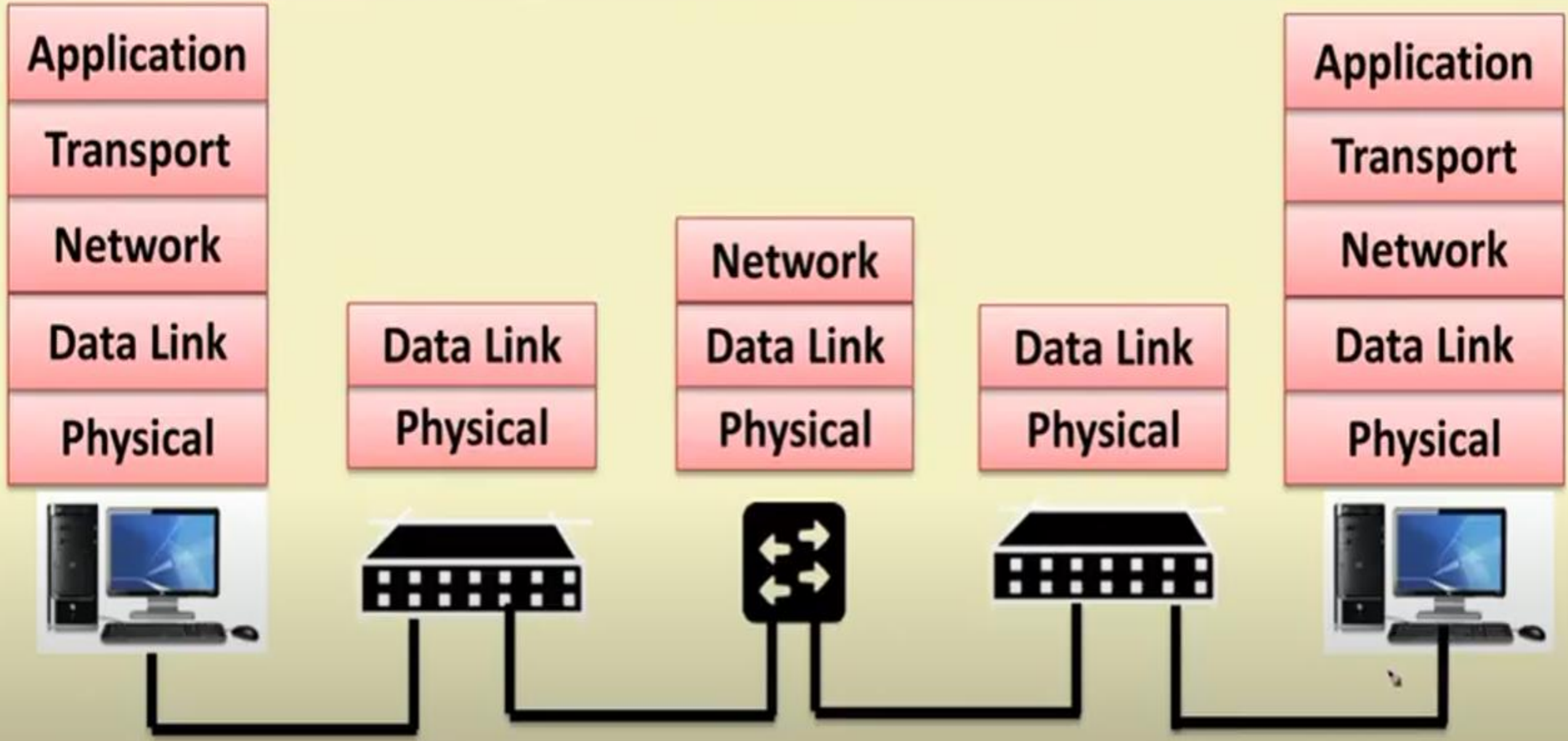
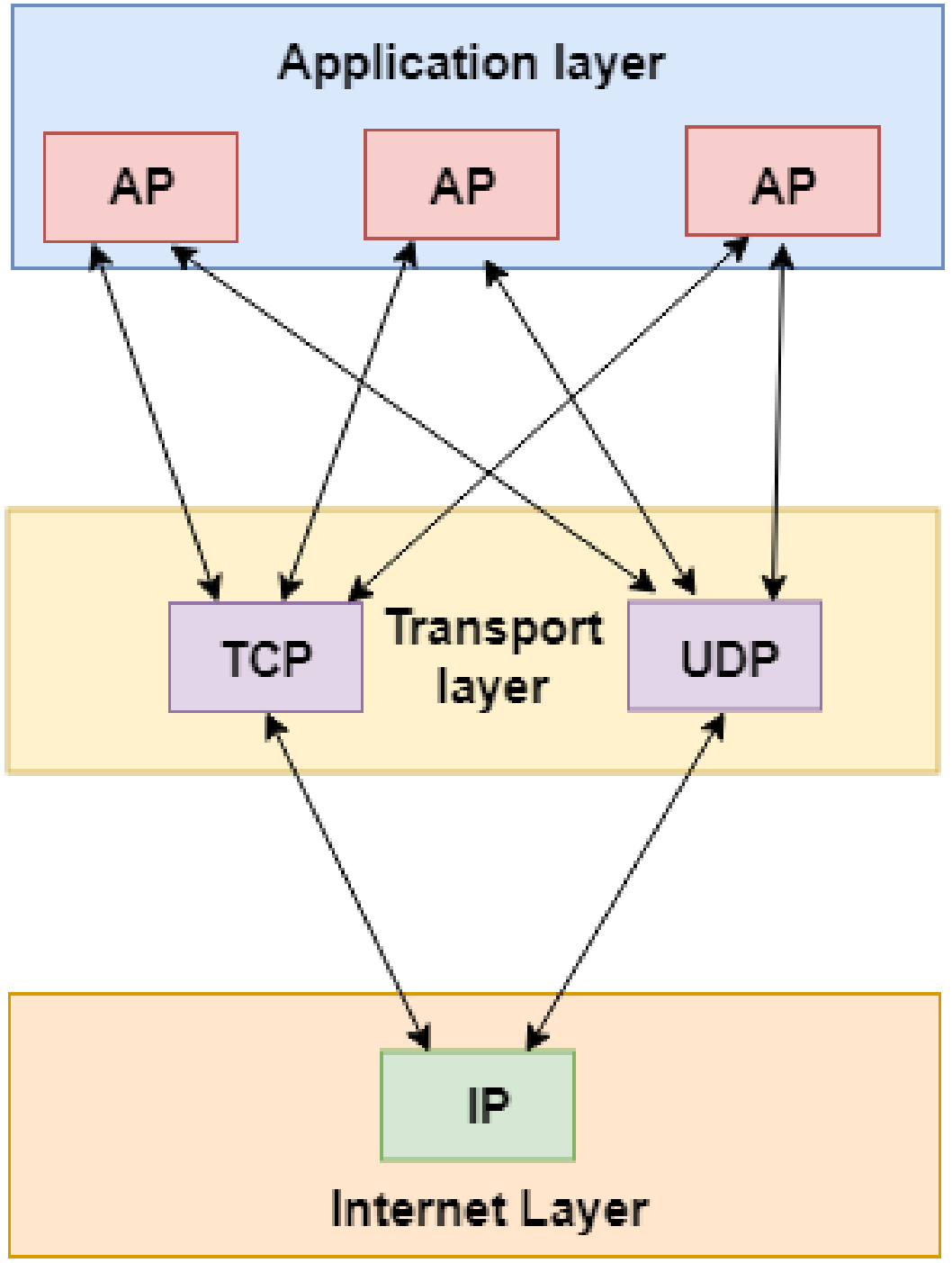
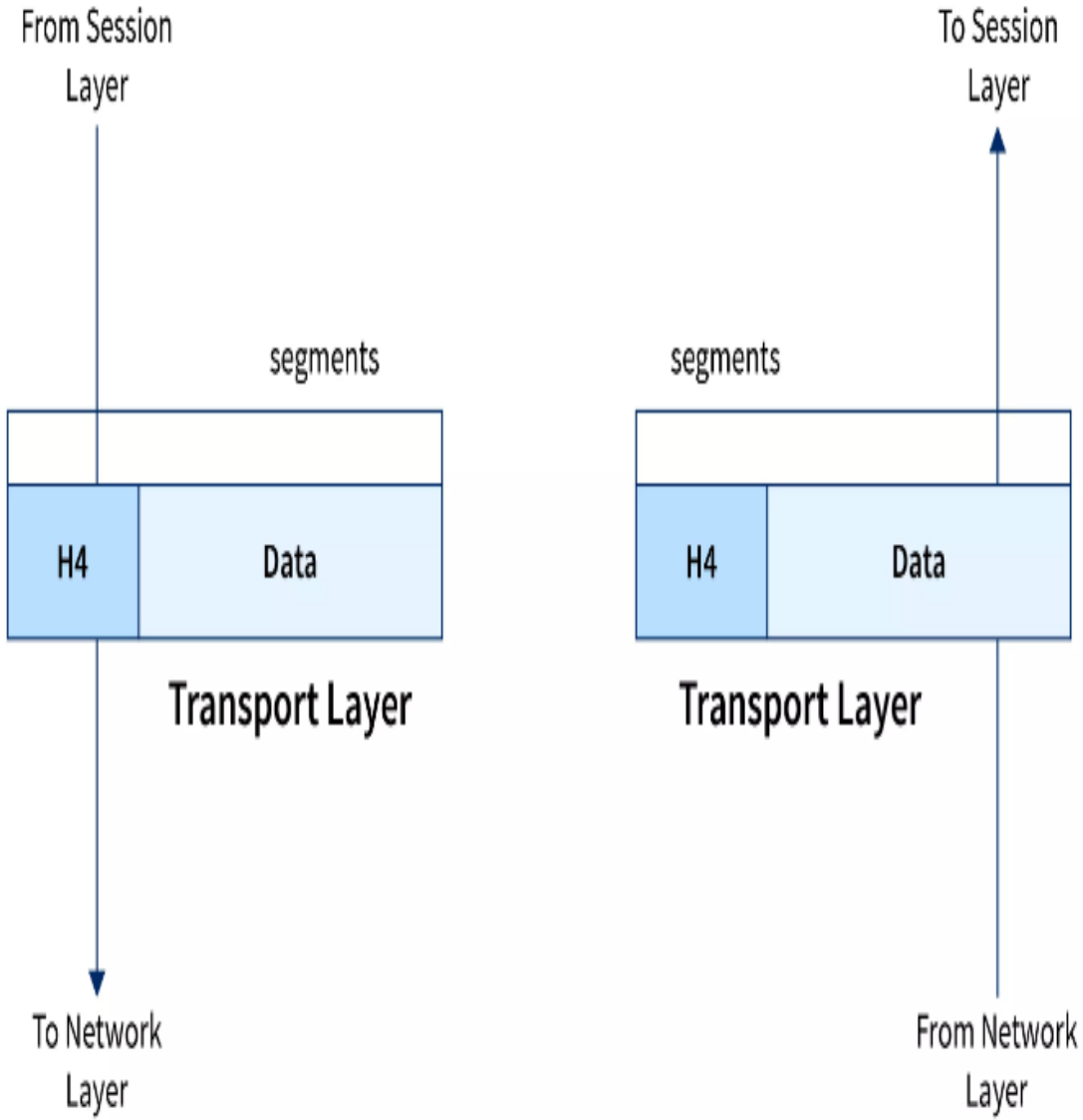
Transport Layer - I (Services)



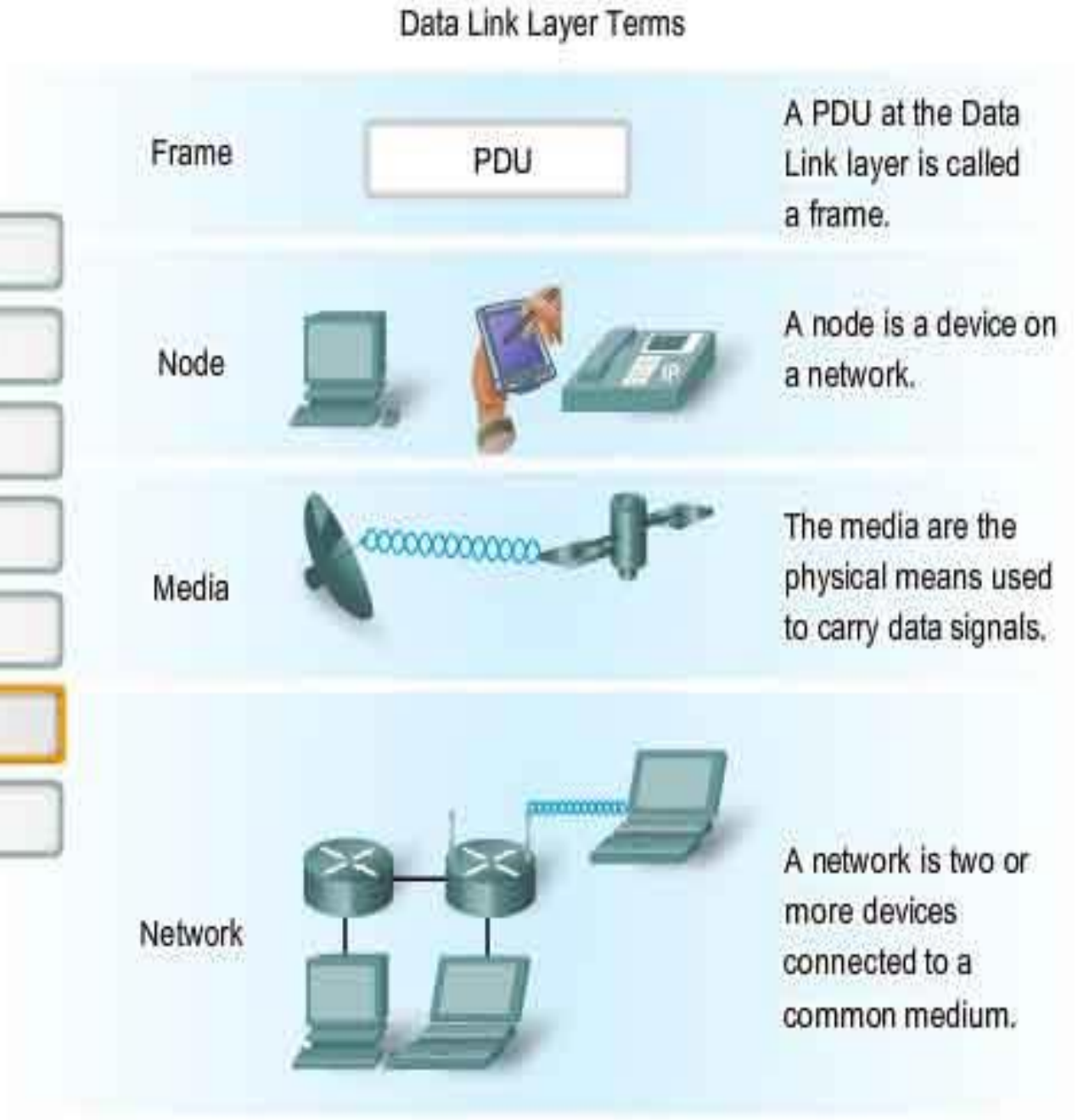
The transport layer provides a logical communication between application processes running on different hosts. Although the application processes on different hosts are not physically connected, application processes use the logical communication provided by the transport layer to send the messages to each other.



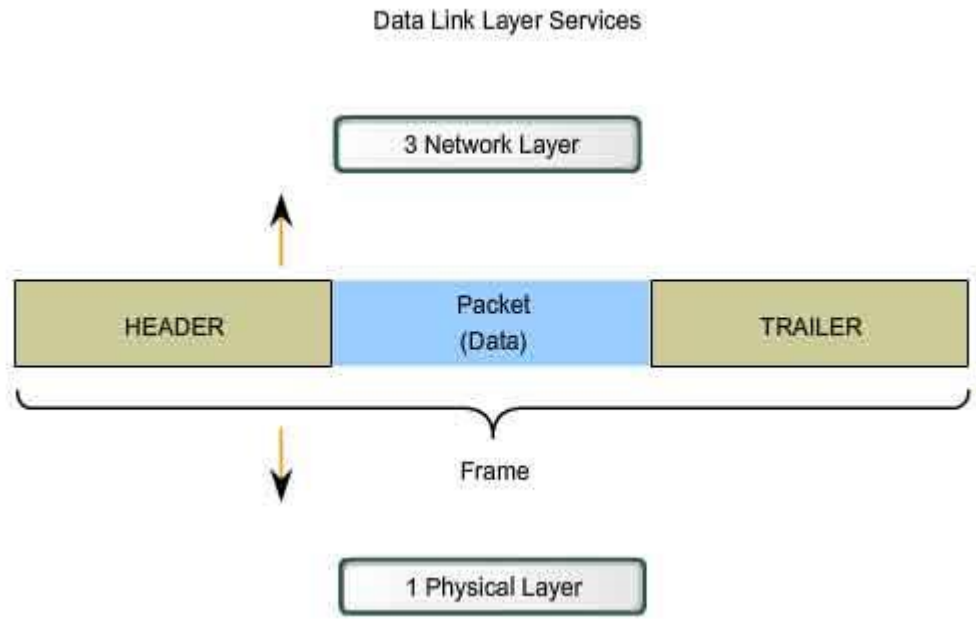
The transport layer is the fourth layer of the OSI model which is responsible for the process to process delivery of data. The network layer divides the data received from the transport layer in the form of packets. The network layer provides two ways of communication namely - connection-oriented and connection-less.



Network layer: Handles the routing and sending of data between different networks. The most important protocols at this layer are IP and ICMP. 2. Data link layer: Handles communications between devices on the same network.

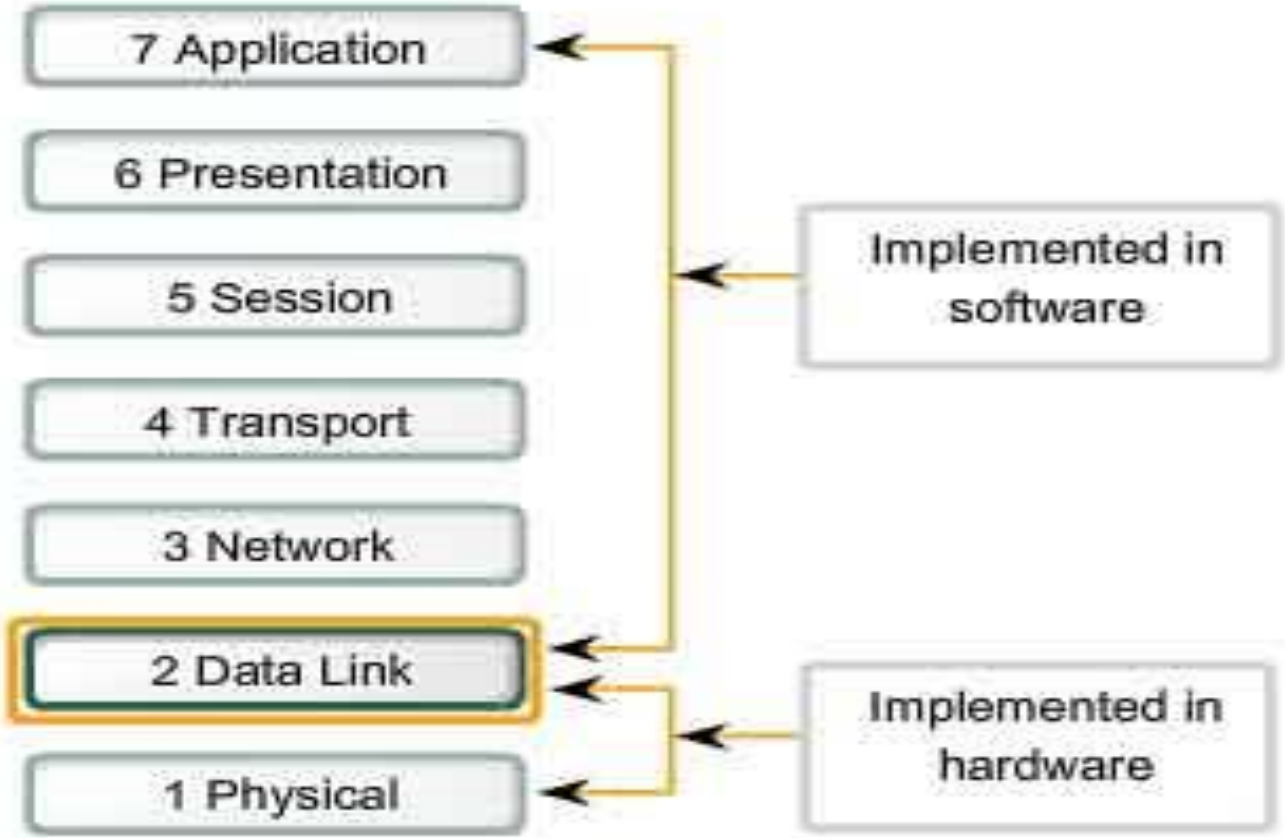


The Data Link layer is responsible for controlling the transfer of frames across the media.



Connecting Upper Layer Services to the Media

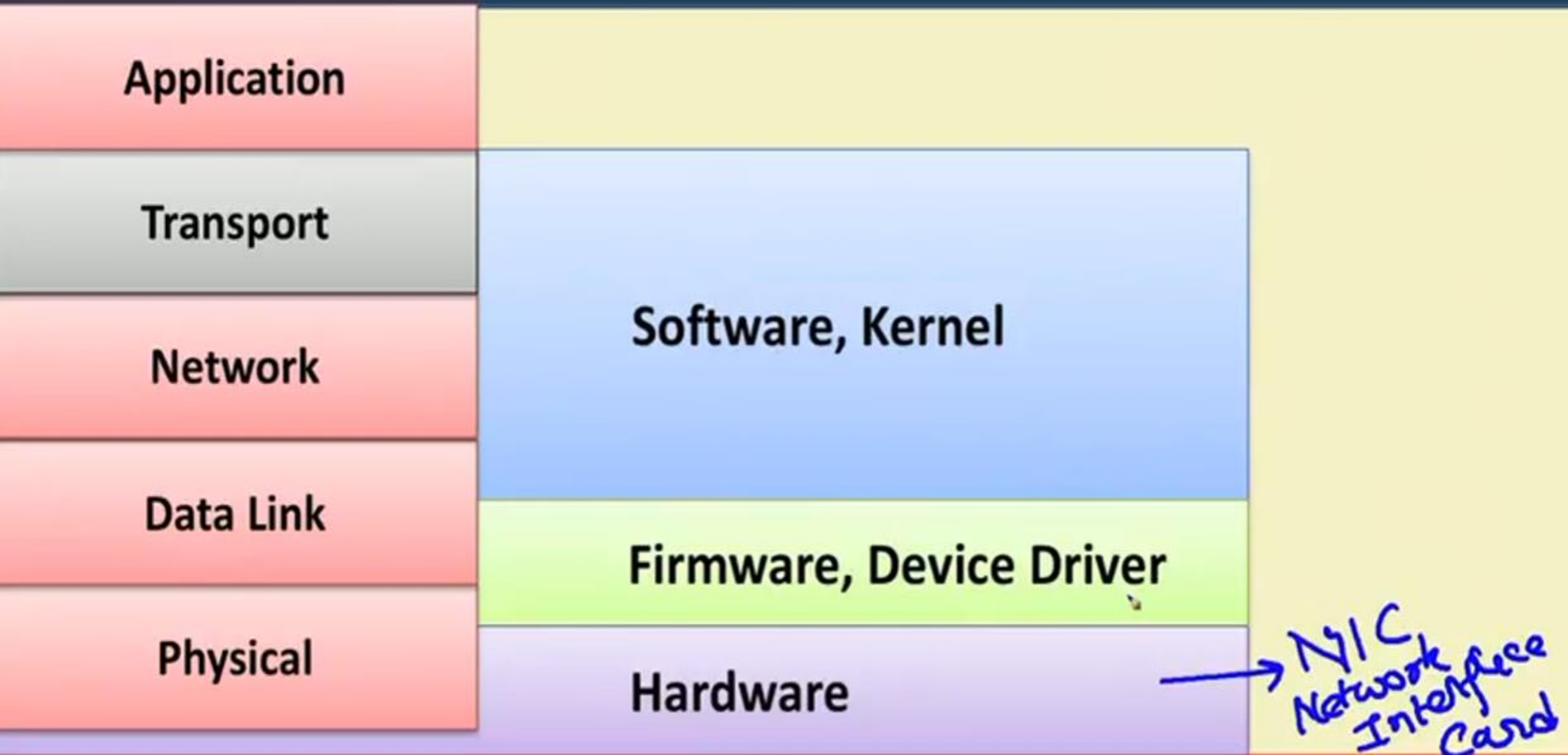
The Data Link layer links the software and hardware layers.



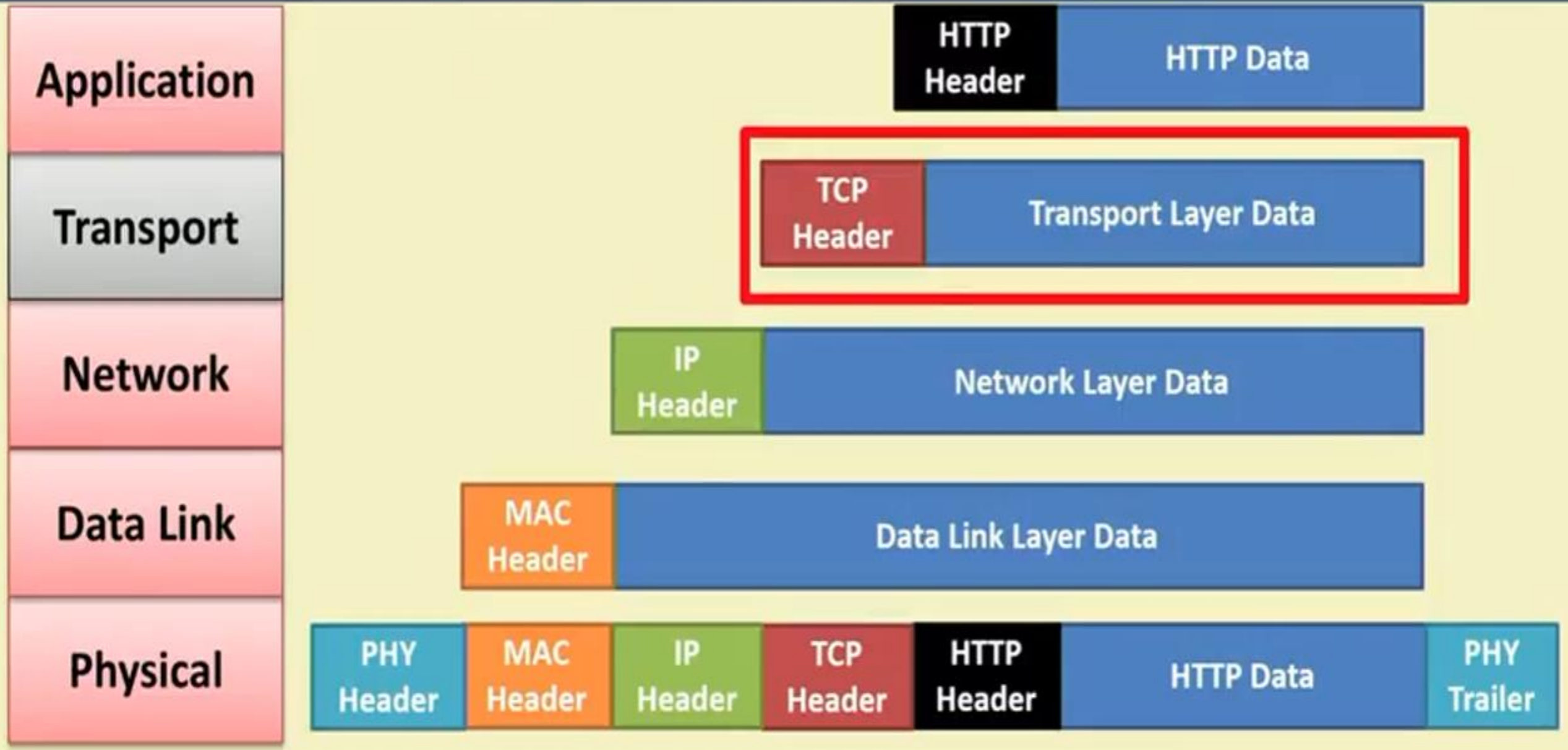
Physical devices devoted to the Data Link layer have both hardware and software components.

