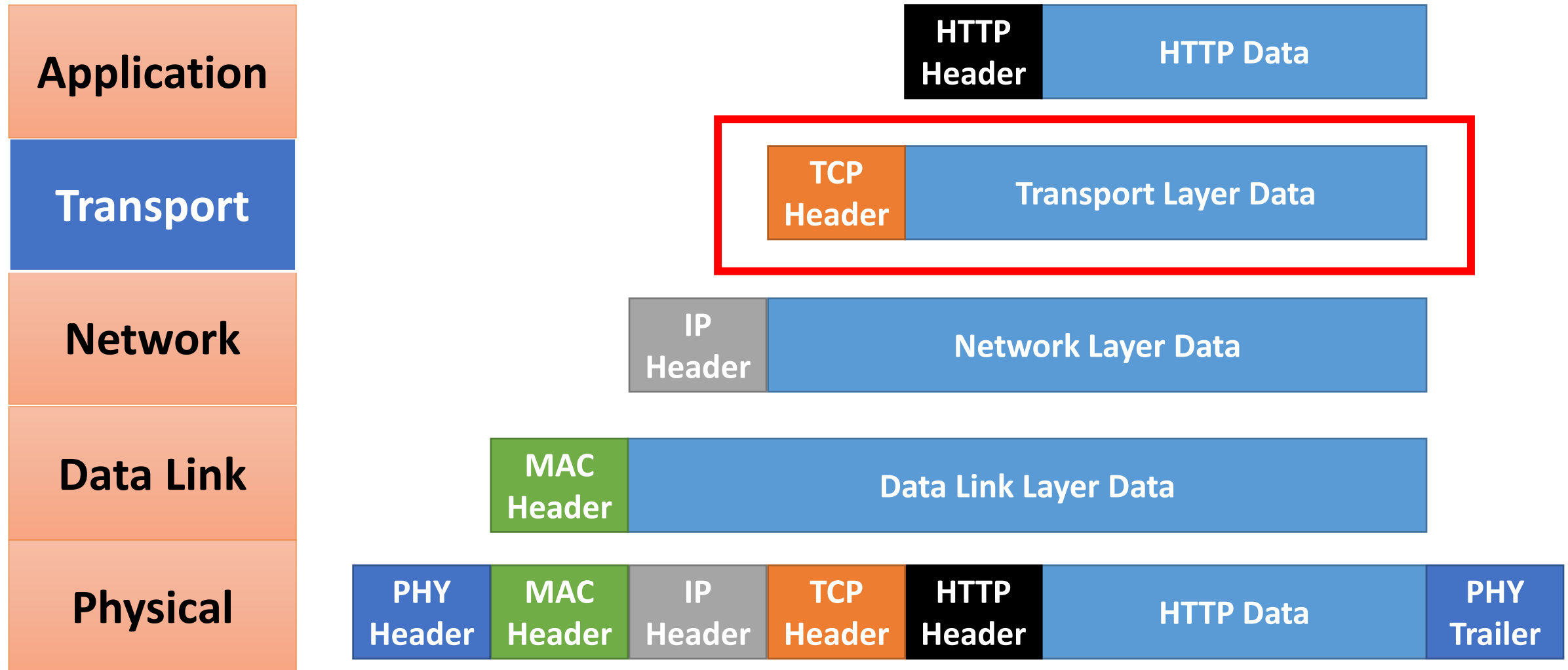


# How Application Data Passes Through Different Layers



# Transport Layer Services

End to end  
packet delivery

Connection  
Establishment

Reliable Data  
Delivery

Flow and  
Congestion  
Control

Ordered Packet  
Delivery

**UDP**

**Transport**

**TCP**

Datagram delivery (unreliable)

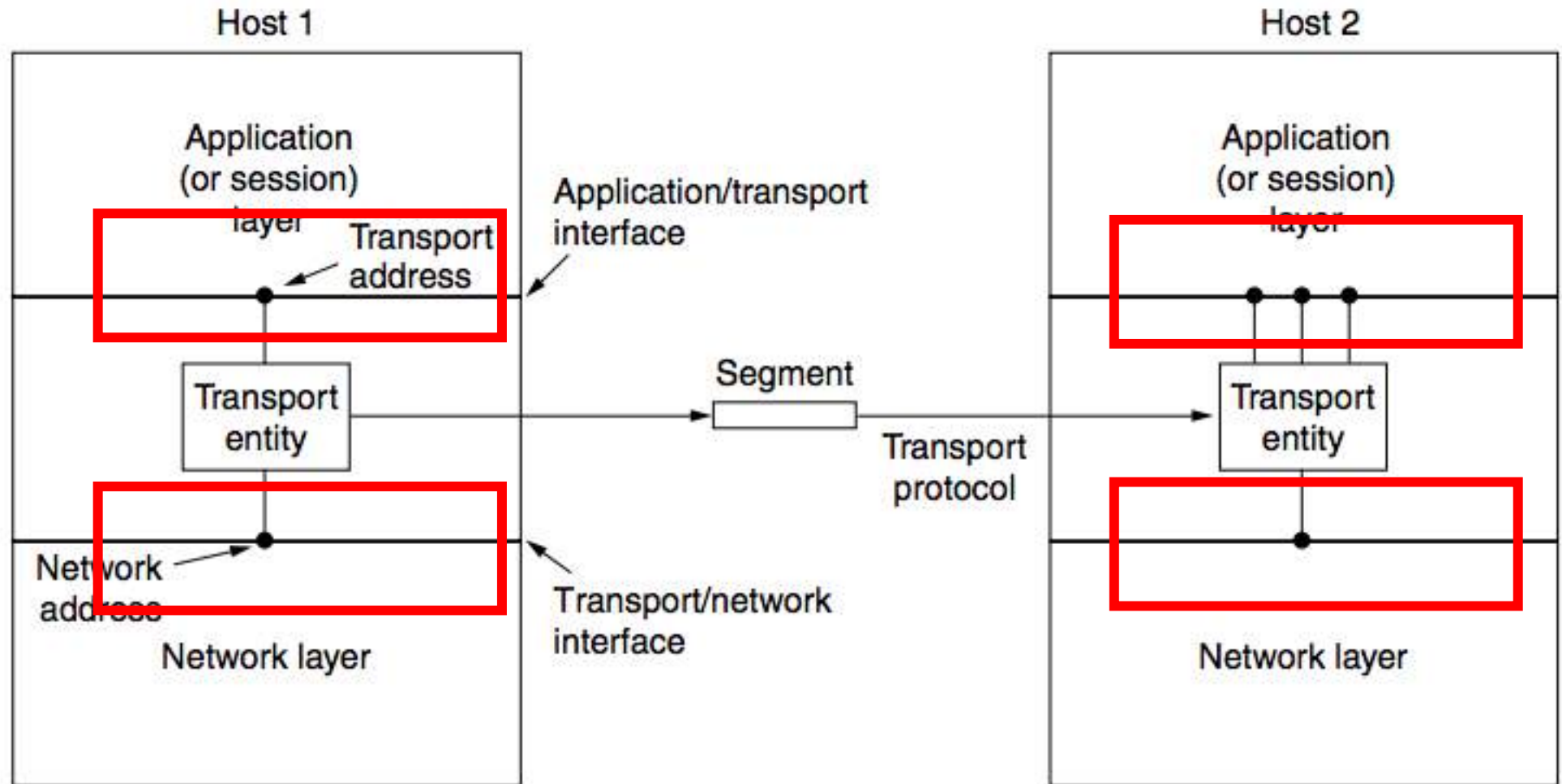
**Network**

**Data Link**

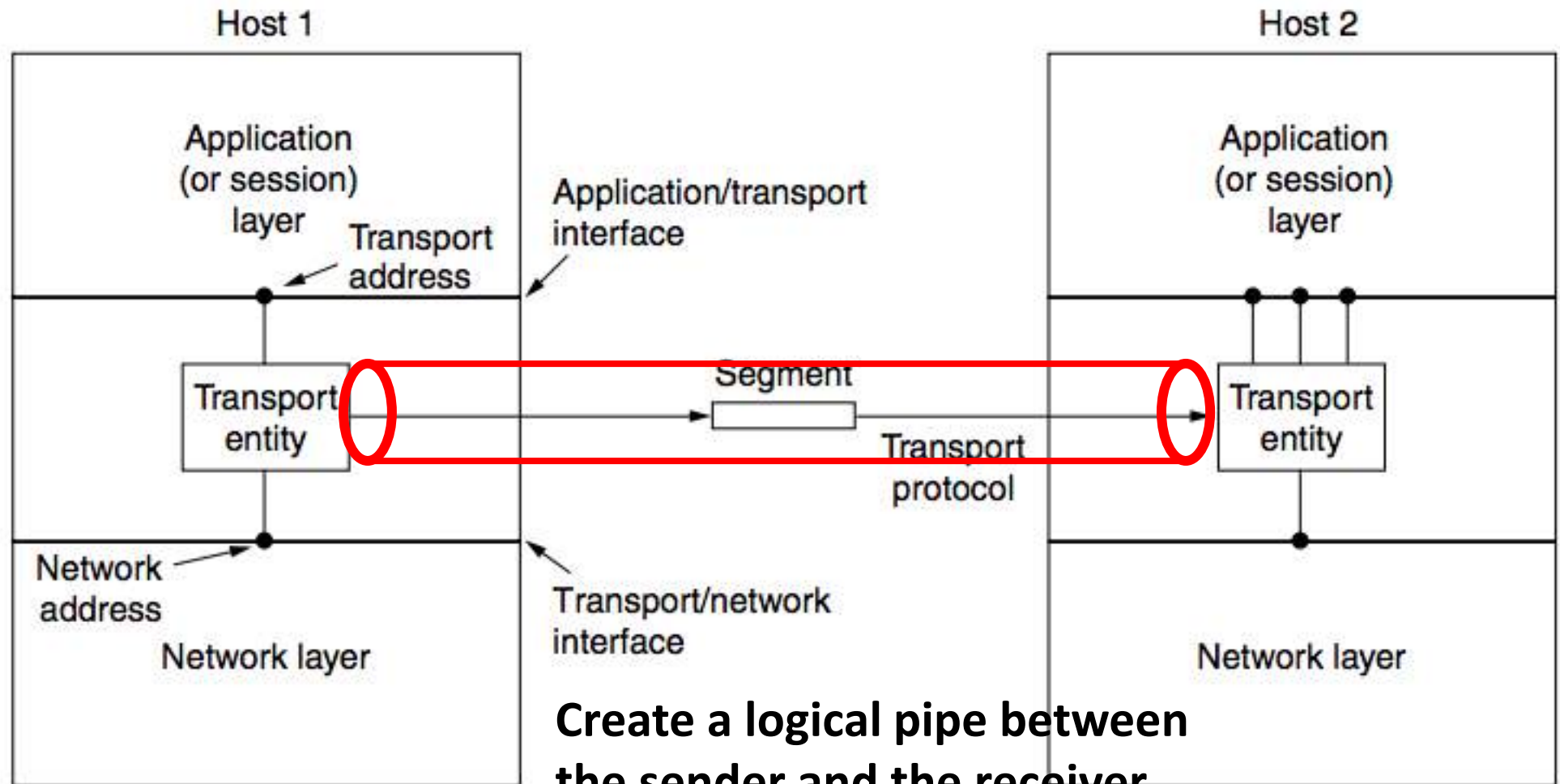
# Transport Layer – Interfacing with Application and Network

