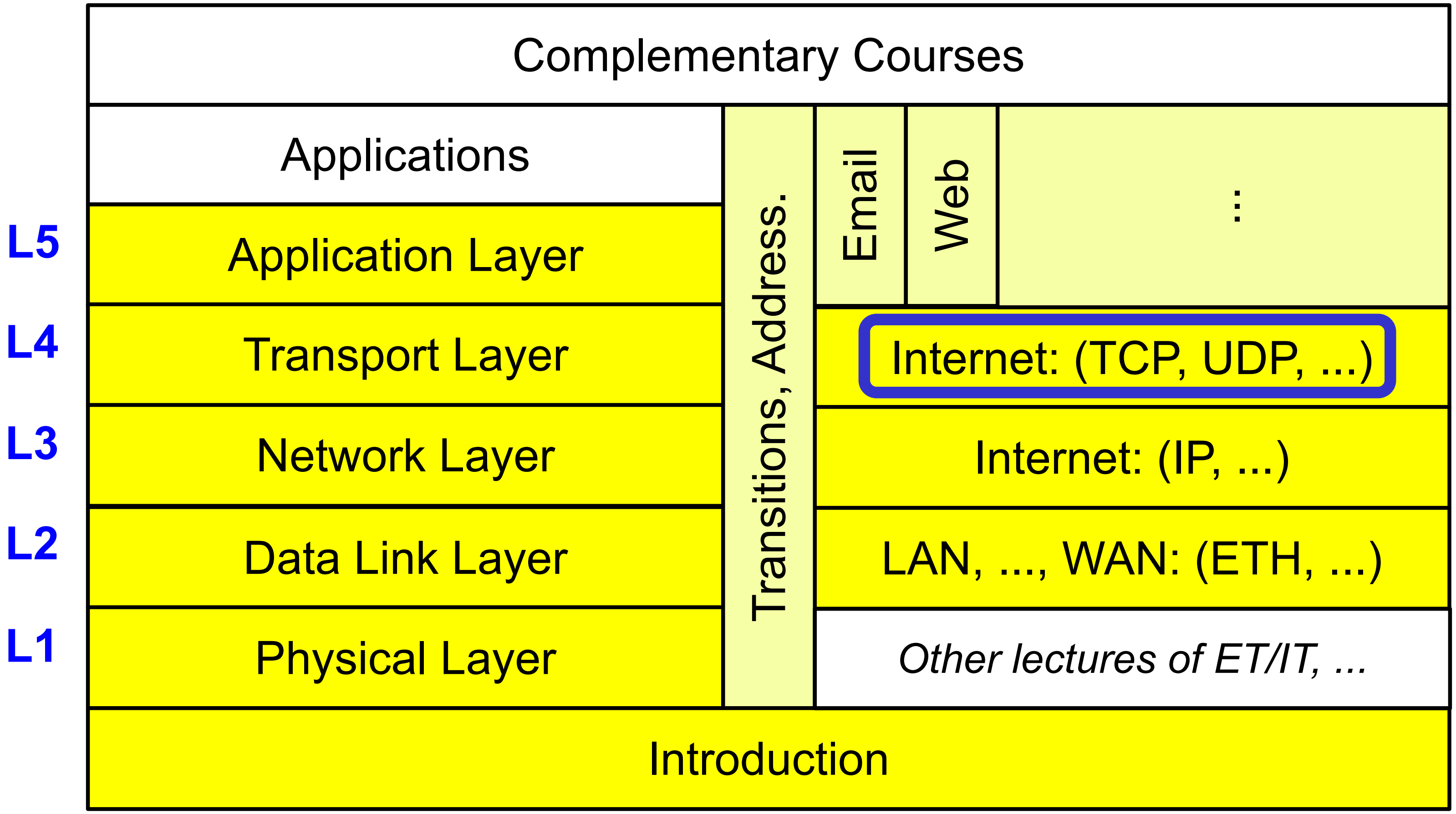
Computer Networks I

Transport Layer: Internet Protocols

Prof. Dr.-Ing. **Lars Wolf**

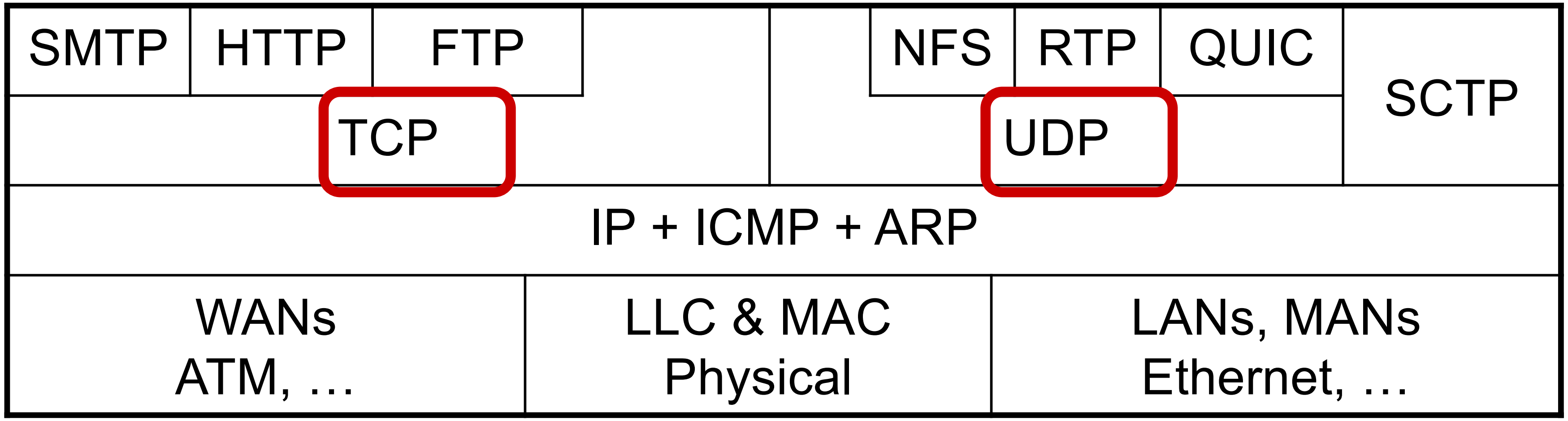
IBR, TU Braunschweig
Mühlenpfordtstr. 23, D-38106 Braunschweig, Germany,
Email: wolf@ibr.cs.tu-bs.de

Scope


Complementary Courses
Applications
Transitions, Address.
Email
Web
⋮
L5
