COMPUTER NETWORKS

HOME WORK - 2

Submitted By:
Manish Choudhary

Answer 1(a):

End to end reliability means performing the reliability operations (ARQ protocol) on the end hosts to ensure the reliable transmission of data from the sender to the receiver. In an alternative implementation, we can also run the ARQ protocol on every router-to-router hop to provide hop-by-hop reliability, thus, freeing the end hosts from this responsibility.
There are advantages and disadvantages of both implementations. Let's consider both together, disadvantage in one means that it would be better in other implementation.

First, the cost of implementing hop-by-hop reliability would be higher as ARQ protocol needs to be implemented on each intermediate node and the required hardware support should be provided to perform reliability checks. Thus, it is hard to perform reliability check on on hop-by-hop basis especially when the end systems don't have any control over the network. On the other side, end-to-end reliability is easier to implement and more tractable.

The hop-by-hop implementation doesn't provide that level of reliability. There are many scenarios in which error may go undetected in hop-by-hop reliability check. For example, in case of a file transfer from one source to destination, errors may occur in reading/writing data from/to disks or in buffering/copying data at hosts or intermediate nodes. A router would receive data from another router, would perform checksum and would keep the data in buffer. But, it may perform some erroneous step while copying data to transfer to the next router. As the checksum would be computed on this new data, this error would go undetected. Even at the receiver end, disk or host I/O buses can introduce bit errors. Thus, integrity check should be performed on the file rather than the packets which can be done at the end systems only.

The primary purpose of the network is the data delivery. Reliability is secondary requirement. Thus, the network should not be loaded with the responsibility of reliability check and it may hinder the primary functionality of the network. Internet is built on the assumption that it requires minimal support from underlying network. Thus, end systems are expected to take care of all the secondary requirements.

Hop-by-hop reliability may not ensure that the data has reached to the destination. Even if it ensures this, it can't guarantee that the host acted on the received data. If we want to make sure that Host is acting on the data (delivering it to the end application), we need end-to-end implementation of ARQ protocol. For example, the window size increases/decreases dynamically as the host delivers the segments to the application in case of end-to-end implementation.

The hop-by-hop implementation doesn't provide fairness to the applications. There can be some services/applications that won't require reliable transmission. In that case, hop-by-hop reliability check would be unnecessary. This would introduce unfairness for the applications which don't want to use reliable medium (they may prefer UDP).

According to Saltzer/Kaashoek applications better understand the requirements and thus, can decide what operations need to be performed. Applications may have different semantic requirements from the underlying system and thus, enforcing hop by hop reliability may not be feasible.

End-to-end implementation would be better to provide data authenticity, integrity and secrecy. The encryption /decryption should be performed only by the end applications as network can't be trusted and thus, the key management should be performed by the end systems only. The data should not be exposed to the intermediate nodes.

Hop-by-hop implementation provides better performance, higher throughput and lesser latency. Retransmissions and reliability checks need to be performed between two adjacent nodes and acknowledgement is immediately sent to the previous node. In end-to-end implementation end host is responsible to probe the network for estimating the congestion. The implemented scheme doesn't guarantee congestion free network and packets may get dropped and hence, packets have to travel the whole path again. In hop-by-hop each two adjacent nodes can handle this problem and thus, would result in less congestion. Moreover, the packet needs to retransmitted only between these two nodes.

In case of end-to-end implementation, the end hosts have to be really intelligent so that they can implement and handle the whole reliability check. In case of hop-by-hop end hosts can be expected to be dumb as the intermediate nodes will take care of this.

Today, Internet does only end-to-end reliability because it makes more sense to let the network perform basic functionality and let the end systems/applications do the remaining. As discussed above, there are many requirements and problems in implementing the hop-by-hop reliability. Thus, end-to-end reliability is preferred.

