Computer Network I

Reti di Calcolatori I

Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Informatica

Riccardo Caccavale

(riccardo.caccavale@unina.it)

Transport Layer
Reliable Data Transfer Problem

- Let’s assume we are at the railway station waiting for the train number 6 which should arrive on platform 5.
- The train changes platform, it will arrive on platform 9 instead of platform 5, a message is sent from station control to communicate it:

  “Train 6 will arrive on platform 9”

- What happens if the communication is unreliable:
  - Words could be lost: “Train %&! will arrive on platform 9”
  - Words could be altered: “Train 7 will arrive on platform 9”
  - Words could be swapped: “Train 9 will arrive on platform 7”
  - Words could be duplicated: “Train 66 will arrive on platform 9”
- It is possible you can understand that the message is wrong (for example in case 1), but you may also take the wrong train or miss the right one.

Transport Layer

Reliable Data Transfer

- Error control (e.g., through checksum) allows receiver at least to understand **if the message is corrupted**, but this is not enough. Reliable data transfer **is still one of the main problem of networking**.
  - The problem is not only addressed at the transport layer, but also at the link layer and (often) at the application layer.

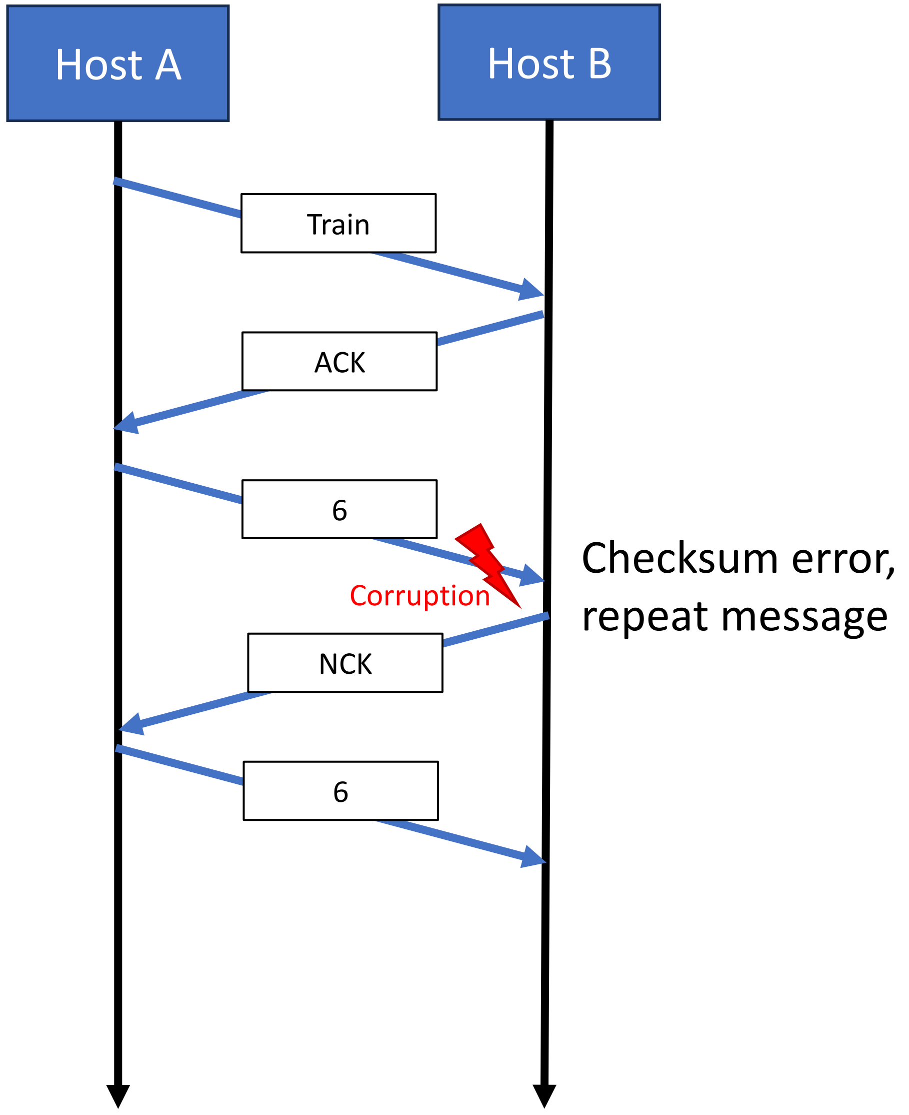
- In a reliable channel 3 elements must be guaranteed:
  1. None of the transmitted bits **are corrupted** (flipped from 0 to 1, or vice versa).
  2. None of the transmitted bits **are lost or repeated**.
  3. All bits are delivered **in the exact order** in which they were sent.

- In general, we must assume the lower-level **network layer to be unreliable**.
  - This is a realistic assumption, for example, TCP is a reliable data transfer protocol that **is implemented on top of an unreliable (IP) end-to-end network layer**.

Transport Layer

Reliable Data Transfer: Packet Corruption and Stop-and-wait

- Let’s assume we have **a channel where bits may only be corrupted** (not lost).
  - Corruption of bits is a typical problem, which is **mainly due to the physical components** of a network as a packet is transmitted, propagated, or buffered several time during the communication.
- **Stop-and-wait**: a first approach requires each message to be acknowledged before sending a new one:
  - **Positive acknowledgments** (ACK) means that the message has been received intact.
  - **Negative acknowledgments** (NCK) means that an error occurred, so the message must be repeated.
- In a computer network, **protocols based on retransmission are also known as ARQ** (Automatic Repeat reQuest) protocols.