Application Layer
L4
Transport Layer
Internet: (TCP, UDP, ...)
L3
Network Layer
Internet: (IP, ...)
L2
Data Link Layer
LAN, ..., WAN: (ETH, ...)
L1
Physical Layer
Other lectures of ET/IT, ...
Introduction

Overview

1 Some Familiar Internet Protocols



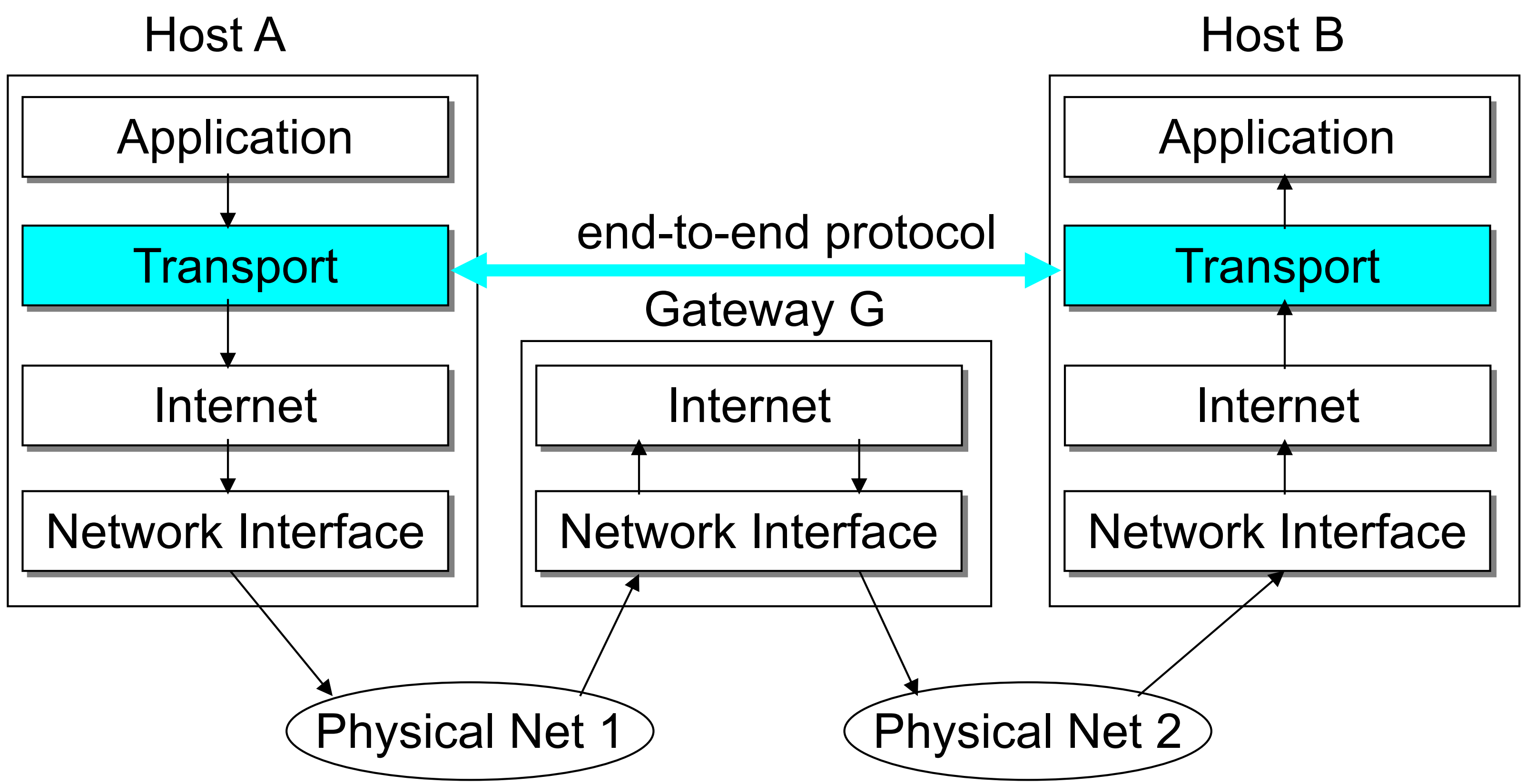
Here only a short introduction to UDP and TCP can be given.