Answer 1(b):

"Echo Checking" is a quality check and error-control technique for data transferred over a network or communications link, in which the received data is stored and also transmitted back to the point of origin, where it is compared with the original data. Thus, in echo checking the data sent is echoed (returned) to the sending end of a transmission.

Echo can be either local echo, where the sending device itself displays the sent data, or remote echo, where the receiving device returns the received data to the sender. The latter can be used as a form of error detection to determine whether the data received at the remote end is the same as data sent or not. Thus, any difference in the data sent and the echoed data would result is error detection and therefore, can be used for reliable data transfer.

For example, terminals are an example where echo check is used for reliable data transmission. As discussed above, echo can be done locally or remotely. In local mode, a character is transmitted to the remote computer and simultaneously printed or displayed on the terminal screen when a key is pressed. In remote mode, when a key is pressed, the character is transmitted to the remote computer but is not displayed on the terminal screen. Instead, the character received by the remote computer is retransmitted or echoed back to the terminal, which is then displayed on the screen. Thus, echo checking is used for error detection in case of remote echo. If the character which is displayed on the screen differs from the actual key pressed on the keyboard, an error must have occurred during the transmission and thus, sender should retransmit that character. Therefore, echo check ensures reliability.

Another example is TELNET protocol, which provides a standardized interface through which a program on one host (TELNET client) may access the resources of another host (TELNET server). Here also, echo checking is used for reliable data transfer.

The two modems communicating with each other also use echo checking. The remote modem echoes whatever it receives from the local modem.

Instead of echo checking in which entire data is sent twice, better alternatives can be used for achieving reliability. These schemes are based on the concept of returning only a small control message or frame to acknowledge correct receipt (or otherwise) of each transmitted frame, rather than retransmission of a complete copy of each transmitted frame. This improves the performance of the system substantially. ARQ protocols are acknowledgement based schemes which are much better and faster as compared to the echo checking mechanism.

In addition, the echoed data (even if echoed the correct data) may get corrupted on the way and may result in the sender retransmitting the data again to the receiver unnecessarily.

References:
1. http://en.wikipedia.org/wiki/Echo_%28computing%29
2. http://www.cs.ucv.ro/staff/dmancas/CD/en/Cap4.rtf
3. IBM (June 1995) - Telnet and the Telnet Protocol

Answer 1(c):

A pair of TCP devices willing to establish a connection needs to go through TCP three-way handshake. During handshake, the TCP layers on the devices must exchange information about the sequence numbers each device wants to use for its first data transmission, as well as parameters that will control how the connection operates. The sequence number synchronization is carried out by sharing SYN (synchronization messages).

TCP has to synchronize the sequence numbers in the beginning of the connection because TCP is a byte stream protocol and it refers to each byte of data individually. Thus, it needs sequence numbers to keep track of which bytes have been sent and received. Since each byte has a sequence number, we can acknowledge each byte, or more efficiently, use a single number to acknowledge a range of bytes received. It helps to keep track of out-of-sequence segments (because of delays caused by the network) and also helps in deciding which segments need to be retransmitted.

We can't start all the connections counting from zero as it introduces the possibility of segments from different connections getting mixed up. Suppose two devices established a TCP connection and one device sent a segment containing bytes 0 through 512 to the other. Let's assume that the segment got delayed because of some problem and eventually, the TCP connection got terminated. Then, the two devices again started a new connection and again used the zero as the starting sequence number.

Now, as soon as the connection was started, the old segment with bytes 0 to 512 appeared. The receiving device would erroneously think these bytes were part of the new connection. Thus, each communicating host has to choose a 32-bit initial sequence number (ISN) for the connection to avoid such problem. The ISN is usually a random number which is generally different for each host.

In addition to the above problem, another reason for choosing a random ISN is to try to avoid TCP sequence prediction attack. If each connection starts with fixed (say zero) sequence number and the attacker knows the same, it would be really easy for the adversary to attack. Attacker can send counterfeit packets to the receiving host which will seem to originate from the sending host, even though these packets may in fact originate from some third host controlled by the attacker. Thus, attacker would hijack the connection and would effectively communicate with the receiver impersonating as the sender. In this way, attacker may get any information from the receiver or force it to perform some action. After this, attacker may terminate the connection using TCP reset attack.

