CS339 Computer Networks Chapter 3 -- Transport Layer

1. Differentiate the Service Scales of Different Layer

- Application Layer: Provides service between **two applications**, knows nothing about the underlying transmission
- Transport Layer: Provides service between **two end systems**, knows nothing about the underlying transmission
- Network Layer: Provides service between **two given IP addresses**, knowing nothing about the underlying transmission
- Link Layer: Provides service between **two adjacent hops/devices in the network**, knowing nothing about the transmission method

2. Different Layers' Transmission Unit

- Application Layer: Message
- Transport Layer: Segment
- Network Layer: Packet
- Link Layer: Frame
- Physical Layer: Bit

3. Ports

3.1 What is Port?

The ports of a host/device is defined using 16-bits binary number, which means one device can have 65536 ports. But by convention, only a few of them are frequently used (0~1023, called well-known port numbers). Port number represents the local address of the process. Namely, one can interpret it as the address of the process in the host.

3.2 How Applications are Differentiated (with my assumption)?

Take `HTTP` as an example, if the applications are not persistent, then they are uniquely defined by a `(IP address, port number)` pair. If the applications are persistent, which means the spawned processes happen at the same time, they will use the same `IP address` and `port number`. But the `target IP address` and `port number` should be different. Actually, the spawned processes are the **sub-processes of the main process**, so they use the same port number. For other applications, they will have their own port number. Like `FTP(20/21 One for control and the other for data)`, `SMTP(25)`. **So the same service will usually have the same port number**.

4. Multiplexing

4.1 Upward Multiplexing

This is the situation when there is only one NIC in that host, then all the applications will use the IP address.

4.2 Downward Multiplexing

This is the situation when there are more than one NIC in that host, then it can use multiple IP addresses to transmit the information. This is usually seen in high-bandwidth channels.

5. Demultiplexing

We can understand the multiplexing as: **collect information from different sockets and then send them out**.

We can understand the demultiplexing as: **direct the information from port to the right sockets**.

Explanation From Textbook:

The job of **delivering the data in a transport layer segment to the correct socket** is called **demultiplexing**.

The job of **encapsulating the data from sockets to create segment, and sending them to the network layer** is called **multiplexing**.

6. UDP Multiplexing and Demultiplexing

The socket in UDP can be represented by `(IP address, port number)` pair. Each process has only **one socket**. When sending message, the transport layer will multiplex the data from different sockets and send them out. The receiver will then collect the segment and demultiplex them into different sockets using the `(destination port number, destination IP address)` pair.

7. TCP Multiplexing and Demultiplexing

The socket in TCP can be represented by `(source IP, source port, destination IP, destination port)`. Each process has only **one socket**. It is possible that two processes have the same `(source IP, source port)` pair, but different `(destination IP, destination port)` pair.

8. Conclusion for Multiplexing and Demultiplexing

The golden rule is: **One process has only one socket, the source port number depends on the protocol.**

9. HTTP Connection

`HTTP` is a special service for the multiplexing and demultiplexing process. When a new `HTTP` connection is set, it will usually not create new `TCP` connections (non-persistent connection), since it is very slow. It will **spawn a new subprocess** in the main process. It will create new sockets for these subprocesses, thus for `HTTP` there **may not be one to one correspondence between socket and process**.

9. UDP Connection

9.1 Only Add Four Fields

`UDP` only adds 4 fields to the application data, `source port, destination port, length, checksum`. The `IP` information is dealt by network layer. Thus the `UDP` header is very small.

9.2 Why Use UDP?

UDP is **faster for short message exchange** and **loss tolerant applications**. Due to:

1. Header is small.
2. No congestion control.
3. No connection state is built.
4. Finer-application layer control.

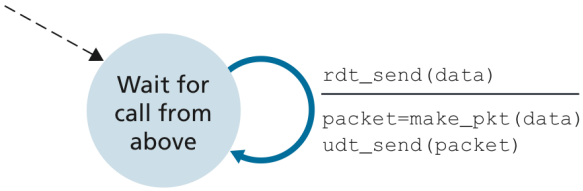
9.3 UDP Checksum

To calculate the checksum of UDP, just adding all the binary numbers together (**using wrapped around, namely, adding overflow to the end of result**). Then get all the digits inversed (1s compliment). This way, adding checksum to the result of addition will give all **1**. If not, there is a wrong bit. Note that UDP does not solve this problem, it only reports this problem.

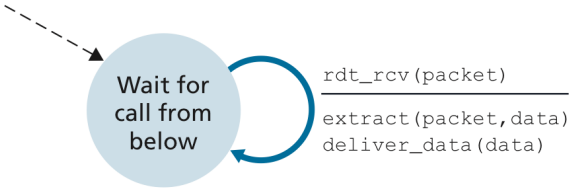
10. Reliable Data Transfer (rdt)

10.1 rdt1.0: Perfectly Reliable Channel

In this case, since the underlying layer has no error, we can just make the sender and receiver as follows:



