RAMCO INSTITUTE OF TECHNOLOGY
Approved by AICTE, New Delhi & Affiliated to Anna University
Accredited by NAAC & An ISO 9001: 2015 Certified Institution
NBA Accredited UG Programs: CSE, EEE, ECE and MECH

Department of Artificial Intelligence and Data Science

Academic Year: 2022 - 2023 (Even Semester)

Degree, & Semester: B. TECH /IV/

Regulation: Anna University R 2021

Course Code & Title: CS3591- Computer Networks

Category of the Course: Professional Core Course

Name of the Faculty member(s) with Designation & Department:

Dr.M. Kaliappan

Professor and Head

Department of Artificial Intelligence and Data Science

Acknowledgement:

James F. Kurose, Keith W. Ross, Computer Networking, A Top-Down Approach Featuring the Internet, Eighth Edition, Pearson Education, 2021

Review Questions & Answers

Problems & Answer

Chapter 3: TRANSPORT LAYER

Chapter 3: TRANSPORT LAYER

1.Suppose the network layer provides the following service. The network layer in the source host accepts a segment of maximum size 1,200 bytes and a destination host address from the transport layer. The network layer then guaran- tees to deliver the segment to the transport layer at the destination host. Suppose many network application processes can be running at the destination host.

a) **Design the simplest possible transport-layer protocol that will get application data to the desired process at the destination host. Assume the operating system in the destination host has assigned a 4-byte port number to each running application process.**
b) **Modify this protocol so that it provides a "return address" to the destination process.**
c) **In your protocols, does the transport layer "have to do anything" in the core of the computer network?**

ANSWER:
a)
The Simple Transport Protocol takes data not exceeding 1196 bytes at the sender side.

- It accepts four byte of destination port number and host address.
- The Simple Transport Protocol gives the destination host address and the resulting segment to the network layer.
- The network layer sends the segment to Simple Transport Protocol at the destination host.
- The Simple Transport Protocol observes the port number.
- Abstracts the data from the segment in the Simple Transport Protocol.
- Finally, send the data to the process recognized by the port number.

b)
Consider the two header fields in the segment:

1. Source port field
2. Destination port field

The Simple Transport Protocol creates application data, source and destination port numbers in the segment. It sends the destination host address to the network layer. Then, The Simple Transport Protocol is receiving host address and provides the process the source port number and the application data.
c)
No, the transport layer does not have to do anything in the core.
The reason is that, the transport layer "lives" in the end systems.

2. Consider a planet where everyone belongs to a family of six, every family lives in its own house, each house has a unique address, and each person in a given house has a unique name. Suppose this planet has a mail service that delivers letters from source house to destination house. The mail service requires that (1) the letter be in an envelope, and that (2) the address of the destination house (and nothing more) be clearly written on the envelope. Suppose each family has a delegate family member who collects and distributes letters for the other family members. The letters do not necessarily provide any indication of the recipients of the letters.

a) **Using the solution to Problem R1 above as inspiration, describe a protocol that the delegates can use to deliver letters from a sending family member to a receiving family member.**
b) **In your protocol, does the mail service ever have to open the envelope and examine the letter in order to provide its service?**

ANSWER:

a)

- Sender has to provide the address of the destination name. It is written by the delegate to the planet's mail service.
- After receive the destination address, the envelop the written on the top details.

b)

No.

The mail service ever has not to open the envelope and examine the letter in order to provide its service.

3. Consider a TCP connection between Host A and Host B. Suppose that the TCP segments traveling from Host A to Host B have source port number x and destination port number y. What are the source and destination port numbers for the segments traveling from Host B to Host A?

ANSWER:

The source and destination port numbers for the segments travelling from Host B to Host A: Source port number y and destination port number x.

4. Describe why an application developer might choose to run an application over UDP rather than TCP.

ANSWER:

The TCP'(transmission control protocol) can choke the application's sending rate at times of bottleneck.

The UDP(user datagram protocol) does not keep joining state and does not track any of the limits.

Even though data transfer by TCP is dependable, some applications do not need dependable TCP data transfer. So, UDP (user datagram protocol) rather than TCP (transmission control protocol):

5.Why is it that voice and video traffic is often sent over TCP rather than UDP in today's Internet? (Hint: The answer we are looking for has nothing to do with TCP's congestion-control mechanism.)

ANSWER:

Most firewalls are configured to block UDP traffic, using TCP for video and voice traffic lets the traffic though the firewalls. So, that voice and video traffic is often sent over TCP rather than UDP in today's Internet.

6. Is it possible for an application to enjoy reliable data transfer even when the application runs over UDP? If so, how?

ANSWER:

The application developer can put consistent data transfer into the application layer protocol. It contains a significant amount of work and debugging.

7. Suppose a process in Host C has a UDP socket with port number 6789. Sup- pose both Host A and Host B each send a UDP segment to Host C with destination port number 6789. Will both of these segments be directed to the same socket at Host C? If so, how will the process at Host C know that these two segments originated from two different hosts?

ANSWER:

Yes.

For each received segment, at the socket interface, the operating system will provide the process with the IP addresses to determine the origins of the individual segments.

8. Suppose that a Web server runs in Host C on port 80. Suppose this Web server uses persistent connections, and is currently receiving requests from two different Hosts, A and B. Are all of the requests being sent through the same socket at Host C? If they are being passed through different sockets, do both of the sockets have port 80? Discuss and explain.

ANSWER:

For each persistent connection, the Web server creates a separate "connection socket". Each connection socket is identified with a four-tuple: (source IP address, source port number, destination IP address, destination port number). When host C receives and IP datagram, it examines these four fields in the datagram/segment to determine to which socket it should pass the payload of the TCP segment. Thus, the requests from A and B pass through different sockets. The identifier for both of these sockets has 80 for the destination port; however, the identifiers for these sockets have different values for source IP addresses. Unlike UDP, when the transport layer passes a TCP segment's payload to the application process, it does not specify the source IP address, as this is implicitly specified by the socket identifier.

9. In our RDT protocols, why did we need to introduce sequence numbers?
ANSWER:
Sequence numbers are necessary for a receiver to find out whether an arriving packet contains new data or is a retransmission in our **RDT** protocols.

10. In our RDT protocols, why did we need to introduce timers?
ANSWERS:
To handle losses in the channel. If the ACK for a transmitted packet is not received within the duration of the timer for the packet, the packet (or its ACK or NACK) is assumed to have been lost. Hence, the packet is retransmitted.

11. Suppose that the roundtrip delay between sender and receiver is constant and known to the sender. Would a timer still be necessary in protocol rdt 3.0, assuming that packets can be lost? Explain.

ANSWERS:
Yes, a timer still is necessary in protocol rdt 3.0.
Reason: Assume the packet is loss, the sender knows the round trip delay time. It is used for estimate the transfer packet time.

12. **Visit the Go-Back-N Java applet at the companion Web site.**

a) Have the source send five packets, and then pause the animation before any of the five packets reach the destination. Then kill the first packet and resume the animation. Describe what happens.
b) Repeat the experiment, but now let the first packet reach the destination and kill the first acknowledgment. Describe again what happens.
c) Finally, try sending six packets. What happens?

ANSWER:

a) Have the source send five packets, and then pause the animation before and of the five packet reach the destination. Then kill the first packet and resume the animation. Describe what happens.
   The packets were received out of order and no packets were acknowledged. The packets were then retransmitted. Once they were retransmitted and received and ACK was sent to the sender.
b) Repeat the experiment, but now let the first packet reach the destination and kill the first ACK. Describe what happens.

-The first ACK was lost but the others made it to the sender and their timers were stopped.

c) Finally, try sending six packets. What happens?
-It only lets you send 5 packets out at once. Before the sixth packet can be sent you must wait forthe first packet to finish the ACK.

13. Repeat R12, but now with the Selective Repeat Java applet. How are Selective Repeat and Go-Back-N different?

ANSWER:
Selective Repeat java applet protocol:

- If sender sent more than one packet, it reaches the destination might be one packet is killed and other packets stored in buffer at end receiver.
- The receiver send acknowledgement to sender, but the sender does not receiver acknowledgement.

Go-Bank-N protocol:

- If sender sent more than one packet, it reaches the destination might be one packet is killed and other packets reached destination without buffering.
- The receiver does not send acknowledgement to sender, but the sender receive acknowledgement.

14. True or false?

a) Host A is sending Host B a large file over a TCP connection. Assume Host B has no data to send Host A. Host B will not send acknowledgments to Host A because Host B cannot piggyback the acknowledgments on data. **Answer: False**

b) The size of the TCP rwnd never changes throughout the duration of the connection.
Answer: False

c) Suppose Host A is sending Host B a large file over a TCP connection. The number of unacknowledged bytes that A sends cannot exceed the size of the receive buffer.
Answer: True

d) Suppose Host A is sending a large file to Host B over a TCP connection. If the sequence number for a segment of this connection is m, then the sequence number for the subsequent segment will necessarily be m + 1. **Answer: False**

e) The TCP segment has a field in its header for rwnd. **Answer: True**

f) Suppose that the last SampleRTT in a TCP connection is equal to 1 sec. The current value of TimeoutInterval for the connection will neces- sarily be ≥ 1 sec. **Answer : False**

g) Suppose Host A sends one segment with sequence number 38 and 4 bytes of data over a TCP connection to Host B. In this same segment the acknowledgment number is necessarily 42. **Answer: False**

15. Suppose Host A sends two TCP segments back to back to Host B over a TCP connection. The first segment has sequence number 90; the second has sequence number 110.

a) **How much data is in the first segment?**

b) **Suppose that the first segment is lost but the second segment arrives at B. In the acknowledgment that Host B sends to Host A, what will be the acknowledgment number?**

ANSWER:

a)

Consider sequence numbers, First segment=90

Second segment=110

Data in the first segment=110-90

=20

b) Consider the first segment is lost but the second segment arrives at B. In the acknowledgment that Host B sends to Host A, then the acknowledgment number will be first segment of sequence number, that is 90.

16. Consider the Telnet example discussed in Section 3.5. A few seconds after the user types the letter 'C,' the user types the letter 'R.' After typing the letter 'R,' how many segments are sent, and what is put in the sequence number and acknowledgment fields of the segments?

ANSWER:

Segment	Sequence number	Acknowledgment field
First segment	seq = 43	ack=80
Second segment	seq = 80	ack=44
Third segment	seq = 44	ack=81

17. Suppose two TCP connections are present over some bottleneck link of rate R bps. Both connections have a huge file to send (in the same direction over the bottleneck link). The transmissions of the files start at the same time. What transmission rate would TCP like to give to each of the connections?

ANSWER:

Consider data,

Two TCP connections are present over some bottleneck link of rate R bps. Both connections have a huge file to send. transmissions of the files start at the same time.

Transmission rate would TCP like to give to each of the connections= R/2.

18. True or false? Consider congestion control in TCP. When the timer expires at the sender, the value of ssthresh is set to one half of its previous value.

ANSWER:
False. The slow start threshold(ssthresh) is set to one half of its previous value in the congestion window.

19. In the discussion of TCP splitting in the sidebar in Section 7.2, it was claimed that the response time with TCP splitting is approximately
4. RTT_{FE} + RTT_{BE} + processing time. Justify this claim.

ANSWER:
Consider the delay in receiving a response for a search query.In slow start, the server requires three TCP windows to deliver the response.
Thus, the time from when an end system initiates a TCP connection until the time when it receives the last packet of the response is roughly 4*round trip time (RTT).
One RTT is used to set up the TCP connection and three RTTs are used for the three windows of data plus the processing time in the data center. These RTT delays can lead to a noticeable delay in returning search results for a significant fraction of queries.
Moreover, there can be significant packet loss in access networks, leading to TCP retransmissions and even larger delays.
Improve the user-perceived performance is to utilize TCP splitting by breaking the TCP connection at the front-end server.
The client establishes a TCP connection to the nearby front-end, and the front-end maintains a persistent TCP connection to the data center with a very large TCP congestion window with TCP splitting.
Hence, it is justified that the response time roughly becomes 4*RTT_{FE} +RTT_{BE} + processing time.
Where, RTT_{FE} is the round-trip time between client and front-end server, and RTT_{BE} is the round-trip time between the front-end server and the data center (back-end server).

PROBLEMS

1.Suppose Client A initiates a Telnet session with Server S. At about the same time, Client B also initiates a Telnet session with Server S. Provide possible source and destination port numbers for

- **a. The segments sent from A to S.**
- **b. The segments sent from B to S.**
- **c. The segments sent from S to A.**
- **d. The segments sent from S to B.**
- **e. If A and B are different hosts, is it possible that the source port number in the segments from A to S is the same as that from B to S?**
- **f. How about if they are the same host?**

ANSWERS:

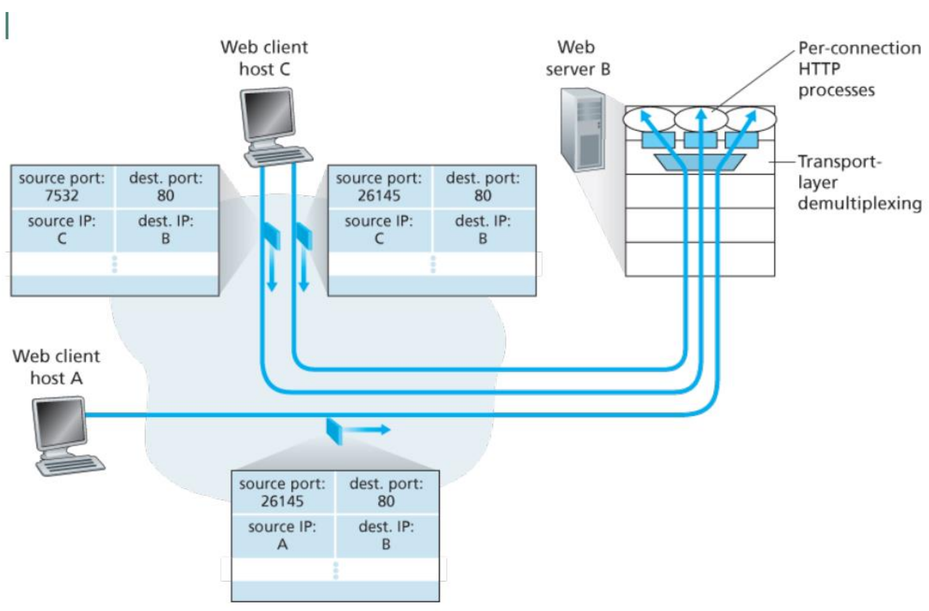
Serial port number Destination port number

a) A → S	467	23
b) B → S	513	23
c) S → A	23	467
d) S → B	23	513

e) Yes

f) NO

2. Consider the Figure. What are the source and destination port values in the segments flowing from the server back to the clients' processes? What are the IP addresses in the network-layer datagrams carrying the transport-layer segments?



ANSWER:
Assume the IP addresses of the hosts A, B, and C are a, b, c, respectively. (Note that a, b, c are distinct.)

- To host A: Source port =80, source IP address = b, destination port = 26145, destination IP address = a
- To host C, left process: Source port =80, source IP address = b, dest port = 7532, destination IP address = c
- To host C, right process: Source port =80, source IP address = b, dest port = 26145, destination IP address = c

3. UDP and TCP use 1s complement for their checksums. Suppose you have the following three 8-bit bytes: 01010011, 01100110, 01110100. What is the 1s complement of the sum of these 8-bit bytes? (Note that although UDP and TCP use 16-bit words in computing the checksum, for this problem you are being asked to consider 8-bit sums.) Show all work. Why is it that UDP takes the 1s complement of the sum; that is, why not just use the sum? With the 1s complement scheme, how does the receiver detect errors? Is it possible that a 1-bit error will go undetected? How about a 2-bit error?

ANSWERS:
Given 8-bit bytes are as follows:

```
    01010011
    01100110
    01110100
```

Calculate the sum of the given 3 bytes.
Add first two 8-bit bytes:

```
0 1 0 1 0 0 1 1
0 1 1 0 0 1 1 0
1 0 1 1 1 0 0 1
```

Now add the result with the 3rd byte.

```
  1 0 1 1 1 0 0 1
  0 1 1 1 0 1 0 0
1 0 0 1 0 1 1 0 1
```

Wrap around the extra bit.

```
0 0 1 0 1 1 0 1
              1
0 0 1 0 1 1 1 0
```

The sum three 8-bit bytes is 00101110. Invert all the bits to get the check sum.
Check sum is 11010001.

Now calculate the 1’s compliment of the sum. Convert all 0’s to 1’s and vice versa to find the 1’s compliment.
The 1’s compliment of (sum) 00101110 is 11010001.

It is clear that the 1’s compliment and the checksum are same.

User Datagram Protocol (UDP) uses the 1’s complement as it is same as the checksum of the sum.

The checksum is used by the receiver (host) to identify the errors in the segment.

The process of detecting errors by the receiver:
The receiver performs the following steps at the receiver end to identify the errors in the segment.

- Add all the bytes including checksum.
- Observe the sum.
    - If it contains all 1’s then the segment has errors.
    - If it contains 1 or more 0’s then the segment contains errors.

