TRANSMISSION CONTROL PROTOCOL

Transmission Control Protocol (TCP) is a connection-oriented, reliable protocol. TCP explicitlydefines connection establishment, data transfer, and connection teardown phases to provide aconnection-oriented service.

TCP Congestion -

Congestion refers to a network state where - The message traffic becomes so heavy that itslows down the network response time. Congestion is an important issue that can arise in Packet Switched Network. **Congestion leads to the loss of packets in transit.**

- Congestion is an important issue that can arise in **Packet Switched Network.**
- Congestion leads to the loss of packets in transit.
- So, it is necessary to control the congestion in network.
- It is not possible to completely avoid the congestion.

TCP Congestion Control :

The important service done by TCP is providing end to end transmission. This servicehas a significant role in congestion control in the network. Congestion free network which is an ideal network means taking the advantage of all the bandwidth in the network.

The sender's window size is determined not only by the receiver but also by congestion in the network. The sender has two pieces of information: the receiver-advertised window sizeand the congestion window size. The actual size of the window is the minimum of these two.

Actual window size = minimum (rwnd, cwnd)

Congestion Policy :

TCP's general policy for handling congestion is based on three phases: slow start, congestion avoidance, and congestion detection. In the slow-start phase, the sender startswith a very slow rate of transmission, but increases the rate rapidly to reach a threshold.
When the threshold is reached, the data rate is reduced to avoid congestion. Finally if

congestion is detected, the sender goes back to the slow-start or congestion avoidancephase based on how the congestion is detected.

Policy Transition :

We discussed three congestion policies in TCP. Now the question is when each of these policies is used and when TCP moves from one policy to another. To answer these questions, we need to refer to three versions of TCP: Taho TCP, Reno TCP, and New RenoTCP.

Taho TCP :

The early TCP, known as Taho TCP, used only two different algorithms in their congestion policy: slow start and congestion avoidance. We use Figure 24.31 to show the FSM for thisversion of TCP. However, we need to mention that we have deleted some small trivial actions, such as incrementing and resetting the number of duplicate ACKs, to make the FSM less crowded and simpler.

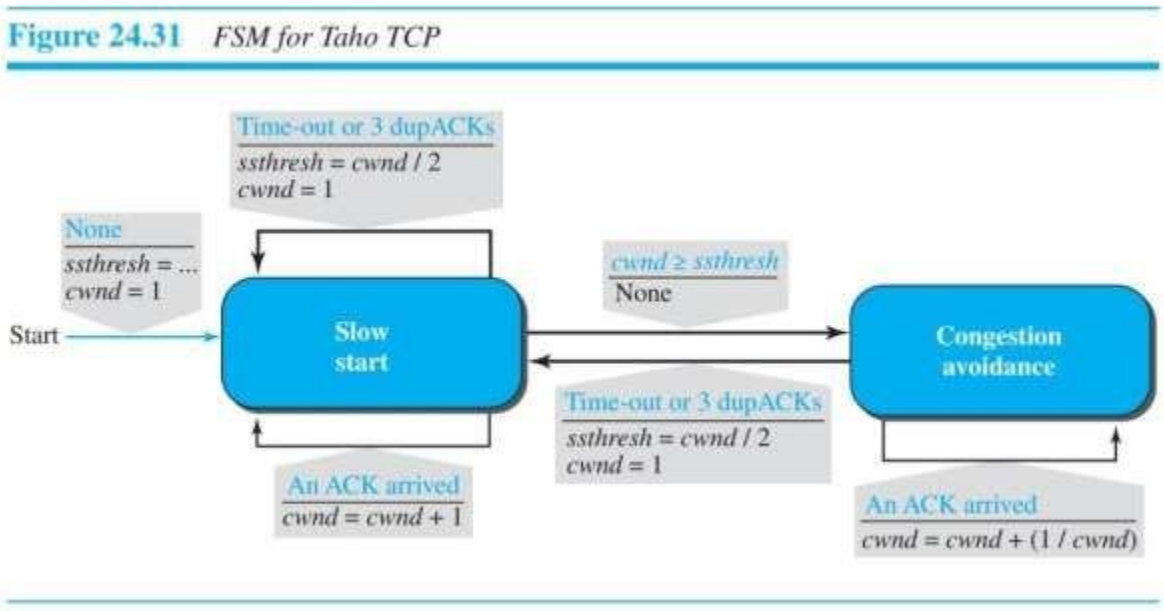


Figure 24.31 *FSM for Taho TCP*

Taho TCP treats the two signs used for congestion detection, time-out and three duplicate ACKs, in the same way. In this version, when the connection is established, TCP starts theslow-start algorithm and sets the ssthresh variable to a pre-agreed value (normally a

