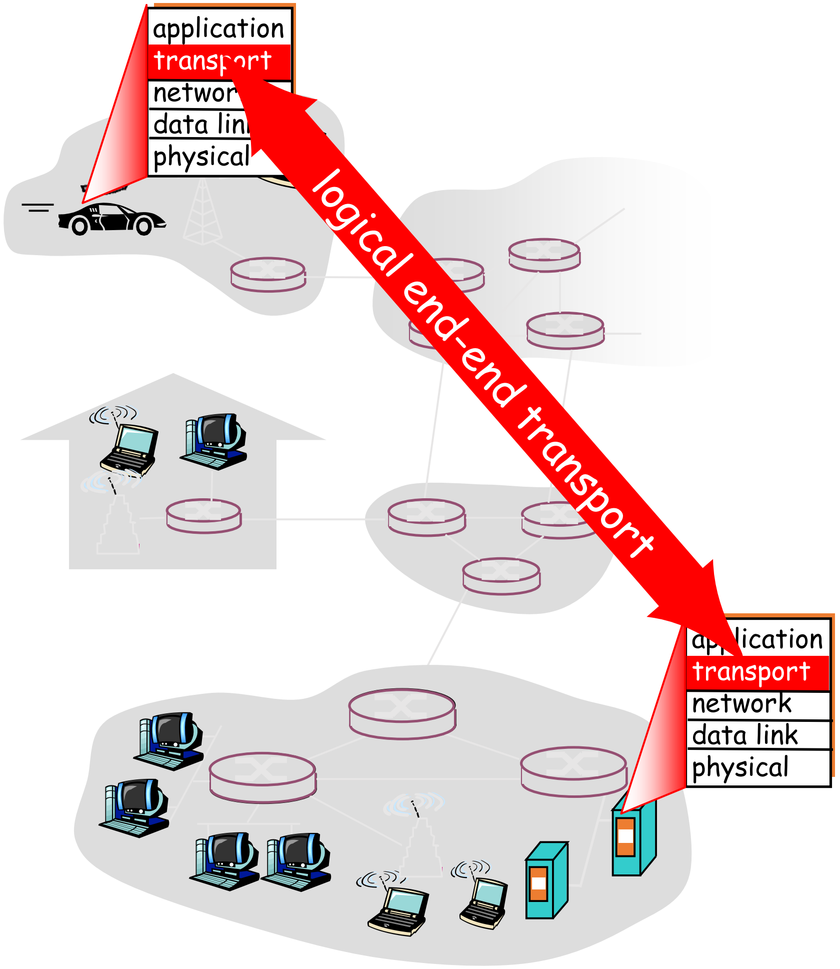
Transport Layer

Transport services and protocols

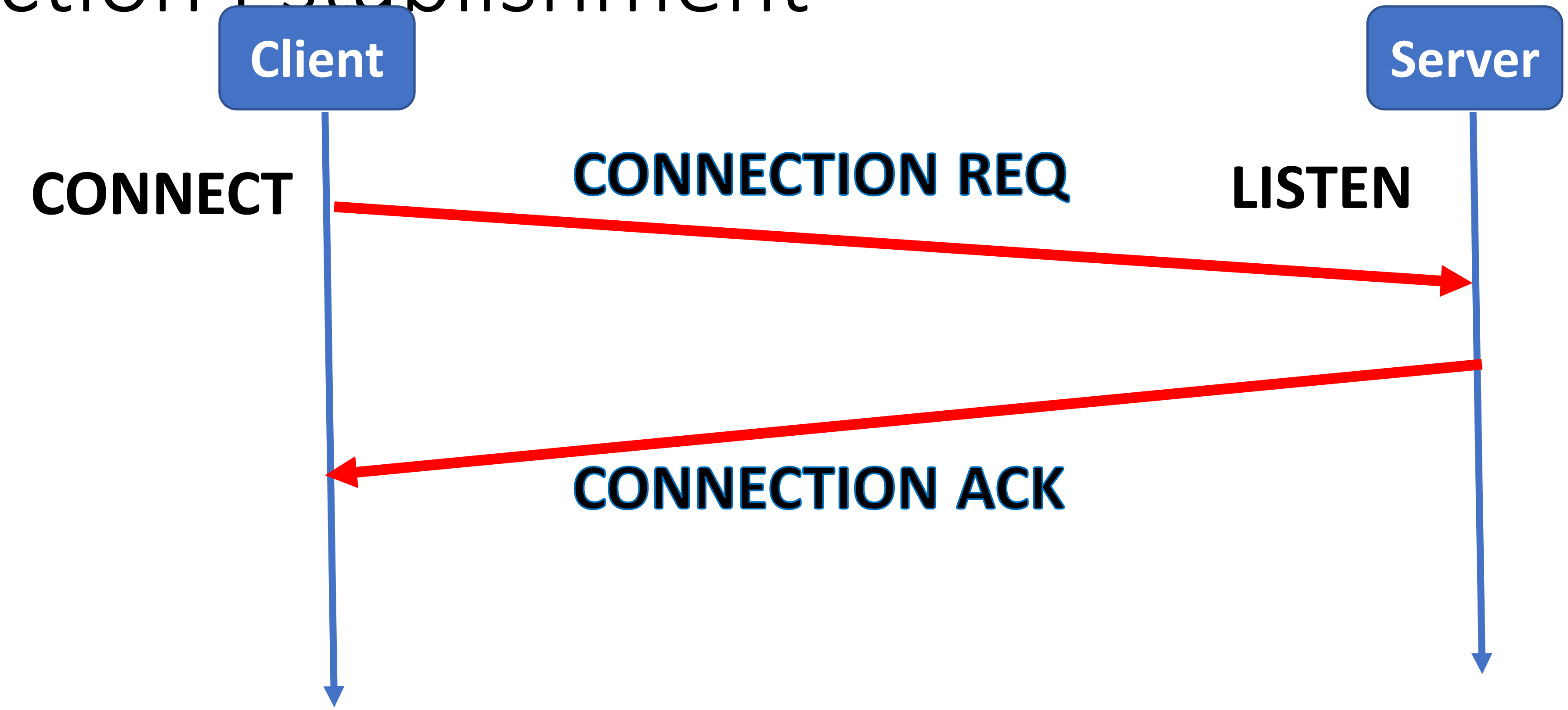
- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



Transport Layer Services

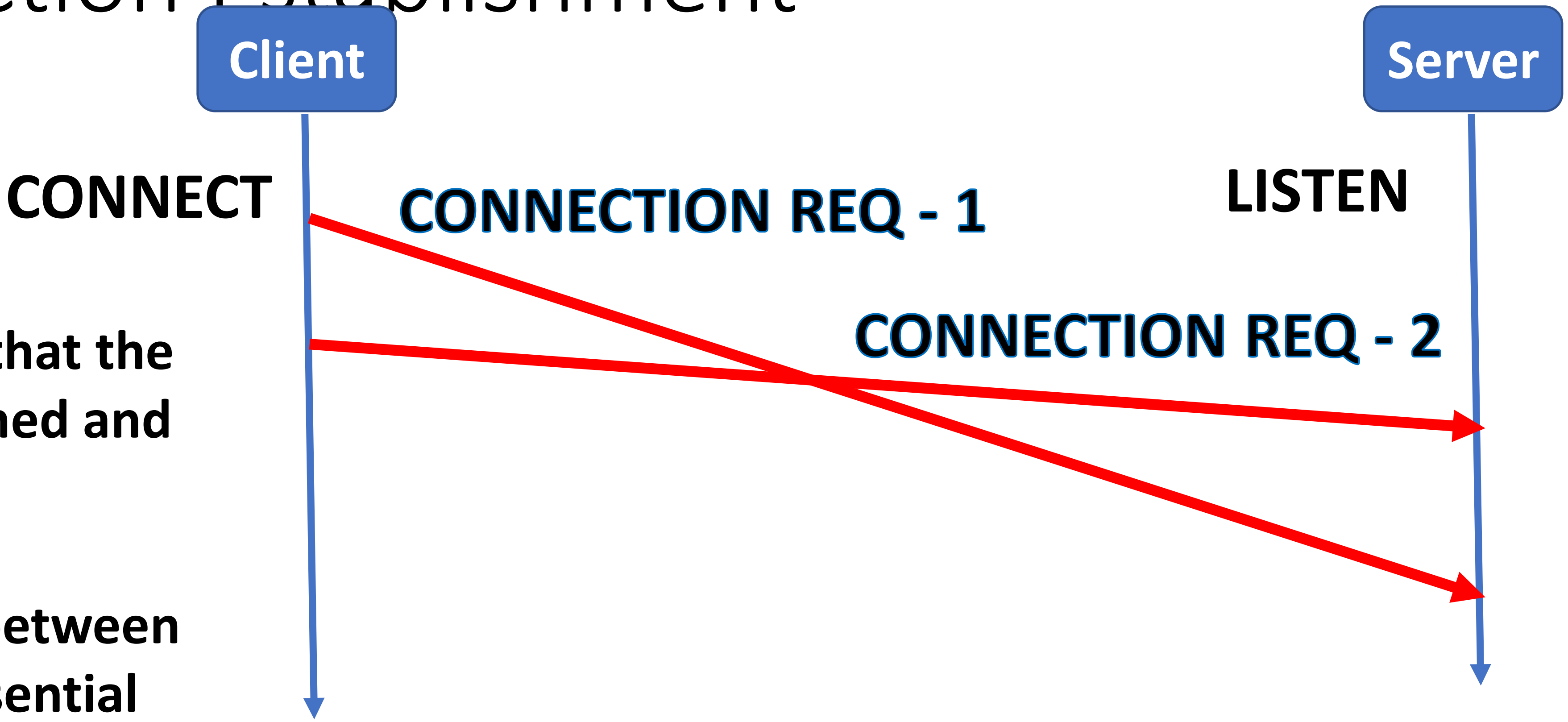
- Connection establishment
- Error and flow control
- Congestion control

Connection Establishment



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

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?

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

- **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

How to handle delayed duplicates

- **Solution 1: Give each connection a unique identifier chosen by the initiating party and put in each segment**
  - Can you see any problem in this approach?

How to handle delayed duplicates

- **Solution 1: Give each connection a unique identifier chosen by the initiating party and put in each segment**
    - Can you see any problem in this approach?
    - Problem 1: Need to store the identifier (or sequence numbers) in a table at both sender and receiver side.

How to handle delayed duplicates

- **Solution 1: Give each connection a unique identifier chosen by the initiating party and put in each segment**
  - Can you see any problem in this approach?
  - Problem 1: Need to store the identifier (or sequence numbers) in a table at both sender and receiver side.

Connection Establishment – Handling Delayed Duplicates

- **Solution 2: Devise a mechanism to kill off aged packets that are still hobbling about (Restrict the packet lifetime)**

Connection Establishment – Handling Delayed Duplicates

- **Solution 2: Devise a mechanism to kill off aged packets that are still hobbling about (Restrict the packet lifetime)**

- Two ways to restrict packet lifetime
  - Timestamping each packet – define the lifetime of a packet in the network, need time synchronization across each router.

Connection Establishment – Handling Delayed Duplicates

- **Solution 2: Devise a mechanism to kill off aged packets that are still hobbling about (Restrict the packet lifetime)**

- Two ways to restrict packet lifetime
    - **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.

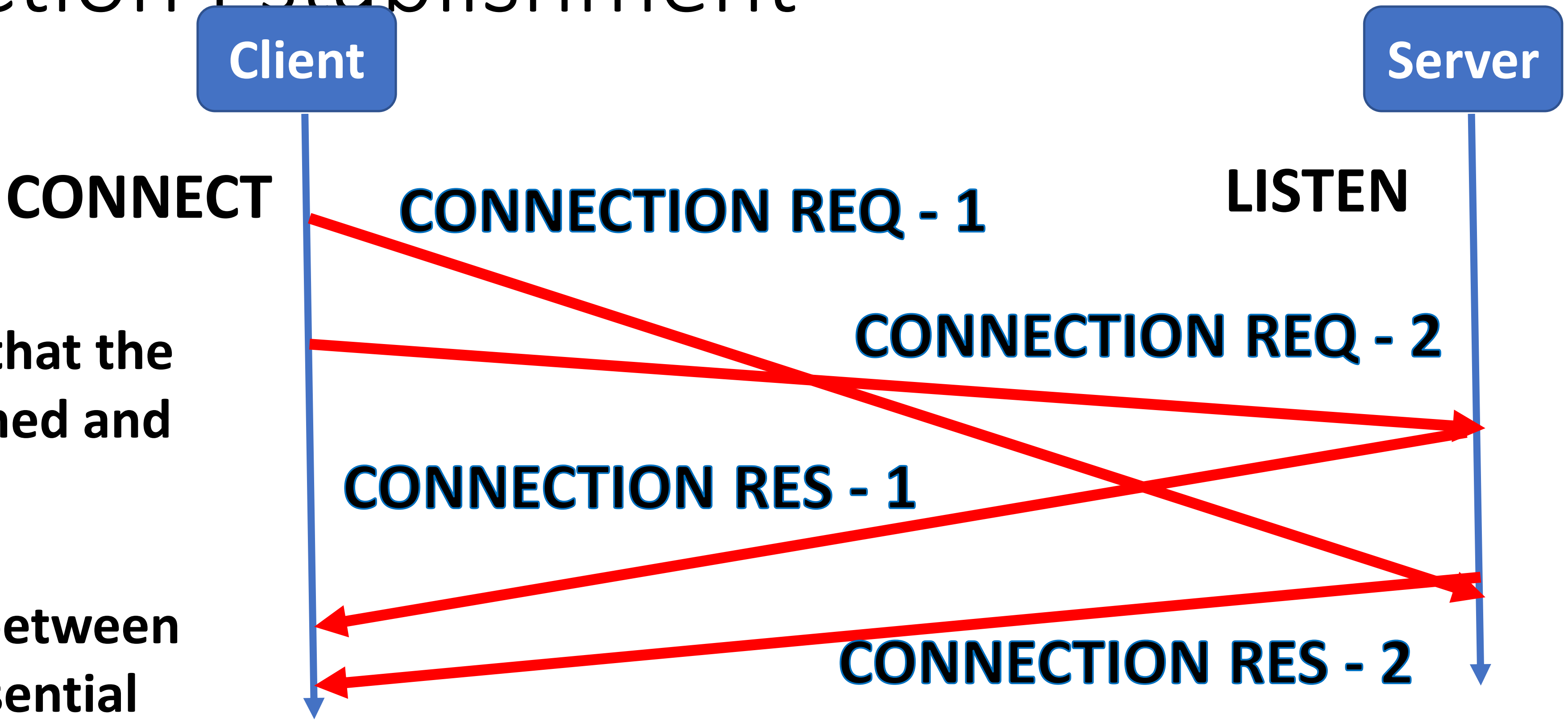
Connection Establishment – Handling Delayed Duplicates

- **Solution 2: Devise a mechanism to kill off aged packets that are still hobbling about (Restrict the packet lifetime)**

- Two ways to restrict packet lifetime
  - **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



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 RES-1 is a new connection response or a response of the CONNECTION RES-2?**

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**

- **May be, we can use Solution 1 and 2 together for meeting the challenge**

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

Connection Establishment – Handling Delayed Duplicates

- Let us define a maximum packet lifetime T

- Rather than a physical clock (clock synchronization in the Internet is difficult to achieve), let us use a virtual clock – **sequence number will represent a virtual clock**

Connection Establishment – Handling Delayed Duplicates

- Let us define a maximum packet lifetime T
- **sequence number generated based on the clock ticks**
- **Label segments with sequence numbers that will not be reused within T secs.**
- **At most one packet with a given sequence number may be outstanding at any given time**

Connection Establishment – Handling Delayed Duplicates

- **At most one packet with a given sequence number may be outstanding at any given time**

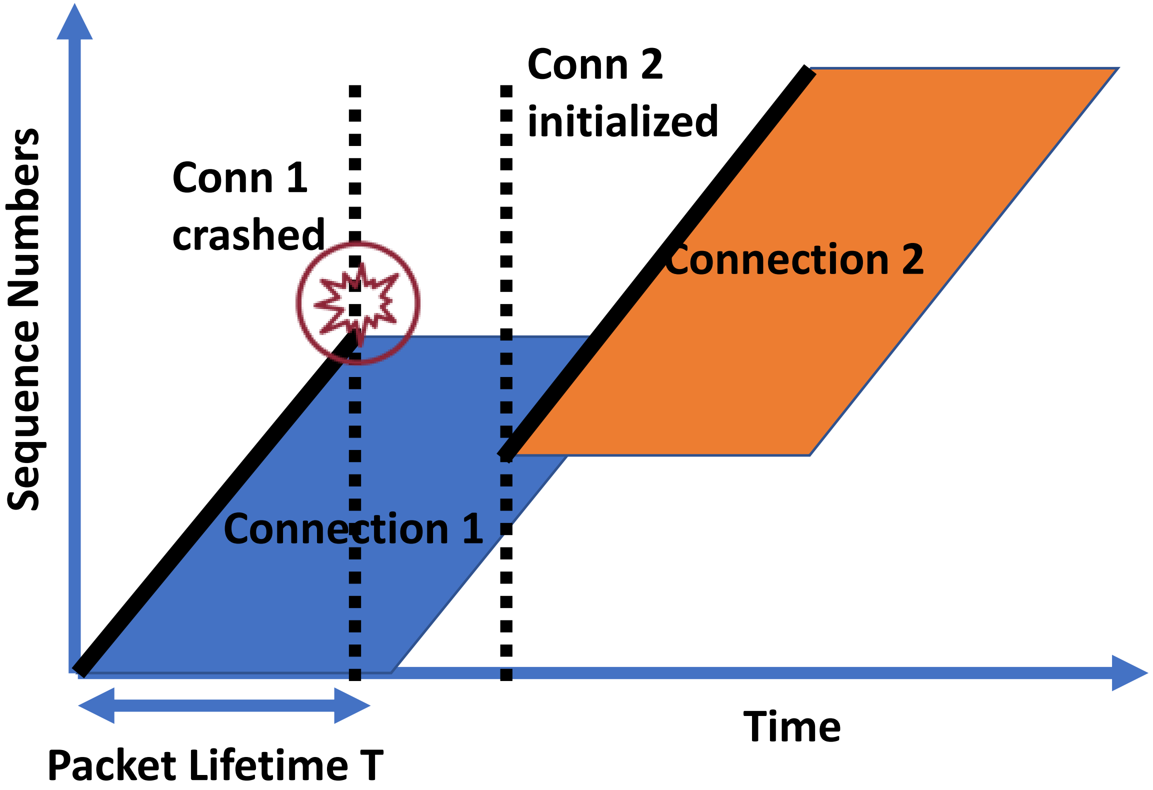
- If a receiver receives two segments having the same sequence number within a duration T, then one packet must be the duplicate. The receiver then discards the duplicate packets.

Connection Establishment – Handling Delayed Duplicates

- **What happens in case of crash of sender/receiver??**

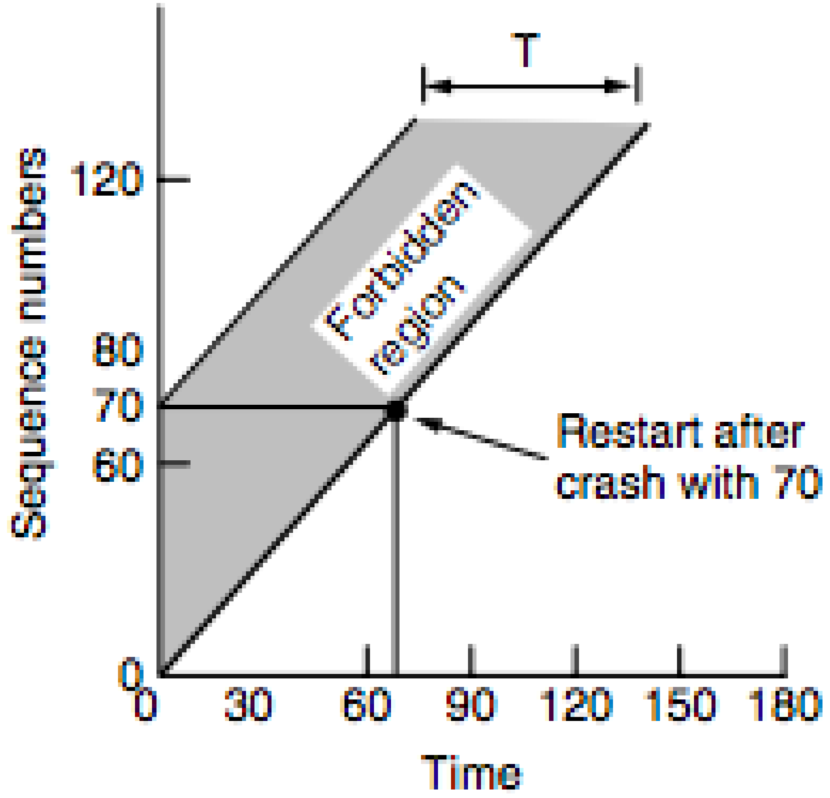
- **Where to start from, what sequence number to use??**

Why Initial Sequence Number is Important



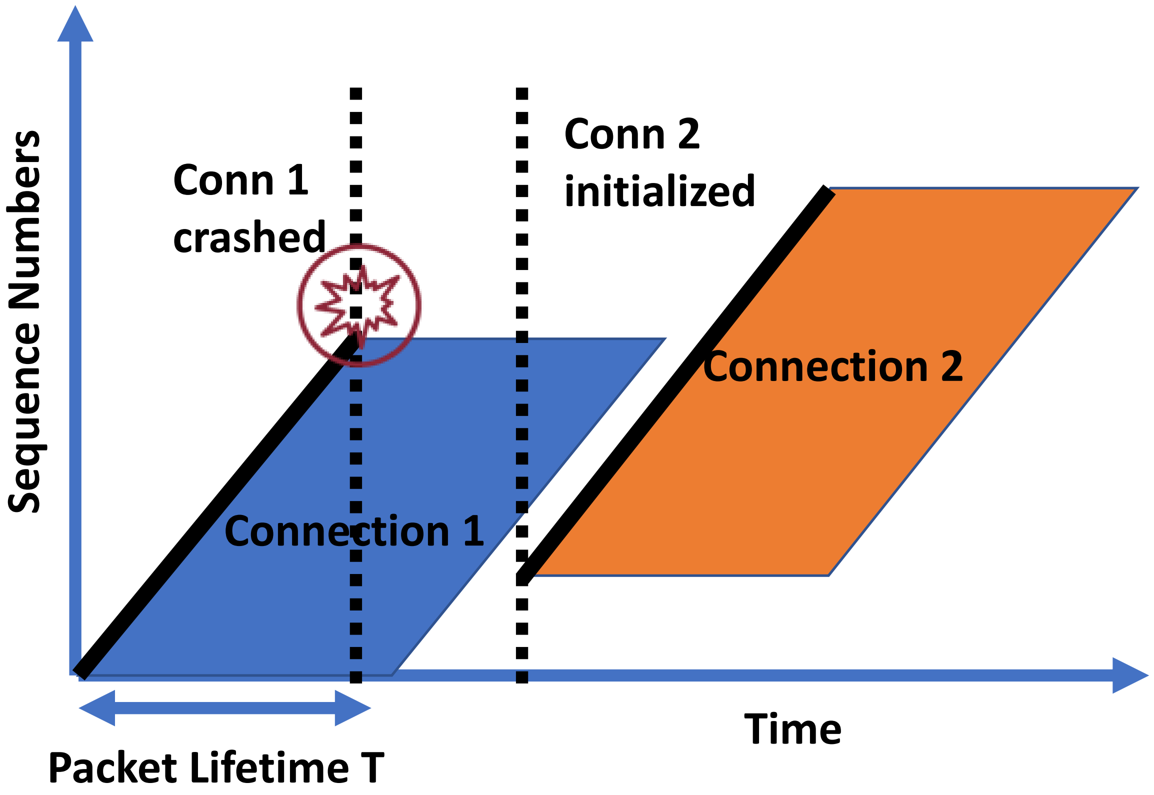
- A Delayed duplicate packet of connection 1 can create a confusion for connection 2