Error types that are identified using 1’s compliment method:

- Using 1’s compliment method, it is possible to detect all the 1-bit errors.
- Using 1’s compliment method, there is a possibility that some 2-bit errors are left undetected.

4. UDP and TCP use 1s complement for their checksums. Suppose you have the following three 8-bit bytes: 01010011, 01100110, 01110100. What is the 1s complement of the sum of these 8-bit bytes? (Note that although UDP and TCP use 16-bit words in computing the checksum, for this problem you are being asked to consider 8-bit sums.) Show all work. Why is it that UDP takes the 1s complement of the sum; that is, why not just use the sum? With the 1s complement scheme, how does the receiver detect errors? Is it possible that a 1-bit error will go undetected? How about a 2-bit error?

ANSWERS:
Given 8-bit bytes are as follows:

01010011
01100110
01110100

Calculate the sum of the given 3 bytes.
Add first two 8-bit bytes:

```
0 1 0 1 0 0 1 1
0 1 1 0 0 1 1 0
1 0 1 1 1 0 0 1
```

Now add the result with the 3rd byte.

```
    1   0   1   1   1   0   0   1
    0   1   1   1   0   1   0   0
1   0   0   1   0   1   1   0   1
```

Wrap around the extra bit.

```
0   0   1   0   1   1   0   1
                            1
0   0   1   0   1   1   1   0
```

The sum three 8-bit bytes is 00101110. Invert all the bits to get the check sum.
Check sum is 11010001.

Now calculate the 1’s compliment of the sum. Convert all 0’s to 1’s and vice versa to find the 1’s compliment.
The 1’s compliment of (sum) 00101110 is 11010001.
It is clear that the 1’s compliment and the checksum are same.
User Datagram Protocol (UDP) uses the 1’s complement as it is same as the checksum of the sum.
The checksum is used by the receiver (host) to identify the errors in the segment.
The process of detecting errors by the receiver:
The receiver performs the following steps at the receiver end to identify the errors in the segment.

- Add all the bytes including checksum.
- Observe the sum.
    - If it contains all 1’s then the segment has errors.
    - If it contains 1 or more 0’s then the segment contains errors.

Error types that are identified using 1’s compliment method:

- Using 1’s compliment method, it is possible to detect all the 1-bit errors.
- Using 1’s compliment method, there is a possibility that some 2-bit errors are left undetected.

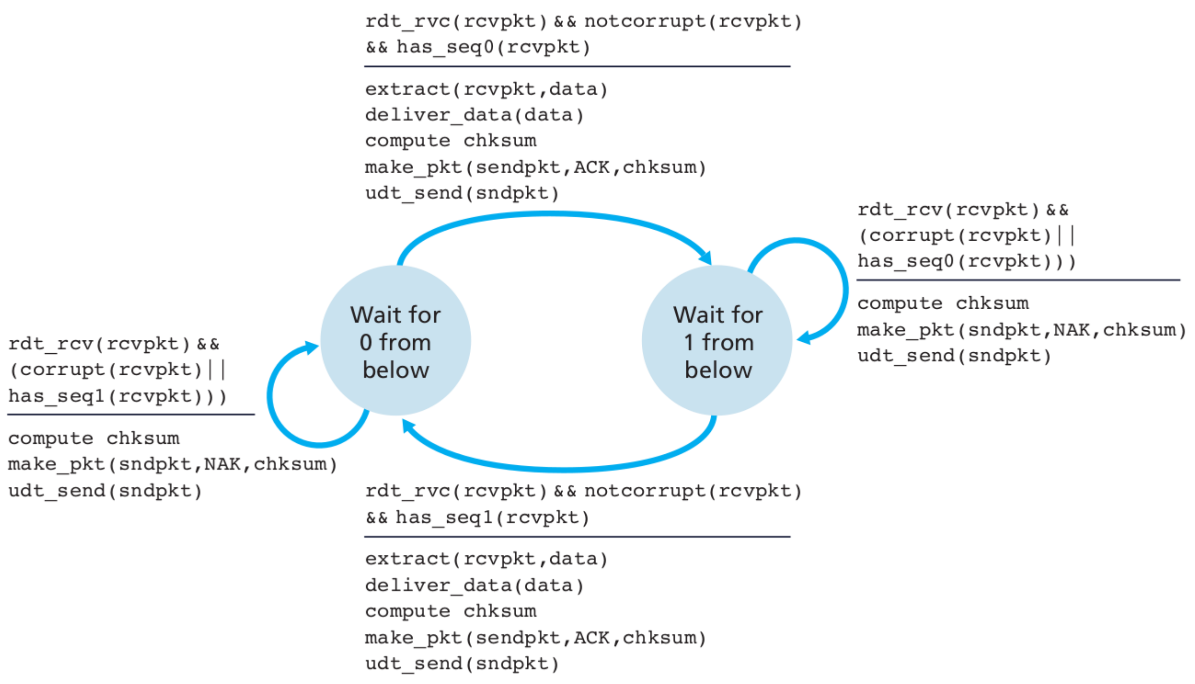
5. Suppose that the UDP receiver computes the Internet checksum for the received UDP segment and finds that it matches the value carried in the checksum field. Can the receiver be absolutely certain that no bit errors have occurred? Explain.
ANSWER:
Certainty of no bit errors at receiver end using checksum using UDP:

The receiver in the UDP (User Datagram Protocol) verifies the received segment by calculating internet checksum and comparing it with the value in the checksum field.

- The 1's compliment of the sum is considered as checksum. So, when this check sum is used to detect the errors in the packet, the errors remain in under cover.
- In case, if two 16-bit words are added, then there is a scope for flipping the 0's and 1's. If the bits are flipped, the sum will be same and error can't detected.

Therefore, it is not possible to the receiver in UDP to be sure that there are no bit errors have occurred.

6. Consider our motivation for correcting protocol rdt2.1. Show that the receiver, shown in Figure, when operating with the sender shown in Figure, can lead the sender and receiver to enter into a deadlock state, where each is waiting for an event that will never occur.


rdt_rvc(rcvpkt) && notcorrupt(rcvpkt) && has_seq0(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
compute chksum
make_pkt(sendpkt,ACK,chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)|| has_seq0(rcvpkt)))
compute chksum
make_pkt(sndpkt,NAK,chksum)
udt_send(sndpkt)
Wait for 0 from below
Wait for 1 from below
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)|| has_seq1(rcvpkt)))
compute chksum
make_pkt(sndpkt,NAK,chksum)
udt_send(sndpkt)
rdt_rvc(rcvpkt) && notcorrupt(rcvpkt) && has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
compute chksum
make_pkt(sendpkt,ACK,chksum)
udt_send(sndpkt)

ANSWER:
Consider the states of sender and receiver as follows:

Sender state as "wait for call 1 from above".

Receiver state as "wait for 1 from below".

- Initially sender sends a packet with sequence number 1.

- After sending the packet, the state of the sender will be “wait for ACK or NAK 1” (indicates that sender waits for acknowledgement ACK or NAK from receiver).
- The receiver receives the packet 1 and sends the ACK.
- After acknowledging the packet, the state of the receiver is “wait for 0 from below” (indicates that the receiver is waiting for packet 0).
- Assume that sender receives the corrupted ACK sent by the receiver for packet 1.
- Since the sender received the corrupted ACK, it transmits the packet 1 again.
- The receiver sends a NAK as it received packet with sequence number 1 again instead of packet with sequence number 0.
- The sender sends the packet with sequence number 1 again.
- The receiver sends NAK again to sender.

Therefore, the sender repeats the sending packet with sequence number 1 and the receiver sends NAK again to sender. This raises a deadlock stage for both sender and the receiver.

7. In protocol *rdt3.0,* the ACK packets flowing from the receiver to the sender do not have sequence numbers (although they do have an ACK field that contains the sequence number of the packet they are acknowledging). Why is it that our ACK packets do not require sequence numbers?

ANSWER:
The protocol rtd3.0 is used to transfer data from sender to receiver.

- If a sender transfers the packet to the receiver, then receiver will receive and send ACK(Acknowledgement) to the sender for conformation.
- If sender received ACK then go to the next level.
- In this process, needs sequence number to the sender for finding duplicate packets data or ACK data.
- If the sender finds any duplicate ACK, then ignore it. In this process ACK packets does not require sequence number.

So, ACK packets does not require sequence numbers.

8. Draw the FSM for the receiver side of protocol *rdt3.0*.

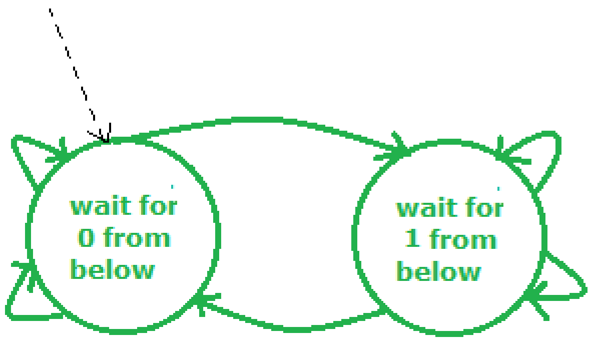
ANSWER:
To transfer the data over a channel, rdt3.0 protocol is useful. It is a reliable protocol.

The sender transmits the packet and the receiver acknowledges it by sending an ACK and confirms the packet is received.

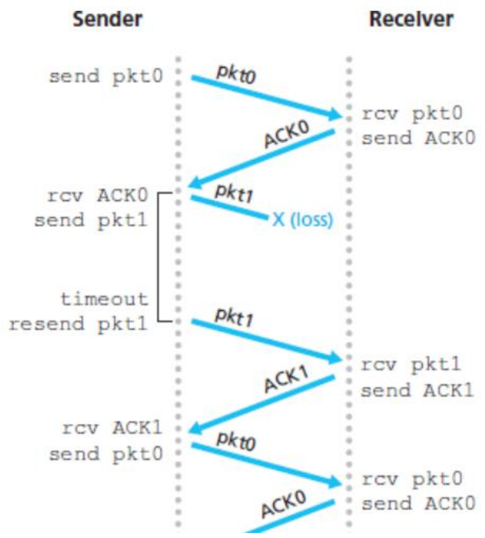
The rdt3.0 protocol allows duplicate packets into the sender-to-receiver data stream by adding timeout. This is not possible in rdt2.0.

The rdt3.0 and rdt2.0 works in the same way at receiver end. Both handles duplicate packets.

FSM diagram for the receiver side of the protocol rdt3.0 is follows:



9. Give a trace of the operation of protocol rdt3.0 when data packets and acknowledgment packets are garbled. Your trace should be similar to that used in Figure .



ANSWER:

The following figures illustrates the tracing of the rdt3.0 protocol when data packets or the acknowledge (ACK) packets are corrupted.

The sender's state is "wait for call from above" and the receiver's state is "wait for 0 from below".

Figure illustrating data sent by the sender is corrupted:

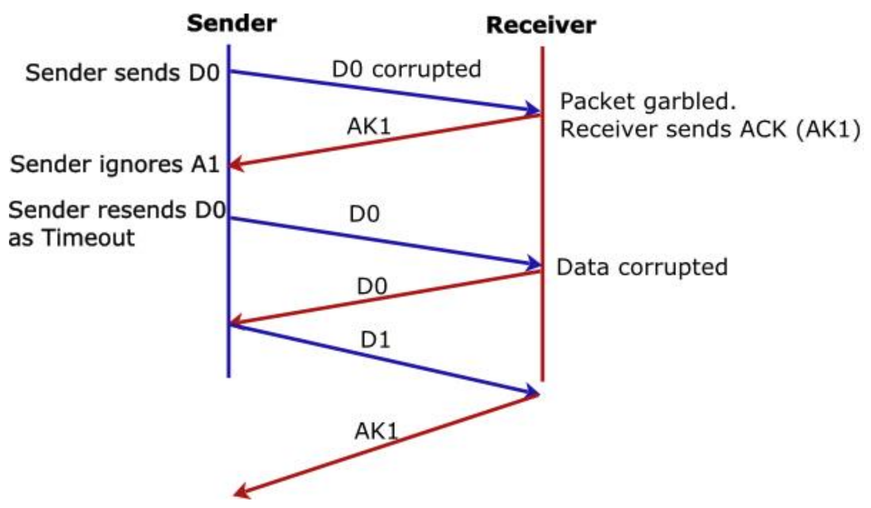
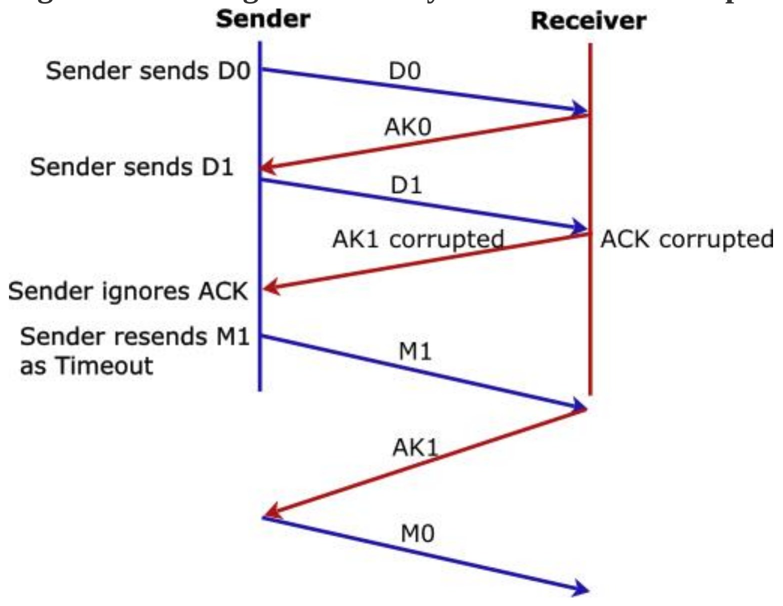
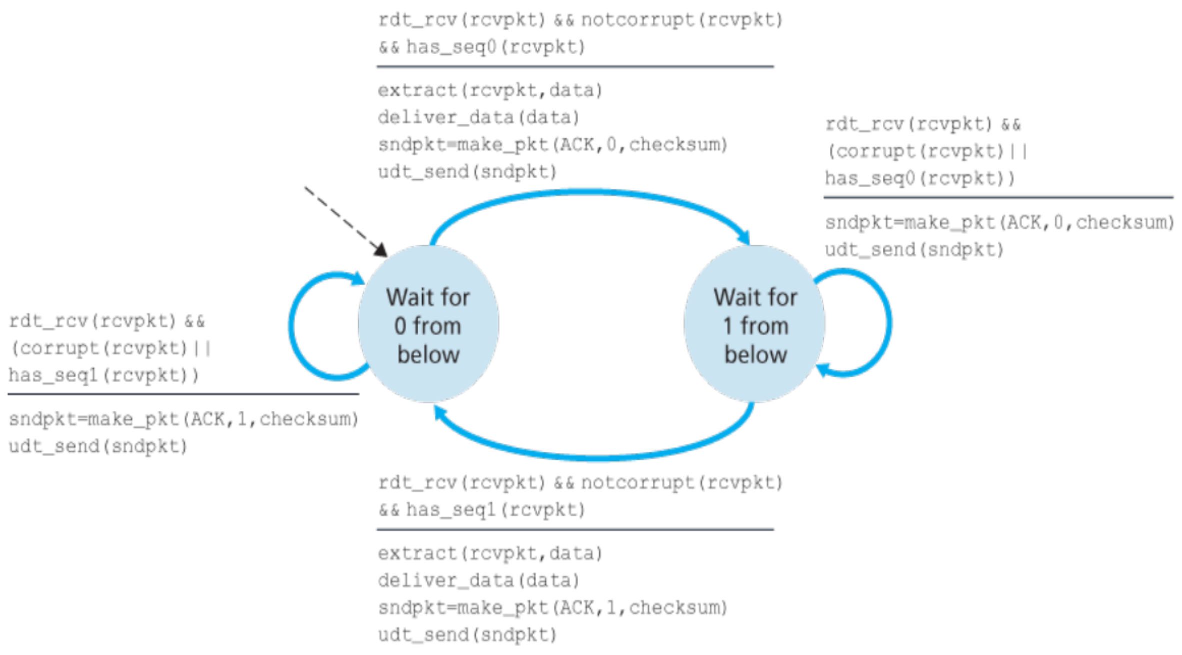


Figure illustrating ACK sent by the receiver is corrupted:



11. Consider the rdt2.2 receiver in Figure, and the creation of a new packet in the self-transition (i.e., the transition from the state back to itself) in the Wait- for-0-from-below and the Wait-for-1-from-below states: sndpkt=make_ pkt(ACK,0,checksum) and sndpkt=make_pkt(ACK,0, checksum). Would the protocol work correctly if this action were removed from the self-transition in the Wait-for-1-from-below state? Justify your answer. What if this event were removed from the self-transition in the Wait- for-0-from-below state? [Hint: In this latter case, consider what would happen if the first sender-to-receiver packet were corrupted.]


rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq0(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK,0,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)|| has_seq0(rcvpkt))
sndpkt=make_pkt(ACK,0,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)|| has_seq1(rcvpkt))
sndpkt=make_pkt(ACK,1,checksum)
udt_send(sndpkt)
Wait for 0 from below
Wait for 1 from below
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK,1,checksum)
udt_send(sndpkt)

ANSWER:
If the sending of this message were removed, the sending and receiving sides would deadlock, waiting for an event that would never occur. Here's a scenario:
- Sender sends pkt0, enter the "Wait for ACK0 state", and waits for a packet back from the receiver
- Receiver is in the "Wait for 0 from below" state, and receives a corrupted packet
from the sender. Suppose it does not send anything back, and simply re-enters the 'wait for 0 from below" state.
Now, the ender is awaiting an ACK of some sort from the receiver, and the receiver is waiting for a data packet form the sender – a deadlock!

