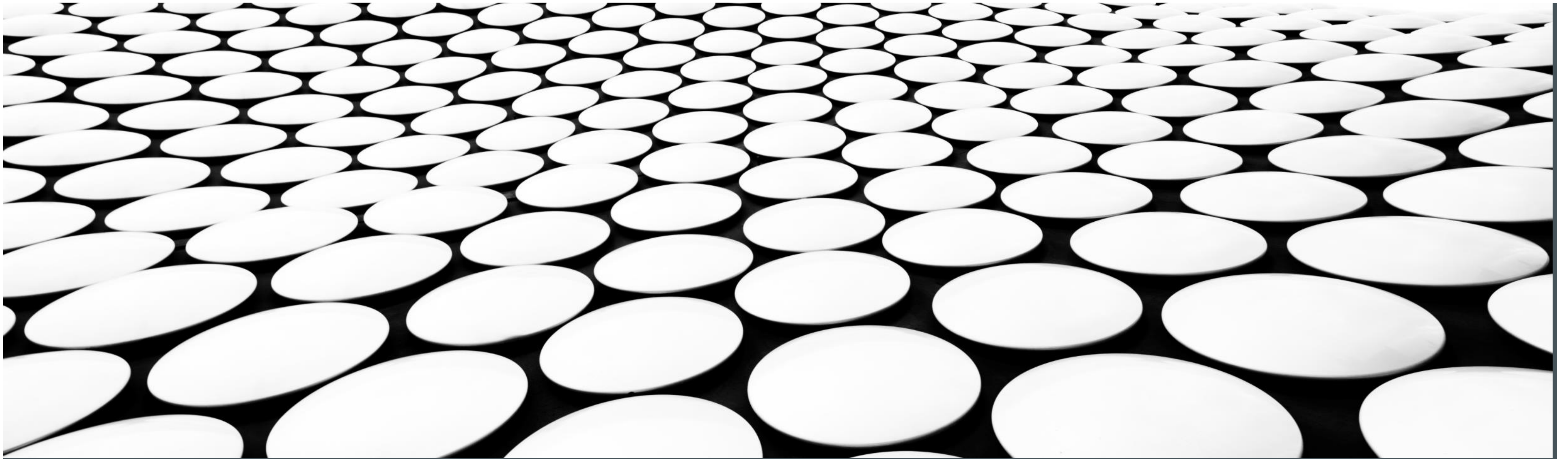
COMPUTER NETWORKS

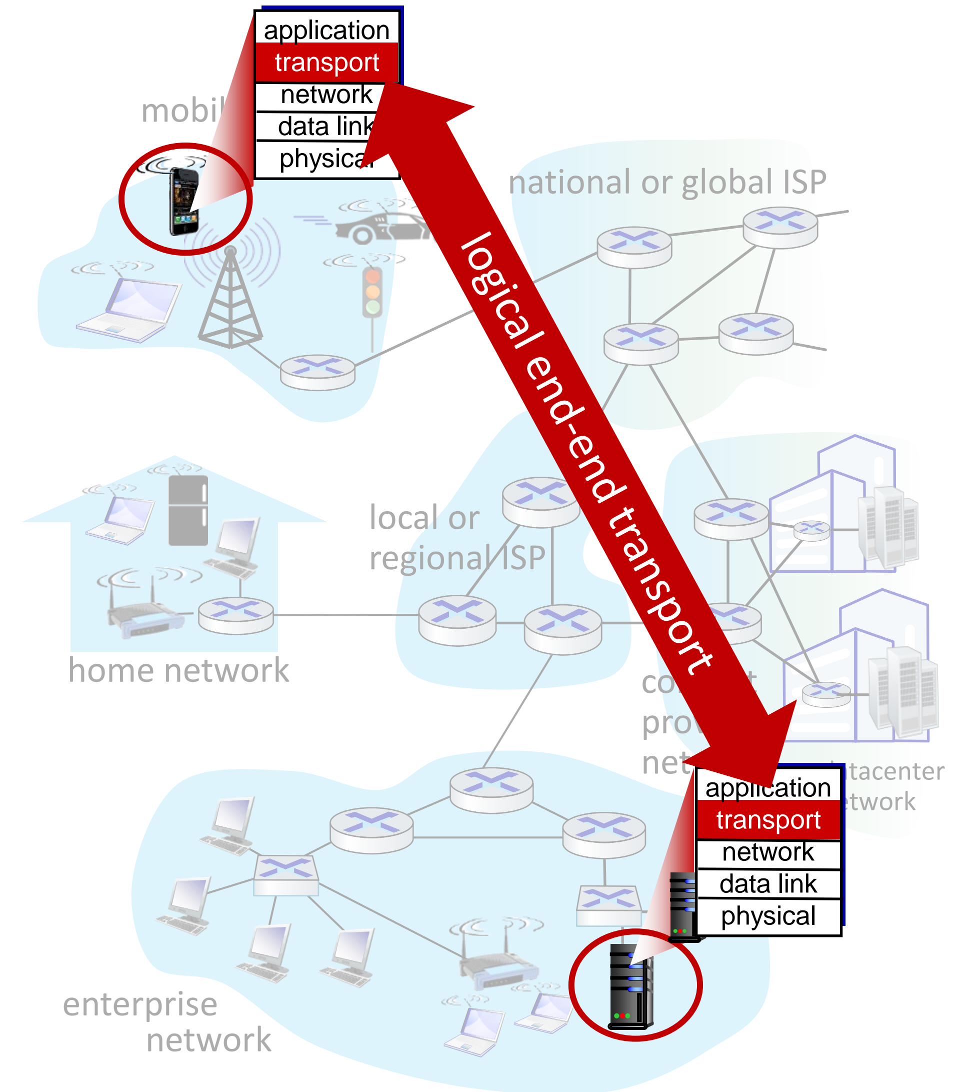
TRANSPORT (CH.24-25)

HEMANT GHAYVAT, (hemant.ghayvat@lnu.se)



TRANSPORT SERVICES AND PROTOCOLS

- Logical communication
- Multiplexing and demultiplexing
- Segmentation followed by Reliability and in order delivery
- Error Control
- Flow Control



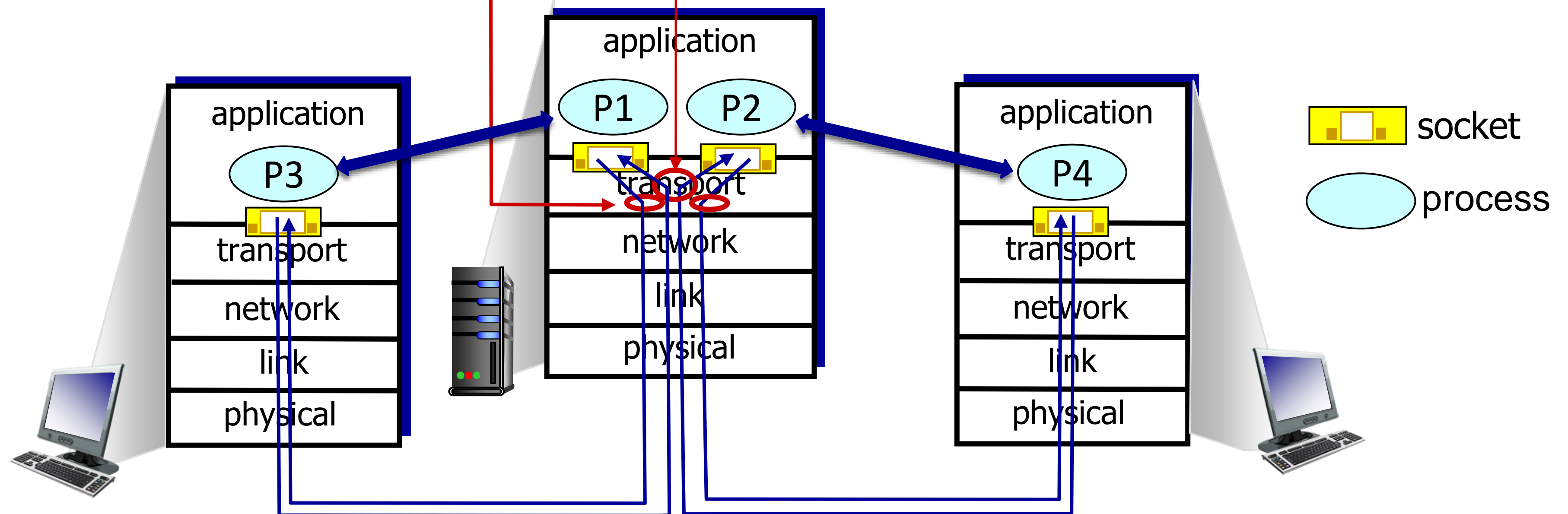
MULTIPLEXING/DEMULTIPLEXING

multiplexing at sender:

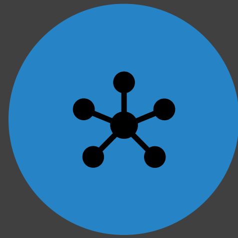
handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:

use header info to deliver received segments to correct socket



TOPICS TO BE COVERED IN THIS SESSION



CONNECTION VS CONNECTIONLESS PARADIGMS

UDP

TCP

PACKET LOSS AND RETRANSMISSION

CONGESTION AND PERFORMANCE

NEEDS AN UPPER LAYER TO PERFORM THESE SERVICES !

THE TRANSPORT LAYER IMPLEMENTS

(De)multiplexing

Error detection

Reliable delivery

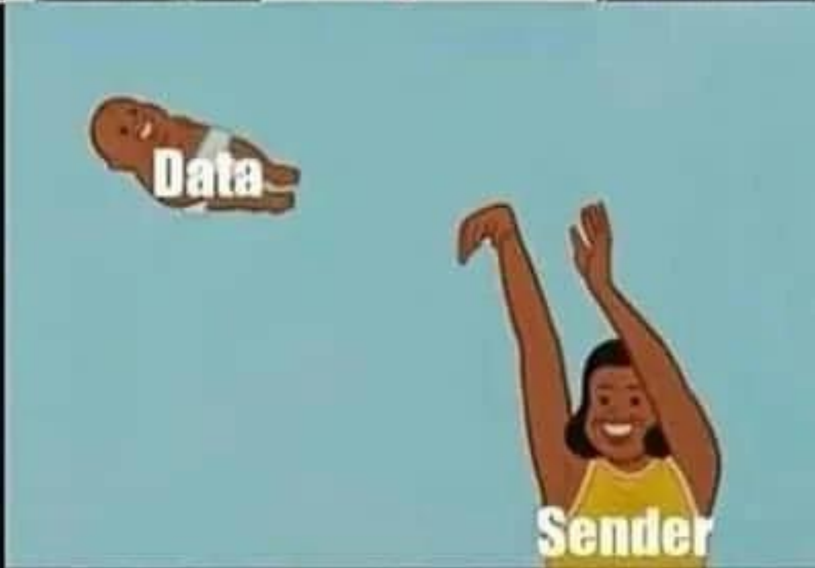
Flow control

TRANSPORT LAYER: ROADMAP

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

TCP

UDP

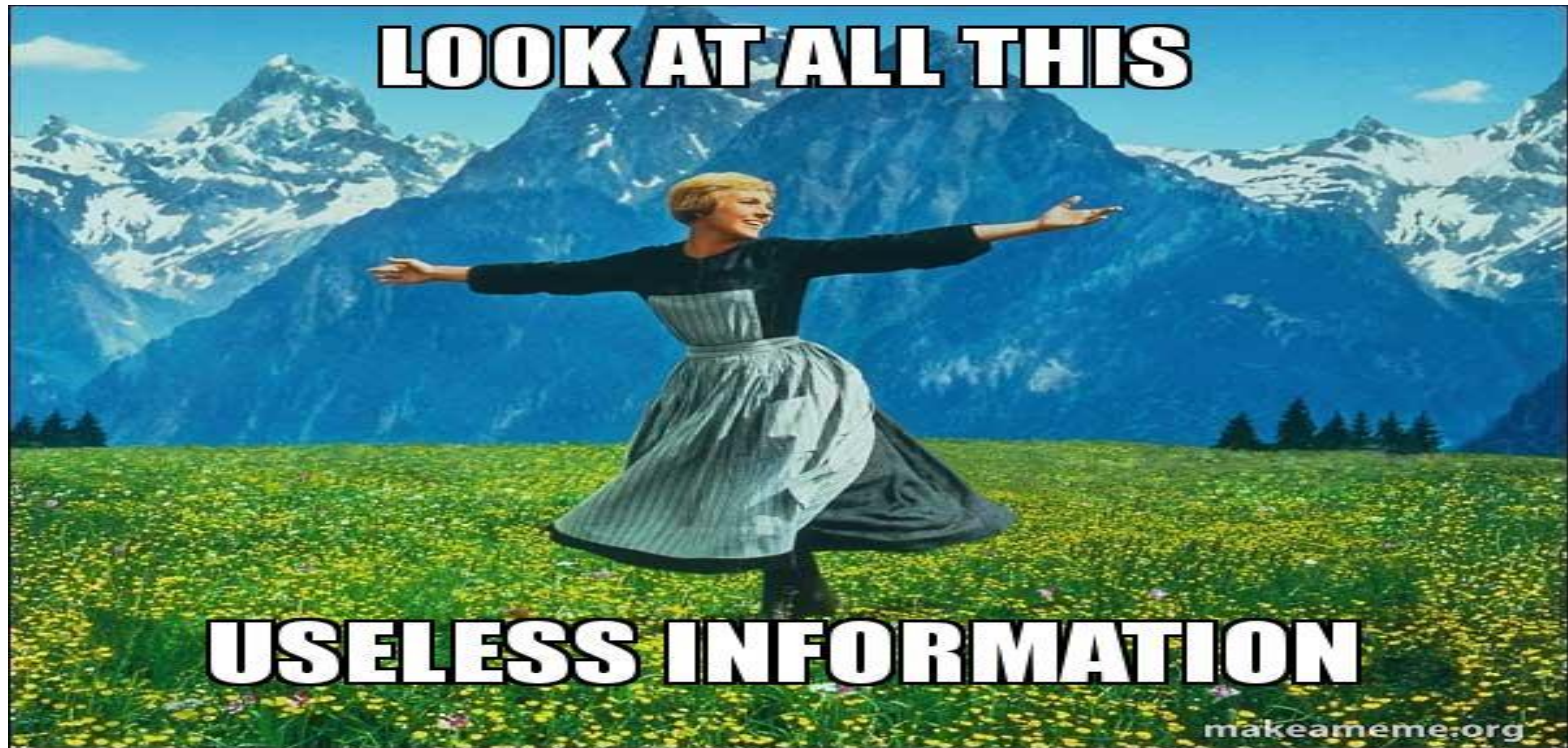


UDP: USER DATAGRAM PROTOCOL

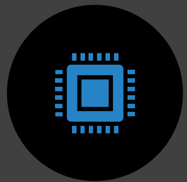
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add RTT delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control
  - UDP can blast away as fast as desired!
  - can function in the face of congestion


LOOK AT ALL THIS
USELESS INFORMATION
makeameme.org

WHY WOULD ANYONE USE UDP?