Port Number

IP Address



# Transport Layer – Interfacing with Application and Network



**Create a logical pipe between the sender and the receiver and **monitor the data transmission through this pipe****

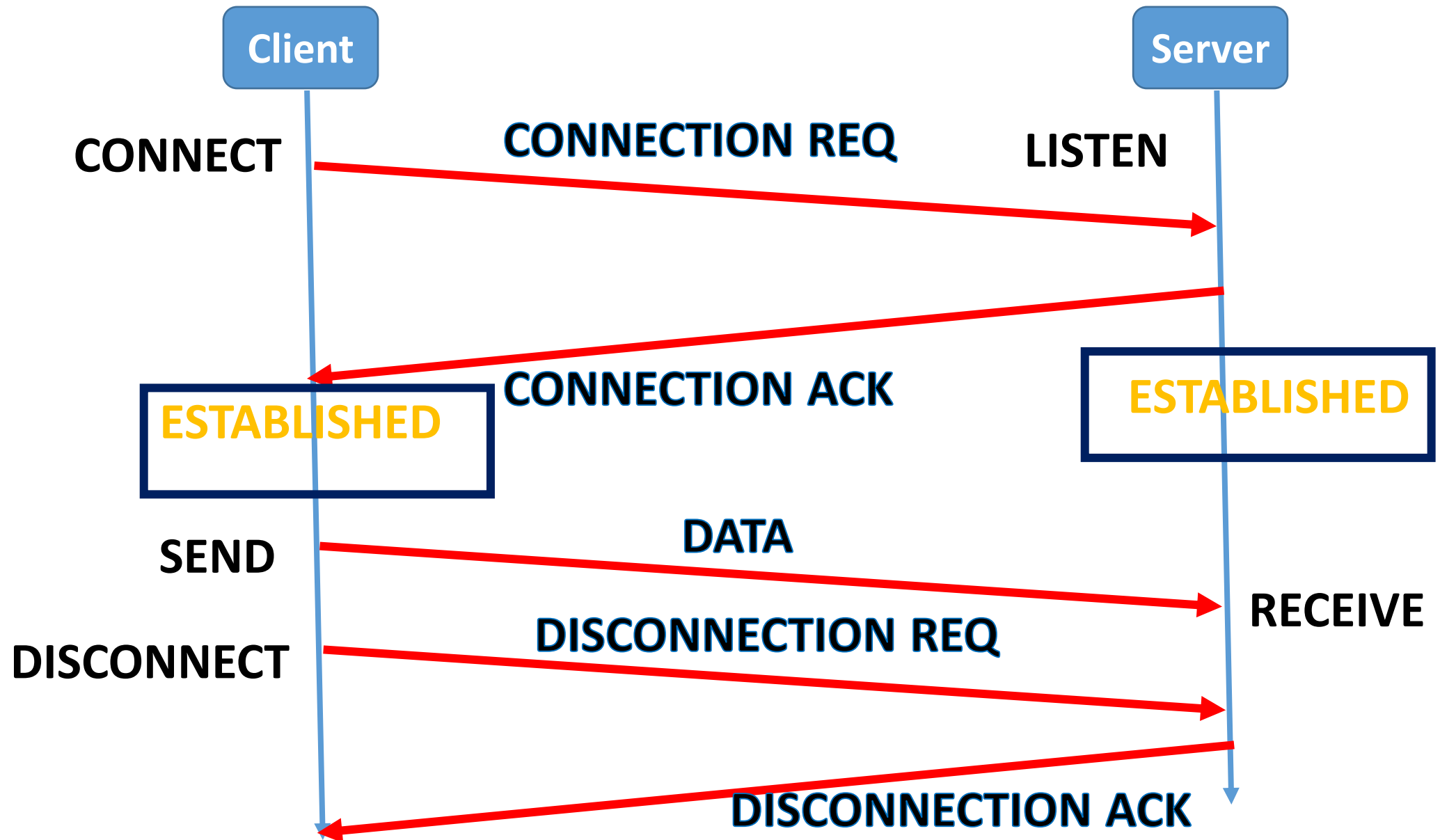
# Transport Service Primitives

- To allow users to access transport service, the transport layer must provide some operations to the application programs.
- Let us look into a hypothetical transport service primitives, that are provided to the application layer

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	Request a release of the connection

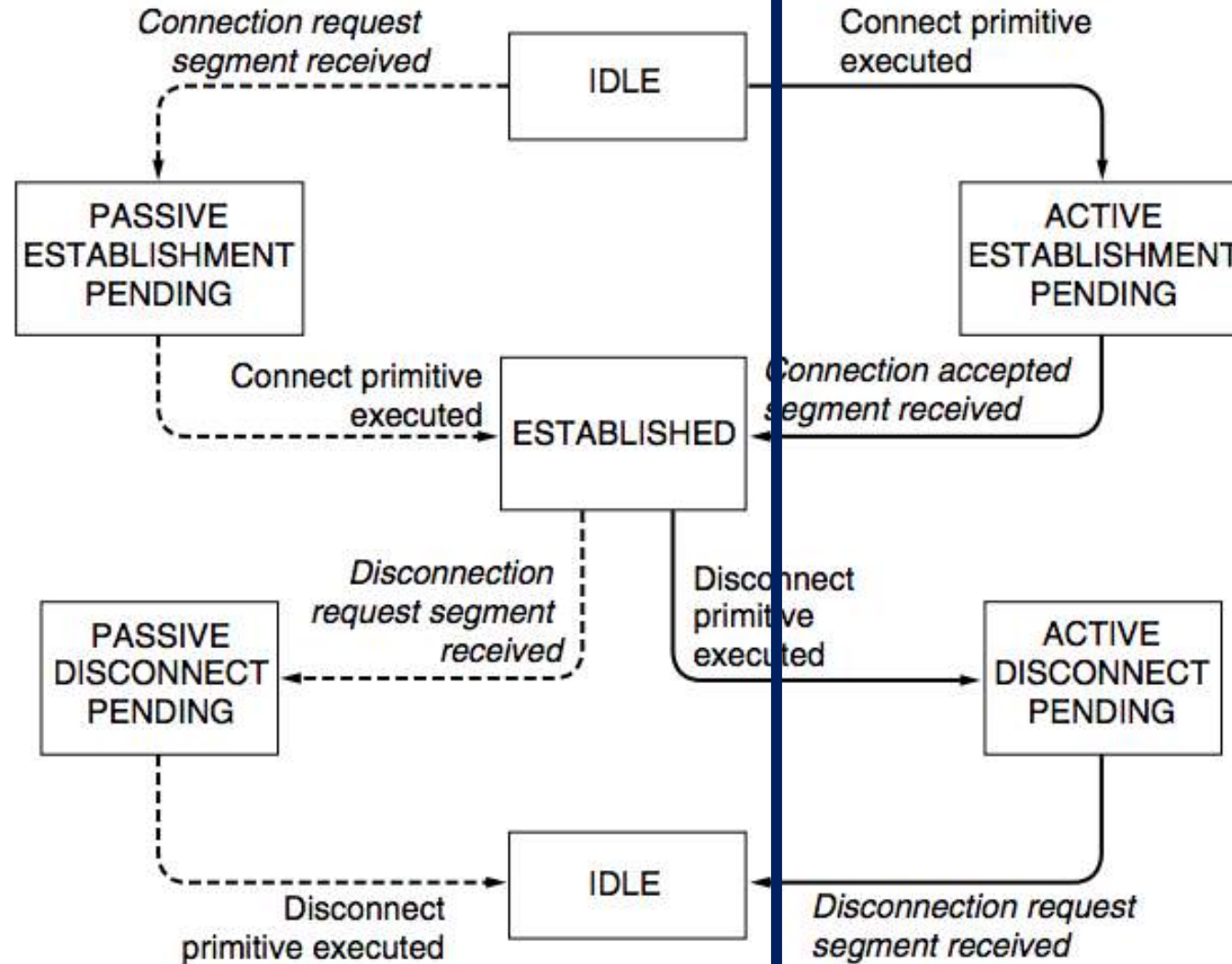
The transport layer needs to remember the state of the pipe, so that appropriate actions can be taken. We need a **stateful protocol** for transport layer.

# Transport Service Primitive – Connection Establishment



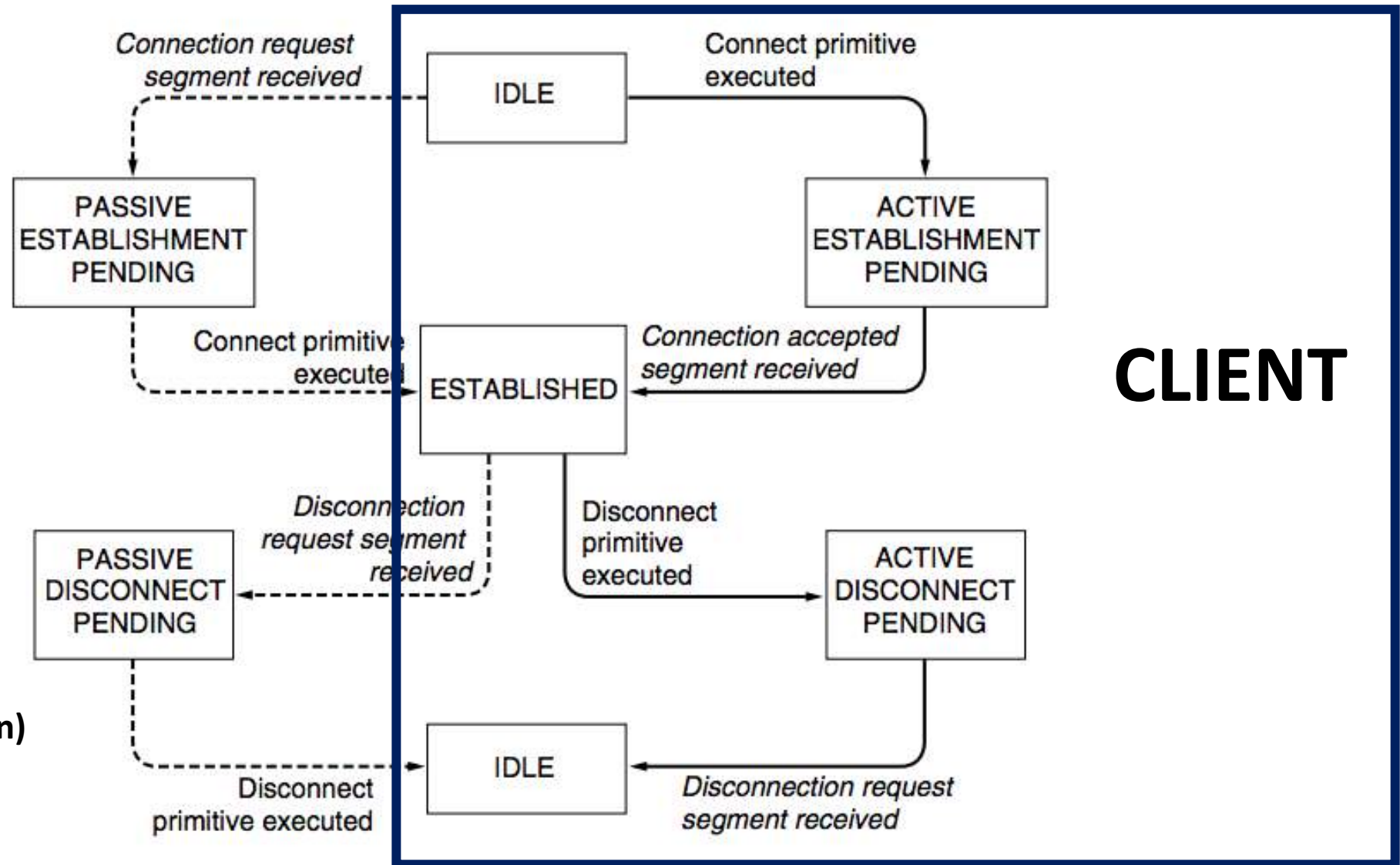
# Transport Layer Protocol – State Diagram