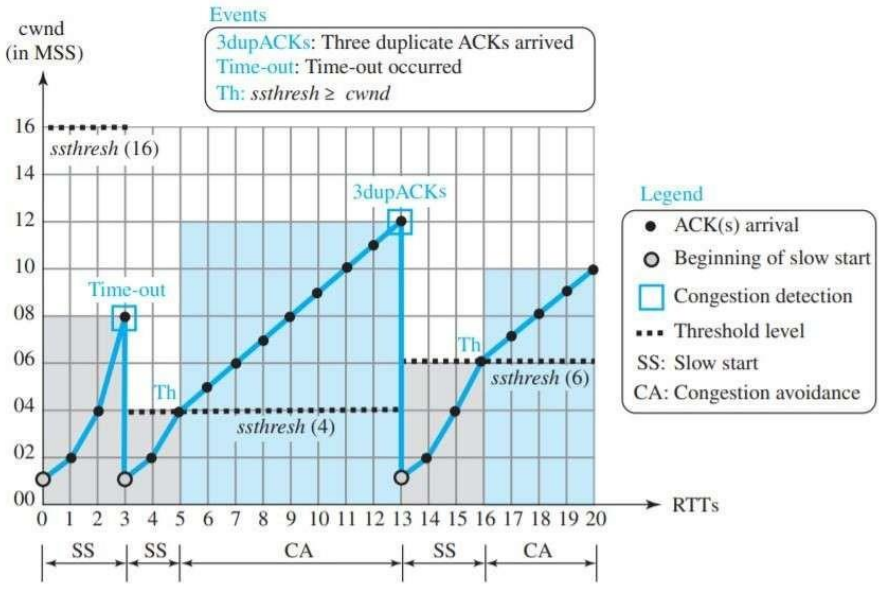
multiple of MSS) and the cwnd to 1 MSS. In this state, as we said before, each time an ACKarrives, the size of the congestion window is incremented by 1. We know that this policy is very aggressive and exponentially increases the size of the window, which may result in congestion. If congestion is detected (occurrence of time-out or arrival of three duplicate ACKs), TCP immediately interrupts this aggressive growth and restarts a new slow start algorithm by limiting the threshold to half of the current cwnd and resetting the congestion window to 1. In other words, not only does TCP restart from scratch, but it also learns how to adjust the threshold. If no congestion is detected while reaching the threshold, TCP learns that the ceiling of its ambition is reached; it should not continue at this speed. It moves to the congestion avoidance state and continues in that state.

In the congestion-avoidance state, the size of the congestion window is increased by 1 each time a number of ACKs equal to the current size of the window has been received. Forexample, if the window size is now 5 MSS, five more ACKs should be received before the size of the window becomes 6 MSS. Note that there is no ceiling for the size of the congestion window in this state; the conservative additive growth of the congestion window continues to the end of the data transfer phase unless congestion is detected.

If congestion is detected in this state, TCP again resets the value of the ssthresh to half of the current cwnd and moves to the slow-start state again. Although in this version of TCP the size of ssthresh is continuously adjusted in each congestion detection, this does not mean that it necessarily becomes lower than the previous value. For example, if the originalssthresh value is 8 MSS and congestion is detected when TCP is in the congestion avoidance state and the value of the cwnd is 20, the new value of the ssthresh is now 10, which means it has been increased.

