IND426 Tutorial 9 (Solutions)
(Roberto Togneri 1998)

1. Consider what happens when an obsolete SYN \( i \) is received by B from A. Assume B is listening for connection requests from A (asymmetric operation).
   (a) Show how a 3-way handshake will correctly handle this.
   (b) Assume that immediately after the obsolete SYN \( i \) is received the correct SYN \( x \) is also received. Show how a 3-way handshake can be used to reject the obsolete SYN \( i \) and still allow the correct SYN \( x \) to form a connection. Assume duplicate SYN requests are ignored.
   (c) In Figure 17.10 (dec5e) the behaviour of case (b) for a 2-way handshake is shown. But show what happens if only the obsolete SYN \( i \) is received (as is the case in (a)).
   (d) Show how the 2-way handshake can result in a deadlock situation.

Solution
(a)
- B responds to the SYN \( i \) with a SYN \( y \), ACK \( i \)
- A realises something is amiss (since it doesn’t recognise the ACK \( i \)) and sends RST \( y \)
- B receives RST \( y \) and aborts connection.

(b)
- B responds to the SYN \( i \) with SYN \( y \), ACK \( i \) to obsolete SYN \( i \)
- B ignores the correct but duplicate SYN \( x \)
- A will send RST \( y \) in response to SYN \( y \), ACK \( i \) (A doesn’t recognise ACK \( i \))
- B will abort the connection associated with SYN \( i \)
- A will timeout and resend SYN \( x \) (since it did not receive an ACK \( x \) from B)
- B responds with SYN \( y \), ACK \( x \)
- A responds with ACK \( y \) and the connection is established.

(c)
- B responds with SYN \( y \) and opens connection to A
- A responds with RST \( y \) since it did not initiate any connection request
- B aborts connection when it sees the RST \( y \)

(d)
- B receives an obsolete SYN \( i \) request, responds with SYN \( y \) and opens connection
- The SYN \( y \) is lost and the connection remains open indefinitely.

Solution
(a) No.

(b) The sender cannot allow the window sequence number to wrap-around in any one 60 second interval (30 seconds for data packet lifetime plus an additional 30 seconds for the corresponding ACK packet). With an 8-bit sequence number up to 256 packets can be sent in a 60 second interval. Thus the maximum data rate is \((256 \times 128 \times 8) / 60 = 4369\) bps.

(c) There is no direct provision in TCP for lost credit allocations other than the fact that such allocations are repeated (together with the current acknowledgment) in subsequent outgoing data segments. There is also no provision for out of order credit allocations other than relying on the out-bound data SEQ and ACKN values. However, this will not work if there is a change in credit allocation but no more out-bound data is being generated and no in-bound data is arriving since the same SEQ and ACKN will be used (NOTE: an empty segment will use the SEQ value of the most recent out-bound data octet, hence a sequence of empty segments will have the same SEQ, and with no in-bound data the ACKN will also be the same ACKN of the most recent in-bound data octet received).

[Note: With TCP a SYN \( x \) will require an ACK \( x +1 \) rather than ACK \( x \)]
3. Consider the following observed round-trip times for the first 20 segments on a connection: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10. Calculate what the TCP RTO(i) value is for \( i = 1, 2, \ldots, 15 \). Assume the typical values are used for the timer management parameters.

**Solution**

We set SRTT(0) = SDEV(0) = 0 (since RTT is initially assumed to be 0), then:

\[
\begin{align*}
\text{SRTT}(i) & = 0.875 \times \text{SRTT}(i-1) + 0.125 \times \text{RTT}(i) \\
\text{SERR}(i) & = \text{RTT}(i) - \text{SRTT}(i-1) \\
\text{SDEV}(i) & = 0.75 \times \text{SDEV}(i-1) + 0.25 \times |\text{SERR}(i)| \\
\text{RTO}(i) & = \text{SRTT}(i) + 4 \times \text{SDEV}(i)
\end{align*}
\]

And if you are dab hand at Excel you will instantly obtain:

<table>
<thead>
<tr>
<th>( i )</th>
<th>RTT</th>
<th>SRTT</th>
<th>SERR</th>
<th>SDEV</th>
<th>RTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.1250</td>
<td>1.0000</td>
<td>0.2500</td>
<td>1.1250</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.3594</td>
<td>1.8750</td>
<td>0.6563</td>
<td>2.9844</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.6895</td>
<td>2.6406</td>
<td>1.1523</td>
<td>5.2988</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1.1033</td>
<td>3.3105</td>
<td>1.6919</td>
<td>7.8708</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1.5904</td>
<td>3.8967</td>
<td>2.2431</td>
<td>10.5628</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>2.1416</td>
<td>4.4096</td>
<td>2.7847</td>
<td>13.2805</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>2.7489</td>
<td>4.8584</td>
<td>3.3032</td>
<td>15.9615</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>3.4053</td>
<td>5.2511</td>
<td>3.7902</td>
<td>18.5659</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>4.1046</td>
<td>5.5947</td>
<td>4.2413</td>
<td>21.0698</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>4.8415</td>
<td>5.8954</td>
<td>4.6548</td>
<td>23.4608</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>5.4863</td>
<td>5.1585</td>
<td>4.7807</td>
<td>24.6093</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>6.0505</td>
<td>4.5137</td>
<td>4.7140</td>
<td>24.9064</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>6.5442</td>
<td>3.9495</td>
<td>4.5228</td>
<td>24.6356</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>6.9762</td>
<td>3.4558</td>
<td>4.2561</td>
<td>24.0005</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>7.3542</td>
<td>3.0238</td>
<td>3.9480</td>
<td>23.1462</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>7.6849</td>
<td>2.6458</td>
<td>3.6225</td>
<td>22.1747</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>7.9743</td>
<td>2.3151</td>
<td>3.2956</td>
<td>21.1568</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>8.2275</td>
<td>2.0257</td>
<td>2.9781</td>
<td>20.1401</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>8.4491</td>
<td>1.7725</td>
<td>2.6767</td>
<td>19.1560</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>8.6429</td>
<td>1.5509</td>
<td>2.3953</td>
<td>18.2241</td>
</tr>
</tbody>
</table>

4. Consider a long file transfer where the sender transport entity transfers data in 1000 octet segments but the receiver’s transport user reads data in 50 octet blocks. Assume a window update policy of immediate advertisement of the current available credit and a send policy of sending as soon as credit is available. The resultant poor performance of the file transfer because of these policies is known as the Silly Window Syndrome (SWS). Explain the poor performance that occurs under the above conditions and how the standard TCP policies avoid this condition.

**Solution**

Consider a full receiver buffer (credit = 0). The transport user reads the next 50-octet block and the available credit is now 50 octets which the receiver transport entity promptly uses to notify the sender. In response to the 50-octet credit advertisement the sender immediately obliges with a 50-octet data segment. The receiver’s buffer is now full again and a 0 credit is advertised (which stops the sender). When the transport user reads the next 50-octet the process repeats. The poor performance comes from the sender using small 50-octet data segments and only being able to send one of these at a time since the receiver’s buffer is almost always full.

The TCP window update policy is to wait for WIN to be large enough and the send policy is to wait for a segment to contain enough data. Both policies aid in preventing the occurrence of SWS.

5. (a) Give a potential disadvantage of Nagle’s algorithm for X-windows traffic.
(b) Consider the effect of using slow-start on a line with a 10-msec round-trip time and no congestion. The receive window is 24 KB and the maximum segment size is 2 KB. How long does it take before the first full window can be sent?
(c) Suppose that the TCP congestion window is set to 18 KB and a timeout occurs. How big will the window be if the next ten transmission bursts are all successful? Assume that the maximum segment size is 1 KB.

**Solution**

(a) Each mouse action will be aggregated and delivered to the X-windows client in one segment. This will cause the X-windows mouse motion event responses to be erratic or bursty rather than simply delayed.

(b) The congestion window is initially set to 1 MSS = 2 KB and the doubles with each successive RTT: 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, etc. After 4 RTT’s = 40 msec the congestion window will be set to 32 KB and hence the full receive window of 24 KB will be used.

(c) If the congestion window is 18 KB when the timeout occurs than a threshold, TH = 9 KB is chosen and slow-start proceeds as follows: First 4 successful transmissions: 1 2 4 8 (exponential) and then for the next 6 successful transmissions: 9 10 11 12 13 14 (linear). Thus the window will be 14 KB at the end of ten successful transmissions.