Redes de computadoras, 8 de diciembre de 2024

Transferencia confiable, TCP y control de congestión

Martínez Buenrostro Jorge Rafael

correo, molap96@gmail.com
Universidad Autónoma Metropolitana
Unidad Iztapalapa, México

Knowledge Checks

Connectionless Transport: UDP

UDP header files

Which fields are in a UDP segment header? Source port, destination port, length (of UDP header plus payload), internet checksum

UDP segment length field

Why is the UDP header length field needed? Because the payload section can be of variable length, and this lets UDP know where the segment ends

Internet Checksum and UDP

Over what set of bytes is the checksum field in the UDP header computed over? The entire UDP segment, except the checksum field itself, and the IP senders and receive address fields

- On the sending side, the UDP sender will take each application-layer chunk of data written into a UDP socket and send it in a distinct UDP datagram. And then on the receiving side, UDP will deliver a segment's payload into the appropiate socket, preserving the application-defined message boundary.

What is checksum?

The next statements are true about a checksum:

- A checksum is computed at a sender by considering each byte within a packet as a number, and then adding these numbers (each number representing a bytes) together to compute a sum (which is known as a checksum)
- The sender-computed checksum value is often included in a checksum field within a packet header

- The receiver of a packet with a checksum field will add up the received bytes, just as the sender did, and compare this locally-computed checksum with the checksum value in the packet header. If these two values are **different** then the receiver **knows** that one of the bits in the received packet has been changed during transmission from sender to receiver.

Computing the internet Checksum (1)

Compute the Internet checksum for these two 16-bit words: 11110101 11010011 and 10110011 01000100 To compute the Internet checksum of a set of 16-bit words, we compute the one's complement sum of the two words. That is, we add the two numbers together, making sure that any carry into the 17th bit of this initial sum is added back into the 1's place of the resulting sum); we then take the one's complement of the result. This means that the Internet checksum is **01010110 11100111**

Computing the internet Checksum (2)

Compute the Internet checksum for these two 16-bit words: 01000001 11000100 and 00100000 00101011 To compute the Internet checksum of a set of 16-bit words, we compute the one's complement sum of the two words. That is, we add the two numbers together, making sure that any carry into the 17th bit of this initial sum is added back into the 1's place of the resulting sum); we then take the one's complement of the result. This means that the Internet checksum is **10011110 00010000**

`https://traductordebinario.com/calculadora-de-sumas-binario/`
`https://www.allmath.com/es/calculadora-del-complemento-a-uno.php`

UDP Checksum: how good is it?

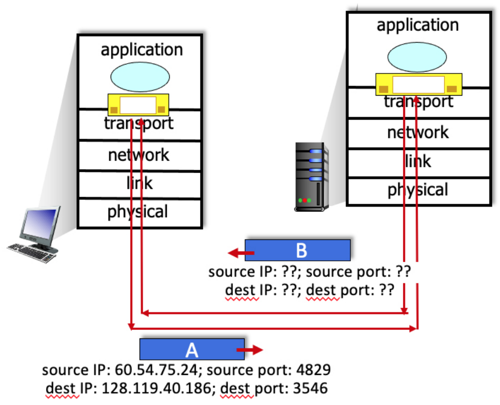
When computing the Internet checksum for two numbers, a single flipped bit (i.e., in just one of the two numbers) will always result in a changed checksum

UDP Checksum: how good is it?

When computing the Internet checksum for two numbers, a single flipped bit (i.e., in just one of the two numbers) will NOT result in a changed checksum.

IP addresses and port numbers in a UDP segment sent in reply

Now consider the UDP datagram (and the IP datagram that will encapsulate it) sent in reply by the application on host 128.119.40.186 to the original sender host, labeled B in the figure above.


application
transport
network
link
physical
application
transport
network
link
physical
B
source IP: ??; source port: ??
dest IP: ??; dest port: ??
A
source IP: 60.54.75.24; source port: 4829
dest IP: 128.119.40.186; dest port: 3546

- The source port number of the UDP segment (B) sent in reply is: **3546**
- The source IP address of the IP datagram containing the UDP segment (B) sent in reply is: **128.119.40.186**
- The destination port number of the UDP segment (B) sent in reply is: **4829**
- The destination IP address of the IP datagram containing the UDP segment (B) sent in reply is: **60.54.75.24**

Principles of Reliable Data Transfer

Reliable data transfer protocol mechanisms

Consider the purpose/goals/use of different reliable data transfer protocol mechanism. For the given purpose/goal/use this is the RDT mechanism that is used to implement the given purpose/goal/use

