CS2105 Introduction to Computer Networks

Prepared by Clinton Law (AY19/20 Sem1)

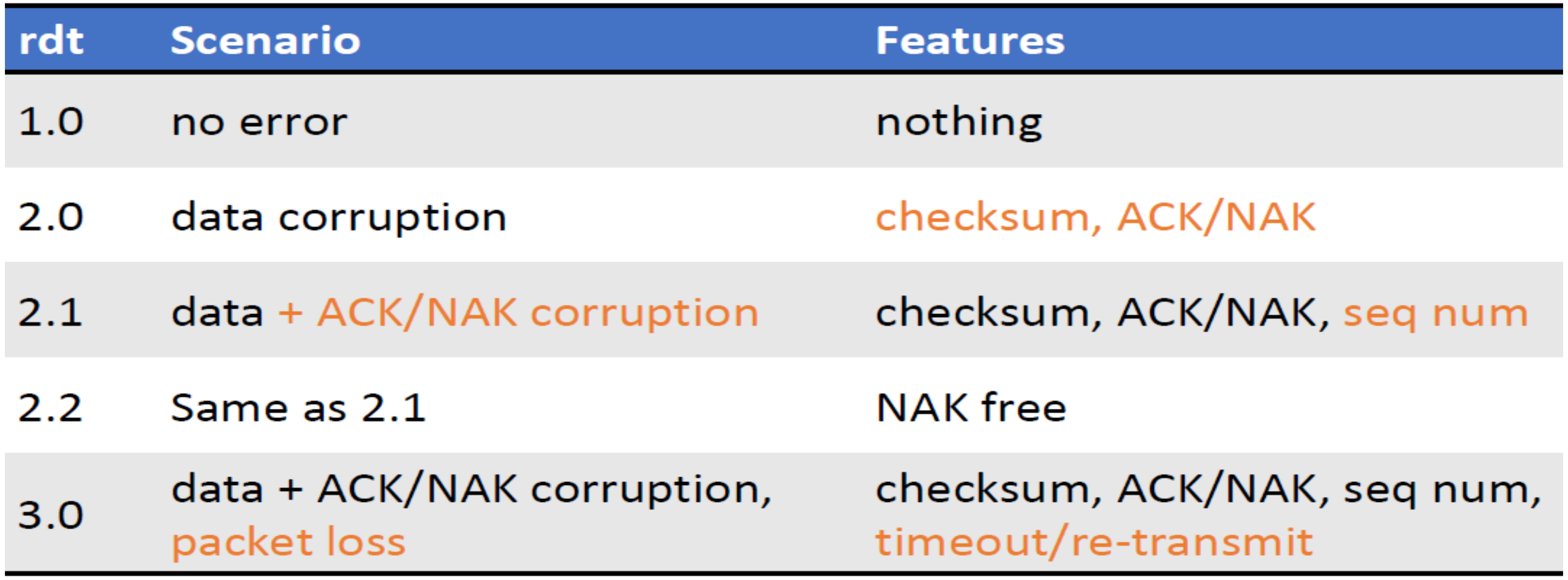
Admin

When you check your emails for the slides, do take time to read the **content** of the email.

I try to explain concepts that were left hanging in class there, which can be useful for midterm revision.