12. The sender side of rdt3.0 simply ignores (that is, takes no action on) all received packets that are either in error or have the wrong value in the acknum field of an acknowledgment packet. Suppose that in such circum- stances, rdt3.0 were simply to retransmit the current data packet. Would the protocol still work? (Hint: Consider what would happen if there were only bit errors; there are no packet losses but premature timeouts can occur. Consider how many times the nth packet is sent, in the limit as n approaches infinity.)

ANSWER:
The protocol rtd3.0 is used to transfer data from sender to receiver.

- If a sender transfers the packet to the receiver, then receiver will receive and send ACK(Acknowledgement) to the sender for conformation.
- If the receiver receives a packet where bits are not in order or error has occurred then it will not send acknowledgment.
- The sender will retransmit packet after time out.
- If a packet is send several times, then protocol might look inefficiency.

- The reason is that, it has on trying to send a certain packet and the remaining uninterrupted waiting packets have to wait until the certain packet is delivered.

So, it is better to allow for premature timeouts to occur to avoid this problem.

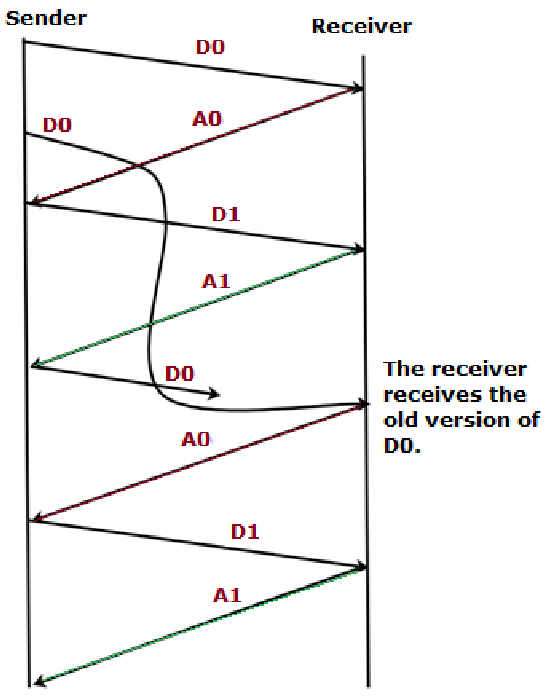
13. Consider the rdt 3.0 protocol. Draw a diagram showing that if the net-work connection between the sender and receiver can reorder messages (that is, that two messages propagating in the medium between the sender and receiver can be reordered), then the alternating-bit protocol will not work correctly (make sure you clearly identify the sense in which it will not work correctly). Your diagram should have the sender on the left and the receiver on the right, with the time axis running down the page, showing data (D) and acknowledgment (A) message exchange. Make sure you indicate the sequence number associated with any data or acknowledgment segment.

ANSWER:

rdt 3.0 protocol's message exchange procedure:

rdt 3.0 stands for reliable data transfer protocol. The following are the steps involved in message exchange.

The following diagram represents the sender and receiver connected through a network. The messages reordering is illustrated:



In the above diagram D0, D1 indicate data with sequence number 0 and 1 respectively. A0, A1, indicates acknowledgements with sequence number 0 and 1 respectively.

Step by step process of message exchange:

- The sender sends the data D0 to the receiver expects an acknowledgment from receiver. So, the sender waits for the acknowledgment for a specific period of time.

- Since the sender does not received the acknowledgement from the receiver, it assumes that the data D0 is not received by the receiver. So, the sender sends D0 again.
- After sending the data D0 for the second time, the sender receives the acknowledgement A0 from the receiver, it sends the data D1.
- The receiver receives the data D1 and sends the acknowledgement A1 to the sender.
- The receiver receives the retransmitted data D0 and sends the acknowledgement A0 again to the sender assuming that the previous acknowledgement is not received by the sender. So, the old version of data D0 is acknowledged by the receiver.
- The sender repeats the data D1 transmission and the receiver sends the acknowledgement A1 to the sender.

The new version of the data D0 received by the receiver is replaced by the old version of the data D0 due to the reordering.

Therefore, the protocol is not working well and message exchange is improper. Data is not transmitted as intended.

14. Consider a reliable data transfer protocol that uses only negative acknowledg- ments. Suppose the sender sends data only infrequently. Would a NAK-only protocol be preferable to a protocol that uses ACKs? Why? Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?

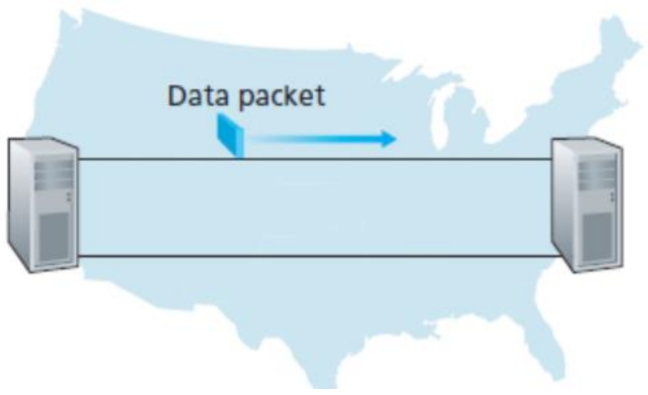
ANSWER:
If the sender sends the data infrequently (occasionally), then the NAK-only protocol is not preferred. It is good to use the protocol that uses ACKs.

- The main disadvantage of NAK-only protocol is that it can detect that the packet has been lost only when the next packet is received by the receiver.
- The NAK-only protocol realize the loss of packet after a long time when it receives the data packet with wrong sequence number as the sender sends the data packets occasionally.
- When the receiver realizes the packet loss, it sends NAK to the sender and the sender has to retransmit both the lost and next packet.
- If the sender sends the data frequently, then the NAK-only protocol is preferred. The protocol that uses ACKs is not preferred as it require to send more number of acknowledgements.

When the sender sends data frequently and the data loss rate is very less, then a NAK-only protocol is preferable to a protocol that uses ACKs.

- The receiver sends the NAK as it realizes the packet loss quickly as the data transmitted frequently.
- Since the data loss is less, the number of NAK in NAK-only protocol is less when compared with protocol that uses ACKs.

15. Consider the cross-country example shown in Figure . How big would the window size have to be for the channel utilization to be greater than 98 percent? Suppose that the size of a packet is 1,500 bytes, including both header fields and data.



ANSWER:

Consider two systems A and B.
The round-trip propagation delay between A and B (*RTT*) = 30 ms.
Transmission rate of the link between A and B (*R*) = 1Gbps = 10^9bps.
The size of the data packet (*L*)= 1500 bytes (1500x8 bits)

The following is the formula to calculate the time required to transmit the packet over the 1 Gbps link:

$$\begin{aligned} D_{\text{trans}} &= \frac{L}{R} \\ &= \frac{1500\times 8\,\text{bits/packet}}{10^9\,\text{bits/sec}} \\ &= 12\text{ms} \\ &= 0.012\ \mu\text{s} \end{aligned}$$

where D_{trans} is the time required.
Calculate the window size such that the channel utilization is greater than 98%.

$$\begin{aligned} \text{ChanelUtilisation} &= N \times \frac{\frac{L}{R}}{\frac{L}{R} + RTT} \\ \frac{98}{100} &= N \times \frac{0.012}{30 + 0.012} \\ 0.98 \times 30.012 &= N \times 0.012 \\ N &= \frac{29.41176}{0.012} \\ N &= 2450.98 \end{aligned}$$

Therefore, to utilize 98% of the channel, the window size should be 2451 packets approximately.

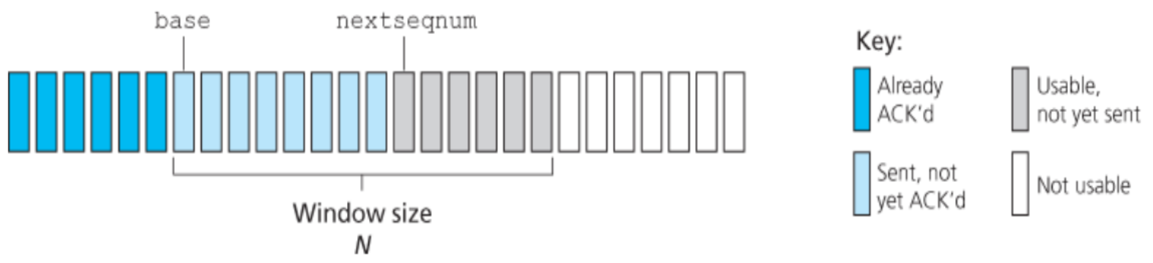
Chapter 3: TRANSPORT LAYER

16.Suppose an application uses rdt 3.0 as its transport layer protocol. As the stop- and-wait protocol has very low channel utilization (shown in the cross- country example), the designers of this application let the receiver keep sending back a number (more than two) of alternating ACK 0 and ACK 1 even if the corresponding data have not arrived at the receiver. Would this application design increase the channel utilization? Why? Are there any potential problems with this approach? Explain.

ANSWER:

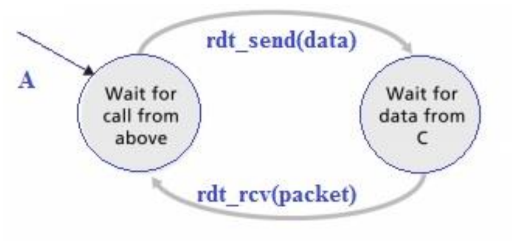
- Suppose an application uses rdt 3.0 as its transport layer protocol. Then It stop-and-wait protocol is used very low utilization in the application.
- The application allows the receiver to send the acknowledgements and sends the next data packet. It is used as a pipelined data in the channel.
- There is a chance of missing data before it reaches the receiver.
- So the sender (who is using rtd 3.0 protocol) will not retransmit the data. The missing or lost some data.
- Thus, designing of the application need to adopt some mechanism to over come this problem.

17. Consider two network entities, A and B, which are connected by a perfect bi-directional channel (i.e., any message sent will be received correctly; the channel will not corrupt, lose, or re-order packets). A and B are to deliver data messages to each other in an alternating manner: First, A must deliver a message to B, then B must deliver a message to A, then A must deliver a mes- sage to B and so on. If an entity is in a state where it should not attempt to deliver a message to the other side, and there is an event like rdt_send(data) call from above that attempts to pass data down for transmission to the other side, this call from above can simply be ignored with a call to rdt_unable_to_send(data), which informs the higher layer that it is currently not able to send data. [Note: This simplifying assumption is made so you don't have to worry about buffering data.]. Draw a FSM specification for this protocol (one FSM for A, and one FSM for B!). Note that you do not have to worry about a reliability mechanism here; the main point of this question is to create a FSM specification that reflects the synchronized behavior of the two entities. You should use the following events and actions that have the same meaning as protocol rdt1.0 in Figure : rdt_send(data), packet = make_pkt(data), udt_send(packet), rdt_rcv(packet), extract (packet,data), deliver_data(data). Make sure your protocol reflects the strict alternation of sending between A and B. Also, make sure to indicate the initial states for A and B in your FSM descriptions.



ANSWER:

FSM specification for this protocol (one FSM for A, and one FSM for B!):

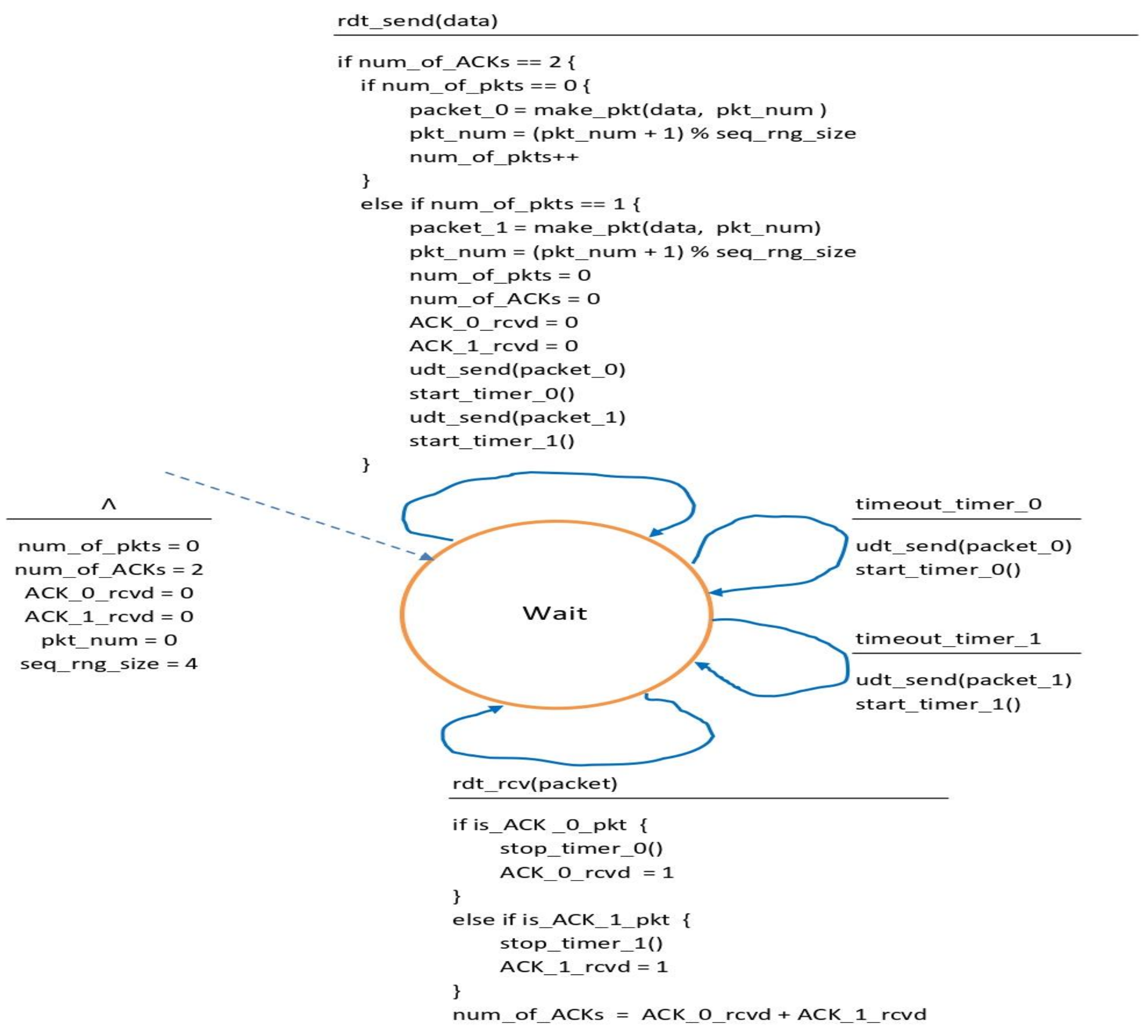


18. **In the generic SR protocol that the sender transmits a message as soon as it is available (if it is in the window) without waiting for an acknowledgment. Suppose now that we want an SR protocol that sends messages two at a time. That is, the sender will send a pair of messages and will send the next pair of messages only when it knows that both messages in the first pair have been received correctly. Suppose that the channel may lose messages but will not corrupt or reorder messages. Design an error- control protocol for the unidirectional reliable transfer of messages. Give an FSM description of the sender and receiver. Describe the format of the packets sent between sender and receiver, and vice versa. If you use any procedure calls other than those in Section 3.4 (for example, u*dt_send()*, *start_timer()*, *rdt_rcv()*, and so on), clearly state their actions. Give an example (a timeline trace of sender and receiver) showing how your protocol recovers from a lost packet.**

ANSWER:

Extended FSMs for Sender and Receiver, Descriptions of packets' format and Timeline are shown below

Sender (extended FSM)



```
rdt_send(data)
--------------------------------------------------
if num_of_ACKs == 2 {
   if num_of_pkts == 0 {
         packet_0 = make_pkt(data,  pkt_num )
         pkt_num = (pkt_num + 1) % seq_rng_size
         num_of_pkts++
   }
   else if num_of_pkts == 1 {
         packet_1 = make_pkt(data,  pkt_num)
         pkt_num = (pkt_num + 1) % seq_rng_size
         num_of_pkts = 0
         num_of_ACKs = 0
         ACK_0_rcvd = 0
         ACK_1_rcvd = 0
         udt_send(packet_0)
         start_timer_0()
         udt_send(packet_1)
         start_timer_1()
   }
```

```
Λ
--------------------
num_of_pkts = 0
num_of_ACKs = 2
ACK_0_rcvd = 0
ACK_1_rcvd = 0
pkt_num = 0
seq_rng_size = 4
```

```
timeout_timer_0
--------------------
udt_send(packet_0)
start_timer_0()
```

```
timeout_timer_1
--------------------
udt_send(packet_1)
start_timer_1()
```

```
rdt_rcv(packet)
--------------------------------------------------
if is_ACK _0_pkt  {
      stop_timer_0()
      ACK_0_rcvd  = 1
}
else if is_ACK_1_pkt  {
      stop_timer_1()
      ACK_1_rcvd = 1
}
num_of_ACKs  =  ACK_0_rcvd + ACK_1_rcvd
```

