicating the sequence number of the segment. Please note that the size of the last segment may be smaller than 1 KB, 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 packet in **big-endian** form. 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 acknowledgment packets carried as UDP segments. The receiver assumes that sequence numbers start from **1** and it is incremented by 1 for each segment. Note that it is possible to implement Selective Repeat with only 2N unique sequence numbers; however, all sequence numbers are unique in this assignment to make it easier to implement.

Since the receiver does not know the size of the file being sent, the sender should signal the end of transmission to the receiver. After your sender program sends all the segments and receives all the acknowledgements, it should send **2 bytes of zeros**, which is an indicator for the end of the transmission. Note that no artificial delay/drop is applied to this special packet, for your convenience. Therefore, you should terminate the sender program after sending this signal without checking for acknowledgements.

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

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

```
python Sender.py <file_path> <receiver_port> <window_size_N> <retransmission_timeout>
```

Selective Repeat Receiver

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

```
python receiver.py <bind_port> <window_size_N> <packet_loss_probability_p>
<max_packet_delay_Dmax>
```

The receiver binds to the local host address (127.0.0.1) at the specified port (bind_port) and implements Selective Repeat protocol receiver entity. 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. Therefore, packet losses are implemented by the receiver code that is provided to you. The receiver drops the data packets with some specified probability **p**, and it delays the reception of the data packets with the amount of delay randomly chosen from the interval [0, **Dmax**]. This will emulate the effects of network delay. Dmax is in milliseconds.

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. Then, it will write the file it received to the same directory as itself, named as **received.png**. This process may take a few seconds. 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.

Threading

In Selective Repeat, each segment has its own retransmission timer, which are independent from each other. Your program must retransmit a segment once its timer expires while keeping track of other segments' timers and 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. 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, N (Selective Repeat window size) separate threads are initialized in the main thread for each data segment to be sent. A draft for the run() methods of these threads are provided below:

```
public void run() {
	try {
		while(true) {
			// Send packet
			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 the packet of that thread. A rough pseudocode for the main thread according to this design can be as follows:

```
loop until all packets are sent and all acks received {
        for segment_no from nextseqnum to send_base + N – 1 {
                start sender thread for segment_no;
        }

        ACK = receive_ACK(socket);

        if ACK in [send_base, send_base + N - 1] {
                if thread with segment_no == ACK is still running {
                        thread.interrupt();
                        advance send_base;
                }
        }
}
```

Please note that these pseudocodes and this design pattern are just examples to give you some idea about how the concurrency requirement of the assignment can be fulfilled. You are free to implement the program in any way you want as long as you achieve concurrency. That is, you can use more advanced concurrent design patterns, techniques, primitives, etc. Also, keep in mind that even if you decide to use the pseudocodes we provide, you might still need to make changes to them since they are oversimplified drafts and do not fully cover the requirements of the assignment.

Report

You will transfer the image.png file given to you with this assignment and measure the Average Throughput for this file transfer under different settings as listed below. In your report, you need to provide the following plots and include a discussion about how your results 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 Dmax = 150 ms, timeout = 180 ms, N = 30.
- Average Throughput (in bps) vs. window size (N) for N ∈ {10, 30, 50, 70, 90}. Use Dmax = 150 ms, timeout = 180 ms, p = 0.1.

- Average Throughput (in bps) vs timeout for timeout ∈ {60, 100, 140, 180, 220}. Use Dmax = 150 ms, p = 0.1, N = 30.

Final Remarks

- For UDP communication, you should use DatagramSocket and DatagramPacket classes in Java, and socket.SOCK_DGRAM in 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.
- Sending an image is not different than sending a txt file, or any other file. Therefore, 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. For Java, DO NOT use ImageIO while reading the input image. You can use FileInputStream to read any files as byte arrays. For Python, do not use any image reader libraries, like PIL. The standard 'open' function can handle the bytes if you supply correct parameters.
- 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 uncomment line 64 of the receiver.py to investigate what the receiver has received.
- 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.
- 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.
- Please contact your assistants furkan.guzelant@bilkent.edu.tr or bora.tuncer@bilkent.edu.tr if you have any questions about the assignment.

Submission rules

You need to apply all the following rules in your submission. **You will lose points if you do not obey the submission rules below or your program does not run as described in the assignment above.**

- The assignment should be submitted to respective Moodle page.
- You need to do the project individually.
- The name of the submission should start with [CS421_PA2], and include your name and student ID. For example, the name must be

  [CS421_PA2]AliVelioglu20111222

  if your name and ID are Ali Velioglu and 20111222,. You can work in groups of two.
- All the files must be submitted in a zip file whose name is given above.
- The standard rules for plagiarism and academic honesty apply; if in doubt refer to Academic Integrity Guidelines for Students and Academic Integrity, Plagiarism & Cheating.