Assignment 2

Due date: December 2, 2013 at 20:00 CEST

This is an individual assignment. You may discuss it with others, but your code and documentation must be written on your own.

Introduction

The goal of this assignment is to implement a modified version of the *Go-Back-N* transport protocol, which we will call *SR-Go-Back-N* protocol. This protocol allows applications to send a stream of data reliably, using an unreliable datagram network service.

You must implement a unidirectional transport-level service that uses the *SR-Go-Back-N* protocol. To do that, you must implement two Java classes, **SGBNSender** and **SGBNReceiver**, that implement the two sides of the transport layer. Your classes must be integrated with the **transport** library that you can download from the course web page. In fact, **SGBNSender** and **SGBNReceiver** must extend the **transport.Sender** and **transport.Receiver** base (abstract) classes, respectively. On the course web page you can also find two simple examples showing the use of the transport library and how to override its methods.

SR-Go-Back-N Specifications

SR-Go-Back-N is very similar to *Go-Back-N* as specified in the textbook. As in *Go-Back-N*, the *SR-Go-Back-N* sender maintains a sliding window of up to W pending segments. That is, segments that have been sent but not acknowledged. As in *Go-Back-N* the window size is fixed. Acknowledgments are cumulative, so the window "slides" by updating the *base* of the pending sequence whenever the sender receives an acknowledgment for a sequence number within the window. The acknowledgment must carry the sequence number of the last segment delivered in order.

Differently from *Go-Back-N*, in *SR-Go-Back-N* segments received out of order are not discarded. The receiver stores them and sends one or more special packets to the sender (*SR* packet) to ask for the missing packets.

Whenever the sender receives a *SR* packet it resends the related packet. Exactly as in *Go-Back-N*, the sender keeps a timeout and resend from the base whenever the timeout expires.

Requirements

In your implementation, **SGBNSender** must implement the following methods:

- **reliableSend** responds to requests from the application layer.
- **unreliableReceive** receives packets from the network (e.g., acknowledgments).

while **SGBNReceiver** must implements the following methods:

- **unreliableReceive** receives packets from the network (e.g., data packets from the sender).

Within your code, you can use the methods provided by the **transport.Network** class. The on-line documentation describes them all in detail. These are the most important ones:

- **unreliableSend** sends a packet through the network to the other endpoint.
- **setTimeout** and **cancelTimeout** starts and cancel a timer, respectively. The action specified with setTimeout is executed when the timer expires (if the timer is not canceled before that time).
- **blockSender** blocks the sender application, preventing further calls to **reliableSend**. The sender application can be resumed with the **resumeSender** method. You may have to block the sender at the transport-level when the window is full.

Notice that the transport library does not have a shutdown method for the sender, so the sender application will simply disconnect and terminate as soon as your transport level implementation is done sending the last segment. However, according to the Go-Back-N specification seen in class, the receiver application will not realize that the sender has terminated and will simply wait for more data, therefore your implementation of the receiver must have an appropriate termination condition (e.g., after a reasonably long timeout).

References and Tools

Check out the lecture notes about reliable data transfer as well as Section 3.4 of the textbook. Also, consult the documentation of the transport library to familiarize yourself with its operations and methods. Discussing your solution with the TA before the deadline is not only allowed but it is also encouraged.

Testing your Implementation

The **transport** library implements two applications, **FileSender** and **FileReceiver**, which can be used to test your implementation. As the names suggest, these applications exchange a file. The syntax to call them from the command line is:

java -cp transport.jar:. transport.FileSender *sender-impl* \
  localport hostname remoteport filename [error]

java -cp transport.jar:. transport.FileReceiver *receiver-impl* \
  localport hostname remoteport filename [error]

Where:

- *sender-implementation* and *receiver-implementation* are the class names of the implementation of the transport protocol you want to use. For example, you should set them to **SGBNSender** and **SGBNReceiver** to test your implementation.
- We use **-cp transport.jar:.** to set the classpath to the class files of the library and to the current directory, where you should have your class files
- *localport hostname remoteport* have the usual meaning.
- *filename* is the file read and sent by the sender, or received and written by the receiver.
- *error* is the percentage of lost packets. Notice that these errors are introduced in addition to the errors that might occur naturally across the network.

Notes on the Grading Policy

You are asked to implement the *SR-Go-Back-N* protocol, not *Go-Back-N*. This means that the extra features of this protocol are necessary to obtain a passing grade. A minimal level of quality of your code is also necessary to obtain a passing grade. In particular, you should follow the following general guidelines:

- use meaningful variables names;
- write modular code, for functionality but also for readability. For example, method bodies are typically less than 50 non-comment lines of code. If your code has overly long method bodies, you should consider breaking them into smaller logical units each implemented in a separate method;
- indent your code, and do that consistently; and
- document your code. Do not paraphrase the code in English. Instead, explain the *purpose* of non-obvious code elements (methods, variables, blocks, variables, conditionals, etc.).

Submission Instructions

You must submit a single *tar.gz* or *zip* archive containing only three files: **SGBNSender.java**, **SGBNReceiver.java** and **README.txt**. The **README.txt** file should contain a brief description of your implementation, possibly a list of limitations or errors you are aware of but that you were not able to fix, and *clear references to any and all external material you might have used,* including discussions with or help from other students.

Do not include any other files or folder. In particular, you may use the text editor or the IDE of your choice, but do not include project files and folders. Name your archive file following this format: **assign02-***lastname-firstname***.tar.gz**. Submit the *tar.gz* or *zip* archive through the Moodle system. The deadline is firm.