Connection Establishment – Handling Delayed Duplicates

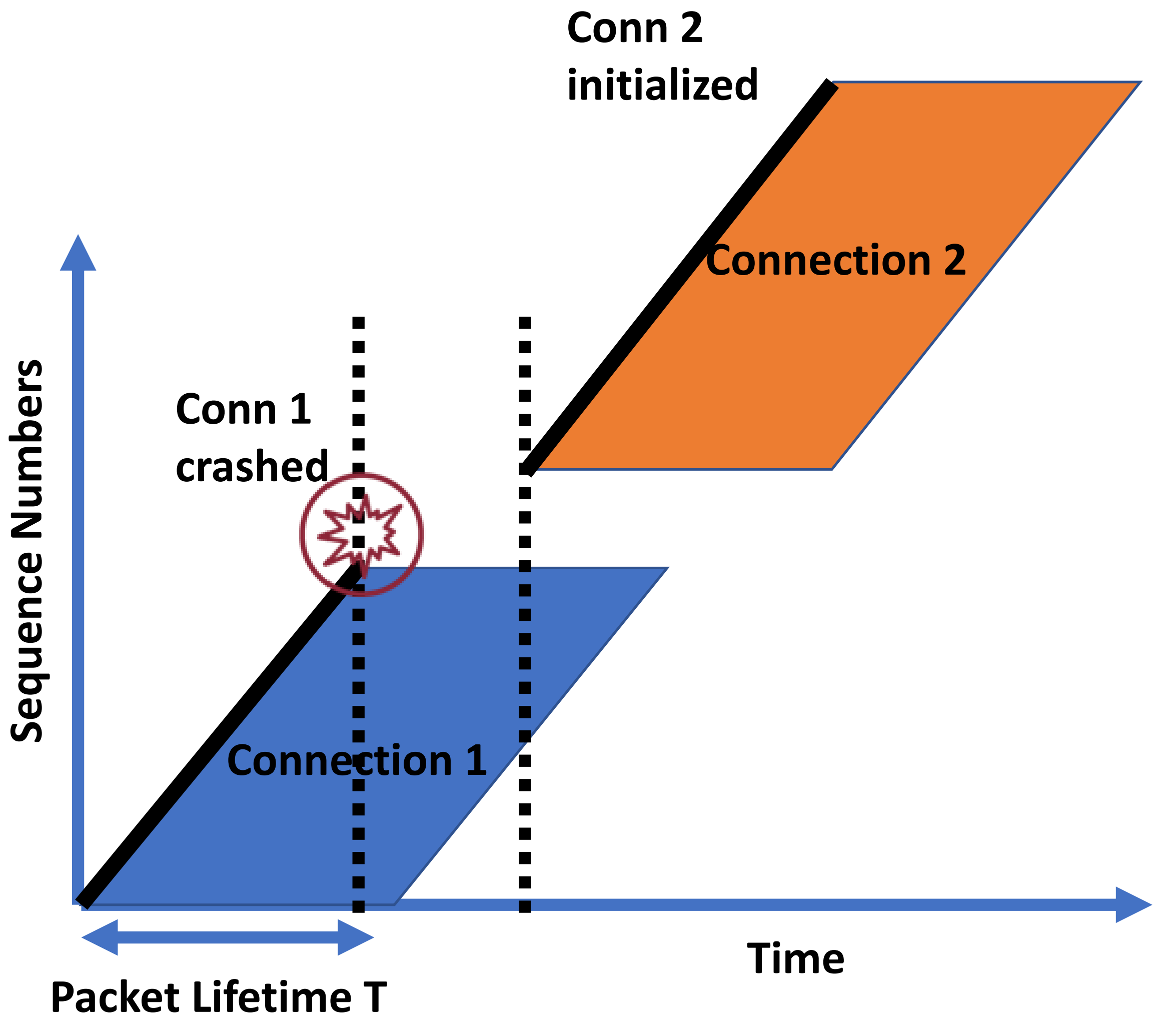


- **Adjust the initial sequence numbers properly - A host does not restart with a sequence number in the forbidden region, based on the sequence number it used before crash and the time duration T.**

What We Ideally Want? Or …



What We Ideally Want? Either …

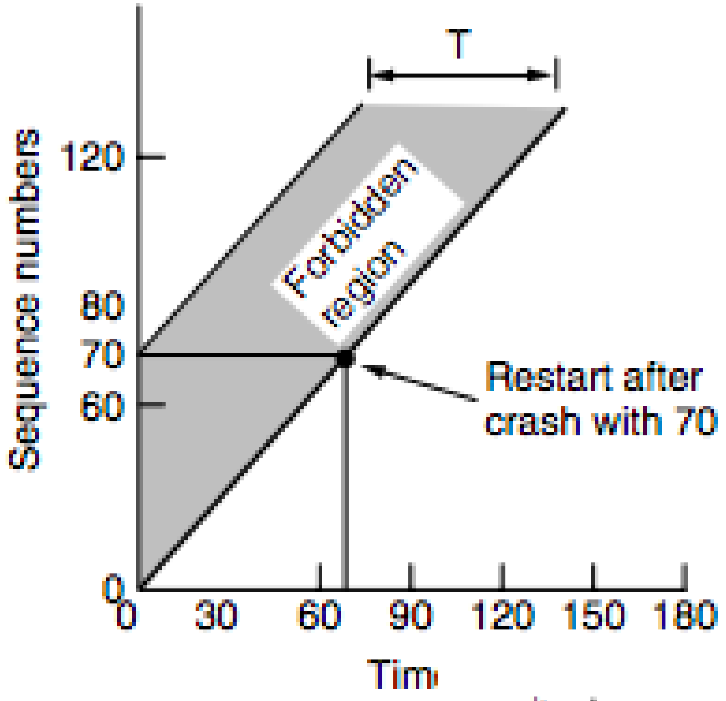


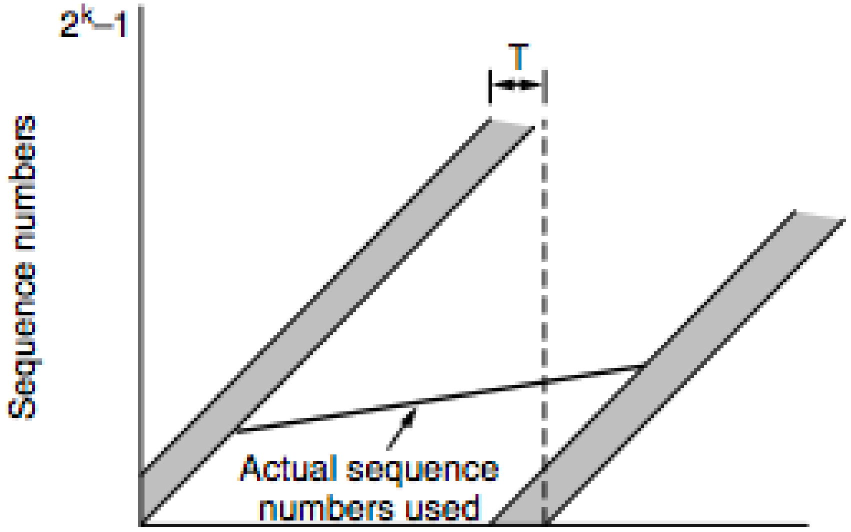
One problem while choosing a sequence number

- The period T and the rate of packets per second determine the size of the sequence number

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





Adjusting the Sending Rate based on Sequence Numbers

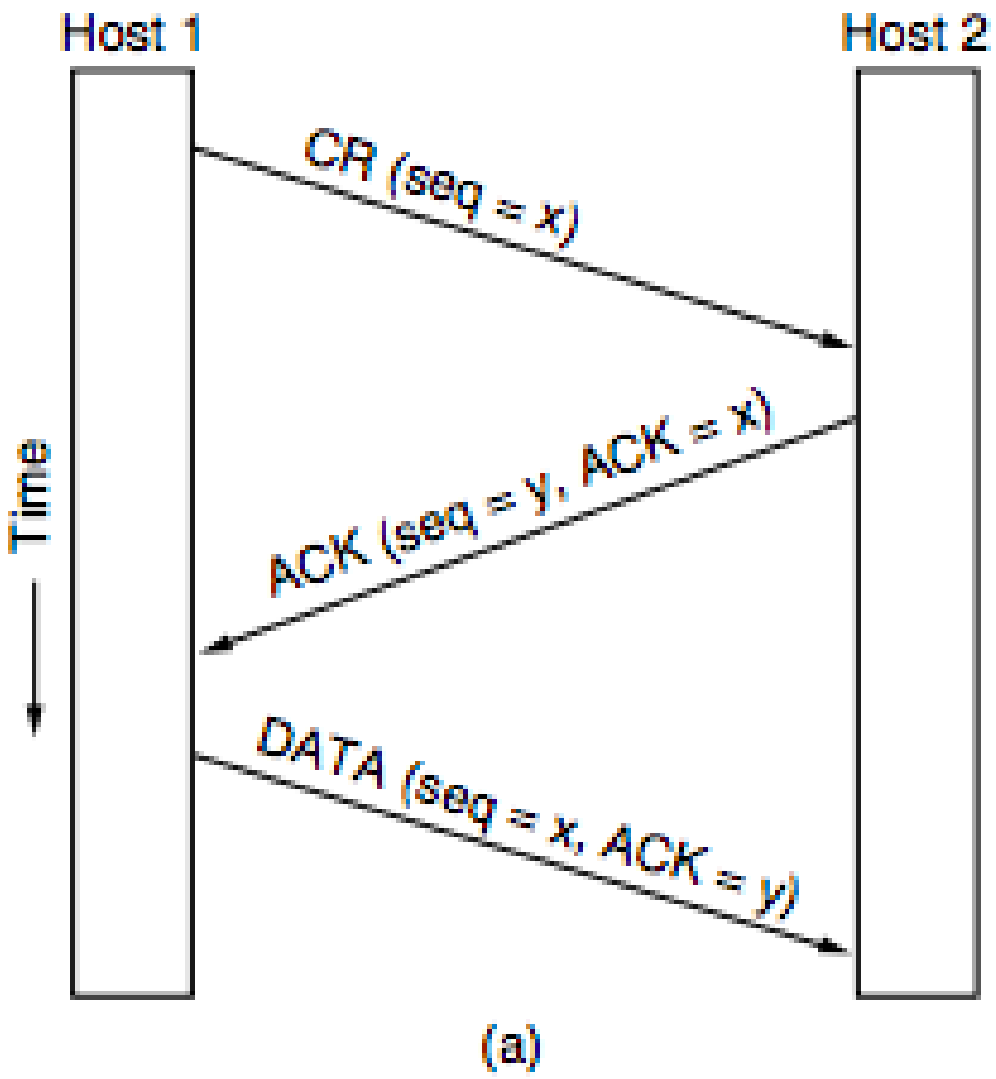
- The maximum data rate on any connection is one segment per clock tick
    - Clock ticks (inter-packet transmission duration) is adjusted based on the sequences acknowledged – **ensure that no two packets are there in the network with same sequence number**
    - **We call this mechanism as self-clocking (used in TCP)**
    - Ensures that the sequence numbers do not warp around too quickly (RFC 1323)

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**

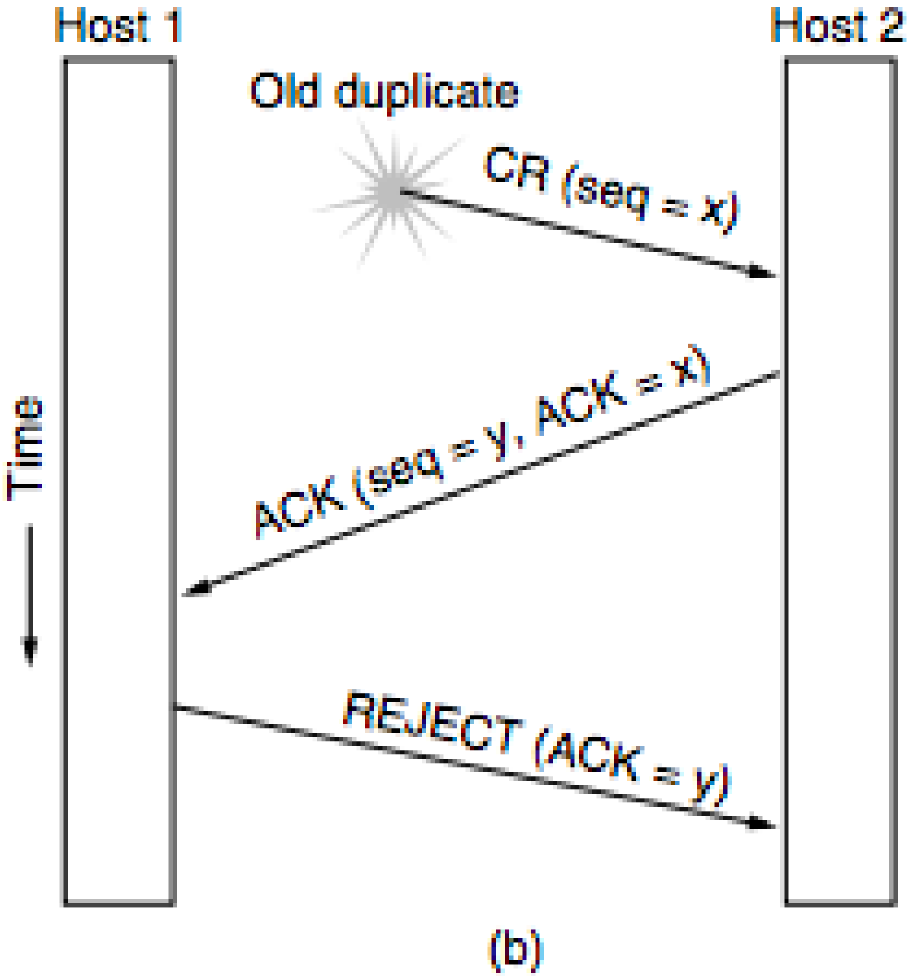
Three Way Handshake



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

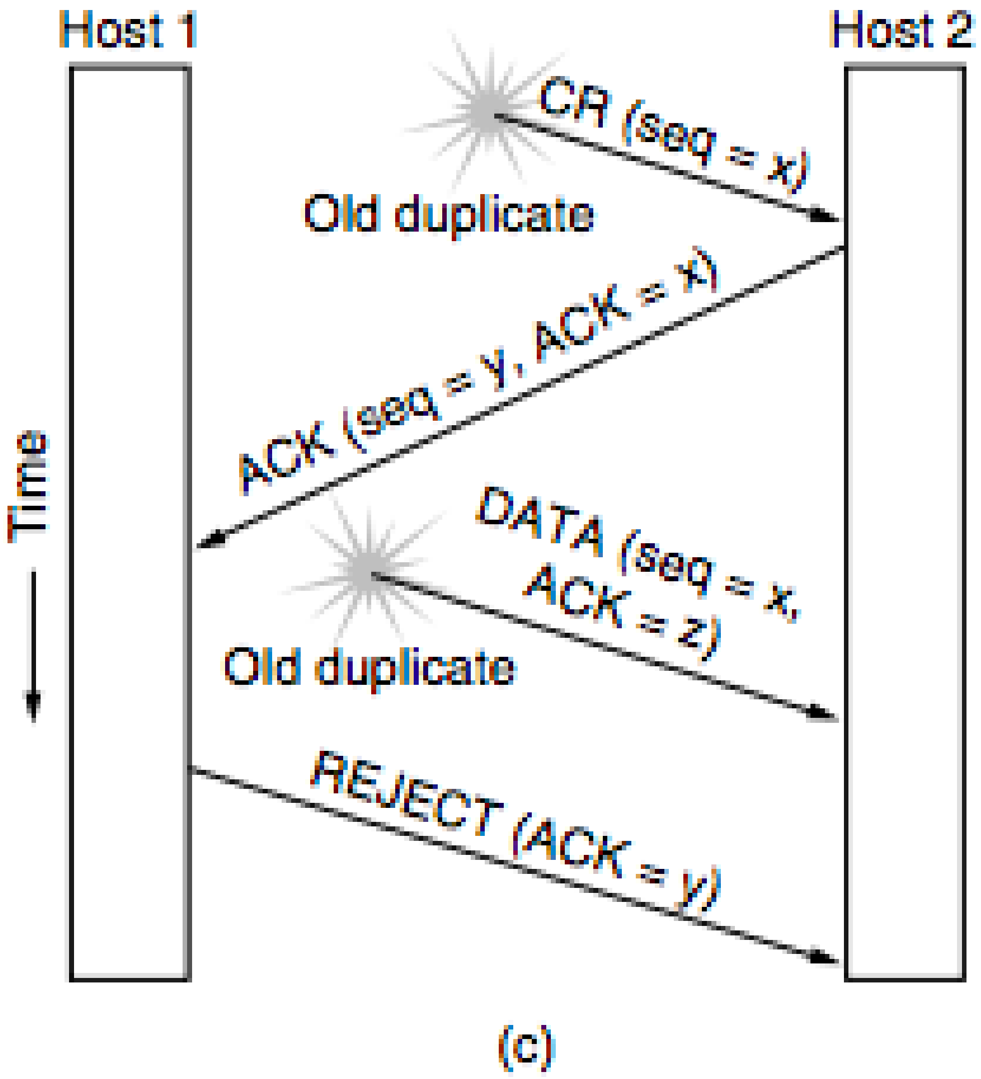
- By looking at the ACK in DATA, Host 2 ensures that sequence number y does not belong to the forbidden region of any previously established connection

Three Way Handshake – CONNECTION REQUEST is a Delayed Duplicate



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

Three Way Handshake – CONNECTION REQUEST and ACKNOWLEDGEMENT both are Delayed Duplicates



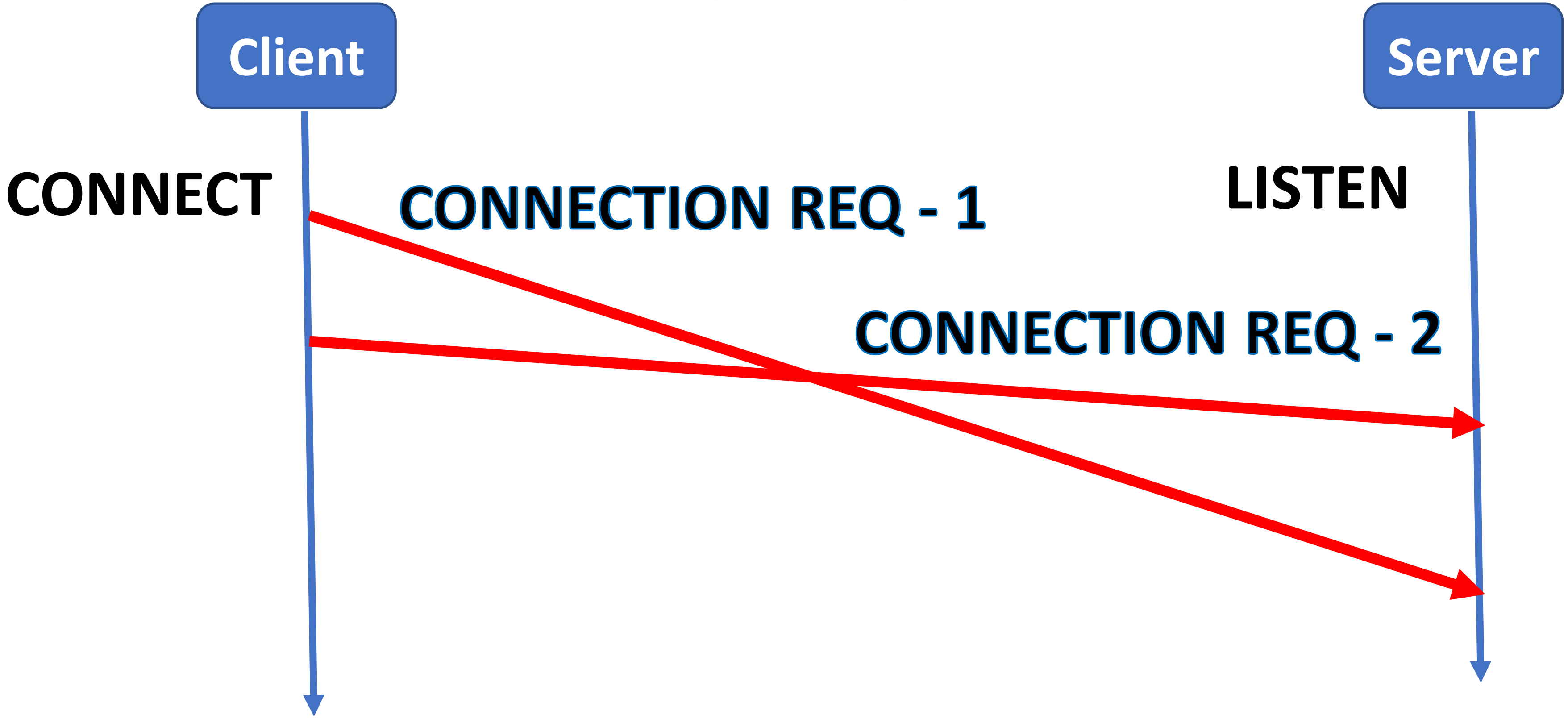
Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Sequence Number Adjustment

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

Recap

- Connection Establishment procedure – **Delayed Duplicates problem**



Recap

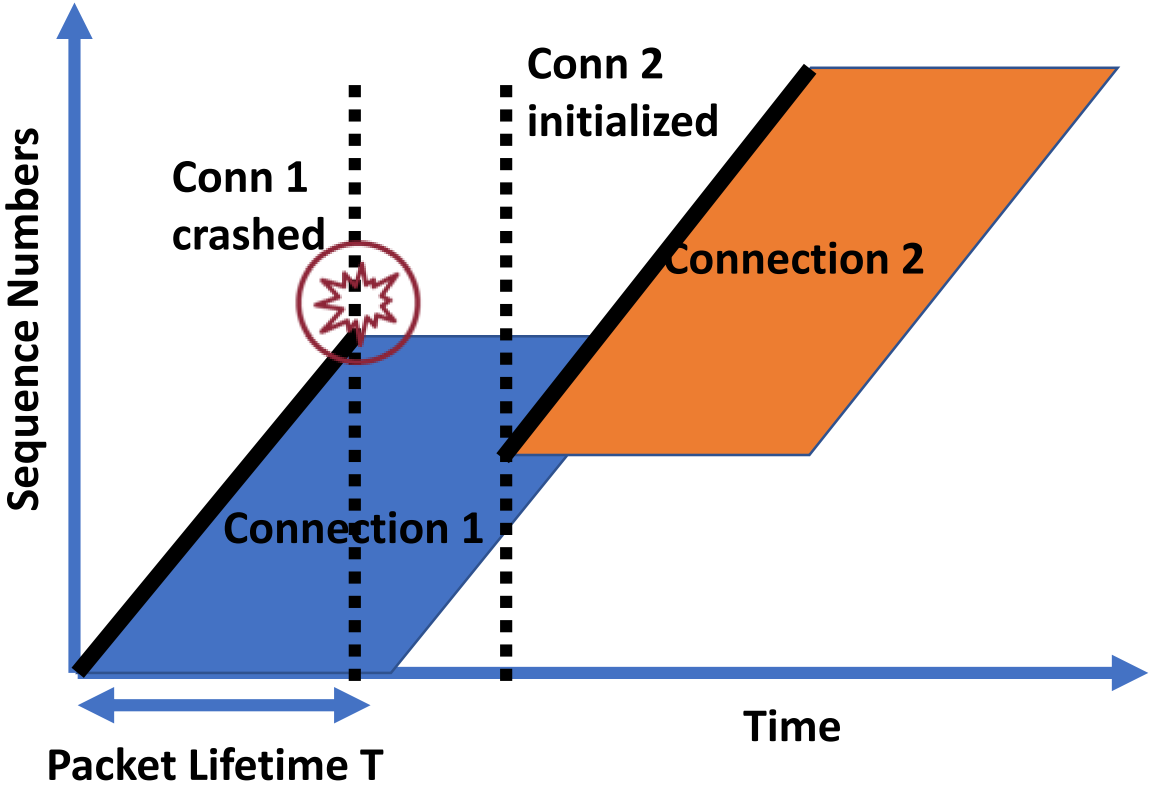
- **Connection Establishment – Handling Delayed Duplicates**

- **Solution:**
  - Let us define a maximum packet lifetime T
  - Label segments with sequence numbers that will not be reused within T secs.