References:

1. http://www.tcpipguide.com/free/t_TCPConnectionEstablishmentSequenceNumberSynchroniz.htm
2. http://en.wikipedia.org/wiki/TCP_sequence_prediction_attack

<u>**Answer 1(d):**</u>

TCP Reno is the most common used flavour which uses both “Fast Retransmit” and “Fast Recovery” algorithms.

Here, a receiver should send an immediate a duplicate ACK when it gets an out-of-order segment. This duplicate ACK informs the sender that an out-of-order segment was received. From the sender's perspective, this duplicate ACK may be caused by several network problems like dropped segments, re-ordering of data segments by the network or replication of ACK.

When the sender receives first duplicate ACK, it enters into fast retransmit mode in which it waits for 2 more duplicate ACKs to receive before retransmitting the lost packet. “Fast Retransmit” algorithm is used to detect and repair loss based on the incoming duplicate ACKs. Thus, the fast retransmit algorithm uses the arrival of 3 duplicate ACKs as an indication that a segment has been lost. After receiving 3 duplicate ACKs, TCP performs a retransmission of what appears to be the missing segment, without waiting for the retransmission timer to expire.

After fast retransmit mode, “fast recovery” algorithm is run which governs the transmission of new data until non-duplicate ACK arrives. It does the fast recovery (and not the slow start) because the receipt of the duplicate ACKs indicates that the segments are most likely leaving the network and are in the receiver’s buffer; and thus, not consuming the network resources. This means that TCP sender can continue to transmit new segments until non-duplicate ACK is received.

The fast retransmit and fast recovery algorithms are usually implemented together as follows.

1. On receiving the first duplicate ACK, it enters into fast retransmit mode.

2. When the third duplicate ACK is received, TCP sets ssthresh to half of current window. The missing segment is retransmitted by the sender.

3. The recovery mode starts in which cwnd is set to (ssthresh + 3*SMSS). This artificially "inflates" the congestion window by the number of segments (three) that have left the network and which the receiver has buffered.

4. For each additional duplicate ACK received, cwnd is incremented by SMSS. This artificially inflates the congestion window in order to reflect the additional segment that has left the network.

5. The previously unsent data can be sent to the receiver if the window allows.

6. When non-duplicate ACK arrives at the sender, TCP sets cwnd to ssthresh. This is deflating window and this marks the end of Fast recovery mode.

Fast Recovery as the name suggests is done to recover fast from the loss of the segment by increasing the cwnd before the reception of a non-duplicate acknowledgement. This is based on the logic that each duplicate ACK indicates that a segment has left the network and thus, the window size can be increased, which allows to send new data.

References:

1. TCP Congestion Control: RFC 5681

Answer 2 (a):

Choke packet is a control packet generated at a congested node and transmitted back to a source node to restrict traffic flow. Either a router or a destination end system may send this message to a source end system, requesting that it reduces the rate at which it is sending the traffic. This forces the node or transmitter to reduce its rate at which data is sent.

Thus, choke packets are used for congestion and flow control over a network. The source node is addressed directly by the router using the IP address; forcing it to decrease its sending rate. The source node acknowledges this by reducing the sending rate by some percentage.

An Internet Control Message Protocol (ICMP) source quench packet is a type of choke packet normally used by routers.

Algorithm for Congestion Control using choke packets:

1. Sender establishes a connection with the receiver and both agree upon transmission rate at which the data will be sent.

2. Sender starts sending data to the receiver through the network at the rate they have agreed upon.

3. In the network, each router monitors its resources and the utilization at each of its output line. A threshold value is decided, above which the line would be considered congested. Thus, whenever the resource utilization crosses this threshold, the corresponding output line enters in a warning state indicating that it is congested.

