Network Systems (201300179/201400431), Test 1

February 12, 2016, 13:45–15:15

With answers

- This is an open-book exam: you are allowed to use the book by Peterson & Davie and the reader that belongs to this module, and the handout about peer-to-peer communication (i.e., the part of the Kurose&Ross book distributed via Blackboard). Furthermore, use of a dictionary is allowed. Use of a simple (non-graphical) calculator is allowed.
- Other written materials, and laptops, tablets, graphical calculators, mobile phones, etc., are not allowed. ***Please remove any such material and equipment from your desk, now!***
- Although the questions are stated in English, you may answer in English or Dutch, whichever you are more comfortable with.
- You should always explain or motivate your answers, with so much detail that the grader can judge whether you understand the material; so just saying "yes" or giving a formula without explanation is not enough.
- Visiting the toilet without explicit permission of the supervisor is not allowed. During the last 30 minutes of the exam, no toilet visits are allowed.

---

1. Protocols and performance

Consider SMTP delivering e-mails over a transatlantic cable, with an RTT of 100 ms and a bandwidth of 100 Mbps. Assume for now that the underlying protocols do not use sliding windows or otherwise limit how many packets can be sent.

The SMTP protocol consists of a number of messages/command (`HELO`, etc.) sent to the server, to each of which the server responds.

3 pt (a) Assuming that the client cannot send the next message before it gets a response from the server to the previous one, how many RTTs does it take to transfer one 10 kB e-mail?

4 (`HELO`, `MAIL FROM`, `RCPT TO`, `DATA`), but one can also argue 5 or 6 by including setup of tcp connection, and the QUIT message.

The transmission time for the messages and the e-mail itself is negligible here (10 kB = 80 kb takes 0.8 ms, that's less than a percent of the RTT).

2 pt (b) Same question, but assuming that the client can send the next message before it gets a response from the server to the previous one.

1 RTT suffices since everything can now be sent in one go, without any intermediate waiting. Or 2, if you include the setup of the tcp connection; or 0.5, if you don't even include the wait for a confirmation of the whole exchange.

2 pt (c) The protocol difference between the previous two questions corresponds to a difference between HTTP/1.0 and HTTP/1.1. Which difference?

Pipelining: in HTTP/1.1 this means the next GET request can be sent before the previous one has been answered; same here with sending the next message/command.

Many students answered "persistent connections", but that's not right here: if persistent connections are used without pipelining, the client still needs to wait for one GET request to be finished before it can send the next.

Now let's move to another protocol layer, which is used by SMTP to enable reliable data transfer over this (unreliable) transatlantic cable

2 pt (d) Would this protocol layer be "above" or "below" the protocol layer in which SMTP belongs? Explain.

Below, because SMTP needs to use it to reliably transfer its messages; in general, the higher-layer *uses* the service provided by the lower layer.

Let's assume this reliable data transfer is implemented using a sliding-window protocol, using 10-bit sequence numbers.

3 pt (e) Draw a time-sequence diagram showing how things can go wrong if SWS=1024 is used.
Of course, you don't need to draw a thousand packets; just draw a few and indicate which range of sequence numbers they have, with sufficient detail that it is clear what you mean.

[I describe the answer in words here, although for full grade you should draw the appropriate diagram.]

Send packets 0 – 1023, which all arrive. Ack for packet 0 is lost, all other acks do arrive. After timeout packet 0 will be retransmitted, which will be accepted by the receiver as a new packet ("packet 1024") rather than as a retransmit of the earlier one with the same sequence number.

It is essential that your diagram shows that between the original transmission of packet 0, and its retransmission, other packets are received correctly; otherwise, the receiver's window will not have moved on yet, and it will not accept the retransmission as a new packet.

Note: it was silently assumed here that packets can't overtake each other (otherwise, any finite sequence number space can be too small, so the question becomes rather meaningless).

3 pt (f) Assuming SWS is safely chosen as 1023, how large should the packets at least be to be able to fully utilize the link bandwidth? Explain.