a. rdt1.0: sending side

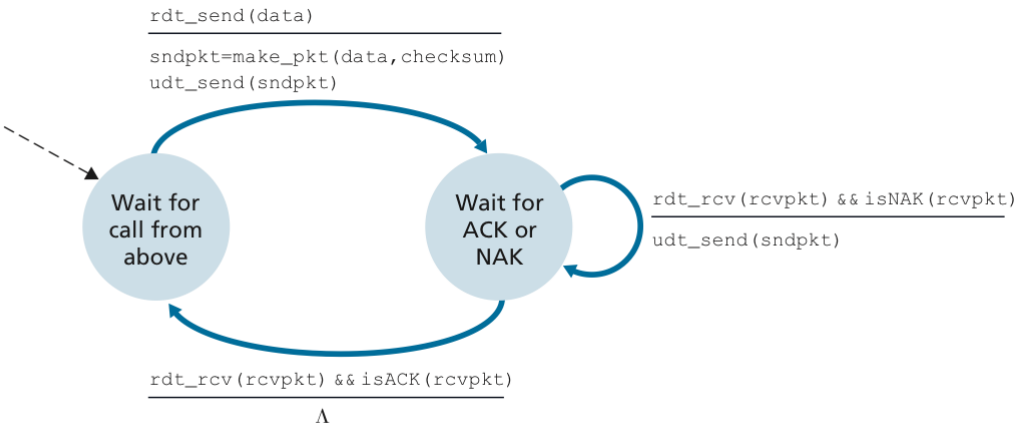


b. rdt1.0: receiving side

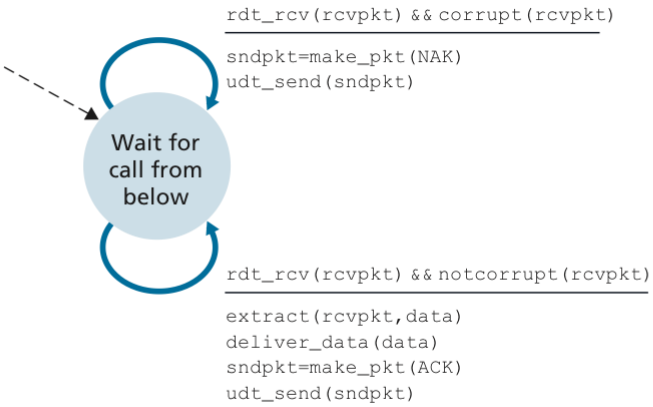
Nothing hard here.

10.2 rdt2.0: Consider Bit Error Happening to Data -- Introduce `ACK/NAK` and Retransmission

In this case, the receiver will repeat with `ACK` or `NAK` to the sender. If the packet arrives with some wrong bits, the receiver will repeat `NAK`. If the packet arrives correctly, the receiver will repeat `ACK`. Then the sender will decide whether to retransmit the packet or not:



a. rdt2.0: sending side



b. rdt2.0: receiving side

10.3 rdt2.1: Consider Bit Error Happen both to Data and `ACK/NAK` -- Add Seq. Number for Packet

In this case, we need to add more states. The sender will treat the corrupted `ACK/NAK` as if `NAK` (to prevent it is indeed a `NAK`), and retransmit the packet. The sender will only continue to send the next packet if a **non-corrupted** `ACK` is received. Meanwhile, the receiver will stay in the current state if it receives a corrupted data (return `NAK` as 2.0), or receives a non-corrupted data but with different seq. number (retransmission data, return `ACK` and stay). It will only enter the next state when there is **a non-corrupted packet with right seq. number** is received. **Note that a checksum is added for checking correctness**:

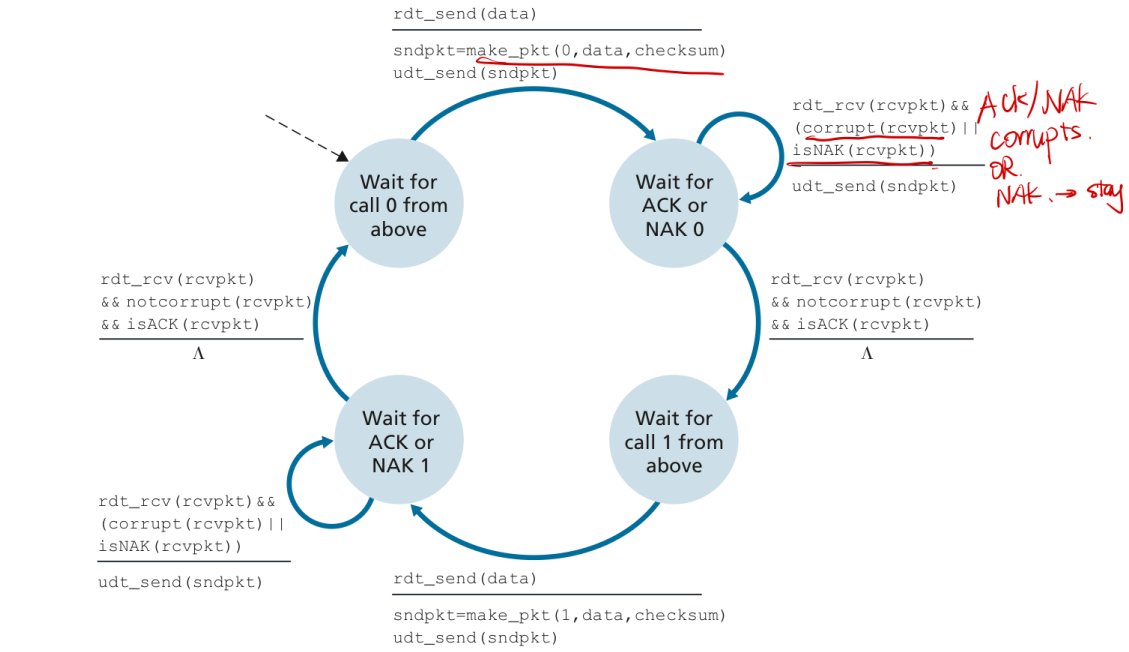


Figure 3.11 ♦ `rdt2.1` sender

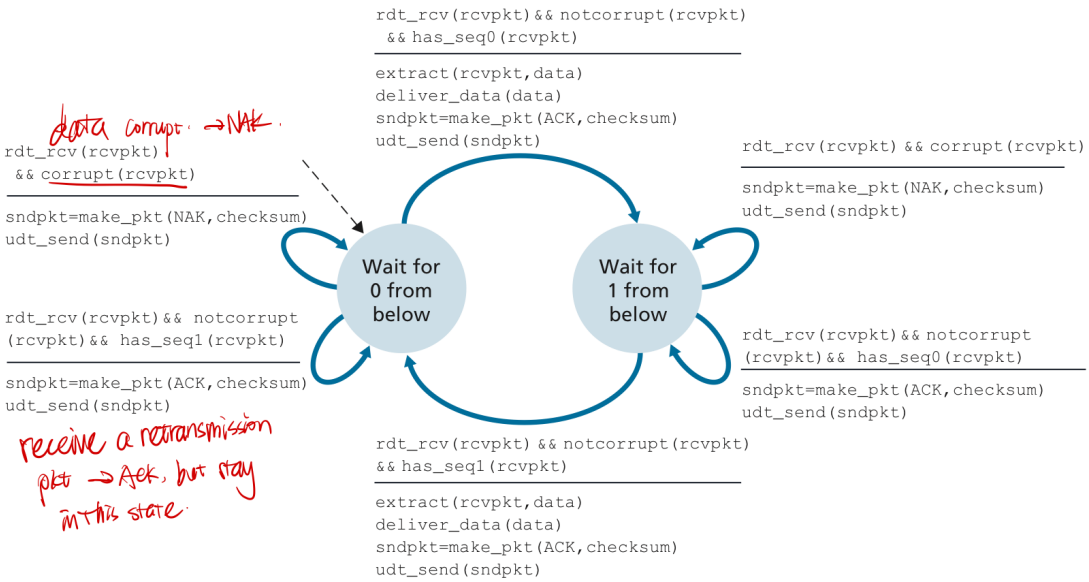


Figure 3.12 ♦ `rdt2.1` receiver

10.4 rdt2.2: No `NAK` -- Add seq. number to `ACK`

In rdt2.2, it uses two duplicated `ACK` to represent packet corruption/receiving retransmission packet. Namely, it combine the retransmission packet case with the corrupt packet case. Thus, the corrupt packet case will no long use `NAK`, but the **`ACK` of last packet** as return message.

When sender receives packet with certain `ACK` number, if the sender is indeed doing retransmission (`ACK` number is right), it will go to transmit the next packet. If the sender is sending another packet (`ACK` number is wrong, i.e. duplicated with the former `ACK`), it knows this packet fails to reach receiver correctly, and it will resent it.

10.5 rdt3.0: Consider Packet Loss -- Add Timer

In this case, we consider that the packet can be fully dropped in the network. One of the strategy is adding a **timer**, no matter the packet or `ACK` is lost or they are just overly delayed, just **retransmit when timer expires.** This may cause duplicate packet, but the seq. number duplication can be handled by rdt2.2:

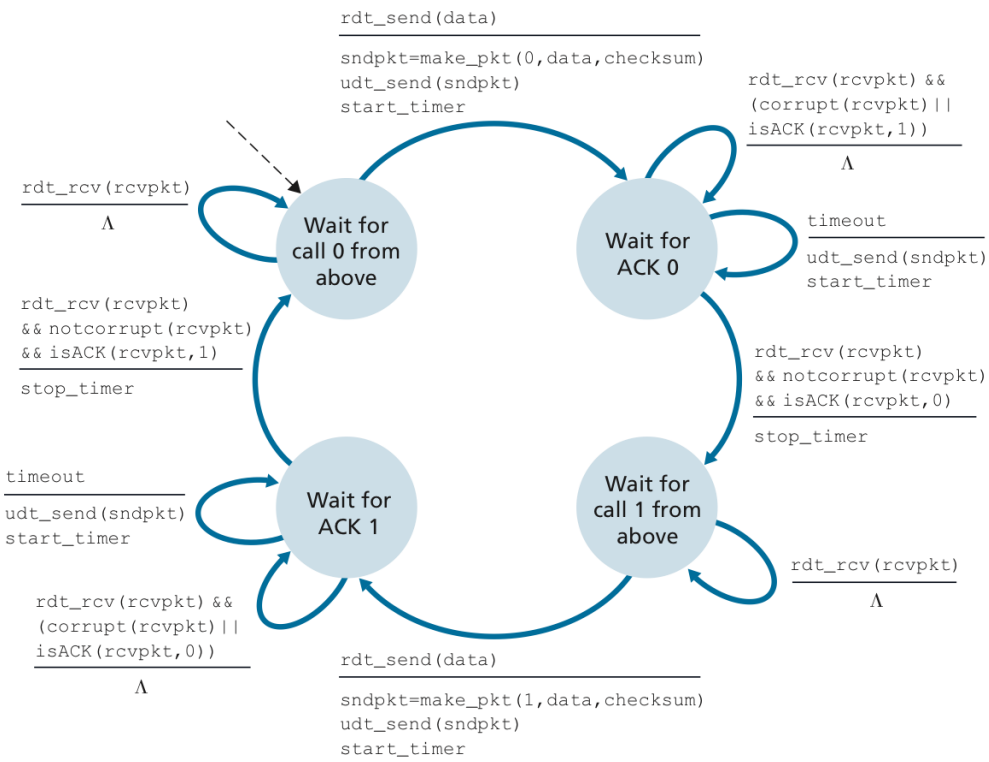


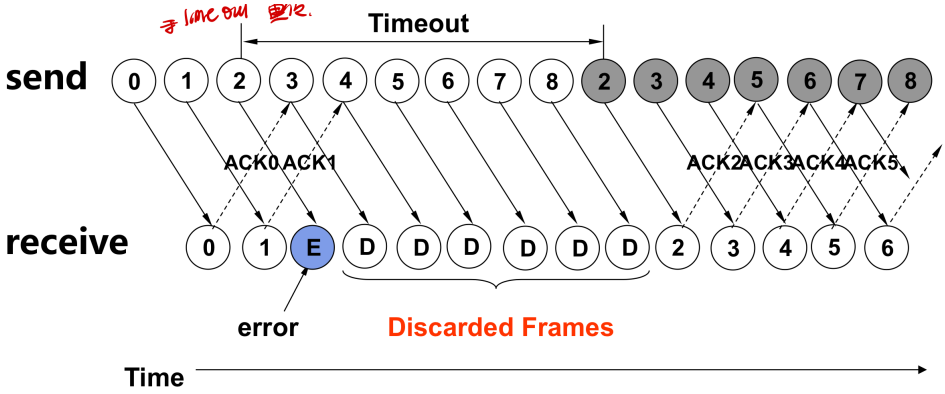
Figure 3.15 ♦ `rdt3.0` sender

11. Pipelined Reliable Data Transfer Protocols

rdt3.0 is still a **drop and wait protocol**, thus it is slow. Thus we transmit packets in a pipeline. When there are wrong bits/loss in packets, we need to retransmit them.