Further (many) details about UDP, TCP, and also SCTP are given in Computer Networks II.

Internet Transport Layer (in General & Addressing)

Lowest level end-to-end protocol
- header generated by sender is interpreted only by destination
- routers / gateways view transport header as part of the payload

Adds extra functionality to the best effort packet delivery service provided by IP
- makes up for shortcomings of core network



Some Functions of Transport Protocols

Multiplexing/demultiplexing data for multiple applications
- uses “port’’ abstraction

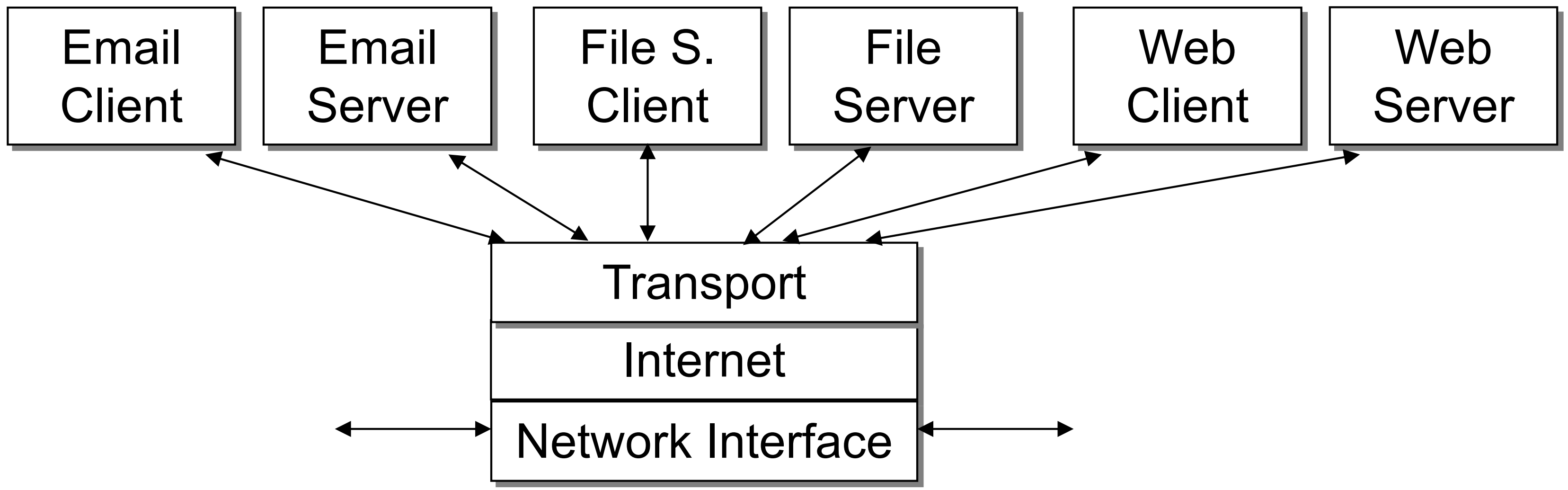
Connection establishment
- logical end-to-end connection