» Fine control over what data is sent and when

» As soon as app process writes into socket

» … UDP will package data and send packet.

» No delay for connection establishment

» No connection state

» No buffers, sequence #'s, …

» Small packet header overhead

» Header only 8B.

UDP: USER DATAGRAM PROTOCOL

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
  - HTTP/3
- if reliable transfer needed over UDP (e.g., HTTP/3):
  - add needed reliability at application layer
  - add congestion control at application layer

UDP: USER DATAGRAM PROTOCOL [RFC 768]

INTERNET STANDARD

RFC 768 J. Postel
ISI
28 August 1980

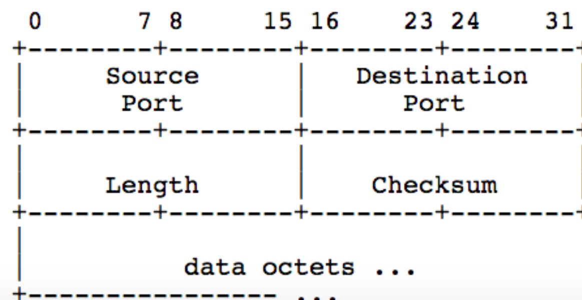
User Datagram Protocol

Introduction

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

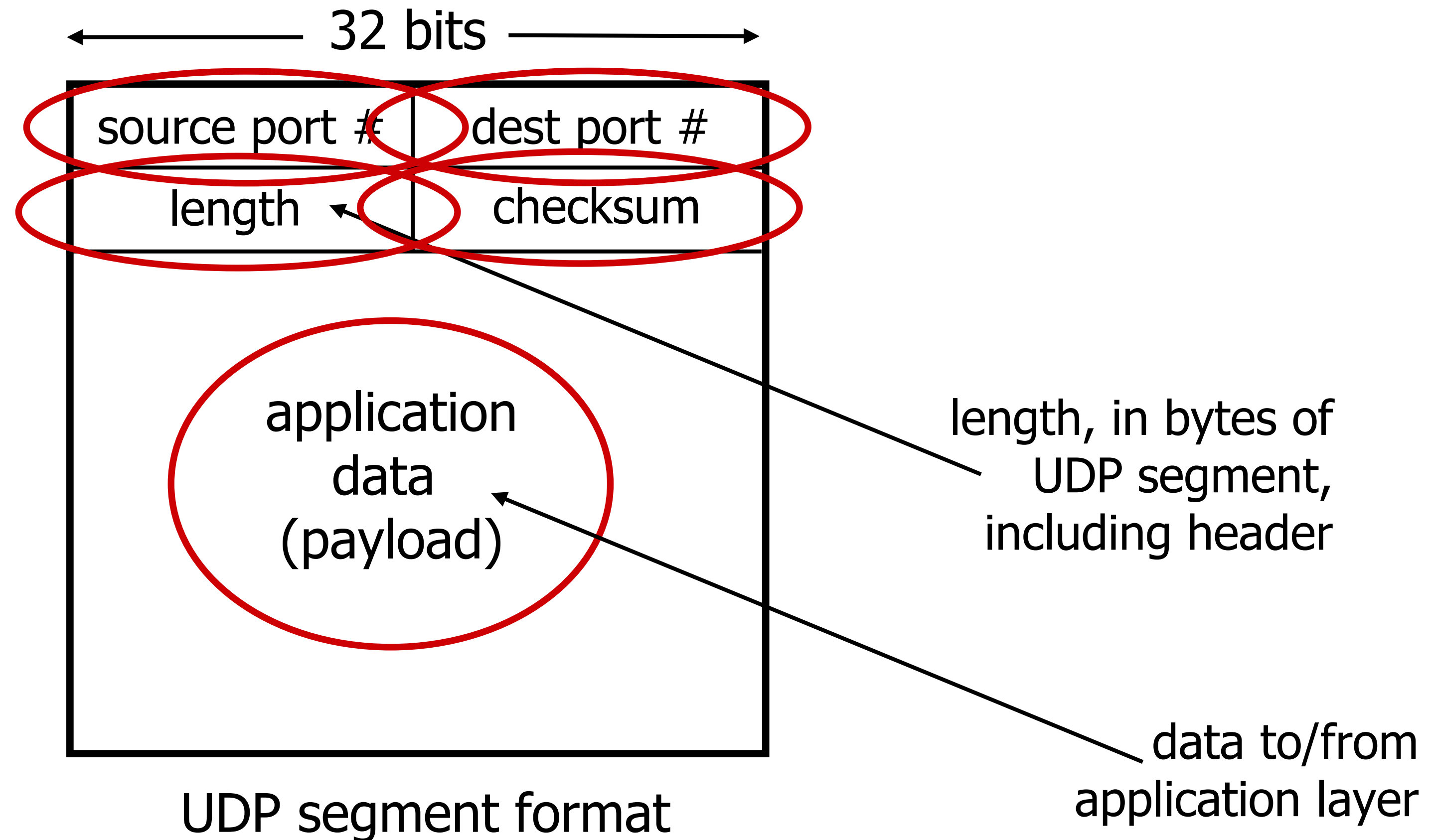
This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format



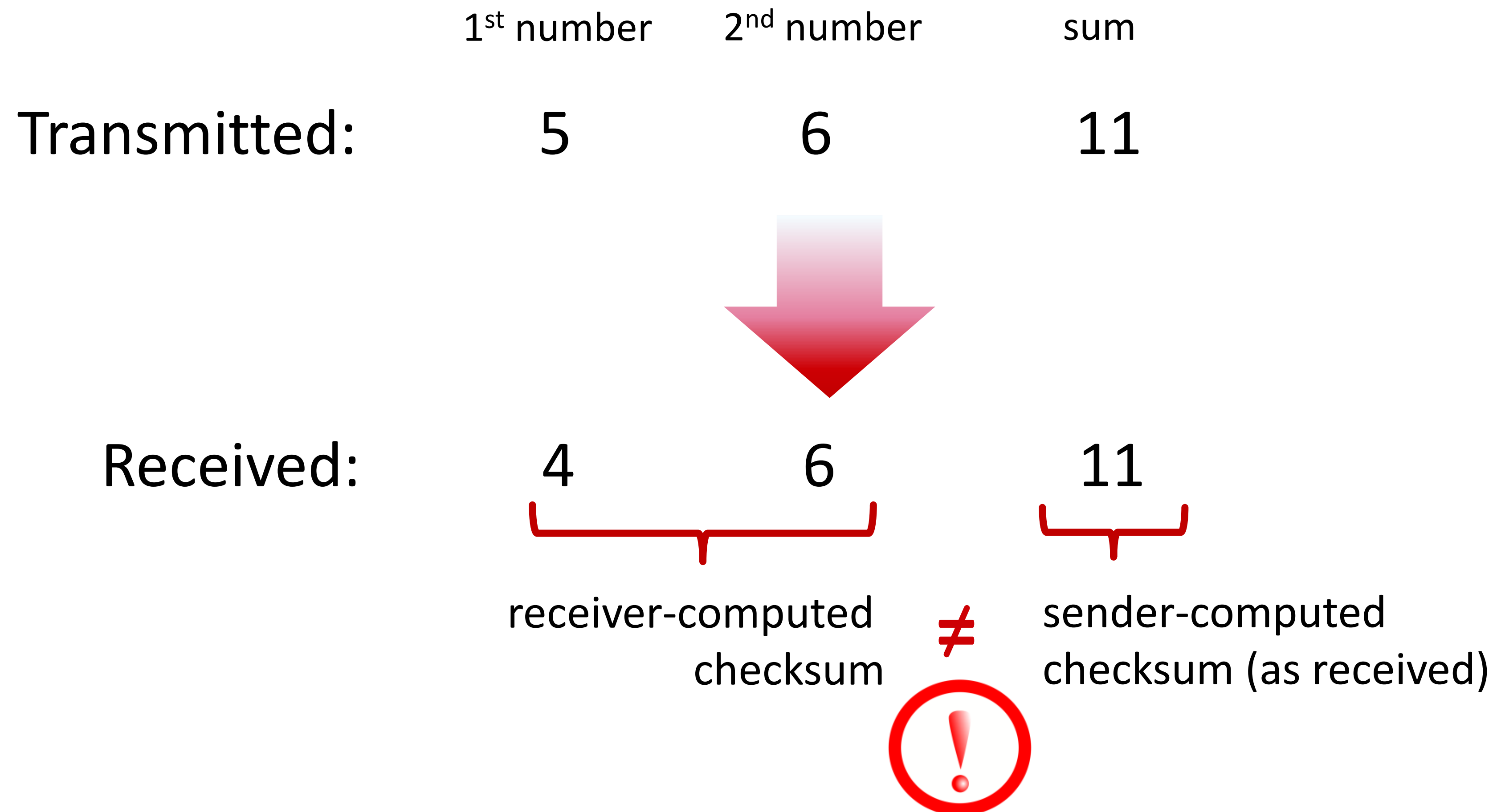
UDP Header

UDP SEGMENT HEADER



UDP CHECKSUM

Goal: detect errors (*i.e.*, flipped bits) in transmitted segment

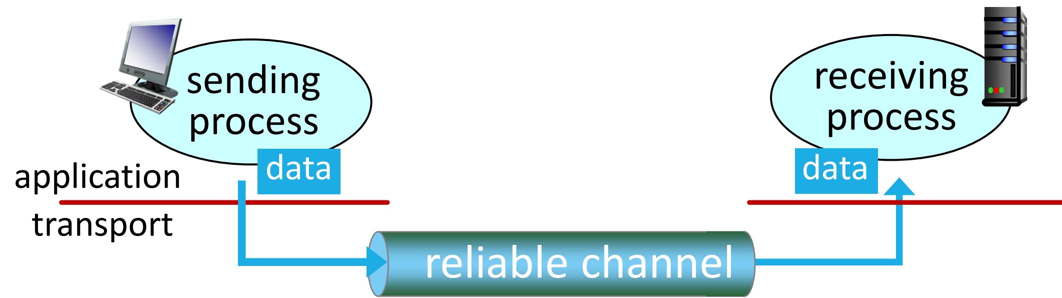


SUMMARY: UDP

- "no frills" (without added feature) protocol:
  - segments may be lost, delivered out of order
  - best effort service: "send and hope for the best"
- UDP has its plusses:
  - no setup/handshaking needed (no RTT incurred)
  - can function when network service is compromised
  - helps with limited reliability (checksum)
- Multimedia streaming (VoIP, Livestreaming movie, Online Gaming, video conferencing, ...) :
  - Retransmitting lost/corrupted packets is not worthwhile,
  - anyways by the time the packet is retransmitted, it is too late

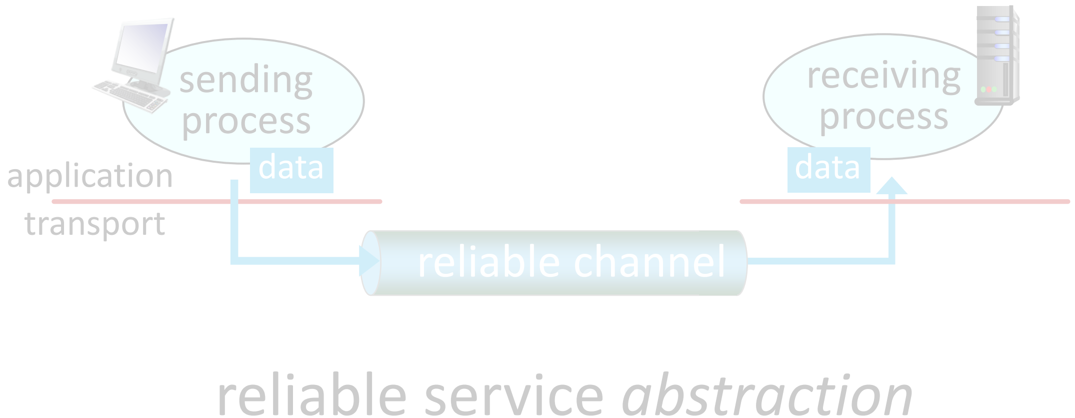
PRINCIPLES OF RELIABLE DATA TRANSFER

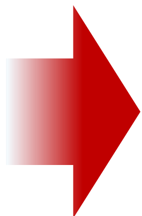
Do you remember Simplex, Half Duplex and Full duplex!!!!!

