1. Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 248. Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 40 and 50 bytes of data, respectively. In the first segment, the sequence number is 249, the source port number is 503, and the destination port number is 80. Host B sends an acknowledgement whenever it receives a segment from Host A.

   (a) In the second segment from Host A to Host B, what are the sequence number, source port number, and destination port number? In the second segment from Host A to B, the sequence number is 289, source port number is 503 and destination port number is 80.

   (b) If the first segment arrives before the second segment, in the acknowledgment of the first arriving segment, what is the acknowledgement number, the source port number, and the destination port number?
   If the first segment arrives before the second, in the acknowledgement of the first arriving segment, the acknowledgement number is 289, the source port number is 80 and the destination port number is 503.

   (c) If the second segment arrives before the first segment, in the acknowledgement of the first arriving segment, what is the acknowledgement number?
   If the second segment arrives before the first segment, in the acknowledgement of the first arriving segment, the acknowledgement number is 249, indicating that it is still waiting for bytes 249 and onwards.

   (d) Suppose the two segments sent by A arrive in order at B. The first acknowledgement is lost and the second acknowledgement arrives after the first timeout interval, as shown in the diagram on the next page. Draw a timing diagram, showing these segments and all other segments and acknowledgements send. (Assume there is no additional packet loss.) For each segment in you figure, provide the sequence number and the number of bytes of data; for each acknowledgement that you add, provide the acknowledgement number.
   See diagram:

2. Consider SYN cookies.

   (a) Why is it necessary for the server to use a special initial sequence number in the SYNACK?
   The server uses special initial sequence number (that is obtained from the hash of source and destination IPs and ports) in order to defend itself against SYN FLOOD attack.

   (b) Suppose that an attacker knows that a target Host uses SYN cookies. Can the attacker create half-open or fully open connections by simply sending an ACK packet to the target? Why or why not?
   No, the attacker cannot create half-open or fully open connections by simply sending and ACK packet to the target. Half-open connections are not possible since a server using SYN cookies does not maintain connection variables and buffers for any connection before full connections are established. For establishing fully open

connections, an attacker should know the special initial sequence number corresponding to the (spoofed) source IP address from the attacker. This sequence number requires the "secret" number that each server uses. Since the attacker does not know this secret number, she cannot guess the initial sequence number.

3. Consider the TCP procedure for estimating RTT. Suppose that $\alpha = 0.15$. Let `SampleRTT1` be the most recent sample RTT, let `SampleRTT2` be the next most recent sample RTT, and so on.

   (a) For a given TCP connection, suppose 4 acknowledgements have been returned with corresponding sample RTTs `SampleRTT4`, `SampleRTT3`, `SampleRTT2`, and `SampleRTT1`. Express `EstimatedRTT` in terms of the four sample RTTs.

$$\begin{aligned}
\texttt{EstimatedRTT}^{(1)} &= \texttt{SampleRTT1} \\
\texttt{EstimatedRTT}^{(2)} &= x\texttt{SampleRTT1} + (1-x)\texttt{SampleRTT2} \\
\texttt{EstimatedRTT}^{(3)} &= x\texttt{SampleRTT1} + (1-x)\left[x\texttt{SampleRTT2} + (1-x)\texttt{SampleRTT3}\right] \\
&= x\texttt{SampleRTT1} + (1-x)x\texttt{SampleRTT2} + (1-x)^2\texttt{SampleRTT3} \\
\texttt{EstimatedRTT}^{(3)} &= x\texttt{SampleRTT1} + (1-x)x\texttt{SampleRTT2} + (1-x)^2x\texttt{SampleRTT3} \\
&\quad +(1-x)^3\texttt{SampleRTT4}
\end{aligned}$$

   (b) Generalize your formula for n sample RTTs.

$$\texttt{EstimatedRTT}^{(n)} = x\sum_{j=1}^{n-1}(1-x)^{j-1}\texttt{SampleRTTj} + (1-x)^{n-1}\texttt{SampleRTTn}$$

   (c) For the formula you just derived above let n approach infinity. Comment on why this averaging procedure is called an exponential moving average.

$$\texttt{EstimatedRTT}^{(\infty)} = \frac{x}{1-x}\sum_{j=1}^{\infty}(1-x)^{j}\texttt{SampleRTTj}$$

   The weight given to past samples decays exponentially.

4. Why do you think TCP avoids measuring the `SampleRTT` for retransmitted segments? Let's look at what could wrong if TCP measures `SampleRTT` for a retransmitted segment. Suppose the source sends packet P1, the timer for P1 expires, and the source then sends P2, a new copy of the same packet. Further suppose the source measures `SampleRTT` for P2 (the retransmitted packet). Finally suppose that shortly after transmitting P2 an acknowledgment for P1 arrives. The source will mistakenly take this acknowledgment as an acknowledgment for P2 and calculate an incorrect value of `SampleRTT`.

