TCP/IP Extra Problem Set – Sample Solutions

Question 1:

2bits Netmask 255.255.255.192
This would provide 4 subnets each with 62 host addresses
subnet 0 (00) Host Addresses 192.168.255.1 to 192.168.255.62
subnet 1 (01) Host addresses 192.168.255.65 to 192.168.255.126
subnet 2 (10) Host addresses 192.168.255.129 to 192.168.255.190
subnet 3 (11) Host addresses 192.168.255.193 to 192.168.255.254.
We can ignore the first 24 bits in the IP address represented by the decimals 192.168.255 since those are for the baseline network which we are subnetting. If we look at subnet 0 then the last eight bits will always start with 00 which means that the decimal equivalent of their last 8 bits can be from 0 to 63. The all zeros and the all ones for a host address cannot be used which eliminates 0 and also 63 thus we have 62 left from 1 to 62. Looking at subnet 2 the first two bits of the last octet for all host addresses are always 10 thus the available combinations for the last six becomes decimal equivalent of 128-191. Again all 0s and all 1s cannot be used for host address (last six bits) thus the last octet will be 129-190 which provides for 62 host addresses again.

Question 2:

The first step is to determine the number of bits required to define the 3 subnets, which are two bits. The remainder 4 bits can represent the 12 hosts.

193.1.1.0/26 = **11000001.00000001.00000001.00000000**

Network Prefix **Subnet ID** **Host ID**

i) The extended network prefix is [**11000001.00000001.00000001.0000**]
ii) 28 bits
iii)

Subnet 0, first IP: **11000001.00000001.00000001.00000001**

Subnet 0, last IP: **11000001.00000001.00000001.00001110**

Subnet 1, first IP: **11000001.00000001.00000001.00010001**

Subnet 1, last IP: **11000001.00000001.00000001.00011110**

Subnet 2, first IP: **11000001.00000001.00000001.00100001**

Subnet 2, last IP: **11000001.00000001.00000001.00101110**

Question 3:

Infinite capacity means that after the slow start phase and without any flow control action the sender will always transmit based on the maximum window size. TCP is based on the self-clocked mechanism so the sender will transmit new data with the receipt of the ACKs. So, in this case the transmission rate will be: (64 . 1024 . 8[*bits/ sec*]) / (20 . 10^{-3}[*sec*]) = 26.2*Mbps*

By doubling the RTT the transmission rate will be half (13.1Mbps)

Question 4:

Consider the first network. The frame payload can be up to 1024 bytes (the MTU), which means up to 1004 bytes of IP datagram data (because we have 20 bytes of IP header). Because 1004 is not a multiple of 8, the maximum that a fragment can contain is 1000 bytes. We need to transfer 2048 + 20 = 2068 bytes of data. (The IP datagram data is the complete TCP packet - 20 bytes of TCP header plus 2048 bytes of data.) 2068 bytes of data can be broken int fragments of 1000, 1000 and 68. The sizes and offsets of the 3 fragments would be

Fragment 1: 1020 bytes, offset field value = 0
Fragment 2: 1020 bytes, offset field value = 125
Fragment 3: 88 bytes, offset field value = 250

(Remember, the packet size is data plus 20 byte of IP header. Fragment offsets are divided by 8 to fit in the 13 bit fragment offset field.) Over the second network, we can transmit up to 492 bytes of IP data (512 - 20). Because 492 is
not a multiple of 8, the maximum a fragment can contain is 488 bytes. The fragment of 88 bytes will not be fragmented further. The fragments of 1020 bytes WILL be fragmented further. Their 1000 bytes of data will be placed in fragments of size 488, 488 and 24.
The fragments arriving at the destination should be (not necessarily in this order of course):
Fragment 1: 488 + 20 = 508 bytes, offset field value = 0/8 = 0
Fragment 2: 488 + 20 = 508 bytes, offset field value = 488/8 = 61
Fragment 3: 24 + 20 = 44 bytes, offset field value = 61 + 488/8 = 122
Fragment 4: 488 + 20 = 508 bytes, offset field value = 122 + 24/8 = 125
Fragment 5: 488 + 20 = 508 bytes, offset field value = 125+ 488/8 = 186
Fragment 6: 24 + 20 = 44 bytes, offset field value = 186 + 488/8 = 247
Fragment 7: 88 bytes, offset field value = 250 (as above, not fragmented further)

Question 5:

If the RTT estimate is far too small, the retransmission timer will be too small and will timeout prematurely – resulting in the retransmission of packets that may not be lost. If the

RTT estimate is far too large, the retransmission timer will be too large. In this case, the packet will not be transmitted until a long time after it has been lost – resulting in long error recovery delays.