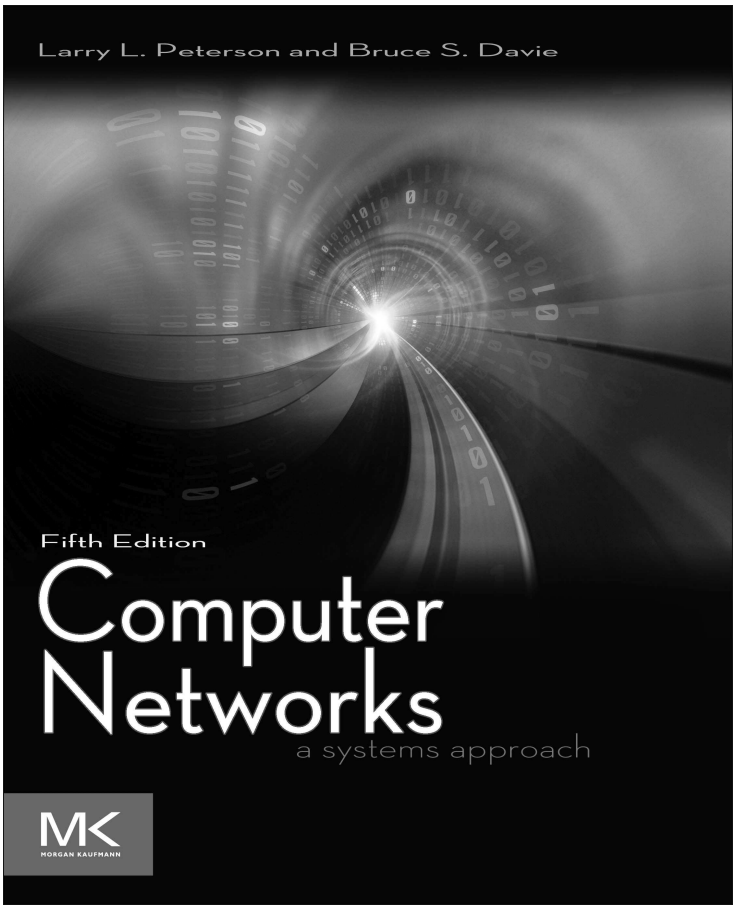
Computer Networks: A Systems Approach
Fifth Edition
Solutions Manual

Larry Peterson and Bruce Davie

2011



Dear Instructor:

This *Instructors' Manual* contains solutions to most of the exercises in the fifth edition of Peterson and Davie's *Computer Networks: A Systems Approach*.

Exercises are sorted (roughly) by section, not difficulty. While some exercises are more difficult than others, none are intended to be fiendishly tricky. A few exercises (notably, though not exclusively, the ones that involve calculating simple probabilities) require a modest amount of mathematical background; most do not. There is a sidebar summarizing much of the applicable basic probability theory in Chapter 2.

An occasional exercise is awkwardly or ambiguously worded in the text. This manual sometimes suggests better versions; also see the errata at the web site.

Where appropriate, relevant supplemental files for these solutions (*e.g.* programs) have been placed on the textbook web site, `http://mkp.com/computer-networks`. Useful other material can also be found there, such as errata, sample programming assignments, PowerPoint lecture slides, and EPS figures.

If you have any questions about these support materials, please contact your Morgan Kaufmann sales representative. If you would like to contribute your own teaching materials to this site, please contact our Associate Editor David Bevans, `D.Bevans@elsevier.com`.

We welcome bug reports and suggestions as to improvements for both the exercises and the solutions; these may be sent to netbugsPD5e@elsevier.com.

Larry Peterson
Bruce Davie
March, 2011

Solutions for Chapter 1

3. We will count the transfer as completed when the last data bit arrives at its destination. An alternative interpretation would be to count until the last ACK arrives back at the sender, in which case the time would be half an RTT (25 ms) longer.

   (a) 2 initial RTT's (100ms) + 1000KB/1.5Mbps (transmit) + RTT/2 (propagation = 25ms)
   $\approx$ 0.125 + 8Mbit/1.5Mbps = $0.125 + 5.333$ sec $= 5.458$ sec. If we pay more careful attention to when a mega is 10^6 versus 2^{20}, we get 8,192,000 bits/1,500,000 bps = 5.461 sec, for a total delay of 5.586 sec.

