CS 421 – Computer Networks
Programming Assignment II
Application Layer Reliable Protocol

Due: 7 May 2018

In this programming assignment, you will program a client which communicates with a server reliably. Your client program will use UDP at the transport layer and ensure reliability at the application layer. Your client must be a console application (no graphical user interface, GUI, is required) and should be named as RDTClient (i.e., the name of the class that includes the main method should be RDTClient). The server code will be provided with the assignment.

Reliable Communication

Your client program will send data to and receive data from the provided server. Since you will most probably test your client code on a single machine, i.e., your own, the server we provide will simulate loss through the random dropping of messages. The server will drop the following message with some probability:

- Your client's outgoing data message
- Your client's outgoing acknowledgment message
- The server's outgoing data message
- The server's outgoing acknowledgment message

You will implement rdt3.0 as described in Chapter 3 (Transport Layer) of the textbook by Kurose and Ross at http://www.cs.bilkent.edu.tr/~tugrul/CS421/Chapter3.pdf and given in Figure 1. For this reason, you will need to use messages with sequence numbers as in:

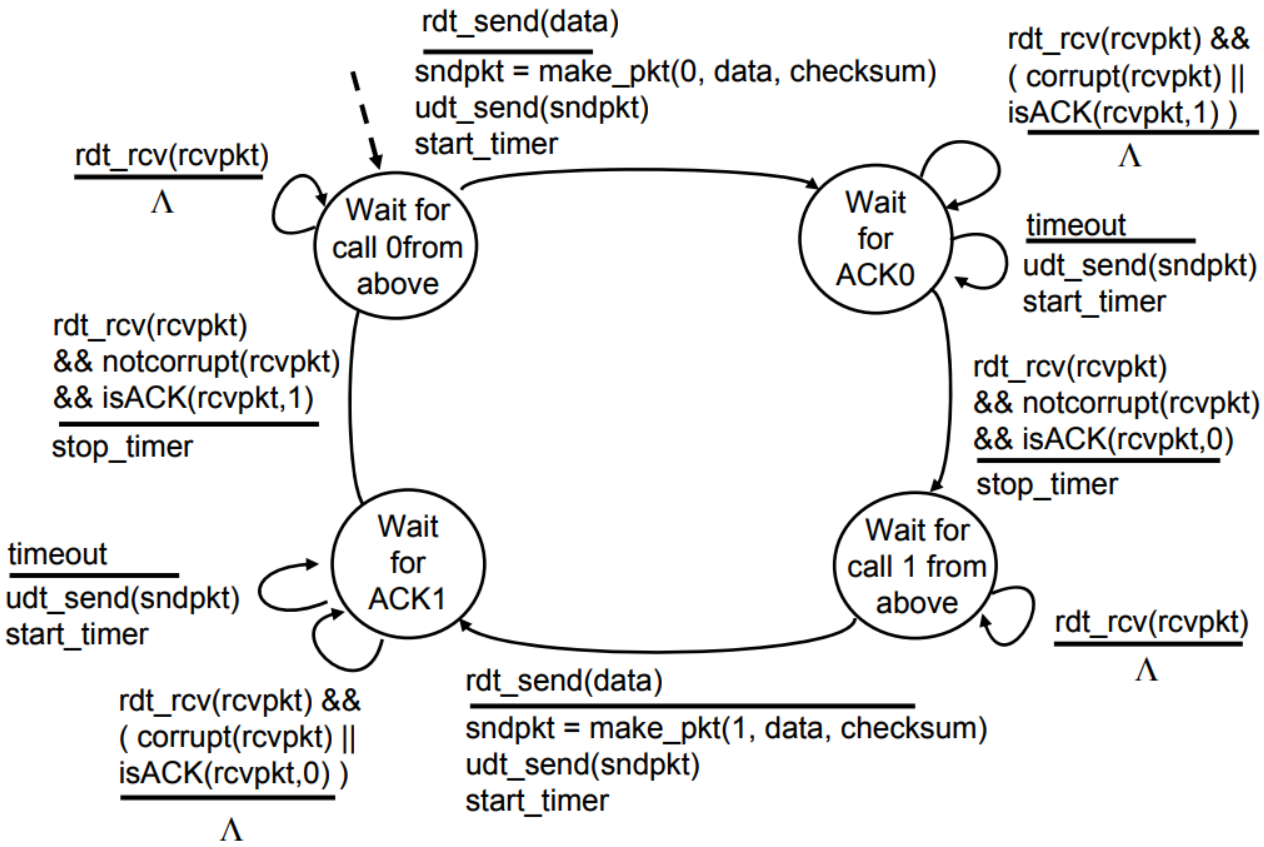

Figure 1. The state diagram of the rdt3.0 protocol

The first byte of the message contains a number which indicates the type of the message. There are three types of messages: DATA, ACK, END as shown in Table 1.

