CS-340 Introduction to Computer Networking

Lecture 5: Reliable Transport

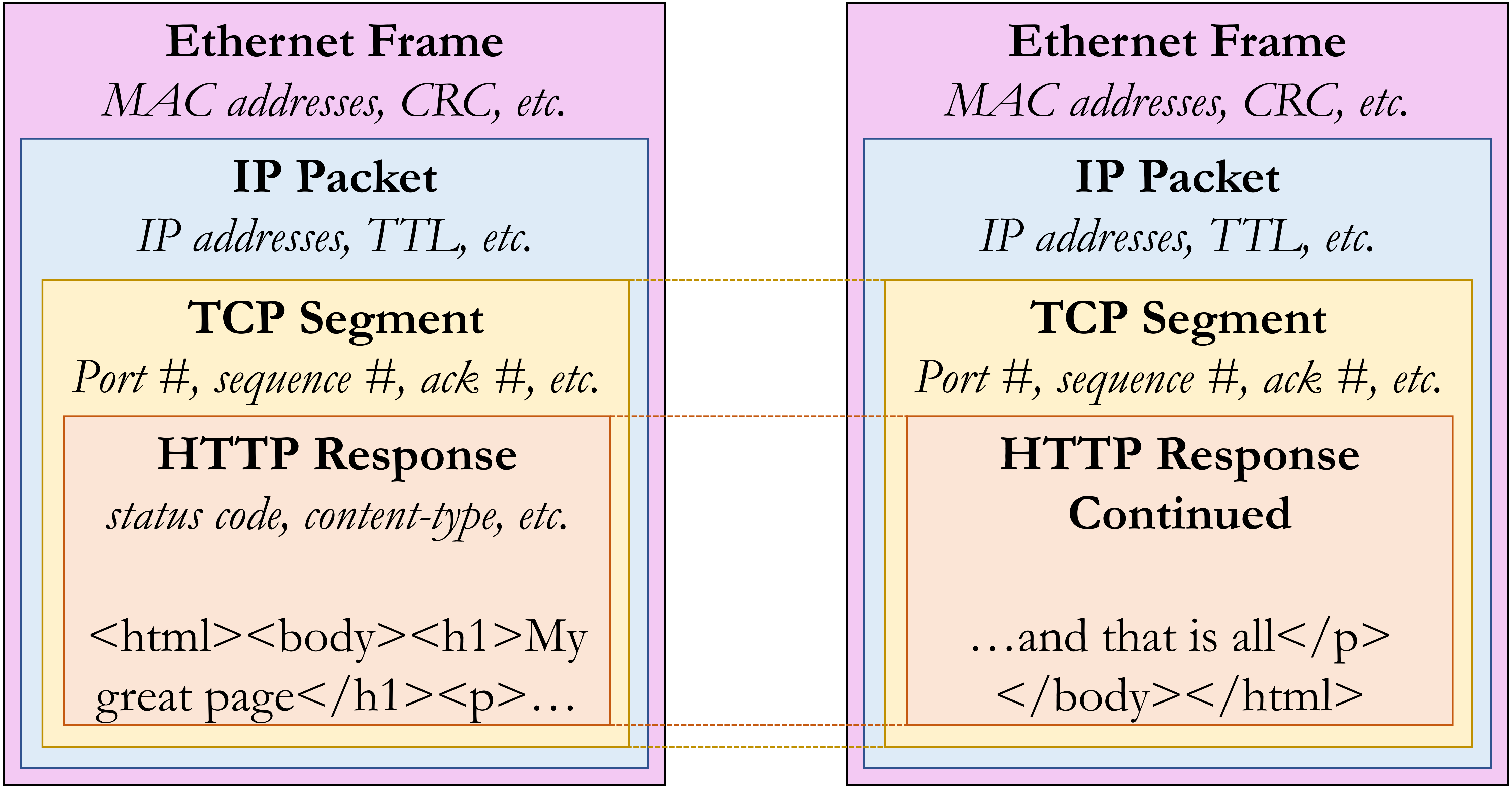
Steve Tarzia

Many diagrams & slides are adapted from those by J.F Kurose and K.W. Ross

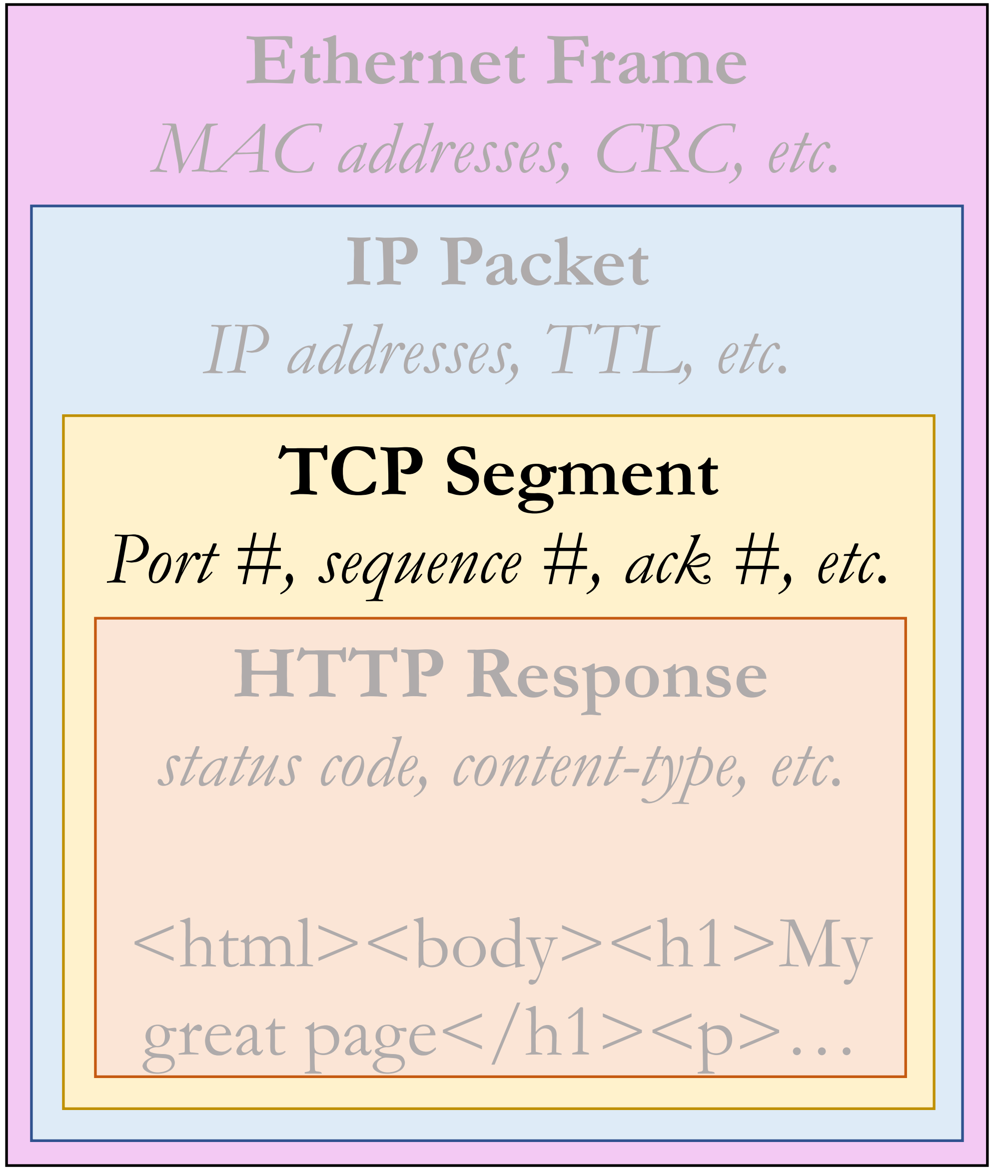
Last Lecture: Domain Name Service

- DNS is the Internet's directory service
- It's distributed and hierarchical
  - 13 Root servers are run by ICANN
  - Top level domain (TLD) servers manage com, org, edu, cn, au, uk, *etc.*
  - Each subdomain has a set of authoritative nameservers
- Various types of records exist to do more than just map name → IP
- Domain registrars are accredited by each TLD to sell names.
- Dynamic DNS servers can cleverly craft their responses to provide:
  - Load balancing and fault tolerance in a cluster of servers
  - Content Delivery Networks, that direct you to the closest service "mirror"
  - Captive portals