   (b) To the above we add the time for 999 RTTs (the number of RTTs between when packet 1 arrives and packet 1000 arrives), for a total of $5.586 + 49.95 = 55.536$.

   (c) This is 49.5 RTTs, plus the initial 2, for 2.575 seconds.

   (d) Right after the handshaking is done we send one packet. One RTT after the handshaking we send two packets. At n RTTs past the initial handshaking we have sent $1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1$ packets. At $n = 9$ we have thus been able to send all 1,000 packets; the last batch arrives 0.5 RTT later. Total time is 2+9.5 RTTs, or .575 sec.

4. The answer is in the book.

5. Propagation delay is 4×10^3 m/(2×10^8 m/s) = 2×10^{-5} sec = 20 μs. 100 bytes/20 μs is 5 bytes/μs, or 5 MBps, or 40 Mbps. For 512-byte packets, this rises to 204.8 Mbps.

6. The answer is in the book.

7. Postal addresses are strongly hierarchical (with a geographical hierarchy, which network addressing may or may not use). Addresses also provide embedded "routing information". Unlike typical network addresses, postal addresses are long and of variable length and contain a certain amount of redundant information. This last attribute makes them more tolerant of minor errors and inconsistencies. Telephone numbers, at least those assigned to landlines, are more similar to network addresses: they are (geographically) hierarchical, fixed-length, administratively assigned, and in more-or-less one-to-one correspondence with nodes.

8. One might want addresses to serve as *locators*, providing hints as to how data should be routed. One approach for this is to make addresses *hierarchical*.

   Another property might be *administratively assigned*, versus, say, the *factory-assigned* addresses used by Ethernet. Other address attributes that might be relevant are *fixed-length* v. *variable-length*, and *absolute* v. *relative* (like file names).

If you phone a toll-free number for a large retailer, any of dozens of phones may answer. Arguably, then, all these phones have the same non-unique "address". A more traditional application for non-unique addresses might be for reaching any of several equivalent servers (or routers). Non-unique addresses are also useful when global reachability is not required, such as to address the computers within a single corporation when those computers cannot be reached from outside the corporation.

9. Video or audio teleconference transmissions among a reasonably large number of widely spread sites would be an excellent candidate: unicast would require a separate connection between each pair of sites, while broadcast would send far too much traffic to sites not interested in receiving it. Delivery of video and audio streams for a television channel only to those households currently interested in watching that channel is another application.

   Trying to reach any of several equivalent servers, each of which can provide the answer to some query, would be another possible use, although the receiver of many responses to the query would need to deal with the possibly large volume of responses.

10. STDM and FDM both work best for channels with constant and uniform bandwidth requirements. For both mechanisms bandwidth that goes unused by one channel is simply wasted, not available to other channels. Computer communications are bursty and have long idle periods; such usage patterns would magnify this waste.

    FDM and STDM also require that channels be allocated (and, for FDM, be assigned bandwidth) well in advance. Again, the connection requirements for computing tend to be too dynamic for this; at the very least, this would pretty much preclude using one channel per connection.

    FDM was preferred historically for TV/radio because it is very simple to build receivers; it also supports different channel sizes. STDM was preferred for voice because it makes somewhat more efficient use of the underlying bandwidth of the medium, and because channels with different capacities was not originally an issue.

11. 10 Gbps = 10^{10} bps, meaning each bit is 10^{-10} sec (0.1 ns) wide. The length in the wire of such a bit is .1 ns $\times$ 2.3×10^8 m/sec = 0.023 m or 23mm

12. x KB is $8 \times 1024 \times x$ bits. y Mbps is $y \times 10^6$ bps; the transmission time would be $8 \times 1024 \times x/y \times 10^6$ sec = $8.192x/y$ ms.

13. (a) The minimum RTT is $2 \times 385,000,000$ m / 3×10^8 m/s = 2.57 seconds.

    (b) The delay$\times$bandwidth product is 2.57 s$\times$1 Gbps = 2.57Gb = 321 MB.