Reno TCP:

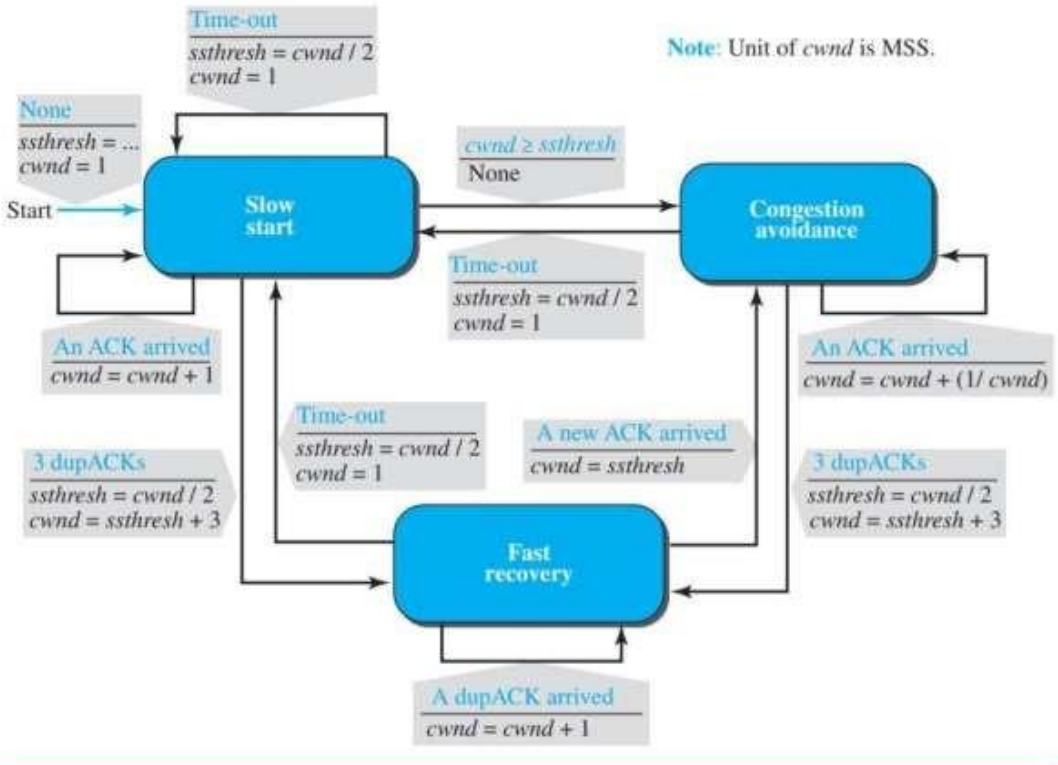
Figure 24.32 *Example 24.9*



A newer version of TCP, called Reno TCP, added a new state to the congestion-control FSM, called the fast-recovery state. This version treated the two signals of congestion, time-out and the arrival of three duplicate ACKs, differently. In this version, if a time-out occurs, TCP moves to the slow-start state (or starts a new round if it is already in this state); on the other hand, if three duplicate ACKs arrive, TCP moves to the fast-recovery state and remains there as long asmore duplicate ACKs arrive. The fast-recovery state is a state somewhere between the
slow-start and the congestion-avoidance states. It behaves like the slow start, in which the cwndgrows exponentially, but the cwnd starts with the value of ssthresh plus 3 MSS (instead of 1).
When TCP enters the fast-recovery state, three major events may occur. If duplicate ACKs continue to arrive, TCP stays in this state, but the cwnd grows exponentially. If a time-out occurs, TCP assumes that there is real congestion in the network and moves to the slow-start state. If a new (non duplicate) ACK arrives, TCP moves to the congestion-avoidance state, but deflates the size of the cwnd to the ssthresh value, as though the three duplicate ACKs have notoccurred, and transition is from the slow-start state to the congestion-avoidance state. Figure
24.33 shows the simplified FSM for Reno TCP. Again, we have removed some trivial events tosimplify the figure and discussion.

.

Figure 24.33 *FSM for Reno TCP*



New Reno TCP :

A later version of TCP, called NewReno TCP, made an extra optimization on the Reno TCP. Inthis version, TCP checks to see if more than one segment is lost in the current window when three duplicate ACKs arrive. When TCP receives three duplicate ACKs, it retransmits the lost segment until a new ACK (not duplicate) arrives. If the new ACK defines the end of the window when the congestion was detected, TCP is certain that only one segment was lost. However, if the ACK number defines a position between the retransmitted segment and the end of the window, it is possible that the segment defined by the ACK is also lost. NewReno TCP retransmits this segment to avoid receiving more and more duplicate ACKs for it.