Recall the four main layers on the Internet

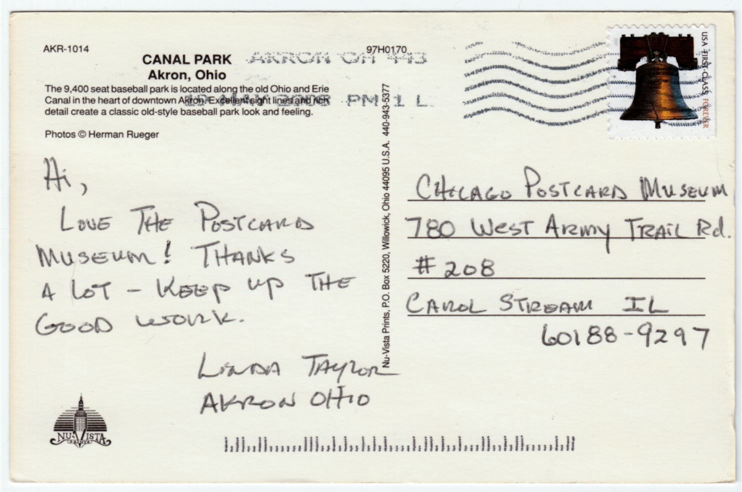


Each layer solves a subset of problems



- Link layer: shares a physical channel among several transmitters/receivers
- Network layer: routes from source to destination, along many hops.
- **Transport layer:**
  - Creates connections/sockets used by apps.
  - Multiplexing (>1 connection per machine)
  - Ordering, • Acknowledgement, • Pacing
- HTTP layer:
  - Resource urls, • Response codes,
  - Caching, • Content-types, • Compression
- None of the layers shown provide *security*.

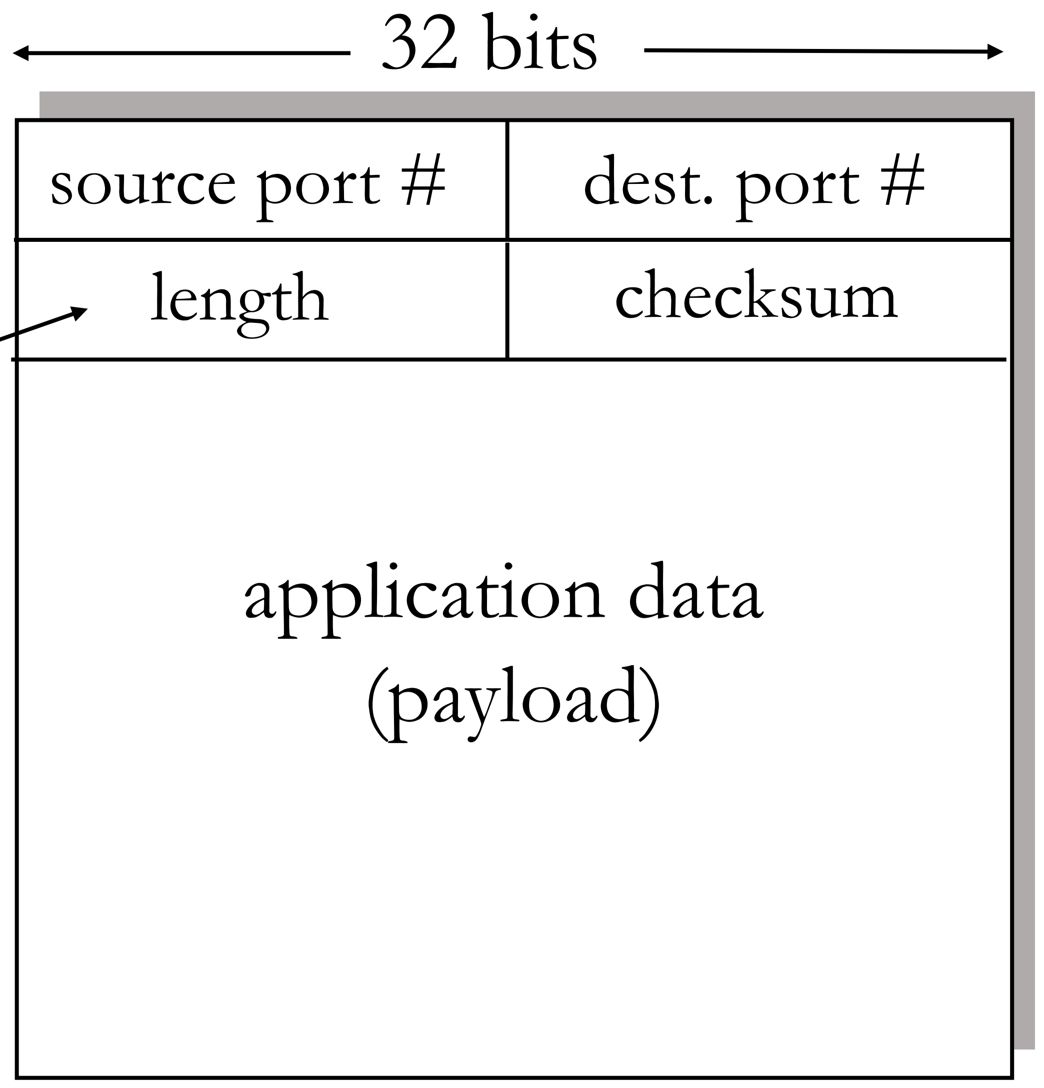
User Datagram Protocol (UDP)



- The simplest ***transport protocol*** on the Internet (simpler than TCP).
  - "*transport*" was a bad naming choice.
- Does not provide much more than the IP layer below.
  - **Datagrams are packets** sent between software applications.
  - IP layer provides "best effort" delivery. Packets may be dropped.
  - Thus, UDP is also unreliable.
- Adds to each packet:
  - A ***port number***, to distinguish different services on the machine.
    - Only one process can "listen" for packets on a given port number.
  - A ***checksum*** to verify that packet data was not corrupted.

UDP header fields

length, in bytes of UDP packet, including header



UDP packet format

Checksum is a simple way to detect data corruption

- Break the data into a sequence of 16-bit integers
- Add the integers
- *Wrap* the carry-out bits to the least-significant position.
- Finally, invert the result.

