EN.601.414/614 Computer Networks

Transport Layer

Xin Jin

Spring 2019 (MW 3:00-4:15pm in Shaffer 301)

JOHNS HOPKINS UNIVERSITY

https://github.com/xinjin/course-net

Recap: Web components

- **Infrastructure:**
  - Clients
  - Servers (DNS, CDN, Datacenters)
- **Content:**
  - URL: naming content
  - HTML: formatting content
- **Protocol for exchanging information: HTTP**

Recap: HTTP

- **Client-server architecture**
  - Server is “always on” and “well known”
  - Clients initiate contact to server
- **Synchronous request/reply protocol**
  - Run over TCP, Port 80
- **Stateless**
- **ASCII format**
  - Before HTTP/2

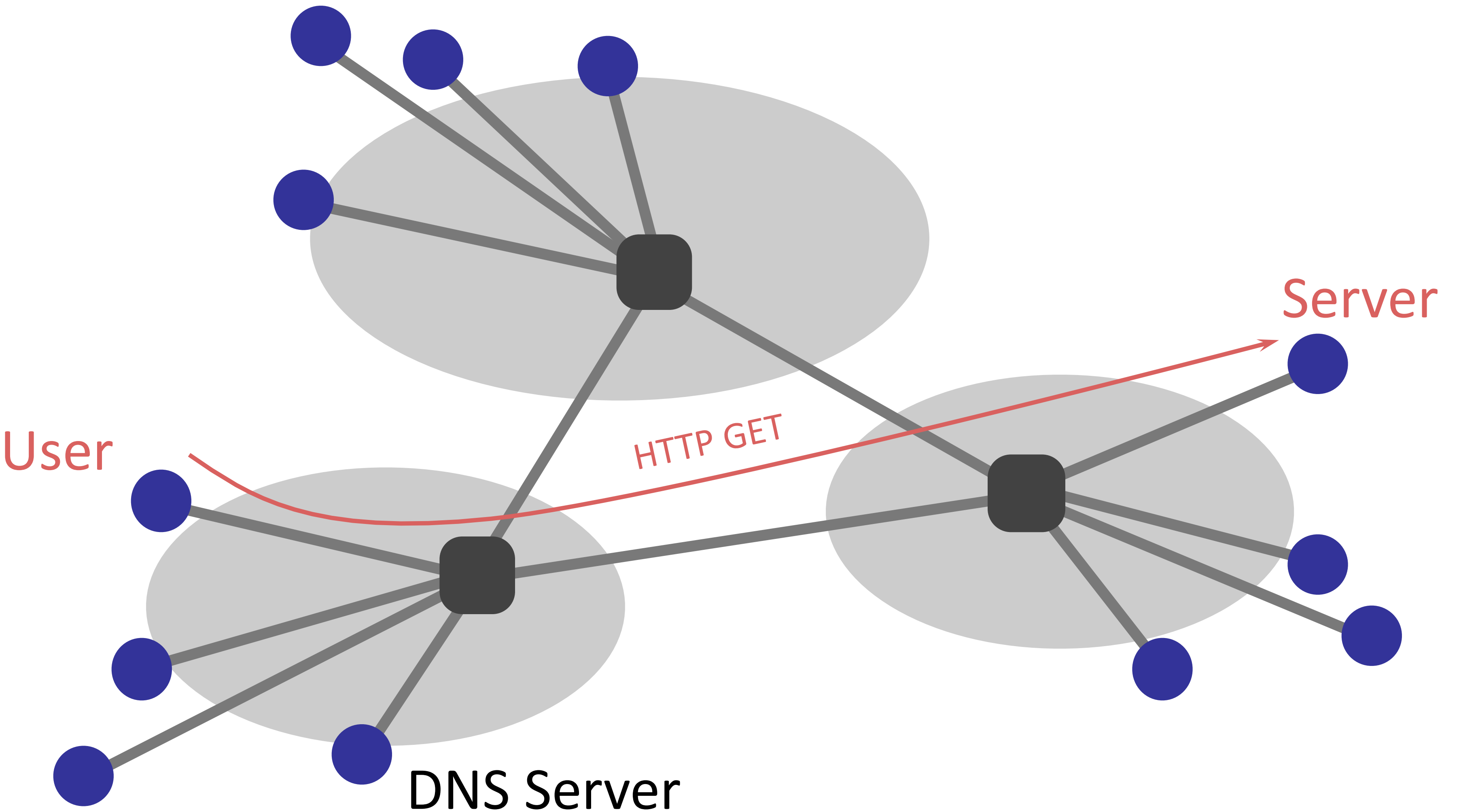
Recap: CDN

- **Caching and replication as a service**
- **Large-scale distributed storage infrastructure (usually) administered by one entity**
  - e.g., Akamai has servers in 20,000+ locations
- **Combination of caching and replication**
  - Pull: Direct result of clients' requests (caching)
  - Push: Expectation of high access rate (replication)
- **Can do some processing to handle dynamic webpage content**

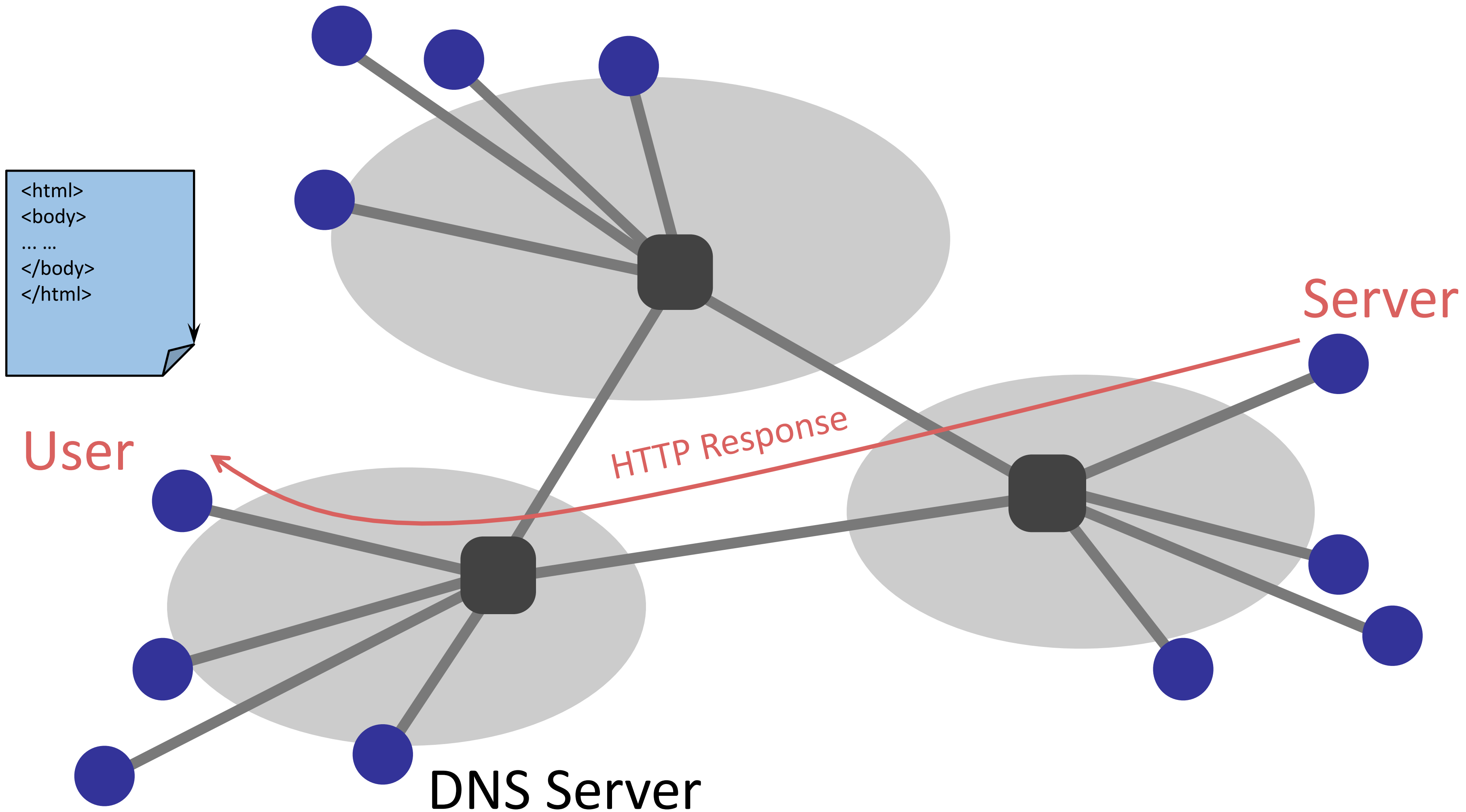
Recap: DNS

- **Resolve names to addresses**
    - ➢Uniqueness: no naming conflicts
    - ➢Scalable: many names and frequent updates
    - ➢Distributed, autonomous administration
        - Ability to update my own (machines') names
        - Don't have to track everybody's updates
    - ➢Highly available
    - ➢Lookups are fast
- **Level of indirection**