    (c) This represents the amount of data the sender can send before it would be possible to receive a response.

(d) We require at least one RTT from sending the request before the first bit of the picture could begin arriving at the ground (TCP would take longer). 25 MB is 200Mb. Assuming bandwidth delay only, it would then take 200Mb/1000Mbps = 0.2 seconds to finish sending, for a total time of $0.2 + 2.57 = 2.77$ sec until the last picture bit arrives on earth.

14. The answer is in the book.

15. (a) Delay-sensitive; the messages exchanged are short.

    (b) Bandwidth-sensitive, particularly for large files. (Technically this does presume that the underlying protocol uses a large message size or window size; stop-and-wait transmission (as in Section 2.5 of the text) with a small message size would be delay-sensitive.)

    (c) Delay-sensitive; directories are typically of modest size.

    (d) Delay-sensitive; a file's attributes are typically much smaller than the file itself.

16. (a) On a 100 Mbps network, each bit takes $1/10^8 = 10$ ns to transmit. One packet consists of 12000 bits, and so is delayed due to bandwidth (serialization) by 120 μs along each link. The packet is also delayed 10 μs on each of the two links due to propagation delay, for a total of 260 μs.

    (b) With three switches and four links, the delay is

    $$4 \times 120\mu\text{s} + 4 \times 10\mu\text{s} = 520\mu\text{s}$$

    (c) With cut-through, the switch delays the packet by 200 bits = 2 μs. There is still one 120 μs delay waiting for the last bit, and 20 μs of propagation delay, so the total is 142 μs. To put it another way, the last bit still arrives 120 μs after the first bit; the first bit now faces two link delays and one switch delay but never has to wait for the last bit along the way.

17. The answer is in the book.

18. (a) The effective bandwidth is 100 Mbps; the sender can send data steadily at this rate and the switches simply stream it along the pipeline. We are assuming here that no ACKs are sent, and that the switches can keep up and can buffer at least one packet.

    (b) The data packet takes 520 μs as in 16(b) above to be delivered; the 400 bit ACKs take 4 μs/link to be sent back, plus propagation, for a total of 4×4 μs $+4 \times 10$ μs = 56 μs; thus the total RTT is 576 μs. 12000 bits in 576 μs is about 20.8 Mbps.

    (c) $100 \times 4.7 \times 10^9$ bytes / 12 hours = 4.7×10^{11} bytes/(12×3600 s) $\approx$ 10.9 MBps = 87 Mbps.

19. (a) 100×10^6bps $\times$ 10×10^{-6} sec = 1000 bits = 125 bytes.

(b) The first-bit delay is 520 μs through the store-and-forward switch, as in 16(a). 100×10^6bps $\times$ 520$\times 10^{-6}$ sec = 52000 bits = 650 bytes.

(c) 1.5$\times 10^6$ bps $\times$ 50 $\times$ 10^{-3} sec = 75,000 bits = 9375 bytes.

(d) The path is *through* a satellite, *i.e.* between two ground stations, not *to* a satellite; this ground-to-satellite-to-ground path makes the total one-way travel distance 2$\times$35,900,000 meters. With a propagation speed of c = 3×10^8 meters/sec, the one-way propagation delay is thus 2$\times$35,900,000/c = 0.24 sec. Bandwidth$\times$delay is thus 1.5×10^6 bps $\times$ 0.24 sec = 360,000 bits $\approx$ 45 KBytes

20. (a) Per-link transmit delay is 10^4 bits / 10^8 bps = 100 μs. Total transmission time including link and switch propagation delays = $2\times 100+2\times 20+35 = 275$ μs.

(b) When sending as two packets, the time to transmit one packet is cut in half. Here is a table of times for various events:

T=0	start
T=50	A finishes sending packet 1, starts packet 2
T=70	packet 1 finishes arriving at S
T=105	packet 1 departs for B
T=100	A finishes sending packet 2
T=155	packet 2 departs for B
T=175	bit 1 of packet 2 arrives at B
T=225	last bit of packet 2 arrives at B

This is smaller than the answer to part (a) because packet 1 starts to make its way through the switch while packet 2 is still being transmitted on the first link, effectively getting a 50 μs head start. Smaller is faster, here.