Additive Increase, Multiplicative Decrease :

Out of the three versions of TCP, the Reno version is most common today. It has been observed that, in this version, most of the time the congestion is detected and taken careof by observing the three duplicate ACKs. Even if there are some time-out events, TCP

recovers from them by aggressive exponential growth. In other words, in a long TCP connection, if we ignore the slow-start states and short exponential growth during fast recovery, the TCP congestion window is cwnd = cwnd + (1 / cwnd) when an ACK arrives (congestion avoidance), and cwnd = cwnd / 2 when congestion is detected, as though SS does not exist and the length of FR is reduced to zero. The first is called additive increase; the second is called multiplicative decrease. This means that the congestion window size, after it passes the initial slow-start state, follows a saw tooth pattern called additive increase, multiplicative decrease (AIMD), as shown in Figure 24.35.

Figure 24.35 *Additive increase, multiplicative decrease (AIMD)*

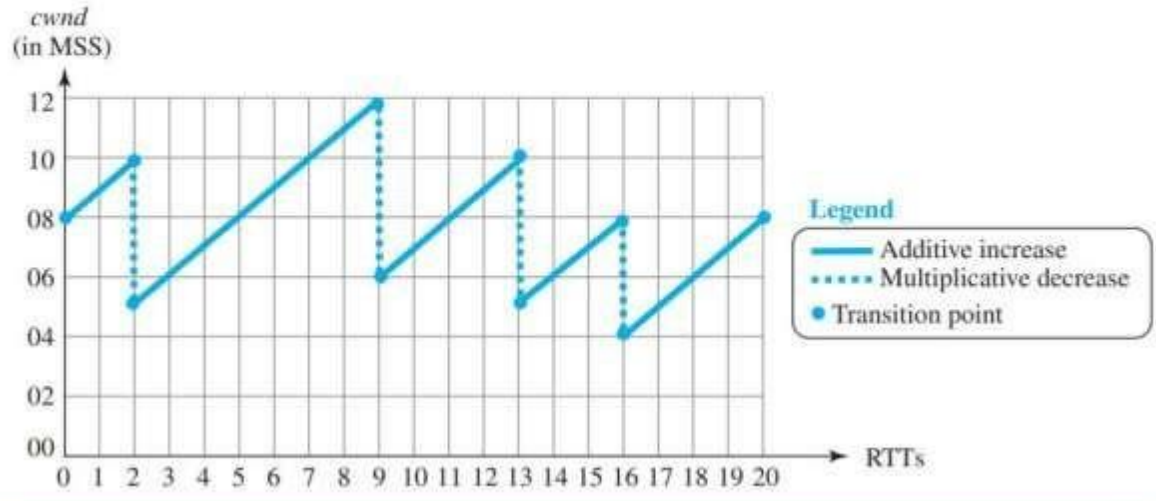


Table 1. The four phases of congestion control algorithm

Phase 1	Slow Start Algorithm
Phase 2	Congestion avoidance Algorithm
Phase 3	Fast retransmission Algorithm
Phase 4	Fast recovery Algorithm

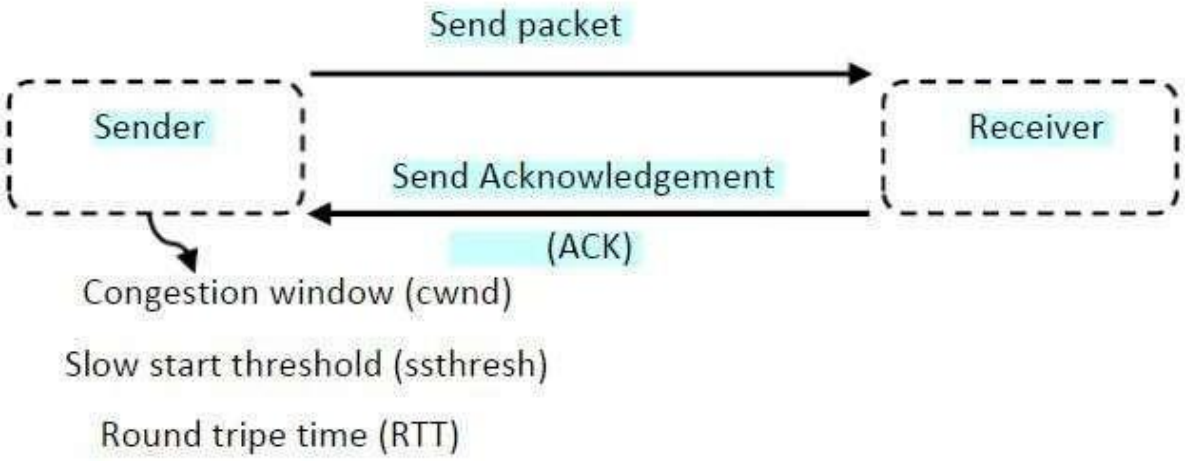


Fig. 1 Common factors used in congestion control

TCP Taho :

TCP Taho uses only the first three algorithm mentioned in Table 1 which are Slow Start, Congestion Avoidance, and Fast Retransmit . To simplify the idea discussed in algorithm steps with and without congestion will be shown here.