Receiver (extended FSM)

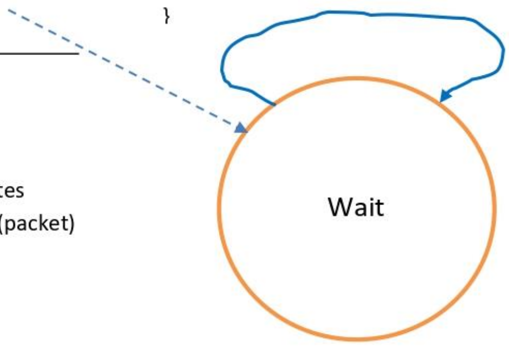
```
rdt_rcv(packet)
________________________________________________
pkt_num = extract_pkt_num(packet)

if (pkt_num == expected_pkt_num) {
    extract(packet, data)
    deliver_data(data)
    expected_pkt_num = (expected_pkt_num + 1) % seq_rng_size

    if pkt_was_buffered == 1 {
        extract_data(buffered_pkt, data)
        deliver_data(data)
        pkt_was_buffered = 0
        expected_pkt_num = (expected_pkt_num + 1) % seq_rng_size
    }
}
else if (pkt_num == expected_pkt_num + 1) && pkt_was_buffered == 0 {
    pkt_was_buffered = 1
    buffered_pkt = packet
}

if (pkt_num % 2) {
    ACK_1_pkt = makepkt(ACK, 1)
    udt_send(ACK_1_pkt)
}
else {
    ACK_0_pkt = makepkt(ACK, 0)
    udt_send(ACK_0_pkt)
}
```

```
              Λ
________________________________
seq_rng_size = 4
expected_pkt_num = 0
packet_was_buffered = 0
buffered_pkt   – array of bytes
                 with sizeof(packet)
```



Description of the packets' format

Packets' sequence numbers range = {0, 1, 2, 3} (four numbers are required so that the receiver be able to distinguish between new packets and the packets resent because of lost ACKs as only two packets can be sent consecutively before being ACKed. The number of consecutively sent packets – 2 – can be seen as an equivalent of *sending window size* in Selective Repeat protocol. Thus a range of packets' sequence numbers has to be at least twice as large as sending window, so 2 * 2 = 4 sequence numbers are required)

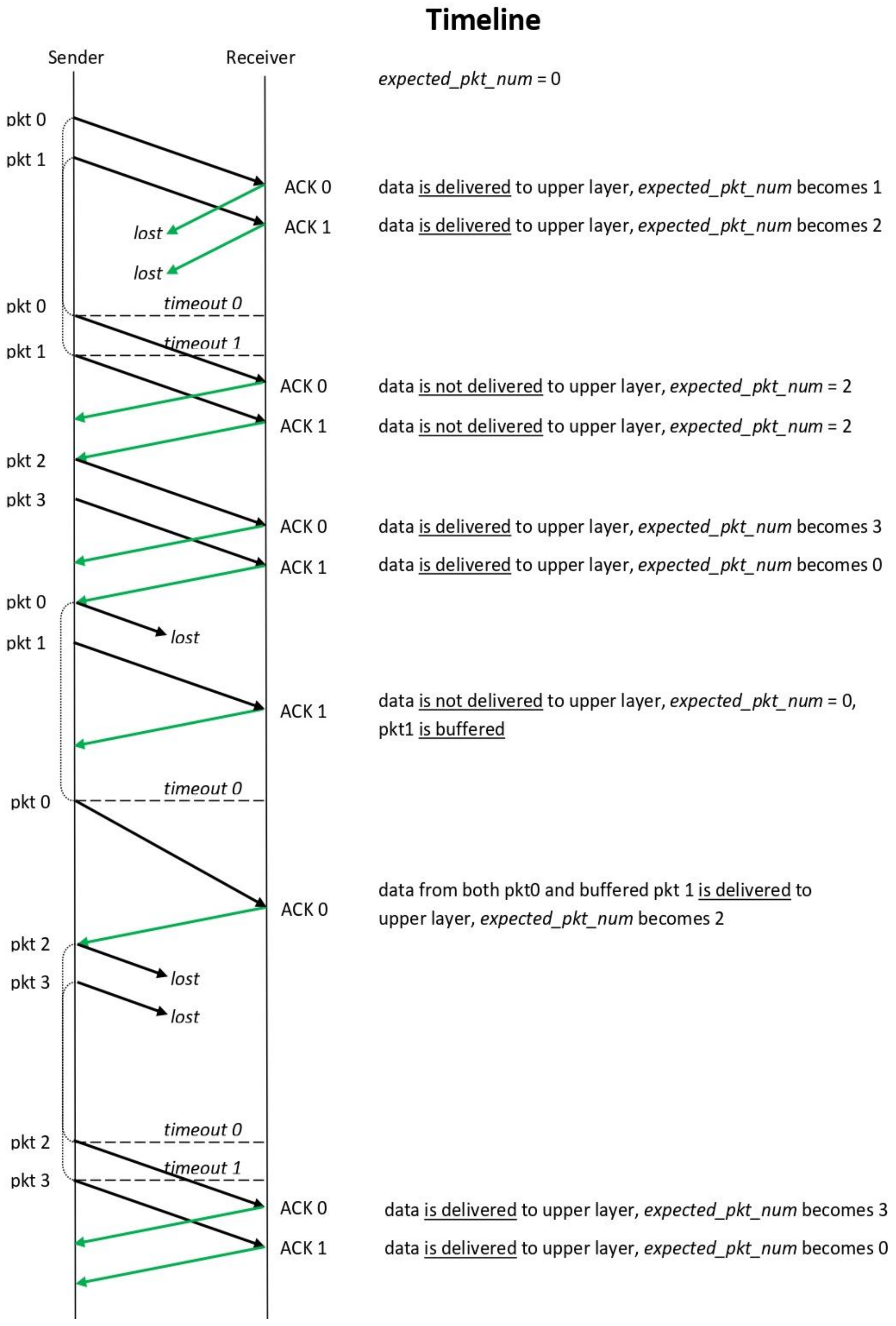
As only two packets can be sent consecutively before being ACKed, range of ACK packets sequence numbers = {0, 1}:

- If a packet being received has even number (0 or 2), corresponding ACK packet should have number 0.
- If a packet being received has odd number (1 or 3), corresponding ACK packet should have number 1.

<u>Sender to receiver packet format:</u>
pkt_num (0 or 1 or 2 or 3) : data

<u>Receiver to sender packet format:</u>
pkt_num (0 or 1) : ACK

*names of the pseudo-procedures in extended FSM's are self-explanatory


Timeline
Sender
Receiver
expected_pkt_num = 0
pkt 0
pkt 1
ACK 0
data is delivered to upper layer, expected_pkt_num becomes 1
ACK 1
data is delivered to upper layer, expected_pkt_num becomes 2
lost
lost
pkt 0
timeout 0
pkt 1
timeout 1
ACK 0
data is not delivered to upper layer, expected_pkt_num = 2
ACK 1
data is not delivered to upper layer, expected_pkt_num = 2
pkt 2
pkt 3
ACK 0
data is delivered to upper layer, expected_pkt_num becomes 3
ACK 1
data is delivered to upper layer, expected_pkt_num becomes 0
pkt 0
lost
pkt 1
ACK 1
data is not delivered to upper layer, expected_pkt_num = 0, pkt1 is buffered
timeout 0
pkt 0
ACK 0
data from both pkt0 and buffered pkt 1 is delivered to upper layer, expected_pkt_num becomes 2
pkt 2
lost
pkt 3
lost
timeout 0
pkt 2
timeout 1
pkt 3
ACK 0
data is delivered to upper layer, expected_pkt_num becomes 3
ACK 1
data is delivered to upper layer, expected_pkt_num becomes 0

19. Consider a scenario in which Host A wants to simultaneously send packets to Hosts B and C. A is connected to B and C via a broadcast channel—a packet sent by A is carried by the channel to both B and C. Suppose that the broad- cast channel connecting A, B, and C can independently lose and corrupt packets (and so, for example, a packet sent from A might be correctly received by B, but not by C). Design a stop-and-wait-like error-control protocol for reliably transferring packets from A to B and C, such that A will not get new data from the upper layer until it knows that both B and C have correctly received the current packet. Give FSM descriptions of A and C. (Hint: The FSM for B should be essentially the same as for C.) Also, give a description of the packet format(s) used.

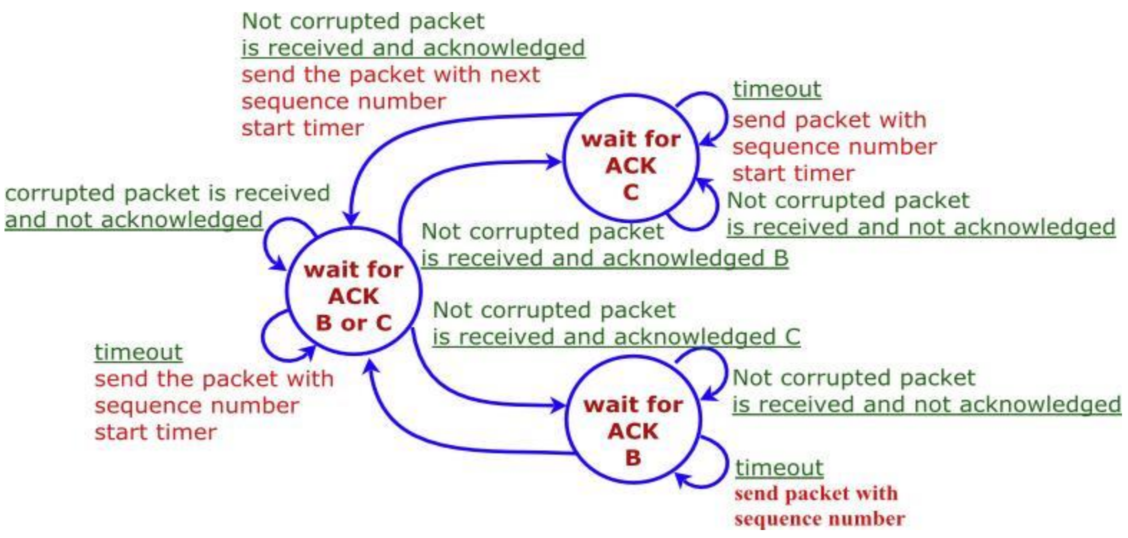
ANSWER:

Description of Stop-and-wait like error-control protocol:

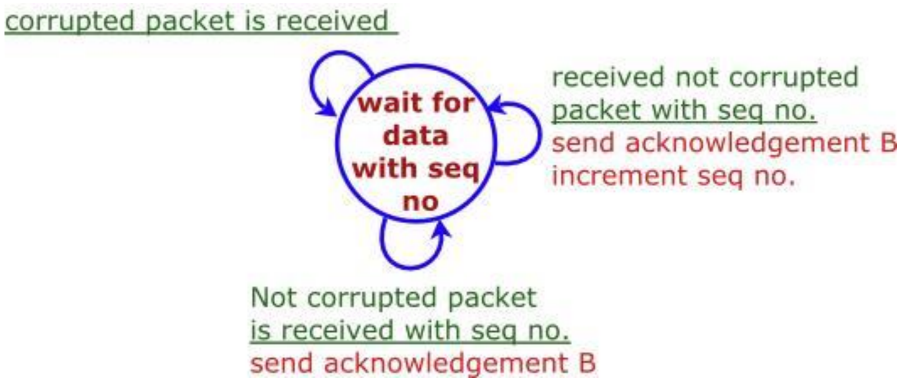
- While transmitting the message over the channel, the message may be lost in Stop-and-wait like error-control protocol.
- It is required to use the sequence numbers for the packets, to detect and retransmit the lost packets.
- In this protocol, it is also possible that the sender sends the lost messages again and again that are received by the receiver.

Following is the Finite State Machines (FSM) of sender and receiver:

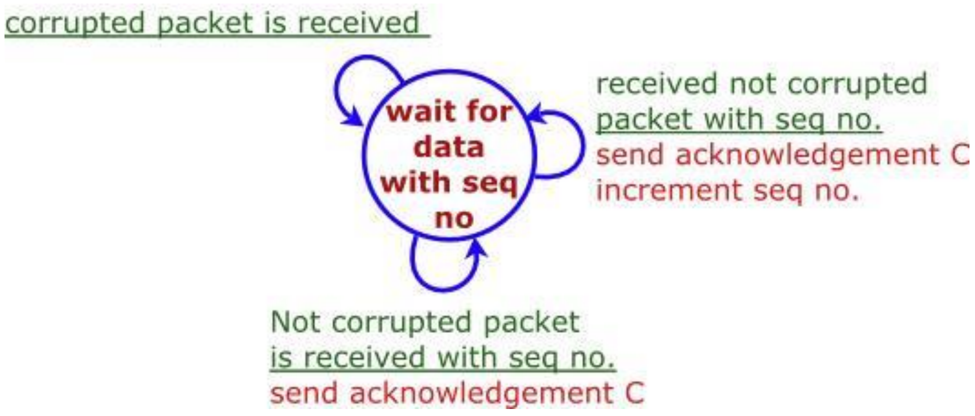
FSM of the sender:



FSM of the receiver B:


corrupted packet is received
wait for data with seq no
received not corrupted packet with seq no.
send acknowledgement B
increment seq no.
Not corrupted packet is received with seq no.
send acknowledgement B

FSM of the receiver C:


corrupted packet is received
wait for data with seq no
received not corrupted packet with seq no.
send acknowledgement C
increment seq no.
Not corrupted packet is received with seq no.
send acknowledgement C

Sender to Receiver data packet format:

Seqnum	data

Receiver to Sender acknowledgement format:

Acknum	B|C

Flow of the Sender FSM:

The data in the sender FSM (Finite state machine) changes the states as follows:

- Initialize the session and start the sequence number with 0.
- Build the packet with data and sequence number.
- Start the timer and broadcast the packet to hosts in the channel.
- Wait for some time for acknowledgement.
- If acknowledgement is not received in sepecified time, then broadcast the packet to all hosts in the channel.

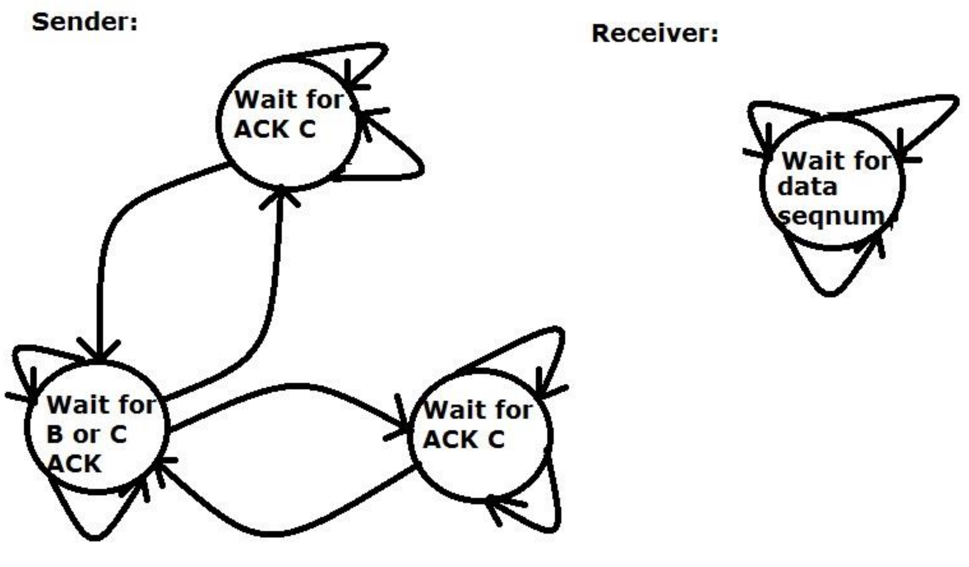
Flow of the Receivers FSM:

The acknowledgement in the receiver FSM (Finite state machine) changes the states as follows:

- Initialize the session and start the acknowledgement number with 0.
- Wait to receive the uncorrupted data packet from all hosts with sequence number same as acknowledgement number.
- If packet's sequence number received do not match with the acknowledgement number, then lose the acknowledgement and make the packet with acknowledgement number 1.
- If packet's sequence number received match with the acknowledgement number, send acknowledgement number to the sender.
- Extract and process the data packet.

