Lesson 3.3: Transport Layer

CSC450 – COMPUTER NETWORKS | WINTER 2019-20

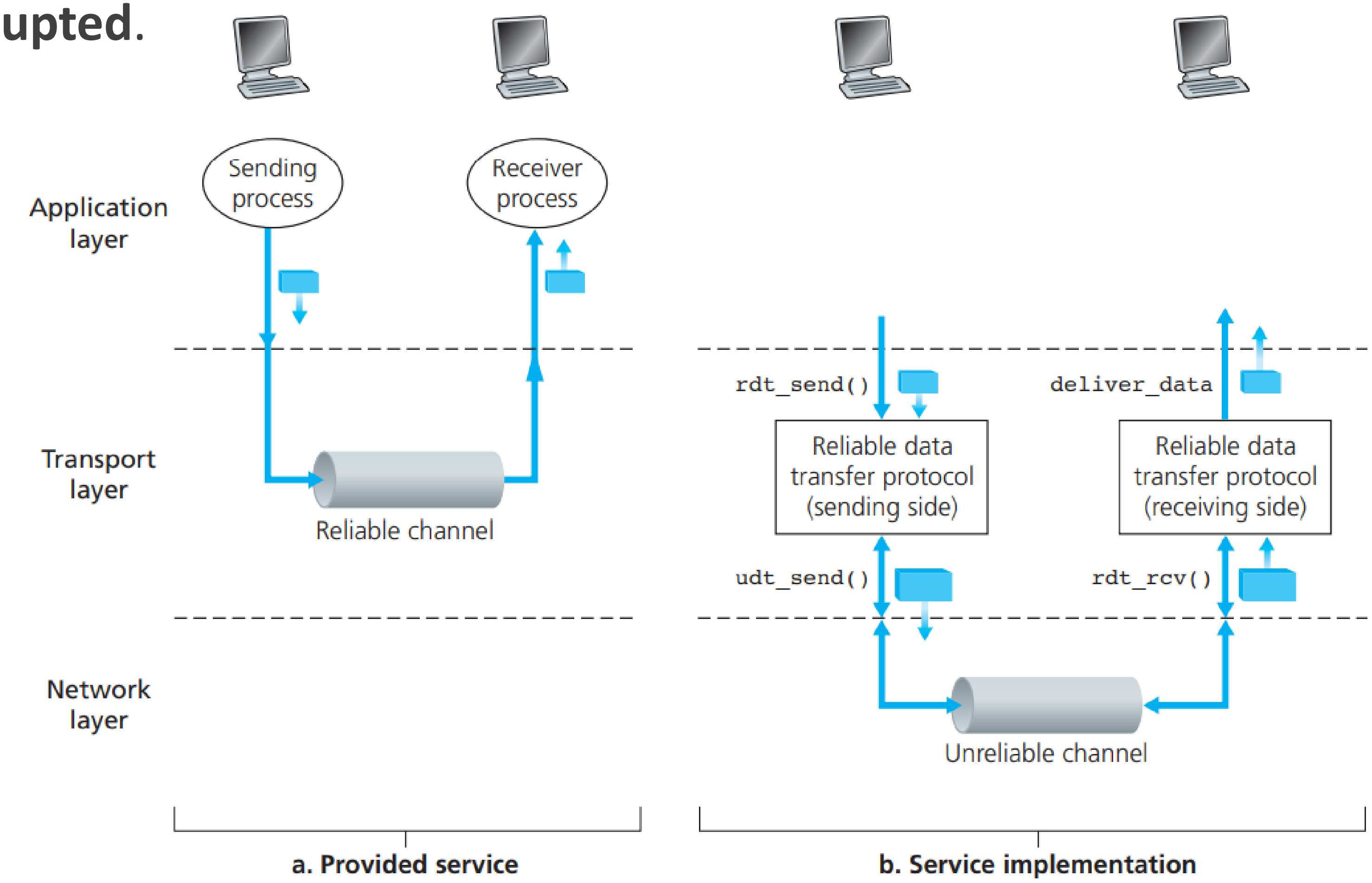
DR. ANDREY TIMOFEYEV

OUTLINE

- Reliable data transfer.
  - Principles.
  - Automatic repeat request (ARQ).
  - Stop-and-wait.
  - Sliding widow.
    - Go-back-N (GBN).
    - Selective repeat (SR).
- TCP reliable data transfer.

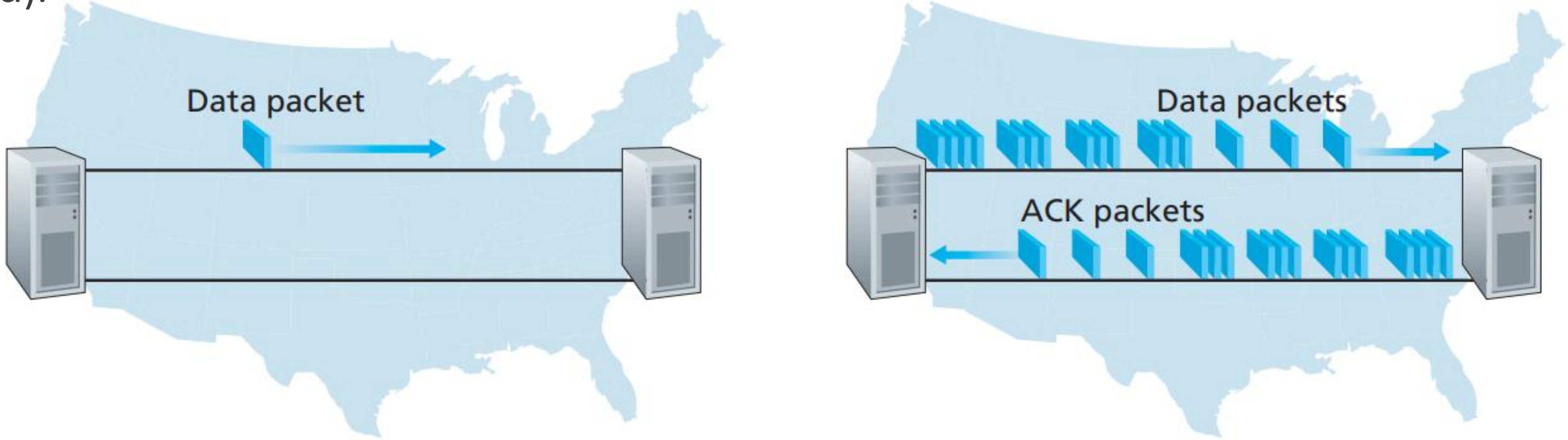
PRINCIPLES OF RELIABLE DATA TRANSFER

- Reliable data transfer **guarantees**:
  - **No bits** in transferred segments are **corrupted**.
    - Error-checking mechanism (checksum).
  - **No** transferred **segments** are **lost**.
    - ACKs & timeouts.
  - **All** transferred **segments** are delivered in **order**.
    - Sequence numbers.



AUTOMATIC REPEAT REQUEST (ARQ)

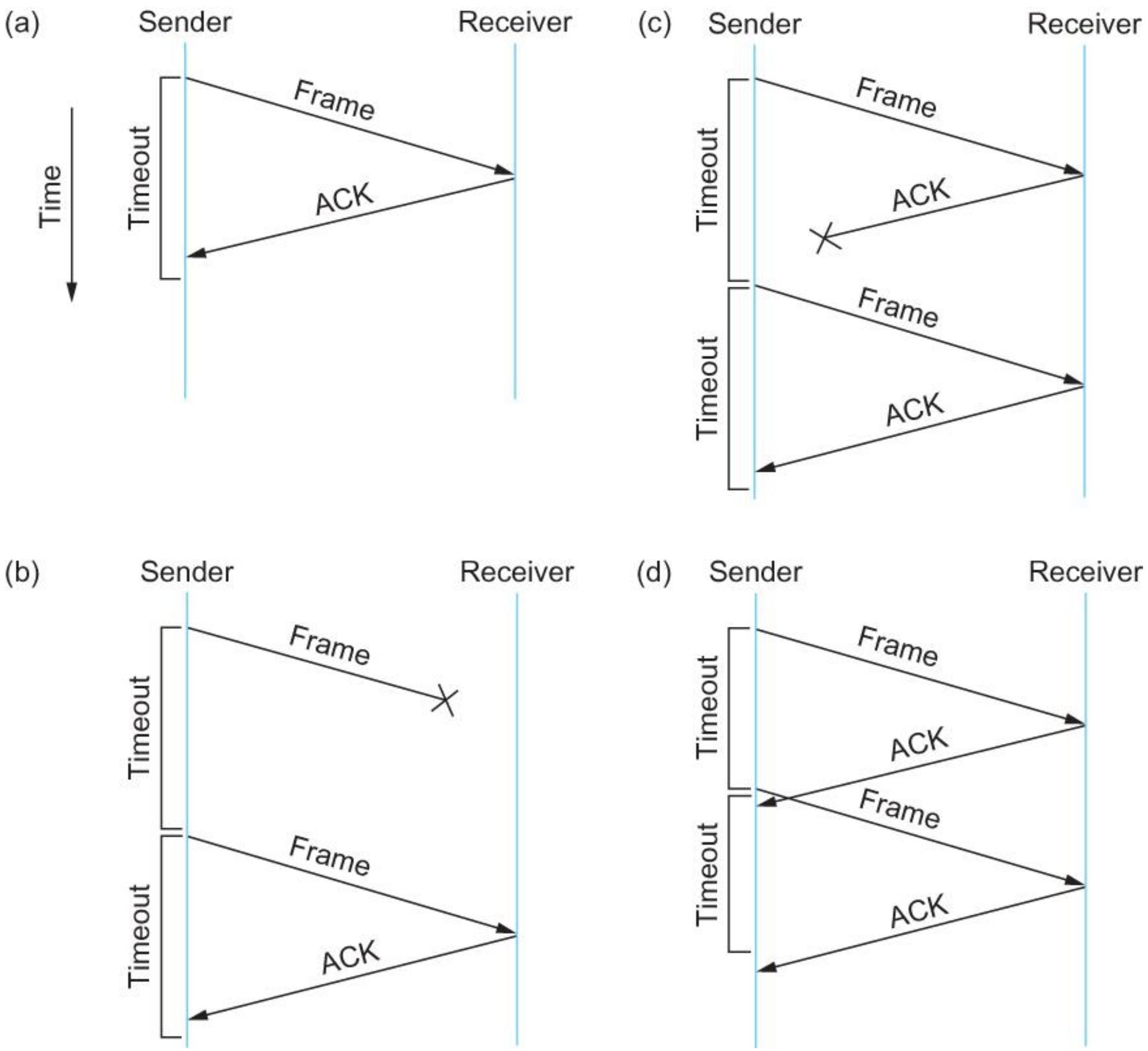
- **Automatic repeat reQuest** (**ARQ**) method is used to assure **reliable segment delivery**.
- ARQ is based on combination of two fundamental mechanisms: **acknowledgements** (**ACKs**) & **timeouts**.
  - If sender does not receive **ACK** after **timeout** then it **retransmits** the **segment**.
- Two main **ARQ** methods:
  - **Stop-and-wait** (serial).
  - **Sliding window** (pipelined).