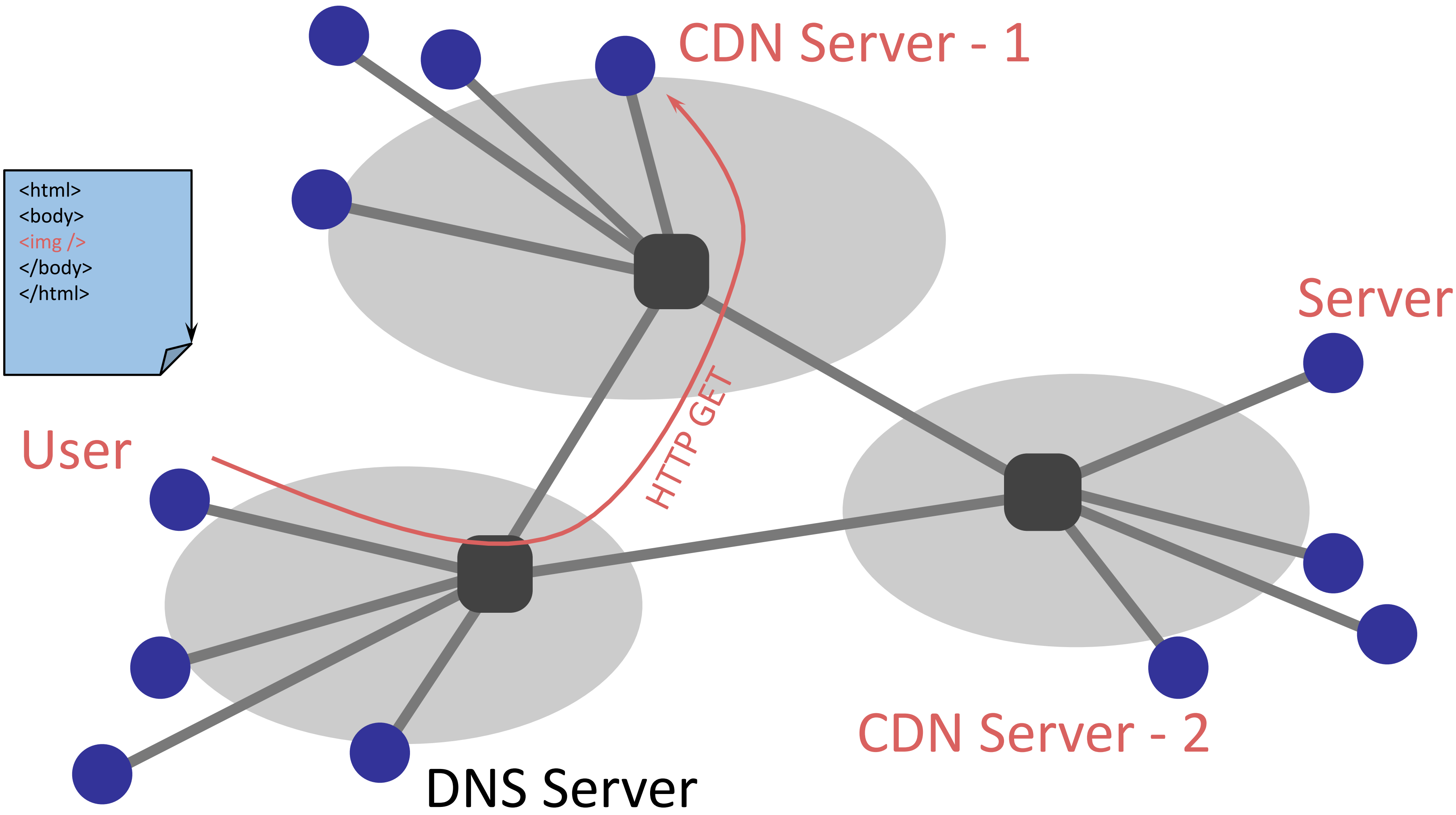
Putting it all together



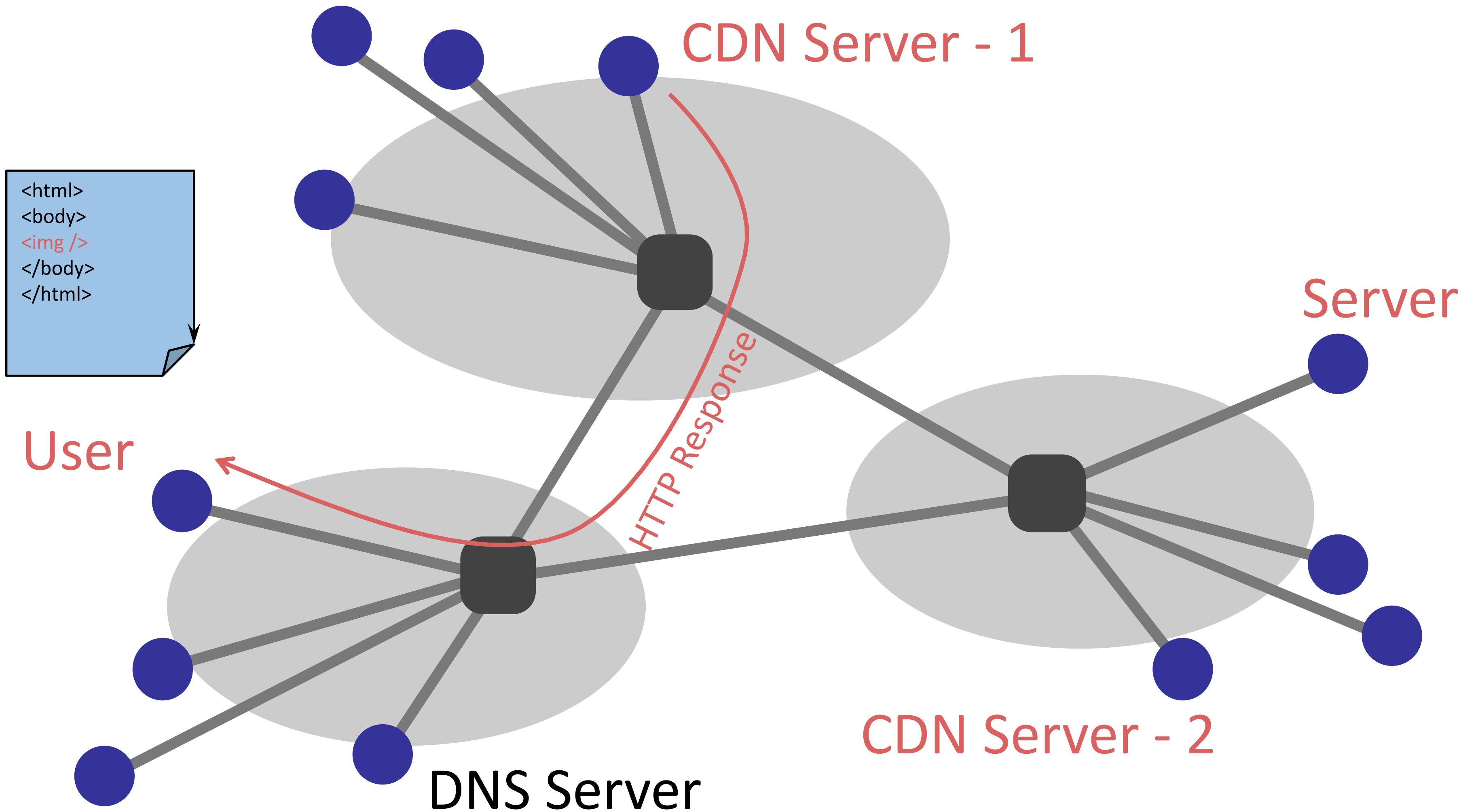
Putting it all together



Putting it all together



Putting it all together

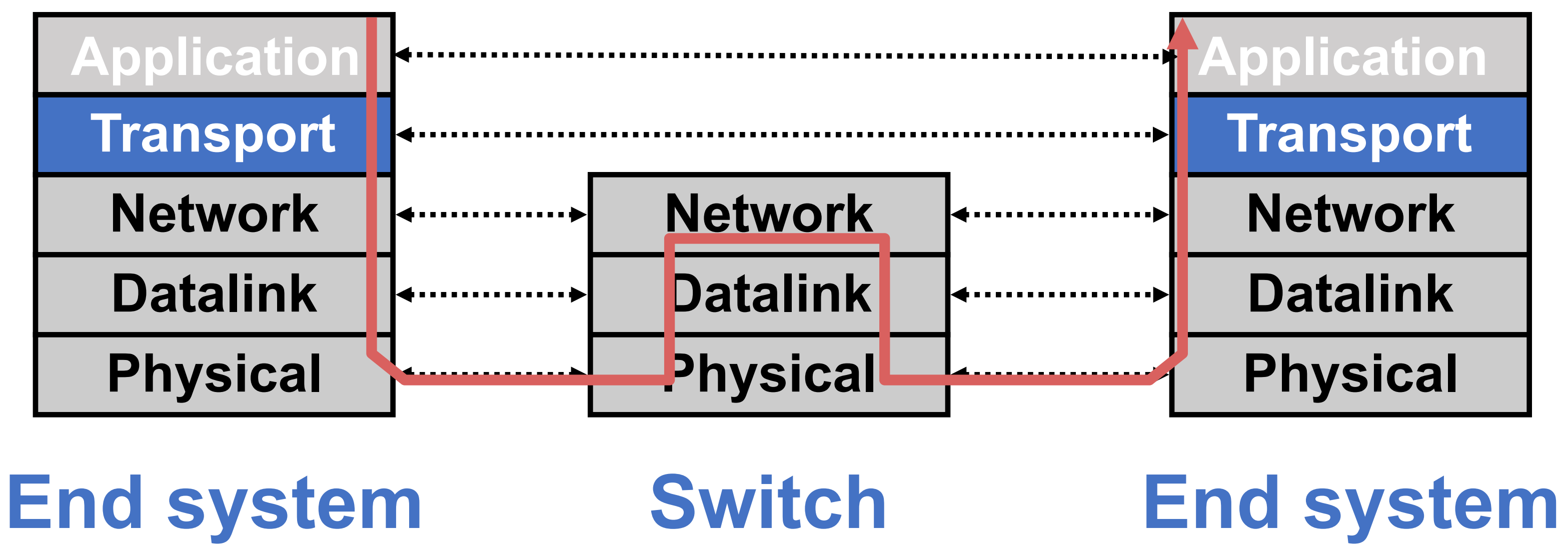


Agenda

- **Transport layer basics**
- **UDP**
- **Designing a reliable transport protocol**

Transport layer

- **Layer at end hosts, between the application and network layer**



Why a transport layer?