20. Consider a scenario in which Host A and Host B want to send messages to Host C. Hosts A and C are connected by a channel that can lose and corrupt (but not reorder) messages. Hosts B and C are connected by another channel (independent of the channel connecting A and C) with the same properties. The transport layer at Host C should alternate in delivering messages from A and B to the layer above (that is, it should first deliver the data from a packet from A, then the data from a packet from B, and so on). Design a stop-and- wait-like error-control protocol for reliably transferring packets from A and B to C, with alternating delivery at C as described above. Give FSM descriptions of A and C. (Hint: The FSM for B should be essentially the same as for A.) Also, give a description of the packet format(s) used.

ANSWER:
The sender sent a message to many times to the user. In this situation. the channel has lose message if previously received message already.
The following stop-and- wait-like error-control protocol for reliably transferring packets from A and B to C, with alternating delivery at C. The sender received ACK from B and C (or) neither C nor B.



21. Suppose we have two network entities, A and B. B has a supply of data mes- sages that will be sent to A according to the following conventions. When A gets a request from the layer above to get the next data (D) message from B, A must send a request (R) message to B on the A-to-B channel. Only when B receives an R message can it send a data (D) message back to A on the B-to- A channel. A should deliver exactly one copy of each D message to the layer above. R messages can be lost (but not corrupted) in the A-to-B channel; D messages, once sent, are always delivered correctly. The delay along both channels is unknown and variable.Design (give an FSM description of) a protocol that incorporates the appro- priate mechanisms to compensate for the loss-prone A-to-B channel and implements message passing to the layer above at entity A, as discussed above. Use only those mechanisms that are absolutely necessary.

ANSWER:
Because the A-to-B channel can lose request messages, A will need to timeout and retransmit its request messages (to be able to recover from loss). Because the channel delays are variable and unknown, it is possible that A will send duplicate requests (i.e., resend a request message that has already been received by B). To be able to detect duplicate request messages, the protocol will use sequence numbers. A 1-bit sequence number will suffice for a stop-and-wait type of request/response protocol.
A (the requestor) has 4 states:
1- “Wait for Request 0 from above.” Here the requestor is waiting for a call from above to request a unit of data. When it receives a request from above, it sends a request message, R0, to B, starts a timer and makes a transition to the “Wait for D0” state. When in the “Wait for Request 0 from above” state, A ignores anything it receives from B.
2- “Wait for D0”. Here the requestor is waiting for a D0 data message from B. A timer is always running in this state. If the timer expires, A sends another R0 message, restarts the timer and remains in this state. If a D0 message is received from B, A stops the time and transits to the “Wait for Request 1 from above” state. If A receives a D1 data message while in this state, it is ignored.
3- “Wait for Request 1 from above.” Here the requestor is again waiting for a call from above to request a unit of data. When it receives a request from above, it sends a request message, R1, to B, starts a timer and makes a transition to the “Wait for D1” state. When in the “Wait for Request 1 from above” state, A ignores anything it receives from B.
4- “Wait for D1”. Here the requestor is waiting for a D1 data message from B. A timer is always running in this state. If the timer expires, A sends another R1 message, restarts the timer and remains in this state. If a D1 message is received from B, A stops the timer and transits to the “Wait for Request 0 from above” state. If A receives a D0 data message while in this state, it is ignored.
The data supplier (B) has only two states:
1- “Send D0.” In this state, B continues to respond to received R0 messages by sending D0, and then remaining in this state. If B receives a R1 message, then it knows its D0 message has been received correctly. It thus discards this D0 data (since it has been received at the other side) and then transits to the “Send D1” state, where it will use D1 to send the next

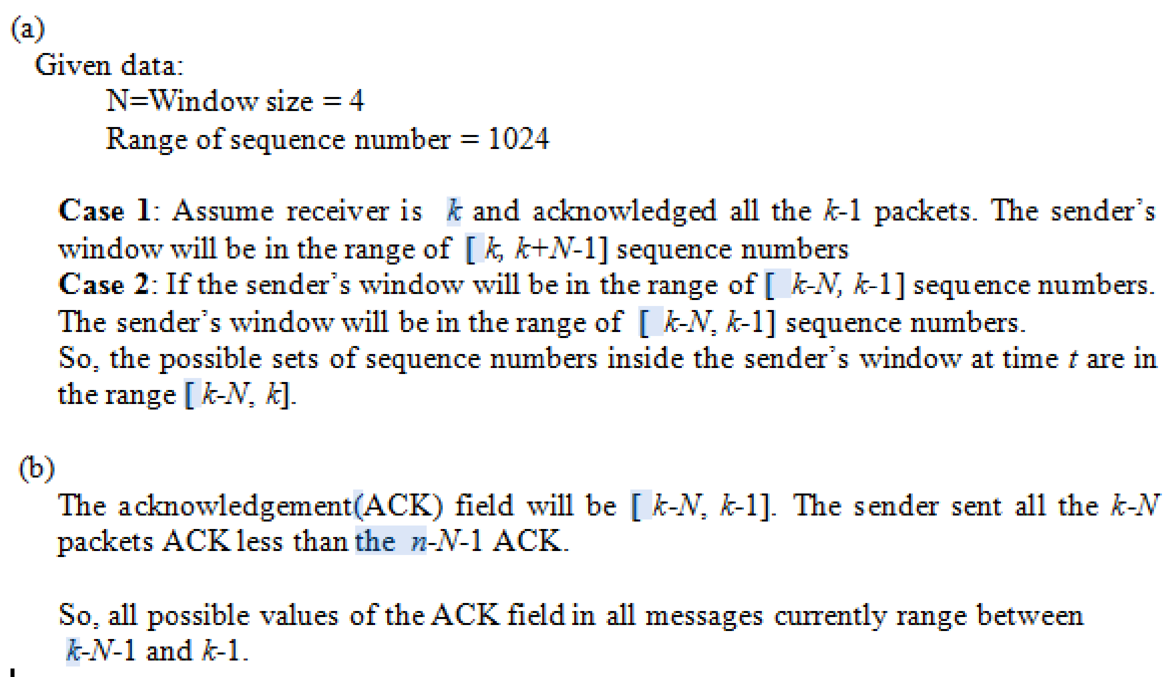
requested piece of data.
2- “Send D1.” In this state, B continues to respond to received R1 messages by sending D1, and then remaining in this state. If B receives a R1 message, then it knows its D1 message has been received correctly and thus transits to the “Send D1” state.

22. Consider the GBN protocol with a sender window size of 4 and a sequence number range of 1,024. Suppose that at time t, the next in-order packet that the receiver is expecting has a sequence number of k. Assume that the medium does not reorder messages.
Answer the following questions:

a) What are the possible sets of sequence numbers inside the sender’s window at time t? Justify your answer.

b) What are all possible values of the ACK field in all possible messages currently propagating back to the sender at time t? Justify your answer.

ANSWER:



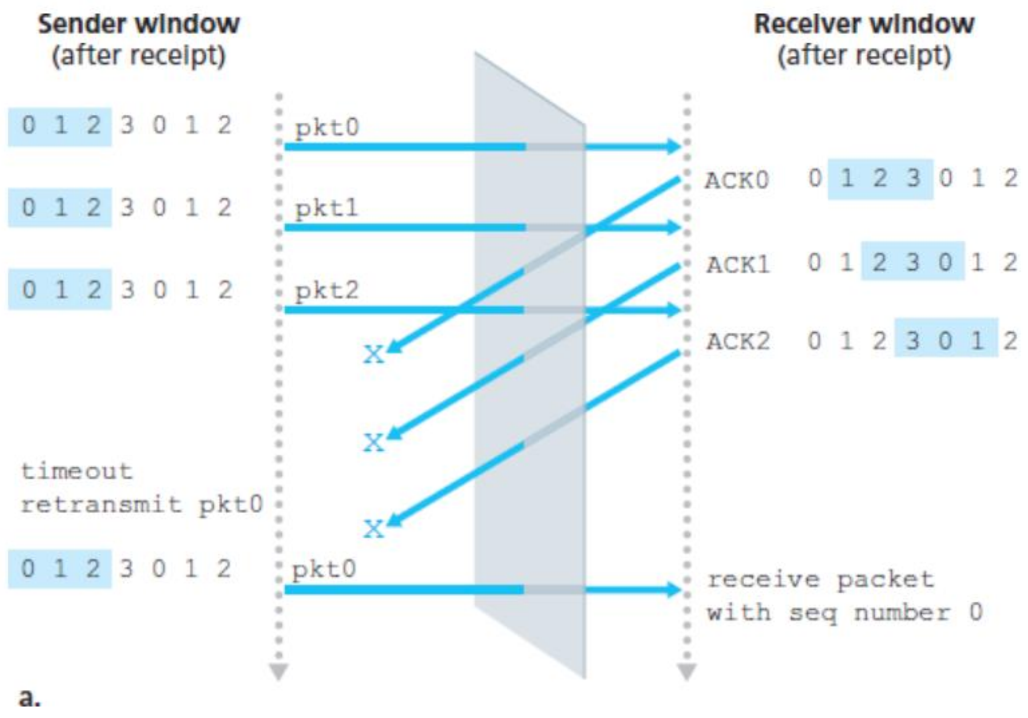
(a)
Given data:
N=Window size = 4
Range of sequence number = 1024

Case 1: Assume receiver is k and acknowledged all the k-1 packets. The sender’s window will be in the range of [k, $k+N$-1] sequence numbers
Case 2: If the sender’s window will be in the range of [k-N, k-1] sequence numbers. The sender’s window will be in the range of [k-N, k-1] sequence numbers.
So, the possible sets of sequence numbers inside the sender’s window at time t are in the range [k-N, k].

(b)
The acknowledgement(ACK) field will be [k-N, k-1]. The sender sent all the k-N packets ACK less than the n-N-1 ACK.

So, all possible values of the ACK field in all messages currently range between k-N-1 and k-1.

23. Consider the GBN and SR protocols. Suppose the sequence number space is of size k. What is the largest allowable sender window that will avoid the occurrence of problems such as that in Figure for each of these protocols?



Sender window (after receipt)
Receiver window (after receipt)
0 1 2 3 0 1 2
pkt0
pkt1
pkt2
ACK0
ACK1
ACK2
X
timeout
retransmit pkt0
receive packet with seq number 0
a.

ANSWER:

In order to avoid the scenario of Figure, we want to avoid having the leading edge of the receiver's window (i.e., the one with the “highest” sequence number) wrap around in the sequence number space and overlap with the trailing edge (the one with the "lowest" sequence number in the sender's window). That is, the sequence number space must be large enough to fit the entire receiver window and the entire sender window without this overlap condition. So, we need to determine how large a range of sequence numbers can be covered, at any given time, by the receiver and sender windows.

Suppose that the lowest-sequence number that the receiver is waiting for is packet m. In this case, it’s window [m, m+w-1] and it has received (and ACKed) packet m-1 and the w-1 packets before that, where w is the size of the window. If none of those w ACKs have been yet received by the sender, then ACK messages with values of [m-w,m-1] may still be propagating back. If no ACKs with these ACK numbers have been received by the sender, then the sender's window would be [m-w, m-1].

Thus, the lower edge of the sender's window is m-w, and the leading edge of the receiver’s window is m+w-1. In order for the leading edge of the receiver's window to not overlap with the trailing edge of the sender's window, the sequence number space must thus be big enough to accommodate 2w sequence numbers. That is, the sequence number space must be at least twice as large as the window size

24. Answer true or false to the following questions and briefly justify your answer:

- h) **With the SR protocol, it is possible for the sender to receive an ACK for a packet that falls outside of its current window. True**
- i) **With GBN, it is possible for the sender to receive an ACK for a packet that falls outside of its current window. True**
- j) **The alternating-bit protocol is the same as the SR protocol with a sender and receiver window size of 1. True**
- k) **The alternating-bit protocol is the same as the GBN protocol with a sender and receiver window size of 1. True**

25. We have said that an application may choose UDP for a transport protocol because UDP offers finer application control (than TCP) of what data is sent in a segment and when.

- a) **Why does an application have more control of what data is sent in a segment?**
- b) **Why does an application have more control on when the segment is sent?**

ANSWER:

a)

In TCP protocols:

- If the send the segment application then data is not assigned directly.
- Then data store in buffer then decide to connect the segement.
- So,the application does not have more control to sent data in the TCP protocol.

For UDP protocols:

- If the send the segment application then data is assigned directly
- Then the data in the segment is then sent.
- So, the application has more control to sent data in the UDP protocol.

b)

- In TCP protocol **cannot** expect the time between sent data and arrived data. So, an application **does not** have more control on when the segment is sent in TCP.
- n TCP protocol **can** expect the time between sent data and arrived data. So, an application have more control on when the segment is sent in TCP.

26. Consider transferring an enormous file of L bytes from Host A to Host B. Assume an MSS of 536 bytes.

- a) **What is the maximum value of L such that TCP sequence numbers are not exhausted? Recall that the TCP sequence number field has 4 bytes.**
- b) **For the L you obtain in (a), find how long it takes to transmit the file. Assume that a total of 66 bytes of transport, network, and data-link header are added to each segment before the resulting packet is sent out over a 155 Mbps link. Ignore flow control and congestion control so A can pump out the segments back to back and continuously.**

ANSWER:

a)

The size of TCP sequence number field = 4 bytes

=4*8 bits

=32 bits.

So, the sequence numbers of bits are 2^{32}

The maximum file size sent from Host A to Host B representable by 2^{32}

$= 2^2 \times 2^{30}$bytes

$=2^2$ Gbytes

=4 Gbytes

b)

Maximum segment size (MSS) = 536 bytes

Segments data=$2^{32}/536$

=8012999

Total header fields = 66 bytes.

Total number of bytes through the 155Mbps link $= 8012999 \times 66$ bytes

$= 528857934$ bytes

Transmitted data $= (2^{32} + 528857934)$

$=4.824 \times 10^9$ bytes

$$\text{Transmit time} = \frac{4.824 \times 10^9 \times 8\text{bits}}{155 \times 10^6 \text{ bps}} \approx \boxed{249} \text{ seconds}$$

27. Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 126. Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 80 and 40 bytes of data, respectively. In the first segment, the sequence number is 127, the source port number is 302, and the destination port number is 80. Host B sends an acknowledgment whenever it receives a segment from Host A.

a) **In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number?**

b) **If the first segment arrives before the second segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number, the source port number, and the destination port number?**

c) **If the second segment arrives before the first segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number?**

d) **Suppose the two segments sent by A arrive in order at B. The first acknowledgment is lost and the second acknowledgment arrives after the first time- out interval. Draw a timing diagram, showing these segments and all other segments and acknowledgments sent. (Assume there is no additional packet loss.) For each segment in your figure, provide the sequence number and the number of bytes of data; for each acknowledgment that you add, provide the acknowledgment number.**

ANSWER:

Given data:

- Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 126.
- The first and second segments contain 80 and 40 bytes of data, respectively.
- The first segment of sequence number is 127.
- The source port number is 302.
- The destination port number is 80

a)

Sequence number = first segment of sequence number+ destination port number
=127+80
=207

So, sequence number=207
Source port number = 302
Destination port number= 80

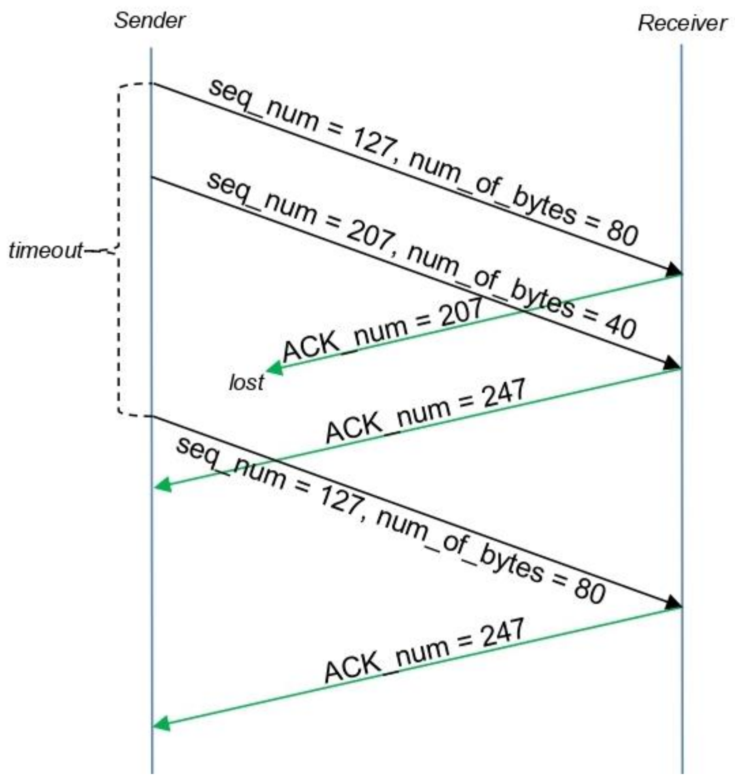
b)

Acknowledgement number= 207
Source port number = 80
Destination port number= 302

c)

Acknowledgement number=127

d) timing diagram


Sender
Receiver
seq_num = 127, num_of_bytes = 80
seq_num = 207, num_of_bytes = 40
timeout
ACK_num = 207
lost
ACK_num = 247
seq_num = 127, num_of_bytes = 80
ACK_num = 247

28. Host A and B are directly connected with a 100 Mbps link. There is one TCP connection between the two hosts, and Host A is sending to Host B an enormous file over this connection. Host A can send its application data into its TCP socket at a rate as high as 120 Mbps but Host B can read out of its TCP receive buffer at a maximum rate of 50 Mbps. Describe the effect of TCP flow control.