- **At most one packet with a given sequence number may be outstanding at any given time**

Recap

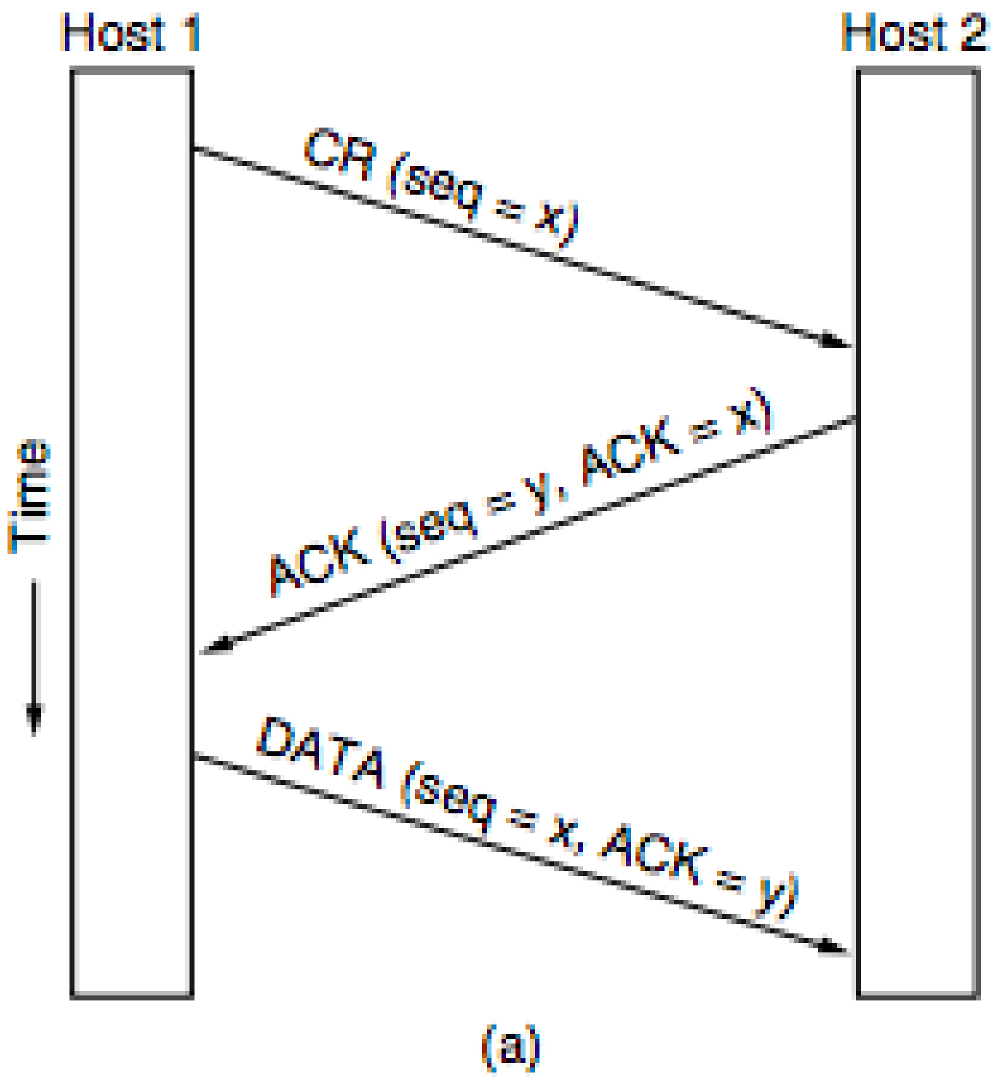
- Sequence Number Adjustment
  - **Choosing an Initial sequence number is important – as it does not clash with the sequence numbers currently used by another connection**



Recap

- Sequence Number Adjustment
  - **3-way handshake:** The valid range of sequence numbers must be positively synchronized between the sender and the receiver, whenever a connection is used.
  - Receiver need to inform sender correctly which connection it is acknowledging, so that the sender can accordingly distinguish an old connection request from the current/new one

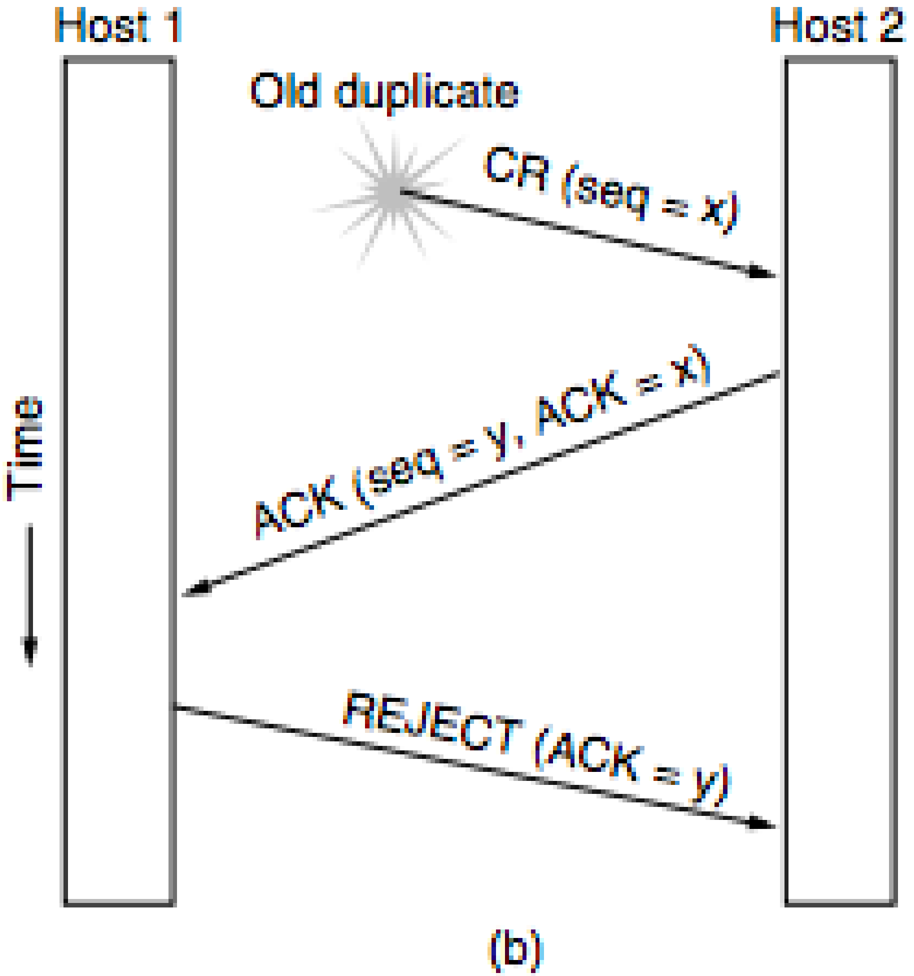
Recap - Three Way Handshake



- 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

Three Way Handshake – CONNECTION REQUEST is a Delayed Duplicate

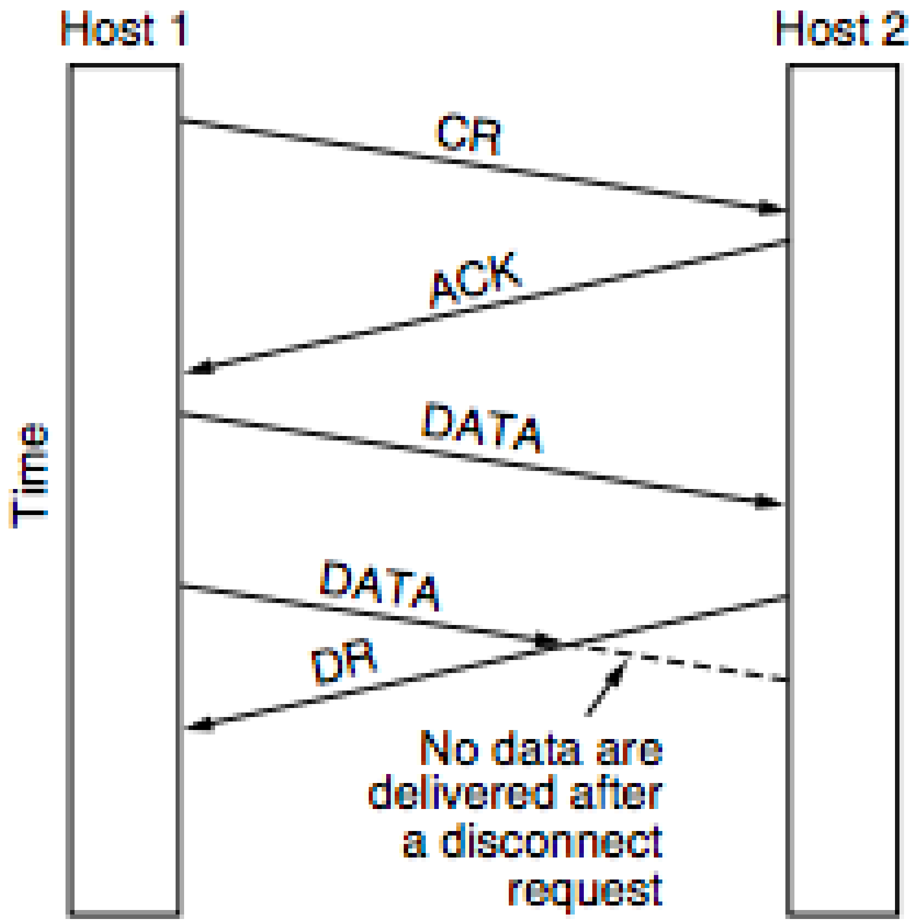


Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

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

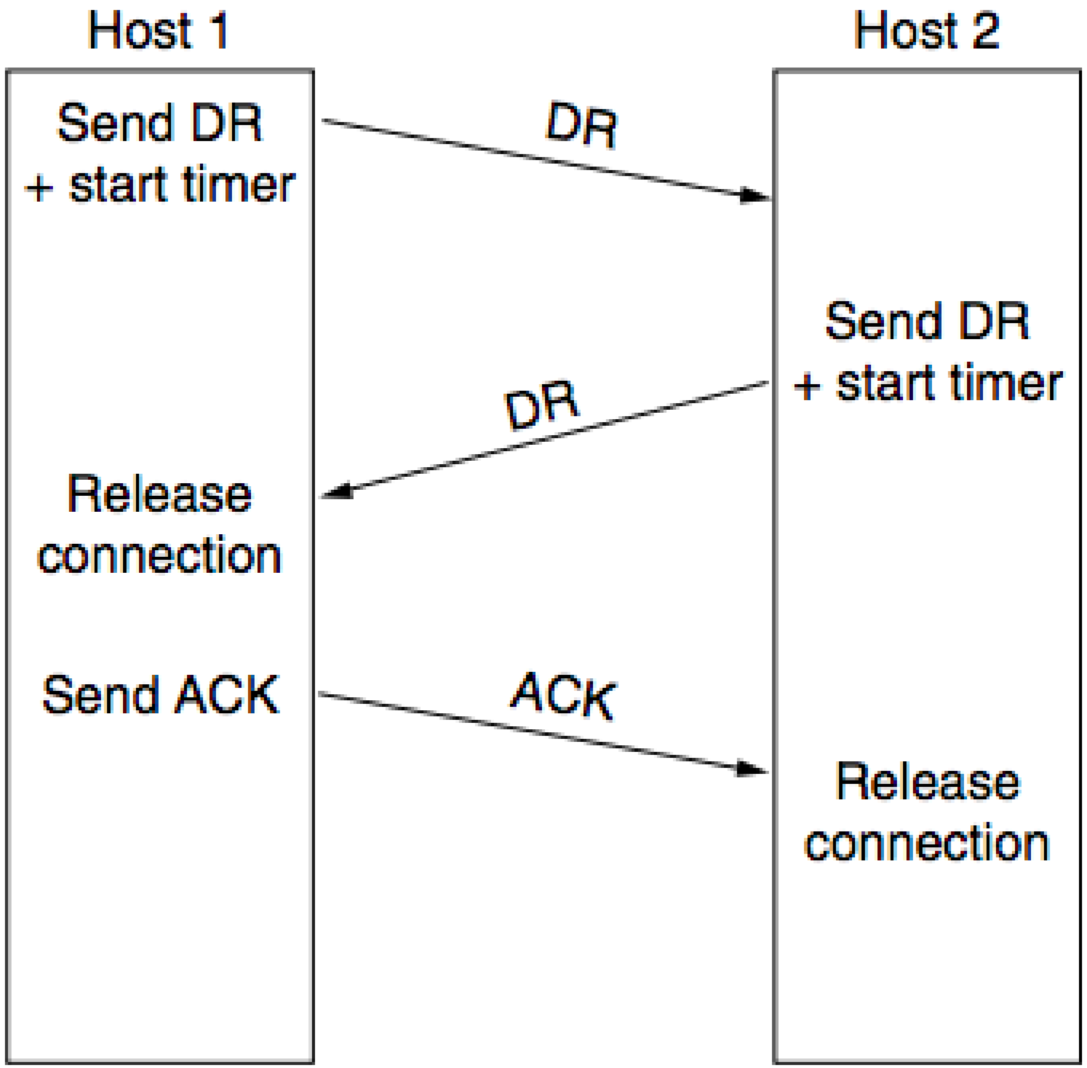
- Treats the connection as two separate unidirectional connections and requires each one to be released separately

- 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?
    - **Host 1: “I am done”**
    - **Host 2: ”I am done too”**

- **Does this protocol work good always?**

Connection Release



Host 1

Host 2

Send DR
+ start timer

DR

Send DR
+ start timer

DR

Release
connection

Send ACK

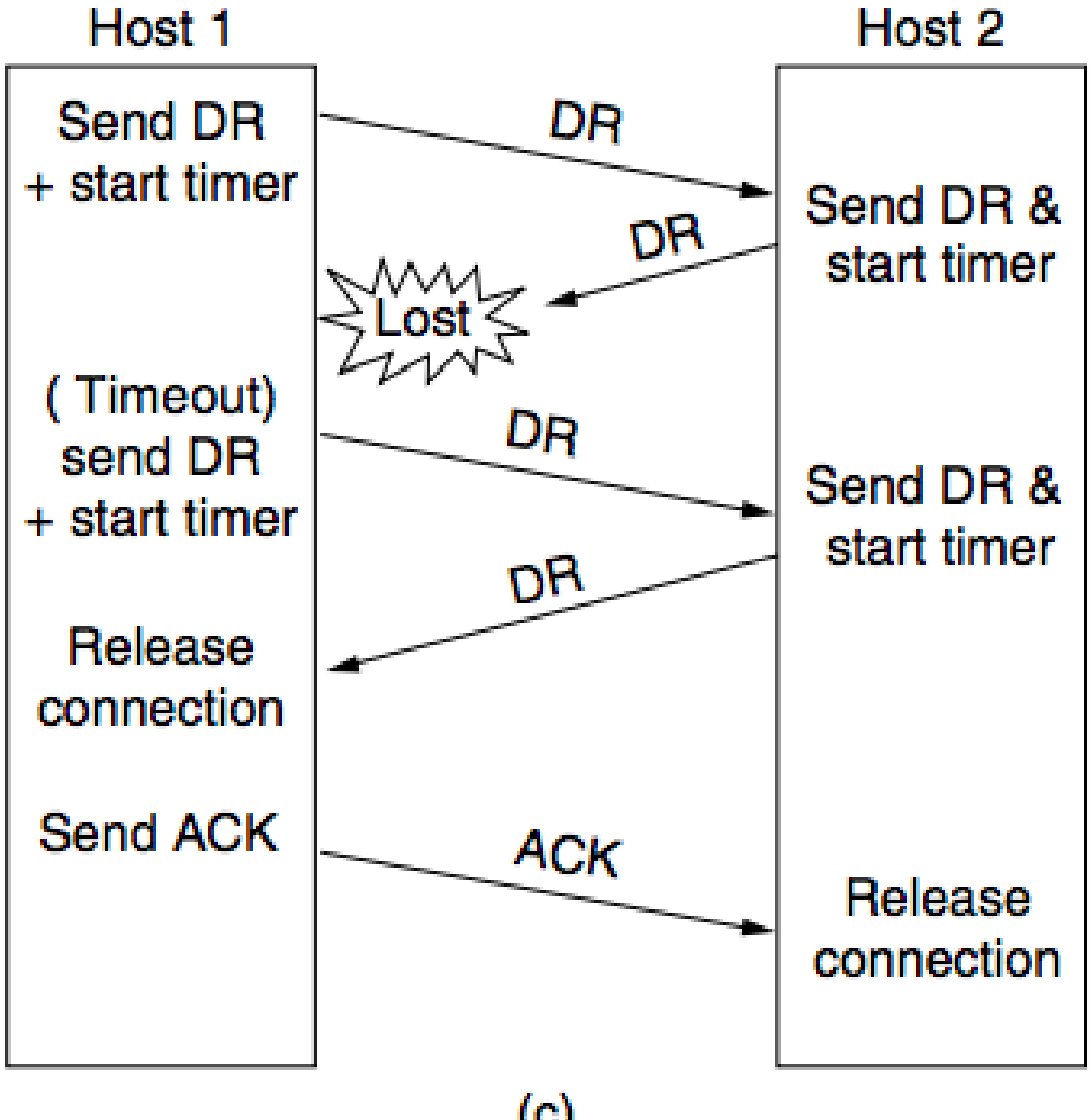
ACK

Release
connection

(a)

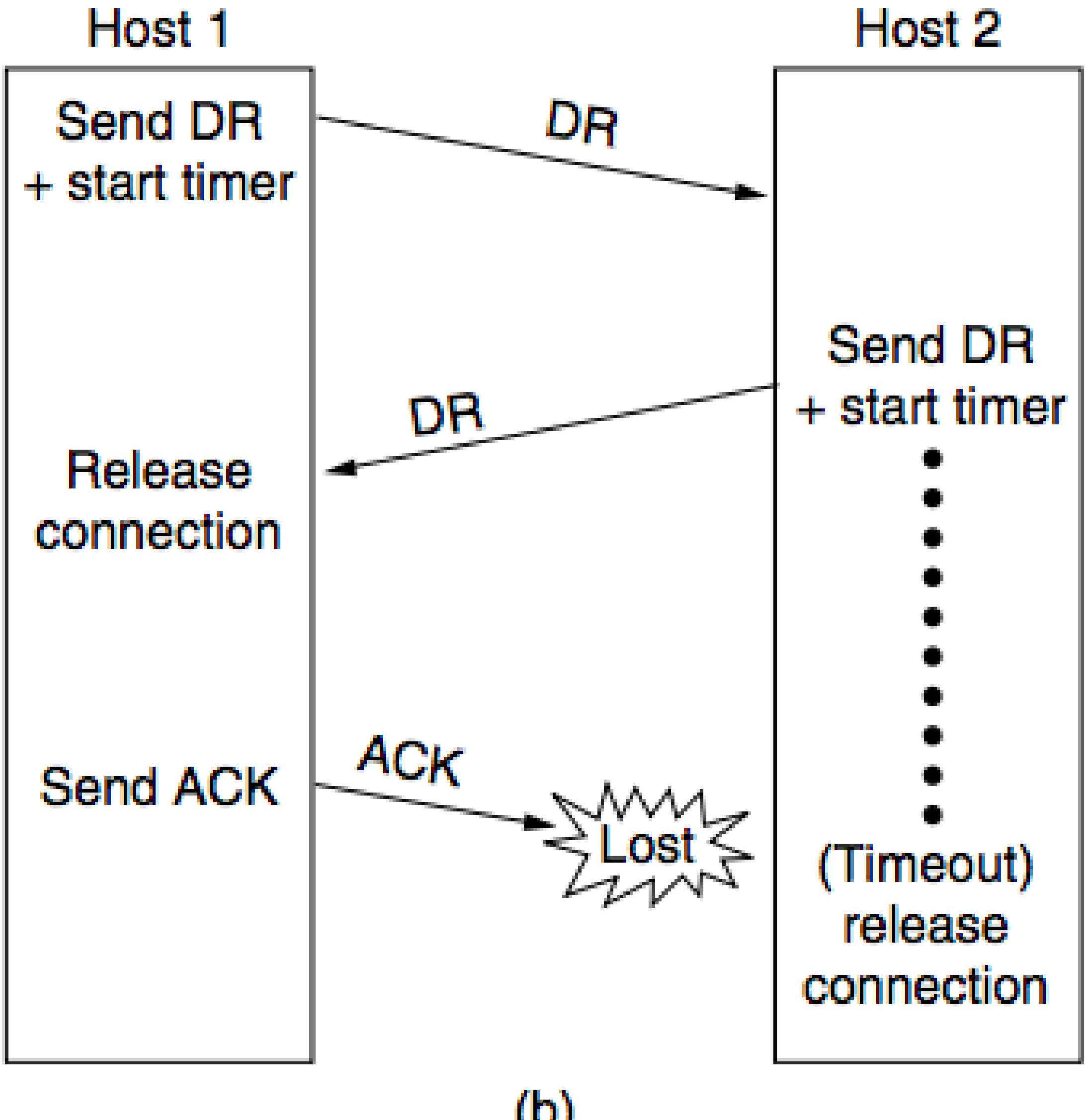
Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Connection Release – Response Lost