NAK	Lets the sender know that a packet was NOT received correctly at the receiver
Checksum	Used by the sender or receiver to detect bits flipped during a packet's transmission
Sequence numbers	Allows for duplicate detection at receiver
ACK	Lets the sender know that a packet was received correctly at the receiver
Retransmission	Allows the receiver to eventually receive a packet that was corrupted or lost in an earlier transmission

Cumulative ACK

What is meant by cumulative acknowledgment, ACK(n)? A cumulative ACK(n) acks all packets with a sequence number up to and including n as being received

Stop-and-wait: channel utilization

Suppose a packet is 10k bits long, the channel transmission rate connecting a sender and receiver is 10 Mbps, and the round-trip propagation delay is 10 ms. What is the maximum channel utilization of a stop-and-wait protocol for this channel?

$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{10,000/10,000,000}{0{,}01 + 10,000/10,000,000} = 0{,}09 \approx 0{,}1$$

where

L	packet size
R	transmission rate
RTT	Round-trip propagation

Channel utilization with pipelining

Suppose a packet is 10K bits long, the channel transmission rate connecting a sender and receiver is 10 Mbps, and the round-trip propagation delay is 10 ms. What is the channel utilization if a pipelined protocol with an arbitrarily high level of pipelining for this channel?

$$U_{sender} = \frac{N * L/R}{RTT + L/R}$$

where

L	packet size
R	transmission rate
RTT	Round-trip propagation
N	packet pipelining increased utilization

Channel utilization with pipelining (more)

Suppose a packet is 10K bits long, the channel transmission rate connecting a sender and receiver is 10 Mbps, and the round-trip propagation delay is 10 ms. How many packets can the sender transmit before it starts receiving acknowledgment back? The answer is 10 packets

Pipelining

The next statements are true about pipelining:

- A pipelined sender can have trasmitted multiple packets for which the sender has yet to receive an ACK from the receiver
- With a pipelined sender, there may be transmitted packet ïn flight propagating through the channel - packets that the sender has sent but that the receiver has not yet received

Packet buffering in Go-Back-N

What are some reasons for discarding received-but-out-of-sequence packets at the receiver in GBN?

- The sender will resedn that packet in any case
- The implementation at the receiver is simpler

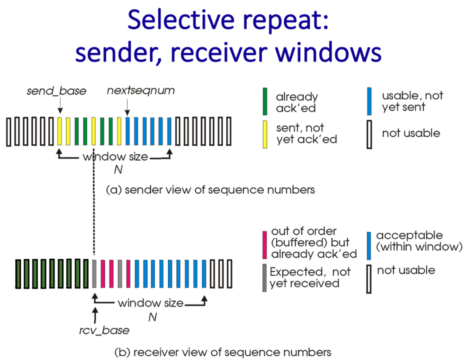
Packet buffering in Go-Back-N (more)

What are some reasons for *not* discarding received-but-out-of-sequence packets at the receiver in GBN?

- Even though that packet will be retransmitted, its next retransmission could be corrupted, so don't discard a perfectly well-received packet

Receiver operation in Selective Repeat

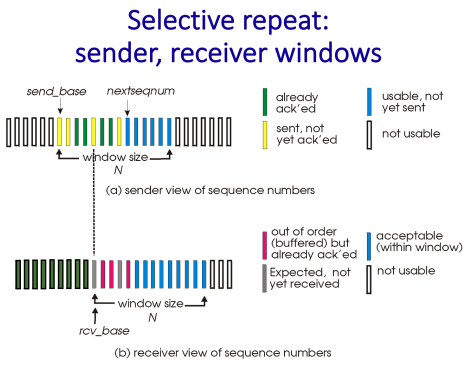
In the SR receiver window (see diagram below), why haven't the red packets been delivered yet?


Selective repeat:
sender, receiver windows
send_base
nextseqnum
window size
N
already ack'ed
sent, not yet ack'ed
usable, not yet sent
not usable
(a) sender view of sequence numbers
out of order (buffered) but already ack'ed
Expected, not yet received
acceptable (within window)
not usable
window size
N
rcv_base
(b) receiver view of sequence numbers

There is a packet with a lower sequence number than any of the red packets that has yet to be received, so in-order dalivery of data in the red packets up to the application layer is not possible

Receiver operation in Selective Repeat (more)

In SR, why does the receiver have to acknowledge packets with sequence numbers that are less than those in its window, which starts at *rcv-base*


Selective repeat:
sender, receiver windows
send_base
nextseqnum
window size
N
already ack'ed
sent, not yet ack'ed
usable, not yet sent
not usable
(a) sender view of sequence numbers
out of order (buffered) but already ack'ed
Expected, not yet received
acceptable (within window)
not usable
window size
N
rcv_base
(b) receiver view of sequence numbers

Because the sender may not have received an ACK for that packet yet.