Serial vs. pipelined ARQ

STOP-AND-WAIT: ACKNOWLEDGEMENT

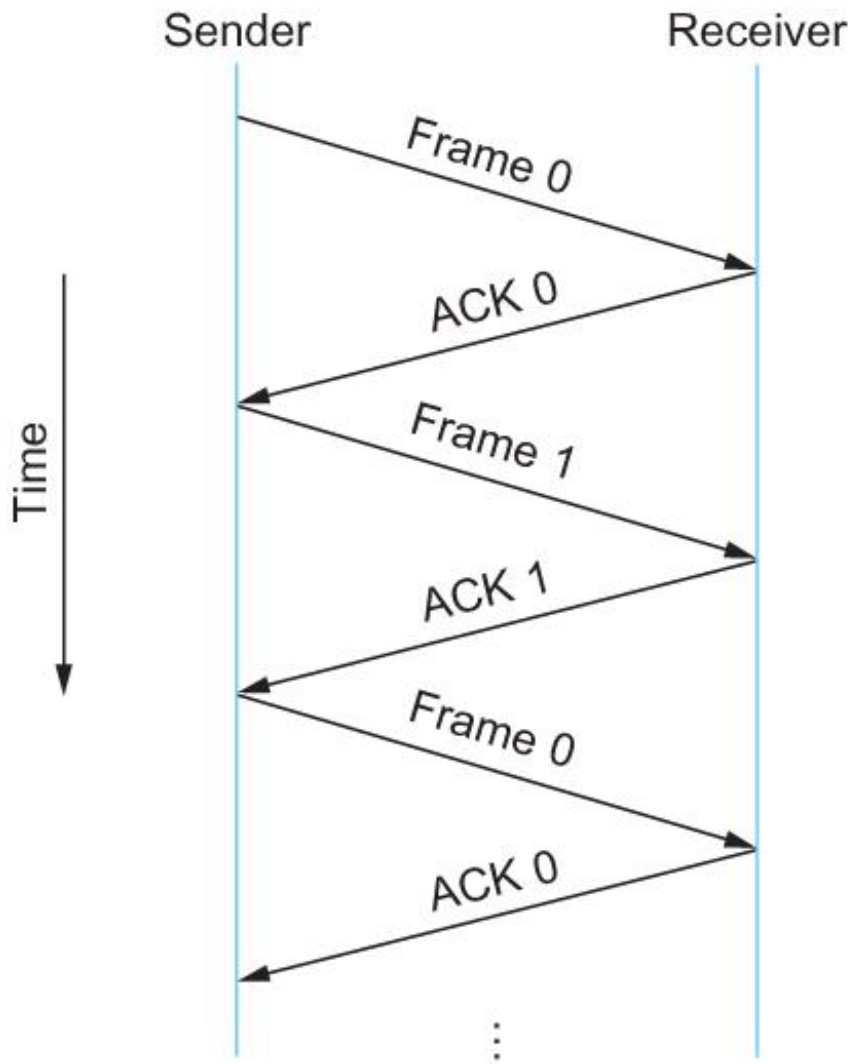
- **Stop-and-wait** algorithm:
  - After **transmitting** one segment, sender **waits** for **ACK** before **transmitting** next.
  - If **ACK** does **not arrive** after **timeout**, sender **retransmits** original segment.
- If **ACK** is **lost** or **delayed** in arriving (c & d):
  - Sender **times out** and **retransmits original** segment.
  - **Receiver** treats this **segment** as **next segment**.
    - It correctly received and ACKed previous segment.
  - Problem**: duplicate** copies of **segments** are delivered.
  - Solution: **alternating-bit approach**.
    - Use 1-bit **sequence** number (0/1).