After the last bit of the first packet leaves the sender, it takes 1 RTT before the ack for that packet gets back to the sender, so in order to keep transmitting continuously, we shouldn't be finished transmitting the other 1022 packets by then. So we can send at most 1022 packets per RTT, i.e., per 0.1 s. That's $0.1/1022$ seconds per packet, and thus $10^8 \cdot 0.1/1022 = 9785$ bits per packet, or 1223 bytes per packet.

Most students computed with 1023 rather 1022 packets, which I've also accepted as a correct answer.

---

2. Information theory and error-correcting codes

The HTTP protocol has a number of response messages; let's assume only the following are used, with the associated probabilities:

`200 OK`	80%
`304 Not modified`	5%
`403 Forbidden`	5%
`404 Not found`	10%

3 pt (a) Down to how few bits (on average per message) can these messages be compressed?

Calculate Shannon information using formula: 1.022 bits.

3 pt (b) Propose a way to encode the observations in less than 1.5 bits on average, and show that your code indeed achieves this.

One solution: from top to bottom in the table: 0, 110, 111, 10; average is 0.8*1 + 0.05*3 + 0.05*3 + 0.1*2 = 1.3 bits.

Be careful to make your code uniquely decodable!

Consider a binary symmetric channel with a raw speed of 100 bits/s and an error probability of 1%.

3 pt (c) Compute this channel's Shannon capacity, and explain what it means.

Formula gives 91.9 bit/s; this means that for any data rate below 91.9 bit/s, and for any desired (low) error rate, there exists an error correcting code allowing us to achieve that error rate at that data rate.

2 pt (d) Would the use of a 4×4 parity matrix code be a good way to get close to this channel's capacity? Explain.

No. A 4×4 parity matrix has 9 bits of data for 16 bits sent over the channel, so it leaves only 56 bits per second for the user data. That's much less than the 91 bits/s that we were promised by Shannon. Also, such a parity matrix can correct only 1 bit error in those 16 bits, which is not enough to get the bit error rate really (i.e., arbitrarily) low.

---

3. Peer-to-peer applications

Consider distribution of a large file among a set of peer nodes.

2 pt (a) In the lectures we derived a formula for how long this would take. This formula was independent of the RTTs. Why is the RTT not important?

Because P2P would typically be used for distributing very large files, for which the transmission time over a link is much larger than the RTTs involved, so the latter's influence is negligible.

Assume we have N peers and no separate server. All peers have the same download speed d bits/s, and the same upload speed u bits/s. However, only half of the peers are willing to upload (the others are impolitely called "leechers")[1]. One peer initially has a file of F bits.

3 pt (b) Give an expression for how long it takes until all peers have the file; explain.

$\max\left(\frac{F}{u}, \frac{F}{d}, \frac{(N-1)F}{Nu/2}\right)$. The first term is how long it takes for the original owner to send the file; the second how long it takes for a single peer to receive the file; and the third term comes from the need to upload F bits to $N-1$ peers by only $N/2$ peers contributing u each. The last term can be rewritten a bit and for $N \geq 2$ it is always larger than the first term, so we get: $\max\left(\frac{F}{d}, \frac{2F}{u} \cdot \frac{N-1}{N}\right)$.

(This assumes the file originally is at a non-leecher; if it's at a leecher, then obviously the others will never get the file.)

Instead of deriving the formula as above, you could also start with the standard formula from the handout, and substitute: the server's role in that formula is taken by the peer who initially has the file, so $u_s = u$, and we have $N-1$ other peers left; and u_i then is u for half of the peers, and 0 for the other half, which you can substitute into $\sum_i u_i$ in the formula.

2 pt (c) Is the scalability significantly affected by the fact that only half of the peers are willing to upload? Explain.

Not really: the download time is still independent of N for large N, like in the standard case where all peers do upload.

End of this exam.

[1] Real P2P protocols, like BitTorrent, take measures to exclude "leechers", but we assume here that the leechers are simply allowed to download the data without uploading anything.