4. For each incoming data packet, the state of the corresponding output line is checked. If it is in warning state, a choke packet is sent to the source using the IP address, giving it a feedback to reduce the traffic. This source quench packet tells the sender that congestion has occurred and that it needs to reduce its rate at which it is sending the data.

5. The data packet received at the router, is tagged (by setting a bit in its header) so that it will not generate any more choke packets at downstream router, and is then forwarded in the usual way.

6. When the source receives the choke packet, it gets to know that the network has got congested and thus, it should reduce the traffic rate to the specified destination by fraction (for example, it may reduce the data rate by 50%).

7. After getting the choke packet and reducing the rate, source starts a timer during which it ignores the choke packets referring to that destination. Once this time period expires, if another choke packet arrives, the host reduces the data rate again, and so on.

8. After the timer expires, the sender would start increasing its data rate linearly (for example, by 1 for every ACK received) until further congestion occurs.

References:
1. Routing and Congestion Control: Version 2 CSE IIT Kharagpur

Answer 2(b):

Advantages:

1. In this scheme, congestion control is governed by the network and not by the sender. Thus, this scheme is dynamic. Source sends as much data as it wants and it is network's duty to detect the congestion and inform the sender about network congestion asking it to slow down. Thus, sender doesn't need to probe the network at the start of the connection to guess the appropriate data rate. It can start to send the data at the rate at which it has agreed upon with the receiver.

Disadvantages:

1. This scheme is not practical to implement especially in the case of multiple senders. If there is large number of senders, same number of source quenches has to be sent, which would result in more congestion and may cause congestion for other routers in the network.

2. It is difficult to estimate an appropriate value by which the sender should slow down on receiving the choke packet.

3. It is unfair as we are asking all the senders to reduce their data rates irrespective of their initial data rate. It is also difficult to identify the sender responsible for the congestion. For example, suppose, there are three senders A, B and C, sending data at rates 850 Mbps, 5 Mbps and 3 Mbps respectively and are competing for a link of 100 Mbps. It would result in congestion and thus, the router would send choke packets to each of the senders asking to reduce the rate by half. Thus, the rates become 425 Mbps, 2.5 Mbps and 1.5 Mbps. The network would still be congested and it is not fair to ask B and C to reduce the rates as they are not responsible for the congestion.

4. The scheme also results in a problem called "Unquenching Problem", where there is no way to inform the sender that congestion is over and it can increase the data rate. The sender is responsible for increasing the data rate.

Answer 3 (a):

Max. Window Size W = 32 segments
RTT = 100 msec = 0.1 sec
Loss is identified by three duplicate ACK only.
Range for loss Rate = 0.5% to 5%
SMSS = 512 Bytes