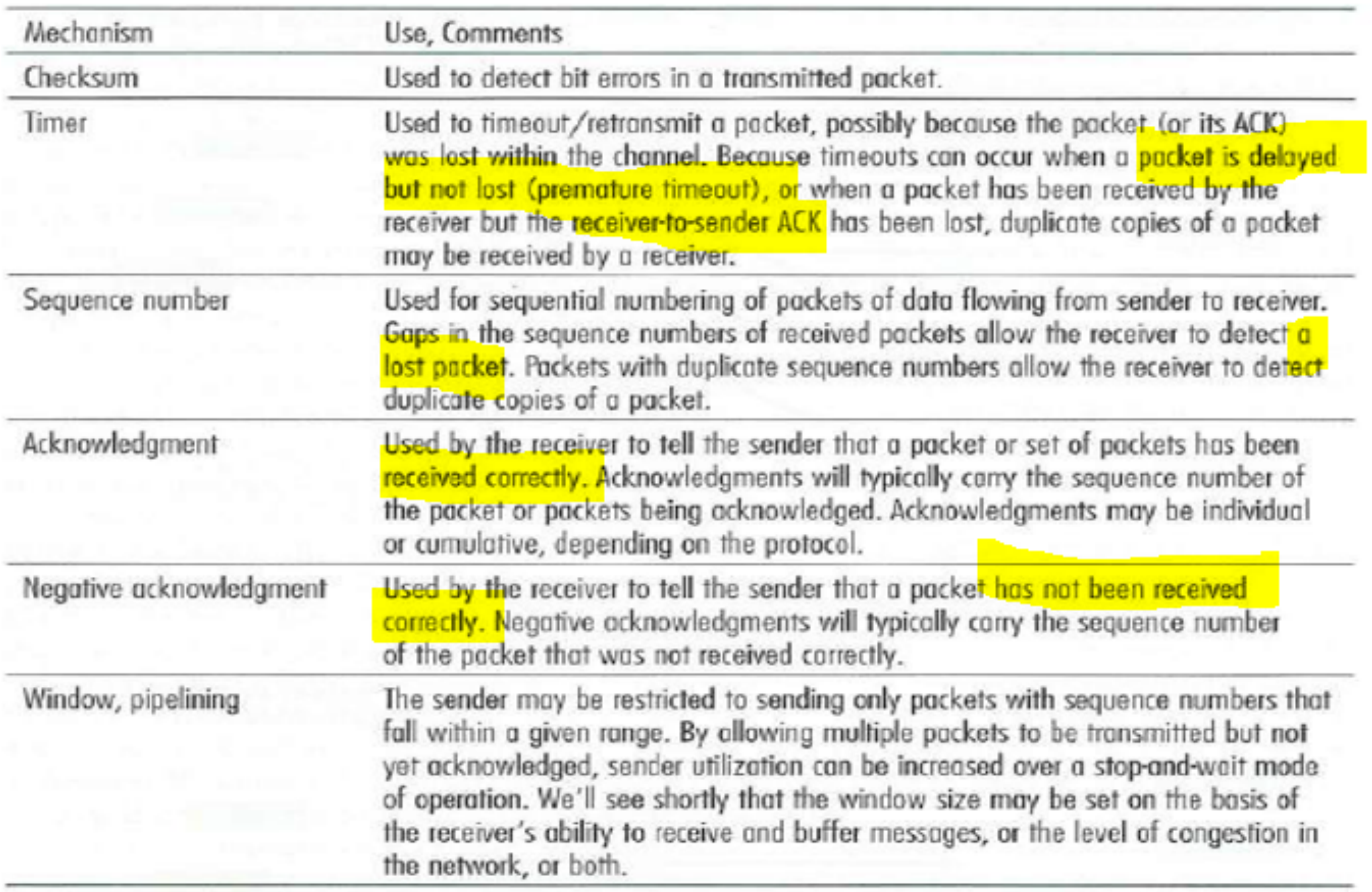
Recap

- RDT
  - Start from a perfect protocol and slowly introduce errors
    - Reduce dependence on unreliable network layer
  - FSMs model how all possible events are connected to each other by means of event occurrences and



rdt	Scenario	Features
1.0	no error	nothing
2.0	data corruption	checksum, ACK/NAK
2.1	data + ACK/NAK corruption	checksum, ACK/NAK, seq num
2.2	Same as 2.1	NAK free
3.0	data + ACK/NAK corruption, packet loss	checksum, ACK/NAK, seq num, timeout/re-transmit

Recap



Mechanism	Use, Comments
Checksum	Used to detect bit errors in a transmitted packet.
Timer	Used to timeout/retransmit a packet, possibly because the packet (or its ACK) was lost within the channel. Because timeouts can occur when a packet is delayed but not lost (premature timeout), or when a packet has been received by the receiver but the receiver-to-sender ACK has been lost, duplicate copies of a packet may be received by a receiver.
Sequence number	Used for sequential numbering of packets of data flowing from sender to receiver. Gaps in the sequence numbers of received packets allow the receiver to detect a lost packet. Packets with duplicate sequence numbers allow the receiver to detect duplicate copies of a packet.
Acknowledgment	Used by the receiver to tell the sender that a packet or set of packets has been received correctly. Acknowledgments will typically carry the sequence number of the packet or packets being acknowledged. Acknowledgments may be individual or cumulative, depending on the protocol.
Negative acknowledgment	Used by the receiver to tell the sender that a packet has not been received correctly. Negative acknowledgments will typically carry the sequence number of the packet that was not received correctly.
Window, pipelining	The sender may be restricted to sending only packets with sequence numbers that fall within a given range. By allowing multiple packets to be transmitted but not yet acknowledged, sender utilization can be increased over a stop-and-wait mode of operation. We'll see shortly that the window size may be set on the basis of the receiver's ability to receive and buffer messages, or the level of congestion in the network, or both.