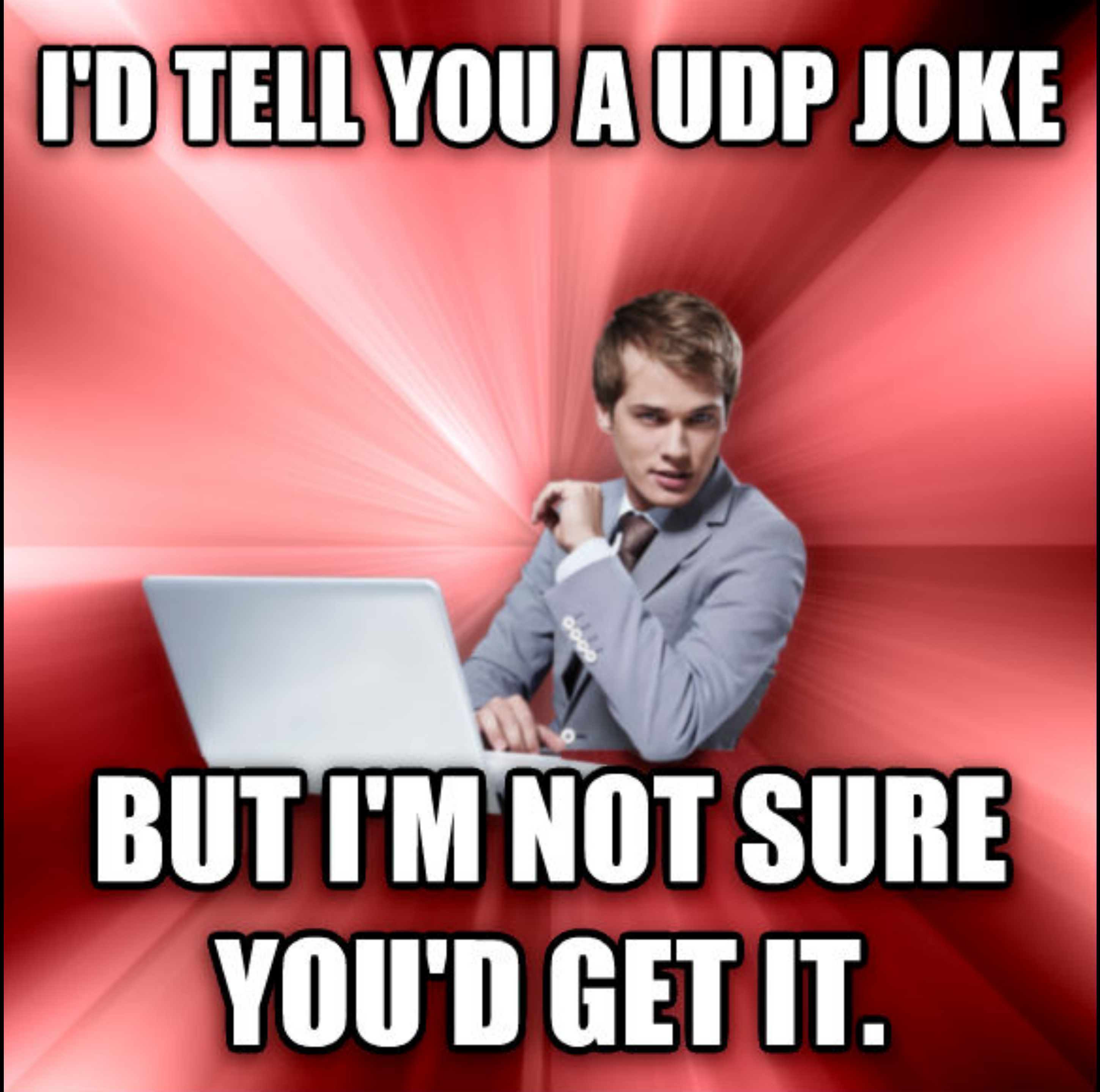
Checksum is redundant information – a summary of the packet data.

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Checksum in action

- Sender wants to send data:
  "Hello there, here is my message."
- UDP library in the sender computes a checksum as follows:
  - "He" + "ll" + "o " + "th" + "er" + "e," + " h" + "er" + "e " + "is" " m" + "y " + "me" + "ss" + "ag" + "e." = 0xB51
  - Wrap around: 0x51 + 0xB = 0x5C
  - Flip bits:
    0101 1100 → 1010 0011 = **0xA3**
- Sender adds 0xA3 checksum to UDP header of the packet.

- Receiver wants to verify the following message:
  "Hello there, here is my massage."
- UDP packet's checksum says checksum is 0xA3.
- Receiver calculates checksum of the received message, and finds that it *does not* equal 0xA3 (because a bit was flipped).
- Receiver drops the packet.
- Checksum does not repair errors, it simply lets us detect errors.


I'D TELL YOU A UDP JOKE
BUT I'M NOT SURE YOU'D GET IT.

TCP provides ***streaming*** connections to apps

TCP is usually implemented by the OS. An OS library handles the following:

- **Ordering**:
  - Data must be *packetized* (chunked) by the sender and *reassembled* by receiver
  - Reassembly is done in the proper *order*, regardless of delivery order.
- **Acknowledgement**: *(almost, but not exactly "reliability")*
  - Delivery of each packet is acknowledged, so lost packets can be *retransmitted*.
- **Pacing**:
  - Sender adjusts packet send rate so neither receiver nor network are overwhelmed.
  - Avoid filling up queues and dropping packets.

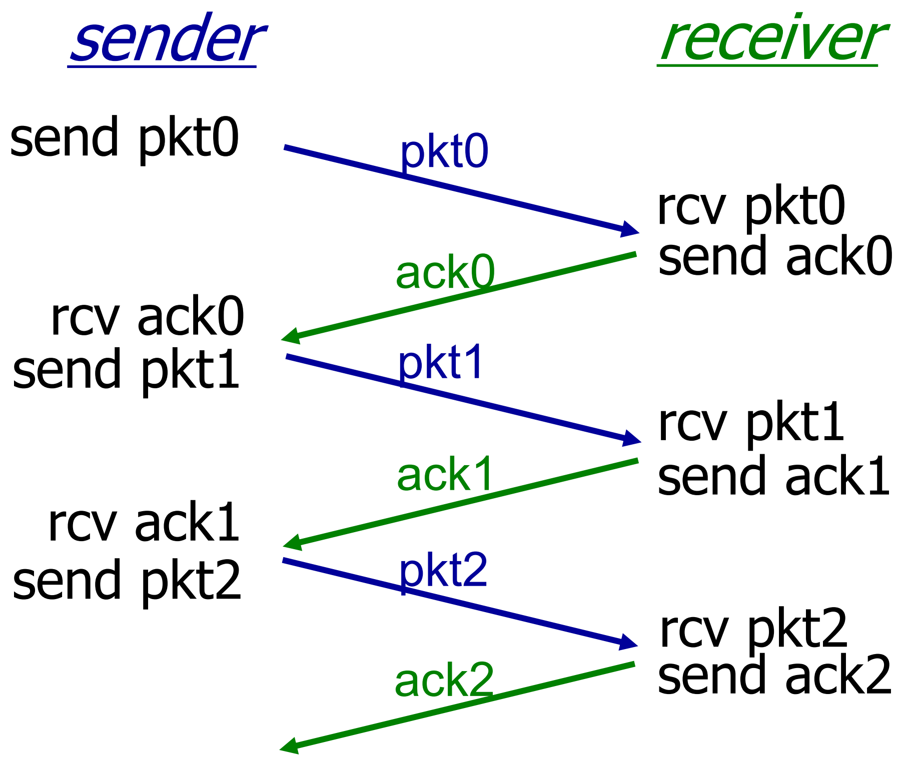
TCP deals with many underlying Internet problems:

- Packet loss/corruption *(ack./checksum)*
- Packet reordering *(seq. numbers)*
- Finite link speed & Q size *(flow & congestion control)*
- Finite packet size *(seq. numbers)*

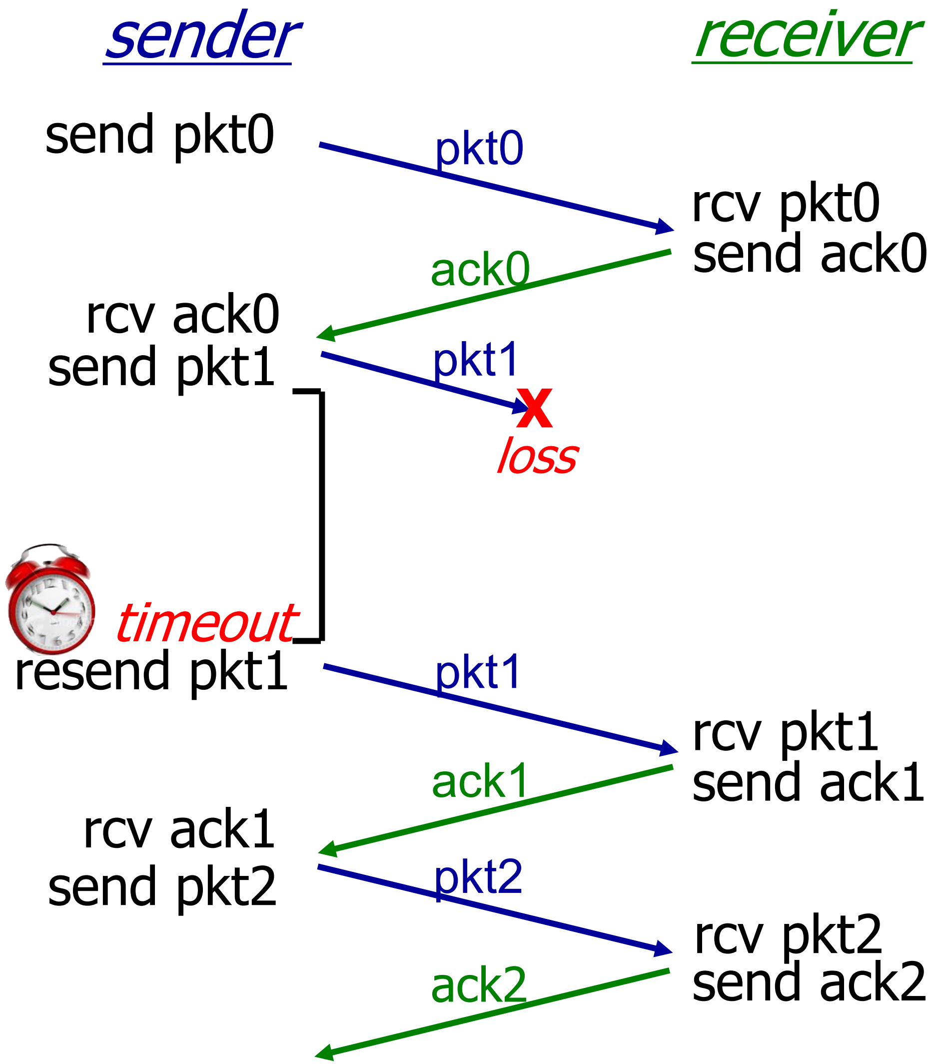
Human solutions to message loss

- How do people deal with "message loss" on the telephone?
- Listener may say *"OK"* or *"mm-hmm"* after each sentence.
  - Called *positive acknowledgement* or *ACK*.
  - If talker does not hear an ACK, then maybe she repeats herself, or asks "are you still there?"
- Listener may say *"What?"* or *"Can you repeat that?"* if message was corrupted or lost.
  - Called *negative acknowledgement* or *NACK*.
  - Talker retransmits the message in response.
- What happens if acknowledgements are lost?
  - Positive: talker cannot make progress, gives up.
  - Negative: listener cannot recover missed messages, gives up.