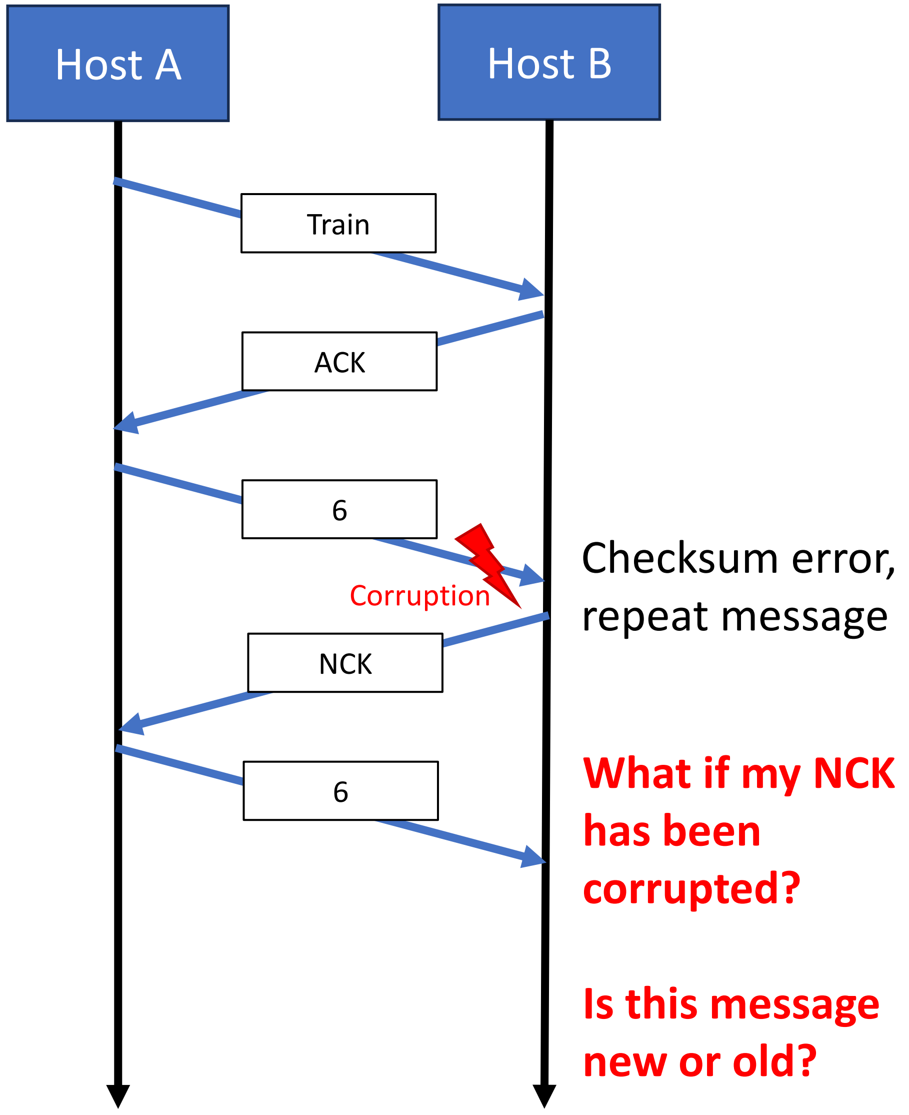


Transport Layer

Reliable Data Transfer: Stop-and-wait

- There are a couple of **issues** with this approach:
  - What happens if the **ACK/NCK message is corrupted** itself?
  - How can Host B be sure that **a message is the repetition instead of a new message**?

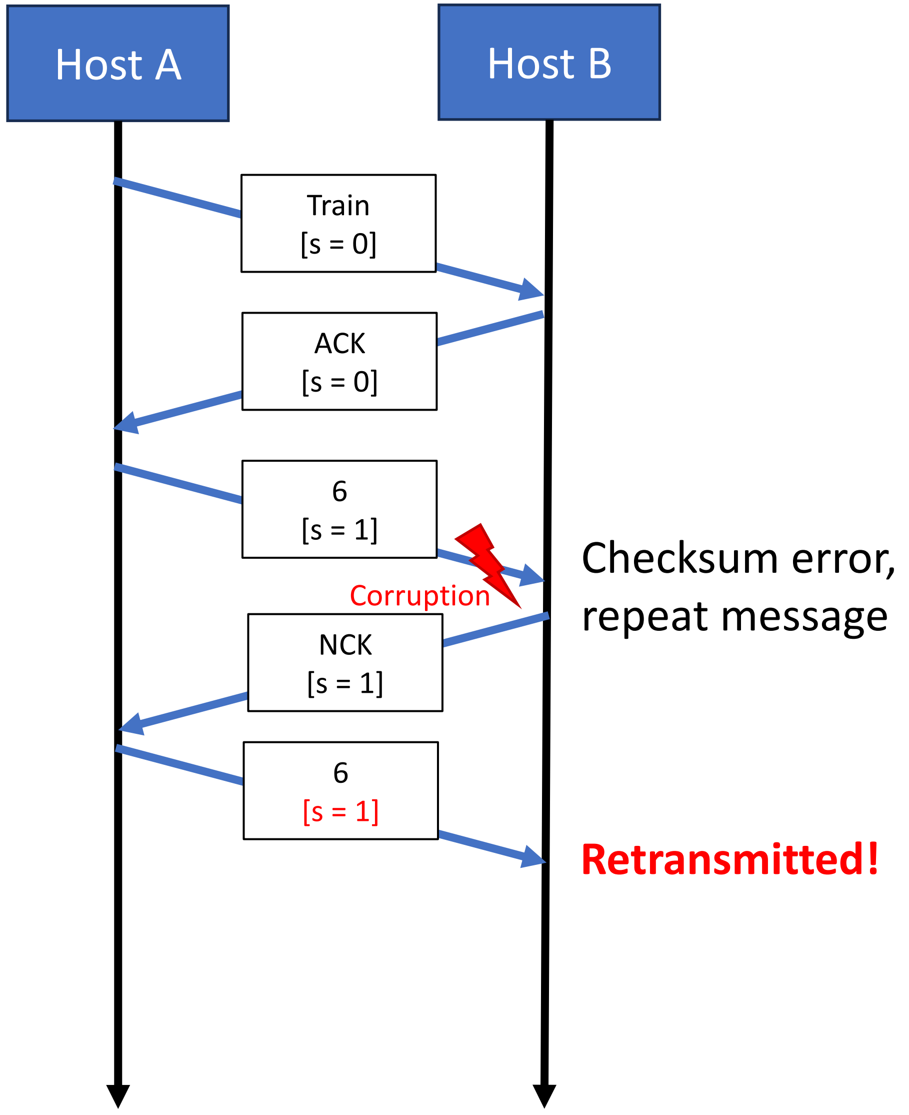
- **Sequence number**: add a new field to the headers of packets which specifies the order in which packages should be received.


Host A
Host B
Train
ACK
6
Corruption
Checksum error, repeat message
NCK
6
What if my NCK has been corrupted?
Is this message new or old?