Error control
- hides unreliability of network layer from applications
- some types of errors:
  - corruption, loss, duplication, reordering

End-to-end flow control
- to avoid flooding the receiver

Congestion control
- to avoid flooding the network

Transport Layer: End-to-End Communications



- communication between applications required
- applications communicate
  - locally by interprocess communication
  - between systems via TRANSPORT SERVICES

Transport layer

- **interprocess end-to-end** communication via communication networks

Internet protocol IP

- enables only **endsystem-to-endsystem** communication

2 UDP – User Datagram Protocol

Specification:
- RFC 768

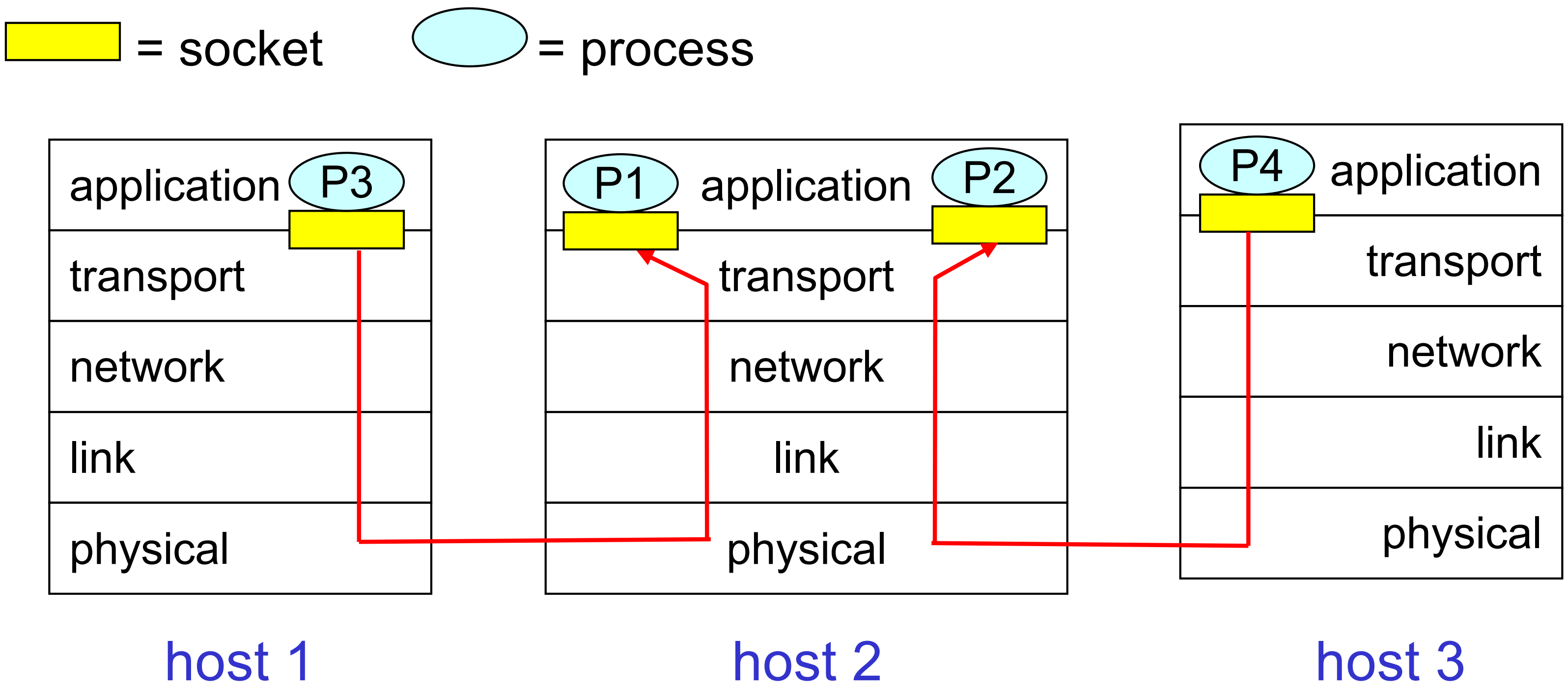
UDP is a simple transport protocol
- unreliable
- connectionless
- message-oriented