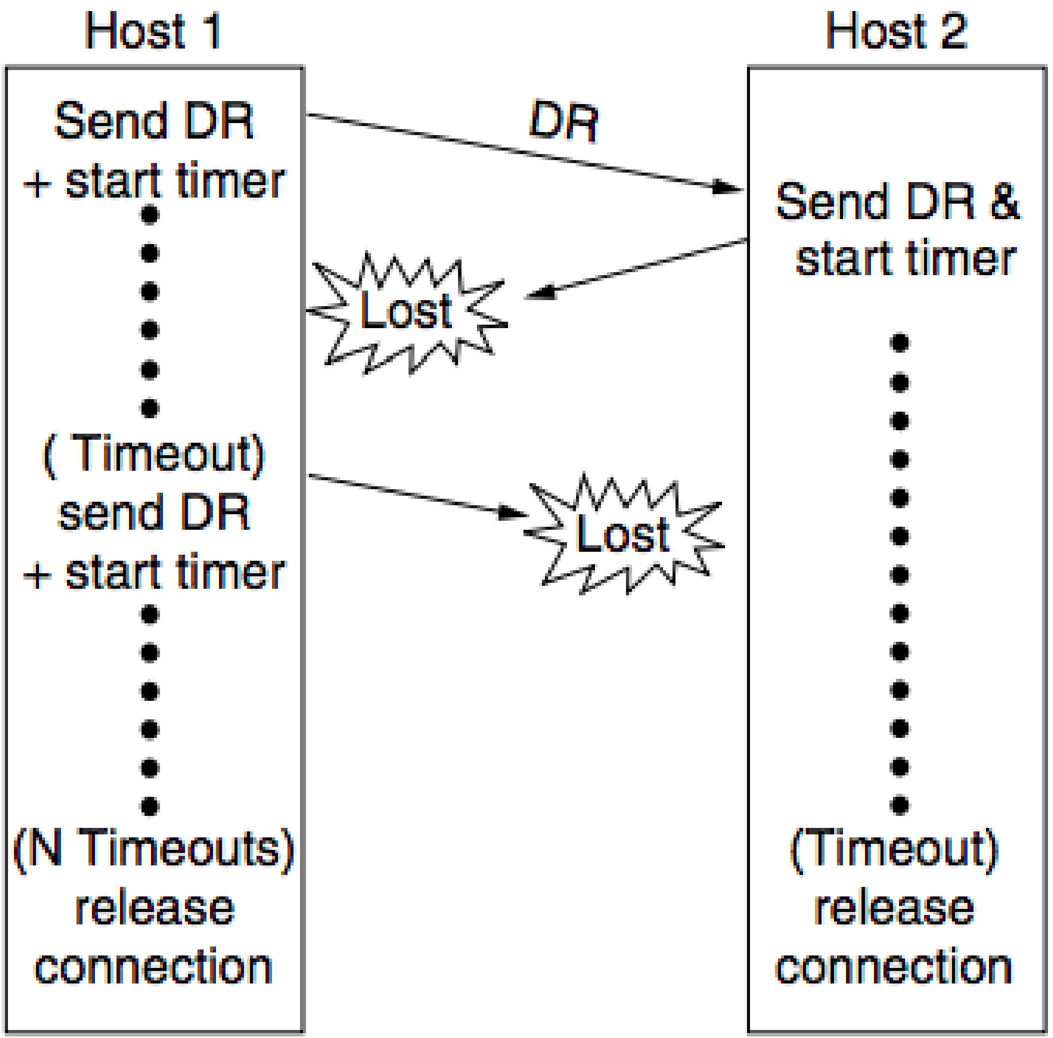
Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Connection Release – Final ACK Lost



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

Connection Release – Response Lost and Subsequent DRs Lost

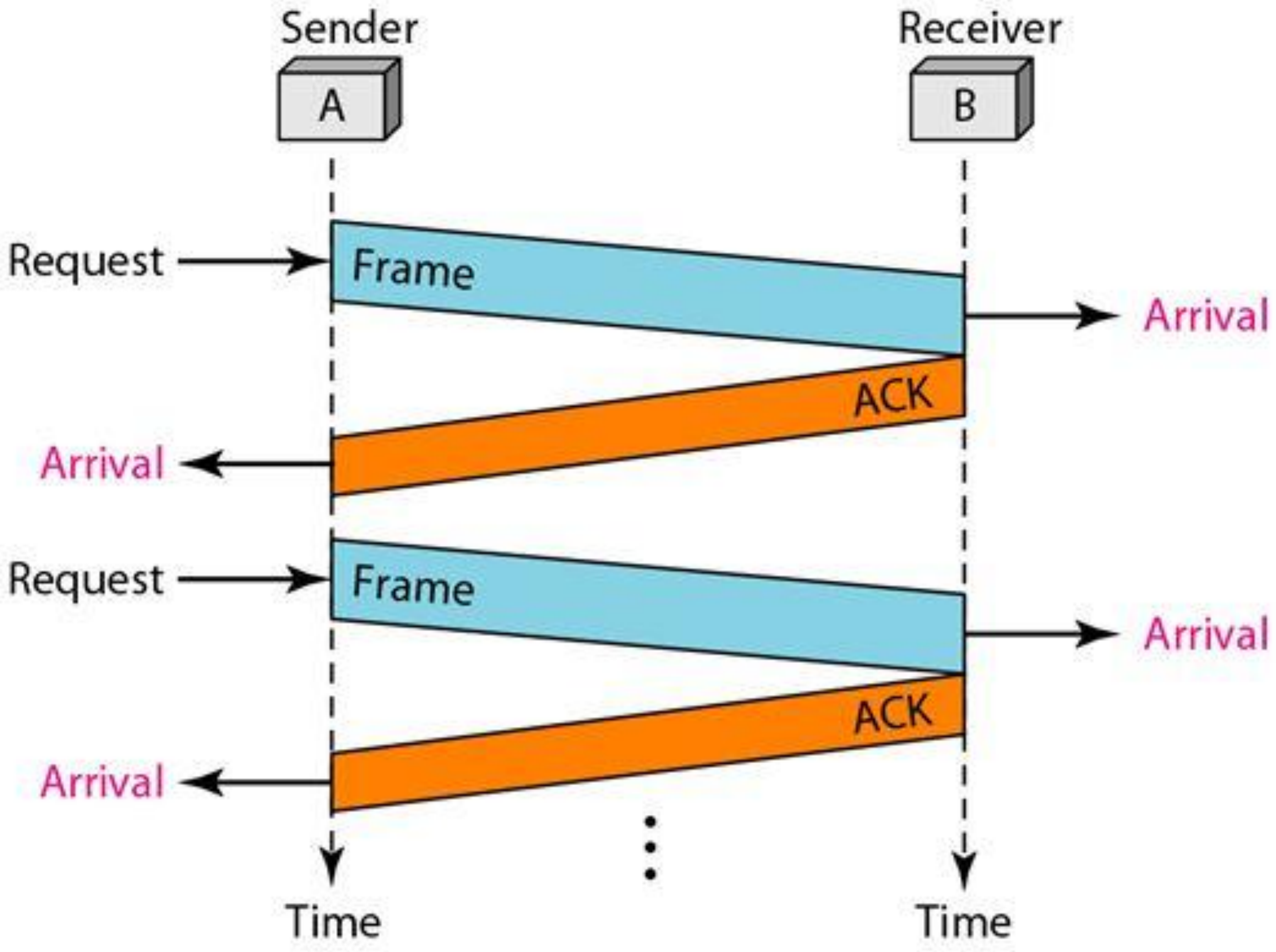


Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Flow Control Algorithms

- **Stop and Wait Flow**

sender sends one frame and then waits for an acknowledgement before proceeding to send next frame.

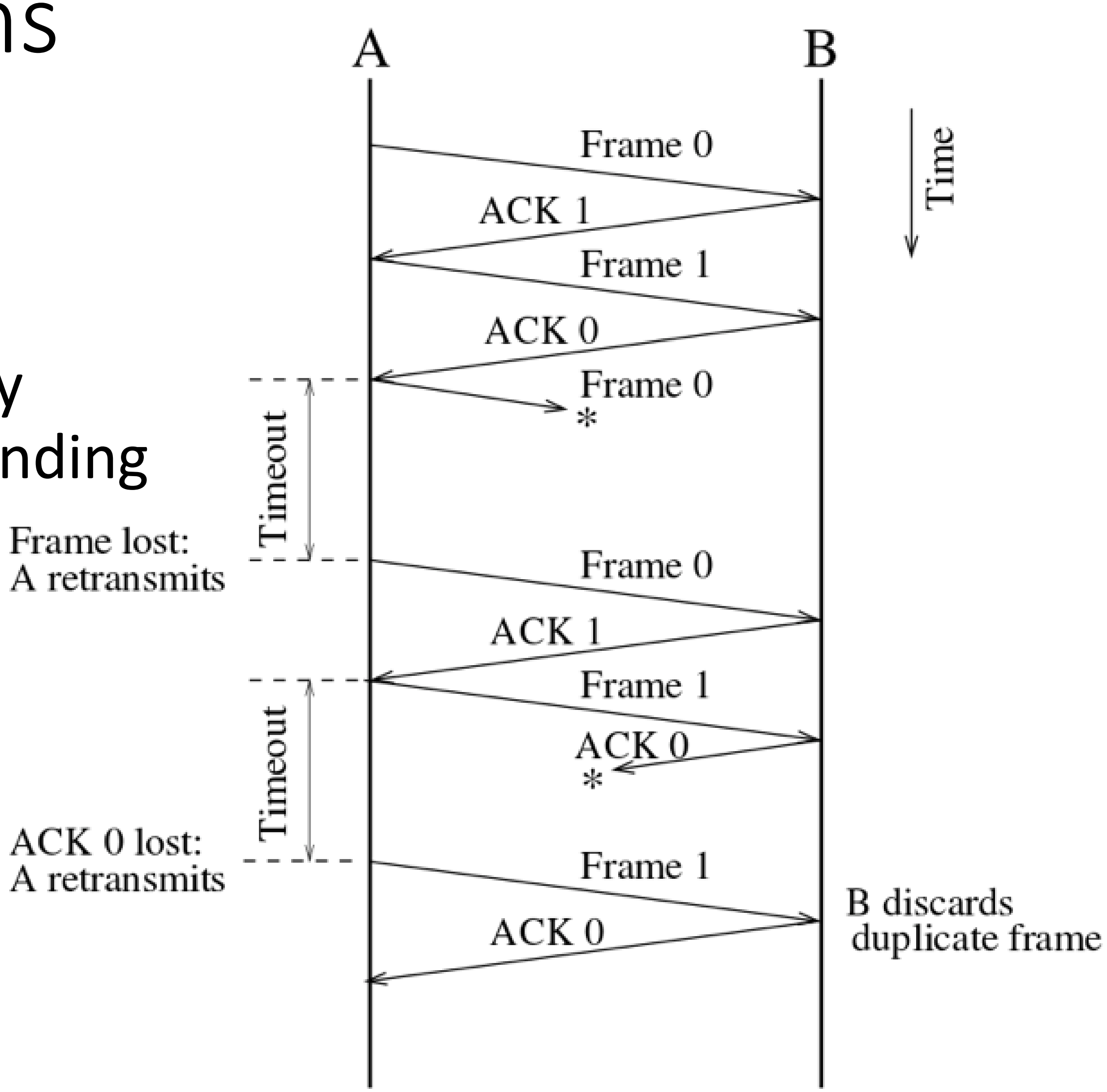


Flow Control Algorithms

- **Stop and Wait (Noisy Channel):**

- Use sequence numbers to individually identify each frame and the corresponding acknowledgement

- **Automatic Repeat Request (ARQ)**



Flow Control Algorithms

- Three things to consider when designing a reliable flow control algorithm
    - Sequence Numbers – for distinguishing older packets and new ones (both data as well as ACKs).
    - Receiver Feedback – using ACKs and NAKs
    - Retransmission – after a fixed amount of Time, in case of lost data or ACKs

Stop-and-Wait ARQ:

- When a frame is sent, the sender starts the timeout counter.
- If acknowledgement of frame comes in time, the sender transmits the next frame in queue.
- If acknowledgement does not come in time, the sender assumes that either the frame or its acknowledgement is lost in transit. Sender retransmits the frame and starts the timeout counter.
- If a negative acknowledgement is received, the sender retransmits the frame.

Problem with Stop and Wait

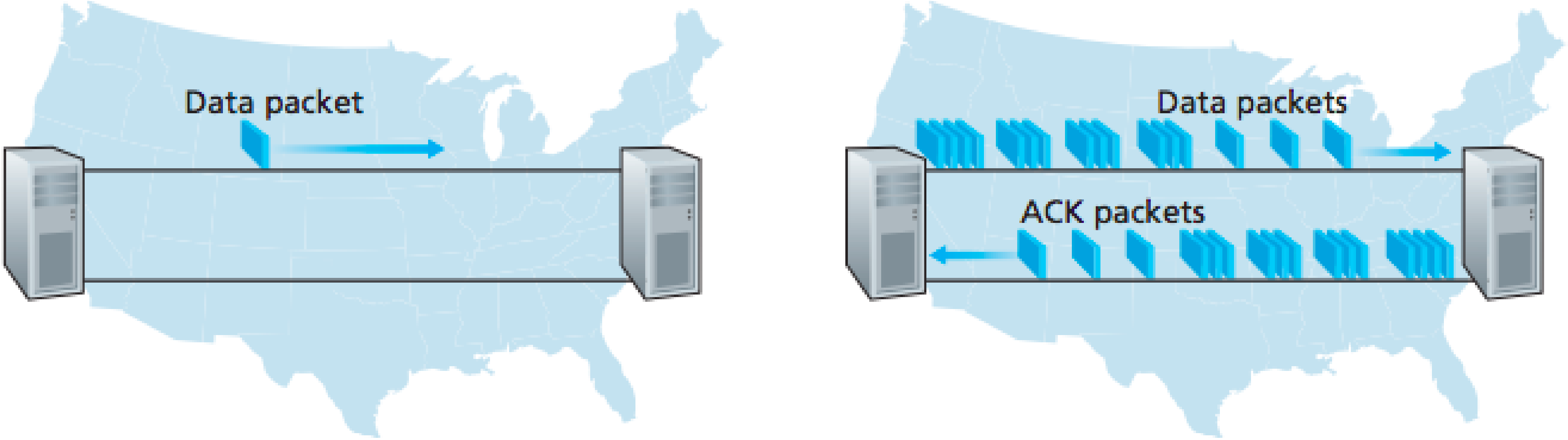
- Every packet needs to wait for the acknowledgement of the previous packet.

- 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

Source: Computer Networks, Kurose, Ross

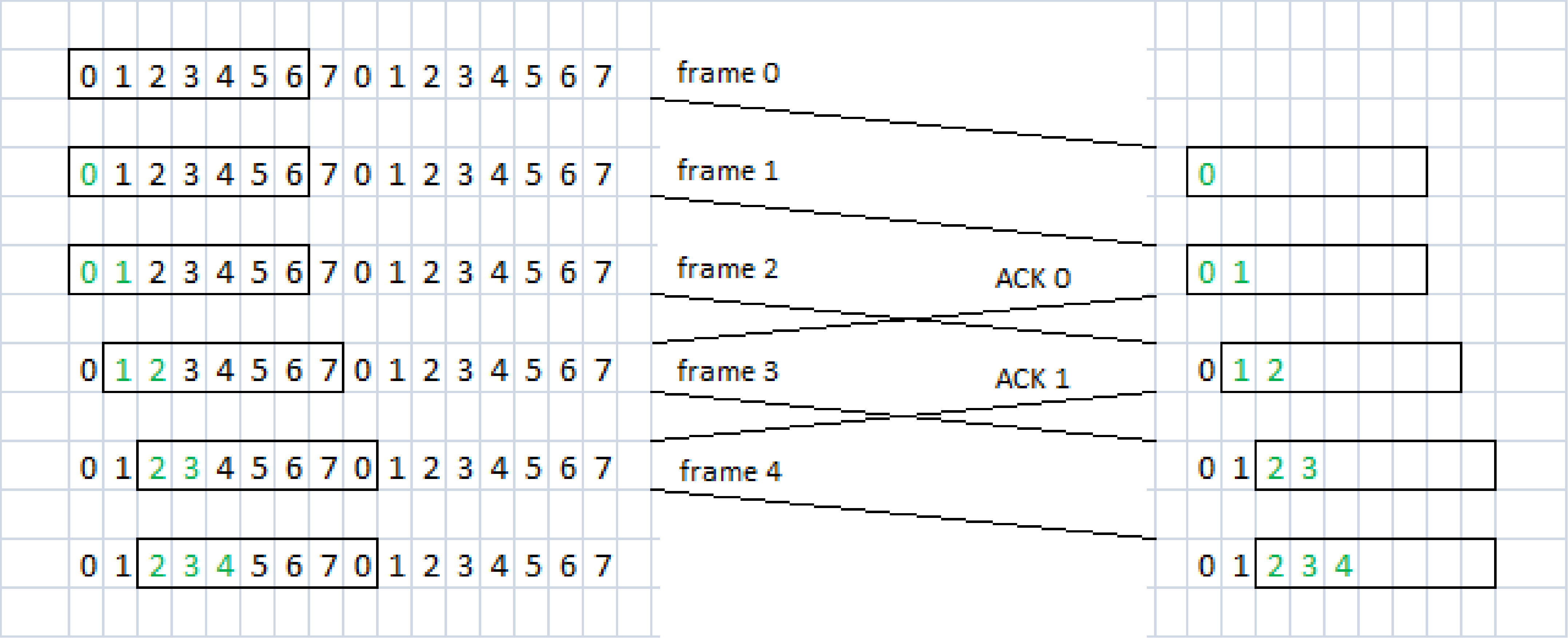
Sliding Window Protocols

- 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

Source: http://ironbark.xtelco.com.au/subjects/DC/lectures/13/

Sliding Window for a 3 bit Sequence Number

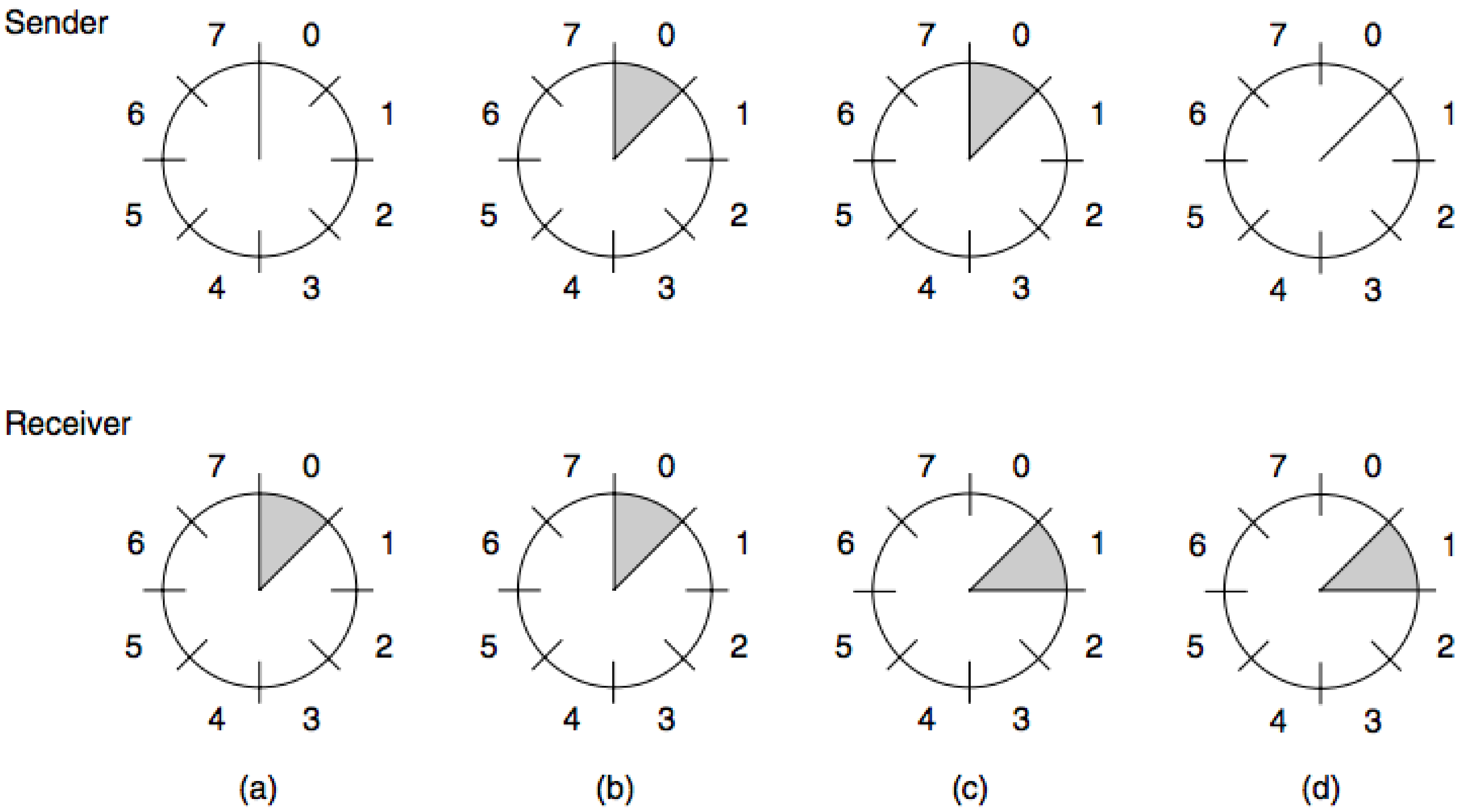


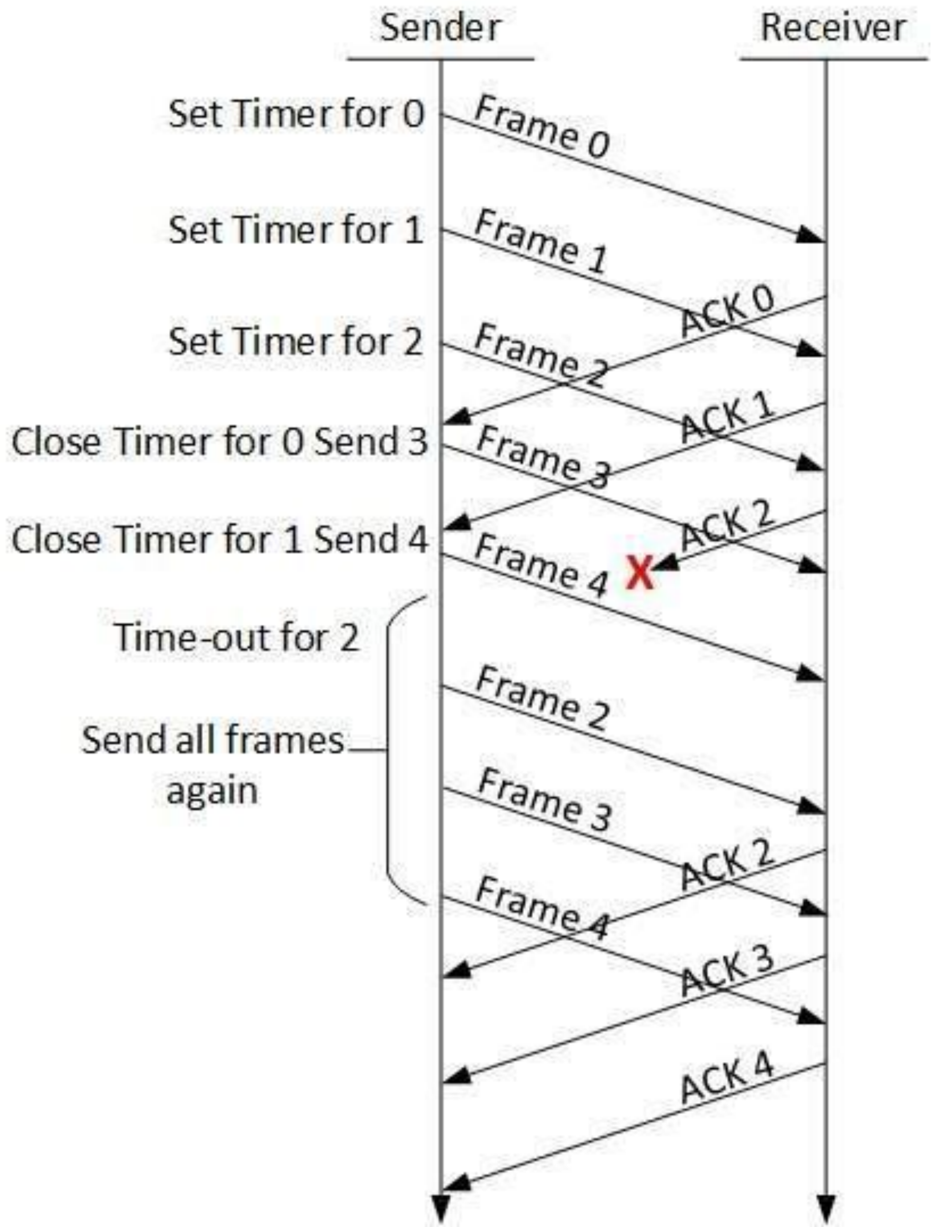
Figure 3-15. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