Transport Layer

Reliable Data Transfer: Stop-and-wait

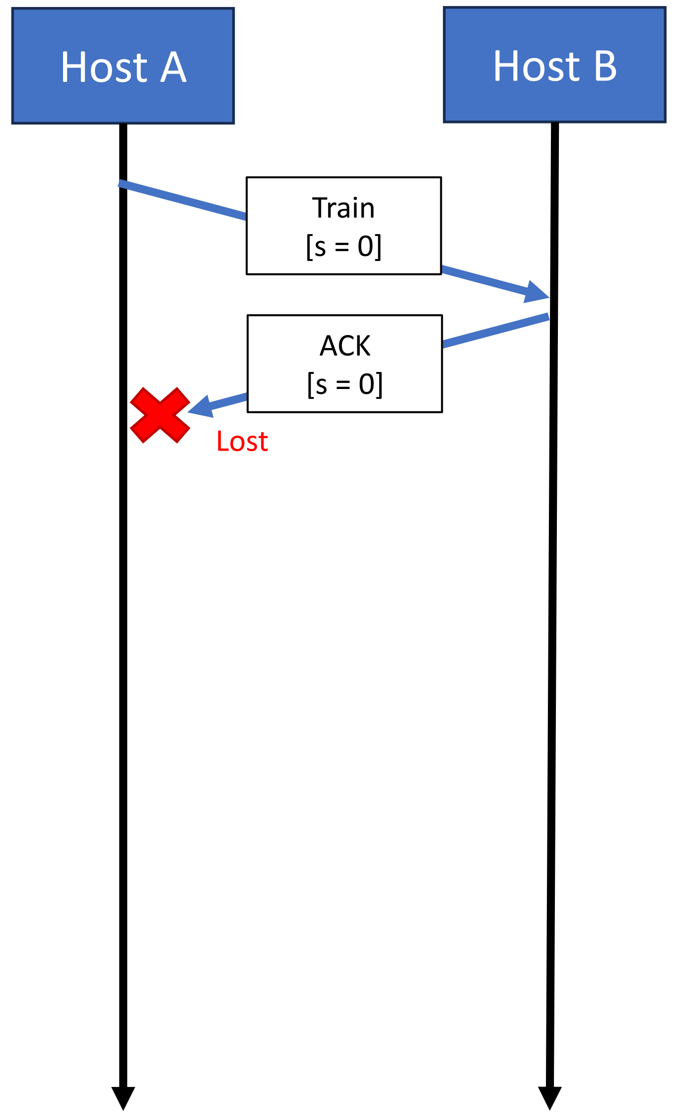
- In a stop-and wait approach we always have **one message** on the line.
- There is no need to specify the whole sequence of packets, we just need to **differentiate between the current packet and the previous one**.
- Therefore, in this case, **we just need to add one bit** (*s = 0/1*) to the headers of packets.



Transport Layer

Reliable Data Transfer: Packet Loss and Stop-and-wait

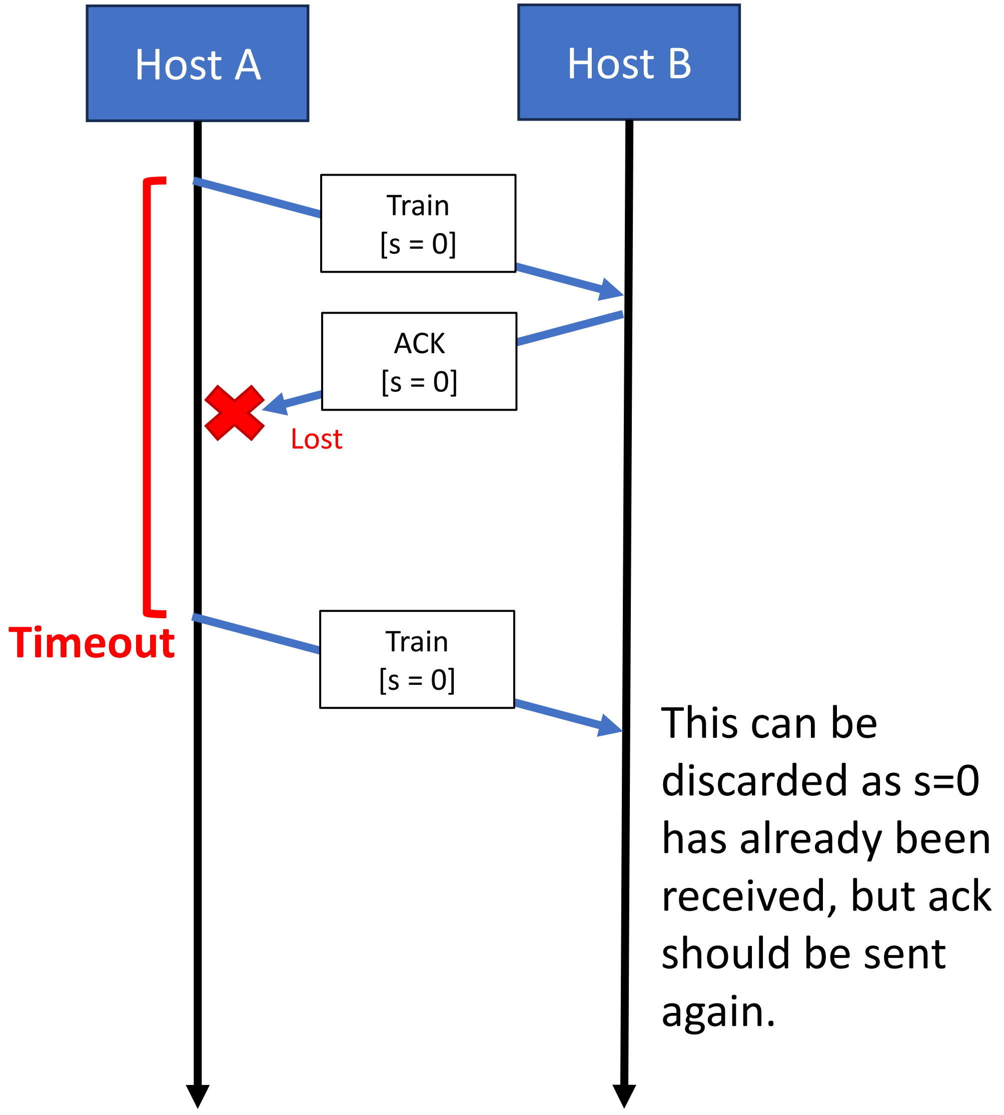
- Let's assume we now have **a channel where packets may also be lost**.
  - This is also quite reasonable as **network devices may have buffer overflow** in case of intense traffic.
- Here we have a clear problem with stop-and-wait approach: the loop. **If one message is lost, hosts will never send a new one**.
  - Notice that this happens whether we loose message or acknowledgment, as in both cases host A will not be triggered to send a new message.