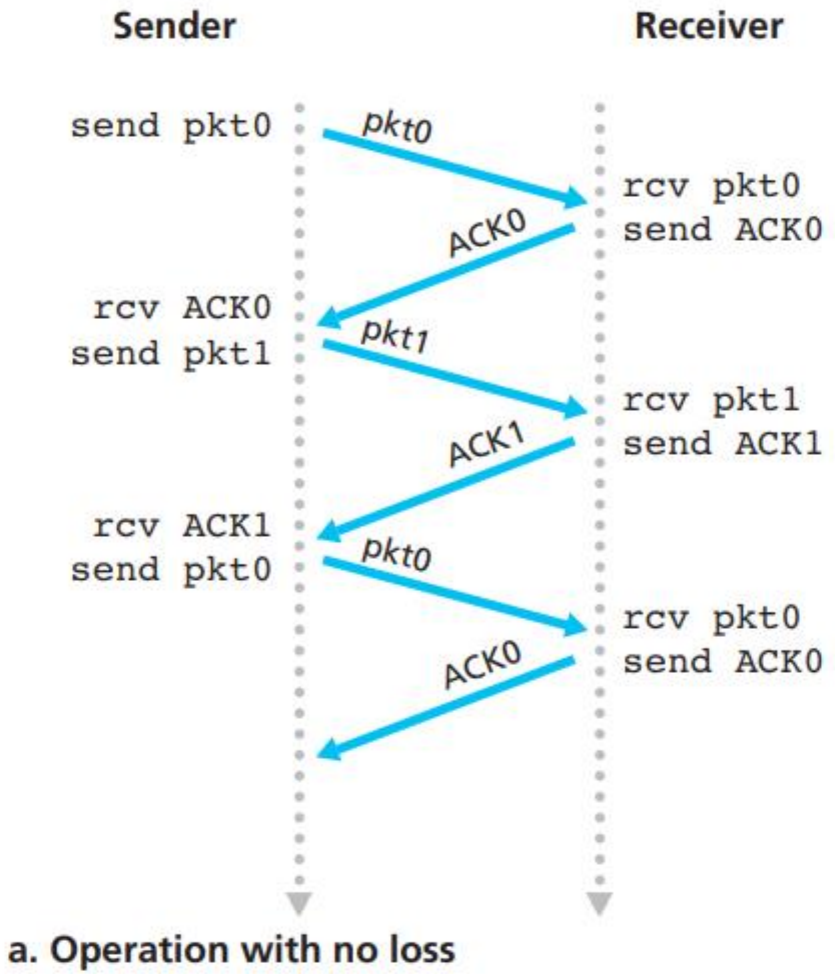
Scenarios of stop-and-wait algorithm

STOP-AND-WAIT: ALTERNATING BIT

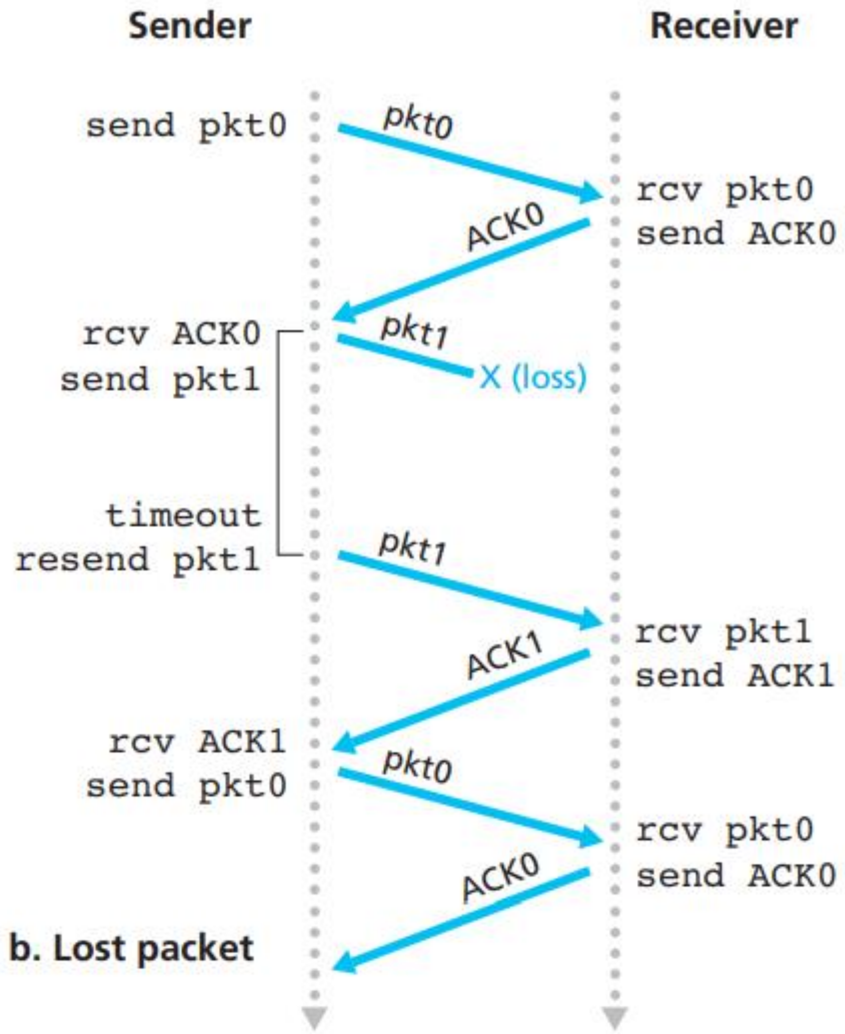
- **Alternating-bit** approach:
  - Each **segment** has a **sequence** bit set to *0* or *1*.
  - When sender **retransmits** segment:
    - Receiver can determine if it is **first** or **second** copy of segment.
      - **Ignores** segment if it is **second** copy.
      - **ACKs** segment.
        - In case the first ACK was lost.



STOP-AND-WAIT: ALTERNATING BIT EXAMPLES (1)



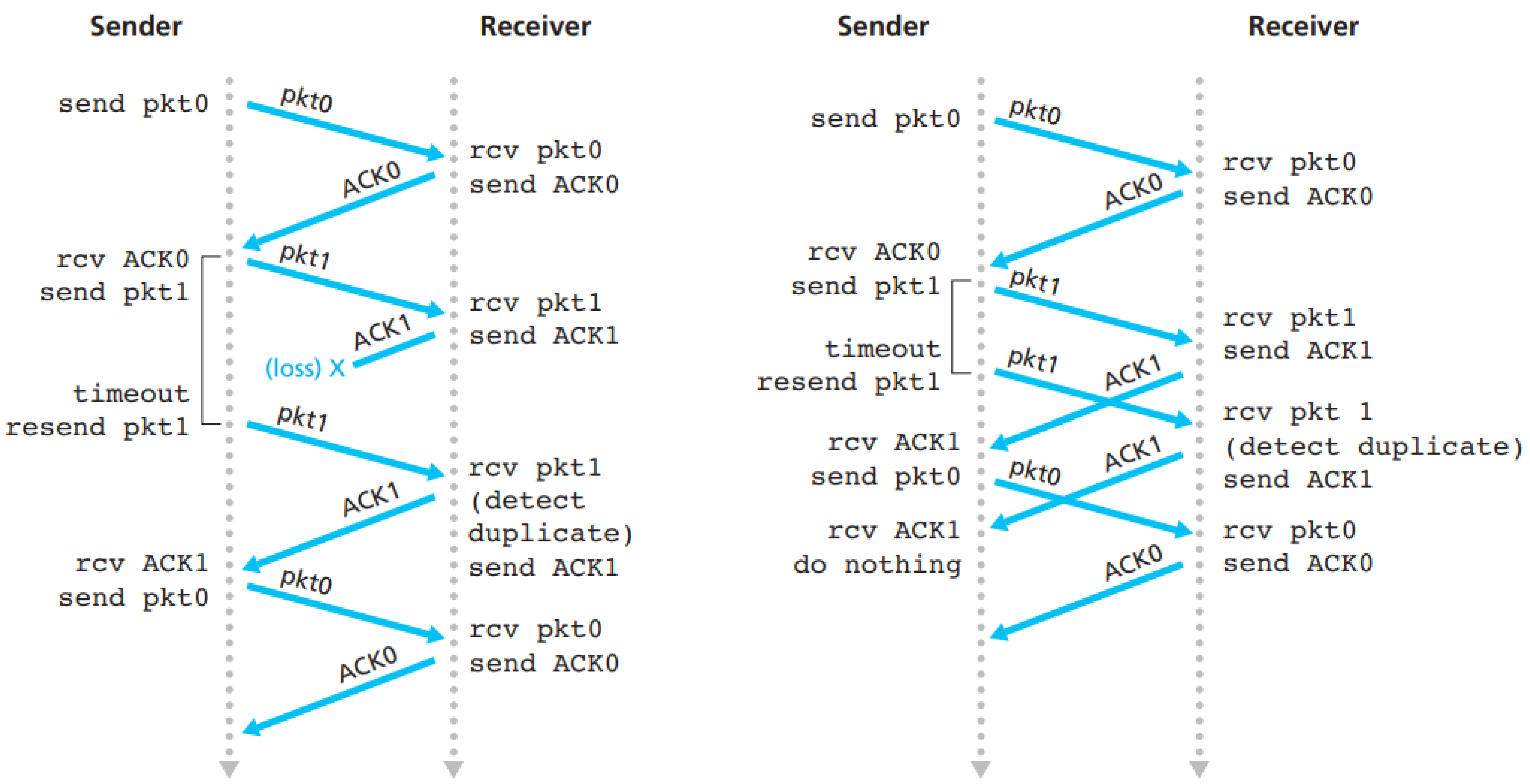
a. Operation with no loss



b. Lost packet

Stop-and-wait with alternating bit

STOP-AND-WAIT: ALTERNATING BIT EXAMPLES (2)



Sender

send pkt0

rcv ACK0
send pkt1

timeout
resend pkt1

rcv ACK1
send pkt0

Receiver

rcv pkt0
send ACK0

rcv pkt1
send ACK1

rcv pkt1
(detect
duplicate)
send ACK1

rcv pkt0
send ACK0

c. Lost ACK

Sender

send pkt0

rcv ACK0
send pkt1

timeout
resend pkt1

rcv ACK1
send pkt0

rcv ACK1
do nothing

Receiver

rcv pkt0
send ACK0

rcv pkt1
send ACK1

rcv pkt 1
(detect duplicate)
send ACK1

rcv pkt0
send ACK0

d. Premature timeout

Stop-and-wait with alternating bit

STOP-AND-WAIT: TIMEOUT

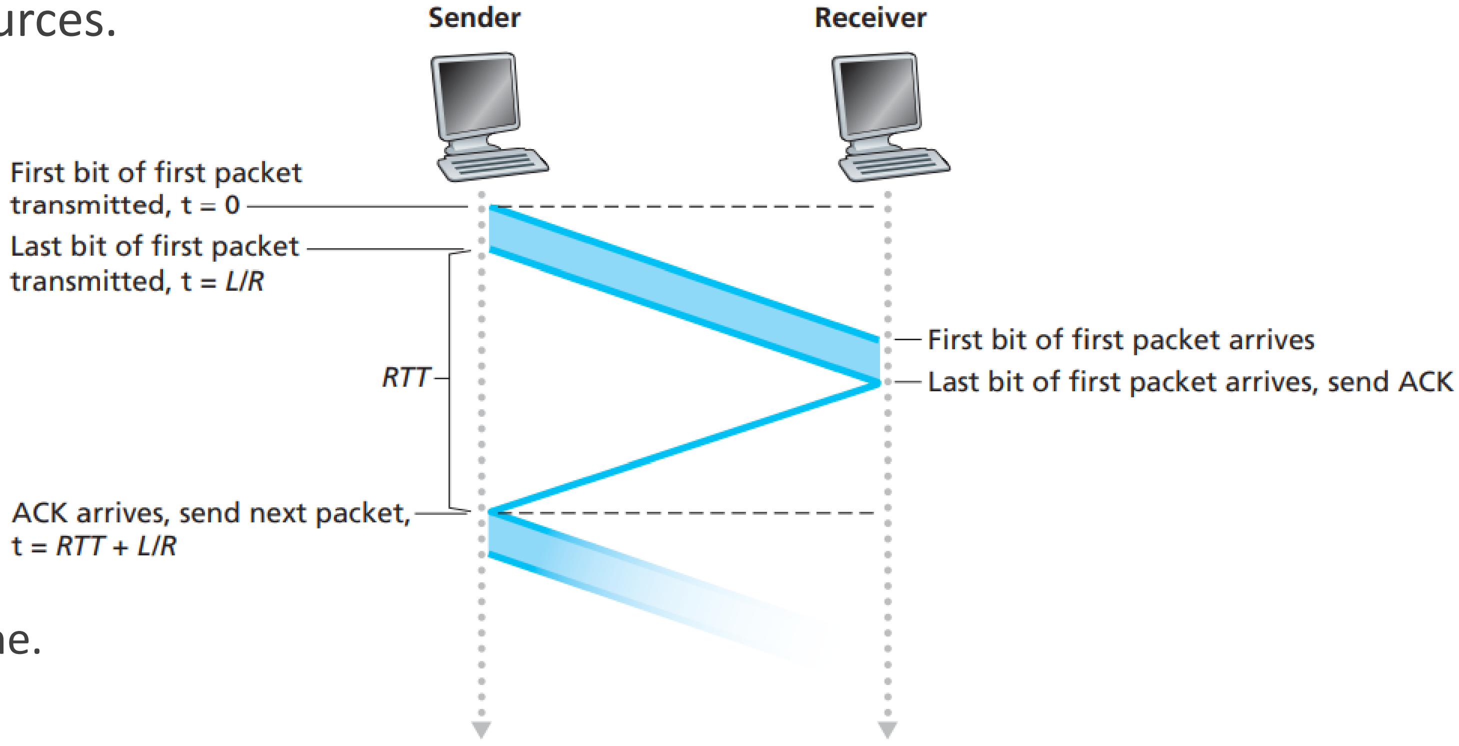
- **Retransmission approach** requires a mechanism that **interrupts** sender after given amount of time (**timeout**) has **expired**.
- Countdown **timer** handled by the **sender**.
  - **Starts** the timer each time a **segment** (original or retransmission) is **sent**.
  - **Stops** timer when segment is **ACKed**.
  - **Retransmits** segment if timer **exceeded** timeout.