Connection-oriented Transport: TCP

TCP reliability semantics

[**FALSE**] On the sending side, the TCP sender will take each application-layer chunk of data written into a TCP socket and send it in a distinct TCP segment. And on the receiving side, TCP will deliver a segment's payload into the appropiate socket, preserving the application-defined message boundary.

TCP segment format

Socket port number This field contains the port number associated with the sending socket for this TCP segment.

Data (*or payload*) This field contains application data that was written into a socket by the sender of this TCP segment.

Sequence number This field contains the index in the sender-to-receiver byte stream of the first byte of that data in the payload carried in this segment.

ACK number field This field contains the index in the byte stream of the next in-order byte expected at the receiver.

ACK bit If set, this segment cumulatively ACKs in the byte all data bytes up to, but not including, the byte index in the ACK value field of this segment.

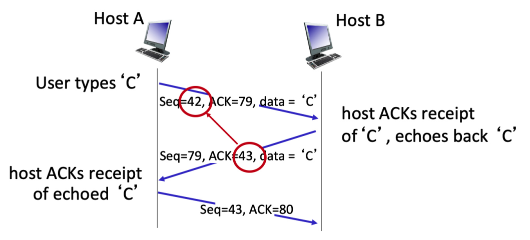
Receiver advertised window This field contains the number of available bytes in the TCP receiver's buffer.

Checksum This field contains the Internet checksum of the TCP segment and selected fields in the IP datagram header.

Header length field This field contains the number of bytes in the TCP header.

TCP sequence numbers and ACKs (1)

Consider the TCP Telnet scenario below. Why is it that the receiver sends an ACK that is one larger than the sequence number in the received datagram?

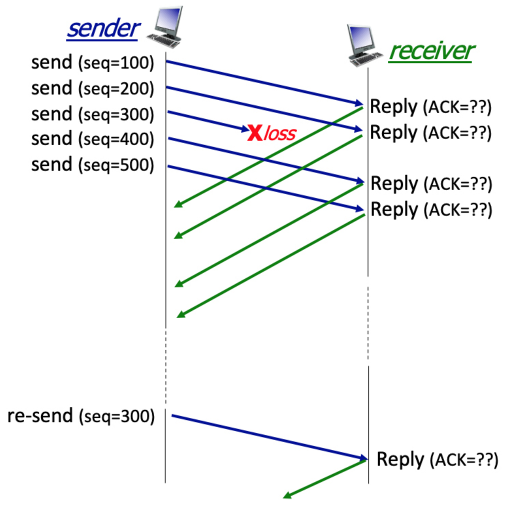

Host A
Host B
User types 'C'
Seq=42, ACK=79, data = 'C'
host ACKs receipt of 'C', echoes back 'C'
Seq=79, ACK=43, data = 'C'
host ACKs receipt of echoed 'C'
Seq=43, ACK=80

simple telnet scenario

Because the send-to receiver segment carries only one byte of data, and after that segment is received, the next expected byte of data is just the next byte (*i.e., has an index that is one larger*) in the data stream.

TCP sequence numbers and ACKs (2)

Suppose that as shown in the figure below, a TCP sender is sending segments with 100 bytes of payload. The TCP sender sends five segments with sequence numbers 100, 200, 300, 400, and 500. Suppose that the segment with sequence number 300 is lost. The TCP receiver will buffer correctly-received but not-yet-in-order segments for later delivery to the application layer (*once missing segments are later received*)



- After receiving segment 100, the receiver responds with an ACK with value **200**.
- After receiving segment 200, the receiver responds with an ACK with value **300**.
- After receiving segment 500, the receiver responds with an ACK with value **300, a duplicate ACK**.
- After receiving the retransmitted segment 300, the receiver responds with an ACK with value **600**.
- The TCP receiver does not respond in the example, with an ACK with value **400**.