PC NIC

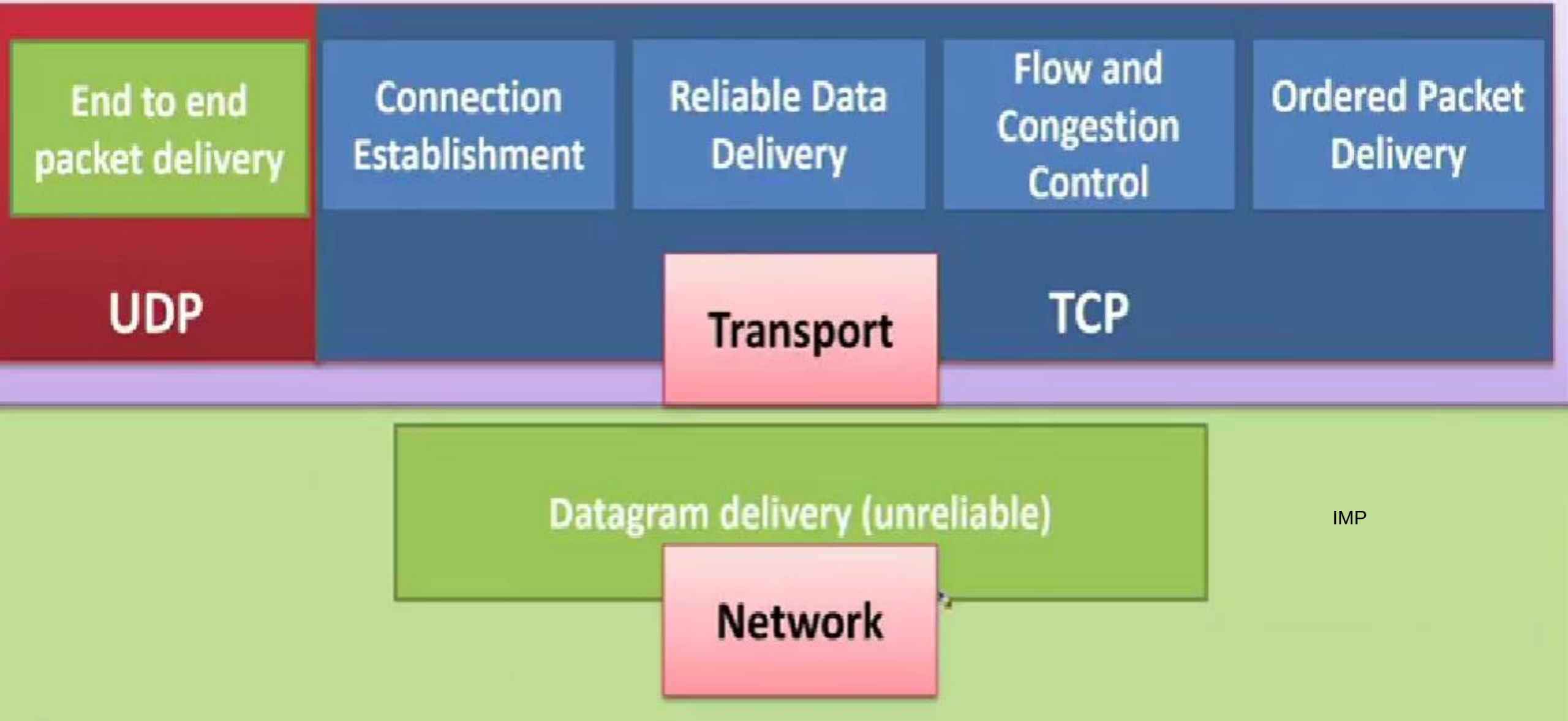
Protocol Stack Implementation in a Host



How Application Data Passes Through Different Layers



Transport Layer Services



IMP

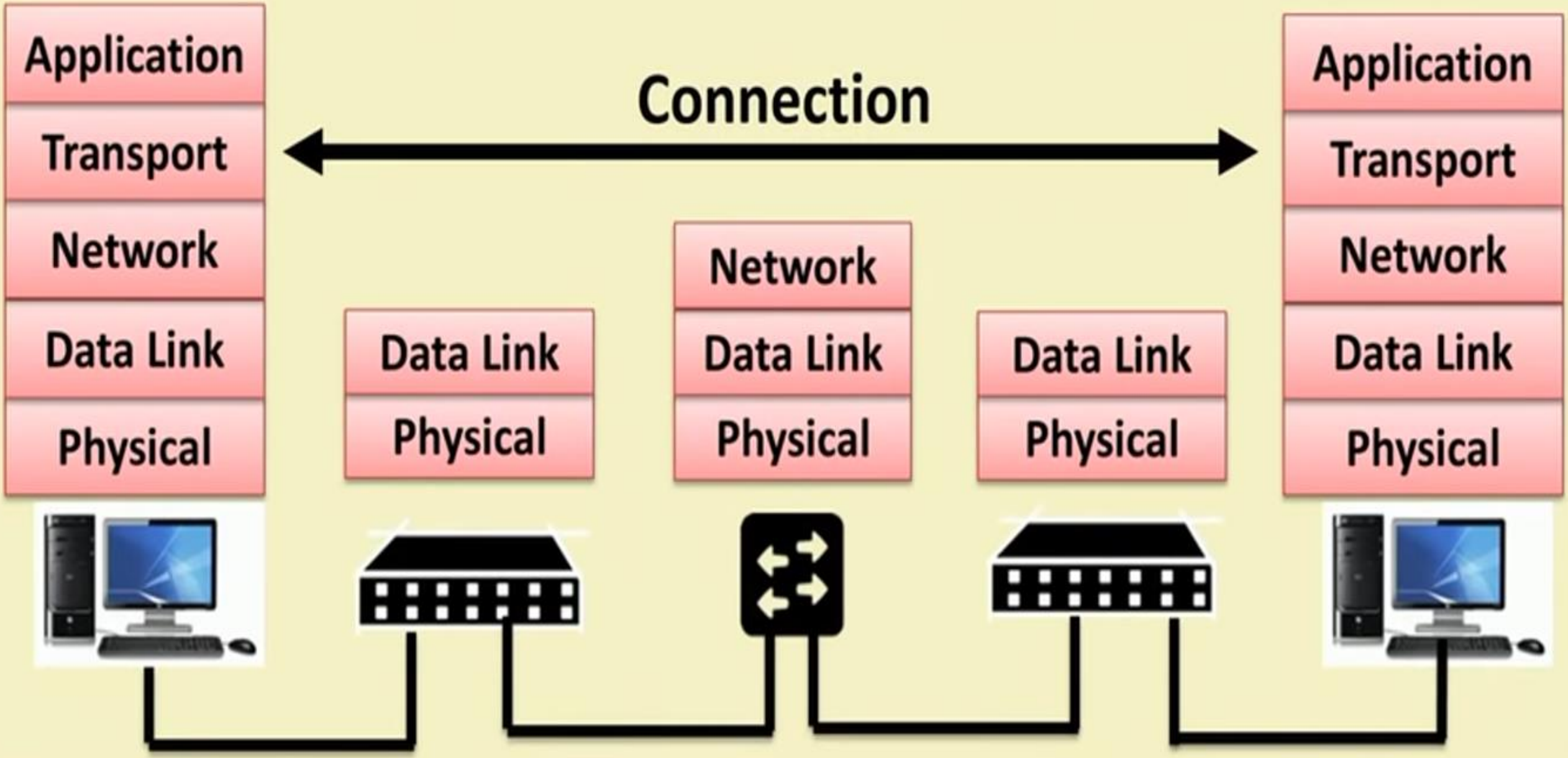
Flow Control

TCP uses Sliding Window mechanism at octet level. The window size can be variable over time. This is achieved by utilizing the concept of "Window Advertisement" based on :

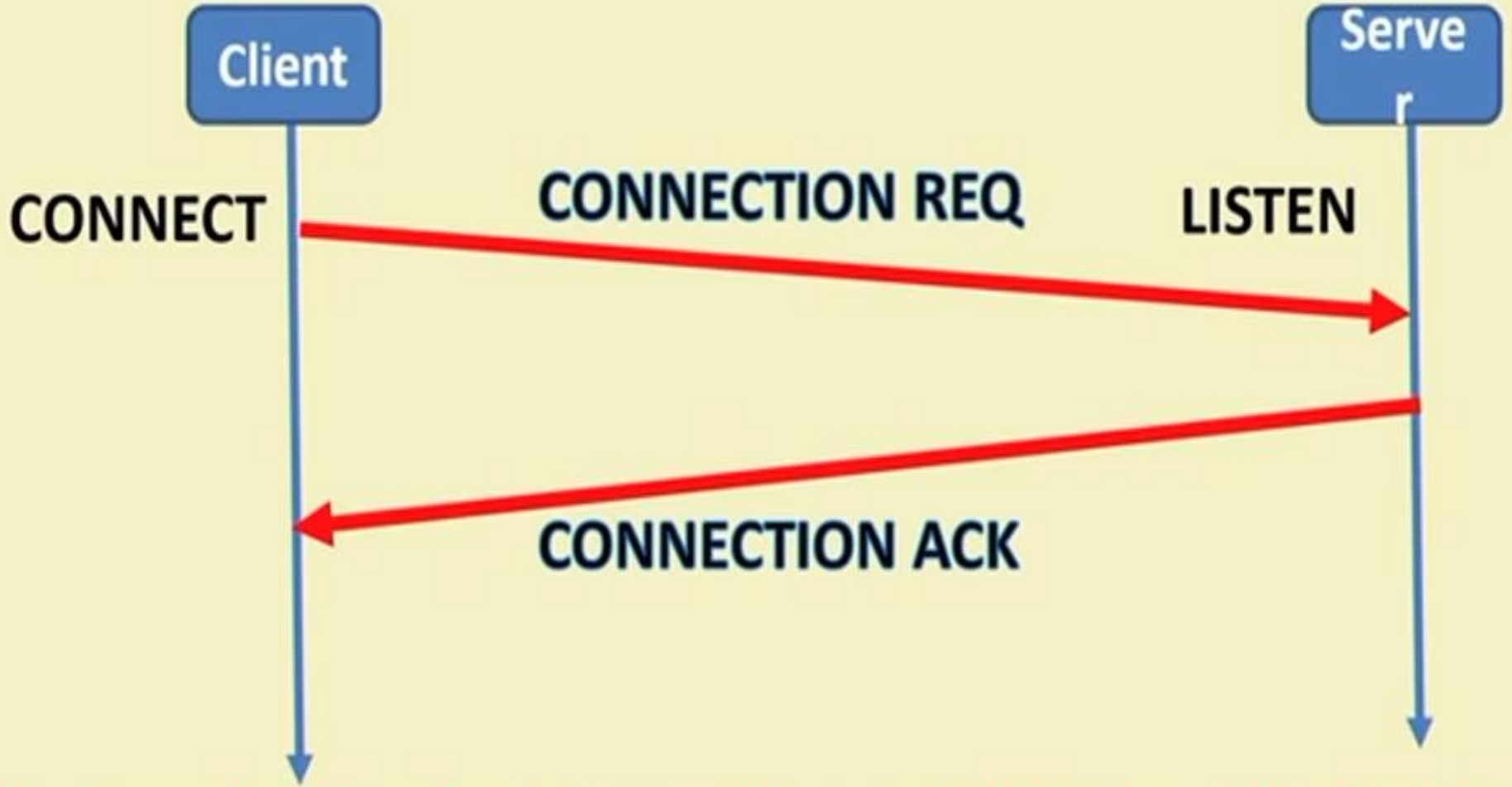
1. **Buffer availabilty at the receiver**
2. **Network conditions (traffic load etc.)**

In the former case receiver varies its window size depending upon the space available in its buffers. The window is referred as RECEIVE WINDOW (Recv_Win). When receiver buffer begin to fill it advertises a small Recv_Win so that the sender does'nt send more data than it can accept. If all buffers are full receiver sends a "Zero" size advertisement. It stops all transmission. When buffers become available receiver advertises a Non Zero widow to resume retransmission. The sender also periodically probes the "Zero" window to avoid any deadlock if the Non Zero Window advertisement from receiver is lost. The Variable size Recv_Win provides efficient end to end flow control. The second case arises when some intermediate node (e.g. a router) controls the source to reduce transmission rate. Here another window referred as CONGESTION WINDOW (C_Win) is utilized. Advertisement of C_Win helps to check and avoid congestion.

Transport Layer - II (Connection)



Connection Establishment

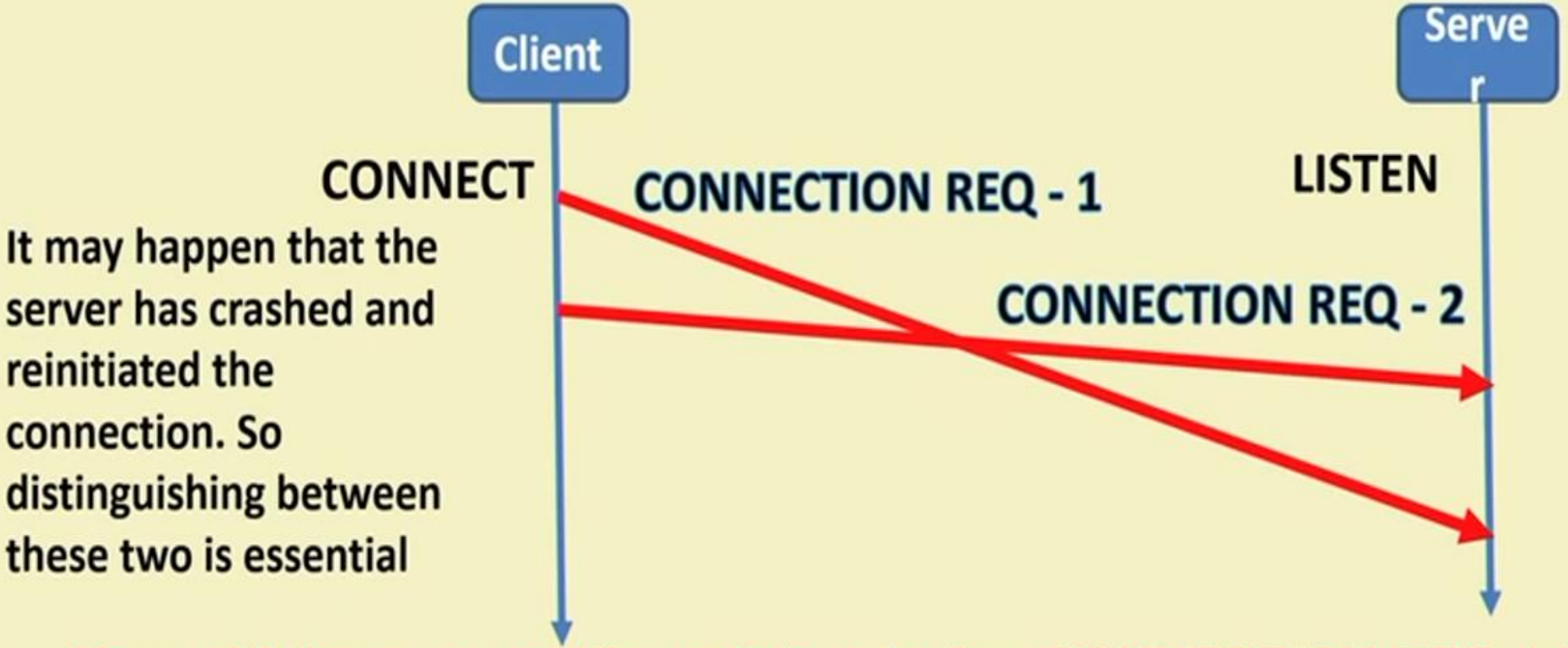


- This is a simple primitive for connection establishment – but does this work good?

Connection Establishment

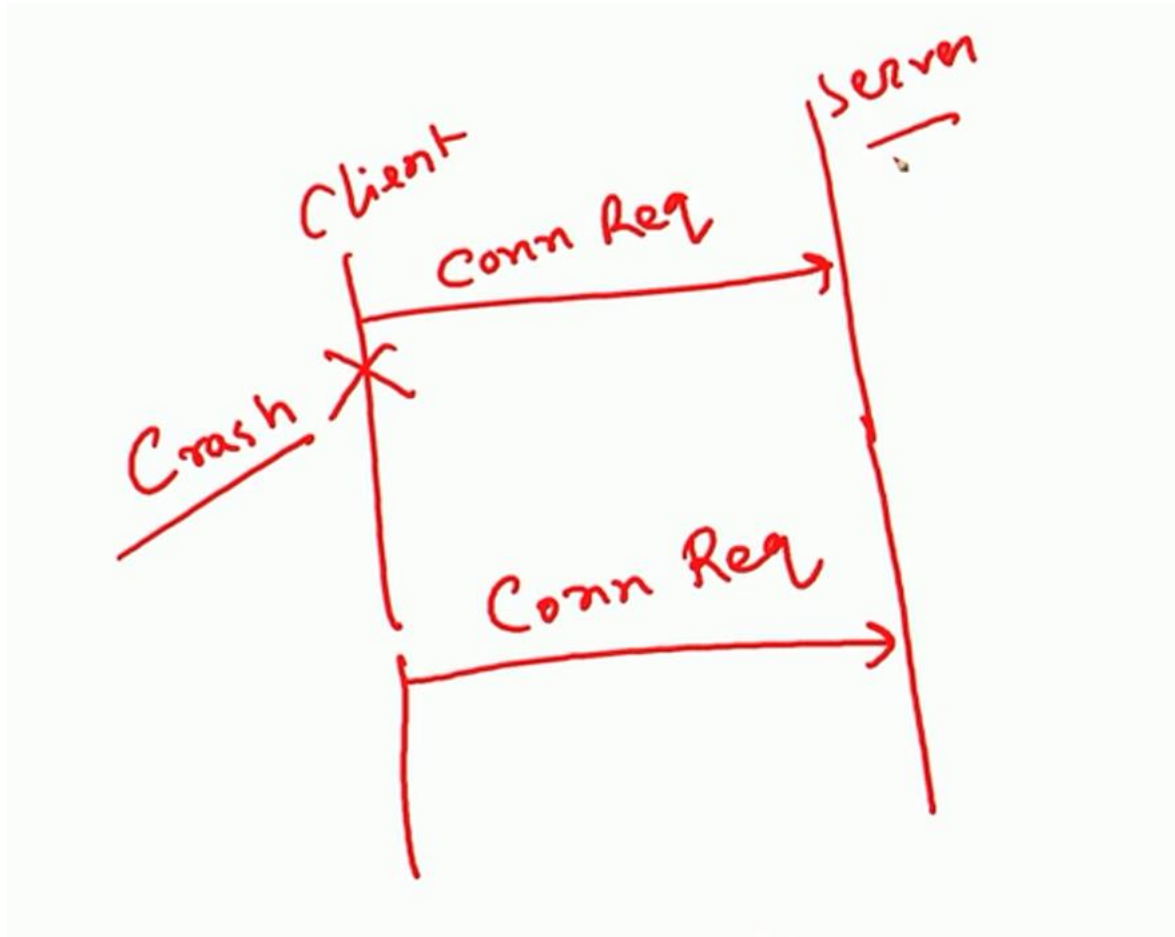
- Consider a scenario when the network can lose, delay, corrupt and duplicate packets (the underline network layer uses unreliable data delivery)
- Consider retransmission for ensuring reliability – every packet uses different paths to reach the destination
- Packets may be delayed and got struck in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

Connection Establishment



It may happen that the server has crashed and reinitiated the connection. So distinguishing between these two is essential

- How will the server differentiate whether CONNECTION REQ-1 is a new connection request or a duplicate of the CONNECTION REQ-2?


Client
Server
Conn Req
Crash
Conn Req

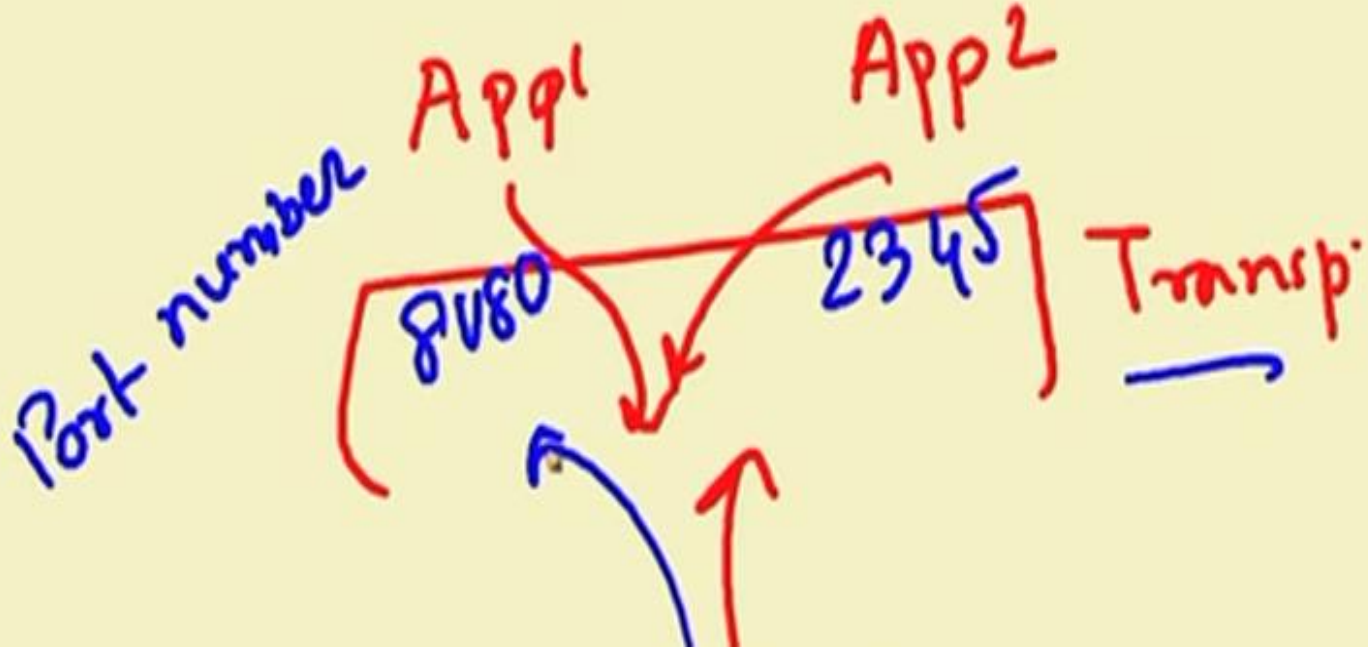
Connection Establishment

- **Protocol correctness versus Protocol performance – an eternal debate in computer networks …**

- 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

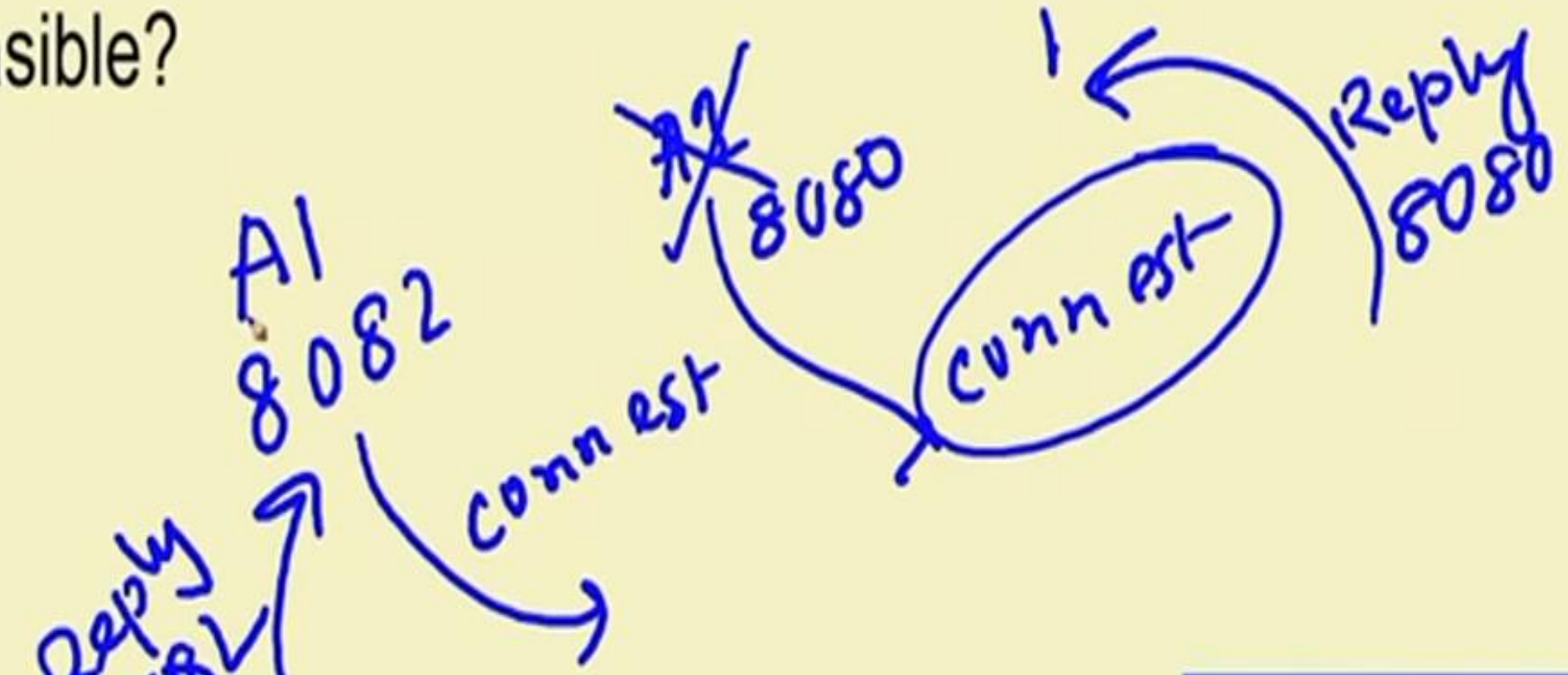
Connection Establishment – Handling Delayed Duplicates

- **Solution 1: Use Throwaway Transport Address (Port Numbers)**
  - Do not use a port number if it has been used once already – Delayed duplicate packets will never find their way to a transport process
  - Is this solution feasible?



Connection Establishment – Handling Delayed Duplicates

- **Solution 1: Use Throwaway Transport Address (Port Numbers)**
  - Do not use a port number if it has been used once already – Delayed duplicate packets will never find their way to a transport process
  - Is this solution feasible?



Connection Establishment – Handling Delayed Duplicates

- **Solution 1: Use Throwaway Transport Address (Port Numbers)**
  - Do not use a port number if it has been used once already – Delayed duplicate packets will never find their way to a transport process
  - Is this solution feasible?

- **Solution 2: Give each connection a unique identifier chosen by the initiating party and put in each segment**
  - Can you see any problem in this approach?

Connection Establishment – Handling Delayed Duplicates

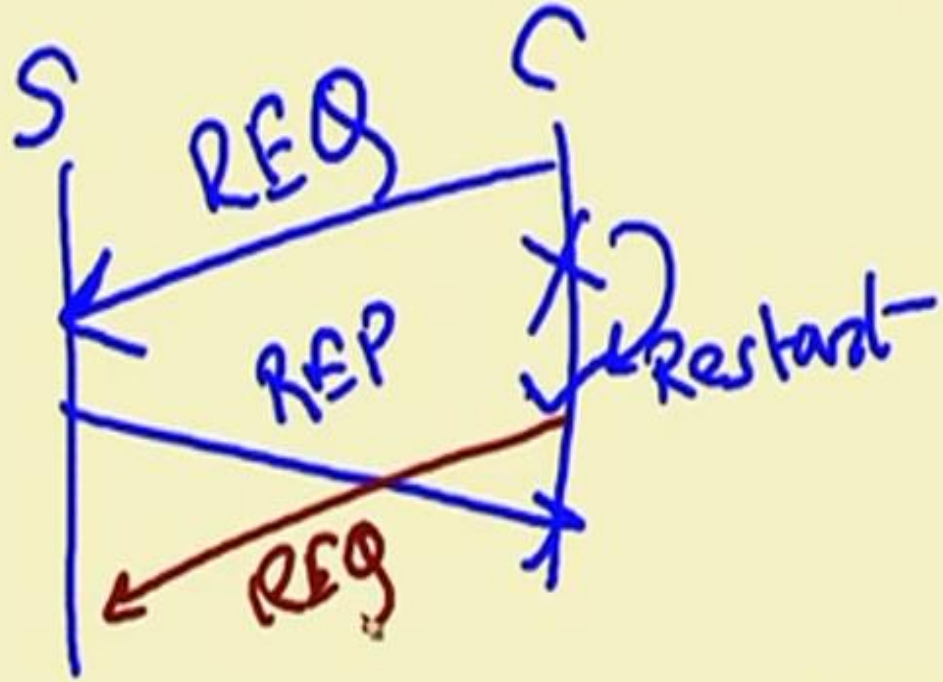
- **Solution 3:** Devise a mechanism to kill off aged packets that are still hobbling about (Restrict the packet lifetime)
  - Makes it possible to design a feasible solution

Connection Establishment – Handling Delayed Duplicates

- 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.

Connection Establishment – Handling Delayed Duplicates

- 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**

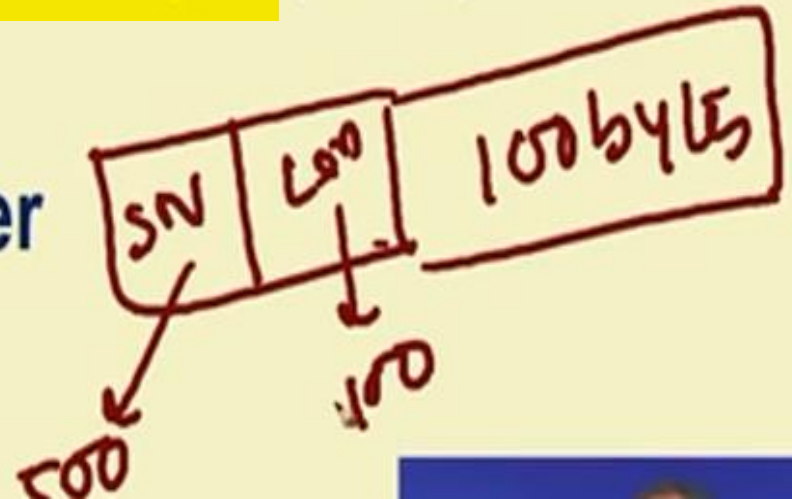
Connection Establishment – Handling Delayed Duplicates

- **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**

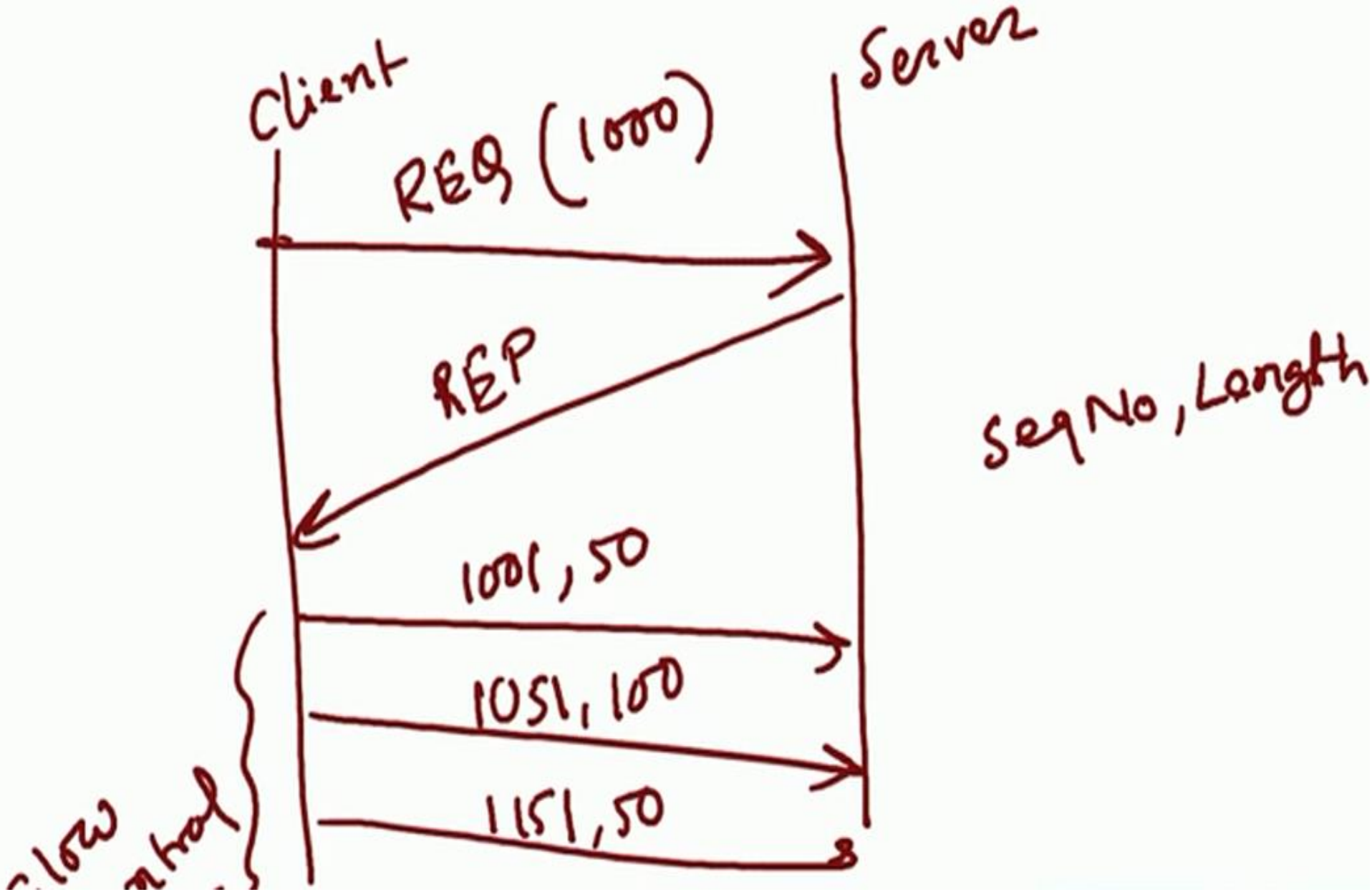
Sequence Number Adjustment

- **Two important requirements (*Tomlinson 1975, Selecting Sequence Numbers*)**
  - **R1.** 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**

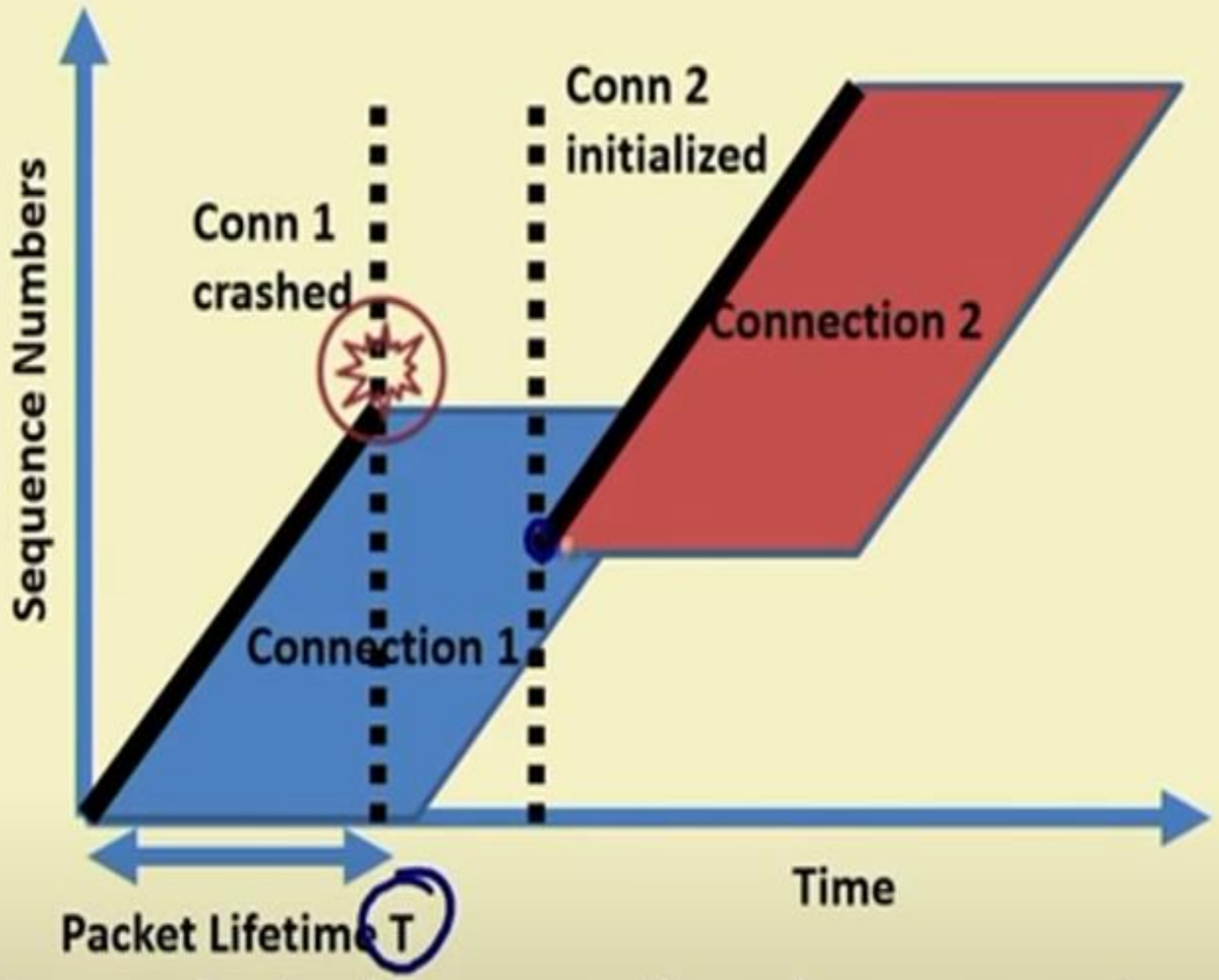


Sequence Number Adjustment

- **Two important requirements (*Tomlinson 1975, Selecting Sequence Numbers*)**
  - **R2:** 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**