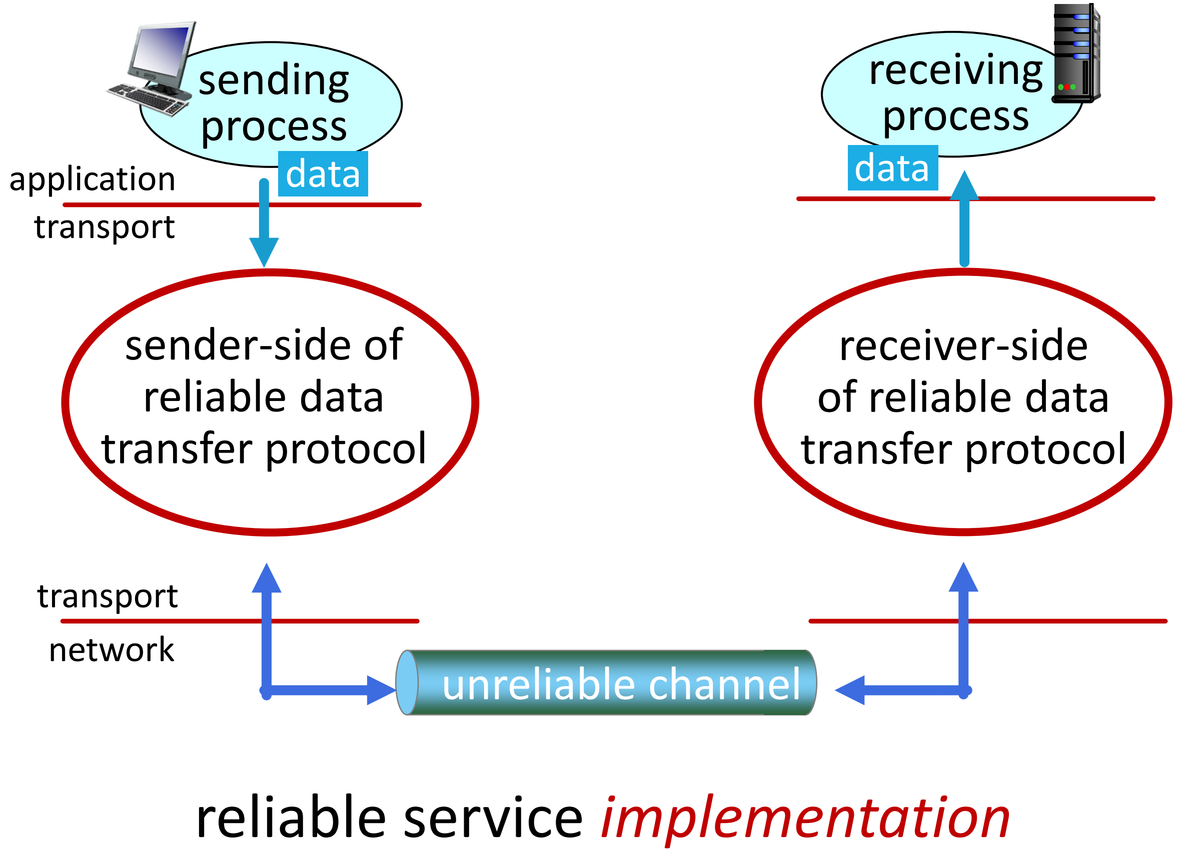


reliable service *abstraction*

PRINCIPLES OF RELIABLE DATA TRANSFER







reliable service *implementation*

CHALLENGES OF RELIABLE DATA TRANSFER

- **Over a perfectly reliable channel:**

  All data arrives in order, just as sent. Simple!

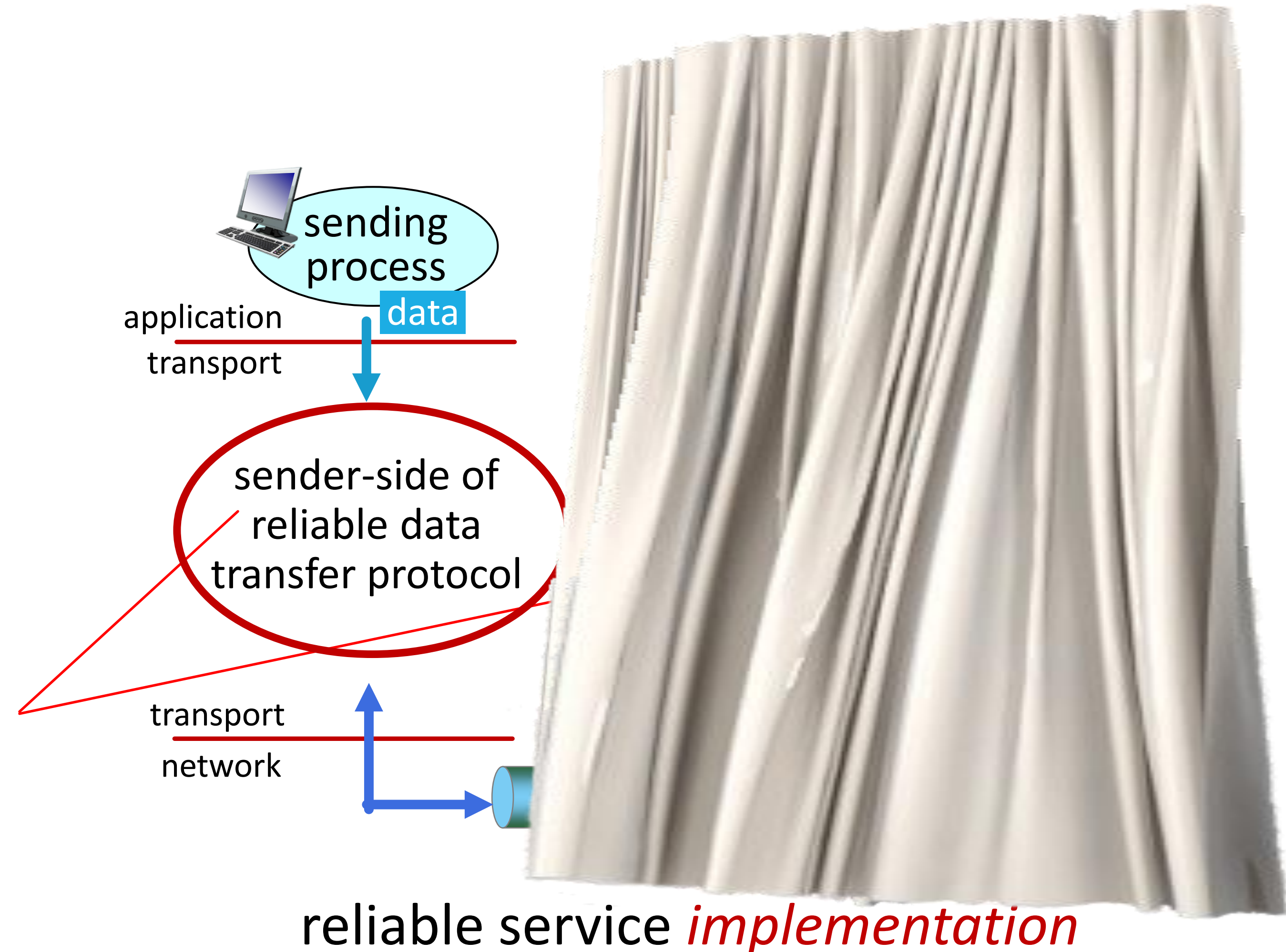
- **Over a channel with bit errors or Unreliable channel:**

  1. All data arrives in order, but some bits corrupted.
  2. Receiver detects errors and says “please repeat that”.
  3. Sender retransmits corrupted data

PRINCIPLES OF RELIABLE DATA TRANSFER

Sender, receiver do *not* know the “state” of each other, e.g., was a message received?

- unless communicated via a message



reliable service *implementation*

CHALLENGES OF RELIABLE DATA TRANSFER

Over a lossy channel with bit errors:

» Some data is missing, and some bits are corrupted.

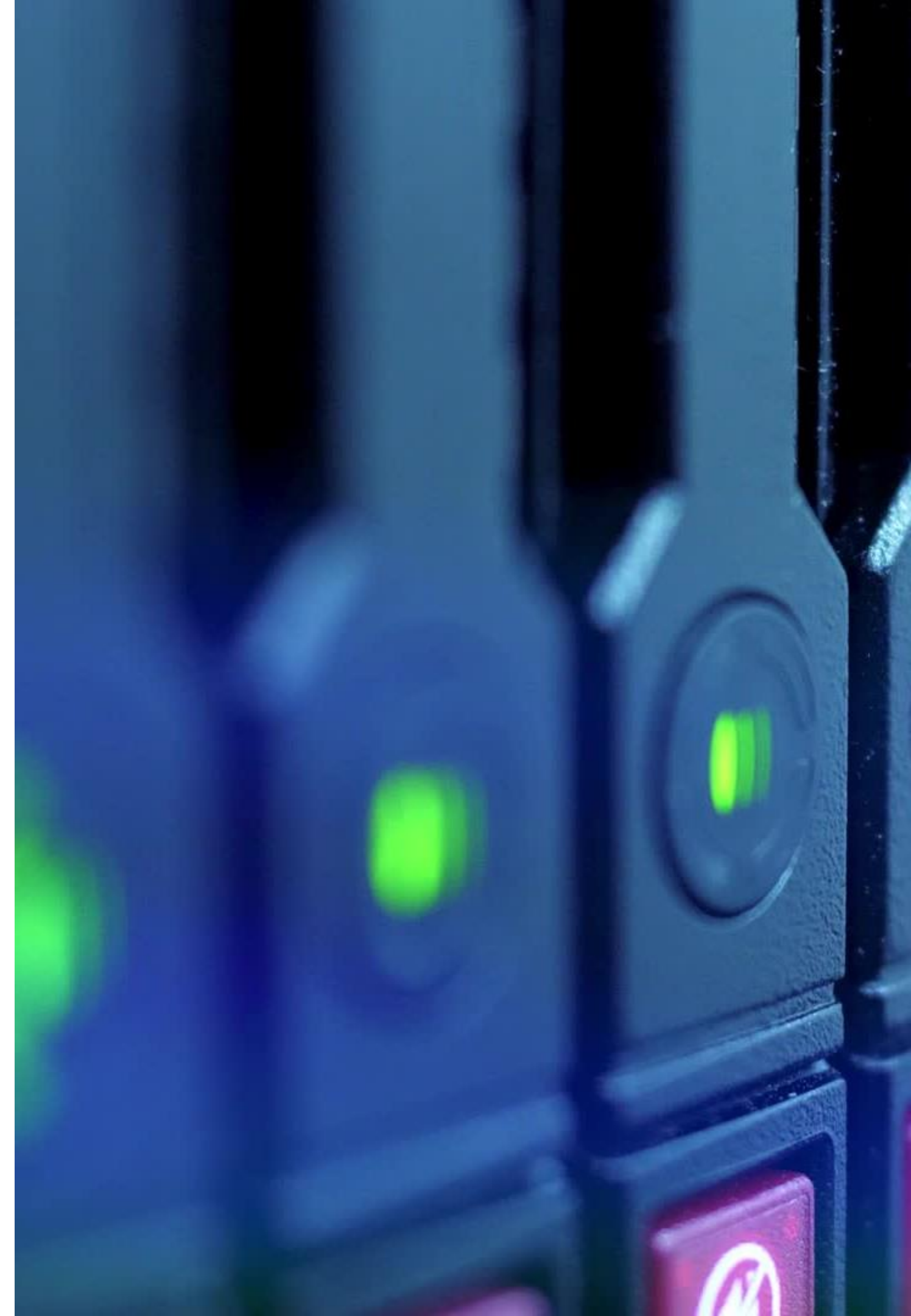
» Receiver detects errors but cannot always detect loss.

» Sender must wait for acknowledgment ("ACK" or "OK")

» ...and retransmit data after some time if no ACK arrives.