Stop-and-wait animation

STOP-AND-WAIT: FLAW

- **Performance** is the main **flaw** of **stop-and-wait** algorithm.
  - Allows sender to have only **one unACKed segment** at a time.
    - Protocol limits the full use of physical resources.
    - “Pipe is not full”.

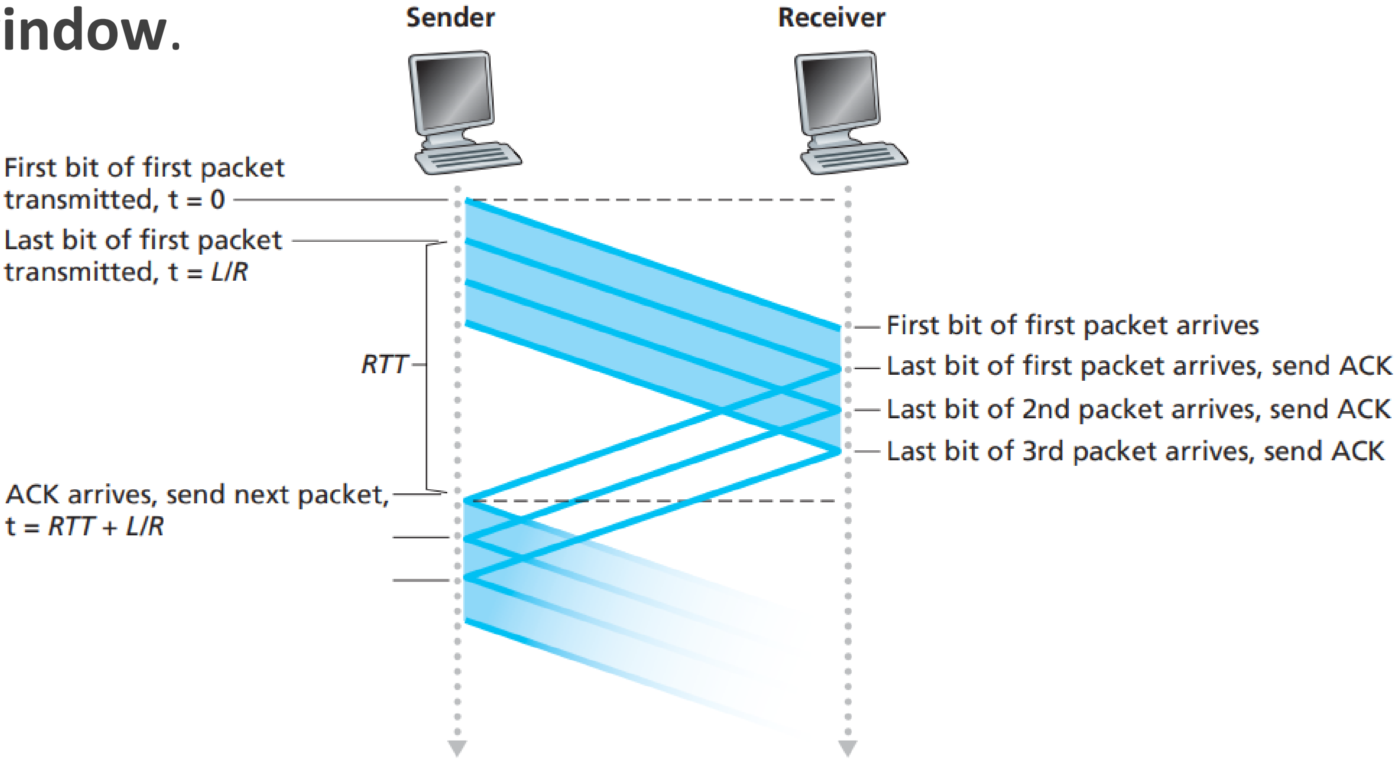
- **Example**:
  - Propagation delay (*PD*) = 15 ms.
  - Transmission rate (*R*) = 1 Gbps (10^9 bits/s).
  - Packet size (*L*) = 1000 bytes ($8*10^3$ bits).
  - Transmission delay (*TD*) = 0.008 ms.
  - Utilization (*U*) = TD / (RTT + TD) = 0.0027
    - Sender is busy only 0.0027 (<1%) of sending time.
    - Sender is able to send 8000 bits in 30.008 ms.
      - Only 267 Kb per seconds on a link of 1 Gb per second.


Sender
Receiver
First bit of first packet transmitted, t = 0
Last bit of first packet transmitted, t = L/R
RTT
First bit of first packet arrives
Last bit of first packet arrives, send ACK
ACK arrives, send next packet, t = RTT + L/R

Stop-and-wait serial operation

SLIDING WINDOW

- **Performance flaw** of **stop-and-go** approach can be **mitigated** by **pipelining**.
  - **Sender** allows multiple “**in-flight**” **yet-to-be-acknowledge** segments.
    - **Increased** range of **sequence** numbers.
    - **Buffering** at **sender** & **receiver**.
- **Pipelining** is based on an idea of **sliding window**.
- Two types of **sliding window** protocols:
  - Go-back-N (**GBN**).
  - Selective repeat (**SR**).


Sender
Receiver
First bit of first packet transmitted, t = 0
Last bit of first packet transmitted, t = L/R
RTT
ACK arrives, send next packet, t = RTT + L/R
First bit of first packet arrives
Last bit of first packet arrives, send ACK
Last bit of 2nd packet arrives, send ACK
Last bit of 3rd packet arrives, send ACK

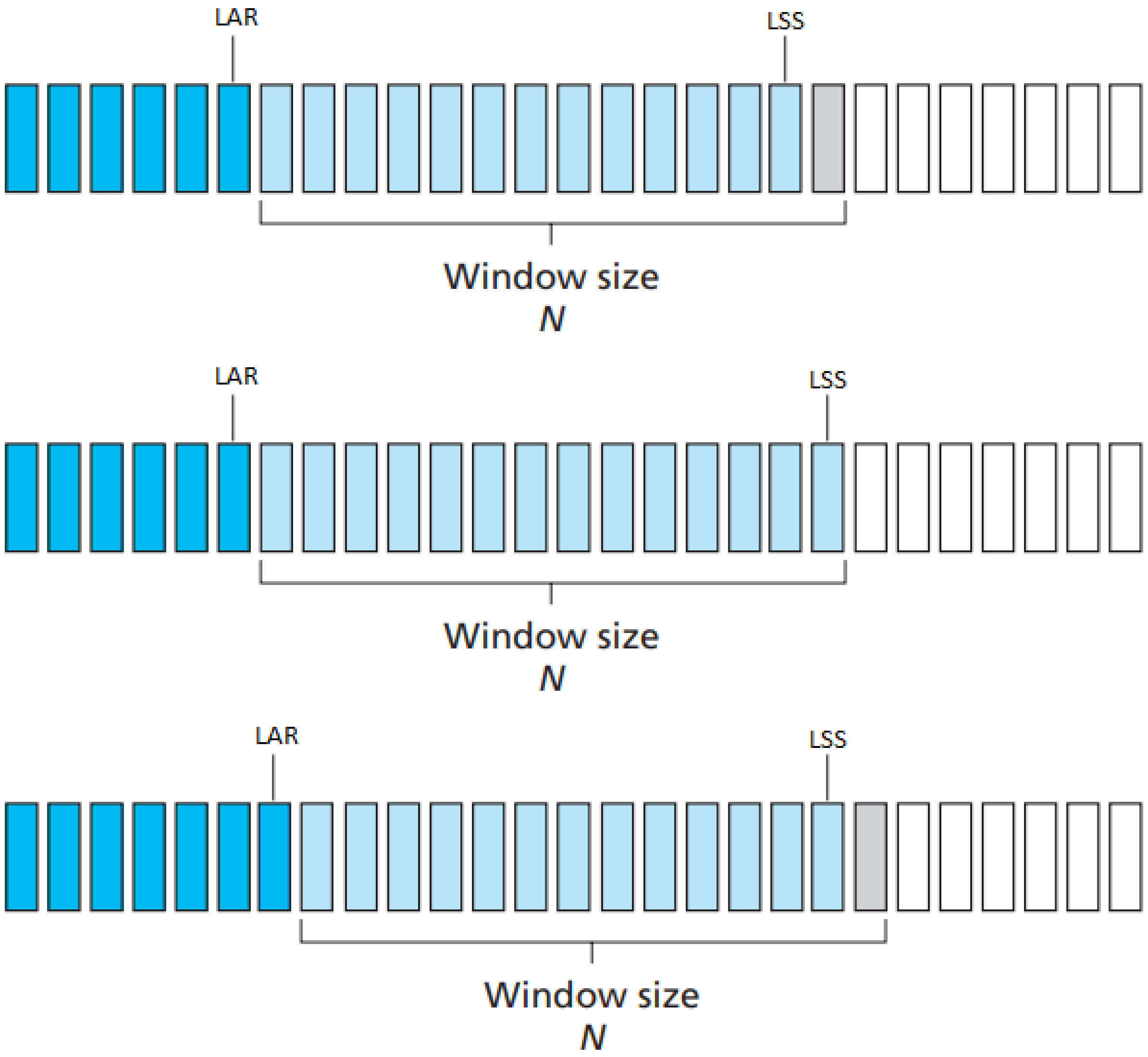
Pipelined operation

GO-BACK-N (1)