“best effort” service, UDP segments may be:
- lost
- delivered out of order to app

connectionless:
- no handshaking between UDP sender, receiver
- each UDP segment (message) handled independently of others

UDP: Characteristics

UDP is mostly IP with a short transport header
- source and destination port
- ports allow for dispatching of messages to receiver process



From *Kurose/Ross*

UDP: Characteristics

no connection establishment
- no delay for connection setup

simple
- no connection state at sender, receiver

small segment header
- less overhead

no flow control and no congestion control:
- UDP can send as fast as desired / application may transmit
  - as fast as it can/wants to and
  - as fast as the network permits

no error control or retransmission
- no guarantee about packet sequencing
- packet delivery to receiver not ensured
- possibility of duplicated packets

may be used with broadcast / multicast and streaming

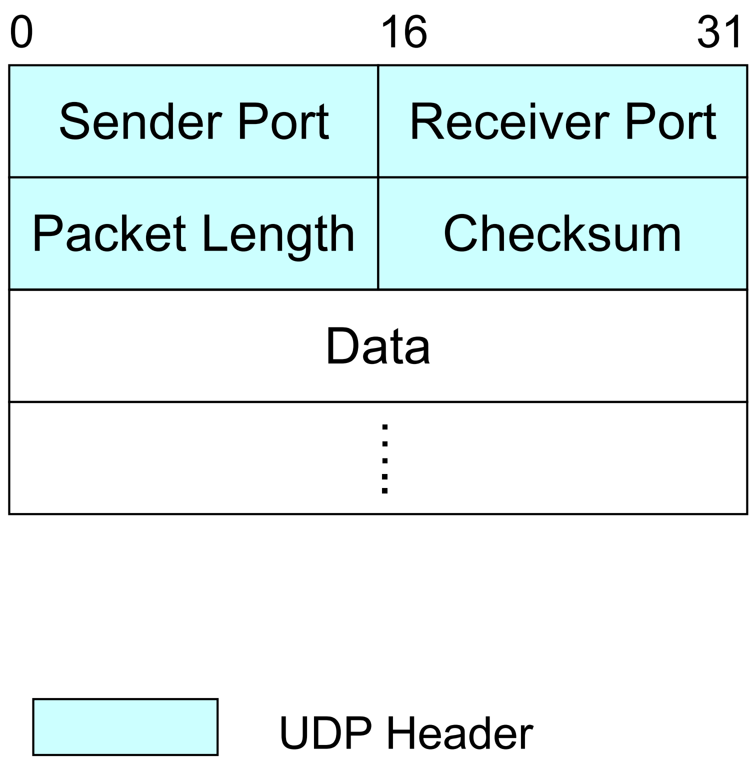
UDP: Message Format

Sender port
- 16 bit sender identification
- is optional
- use: response may be sent there

Receiver port
- receiver identification

Packet length
- in bytes (including UDP header)
- minimum: 8 (byte) i.e. header without data

Checksum
- of header and data for error detection
- use of checksum optional



3 TCP – Transmission Control Protocol

Motivation: network layer provides unreliable connectionless service

- packets and messages may be
  - duplicated, delivered in wrong order, faulty
- given such an unreliable service
  - each application would have to implement error detection and correction separately
- network or service can
  - impose packet length
  - define additional requirements to optimize data transmission
  - i.e. each application would have to be adapted separately

➔ avoid to reinvent the wheel for every application

➔TCP is the Internet transport protocol providing

- reliable end-to end byte stream over an unreliable internetwork

Specification:

- RFC 793: originally
- Enhancements, further details also in, e.g., RFC 1122, RFC 1323

3.1 TCP Features

Reliable, bidirectional, unstructured byte stream between two communicating peers
- fully ordered, fully reliable

Connections established & torn down
- three-way handshake connection setup
- disconnect per direction

Multiplexing/ demultiplexing
- ports at both ends

Error control
- checksums, sequence numbers, automatic retransmissions

→ users see correct, ordered byte sequences

End-to-end flow control
- avoid to overwhelm machine on other side
- credit mechanism (dynamic window size)

Congestion avoidance
- avoid to create traffic jam within network

3.2 TCP Protocol Basics

A key feature:

- every byte on TCP connection has its own 32 bit sequence number

Separate sequence numbers used for

- data
- acknowledgements
- window mechanism

Remember:

- 32 bit sequence number space was big in early days of the Internet
- nowadays, it can be consumed very fast

TCP Protocol Basics (2)

TCP entitites exchange data in form of **segments**

TCP segment consists of

- fixed 20 byte header (plus optional part)
- zero or more data bytes

TCP software (entity) decides on segment size to be used

- data from several writes can be combined to 1 segm.
- or data from one write can be split into several segm.