Table 3.1 ♦ Summary of reliable data transfer mechanisms and their use

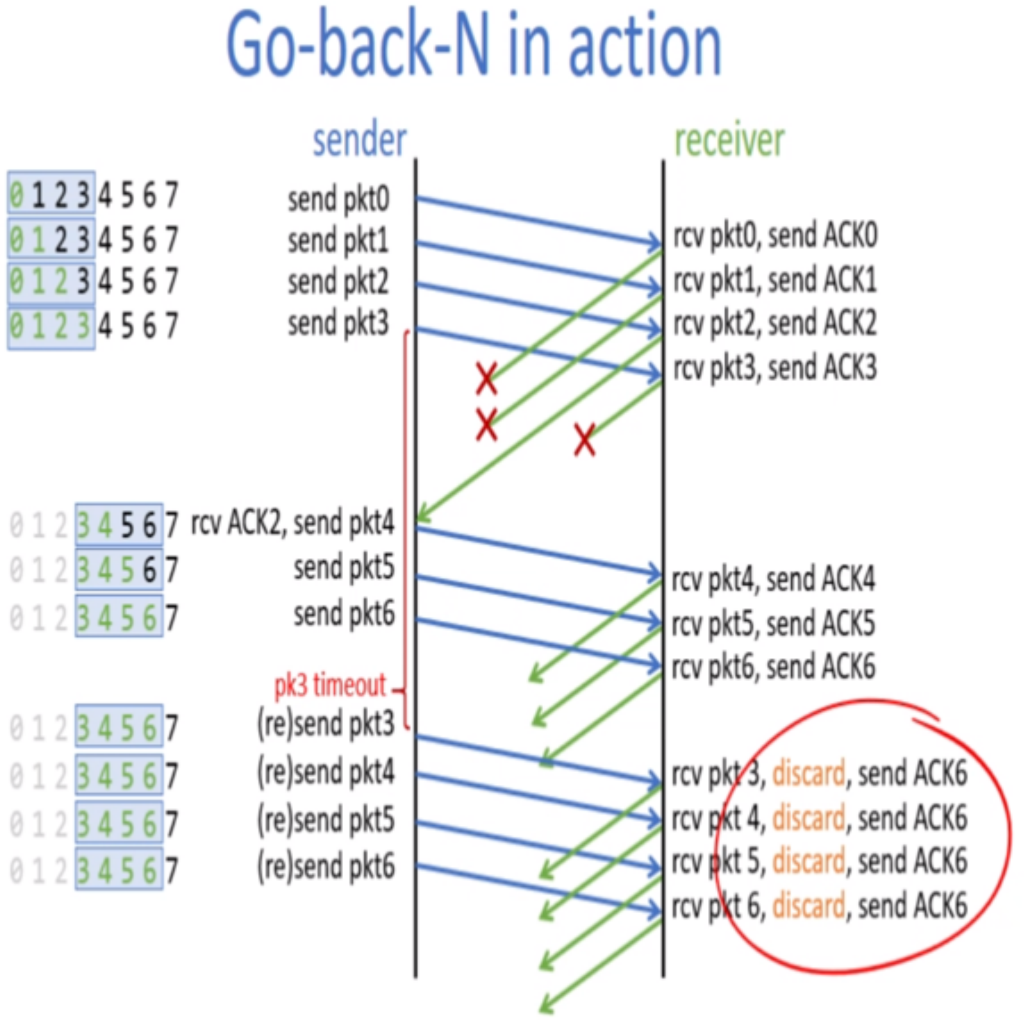
Recap

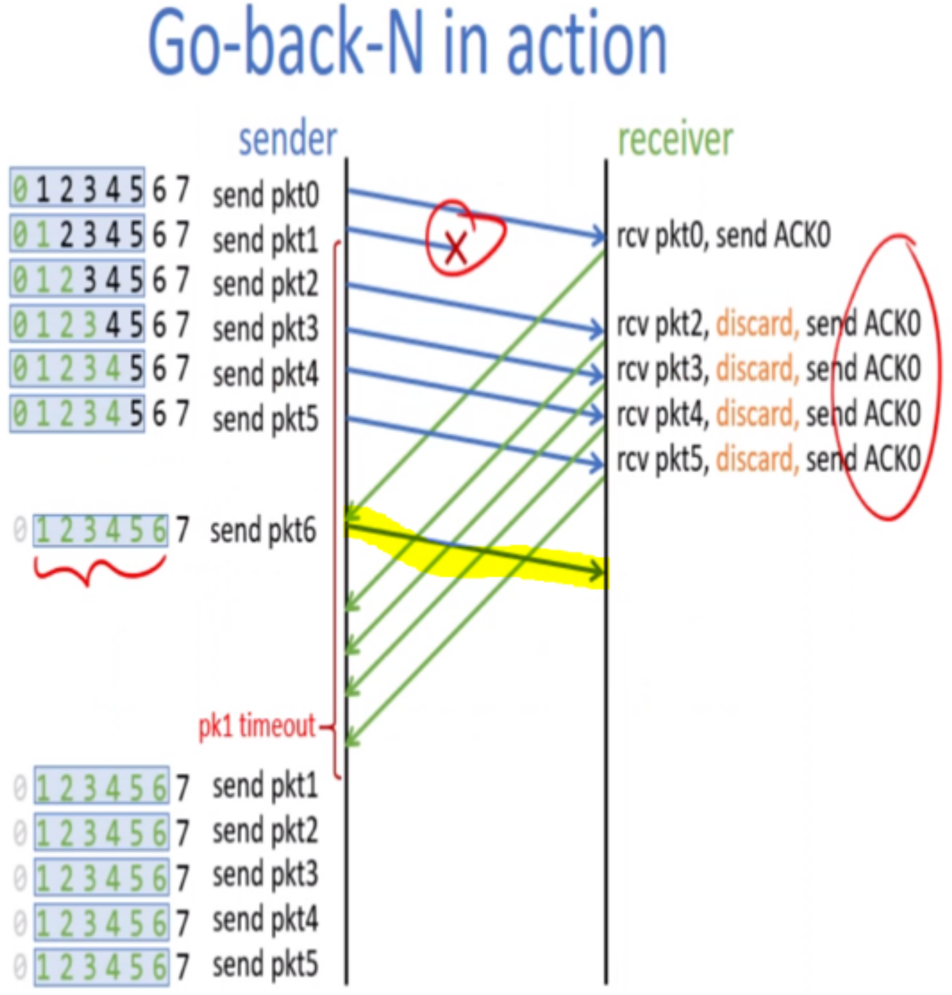
- RDT models employ a stop & wait protocol
  - Utilisation of link is very small ($\frac{\mathbf{T_{used}}}{\mathbf{T_{total}}}$)
- Pipeline
  - Alternating bit protocols cannot be used as need to keep track of more packets
  - Concept of the buffer
    - GBN
    - SR

Recap

- Go-Back N
  - ACKs are **cumulative**
    - If ACK *n* has been received, I can be sure that packets 0 to *n* have been received
    - Don't have to receive every single ACK to be sure that all packets have been received, just need the "latest" ACK
  - Sender
    - Has a size *N* buffer to keep track of *N* unACKed packets
      - Transmission will stop if buffer is full, even if there are more packets to be sent
    - Maintains a timer for **oldest unACKed packet**
      - On timeout, retransmit the whole buffer
        - Why?
  - Receiver
    - Has no buffer
      - Discards all out of order packets
      - Just need to track expected seq num