21. (a) Without compression the total time is 1 MB/$bandwidth$. When we compress the file, the total time is

$$compression\_time + compressed\_size/bandwidth$$

Equating these and rearranging, we get

$$bandwidth = compression\_size\_reduction/compression\_time$$

= 0.5 MB/1 sec = 0.5 MB/sec for the first case,
= 0.6 MB/2 sec = 0.3 MB/sec for the second case.

(b) Latency doesn't affect the answer because it would affect the compressed and uncompressed transmission equally.

22. The number of packets needed, N, is $\lceil 10^6/D \rceil$, where D is the packet data size. Given that overhead = 50$\times N$ and loss = D (we have already counted the lost packet's header in the overhead), we have overhead+loss = $50 \times \lceil 10^6/D \rceil + D$.

D	overhead+loss
1000	51000
10000	15000
20000	22500

The optimal size is 10,000 bytes which minimizes the above function.

23. Comparison of circuits and packets result as follows :

    (a) Circuits pay an up-front penalty of 1024 bytes being sent on one round trip for a total data count of $2048 + n$, whereas packets pay an ongoing per packet cost of 24 bytes for a total count of $1024 \times n/1000$. So the question really asks how many packet headers does it take to exceed 2048 bytes, which is 86. Thus for files 86,000 bytes or longer, using packets results in more total data sent on the wire.

    (b) The total transfer latency for packets is the sum of the transmit delays, where the per-packet transmit time t is the packet size over the bandwidth b ($8192/b$), introduced by each of s switches ($s \times t$), total propagation delay for the links ($(s+2) \times 0.002$), the per packet processing delays introduced by each switch ($s \times 0.001$), and the transmit delay for all the packets, where the total packet count c is $n/1000$, at the source ($c \times t$). Resulting in a total latency of $(8192s/b) + 0.003s + 0.004 + (8.192n/b) = (0.02924 + 0.000002048n)$ seconds. The total latency for circuits is the transmit delay for the whole file ($8n/b$), the total propagation delay for the links, and the setup cost for the circuit which is just like sending one packet each way on the path. Solving the resulting inequality $0.02924 + 8.192(n/b) > 0.076576 + 8(n/b)$ for n shows that circuits achieve a lower delay for files larger than or equal to 987,000 B.

    (c) Only the payload to overhead ratio size effects the number of bits sent, and there the relationship is simple. The following table show the latency results of varying the parameters by solving for the n where circuits become faster, as above. This table does not show how rapidly the performance diverges; for varying p it can be significant.

s	b	p	pivotal n
5	4 Mbps	1000	987000
6	4 Mbps	1000	1133000
7	4 Mbps	1000	1280000
8	4 Mbps	1000	1427000
9	4 Mbps	1000	1574000
10	4 Mbps	1000	1721000
5	1 Mbps	1000	471000
5	2 Mbps	1000	643000
5	8 Mbps	1000	1674000
5	16 Mbps	1000	3049000
5	4 Mbps	512	24000
5	4 Mbps	768	72000
5	4 Mbps	1014	2400000

    (d) Many responses are probably reasonable here. The model only considers the network implications, and does not take into account usage of processing or state storage capabilities on the switches. The model also ignores the presence of other traffic or of more complicated topologies.

24. The time to send one 12000-bit packet is 12000 bits/100 Mbps = 120 μs. The length of cable needed to exactly contain such a packet is 120 μs $\times$ 2×10^8 m/sec = 24,000 meters.

    12000 bits in 24000 meters is 50 bits per 100 m. With an extra 10 bits of delay in each 100 m, we have a total of 60 bits/100 m or 0.6 bits/m. A 12000-bit packet now fills 12000/(.6 bits/m) = 20,000 meters.

25. For music we would need considerably more bandwidth, but we could tolerate high (but bounded) delays. We could *not* necessarily tolerate higher jitter, though; see Section 6.5.1.

    We might accept an audible error in voice traffic every few seconds; we might reasonably want the error rate during music transmission to be a hundredfold smaller. Audible errors would come either from outright packet loss, or from jitter (a packet's not arriving on time).