- limits
  - each segm. (incl. TCP header) must fit into IP payload
  - segment must fit into maximum transfer unit (MTU) of visited networks
    - each network may have MTU, depending on L2 technology
    - often 1500 byte (Ethernet payload size), typical upper bound on segm. size

TCP Protocol Basics (3)

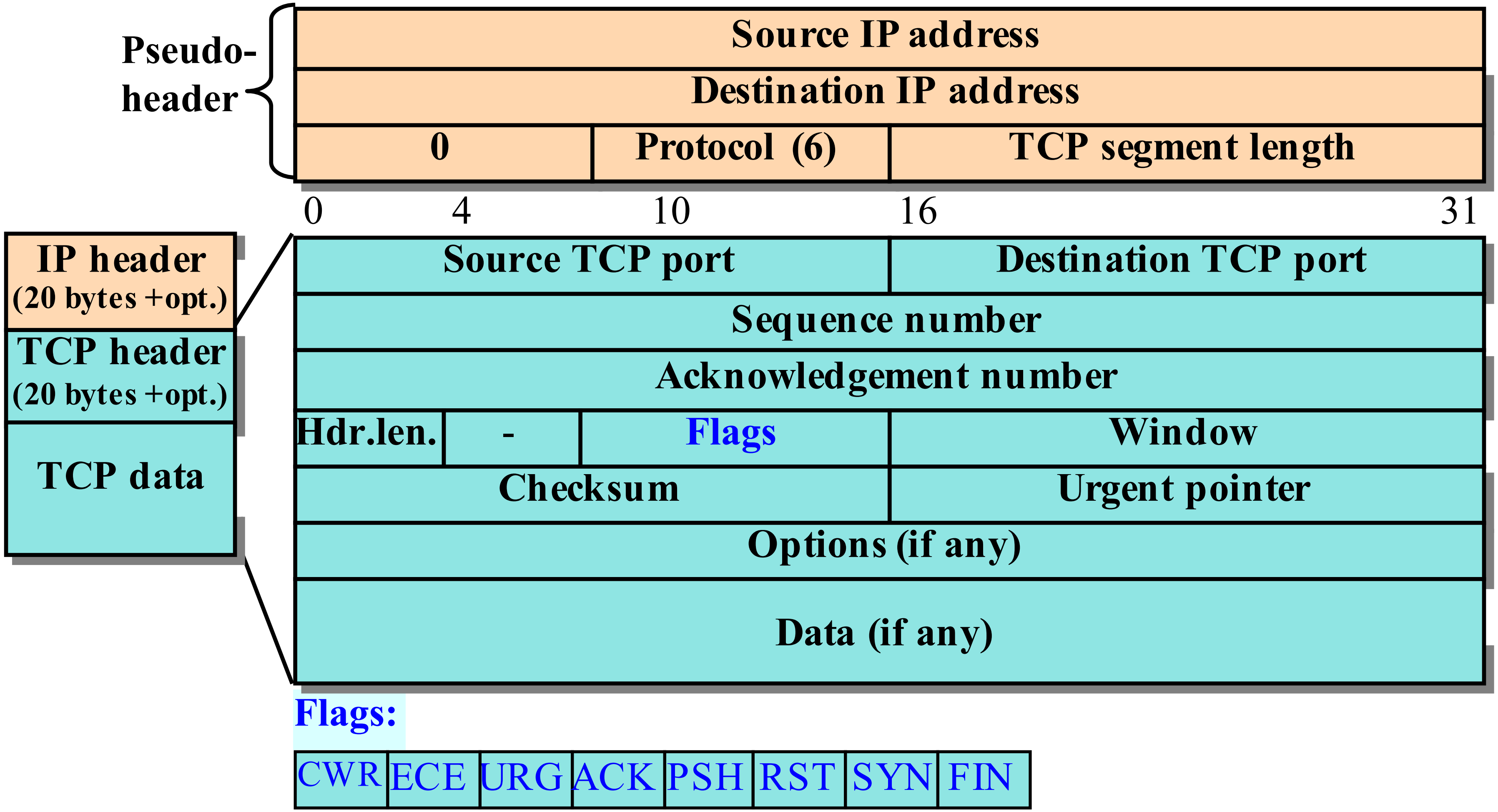
TCP uses: Sliding window protocol

- sender starts timer when it transmits segment

- receiver sends back segment to acknowledge
  - with data, if any, otherwise without data
  - acknowledgment number equal to next sequence number it expects to receive