Recap



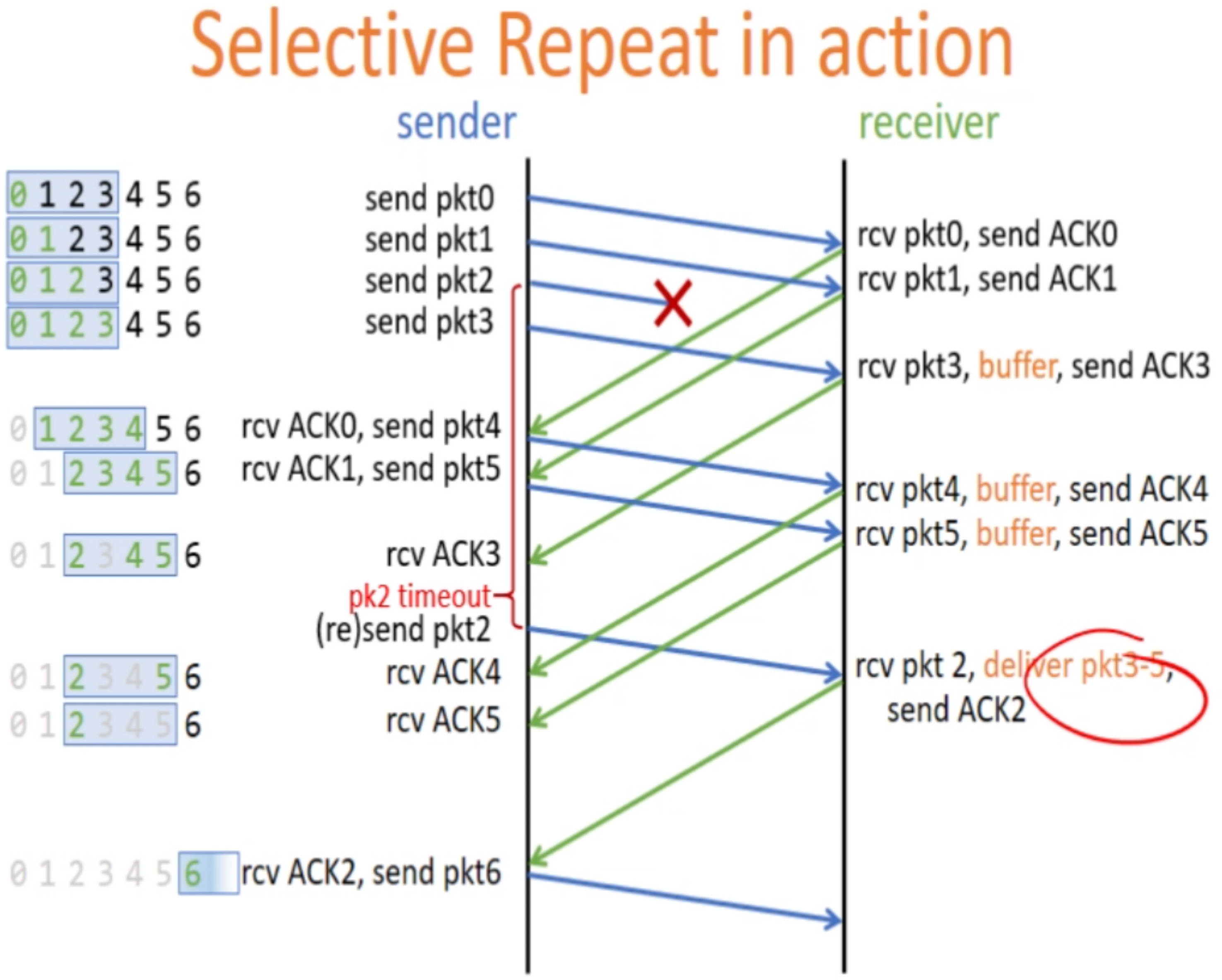


packet 6 is ignored

Recap

- Selective Repeat (SR)
  - ACKs are specific for their individual packet, **NOT CUMULATIVE**
    - If I receive an ACK *n*, I can only be sure that packet *n* has been received, nothing else.
  - Sender
    - Maintains a timer for **each unACKed packet**
      - On timeout, only retransmit the timedout packet
  - Receiver
    - Flexibility requires a buffer (again, expensive)
      - Having independent ACKs allows for out of order packets but need to be held in a buffer till the packets form a sequential stream to be pushed into the app layer

Recap



Selective Repeat in action
sender
receiver
0 1 2 3 4 5 6
0 1 2 3 4 5 6
0 1 2 3 4 5 6
0 1 2 3 4 5 6
send pkt0
send pkt1
send pkt2
send pkt3
rcv pkt0, send ACK0
rcv pkt1, send ACK1
rcv pkt3, buffer, send ACK3
0 1 2 3 4 5 6
0 1 2 3 4 5 6
rcv ACK0, send pkt4
rcv ACK1, send pkt5
rcv pkt4, buffer, send ACK4
rcv pkt5, buffer, send ACK5
0 1 2 3 4 5 6
rcv ACK3
pk2 timeout
(re)send pkt2
rcv pkt 2, deliver pkt3-5, send ACK2
0 1 2 3 4 5 6
rcv ACK4
0 1 2 3 4 5 6
rcv ACK5
0 1 2 3 4 5 6
rcv ACK2, send pkt6

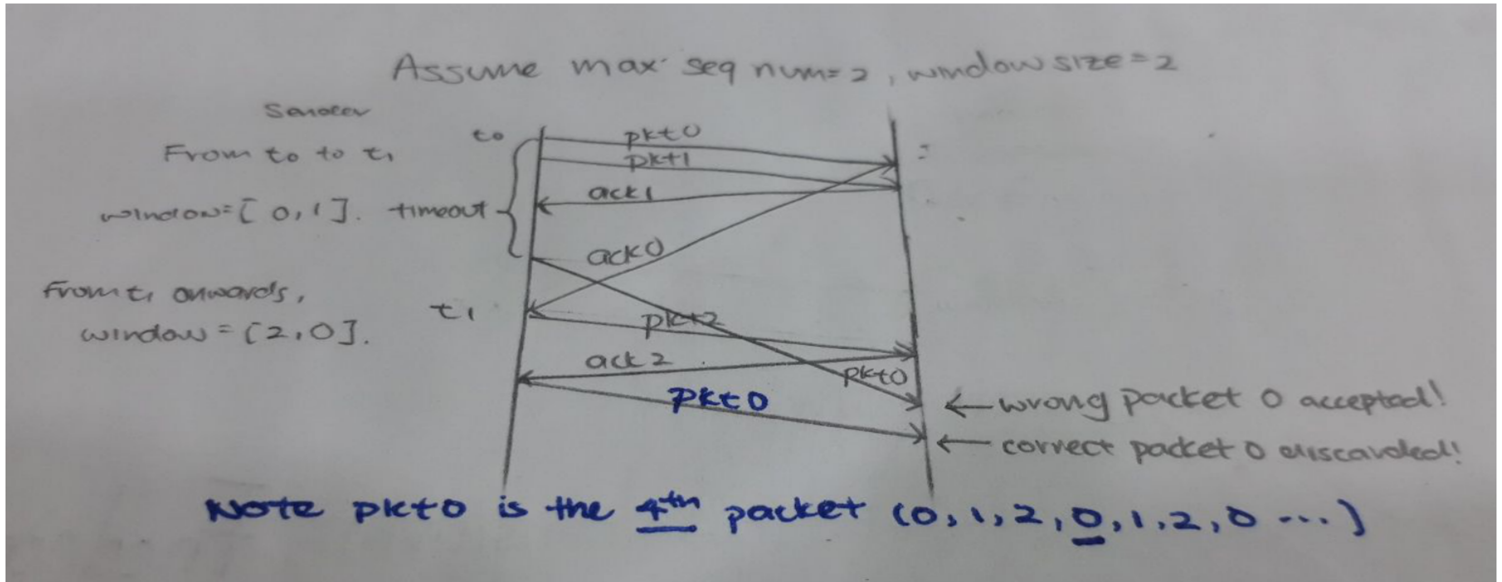
Recap

GBN	**SR**
Cumulative ACKs	Individual ACKs
1 timer (send_base, restarts when win move)	Individual timers (expensive)
Sender: Buffer required Receiver: Keep track of seq num expected	Sender: Buffer required Receiver: Buffer required (exp)
No buffer$_{\text{receiver}}$ -> discards out of order pkts	Can hold out of order pkts
Losing ACK doesn't matter – **no rtx of ACK**	Losing ACK → timeout → rtx
Max window size, w == Max seq #	**w <= 0.5 * Max seq #**
Retransmission only on timeout (no NACKs)	
#unACKed packets = N packets in pipeline	
Addresses the problem of poor utilisation via pipelining.	
Both models alternating bit protocol **if window size == 1.**	
Retransmit whole buffer	Retransmit single pkt

