Solutions for Homework #2

EE122: Introduction to Communication Networks

(Fall 2006)

Department of Electrical Engineering and Computer Sciences
College of Engineering
University of California, Berkeley

Vern Paxson / Sukun Kim / Dilip Antony Joseph

1. Problems from Peterson & Davie.

   (a) **Exercise 3.1** (11 points)

   Virtual Circuit Table Entries for Switch 1

In Port	In VCI	Out Port	Out VCI
2	0	1	0
3	0	1	1
0	0	1	2
1	3	2	1

   Virtual Circuit Table Entries for Switch 2

In Port	In VCI	Out Port	Out VCI
3	0	0	0
3	1	1	0
2	0	0	1
3	2	0	2
1	1	0	3
1	2	3	3

   Virtual Circuit Table Entries for Switch 3

In Port	In VCI	Out Port	Out VCI
0	0	3	0
0	1	2	0
0	2	3	1
0	3	1	0

Virtual Circuit Table Entries for Switch 4

In Port	In VCI	Out Port	Out VCI
3	0	1	0
2	0	3	1
0	0	3	2

(b) **Exercise 3.6** (20 points)

In Figure 3.9, instead of rotating the list of ports without modification, overwrite the outgoing port number with the incoming port number, then rotate. In Switch 1, the rightmost number of (3, 0, 1) is 1, and indicates the outgoing port. Overwrite it with the incoming port 2 (3, 0, 2), then rotate to the right (2, 3, 0). When leaving Switch 2, this will become (3, 2, 3). After Switch 3, the list becomes (0, 3, 2). This is the list Host B receives, and indicates the ports to take in each switch from the left to the right.

Since the order is reversed, we need a bit to specify from which end to read first. In this case, we specify to read from the left, and rotate to the left. At Switch 3, the leftmost port number of (0, 3, 2) is 0. Therefore, overwrite 0 with incoming port 3 (3, 3, 2), then rotate to the left (3, 2, 3). Since we are reading from the left, the list needs be rotated to the left. The list after Switch 3 becomes (3, 2, 3), and after Switch 2 it becomes (2, 3, 0). After Switch 1, Host A receives (3, 0, 1). Host A can send again, setting the bit to read from the right again.

(c) **Exercise 4.4** (20 points)

The size of the TCP message is 2,068 bytes, including header and data. Unlike the IP header, the TCP header is transparent to the network, and is not placed in each fragmented packet.

The MTU of a network is the largest IP datagram that the network can carry, not including the link-layer header. In the first network, the MTU is 1,024 bytes, for which 20 bytes are used for the IP header, leaving 1,004 bytes for IP data payload. Since fragmentation occurs on 8-byte boundaries, only 1,000 bytes of data can be put into the first fragment. It will have an offset of 0. The second fragment will also include 1,000 bytes of data, this time with an offset of 125 (since $125 \cdot 8 = 1,000$). The third packet will have the remaining 68 bytes, with an offset of 250 ($250 \cdot 8 = 2,000$).

In the second network, the MTU is 512 bytes. Considering 20 bytes of IP header and 8-byte granularity, the IP data payload can be up to 488 bytes. Of the three packets from the first network, the first packet will be fragmented into 488, 488, and 24 byte packets, with offsets of 0, 61, and 122, repectively. The second packet is similarly fragmented. However, since its original offset is 125, the starting offset iof its fragmented packets is also 125, instead of 0. This is because fragment offsets are always in terms of the

original, total IP datagram. Therefore, the 3 fragments have offsets of 125, 186, 247 respectively. The third packet of 24 bytes is small enough that fragmentation does not occur.

Therefore, the receiver will receive 7 packets:

Fragment	Size	Offset
1	488	0
2	488	61
3	24	122
4	488	125
5	488	186
6	24	247
7	68	250

2. **The Design of IP.**

   The length field in the IP header is 4 bits wide and thus the largest value it can encode is $1111_2 = 15$. The length is specified in units of 4 bytes, and therefore the longest possible IP header is $15 \cdot 4 = 60$ bytes. In the IP header, 20 bytes are always consumed by the non-optional fields. This leaves at most 40 bytes for use in options.

   The source routing options require 3 bytes for meta-information, which leaves 37 bytes for the IP addresses specifying the intermediary routers. Therefore, up to 9 routers can be included.