- **GBN overview**:
  - **Sender** can have up to *N* **unACKed segments** in pipeline.
  - **Receiver** only sends **cumulative ACK**.
    - Doesn’t **ACK** segment if there’s a **gap**.
    - **Discards** segments that arrived out-of-order.
  - **Sender** has a **single timer** for the **oldest unACKed** segment.
    - When timer **expires** sender **retransmits all** unACKed segments.
    - **Major flaw**, since a **single segment error** can cause **unnecessary retransmission** of large number of segments.

GO-BACK-N (2)

- **Sender side**:
  - Sender window size (**N**).
    - Max number of unACKed segments.
  - Last acknowledgement received (**LAR**).
  - Last segment sent (**LSS**).
  - Can send segments while (**LSS – LAR <= N**).
- **Message** comes from **application** layer:
  - LSS – LAR = N.
  - Turn message into segment and send it.
- **Higher ACK** comes from **receiver**:
  - Window advances, buffer is freed.
  - LSS – LAR <= N.
  - Can send more messages.



Sender view of sequence numbers in GBN

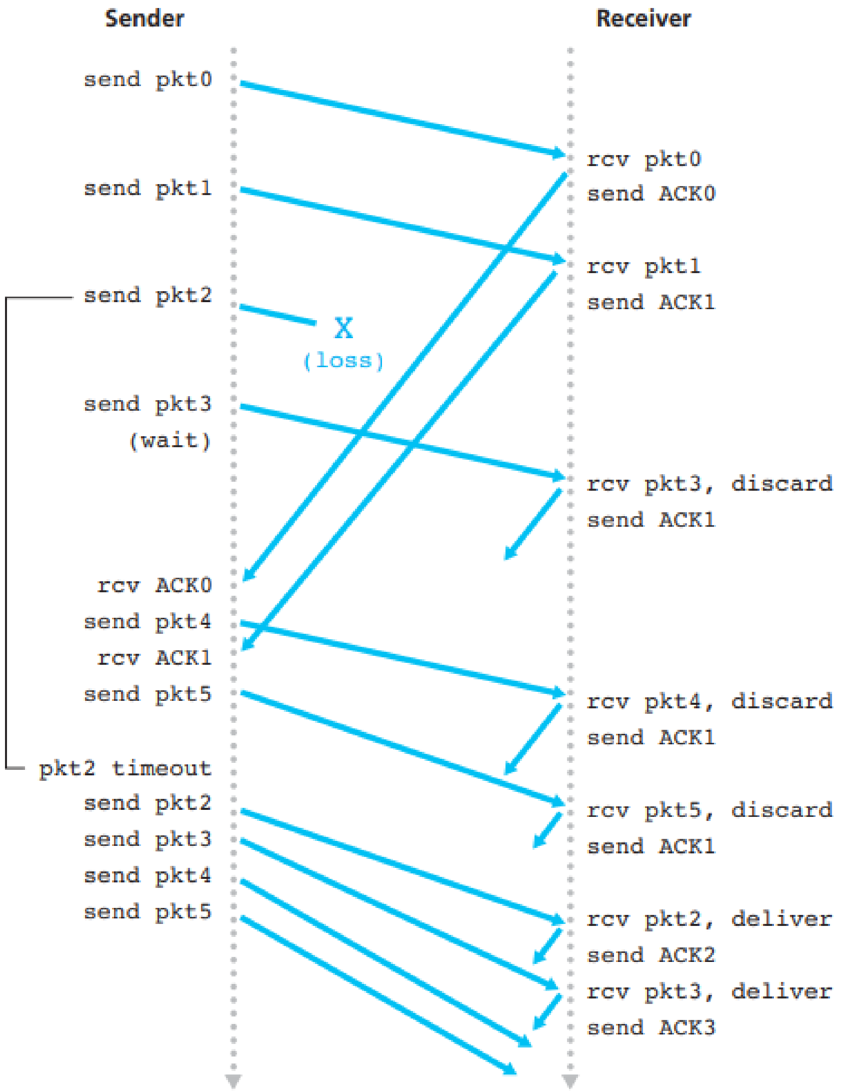
GO-BACK-N (3)

- **Receiver side**:
  - Segment arrived **correctly** & **in-order** → send **ACK** to the sender.
    - Segment with **highest in-order** sequence number (*expected sequence number*).
  - Any **out-of-order** segments are **discarded**.
    - No **buffering** on receiver side.
    - ACK sent with the **highest in-order** sequence number.

GO-BACK-N: OPERATION

- **GBN example**:
  - Window size N = 4.
  - Sender sends segments 0-3 and waits for ACKs.
  - As ACKs arrives → window slides forwards → next segments are sent.
  - On receiver – segment 2 is lost → segments 3-5 discarded.
  - On sender – segment 2 timeout → segments 2-5 retransmitted.

- Improvement – **selective acknowledgements**.
  - Receiver **ACKs** exactly those segments it has **received**.
  - **Selective repeat** (**SR**) approach.



GBN in operation

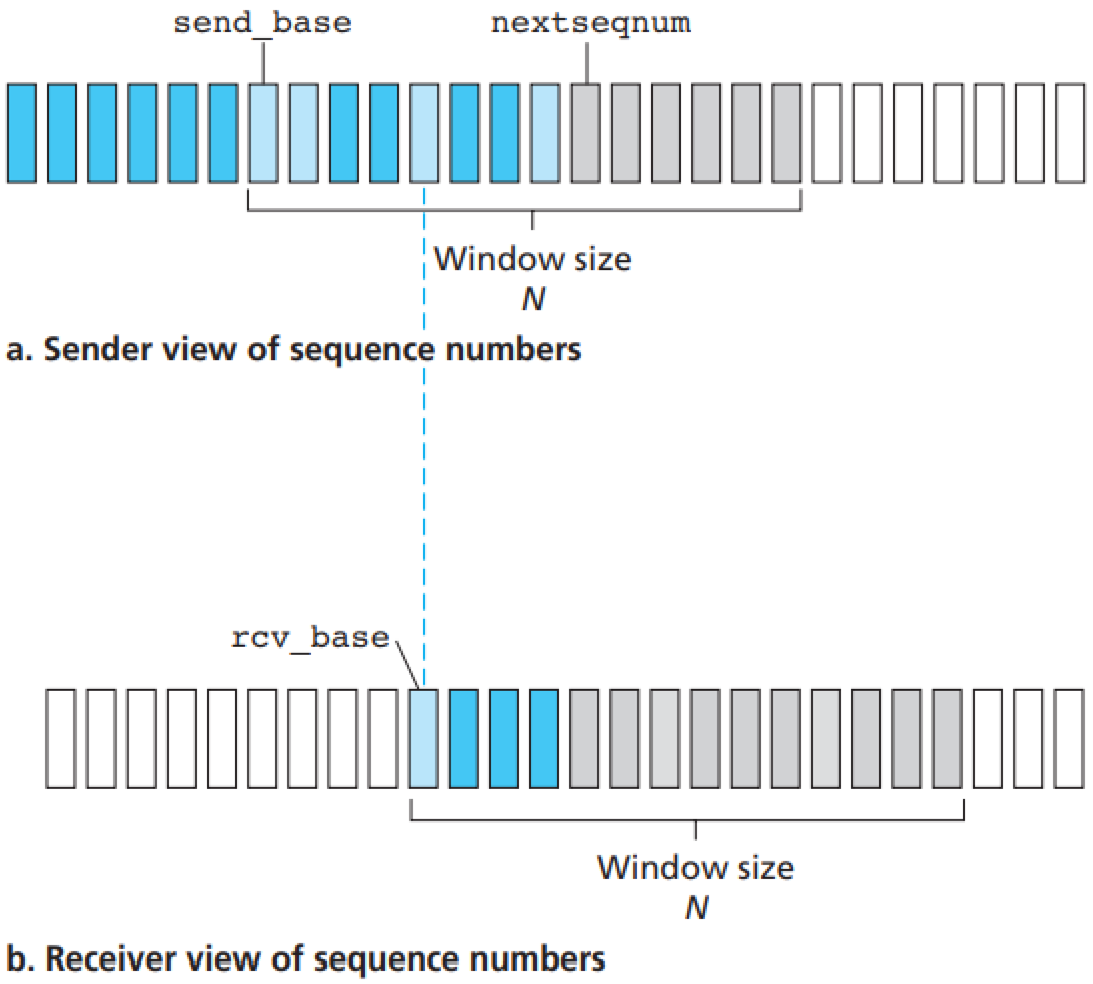
GBN animation

SELECTIVE REPEAT (1)

- **SR overview**:
  - **Sender** can have up to *N* **unACKed segments** in pipeline.
  - **Receiver** sends **individual ACKs** for each segment.
  - **Sender** maintains **individual timer** for each **unACKed** segment.
    - When timer **expires** sender **retransmit only** that **unACKed** segment**.**

SELECTIVE REPEAT (2)

- **Sender side**:
  - **Message** comes from **application** layer:
    - If next available seq# in window, **send** segment.
  - **Segment** timeout:
    - **Resend** segment, **restart** timer.
  - **ACK** of **segment** in window:
    - **Mark** segment as **received**.
    - If **smallest unACKed** segment, advance **window** base to next **unACKed** seq#.