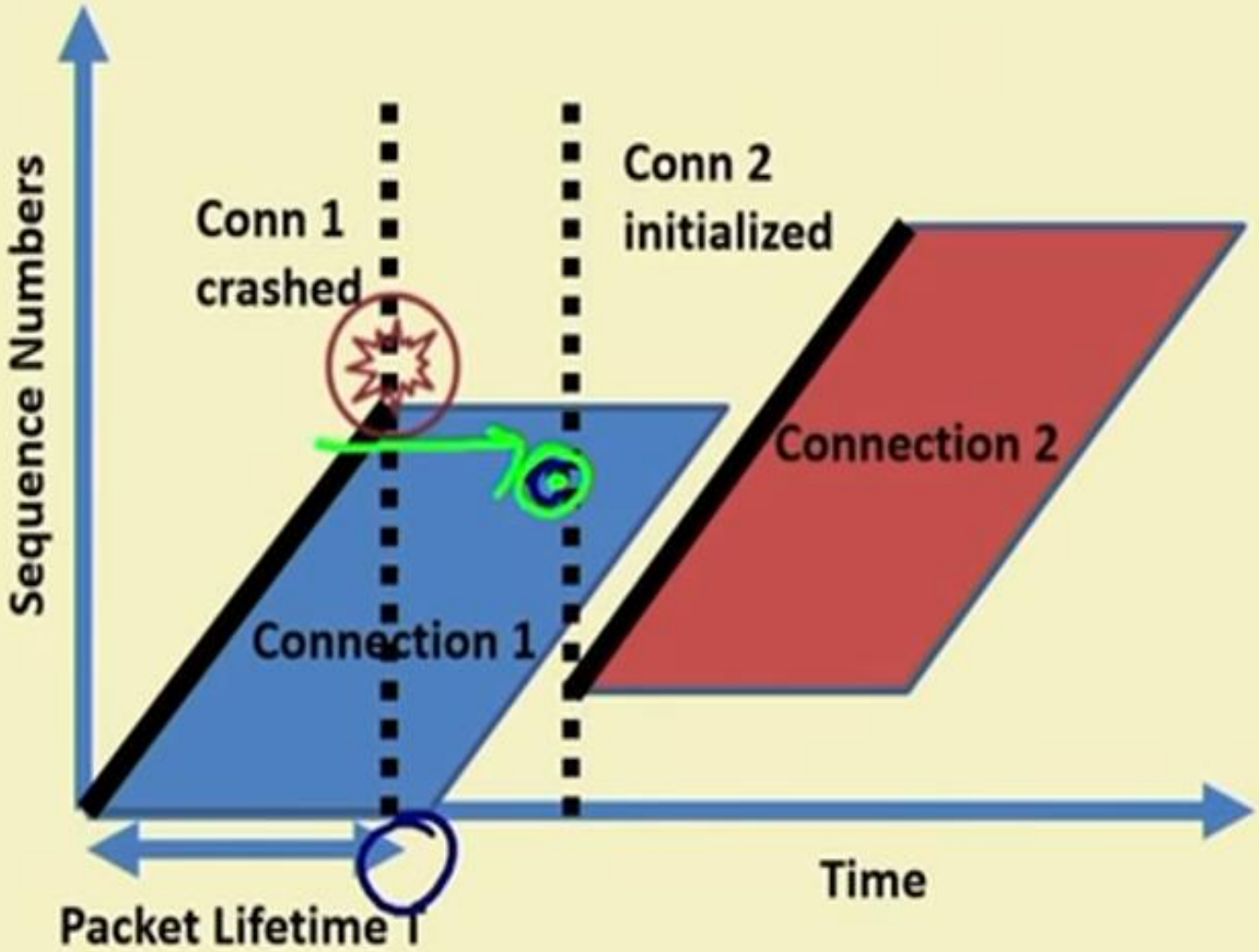

Client
Server
REQ (1000)
REP
SeqNo, Length
1001, 50
1051, 100
1151, 50

Initial Sequence Number during Connection Establishment

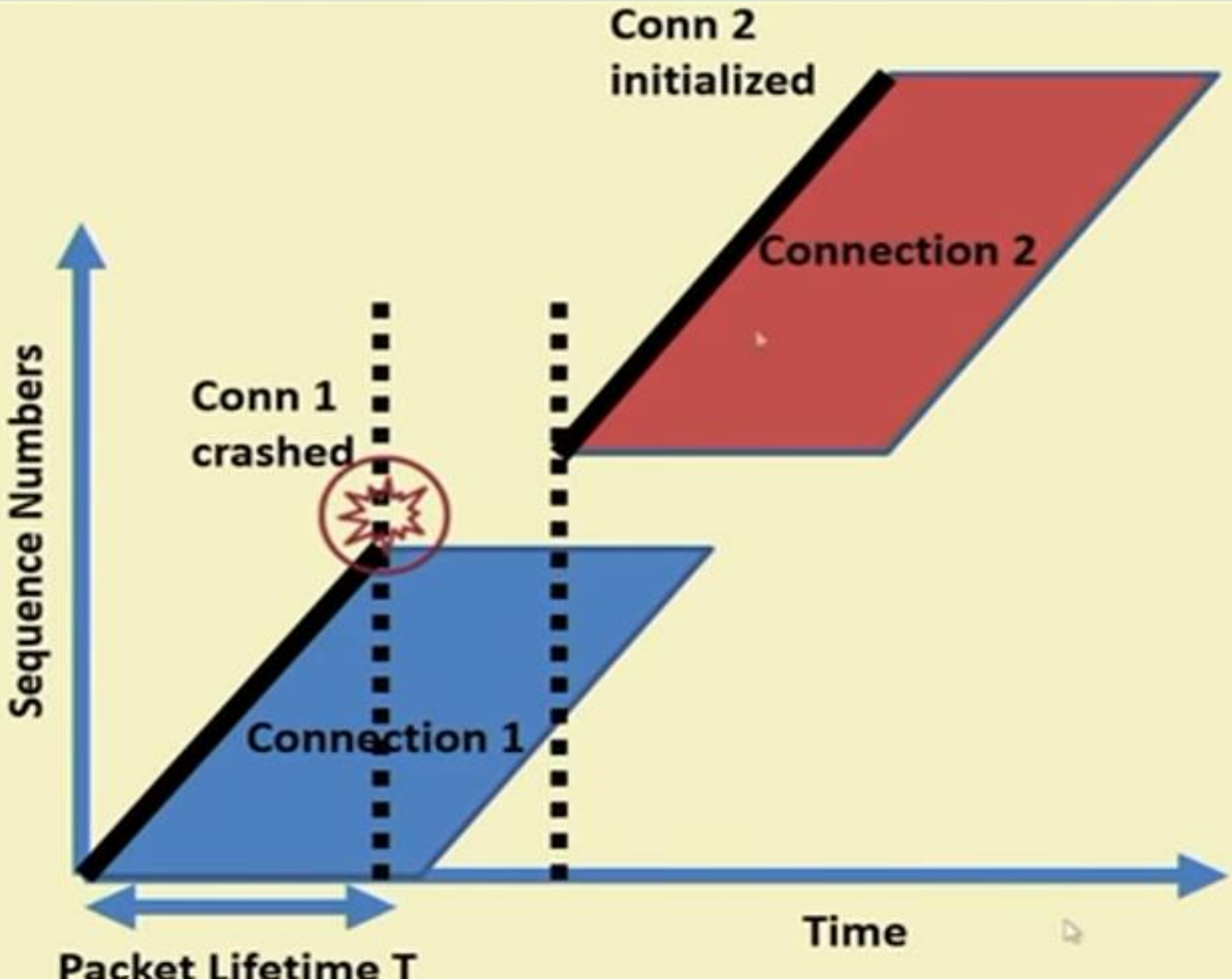


- A Delayed duplicate packet of connection 1 can create a confusion for connection 2

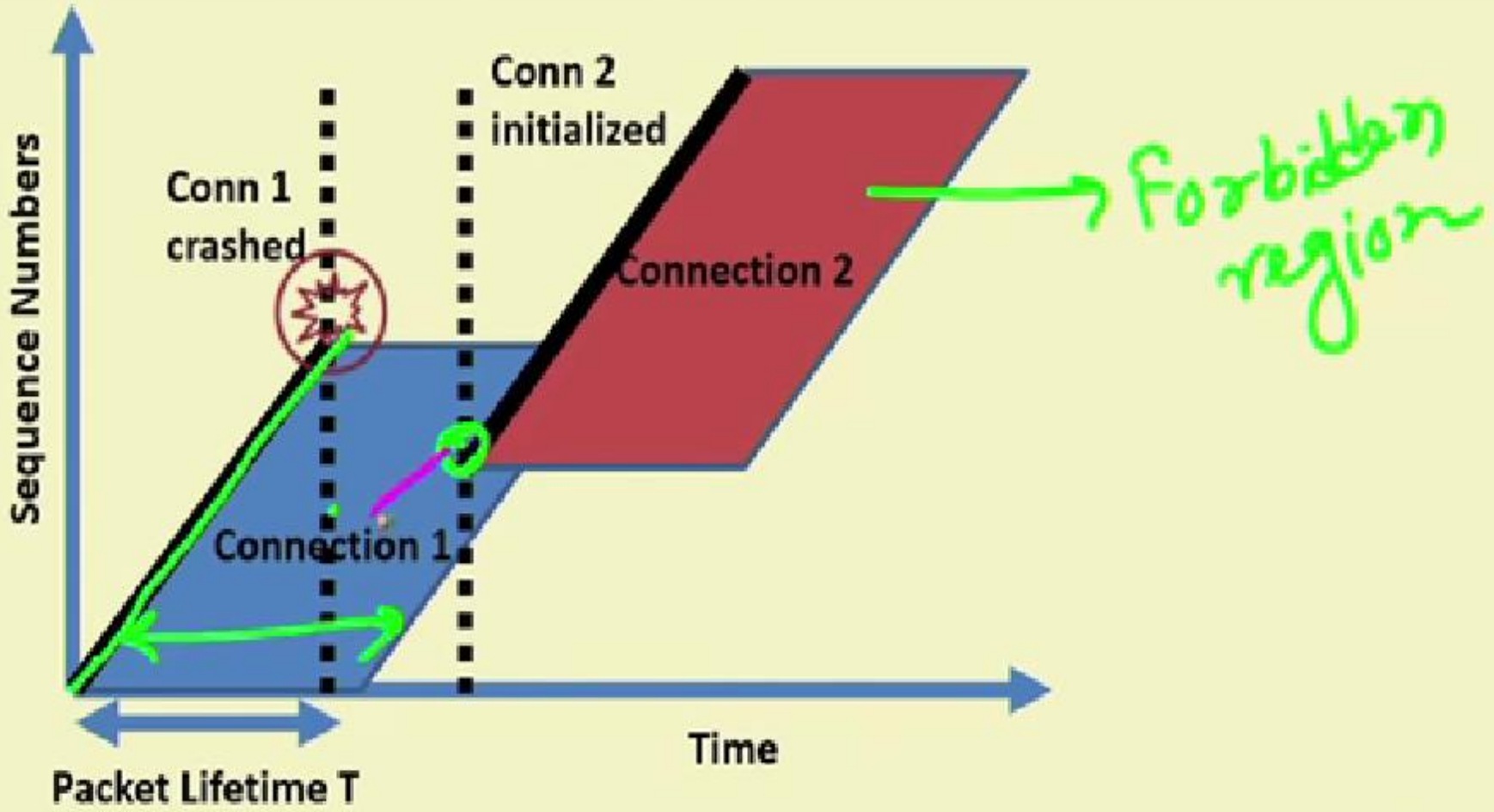
What We Ideally Want? Either …



What We Ideally Want? Or …

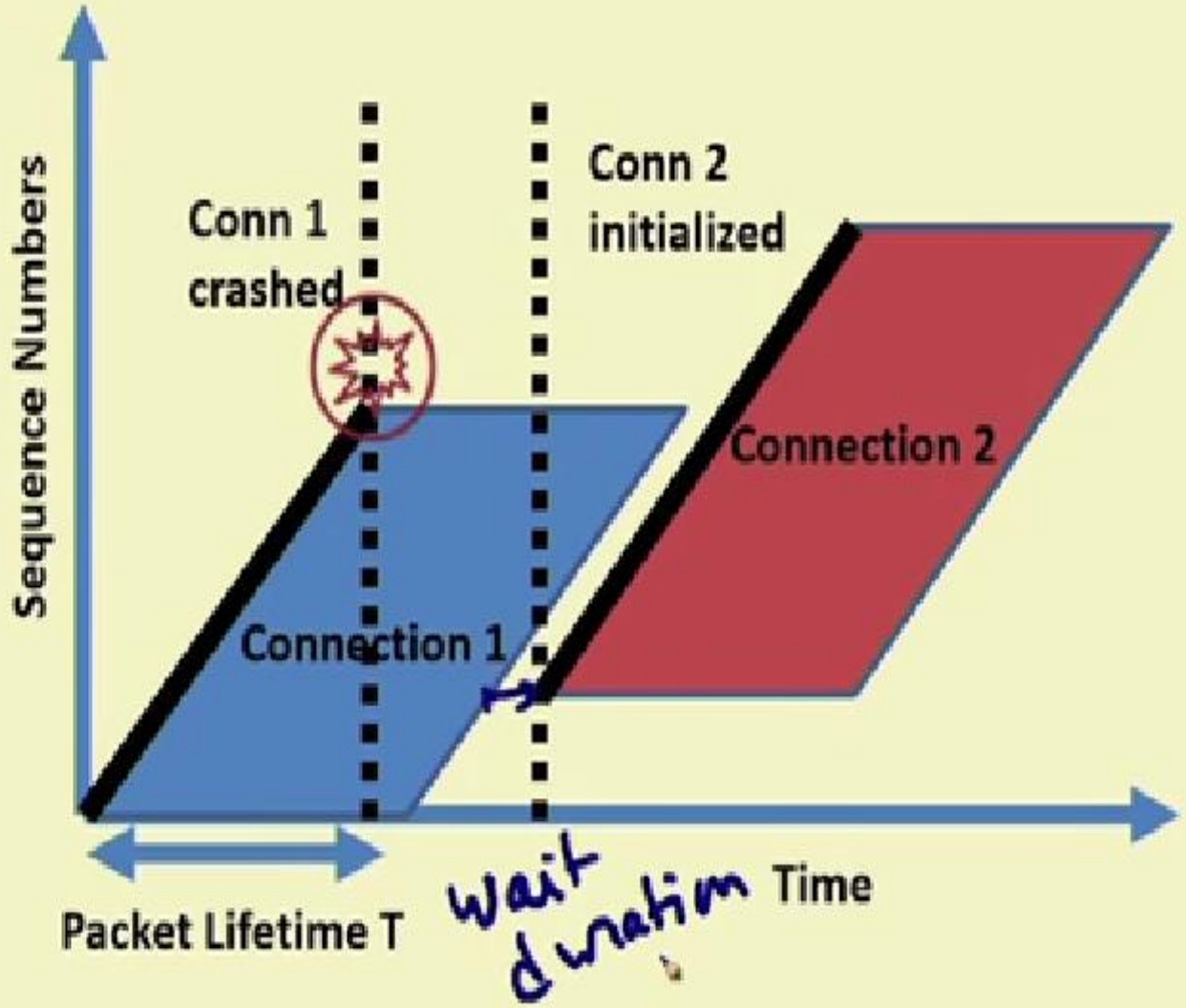


Initial Sequence Number during Connection Establishment

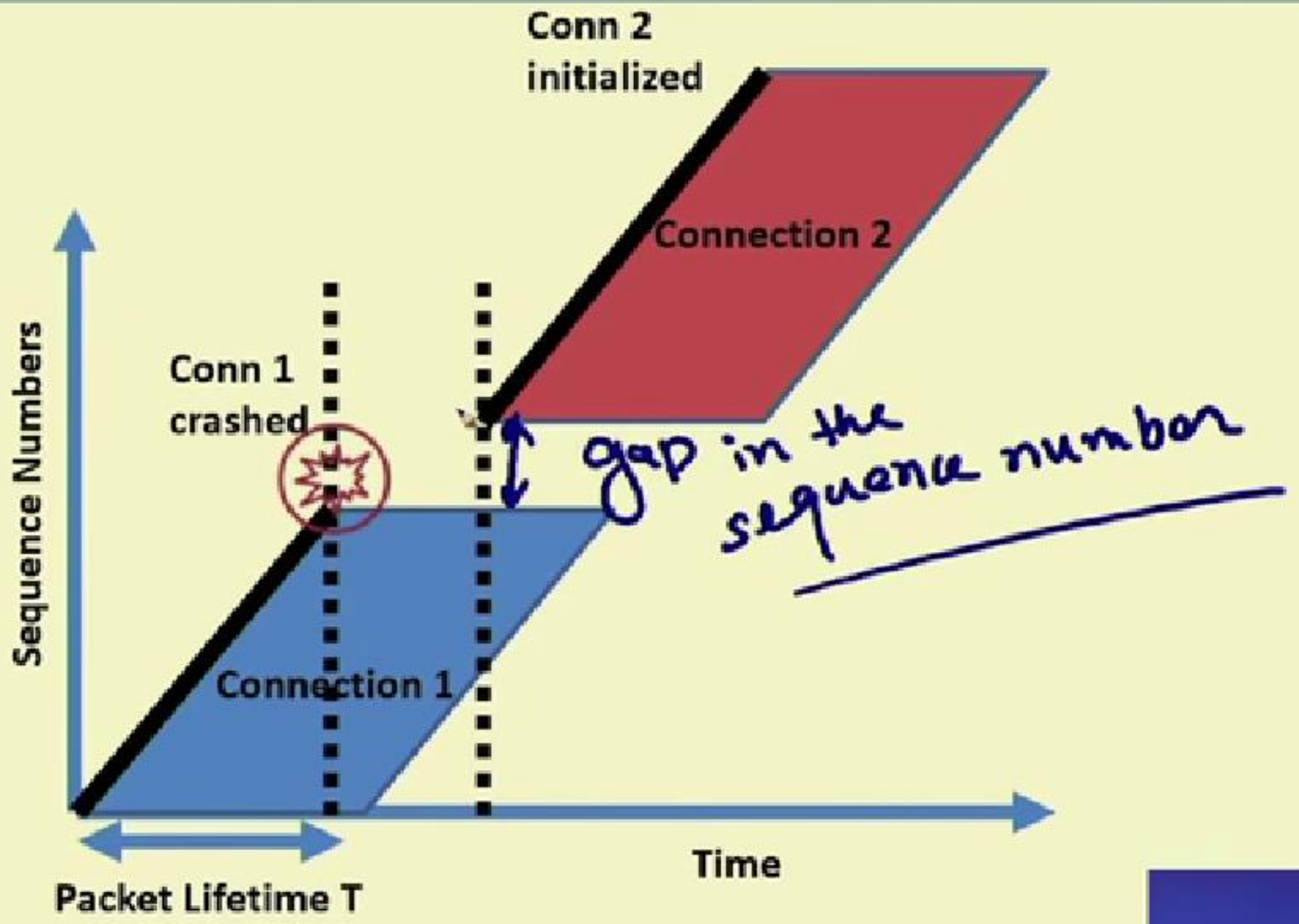


- A Delayed duplicate packet of connection 1 can create a confusion for connection 2

What We Ideally Want? Either ...

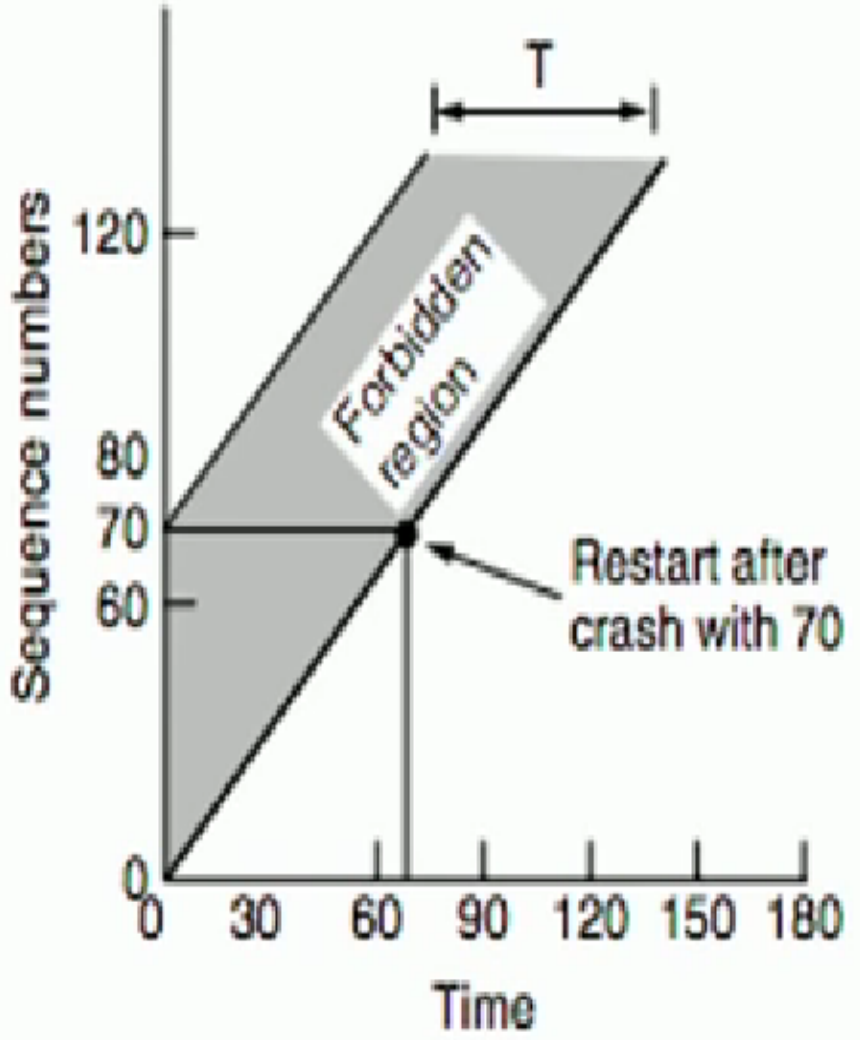


What We Ideally Want? Or …



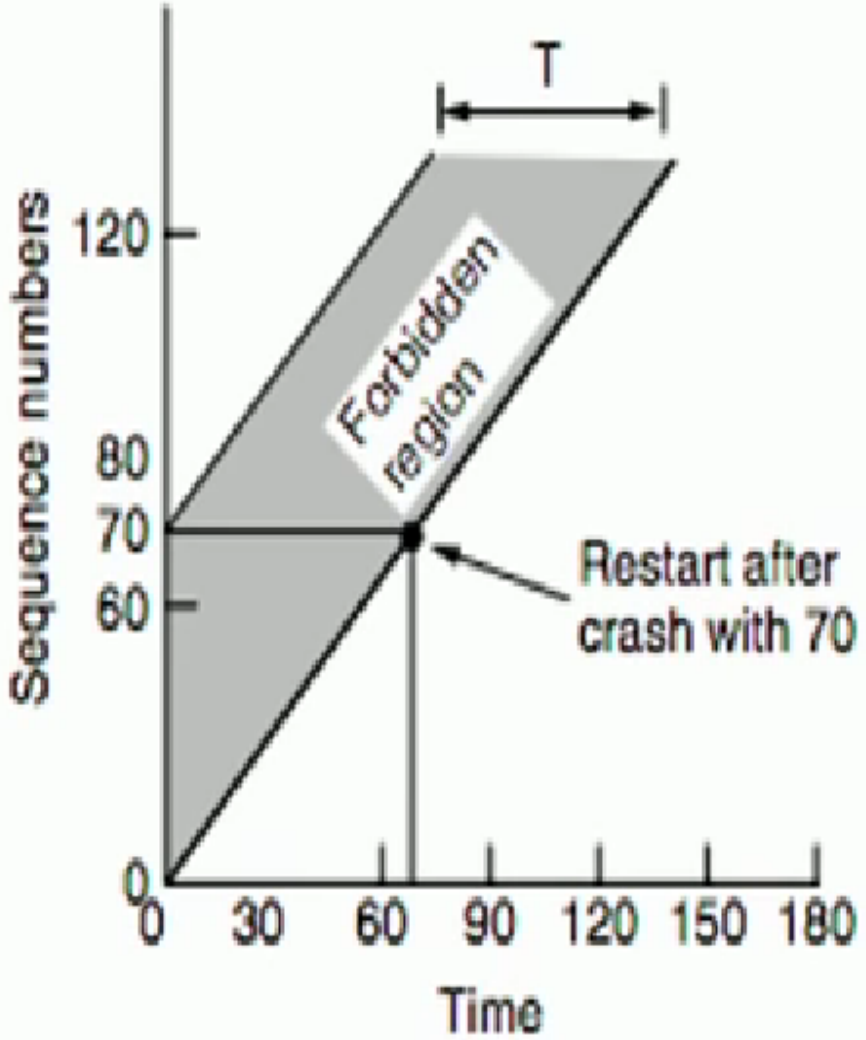
Connection Establishment – Handling Delayed Duplicates

- Receiver receives two segments having the same sequence number within a duration T
  - One packet must be the duplicate
  - The receiver discards the duplicate packets.



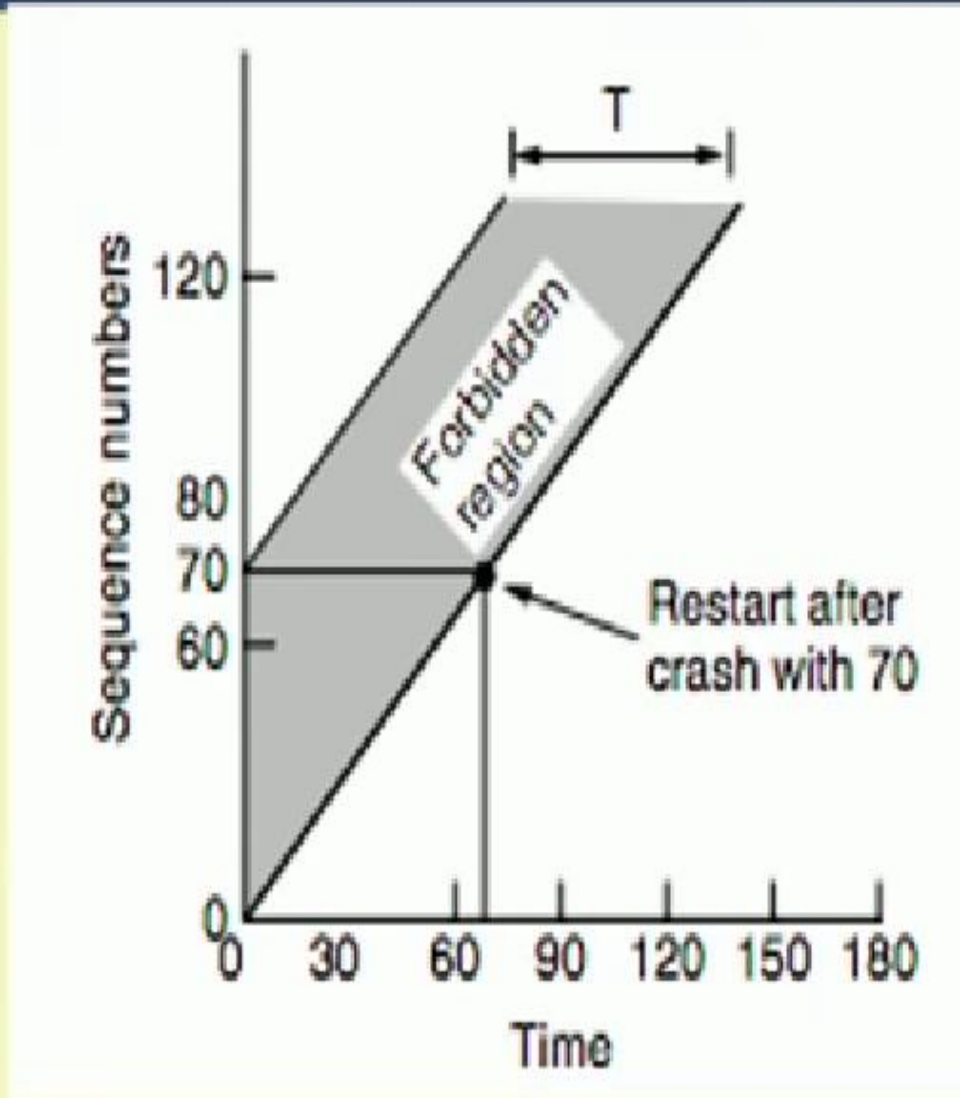
Connection Establishment – Handling Delayed Duplicates

- 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**



Connection Establishment – Handling Delayed Duplicates

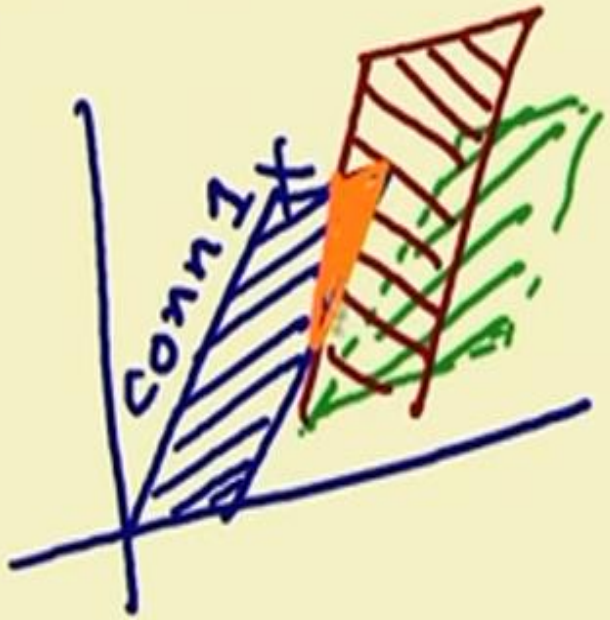
- **Adjust the initial sequence numbers properly** - A host does not restart with a sequence number in the forbidden region, based on the sequence number it used before crash and the time duration T.