Transport Layer

Reliable Data Transfer: Packet Loss and Stop-and-wait

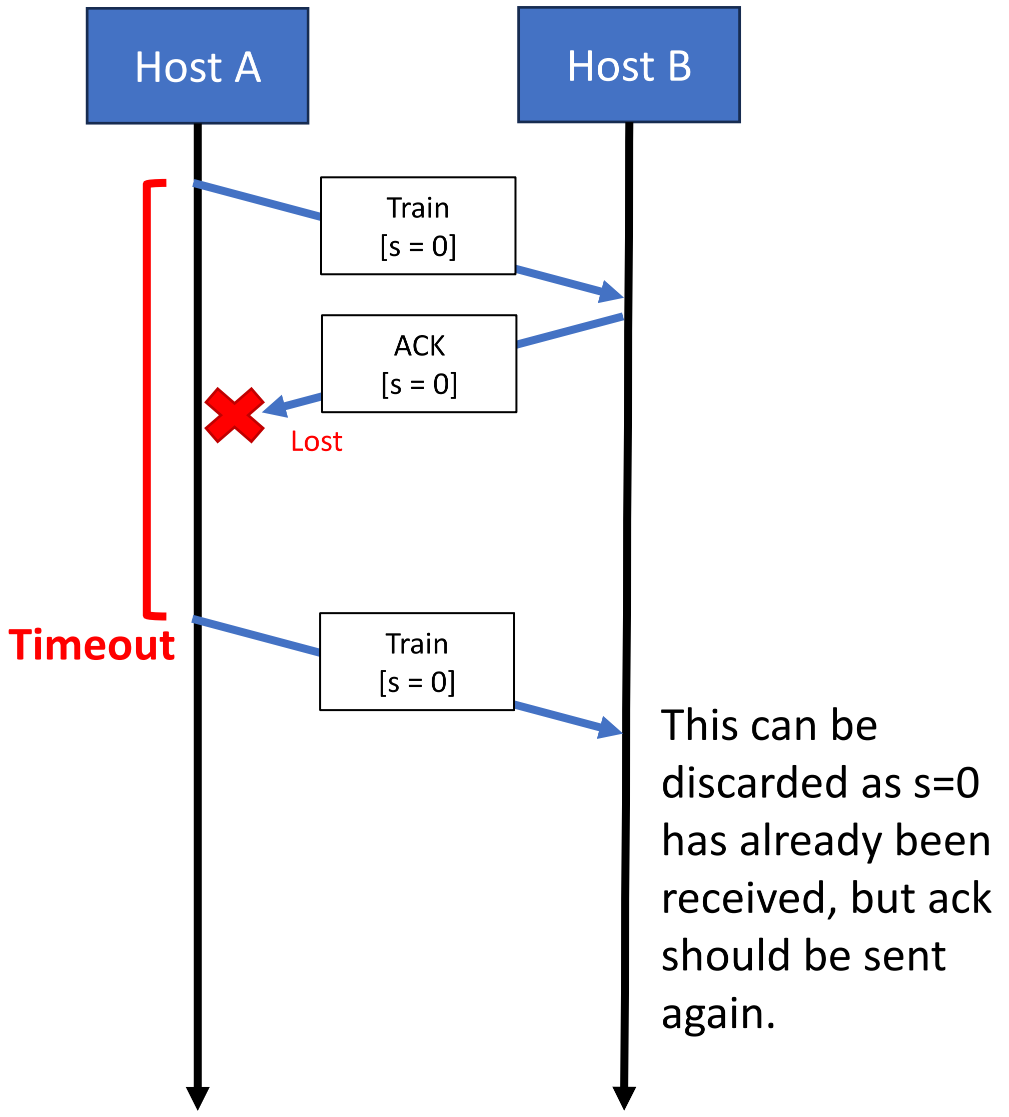
- **Timeout**: a very simple yet effective solution is to **add a timeout on the sender-side** that, once expired, allows the Host to try again.
    - A similar approach can be used in UDP-based DNS.
- This works **both in the case of packet loss and in acknowledgment loss**.
    - In case of duplicate the receive has just to discard the packet.


Host A
Host B
Train
[s = 0]
ACK
[s = 0]
Lost
Timeout
Train
[s = 0]
This can be discarded as s=0 has already been received, but ack should be sent again.

Transport Layer

Reliable Data Transfer: Packet Loss and Stop-and-wait

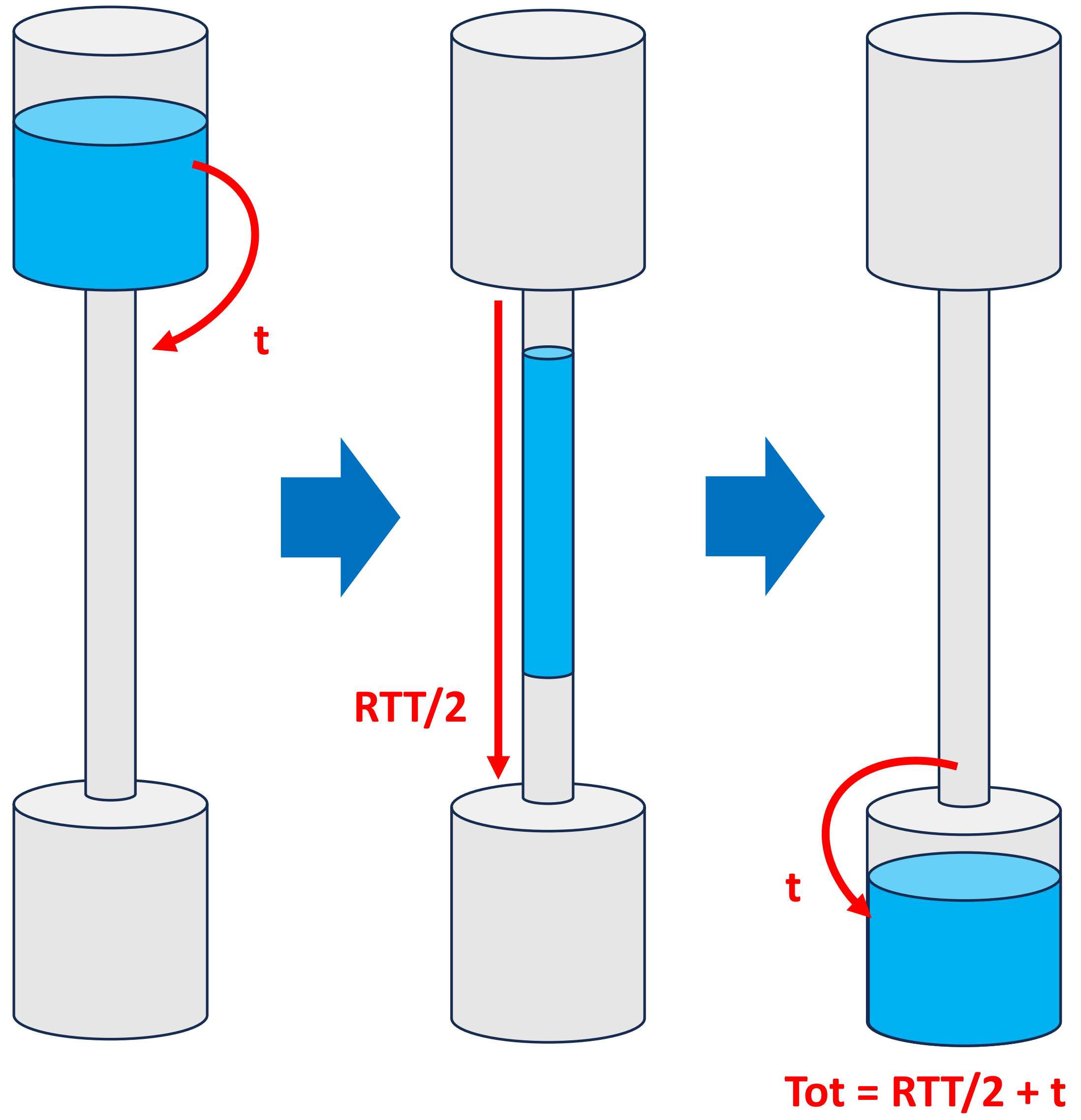
- The challenge here **is how to estimate a suitable timeout** (it depends on the RTT).
  - Too long timeout causes communication to be slowed.
  - Too short timeout causes the packets to overlap.
- We can say that a reasonable timeout should be somehow longer than the RTT.
- Timeout saves stop-and-wait from loop, but the **performance are quite bad**…


Host A
Host B
Train [s = 0]
ACK [s = 0]
Lost
Timeout
Train [s = 0]
This can be discarded as s=0 has already been received, but ack should be sent again.

