1. **TCP segments**
   a. Consider sequence numbers,
      First segment=80
      Second segment= 92
      Data in the first segment=92 - 80
      = 12

   b. Consider the second segment is lost but the first and third segment arrives at B. In the acknowledgment that Host B sends to Host A, then the acknowledgment number will be second segment of sequence number, that is **92**.

   c. Sequence number of A = 80
      Sequence number of B = 92
      Sequence number of C = 20 (New)

      All the three numbers have arrived at host B.
      ACK number = 80 + 92 + 20 = 192
      Host B will send 193 as the ACK number

2. **Throughput issues.**
   a. The utilization over two links is divided into two parts. For the first part, i.e., for the first scenario, the utilization if given as 120/256 = 47%
   For the second part, i.e., for the second scenario the utilization is given by 33/256 = 13%.

   b. The window size is given by W = ER * RTT.

      For the first scenario
      W = (120 * 10^3) * (128 * 10^3) = 15360 bits = **1920 bytes**

      For the second scenario
      W = (33 × 10^3) × (500 × 10^3) = 16500 bits = **2063 bytes**

   c. For the window size to be equal to the satellite link we equate both.
      ER = (256 * 10^3) = W / (500 * 10^3)
      W = 128,000 bits = 16,000 bytes

3. Pipelined protocols
   a. Differences between go-back-N and selective repeat pipelined protocols
      i. GO-BACK-N
         - In Go-Back-N Protocol, if the sent frame are find suspected then all the frames are re-transmitted from the lost packet to the last packet transmitted.
         - Sender window size of Go-Back-N Protocol is N.
         - In Go-Back-N Protocol, neither sender nor at receiver need sorting.

- In Go-Back-N Protocol, type of Acknowledgement is cumulative.

ii. Selective repeat
- In selective Repeat protocol, only those frames are re-transmitted which are found suspected.
- Sender window size of selective Repeat protocol is also N.
- In selective Repeat protocol, receiver side needs sorting to sort the frames.
- In selective Repeat protocol, type of Acknowledgement is individual.

4. Sending and receiving an acknowledgement from either side's FIN assures the other side that the message is sent and received. TCP entity must wait equal to twice the maximum expected segment lifetime. This connection is held in limbo to allow all connection messages (e.g., late duplicates) that may still exists out in the network to arrive or be discarded. It can be considered as the wait time. But the messages received after that can be discarded. If any messages arrive, TCP will know that they belong to a defunct connection and will discard them.'

5. Calculation for estimated RTT.

a. $\alpha = 0.125$
Estimated RTT = x
X = (1-.125)x +.125*500
X(.125) = 6.25
X= 6.25/1.25 = 50
X= 50

b. $\alpha = 0.75$
Estimated RTT = x
X = (1- .75) x +.75*500
X(.75) = 375
X= 375/.75
X=500

c. The performance of the sliding window protocol depends heavily on the timeout value, which in terms depends on the round trip time (RTT) between the communicating end-points. If the timeout value is too small, the source will time out too fast resulting in unnecessary retransmissions. On the other hand, if the timeout value is too large, the source will take too long to recover from errors.
To achieve good performance in data transfer TCP must determine the RTT for each connection .The value for alpha is a number between 0 and 1.
To make the matter even more difficult, the RTT changes over time due to changing conditions within the network. Therefore, TCP must continuously estimates the RTT and adjust the time out for each fragment transmitted.

6.

a. Calculation of SRTT(19)

SRTT (K + 1) = α * SRTT (K) + (1 – α) * RTT (K + 1)

SRTT(n) = α × SRTT(0) + (1 - α) × RTT × (α^n-1 + α^n-2 + … + α + 1)
= α × SRTT(0) + (1 - α) × RTT × (1- α^n)/(1- α)
Substituting the values we get,
SRTT(19) = 1.1 sec

b. Taking values from the equations above.
Substituting the values we get,
SRTT(19) = 2.9 sec; this is the case because the value of α is taken arbitrarily.