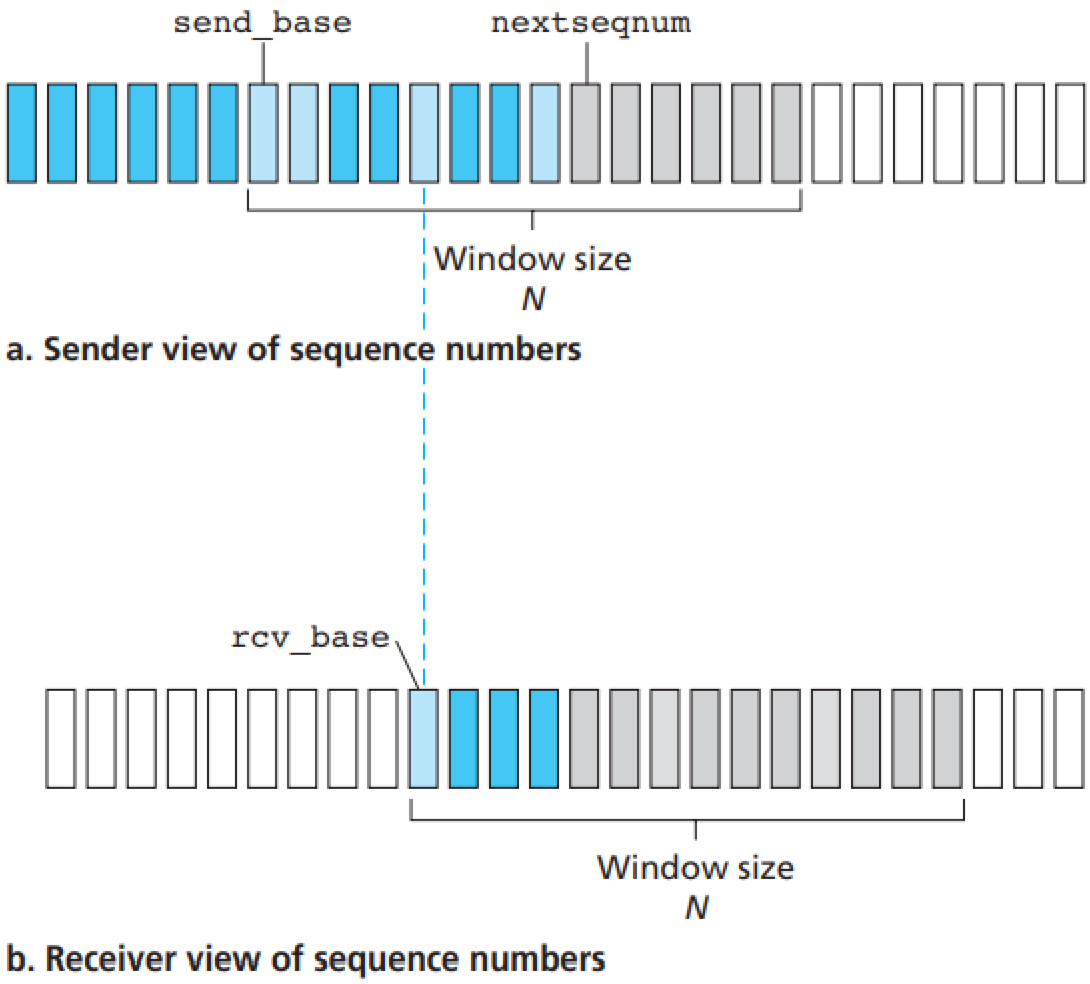
a. Sender view of sequence numbers

b. Receiver view of sequence numbers

Sender & receiver view of sequence number in SR

SELECTIVE REPEAT (3)

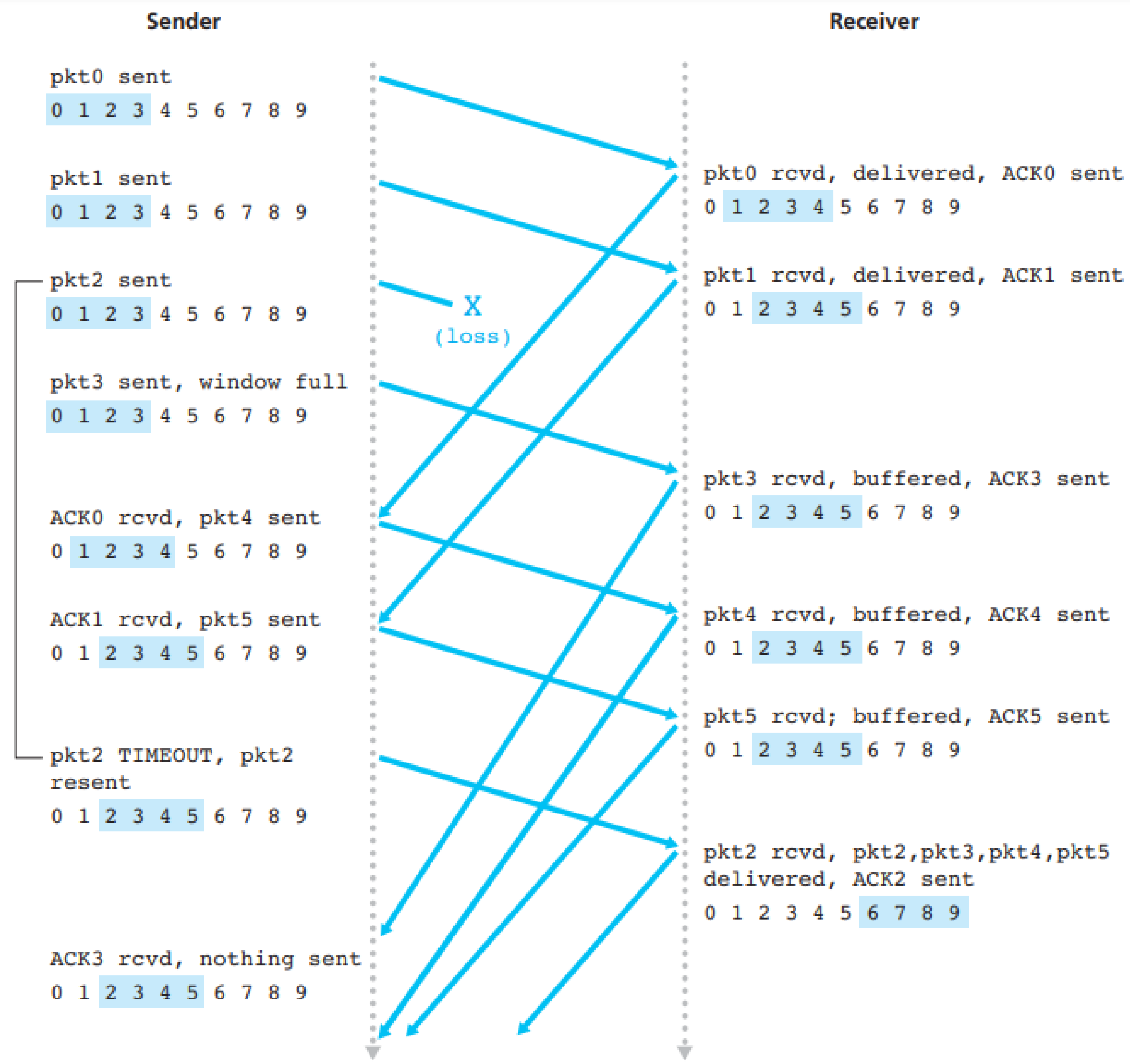
- **Receiver side**:
  - **Arrived segment** in window:
    - Send **ACK**.
    - **Out-of-order**: **buffer**.
    - **In-order**: **deliver**.
      - Also deliver **buffered in-order** segments.
      - Advance **window** to **next** not-yet-received segment.
  - **Segment** in [*rcv_base - N, rcv_base - 1*]:
    - Send **ACK**.
  - **Otherwise**:
    - **Discard**.


send_base
nextseqnum
Window size
N
a. Sender view of sequence numbers
rcv_base
Window size
N
b. Receiver view of sequence numbers

Sender & receiver view of sequence number in SR

SELECTIVE REPEAT: OPERATION

- **SR example**:
  - Window size N = 4.
  - Sender sends segments 0-3 and waits for ACKs.
  - ACKs arrive → window slides forward → next segments are sent.
  - On receiver – segment 2 lost, segments 3-5 buffered.
  - On sender – segment 2 timeout → segment 2 retransmitted, timer reset.
  - On receiver – segment 2 arrives → segments 2-5 are delivered.



