1 Overview

This document specifies a framing protocol for CS 685 Network Algorithmics. This is a simple byte-oriented, delimiter-based protocol that can accommodate frames of arbitrary length. A byte-stuffing technique makes it possible for the receiver to distinguish payload bytes that happen to have the same value as the control (delimiter and escape) bytes from those that are actually functioning as delimiters.

2 Constants

This protocol uses four constant values: the control bytes used in the protocol are listed in the following table (the names are taken from the ASCII code):

<table>
<thead>
<tr>
<th>Name</th>
<th>Value (decimal)</th>
<th>Value (hexadecimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STX</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SYN</td>
<td>22</td>
<td>16</td>
</tr>
<tr>
<td>ETX</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>DLE</td>
<td>16</td>
<td>10</td>
</tr>
</tbody>
</table>

3 Frame Layout

The layout of the frame is shown in Figure 1. Each frame begins with the three-byte sequence SYN SYN STX (decimal byte values 22, 22, 2). The end of the frame is marked by the single delimiter ETX (decimal value 3). The sending protocol implementation inserts the byte DLE before each occurrence of SYN, STX, ETX, and DLE in the payload; the receiving protocol implementation removes any such “stuffed” bytes before passing the payload to the higher layer. Thus, the framing mechanism is completely transparent to the higher layer(s).

As an example, consider a payload consisting of four bytes with the values 0, 1, 2, 3 in that order. The result of framing that payload would be a sequence of 10 bytes with the values:

22, 22, 2, 0, 1, 16, 2, 16, 3, 3

Note that a payload of length \( n \) can become of a frame of length \( 2n + 4 \) when it is transmitted “on the wire”. 
It may seem that stuffing SYN and STX in the payload is unnecessary, since the receiver is only looking for ETX while processing the payload. Note, however, that framing protocols should be designed under the assumption that a receiver might begin receiving at any point in the input stream of bytes, in particular in the middle of a payload. The specified stuffing algorithm ensures that the start delimiter sequence occurs only at the beginning of a frame, and thus allows the receiver to unambiguously identify the first complete frame it receives (assuming no errors occur in transmission).

4 Sender Processing

Given a payload, the sender prepends the 3-byte header. Then it processes each byte of the payload in order, inserting the single byte DLE in front of each occurrence of DLE, SYN, STX, or ETX in the payload. Finally, the sender appends the single byte ETX to the stuffed payload.

Generally the sender should transmit frames back-to-back wherever possible. When there is no data to transmit, the sender may transmit SYNs continuously.

5 Receiver Processing

The receiver discards all bytes received until it recognizes the start delimiter sequence SYN SYN STX. It then begins assembling the payload a byte at a time, as follows: If the input byte is ETX, it is discarded, the current payload is delivered to the higher layer, and the process starts over. Otherwise, if the input byte is DLE, it is discarded and the next byte is appended to the current payload without examining it. If the input byte is not ETX or DLE, it is appended to the current payload.

Note that the byte following DLE in the payload is not examined; it could thus be something other than SYN, STX, ETX, or DLE. Given a properly-functioning sender, this situation can only occur if a frame is corrupted in transit. Such errors must be detected by an error-detection protocol; in any case delivering the subsequent byte seems harmless.