NAME : RADHIKA PATWARI
ROLL NO. : 18CS10062

DATE : 13/3/2021
Computer Networks[CS31006] Class test 3
==========

1.
TCP acknowledgments are cumulative and correctly received but out-of-order segments are not individually ACKed by the receiver. TCP sender need only maintain the smallest sequence number of a transmitted but unacknowledged byte (base) and the sequence number of the next byte to be sent (nextseqnum). In this way, TCP looks a lot like a GBN-style protocol.

But there are some differences between TCP and Go-Back-N. TCP accepts out-of-order segments but resends only one missing segment only when it timeouts. Consider when the sender sends a sequence of segments 1, 2, . . . , N, and all of the segments arrive in order without error at the receiver. Further suppose that the acknowledgment for packet n < N gets lost, but the remaining n–1 acknowledgments arrive at the sender before their respective timeouts. In this example, GBN would retransmit not only packet n, but also all of the subsequent packets n + 1, n + 2,. . . , N. TCP, on the other hand, would retransmit at most one segment, here, segment n. Moreover, TCP would not even retransmit segment n if the acknowledgment for segment n + 1 arrived before the timeout for segment n.

2.
During a three-way handshake for establishing TCP connection, the client sends a SYN segment, then the server sends a SYN ACK in response, and awaits an ACK segment from the client. At the same time, the server allocates resources(TCP buffer space, window size parameters..etc) to provide a half-open connection to this client. If the client does not send an ACK to complete the third step of this 3-way handshake, eventually (often after a minute or more) the server will terminate the half-open connection and reclaim the allocated resources.
This leads to an attack called SYN flood attack. In this attack, the attacker(s) send a large number of TCP SYN segments, without completing the third handshake step. With this deluge of SYN segments, the server’s connection

resources become exhausted as they are allocated (but never used) for half-open connections; legitimate clients are then denied service.

Solution : Using SYN cookies
When the server receives a SYN segment, it does not know if the segment is coming from a legitimate user or is part of a SYN flood attack. So, instead of creating a half-open TCP connection for this SYN, the server creates an initial TCP sequence number that is a complicated function (hash function) of source and destination IP addresses and port numbers of the SYN segment, as well as a secret number only known to the server. This initial sequence number is called cookie.
The server then sends the client a SYNACK packet with this special initial sequence number and forgets the cookie or any other state information corresponding to the SYN.
A legitimate client will return an ACK segment. When the server receives this ACK, it verifies the ACK using cookies. The server runs the same hash function using the source and destination IP address and port numbers in the ACK (whose values are the same as in SYNACK as the client is the same) and the secret number. If the result of the function plus one is the same as the acknowledgment (cookie) value in the client's SYNACK, the server concludes that the ACK corresponds to an earlier SYN segment and is hence valid. The server then creates a fully open connection along with a socket.
On the other hand, if the client does not return an ACK segment, then the original SYN has done no harm to the server, since the server hasn't yet allocated any resources in response to the original SYN.

3.
Given delay = 50 ms and bandwidth = 2 Gbps
Bandwidth delay product = delay x bandwidth = 50ms x 2Gbps = 10^8 bits = 125x10^5 bytes

16-bit sequence implies bytes range from 0 to 2^16-1 (65535) bytes : total 65536 bytes at any point of time

Thus , for sliding window protocol, the maximum window size can be of maximum size ~ 2^16

But the link can hold upto 125X10^5 bytes at any point of time. Hence it is safe to use 16-bit sequence number field for sliding window based flow control algorithms.

4.
When sending tcp sends data in large blocks, the receiver buffer may get filled. When the receiving application on the receiver tcp service reads data very slowly (as worse as one byte per second) from the receiver buffer, the buffer slowly becomes empty. Now the receiver tcp service may send window updates piggyback on acknowledgement segments indicating 1 byte is empty at the receiver buffer, so the sender can now send data. In this way, as the buffer becomes empty, window updates per byte are sent. (by the time the receiver buffer becomes 4 byte empty, nearly 4 window updates are sent as the application is reading data slowly). But each segment was at least 20 bytes size(no data, but contained tcp header). Thus there is a wastage of bandwidth (4x20 bytes sent for just 4 bytes update).

Clark's algorithm states that instead of sending window updates for every byte, updates should be sent only when there is sufficient space on the receiver buffer (at least half the window size). This saves overall bandwidth.

5.
Roll no : 18CS10062 (C=3,S=19)

Parameter alpha = (1+8+3+19+1+0+0+6+2)mod 5/10 + 1/2 = 0/10 + 1/2 = ½
Parameter beta = alpha/2 = 1/4

SRTT_initial = 0ms
RTT_VAR_initial = 0ms
RTT_1 = 1100 ms
RTT_2 = 1500 ms

After receiving acknowledgement for 1st TCP segment :
SRTT_1 = alpha x SRTT_initial + (1 - alpha) x RTT_1
=> SRTT_1 = ½ x 0 + (1-½)x1100 = 1100/2 = 550 ms

RTT_VAR_1 = beta x RTT_VAR_initial + (1 - beta) x abs (SRTT_1 - RTT_1)
=> RTT_VAR_1 = ¼ x 0 + (1-¼)x | 550 - 1100 | = 825/2 = 412.5 ms

RTO = max(SRTT_1 + 4 X RTT_VAR_1 , 1 sec) = max(2200 , 1000) = 2200 ms

After receiving acknowledgement for 2nd TCP segment :

SRTT_2 = alpha x SRTT_1 + (1 - alpha) x RTT_2
=> SRTT_2 = ½ x 550 + (1-½)x1500 = 1100/2 = 1025 ms

RTT_VAR_2 = beta x RTT_VAR_1 + (1 - beta) x abs (SRTT_2 - RTT_2)
=> RTT_VAR_2 = ¼ x 825/2 + (1-¼)x | 1025 - 1500 | = 3675/8 ms

RTO = max(SRTT_2 + 4 X RTT_VAR_2 , 1 sec) = max(2862.5,1000) = 2862.5 ms ~ 2863 ms

Thus , RTO after acknowledgement of 2nd TCP segment = 2862.5 ms ~ 2863 ms

6.

a)False;
Ordinary implementations of TCP use Duplicate Acknowledgement(DUPACK) to indicate the presence of missing segments.

b) False;
MSS is calculated during a three-way handshake by TCP using path MTU discovery during the start of connection establishment.

c) False;
During the Slow Start phase, the CWnd parameter in TCP Tahoe increases exponentially

d) True;
Sender side is waiting for acknowledgements and buffering data due to nagle's algorithm and the receiver side is waiting for 500 ms for receiving more data due to delayed acknowledgement. Even after 500 ms , the acknowledgement sent by the receiver can get lost.-- starvation on both sides

e) False;
URG flag forces the out-of-order delivery of segments to receiver and to receiving application even if receiver window size is 0.In PSH flag, receiver will deliver packets to receiving application in correct order, hence this is less powerful.