****GBN** rtx whole buffer because out of order discarded → whole window must rtx

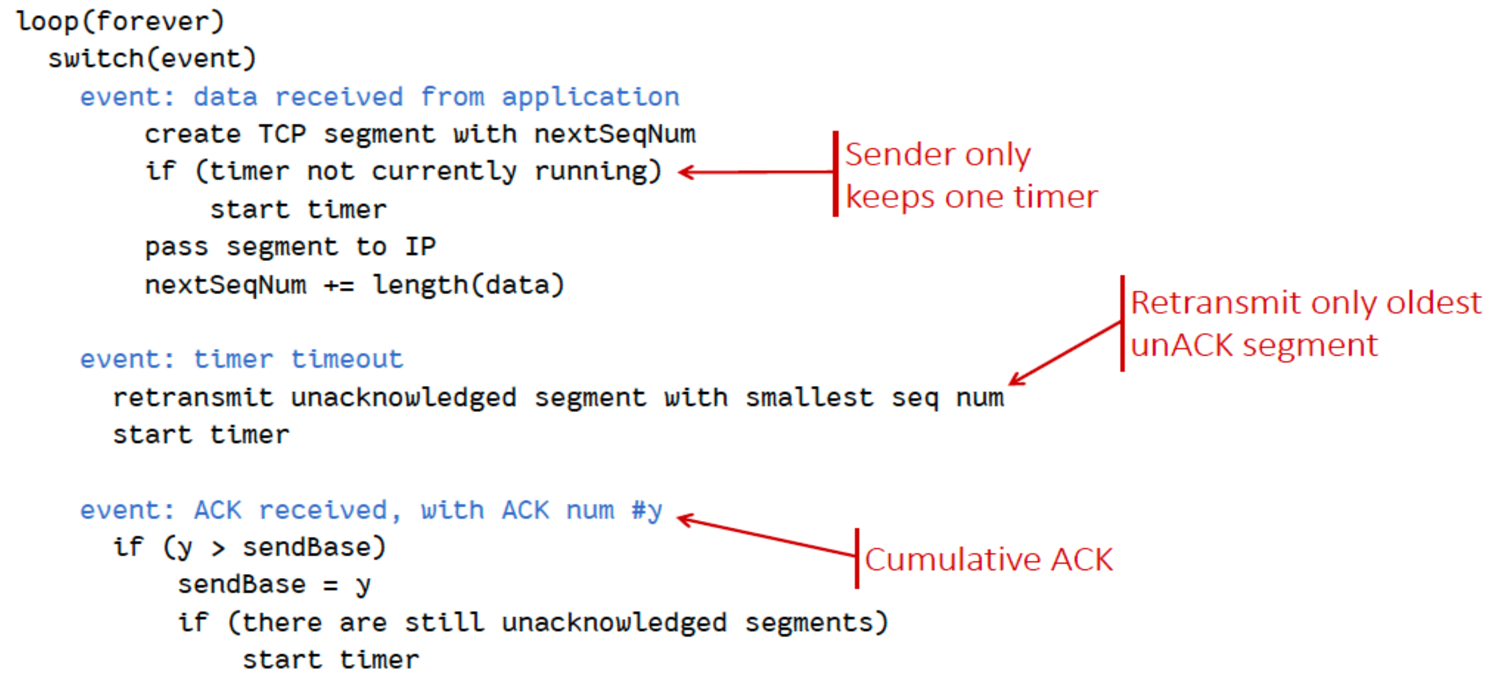
Recap

- Max buffer size
  - Buffer size is limited by the possibility of having a repeated packet of the same seq num within the same buffer where the receiver cannot distinguish between the 2
  - GBN
    - Trivial (*n, where n is the maximum seq num*)
  - SR
    - 0.5*n, where n is the maximum seq num*
    - Example of failure scenario if (max seq num / 2) > window size


Assume max seq num=2, window size=2
Sender
From t0 to t1
window=[0,1].
timeout
t0
pkt0
pkt1
ack1
ack0
From t1 onwards,
window=[2,0].
t1
pkt2
ack2
pkt0
pkt0
←wrong packet 0 accepted!
←correct packet 0 discarded!
Note pkt0 is the 4th packet (0,1,2,0,1,2,0...)

Recap

- TCP
  - Seq num (range, start)
    - Span a huge number to prevent duplicate sequence numbers to reduce duplicate packets which could lead to message reordering
    - Start from a random point so as to prevent malicious attackers from pre-empting the next sequence number due to its predictability
  - Sender events