Sender

pkt0 sent
0 1 2 3 4 5 6 7 8 9

pkt1 sent
0 1 2 3 4 5 6 7 8 9

pkt2 sent
0 1 2 3 4 5 6 7 8 9

pkt3 sent, window full
0 1 2 3 4 5 6 7 8 9

ACK0 rcvd, pkt4 sent
0 1 2 3 4 5 6 7 8 9

ACK1 rcvd, pkt5 sent
0 1 2 3 4 5 6 7 8 9

pkt2 TIMEOUT, pkt2 resent
0 1 2 3 4 5 6 7 8 9

ACK3 rcvd, nothing sent
0 1 2 3 4 5 6 7 8 9

X
(loss)

Receiver

pkt0 rcvd, delivered, ACK0 sent
0 1 2 3 4 5 6 7 8 9

pkt1 rcvd, delivered, ACK1 sent
0 1 2 3 4 5 6 7 8 9

pkt3 rcvd, buffered, ACK3 sent
0 1 2 3 4 5 6 7 8 9

pkt4 rcvd, buffered, ACK4 sent
0 1 2 3 4 5 6 7 8 9

pkt5 rcvd; buffered, ACK5 sent
0 1 2 3 4 5 6 7 8 9

pkt2 rcvd, pkt2,pkt3,pkt4,pkt5 delivered, ACK2 sent
0 1 2 3 4 5 6 7 8 9

SR in operation

SR animation

TCP RELIABLE DATA TRANSFER (1)

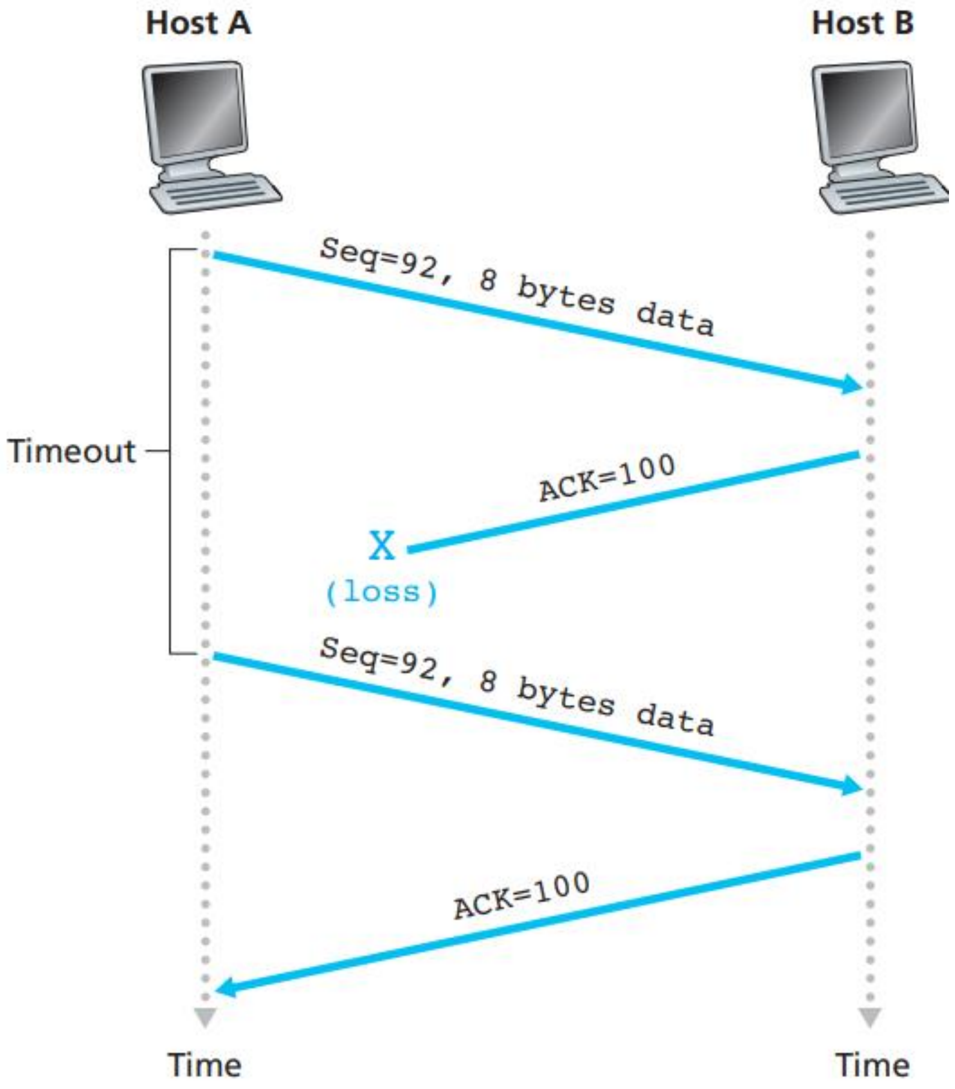
- **Principles** of **reliable data transfer** in **TCP** protocol:
  - **Pipelined** segments.
  - **Cumulative** ACKs.
  - **Single** retransmission timer.
  - **Combination** of GBN and SR.
- **Retransmission triggered** by:
  - **Timeouts**.
    - **Timeout** interval **doubles** after every retransmit (helps congestion control).
  - **Duplicated ACKs**.
    - Faster **detection** of **lost** segments (before timeout) → **faster retransmission**.

TCP RELIABLE DATA TRANSFER (2)

- **Sender side**:
  - **Message** comes from **application layer**.
    - **Create segment** with seq# (byte stream number).
    - If no timer **currently** running – **start** a timer.
      - Timer is set for the **oldest unACKed** segment.
      - **Estimated** expiration interval.
  - **Segment timeout**.
    - **Retransmit** segment that caused timeout.
    - **Restart** timer.
  - **ACK received**.
    - If ACKs **previously unACKed** segment:
      - **Update** what was previously ACKed.
      - **Restart** timer if any unACKed segments.
    - Else, count for **duplicated** ACKs.

TCP RELIABLE DATA TRANSFER: EXAMPLES (1)

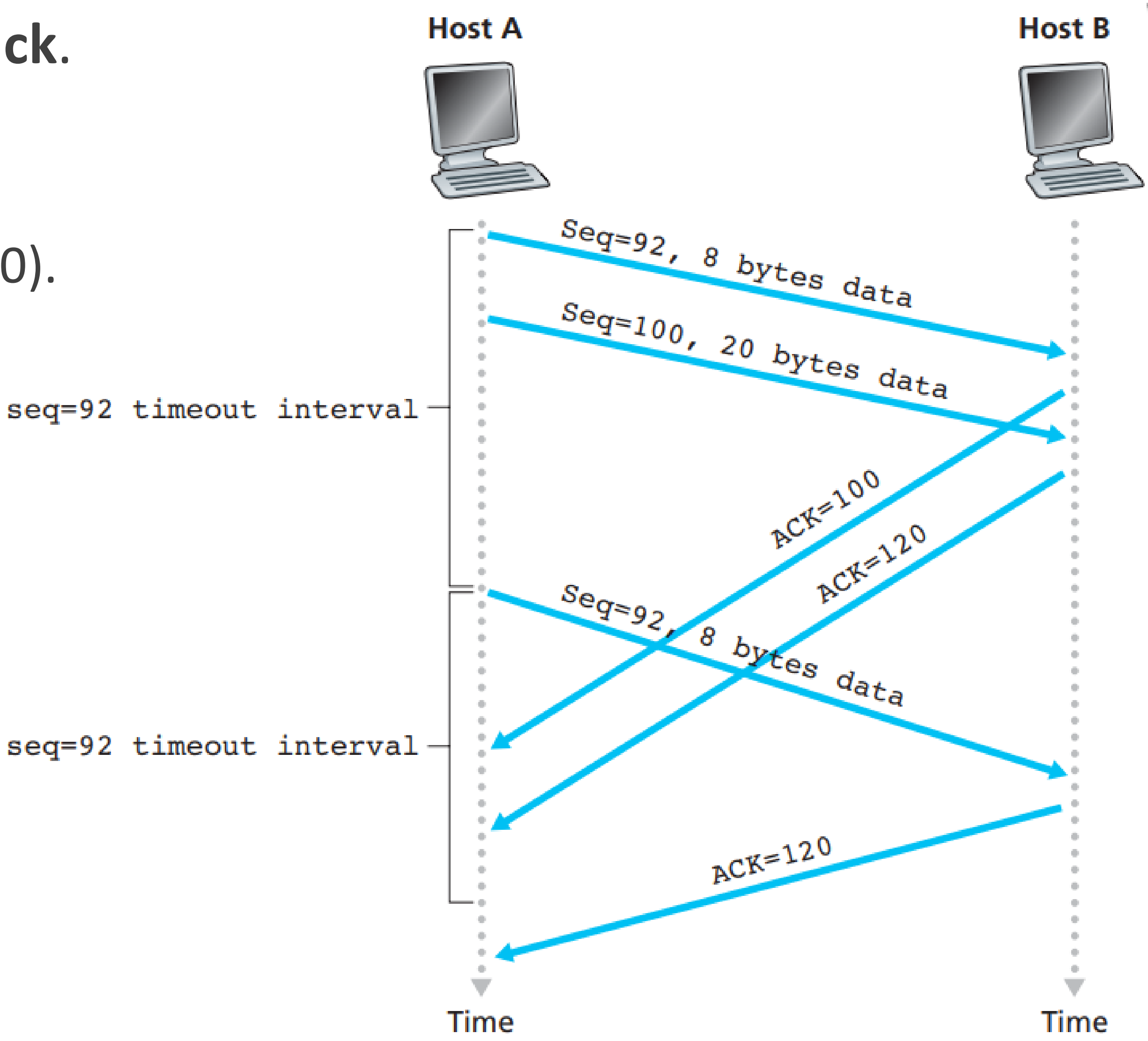
- Host A sends **single segment** to Host B.
  - Segment seq# = 92.
  - Segment size = 8 bytes.
  - ACK form receiver is lost.
  - Sender – segment timeout → retransmit.
  - Receiver – data has been already received → discard.
  - Receiver – ACK duplicated segment.