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

Sliding Window Protocols in Noisy Channels

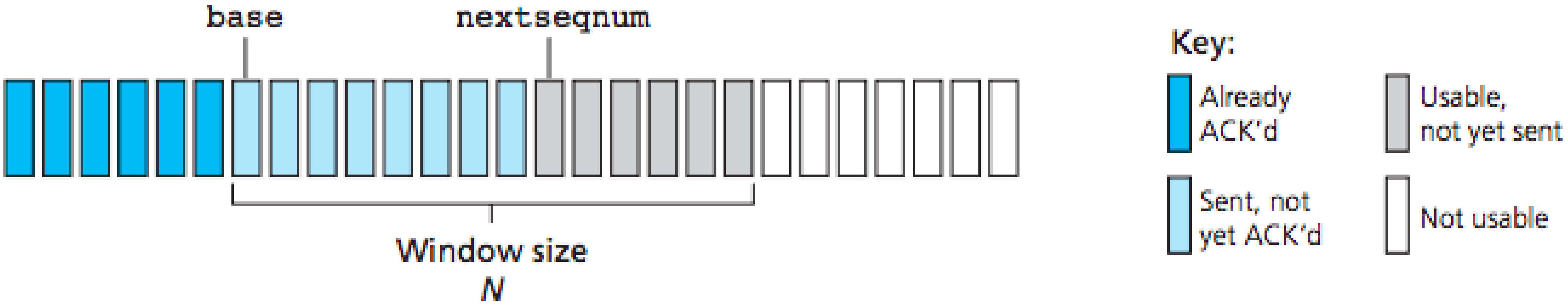
- **A timeout occurs if a segment (or the acknowledgment) gets lost**
- **How does the flow and error control protocol handle a timeout?**
- **Go Back N ARQ:** If segment N is lost, all the segments from segment N are retransmitted
- **Selective Repeat (SR) ARQ:** Only the lost packets are selectively 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

Go Back N ARQ – Sender Window Control



Source: Computer Networks, Kurose, Ross

Go Back N ARQ – A Bound on Window Size

- **Outstanding Frames –** Frames that have been transmitted, but not yet acknowledged

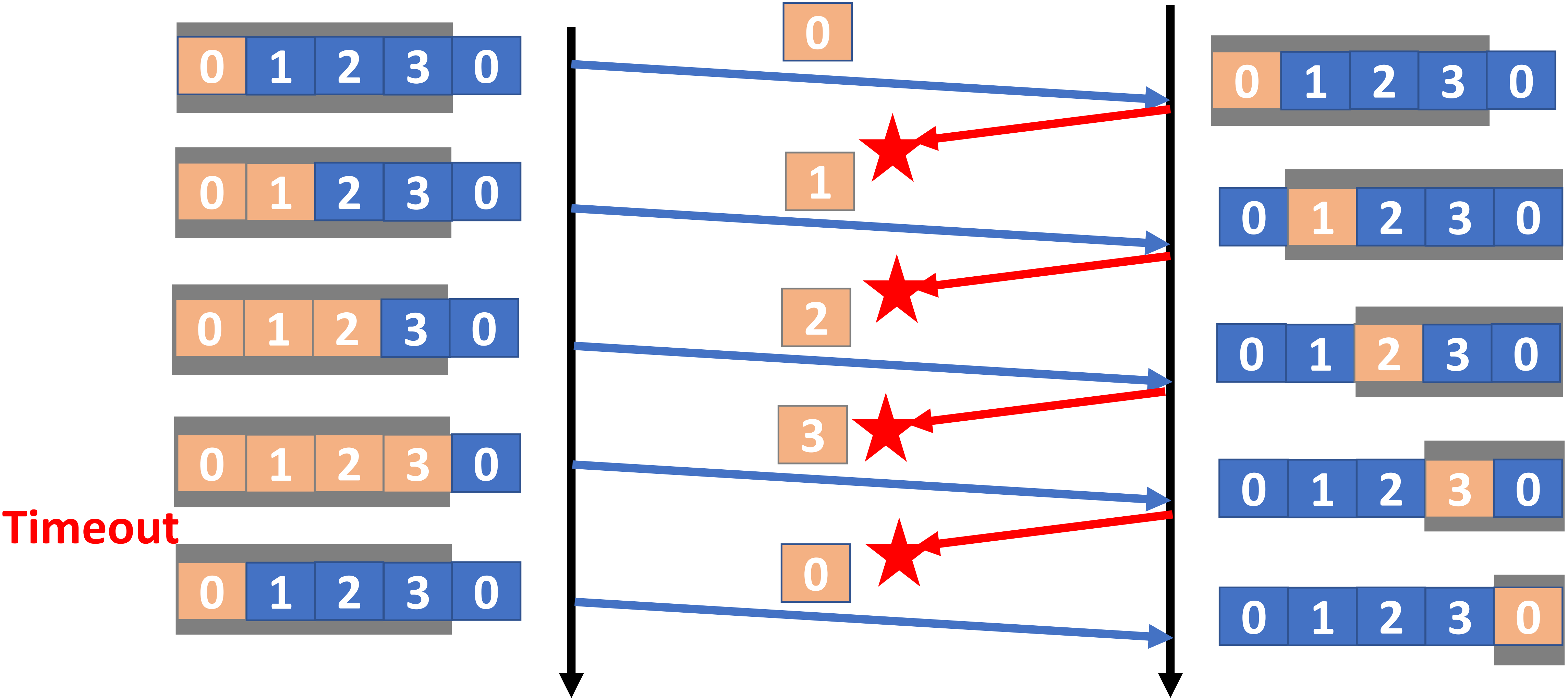
- **Maximum Sequence Number (MAX_SEQ): MAX_SEQ+1** distinct sequence numbers are there
  - **0,1,…,MAX_SEQ**

- **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)**

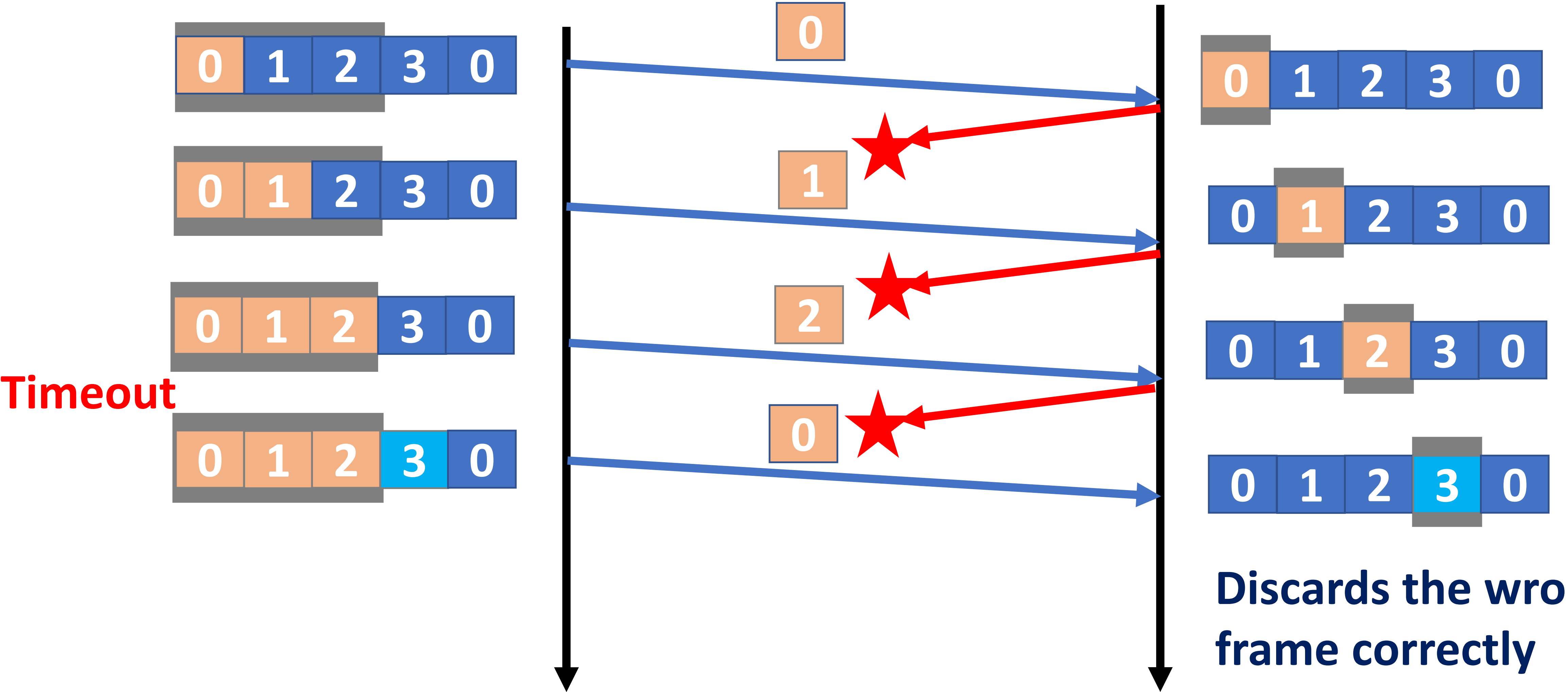
Go Back N ARQ – A Bound on Window Size

- **Let MAX_SEQ = 3, Window Size = 4**

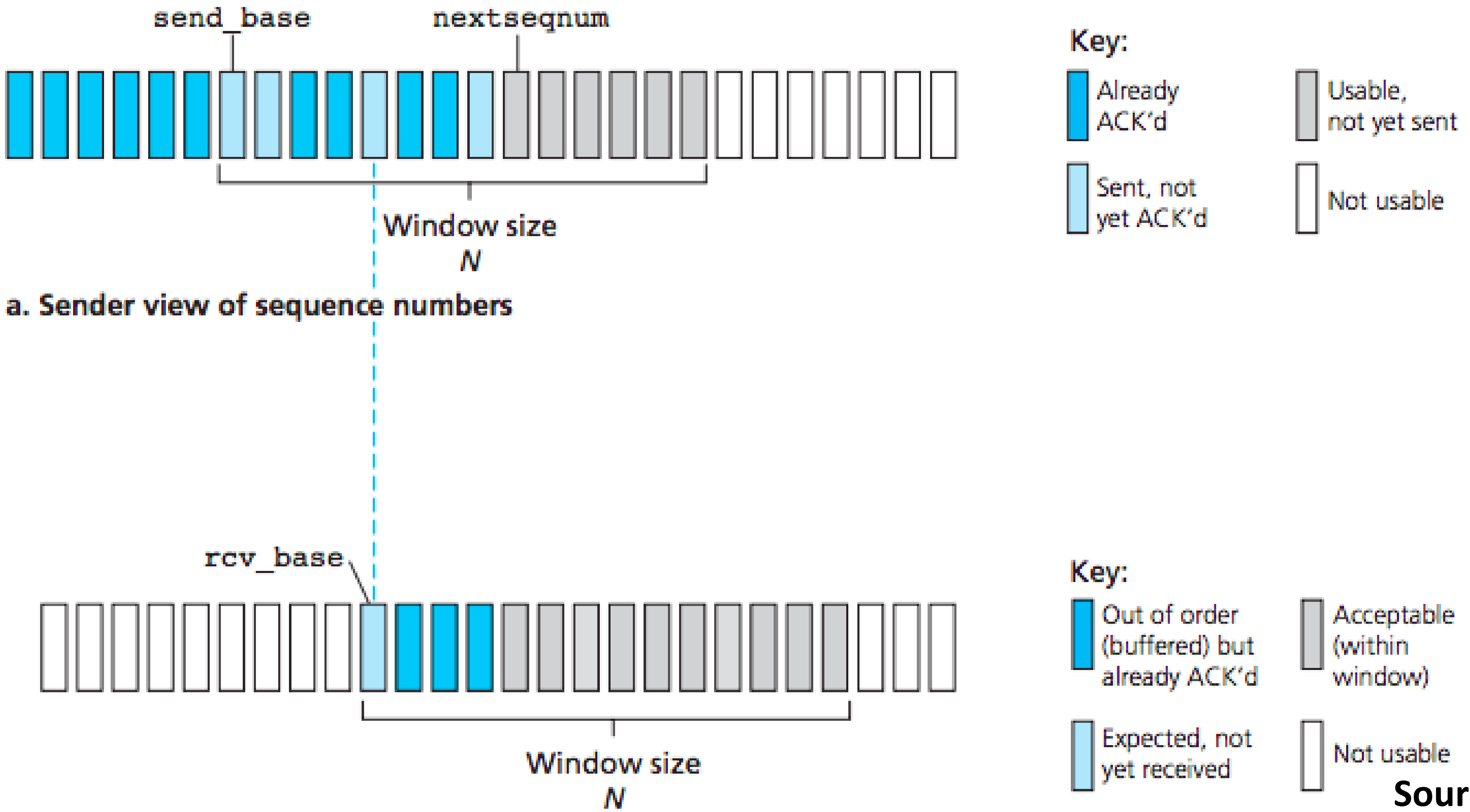


Go Back N ARQ – A Bound on Window Size

- **Let MAX_SEQ = 3, Window Size = 3**



Selective Repeat (SR) – Window Control

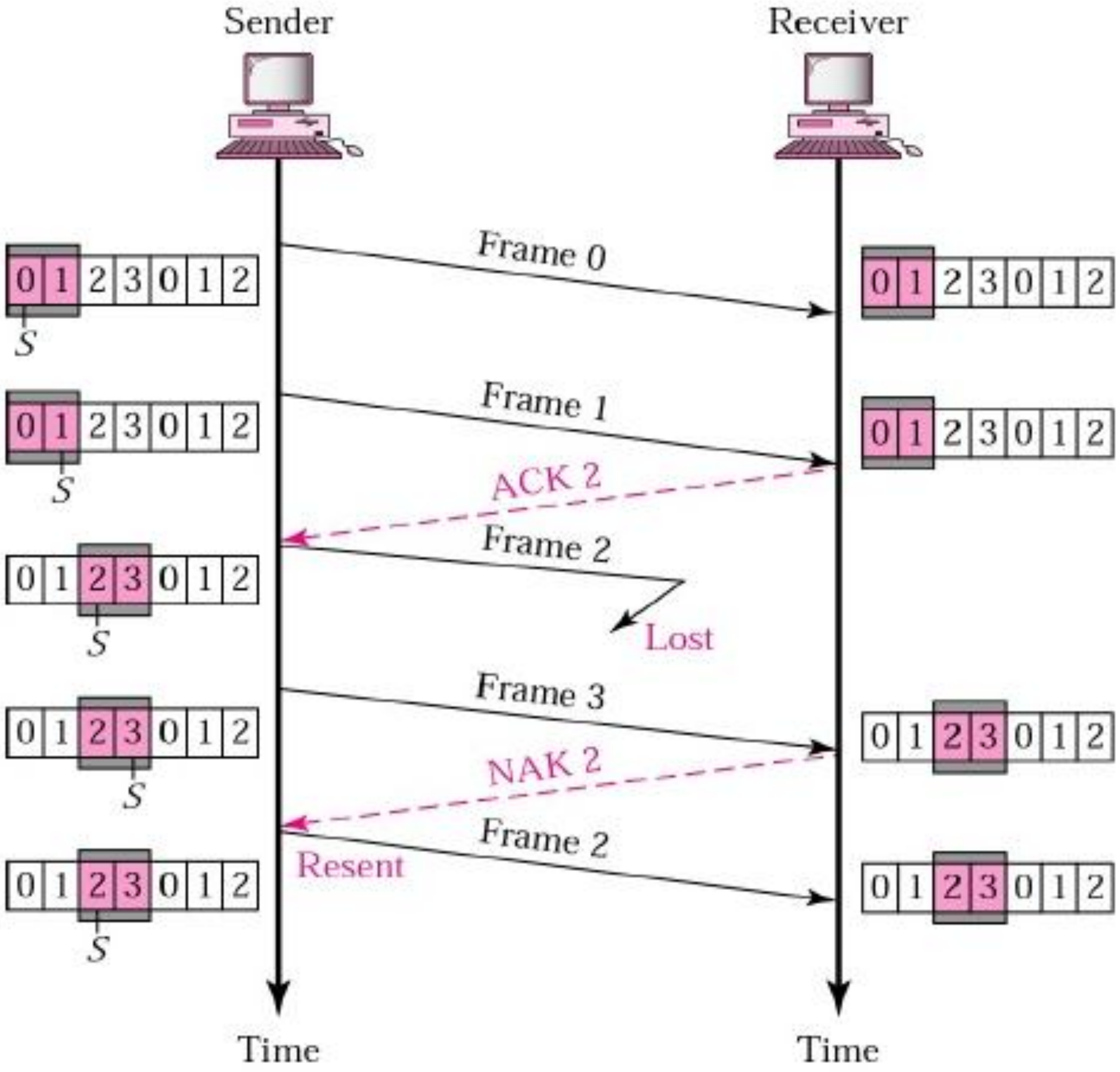


a. Sender view of sequence numbers

b. Receiver view of sequence numbers

Source: Computer Networks, Kurose, Ross

Selective Repeat ARQ



Selective Repeat – A Bound on Window Size

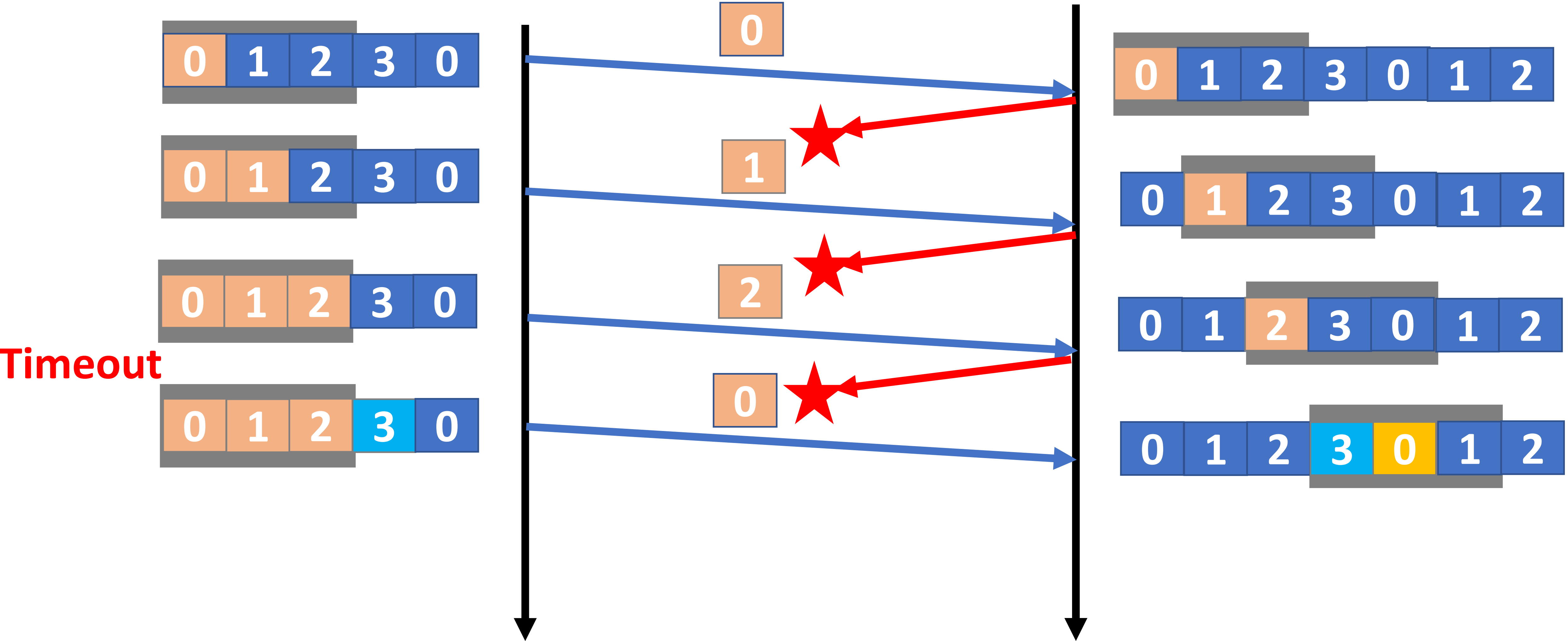
- **Maximum Sequence Number (MAX_SEQ): MAX_SEQ+1** distinct sequence numbers are there
  - **0,1,…,MAX_SEQ**

- **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

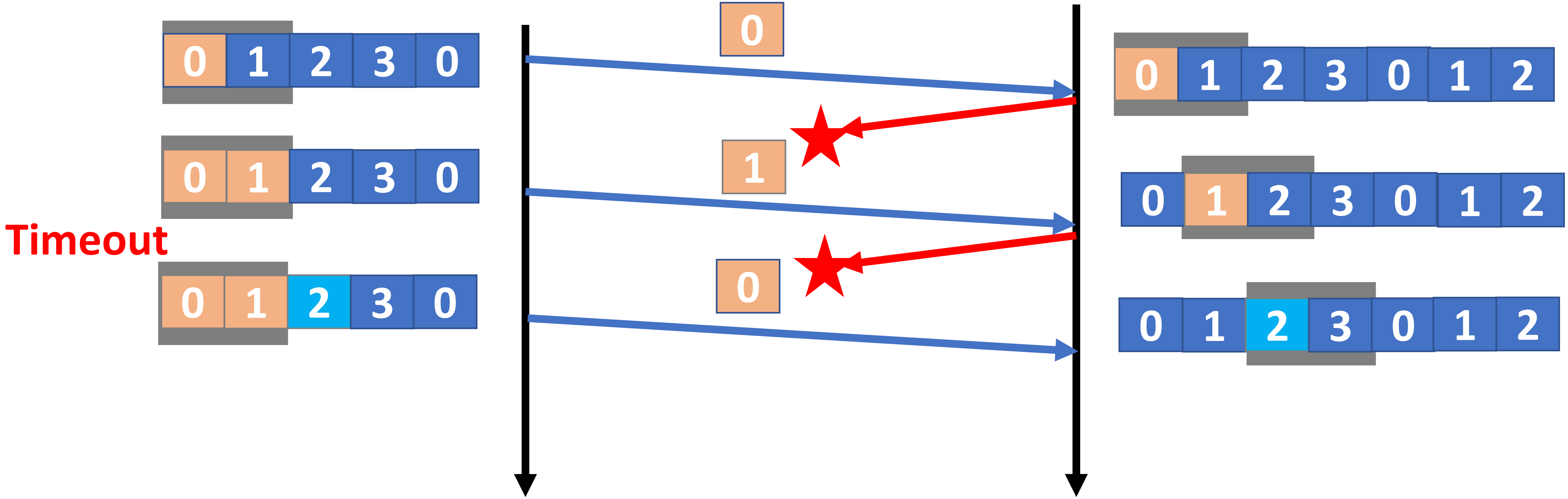
Selective Repeat – A Bound on Window Size