ANSWER:
The effect of TCP flow control:
If host A faster tahn host B sending data, then buffer fill up is start. If the buffer filled completely then send data from host B to host A until the remove buffer data and intimate to continue sending data or not.
Then filled again buffer then repeat the process from host A to host B.

29. SYN cookies

a) **Why is it necessary for the server to use a special initial sequence number in the SYNACK?**
b) **Suppose 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?**
c) **Suppose an attacker collects a large amount of initial sequence numbers sent by the server. Can the attacker cause the server to create many fully open connections by sending ACKs with those initial sequence numbers? Why?**

ANSWER:

a) 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) 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.

c) No, the sever can simply add in a time stamp in computing those initial sequence numbers and choose a time to live value for those sequence numbers, and discard expired initial sequence numbers even if the attacker replay them.

30. Consider the network shown in Scenario 2 in Section 3.6.1. Suppose both sending hosts A and B have some fixed timeout values.

d) Argue that increasing the size of the finite buffer of the router might possibly decrease the throughput λ_{out} .

e) Now suppose both hosts dynamically adjust their timeout values (like what TCP does) based on the buffering delay at the router. Would increasing the buffer size help to increase the throughput? Why?

ANSWER:

a) If timeout values are fixed, then the senders may timeout prematurely. Thus, some packets are re-transmitted even they are not lost.

b) If timeout values are estimated (like what TCP does), then increasing the buffer size certainly helps to increase the throughput of that router. But there might be one potential problem. Queuing delay might be very large, similar to what is shown in Scenario 1.

31. Suppose that the five measured Sample RTT values (see Section 3.5.3) are 106 ms, 120 ms, 140 ms, 90 ms, and 115 ms. Compute the Estimated RTT after each of these Sample RTT values is obtained, using a value of α = 0.125 and assuming that the value of Estimated RTT was 100 ms just before the first of these five samples were obtained. Compute also the DevRTT after each sample is obtained, assuming a value of β = 0.25 and assuming the value of DevRTT was 5 ms just before the first of these five samples was obtained. Last, compute the TCP Timeout Interval after each of these samples is obtained.

ANSWER:

Calculate the EstimatedRTT after obtaining the first sample RTT=106*ms*,

*EstimatedRTT = α * SampleRTT+(1- α) * EstimatedRTT*

EstimatedRTT =0.125 * 106 + (1-0.125) * 100

=0.125* 106 + 0.875 * 100

=13.25 + 87.5

=100.75*ms*

Calculate the DevRTT after obtaining the first sample RTT:

DevRTT **=** *β * | SampleRTT- EstimatedRTT|+(1- β)* DevRTT*

=0.25 * |106-100.75| + (1-0.25) *5

=0.25 *5.25 + 0.75 * 5

=1.3125 + 3.75

=5.0625*ms*

Calculate the Timeout Interval after obtaining the first sample RTT:

TimeoutInterval = EstimatedRTT +4 DevRTT*

= 100.75 + 4 *5.0625

=121*ms*

Calculate the EstimatedRTT after obtaining the second sample RTT=120*ms*,

*EstimatedRTT = α * SampleRTT+(1- α) * EstimatedRTT*

EstimatedRTT =0.125 * 120 + (1-0.125) * 100.75
=0.125* 120 + 0.875 * 100.75
=15 + 88.15625
=103.15625*ms*

Calculate the DevRTT after obtaining the second sample RTT:

DevRTT **=** *β * | SampleRTT- EstimatedRTT|+(1- β)* DevRTT*
=0.25 * |120-103.15625| + (1-0.25) *5.0625
=0.25 *16.84 + 0.75 * 5.0625
=4.21 + 3.79
=8*ms*

Calculate the Timeout Interval after obtaining the second sample RTT:

TimeoutInterval = EstimatedRTT +4 DevRTT*
= 103.15 + 4 *8
=135.15*ms*

Calculate the EstimatedRTT after obtaining the third sample RTT=140*ms*:

*EstimatedRTT = α * SampleRTT+(1- α) * EstimatedRTT*

EstimatedRTT =0.125 * 140 + (1-0.125) * 103.15
=0.125* 140 + 0.875 * 103.15
=17.5 +90.26
=107.75*ms*

Calculate the DevRTT after obtaining the third sample RTT:

DevRTT **=** *β * | SampleRTT- EstimatedRTT|+(1- β)* DevRTT*
=0.25 * |140-107.75| + (1-0.25) *8
=0.25 *32.25 + 0.75 * 8
=8.06 + 6
=14.06*ms*

Calculate the Timeout Interval after obtaining the third sample RTT:

TimeoutInterval = EstimatedRTT +4 DevRTT*
= 107.75 + 4 *14.06
=164*ms*

Calculate the EstimatedRTT after obtaining the fourth sample RTT=90*ms*:

*EstimatedRTT = α * SampleRTT+(1- α) * EstimatedRTT*

EstimatedRTT =0.125 * 90 + (1-0.125) * 107.75
=0.125* 90 + 0.875 * 107.75
=11.25 +94.28
=105.53*ms*

Calculate the DevRTT after obtaining the fourth sample RTT:

DevRTT **=** *β * | SampleRTT- EstimatedRTT|+(1- β)* DevRTT*
=0.25 * |90-105.53| + (1-0.25) *14.06
=0.25 *15.53 + 0.75 * 14.06
=3.88 + 10.545

=14.42*ms*

Calculate the Timeout Interval after obtaining the fourth sample RTT:

TimeoutInterval = EstimatedRTT +4 DevRTT*

= 105.53 + 4 *14.42

=163.21*ms*

Calculate the EstimatedRTT after obtaining the fifth sample RTT=115*ms*:

*EstimatedRTT = α * SampleRTT+(1- α) * EstimatedRTT*

EstimatedRTT =0.125 * 115 + (1-0.125) * 105.53

=0.125* 115 + 0.875 * 105.53

=14.375 +92.34

=106.715*ms*

Calculate the DevRTT after obtaining the fifth sample RTT:

DevRTT **=** *β * | SampleRTT- EstimatedRTT|+(1- β)* DevRTT*

=0.25 * |115-106.715| + (1-0.25) *14.42

=0.25 *8.285 + 0.75 * 14.42

=2.07 + 10.815

=12.885*ms*

Calculate the Timeout Interval after obtaining the fifth sample RTT:

TimeoutInterval = EstimatedRTT +4 DevRTT*

= 106.715 + 4 *12.885

=158.255*ms*

31. Suppose that the five measured SampleRTT values (see Section 3.5.3) are 106 ms, 120 ms, 140 ms, 90 ms, and 115 ms. Compute the EstimatedRTT after each of these SampleRTT values is obtained, using a value of œ = 0.125 and assuming that the value of EstimatedRTT was 100 ms just before the first of these five samples were obtained. Compute also the DevRTT after each sample is obtained, assuming a value of β = 0.25 and assuming the value of DevRTT was 5 ms just before the first of these five samples was obtained. Last, compute the TCP TimeoutInterval after each of these samples is obtained.

ANSWER:

Refer the section 3.5.3 for five measured SampleRTT in the text book.

Given data:

SampleRRT (Sample Round Trip time) values are 106 ms, 120 ms, 140 ms, 90 ms and 115 ms.

EstimatedRTT value = 100 ms

DevRTT value = 5 ms

$\alpha = 0.125$

$\beta = 0.25$

Calculation for estimatedRTT (estimated Round Trip time) values for SampleRRT values:

Formula for estimatedRTT:

$$EstimatedRTT = (1 - \alpha) . EstimatedRTT + \alpha . SampleRTT$$

Case 1: Choose SampleRRT value is 106 ms:

$$\begin{aligned} EstimatedRTT_{106} &= (1 - \alpha) \times EstimatedRTT + \alpha \times SampleRTT \\ &= (1 - 0.125) \times 100 + 0.125 \times 106 \\ &= 100.75 \end{aligned}$$

So, EstimatedRTT$_{106}$ **is 100.75**

Case 2: Choose SampleRRT value is 120 ms:

$$\begin{aligned} EstimatedRTT_{120} &= (1 - \alpha) \times EstimatedRTT + \alpha \times SampleRTT \\ &= (1 - 0.125) \times 100.75 + 0.125 \times 120 \\ &= 103.156 \end{aligned}$$

So, EstimatedRTT$_{120}$ is **103.156**

Case 3: Choose SampleRRT value is 140 ms:

$$\begin{aligned} EstimatedRTT_{140} &= (1 - \alpha) \times EstimatedRTT + \alpha \times SampleRTT \\ &= (1 - 0.125) \times 103.156 + 0.125 \times 140 \\ &= 107.761 \end{aligned}$$

So, EstimatedRTT$_{140}$ is **107.761**

Case 4: Choose SampleRRT value is 90 ms:

$$\begin{aligned} EstimatedRTT_{90} &= (1 - \alpha) \times EstimatedRTT + \alpha \times SampleRTT \\ &= (1 - 0.125) \times 107.761 + 0.125 \times 90 \\ &= 105.54 \end{aligned}$$

Hence, EstimatedRTT$_{90}$ is **105.54**

Case 5: Choose SampleRRT value is 115 ms:

$$\begin{aligned} EstimatedRTT_{115} &= (1 - \alpha) \times EstimatedRTT + \alpha \times SampleRTT \\ &= (1 - 0.125) \times 105.54 + 0.125 \times 115 \\ &= 106.722 \end{aligned}$$

Hence, EstimatedRTT$_{115}$ is **106.722**

Calculation for `DevRTT` and `TimeoutInterval` values for `SampleRRT` values:

Formula for `DevRTT`:

$DevRTT = (1 - \beta) . DevRTT + \beta . \mid SampleRTT - EstimatedRTT \mid$

Formula for `TimeoutInterval`:

$TimeoutInterval = EstimatedRTT + 4.DevRTT$

Case 6: Choose SampleRRT value is 106 ms:

$$\begin{aligned} DevRTT_{106} &= (1-\beta)\times DevRTT + \beta\times\mid SampleRTT - EstimatedRTT \mid \\ &= (1-0.25)\times 5 + 0.25\times\mid 106-100.75\mid \quad (\because EstimatedRTT = 100.75 \text{ from Case } 1) \\ &= 5.06 \end{aligned}$$

So, `DevRTT`$_{106}$ is 5.06

$$\begin{aligned} TimeoutInterval_{106} &= EstimatedRTT + 4.DevRTT \\ &= 100.75 + 4\times 5.06 \\ &= 120.99 \end{aligned}$$

So, `TimeoutInterval`$_{106}$ is 120.99.

Case 7: Choose SampleRRT value is 120 ms:

$$\begin{aligned} DevRTT_{120} &= (1-\beta)\times DevRTT + \beta\times\mid SampleRTT - EstimatedRTT \mid \\ &= (1-0.25)\times 5.06 + 0.25\times\mid 120-103.156\mid \quad (\because EstimatedRTT = 103.156 \text{ from Case } 2) \\ &= 8 \end{aligned}$$

So, `DevRTT`$_{120}$ is 8.

$$\begin{aligned} TimeoutInterval_{120} &= EstimatedRTT + 4.DevRTT \\ &= 103.156 + 4\times 8 \\ &= 135.156 \end{aligned}$$

So, `TimeoutInterval`$_{120}$ is 135.156.

Case 8: Choose SampleRRT value is 140 ms:

$$\begin{aligned} DevRTT_{140} &= (1-\beta)\times DevRTT + \beta\times\mid SampleRTT - EstimatedRTT \mid \\ &= (1-0.25)\times 8 + 0.25\times\mid 140-107.761\mid \quad (\because EstimatedRTT = 107.761 \text{ from Case } 3) \\ &= 14.05 \end{aligned}$$

So, `DevRTT`$_{140}$ is 14.05.

$$\begin{aligned} TimeoutInterval_{140} &= EstimatedRTT + 4.DevRTT \\ &= 107.761 + 4\times 14.05 \\ &= 163.961 \end{aligned}$$

Hence, `TimeoutInterval`$_{140}$ is 163.961.

Case 9: Choose SampleRRT value is 90 ms:

$$DevRTT_{90} = (1-\beta) \times DevRTT + \beta \times |\ SampleRTT - EstimatedRTT\ |$$
$$= (1-0.25) \times 14.05 + 0.25 \times |90 - 105.54| \quad (\because EstimatedRTT = 105.54 \text{ from Case } 4)$$
$$= 6.73$$

So, , **$DevRTT_{90}$ is 6.73**.

$$TimeoutInterval_{90} = EstimatedRTT + 4.DevRTT$$
$$= 105.54 + 4 \times 6.73$$
$$= 132.46$$

S0, $TimeoutInterval_{90}$ is 132.46.

Case 10: Choose SampleRRT value is 115 ms:

$$DevRTT_{115} = (1-\beta) \times DevRTT + \beta \times |\ SampleRTT - EstimatedRTT\ |$$
$$= (1-0.25) \times 6.73 + 0.25 \times |115 - 106.722| \quad (\because EstimatedRTT = 106.722 \text{ from Case } 5)$$
$$= 7.10$$

So, $DevRTT_{115}$ is 7.10.

$$TimeoutInterval_{115} = EstimatedRTT + 4.DevRTT$$
$$= 106.722 + 4 \times 7.10$$
$$= 135.122$$

So, $TimeoutInterval_{115}$ is 135.122.

33. In Section 3.5.3, we discussed TCP's estimation of RTT. Why do you think TCP avoids measuring the SampleRTT for retransmitted segments?

ANSWER:

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.

Let's look at what could be 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.

34. What is the relationship between the variable SendBase in Section 3.5.4 and the variable LastByteRcvd in Section 3.5.5?

ANSWER:

Relationship between the variables, SendBase and LastByteRcvd:

SendBase-1 <=LastByteRcvd

<u>Detailed Explanation:</u>

- SendBase-1 is used to find the ***sequence number*** of the last byte(Refer Section 3.5.4 from the text book)
- LastByteRcvd is used to find the **number of the last byte** in the stream of data arrival from network to buffer. So, SendBase is the LastByteRcvd at the receiver end.

35. In Section 3.5.4, we saw that TCP waits until it has received three duplicate ACKs before performing a fast retransmit. Why do you think the TCP designers chose not to perform a fast retransmit after the first duplicate ACK for a segment is received?

ANSWER:

Suppose packets n, n+1, and n+2 are sent, and that packet n is received and ACKed. If packets n+1 and n+2 are reordered along the end-to-end-path (i.e., are received in the order n+2, n+1) then the receipt of packet n+2 will generate a duplicate ack for n and would trigger retrans- mission under a policy of waiting only for second duplicate ACK for retransmission. By waiting for a triple duplicate ACK, it must be the case that two packets after packet nare correctly received, while n+1 was not received. The designers of the triple duplicate. ACK scheme probabl- y felt that waiting for two packets (rather than 1) was the right tradeoff between triggering a quick retransmission when needed, but not retran- smitting prematurely in the face of packet reordering.

36. In Section 3.5.4, we saw that TCP waits until it has received three duplicate ACKs before performing a fast retransmit. Why do you think the TCP designers chose not to perform a fast retransmit after the first duplicate ACK for a segment is received?

ANSWER:

Packets can arrive out of order from the Internet Protocol layer. So, whenever an out of order packet would be received, it would generate a duplicate acknowledgement(ACK).If we perform retransmission after the first duplicate ACK, it would lead the sender to introduce too many redundant packets in the network.

Hence, the TCP designers chose not to perform a fast retransmit after the first duplicate ACK for a segment is received.