TCP RTT Estimation: EWMA

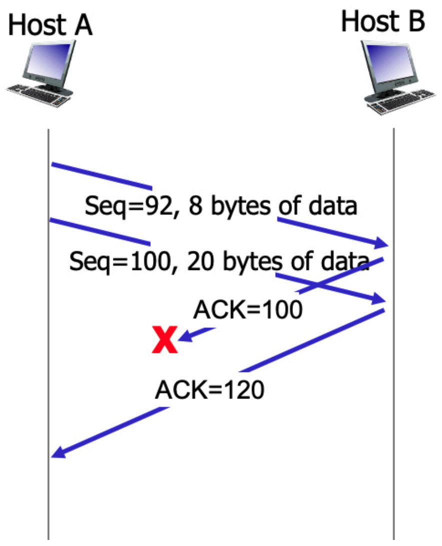
Consider TCP use of an exponentially weighted moving average (*EWMA*) to compute the nth value of the estimated RTT:

$$EstimatedRTT_n)(1 - a) * EstimatedRTT_{n-1} + a * SampleRTT_n$$

[**FALSE**] with this EWMA algorithm the alue of $EstimatedRTT_n$ has no dependence on the earlier sample, $SampleRTT_{n-1}$.

TCP timer management

Consider the TCP Telnet below. What timer-related action does the sender take on the receipt of ACK 120?



Cancels any running timers.

TCP Flow Control

With TCP's flow control mechanism, where the receiver tells the sender how much free buffer space it has (*and the sender always limits the amount of outstanding, unACKed, in-flight to less than this amount*), it is not possible for the sender to send more that the receiver has room to buffer.

TCP connection management

SYN MESSAGE A message from client to server initiating a connection request.

SYNACK MESSAGE A message from server to client ACKing receipt of a SYN message and indicating the willingness of the server to establish a TCP connection with the client.

FIN MESSAGE A message indicating that the sending side is initiating the protocol to terminate a connection.

FINBACK MESSGE A message sent in response to a request to terminate a connection. ACKing that the side receiving this message is algo willing to terminate the connection.

RESET MESSAGE A general purpose error message used during connection set up or tear down to let the other side know that an error has ocurred, and that the referenced connection should be shut down.

TCP Fast Retransmit

Consider the TCPs Fast Retransmit optmization. Of course, the sender doesn't know for sure that the segment with sequence #100 is actually lost (*it can't see into the channel*). Can a sender get three duplicate ACKs for a segment that in fact has *not* been lost? The following statements are true. Suppose a channel can lose, but will not corrupt, messages.

- If the channel can reorder messages, a triple duplicate ACK can occure even though a message is not lost; since it's possible that a message has just been reordered and has not yet arrived when the three ACKs were generated.
- If the channel cannot reorder messages, a triple duplicate ACK indicates to the sender that a segment loss has happened for sure. Actually (*again assuming the channel cannot corrupt or reorder messages*), even a *single* duplicate ACK would indicate that a segment loss has happend for sure.

Principles of Congestion Control

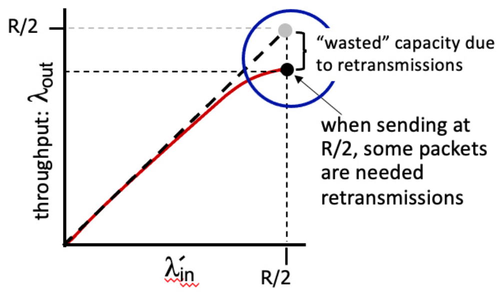
Congestion control versus flow control

Between the next options a **a glass overflowing** and **a talking head** are concepts that need flow control. The others would suggest the need for congestion control

- A penguin crowd
- Car traffic
- A crowd of people
- A glass overflowing
- A talking head

Two congested senders

Consider the figure below, which shows the application-to-application throughput achieved when two senders are competing at a shared bottleneck link. Suppose that then the overall arrival rate, $lambd_{in}(for\ each\ sender)$ is closer to R/2, the throughput to the application layer (*at each receiver*), $lambda_{out}$, is equal to $0{,}8 * lambda_{in}$.



What fraction of the packets transmitted at the sender are retransmissions? **0.20**

Different approaches towards congestion control

Here are some options of how the sender detects congestion:

End-end The sender infers segment loss from the absence of an ACK from the receiver.

Network-assisted Bits are set at a congested router in a sender-to-receiver datagram, and bits are in the returned to the sender in a receiver-to-sender ACK, to indicate congestion to the sender.

Delay-based The sender measure RTTs and uses the current RTT measurement in onfer the level of congestion.