- **Let MAX_SEQ = 3, Window Size = 3 [(MAX_SEQ+1)/2+1]**



Selective Repeat – A Bound on Window Size

- **Let MAX_SEQ = 3, Window Size = 2**



Discards the wrong frame correctly

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 x 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 (~1000 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 (~1000 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**

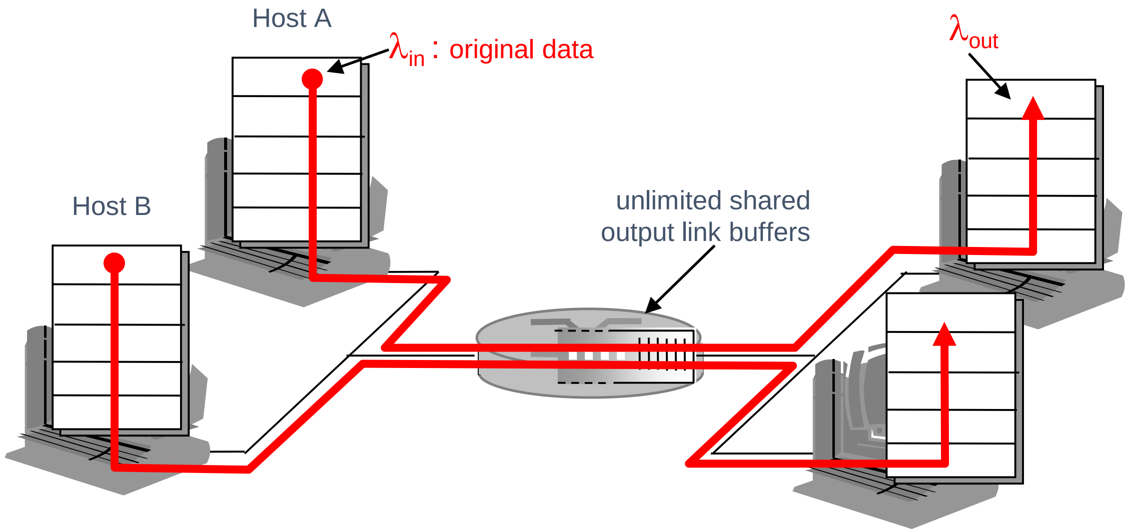
Principles of Congestion Control

Congestion:

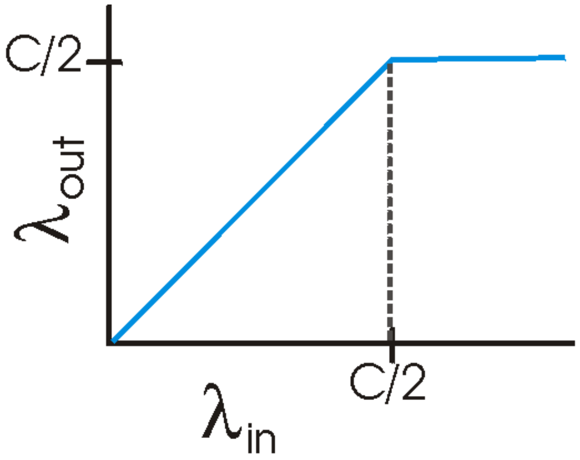
- informally: “too many sources sending too much data too fast for *network* to handle”
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)

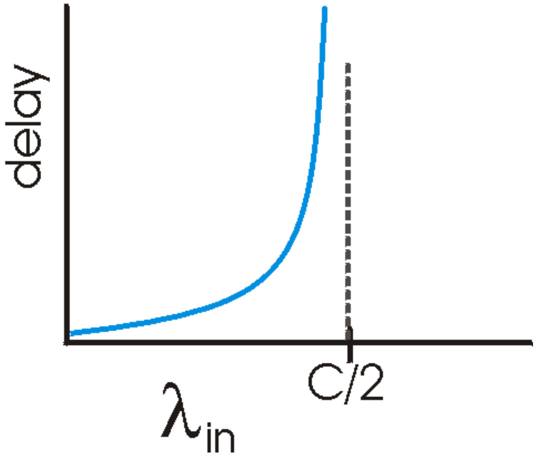
Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- no retransmission



Host A
λ_{in} : original data
λ_{out}
Host B
unlimited shared output link buffers

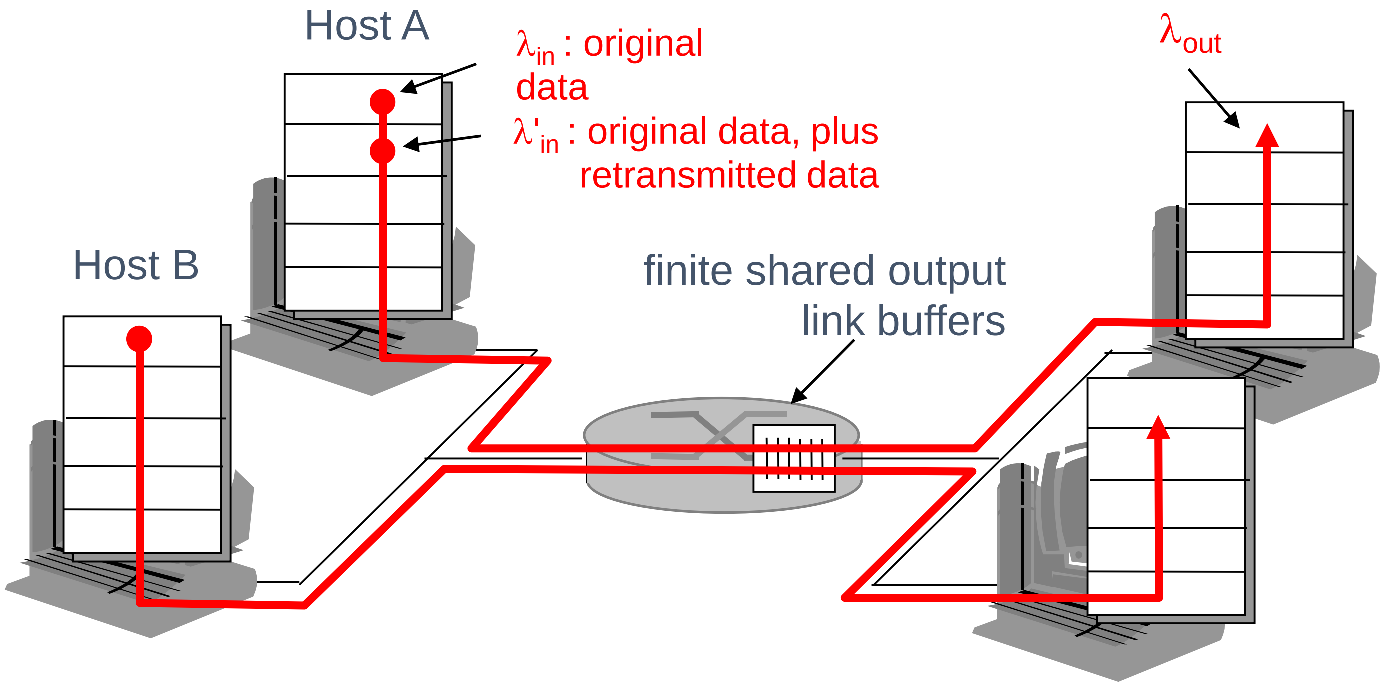




- large delays when congested
- Because link capacity is finite and limited
- In such a case, we can maximum achievable throughput

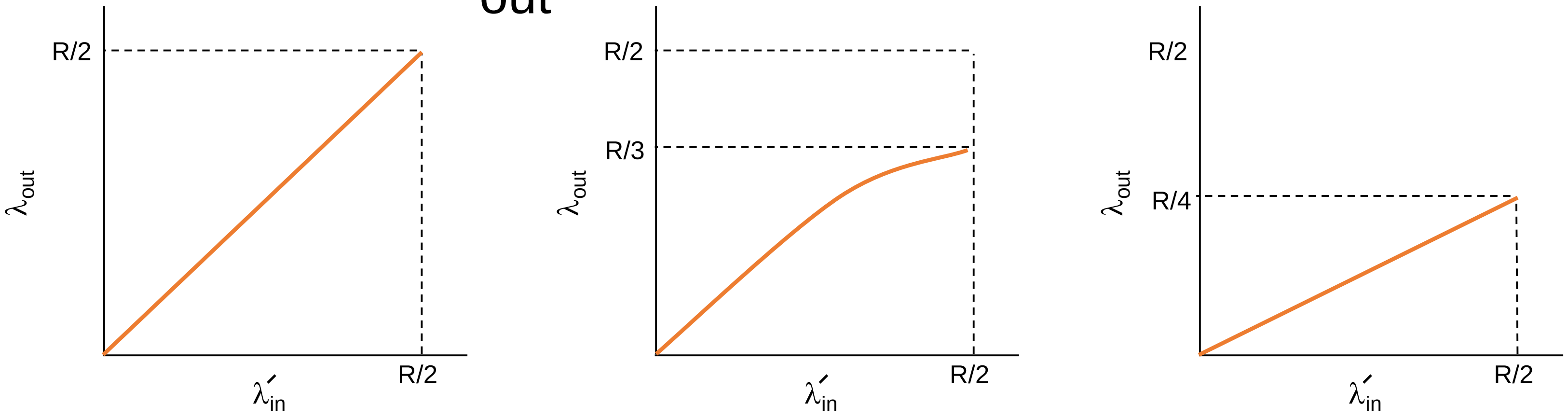
Causes/costs of congestion: scenario 2

- one router, *finite* buffers
- sender retransmission of lost packet


Host A
Host B
λ_{in} : original data
λ'_{in} : original data, plus retransmitted data
finite shared output link buffers
λ_{out}

Causes/costs of congestion: scenario 2

- always: $\lambda_{in} = \lambda_{out}$(goodput)
- “perfect” or ideal retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- retransmission of delayed (not lost) packet makes λ'_{in} larger (than perfect case) for same λ_{out}



a. b. c.

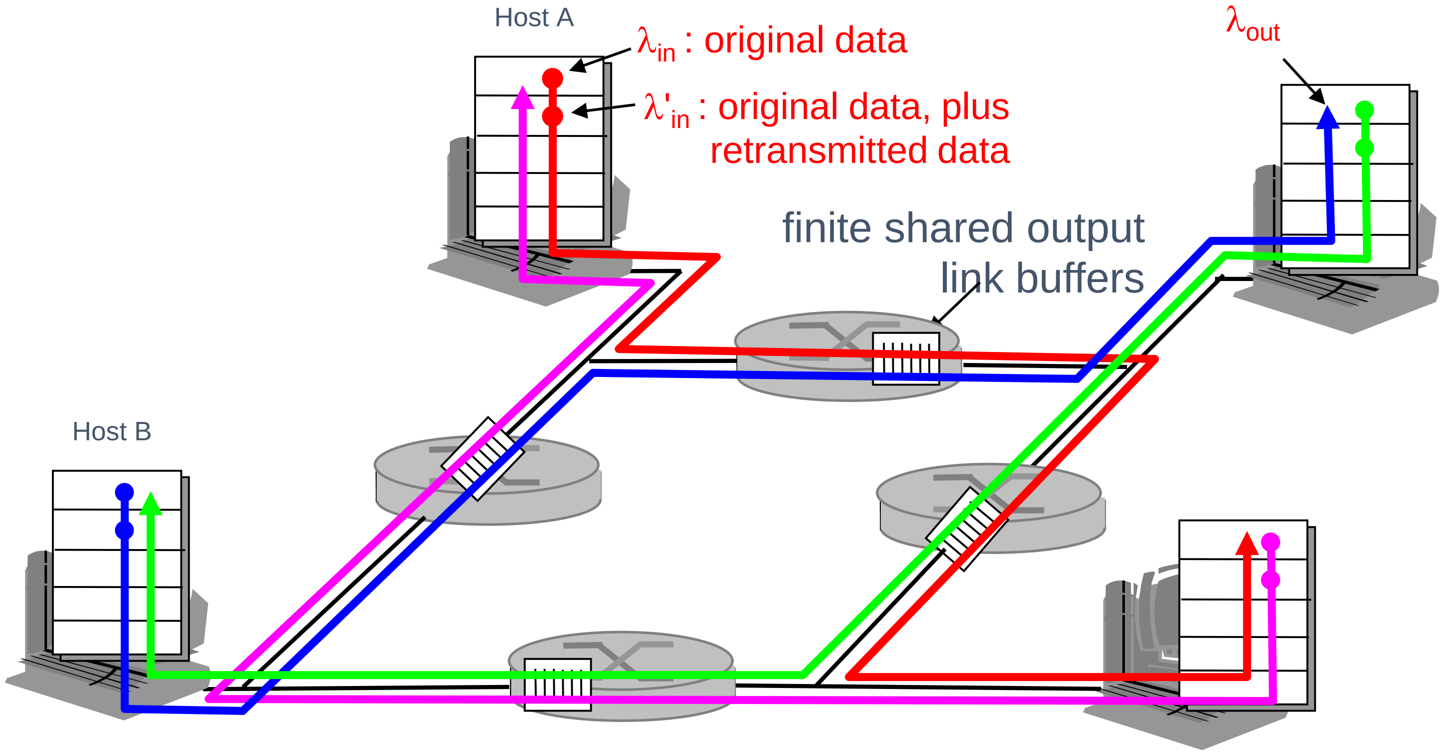
“costs” of congestion:

- ❐ more work (retrans) for given “goodput”
- ❐ unneeded retransmissions: link carries multiple copies of pkt

Causes/costs of congestion: scenario 3

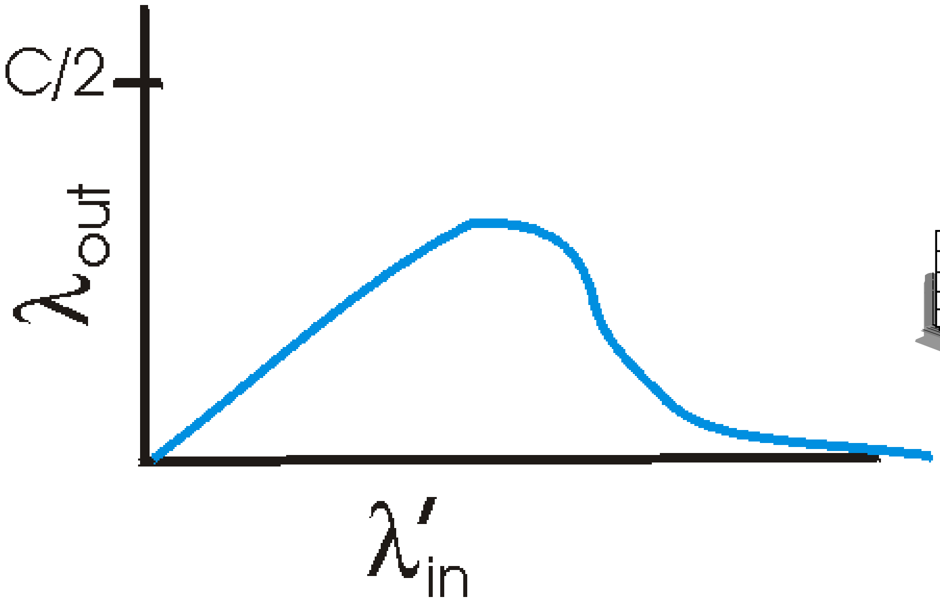
- four senders
- multihop paths
- timeout/retransmit

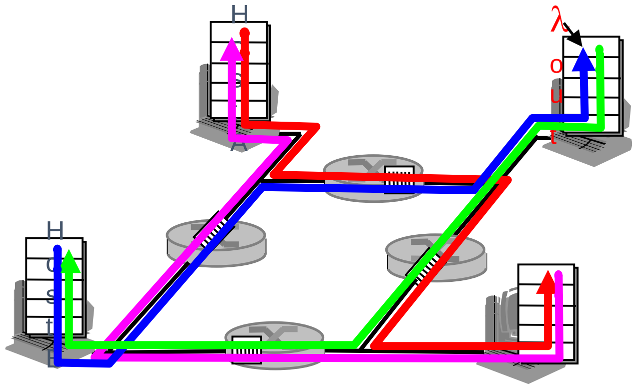
Q: what happens as λ_{in} and λ'_{in} increase ?



Host A
λ_{in} : original data
λ'_{in} : original data, plus retransmitted data
λ_{out}
finite shared output link buffers
Host B

Causes/costs of congestion: scenario 3





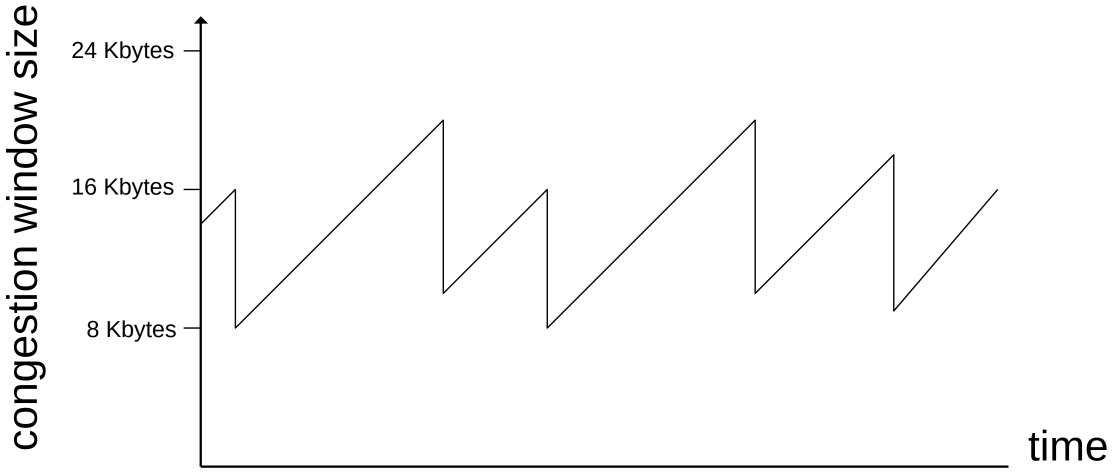
Another "cost" of congestion:

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

TCP congestion control: additive increase, multiplicative decrease

- *Approach:* increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **CongWin** by 1 MSS every RTT until loss detected
  - *multiplicative decrease*: cut **CongWin** in half after loss

Saw tooth behavior: probing for bandwidth



AIMD – Efficient and Fair Operating Point for Congestion Control

- **Additive Increase Multiplicative Decrease (AIMD)** – Chiu and Jain (1989)

- Let *w(t)* be the sending rate. *a (a > 0)* is the additive increase factor, and *b (0<b<1)* is the multiplicative decrease factor

$$w(t+1) = \begin{cases} w(t) + a & \text{if congestion is not detected} \\ w(t) \times b & \text{if congestion is detected} \end{cases}$$

TCP Congestion Control

- Based on implementation of AIMD using a window and with packet loss as the binary signal

- TCP maintains a **Congestion Window (CWnd) –** number of bytes the sender may have in the network at any time

- **Sending Rate = Congestion Window / RTT**

- **Sender Window (SWnd) = Min (CWnd, RWnd)**

- RWnd – Receiver advertised window size

TCP Congestion Control

- How to Implement AIMD in real world scenario

- 3 Steps:
  - Slow Start (Exponential growth)
  - Additive Increase (Congestion Avoidance)
  - Fast Retransmit and Recovery (Once congestion is detected)

Increase Rate Exponentially at the Beginning – The Slow Start

- AIMD rule will take a very long time to reach a good operating point on fast networks if the CWnd is started from a small size.

- A 10 Mbps link with 100 ms RTT
  - Appropriate CWnd = BDP = 1 Mbit
  - 1250 byte packets -> 100 packets to reach BDP
  - CWnd starts at 1 packet, and increased 1 packet at every RTT
  - 100 RTTs are required 10 sec before the connection reaches to a moderate rate

- **Slow Start - Exponential increase of rate to avoid slow convergence**
  - **Rate is not slow at all ! ⍰**
  - **CWnd is doubled at every RTT**

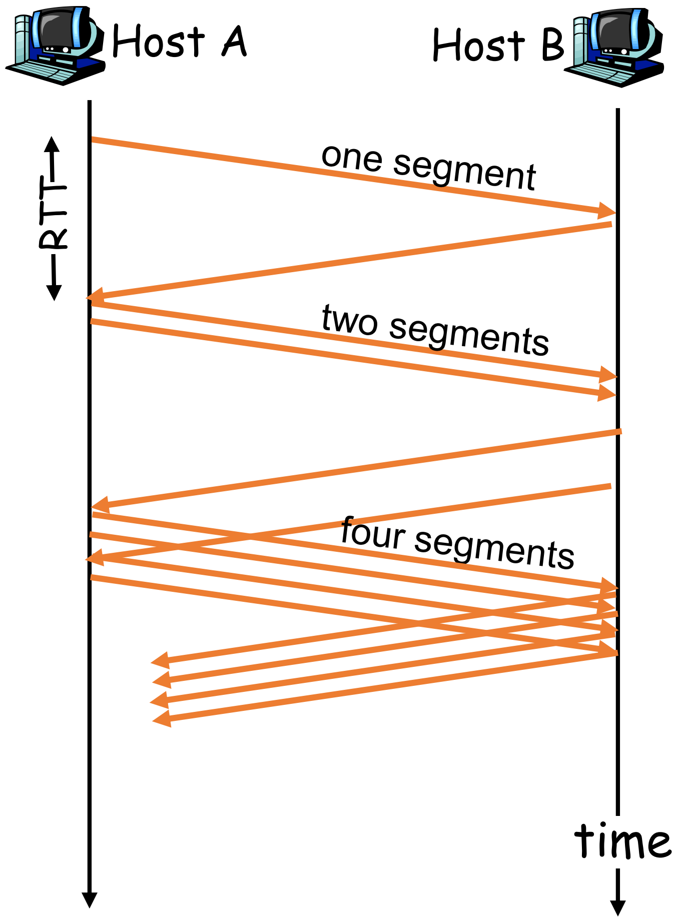
TCP Slow Start

- When connection begins, **`CongWin`** = 1 MSS
    - Example: MSS = 500 bytes & RTT = 200 msec
    - initial rate = 20 kbps
- available bandwidth may be >> MSS/RTT
    - desirable to quickly ramp up to respectable rate

- When connection begins, increase rate exponentially fast until first loss event

TCP Slow Start (more)

- When connection begins, increase rate exponentially until first loss event:
  - double **CongWin** every RTT
  - done by incrementing **CongWin** for every ACK received
- Summary: initial rate is slow but ramps up exponentially fast



Slow Start Threshold

- Slow start causes exponential growth, eventually it will send too many packets into the network too quickly.