TCP SUPPORT FOR RELIABLE DELIVERY

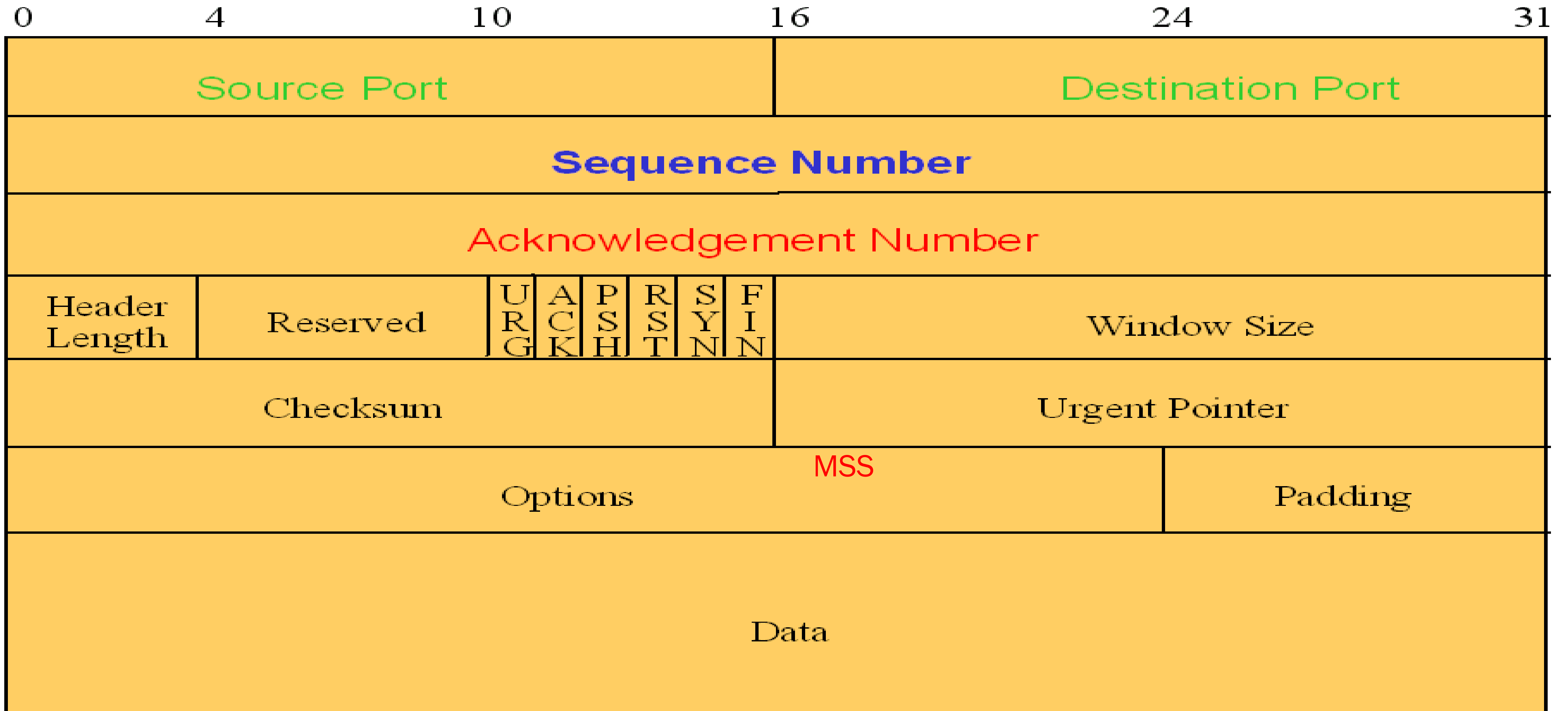
- **Detect bit errors: checksum**
- » Used to detect corrupted data at the receiver
- » …leading the receiver to drop the packet.
- **Detect missing data: sequence number**
- » Used to detect a gap in the stream of bytes
- » ... and for putting the data back in order.
- **Recover from lost data: retransmission**

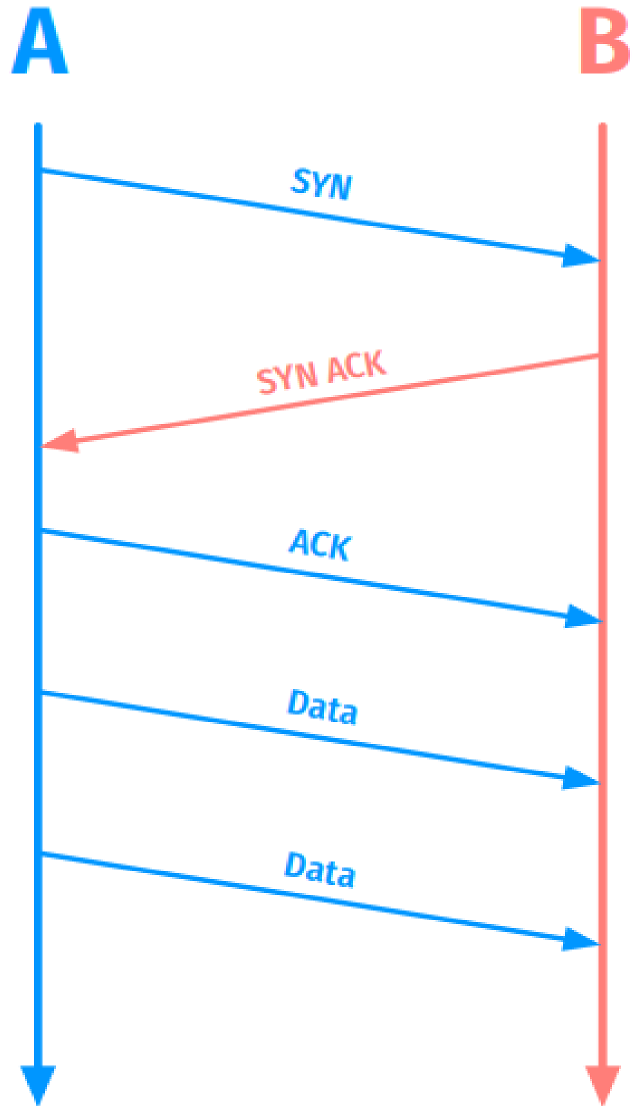


CONNECTION ESTABLISHMENT

RESERVING THE BUFFER, CPU AND BW TO ESTABLISH CONNECTION PROPERLY

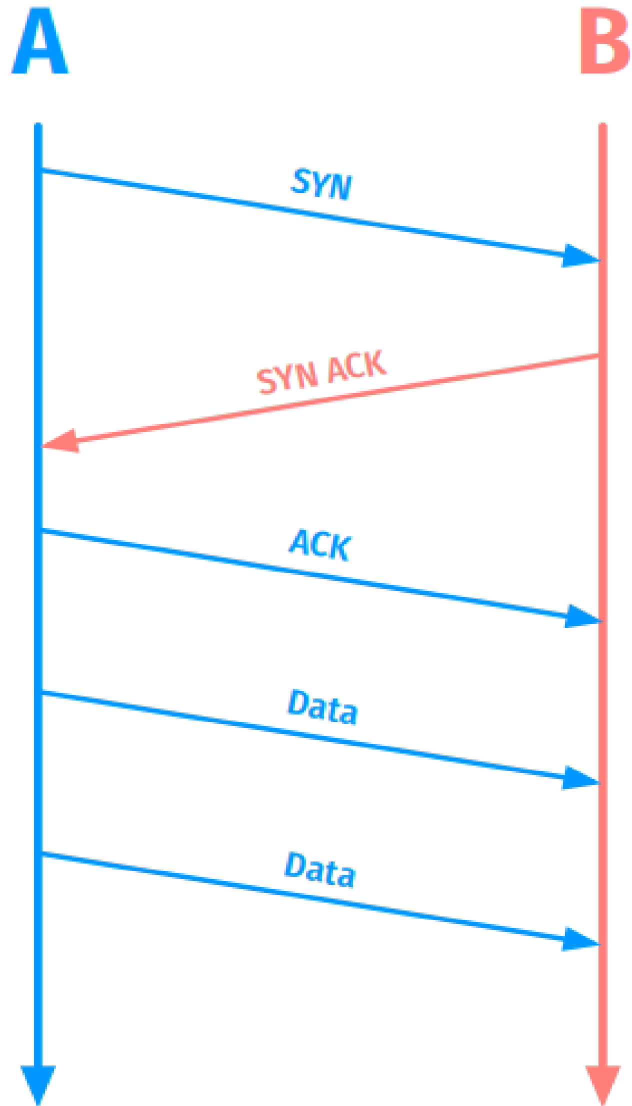
TCP HEADER





ESTABLISHING A TCP CONNECTION

- Each host tells its Initial Sequence Number (ISN) to the other host



ESTABLISHING A TCP CONNECTION

- Three-way handshake to establish connection:

1. Host A sends a SYNchronize to host B.

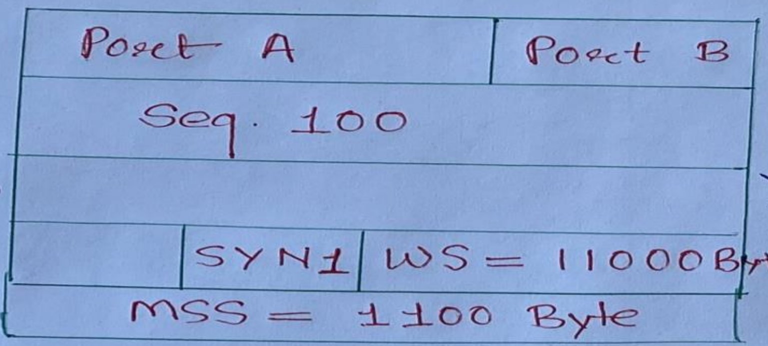
2. Host B returns a SYNchronize ACKnowledgment.

3. Host A sends an ACK to acknowledge the previous SYN.

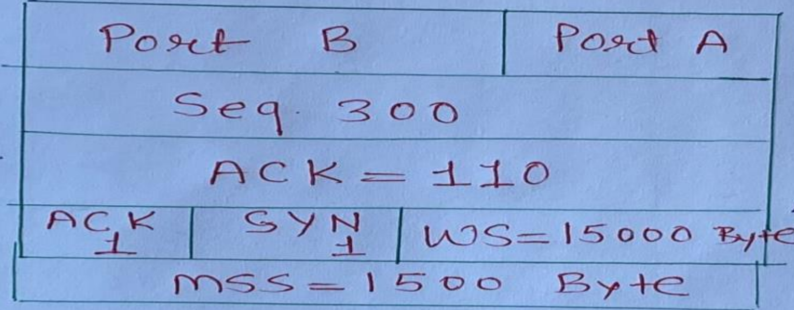
Client (A)

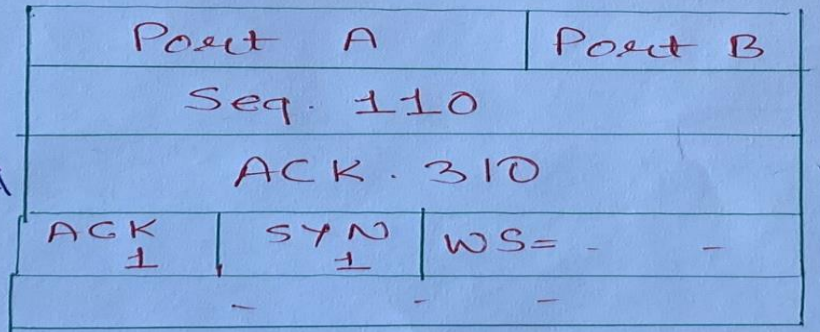
Server (B)

Always active

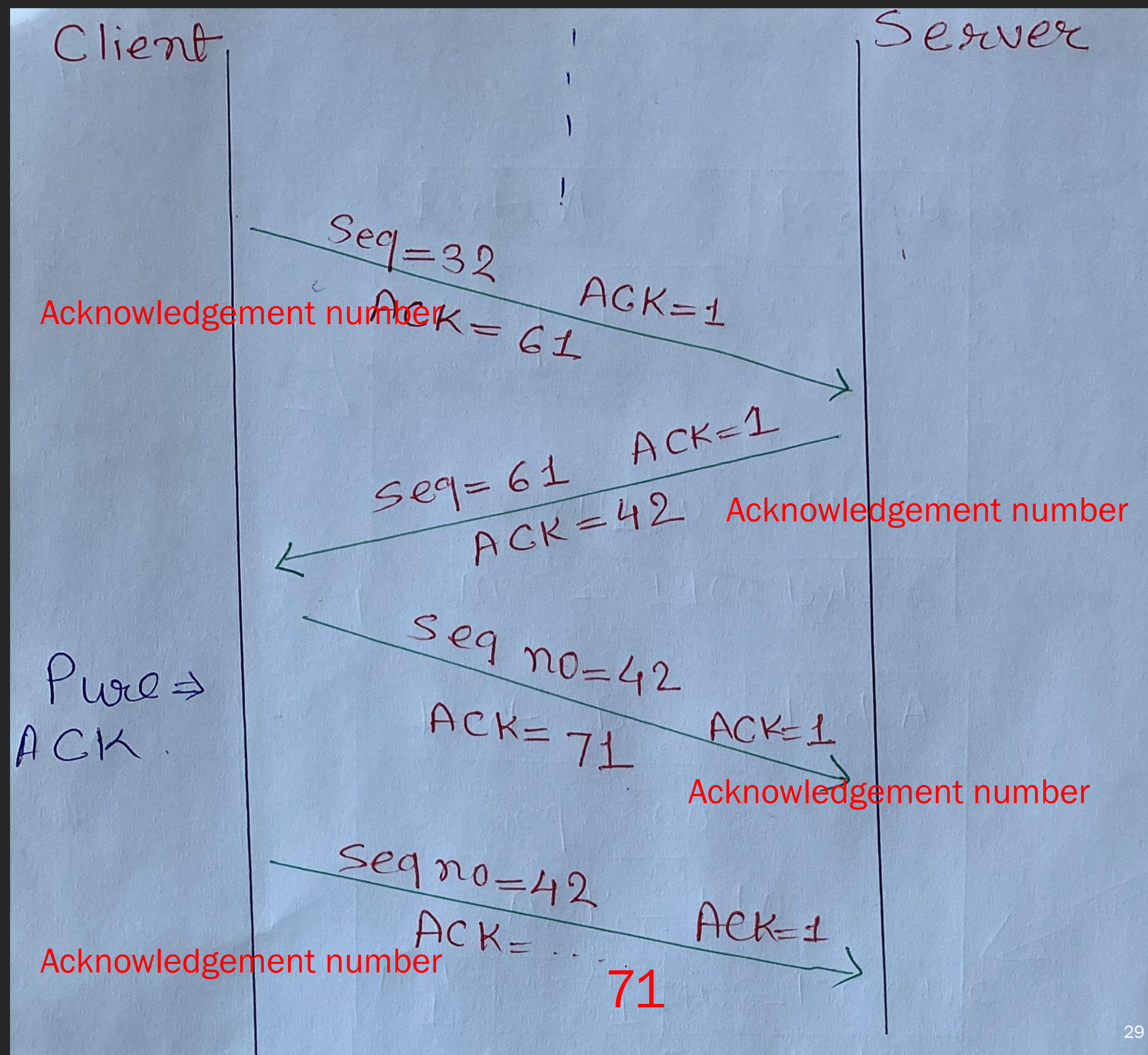


Acknowledgement number





PIGGYBACKING AND PURE ACK



ANY QUESTIONS???