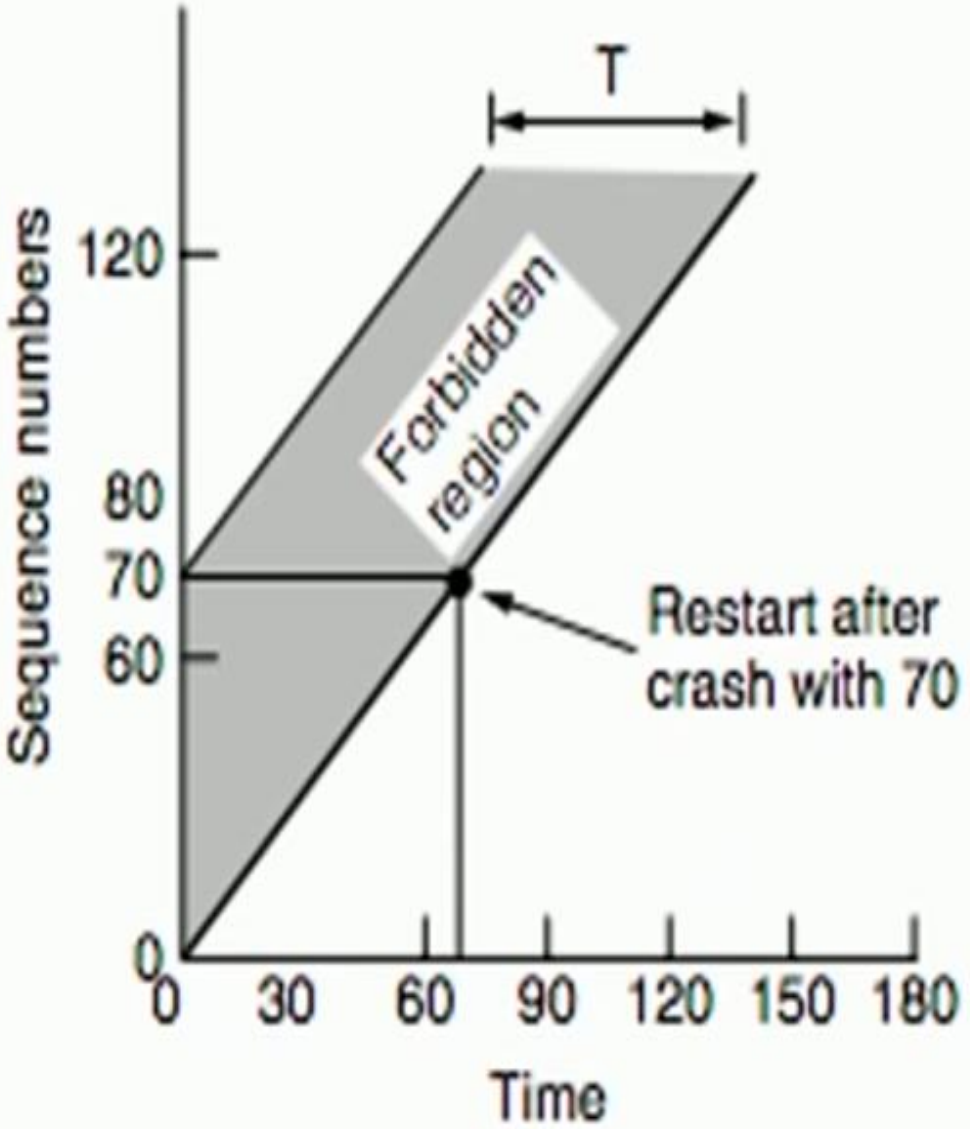


Packet Sequence Numbers are Out of the Forbidden Region

Two possible source of problems

1. A host sends too much data too fast on a newly opened connection

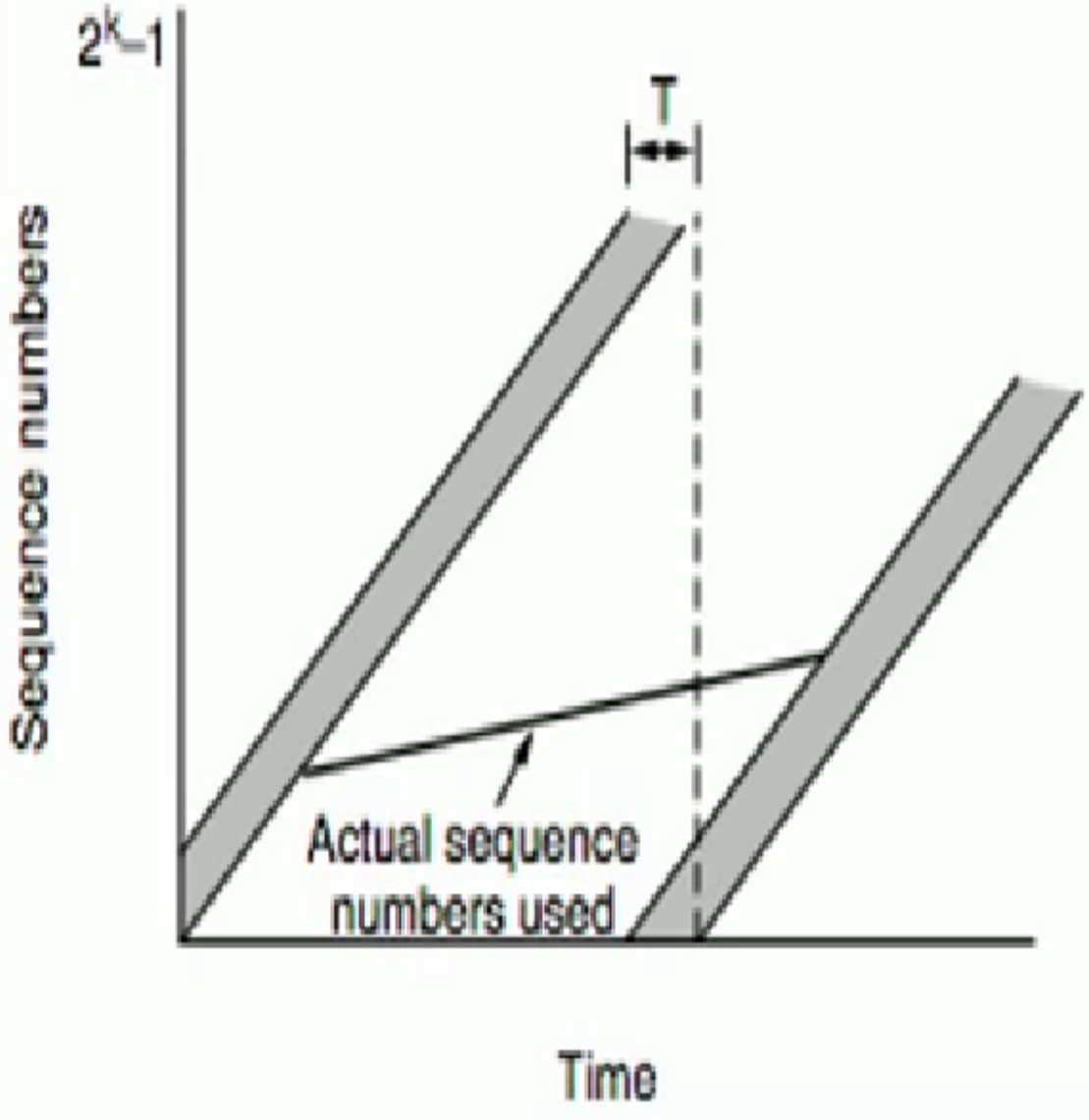




Packet Sequence Numbers are Out of the Forbidden Region

Two possible source of problems

2. The data rate is too slow that the sequence number for a previous connection enters the forbidden region for the next connection



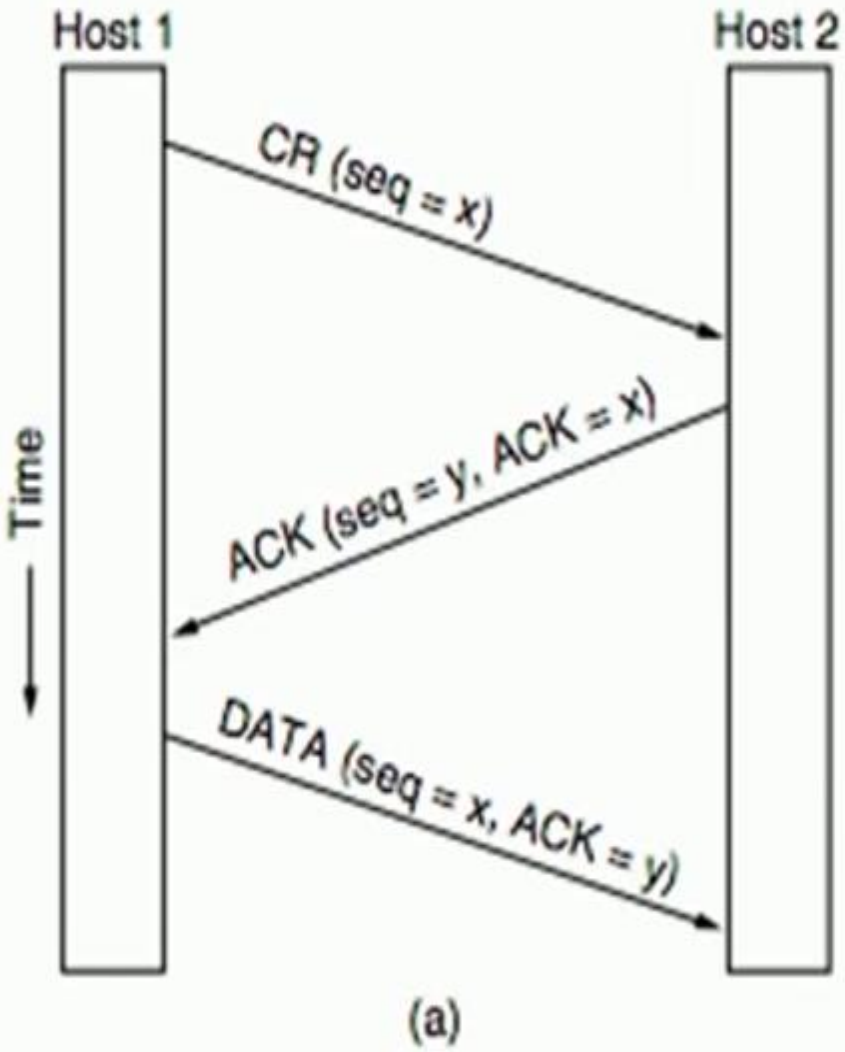
Adjusting the Sending Rate based on Sequence Numbers

- 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

Adjusting the Sending Rate based on Sequence Numbers

- **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**

Three Way Handshake



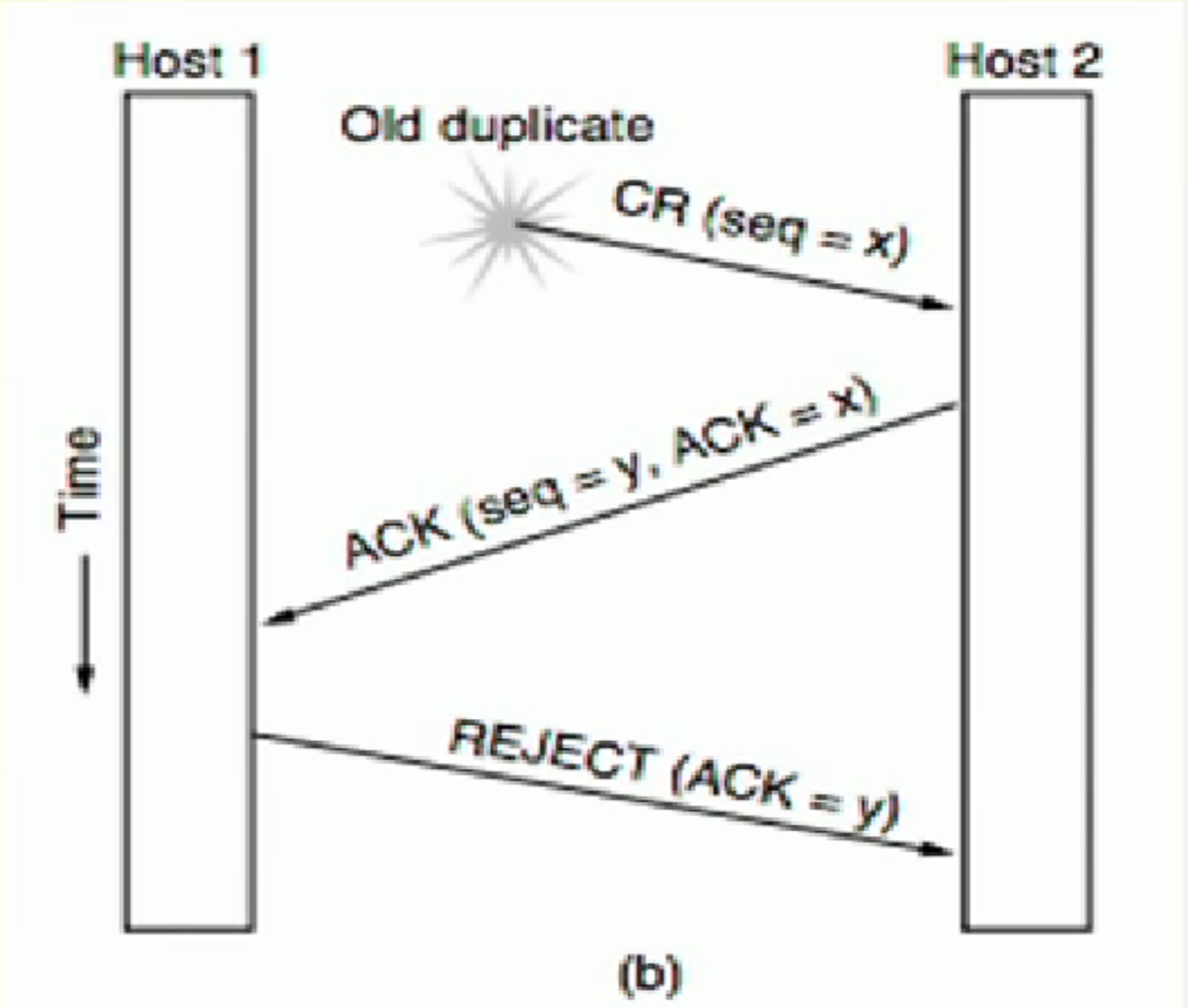
- By looking at the ACK, Host 1 ensures that Sequence number x does not belong to the forbidden region of any previously established connection

- By looking at the ACK in DATA, Host 2 ensures that sequence number y does not belong to the forbidden region of any previously established connection

rce: Computer Networks (5th Edition) by Tanenbaum,

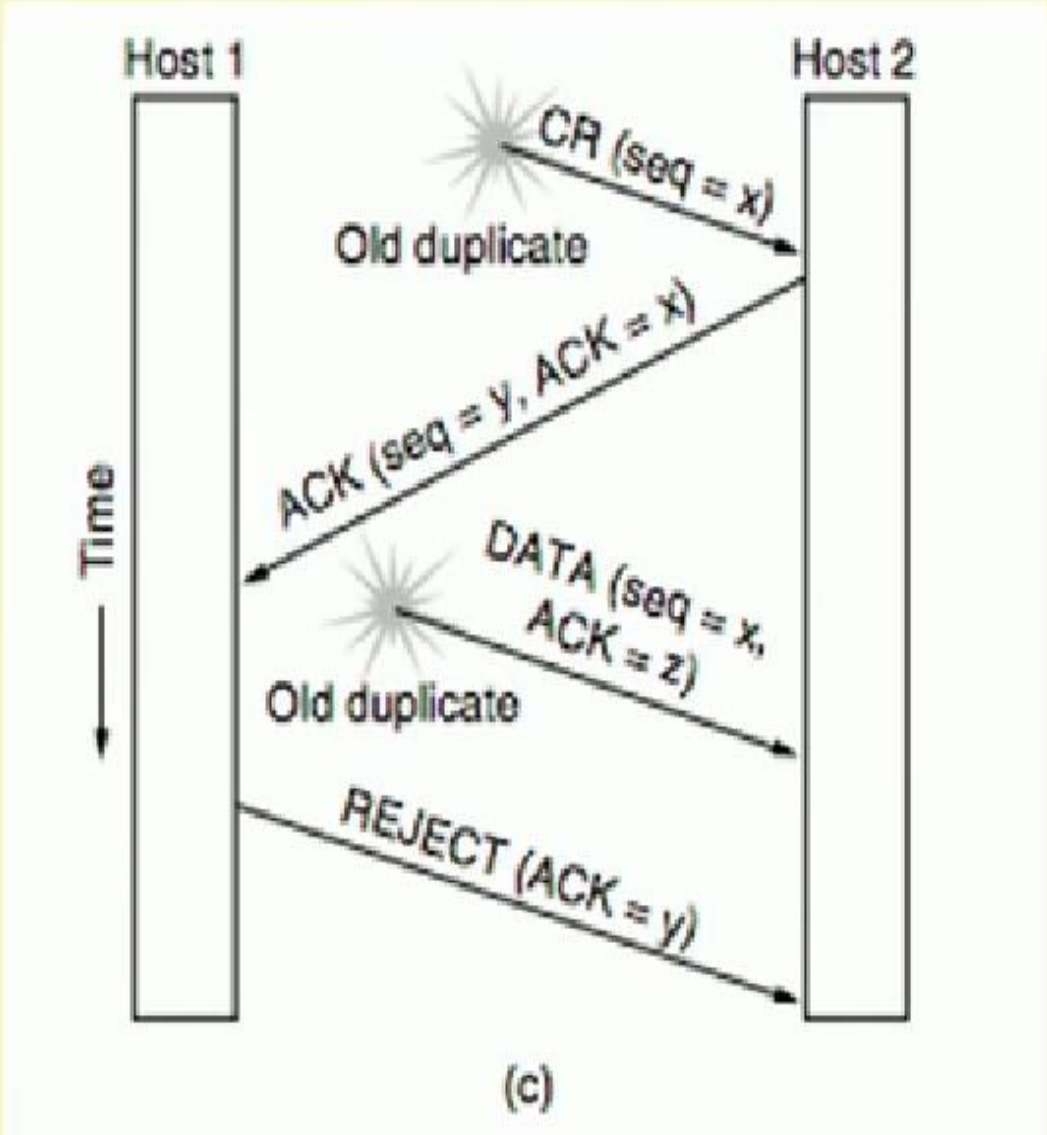
CONNECTION REQUEST is a Delayed Duplica

Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell



CONNECTION REQUEST and ACK both are Delayed Duplicates

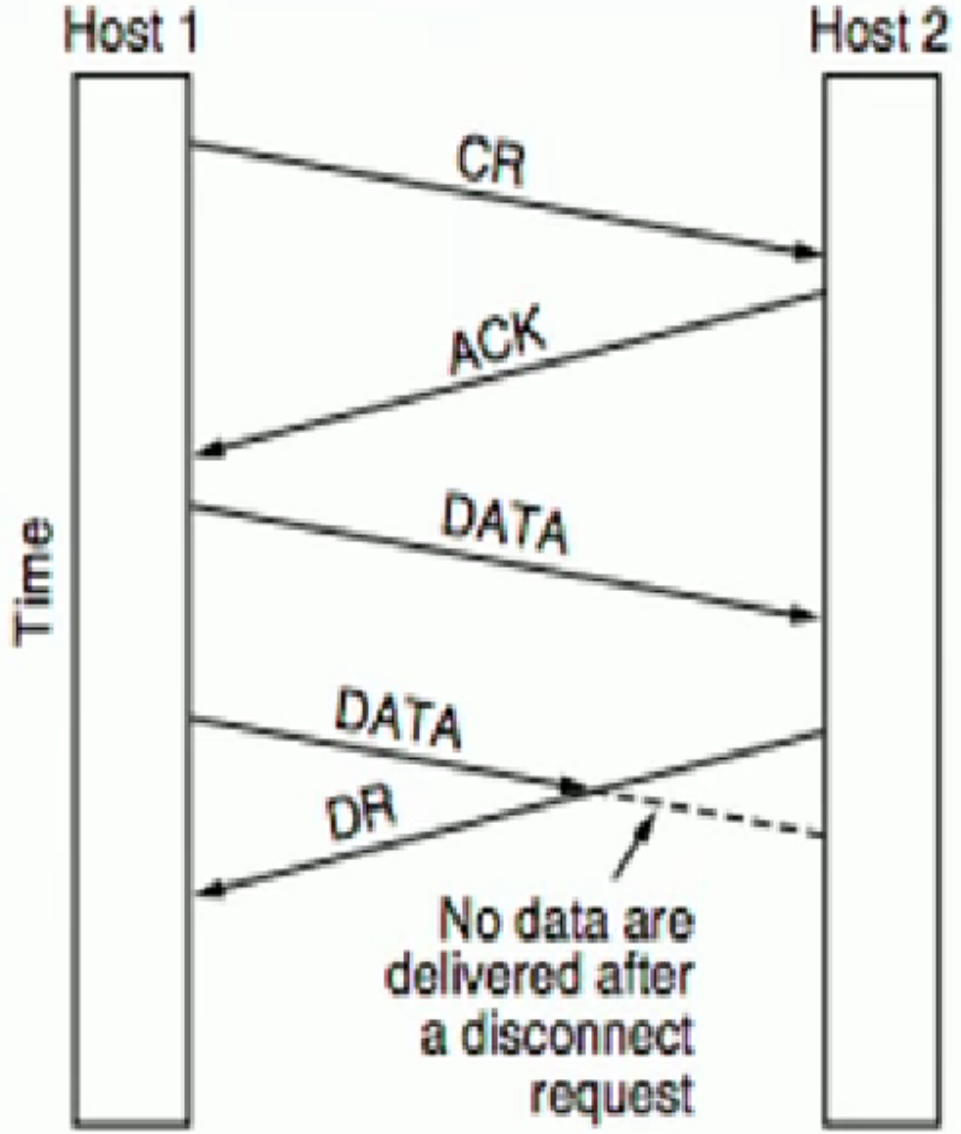
Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell



(c)

Connection Release – Asymmetric Release

- When one party hangs up, the connection is broken
- This may results in data loss



Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

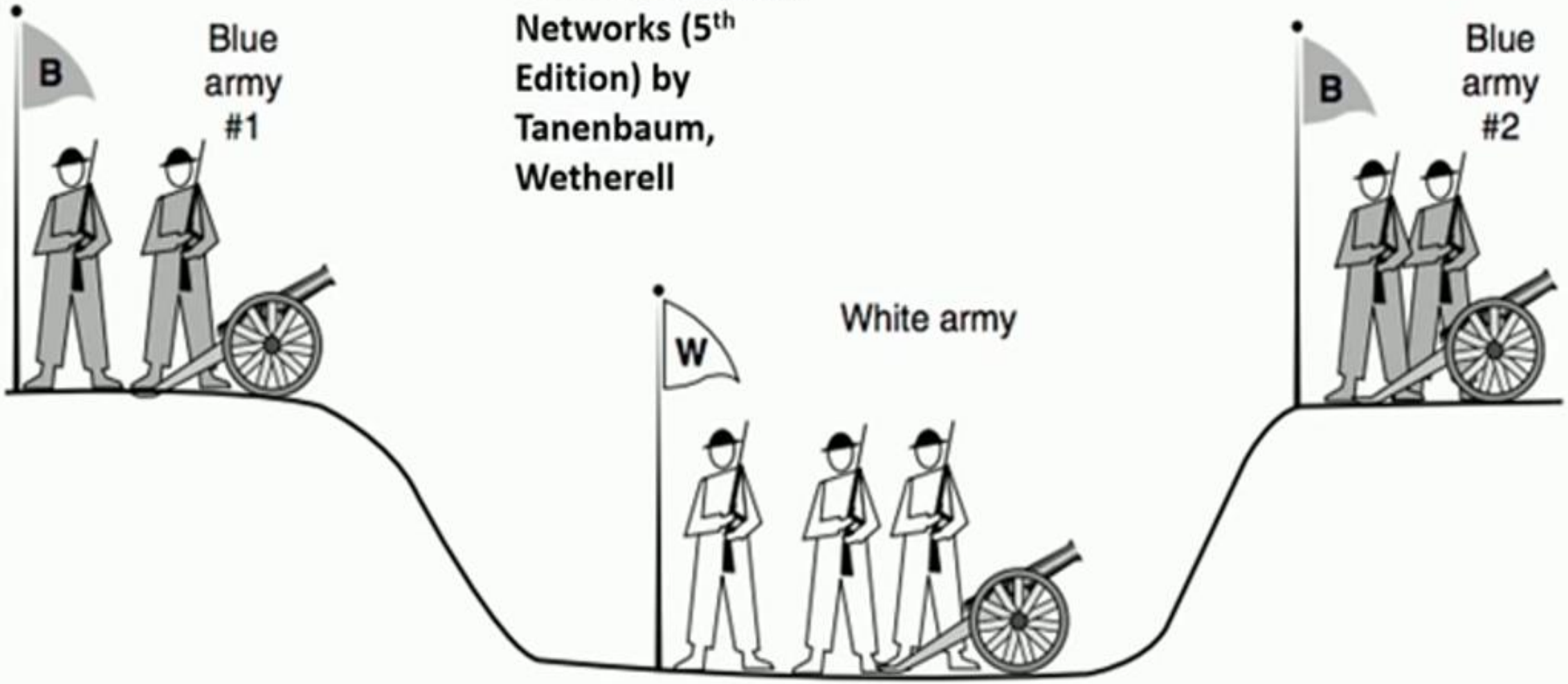
Connection Release – Symmetric Release

- Treats the connection as two separate unidirectional connections and requires each one to be released separately
- Does the job when each process has a fixed amount of data to send and clearly knows when it has sent it.
- What can be a protocol for this?

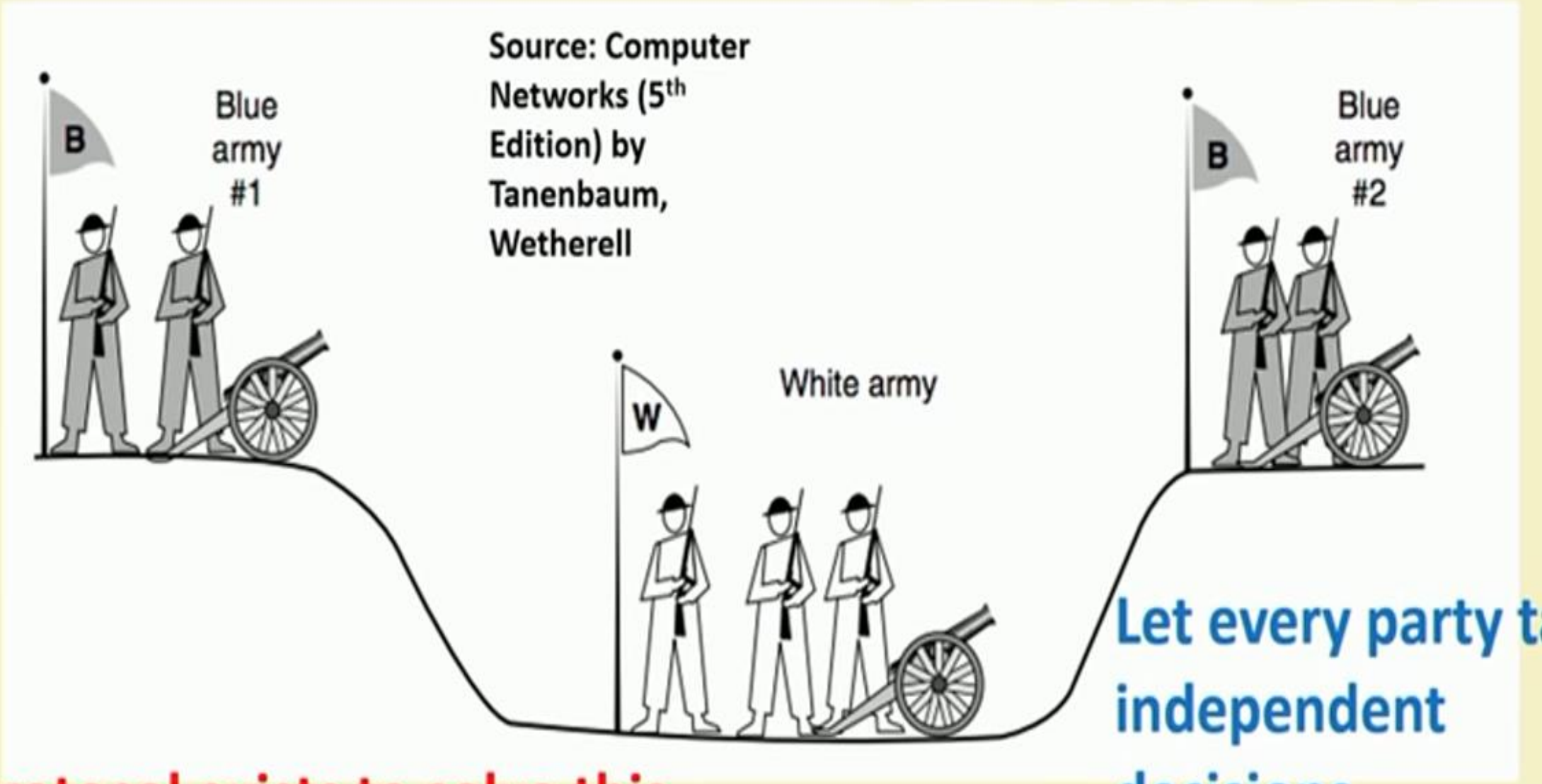
Connection Release – Symmetric Release

- Treats the connection as two separate unidirectional connections and requires each one to be released separately
- Does the job when each process has a fixed amount of data to send and clearly knows when it has sent it.
- What can be a protocol for this?
  - **Host 1: “I am done”**
  - **Host 2: ”I am done too”**
- **Does this protocol work good always?**