11.1 Go-Back-N

When the receiver fails to receive packet k, it will discard all the packets after k. The timer for k on sender side will expire, packet k and packets after k will be retransmitted. Note that when a `ACK` arrives, the timer will update itself to the next packet in the air. The name of "Go-Back-N" means that the sending window is `2^n-1` bits long (with the seq. number to be `n` bits):



One detail: When receiver receives a frame that is going to be discarded, it will return a `ACK` corresponding to the last packet it received. For example, for the diagram above, the receiver will actually send `ACK1` when it receives packet 3-8.

11.2 Selective Repeat

The selective repeat method will require both the sender and receiver have the **same window size**. In the case of packet loss, the receiver cannot send the packet to the upper layer. It buffers the packets after that packet. Then the timer will expire at the time when the sliding window on the sender side moves to that packet place. **The packet is now resent**. The receiver will deliver all the packets in the window **once in all** and move the window directly to the next unreceived packet.

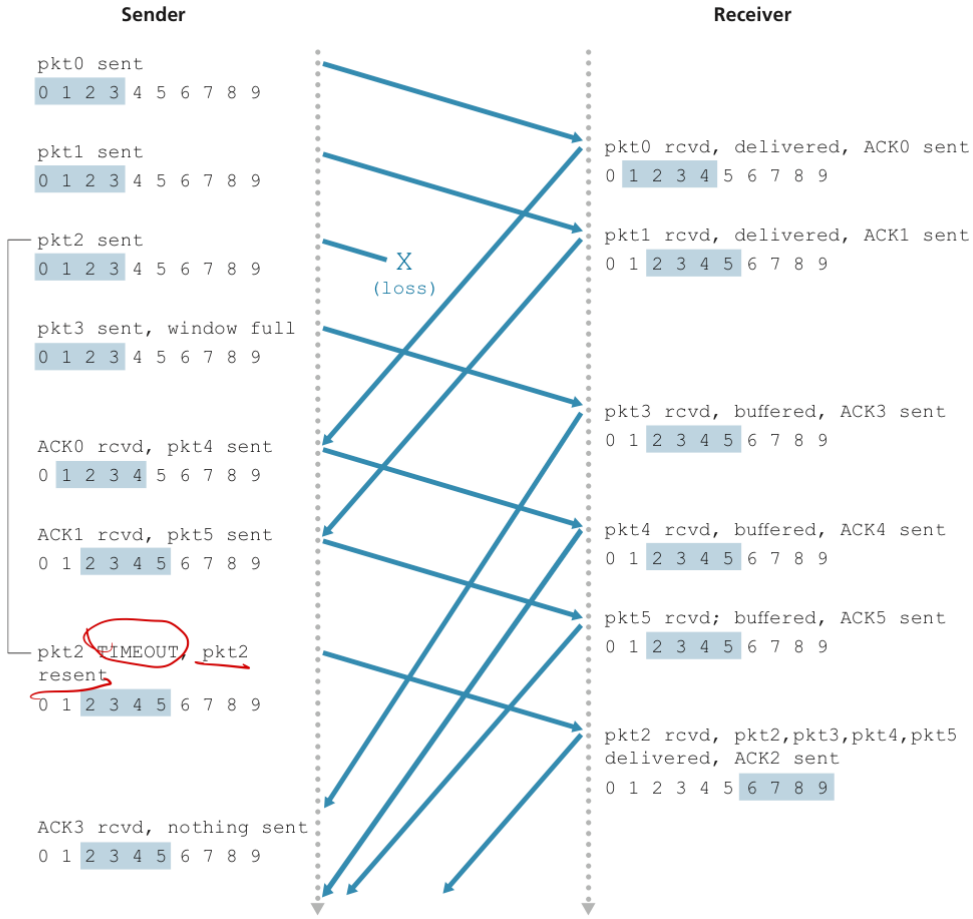


Figure 3.26 ♦ SR operation

One detail: The sliding window size must be smaller than half of the # of seq. number. Otherwise, there will be cases when losing different packets makes no difference.

12. Error Correction

The ways of error detection have been discussed in rdt part, including `ACK`, seq. number, timer and retransmission. Here we consider to directly correct the error instead of retransmitting again, using the redundant information like checksum.

12.1 Parity Checking

Parity Checking just adds one bit of parity bit to the data. It ensures that the data and the parity bit together has odd number of `1`. So if there is odd number of wrong bits, the total number of `1` will not be odd, the error **is detected, but cannot be fixed**. To fix the problem, we need a two dimension matrix with each row and column having a parity bit. So we can find the wrong bits with the coordinates.

12.2 Checksum

Checksum is calculated by adding all the data with (wrapping around skill), and inversing all the bits. So when adding all the data and checksum together, if there is a bit that is `0`, the bit will be the wrong bit. But it is possible that there are multiple wrong bits and the checksum may not find the wrong bits.

12.3 CRC

CRC is similar to checksum. It adds remainder of `G(x)(generator fo degree r)` to the end of the data. This remainder can calculated as follows:

1. Append **degree of G(x) (e.g. 4 if the divider has 5 digits)** to the end of the frame
2. Do long division with **XOR subtraction**.
3. The remainder must be **degree of G(x)** bits long.

When checking, if the data together with the added remainder can be divide by `G(x)`, then there is no problem.

12.4 Hamming Code

https://www.bilibili.com/video/BV1tL4y1h7Fd?from=search&seid=6864231166590858 26&spm_id_from=333.337.0.0

https://www.bilibili.com/video/BV1tf4y1A7NX/?spm_id_from=333.788.recommend_more_video.-1

Hamming code can only correct **one** bit error.

13. Flow Control

Here we just give the formula for the utilization of client:

$$U_{sender} = \frac{\#\text{ of packets in window} \times L/R}{RTT + L/R}$$

If we keep the window

$$W = RTT \times R$$

The pipeline is filled by packets, the efficiency is `100%`.

14. TCP Segment Structure

Focus on the source port number, destination port number, sequence number, acknowledgement number and window size.

14.1 About the Seq. Number and Acknowledge Number

The Seq. number is the seq. number of the current packet, the `ACK` number is the seq. number the sender expects to get back.