Transport Layer

Reliable Data Transfer: Water Example

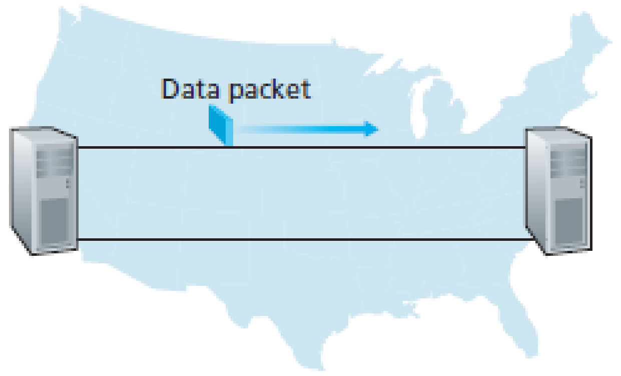
- We can see **data transfer as a water flow problem**:
  - There is **the time (t) to move the water from the tank to the tube**. While entering **the water is also moving** toward the tube.
  - After **t all the water from the tank is inside the tube** (the initial tank is empty as well as the destination tank).
  - It **takes a certain time (RTT/2) for the water to flow** into the tube.
  - Finally, **the water arrives at the destination tank** and (by assuming same download/upload time) it takes the same time t to pour out of the tube (RTT/2 + t to receive the last drop).



Transport Layer

Reliable Data Transfer: Performance of Stop-and-Wait

- Let's consider the idealized case of **two hosts located on the opposite coasts** of the United States.



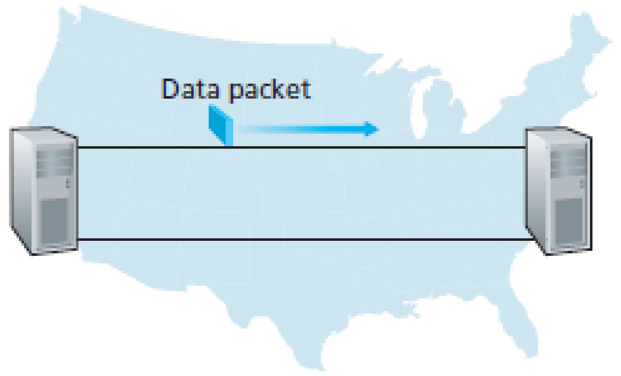
- The speed-of-light RTT between these two end systems **is approximately 30 milliseconds** (0.03 sec.). Let's assume to have:
  - A channel with a transmission rate (*R*) of 1 Gbps (10^9 bits per second).
  - A packet size (*L*) of 1000 bytes (8000 bits).
- The time (*t*) needed to transmit the packet to the channel is:

$$t = \frac{L}{R} = \frac{8000}{10^9} = 0.000008 \text{ (8 microseconds)}$$

Transport Layer

Reliable Data Transfer: Performance of Stop-and-Wait

- Let’s consider the idealized case of **two hosts located on the opposite coasts** of the United States.

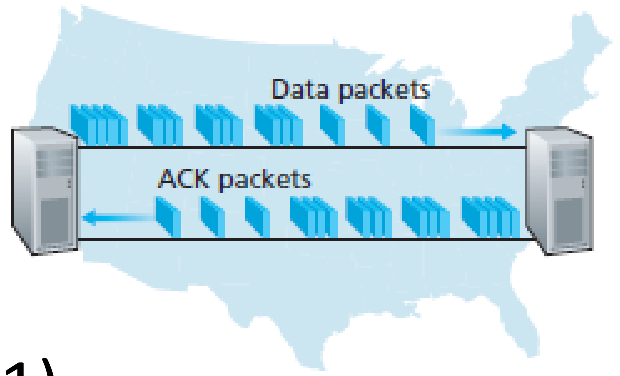


- In a **stop-and-wait** protocol:
  - the **sender begins sending** the packet at t_0,
  - the **last bit enters the channel** at t_0 + 0.000008 sec.,
  - the **packet takes 0.015 sec. to reach** the receiver,
  - the **last bit is received** at t = RTT/2 + L/R = 0.015008 sec.,
  - assuming that ACK packets are extremely small (their **transmission time can be neglected**) the ACK emerges back at the sender at t = RTT + L/R = 0.030008 sec.
- In 0.030008 sec. of total transmission time, the sender was waiting almost all the time (99.973% of the time).

Transport Layer

Reliable Data Transfer: Performance of Pipelining

- **Pipelining**: instead of stop-and-wait, the sender is **allowed to send multiple packets** without waiting for acknowledgments.

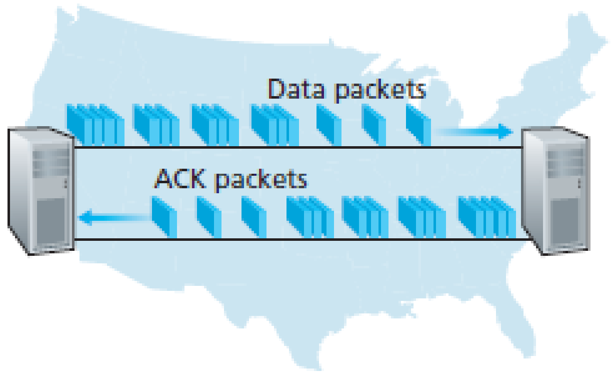


- In pipelining approaches (e.g., 3 packets instead of 1):
  - the **sender begins sending the 3 packets** at t_0,
  - the **last bit of the last packet enters the channel** at t_0 + 0.000024 sec.,
  - the **packets take 0.015 sec. to reach** the receiver,
  - the **last bit is received** at t = RTT/2 + L/R = 0.015024 sec.,
  - assuming that ACK packets are extremely small (their **transmission time can be neglected**) the ACK emerges back at the sender at t = RTT + L/R = 0.030024 sec.
- In 0.030024 sec. **the sender was waiting 0.035% less** (99.920% instead of 99.973%).

Transport Layer

Reliable Data Transfer: Performance of Pipelining

- **Pipelining**: instead of stop-and-wait, the sender is **allowed to send multiple packets** without waiting for acknowledgments.