Naïve ACKs

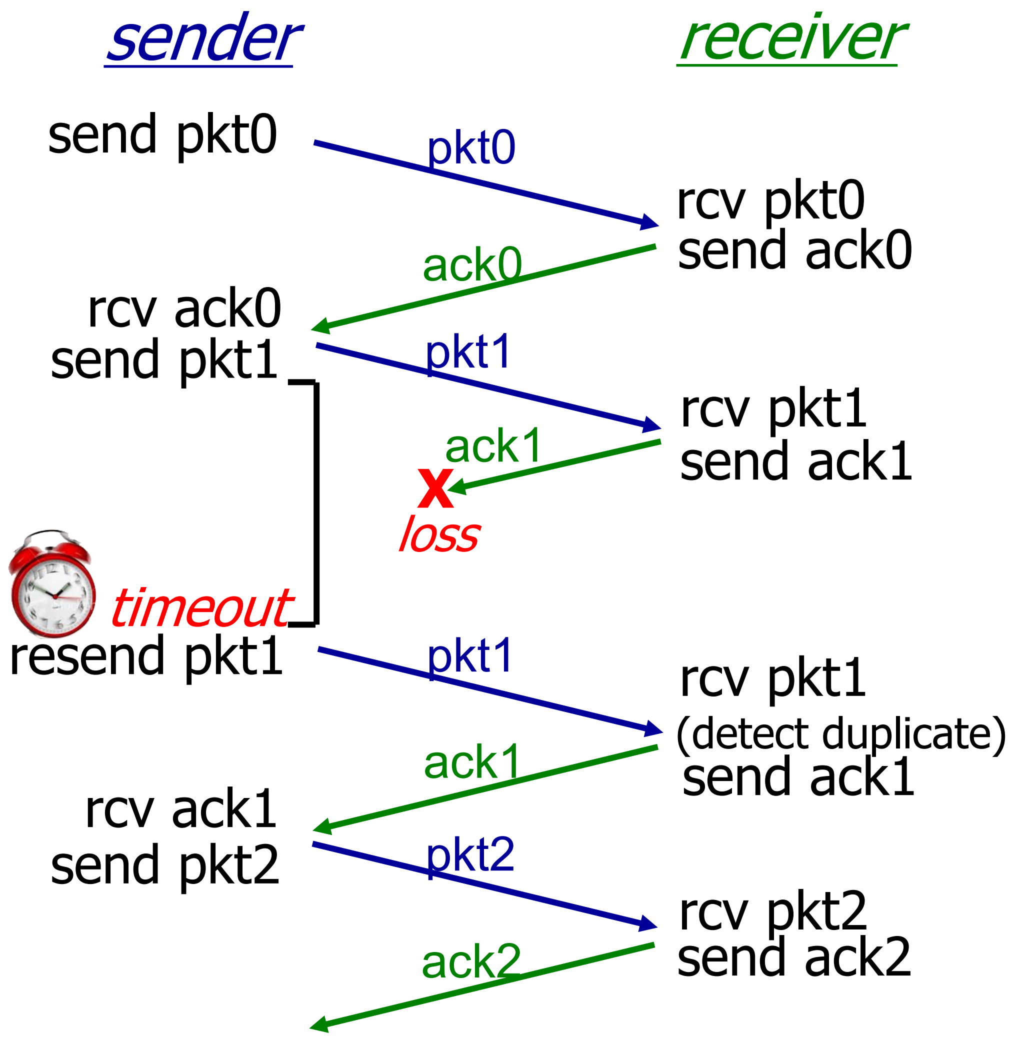


(a) no loss

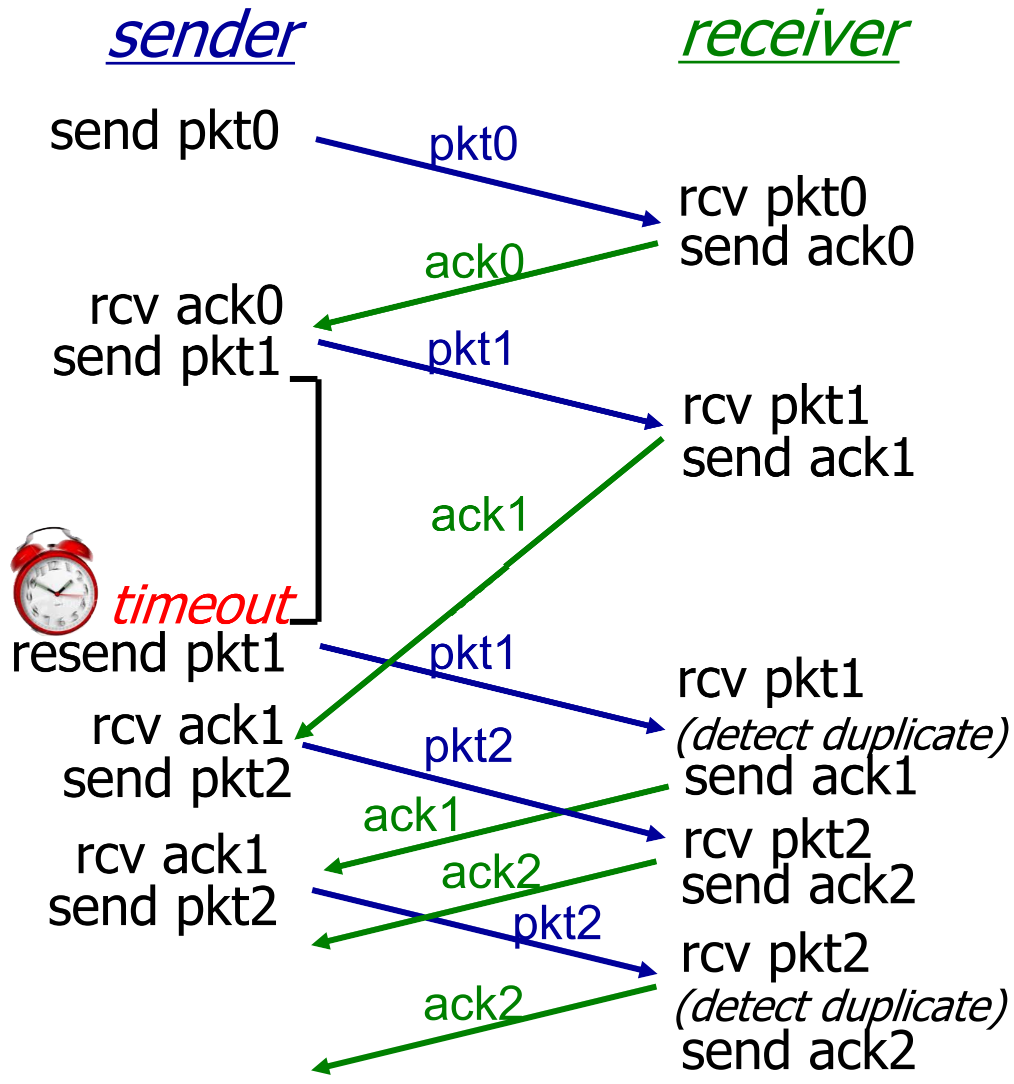


(b) packet loss

Naïve ACKs *(continued)*



(c) ACK loss



(d) premature timeout/ delayed ACK

Naïve ACK correctness

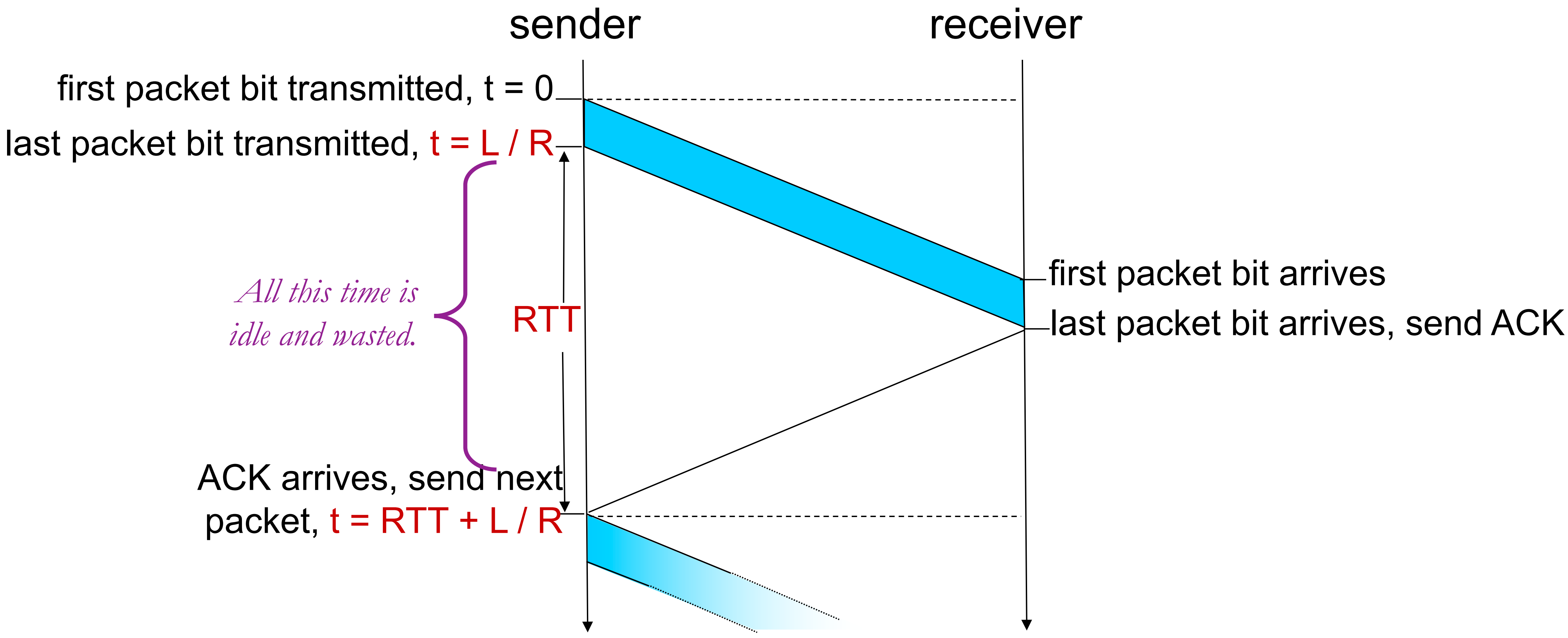
- Solves packet loss/corruption, and ordering
  - Do not send packet *n* until we get ACK *n-1*.
- **Timeout** is necessary to decide when a packet is lost
  - Sender cannot ever really know the status of a packet, unless got an ACK.
  - If timeout is premature, then sender may retry too soon. That's OK because both sender and receiver can simply discard old/duplicate packets:
  - If sender already got ACK *n*, then there is no need to send packet *n* in response to ACK *n-1*.
  - If receiver already got packet *n*, then there is no need to send ACK *n-1* in response to packet *n-1*.
- At most, twice the necessary data will be "in flight."