TCP Congestion Control

TCPs AIMD algorithm

The following statements are true about TCP's Additive-increase-multiplicative-decrease (*AIMD*) algorithm:

- AIMD cuts the congestion window size, cwnd, in half whenever loss is detected by a triple duplicate ACK.
- AIMD cuts the congestion window size, cwnd, i to 1 whenever a timeout occurs.
- AIMD is a end-end approach to congestion control

TCPs AIMD algorithm (2)

How is the sending rate typically regulated in a TCP implementation? By keeping a window size cwnd over the sequence number space, and making sure that no more than cwnd bytes of data are outstanding (*i.e., unACKnowledged*). The size of cwnd ir regulated by AIMD.

TCPs Slowstart algorithm

Consider the transport-layer flows interacting at a congested link. In the face of such congestion, what happens at this link to a transport-layer flow that does not cut back o its sending rate? Nothing different from the other flows crossing the congested link.

Ejercicios interactivos

Internet checksum

Consider the two 16-bit words below. Recall that to compute the Internet checksum of a set of 16-bit words, we compute the one's complement sum of the two words. That is, we add the two numbers together, making sure that any carry into the 17th bit of this initial sum is added back into the 1's place of the resulting sum; we then take the one's complement of the result. Compute the Internet checksum value for these 16-bit words:

10110011 00100111 *this binary number is 45,863 decimal*

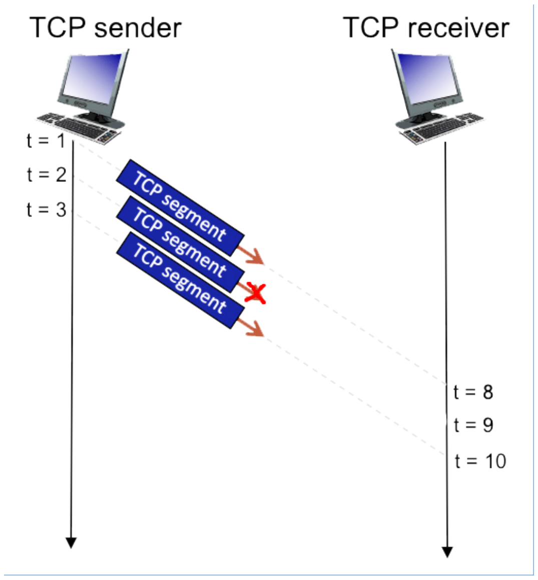
10011100 11100010 *this binary number is 40,162 decimal*

1. What is the sum of these two 16 bit numbers? **101000000001010**

2. Using the sum from question 1, what is the checksum? **1010111111110101**

Números de secuencia y de ACK, con pérdida de segmentos

Consider the figure below in which a TCP senderand receiver communicate over a connection in which the sender-¿receiver segments may be lost. The TCP sender sends an initial window of three segments. Suppose the initial value of the sender-¿receiver sequence number is 389 and the first three segments each contain 431 bytes. The delay between the sender and receiver is seven time units, and so the first segment arrives at the receiver at $t = 8$. As shown in figure below, one of the three segments are lost between the segment and receiver.

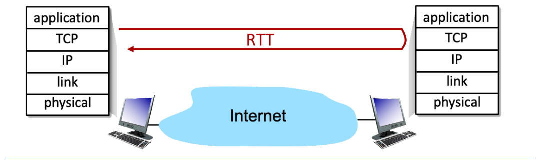


1. Give the sequence numbers associated with each of the three segments sent by the sender. **a=389, b=820, c=1251**

2. Give the ACK numbers the receiver sends in response to each of the segments. If a segment never arrives use 'x' to detomate it. **a=820, b=x, c=820**

RTT y timeout en TCP

Suppose that TCP's current estiamted values for the round trip time (*estimatedRTT*) and deviation in the RTT (*DevRTT*) are 360msec and 25msec, respectively. Suppose that the next three measured values of the RTT are 380msec, 400msec, and 220msec respectively.



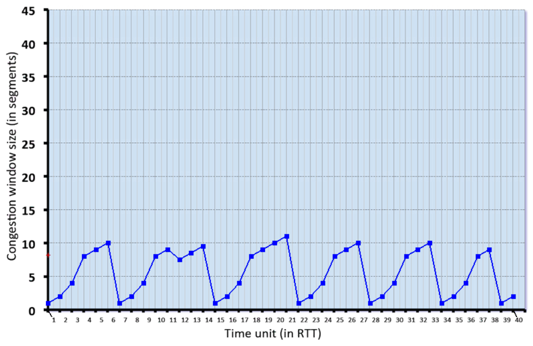
Compute TCP's new value of *DevRTT*, *estimatedRTT*, and the TCP timeout value after each of these three measured RTT values is obtained. Use the values of $\alpha = 0{,}125$, and $\beta = 0{,}25$.