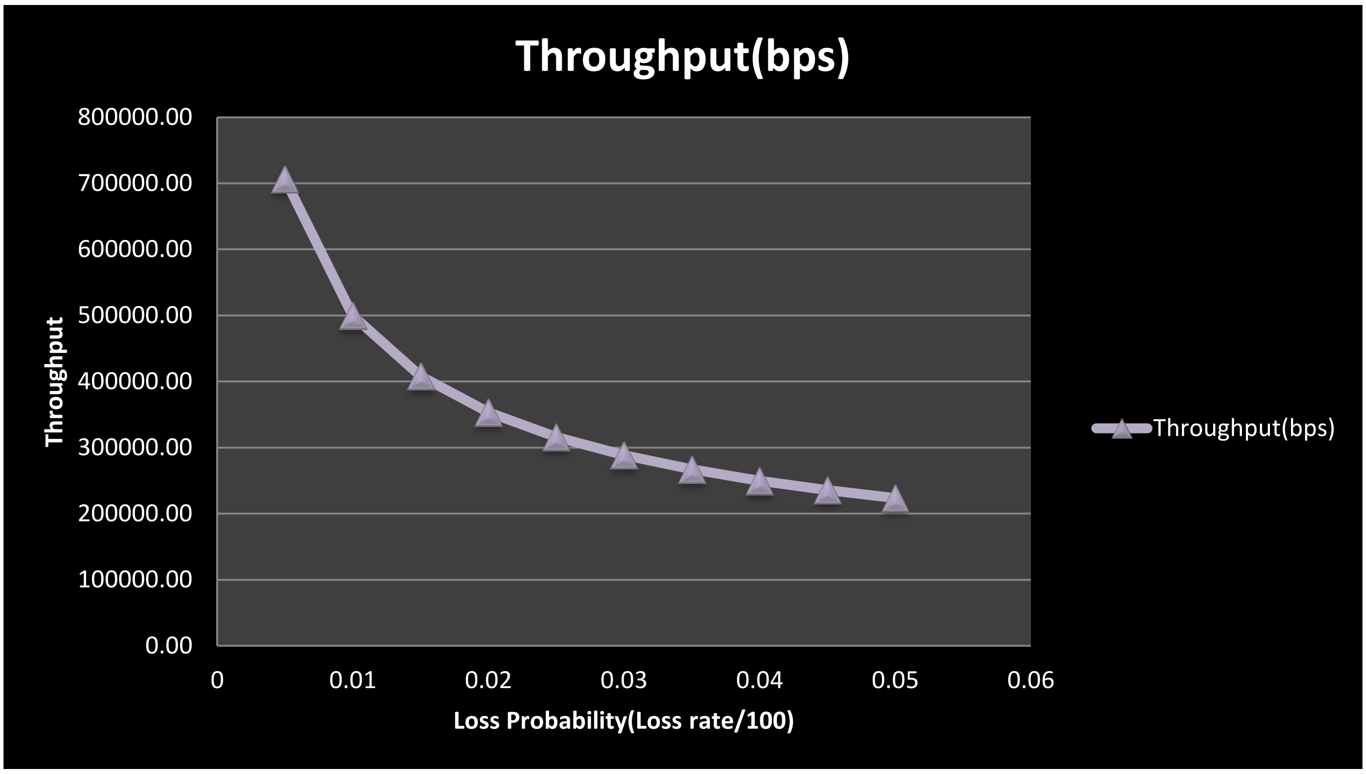
As the probability rate is from 0.5% to 5%, the loss probability would range from 0.005 to 0.05.
The throughput is first calculated using the formula (as each packet is acknowledged by the receiver):

Throughput = 1.22/ (RTT *sqrt (Loss Probability))

Here, we got the throughout in segments per second. So, it is multiplied with SMSS*8 to convert it into bits per second. The results are shown below in tabular form:

Loss Probability(P)	SQRT(P)	Throughput(seg/sec)	Throughput(bps)
0.005	0.070710678	172.5340546	706699.49
0.01	0.1	122	499712
0.015	0.122474487	99.61258287	408013.14
0.02	0.141421356	86.2670273	353349.74
0.025	0.158113883	77.15957491	316045.62
0.03	0.173205081	70.43673284	288508.86
0.035	0.187082869	65.21174303	267107.30
0.04	0.2	61	249856
0.045	0.212132034	57.51135154	235566.50
0.05	0.223606798	54.56005865	223478.00

Below is the graph plotted for the throughput in bits per second for the loss probability ranging from 0.005 to 0.05.