- if sender's timer goes off before acknowledgement arrives, segment is retransmitted

3.3 TCP Segments



3.4 TCP Header

Source Port & Destination Port
- local endpoints of connection (16 bit each)

Sequence Number
- sequence number of first data byte in this segment (each byte is numbered)

Acknowledgement Number
- byte number being acknowledged, specifies next byte expected

Header length (HL) / Data Offset
- length of header including options in 32-bit words
- indicates start of data within segment (in 32-bit words)

Reserved (RESV)
- not used

0 16 32

Source Port	**Dest. Port**
Sequence Number	
Acknowledgment Number (Ack. No.)	
HL/RESV/Flags	**Window**
Checksum	**Urgent Pointer**
Options..	

TCP Header:

HL → Header length including options (4 bits)
RESV→ Reserved bits (4 bits)
Flags: (8 bits)

TCP Header (2)

Flags (bits from left to right)

- CWR:
  - Congestion Window Reduced
- ECE:
  - ECN-Echo
- URG:
  - urgent pointer is being used
- ACK:
  - Ack No. is valid (if 0 then no acknowledgement in segment)
- PSH:
  - data transferred with PUSH, receiver should give received data to application immediately
- RST:
  - Reset, connection is being reset
- SYN:
  - used to establish connection (synchronize sequence numbers)
  - SYN=1 & ACK=0: connection request
  - SYN=1 & ACK=1: connection accept
- FIN:
  - release connection

0 16 32

Source Port	**Dest. Port**
Sequence Number	
Acknowledgment Number (Ack. No.)	
HL/RESV/Flags	**Window**
Checksum	**Urgent Pointer**
Options..	

TCP Header:

HL → Header length including options (4 bits)
RESV→ Reserved bits (4 bits)
Flags: (8 bits)

TCP Header (3)

(Advertised) Window:

- buffer size in bytes as advertised by the receiver
- used for variable-sized sliding window
- window size field indicates #bytes sender may transmit
  - beginning with byte being acknowledged
- window size=0 is valid:
  - bytes up to ACK-1 received,
  - but receiver not ready for more data at the moment,
  - permission for sending more data can be notified later by
    - sending segment with same ACK and non-zero advertised window

0	16	32
Source Port	Dest. Port	
Sequence Number		
Acknowledgment Number (Ack. No.)		
HL/RESV/Flags	Window	
Checksum	Urgent Pointer	
Options..		

TCP Header (4)

Checksum

- checksum over
  - header,
  - data and
  - “pseudoheader”
- for calculation, TCP checksum field is set to 0 and data field padded with an additional zero byte (in case length of data is odd)
- uses one’s complement of sum of 16-bit halfwords in one’s complement arithmetic
- receiver’s calculation on entire segment including checksum field should result in 0

0	16	32
Source Port	Dest. Port	
Sequence Number		
Acknowledgment Number (Ack. No.)		
HL/RESV/Flags	Window	
Checksum	Urgent Pointer	
Options..		

TCP Header (5)

Pseudoheader

- is a conceptual (imaginary) header used in the checksum computation
- contains parts of the IP header
  - source IP address
  - destination IP address
  - protocol number for TCP (6)
  - length of TCP segment (including header)
- allows detection of misdelivered packets
- but violates protocol hierarchy

Pseudoheader

0–7	8–15	16–31
Source Address		
Destination Address		
00000000	Protocol	TCP segment length

TCP Header (6)

Urgent Pointer
- byte offset to the current sequence number at which important data starts

0	16	32
Source Port	Dest. Port	
Sequence Number		
Acknowledgment Number		
HL/Flags	Window	
Checksum	Urgent Pointer	
Options..		