1. What is the estimatedRTT after the first RTT? **346.25**
2. What is the RTT Deviation for the first RTT? **42.5**
3. What is the TCP timeout for the first RTT? **516.25**
4. What id the estimatedRTT after the second RTT? **330.47**
5. Wht is the RTT Deviation for the second RTT? **63.44**
6. What is the TCP timeout for the second RTT? **584.22**
7. What is the estimatedRTT after the third RTT? **327.91**
8. What is the RTT Deviation for the third RTT? **52.7**
9. What is the TCP timeout for the third RTT? **538.69**

Evolución de la ventana de congestión de TCP

Consider the figure below, which plots the evolution of TCP's congestion window at the beginning of each time unit (*where the unit of time is equal to the RTT*). In the abstract model for this problem, TCP sends a "flight.ºf packets of size cwnd at the beginning of each time unit. The result of sending that flight of packets is that either:

1. All packets are ACKed at the end of the time unit
2. There is a timeout for the first packet
3. There is a triple duplicate ACK for the first packet



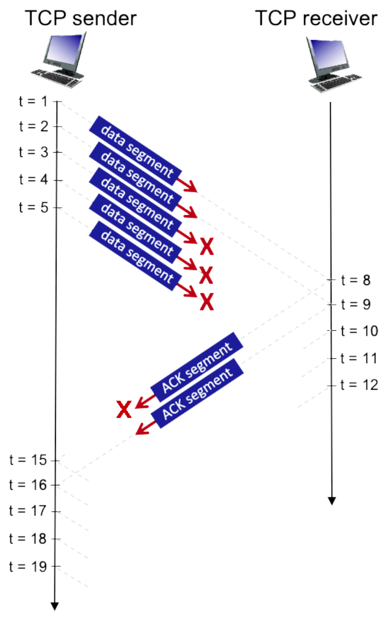
In this problem, you are asked to reconstruct the sequence of events (*ACKs, losses*) that resulted in the evolution of TCP's cwnd shown below. Consider the evolution of TCP's congstion window in the example above and answer the following questions. The initial value of cwnd is 1 and the initial value od ssthresh is 8.

1. Give the times at which TCP is in slow start. **1,2,3,7,8,9,15,16,17, 22,23,24,28,29,30,34,35,36,39,40**
2. Give the times at which TCP is in congestion avoidance. **4,5,6,10,11, 13,14,18,19,20,21,25,26,27,31,32,33,37,38**
3. Give the times at which TCP is in fast recovery. **12**
4. Give the times at which packets are lost via timeout. **6,14,21,27,33,38**
5. Give the times at which packets are lost via *triple ACK*. **11**
6. Give the times at which the value of *ssthresh* changes. **7,12,15,22,28,39**

Retransmisiones en TCP

Consider the figure below in which a TCP sender and receiver communicate over a connection in which the segments can be lost. The TCP sender wants to send a total of 10 segments to the receiver and sends an initial window of 5 segments at t = 1, 2, 3, 4, and 5, respectively. Suppose the initial value of the sequence number is 11 and every segment sent to the receiver each contains 525 bytes. The delay between the sender and receiver is 7 time units, and so the first segment arrives at the receiver at t = 8, and an ACK for this segment arrives at t = 15. As shown in the figure, 3 of the 5 segments is lost between the sender and the receiver, but one of the ACKs is lost. Assume there are no timeouts and any out of order segments received are thrown out.

1. What is the sequence number of the segment sent at t=1? **11**
2. What is the sequence number of the segment sent at t=2? **536**
3. What is the sequence number of the segment sent at t=3? **1061**



4. What is the sequence number of the segment sent at t=4? **1586**
5. What is the sequence number of the segment sent at t=5? **2111**
6. What is the value of the ACK sent at t=8? **536**
7. What is the value of the ACK sent at t=9 **1061**
8. What is the value of the ACK sent at t=10 **x**
9. What is the value of the ACK sent at t=11? **x**
10. What is the value of the ACK sent at t=12? **x**
11. What is the sequence number of the segment sent at t = 15? **x**
12. What is the sequence number of the segment sent at t = 15? **2636**
13. What is the sequence number of the segment sent at t = 17? **3161**
14. What is the sequence number of the segment sent at t = 18? **x**
15. What is the sequence number of the segment sent at t = 19? **x**