A) The procedure without congestion or loss packet will be as following (All terms taken from Fig.1). The procedure start with congestion window of size equal to one packet and any time we have a complete transmission (the source received an acknowledgement for sent packet before timeout status) the congestion window growby one packet.

cwnd<= ssthresh Connection start : cwnd=1 packet.
Each ACK : cwnd+=1 packet.

B) The procedure when we have any one of the following states: the congestion window exceeds the ssthresh, timeout, or receives three duplicate acknowledgements by source.

ssthresh = cwnd/2cwnd =
1
Congestion avoidance
Fast retransmission
Slow start state: when ACK received for retransmitted packet.

TCP Reno :

TCP Reno uses the entire four algorithms for congestion control mentioned in Table 1. The new procedure looks like the one used in TCP Tahoe when respond to timeout. Butfor three duplicate acknowledgements another procedure will be
used which is like following:

ssthresh = cwnd/2.
cwnd = ssthresh + 3. (fast recovery)first
ACK: cwnd +=1.
next ACK cwnd = ssthresh.

The simulation uses two senders and two receivers to test different scenarios of congestion control and Round-trip time RTT. According to the results of simulation, the raise of RTT occurs only when there is a congestion in the network. The raise in RTT property is used to timing the sender with the difference of RTT in current cycle and the previous one. The negative sum of RTT differences means we have no congestion in the current cycle of transmission while the positive sum of RTT differences means the transmission is in congestion state or will have a congestion innext cycles. The
algorithm of TCP Reno will be like the following:

RTT_diff_sum=0;
when the sender recieves a ACK;
RTT_diff = RTT - RIT_last;
RTT_diff_sum + = RTT_diff;
if(dup_ACKs> =3)
(if(RTT_diff_sum >O) Ilin congestion state
{ssthresh=min(cwnd,rcv_window)/2;
cwnd = ssthresh+ 3*MSS; //lower transmission rate
retransmit the lost packet;
} else //in non-congestion state
{ only retransmit the lost packet;

```
}
}
```

TCP Reno is not the best choice to use with wireless network. The new proceduredoesn't provide a way to differentiate between packet loss caused by congestion in network and packet loss when network suffer from random bit error in wirelesslinks.

TCP New Reno :

A very novel TCP version has been proposed. The design provides a new mechanism to differentiate between packets loss caused by high bit error and packetsloss caused by network congestion. Sender, receiver and middle
router all of these parts cooperate to detect congestion and control it.

The new modified Transmission Control Protocol (TCP Reno) is able to monitor the lossof wireless packets in real time. By detecting a router's buffer mechanism in response tocongestion occupancy, the modified Reno is able to monitor wireless packet loss; thus being able to react accordingly and decrease the rate of wireless package loss. This is important because when high volumes of information packets are sent it can have problems reaching the desired recipient. Communication over wireless links, between computer networks and various systems, is filled with random rates of high bit error and connectivity that is intermittent due to the frequency of handoffs.

Mechanisms such as random early detection (RED) were used to find possible problematic packets in their early stages.

The modified TCP Reno uses Explicit Congestion Notification (ECN), which is an extension of REDs. This mechanism can allow the system to properly tell the differencebetween random wireless link errors and errors caused by the congestion of network links. It can also monitor the rate at which wireless packets are lost in a manner that helps the sender select the appropriate segment size at the right moment when packet loss is identified. These modifications to Reno, after plenty of tests and simulations, have shown to have merit and even improve the TCP efficiency; it can do this with no changes in the protocol itself which makes the modified Reno easy enough to use.