- **IP packets are addressed to a host but end-to-end communication is between application processes at hosts**
  - ➢ Need a way to decide which packets go to which applications (multiplexing/demultiplexing)
- **IP provides a weak service model (best-effort)**
  - ➢ Packets can be corrupted, delayed, dropped, reordered, duplicated
  - ➢ No guidance on how much traffic to send and when
  - ➢ Dealing with this is tedious for application developers

Multiplexing & demultiplexing

- **Multiplexing (Mux)**
  - ➢Gather and combining data chunks at the source host from different applications and delivering to the network layer
- **Demultiplexing (Demux)**
  - ➢Delivering correct data to corresponding sockets from a multiplexed stream

Role of the transport layer

- **Communication between processes**
  - Mux and demux from/to application processes
  - Implemented using *ports*

Role of the transport layer

- **Communication between processes**
- **Provide common end-to-end services for app layer [optional]**
  - Reliable, in-order data delivery
  - Well-paced data delivery
    - Too fast may overwhelm the network
    - Too slow is not efficient

Role of the transport layer

- **Communication between processes**
- **Provide common end-to-end services for app layer [optional]**
- **TCP and UDP are the common transport protocols**
  - Also SCTP, MPTCP, SST, RDP, DCCP, …

Role of the transport layer

- **Communication between processes**
- **Provide common end-to-end services for app layer [optional]**
- **TCP and UDP are the common transport protocols**
- **UDP is a minimalist transport protocol**
    - Only provides mux/demux capabilities

Role of the transport layer

- **Communication between processes**
- **Provide common end-to-end services for app layer [optional]**
- **TCP and UDP are the common transport protocols**
- **UDP is a minimalist transport protocol**
- TCP offers a reliable, in-order, byte stream abstraction
  - ➢ With congestion control, but w/o performance guarantees (delay, b/w, etc.)

Applications and sockets

- **Socket: software abstraction for an application process to exchange network messages with the (transport layer in the) operating system**
- **Two important types of sockets**
  - UDP socket: TYPE is SOCK_DGRAM
  - TCP socket: TYPE is SOCK_STREAM

Ports

- **16-bit numbers that help distinguishing apps**
  - ➢ Packets carry src/dst port no in transport header
  - ➢ Well-known (0-1023) and ephemeral ports
- **OS stores mapping between sockets and ports**
  - ➢ Port in packets and sockets in OS
  - ➢ For UDP ports (SOCK_DGRAM)
    - OS stores (local port, local IP address) ←→ socket
  - ➢ For TCP ports (SOCK_STREAM)
    - OS stores (local port, local IP, remote port, remote IP) ←→ socket

UDP: User Datagram Protocol

- **Lightweight communication between processes**
  - ➢Avoid overhead and delays of order & reliability
- **UDP described in RFC 768 – (1980!)**
  - ➢Destination IP address and port to support demultiplexing

SRC port	DST port
Length	Checksum
DATA	

UDP (cont'd)

- **Optional error checking on the packet contents**
  - (checksum field = 0 means "don't verify checksum")
  - See text on how checksums are calculated
- **Source port is also optional**
  - Useful to respond back to the sender in some cases

Why a transport layer?

- **IP packets are addressed to a host but end-to-end communication is between application processes at hosts**
  - Need a way to decide which packets go to which applications (mux/demux)
- **IP provides a weak service model (best-effort)**
  - Packets can be corrupted, delayed, dropped, reordered, duplicated
  - No guidance on how much traffic to send and when
  - Dealing with this is tedious for application developers

Reliable transport

- **In a perfect world, reliable transport is easy**

@Sender

- Send packets

@Receiver

- Wait for packets

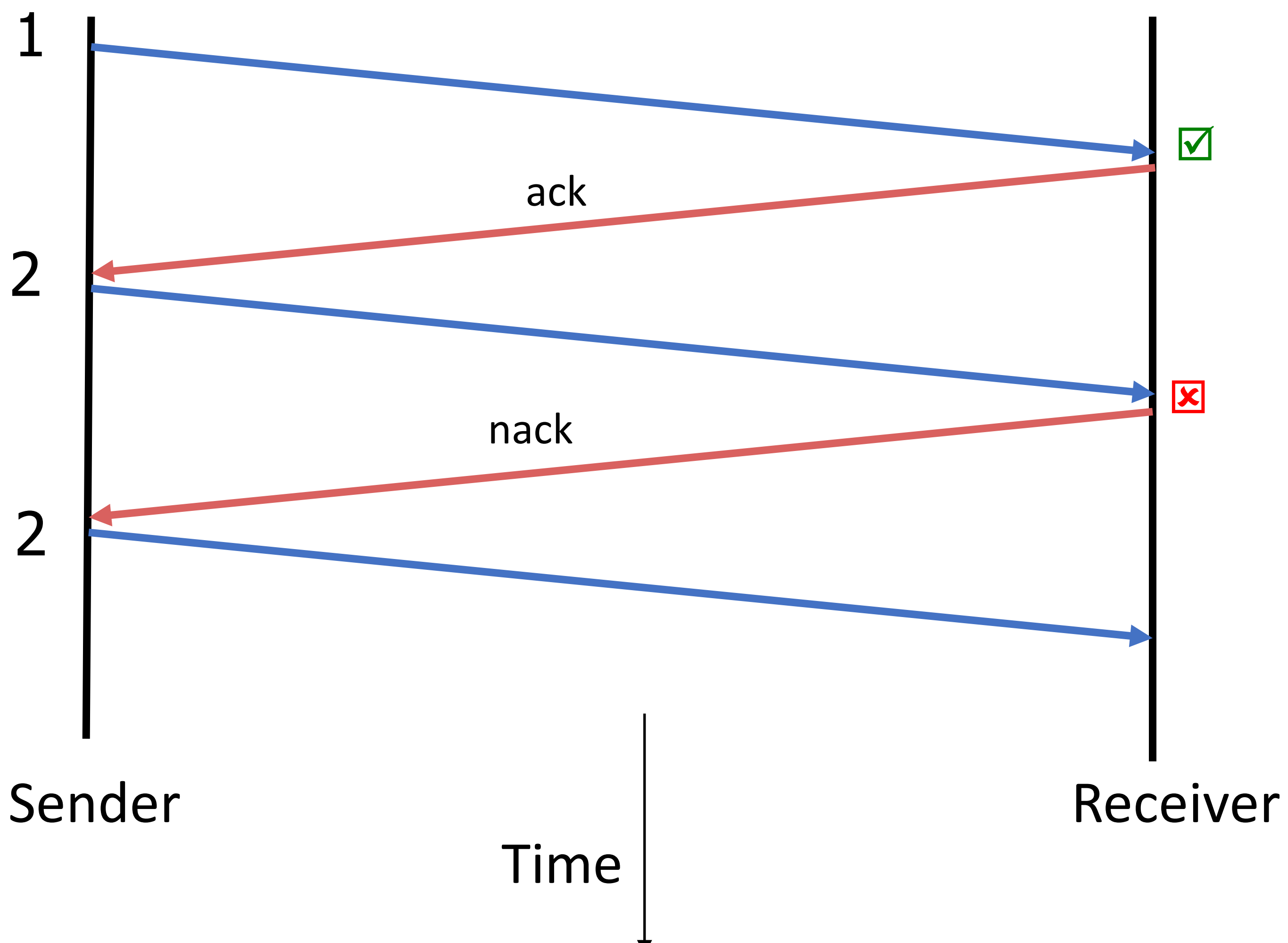
Reliable transport

- **In a perfect world, reliable transport is easy**
- **All the bad things best-effort can do**
  - A packet is corrupted (why?)
  - A packet is lost (*why?*)
  - A packet is delayed (*why?*)
  - Packets are reordered (*why?*)
  - A packet is duplicated (*why?*)