**SERVER**





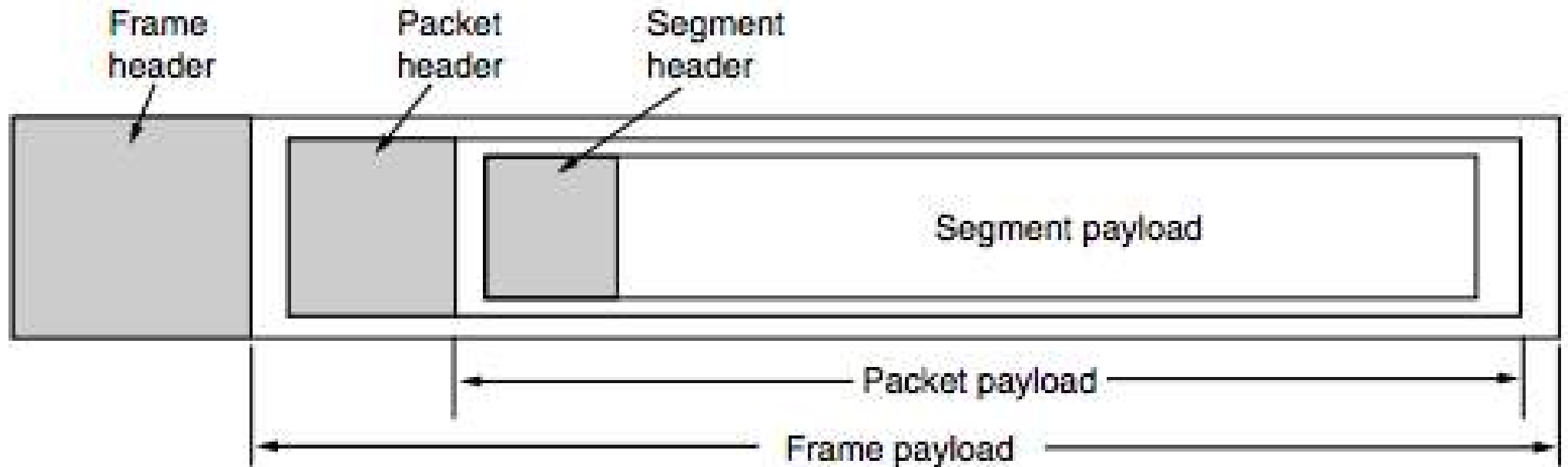
# Transport Layer Protocol – State Diagram



Source: Computer Networks (5<sup>th</sup> Edition) by Tanenbaum, Wetherell

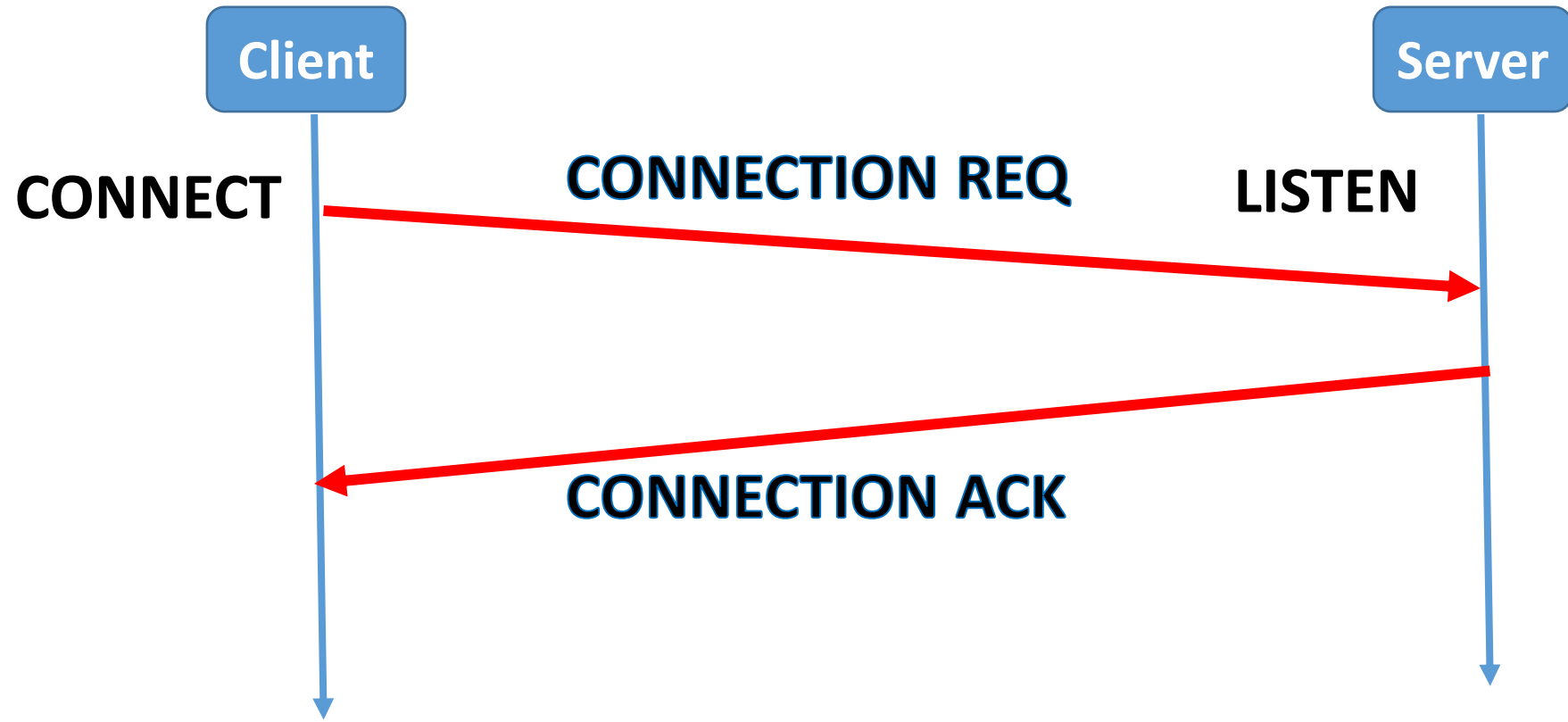


# Segment, Packet (or Datagram) and Frame



Source: Computer  
Networks (5<sup>th</sup> Edition)  
by Tanenbaum,  
Wetherell

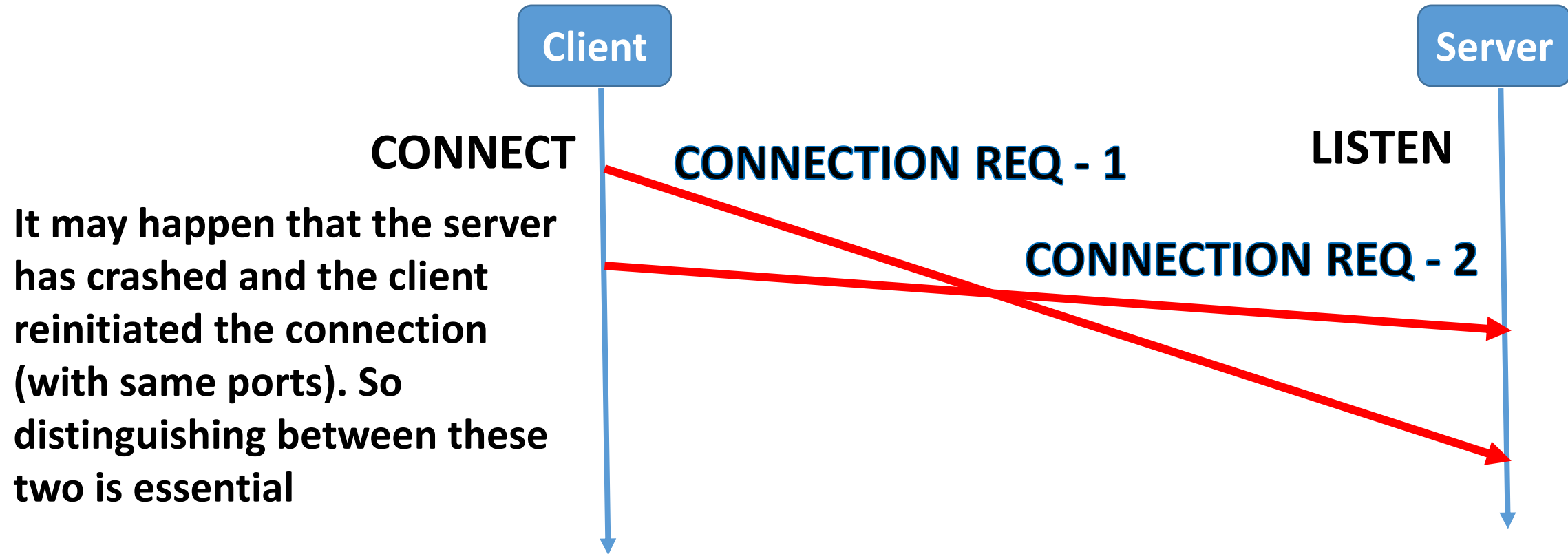
# Connection Establishment



# Connection Establishment

- Consider a scenario when the network can lose, delay, corrupt and duplicate packets (the underline network layer uses unreliable data delivery)
- Consider retransmission for ensuring reliability – every packet uses different paths to reach the destination
- Packets may be delayed and got stuck in the network congestion, after the timeout, the sender assumes that the packets have been dropped, and retransmits the packets

# Connection Establishment



# Connection Establishment

- Delayed duplicates create a huge confusion in the packet switching network. A major challenge in packet switching network is to develop **correct** or **at least acceptable** protocols for handling delayed duplicates

# Connection Establishment – Handling Delayed Duplicates

- **Solution 1: Use Throwaway Transport Address (Port Numbers)**
- **Solution 2: Give each connection a unique identifier chosen by the initiating party and put in each segment**

# Connection Establishment – Handling Delayed Duplicates

- **Solution 3: Devise a mechanism to kill off aged packets that are still hobbling about (Restrict the packet lifetime) – Makes it possible to design a feasible solution**
- Three ways to restrict packet lifetime
  - Restricted Network Design – Prevents packets from looping (bound the maximum delay including congestion)
  - Putting a hop count in each packet – initialize to a maximum value and decrement each time the packet traverses a single hop (most feasible implementation)
  - Timestamping each packet – define the lifetime of a packet in the network, need time synchronization across each router.
- **Design Challenge: We need to guarantee not only that a packet is dead, but also that all acknowledgements to it are also dead**



# Connection Establishment – Handling Delayed Duplicates

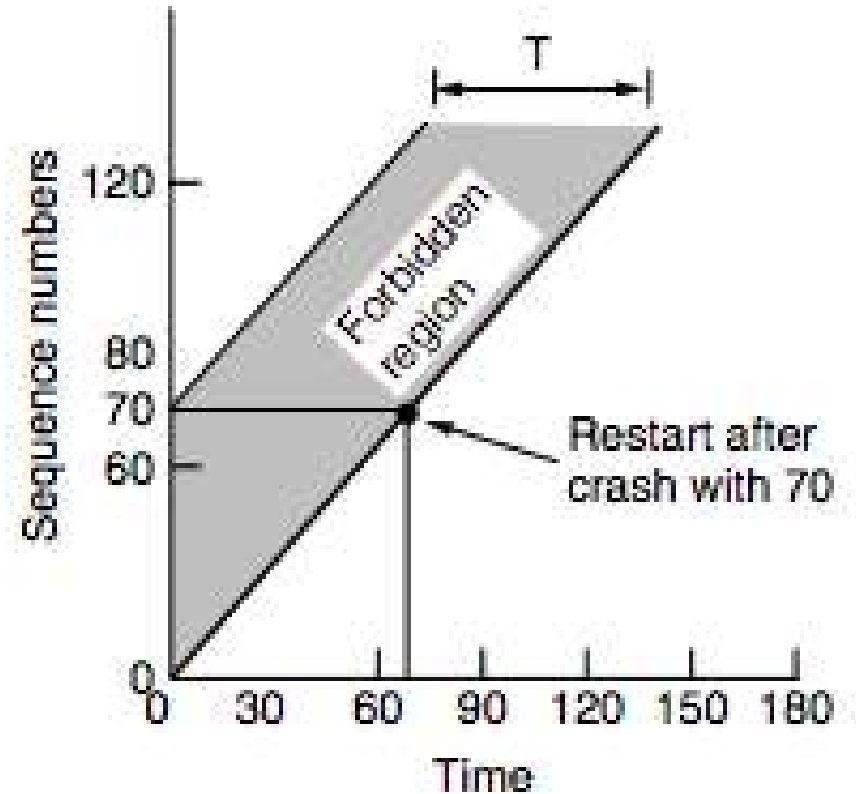
- Let us define a maximum packet lifetime  $T$  – If we wait a time  $T$  secs after a packet has been sent, we can be sure that all traces of it (packet and its acknowledgement) are now gone
- Rather than a physical clock (clock synchronization in the Internet is difficult to achieve), let us use a virtual clock – **sequence number generated based on the clock ticks**
- **Label segments with sequence numbers that will not be reused within  $T$  secs.**
- The period  $T$  and the rate of packets per second determine the size of the sequence number – **at most one packet with a given sequence number may be outstanding at any given time**