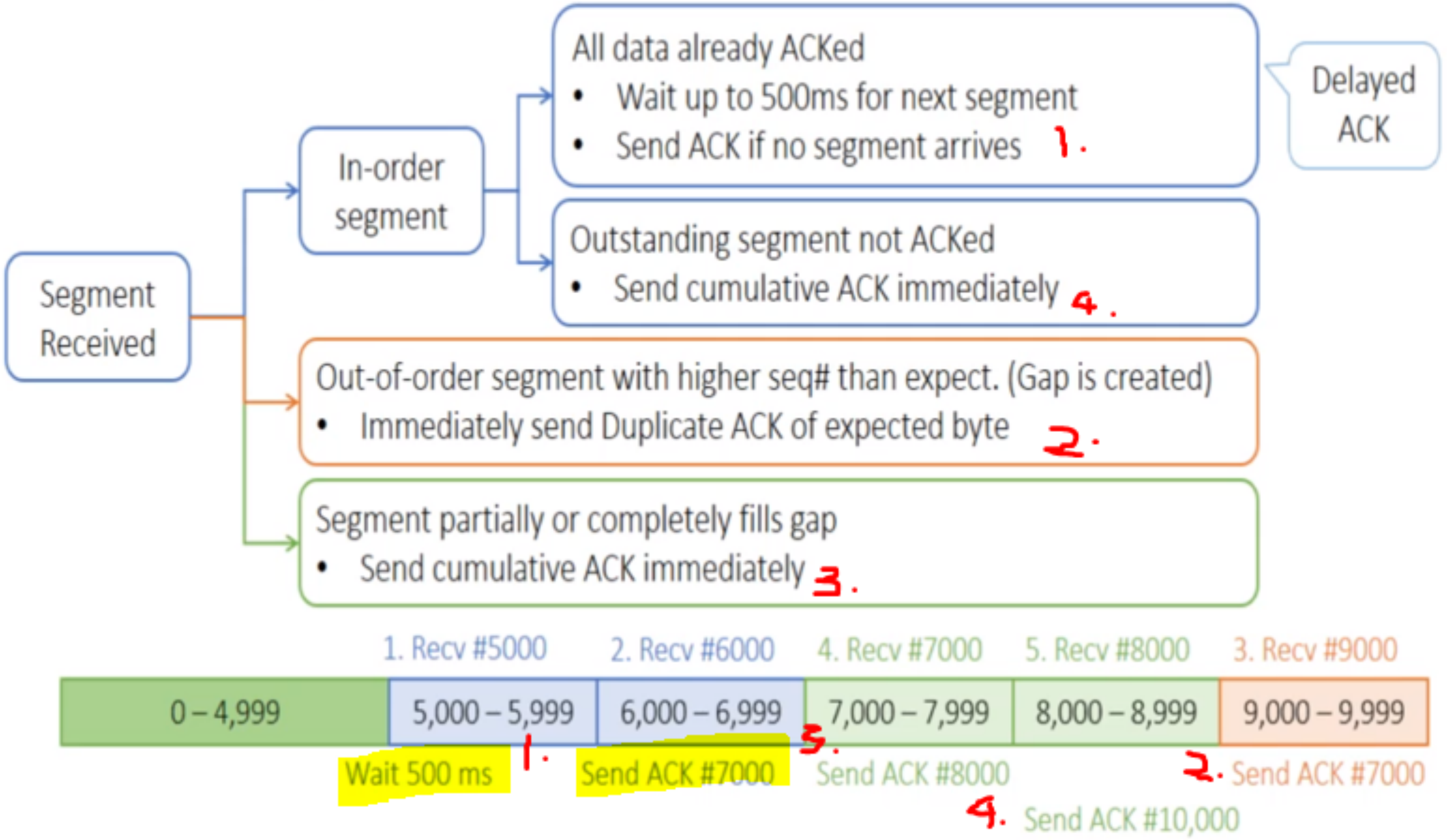


```
loop(forever)
  switch(event)
    event: data received from application
        create TCP segment with nextSeqNum
        if (timer not currently running)
            start timer
        pass segment to IP
        nextSeqNum += length(data)

    event: timer timeout
      retransmit unacknowledged segment with smallest seq num
      start timer

    event: ACK received, with ACK num #y
      if (y > sendBase)
          sendBase = y
          if (there are still unacknowledged segments)
              start timer
```

Sender only keeps one timer
Retransmit only oldest unACK segment
Cumulative ACK

Recap

- TCP receiver events


Segment Received
In-order segment
All data already ACKed
• Wait up to 500ms for next segment
• Send ACK if no segment arrives 1.
Delayed ACK
Outstanding segment not ACKed
• Send cumulative ACK immediately 4.
Out-of-order segment with higher seq# than expect. (Gap is created)
• Immediately send Duplicate ACK of expected byte 2.
Segment partially or completely fills gap
• Send cumulative ACK immediately 3.
1. Recv #5000
2. Recv #6000
4. Recv #7000
5. Recv #8000
3. Recv #9000
0 – 4,999
5,000 – 5,999
6,000 – 6,999
7,000 – 7,999
8,000 – 8,999
9,000 – 9,999
Wait 500 ms 1.
Send ACK #7000 5.
Send ACK #8000
2. Send ACK #7000
4. Send ACK #10,000