Naïve ACK performance

- It's a "**stop and wait**" protocol.
- Round-trip time (RTT) of packets dominates performance.
- Eg., An ISP's fiber link from New Jersey to San Jose, CA:
  1 Gbps link, 15 ms propagation delay, 1.5 kByte packet size:
  - RTT = 2 (15 ms + 1.5 kByte * 8 bit/Byte / 1 Gbps) = 30.01 ms
  - RTT is dominated by the 30 ms round-trip propagation delay.
  - Effective throughput is just 1.5 kByte * 8 bit/Byte / 30.01 ms = **250 kbps**

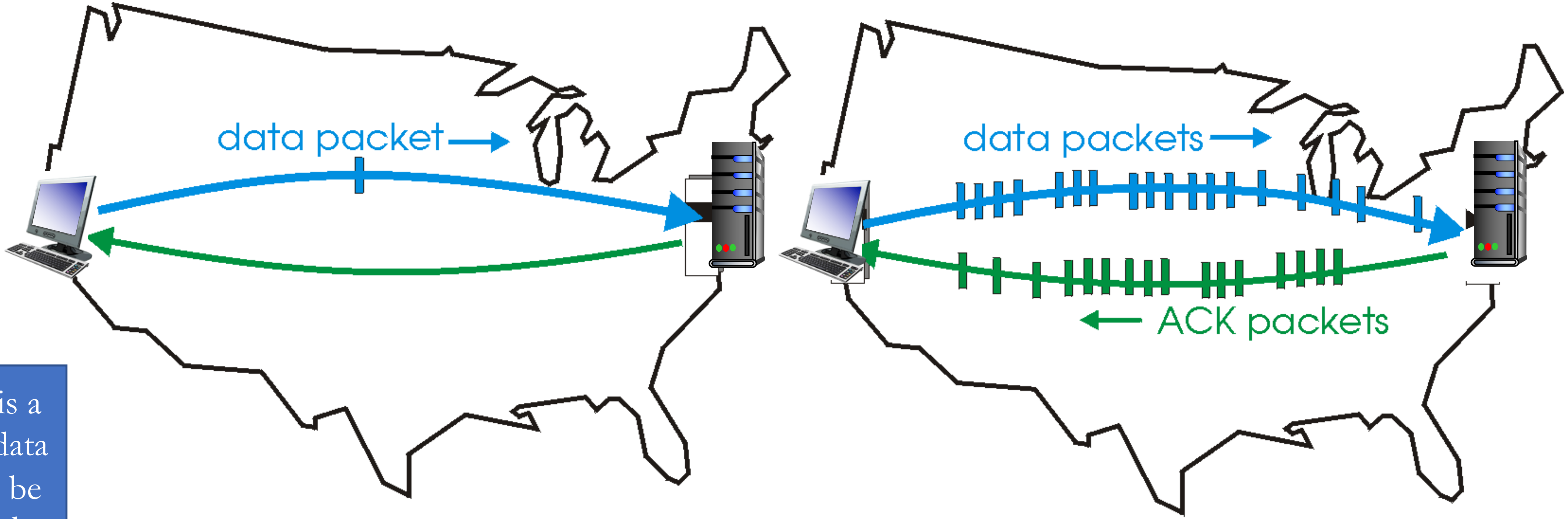
- Performance with ACKs is **4000×** slower than without ACKs.

Stop and wait illustration



Pipelining hides latency to increase throughput

- Pipelining: allow multiple "in-flight" packets, not yet ACK-ed.



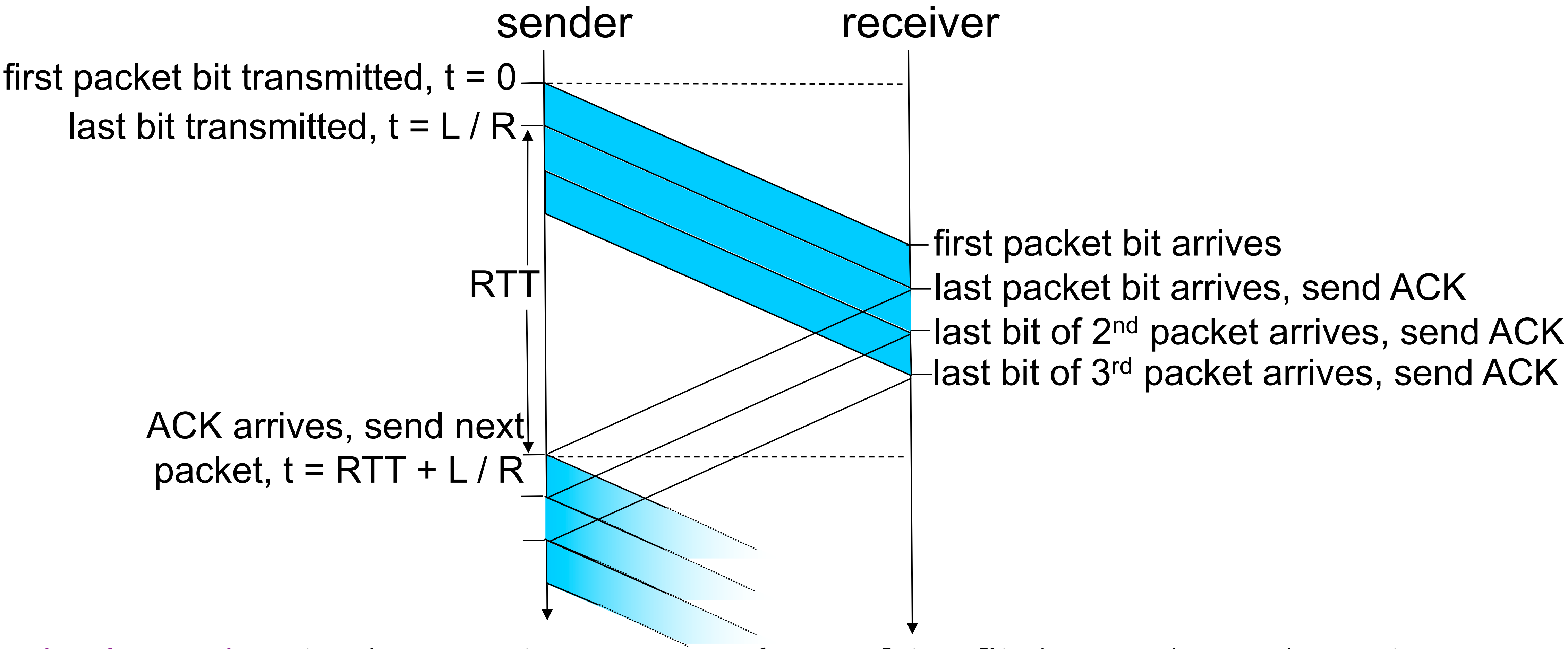
(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

A **buffer** is a queue of data waiting to be consumed.

- Packet **buffering** & acknowledgement become more complex.
- Later we'll talk about flow/congestion control to prevent overwhelming the receiver/network.