# Sequence Number Adjustment

- **Two important requirements**
  - Sequence numbers must be chosen such that a particular sequence number never refers to more than one byte (for byte sequence numbers) at any one time (**how to choose the initial sequence number**)
  - The valid range of sequence numbers must be positively synchronized between the sender and the receiver, whenever a connection is used (**three way handshaking followed by the flow control mechanism – once connection is established, only send the data with expected sequence numbers**)

# Connection Establishment – Handling Delayed Duplicates

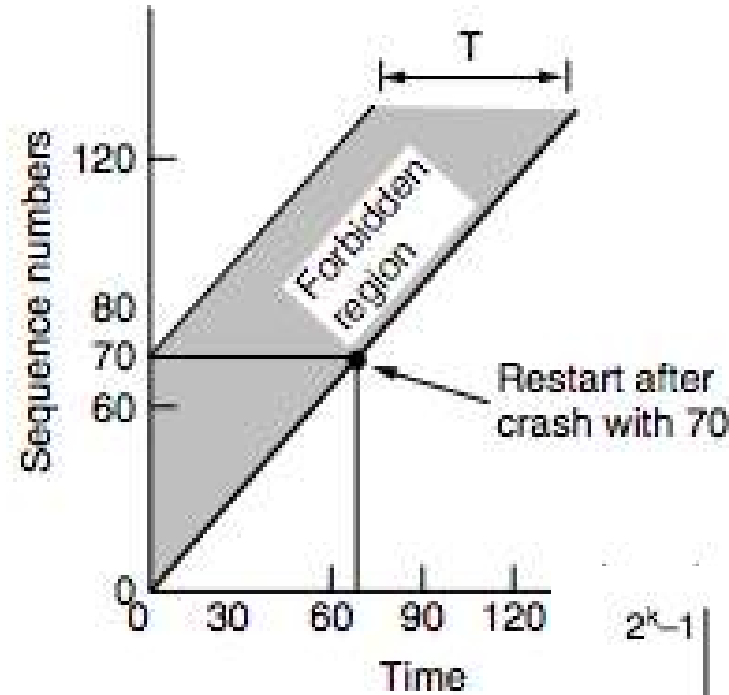
- If a receiver receives two segments having the same sequence number within a duration  $T$ , then one packet must be the duplicate. The receiver then discards the duplicate packets.
- For a crashed device, the transport entity remains idle for a duration  $T$  after recovery, to ensure that all packets from the previous connection are dead – **not a good solution**



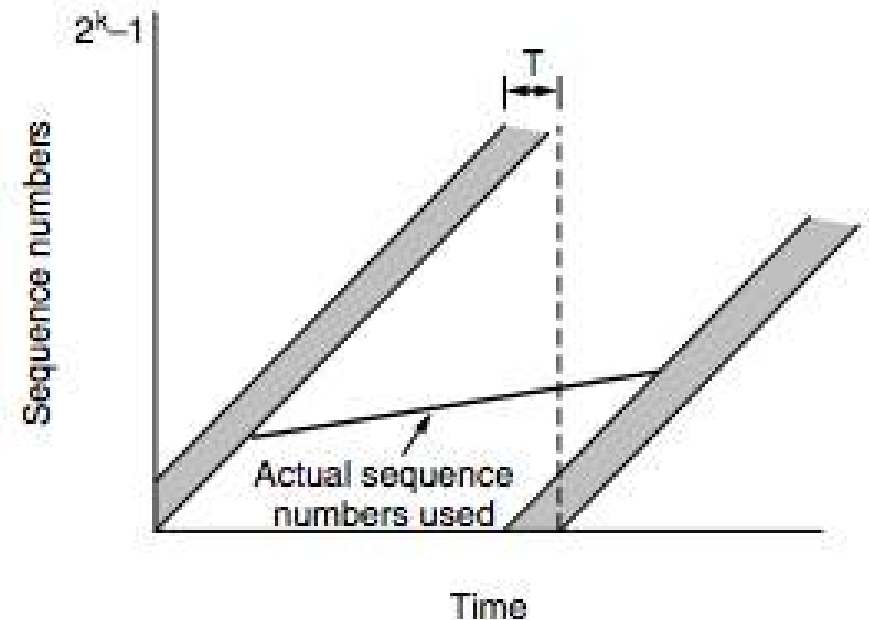
- **Adjust the initial sequence numbers properly - A host does not restart with a sequence number in the forbidden region, based on the sequence number it used before crash and the time duration  $T$ .**

# How do We Ensure that Packet Sequence Numbers are Out of the Forbidden Region

- Two possible source of problems
  - A host sends too much data too fast on a newly opened connection
- The data rate is too slow that the sequence number for a previous connection enters the forbidden region for the next connection



Source: Computer Networks (5<sup>th</sup> Edition) by Tanenbaum, Wetherell

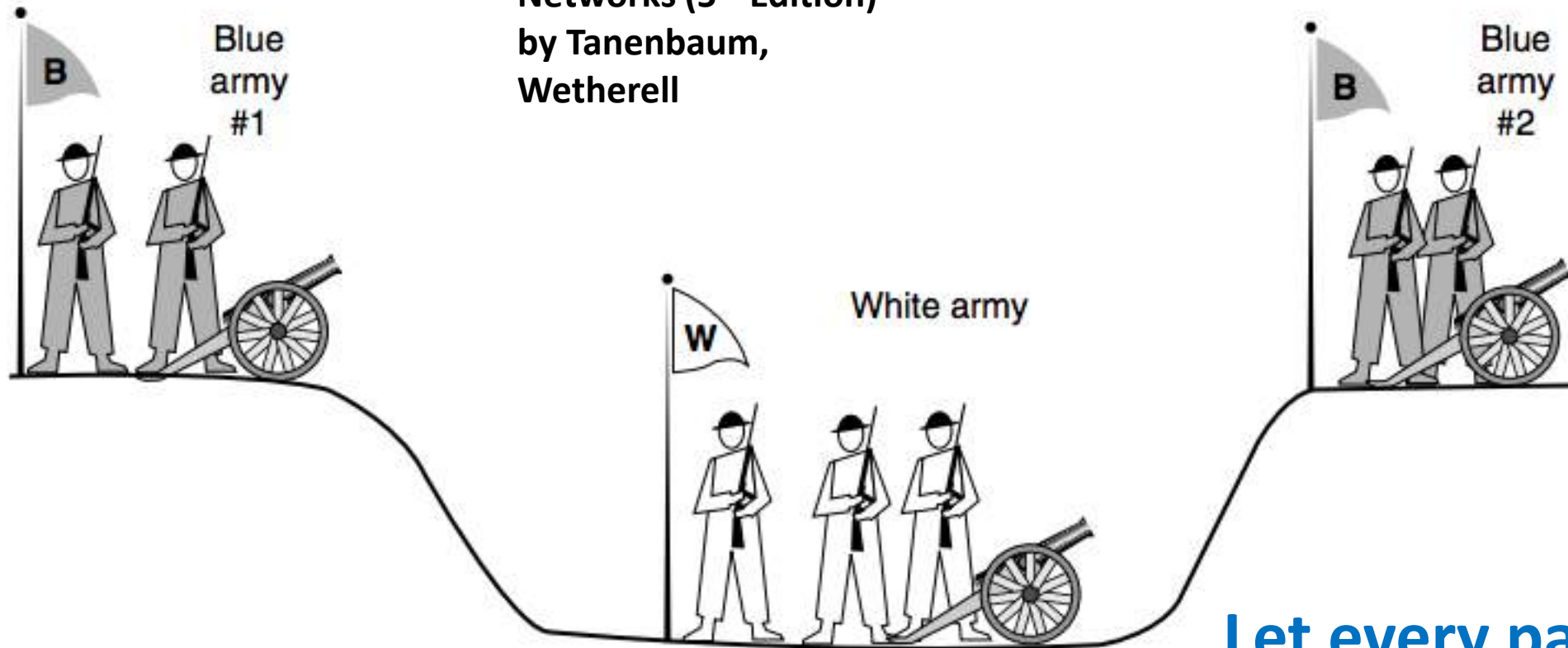


# Adjusting the Sending Rate based on Sequence Numbers

- The maximum data rate on any connection is one segment per clock tick
  - Clock ticks (inter-packet transmission duration) is adjusted based on the sequences acknowledged – **ensure that no two packets are there in the network with same sequence number**
  - **We call this mechanism as self-clocking (used in TCP)**
  - Ensures that the sequence numbers do not warp around too quickly (RFC 1323)
- **We do not remember sequence number at the receiver: Use a three way handshake** to ensure that the connection request is not a repetition of an old connection request
  - The individual peers validate their own sequence number by looking at the acknowledgement (ACK)
  - **Positive synchronization among the sender and the receiver**

# The Two Army Problem

Source: Computer  
Networks (5<sup>th</sup> Edition)  
by Tanenbaum,  
Wetherell



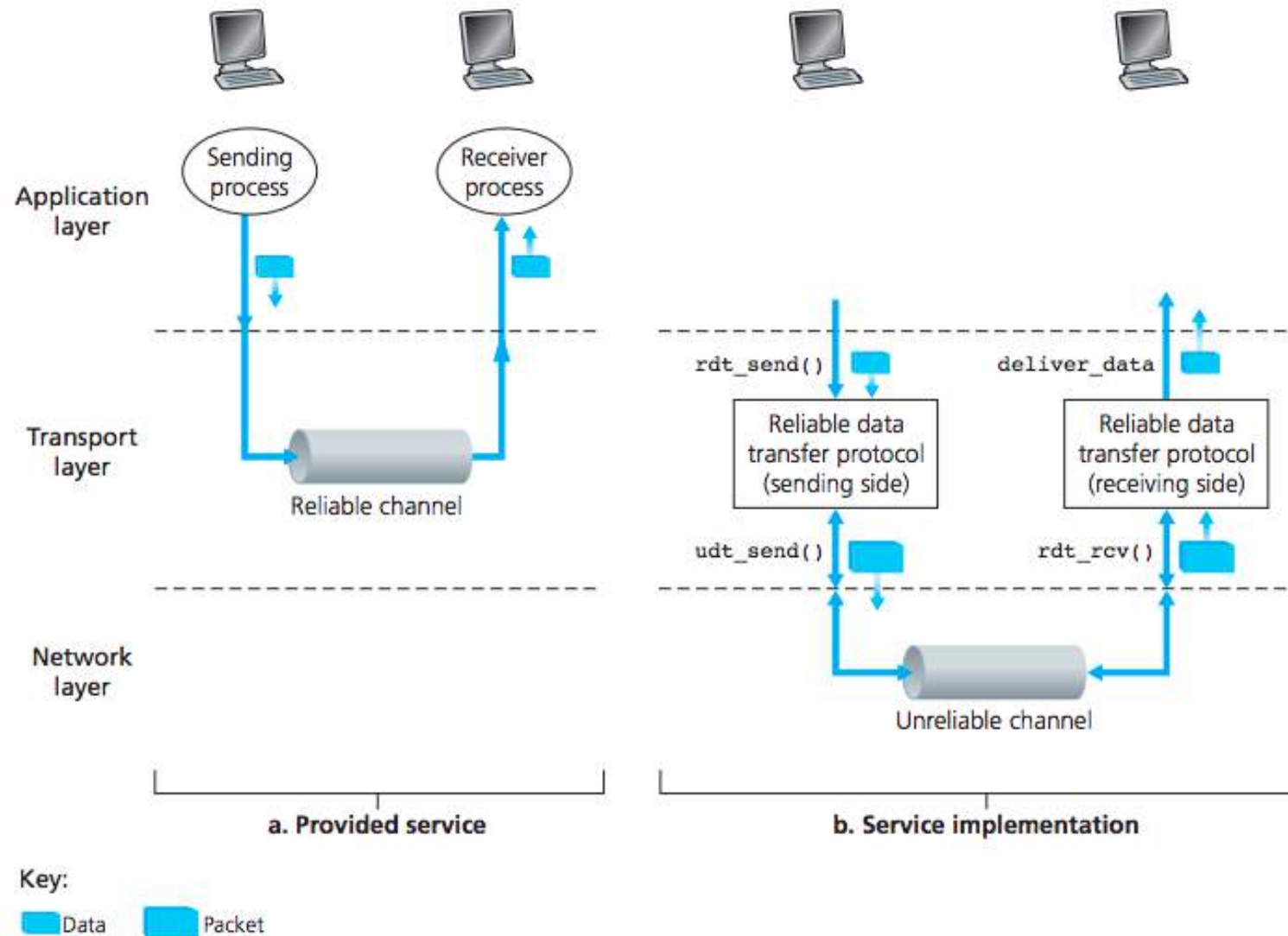
Let every party take  
independent  
decisions