Note: In the above lecture, the possible questions are

1. **Explain Reno TCP**
2. **Explain taho TCP**
3. **Explain new Reno TCP**

2. Explain Sliding window Protocol with Go back N and Selective Repeat Protocol

Ans: Please refer IV unit PPT

3. Derive Estimated RTT, Dev RTT and Timeout Interval?

The sample RTT, denoted Sample RTT, for a segment is the amount of time between when the segment is sent (that is, passed to IP) and when an acknowledgment for the segment is received. Instead of measuring a Sample RTT for every transmitted segment, most TCP implementations take only one Sample RTT measurement at a time. Also, TCP never computes a Sample RTT for a segment that has been retransmitted; it only measures Sample RTT for segments that have been transmitted once.

Obviously, the Sample RTT values will fluctuate from segment to segment due to congestion in the routers and to the varying load on the end systems. Because of this fluctuation, any given Sample RTT value may be atypical. In order to estimate a typical RTT, it is therefore natural to take some sort of average of the Sample RTT values. TCP maintains an average, called Estimated RTT, of the Sample RTT values. Upon obtaining a new Sample RTT, TCP updates Estimated RTT according to the following formula:

Estimated RTT = (1 – α) • Estimated RTT + α • Sample RTT

The new value of Estimated RTT is a weighted combination of the previous value of Estimated RTT and the new value for Sample RTT. The recommended value of α is α = 0.125 (that is, 1/8)

Estimated RTT = 0.875 • Estimated RTT + 0.125 • Sample RTT

Note that Estimated RTT is a weighted average of the Sample RTT values. Since the weight of a given Sample RTT decays exponentially fast as the updates proceed it is called as exponential weighted moving average (EWMA). In addition we have, Dev RTT, as an estimate of how much Sample RTT typically deviates from Estimated RTT:

Dev RTT = (1 – β) • Dev RTT + β •|Sample RTT – Estimated RTT|

Note that DevRTT is an EWMA of the difference between SampleRTTand EstimatedRTT. The recommended value of β is 0.25.

Timeout Interval = Estimated RTT + 4 * Dev RTT

Numerical Questions:

4. Suppose that five measured Sample RTT values are 106ms, 120ms, 140ms, 90ms, and 115ms. Compute the Estimated RTT after each of these Sample RTT values is obtained, using a value of alpha = 0.125 and assuming that the value of Estimated RTT was 100ms just before the first of these 5 sample were obtained.

Solutions: (Using above formula with help of values, find out the answer below)

Estimated RTT = 0.10075sec for 106 ms ; Estimated RTT = 0.10315sec for 120ms

Dev RTT = 0.0050625sec for 106 ms ; Dev RTT = 0.0080075sec for 120ms

5. Represent a 3500-byte datagram that has arrived at router R1 and needs to be sent to R2 over a connection with an MTU of 500 bytes. Assume that the size of the IP header is 20 bytes. What is the total number of fragments sent to the destination? The parameters associated with each of these fragments should be shown.

Explanation:

MTU = 100 bytes - 3500
Size of IP header = 20 bytes
So, size of data that can be transmitted in one fragment = 100 – 20 = 80 bytes 3500-20 = 3480

Size of data to be transmitted = Size of datagram – size of header = 1000 – 20 = 980 bytes – 500-20= 480

Now, we have a datagram of size 1000 bytes.
So, we need ceil(980/80) = 13 fragments. = 3480/480= 7.25=8

Thus, there will be 13 fragments of the datagram. – 8.

6. **Consider distributing a file of F=15 Gbit to N peers. The server has an upload rate of Us=30 mbps, and each peer has a download rate of di=2 mbps and an upload rate of u. for N=10&100 & u=300kbps&700kbps, prepare a chart giving the minimum distribution time for each of the combination of N & u for both client-server distribution & P2P distribution.**

Consider the data:

F = 15 Gbits =15360 Mbits (Convert 1 Gbits=1024 Mbits)

s = 30 Mbps

d_i = 2 Mbps

minimum distribution time :

$$D_{cs} = \max\left\{\frac{NF}{u_s}, \frac{F}{d_{\min}}\right\}$$
$$= \max\left\{\frac{10\times 15360}{30}, \frac{15360}{2}\right\}$$
$$= \max\{5120, 7680\}$$
$$= 7680\,\text{sec}$$

So, the minimum distribution time for each of the combinations of N and u for both client-server distribution and P2P distribution=7680 seconds

7. **In GB4, if every 6th packet being transmitted is lost and if we have to spend 10 packets then how many transmissions are required?**

   Refer 4 unit ppt