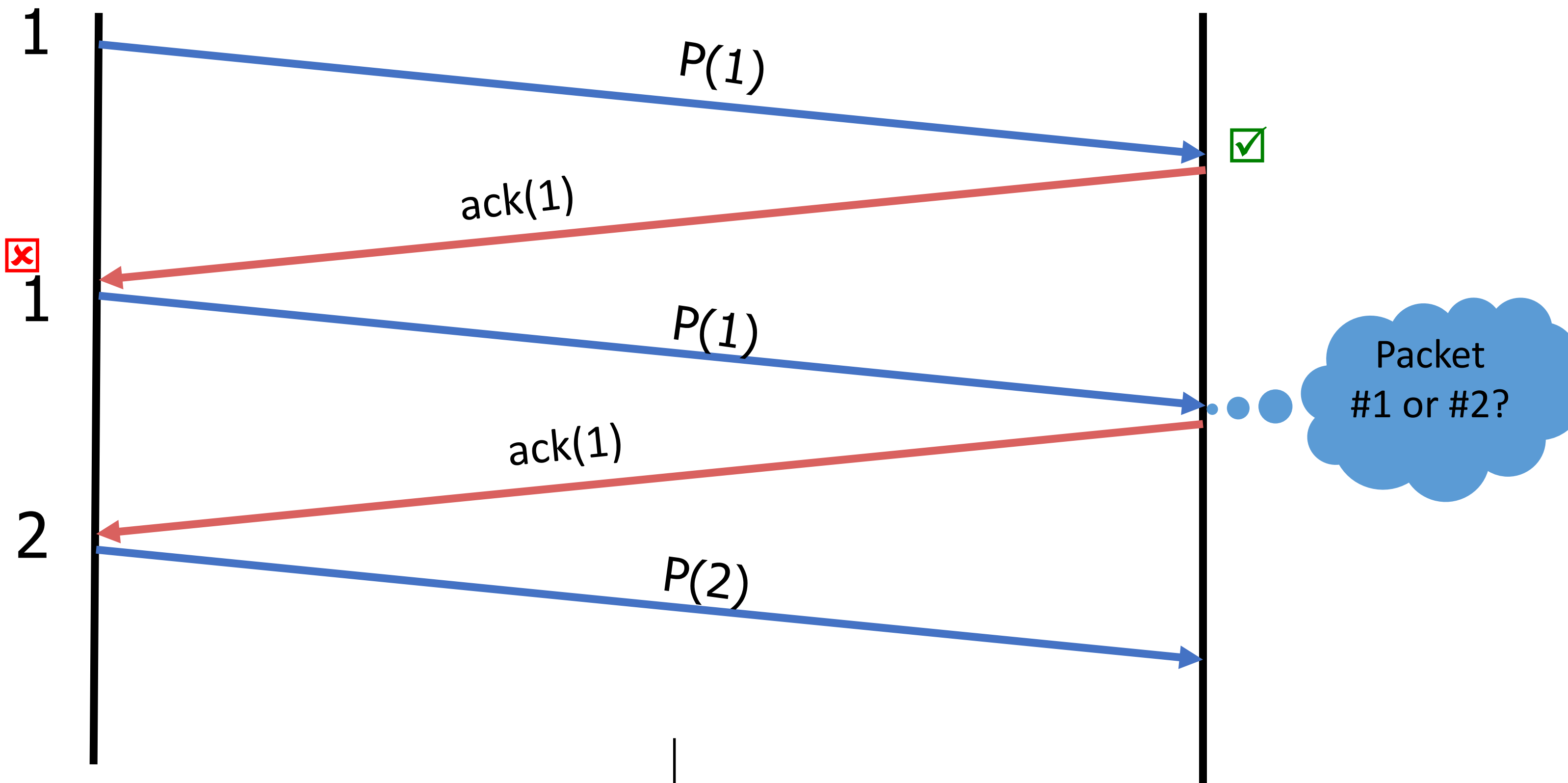
Reliable transport

- **Mechanisms for coping with bad events**
  - Checksums: to detect corruption
  - ACKs: receiver tells sender that it received packet
  - NACK: receiver tells sender it did not receive packet
  - Sequence numbers: a way to identify packets
  - Retransmissions: sender resends packets
  - Timeouts: a way of deciding when to resend packets
  - *Forward error correction: a way to mask errors without retransmission*
  - *Network coding: an efficient way to repair errors*