The Two Army Problem



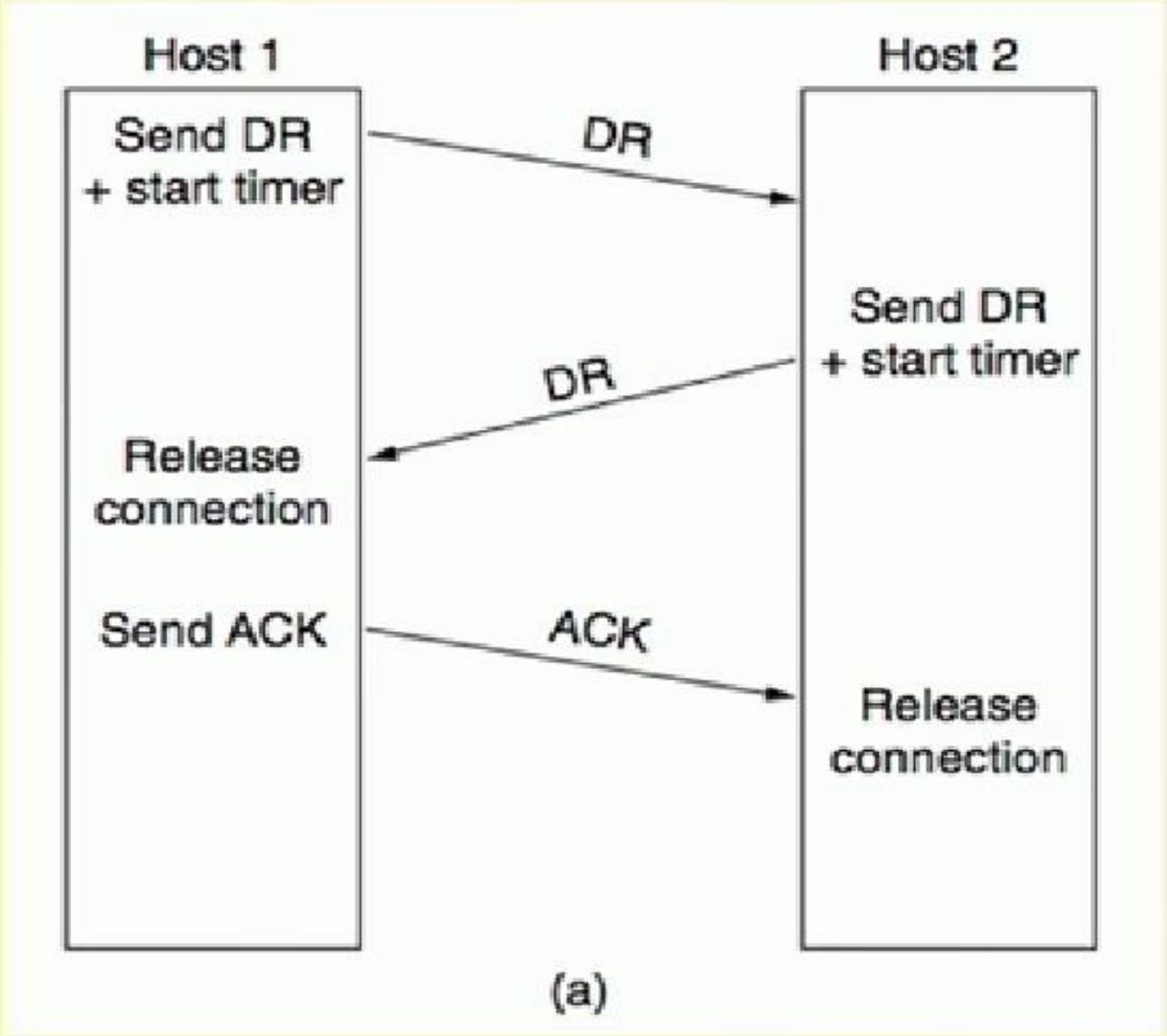
The Two Army Problem



Let every party take independent decisions

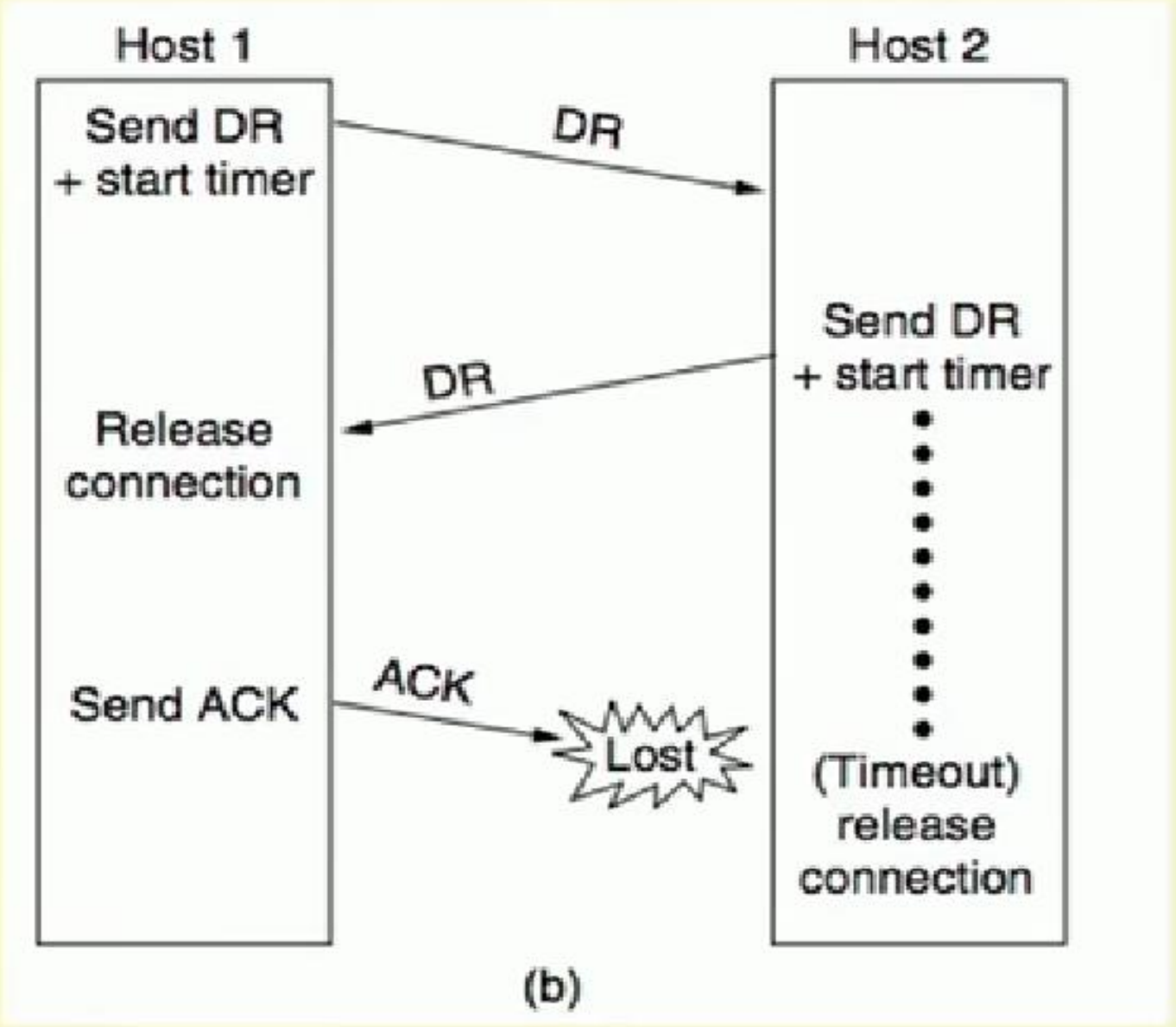
No protocol exists to solve this

Connection Release



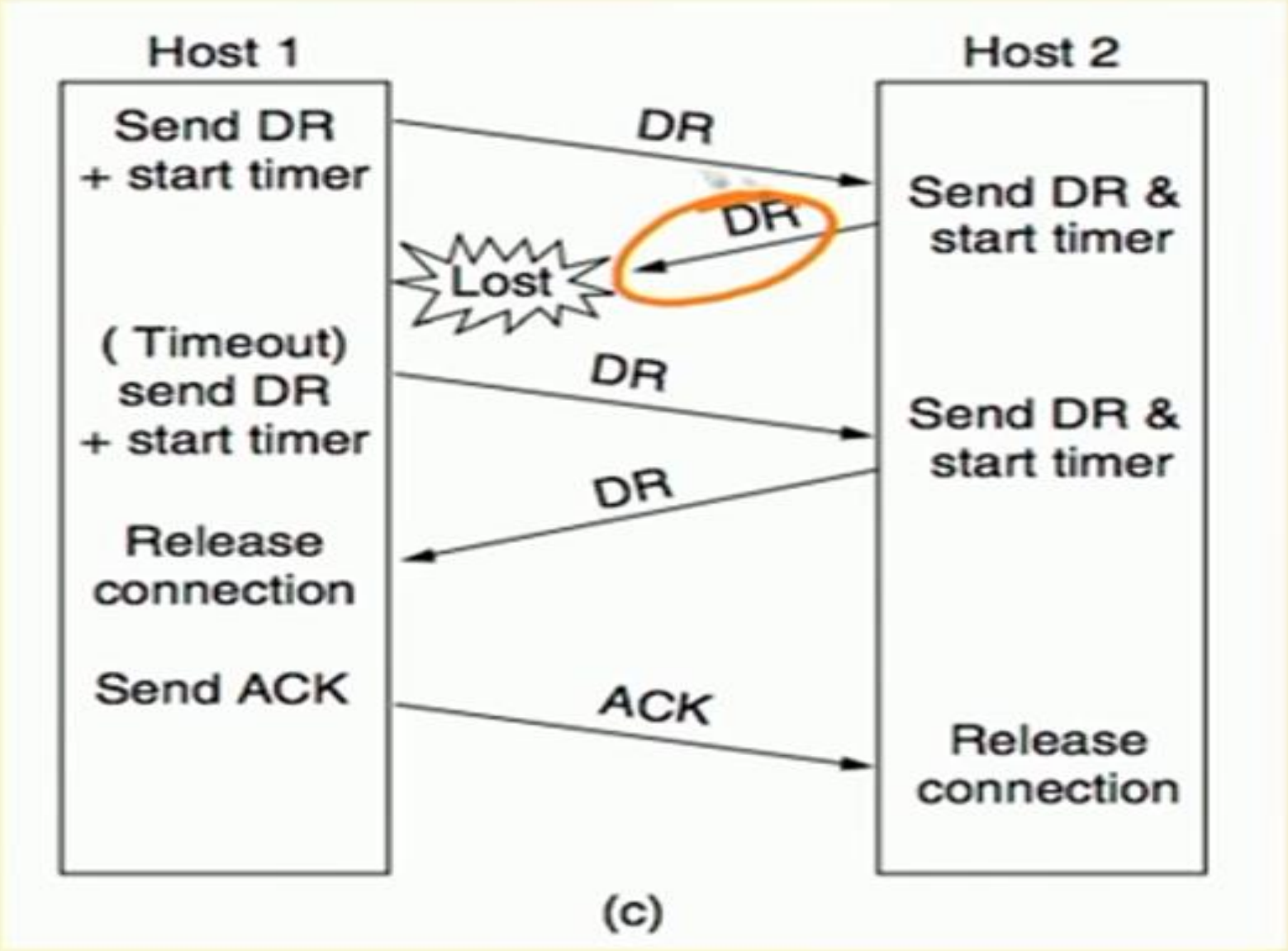
Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Connection Release – Final ACK Lost



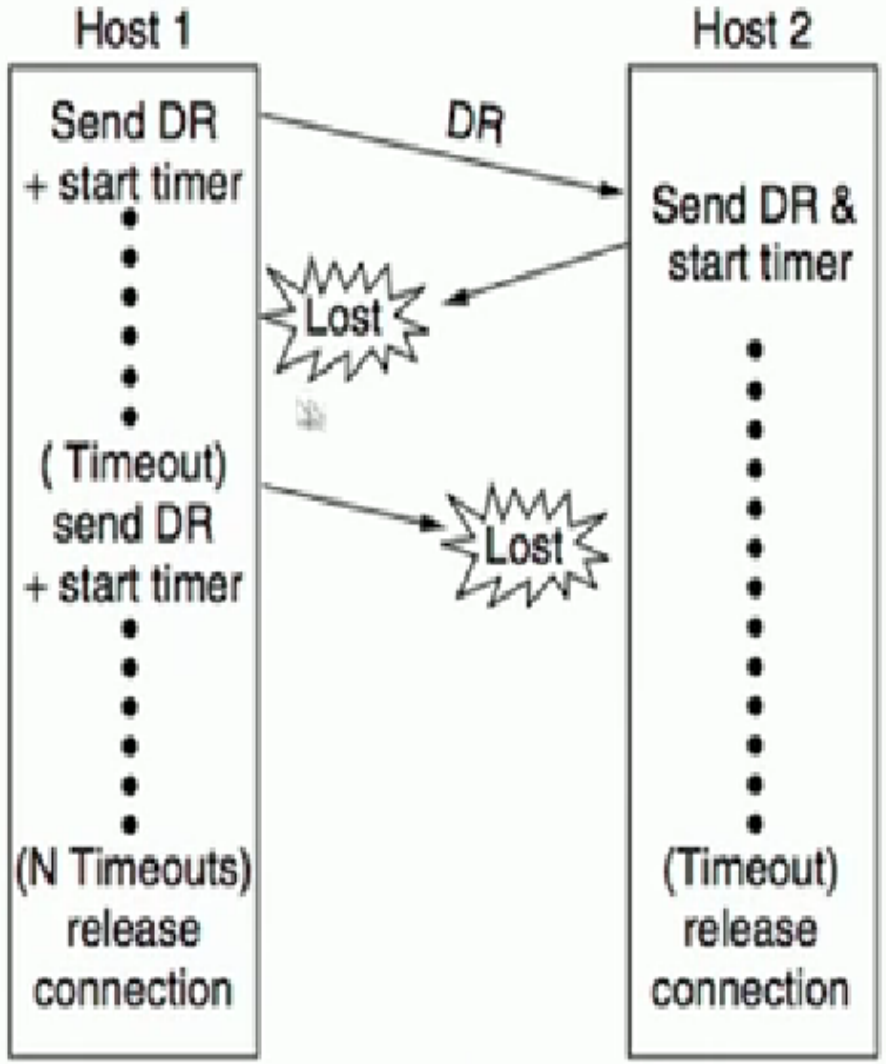
Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Connection Release – Response Lost



works (5th
n, Wetherell

Connection Release – Response Lost and Subsequent DRs Lost

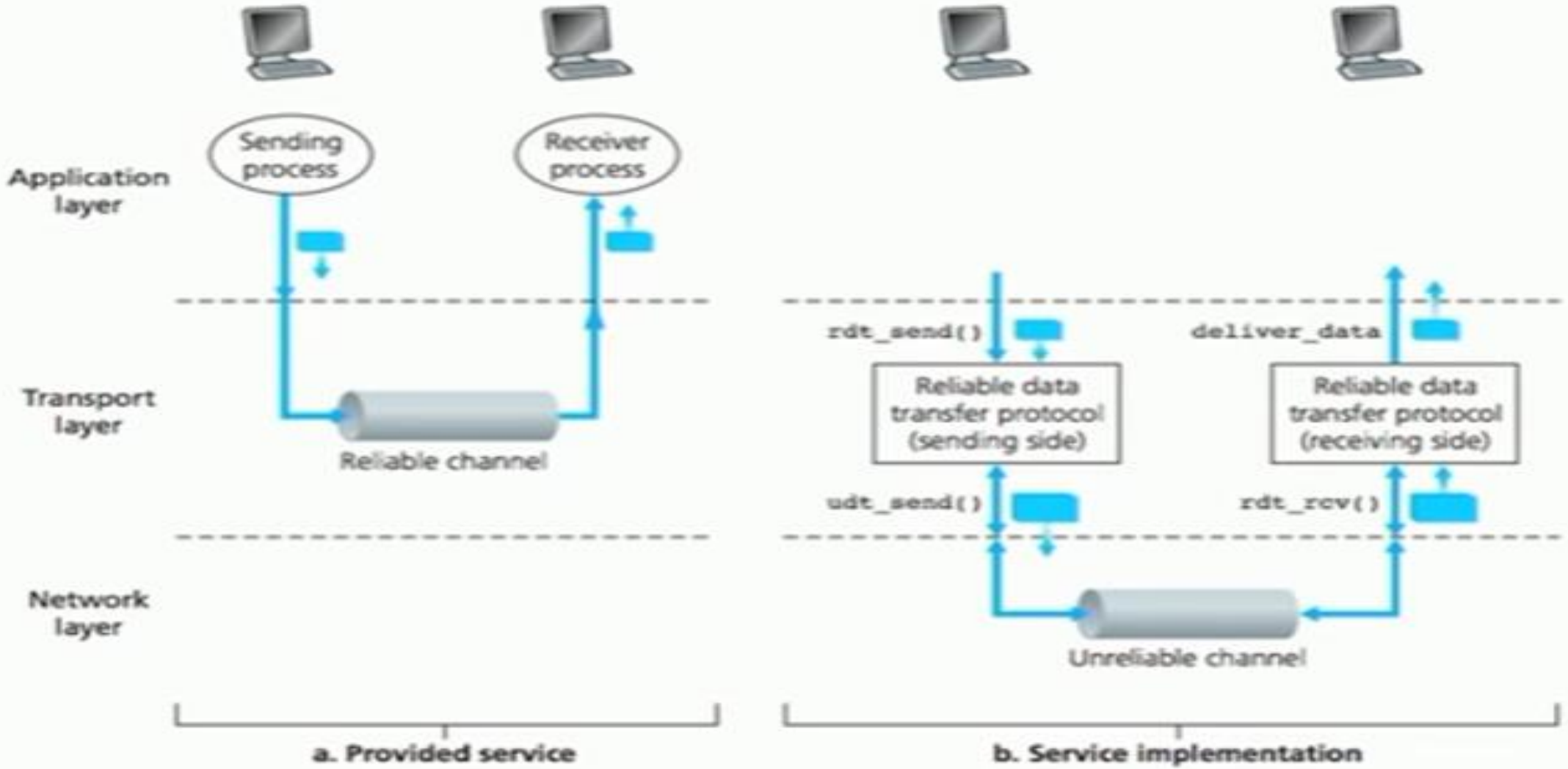


(d)

Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Flow Control and Reliability

Ensure Reliability at the Transport Layer



Source: Computer Networks, Kurose, Ross

Error Control and Flow Control

- These features are used in both Data Link Layer and Transport Layer – Why?
- Flow control and error control at the transport layer are essential
- Flow control and error control at the data link layer improve performance

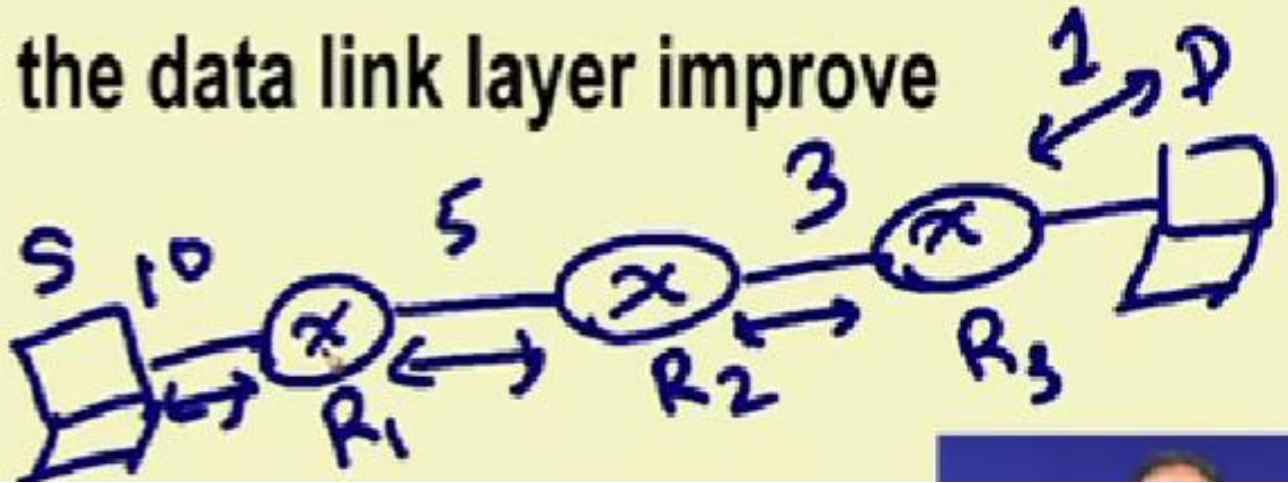
Error Control and Flow Control

- These features are used in both Data Link Layer and Transport Layer – Why?

Average rate is 1 MBPS

- Flow control and error control at the transport layer are essential

- Flow control and error control at the data link layer improve performance

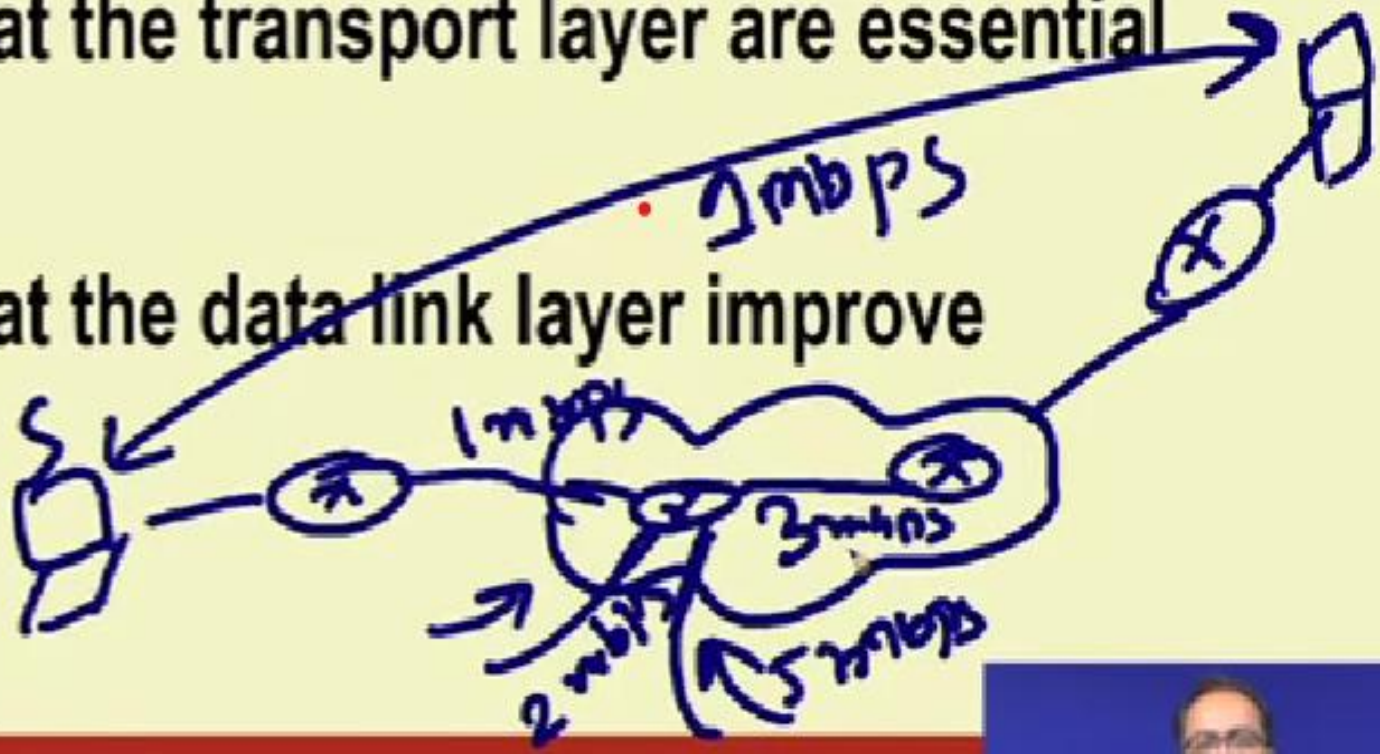


Error Control and Flow Control

- These features are used in both Data Link Layer and Transport Layer – Why?

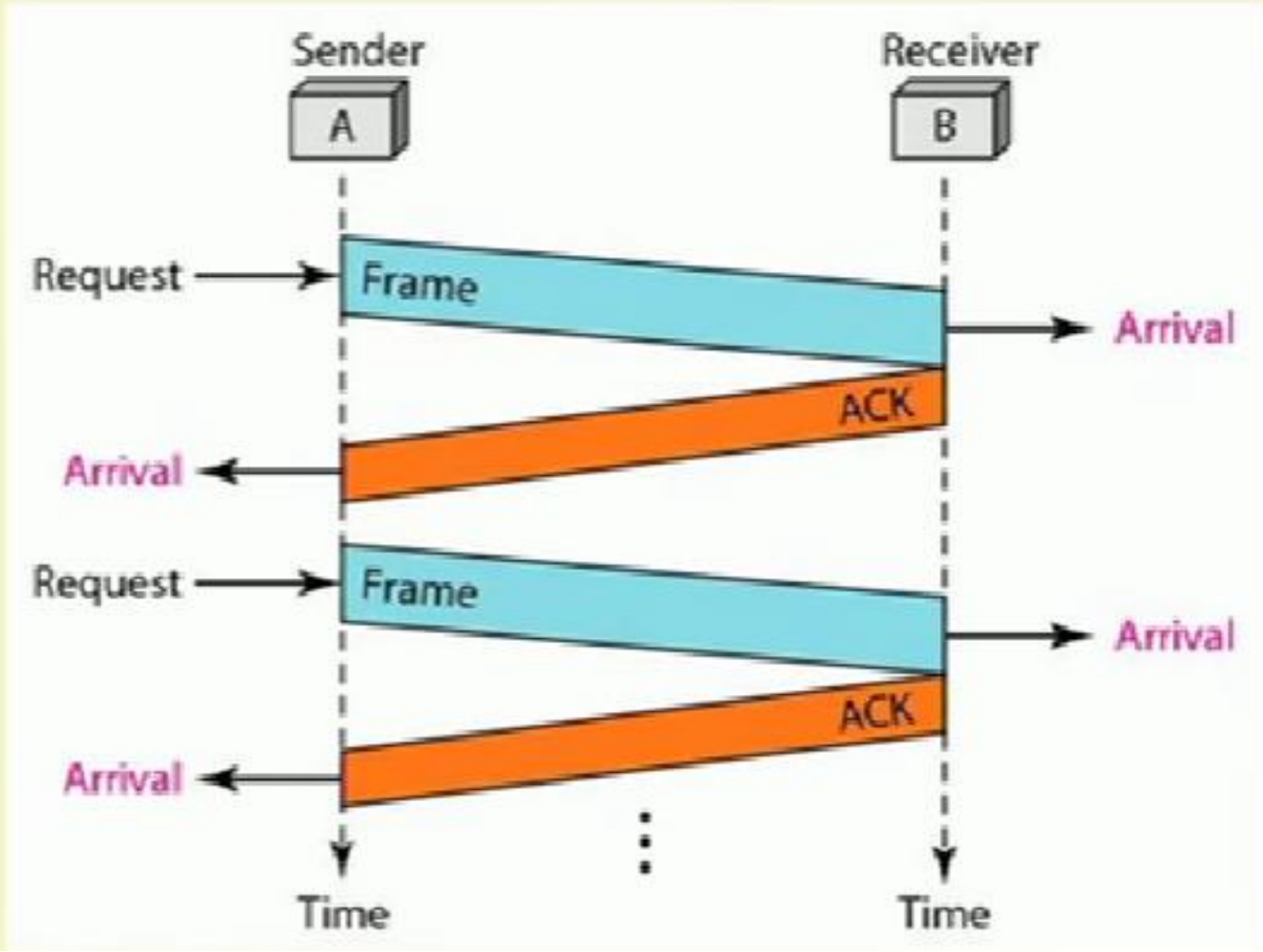
- Flow control and error control at the transport layer are essential

- Flow control and error control at the data link layer improve performance



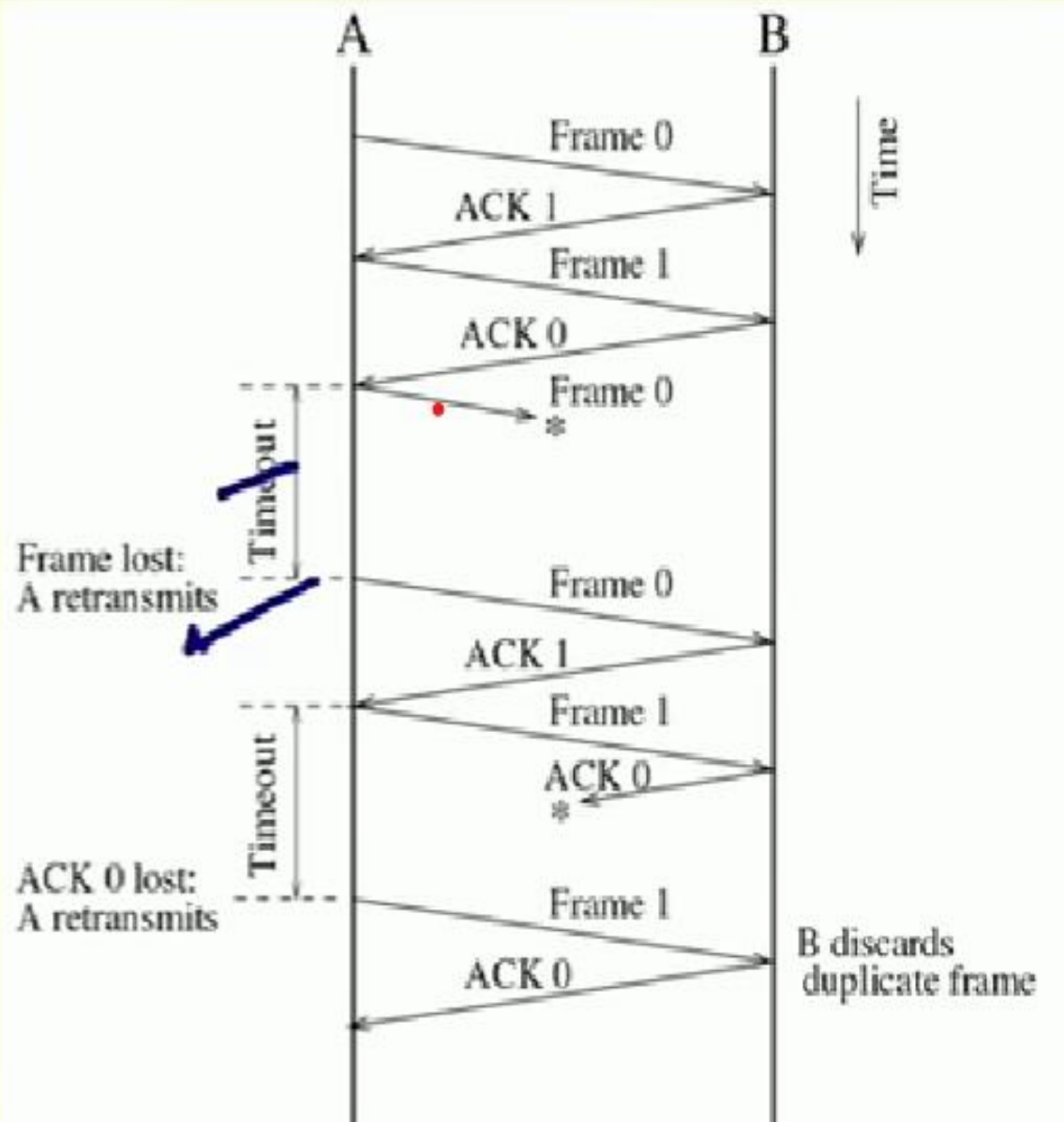
Flow Control Algorithms

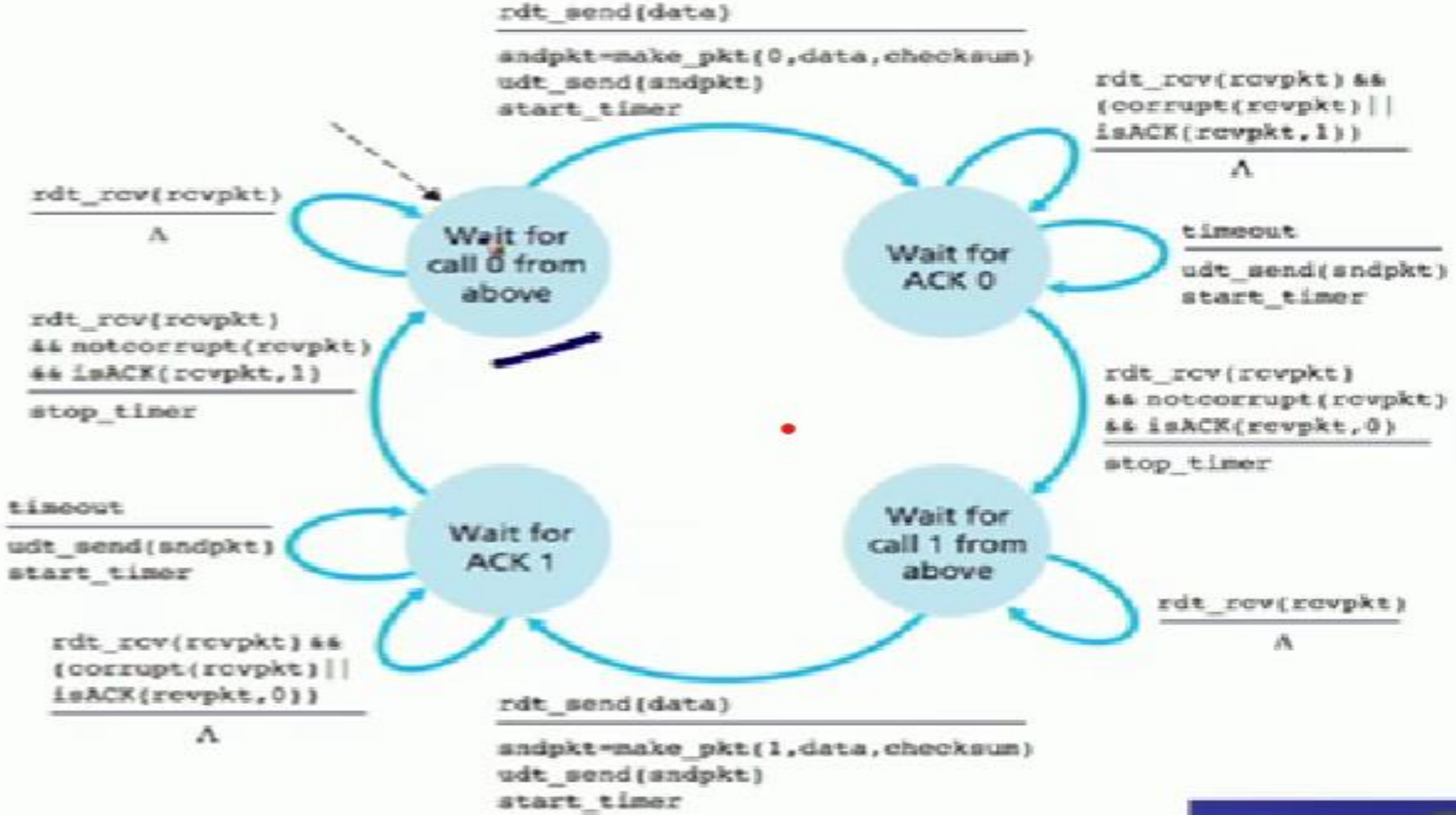
- **Stop and Wait Flow Control (Error Free Channel):**



Flow Control Algorithms

- **Stop and Wait (Noisy Channel):**
- Use sequence numbers to individually identify each frame and the corresponding acknowledgement
- **What can be a maximum size of the sequence number in Stop and Wait?**
- **Automatic Repeat Request (ARQ)**




rdt_send(data)
sndpkt=make_pkt(0,data,checksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt)||
isACK(rcvpkt,1))
Λ
rdt_rcv(rcvpkt)
Λ
Wait for call 0 from above
Wait for ACK 0
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& isACK(rcvpkt,1)
stop_timer
rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& isACK(rcvpkt,0)
stop_timer
timeout
udt_send(sndpkt)
start_timer
Wait for ACK 1
Wait for call 1 from above
rdt_rcv(rcvpkt)
Λ
rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt)||
isACK(rcvpkt,0))
Λ
rdt_send(data)
sndpkt=make_pkt(1,data,checksum)
udt_send(sndpkt)
start_timer