TEARING DOWN THE CONNECTION

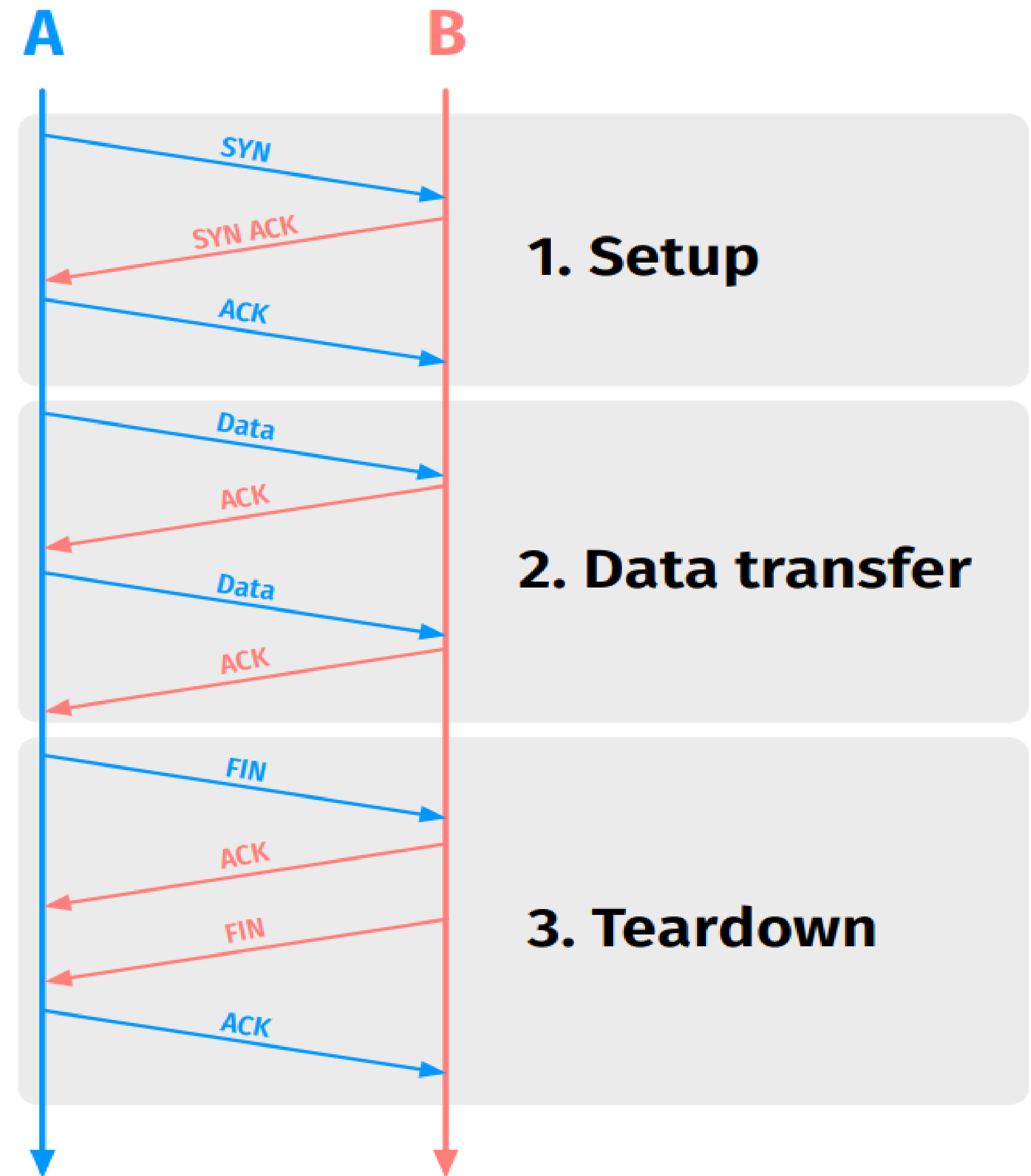
- Closing a connection:
  - » Process done writing, invokes close().
  - » Once TCP sends all outstanding byte, TCP sends a FINish message.
- Receiving a FINish:
  - » Process reading data from socket.

TEARING DOWN THE CONNECTION

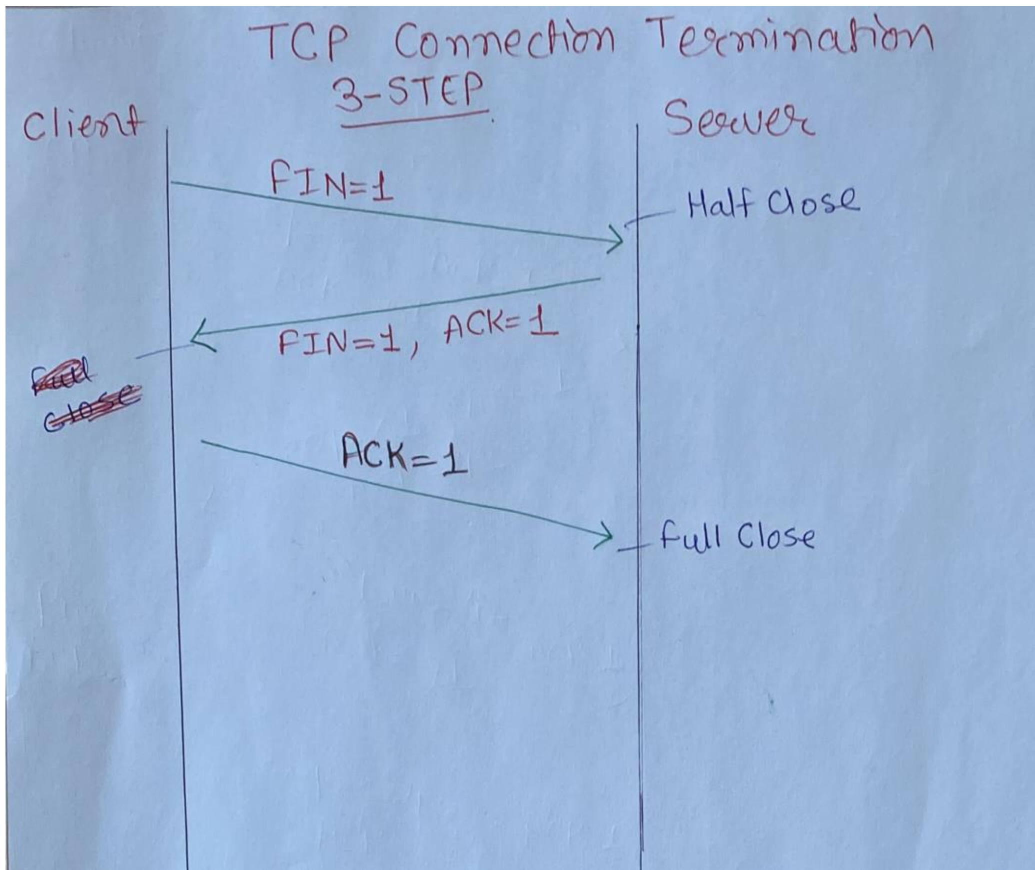
- Tear-down is two-way:

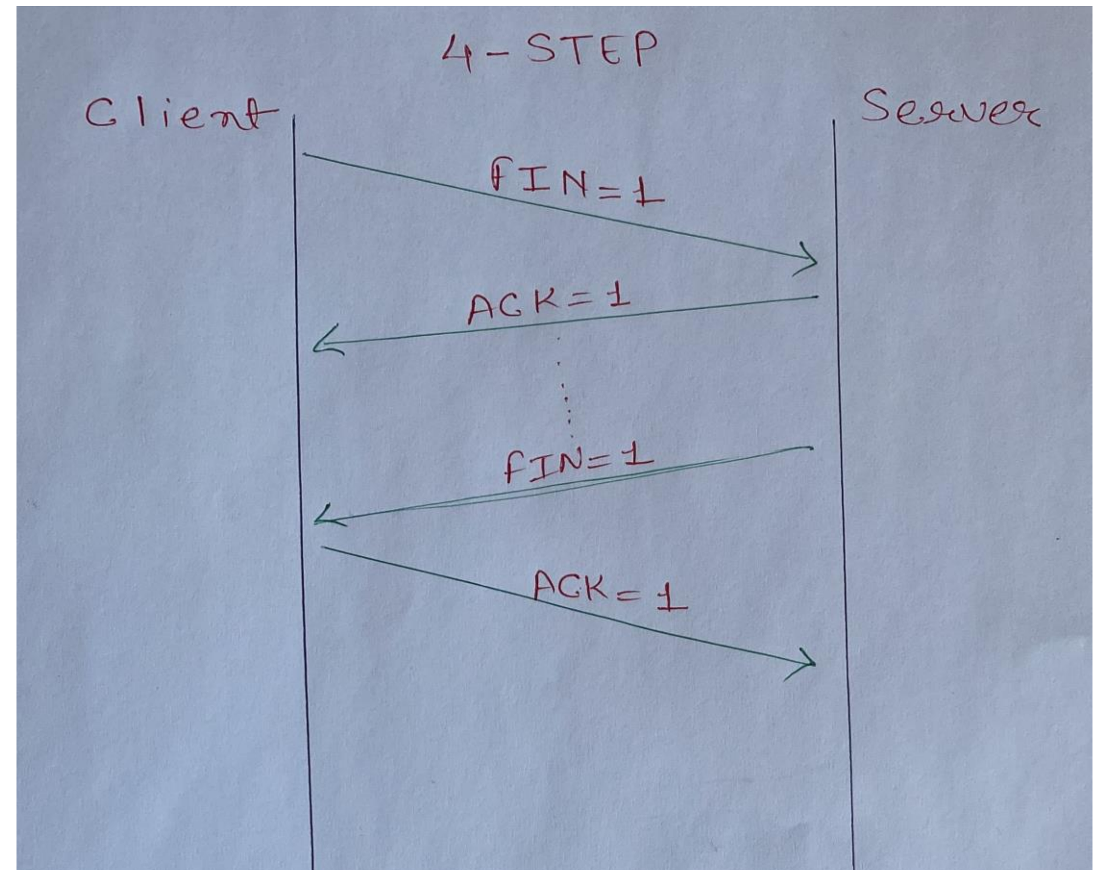
  » FIN to close, but receive remaining

  » other host ACKs the FIN



TCP CONNECTION TERMINATION : 3 AND 4 STEPS





WHAT IF THE SYN PACKET GETS LOST?

- Suppose the SYN packet gets lost:
  - Packet lost inside network, or
  - server rejects packet (e.g., listen queue is full).
- Eventually, a none SYN-ACK arrives:
  - Sender sets timer and waits, retransmitting if needed.

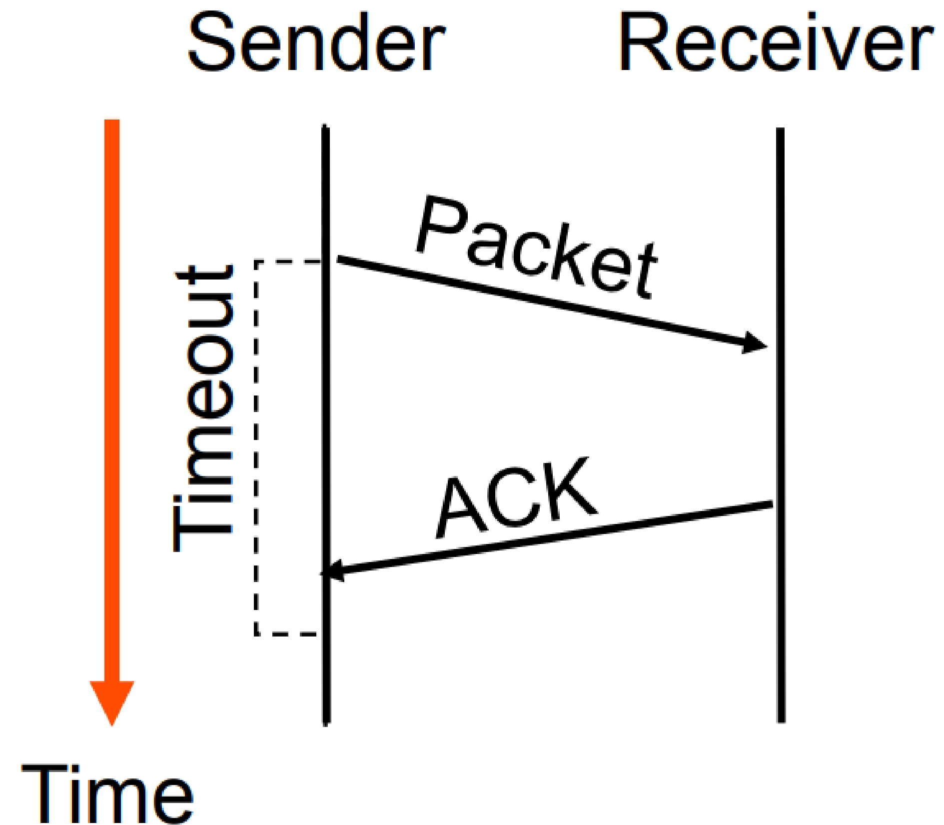
WHAT IF THE SYN PACKET GETS LOST?

