CS446/646 Spring 2018

Solution to Test 1

There are 5 problems, each problem equally worth.

Problem 1. Two hosts A and B are connected by a link of length m meters. The link's rate is R bps. The signal's propagation speed is s meters/s. A has to send a message of size L bits to B

a) What is the propagation delay?
b) What is the transmission delay?
c) What is the end-to-end delay, ignoring the queuing and processing delays

Solution:

a) Propagation delay = m/s
b) Transmission delay = L/R

End to end delay = transmission delay + propagation delay = m/s + L/R

Problem 2. Two hosts, A and B, are 4 hops apart in a packet-switched network (i.e., three routers on the path from A to B). All 4 links from A to B are at rate 10Mbps. Host A needs to send a file of size 10Gb to B. Ignore queuing, propagation, and processing delays. Assume packets do not contain headers.

a) What is the end-to-end delay if A sends the entire file as a packet
b) What is the end-to-end delay if A breaks the file up into packets of size 10Kb each?

Solution: Suppose that the three routers are R1, R2, and R3 and the path of transmission is A->R1->R2->R3->B.

*a) End to end delay = sum {end-to-end delay of each of the 4 links} = 4 * (10Gb/10Mbps) = 4000s*
b) The number of packets is 10Gb/10Kb = 1,000,000 packets. Packet transmission time is 10Kb/10Mbps=0.001s. The table below represents the time each packets reaches each node

time	*R1*	*R2*	*R3*	*B*
0.001	*1st pkt*			
0.002	*2st pkt*	*1st pkt*		
0.003	*3st pkt*	*2st pkt*	*1st pkt*	
0.004	*4st pkt*	*3st pkt*	*2st pkt*	*1st pkt*
0.005	*5st pkt*	*4st pkt*	*3st pkt*	*2st pkt*
…				
1,000	*1,000,000th pkt*	*999,999th pkt*	*999,998th pkt*	*999,997th pkt*
1,000.001		*1,000,000th pkt*	*999,999th pkt*	*999,998th pkt*
1,000.002			*1,000,000th pkt*	*999,999th pkt*

1,000.003				*1,000,000th pkt*

Thus, it takes 1,000.003 seconds for the entire file to reach B.

Problem 3. A client sends a 128-byte request to a server located 100km away over a 1Gbps optical fiber. Light travels at the speed of light in optical fiber. What is the efficiency of the line during this remote procedure call.

Solution:

The round-trip propagation delay between the client and the server, RTT, is:

$$RTT = \frac{2 \times 100 \times 10^3 m}{2.998 \times 10^8 m/s} = 6.67 \times 10^{-4} s$$

The time needed to actually transmit the request into the 1 Gbps fiber is:

$$d_{trans} = \frac{request\ size\ L}{transmission\ rate\ R} = \frac{128 \times 8 bits}{10^9 bits/s} = 1.024 \times 10^{-6} s$$

Assuming for simplicity that ACK packets are extremely small, so that we can ignore their transmission time and that the server can send an ACK as soon as the last bit of a data request is received, the ACK emerges back at the client at time $t = RTT + d_{trans}$. The efficiency is:

$$E = \frac{d_{trans}}{RTT + d_{trans}} \times 100\% = \frac{1.024 \times 10^{-6}}{6.67 \times 10^{-4}} \times 100\% = 0.154\%$$

Thus, the efficiency of the line during this remote procedure call is 0.154%.

Problem 4. Both UDP and TCP use port numbers to identify the destination entity when delivering a message. Give 2 reasons for why these protocols invented a new abstract ID (port numbers), instead of using process IDs, which already existed when these two protocols were designed.

Solution:

First, a process can have multiple connections to multiple clients at a time (a process may have more than one socket), hence simple connection identifiers like process ID would not be unique. Second, an application process is assigned a process identifier number (process ID), which may be different each time when the process is started. Process IDs also differ between operation system platforms, hence they are not uniform. So, these protocols need to invented a new uniform and unique abstract ID(port number).

Problem 5. UDP and TCP use 1s complement for checksum. Suppose you have the following three 8-bit bytes: 01010011, 01010100, 01110100. What is the 1s complement of the sum of these bytes? Show all work. Why is it that UDP takes the 1s complement of the sum; that is, why not just use the sum? With the 1s complement scheme, how does the receiver detect errors? Is it possible that a 1-bit error will go undetected? How about a 2-bit error?

Solution:

sum of three bytes	0 0 0 1 1 1 0 0
checksum	1 1 1 0 0 0 1 1

Thus, the 1s complement of the sum of these bytes is 11100011.

To detect errors, the receiver adds the four 8-bit bytes (the three original bytes 01010011, 01010100, 01110100 and the checksum 11100011). If no errors are introduced into the packet, then the sum at the receiver will be 11111111. If the sum contains a zero, the receiver knows there has been an error. All one-bit errors will be detected, but two-bit errors may cannot be detected. For example, if the last digit of the first word is converted to a 0 and the last digit of the second word is converted to a 1, the sum of the four words will also be 11111111, so the errors aren't detected.

Problem 6. Consider a channel that can lose packets but has a maximal delay that is known. Modify protocol rdt2.1 (we discussed this in the class, see diagram of this protocol in the lecture slides) to include sender timeout and retransmit. Informally argue why your proposed protocol can communicate correctly over this channel.

Solution:

- *Here, we add a timer, whose value is greater than the known round-trip propagation delay. We add a timeout event to the "Wait for ACK or NAK0" and "Wait for ACK or NAK1" states. If the timeout event occurs, the most recently transmitted packet is retransmitted. Let us see why this protocol will still work with the rdt2.1 receiver.*
  - *Suppose the timeout is caused by a lost data packet, i.e., a packet on the sender- to-receiver channel. In this case, the receiver never received the previous transmission and, from the receiver's viewpoint, if the timeout retransmission is received, it look exactly the same as if the original transmission is being received.*
  - *Suppose now that an ACK is lost. The receiver will eventually retransmit the packet on a timeout. But a retransmission is exactly the same action that is take if an ACK is garbled. Thus the sender's reaction is the same with a loss, as with a garbled ACK. The rdt 2.1 receiver can already handle the case of a garbled ACK.*

Problem 7. Consider a reliable data transfer protocol that uses only negative acknowledgements. Suppose that the sender sends data only infrequently. Would a NAK-only protocol be preferable to a protocol that uses ACK only? Why? Now, suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?

Solution:

- *In a NAK only protocol, the loss of packet x is only detected by the receiver when packet x+1 is received. That is, the receivers receives x-1 and then x+1, only when x+1 is received does the receiver realize that x was missed. If there is a long delay between the transmission of x and the transmission of x+1, then it will be a long time until x can be recovered, under a NAK only protocol.*
- *On the other hand, if data is being sent often, then recovery under a NAK-only scheme could happen quickly. Moreover, if errors are infrequent, then NAKs are only occasionally sent (when needed), and ACK are never sent – a significant reduction in feedback in the NAK-only case over the ACK-only case.*