7. TCP

a. Differences between TCP flow and TCP congestion control

1. TCP flow Control

i. In flow control, Traffics are controlled which are flow from sender to a receiver.
ii. Data link layer and Transport layer handle it.
iii. In this, Receiver's data is prevented from being overwhelmed.
iv. In flow control, Only sender is responsible for the traffic.
v. In this, Traffic is prevented by slowly sending by the sender.
vi. In flow control, buffer overrun is restrained in the receiver.

2. TCP congestion control

i. In this, Traffics are controlled entering to the network.
ii. Network layer and Transport layer handle it.
iii. In this, Network is prevented from congestion.
iv. In this, Transport layer is responsible for the traffic.
v. In this, Traffic is prevented by slowly transmitting by the transport layer.
vi. In congestion control, buffer overrun is restrained in the intermediate systems in the network.

b. After the 50 octets arrived at the receiver, it will process and remove the 50 octets and offer a window of 1000 octets. However, the sender will now consider the presence of are 950 octets in transit in the network, hence the useable window is considered only 50 octets. Then the sender will again try sending a 50-octet segment, even though there is no space to push it. Generally, whenever the acknowledgement of a small segment is received, the useable window associated with the acknowledgement will cause another small size segment to be sent. Hence the useable window will break into smaller pieces.

8. Mechanisms to cope with SWS

a. The following strategy should be used:
   When a segment arrives, do not acknowledge immediately and waits until decent space but not more 500 ms to acknowledge.
b. Besides the receiver's strategy, the sender should use the following strategy:
   Sending TCP sends first data piece immediately. Subsequently, it accumulates data until receiving acknowledgement or enough data to fill MSS. Then, it sends it.

9. TO be written.
   a. Assuming cwnd increases by 1 MSS every time a batch of ACKs is received and assuming approximately constant round-trip times.
      Then transmission rate of TCP is written in w bytes/RTT cwnd increases by 1 MSS if every batch of ACKs received.

      The below steps are take for cwnd to increase from 6 MSS to 12 MSS:
      1 RTTs to to 7 MSS.
      2 RTTs to 8 MSS.
      3 RTTs to 9 MSS.
      4 RTTs to 10 MSS.
      5 RTTs to 11MSS.
      6 RTTs to 12 MSS.

   b. Connection up through time = 6 RTT
      Average throughout (in terms of MSS and RTT) =(6+7+8+9+10+11)/6
      = 51/6
      =8.5 MSS/RTT

10.

   a. Estimation of the round trip time and its variation **(Jacobson's algorithm)**
      i. For each sent segment, the time RTT to the arrival of the acknowledgment is measured.
      ii. The difference of the measured round trip time and the current smoothed estimate SRTT is calculated SERR = RTT – SRTT.
      iii. The smoothed estimate SRTT is updated (exponential averaging) SRTT ← (1− g) × SRTT + g × RTT
      iv. Similarly, the estimate for the delay variance SDEV is updated SDEV ← (1 − h) × SDEV + h × |SERR| – the recommended values for the coefficients g and h are
      g = 1/8 = 0.125 'average of the last 8 measurements'
      h = 1/4 = 0.25 'average of the last 4 measurements'

   b. W = (109* 0.06) / (16 * 103* 8) = 460 segments. Similar to the last question, now the window size should be 28= 256 segments. It will take **460 – 256 = 204 RTT.**
      Therefore, the total time to reach the full window is **(8 + 204) * 0.06 = 12.7 seconds.**

c. W = (109* 0.06) / (576 * 8) = 13000 segments. Assuming the sender was transmitting at the full window before the timeout, then our congestion threshold is 6500 segments. TCP will use exponential growth until the congestion threshold is passed, where additive increase begins. So, we can use exponential increase to a window size of: 213=8192, which will take 13 RTT. Now we do additive increase to get to the full window. This will take 13000 - 8192 = 4808 RTT using additive increase. Therefore, the total time to reach the full window is (13 + **4808) RTT = 4821* 0.06 = 289.3 seconds.**