3. **Subnetting and Classless Interdomain-Routing (CIDR).**

   (a) Since the first 22 bits of the blocks are the same, we would have:

   Subnet Mask: 255.255.252.0
   Subnet Number: 147.17.204.0

   (b) For classful addressing, routers outside of the ISP could only view its blocks aggregated into a full class B network: **147.17.0.0/16**

   (c) Similar to the first case, the 4 class C networks have the same top 22 bits. Therefore, with CIDR those four networks can be advertised as a single entry: **197.17.204.0/22**

4. **Comparison of Stop-and-Wait and Sliding Window.**

   First, some preliminaries:

- Each data packet holds 1,460 bytes of payload, so to transfer 90,000 bytes of payload will require 61 full-sized packets plus one 940-byte packet ($61 \cdot 1460 + 940 = 90000$).
- For a full-sized packet, the time to send it from Berkeley to San Francisco is:

$$T_{\text{data}} = 5 \text{ ms} + \frac{(1460 + 40) \text{ bytes} \cdot 8 \text{ bits/byte}}{100 \text{ Mbps}} = 5.12 \text{ ms}$$

- The same calculation for the 940-byte packet yields a latency of $T_{\text{partial}} = 5.0784$ ms.
- Similarly, the time to send back a 40-byte acknowledgement is $T_{\text{ack}} = 5.0032$ ms.
- The round-trip time (RTT) for a full-sized packet is therefore:

$$\text{RTT} = T_{\text{data}} + T_{\text{ack}} = 5.12 \text{ ms} + 5.0032 \text{ ms} = 10.1232 \text{ ms}$$

Given these, we proceed as follows:

(a) To send the file with Stop-and-Wait, each full-sized round requires time RTT $= T_{\text{data}} + T_{\text{ack}}$ (see Figure 1). We include the final partial packet to this to get:

$$T_{\text{stop-and-wait}} = 61 \cdot 10.1232 + (5.0784 + 5.0032) = 627.5968 \text{ ms}$$

(b) The throughput is the amount of data transferred divided by the time it took to send it, so:

$$\text{Throughput} = 90000\text{B}/0.6275968 = 143.4042 \text{ KBps} = 1.147233 \text{ Mbps}$$

(c) An important insight is that each acknowledgment immediately allows us to send another packet (because the window slides forward; see Figure 2). Therefore, for a window of 3 packets #1, #4, #7, and so on are each sent with an interval of $T_{\text{data}} + T_{\text{ack}}$ between them. Similarly for #2, #5, #8, . . ., and #3, #6, #9, ...

Let us now consider the last packet we send, which is #62 (and is a partial packet rather than a full one like the others). It is part of the series #2, #5, ...

Packet #2 is sent just after transmitting #1, so at time:

$$T_2 = \frac{(1460 + 40) \text{ bytes} \cdot 8 \text{ bits/byte}}{100 \text{ Mbps}} = 0.12 \text{ ms}$$

Packet #5 is sent at $T_2 +$ RTT; #8 at $T_2 + 2 \cdot$ RTT, etc. (Again, see Figure 2.)

In particular, packet #62 is sent at

$$T_2 + 20 \cdot \text{ RTT} = 0.12 + 20 \cdot 10.1232 = 202.584 \text{ ms} \quad (1)$$

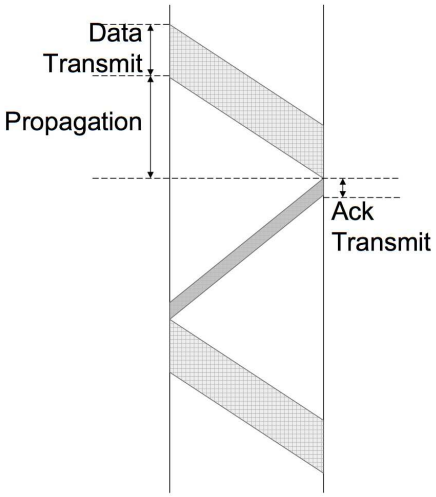

Figure 1: Stop-and-Wait

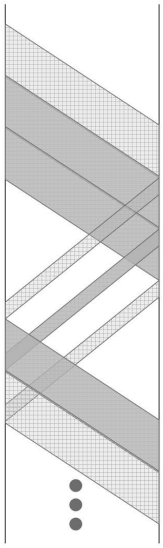

Figure 2: Sliding Window with SWS=RWS=3

Since this is the last packet, we are done when its acknowledgment arrives, which takes

$$T_{\text{partial}} + T_{\text{ack}} = 5.0784 + 5.0032 = 10.0816 \text{ ms} \tag{2}$$

Therefore, the entire transmission takes $202.584 + 10.0816 = 212.6656$ ms. The corresponding throughput is

$$\text{Throughput} = 90000\text{B}/0.2126656 = 423.1996 \text{ KBps} = 3.385597 \text{ Mbps}$$

This reflects a gain of nearly 3 times—not quite a full factor of 3 because the window doesn't allow us to fully parallelize transmission, but rather to pipeline it.