Dealing with packet corruption

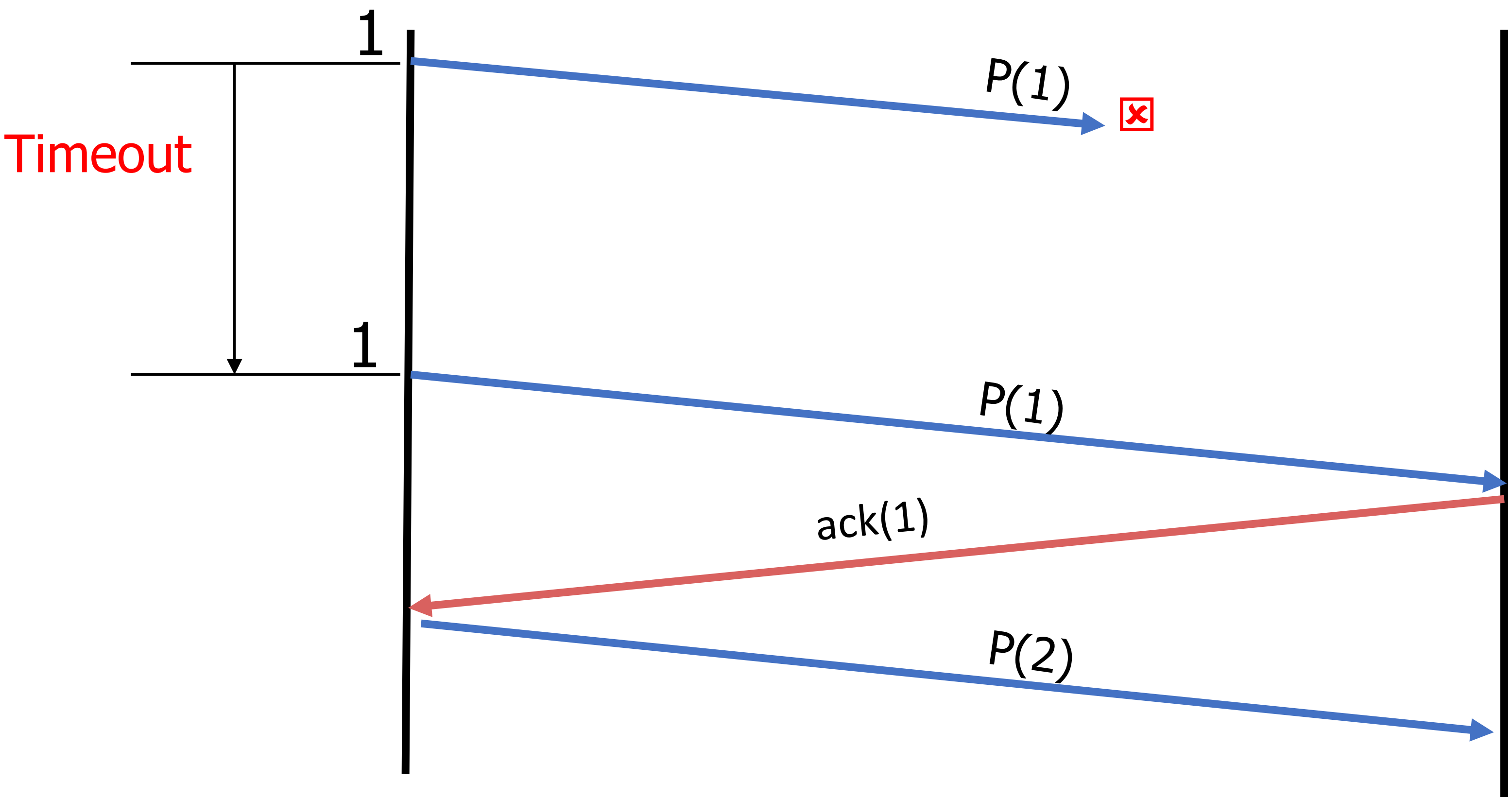


Dealing with packet corruption



Data and ACK packets carry sequence numbers

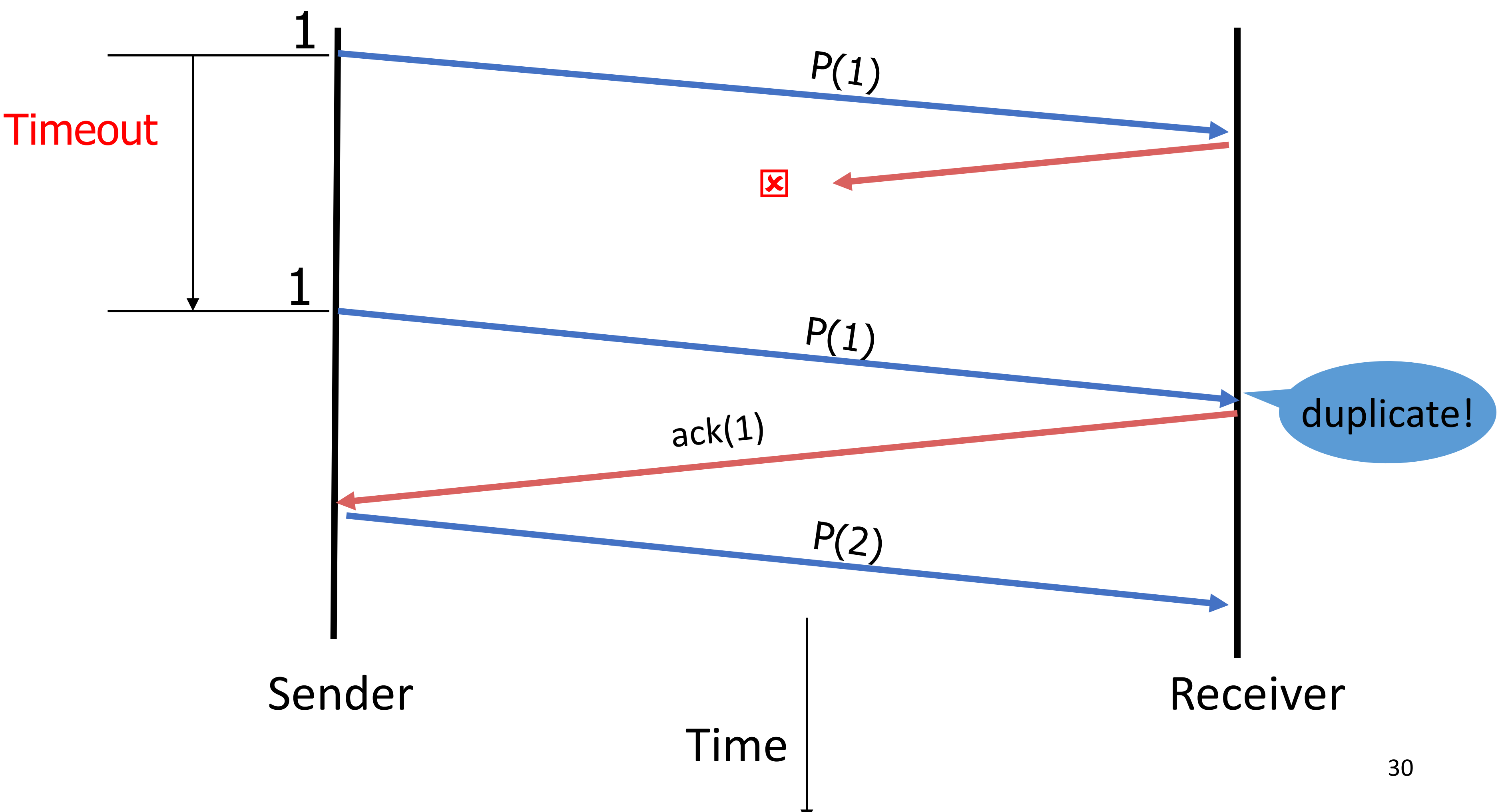
Dealing with packet loss



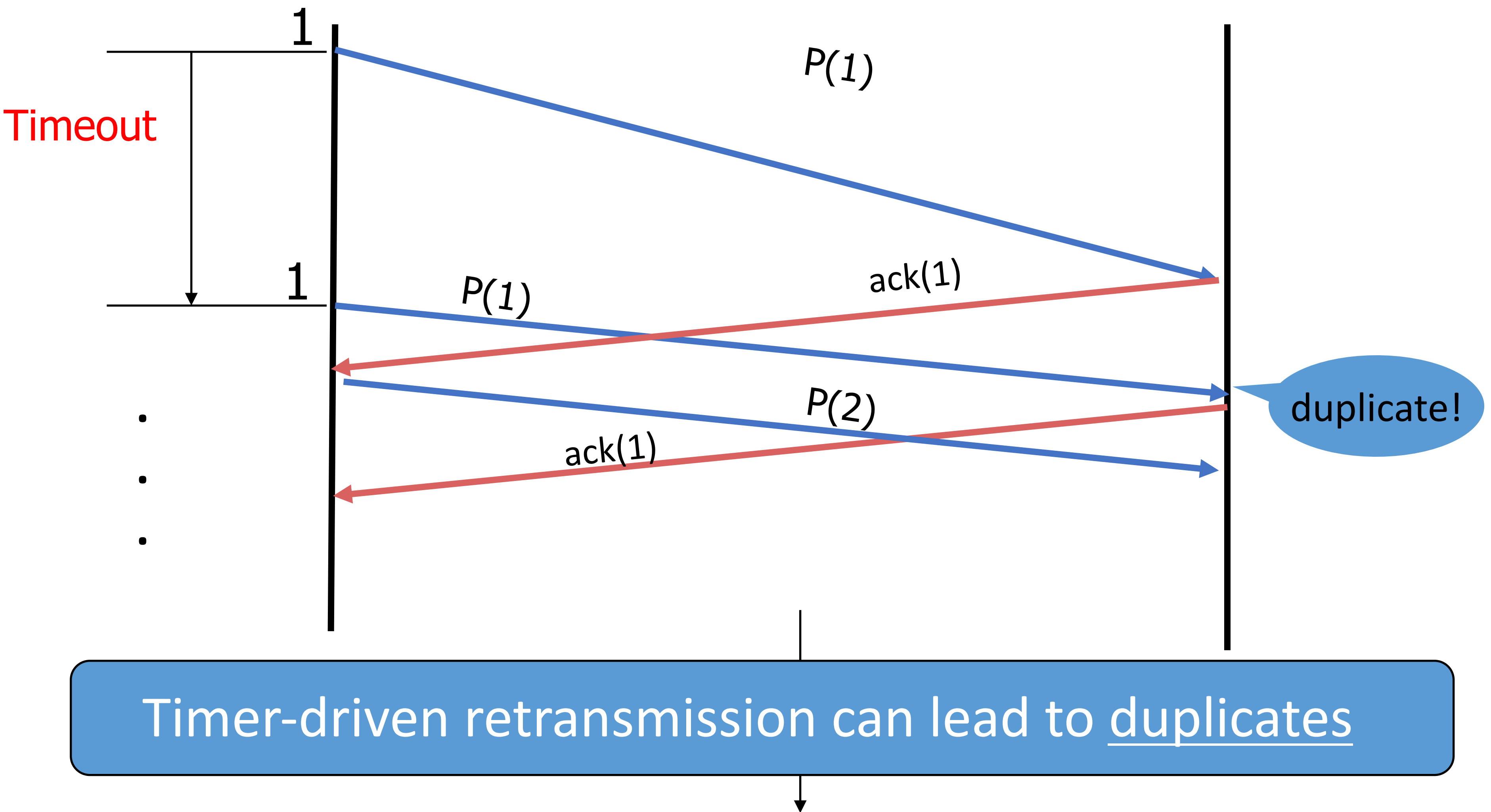
Timer-driven loss detection

Set timer when packet is sent; retransmit on timeout

Dealing with packet loss (of ack)



Dealing with packet duplicate



Timer-driven retransmission can lead to <u>duplicates</u>

Components of a solution

- **Checksums (to detect bit errors)**
- **Timers (to detect loss)**
- **Acknowledgements (positive or negative)**
- **Sequence numbers (to deal with duplicates)**

Designing a reliable transport

A Solution: "Stop and Wait"

@Sender

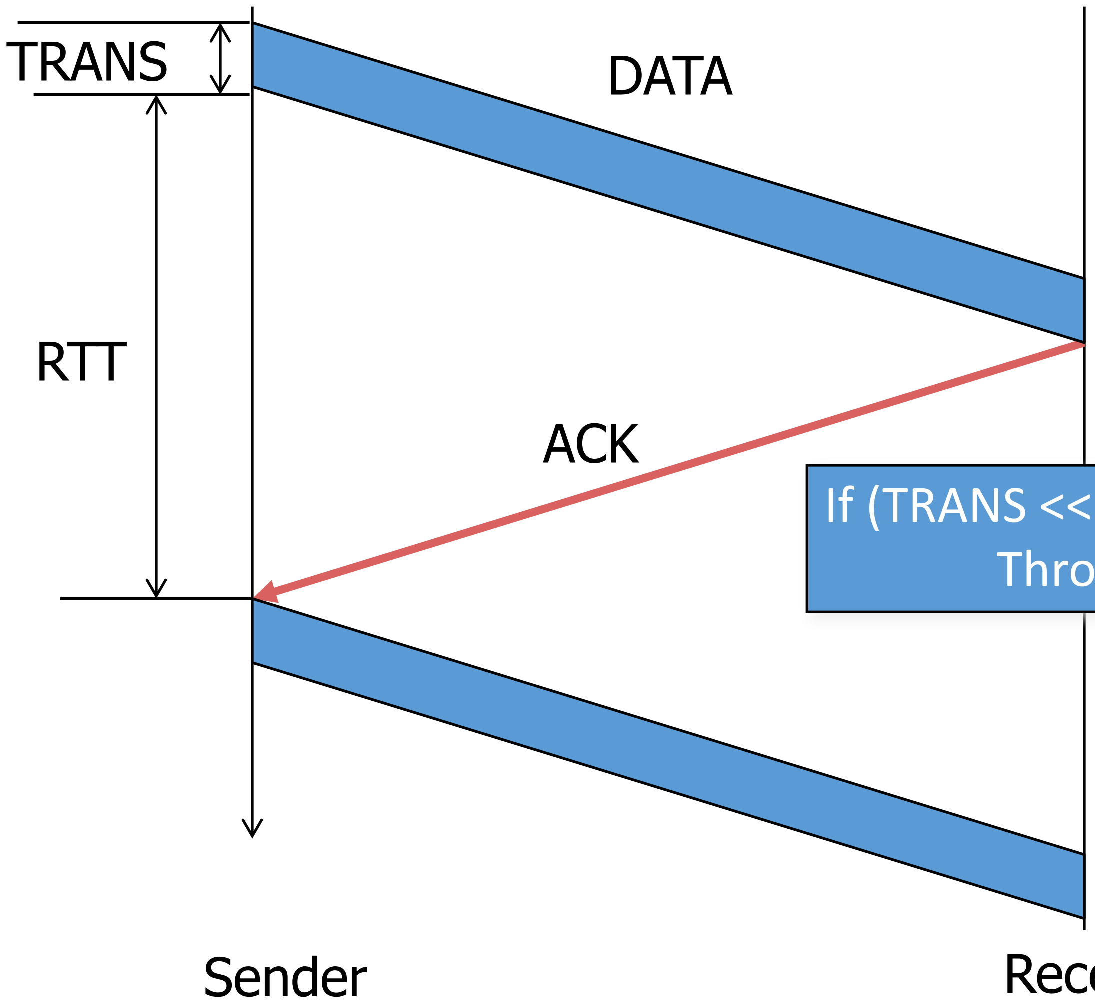
- Send packet(I); (re)set timer; wait for ack
- If (ACK)
  - I++; repeat
- If (NACK or TIMEOUT)
  - repeat