5. After a timeout event, the timeout period is doubled. This is a form of congestion control. Why does TCP need a window based congestion-control mechanism in addition to this doubling-timeout-interval mechanism?
   If TCP were a stop-and-wait protocol, then the doubling of the time out interval would suffice as a congestion control mechanism. However, TCP uses pipelining (and is therefore not a stop-and-wait protocol), which allows the sender to have multiple outstanding unacknowledged segments. The doubling of the timeout interval does not

prevent a TCP sender from sending a large number of first-time-transmitted packets into the network, even when the end-to-end path is highly congested. Therefore a congestion-control mechanism is needed to stem the flow of "data received from the application above" when there are signs of network congestion.

6. Recall the macroscopic description of TCP throughput. In the period of time from when the connection's rate varies from $W/(2 \cdot \text{RTT})$ to W/RTT, only one packet is lost (at the very end of the period).

   (a) Show that the loss rate (fraction of packets lost) is

   $$\frac{1}{\frac{3}{8}W^2 + \frac{3}{4}W}$$

   $$(W/2) + ((W/2) + 1) + \ldots + W = \sum_i = 0^{W/2} W/2 + i = \frac{1}{\frac{3}{8}W^2 + \frac{3}{4}W}$$

   (b) Use the result above to show that if an connection has loss rate L, then its average rate is approximately given by

   $$\approx \frac{1.22MSS}{RTT\sqrt{L}}$$

   For W large,$3/8W^2 >> 3/4W$. Thus $L \approx 8/(3W^2)$ or $W = \sqrt{8/(3L)}$. From the text, we therefore have 3/4 MSS/RTT $\sqrt{8/(3L)} = \approx \frac{1.22MSS}{RTT\sqrt{L}}$

7. In this problem we investigate whether either UDP of TCP provides a degree of end-point authentication.

   (a) Consider a server that receives a request within a UDP packet and responds to that request within a UDP packet (for example, as done by a DNS server). If a client with IP address X spoofs its address with address Y, where will the server send its response?
   The server will send its response to Y.

   (b) Suppose a server receives a SYN with IP source address Y, and after responding with a SYNACK, receives an ACK with IP source address Y with the correct acknowledgement number. Assuming the server chooses a random initial sequence number and there is no "man in the middle," can the server be certain that the client is indeed at Y (and not at some other address X that is spoofing Y)?

   The server can be certain that the client is indeed at Y. If it were at some other address spoofing Y, the SYNACK would have been sent to the address Y, and the TCP in that host would not send the TCP ACK segment back. Even if the attacker were to send an appropriately timed TCP ACK segment, it would not know the correct server sequence number (since the server uses random initial sequence numbers.)

8. Consider sending a large file from a host to another over a TCP connection that has no loss.

(a) Suppose TCP uses AIMD for its congestion control without slow start. Assuming `cwnd` increases by 1 MSS every time a batch of ACKs is received and assuming approximately constant round-trip times, how long does it take for `cwnd` to increase from 5 MSS to 11 MSS (assuming no loss events)?
It takes 1 RTT to increase CongWin to 6 MSS; 2 RTTs to increase to 7 MSS; 3 RTTs to increase to 8 MSS; 4 RTTs to increase to 9 MSS; 5 RTTs to increase to 10 MSS; and 6 RTTs to increase to 11 MSS.

(b) What is the average throughput (in terms of MSS and RTT) for this connection up through time = 6 RTT?
In the first RTT 5 MSS was sent; in the second RTT 6 MSS was sent; in the third RTT 7 MSS was sent; in the forth RTT 8 MSS was sent; in the fifth RTT, 9 MSS was sent; and in the sixth RTT, 10 MSS was sent. Thus, up to time 6 RTT, 5+6+7+8+9+10 = 45 MSS were sent (and acknowledged). Thus, we can say that the average throughput up to time 6 RTT was (45 MSS)/(6 RTT) = 7.5 MSS/RTT.

9. Consider a simplified TCP AIMD algorithm where the congestion window size is measured in number of segments, not in bytes. In additive increase, the congestion window increases by 1 segment in each RTT. In multiplicative decrease, the congestion window size decreases by 1/2. If the result is not an integer, round *down* to the nearest integer. Suppose that two TCP connections C_1 and C_2 are in the congestion avoidance phase. Connection C_1's RTT is 100 msec and connection C_2's RTT is 200 msec. Assume that when the data rate in the link exceeds the link's speed, all TCP connections experience data segment loss.

   (a) Since the link speed was not included, this question will not be graded.

10. When closing a TCP connection, why is the two-segment-lifetime timeout not necessary on the transition from LAST-ACK to CLOSED?
The two-segment-lifetime timeout results from the need to purge old late duplicates, and uncertainty of the sender of the last ACK as to whether it was received. For the first issue we only need one connection endpoint in TIME WAIT; for the second issue, a host in the LAST ACK state expects to receive the last ACK, rather than send it.