- How should TCP sender set timer?
  - Sender has no estimate of RTT
    - Some TCP stacks use a default of 3 or 6 seconds
- Why the reload button in your browser is useful:
  - Opens new connection and "retransmits"

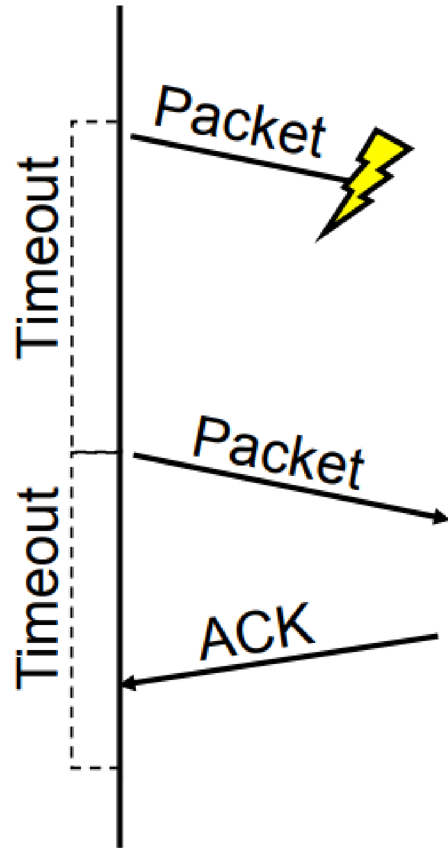
EXCLUSIVE SERVICE OF TCP : ARQ

AUTOMATIC REPEAT REQUEST (ARQ)

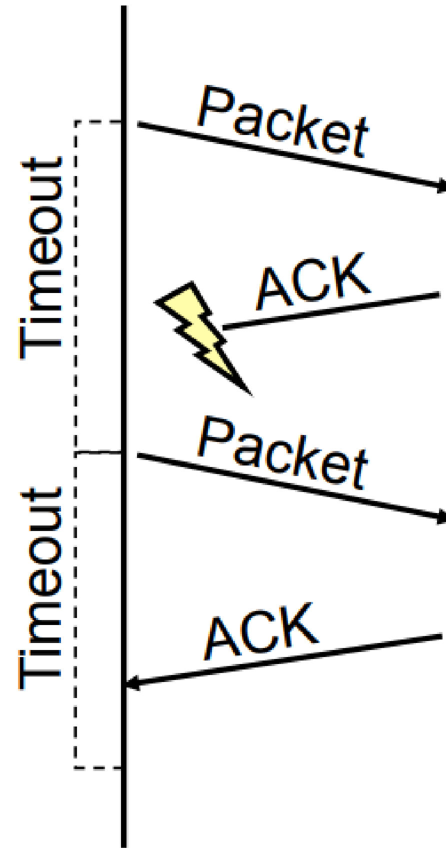
- 1. **Receiver sends ACK** when it receives packet
- 2. **Sender waits for ACK.**
- » if ACK not received within some timeout period,
- resend packet Simplest:
- **"Stop and wait"**



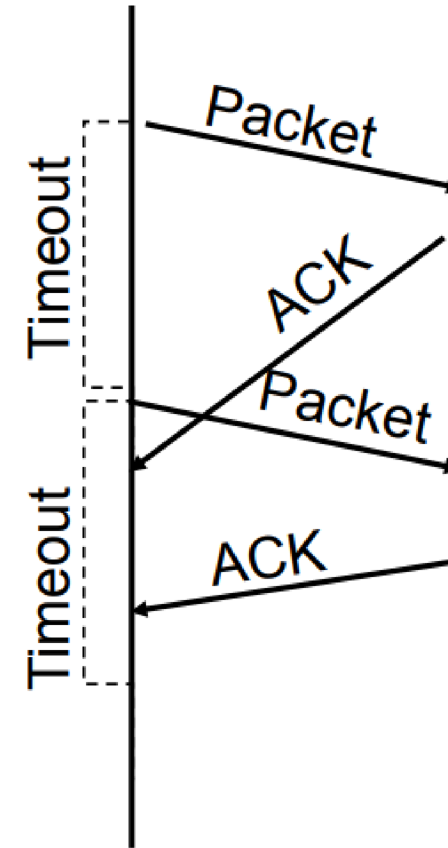
REASONS FOR RETRANSMISSION



Packet lost



ACK lost
DUPLICATE PACKET



Early timeout
DUPLICATE PACKETS

HOW LONG SHOULD THE SENDER WAIT?

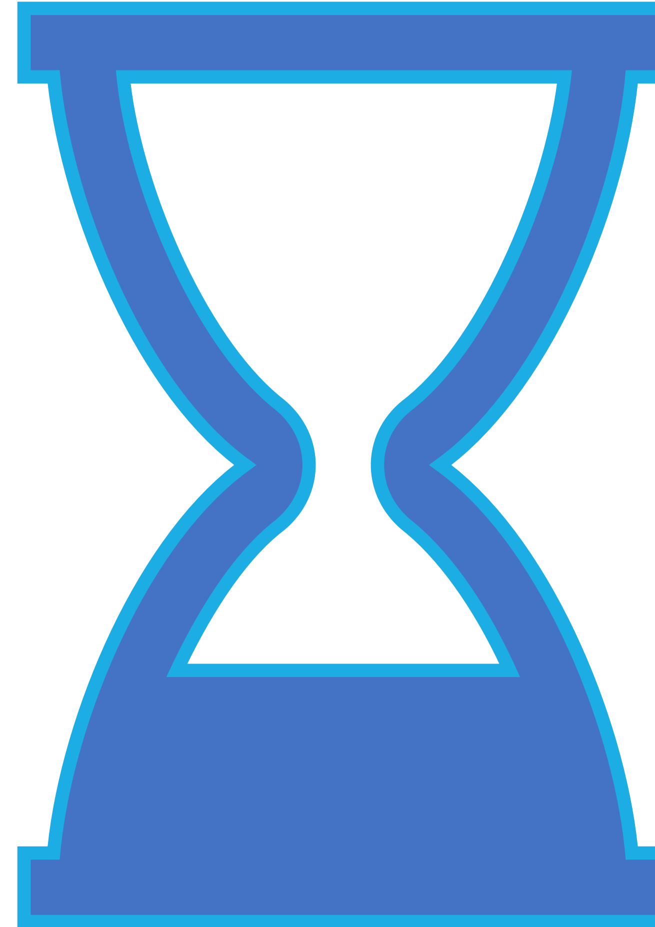
HOW LONG …

- » Too short?

» Wasted retransmissions

- » Too long?

» Excessive delays when packet lost



HOW LONG …

- » TCP sets timeout as function of **Round Trip Time (RTT) or round trip delay**
- » ACK should arrive after **RTT + fudge factor** for queuing
- » How does sender know RTT? Can estimate RTT by watching the ACKs.

TCP ROUND TRIP TIME, TIMEOUT

Q: how to set TCP timeout value?

- longer than RTT, but RTT varies!
- too short: premature timeout, unnecessary retransmissions
- too long: slow reaction to segment loss

Q: how to estimate RTT?

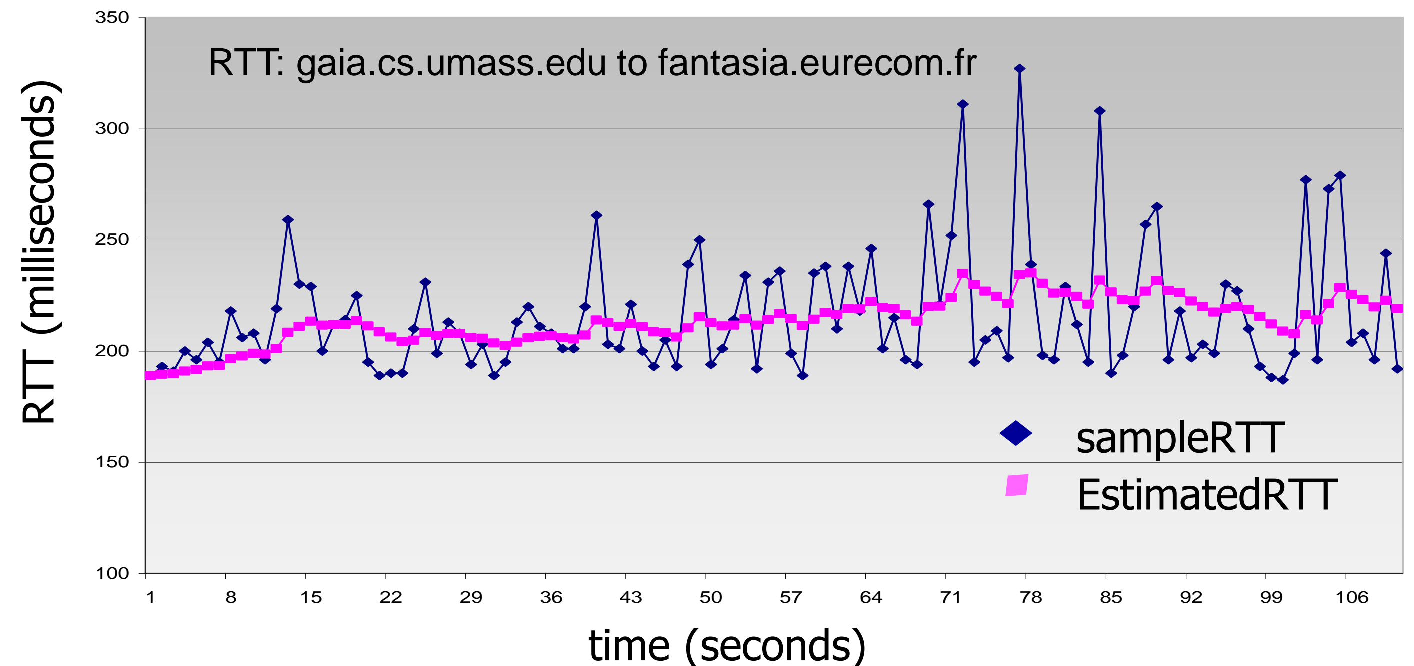
- It could be done by measuring the (actual) sampleRTT, sampleRTT is measured by calculating the past ACKs samples and related time out time.

TCP ROUND TRIP TIME, TIMEOUT

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

- exponential weighted moving average (EWMA)
- influence of past sample decreases exponentially fast
- typical value: α = 0.125

alpha reflects the influence of the most recent measurements on the estimated RTT



TCP ROUND TRIP TIME, TIMEOUT

- timeout interval: **EstimatedRTT** plus "safety margin"
  - large variation in **EstimatedRTT from SampleRTT:** want a larger safety margin

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

estimated RTT

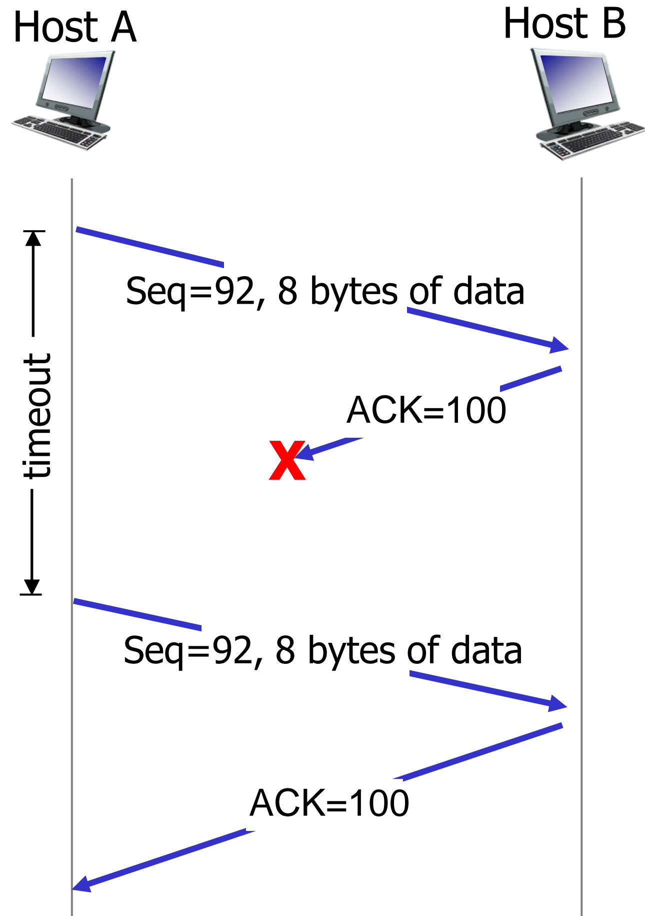
"safety margin"

- **DevRTT**: EWMA of **SampleRTT** deviation from **EstimatedRTT**:

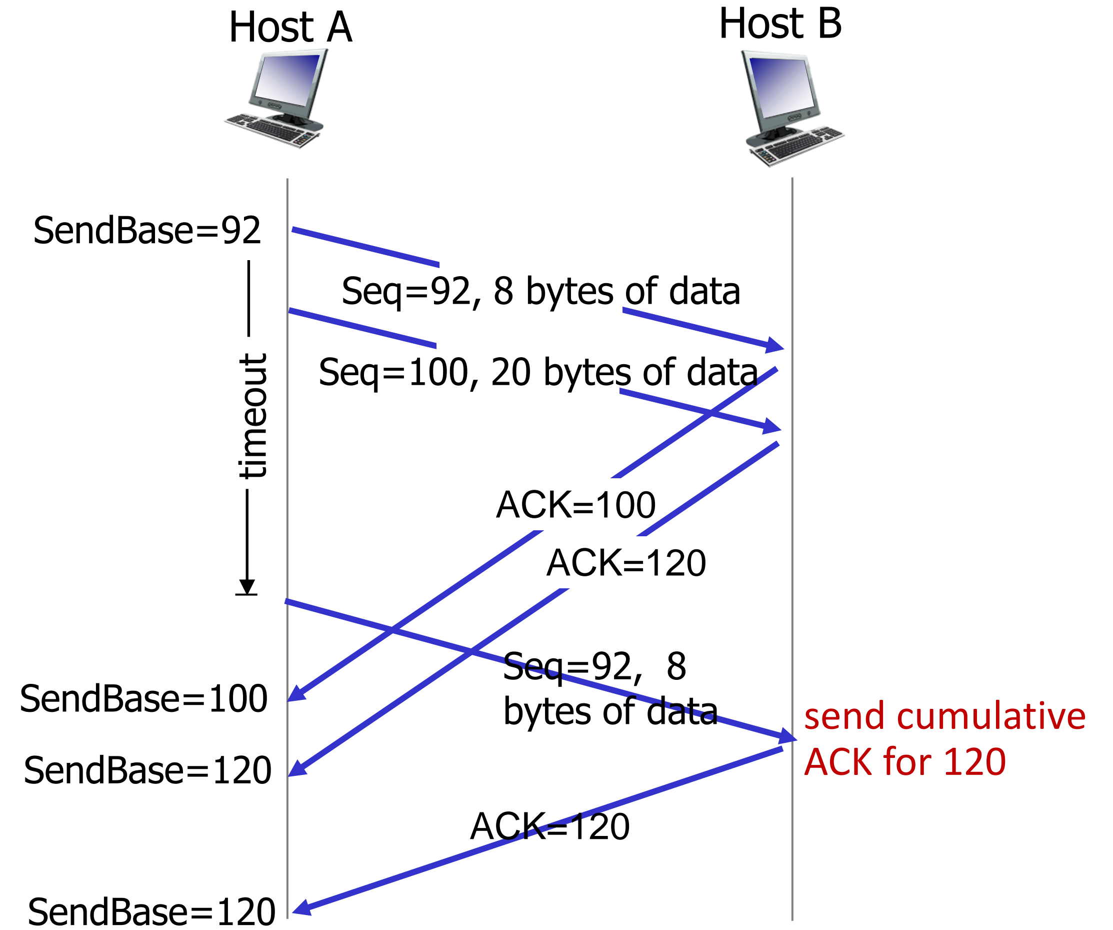
```
DevRTT = (1-β)*RecentlyDevRTT + β*|SampleRTT-EstimatedRTT|
```

(typically, β = 0.25)

TCP: RETRANSMISSION SCENARIOS

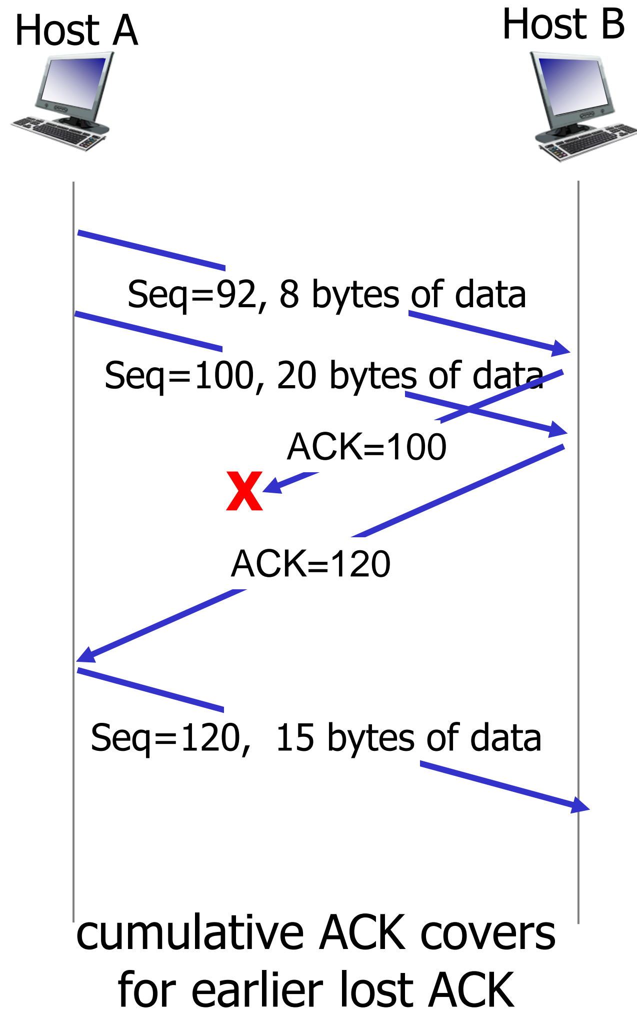


lost ACK scenario



premature timeout

TCP: RETRANSMISSION SCENARIOS



RECEIVER BUFFERING

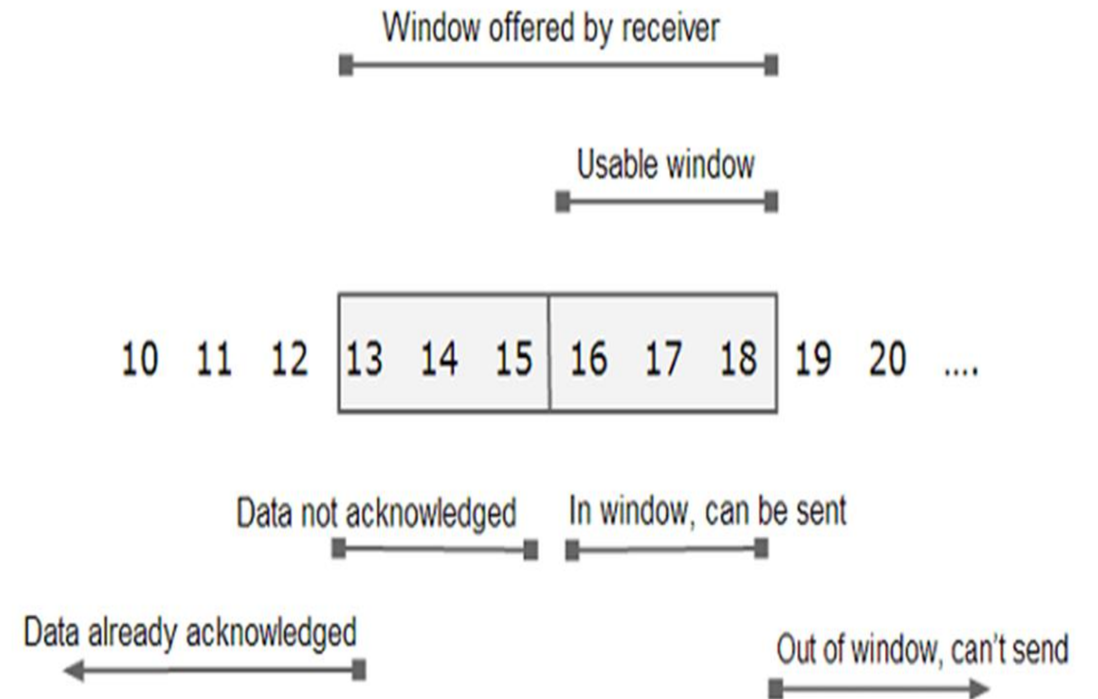
- **Window size:** amount that can be sent without ACK, because receiver can buffer.
- **Receiver advertises window to client:**
  - » Tells amount of free space left (in bytes).
  - » Sender agrees not to exceed this amount.

SLIDING WINDOW

- Sender must maintain a window on its side . This window covers unacknowledged data and the data it can send keeping in mind the window size advertised by the receiver

IN THE CASE SHOWN

- The available window advertised by the receiver is 6. This means that receiver can accept 6 bytes as of now.
- The window at sender side covers bytes ranging from 13 to 18 (I.e. 6 bytes in total).
- Out of this range, 13-15 are the bytes which have been sent but no acknowledgement is yet received for them.
- Bytes 16-18 are the bytes that sender can send as soon as possible.
- If sender starts receiving acknowledgement for bytes 13 to 15, the left end of the window starts closing in.
- The right end starts opening up as more and more window size is advertised by the receiver.
- This window slides towards right depending upon how fast receiver consumes data and sends acknowledgement and hence known as sliding window.



FAST RETRANSMISSION

- Better solution possible under sliding window. Although packet n might have been lost, packets n+1, n+2, and so on might get through.
- **Idea: "Duplicate ACKs" suggest loss:**
  - » ACK says receiver is awaiting nth packet.
  - » Repeated ACKs suggest later packets have arrived.
  - » Sender use "duplicate ACKs" as early hint of loss.

Fast retransmission, sender retransmits data after the triple duplicate ACK.

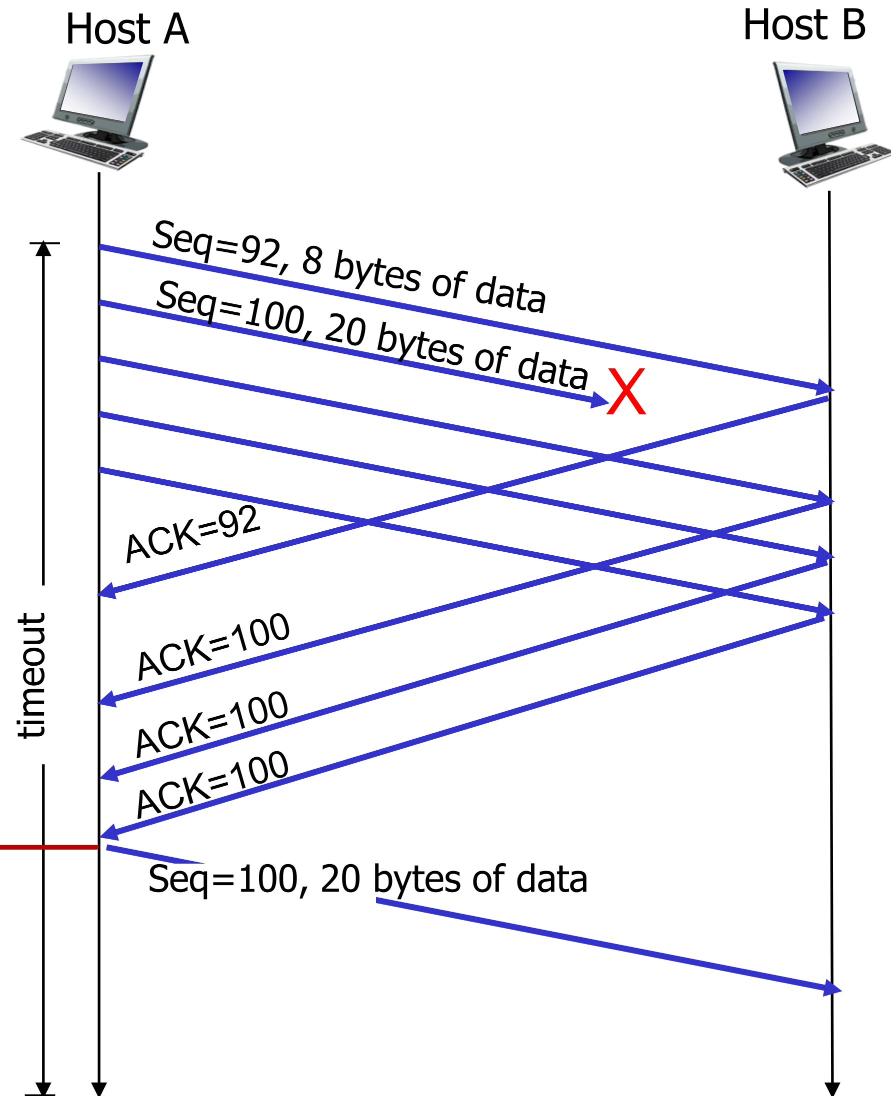
TCP FAST RETRANSMIT

TCP fast retransmit

if sender receives 3 additional ACKs for same data ("triple duplicate ACKs"), resend unACKed segment with smallest seq #

- likely that unACKed segment lost, so don't wait for timeout

Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!