- To keep slow start under control, the sender keeps a threshold for the connection called the **slow start threshold (ssthresh).**

- Initially **ssthresh** is set to BDP (or arbitrarily high), the maximum that a flow can push to the network.

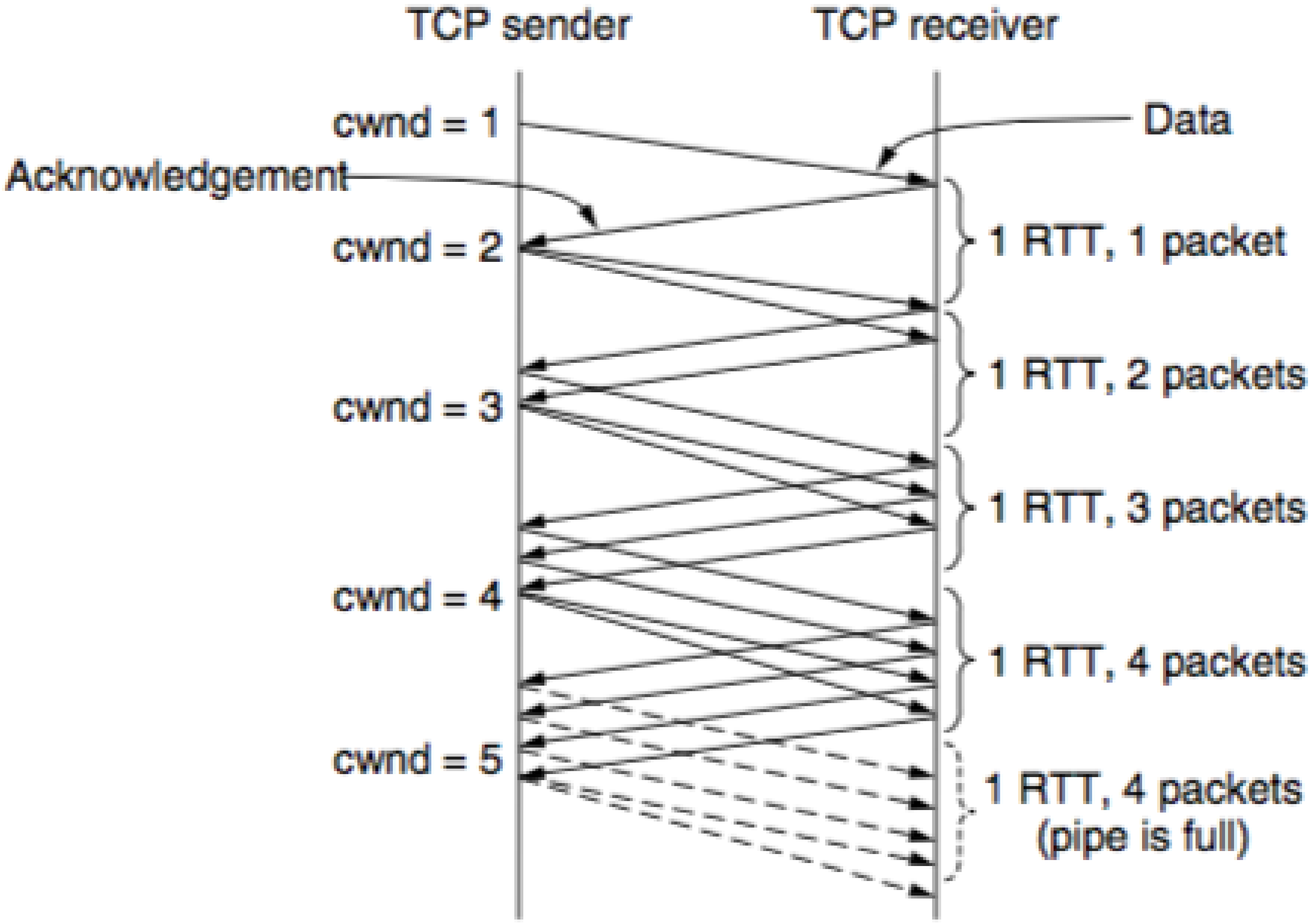
- Whenever a packet loss is detected by a Retransmission Time Out (RTO) or Duplicate ACK (DUPACK), the **ssthresh** is set to be half of the congestion window

Additive Increase (Congestion Avoidance)

- Whenever ssthresh is crossed, TCP switches from slow start to additive increase.
- Usually implemented with an partial increase for every segment that is acknowledged, rather than an increase of one segment per RTT.
- A common approximation is to increase Cwnd for additive increase as follows:

$$\boldsymbol{CWnd = Cwnd + \frac{MSS \times MSS}{CWnd}}$$

Additive Increase – Packet Wise Approximation



Triggering an Congestion

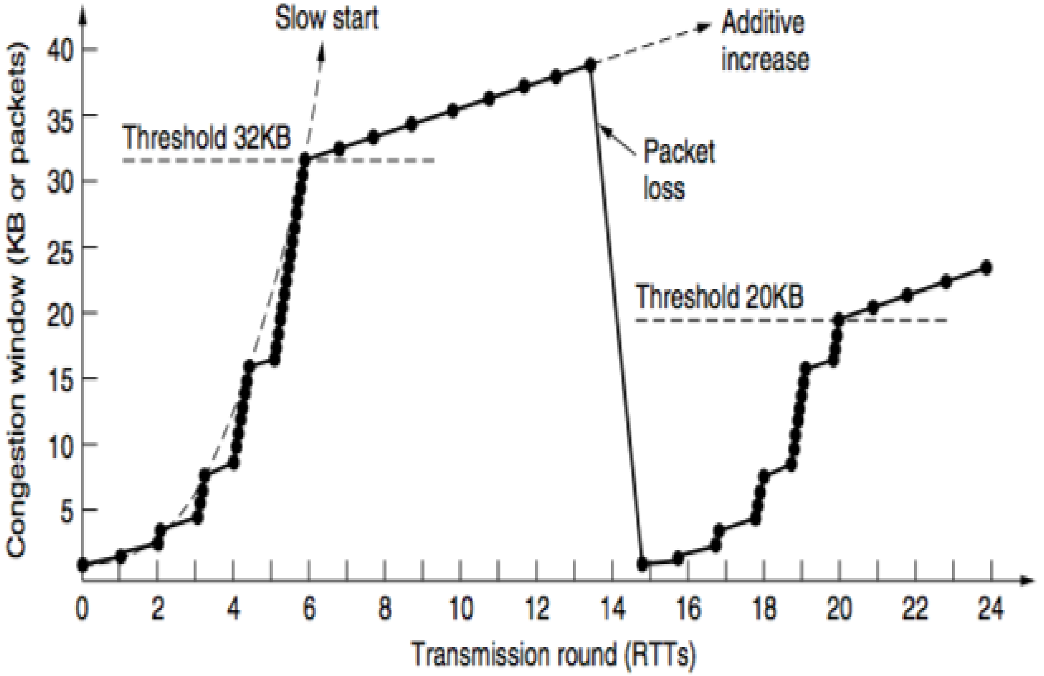
- Two ways to trigger a congestion notification in TCP – (1) RTO, (2) Duplicate ACK

- **RTO**: A sure indication of congestion, however time consuming

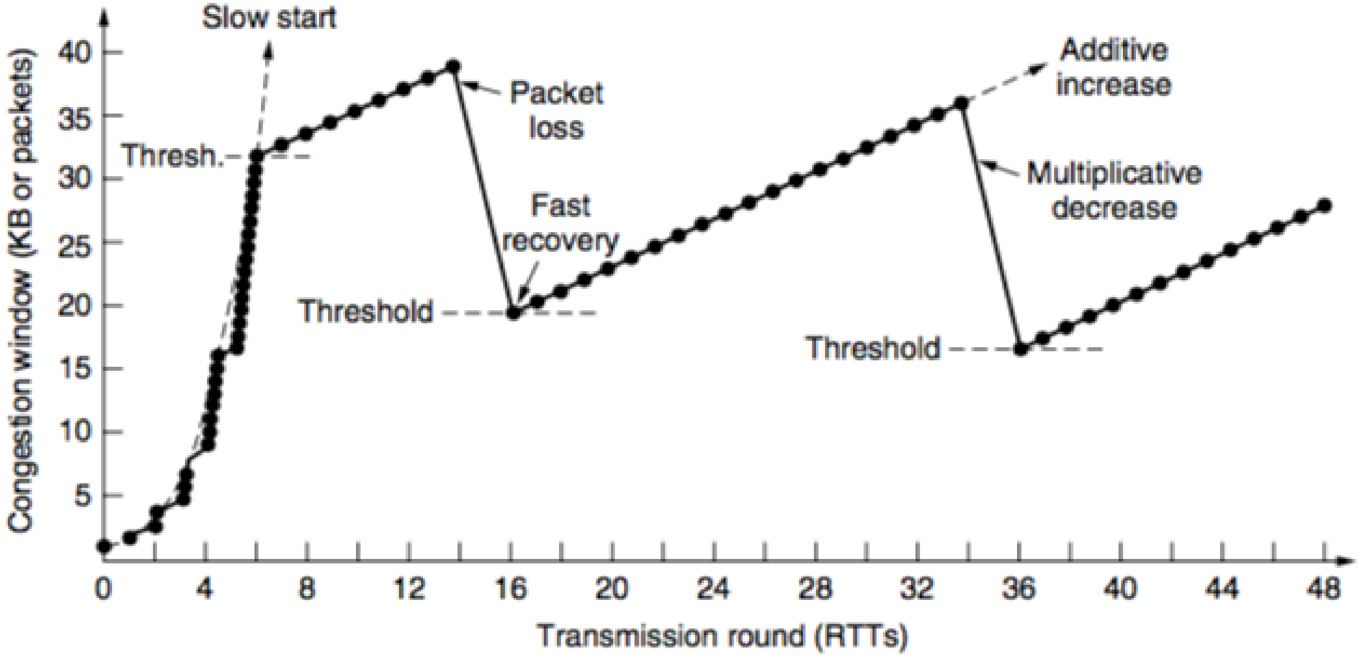
- **Duplicate ACK:** Receiver sends a duplicate ACK when it receives out of order segment
  - **A loose way of indicating congestion**
  - **TCP arbitrarily assumes that THREE duplicate ACKs (DUPACKs) imply that a packet has been lost – triggers congestion control mechanism**
  - The identity of the lost packet can be inferred – **the very next packet in sequence**
  - **Retransmit the lost packet and trigger congestion control**

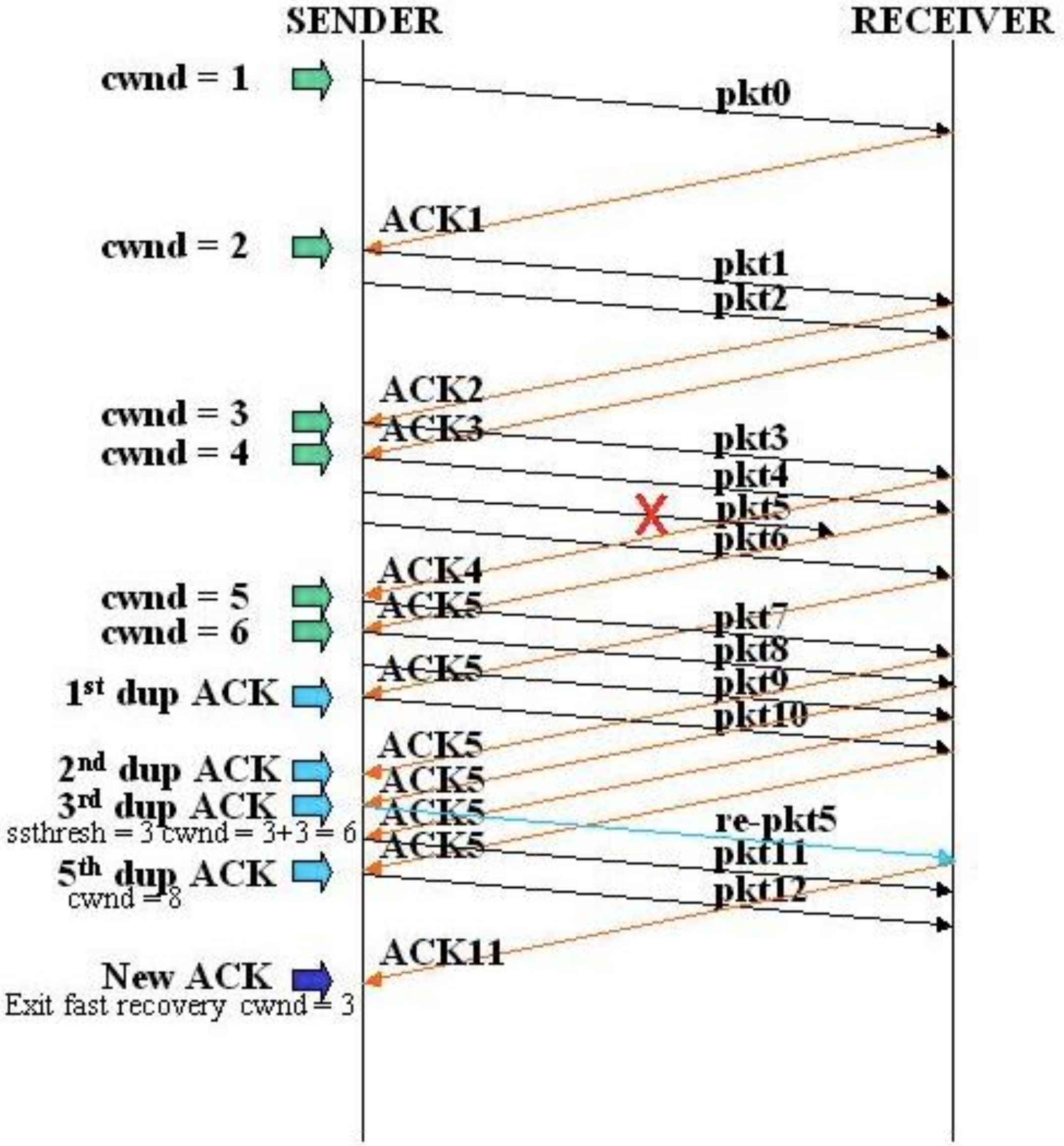
Fast Retransmission – TCP Tahoe

- Use THREE DUPACK as the sign of congestion
- Once 3 DUPACKs have been received,
  - Retransmit the lost packet (**fast retransmission)** – takes one RTT
  - Set ssthresh as half of the current CWnd
  - Set CWnd to 1 MSS



Fast Recovery – TCP Reno





Fast Recovery – TCP Reno

- **Fast recovery:**
  1. set ssthresh to one-half of the current congestion window. Retransmit the missing segment.
  2. set cwnd = ssthresh + 3.
  3. Each time another duplicate ACK arrives, set cwnd = cwnd + 1. Then, send a new data segment if allowed by the value of cwnd.
  4. Once receive a new ACK (an ACK which acknowledges all intermediate segments sent between the lost packet and the receipt of the first duplicate ACK), exit fast recovery. This causes setting cwnd to ssthresh (the ssthresh in step 1). Then, continue with linear increasing due to congestion avoidance algorithm.

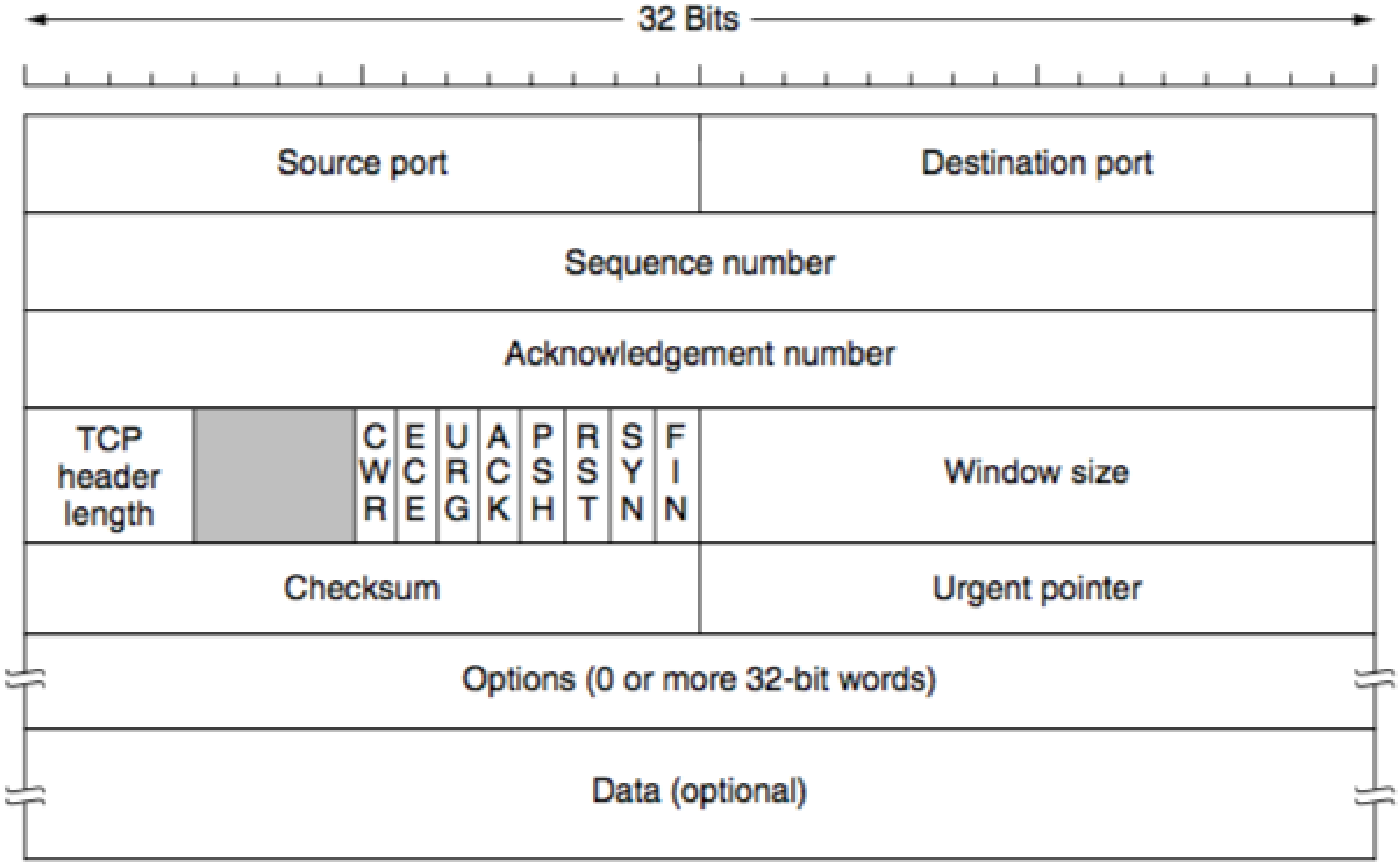
Fast Recovery – TCP Reno

- **Once a congestion is detected through 3 DUPACKs, do TCP really need to set CWnd = 1 MSS ?**

- DUPACK means that **some segments are still flowing in the network** – a signal for temporary congestion, but not a prolonged one

- Immediately transmit the lost segment (**fast retransmit)**, then transmit additional segments based on the DUPACKs received **(fast recovery)**

The TCP Protocol – The Header



32 Bits

Source port | Destination port

Sequence number

Acknowledgement number

TCP header length | | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window size

Checksum | Urgent pointer

Options (0 or more 32-bit words)

Data (optional)

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

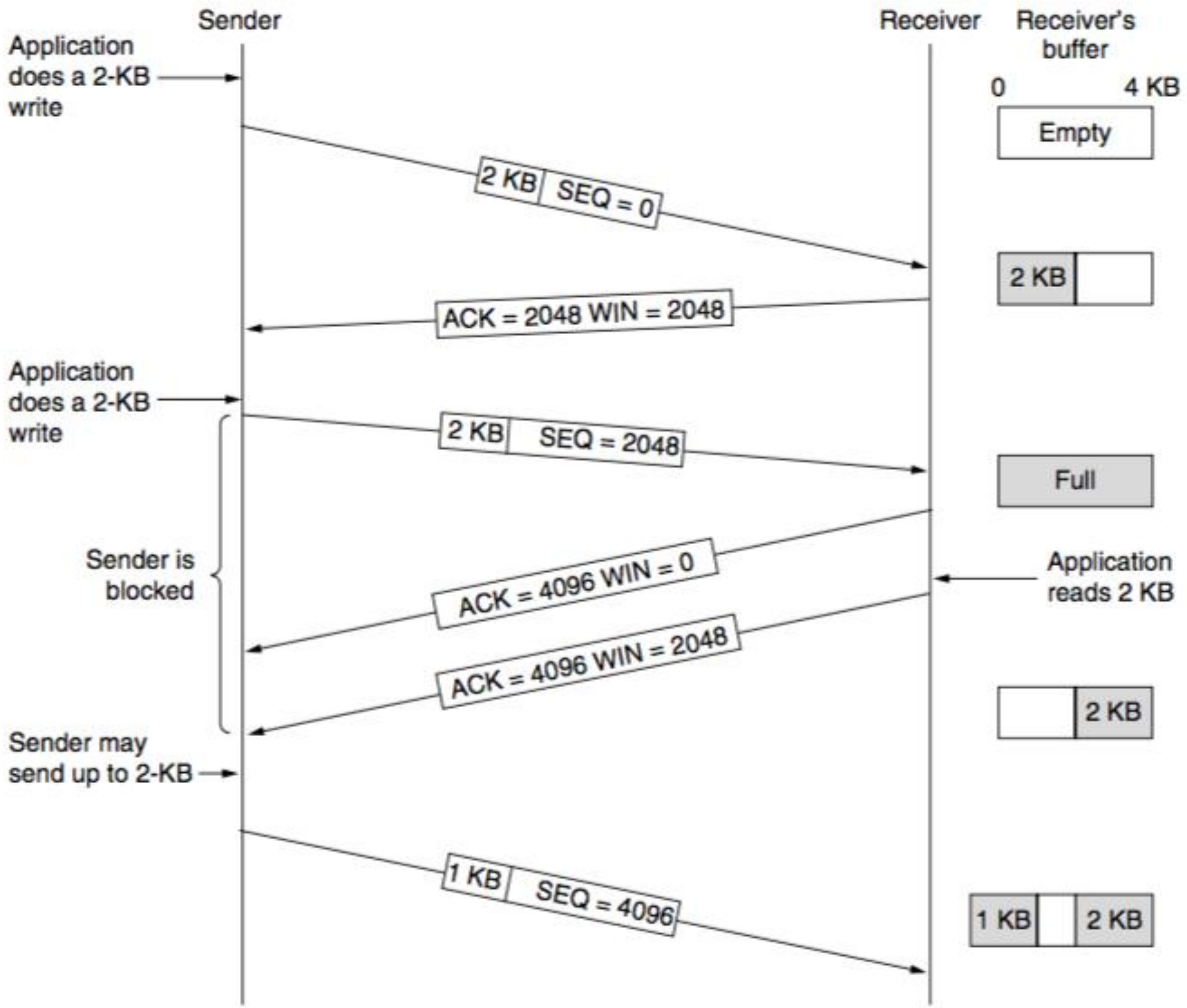
TCP Sequence Number and Acknowledgement Number