# The Two Army Problem

- The blue armies want to synchronize their attacks. However, their only communication medium is to send messengers on foot down into the valley, where they might be captured and the message lost (i.e., they have to use an unreliable communication channel).
- Suppose that the commander of blue army #1 sends a message reading: "I propose we attack at dawn on March 29. How about it?" Now suppose that the message arrives, the commander of blue army #2 agrees, and his reply gets safely back to blue army #1. Will the attack happen? Probably not, because commander #2 does not know if his reply got through. If it did not, blue army #1 will not attack, so it would be foolish for him to charge into battle.
- Now let us improve the protocol by making it a three-way handshake. The initiator of the original proposal must acknowledge the response. Assuming no messages are lost, blue army #2 will get the acknowledgement, but the commander of blue army #1 will now hesitate. After all, he does not know if his acknowledgement got through, and if it did not, he knows that blue army #2 will not attack. We could now make a four-way handshake protocol, but that does not help either.
- **No protocol exists to solve this**



# Ensure Reliability at the Transport Layer



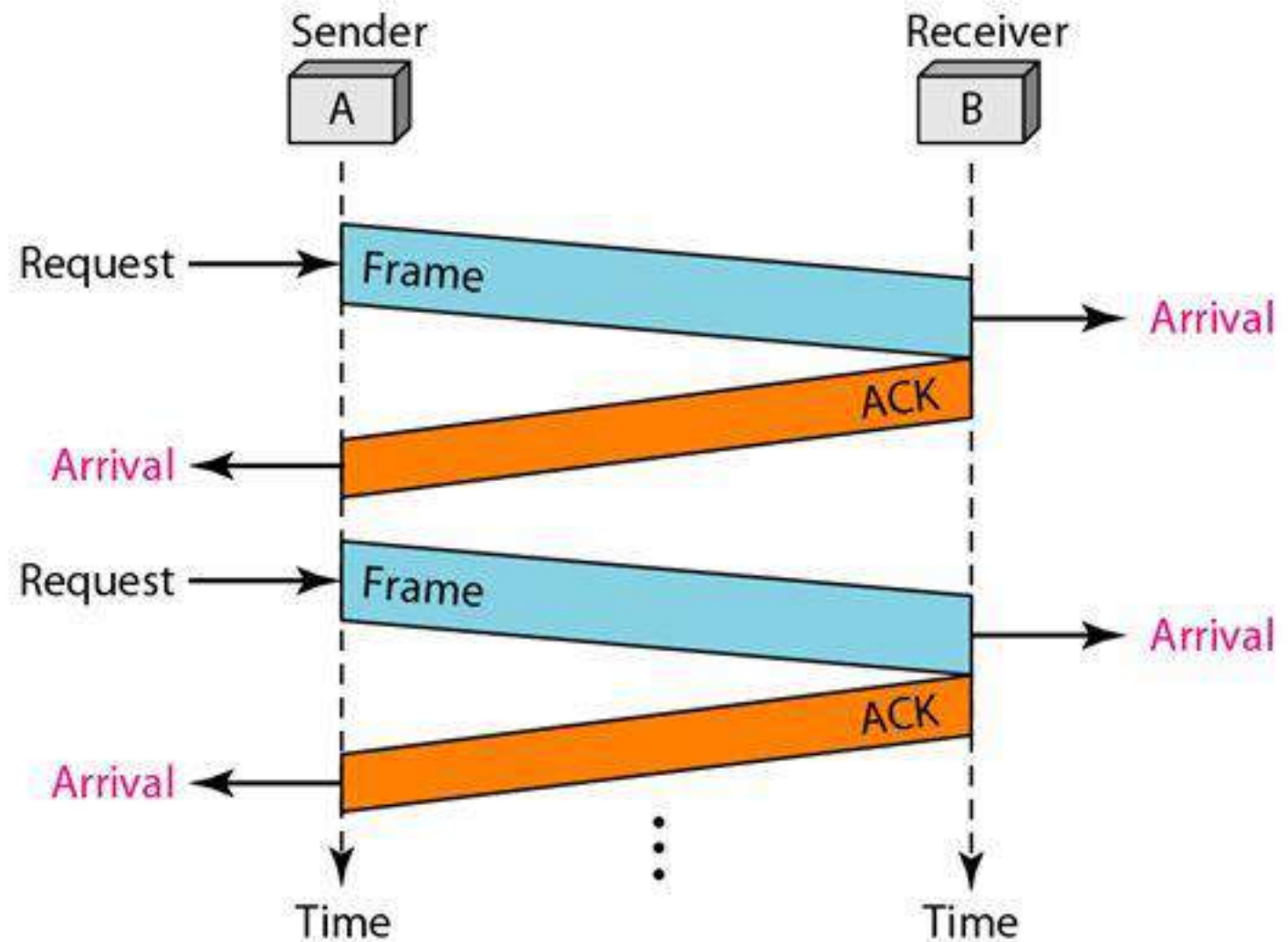
Source: Computer Networks, Kurose, Ross

# Flow control

- In some ways the flow control problem in the transport layer is the same as in the data link layer, but in other ways it is different.
- The basic similarity is that in both layers a sliding window or other scheme is needed on each connection to keep a fast transmitter from overrunning a slow receiver.
- The main difference is that a router usually has relatively few lines, whereas a host may have numerous connections. This difference makes it impractical to implement the data link buffering strategy in the transport layer.

# Flow Control Algorithms

- **Stop and Wait Flow Control (Error Free Channel):**

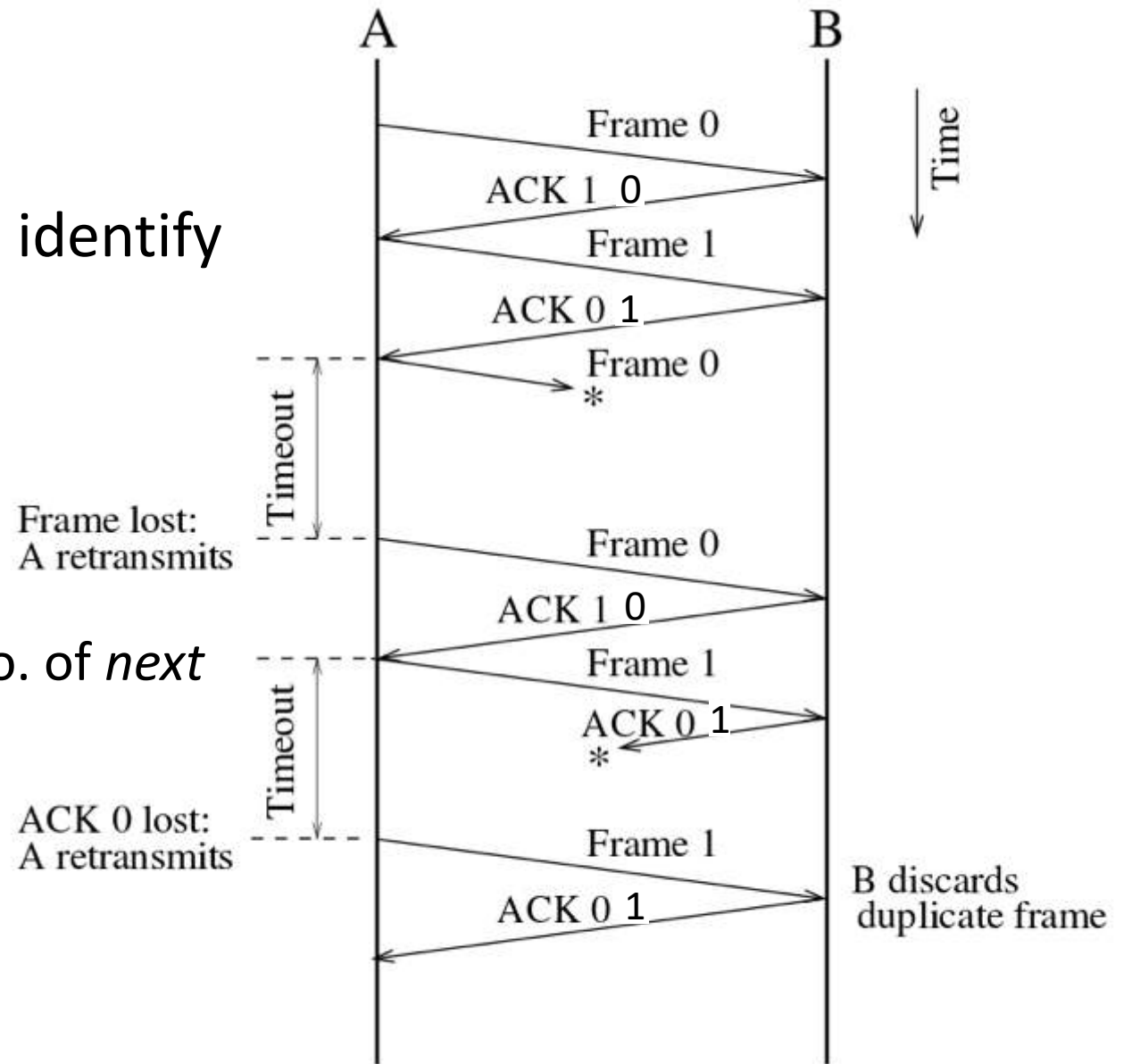


# Flow Control Algorithms

- **Stop and Wait (Noisy Channel):**
- Use sequence numbers to individually identify each frame and the corresponding acknowledgement

Note: acknowledgement contains sequence no. of *next expected frame* in TCP actually

