CO633 Computer Networks 2019-20 : Programming Assignments #1 Sender and #2 Receiver

Advice on tackling the implementation

This appendix contains advice for the benefit of those whose programming skills are a little rusty. The test system used when marking will give credit in proportion to the completeness of your implementation. However, as explained earlier, this is based entirely on how well your code functions when it runs. Code that implements a subset of the required functionality correctly is likely to score a higher mark than a more complete implementation containing a bug that undermines its basic functionality. Hence, while building towards a complete implementation, you should always verify the current subset of functionality is working correctly before extending it.

The approach we suggest for tackling the programming coursework is outlined below. These are not complete instructions. Additional functionality, such as error detection, would be needed to gain full marks. Other approaches are possible too.

#1 Sender

1. This part involves completing method sendMessage in class MessageSender.

2. Compile the project and experiment with TestSender to become familiar with its operation. See the TestSender class documentation and testing examples for instructions. The supplied version of sendMessage just passes a fixed string "sendMessage not yet implemented!" directly to sendFrame without framing or segmentation.

3. As a simple first step, replace the fixed string with the message string. Don't worry about splitting or framing the message yet.

4. A next logical step is to implement a partial solution for the simple case of a short message that fits in a single frame. Set the segment length and checksum fields to "`00`" for now. For example, if the message is **`hello`** the result of this stage will be **`<E-00-hello-00>`**. Hint: only one line of Java needs to be edited for this step!

5. Implement the segment length field. The above test message should yield **`<E-05-hello-00>`**. The number of characters in a String can be obtained using the length method.

6. Implement the checksum field next. The test message above should yield **`<E-05-hello-37>`**. Character values can be obtained using the charAt method. Java treats individual char values as numbers so they needn't be converted to ints before use in arithmetic calculations.

7. Implement segmentation so that long messages are split into several frames according to the MTU. Don't forget the MTU applies to the whole frame including all delimiters, not just the message segment. The frame type field will need to be set to **`D`** or **`E`** accordingly.

#2 Receiver

1. This part involves completing method receiveMessage in class MessageReceiver.

2. Experiment with TestReceiver to become familiar with its operation. See the TestReceiver class documentation and testing examples for instructions. The supplied version of receiveMessage just reads and returns each frame directly to TestReceiver without any processing.

3. Implement a solution to the simple case described above in Sender step (4). The start and end of the message segment are normally at fixed positions relative to the start and end of the frame (i.e. you need

to chop off a fixed number of characters at either end). Hint: Java's String class has a handy method for extracting a portion of a string (a substring).

4. Ditto for Sender step (5). Verify the recorded segment length is correct.
5. Ditto for Sender step (6). Verify the recorded checksum is correct.
6. Ditto for Sender step (7). Loop until the final frame is received and reassemble the message segments as a single string.