Network Systems (201600146/201600197), Test 1

February 16, 2018, 13:45–15:15

Answers

---

1. SpaceY's rocket link

1 pt (a) C.

Many chose D; however, a radio channel is very suitable for statistical multiplexing, it's what e.g. WiFi does.

3 pt (b) Use the formula: $H = 0.8 \log_2 \frac{1}{0.8} + 0.09 \log_2 \frac{1}{0.09} + 0.09 \log_2 \frac{1}{0.09} + 0.02 \log_2 \frac{1}{0.02} = 0.9957$ bits per word.

1 pt (c) B.

The more you already know about a message, the less information it contains. In this case, the previous messages tell you something about the future ones: if there have been many "increase power" messages, then you can be sure the next one will not be "increase power" again.

3 pt (d) Correct answers are, respectively: G, G, E, F, G.

MC03: this code is impossible: twice 'no change' cannot be distinguished from 'decrease power'.
MC04: the three codewords given are fine, but there are no possibilities left that are distinguishable. If you choose e.g. 01, then 01010101... could be either 'self-destruct' repeatedly, or 'no change' followed by 'increase power' repeatedly.
MC05: 000 will work fine.
MC06: either 100 or 111 would be fine.
MC07: choosing 11 would make a correct code, but doesn't satisfy the stated requirement that the average message length must be less than 1.5 bits.

2 pt (e) Use the formula:
$C = 1100 \cdot (1 - 0.99 \log_2 \frac{1}{0.99} - 0.01 \log_2 \frac{1}{0.01}) = 1011.1$ bits per second.

1 pt (f) A.

From (b) we know that the information per message is 0.9957 bits, and we have 1000 messages per second, making for 995.7 bits/s of information. This is less than the channel capacity calculated in (e), so the error probability can be made arbitrarily small.

1 pt (g) C.

Note that D is not correct: you can have codes that indeed need more channel bits per user data bit, but those don't get closer to the channel capacity, as you would be sending *fewer* user bits per second.

---

2. Application protocols

1 pt (a) D.

An IMAP or POP server is needed so users can fetch their mails (to their own computers which are not always on). And an SMTP server is needed so computers from outside this company can deliver their mails for users in the company, and also for the users in the company to send their outgoing mails to (although theoretically they could also connect directly to the SMTP server of the destination, but this is not usually done).

1 pt (b) E.

See above about what the SMTP server is used for. Neither of those two ways of using it should be blocked. However, SpaceY's mail server can (and should) reject mails which have no business on SpaceY's network; otherwise it is an "open relay" which can be abused to spam the world.

2 pt (c) Here, many different answers are possible. Some examples:
- security/privacy: your mail may be stored on someone else's computer.
- how can mail servers outside SpaceY's network find which computer to deliver SpaceY's mail to when that's not a fixed server.

3. Protocols and performance

2 pt (a) Propagation time = $\frac{\text{distance}}{\text{speed}} = \frac{390\,000}{300\,000}$ = 1.3 s

2 pt (b) Transmission time = $\frac{\text{number of bits}}{\text{bit rate}} = \frac{8000}{1000\,000}$ = 0.008 s = 8 ms
Or, properly taking kilobyte as $2^{10} = 1024$ bytes:
Transmission time = $\frac{\text{number of bits}}{\text{bit rate}} = \frac{8192}{1000\,000}$ = 0.008192 s = 8.192 ms