Problem with Stop and Wait

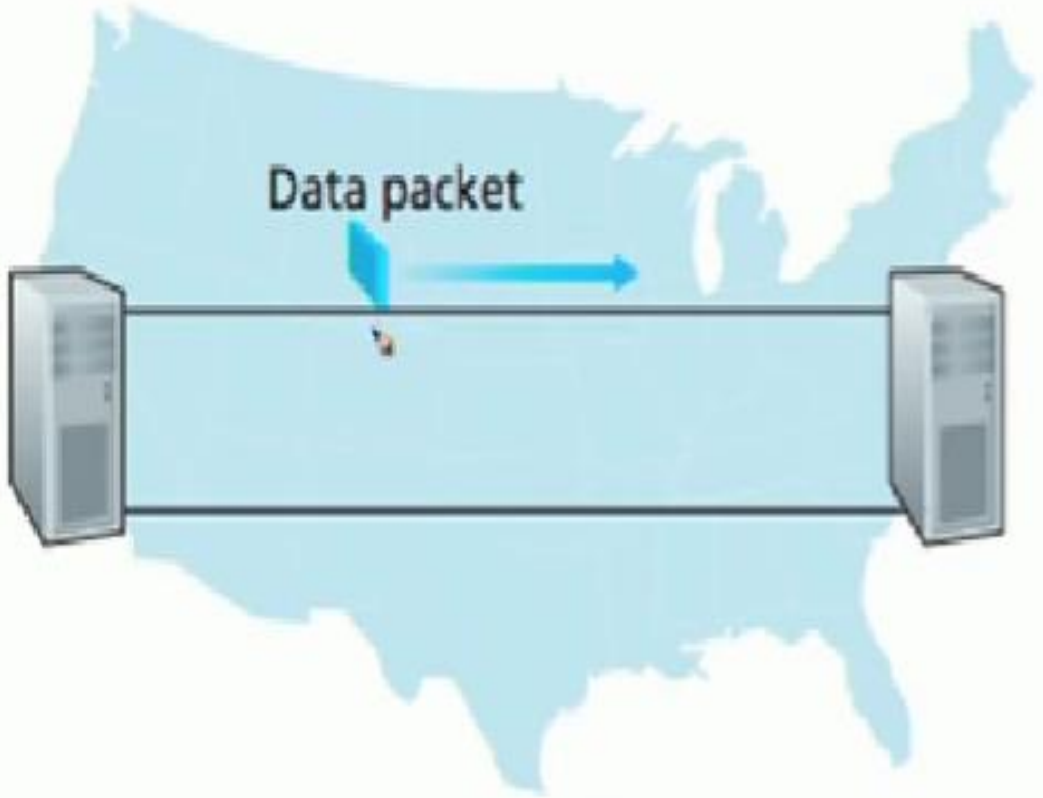
- Every packet needs to wait for the acknowledgement of the previous packet.
- For bidirectional connections – use two instances of the stop and wait protocol at both directions – further waste of resources
- A possible solution: Piggyback data and acknowledgement from both the directions
- Reduce resource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

Problem with Stop and Wait

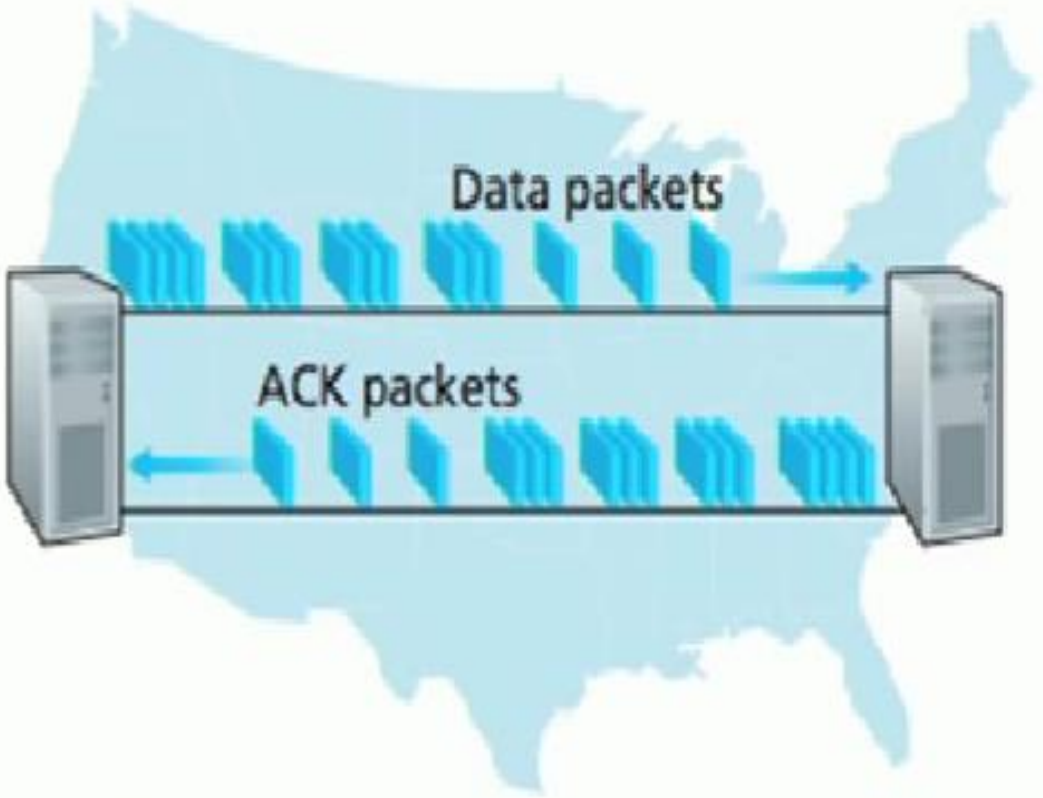
- Every packet needs to wait for the acknowledgement of the previous packet.
- For bidirectional connections – use two instances of the stop and wait protocol at both directions – further waste of resources
- A possible solution: Piggyback data and acknowledgement from both the directions

- 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

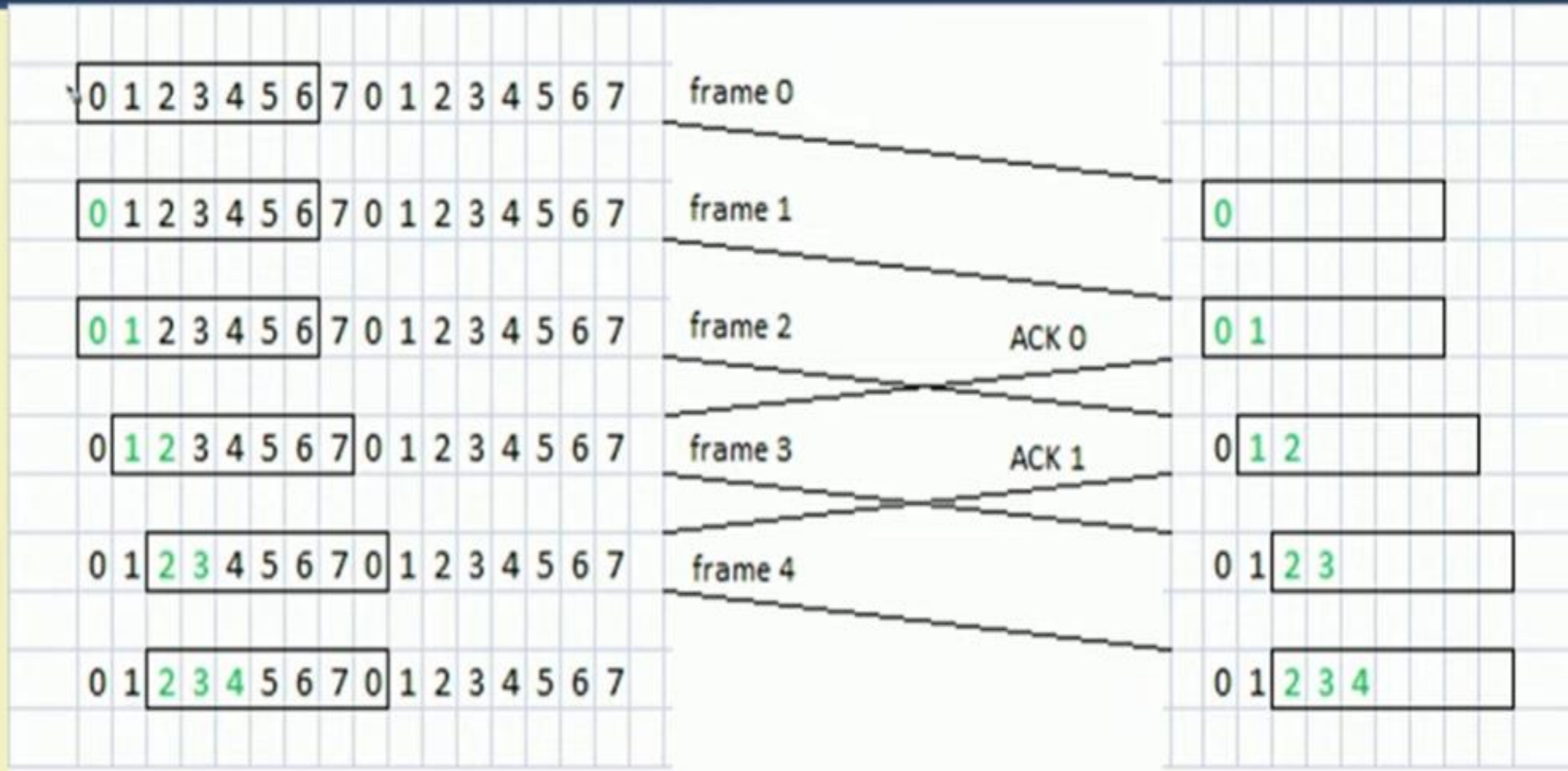
Sliding Window Protocols

- 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



ource:
ttp://ironbark.xtelco.com.au/subjects/DC/
ctures/13/

Sliding window Protocol

Sliding Window for a 3 bit Sequence Number

Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

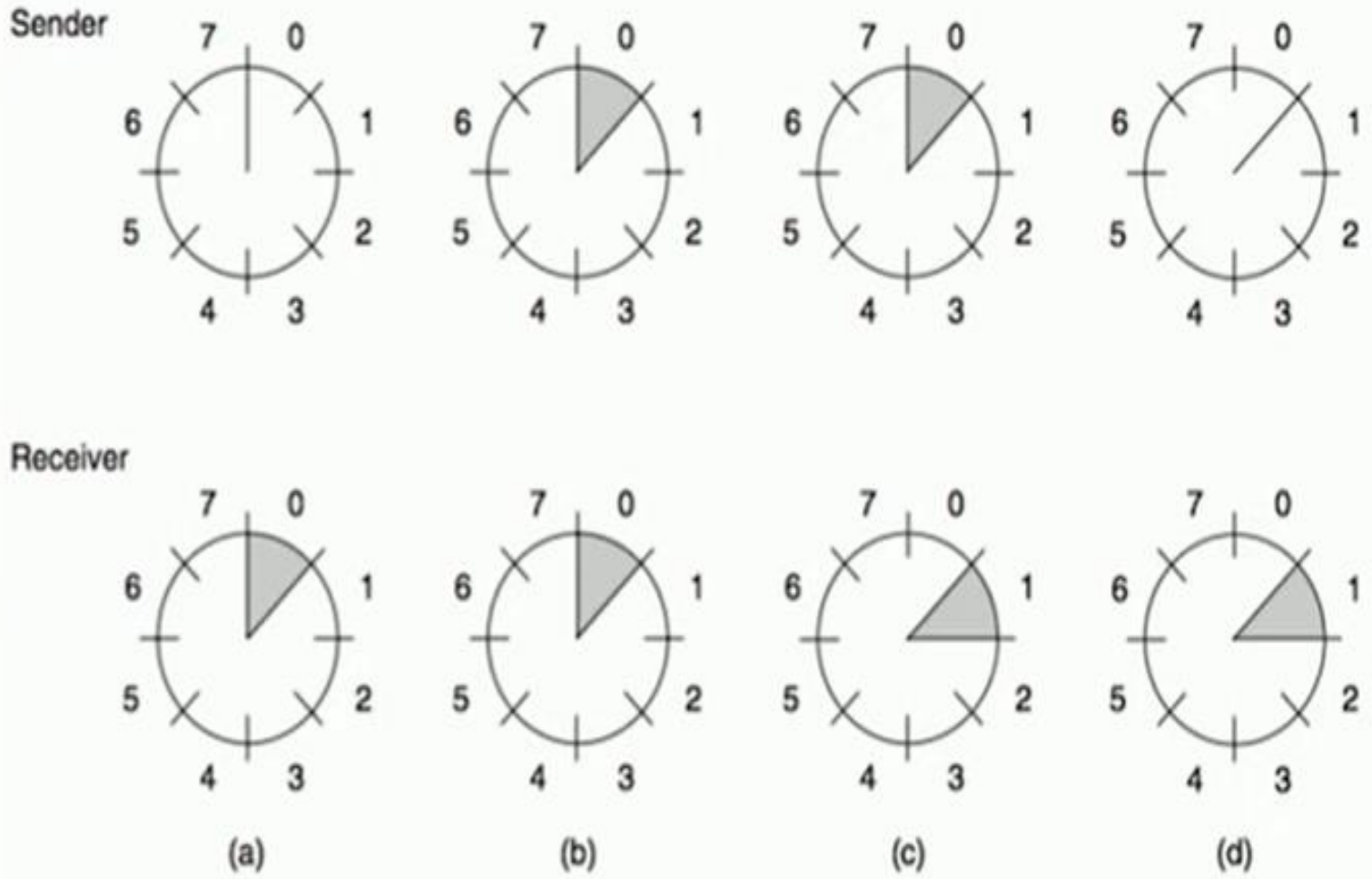
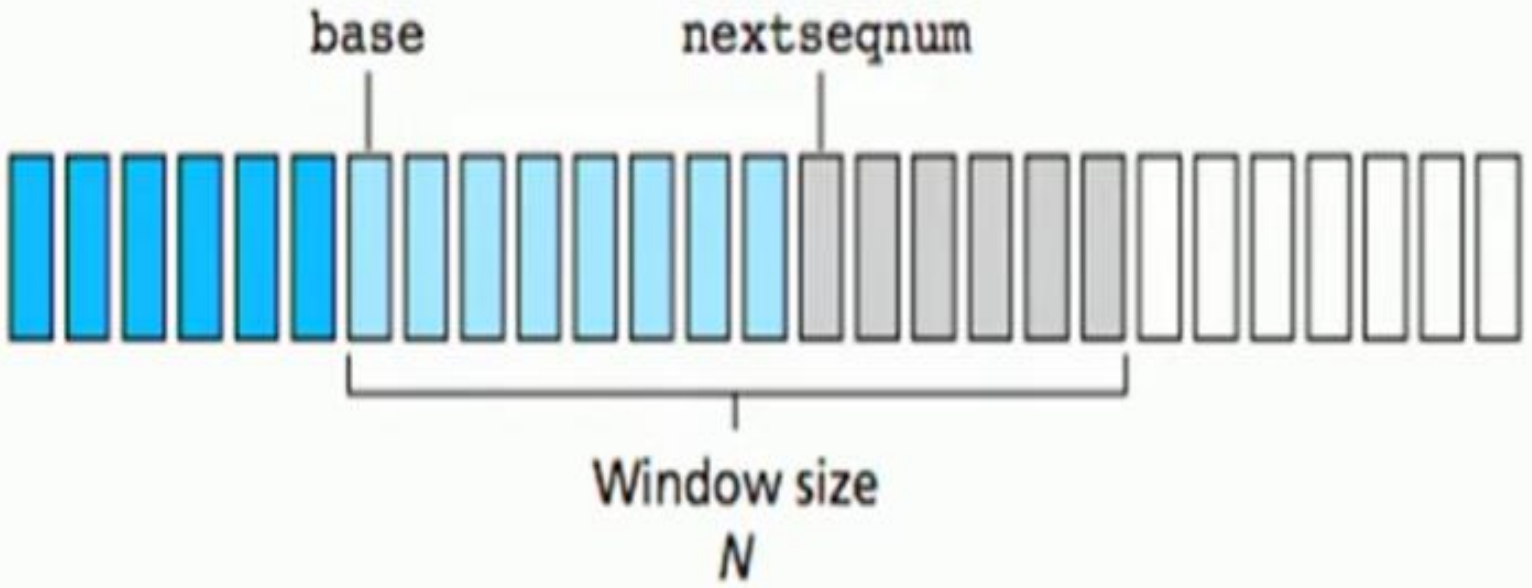


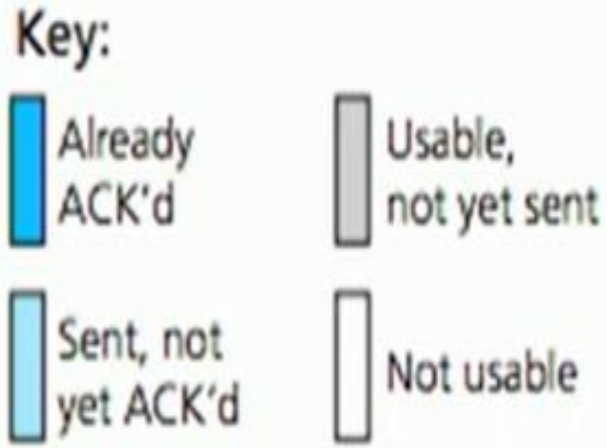
Figure 3-15. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

Sliding Window Protocols in Noisy Channels

- **A timeout occurs if a segment (or the acknowledgment) gets lost**
- **How does the flow and error control protocol handle a timeout?**
- **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)

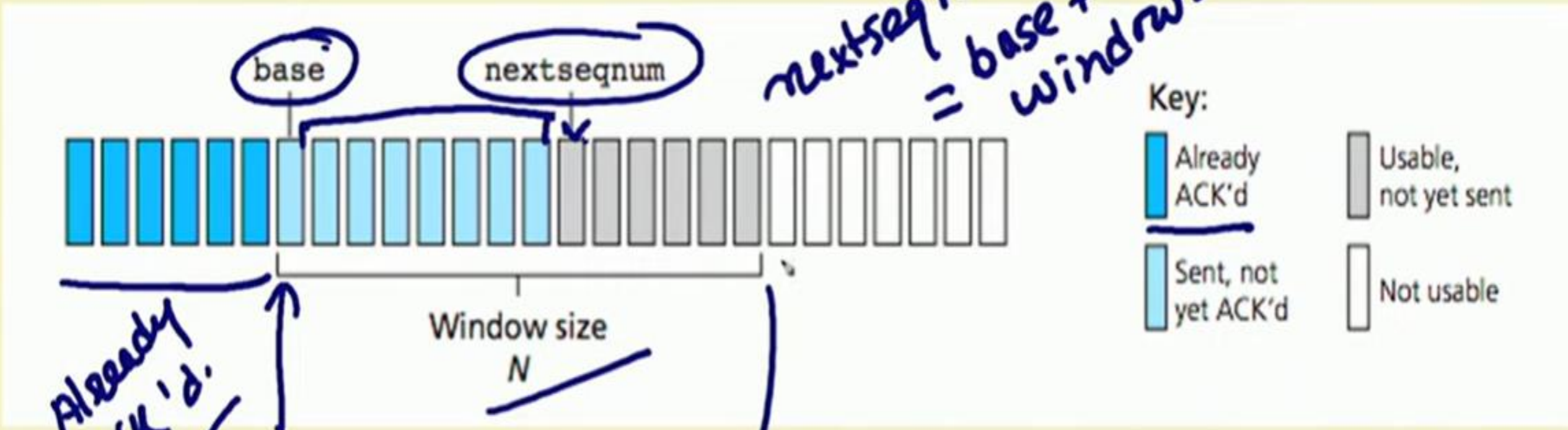
Go Back N ARQ – Sender Window Control





Source: Computer Networks, Kurose, Ross

Go Back N ARQ – Sender Window Control



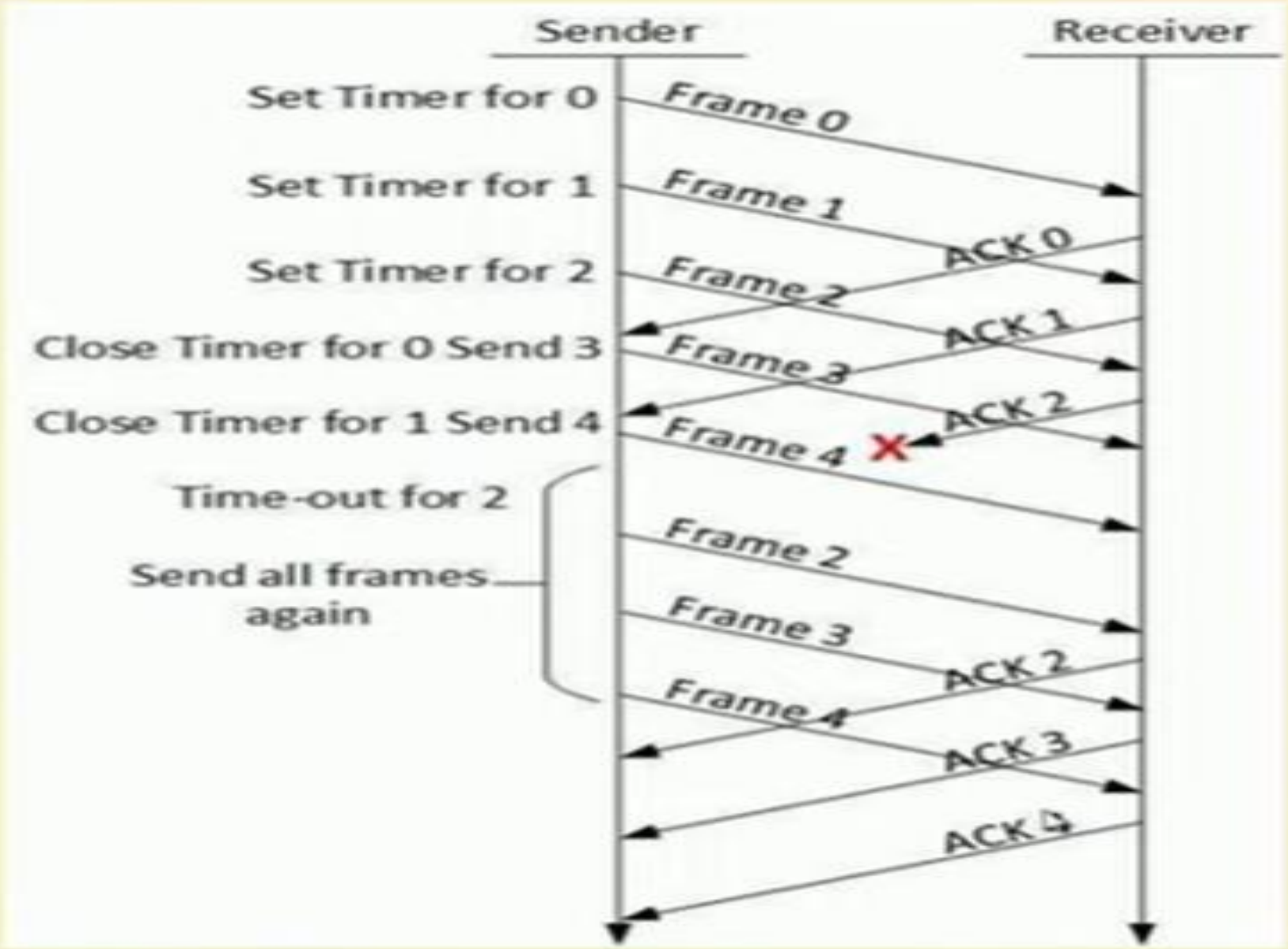
nextseqnum = base + windowSize + 1

Already ACK'd.

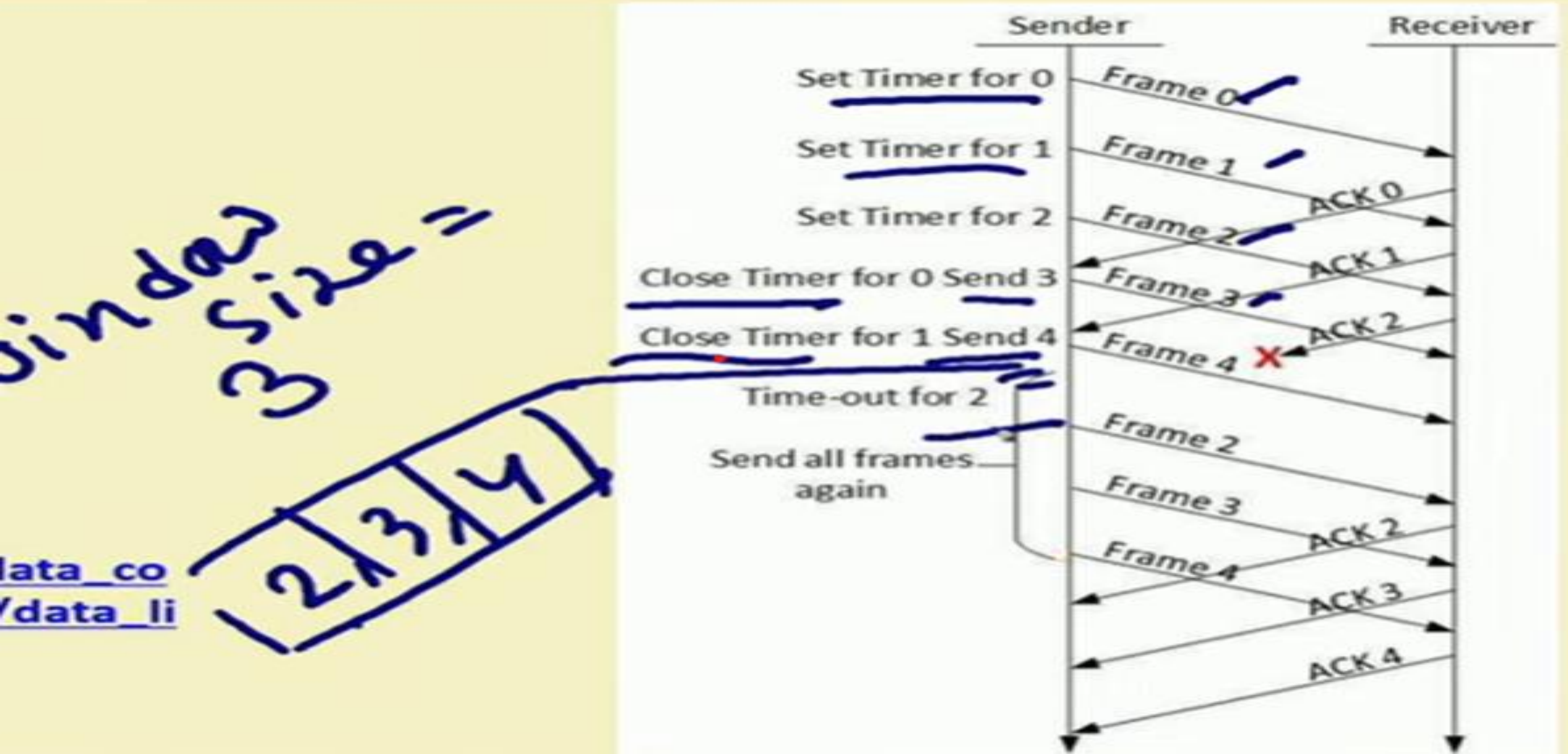
Current window

Source: Computer Networks, Kurose, Ross

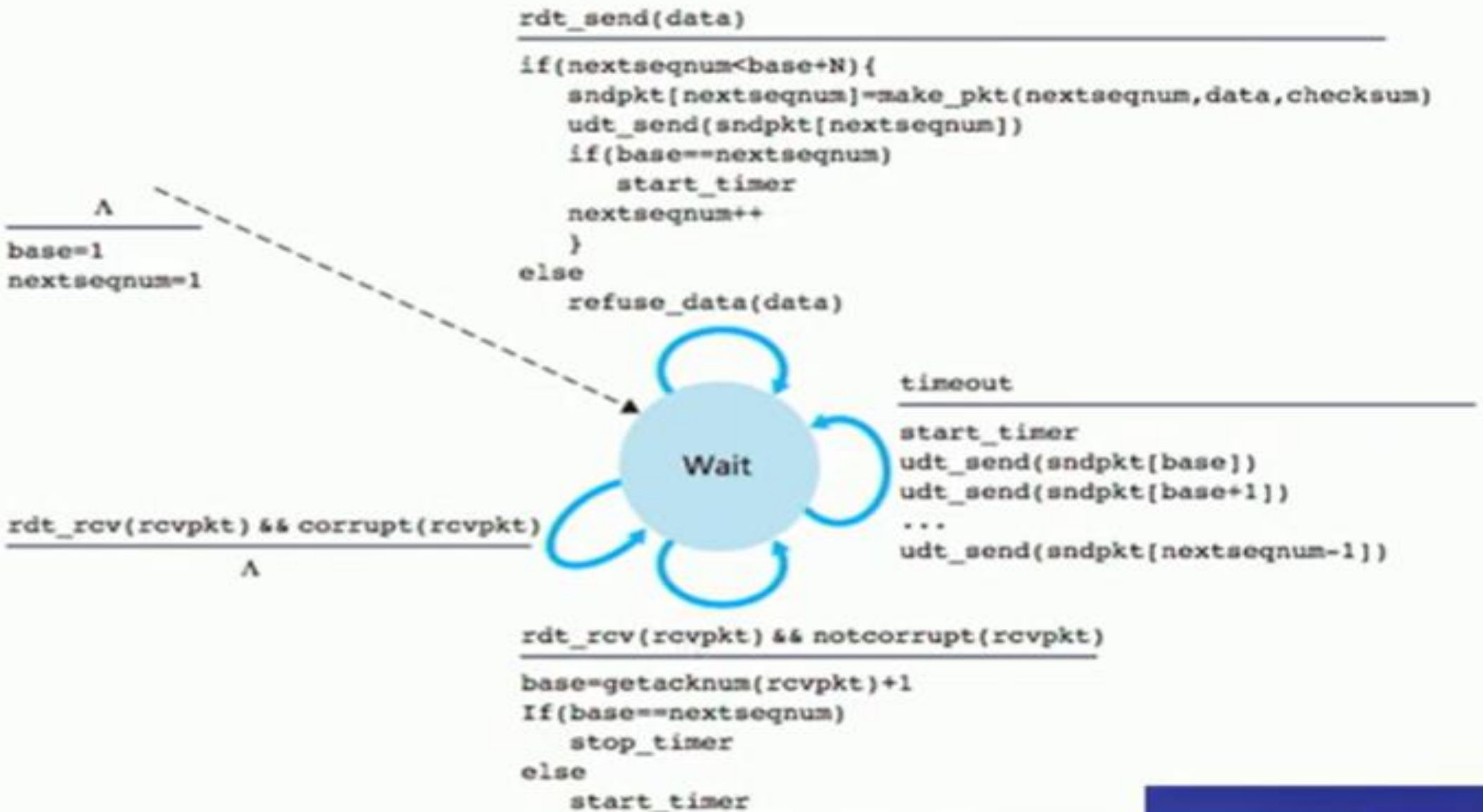
Go Back N ARQ


Sender
Receiver
Set Timer for 0
Frame 0
Set Timer for 1
Frame 1
ACK 0
Set Timer for 2
Frame 2
ACK 1
Close Timer for 0 Send 3
Frame 3
ACK 2
Close Timer for 1 Send 4
Frame 4
X
Time-out for 2
Send all frames again
Frame 2
Frame 3
ACK 2
Frame 4
ACK 3
ACK 4

Go Back N ARQ


Sender
Receiver
Set Timer for 0
Frame 0
Set Timer for 1
Frame 1
ACK 0
Set Timer for 2
Frame 2
ACK 1
Close Timer for 0 Send 3
Frame 3
ACK 2
Close Timer for 1 Send 4
Frame 4
Time-out for 2
Send all frames again
Frame 2
Frame 3
ACK 2
Frame 4
ACK 3
ACK 4