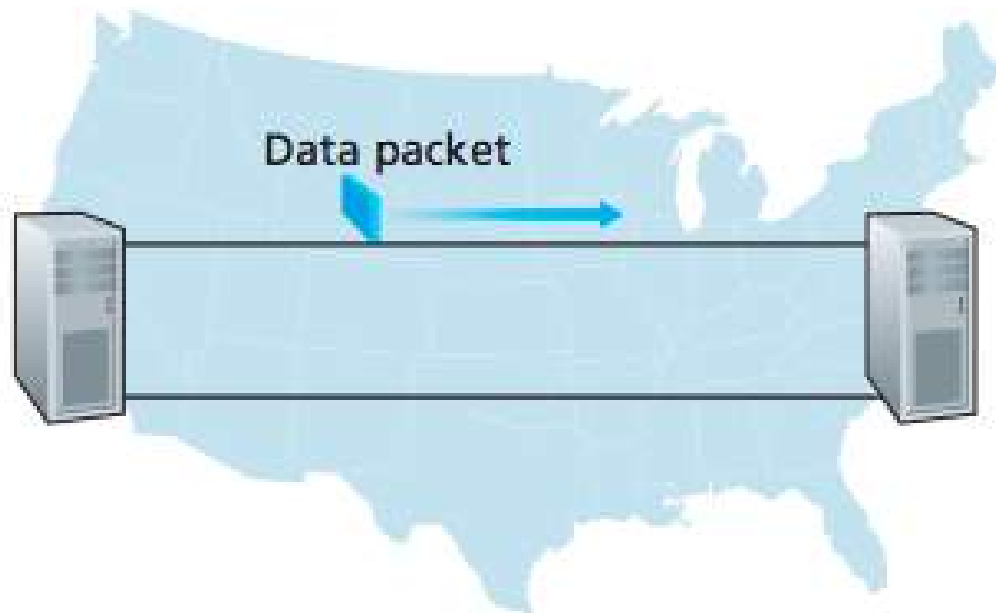
- **Repeat Request (ARQ)**



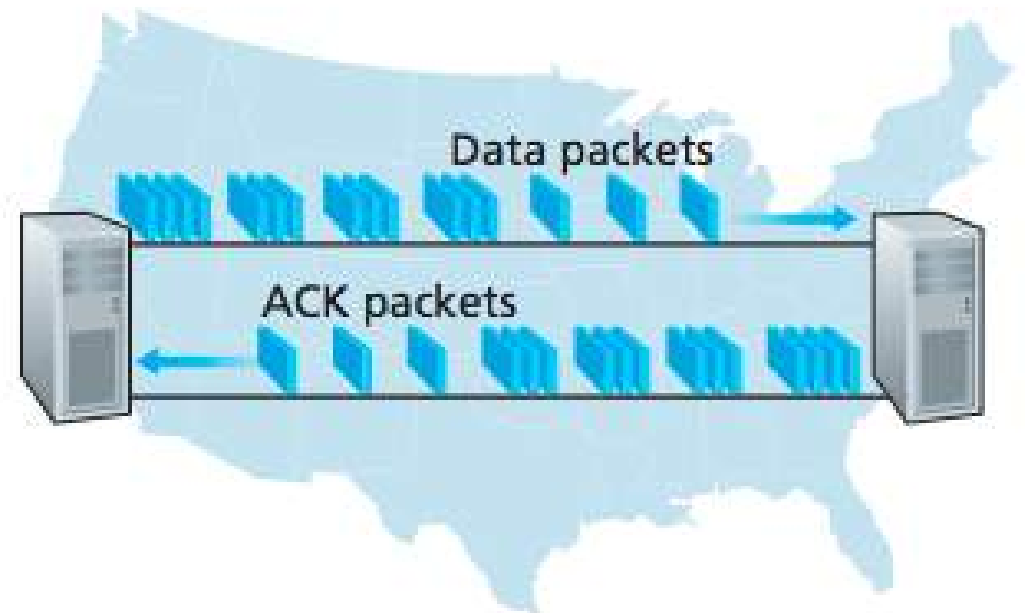
# Problem with Stop and Wait

- Every packet needs to wait for the acknowledgement of the previous packet.
- For bidirectional connections – use two instances of the stop and wait protocol at both directions – further waste of resources
- A possible solution: Piggyback (Piggybacking is the technique of delaying outgoing acknowledgment and attaching it to the next data packet) data and acknowledgement from both the directions.
- Reduce resource waste based on **sliding window protocols (a pipelined protocol)**

# Stop and Wait versus Sliding Window (Pipelined)



**a. A stop-and-wait protocol in operation**



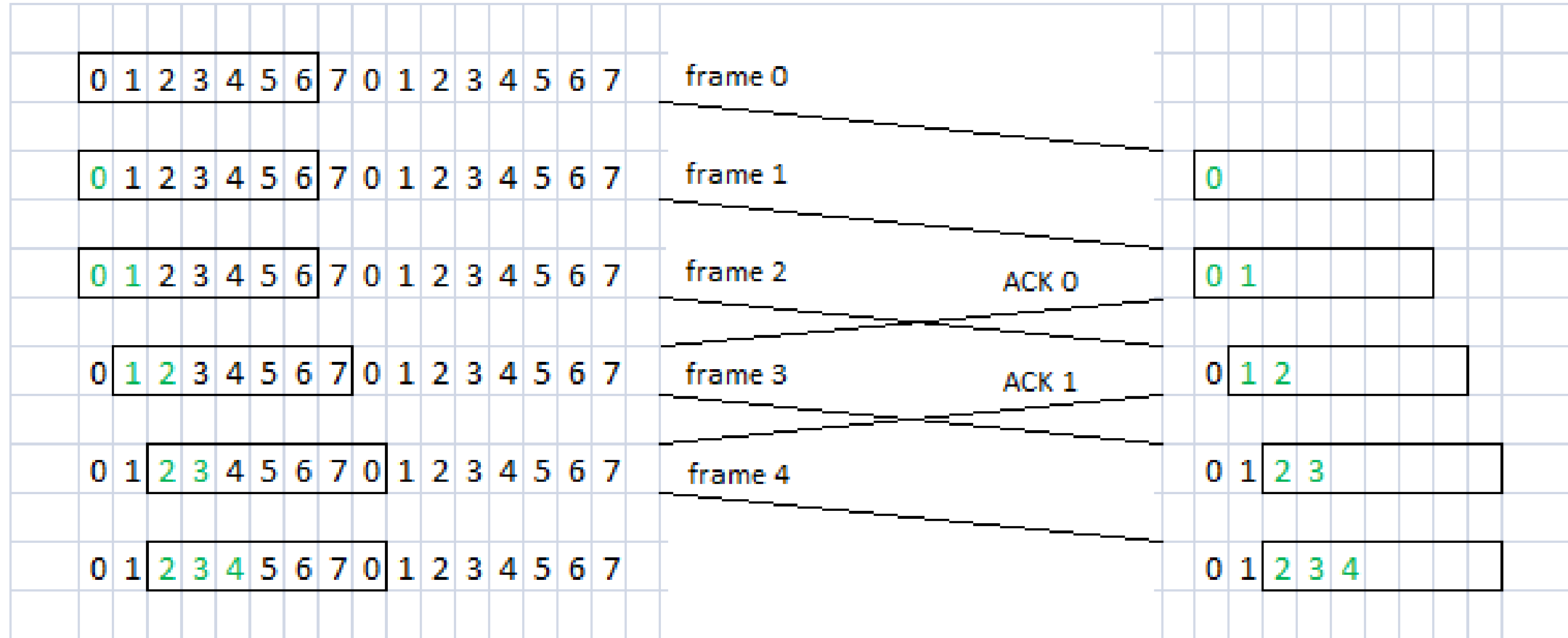
**b. A pipelined protocol in operation**

# Sliding Window Protocols

- Each outbound segment contains a sequence number – from 0 to some maximum ( $2^n-1$  for a  $n$  bit sequence number)
- The sender maintains a set of sequence numbers corresponding to frames it is permitted to send (**sending window**)
- The receiver maintains a set of frames it is permitted to accept (**receiving window**)



# Sliding Window Protocols – Sending Window and Receiving Window

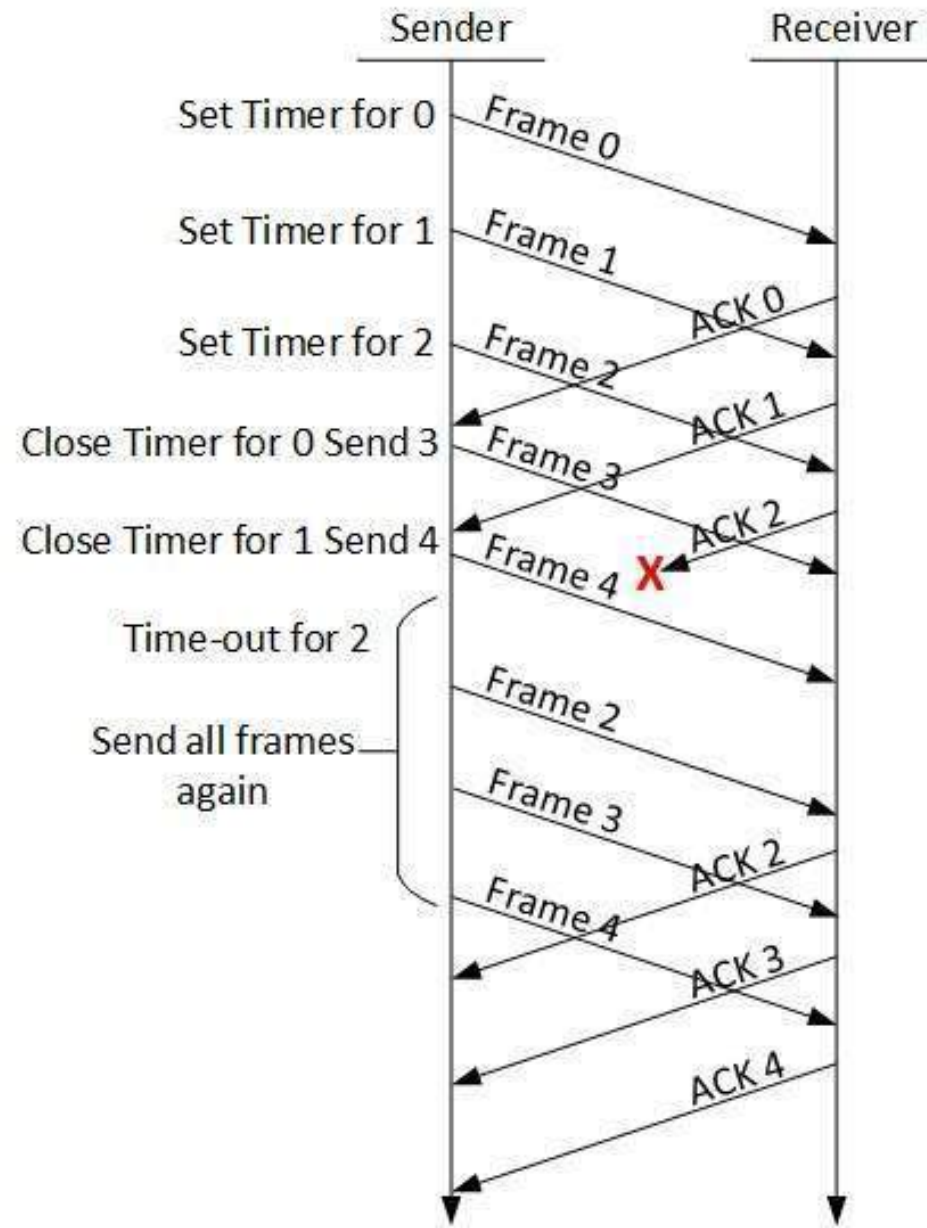


Sliding window Protocol

# Sliding Window Protocols in Noisy Channels

- **A timeout occurs if a segment (or the acknowledgment) gets lost**
- **Go Back N ARQ:** If segment N is lost, all the segments from segment 0 (start of the sliding window) to segment N are retransmitted
- **Selective Repeat (SR) ARQ:** Only the lost packets are selectively retransmitted
  - **Negative Acknowledgement (NAK) or Selective Acknowledgements (SACK):** Informs the sender about which packets need to be retransmitted (not received by the receiver)

# Go Back N ARQ



Source

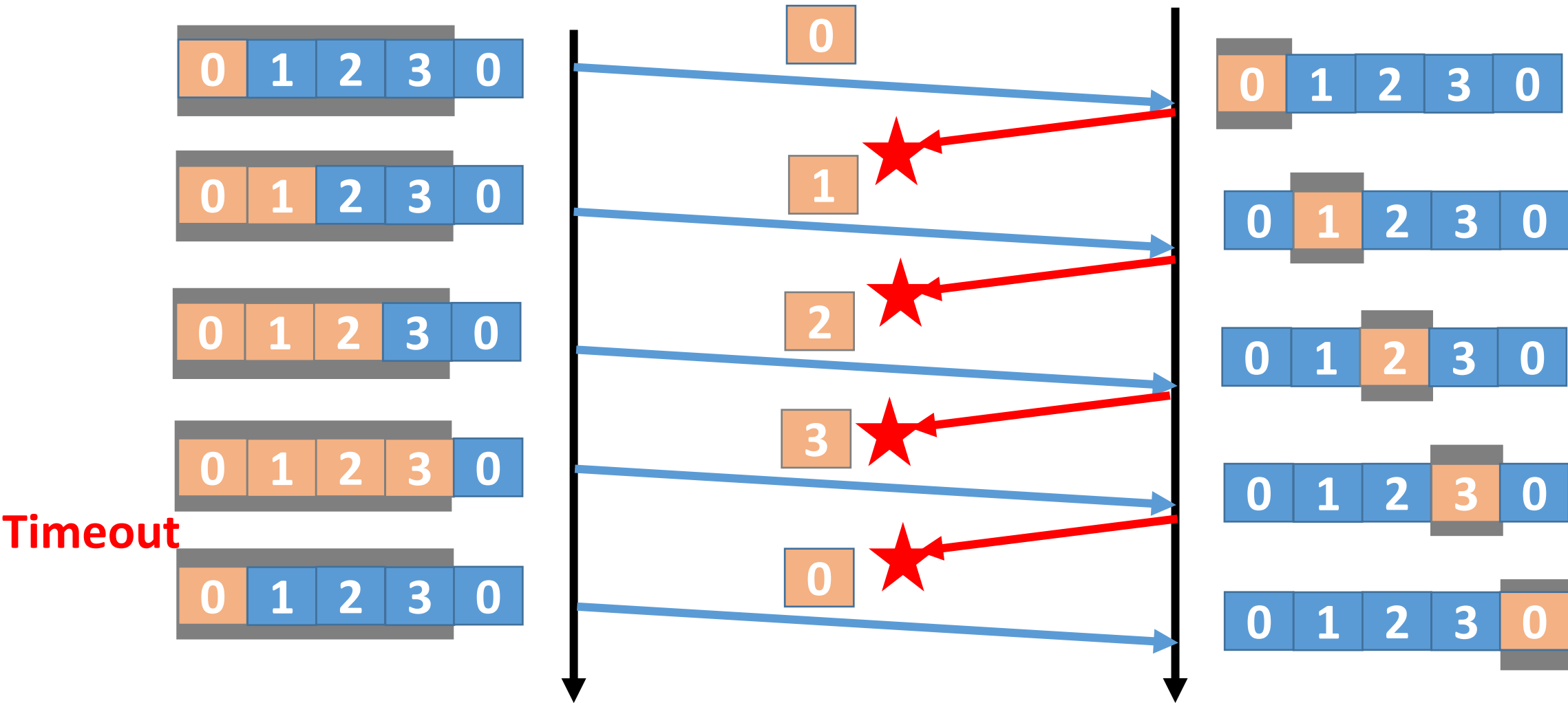
[https://www.tutorialspoint.com/data\\_communication\\_computer\\_network/data\\_link\\_control\\_and\\_protocols.htm](https://www.tutorialspoint.com/data_communication_computer_network/data_link_control_and_protocols.htm)