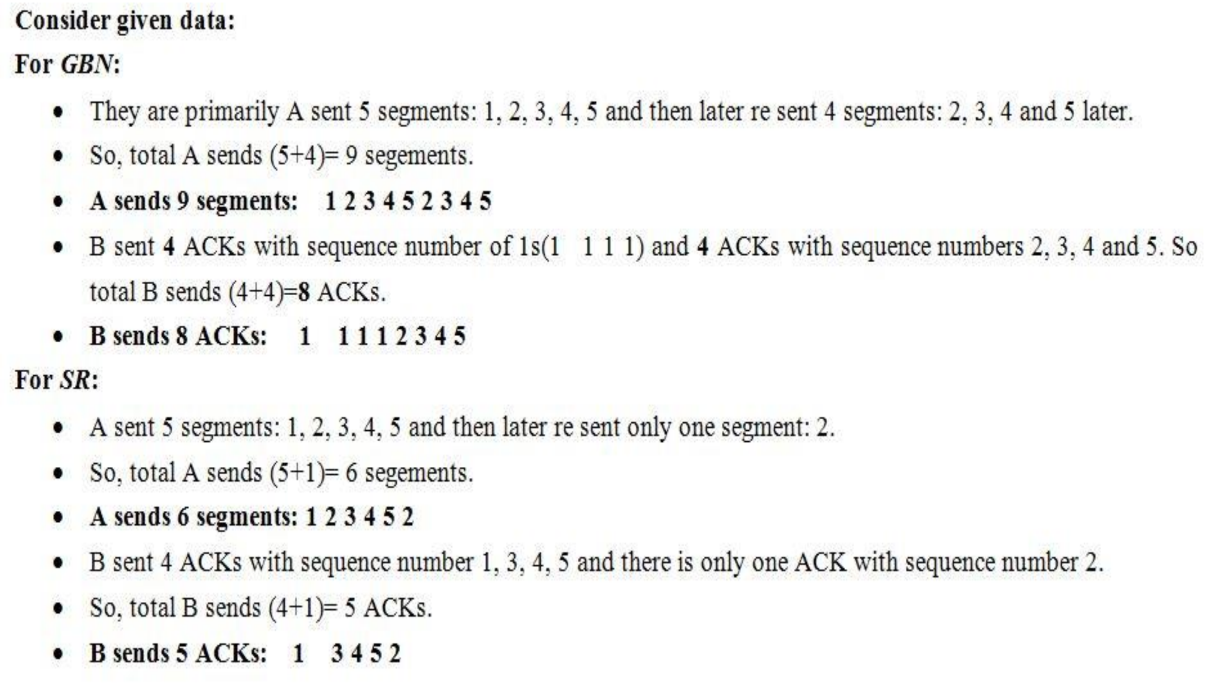
37. Compare GBN, SR, and TCP (no delayed ACK). Assume that the timeout values for all three protocols are sufficiently long such that 5 consecutive data segments and their corresponding ACKs can be received (if not lost in the channel) by the receiving host (Host B) and the sending host (Host A) respec- tively. Suppose Host A sends 5 data segments to Host B, and the 2nd segment (sent from A) is lost. In the end, all 5 data segments have been correctly received by Host B.

a) **How many segments has Host A sent in total and how many ACKs has Host B sent in total? What are their sequence numbers? Answer this question for all three protocols.**

b) **If the timeout values for all three protocol are much longer than 5 RTT, then which protocol successfully delivers all five data segments in shortest time interval?**

ANSWER:

a)



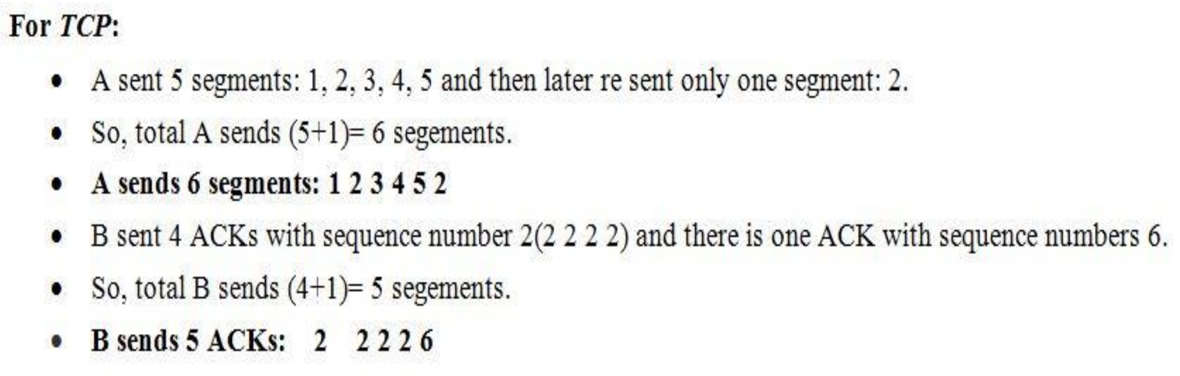
Consider given data:

For *GBN*:

- They are primarily A sent 5 segments: 1, 2, 3, 4, 5 and then later re sent 4 segments: 2, 3, 4 and 5 later.
- So, total A sends (5+4)= 9 segements.
- **A sends 9 segments: 1 2 3 4 5 2 3 4 5**
- B sent **4** ACKs with sequence number of 1s(1 1 1 1) and **4** ACKs with sequence numbers 2, 3, 4 and 5. So total B sends (4+4)=**8** ACKs.
- **B sends 8 ACKs: 1 1 1 1 2 3 4 5**

For *SR*:

- A sent 5 segments: 1, 2, 3, 4, 5 and then later re sent only one segment: 2.
- So, total A sends (5+1)= 6 segements.
- **A sends 6 segments: 1 2 3 4 5 2**
- B sent 4 ACKs with sequence number 1, 3, 4, 5 and there is only one ACK with sequence number 2.
- So, total B sends (4+1)= 5 ACKs.
- **B sends 5 ACKs: 1 3 4 5 2**



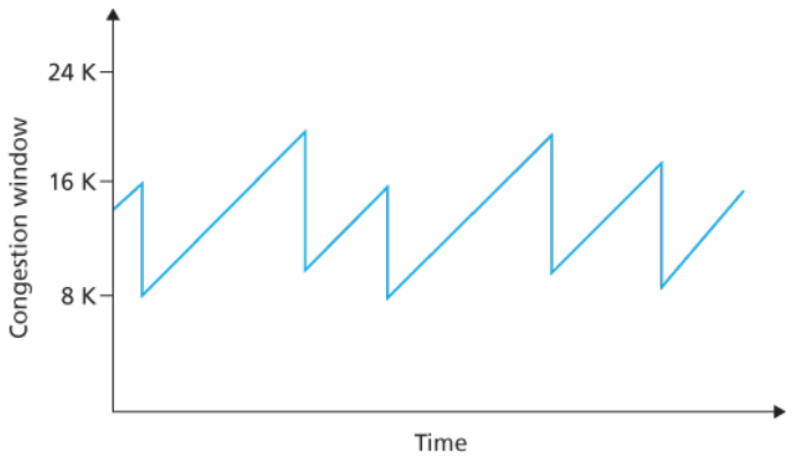
For *TCP*:

- A sent 5 segments: 1, 2, 3, 4, 5 and then later re sent only one segment: 2.
- So, total A sends (5+1)= 6 segements.
- **A sends 6 segments: 1 2 3 4 5 2**
- B sent 4 ACKs with sequence number 2(2 2 2 2) and there is one ACK with sequence numbers 6.
- So, total B sends (4+1)= 5 segements.
- **B sends 5 ACKs: 2 2 2 2 6**

b)

If the timeout values for all three protocol are much longer than 5 RTT, then ***TCP*** protocol successfully delivers all five data segments in shortest time interval. The reason is that TCP uses fast retransmit without waiting time.

38. In our description of TCP in Figure, the value of the threshold, ssthresh, is set as ssthresh=cwnd/2 in several places and ssthresh value is referred to as being set to half the window size when a loss event occurred. Must the rate at which the sender is sending when the loss event occurred be approximately equal to cwnd segments per RTT? Explain your answer. If your answer is no, can you suggest a different manner in which ssthresh should be set?



ANSWER:
If the pace at which the sender is sending is approximately equal to cwnd segments per RTT then the loss event is occurring.
Sound is a variable operated at the sender side. It is used to execute a constraint on the traffic rate.
ssthresh is also called slow start threshold. It is used to store second state value.
When a segment loss event is occurring and the TCP sender identifies the loss, then the value of ssthresh is reset. So, sender rate is cwnd/RTT.

39. Consider Figure (a). If λ'_{in} increases beyond R/2, can λ_{out} out increase beyond R/3? Explain. Now consider Figure (b). If λ'_{in} in increases beyond R/2, can λ_{out} out increase beyond R/4 under the assumption that a packet will be forwarded twice on average from the router to the receiver? Explain.

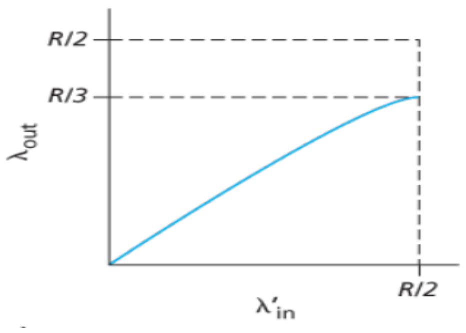


Figure (a)

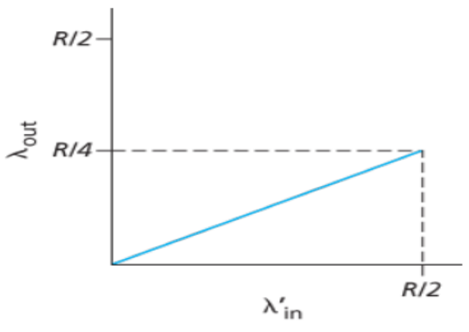


Figure (b)

Consider the given data:

Rate of socket send data $= \lambda_{out}$ bytes/sec.
Rate of network layer send data $= \lambda'_{out}$ bytes/sec.

Rate of application send original data $= \lambda_{in}$ bytes/sec.
Rate of transport layer sending data $= \lambda'_{in}$ bytes/sec

Take the Figure 3.46 (b) from the textbook.

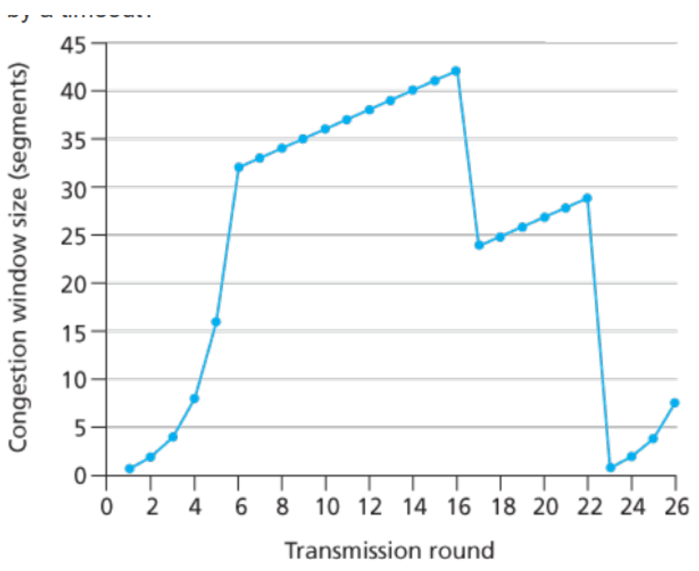
Take the condition that the arrival rate $= \frac{R}{2}$.

The arrival rate $\frac{R}{2}$ is the maximum departure rate of the packets effecting out of the queue. Every third packet or more packets are retransmitted if the arrival rate increases. So, the successfully delivered data throughput will not increase beyond λ_{out}.

The maximum value of $\lambda_{out} = \frac{\left(\frac{R}{2}\right)}{2} = \frac{R}{4}$.

So, the forwarded twice on average from the router to the receive $\frac{R}{4}$ packets.

40. Consider Figure. Assuming TCP Reno is the protocol experiencing the behavior shown above, answer the following questions. In all cases, you should provide a short discussion justifying your answer.



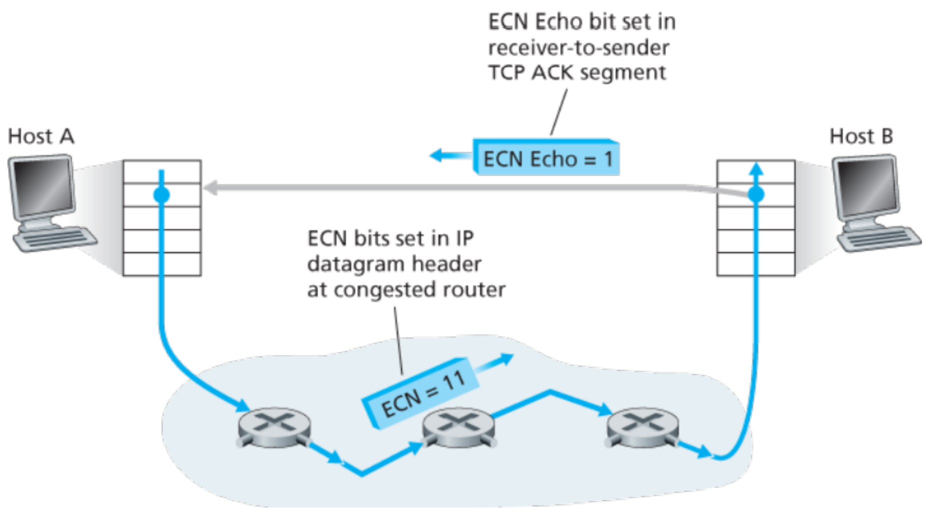
a) **Identify the intervals of time when TCP slow start is operating.**
b) **Identify the intervals of time when TCP congestion avoidance is operating.**
c) **After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?**
d) **After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?**
e) **What is the initial value of ssthresh at the first transmission round?**

f) **What is the value of ssthresh at the 18th transmission round?**
g) **What is the value of ssthresh at the 24th transmission round?**
h) **During what transmission round is the 70th segment sent?**
i) **Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh?**
j) **Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 16th round. What are the ssthresh and the congestion window size at the 19th round?**
k) **Again suppose TCP Tahoe is used, and there is a timeout event at 22nd round. How many packets have been sent out from 17th round till 22nd round, inclusive?**

ANSWER:

a) If TCP slow start is operating, then the intervals of time **1 to 6 and 23 to 26**.
b) If TCP congestion avoidance is operating, then the intervals of time **6 to 23**.
c) After the 16th transmission round, then the segment loss detected by **a triple duplicate ACK**.
d) After the 22nd transmission round, then the segment loss detected by **timeout**.
e) The initial value of ssthresh at the first transmission round **32**.
f) The value of ssthresh at the 18th transmission round **21**.
g) The value of ssthresh at the 24th transmission round **13**.
h) The transmission round is the 70th segment sent is **7**.
i) If a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, then the value is **4**.
j) Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 16th round. Then the ssthresh and the congestion window size at the 19th round is 1 and transmission round is 21.
k) Again suppose TCP Tahoe is used, and there is a timeout event at 22nd round, then the packets have been sent out from 17th round till 22nd round(inclusive) is **52**.

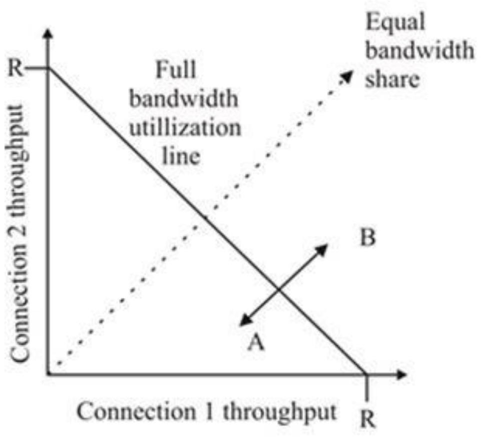
41. Refer to Figure, which illustrates the convergence of TCP's AIMD algorithm. Suppose that instead of a multiplicative decrease, TCP decreased the window size by a constant amount. Would the resulting AIAD algorithm converge to an equal share algorithm? Justify your answer using a diagram similar to Figure.


ECN Echo bit set in receiver-to-sender TCP ACK segment
Host A
Host B
ECN Echo = 1
ECN bits set in IP datagram header at congested router
ECN = 11

ANSWER:

Consider given data:
Suppose that instead of a multiplicative decrease, TCP decreased the window size by a constant amount.

The result diagram is as follows:



42. The doubling of the timeout interval after a timeout event. This mechanism is a form of congestion control. Why does TCP need a window-based congestion-control mechanism in addition to this doubling-timeout-interval mechanism?

ANSWER:
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.

43. Host A is sending an enormous file to Host B over a TCP connection. Over this connection there is never any packet loss and the timers never expire. Denote the transmission rate of the link connecting Host A to the Internet by R bps. Suppose that the process in Host A is capable of sending data into its TCP socket at a rate S bps, where S = 10 · R. Further suppose that the TCP receive buffer is large enough to hold the entire file, and the send buffer can hold only one percent of the file. What would prevent the process in Host A from continuously passing data to its TCP socket at rate S bps? TCP flow control? TCP congestion control? Or something else? Elaborate.

ANSWER:
In this problem, there is no danger in overflowing the receiver since the receiver's receive buffer can hold the entire file. Also, because there is no loss and acknowledgements are returned before timers expire, TCP congestion control does not throttle the sender. However, the process in host A will not continuously pass data to the socket because the send buffer will quickly fill up. Once the send buffer becomes full, the process will pass data at an average rate or R << S.

44. 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 increase from 6 MSS to 12 MSS (assuming no loss events)?

b) What is the average throughout (in terms of MSS and RTT) for this connection up through time = 6 RTT?]

ANSWER:
a)
Given data:
Assuming cwnd increases by 1 MSS every time a batch of ACKs is received and assuming approximately constant round-trip times.

Then transmission rate of TCP is =w bytes/RTT cwnd increases by 1 MSS if every batch of ACKs received.