Go Back N ARQ – Sender



Go Back N ARQ – Receiver

```
rdt_rcv(rcvpkt)
   && notcorrupt(rcvpkt)
   && hasseqnum(rcvpkt,expectedseqnum)
```

```
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(expectedseqnum,ACK,checksum)
udt_send(sndpkt)
expectedseqnum++
```

default

```
udt_send(sndpkt)
```

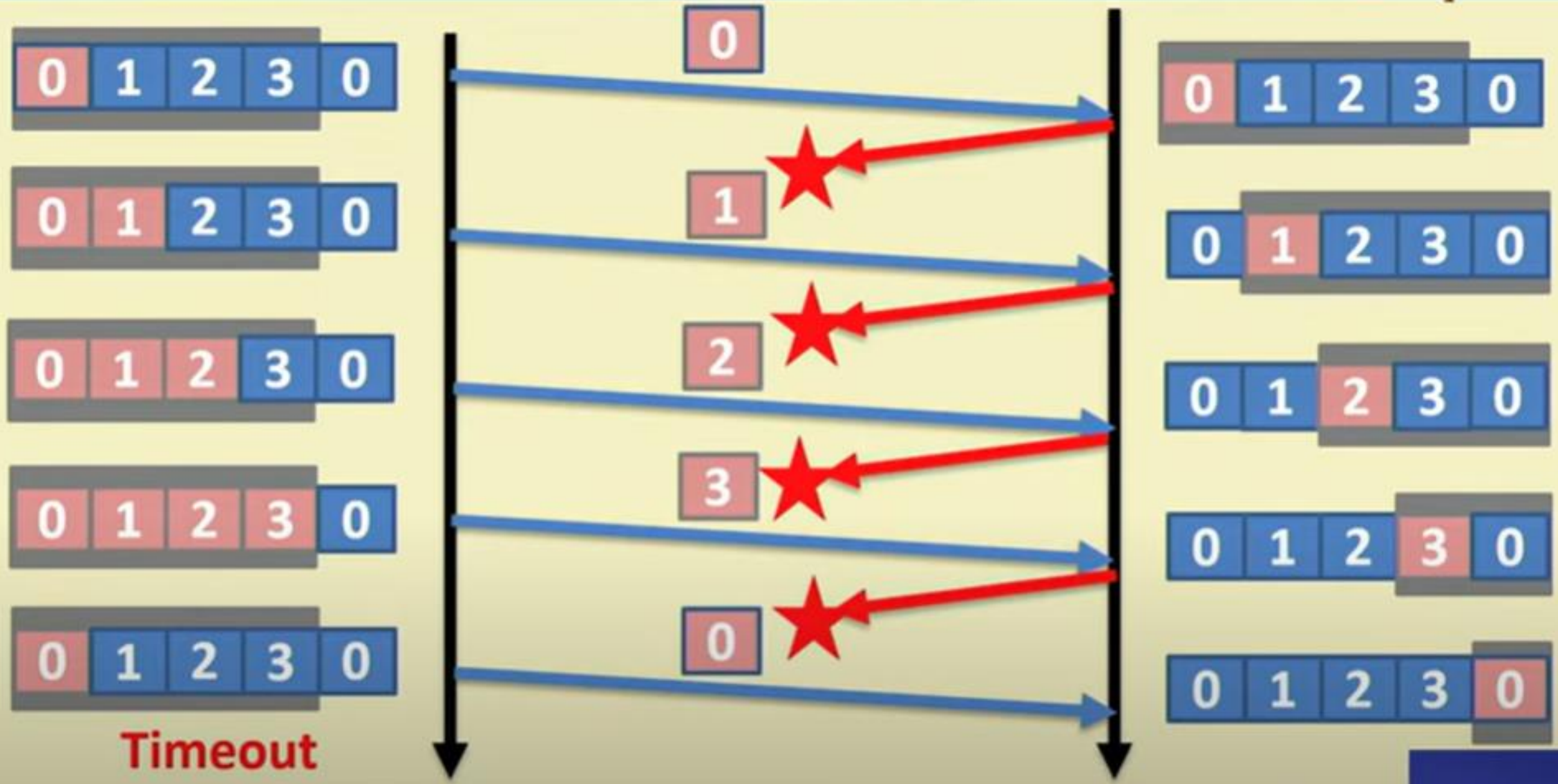
Λ

```
expectedseqnum=1
sndpkt=make_pkt(0,ACK,checksum)
```

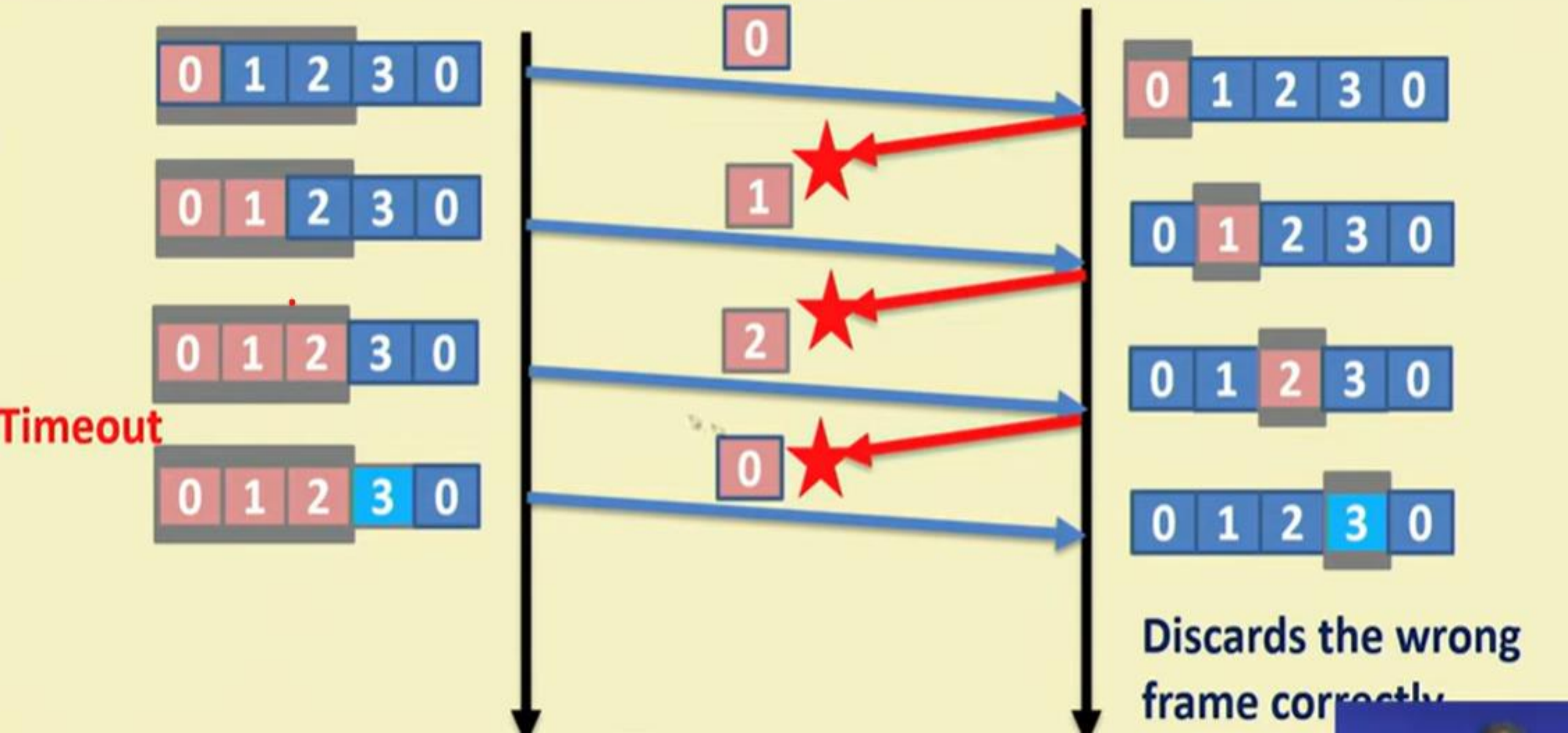
Go Back N ARQ – A Bound on Window Size

- **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)**

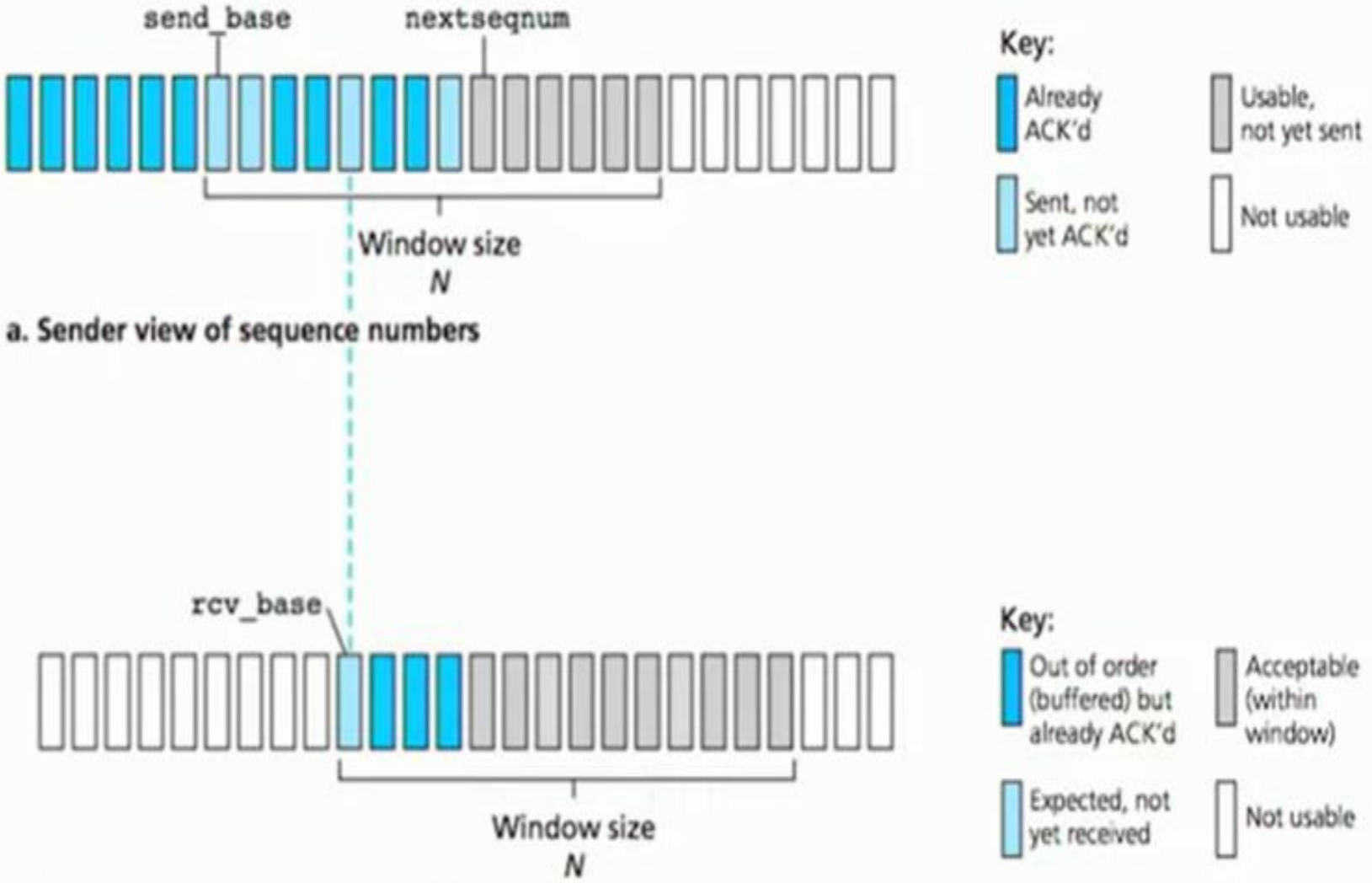
Go Back N ARQ (MAX_SEQ = 3, Window Size = 4)



Go Back N ARQ (MAX_SEQ = 3, Window Size = 3)

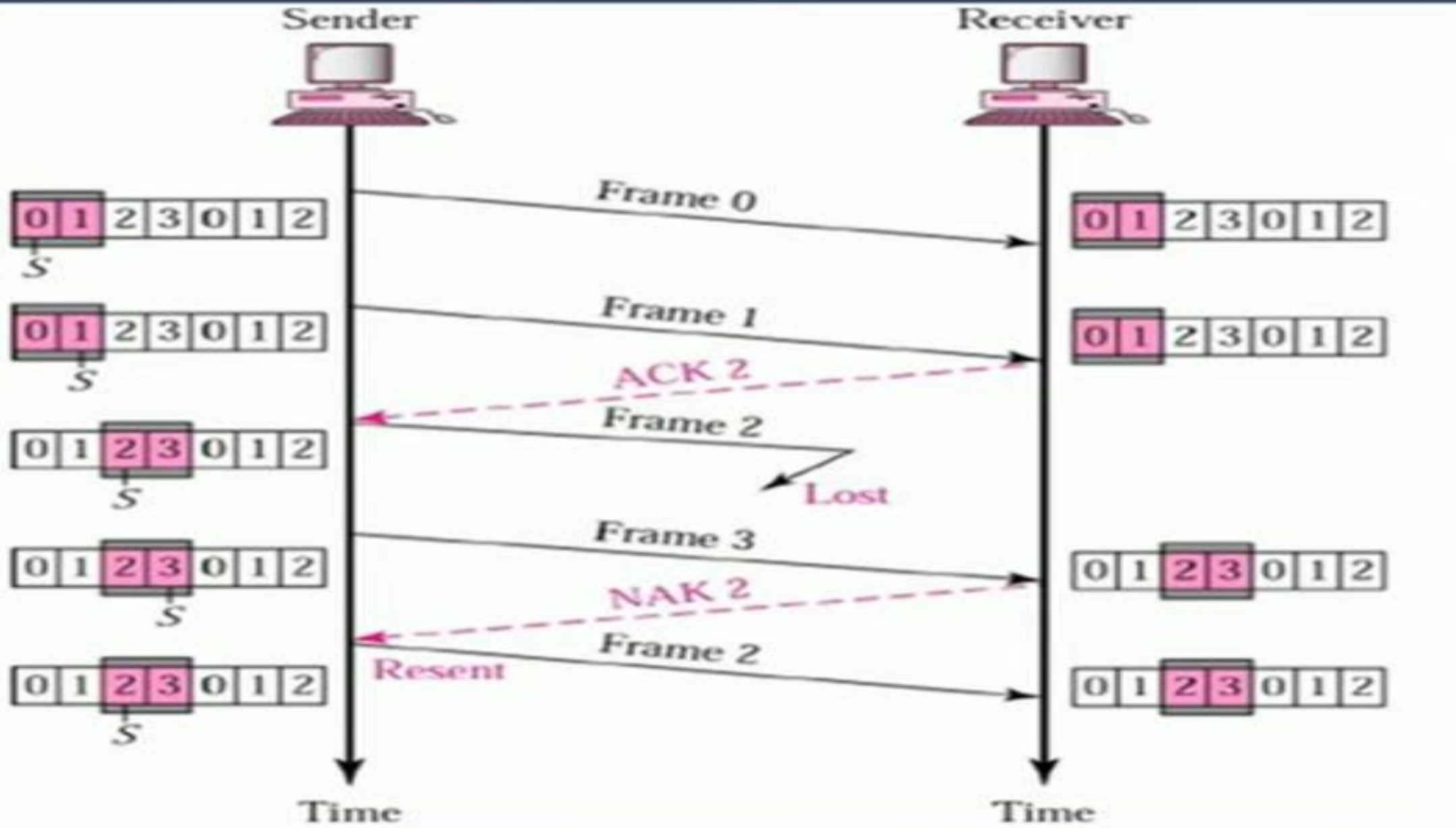


Selective Repeat (SR) – Window Control



Source: Computer Networks, Kurose, Ross

Selective Repeat ARQ



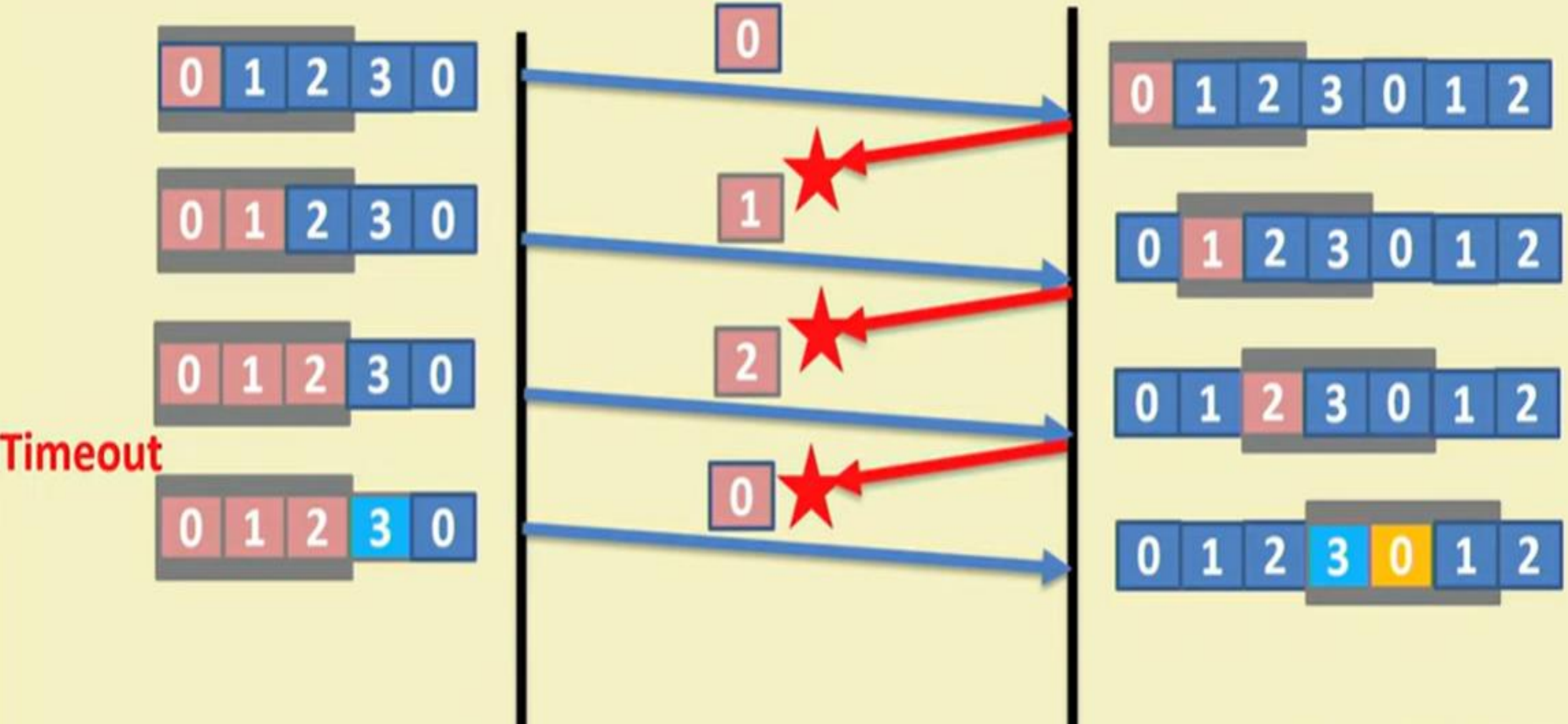
Selective Repeat – A Bound on Window Size

- **Maximum Sequence Number (MAX_SEQ): MAX_SEQ+1** distinct sequence numbers are there
  - **0,1,…,MAX_SEQ**

- **Maximum Number of Outstanding Frames (=Window Size): (MAX_SEQ+1)/2**

- **Example:** Sequence Numbers (0,1,2,…,7) – 3 bit sequence numbers, number of outstanding frames (window size) = 4

Selective Repeat (MAX_SEQ = 3, Window Size = 3)



Principle of Congestion Control

TCP Congestion Control

Principles of congestion control

Congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- manifestations:
  - long delays (queueing in router buffers)
  - packet loss (buffer overflow at routers)

congestion control:

too many senders,
sending too fast

Principles of congestion control

Congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- manifestations:
  - long delays (queueing in router buffers)
  - packet loss (buffer overflow at routers)
- different from flow control!
- a top-10 problem!

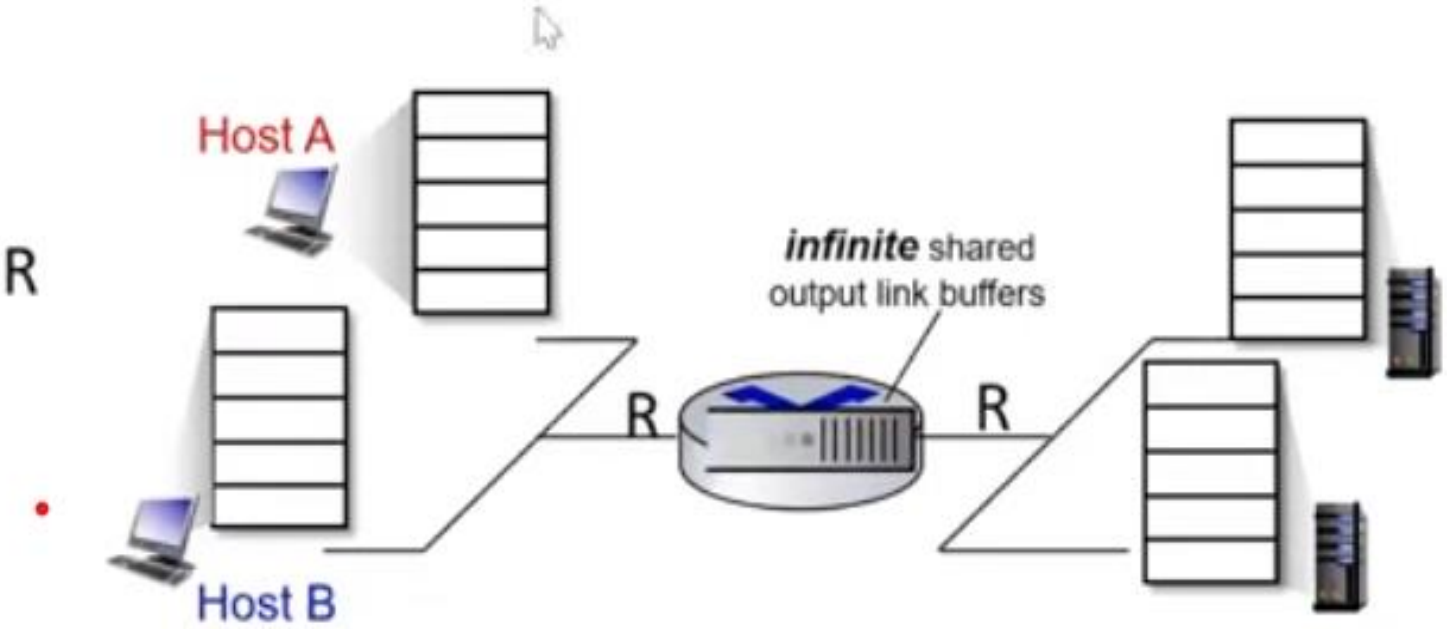
congestion control:
too many senders,
sending too fast

flow control: one sender
too fast for one receiver

Causes/costs of congestion: scenario 1

Simplest scenario:

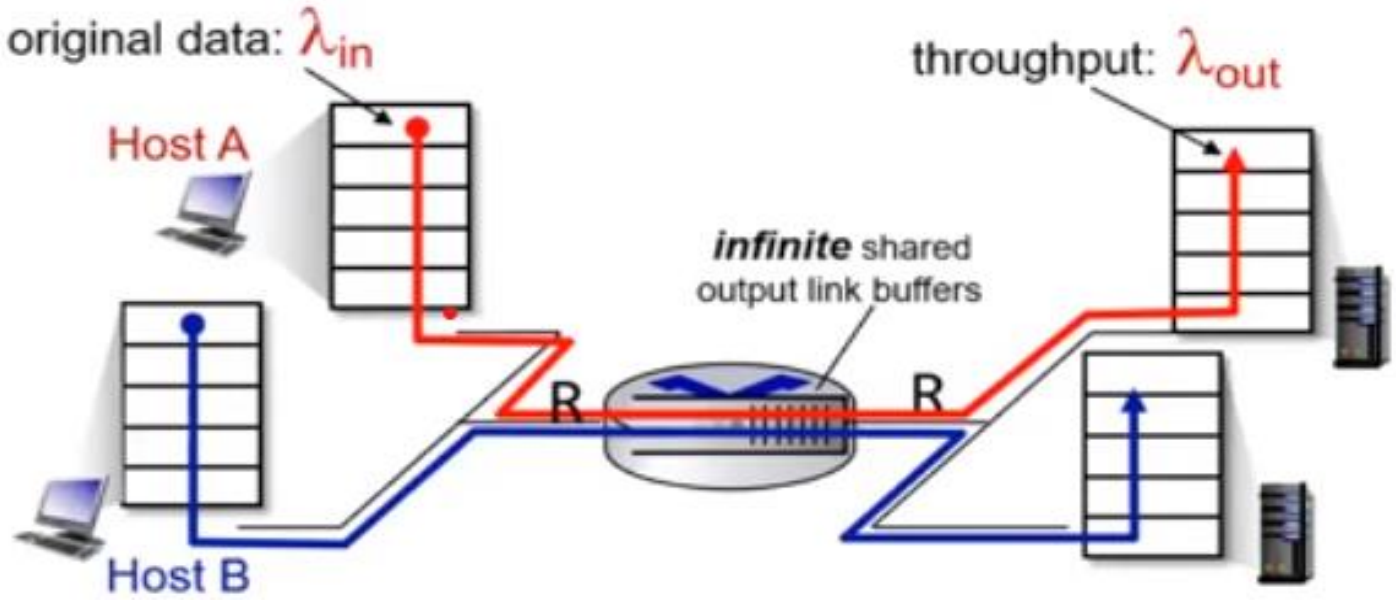
- one router, infinite buffers
- input, output link capacity: R



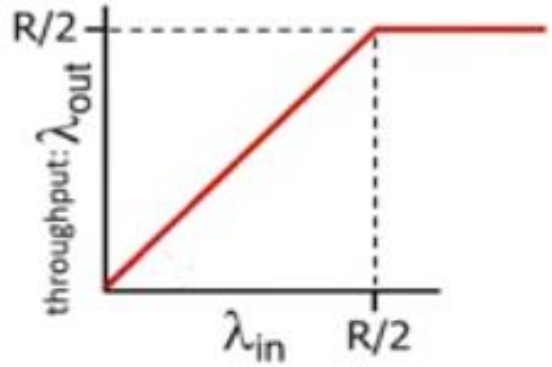
Causes/costs of congestion: scenario 1

Simplest scenario:

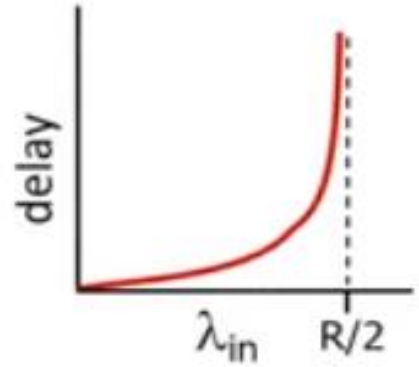
- one router, infinite buffers
- input, output link capacity: R
- two flows
- no retransmissions needed



Q: What happens as arrival rate λ_{in} approaches R/2?



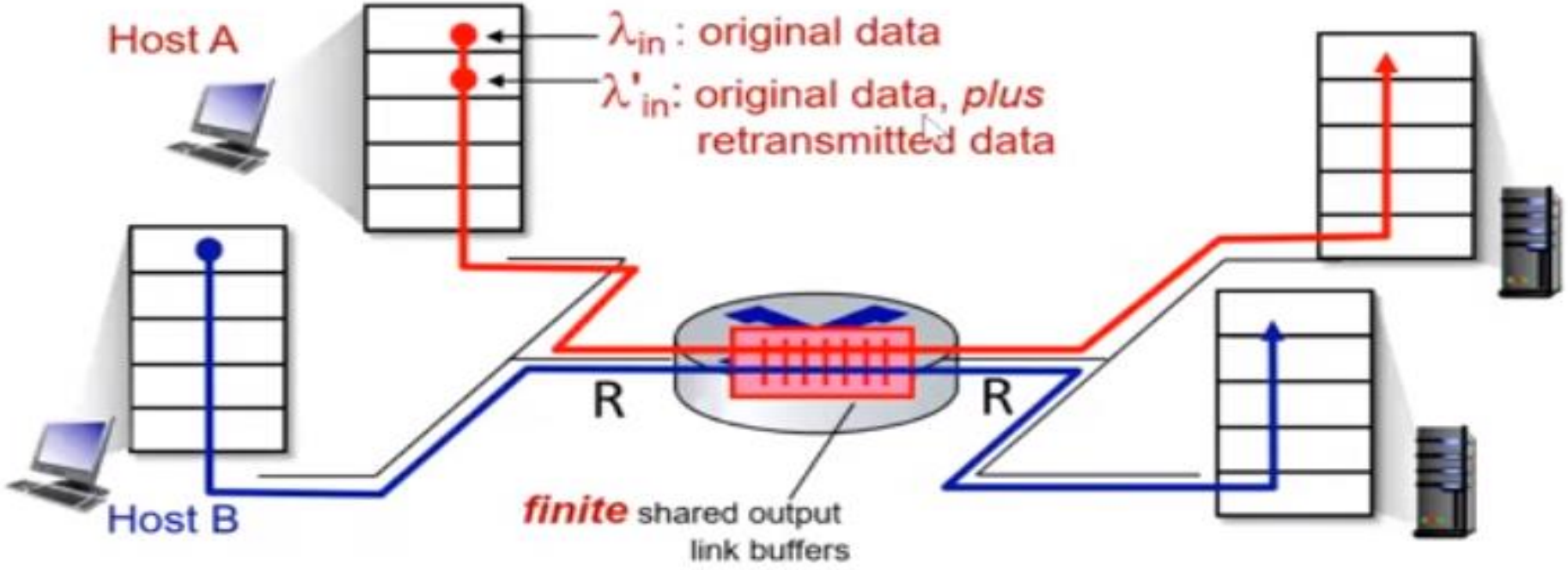
maximum per-connection throughput: R/2



large delays as arrival rate λ_{in} approaches capacity

Causes/costs of congestion: scenario 2

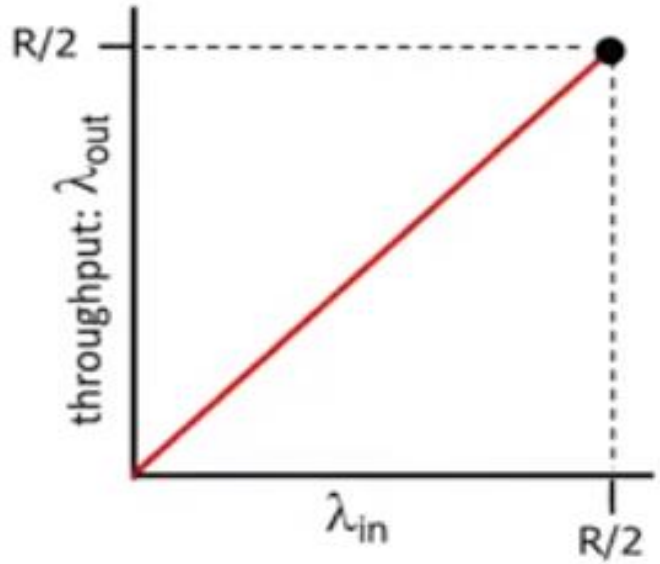
- one router, *finite* buffers
- sender retransmits lost, timed-out packet
  - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes *retransmissions* : $\lambda'_{in} \geq \lambda_{in}$