- 32 bits sequence number and acknowledgement number

- Every byte on a TCP connection has its own 32 bit sequence number – a **byte stream** oriented connection

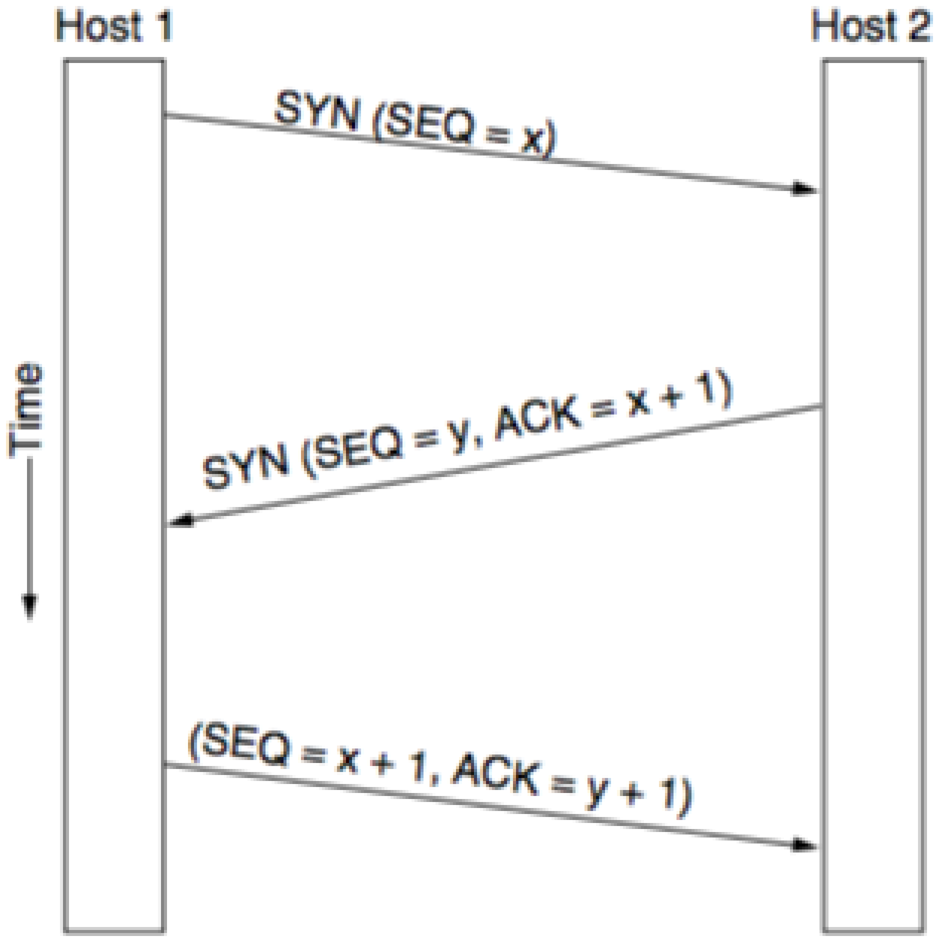
- TCP uses sliding window based flow control – the acknowledgement number contains next expected byte in order, which acknowledges the **cumulative bytes** that has been received by the receiver.
  - ACK number 31245 means that the receiver has correctly received up to 31244 bytes and expecting for byte 31245

TCP Sliding Window

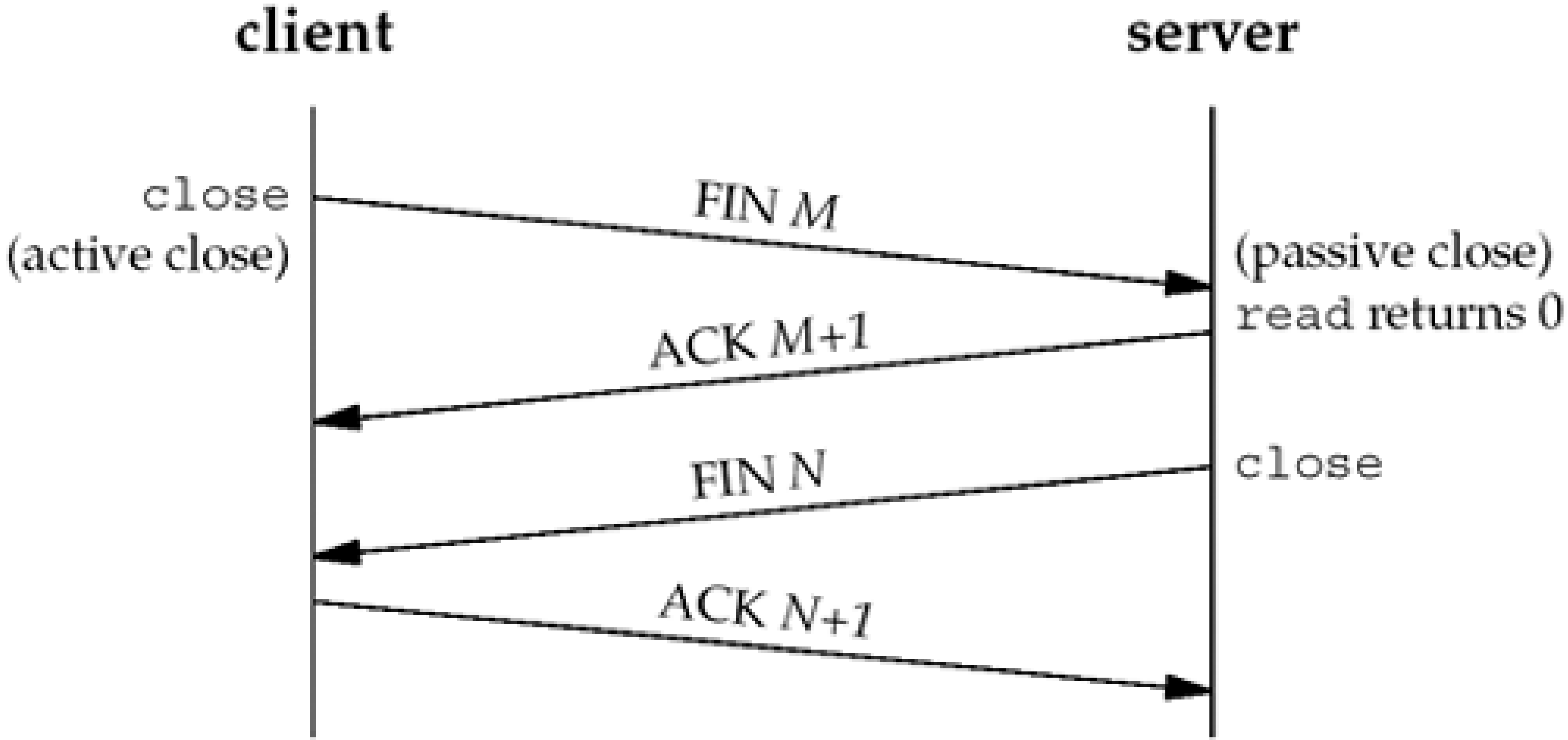


Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

TCP Connection Establishment



TCP Connection Release



Summary: TCP Congestion Control

- When **CongWin** is below **Threshold**, sender in slow-start phase, window grows exponentially.
- When **CongWin** is above **Threshold**, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs, **Threshold** set to **CongWin/2** and **CongWin** set to **Threshold**.
- When timeout occurs, **Threshold** set to **CongWin/2** and **CongWin** is set to 1 MSS.

TCP sender congestion control

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to “Congestion Avoidance”	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT

SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2,
		CongWin = Threshold,
		Set state to “Congestion Avoidance”
			Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.

SS or CA	Timeout	Threshold = CongWin/2,
		CongWin = 1 MSS,
		Set state to “Slow Start”
			Enter slow start

SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

TCP Timer Management

- **TCP Retransmission Timeout (RTO)**: When a segment is sent, a retransmission timer is started
  - **If the segment is acknowledged before the timer expires, the timer is stopped**
  - **If the timer expires before the acknowledgement comes, the segment is retransmitted**

- **What can be an ideal value of RTO ?**

- **Possible solution**: Estimate RTT, and RTO is some positive multiples of RTT

RTT Estimation at the Transport Layer

- Use a dynamic algorithm that constantly adapts the timeout interval, based on continuous measurements of network performance.

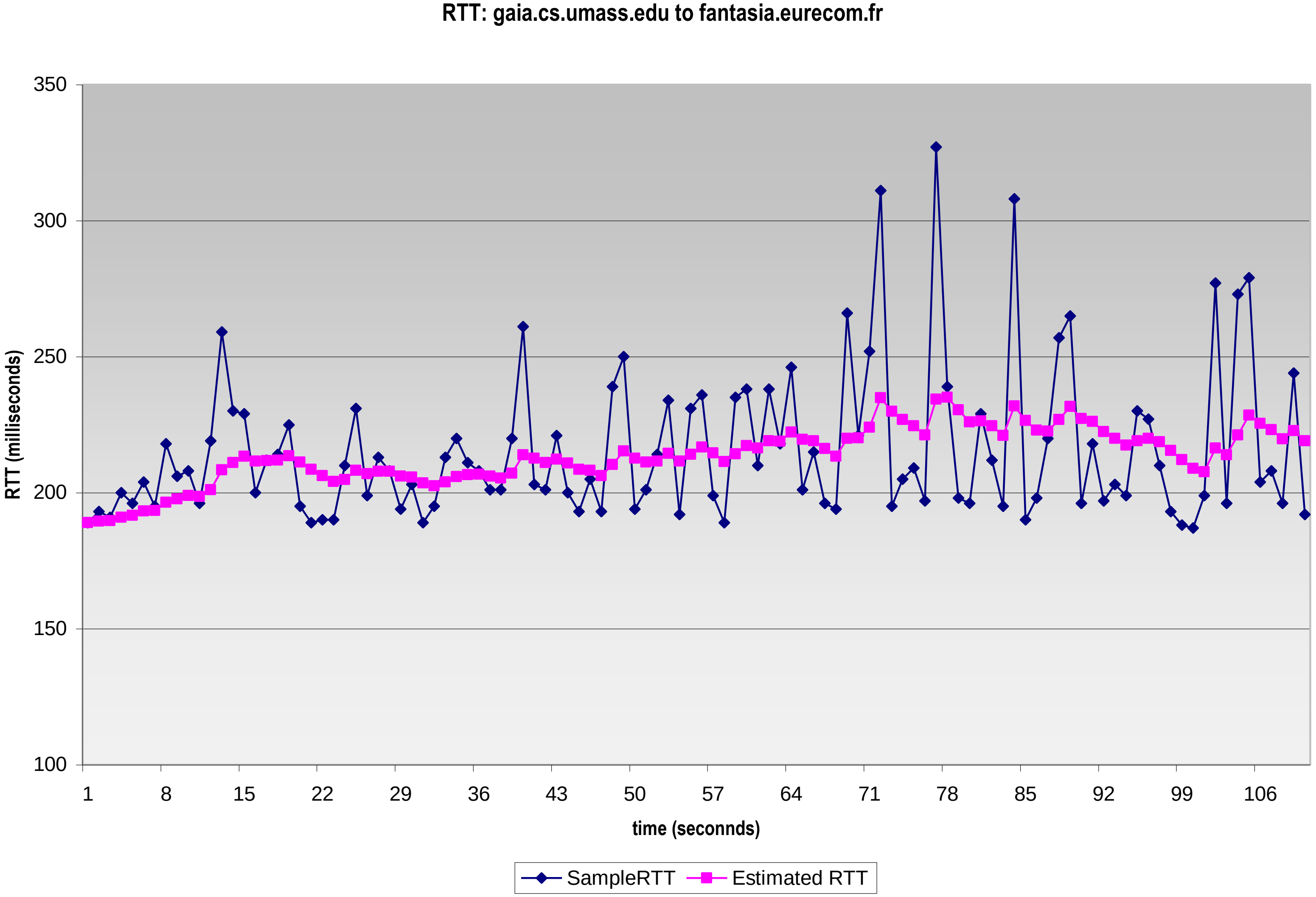
- **Jacobson's algorithm (1988) - used in TCP**
    - For each connection, TCP maintains a variable, ***ERTT (Estimated Round Trip Time)*** – best current estimate of the round trip time to the destination
    - When a segment is sent, a timer is started (both to see how long the acknowledgement takes and also to trigger a retransmission if it takes too long)
    - If the ACK gets back – measure the time (say, *R*)
    - Update ERTT as follows
      $\boldsymbol{ERTT = (1-\alpha)\ ERTT + (\alpha)R}$ **(Exponentially Weighted Moving Average – EWMA)**
    - α is a smoothing factor that determines how quickly the old values are forgotten. Typically α = 0.125

TCP Round Trip Time and Timeout

$$\texttt{EstimatedRTT} = (1-\alpha)*\texttt{EstimatedRTT} + \alpha*\texttt{SampleRTT}$$

- Exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$

Example RTT estimation:



Problem with EWMA

- Even given a good value of ERTT, choosing a suitable RTO is nontrivial.
- Initial implementation of TCP used RTO = 2 times ERTT
- Experience showed that a constant value was too inflexible, because it failed to respond when the **variance went up (RTT fluctuation is high) – happens normally at high load**
- **Consider variation of RTT during RTO estimation.**

RTO Estimation

- Update RTT variation (*RTTVAR*) as follows.

$$\boldsymbol{RTTVAR = \beta\, RTTVAR + (1 - \beta)|ERTT - R|}$$

- Typically β = ¾

- RTO is estimated as follows,

$$\boldsymbol{RTO = ERTT + 4 \times RTTVAR}$$

- **Why 4 ?**
  - **Somehow arbitrary**
  - **Jacobson’s paper is full of clever tricks – use integer addition, subtraction and shift – computation is lightweight**

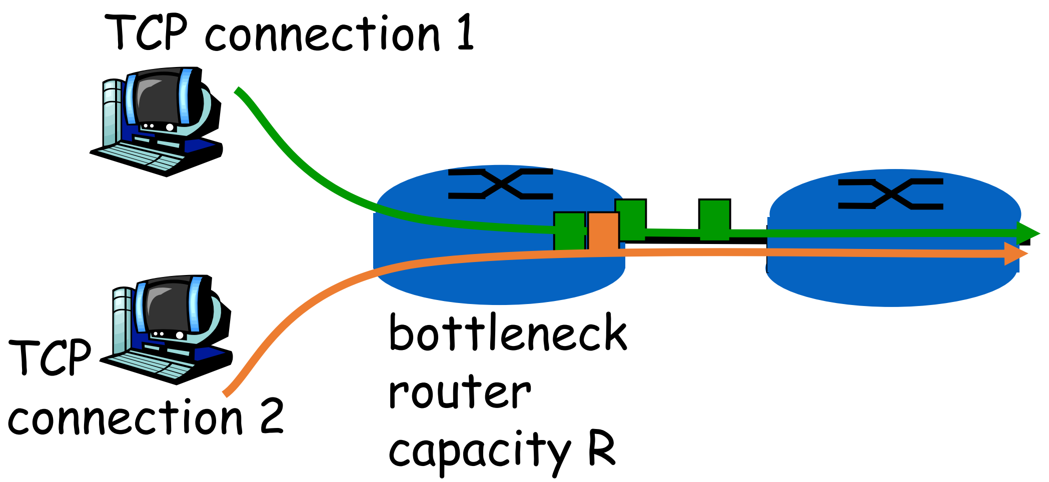
Karn’s Algorithm

- How will you get the RTT estimation, when a segment is lost and retransmitted again?

- **Karn’s algorithm:**
  - Do not update estimates on any segments that has been retransmitted
  - The timeout is doubled each successive retransmission until the segments gets through the first time

TCP Fairness

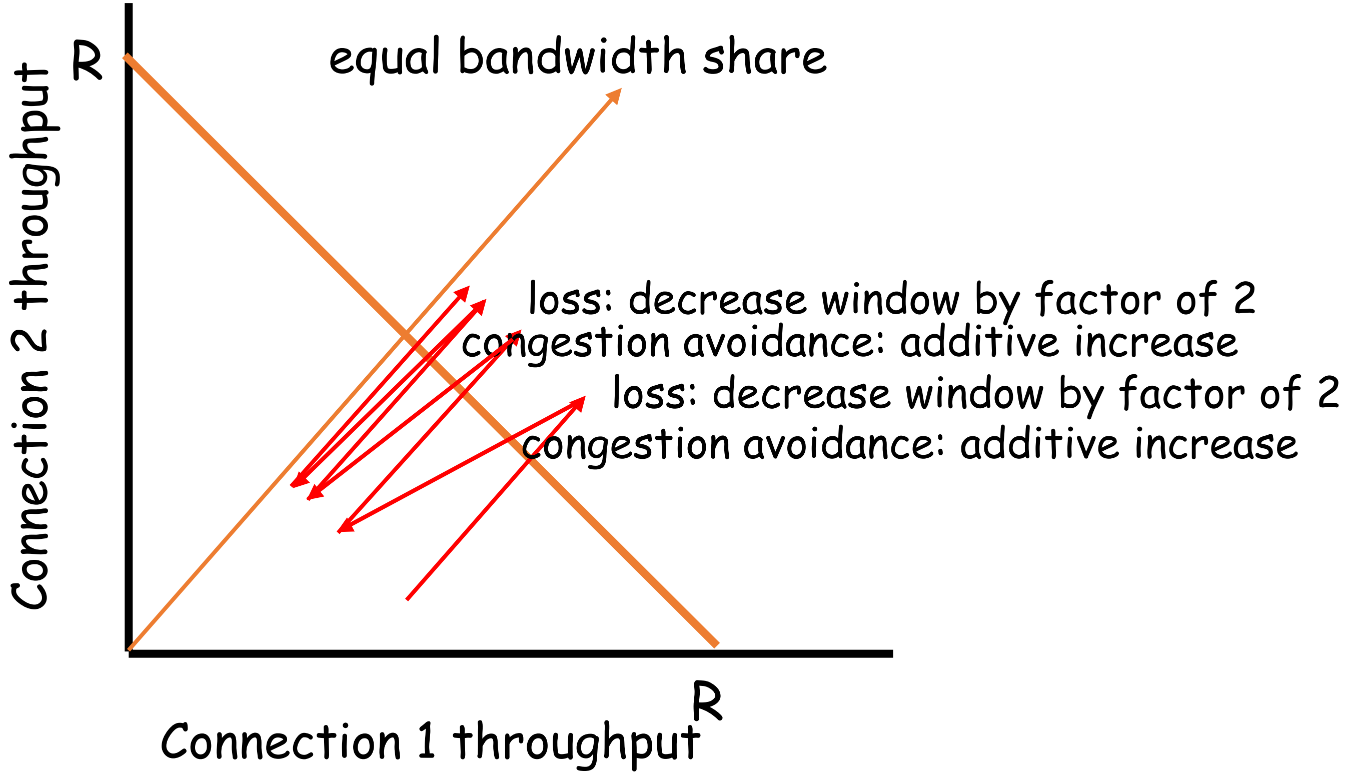
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally


R
equal bandwidth share
loss: decrease window by factor of 2
congestion avoidance: additive increase
loss: decrease window by factor of 2
congestion avoidance: additive increase
Connection 2 throughput
R
Connection 1 throughput

Fairness (more)

Fairness and UDP

- Multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- Instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

Fairness and parallel TCP connections

- nothing prevents app from opening parallel connections between 2 hosts.
- Web browsers do this
- Example: link of rate R supporting 9 connections;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2 !

UDP: User Datagram Protocol [RFC 768]

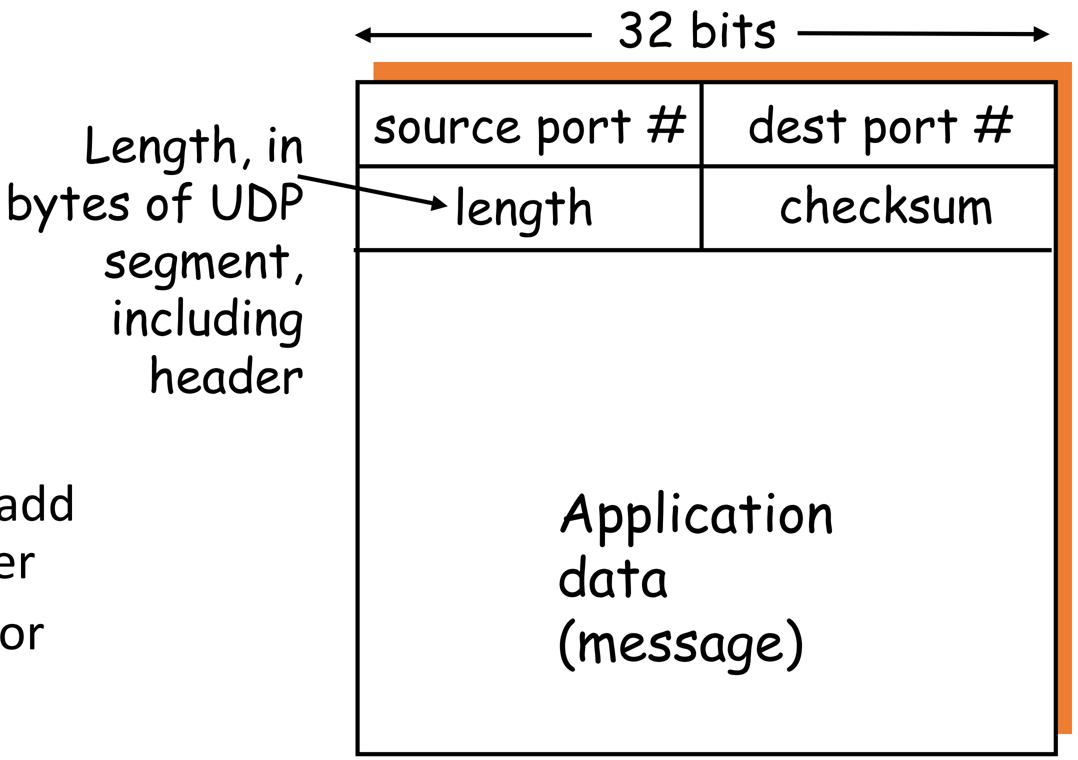
- “best effort” service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

UDP: more

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!


32 bits
source port #
dest port #
length
checksum
Application data (message)
Length, in bytes of UDP segment, including header

UDP segment format

UDP: Summary

- Finer Application-level control over what data is sent and when
- No connection establishment
- No connection state(buffer information, congestion control information, flow control information i.e., seq nums and ack nums

- Small packet header overhead (of 8 bytes)

Transport Layer Summary

- Connection Establishment – Sequence Numbers and 3 way handshake protocol
- Connection Release – Symmetric and Asymmetric release, 3 way handshake
- Flow Control – Stop and Wait, Sliding Window (Go back N and Selective Repeat), Bandwidth Delay Product,
- Error Control – Through Retransmissions
- Congestion Control – AIMD Scheme, TCP Congestion Control
- TCP Segment Structure, TCP Connection Establishment, TCP Connection Release,
- TCP Fairness
- Timeout Calculation
- UDP