Throughput(bps)
800000.00
700000.00
600000.00
500000.00
400000.00
300000.00
200000.00
100000.00
0.00
Throughput
0
0.01
0.02
0.03
0.04
0.05
0.06
Loss Probability(Loss rate/100)
Throughput(bps)

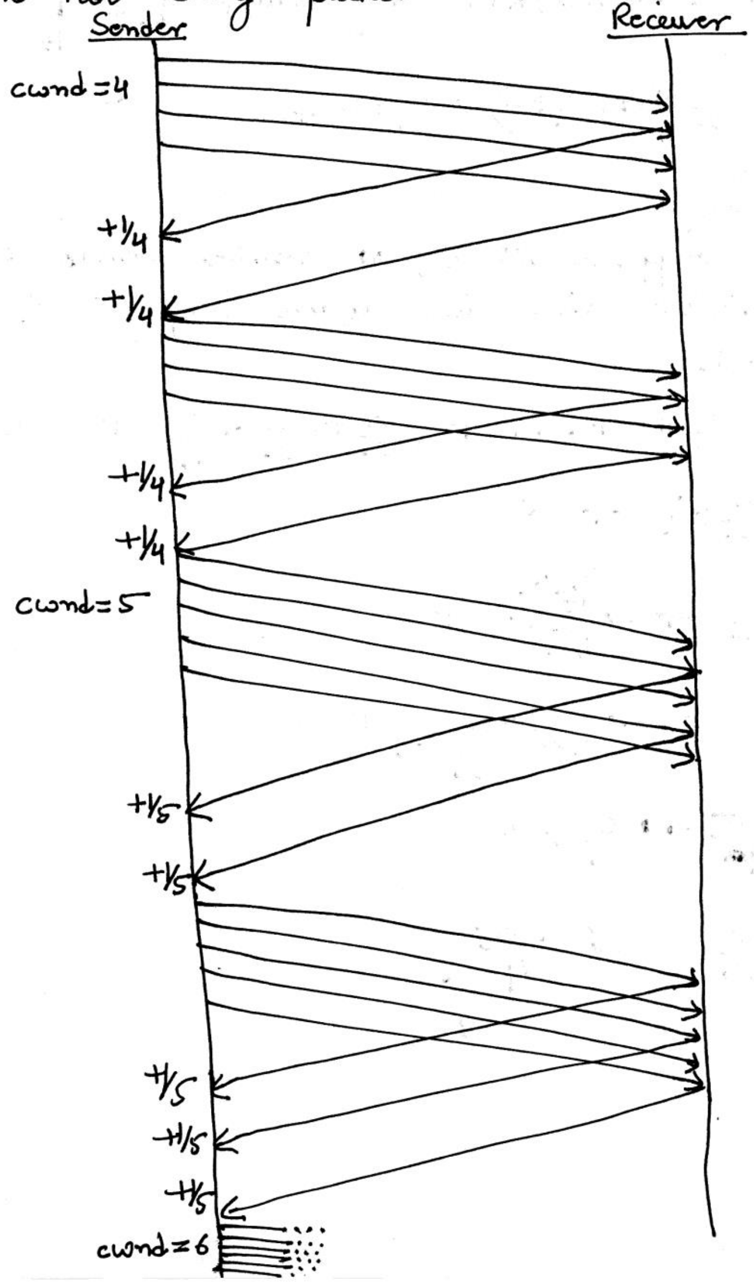
Answer 3(b):

Answer 3(b)

The TCP throughput formula derived in the class assumes that the receiver acknowledges every packet received.
The formula that we derived was :-

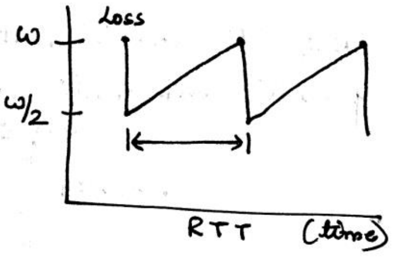
$$\boxed{\text{Throughput} = \frac{1.22}{RTT\sqrt{p}} \text{ segments/sec}}$$

Now, lets consider that the TCP receiver acknowledges every other packet and not every packet.