@Receiver

- Wait for packet
- If packet is OK, send ACK
- Else, send NACK
- Repeat

- **A correct reliable transport protocol, but an extremely inefficient one**

Stop & Wait is inefficient



If (TRANS << RTT) then
Throughput ~ DATA/RTT

Orders of magnitude

- **Transmission time for 10Gbps link:**
  - ~ microsecond for 1500 byte packet
- **RTT:**
  - 1,000 kilometers ~ O(10) milliseconds

Three design decisions

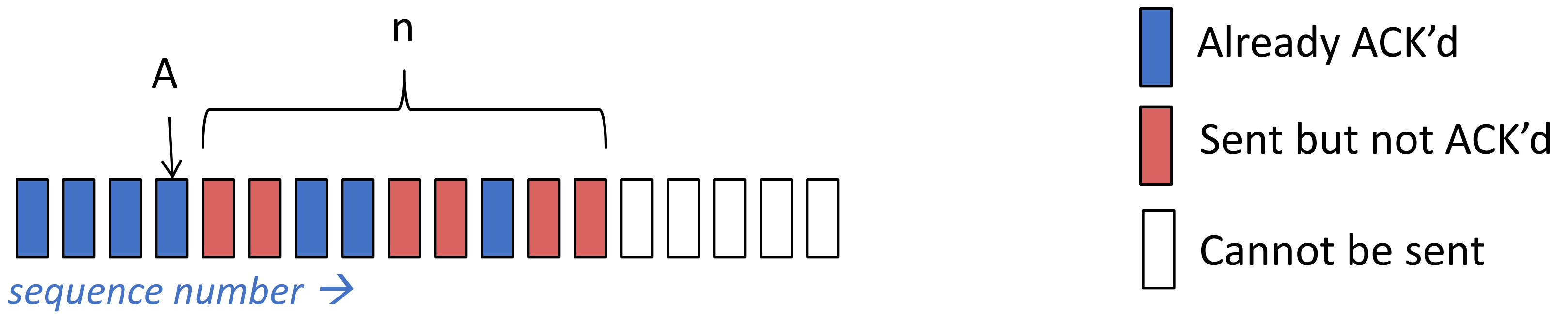
- **Which packets can sender send?**
  - Sliding window
- **How does receiver ack packets?**
  - Cumulative
  - Selective
- **Which packets does sender resend?**
  - Go-Back N (GBN)
  - Selective Repeat (SR)

Sliding window

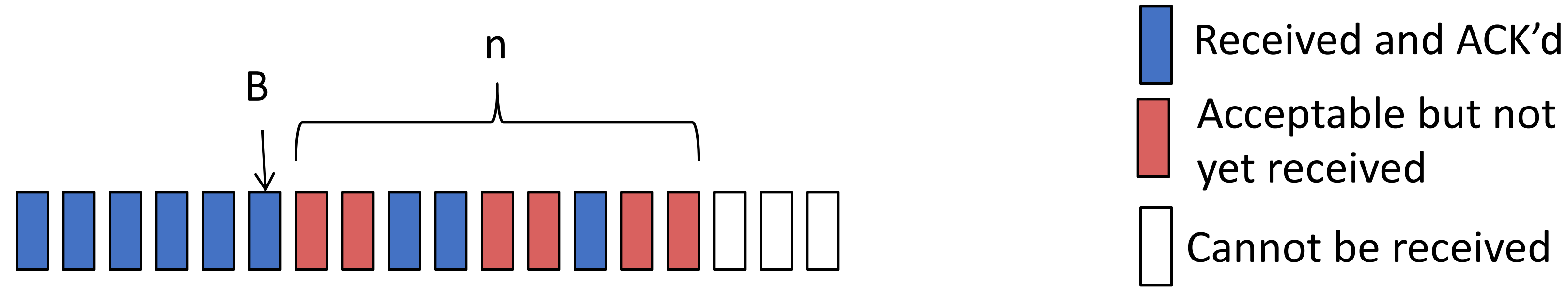
- **Window = set of adjacent sequence numbers**
  - The size of the set is the window size
  - Assume window size is n
- **General idea: send up to n packets at a time**
  - Sender can send packets in its window
  - Receiver can accept packets in its window
  - Window of acceptable packets “slides” on successful reception/acknowledgement
  - Window contains all packets that might still be in transit
- **Sliding window often called “packets in flight”**

Sliding window

- **Let A be the** last ack'd packet of sender without gap**; then window of sender = {A+1, A+2, …, A+n}**



- **Let B be the** last received packet without gap **by receiver, then window of receiver = {B+1,…, B+n}**



Throughput of sliding window

- **If window size is n, then throughput is roughly**
  - MIN(n*DATA/RTT, Link Bandwidth)
- **Compare to Stop and Wait: Data/RTT**

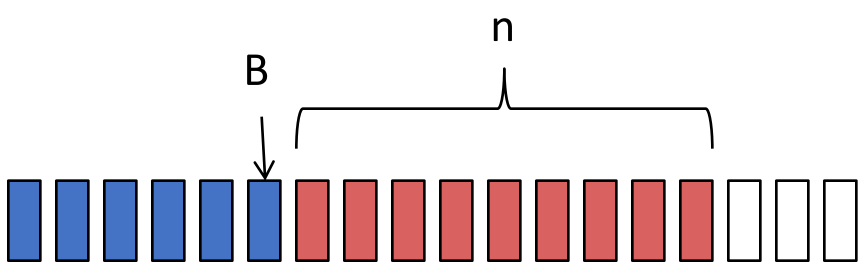
- **What happens when n gets too large?**

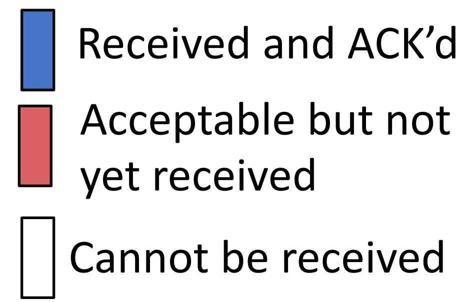
Acknowledgements w/ sliding window

- **Two common options**
  - Cumulative ACKs: ACK carries next in-order sequence number that the receiver expects

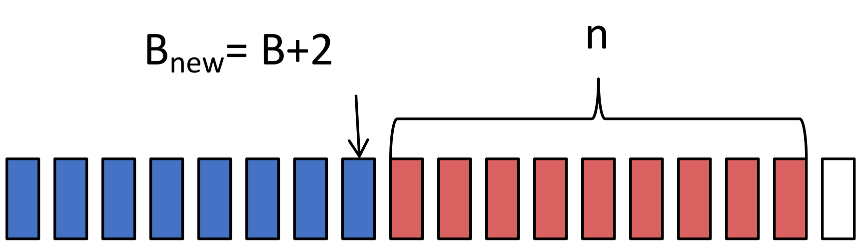
Cumulative acknowledgements

- **At receiver**





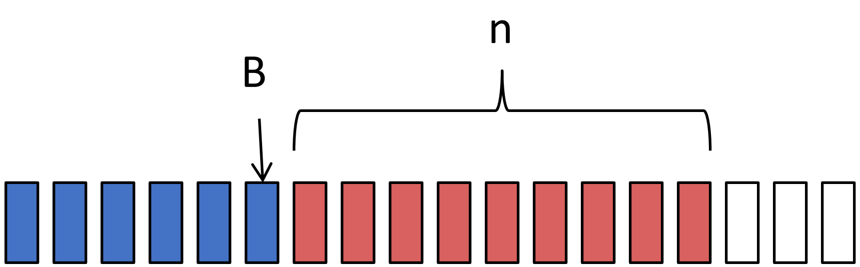
- After receiving B+1, B+2

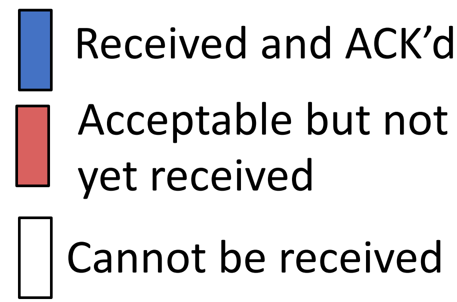


- Receiver sends ACK(B+3) = ACK(B_{new}+1)