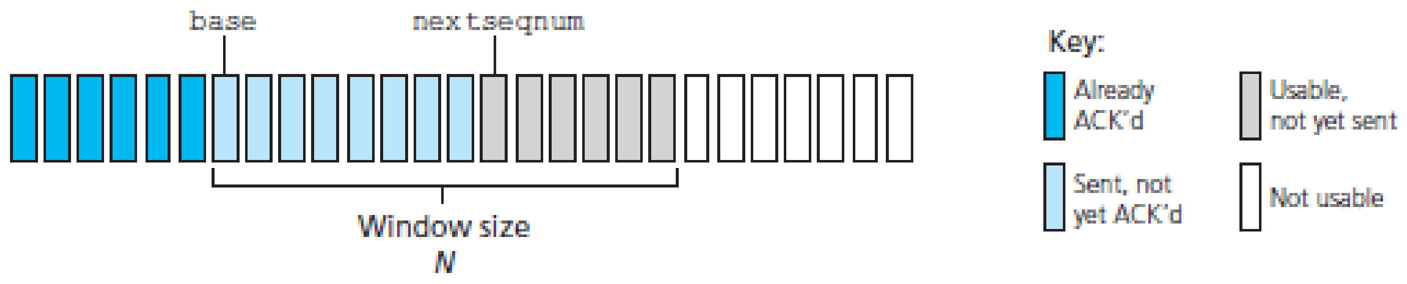


- In pipelining approaches:
  - The **range of sequence numbers must be increased** since each in-transit packet (not counting retransmissions) must have a unique sequence number and there may be multiple, in-transit, unacknowledged packets.
  - We **need buffers to store incoming packets** since we don’t know if packets are correct or there are “holes” in the transmission.
- There are 2 basic protocols using pipelining:
  - **Go-Back-N**.
  - **Selective Repeat**.

Transport Layer

Reliable Data Transfer: Go-Back-N

- **Go-Back-N (GBN)** protocol (aka sliding-window protocol): the sender is allowed to transmit multiple packets (when available) **without waiting for an acknowledgment** but is constrained to have **no more than N unacknowledged packets**.



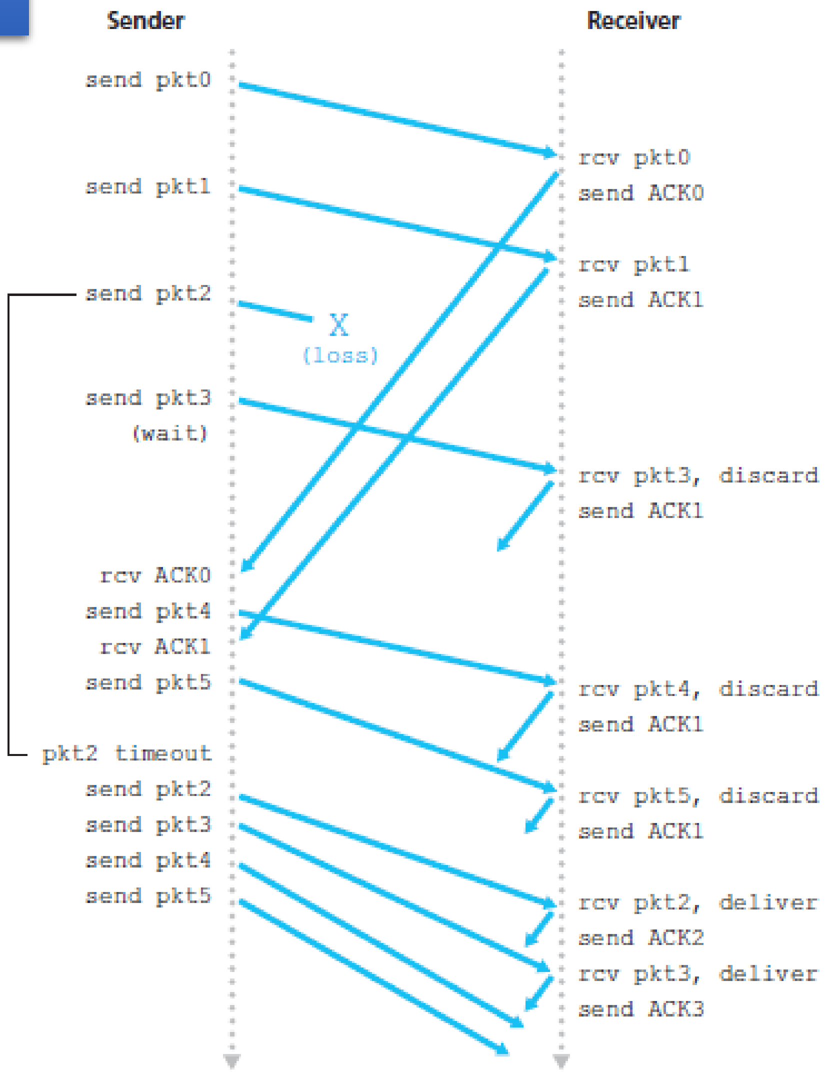
- Base: is the **sequence number of the oldest unacknowledged packet**.
- Nextseqnum: is the **smallest unused sequence number** (next to be sent).
- We have that:
  - Packets in [0, base-1] have been **transmitted and acknowledged**.
  - Packets in [base, nextseqnum-1] have been **sent but not yet acknowledged**.
  - Packets in [nextseqnum, base+N-1] **can be sent**.
  - Packets in [base+N, +inf] **cannot be used** until a new acknowledgment is received.

Transport Layer
Reliable Data Transfer: Go-Back-N

- What about the receiver? In GBN protocol **receiver is quite simple**.

- The receiver has **simply to discard out-of-order packets** (whether they are damaged or not) and to deliver to the upper level (application) in-order packets only.
    - Discarded packets (not acknowledged) will be eventually retransmitted by the sender.

- With this approach good packets are also discarded but the overall process is quite simple:
    - the **sender must maintain the indices of the window**,
    - the **receiver only needs to maintain the sequence number** of the next in-order packet.



Since pkt2 is lost, all pkt3, pkt4, pkt5, must be discarded even if correct.