So, the increment in collision avoidance will happen something similar as shown in above diagram.
Thus, instead of increasing 1 segment/RTT, we are actually increasing effectively 1 segment for every 2 RTTs.
Lets say that the max. ~~segment~~ window size is W. At TDA loss, the window size becomes $\frac{W}{2}$.
As 1 segment is getting increased after every 2 RTTs, it would take (W) rounds to reach to the maximum window size

Lets calculate no. of segments in one tooth,
It would send the segments/RTT in following manner :-



$$\frac{W}{2}+\frac{W}{2}+\left(\frac{W}{2}+1\right)+\left(\frac{W}{2}+1\right)+\left(\frac{W}{2}+2\right)+\left(\frac{W}{2}+2\right)+\dots$$

$$\dots\dots+\underbrace{\left(\frac{W}{2}+\frac{W}{2}\right)}$$

Assuming that loss happens at W, as sender sends W segments, (as shown above) Window size will again drop to W/2.
So, # segments/tooth

$$= 2\left(\frac{W}{2}+\left(\frac{W}{2}+1\right)+\dots+\left(\frac{W}{2}+\left(\frac{W}{2}-1\right)\right)\right)+\left(\frac{W}{2}+\frac{W}{2}\right)$$

$$= 2\cdot\sum_{i=0}^{\frac{W}{2}-1}\left(\frac{W}{2}+i\right)+W$$

Total W rounds.

$$= 2\cdot\sum_{i=0}^{\frac{W}{2}-1}\left(\frac{W}{2}\right)+2\cdot\sum_{i=0}^{W/2-1}(i)+W$$

$$=2\cdot\frac{W}{2}\cdot\frac{W}{2}+2\cdot\frac{\left(\frac{W}{2}-1\right)\left(\frac{W}{2}\right)}{2}+W$$

$$=\frac{W^2}{2}+\frac{W^2}{4}-\frac{W}{2}+W$$

$$=\frac{3W^2}{4}+\frac{W}{2}\approx\frac{3W^2}{4}.$$

Now, lets calculate the loss probability :-

$$P = \frac{1 \text{ segment}}{(\text{\# of segments per tooth})}$$

$$= \frac{1}{3W^2/4}$$

$$= \frac{4}{3W^2}$$

or $\boxed{W = \sqrt{\frac{4}{3P}}}$

So, throughput can be calculated as follows :-

$$\text{Throughput} = \frac{\text{\# of segments in one tooth}}{\text{Time of tooth}}$$

$$= \frac{3W^2/4}{W \times RTT}$$

$$= \frac{3W}{4 \cdot RTT}$$

$$= \frac{3 \times \sqrt{\frac{4}{3P}}}{4 \cdot RTT}$$

$$= \frac{\cancel{1.22}\sqrt{3}}{2 \cdot RTT \cdot \sqrt{P}} \text{ segments/sec.}$$

OR $\boxed{\text{Throughput} = \frac{0.866}{RTT\sqrt{P}} \text{ seg/sec}}$

Answer 4(a):

Suggested attack:

Upon receiving a data segment, the receiver sends a long stream of acknowledgments for the last sequence number received (at the start of a connection this would be for the SYN segment).

Advantage available for the attacker:

The attack is possible because of the third and fourth steps of the fast recovery process which can cause the sender to artificially inflate the window giving the misbehaving receiver an advantage.