## Go Back N ARQ – A Bound on Window Size

- **Outstanding Frames** – Frames that have been transmitted, but not yet acknowledged
- **Maximum Sequence Number (MAX\_SEQ):** MAX\_SEQ+1 distinct sequence numbers are there
  - 0,1,...,MAX\_SEQ
- **Maximum Number of Outstanding Frames (=Window Size):** MAX\_SEQ
- **Example:** Sequence Numbers (0,1,2,...,7) – 3 bit sequence numbers, number of outstanding frames = 7 (**Not 8**)

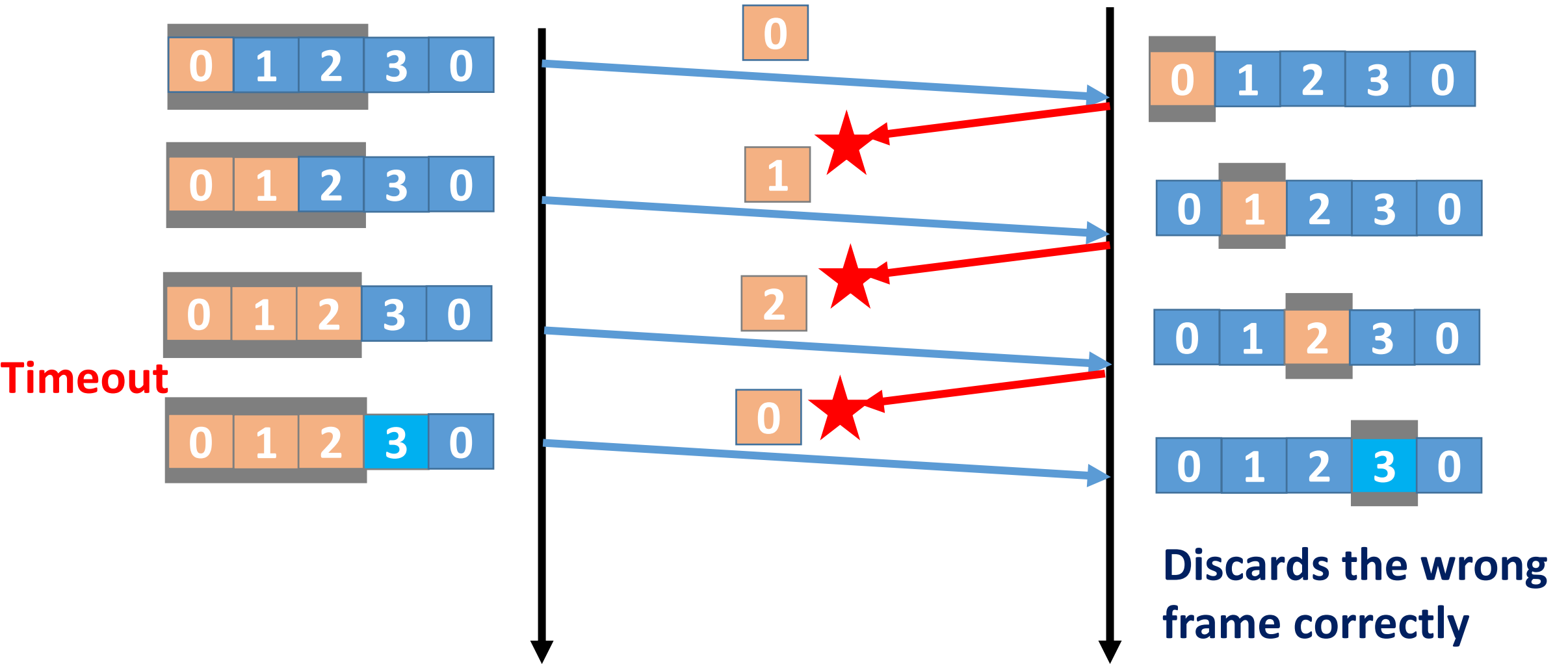
# Go Back N ARQ – A Bound on Window Size

- Let  $\text{MAX\_SEQ} = 3$ , Window Size = 4

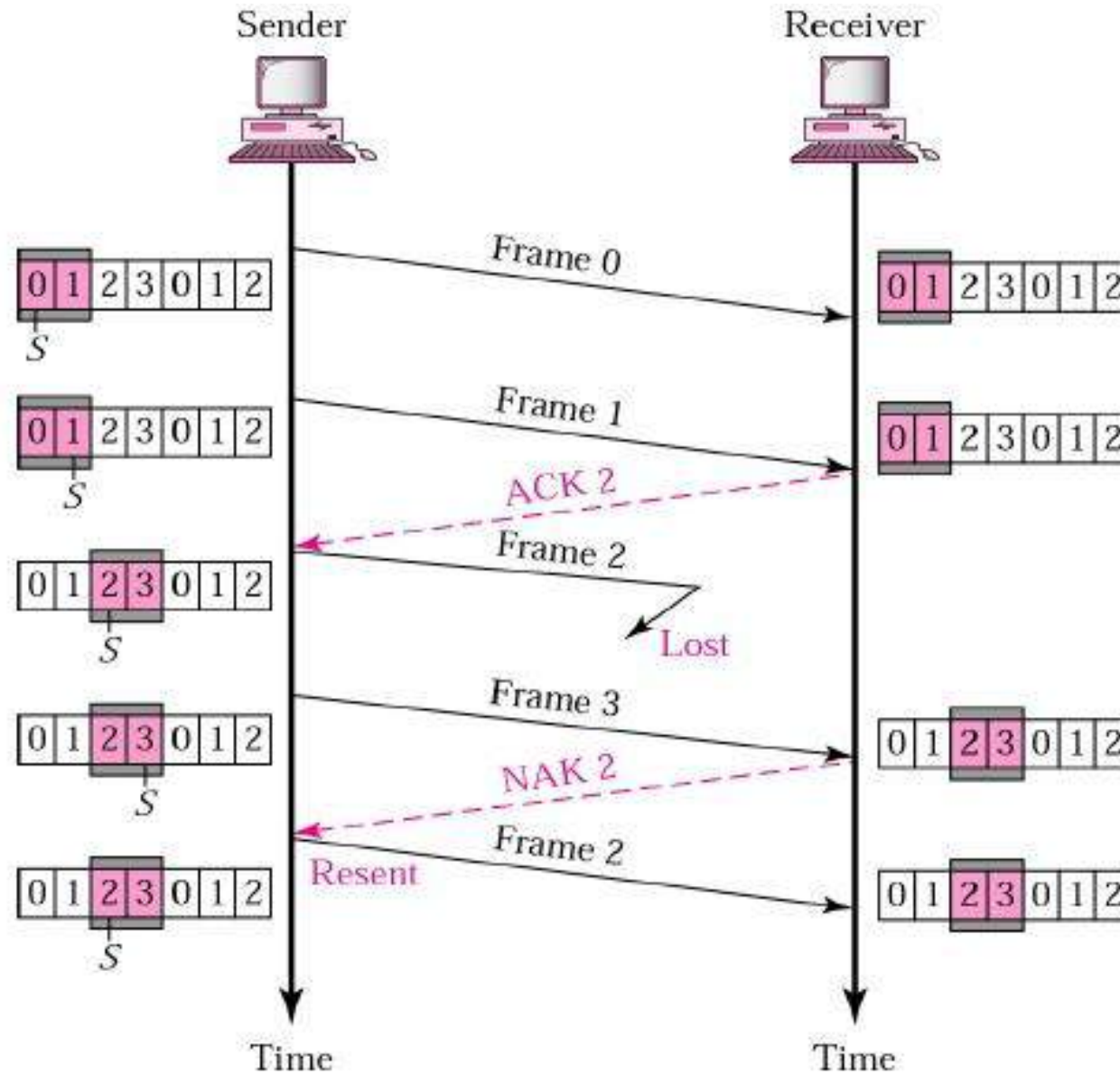


# Go Back N ARQ – A Bound on Window Size

- Let  $\text{MAX\_SEQ} = 3$ , Window Size = 3



# Selective Repeat ARQ



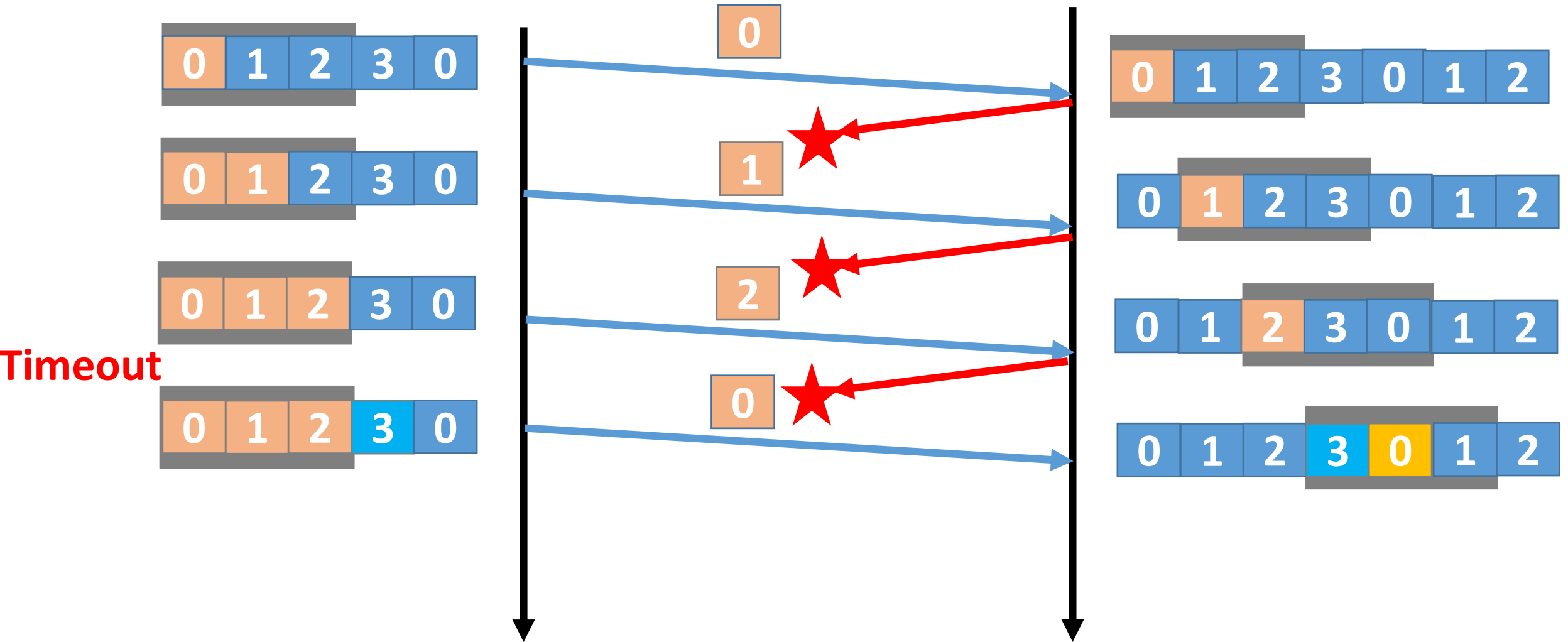


# Selective Repeat – A Bound on Window Size

- **Maximum Sequence Number (MAX\_SEQ):** MAX\_SEQ+1 distinct sequence numbers are there
  - 0,1,...,MAX\_SEQ
- **Then, Max. No. of Outstanding Frames ( = Window Size ):**  $(MAX\_SEQ+1)/2$
- **Example:** Sequence Numbers (0,1,2,...,7) – 3 bit sequence numbers, max. number of outstanding frames (= window size) = 4