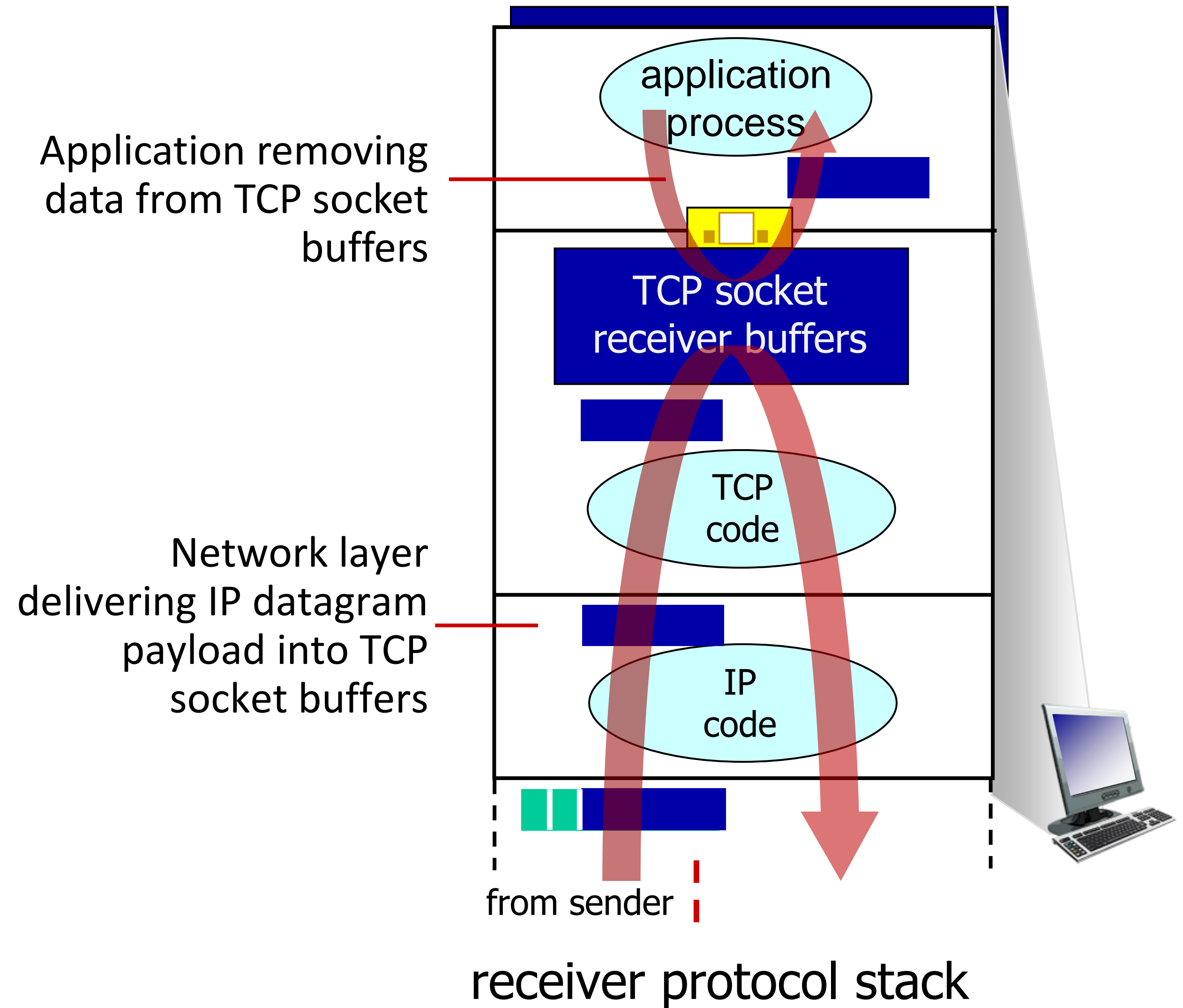


EFFECTIVENESS OF FAST RETRANSMIT

- When does Fast Retransmit work best?
  - » Long transfers: high likelihood of many pkts in flight.
  - » Large window: high likelihood of many packets in flight.
  - » Low loss burstiness: higher likelihood that later packets arrive.

TCP FLOW CONTROL

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



TCP FLOW CONTROL

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?

application process

Application removing data from TCP socket buffers

TCP socket receiver buffers

TCP code

Network layer delivering IP datagram payload into TCP socket buffers

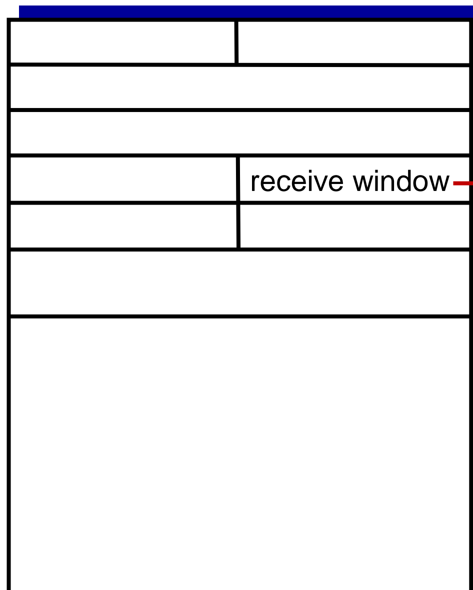
IP code

from sender

receiver protocol stack

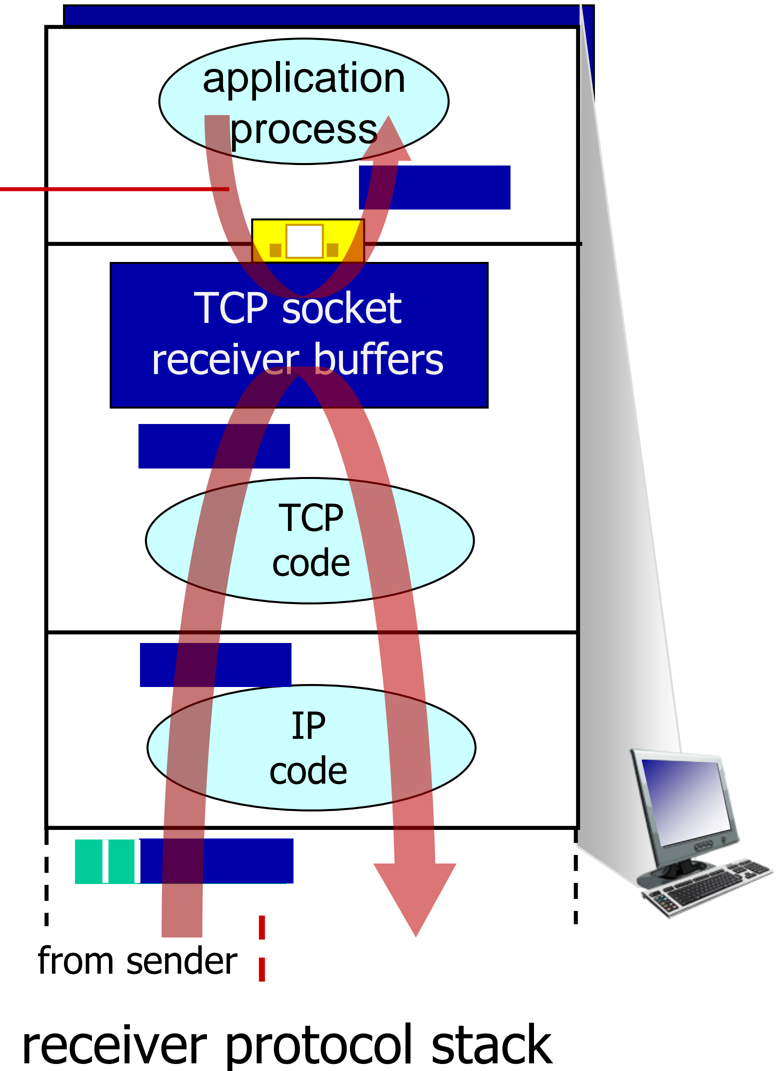
TCP FLOW CONTROL

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



flow control: # bytes receiver willing to accept

Application removing data from TCP socket buffers



receiver protocol stack

TCP FLOW CONTROL

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?

flow control

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

Application removing data from TCP socket buffers

application process

TCP socket receiver buffers

TCP code

IP code

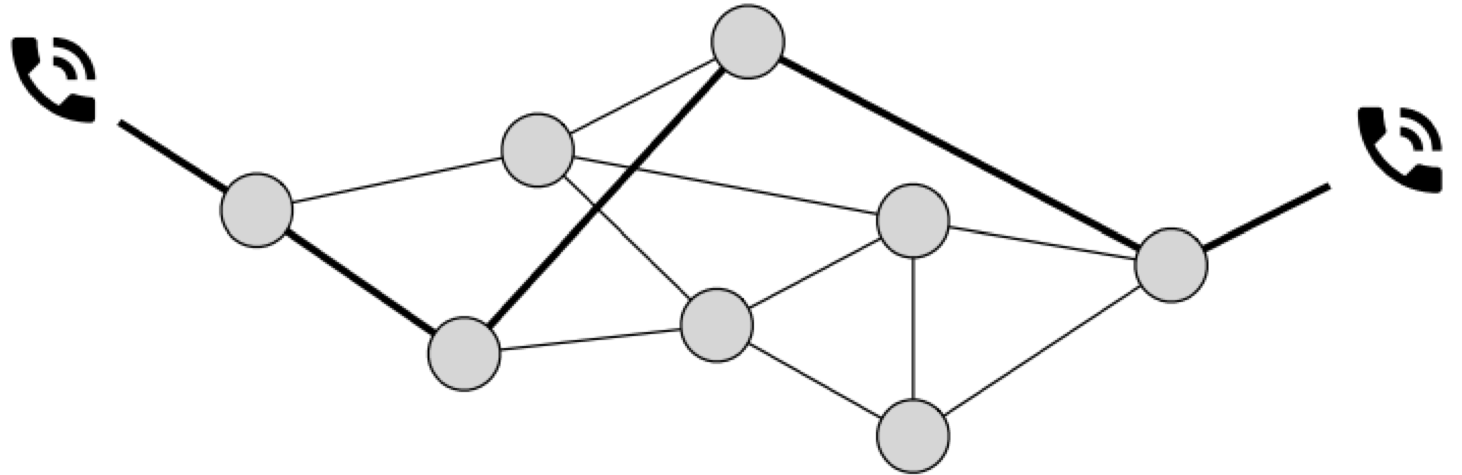
from sender

receiver protocol stack

CONGESTION

No problem under circuit switching

CIRCUIT SWITCHING



- » Source establishes connection to destination:
  - » nodes reserve resources for the connection
  - » circuit rejected if the resources are not available
  - » cannot have more than the network can handle

IP BEST-EFFORT DESIGN PHILOSOPHY

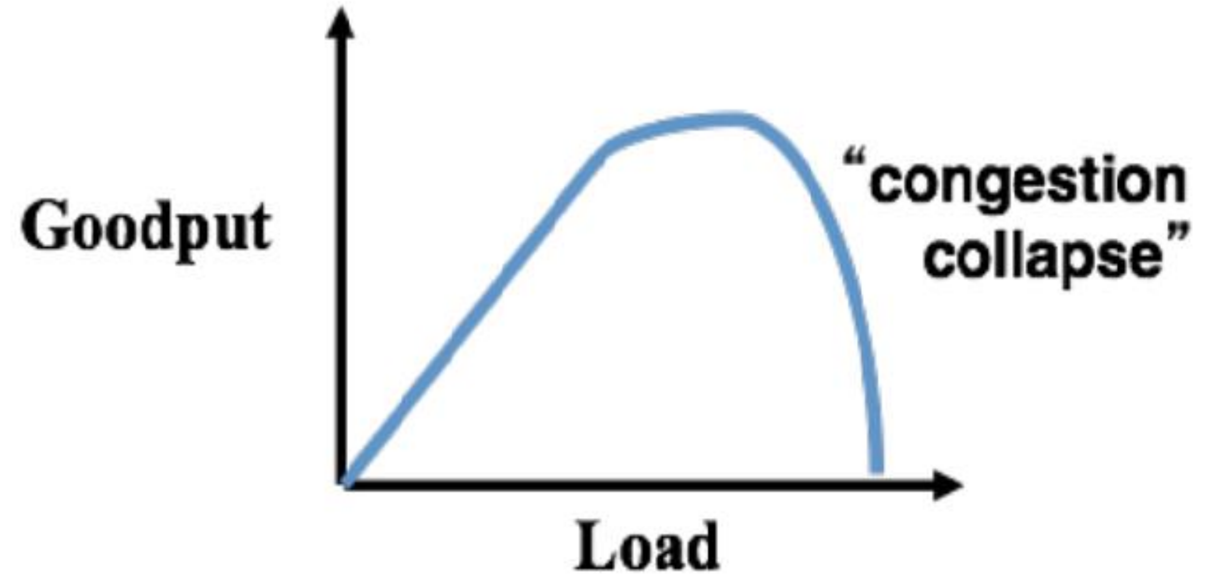
- Best-effort delivery:
  - » Let everybody send.
  - » Network tries to deliver what it can
  - » … and just drops the rest.

CONGESTION IS UNAVOIDABLE

- » Two packets arrive at same time:
  - » router can only transmit one: must buffer or drop other
- » If many packets arrive in short period of time:
  - » router cannot keep up with the arriving traffic
  - » buffer may eventually overflow

THE PROBLEM OF CONGESTION

- » What is congestion?
  - » Load is higher than capacity.
- » What do IP routers do?
  - » Drop the excess packets.
- » Why is this bad?
  - » Wasted bandwidth for retransmissions.



Increase in load that results in a *decrease* in useful work done.

WAYS TO DEAL WITH CONGESTION

- » Ignore the problem:
  - » Many dropped (and retransmitted) packets.
  - » Can cause congestion collapse.
- » Reservations, like in circuit switching:
  - » Pre-arrange bandwidth allocations.
  - » Requires negotiation before sending packets

WAYS TO DEAL WITH CONGESTION

- » Pricing:
  - » Do not drop packets for the high-bidders.
  - » Requires a payment model, and low-bidders still dropped.

» Dynamic adjustment (TCP):

- » Every sender infers the level of congestion.
- » Each adapts its sending rate "for the greater good"

READING INSTRUCTIONS

» CH. 24: ALL » CH. 25: ALL

REFERENCES

- https://www.net.t-labs.tu-berlin.de/teaching/computer_networking
- https://techterms.in/
- https://github.com/HanochShi/Supplements-ComputerNetworking-ATopDownApproach-7th-ed
- https://www.youtube.com/channel/UCJQJ4GjTiq5lmn8czf8ooOQ
- http://www.whatis.com
- http://www.webopedia.com
- Understanding Data Communications & Networks, Shay (1999)
- http://www.daemon.org/ip.html