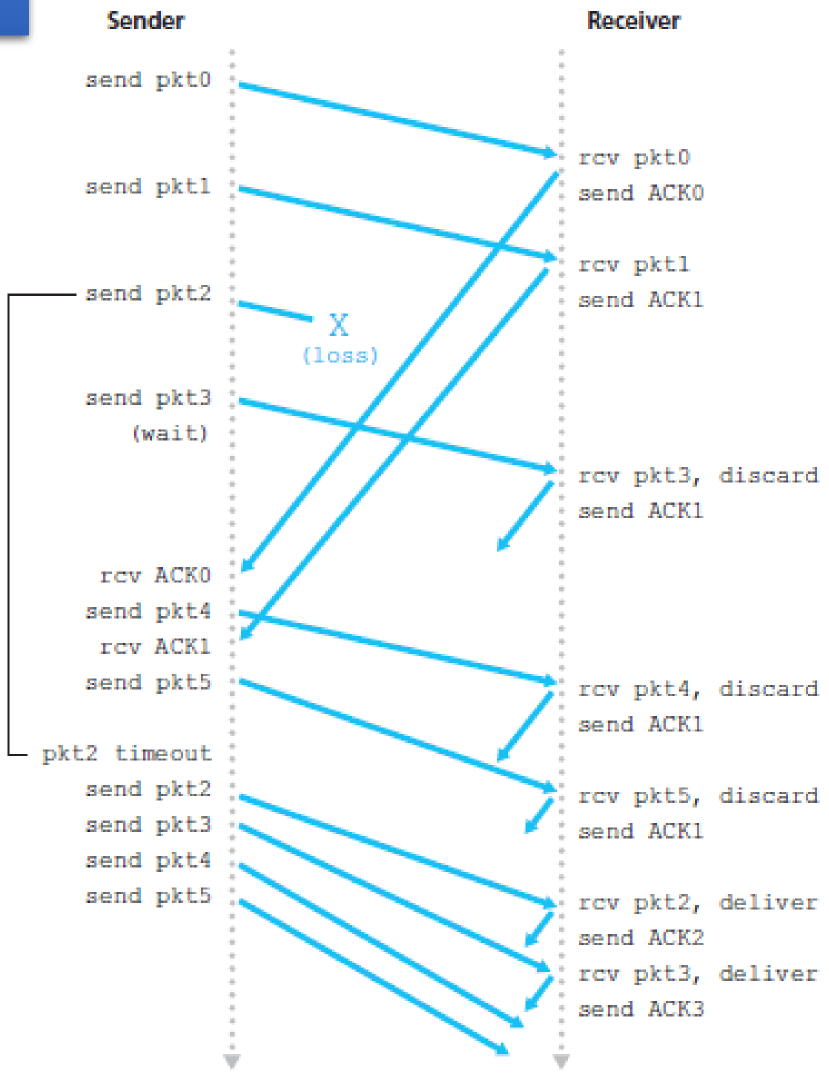
Transport Layer

Reliable Data Transfer: Go-Back-N

- Of course, the disadvantage of throwing away a correctly received packet is that **we need to resend it again**.

- This is a **chain-reaction**, if we are losing packets due to network difficulties many correctly received but out-of-order packets may be discarded forcing the sender to retransmit them.

- The retransmission itself might be lost or damaged and thus **even more retransmissions would be required**.

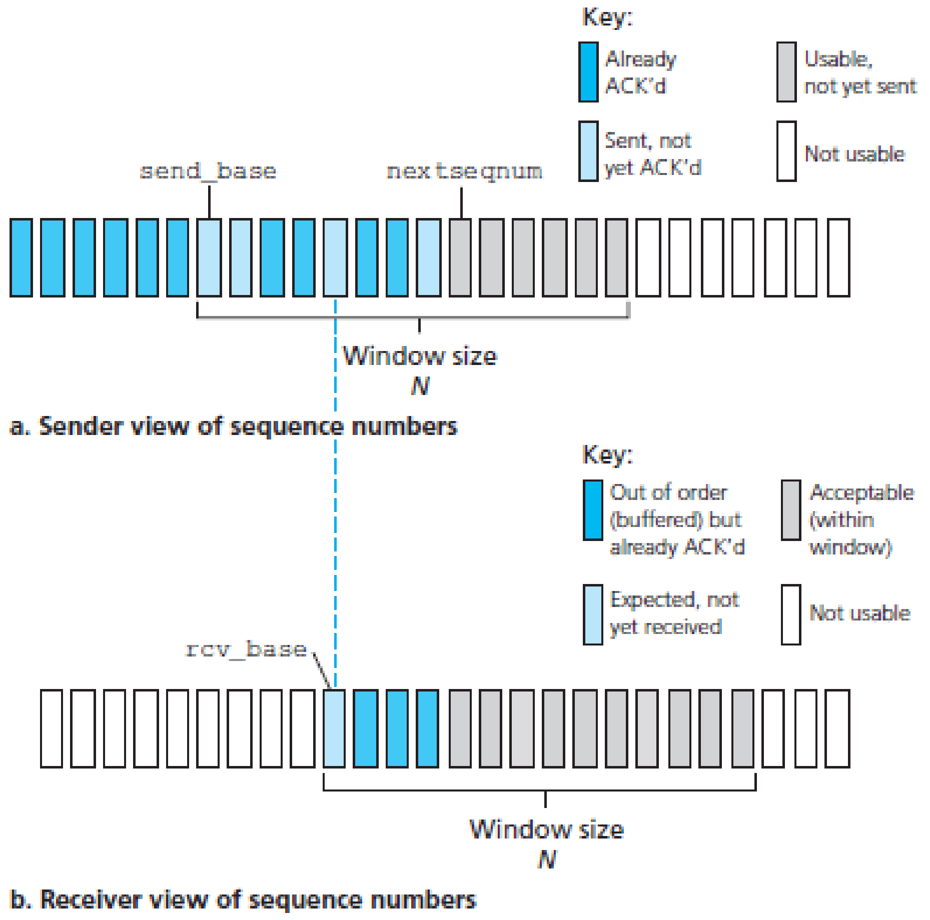


Not only pkt2, but also pkt3, pkt4, pkt5, must be transmitted again.

Transport Layer

Reliable Data Transfer: Selective Repeat

- Despite GBN is more effective than stop-and-wait, it may **suffer from performance problems**.
- Considering large window size**, a single packet error can cause GBN to retransmit a large number of packets**, many unnecessarily.
- In **selective-repeat** (SR) protocols: the sender retransmit **only those packets that were likely received in error** (lost or corrupted):
  - As in GBN **individual packets are acknowledged**.
  - A **window size of N is again used** to limit the number of unacknowledged packets.
  - Unlike GBN, **out-of-order packets are buffered** (stored) and acknowledged by the receiver.