15. TCP Connection Establishment and Release

15.1 Three Way Handshake

The purpose of three way handshake: The first packet means the client is ready to connect, the later two handshakes are used to make sure both hosts can hear each other.

15.2 Four Way Hand Wave

Actually the three way handshake combines one of the packet. The four way hand wave contains:

1. Client send `FIN` to server, indicating it is about to leave. Server replies `ACK`.
2. Server send `FIN` to client, indicating it is about to leave. Client replies `ACK`.
3. Both client and server wait for some time (by keeping a timer) for potential packet loss. If not, shut down connection.

16. TCP Reliable Data Transfer

16.1 Hybrid of GBN and SR

16.2 TCP Fast Retransmission

The retransmission happens when:

1. The timer (TCP keeps a timer for the oldest packet in the air) expires
2. When the sender receives three duplicate (four in total) `ACK`.

The retransmission will only resend one packet.

One extra feature is the piggyback/cumulative `ACK`. When the `ACK` for the next packet is received by sender, it will know that the packet he sends before have already been received, no matter the `ACK`s have arrived or not.

16.3 Timer Management

$$\text{timeout} = RTT + 4D$$

`RTT` is the best `RTT` currently. `D` is the estimation of deviation of `RTT`. These two values are updated using the old value and the current measured value. Namely

$$\text{new value} = \alpha \cdot \text{old value} + (1 - \alpha) \cdot \text{current measured value}$$

17. TCP Flow Control

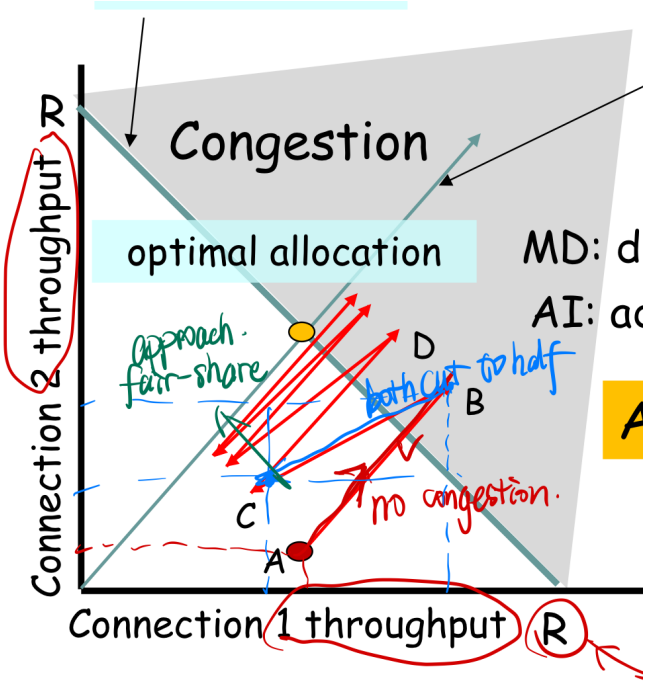
Receiver can use the window size area in the TCP header to indicate that there is no space for the receiving window. If the sender receive this packet, it will **stop its transmission**.

Then the receiver can send the packet again to indicate that there are now some space in the receiving window. This time the sender will restart transmission. If this particular packet is **lost**, the sender may wait for some time and **send a small packet**. If he then receives an `ACK` normally, then he can restart the transmission.

18. TCP Congestion Control

18.1 Bandwidth Allocation-- AIMD

TCP uses the strategy of congestion avoidance, while keeping the bandwidth fairly shared by the users. Thus, there Addictive increase, multiplicative decrease method is used to reach the fairness. That is, it **increases the data transmission rate linearly when there is no congestion**, it **decreases the data transmission rate multiplicatively when it meets congestion**. In this way, original links with high rate will be decreased more than the links with low rate. It will finally reach fairness.



18.2 Congestion Control Algorithm (TCP Reno as example)

1. **Slow Start:** Window size doubles each time until the threshold.
2. **Congestion Avoidance:** Window size grows linearly until meets congestion.
3. **Fast Recovery/Slow start:** If there are three duplicated `ACK`, both the window size and the threshold halves. It then grows linearly, namely, does congestion avoidance. If there is a timeout, the window size becomes 1 while the threshold still halves. It then does the slow start again.

Note: TCP Tahoe will set the window size to 1 and threshold to half of original size no matter there are duplicated `ACK` or packet loss.

19. Fairness

TCP uses Max-min Fairness. It provides fairness in connection, but not in application.

Homework and Quiz Review

Question 1

1 / 1 pts

Suppose that the last SampleRTT in a TCP connection is equal to 1 sec. Then Timeout for the connection will necessarily be set to a value >= 1 sec.

- (●) True
- ( ) False

Explain: calculate the RTT as follows:

$$\mathrm{RTT} = \alpha \cdot \mathrm{RTT} + (1-\alpha) \cdot M = \alpha \cdot \mathrm{RTT} + (1-\alpha)$$

calculate the D as follows:

$$D = \alpha \cdot D + (1-\alpha) \cdot |\mathrm{RTT} - M| = \alpha \cdot D + (1-\alpha) \cdot |\mathrm{RTT} - 1|$$

The timeout can thus be calculated as:

$$\mathrm{timeout} = \mathrm{RTT} + 4D = \alpha(4D + \mathrm{RTT}) + (1-\alpha)(4|\mathrm{RTT} - 1| + 1)$$

