Network Systems (201600146/201600197), Test 3

March 23, 2018, 13:45–15:15

Answers (updated 2018-03-29)

---

1. Addressing and routing

3 pt (a) B, A, B, B, A, C, B, C, B.
The most difficult one was apparentely `100:0:face::1:0` (MC06); many chose B, but in fact this address does belong to the subnet. The subnet has a /33 mask, meaning the first 33 bits identify the network: 16 bits in the first group, 16 bits in the second group, and 1 bit in the third group; both hexadecimal `8000` and `face` start with a 1 bit, so the 33rd bit does match.

1 pt (b) D.

1 pt (c) D.

1 pt (d) A.

1 pt (e) C.
Many chose answer A, but there really is no shortage of IPv4 *multicast* addresses.

---

2. Distance-vector routing

1 pt (a) {(A,0), (B,15), (C,3)}

1 pt (b) {(A,0), (B,4), (C,3), (D,17)}

1 pt (c) {(A,0), (B,4), (C,3), (D,6)}

1 pt (d) {(A,0)}

2 pt (e) B and C will do bouncing/count-to-infinity, in each iteration increasing their estimate of the cost to reach A by 1, until it exceeds the cost of going to A via the B–A link. C started with a cost to A of 3, so this takes 13 iterations (or 12 or 14, depending a bit on where you start counting).

1 pt (f) D.
With split horizon, B would not have told C that it could reach A at cost 4, so C would not mistakenly assume that it could reach A via B at cost 5, and so on.

1 pt (g) A.
When a new link is *added*, the convergence cannot be slow. The bouncing problem only occurs if costs go up, as nodes prefer to use low-cost paths but those are outdated, and these need to 'bounce' back and forth until their cost becomes so high they will no longer be prefered. If costs go down (as could happen due to adding the new link), the preference for low costs is actually also a preference for the fresh, new information.

1 pt (h) C.

1 pt (i) B.
The RREQ packets are in principle flooded over the entire network whenever one nodes needs to find a route to another node. So with more nodes, there will be more such packets, *and* they will need to be handled by more nodes.

