Assignment 2: Go-Back-N-T

Due date: *Wednesday, May 13, 2020 at 22:00*

> *This is an individual assignment. You may discuss it with others, but your code and documentation must be written on your own. You must properly reference any and all sources you used.*

You must implement a variant of the Go-Back-N reliable data transfer protocol called *Go-Back-N-T*. The implementation must be done in Java on top of the *transport* Java library available from the course page. In particular, you must implement a unidirectional transport-level service by providing two Java classes called *GBNTSender* and *GBNTReceiver* that realize the two sides of the transport layer and that implement *transport.Sender* and *transport.Receiver* interfaces, respectively.

Go-Back-N-T Specification

Go-Back-N-T is a simple extension of the Go-Back-N protocol as specified in the textbook. As in Go-Back-N, a Go-Back-N-T sender maintains a sliding window of up to W pending segments, that is, segments that have been sent but not yet acknowledged. As in Go-Back-N, W is a fixed parameter.

As in Go-Back-N, the receiver acknowledges all the segments received in the correct order, and discards out-of-order segments. Also, acknowledgments are cumulative, so the sender window "slides" by updating the *base* of the pending sequence whenever the sender receives an acknowledgment for a sequence number within the window. The sender resends all pending segments if it does not receive any valid acknowledgment after a certain time.

In addition to these standard features, Go-Back-N-T has an *adaptive timeout*. The sender resends all pending segments if it does not receive any valid acknowledgment after a certain time. This timeout value is computed as in TCP, based on a running estimation of the round-trip time (RTT). In particular, the sender measures the real RTT using acknowledgments. A measurement is the time elapsed between the departure of a segment and the arrival of the acknowledgment for that segment. The sender then estimates a stable RTT value using an exponential weighted moving average (EWMA) of the individual measurements, and similarly it also computes the RTT *variability*, also as an expontential weighted moving average. The sender then computes its timeout values based on the average RTT and average RTT variability, exactly as in TCP.

Transport Library API

Your *GBNTSender* must implement the *transport.Sender* interface. In particular, *GBNTSender* must implement a *reliableSend* method to handle requests from the application layer, and an *unreliableReceive* method to handle packets from the network (e.g., acknowledgments). Similarly, *GBNTReceiver* must implement the *unreliableReceive* method to handle 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 most important ones are:

- *unreliableSend* sends a packet through the network to the other endpoint.
- *setTimeout* and *cancelTimeout* start and cancel a timer, respectively. The action specified with *setTimeout* is executed when the timer expires (if the timer is not cancelled before that time). The delay that you have to specify in *setTimeout* is the one computed using the RTT statistics.

- *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 is rather simple, and does not have a connect procedure. So, the sender and receiver classes must be initially synchonized with respect to sequence numbers. Also, there is no shutdown procedure, 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).

Consult the documentation of the transport library to familiarize yourself with its operations and methods.

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 *GBNTSender* and *GBNTReceiver* 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. In order to test the RTT estimation you should run the applications on different machines.
- *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.

Submission Instructions

You should submit a single zip or tar archive containing only three files: GBNTSender.java, GBNTReceiver.java and README. *Do not include any other files or folder*. Make sure that you include all the necessary components to build and run your solution on a standard installation of a Java environment. In particular, make sure your solution works with the most basic command-line tools, outside of any integrated development environment. You may use the IDE of your choice, but do not include their project files and folders. Name your archive file following this format: `assign02-<surname>-<name>`{*.tar.gz,.zip*}.

Add comments to your code to explain sections of the code that might not be clear. Use the README file to add general comments to properly acknowledge any and all external sources of information you may have used, including code, suggestions, and comments from other students. If your implementation has limitations and errors you are aware of (and were unable to fix), then list those as well in the README file.

Submit your solution package through the iCorsi system.