We only need to consider the original RTT is smaller than 1. Then it becomes

$$\mathrm{timeout} = (5\alpha - 4) \cdot \mathrm{RTT} + 5(1-\alpha)$$

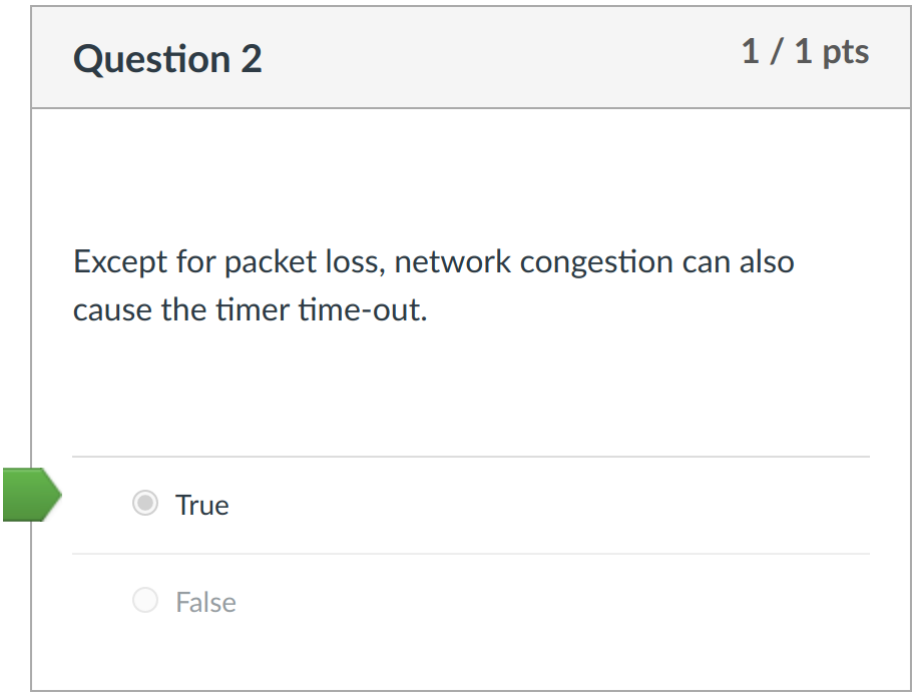
consider

$$\begin{cases} \alpha \to 1 \\ \mathrm{RTT} \to 0 \end{cases}$$

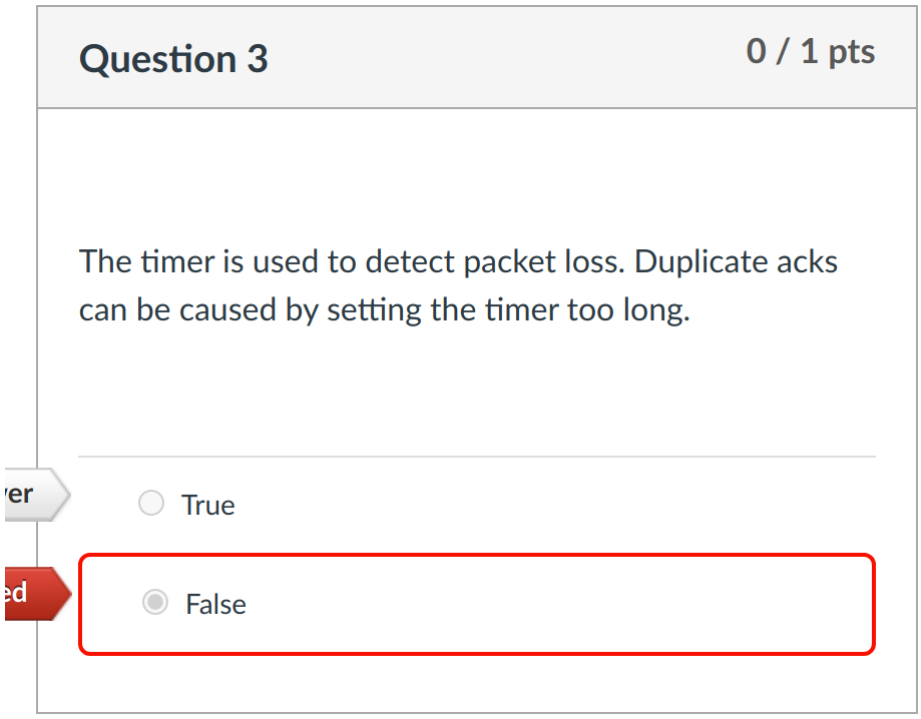
then

$$\mathrm{timeout} \to 0$$

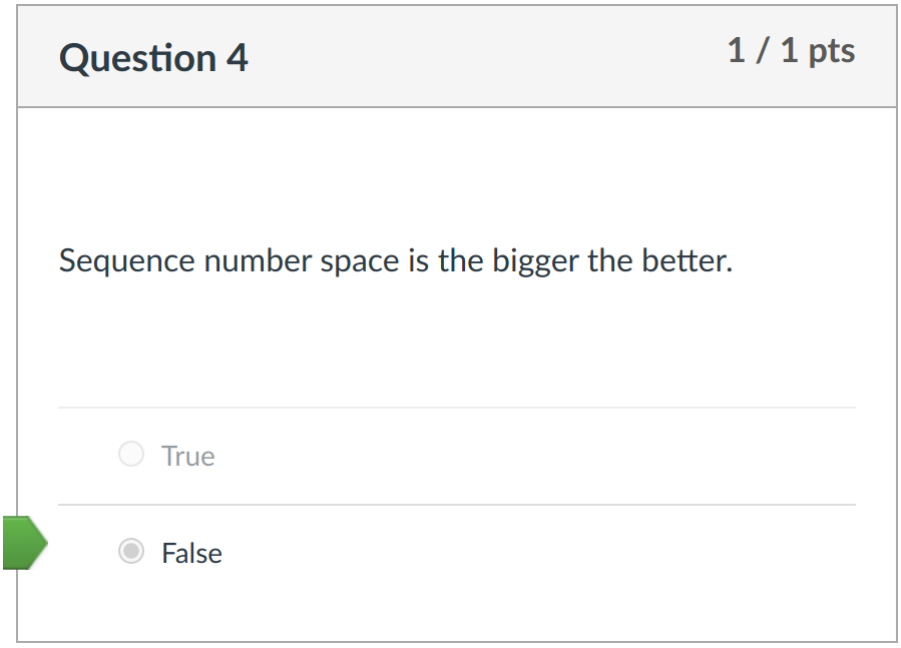
The answer for this question is **Wrong!**