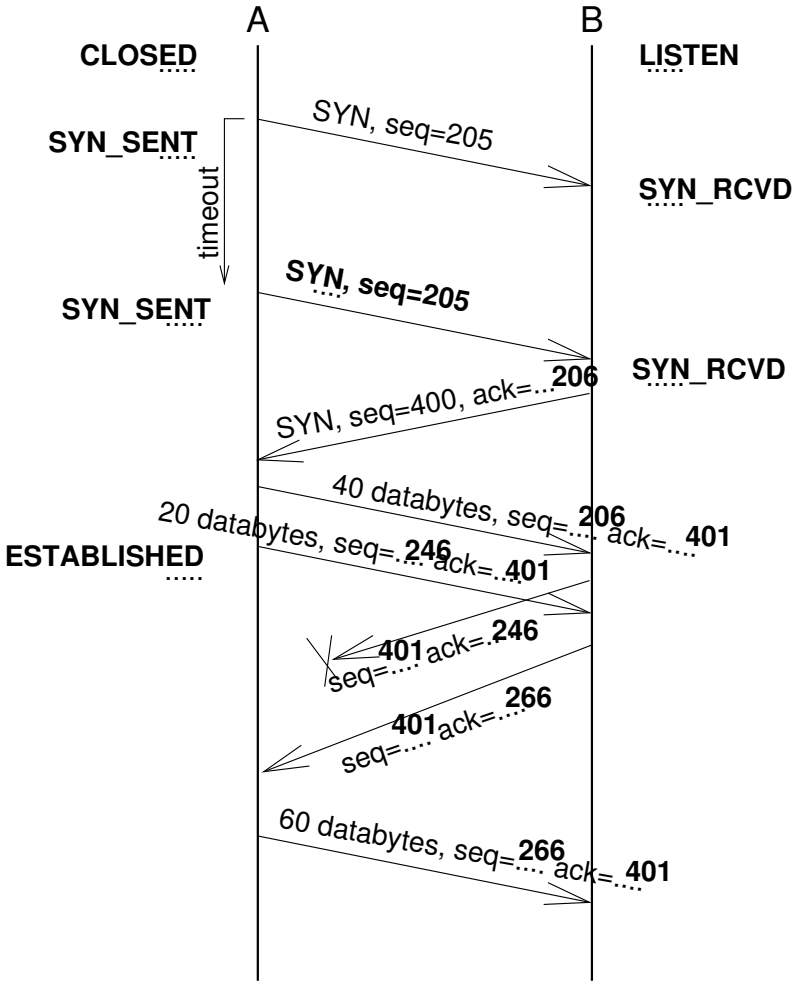
---

3. Transport layer protocols

1 pt (a) D.
For timing measurements, you really don't want the transport layer to do a retransmission, because it gives extra uncertainty: when you do get a response from the other side, containing its idea of the current time, was

it the time at the time of the original request or at the time of the retransmission? Better let the application deal with packet loss in a way it can distinguish between an answer to the first and to the second attempt, by including some kind of identifying information in the request and response.

1 pt (b) B.



5 pt (c)

2 points for the states, 1 point for the SYN packets, and 2 points for the seq/ack numbers in the non-SYN packets.

The initial state of A can also be LISTEN according to the state-transition diagram, but this is rather rare: it would be the situation where the application first decides to listen for an incoming connection, and then changes its mind and decides to open the connection itself.

There's something strange with the first SYN packet: B does receive it, but does not respond. That's a mistake I made: I didn't draw B's response, which must have gotten lost on its way to A. However, you could also interpret this as B not receiving the first SYN packet, and in that case it would stay in the LISTEN state, so we accepted that as well.

Note that B does eventually go to the ESTABLISHED state, namely after it receives the '40 databytes' packet, which also contains the ACK for B's SYN-ACK. Since there were no dots in the diagram there, you didn't need to write it down.

Lots of mistakes were made with the seq/ack numbers in the latter part of the diagram.
Be careful to distinguish between the two directions in which data can flow: in this case, data from A to B has sequence numbers in the 200s, and thus the packets from B to A have ack numbers in that range; and conversely, B's sequence numbers are around 400, so A acks numbers in this range.
Furthermore, be careful about how to increment the numbers. Sequence numbers count bytes, and the SYN and FIN flags each also take up one place in the sequence number space. If packets do not contain user data, such as (in this case) those sent by B, the sequence number does not increase (except for the SYN/FIN flag).

1 pt (d) C.

Since TCP's acknowledgements are cumulative, if TCP has already sent ACK=500, it must have already received the data with SEQ=300 earlier. The fact that another copy of this packet shows up, could e.g. be due to an unnecessary retransmission, if the other side's retransmission timeout is too short. The only useful response is transmitting the ACK=500 again; since the ACK number is cumulative, this also acknowledges the arriving SEQ=300 packet.

1 pt (e) C.
The meaning of the labels at the arrow is explained in the book on page 404–405. Note further that there is no such thing as a *Close* packet (there's no 'Close' flag in the TCP header); *Close* is an action commanded by the application program.

1 pt (f) B or E.
The best is answer E: if we are in `FIN_WAIT_2`, then the other side must have already ack'ed our FIN, so it must have progressed to `CLOSE_WAIT`.
However, this is not completely correct: it is possible that the other side has already sent a FIN too, after ack'ing ours, and thus has progressed to `LAST_ACK`, but that that FIN has not yet reached us (otherwise we'd have gone to `TIME_WAIT`).

When I wrote the problem, the intended answer was B. I intended this to be an example of a TCP connection being open in one direction and already closed in the other direction, as discussed at the top of page 407 in the book. However, as noted above, in that situation the other side should be in `FIN_WAIT_2`, not in `ESTABLISHED`. This was an error in the problem statement.

We've accepted both answer B and E.

1 pt (g) A.
The Window Scaling extension makes it possible for the receiver to communicate a larger window size to the sender than would otherwise fit in the 16 bits available for this in the TCP header.

1 pt (h) C.
The time after which sequence numbers wrap around, only depends on the bandwidth, not on the RTT. But if the RTT is small, the window can be small while still fully utilizing the bandwidth. So on a link with large bandwidth but small RTT, you do need protection against wrapped sequence numbers, but there's no need to scale the window.