(d) The bandwidth-delay product of this path is

$$100 \text{ Mbps} \cdot 10 \text{ ms} = 1 \text{ Mb} = 125 \text{ KB}$$

A window size of 125 KB corresponds to sending *the entire file in a single burst*.

We can double-check this as follows. The transmission time for a single full packet is $T_2 = 0.12$ ms as given by Equation 1 above. Therefore if we send all 61 full-sized packets out back-to-back, we'll have finished transmitting the last one at time $61 \cdot T_2 = 7.32$ ms. This is *before* the first acknowledgement has arrived.

(It didn't have to work out this particular way. If the file were larger than 125 KB, then we would not send it all at once; but after sending the first 125 KB, just as we were ready to send the next packet, the acknowledgment for the first packet would arrive—which would cause the window to slide and for the first time allow us to send a packet beyond initial 125 KB. The general observation is that a window size greater or equal to the bandwidth-delay product allows us to fully use the capacity of the network path. There is no "down time" spent waiting for acknowledgments to advance the window so we can send more; we're always able to send at the rate the network can sustain.)

Therefore, in our particular case, by setting the window to the delay-bandwidth product we will send the final, partial packet at a time 7.32 ms after we begin transmission. Its round-trip time is 10.0816 ms (see Equation 2 above), so we are all done with the transfer after 7.32 + 10.0816 = 17.4016 ms, for a throughput of 5.171938 MB/s = 41.37551 Mbps.

(e) Setting the window any higher won't change the throughput we can achieve, since the window computed using the bandwidth-delay product already allows us to send all our data in a single flight.

More generally, using the bandwidth-delay product for the window allows us to completely "fill the pipe," namely to keep the forward transmission path continuously occupied. Any larger window can't enable us to go any faster, since we're already going as fast as the network can possibly let us go. (The larger window *can* lead to larger "queues" inside the network, since we give the network more load to process in a given amount of time. Ironically, this can lead to some of our packets being dropped due to exhausted queue space, which, as we'll see when we study congestion control, can actually cause the transfer to go much *slower*.)

5. **DNS.**

(a) The namer servers visited:

i. *a.root-servers.net* as the initial root server

ii. *a3.nstld.com* (or another TLD serer) to resolve names in `edu`.

iii. *adns1.berkeley.edu* (or another of UCB's servers: *ucb-ns.nyu.edu*, *ns.v6.berkeley.edu*, *dns2.ucla.edu*, *adns2.berkeley.edu*, *phloem.uoregon.edu*) to resolve names in `berkeley.edu`.

iv. Again, *adns1.berkeley.edu* (or another of EECS's servers: *ns.cs.berkeley.edu*, *ns.eecs.berkeley.edu*, *cgl.ucsf.edu*, *adns2.berkeley.edu*, *vangogh.cs.berkeley.edu*) to resolve names in `eecs.berkeley.edu`.

(b) 128.32.48.169

(c) The same sequence as above occurs, except at step 3 both no Answer is returned nor is any further NS record. (The server will return an SOA [Source of Authority] record, which we haven't discussed, but this is basically its way of saying that the hostname definitely does not exist.)

6. **Email.**

(a) The commands used are as follows:

```
HELO imail.eecs.berkeley.edu
MAIL FROM:<ee122-tb@imail.eecs.berkeley.edu>
RCPT TO:<binetude@calmail.berkeley.edu>
DATA
From: Golden Bear <ee122-tb@imail.eecs.berkeley.edu>
To: EE122 <binetude@calmail.berkeley.edu>
Subject: My Answer To Homework #2, Problem #4
Hi.
I am Golden Bear.
.
QUIT
```

(b)
```
RCPT to:<vern@icir.org>
```

returns

```
550 relay not permitted
```

calmail.berkeley.edu does not allow arbitrary hosts to relay through it. (Note, it's possible that you used a host and/or From address for which it actually would allow the relaying. We did not mark off in this case.)