11. You are hired to design a reliable byte-stream protocol that uses a sliding window (Like TCP). This protocol will run over a 1-Gbps network. The RTT of the network is 100 ms, and the maximum segment lifetime is 30 seconds.

    (a) How many bits would you advertise in the `AdvertisedWindow` and `SequenceNum` fields of your protocol header?

    The advertised window should be large enough to keep the pipe full; delay (RTT)X bandwidth here is 100ms X 1Gbps = 100Mb = 12.5 MB of data. This requires 24 bits if we assume the window is measured in bytes ($2^{24} \approx$ 16million) for the `AdvertisedWindow` field. The sequence number field must not wrap around in the maximum segment lifetime. In 30 seconds, 30 Gb = 3.75 GB can be transmitted. 32 bits allows a sequence space of about 4GB, and so will not wrap in 30 seconds. (If the maximum segment lifetime were not an issue, the sequence number field would still need to be large enough to support twice the maximum window size; (see "Finite Sequence Numbers and Sliding Windows" in Section 2.5.)

(b) How would you determine the numbers given above, and which values might be less certain?

The bandwidth is straightforward from the hardware; the RTT is also a precise measurement but will be affected by any future change in the size of the network. The MSL is perhaps the least certain value, depending as it does on such things as the size and complexity of the network, and on how long it takes routing loops to be resolved.

12. If host A receives two SYN packets from the same port from remote host B, the second may be either a retransmission of the original or, if B has crashed and rebooted, an entirely new connection request.

(a) Describe the difference as seen by host A between these two cases.

If a SYN packet is simply a duplicate, its ISN value will be the same as the initial ISN. If the SYN is not a duplicate, and ISN values are clock generated, then the second SYNs ISN will be different.

(b) Give an algorithmic description of what the TCP layer needs to do upon receiving a SYN packet. Consider the duplicate/new cases above and the possibility nothing is listening to the destination port.

We will assume the receiver is single-homed; that is, has a unique IP address. Let < raddr, rport > be the remote sender, and lport be the local port. We suppose the existence of a table T indexed by < lport, raddr, rport > and containing (among other things) data fields lISN and rISN for the local and remote ISNs. if (connections to lport are not being accepted)
send RST
else if (there is no entry in T for < lport, raddr, rport >) // new SYN
Put < lport, raddr, rport > into a table,
Set rISN to be the received packets ISN,
Set lISN to be our own ISN,
Send the reply SYN+ACK
Record the connection as being in state SYN RECD
else if (T [< lport, raddr, rport >] already exists)
if (ISN in incoming packet matches rISN from the table)
if (ISN in incoming packet matches rISN from the table)
// SYN is a duplicate; ignore it
else
send RST to < raddr, rport >)

13. Suppose a TCP connection, with window size 1, loses every other packet. Those that do arrive have RTT = 1 second. What happens? What happens to `TimeOut`? Do this for two cases:

(a) After a packet is eventually received, we pick up where we left off, resuming with `EstimatedRTT` initialized to its pre-timeout value, and `TimeOut` to double that.

Let $E \geq 1$ be the value for `EstimatedRTT`, and $T = 2 \times E$ be the value for

`TimeOut`. We lose the first packet and back off `TimeOut` to $2 \times T$. Then, when the packet arrives, we resume with `EstimatedRTT` $= E$, `TimeOut` $= T$. In other words, `TimeOut` doesn't change.

(b) After a packet is eventually received, we resume with `Timeout` initialized to the last exponentially backed-off value used for the timeout interval.

Let T be the value for `TimeOut`. When when we transmit the packet the first time, it will be lost and we will wait time T . At this point we back off and retransmit using `TimeOut` $= 2 \times T$. The retransmission succeeds with an RTT of 1 sec, but we use the backed-off value of $2 \times T$ for the next `TimeOut`. In other words, `TimeOut` doubles with each received packet. This is Not Good.

14. Suppose TCP's measured RTT is 1.0 second except that every Nth RTT is 4.0 seconds. What it the largest N, approximately, that doesn't result in timeouts in the steady state (i.e. for which the Jackobson/Karels `TimeOut` remains greater than 4.0 seconds)? Use $\delta = 1/8$.

    We could create a table starting from scratch, using an initial `EstimatedRTT` of 1.0 and seeding the first few rows with a couple instances of `SampleRTT` $= 4.0$ to get `TimeOut` ≥ 4.0 in the first place. N is between 6 and 7 here.

15. Consult *Request for Comments* 793 to find out how TCP is supposed to respond if a FIN or an RST arrives with a sequence number other than `NextByteExpected`. Consider both when the sequence number is within the receive window and when it is not. Answers are found in the RFC