Pipelining increases link utilization



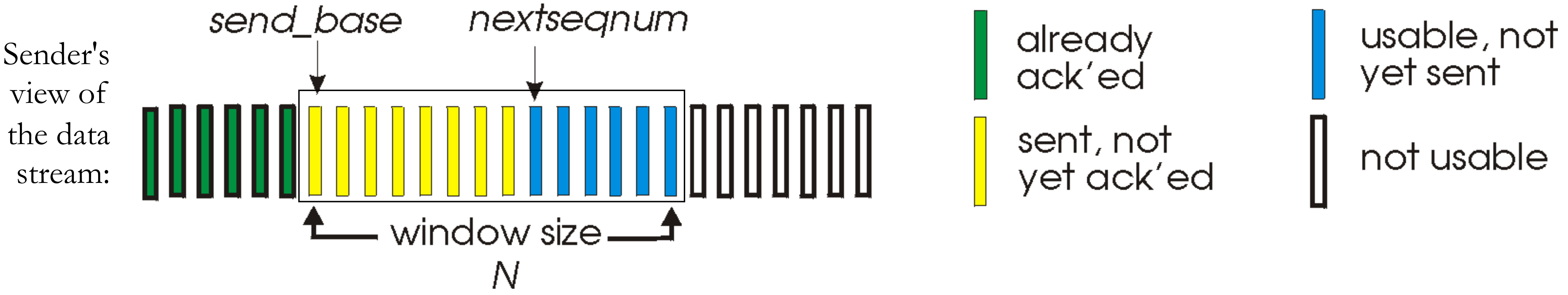
- **Window size** is the maximum number of in-flight packets (here it's 3).
- It's **finite** to limit the data buffering required at sender & receiver, and to limit the load placed on the network.

Sequence numbers

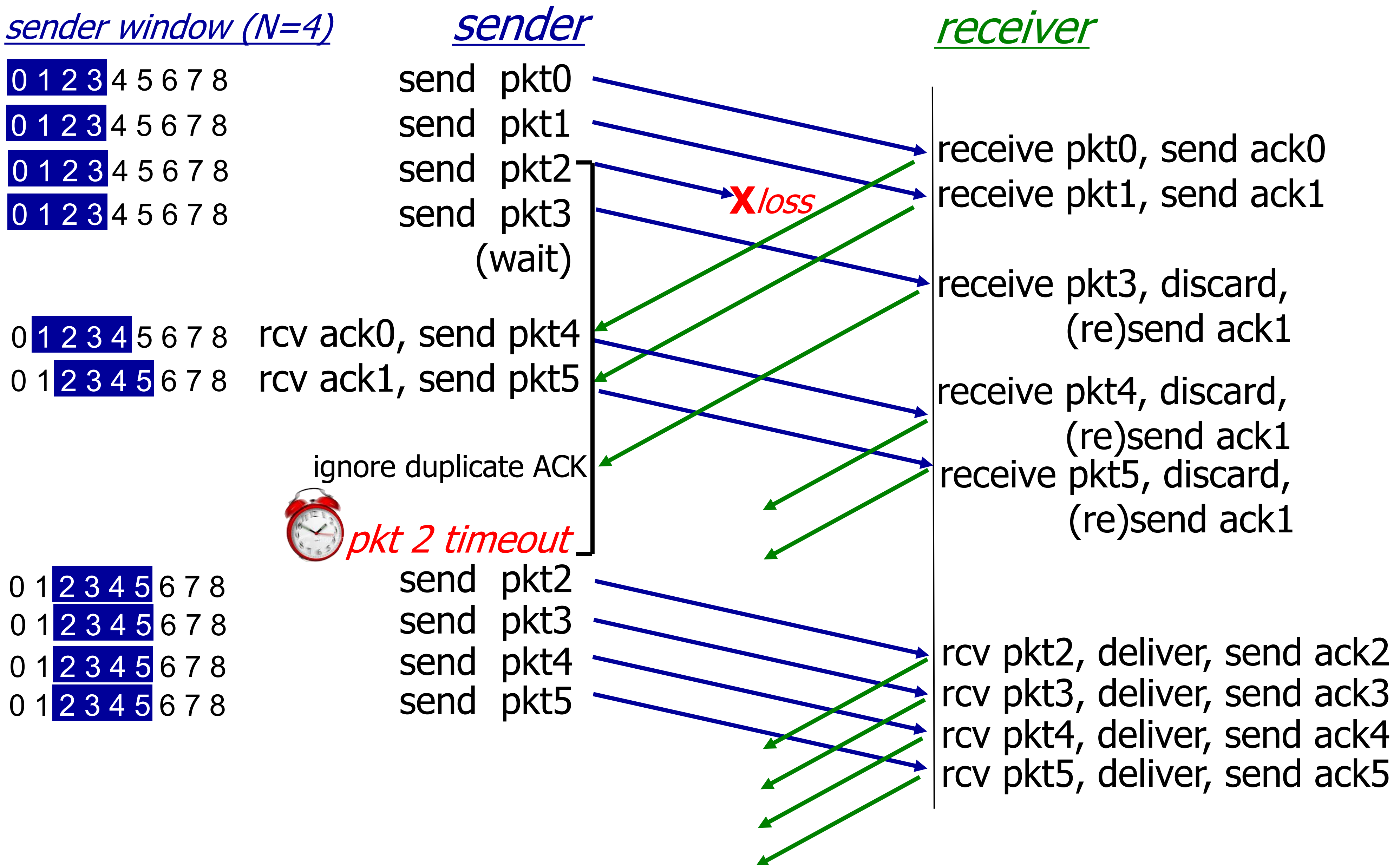
- (*Terminology*: segment = packet = frame = datagram)
- Pipelining ***parallelizes*** the transfer of ACK'ed data.
- Parallelism means we must handle out-of-order delivery.
- **Sequence numbers** identify each data segment with an increasing integer. (First segment has *seq.* # 0, next has *seq.* # 1, then 2, etc.)
- Allows receiver to correctly order and reassemble the received data.
- ACKs also must carry sequence numbers.
  - Sender has multiple data segments in flight, so the ACK must specify which of the several sequence numbers was received.

Pipelining attempt #1: ***Go Back N***

- Window size is N, sender can have up to N packets in flight.
- Receiver sends **cumulative ACK**: "*I got everything up to seq. number* **x**"
  - Discard out-of-order packets, re-send ACK of *last in-order seq. number*
  - If sender does not get an ACK after some **timeout** interval, resend **all** packets starting from packet after the last ACK'ed packet.
- If the sender timeout expires several times without receiving any ACK, then give up on the connection.



Go Back N in action



Go Back N Demo

https://stevetarzia.com/340/gbn.html

Go Back N advantages

- Easy to implement:
  - Sender just stores # of last ACK and maintains a timer.
  - Receiver just stores expected seq number and immediately passes new in-order packets to listening app.