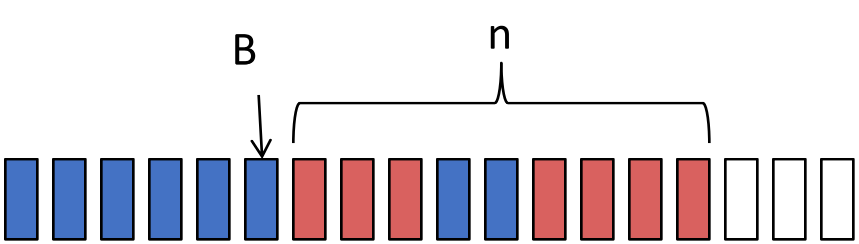
Cumulative acknowledgements (cont'd)

- **At receiver**





- After receiving B+4, B+5



- Receiver sends ACK(B+1)

Acknowledgements w/ sliding window

- **Two common options**
  - Cumulative ACKs: ACK carries next in-order sequence number the receiver expects
  - Selective ACKs: ACK individually acknowledges correctly received packets

- **Selective ACKs offer more precise information but require more complicated book-keeping**

Sliding window protocols

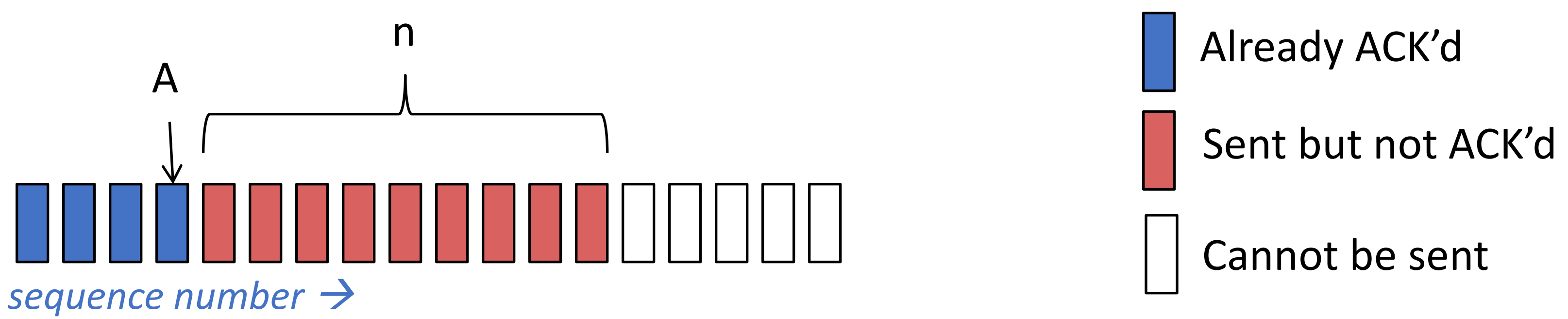
- **Resending packets: two canonical approaches**
  - Go-Back-N
  - Selective Repeat
- **Many variants that differ in implementation details**

Go-Back-N (GBN)

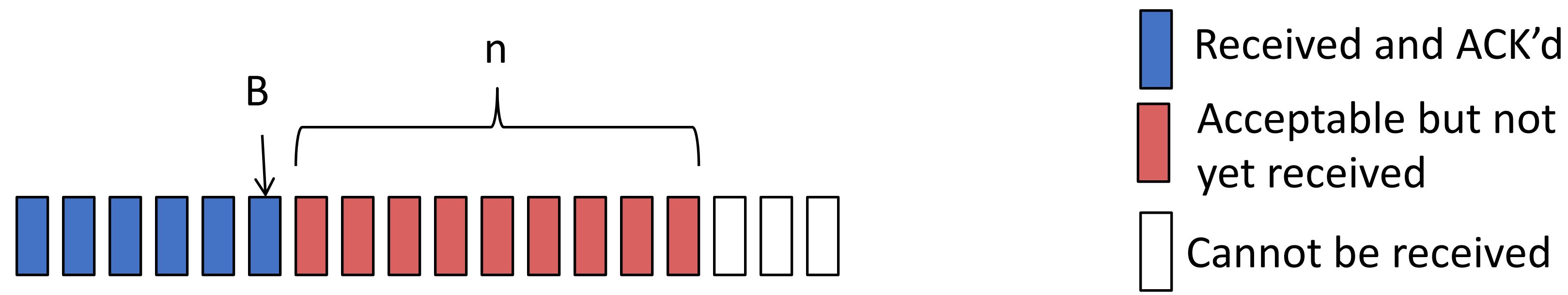
- **Sender transmits up to n unacknowledged packets**
- **Receiver only accepts packets in order**
  - Discards out-of-order packets (i.e., packets other than B+1)
- **Receiver uses cumulative acknowledgements**
  - i.e., sequence# in ACK = next expected in-order sequence#
- **Sender sets timer for 1st outstanding ack (A+1)**
- **If timeout, retransmit A+1, … , A+n**

Sliding window with GBN

- **Let A be the last ack'd packet of sender without gap; then window of sender = {A+1, A+2, …, A+n}**

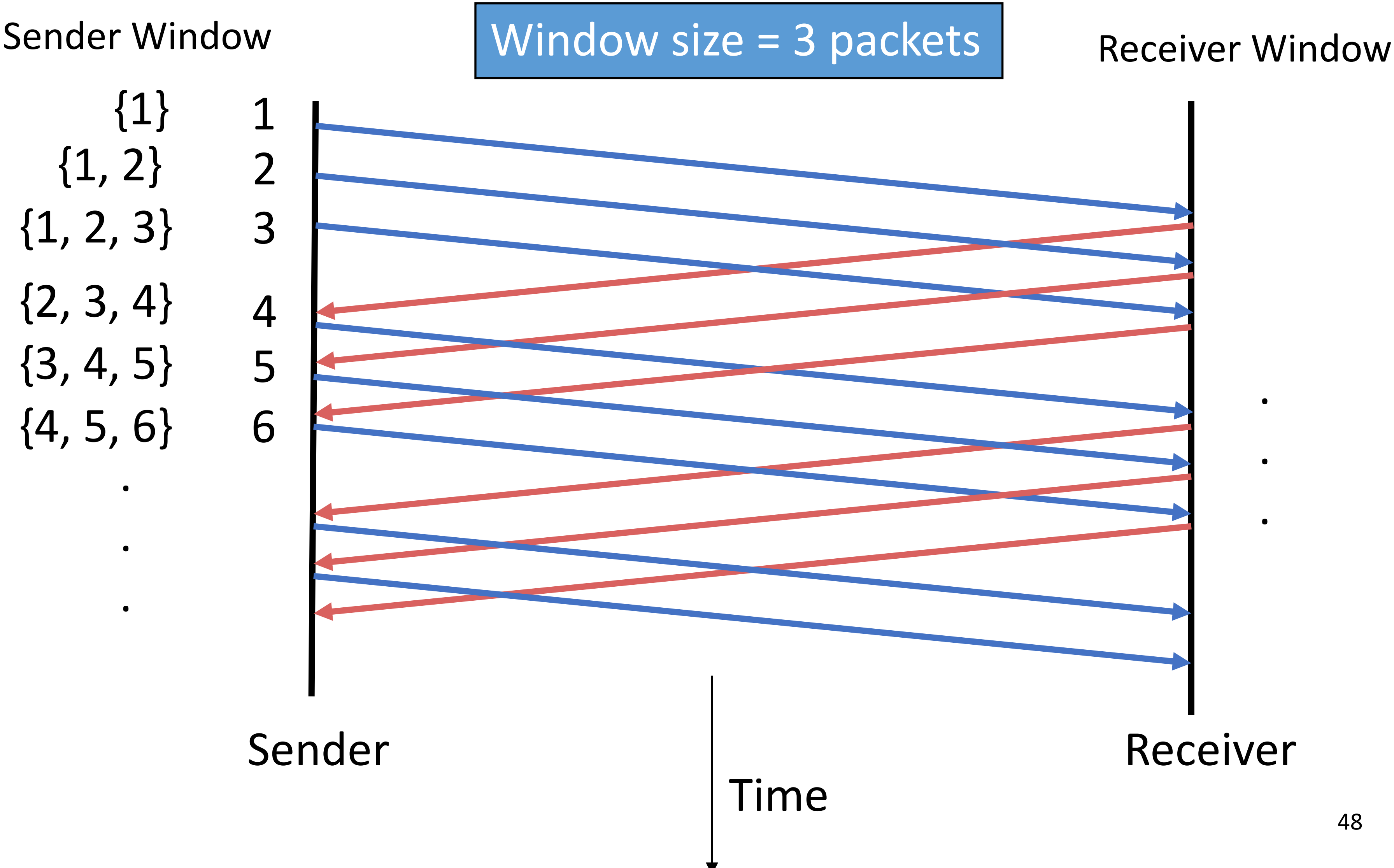


- **Let B be the last received packet without gap by receiver, then window of receiver = {B+1,…, B+n}**