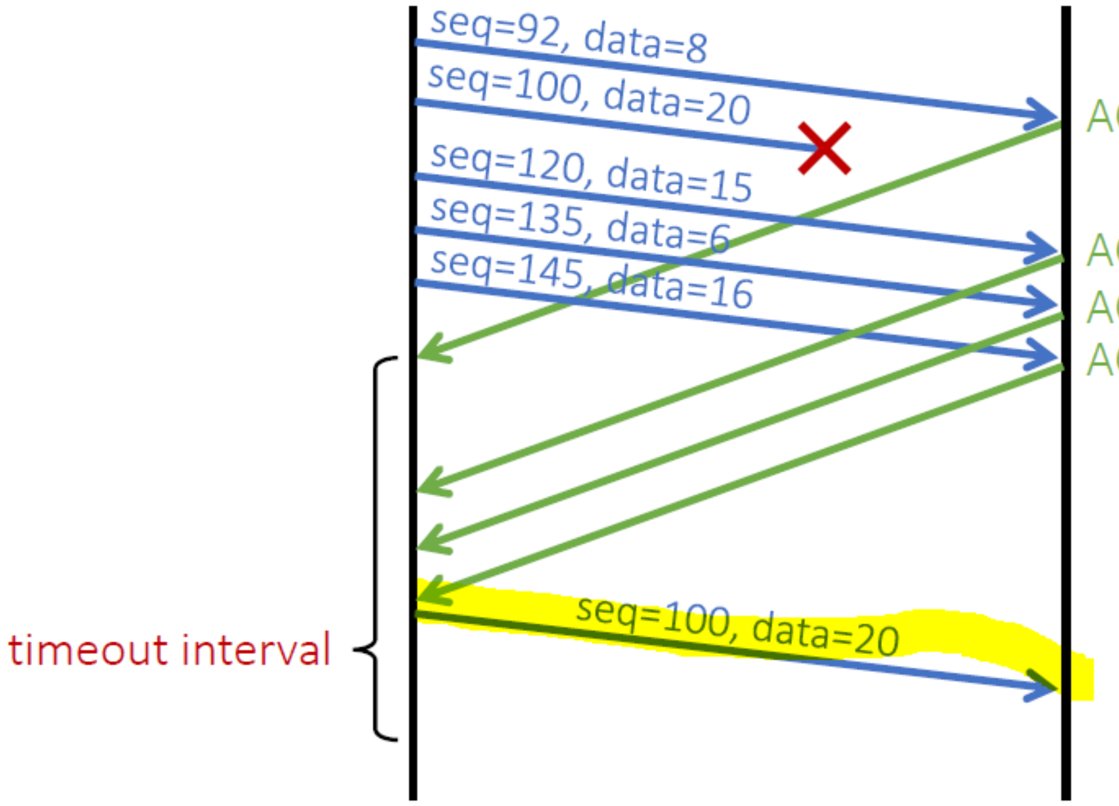
Recap

- TCP timeout value
  - Too long suboptimal
  - Too short many rtx
    - Must minimally be larger than RTT !
    - Use Estimated Weighted Moving Average to estimate RTT

$RTT_{\varepsilon} = (1 - \alpha) \cdot RTT_{\varepsilon} + \alpha \cdot RTT_{s}$, where RTT_s is sample RTT_e is estimate RTT

Recap

- TCP fast retransmission
  - If many duplicate ACKs are received, the sender knows that the network is fine
    - Can immediately send the packet specified by the duplicate ACKs
    - No need to wait for timeout!



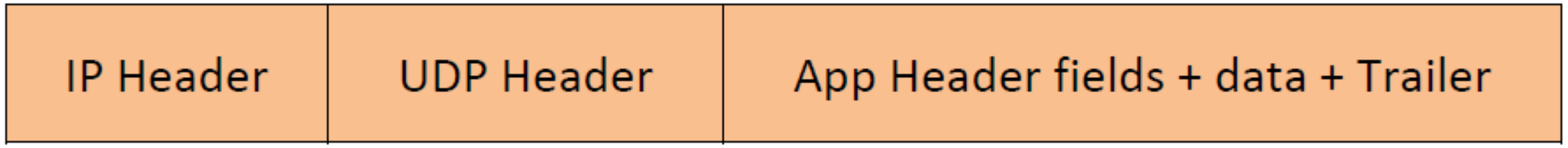
Tutorial Questions

Question 1

[KR, Chapter 3, R6] Is it possible for an application to enjoy reliable data transfer even when the application runs over UDP? If so, how?

Note: this is exactly what you are supposed to do in Assignment 2. ☺

One would have to implement reliability checking and recovery mechanisms (ACK, seq #, checksum, timeout, re-transmission, etc.) at application layer. For example, sender needs to include relevant header/trailer fields in every packet (as illustrated below).

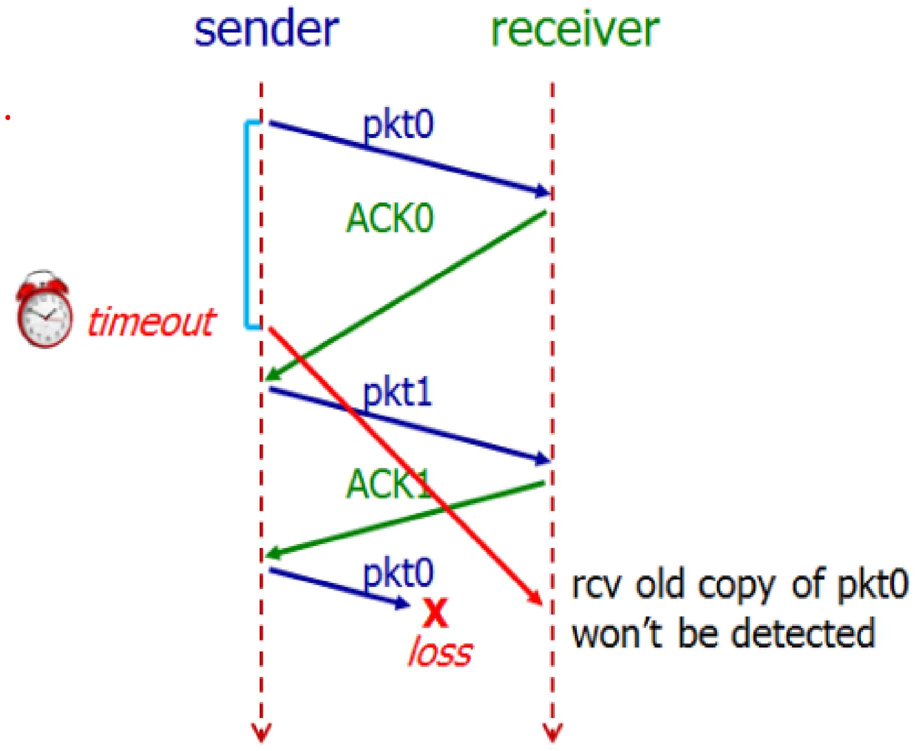


Question 2

Show an example that if the communication channel between the sender and receiver can reorder messages (i.e. two messages are received in different order they are sent), then protocol **rdt3.0** will not work correctly.

Recall that rdt3.0 employs an alternating-bit protocol (seq num is either 0 or 1).

1. Sender sends pkt0
   - Sender **expects ack0.**
2. Receiver receives pkt0 → sends ack0
   - However, this ack0 might have taken a route that is very slow such that the d_{prop} is very long ($\geq$ *timeout*)
   - Receiver **expects pkt1.**
3. Ack0 has not been received within the timeout window → resends pkt0
   - Timeout → Sender **thinks** that initial pkt0 is lost, hence rtx pkt0
   - Still expects ack0
4. Sender **finally** receives ack0 → send pkt1
   - Sender **expects ack1**.
5. Receiver receives pkt1 → send ack1
   - Receiver **expects pkt0.**
6. Sender receives ack1 → sends pkt0
7. OUTDATED pkt 0 is now received by the receiver
   - Receiver does not know this pkt0 is the outdated rtx'd pkt0 in step 3, not the correct pkt 0 sent in step 6!