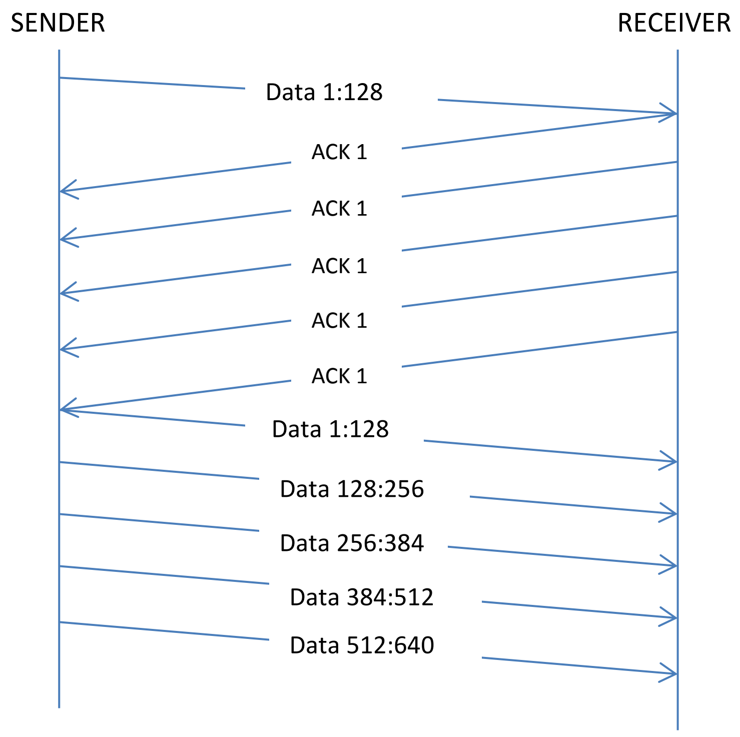
The involved steps are:

*3. The lost segment starting at SND.UNA MUST be retransmitted and cwnd set to ssthresh plus 3*SMSS. This artificially "inflates" the congestion window by the number of segments (three) that have left the network and which the receiver has buffered.*

4. for each additional duplicate ACK received (after the third), cwnd MUST be incremented by SMSS. This artificially inflates the congestion window in order to reflect the additional segment that has left the network.

Example and Explanation:

The attack is possible by a misbehaving receiver because TCP requires that duplicate ACKs be exact duplicates and thus, there is no way to ascertain which data segment they were sent in response to. Therefore, it is impossible to differentiate a “valid” duplicate ACK, from a forged, or “spoofed”, duplicate ACK. Because of this reason, the sender cannot distinguish ACKs that are accidentally duplicated by the network itself from those generated by a receiver. Thus, the receiver can use long stream of duplicate ACKs to force the sender to transmit new segments into the network.



Let's consider an example for the attack, sender sends a data segment of 128 bytes to the receiver. Receiver even on receiving the data segment may pretend as if it didn't receive the segment and thus, would send ACK 1 to the sender. This would be an indication for sender that the next expected byte is 1. Receiver would send another ACK 1 to the sender who would enter into the fast retransmit mode on receiving this first duplicate ACK. Receiver would keep sending the duplicate ACKs. Since duplicate ACKs are indistinguishable, the receiver does not need to wait for new data to send additional acknowledgments.

The first four ACKs for the same sequence number would cause the sender to retransmit the first segment. However, cwnd is now set to its initial value plus 3*SMSS, and increased by SMSS for each additional duplicate ACK, for a total of 4 segments (as per the fast recovery algorithm). As a result, the sender will return data at a rate directly proportional to the rate at which the receiver sends acknowledgments. Just before the timeout, the receiver would acknowledge the missing segment and would enter the fast retransmit again for a new segment, later. In this way, a misbehaving receiver can force the sender to artificially inflate the window and to send more data at a rate higher than appropriate.

Answer 4(b):

RFC 5681 suggests following implementation to mitigate this:

“A TCP may limit the number of times cwnd is artificially inflated during loss recovery to the number of outstanding segments (or, an approximation thereof).”

This is a sender-only heuristic in which sender can maintain a count of the outstanding segments sent above the missing segment. For each duplicate acknowledgment received, this count is decremented and when it reaches zero any additional duplicate acknowledgments received by the sender are ignored. This fix tries to limit the number of segments wrongly sent to contain no more than cwnd – SMSS bytes.

This fix won't work as a receiver can acknowledge the missing segment as the count reaches to zero and then can repeat the same process indefinitely. Thus, receiver is again able to exploit the weakness in the implementation.

References:

1. Savage et. al., TCP congestion control with Misbehaving Receiver.
2. TCP Congestion Control: RFC 5681