Table 1: Message types and their representative values.

Message Type	Value
DATA	0
ACK	1
END	2

The second byte of a message indicates the sequence number of the sent message. For rdt3.0, this number is either 0 or 1. Remember that both the sender and the receiver parts of your program needs to keep track of this number.

For DATA messages, the message contains the payload after the header of 2 bytes. In this assignment, you will use a payload size of 64 bytes, i.e., total size of data messages will be 66 bytes. For ACK and END messages, message size should be 2 bytes.

Client Program

Your client will contain both sender and receiver parts, as does the server.

The server program will be provided to you as a class file and will work according to the following protocol. When a client sends data which ends with the END message, the server will perform some operation on the received data and return some data to the client. For your own tests, this operation will be to **uppercase** the received data**.** However, we will be testing your client also with other operations; so **do not assume** that the sent and received data lengths will be the same in the graded assignment.

The server program that is provided with the assignment runs with the following command in the console:

java RDTServer <source_port_number> <target_port_number>

where *<source_port_number>* is the port number to which the server socket is bound and <*target_port_number>* is the port number to which the client program is bound. After executing this command, the server program runs and starts to listen for incoming UDP segments from the specified port.

The client program that you are asked to implement should run with the following command:

java RDTClient <source_port_number> <target_port_number> <target_file>

where <*port_number>* is the port number to which the server socket is bound, *<target_port_number>* is port number to which the client program is bound, and *<target_file>* is the name of the text file which the client wants to send. Note that since you will be running both the server and the client on the same computer (for the sake of simplicity), you can assume that server and client share the same IP address.

After the server starts running, you will execute the client program with the command described above.

When you start your client, your client should do the following:

1. Send the text file with name *<target_file>* to the server through UDP reliably. Some messages associated with the reliable application layer protocol will be dropped randomly by the server. Your rdt3.0 implementation should handle this.
2. After sending the file and the END message, your client should switch to the listening mode. In this mode, the server will start sending the client an altered version of the text file.

3. Receive the altered text file and save it with the name "*<target_file>_altered.txt*" e.g. if *readme.txt* is the "*<target_file>*, then your client should save in its own root folder a file named *readme_altered.txt*
4. Finally exit and print the total time and total number of message transmissions in the following format:

   >> Transmission complete: Total number of messages: 500, Total number of messages sent: 512

   The above implies that 12 messages were lost.

   Points to Consider

   - RDT timeout is set 1000ms as default in the server.
   - Your program also needs to respond to the END message correctly (and terminate correctly)
   - TCP is prohibited (as server would not respond TCP packets)

Submission Rules

You can do this assignment either individually or together with another student from the **same** course section.

You need to apply all of 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 as an e-mail attachment sent to yarkin.cetin[at]bilkent.edu.tr. Any other methods (Disk/CD/DVD) of submission will not be accepted.
- The subject of the e-mail should start with [CS421_PA2], and include your name and student ID. For example, the subject line should be

  [CS421_PA2]AliVelioglu20111222

  if your name and ID are Ali Velioglu and 20111222. If you are submitting an assignment done by two students, the subject line should include the names and IDs of both group members. The subject of the e-mail should be

  [CS421_PA2]AliVelioglu20111222AyseFatmaoglu20255666

  if group members are Ali Velioglu and Ayse Fatmaoglu with IDs 20111222 and 20255666, respectively.

- All the files must be submitted in a zip file whose name is the same as the subject line **except** the **[CS421_PA2]** part. The file must be a .zip file, not a .rar file or any other compressed file.
- All of the files must be in the **root** of the zip file; directory structures are not allowed. Please note that this also disallows organizing your code into Java packages. The archive should **not** contain:

  – Any class files or other executables,

  – Any third-party library archives (i.e., jar files),

  – Any text 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. For a discussion of academic integrity, please read this document. Refer to 'YÖK Student Disciplinary Rules and Regulations', items 7.e and 7.f, and 'Bilkent University Undergraduate Education Regulations' item 4.9 for possible consequences of plagiarism and cheating.