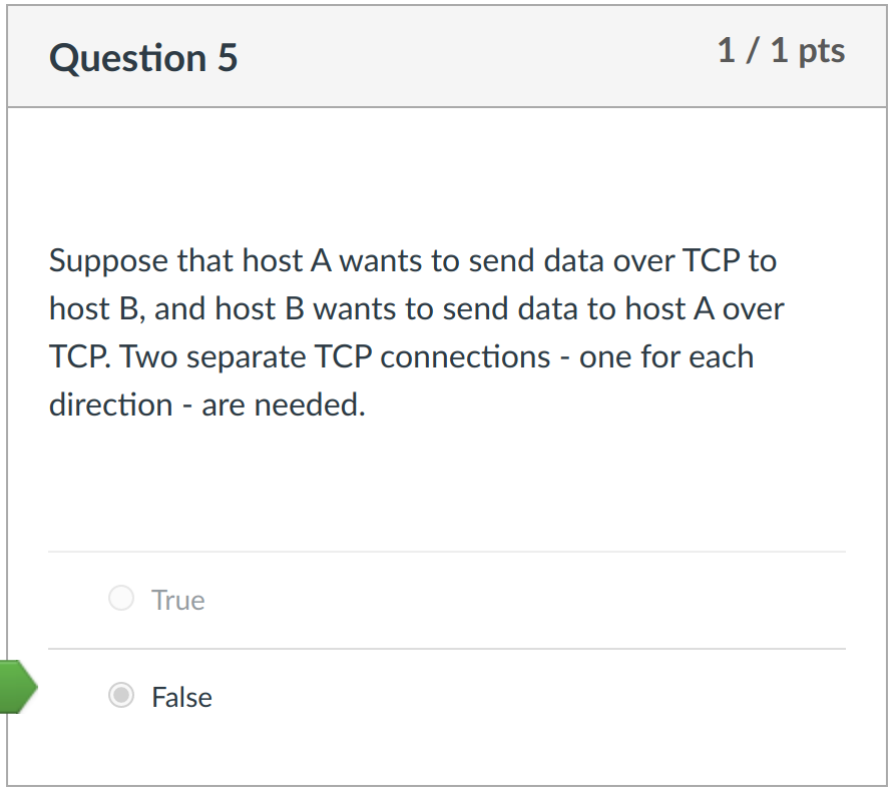
Explain: Both timeout and three duplicated `ACK` are considered as congestion in TCP.



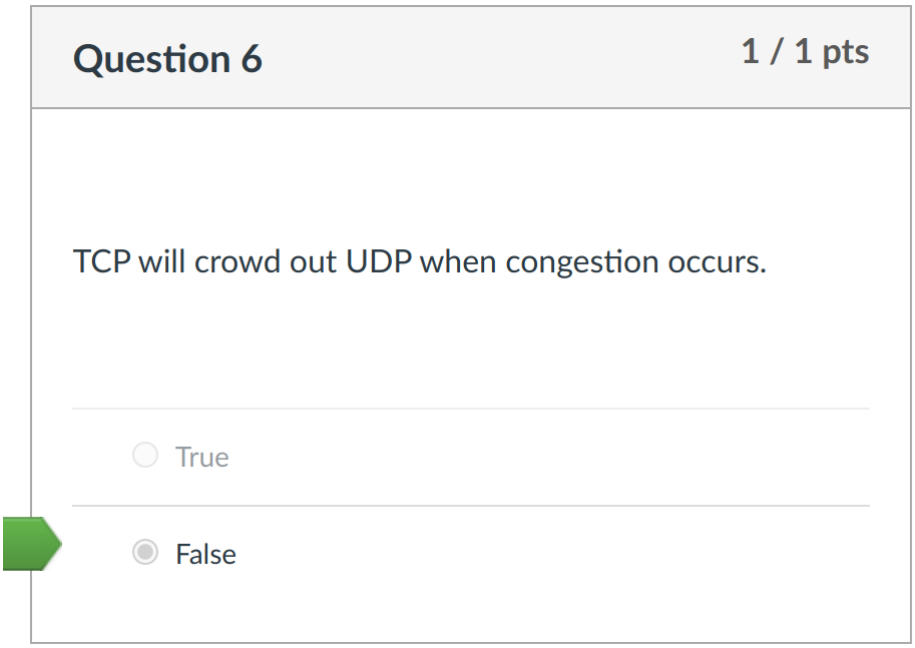
Explain: The best timer time is `2RTT+L/R`, so the sender will immediately resent the packet if it receives no `ACK`. However, `RTT` is unpredictable and the timer may be shorter than ideal case or longer than ideal case. For the shorter case, it just regard it as packet loss **with no duplicate packets, namely, timeout.** For the longer case, it will receive **duplicate packets**. If it receives three duplicate packets, it will regard it as packet loss as well.



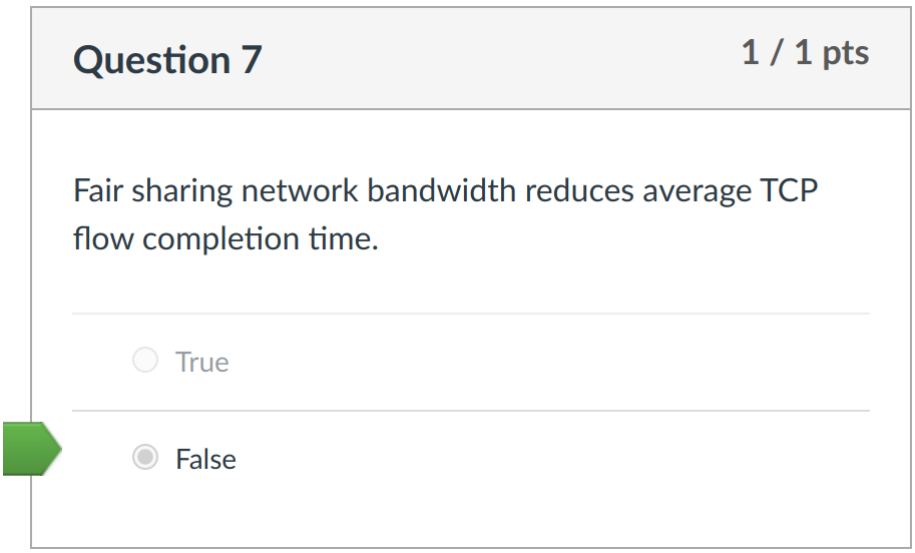
Explain: Definitely not, the header will be longer in that case.