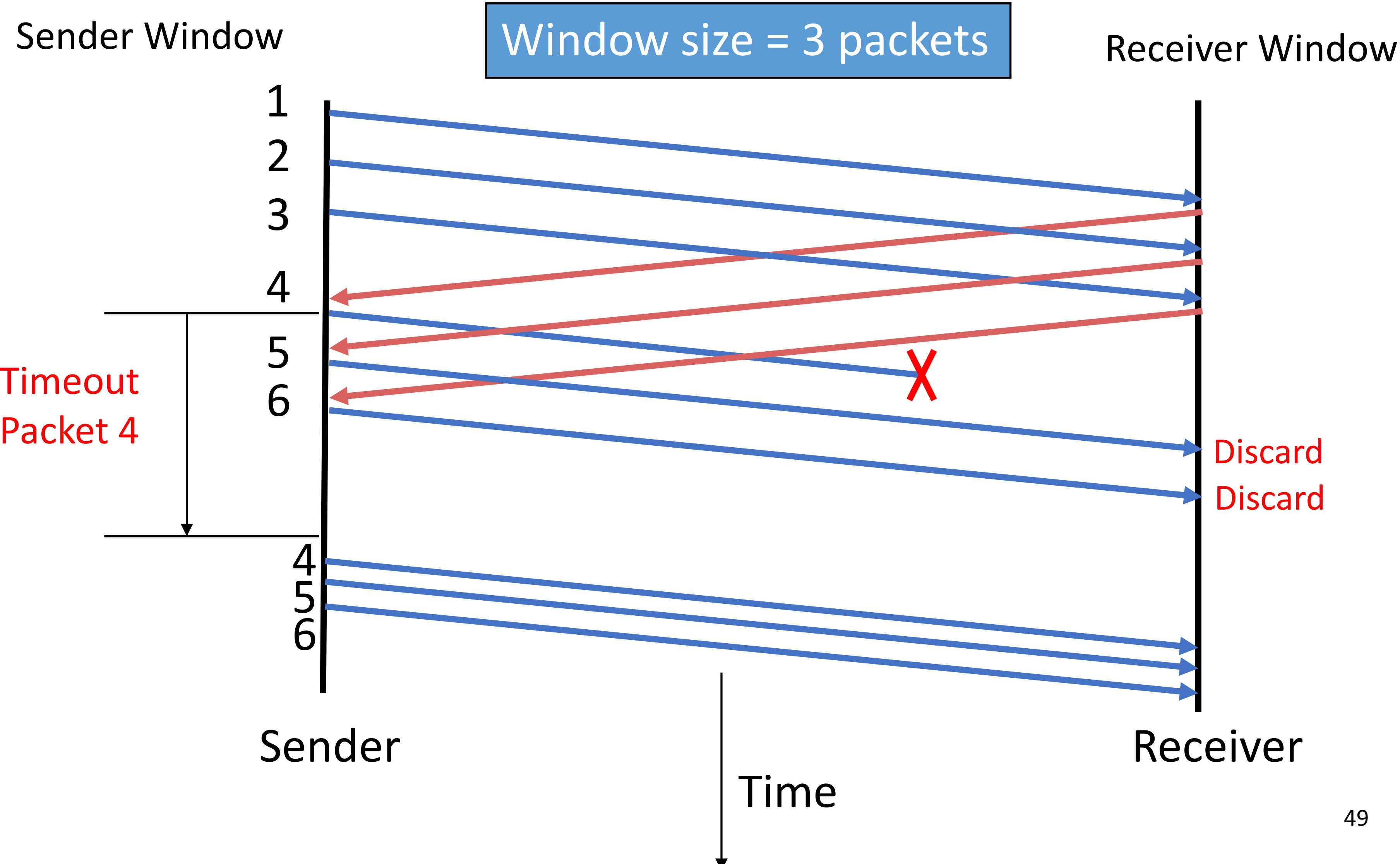


GBN example w/o errors

Window size = 3 packets



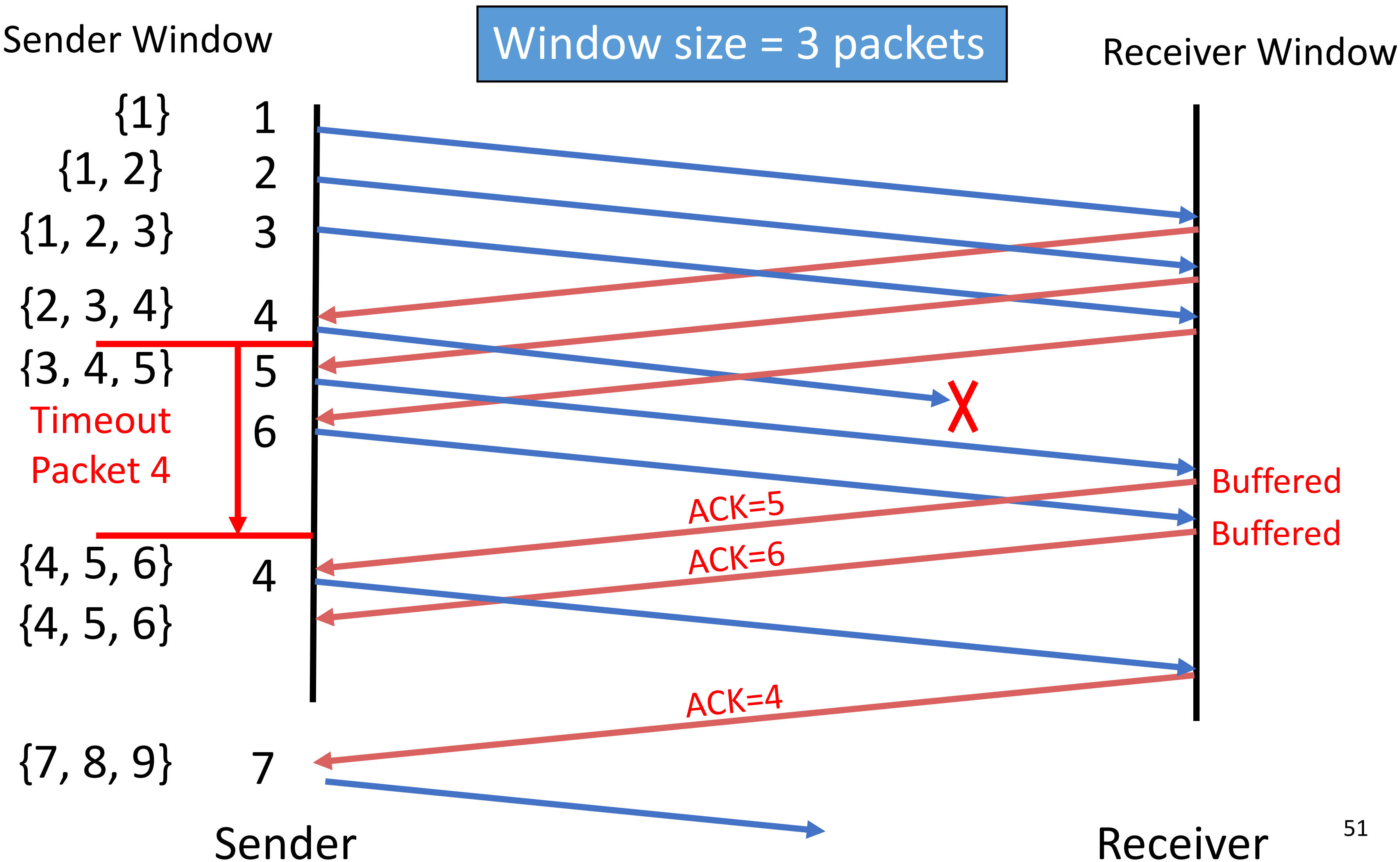
GBN example with errors



Selective Repeat (SR)

- **Sender: transmit up to n unacknowledged packets**
- **Assume packet k is lost, k+1 is not**
  - Receiver: indicate packet k+1 correctly received
  - Sender: retransmit only packet k on timeout
- **Efficient in retransmissions but complex book-keeping**
  - Need a timer/flag per packet

SR example with errors



GBN vs. Selective Repeat

- **When would GBN be better?**
  - When error rate is low; wastes bandwidth otherwise

- **When would SR be better?**
  - When error rate is high; otherwise, too complex

Observations

- **With sliding windows, it is possible to fully utilize a link, provided the window size is large enough.**
- **Sender has to buffer all unacknowledged packets, because they may require retransmission**
- **Receiver may be able to accept out-of-order packets, but only up to its buffer limits**
- **Implementation complexity depends on protocol details (GBN vs. SR)**

Components of a solution

- **Checksums (for error detection)**
- **Timers (for loss detection)**
- **Acknowledgments**
  - Cumulative
  - Selective
- **Sequence numbers (duplicates, windows)**
- **Sliding windows (for efficiency)**
- **Reliability protocols use the above to decide when and what to retransmit or acknowledge**

Summary

- **Transport layer allows applications to communicate with each other**
- **Provides unreliable and reliable mechanisms**
- **Possible to build reliable transport over unreliable medium**

- **Assignment 2:** hands-on experience on building a reliable transport
- **Next lecture:** TCP

Thanks!
Q&A