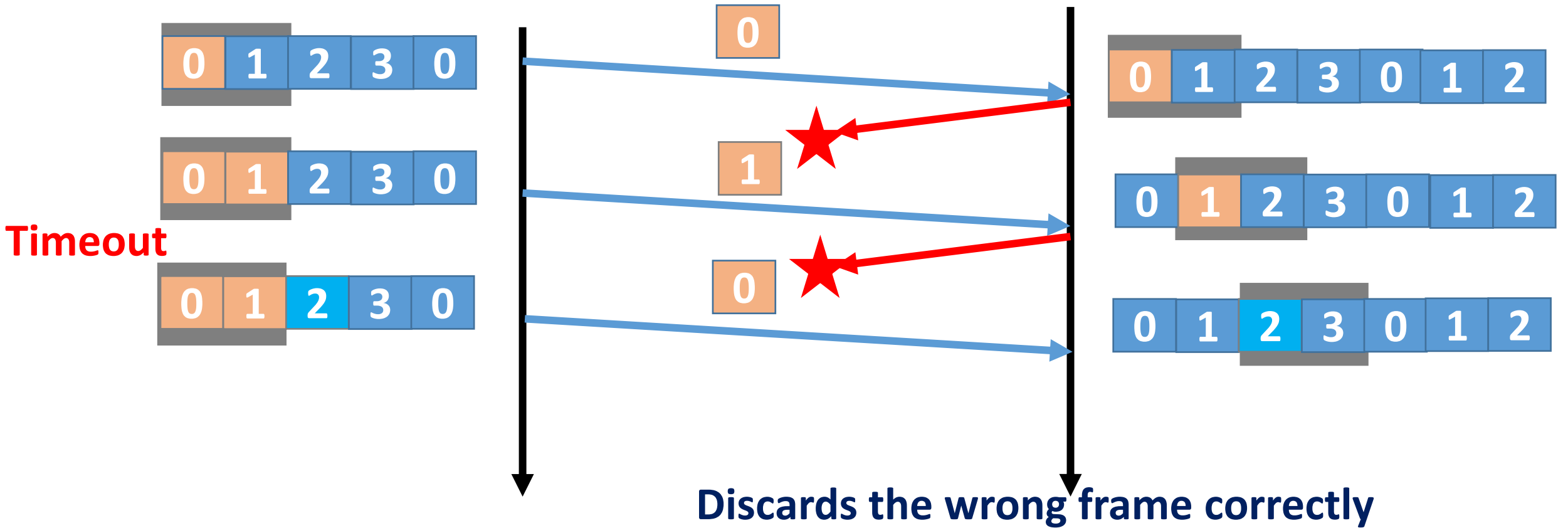
# Selective Repeat – A Bound on Window Size

- Let  $\text{MAX\_SEQ} = 3$ , Window Size = 3 [i.e. above safe limit  $(\text{MAX\_SEQ}+1)/2$ ]



# Selective Repeat – A Bound on Window Size

- Let  $\text{MAX\_SEQ} = 3$ , Window Size = 2



Note: duplicate frames might be discarded by receiver, but an acknowledgement with the frame no. is sent back to the sender, so that the sender window continues to slide forward. Otherwise, sender has no way of knowing that a packet has actually been received earlier (albeit out-of-order).

# Bandwidth Delay Product

- **Bandwidth Delay Product (BDP) = Link Bandwidth x Link Delay** – an important metric for flow control
- Consider Bandwidth = 50 Kbps, one way transit time (delay) = 250 msec
  - BDP 12.5 Kbit
  - Assume 1000 bit segment size; BDP = 12.5 segments
- Consider the event of a segment transmission and the corresponding ACK reception – this takes a round trip time (RTT) – twice the one way latency.
- Maximum number of segments that can be outstanding during this duration =  $12.5 \times 2 = 25$  segments

# Bandwidth Delay Product – Implication on Window Size

- Maximum number of segments that can be outstanding within this duration =  $25 + 1$  (as the ACK is sent only when the first segment is received) = 26
  - This gives the maximum link utilization – **the link will always be busy in transmitting data segments**
- Let **BD** denotes the number of frames equivalent to the BDP, **w** is the maximum window size
- So,  **$w = 2BD + 1$**  gives the maximum link utilization – **this is an important concept to decide the window size for a window based flow control mechanism**

# Implication of BDP on Protocol Design Choice

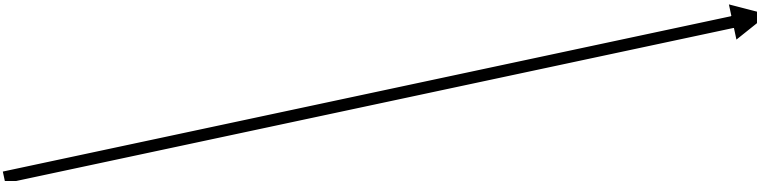
- Consider the link bandwidth = 1Mbps, Delay = 1ms
- Consider a network, where segment size is 1 KB (1024 bytes)
- Which protocol is better for flow control?
  - (a) stop and wait,
  - (b) Go back N,
  - (c) Selective Repeat
- **BDP = 1 Mbps x 1ms = 1 Kb (1024 bits)**
- **The segment size is eight times larger than the BDP -> the link can not hold an entire segment completely**
- **Sliding window protocols do not improve performance**
- **Stop and Wait is better – less complexity**

# Congestion Control in the Network


- Flows enter and exit network dynamically – so applying an algorithm for congestion control is difficult
- **Congestion avoidance:** Regulate the sending rate based on what the network can support

**Sending Rate = minimum (network rate, Receiver rate)**

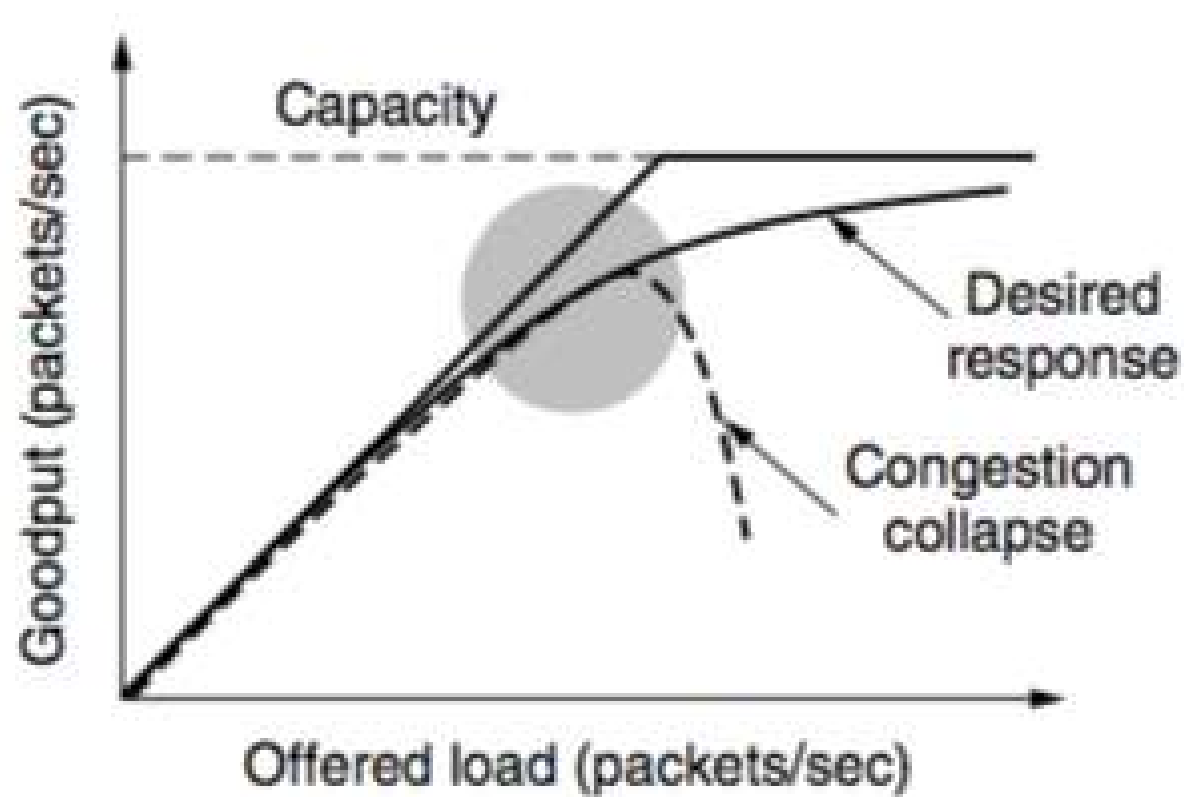
**Gradually increase the network rate  
and observe the effect on flow rates  
(packet loss)**



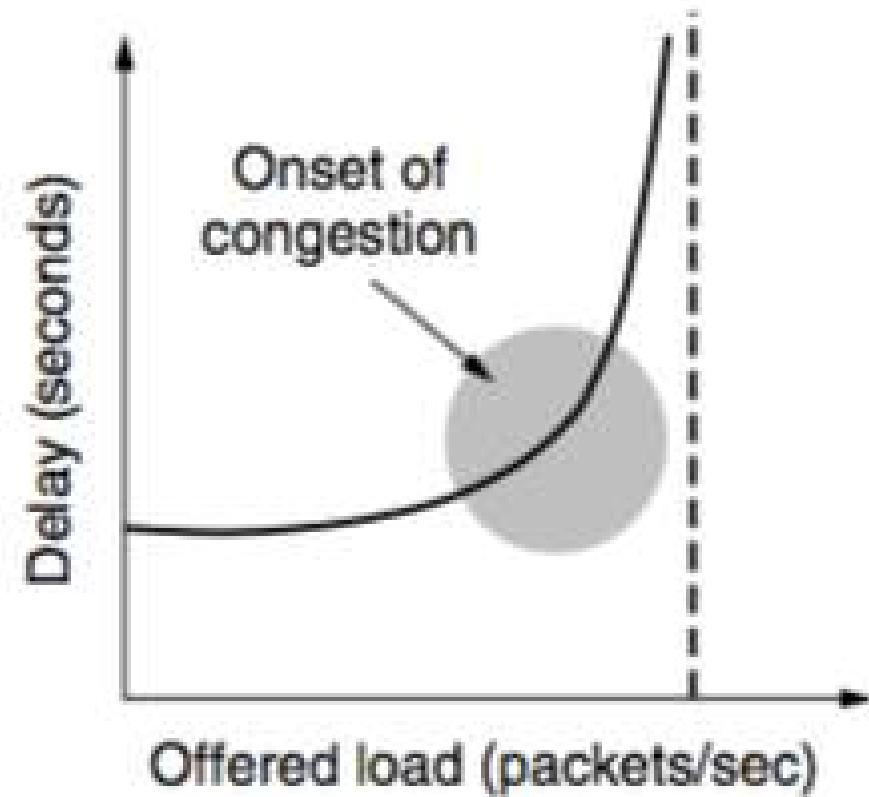
**Comes from flow control –  
receiver advertised  
window size for a sliding  
window flow control**



# Network Congestion – Impact over Goodput and Delay



(a)



(b)

Source: Computer  
Networks (5<sup>th</sup> Edition)  
by Tanenbaum,  
Wetherell



# Congestion Control and Fairness

- Ensure that the rate of all the flows in the network is controlled in a **fair way**
- A bad congestion control algorithm may affect fairness - Some flows can get starved
- Hard fairness in a decentralized network is difficult to implement
- **Max-Min Fairness:** An allocation is max-min fair if the bandwidth given to one flow cannot be increased without decreasing the bandwidth given to another flow with an allocation.