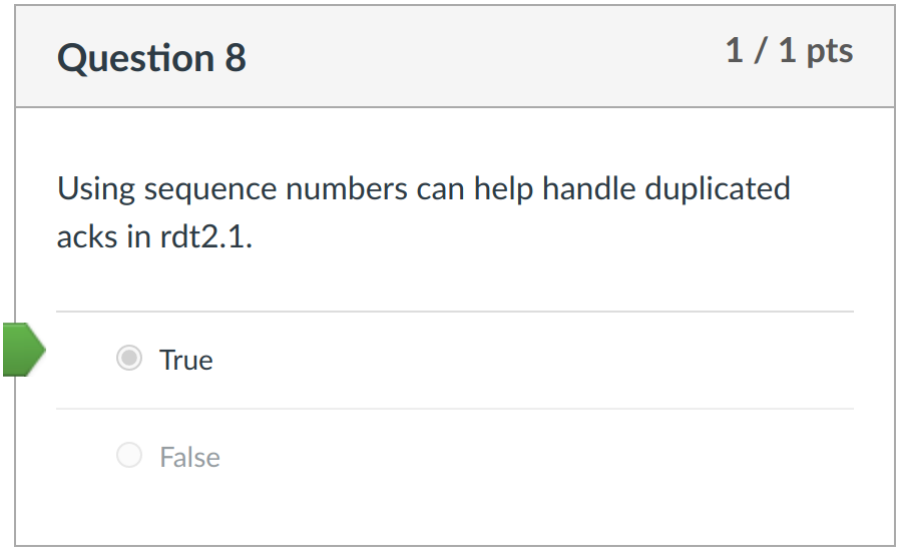
Explain: One TCP is enough, TCP is double-side link.



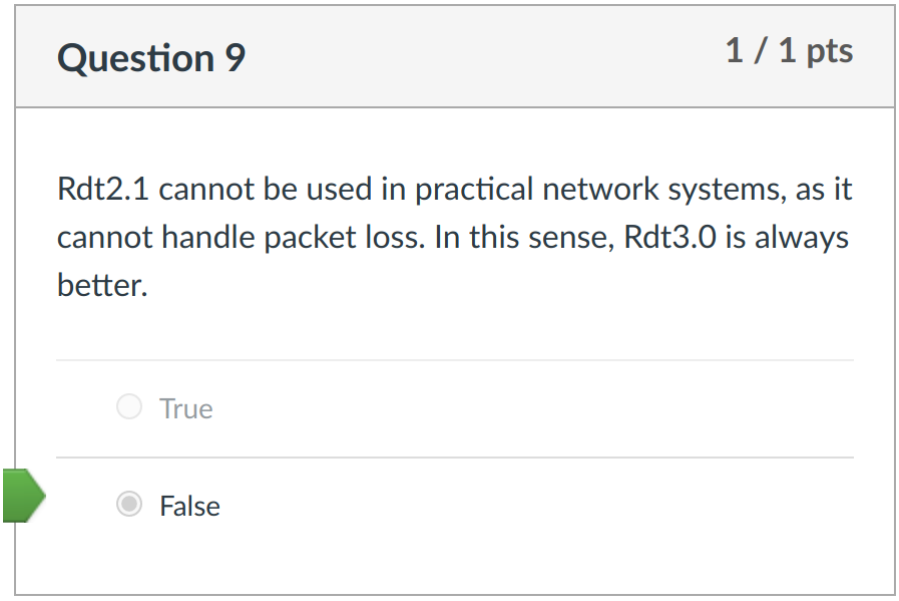
Explain: Since UDP provides best effort service, it will crowd TCP out. TCP has congestion control algorithms.



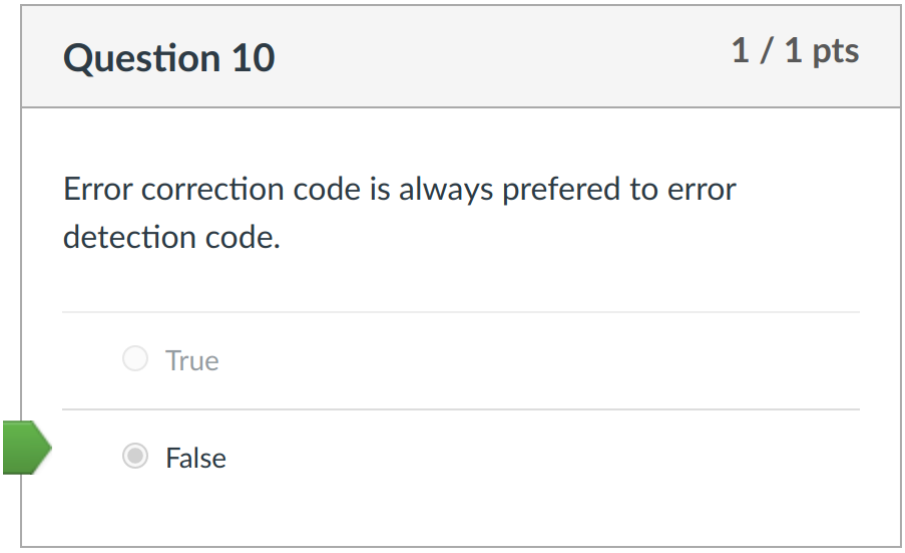
Explain: One thing we need to pay attention is that, the Max-min Fairness of TCP actually increase the average TCP flow completion time. Since the bandwidth is divided and the transmission will be slower than the "dedicate transmission".



Explain: **Weird!** Here my understanding is that, the corrupt `ACK` will be detected by sender. The sender will send a packet with the same sequence number. The receiver will then send the `ACK` again.



Explain: rdt2.1 is better than rdt3.0 in lossless network like RDMA. rdt3.0 introduces timer, which is not necessary in lossless network.



Explain: Error correction takes effort. For UDP, error detection is sufficient and is thus preferred.