Retransmission due to lost ACK

TCP RELIABLE DATA TRANSFER: EXAMPLES (2)

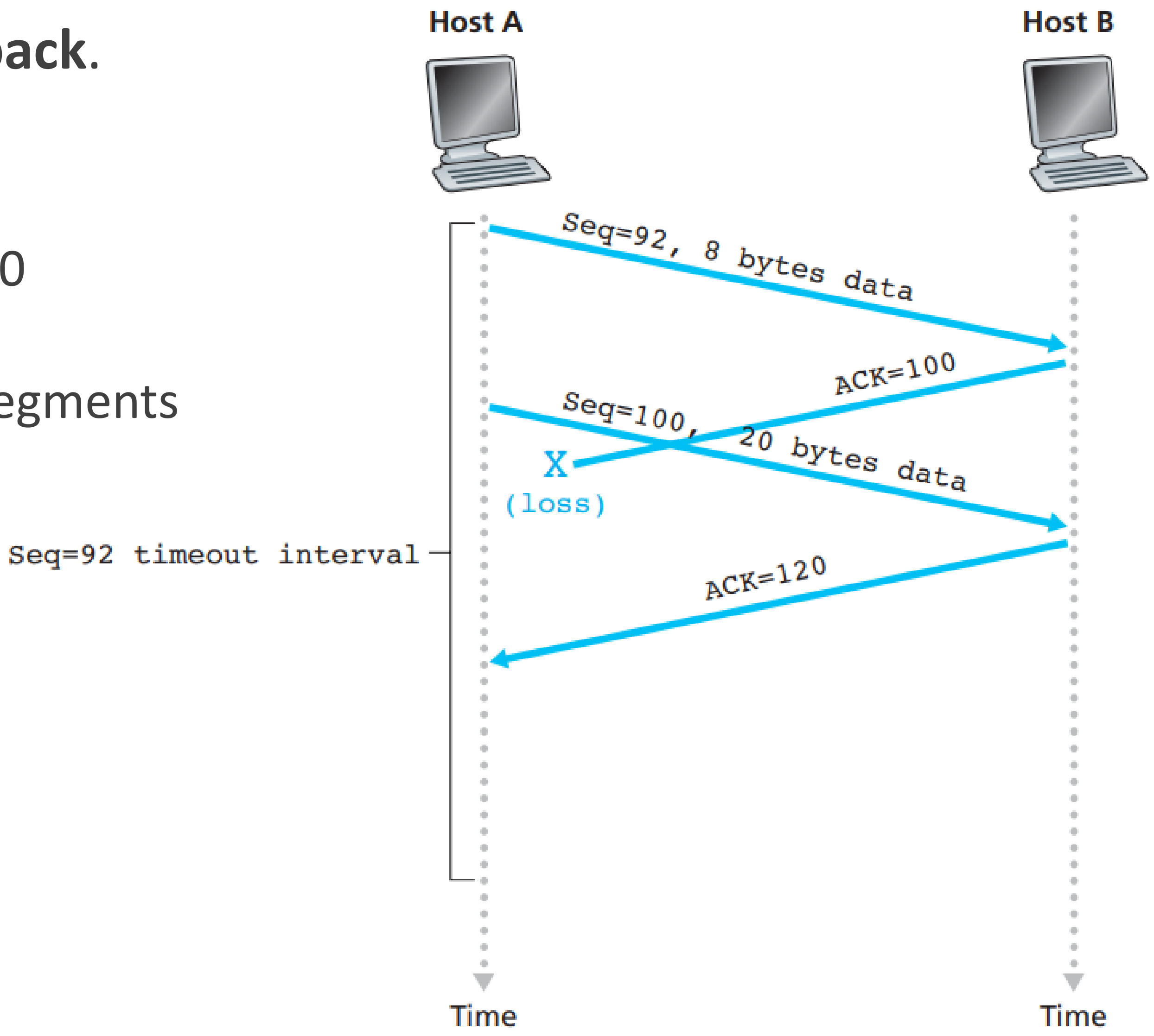
- Host A sends **two segments** to Host B **back-to-back**.
  - First segment seq# = 92, size = 8 bytes.
  - Second segment seq# = 100, size = 20 bytes.
  - Receiver – segments arrive → send ACKs (100 & 120).
  - Sender – first segment timeout before ACKs arrive → retransmit seq# = 92, restart timer.
  - Sender – if second ACK = 120 arrives before second timeout → no retransmit of seq# = 100.



No retransmission of segment 100

TCP RELIABLE DATA TRANSFER: EXAMPLES (3)

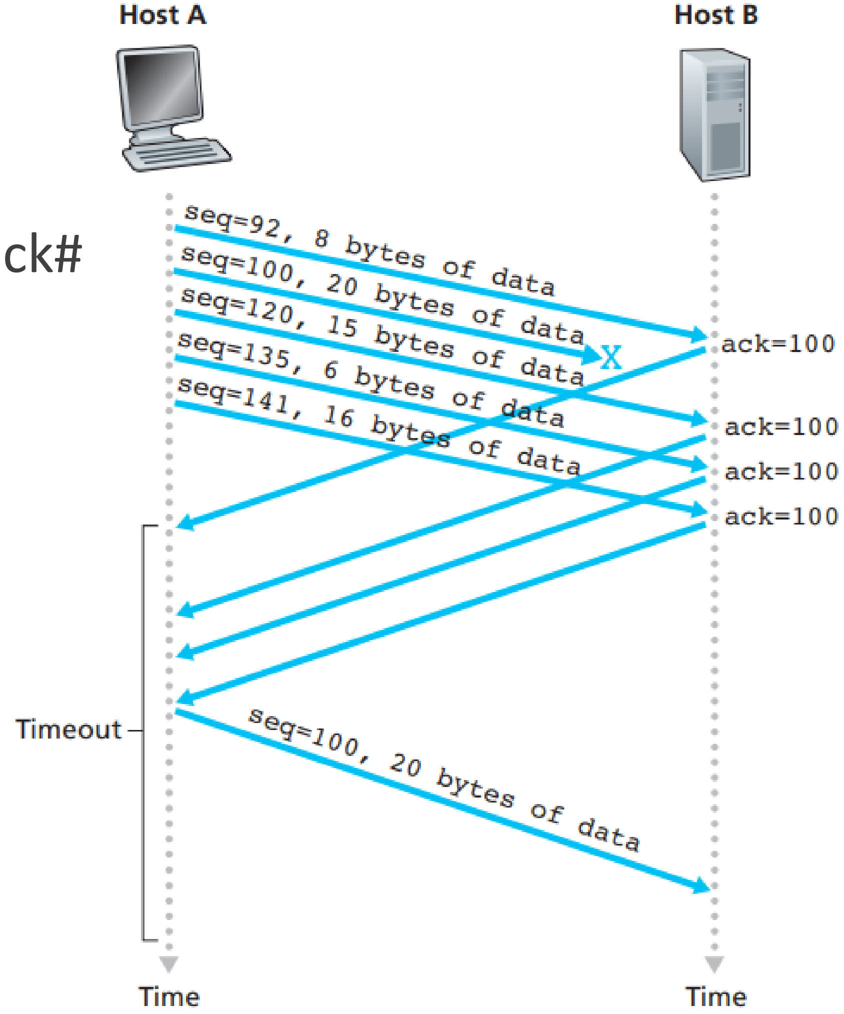
- Host A sends **two segments** to Host B **back-to-back**.
  - First segment seq# = 92, size = 8 bytes.
  - Second segment seq# = 100, size = 20 bytes.
  - Receiver – first ACK = 100 is lost, second ACK = 120 delivered before timeout.
  - Sender – receives ACK = 120 → sender received segments up to byte 119 → no retransmit.



No retransmission of segment 92

TCP RELIABLE DATA TRANSFER: FAST RETRANSMIT

- **Fast retransmission** – three **duplicated** ACKs → **retransmit** before **timeout**.
  - Receiver – segment seq# > next expected in-order seq# → gap.
    - Missing segment due to lost or reordered segments.
  - Receiver – reACKs last in-order received byte → duplicated ACKs.
  - Sender – receives three duplicated ACKs → segment with seq# = ack# is lost → retransmit segment.



Fast retransmit

TCP RELIABLE DATA TRANSFER: GBN VS. SR

- TCP uses **cumulative** ACKs.
  - Trait of **GBN** approach.
- TCP **buffers** received **out-of-order** segments & *potentially* **retransmits** only **lost** segment.
  - Traits of **SR** approach.
- TCP is a **hybrid** of **GBN** & **SR** approaches.

Supplementary material: TCP reliable data transfer pseudo code

TCP animation

SUMMARY

- Principles of reliable data transfer.
- Automated repeat request (ARQ).
- Stop-and-wait.
- Sliding window.
- Go-back-N (GBN).
- Selective repeat (SR).
- TCP reliable data transfer.