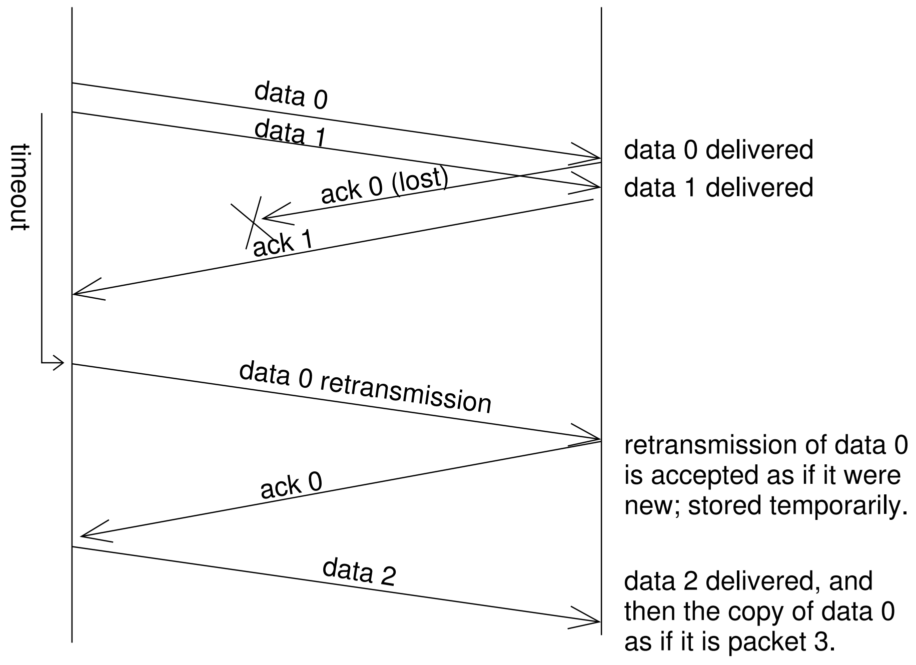
2 pt (c) We need to fill the entire RTT, which is 2.6 s, with packets of 8 ms each, so SWS$\geq$2.6/0.008 = 325. Strictly speaking, we need one more, namely 326, as follows: 8 ms after the start of a data packet, its last bit has left the sender. 1.3 s later it arrives at the receiver, which sends an ack, which takes 1.3 s to reach the sender. So only 8 ms + 2 $\times$ 1.3 s = 2.608 s after the start of a packet can we move the window to no longer include that packet. During that time, 2.608/0.008 = 326 packets can be sent, so in order to transmit without interruptions, the minimum sufficient SWS is 326.
If you nicely used 2^{10} bytes for a kilobyte in the previous question, then of course you get a slightly different answer here: 319.

1 pt (d) A.

Efficiency is reduced because the sender can't send continuously anymore, has to wait for an acknowledgement. Reducing the window size does not make the protocol incorrect: it does not increase the possibilities for confusion between old and new packets with the same sequence number.

2 pt (e) Because RWS=SWS, we need to allow for the window to be twice as large. Thus, we need to distinguish between 2$\times$326 = 652 packets; that takes 10 bits, as $2^9 < 652 < 2^{10}$.

3 pt (f) Many variations are possible, but this is about the simplest:


timeout
data 0
data 1
ack 0 (lost)
ack 1
data 0 delivered
data 1 delivered
data 0 retransmission
ack 0
retransmission of data 0 is accepted as if it were new; stored temporarily.
data 2
data 2 delivered, and then the copy of data 0 as if it is packet 3.

Some noteworthy points:

- The sender's window size is 2; so initially, its window contains packets 0 and 1, and it is not allowed to transmit packet 2. Only after receiving the ack for 0 does its window move to be packets 1 and 2, and then packet 2 can be sent.
- Similarly, the receiver's window initially also consists of 0 and 1; after receiving packet 0, it becomes 1 and 2; then after receiving 1, the window contains 2 and 0. Then it will be willing to receive a packet with number 0 and interpret it as if it is the 4th packet.
- The receiver is not simply going to deliver the packets to the application layer in the order they come in: of course it will order them by their sequence numbers. So when its window contains 2 and 0, and it receives 0, it will store this, but wait for the missing intermediate packet (nr. 2) before delivering them to the application.

1 pt (g) C.

Persistent connections saves one RTT for setting up the connection for each image except for the first. And pipelining allows the ground station to send multiple GET requests without waiting for the response to the previous one, again saving an RTT per image.

1 pt (h) A.

In either case, first one RTT is "wasted" on setting up the connection(s), but after that multiple requests can be sent (either pipelined, or over the parallel connections) so the spacecraft can send multiple images continuously.

1 pt (i) C.

See the derivation of the formula.

2 pt (j) D.

First calculate the lower bound with these numbers: 100 seconds.
Then check option D:

- Server sends F/2=10 Mbit to each peer at 0.1 Mbit/s: this doesn't overload u_S, and takes 100 seconds so finishes in time, so it's feasible.
- Each peer receives 0.1 Mbit/s and immediately sends this to one other peer: this doesn't overload either d or u, so it's feasible.
- Each peer receives 0.1 Mbit/s directly from server, and 0.1 Mbit/s from one other peer, so indeed, in 100 seconds the peer receives F = 20 million bits. Do these bits together form the complete file? If we arrange it such that e.g. the server sends the first half of the file to peers 1—5 and the second half to peers 6—10, and peers 1—5 send their data on to peers 6—10 and vice versa, then indeed each peer has the complete file after 100 seconds.

The other options fail, due to either the server or a peer not having enough upload bandwidth.