    Latency requirements for music, however, might be much lower; a several-second delay would be inconsequential. Voice traffic has at least a tenfold faster requirement here.

26. (a) $640 \times 480 \times 3 \times 30$ bytes/sec = 26.4 MB/sec

    (b) $160 \times 120 \times 1 \times 5$ = 96,000 bytes/sec = 94KB/sec

    (c) 650MB/75 min = 8.7 MB/min = 148 KB/sec

    (d) $8 \times 10 \times 72 \times 72$ pixels = 414,720 bits = 51,840 bytes. At 14,400 bits/sec, this would take 28.8 seconds (ignoring overhead for framing and acknowledgments).

27. The answer is in the book.

28. (a) A file server needs lots of peak bandwidth. Latency is relevant only if it dominates bandwidth; jitter and average bandwidth are inconsequential. No lost data is acceptable, but without real-time requirements we can simply retransmit lost data.

    (b) A print server needs less bandwidth than a file server (unless images are extremely large). We may be willing to accept higher latency than (a), also.

    (c) A file server *is* a digital library of a sort, but in general the world wide web gets along reasonably well with much less peak bandwidth than most file servers provide.

    (d) For instrument monitoring we don't care about latency or jitter. If data were continually generated, rather than bursty, we might be concerned mostly with average bandwidth rather than peak, and if the data really were routine we might just accept a certain fraction of loss.

    (e) For voice we need guaranteed average bandwidth and bounds on latency and jitter. Some lost data might be acceptable; e.g. resulting in minor dropouts many seconds apart.

(f) For video we are primarily concerned with average bandwidth. For the simple monitoring application here, relatively modest video of Exercise 26(b) might suffice; we could even go to monochrome (1 bit/pixel), at which point $160 \times 120 \times 5$ frames/sec requires 12KB/sec. We could tolerate multi-second latency delays; the primary restriction is that if the monitoring revealed a need for intervention then we still have time to act. Considerable loss, even of entire frames, would be acceptable.

(g) Full-scale television requires massive bandwidth. Latency, however, could be hours. Jitter would be limited only by our capacity to absorb the arrival-time variations by buffering. Some loss would be acceptable, but large losses would be visually annoying.

29. In STDM the offered timeslices are always the same length, and are wasted if they are unused by the assigned station. The round-robin access mechanism would generally give each station only as much time as it needed to transmit, or none if the station had nothing to send, and so network utilization would be expected to be much higher.

30. (a) In the absence of any packet losses or duplications, when we are expecting the Nth packet we *get* the Nth packet, and so we can keep track of N locally at the receiver.

(b) The scheme outlined here is the stop-and-wait algorithm of Section 2.5; as is indicated there, a header with at least one bit of sequence number is needed (to distinguish between receiving a new packet and a duplication of the previous packet).

(c) With out-of-order delivery allowed, packets up to 1 minute apart must be distinguishable via sequence number. Otherwise a very old packet might arrive and be accepted as current. Sequence numbers would have to count as high as

$$bandwidth \times 1\,\text{minute}\,/packet\_size$$

31. In each case we assume the local clock starts at 1000.

(a) Latency: 100. Bandwidth: high enough to read the clock every 1 unit.

1000	1100	
1001	1101	
1002	1102	
1003	1104	tiny bit of jitter: latency = 101
1004	1104	

(b) Latency=100; bandwidth: only enough to read the clock every 10 units. Arrival times fluctuate due to jitter.

1000	1100	
1020	1110	latency = 90
1040	1145	
1060	1180	latency = 120
1080	1184	

(c) Latency = 5; zero jitter here:

1000	1005	
1001	1006	
1003	1008	we lost 1002
1004	1009	
1005	1010	

32. Generally, with MAX_PENDING =1, one or two connections will be accepted and queued; that is, the data won't be delivered to the server. The others will be ignored; eventually they will time out.

    When the first client exits, any queued connections are processed.

34. Note that UDP accepts a packet of data from any source at any time; TCP requires an advance connection. Thus, two clients can now talk simultaneously; their messages will be interleaved on the server.