Question 3

[KR, Chapter 3, P29] It is generally a reasonable assumption, when sender and receiver are connected by a single wire, that packets cannot be reordered within the channel between the sender and receiver. However, when the "channel' connecting the two is a network, packet reordering may occur. One manifestation of packet reordering is that old copies of a packet with a sequence or acknowledgement number of *x* can appear, even though neither sender's nor receiver's window contains *x*. With packet reordering, the channel can be thought of as essentially buffering packets and spontaneously emitting these packets at any point in the future. What is the approach taken in practice to guard against such duplicate packets?

The approach taken in practice is to ensure that a sequence number is not reused until the sender is "sure" that any previously sent packets with the same sequence number are no longer in the network.

Firstly, TCP use large sequence number field (32-bit) to lower the chance a sequence number is to be reused. Secondly, a packet cannot "live" in the network forever. For example, IP protocol specifies TTL in packet header to ensure that datagrams do not circulate infinitely in the network. This field is decreased by one each time the datagram arrives at a router along the end-to-end path. If TTL field reaches 0, router will discard this datagram. In practice, a maximum packet lifetime of approximately three minutes is assumed in the TCP extensions for high-speed networks.

Another way to avoid duplicate packets is by having a randomised sequence number. This is an especially useful way to also guard against attackers. Basically, in establishing your TCP 3way handshake, the syn packet will contain the randomised seq number that the hosts will be using in message exchange. Hence, attackers cannot predict which seq number the message exchange starts from and hence cannot send bogus packets that can be disguised within the rest of the packets.

Question 4

[Modified from KR, Chapter 3, P37] Host A is sending data segments to Host B using a reliable transport protocol (either GBN or SR). Assume timeout values are sufficiently large such that all data segments and their corresponding ACKs can be received (if not lost in the channel) by Host B and the Host A respectively. Suppose Host A sends 5 data segments to Host B and the 2nd data segment is lost. Further suppose retransmission is always successful. In the end, all 5 data segments have been correctly received by Host B.

How many segments has Host A sent in total and how many ACKs has Host B sent in total if either GBN or SR protocol is used? What are their sequence numbers? Answer this question for both protocols.

GBN: Host A sends 9 segment. They are initially sent segments 1, 2, 3, 4, 5 and later resent segments 2, 3, 4 and 5. Host B sends 8 ACKs. They are 4 ACKs with seq # 1 and 4 ACKs with seq # 2, 3, 4 and 5.

SR: Host A sends 6 segment. They are initially sent segments 1, 2, 3, 4, 5 and later resent segment 2. Host B sends 5 ACKs. They are 4 ACKs with seq # 1, 3, 4, 5 and 1 ACK with seq # 2 (for resent segment).

Question 5

[KR, Chapter 3, R15] Suppose Host A sends two TCP segments back to back to Host B over a TCP connection. The first segment has sequence number 65; the second has sequence number 92.

a) How much data is in the first segment?

92 – 65 = 27 bytes

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?

65. Note that TCP acknowledgment is cumulative and states the expected in-order sequence number.

Question 6 part 1

[KR, Chapter 3, P26] Consider transferring an enormous file of *L* bytes from Host A to Host B. Assume an MSS of 512 bytes.

1. What is the maximum value of *L* such that TCP sequence numbers are not exhausted? Recall that the TCP sequence number field is 32 bits.

TCP sequence number doesn't increase by one with each TCP segment. Rather, it increases by the number of bytes of data sent. Therefore the size of the MSS is irrelevant here -- the maximum size file that can be sent from A to B without exhausting TCP sequence number is simply 2^{32} bytes.

Question 6 part 2

For the *L* you obtain in (a), find how long it takes to transmit this file. Assume that a total of 64 bytes of transport, network, and data-link header are added to each packet before the resulting packet is sent out over a 155 Mbps link. Ignore flow control, congestion control and assume Host A can pump out all segments back to back and continuously.

Remember to convert bytes into bits!!

Number of packets = L / MSS = 2^{32} / 512 = 8,388,608

64 bytes of headers will be added to each packet. Therefore,

Total bytes sent = 2^{32} + 8388608 * 64 = 4,831,838,208 bytes

Transmission delay = 4831838208 * 8 / (155 * 10^6) ≈ 249 seconds

Summary

- RDT

- Buffers
  - GBN and SR

Extra Questions

A **Go-Back-N** sender just receives an ACK packet with sequence number 14. This ACK number falls within sender's window. Sender's window size is 6. Every packet embeds a k-bit sequence number field (k is an unknown constant). Which of the following definitely CANNOT be the sequence number of the next packet transmitted by the sender?

A. 9

B. 4

C. 15

D. 19

E. 20

Extra Questions

Consider a sender and a receiver communicating using **Selective Repeat** protocol. After transmitting for a while, the first and the last sequence numbers in the sender's window are k and k+3 respectively. Let a packet with sequence number i be p_i. Which of the following statement MUST be TRUE?

A. p_k is sent and acknowledged.

B. If p_{k+2} is not sent, p_{k+1} is also not sent.

C. Receiver is currently expecting p_k.

D. If p_{k+3} is sent, it is still unacknowledged.

E. None of the above

Extra Questions

Which of the following statement about TCP initial sequence number (ISN) is TRUE?

A. ISN is increased by 1 after sending every TCP segment.

B. In bi-directional communication, both directions of communication must choose different ISNs.

C. ISN determines the amount of data that can be transmitted over TCP.

D. ISN is randomly chosen between [0, 2^{32}-1], both inclusive.

E. None of the above

Extra Questions – super tricky!

A huge file is transferred over an existing TCP connection (i.e., 3-way handshake is already done). The connection is still open after transmission. The first and last TCP segments have the sequence numbers 12,345 and 2,105 respectively. MSS is 1,024 bytes and TCP sends as much data as possible in a segment.

How many TCP segments are used to transfer the file, assuming the communication channel is perfectly reliable?

A. 10

B. 4,194,294

C. 4,194,303

D. 33,554,360

E. 8,388,599

Thank you!

Answers:
A, E, D, E

Prepared by Clinton Law (AY19/20 Sem1)