2. [Congestion Control (1).]  

(a) The first ACK number is 10 (the host on the right has received bytes 0–9 and is expecting the next byte to be byte number 10).

(b) The third ACK number is 30 (the host on the right has received bytes 0–9, 10–19, and 20–29 and is expecting the next byte to be 30.)

(c) After receiving (cumulative) acknowledgement for the first 30 bytes, the new value of cwnd will be 60 bytes (each successful ACK increased it by 10); see Figure 1.

(d) The next sequence number will be 30.

(e) The value of ssthresh does not change—it remains at 64K.

(f) We are still in slow start since cwnd < ssthresh.

3. [Congestion Control (2).]  

(a) After any timeout, the value of cwnd is set back to 1 MSS, i.e., to 10.

(b) Since segment 0 is the one that timed out, the next segment sent will begin with byte 0.

(c) Yes, we set ssthresh equal to cwnd/2 = 30/2 = 15 bytes. See Figure 2.

(d) We are still in slow start since cwnd < ssthresh.

4. [Congestion Control (3).] This was a bit tricky; I gave credit for multiple answers.

(a) A triple duplicate ACK has just occurred, causing ssthresh to drop to cwnd/2 = 50/2 = 25. See Figure 3.
Figure 2: See problem 3.

Figure 3: See problem 4.
(b) After the fourth ACK was received, the value of the variable dupACKcount reached three, causing a transition to the “Fast recovery” state. The value of cwnd was set to ssthresh + 3 MSS = 25 + 3(10) = 55. I will give credit if you said “65”, mistakenly thinking that there was one additional duplicate, but 55 is correct. Furthermore, a typo on the slide says “cwnd = ssthresh + 3”, but it really should be “ssthresh + 3 MSS”, so some people may have gotten a value of 28 rather than 55. I gave credit for this, too.

(c) Since the duplicate ACKs are all requesting byte 10, that is the sequence number of the next segment.

(d) We are in the “Fast recovery” state.

5. [Congestion Control (4).]

(a) NOTE: several people asked me about which time values to use at the transition points (e.g., at time 4 are we in slow start or congestion avoidance?). It doesn’t matter—I will give credit as long as it’s clear that you know where, in the graph, things are occurring.

We are in the “slow start” state during time intervals 1–4, 9–12, 14–17, 32–35, and 39–40. We are in “congestion avoidance” during intervals 4–8, 12–13, 17–19, 20–28, 29–31, and 35–38. We are in “fast recovery” at times 20 and 29. (Note: I am assuming that we don’t continue receiving duplicate ACKs.)

(b) Answer combined with (c).

<table>
<thead>
<tr>
<th>Time</th>
<th>Cause</th>
<th>New ssthresh</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Timeout</td>
<td>6 MSS</td>
</tr>
<tr>
<td>13</td>
<td>Timeout</td>
<td>4.5 MSS</td>
</tr>
<tr>
<td>19</td>
<td>Triple ACK</td>
<td>5 MSS</td>
</tr>
<tr>
<td>28</td>
<td>Triple ACK</td>
<td>8 MSS</td>
</tr>
<tr>
<td>31</td>
<td>Timeout</td>
<td>6.5 MSS</td>
</tr>
<tr>
<td>38</td>
<td>Timeout</td>
<td>5.5 MSS</td>
</tr>
</tbody>
</table>

(c) Time | Cause | New ssthresh |
---------|-------|--------------|
8        | Timeout | 6 MSS        |
13       | Timeout | 4.5 MSS      |
19       | Triple ACK | 5 MSS       |
28       | Triple ACK | 8 MSS       |
31       | Timeout  | 6.5 MSS      |
38       | Timeout  | 5.5 MSS      |

6. [Reliable Data Transfer.]

(a) My original answer was “No”, but I later realized that something like the following could happen:

Packet 1 is sent and delayed a very, very long time. Meanwhile, sender times out and resends another copy of packet 1, which arrives and is ACK-ed. Packet 2 is sent and arrives. Then the long-delayed earlier packet 1 finally arrives.

So I will give credit for either answer to this question!

(b) Packet 1 arrives, but the ACK for it is lost. Meanwhile, receiver delivers the packet data to the application. Sender times out and re-sends packet 1. Receiver passes it to the application.

(c) • NAKs would not really help—they are useful only when the receiver “knows what to expect”. Since there are no sequence numbers, it doesn’t know which packets to expect, and since no corrupt packets are allowed, there is nothing to “negatively acknowledge.”

• Sequence numbers would definitely help. Now the receiver can keep track of which packets have been received and avoid delivering duplicate packets to the application. Much has to change in the state diagram!
• The main problem here is: when should the receiver start timing something? It has no idea when to expect new data to arrive, so there is nothing to time!