The below steps are take for cwnd to increase from 6 MSS to 12 MSS:

- 1 RTTs to to 7 MSS.
- 2 RTTs to 8 MSS.
- 3 RTTs to 9 MSS.
- 4 RTTs to 10 MSS.
- 5 RTTs to 11MSS.
- 6 RTTs to 12 MSS.

b)
Given data:
Connection up through time = 6 RTT
Average throughout (in terms of MSS and RTT) =(6+7+8+9+10+11)/6
=8.5 MSS/RTT

45. From when the connection's rate varies from W/(2 · 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 equal to

$$L = \text{loss rate} = \frac{1}{\frac{3}{8}W^2 + \frac{3}{4}W}$$

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

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

ANSWER:

a.
Recall the macroscopic description of TCP throughput. In the period of time from when the connection's rate varies from W/(2 · RTT) to W/RTT, only one packet is lost (at the very end of the period).
The loss rate *L* is the ratio of the number of packets lost over the number of packets sent.
The number of packets sent in a cycle is as shown below:

$$\frac{W}{2}+\left(\frac{W}{2}+1\right)+L+W=\sum_{n=0}^{W/2}\left(\frac{W}{2}+n\right)=\left(\frac{W}{2}+1\right)\frac{W}{2}+\sum_{n=0}^{W/2}n$$
$$=\left(\frac{W}{2}+1\right)\frac{W}{2}+\frac{W/2(W/2+1)}{2}$$
$$=\frac{W^2}{4}+\frac{W}{2}+\frac{W^2}{8}+\frac{W}{4}$$
$$=\frac{3}{8}W^2+\frac{3}{4}W$$

Hence, the loss rate (fraction of packets) $L=\dfrac{1}{\frac{3}{8}W^2+\frac{3}{4}W}$

b.

Loss rate $L=\dfrac{1}{\frac{3}{8}W^2+\frac{3}{4}W}$

For window size W large, $\frac{3}{8}W^2 >> \frac{3}{4}W$

Thus the loss rate $L \approx 8/3W^2$ or $W \approx \sqrt{\frac{8}{3L}}$.

The average throughput of a connection $=\dfrac{3}{4}\sqrt{\dfrac{8}{3L}}\times\dfrac{MSS}{RTT}=\dfrac{1.22\times MSS}{RTT\times\sqrt{L}}$

46. Consider that only a single TCP (Reno) connection uses one 10Mbps link which does not buffer any data. Suppose that this link is the only congested link between the sending and receiving hosts. Assume that the TCP sender has a huge file to send to the receiver, and the receiver's receive buffer is much larger than the congestion window. We also make the following assumptions: each TCP segment size is 1,500 bytes; the two-way propagation delay of this connection is 150 msec; and this TCP connection is always in congestion avoidance phase, that is, ignore slow start.

a) **What is the maximum window size (in segments) that this TCP connection can achieve?**
b) **What is the average window size (in segments) and average throughput (in bps) of this TCP connection?**
c) **How long would it take for this TCP connection to reach its maximum window again after recovering from a packet loss?**

ANSWER:
a). The maximum window size:
10^6*0.15/(8*1.5*10^3) = 125. Therefore, the maximum window size is 125 segments.
b). Average window size is 2*W/3 = 84 segments
and average throughput is 84*1500*8/0.15 = 6.72 Mbps
c). It would take 84/2*0.15 = 6.03 sec.

47. Consider the scenario described in the previous problem. Suppose that the 10Mbps link can buffer a finite number of segments. Argue that in order for the link to always be busy sending data, we would like to choose a buffer size that is at least the product of the link speed C and the two-way propagation delay between the sender and the receiver.

ANSWER:

Let W denote max window size. Let S denote the buffer size. For simplicity, suppose TCP sender sends data packets in a round by round fashion, with each round corresponding to a RTT. If the window size reaches W, then a loss occurs. Then the sender will cut its congestion window size by half, and waits for the ACKs for W/2 outstanding packets before it starts sending data segments again. In order to make sure the link always busying sending data, we need to let the link busy sending data in the period W/(2*C) (this is the time interval where the sender is waiting for the ACKs for the W/2 outstanding packets). Thus, S/C must be no less than W/(2*C), that is, S>=W/2. Let Tp denote the one-way propagation delay between the sender and the receiver. When the window size reaches the minimum W/2 and the buffer is empty, we need to make sure the link is also busy sending data. Thus, we must have W/2/(2Tp)>=C, thus, W/2>=C*2Tp. Thus, S>=C*2Tp.

48. Repeat Problem 43, but replacing the 10 Mbps link with a 10 Gbps link. Note that in your answer to part c, you will realize that it takes a very long time for the congestion window size to reach its maximum window size after recovering from a packet loss. Sketch a solution to solve this problem.

ANSWER:

a) Repeat Problem 43, but replacing the 10 Mbps link with a 10 Gbps link.
The congestion window size to reach its maximum window size after recovering from a packet loss=2200msec.
b) YES
c) YES
d) NO

49. Let T (measured by RTT) denote the time interval that a TCP connection takes to increase its congestion window size from W/2 to W, where W is the maximum congestion window size. Argue that T is a function of TCP's average throughput.
ANSWER:

Here, B represents as the average throughput of a connection.
If a connection has loss rate L, then its average throughput

$$B = \frac{1.22 \times MSS}{RTT\sqrt{L}}$$

Loss rate L is derived from the above formula as shown below:

$$B=\frac{1.22\times MSS}{RTT\sqrt{L}}$$

$$\sqrt{L}=\frac{1.22\times MSS}{RTT\times B}$$

$$L=\left(\frac{1.22\times MSS}{RTT\times B}\right)^2$$

TCP host sends 1/L packets between two consecutive packet losses.
Hence, the time interval T is derived as follows

$$T=\frac{(1/L)\times MSS}{B}$$

$$=\frac{MSS}{L\times B}$$

$$=\frac{MSS}{\left(\frac{1.22\times MSS}{RTT\times B}\right)^2\times B}$$

$$=\frac{RTT^2\times B}{1.22^2\times MSS}$$

$$\text{Hence, } T=\frac{RTT^2\times B}{1.22^2\times MSS}$$

Therefore, T is a function of TCP's average throughput.

50. Consider a simplified TCP's AIMD algorithm where the congestion window size is measured in number of segments, not in bytes. In additive increase, the congestion window size increases by one segment in each RTT. In multiplicative decrease, the congestion window size decreases by half (if the result is not an integer, round down to the nearest integer). Suppose that two TCP connections, C_1 and C_2, share a single congested link of speed 30 segments per second. Assume that both C_1 and C_2 are in the congestion avoidance phase. Connection C_1's RTT is 50 msec and connection C_2's RTT is 100 msec. Assume that when the data rate in the link exceeds the link's speed, all TCP connections experience data segment loss.

a) **If both C_1 and C_2 at time t 0 have a congestion window of 10 segments, what are their congestion window sizes after 1000 msec?**
b) **In the long run, will these two connections get the same share of the bandwidth of the congested link? Explain.**

ANSWER:

The difference between C1 and C2 is that RTT of C1's is only half of the RTT of C2. Therefore, C1 adjusts its window size after 50 msec, but C2 adjusts its window size after 100 msec.
Assume that C1 receives it after 50msec and C2 receives it after 100msec, whenever a loss event happens.
The following simplified model of TCP is that after each RTT, a connection determines if it should increase window size or not.

For C1, to compute the average total sending rate in the link in the previous 50 msec. If that rate exceeds the link capacity, then we assume that C1 detects loss and reduces its window size. But for C2, we compute the average total sending rate in the link in the previous 100msec. If the rate exceeds the link capacity, then we assume that C2 detects loss and reduces its window size.

The two TCP connections, C_1 and C_2, share a single congested link of speed **30 segments** per second.
The RTT for TCP connection of C1 is 50*msec* and RTT of C2 is 100*msec*.

Time in sec	Window size = No. of segments sent in next 50 msec	Average data sending speed = segments per Second (window size/0.05)	Window size = No. of segments sent in next 100msec	Average data sending rate = segments per second
0	10	200	10	100
50	5	100		100
100	2	40	5	50
150	1	20		50
200	1	20	2	20
250	1	20		
300	1	20	1	20
350	2	40		10
400	1	20	1	10
450	2	40		10
500	1	20	1	10
550	2	40		10
600	1	20	1	10
650	2	40		10
700	1	20	1	10
750	2	40		10
800	1	20	1	10
850	2	40		10
900	1	20	1	10
950	2	40		10
1000	1	20	1	10

We conclude that after 1000 msec, C1's and C2's window sizes are 1 segment each based on the above table.
b) In the long run, these two connections will **not** get the same share of the band-width of the congested link because C1's bandwidth share is roughly twice as that of C2's, because C1 has shorter RTT, only half of that of C2, so C1 can adjust its window size twice as fast as C2. According to the above table, for every cycle has 200msec.

For example, in the cycle from 850msec to1000msec, inclusive. The sending rate of C1 is 40+20+40+20 = 120, which is thrice as large as the sending of C2 given by 10+10+10+10 = 40.

51. Consider the network described in the previous problem. Now suppose that the two TCP connections, C1 and C2, have the same RTT of 100 msec. Suppose that at time t 0 , C1's congestion window size is 15 segments but C2's congestion window size is 10 segments.

a) **What are their congestion window sizes after 2200msec?**
b) **In the long run, will these two connections get about the same share of the bandwidth of the congested link?**
c) **We say that two connections are synchronized, if both connections reach their maximum window sizes at the same time and reach their minimum window sizes at the same time. In the long run, will these two connections get synchronized eventually? If so, what are their maximum window sizes?**
d) **Will this synchronization help to improve the utilization of the shared link? Why? Sketch some idea to break this synchronization.**

ANSWER:
a)
Compute the window sizes over the time as shown in the table. The TCP connections C1 and C2 have same window size 2 after 2200msec
The RTT for TCP connections is 100*msec*.
Congestion window size of C1's is **15 segments** and congestion window size of C2's is **10 segments**.

Time is sec	Window size=No.of segments sent in next 100msec	Data sending speed=segments per second(window size/0.1)	Window size=No.of segments sent in next 100msec	Data sending speed=segments per second
0	15	150	10	100
100	7	70	5	50
200	3	30	2	20
300	1	10	1	10
400	2	20	2	20
500	1	10	1	10
600	2	20	2	20
700	1	10	1	10
800	2	20	2	20
900	1	10	1	10
1000	2	20	2	20
1100	1	10	1	10
1200	2	20	2	20

1300	1	10	1	10
1400	2	20	2	20
1500	1	10	1	10
1600	2	20	2	20
1700	1	10	1	10
1800	2	20	2	20
1900	1	10	1	10
2000	2	20	2	20
2100	1	10	1	10
2200	2	20	2	20

b)
These two TCP connections will not get the same share of the bandwidth of the congested link in the long run.

c) In the above table, the maximum window size is 2. So, these two TCP connections will get synchronized at the window sizes is 2.

d) This synchronization will not help to improve the utilization of the shared link because two TCP connections will act as a single link. The window size is varying between minimum and maximum.
An idea to break the synchronization is to add finite buffer to the link and randomly drop the packets in the buffer before buffer overflow.This is caused to different connections cut their window sizes at different times.

52. Consider a modification to TCP's congestion control algorithm. Instead of additive increase, we can use multiplicative increase. A TCP sender increases its window size by a small positive constant a (0 < a < 1) whenever it receives a valid ACK. Find the functional relationship between loss rate L and maximum congestion window W. Argue that for this modified TCP, regardless of TCP's average throughput, a TCP connection always spends the same amount of time to increase its congestion window size from W/2 to W.

ANSWER:

The maximum window size is represented by w.
The total number of segments(S) sent out during the interval when TCP changes its window size from w/2 up to and includes w.
S=w/2 +(w/2)*(1+a)+(w/2)*(1+a)^2 +(w/2)*(1+a)^3 +(w/2)*(1+a)^4 + . . .+ (w/2)*(1+a)^n
Here,n=$\log_{(1+a)}$ 2,then S=w*(2a+1)/(2a).

The Loss rate is derived by
L=1/S= (2a)/ (w*(2a+1)).

The TCP takes time to increase its window size from w/2 to w is calculated by

n*RTT = (log$_{(1+a)}$ 2 * RTT
This is independent of TCP’s average throughput.
TCP’s average throughput is derived by
Throughput B=MSS*S/((n+1)*RTT) =MSS/(L*(k+1)*RTT)
Note that the derived throughput is different from original throughput.

53. In our discussion of TCP futures in Section 3.7, we noted that to achieve a throughput of 10 Gbps, TCP could only tolerate a segment loss probability of 2 · 10 $^{-10}$ (or equivalently, one loss event for every 5,000,000,000 segments). Show the derivation for the values of 2 · 10 $^{-10}$ (1 out of 5,000,000) for the RTT and MSS values given in Section 3.7. If TCP needed to support a 100 Gbps connection, what would the tolerable loss be?

ANSWER:

For 10Gbps:

Throughput (or)Bandwidth(B)= 10 Gbps $= 10\times10^9$ bps $= 10^{10}$ bps

$MaximumSegmentSize(MSS)$ = 1500 bytes=1500×8bits

$Round\ Trip\ Time(RTT)$=100ms=100×10^{-3}sec=0.1sec

Let L denotes the loss rate and B denotes the average bandwidth (or) throughput.

Average bandwidth (or) throughput (B) $=\dfrac{1.22\times MSS}{RTT\sqrt{L}}$

From the above formula, loss rate L is as shown below.

$$B=\frac{1.22\times MSS}{RTT\sqrt{L}}$$
$$\sqrt{L}=\frac{1.22\times MSS}{RTT\times B}$$
$$L=\left(\frac{1.22\times MSS}{RTT\times B}\right)^2$$

Substitute the values of MSS, B and RTT in the above formula.

$$\begin{aligned}L&=\left(\frac{1.22\times MSS}{RTT\times B}\right)^2\\&=\left(\frac{1.22\times1500\times8}{0.1\times10^{10}}\right)^2\\&=\left(\frac{14640}{10^9}\right)^2\\&=(0.00001464)^2\\&=0.0000000002143296\\&=2.14\times10^{-10}\end{aligned}$$

Hence, the tolerable loss L will be 2.14×10^{-10} **or** 2×10^{-10} **(approx).**

54. In our discussion of TCP congestion control in Section 3.7, we implicitly assumed that the TCP sender always had data to send. Consider now the case that the TCP sender sends a large amount of data and then goes idle (since it has no more data to send) at t 1 . TCP remains idle for a relatively long period of time and then wants to send more data at t 2 . What are the advantages and disadvantages of having TCP use the cwnd and ssthresh values from t 1 when starting to send data at t 2 ? What alternative would you recommend? Why?

ANSWER:
An advantage of using the earlier values of cwnd and ssthresh at t2 is that TCP would not have to go through slow start and congestion avoidance to ramp up to the throughput value obtained at t1. A disadvantage of using these values is that they may be no longer accurate. In particular, if the path has become more congested between t1 and t2, the sender will send a large window's worth of segments into an already (more) congested path.

55. In this problem we investigate whether either UDP or 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?**
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 acknowledgment 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)?**

ANSWER:
a) Let UDP packet is received a request to the server. This request accepts the server.
So, the IP address X is deceived with address Y and response to the address Y.
So, adress Y is the matching IP address.

b)
The client address is 'Y'
The *SYNACK* will be send with Y's address.
TCP in the host will not send back the *TCP ACK* segment.
It is not possible to the attacker to send the correct sequence number.
The attacker fails even if he sends an properly timed TCP ACK segment.

56. In this problem, we consider the delay introduced by the TCP slow-start phase. Consider a client and a Web server directly connected by one link of rate R. Suppose the client wants to retrieve an object whose size is exactly equal to 15 S, where S is the maximum segment size (MSS). Denote the round-trip time between client and server as RTT (assumed to be constant). Ignoring protocol headers, determine the time to retrieve the object (including TCP connection establishment) when

a. 4 S/R > S/R + RTT > 2S/R
b. S/R + RTT > 4 S/R
c. S/R > RTT.

ANSWER:

Consider the following data:

The transmission rate of Client and Web = *R*.
Maximum Segment Size (*MSS*) = *S*
Round Trip Time (*RTT*) between client and server is constant.

a)
Calculate the time required to retrieve the object when

$$4S/R > S/R > RTT > 2S/R$$

Round trip of sender and receiver = $RTT + RTT$

Calculate the delay in packet transmission as follows:

$$= (S/R + RTT) + (S/R + RTT) + 12S/R + (RTT + RTT)$$

Since $8S/R + 4S/R = 12S/R$

$$= 4RTT + 14S/R$$

Therefore, the time required to retrieve the object is $4RTT + 14S/R$ when $4S/R > S/R > RTT > 2S/R$.

b)
Calculate the time required to retrieve the object when

$$S/R + RTT > 4S/R$$

Round trip of sender and receiver = $RTT + RTT$

Calculate the delay in packet transmission as follows:

$$= (S/R + RTT) + (S/R + RTT) + (S/R + RTT) + 8S/R + (RTT + RTT)$$

$$= 5RTT + 11S/R$$

Therefore, the time required to retrieve the object is $5RTT + 11S/R$ when $S/R + RTT > 4S/R$.

c)
Calculate the time required to retrieve the object when

$$S/R > RTT$$

Round trip of sender and receiver = $RTT + RTT$

Calculate the delay in packet transmission as follows:

$$= (S/R + RTT) + 14S/R + (RTT + RTT)$$

Since 8S/R+4S/R+2S/R=14S/R

$$= 3RTT + 15S/R$$

Therefore, the time required to retrieve the object is $3RTT + 15S/R$ when $S/R > RTT$.