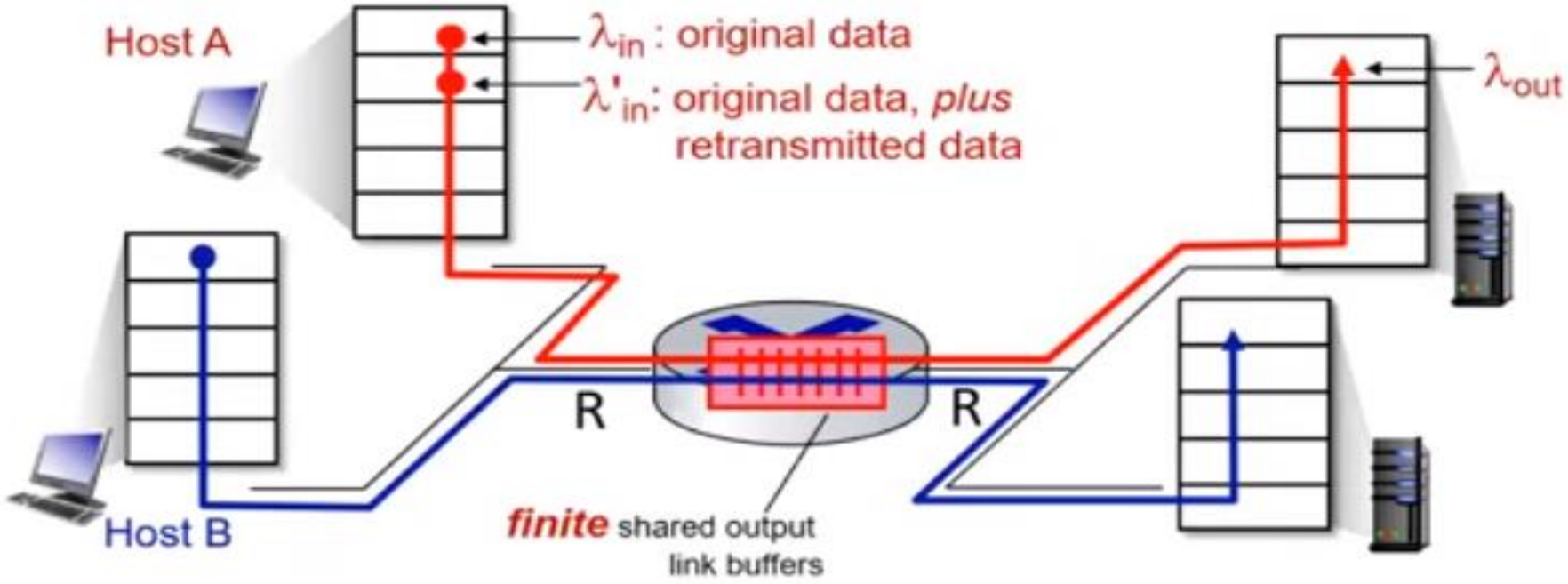


Causes/costs of congestion: scenario 2

Idealization: perfect knowledge

- sender sends only when router buffers available

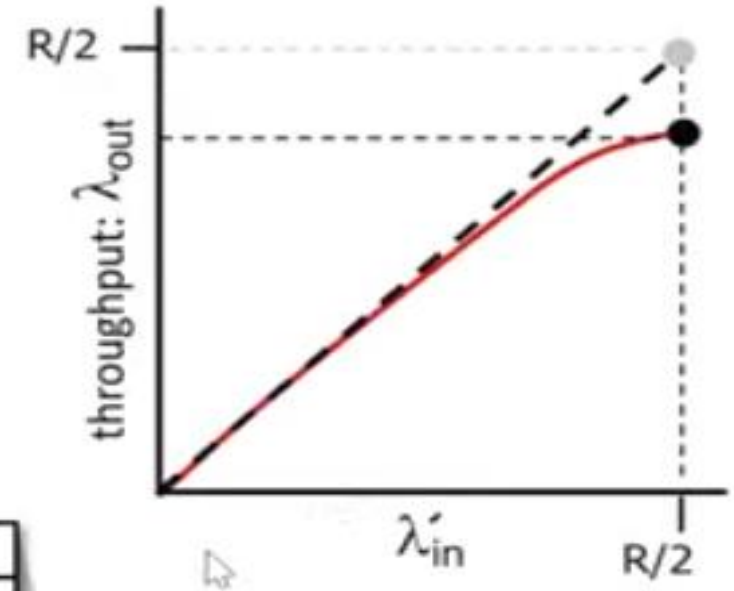


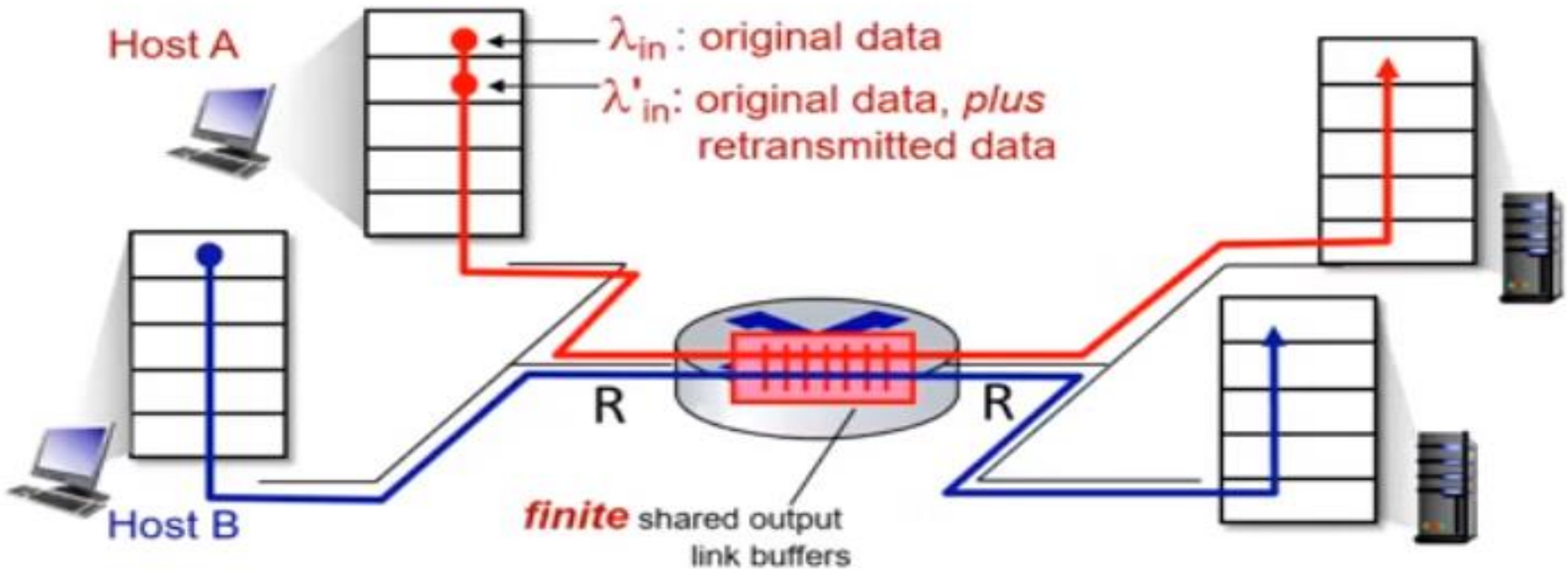


Causes/costs of congestion: scenario 2

Idealization: *some* perfect knowledge

- packets can be lost (dropped at router) due to full buffers
- sender knows when packet has been dropped: only resends if packet *known* to be lost

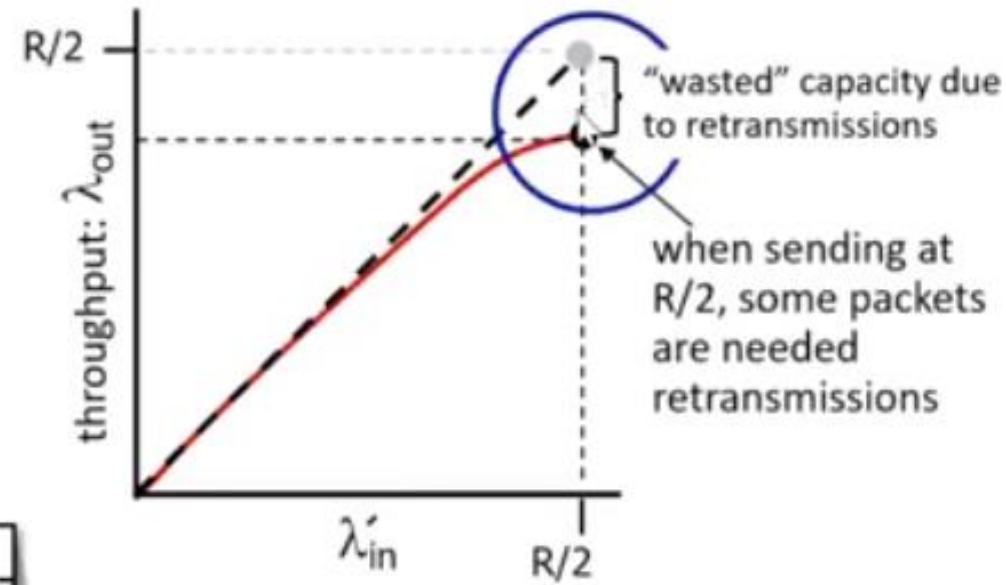


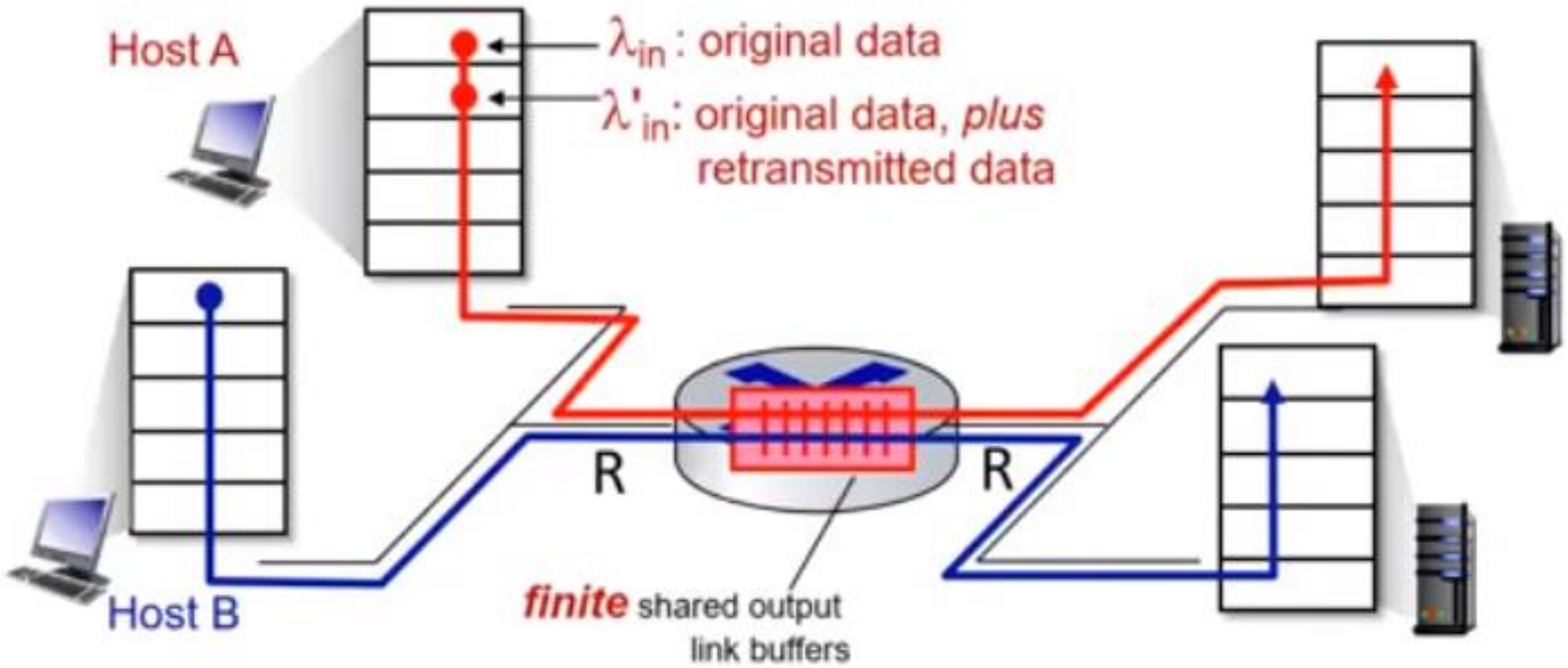


Causes/costs of congestion: scenario 2

Idealization: *some* perfect knowledge

- packets can be lost (dropped at router) due to full buffers
- sender knows when packet has been dropped: only resends if packet *known* to be lost

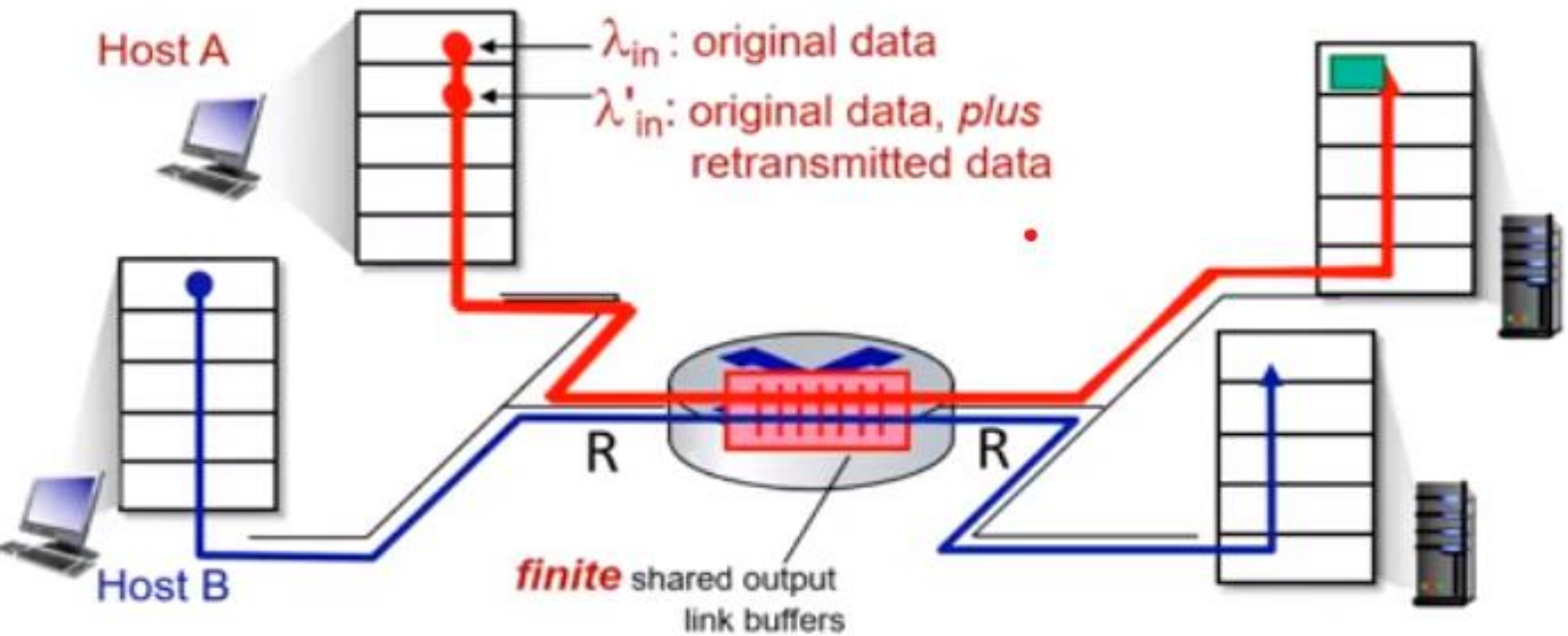


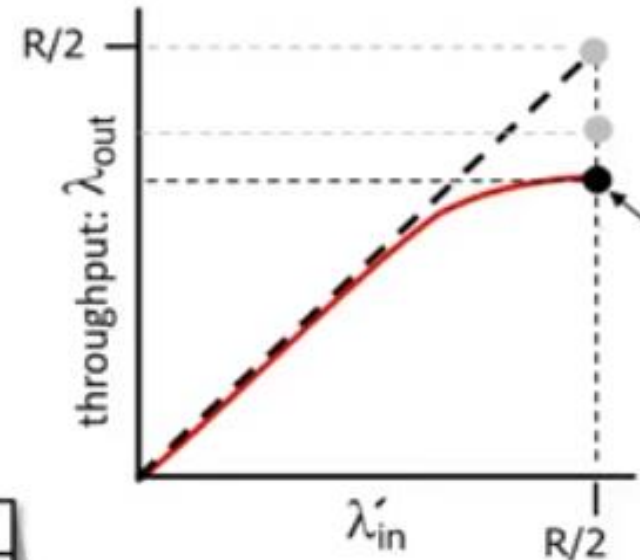


Causes/costs of congestion: scenario 2

Realistic scenario: *un-needed duplicates*

- packets can be lost, dropped at router due to full buffers – requiring retransmissions
- but sender times can time out prematurely, sending *two* copies, *both* of which are delivered



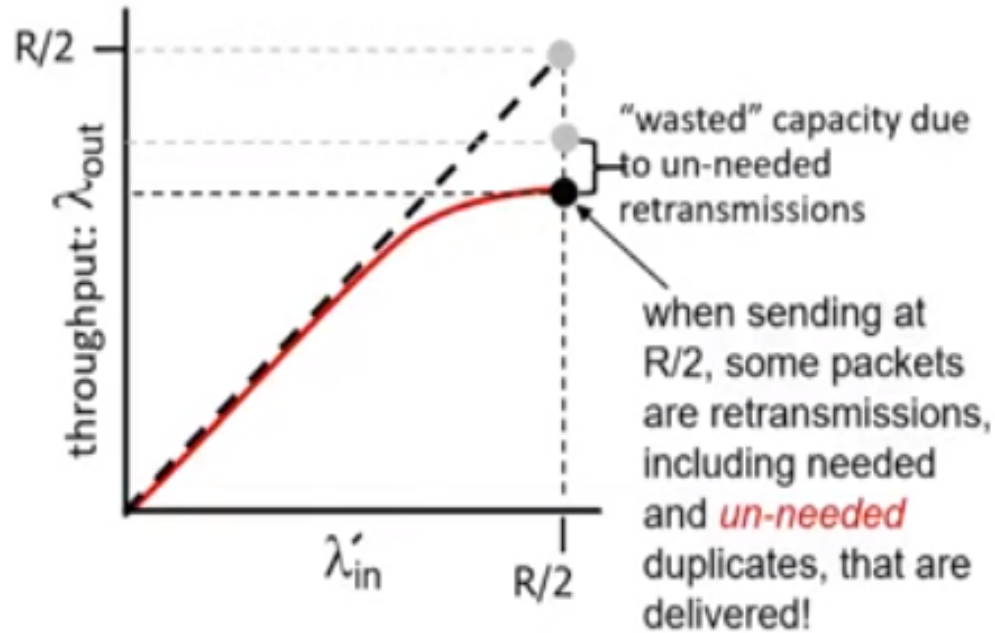


when sending at R/2, some packets are retransmissions, including needed and *un-needed* duplicates, that are delivered!

Causes/costs of congestion: scenario 2

Realistic scenario: *un-needed duplicates*

- packets can be lost, dropped at router due to full buffers – requiring retransmissions
- but sender times can time out prematurely, sending *two* copies, *both* of which are delivered

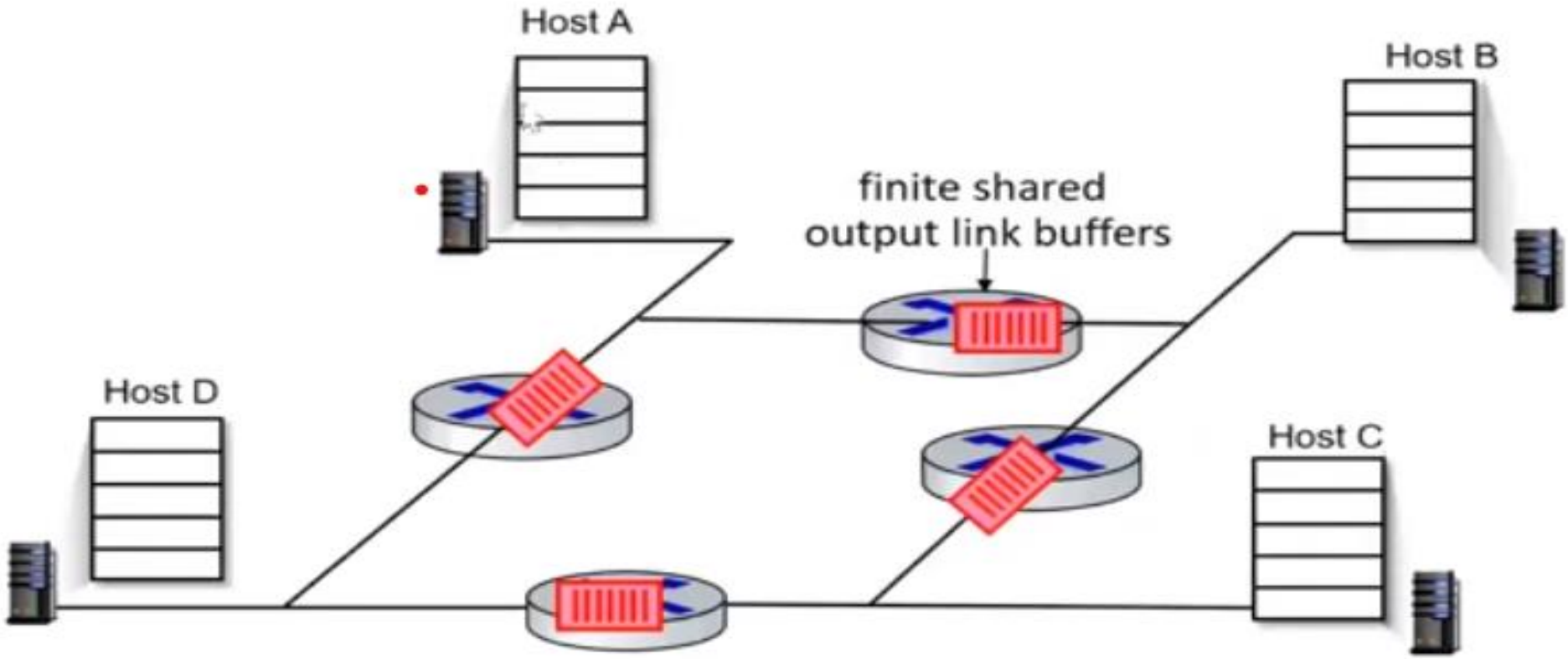


"costs" of congestion:

- more work (retransmission) for given receiver throughput
- unneeded retransmissions: link carries multiple copies of a packet
  - decreasing maximum achievable throughput

Causes/costs of congestion: scenario 3

- *four* senders
- *multi-hop* paths
- timeout/retransmit

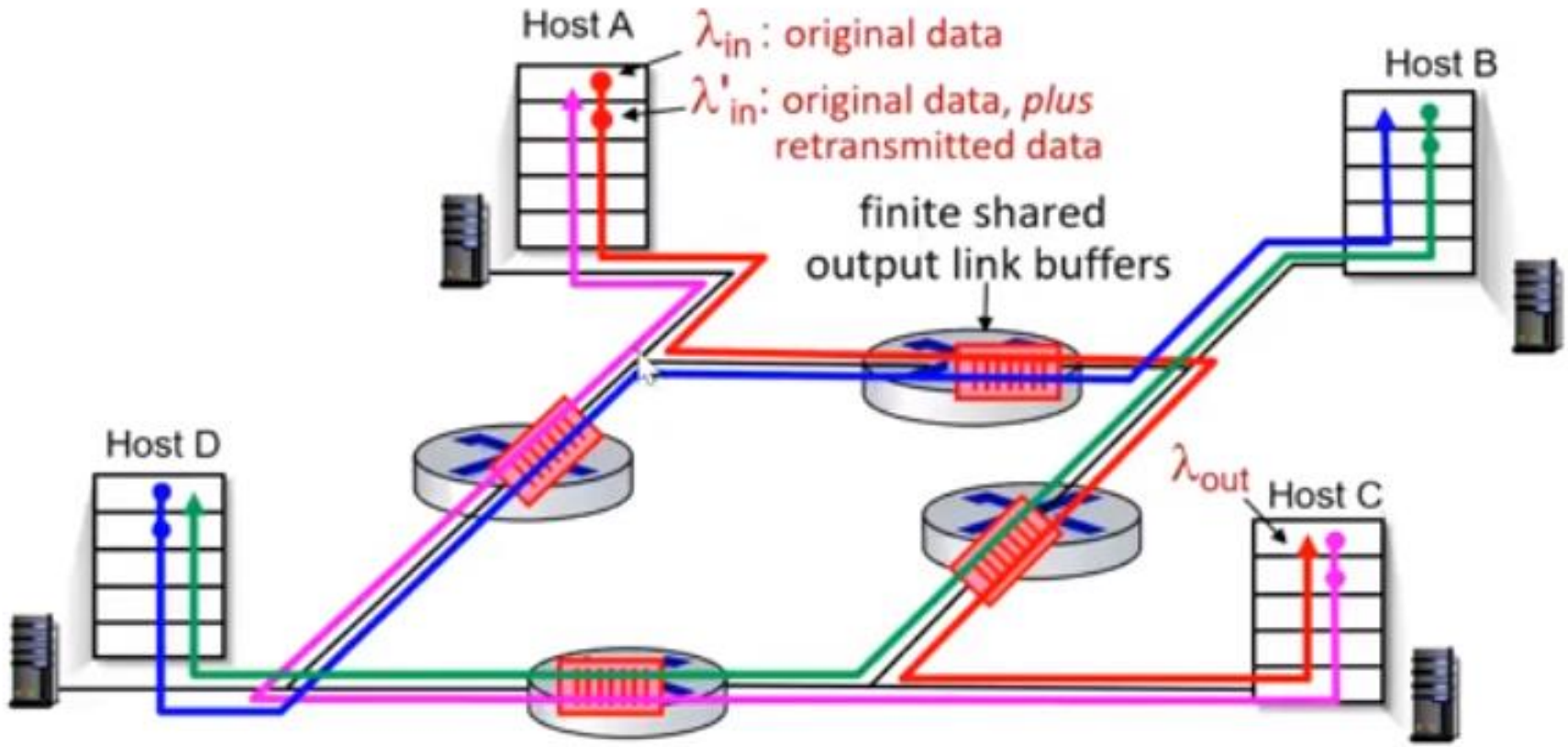


Causes/costs of congestion: scenario 3

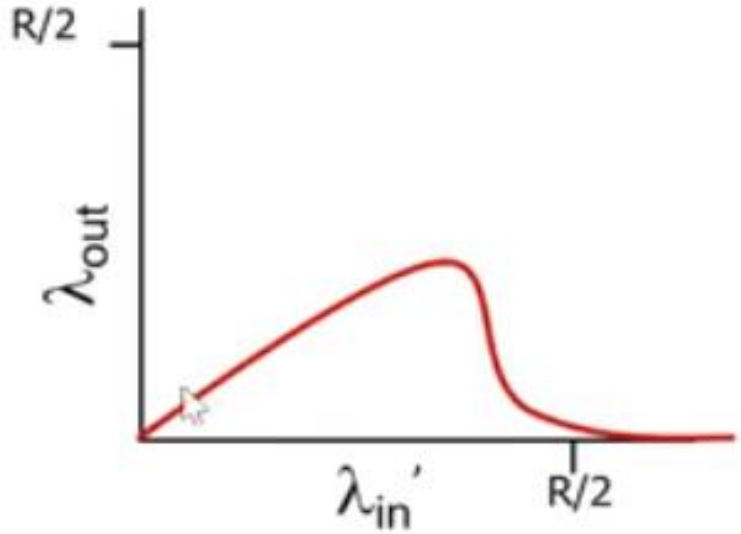
- *four* senders
- *multi-hop* paths
- timeout/retransmit

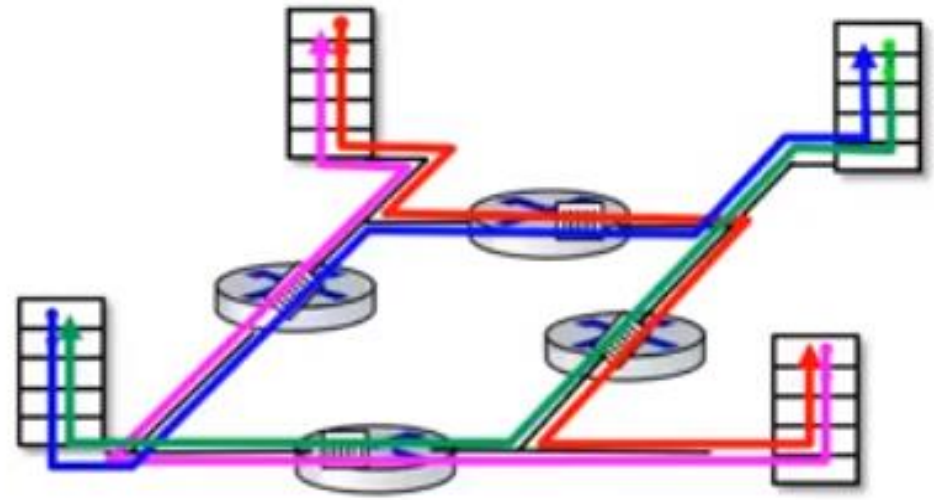
Q: what happens as λ_{in} and λ_{in}' increase ?

A: as red λ_{in}' increases, all arriving blue pkts at upper queue are dropped, blue throughput → 0



Causes/costs of congestion: scenario 3



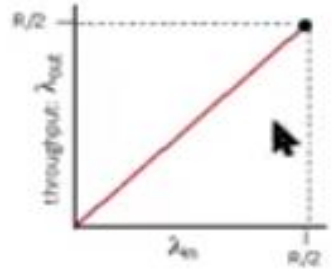


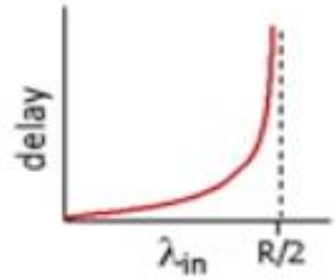
another "cost" of congestion:

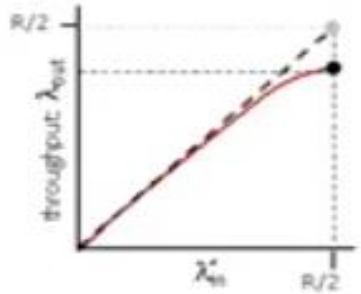
- when packet dropped, any upstream transmission capacity and buffering used for that packet was wasted!

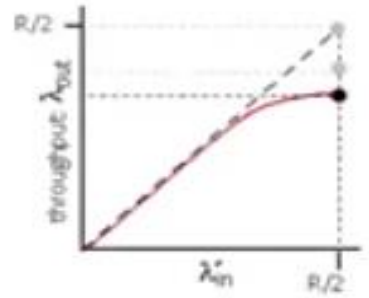
Causes/costs of congestion: insights

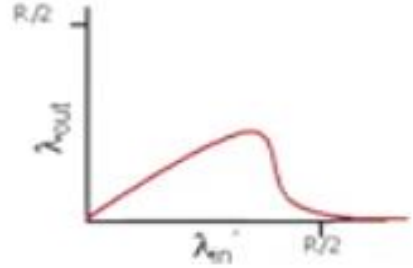
- throughput can never exceed capacity
- delay increases as capacity approached
- loss/retransmission decreases effective throughput
- un-needed duplicates further decreases effective throughput
- upstream transmission capacity / buffering wasted for packets lost downstream







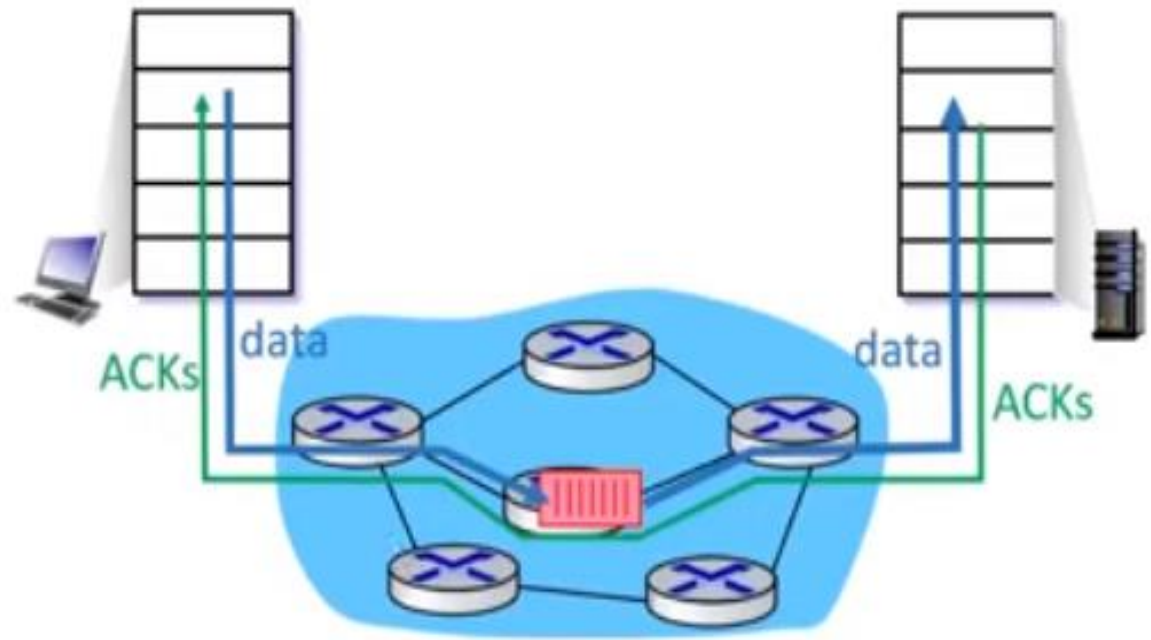




Approaches towards congestion control

End-end congestion control:

- no explicit feedback from network
- congestion *inferred* from observed loss, delay
- approach taken by TCP



Approaches towards congestion control

Network-assisted congestion control:

- routers provide *direct* feedback to sending/receiving hosts with flows passing through congested router
- may indicate congestion level or explicitly set sending rate
- TCP ECN, ATM, DECbit protocols