Go Back N shortcomings

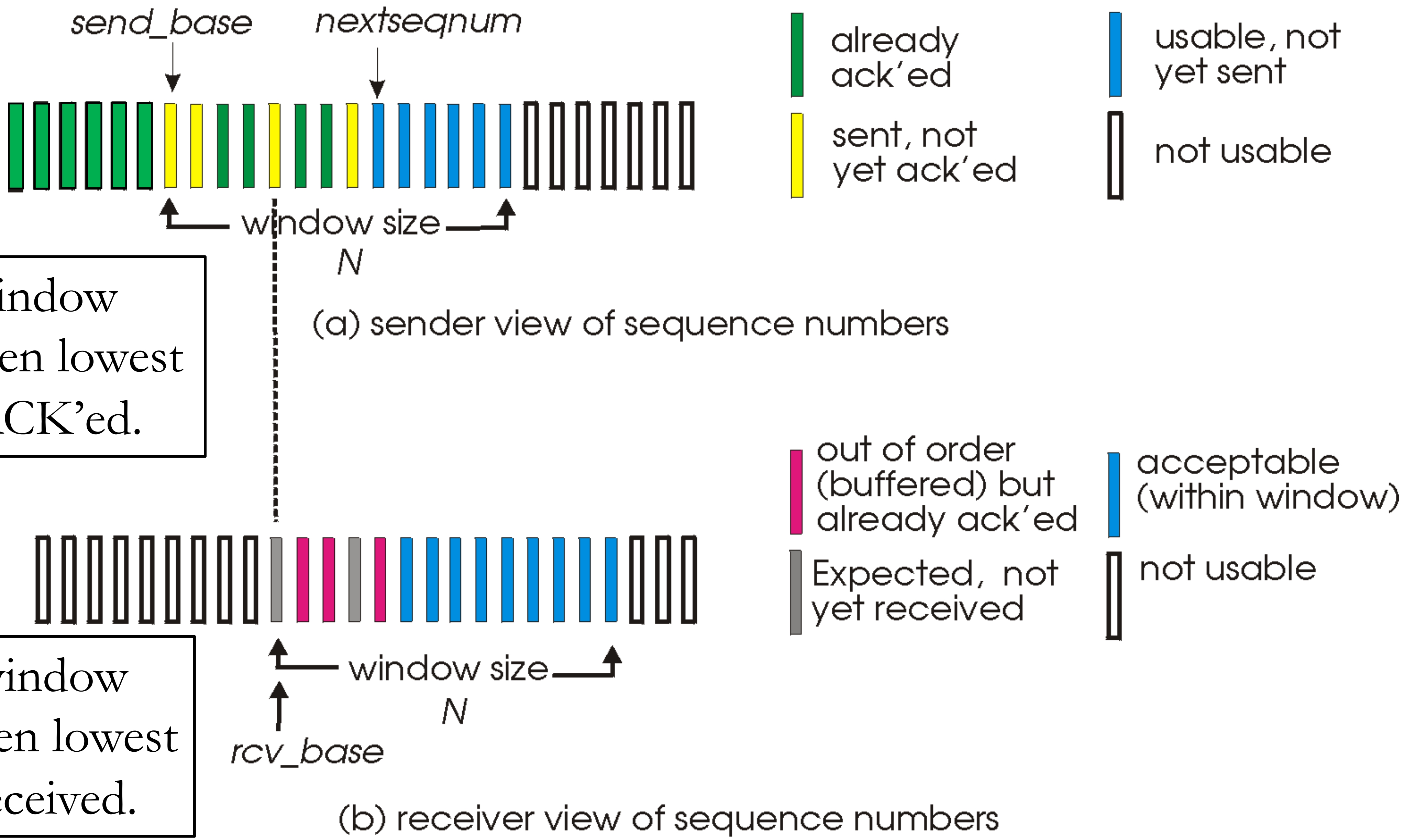
- A **single** lost or **delayed** packet invalidates all the in-flight data.
- Receiver can throw out a lot of good data, just because it's "early."
  - I.e. lacks **receiver buffering**.
- Lose an entire window of data due to one "bad" packet.

Pipelining attempt #2: ***Selective Repeat***

- Receiver **individually ACKs** all received packets.
- Out-of-order packets are stored by receiver and later reassembled
- Sender keeps **many timers**, *one for each in-flight packet*, and will re-send any packets not ACK'ed before timeout.
- Window of size N limits the maximum *range* of un-ACK'ed packets.
  - Receiver drops received packets with seq number outside the window.
  - This prevents packets from old connection from getting inserted into new connection's data stream.

Only re-send an *individual* packet whose transmission or ACK was lost.

Selective Repeat windows



Sender window advances when lowest packet is ACK'ed.

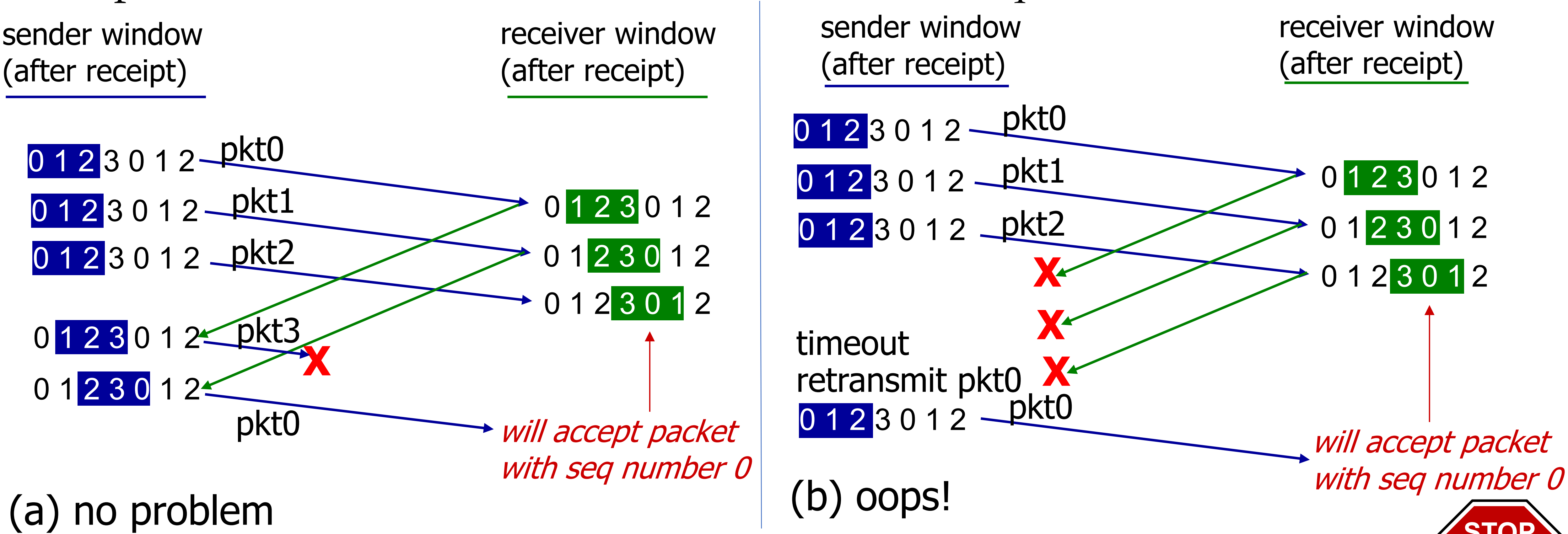
Receiver window advances when lowest packet is received.

Selective Repeat Demo

https://stevetarzia.com/340/sr.html

Seq number reuse can cause confusion

- In TCP, we use a 32-bit number for seq number (0 to 4Gbyte) and it eventually wraps around back to zero.
- Simplified illustration below assumes that 2-bit seq number is used:



- Solution: window length must be < half the max seq number

Recap

- **UDP** is a connectionless, packet-oriented transport protocol.
  - Adds a *port number* and *checksum* to packets.
- **TCP** is a streaming transport protocol.

- Delivery confirmation & ordering is possible by sending *ACKs*
  - After a *timeout*, resend packet that was not ACK'ed.
- *Pipelining* packets allow much better use of link capacity.
  - *Window size* determines the number of allowed in-flight packets
- *Go Back N* is a simple pipelining protocol that uses *cumulative ACKs*.
- *Selective Repeat* adds buffering to the receiver to avoid unnecessary retransmission.

- Next time: TCP details, connection setup, flow/congestion control.