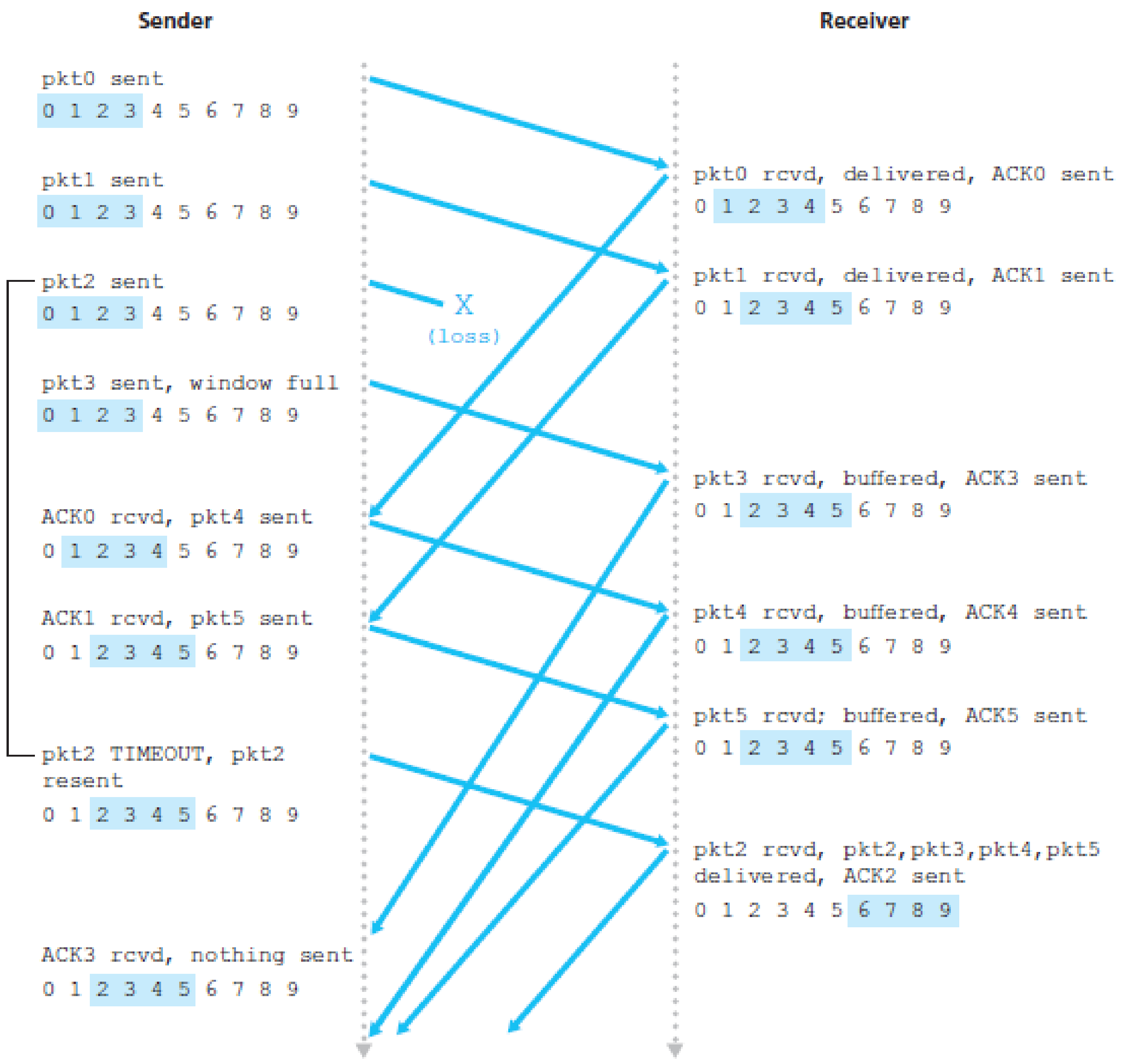

Key:
Already ACK'd
Usable, not yet sent
Sent, not yet ACK'd
Not usable
send_base
nextseqnum
Window size N
a. Sender view of sequence numbers
Key:
Out of order (buffered) but already ACK'd
Acceptable (within window)
Expected, not yet received
Not usable
rcv_base
Window size N
b. Receiver view of sequence numbers

Transport Layer

Reliable Data Transfer: Selective Repeat

- Example of SR operation in the presence of lost packets: the receiver initially buffers pkt3, pkt4, and pkt5, while waiting for pkt2 (lost) to be retransmitted.

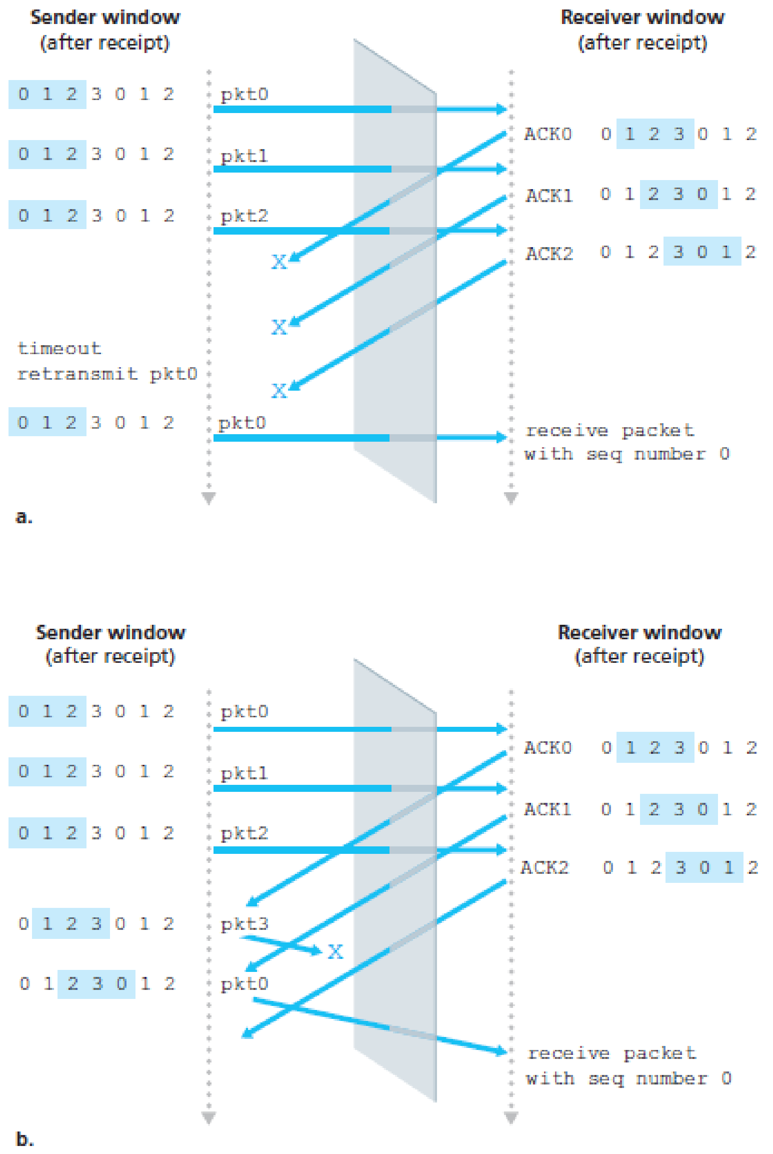
- Despite the previous example, here we avoid to resend pkt3, pkt4, pkt5, in so avoiding additional transmission loss or errors.



Transport Layer

Reliable Data Transfer: Selective Repeat

- One issue in SR is that the window size is related to the sequence number, **and sequence number is finite**.
- In this example we have four packets, **a max sequence number of 3** and a window size of 3.
- Let's assume packets 0 to 2 are correctly received and acknowledged, hence receiver's window moves to 6th packet (i.e., to packets 3, 0, 1):
  - **Case a**: the ACKs for the first three packets are lost and the sender retransmits these packets. *Is packet 0 new or old?*
  - **Case b**: the ACKs are received, packets 3 and 0 are sent but packet 3 is lost. *Is packet 0 new or old?*
- To avoid this issue, **the sequence number must be at least 2 times the window size**.


Sender window (after receipt)
Receiver window (after receipt)
0 1 2 3 0 1 2
pkt0
pkt1
pkt2
ACK0
ACK1
ACK2
x
timeout retransmit pkt0
receive packet with seq number 0
a.
pkt3
b.