Options
for optional facilities, not covered by regular header
- e.g. specify max. TCP payload host may accept (maximum segment size or MSS), default is 536 bytes

- window scale option: allows to use windows bigger than 64KB, shift window size field up to 14 bits to the left, thus enabling max. window 2^30 bytes

- selective repeat instead of go-back-n: NAKs to allow receiver to request a specific segment

3.5 TCP Connection Setup + Release

Connection Setup: One **passive** & one **active** side

- **server**: wait for incoming connection using LISTEN and ACCEPT
- **client**: CONNECT (specifying IP address and port, max. TCP segment size)

Three-Way-Handshake

- Connecting through 3 packets

Agree on initial sequence numbers

- initial sequence number is randomly chosen
- it is not simply 0
- reason:
  - to reduce the chance that sequence numbers of old and new connections overlap

Connection Release

Connection release for pairs of simplex connections

- each direction is **released independently** of other

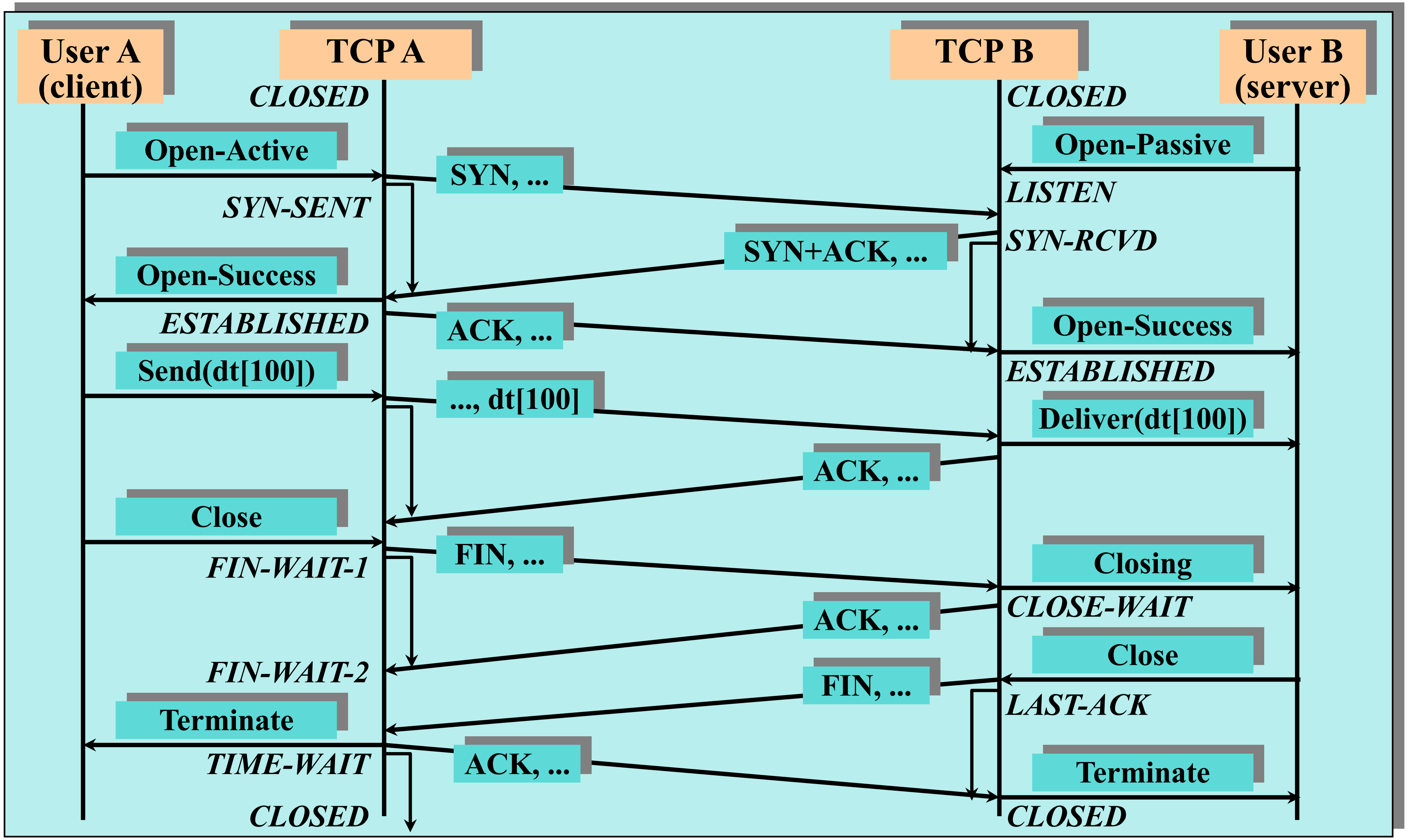
Connection release by either side sending a segment with FIN bit set

- no more data to be transmitted
- when FIN is acknowledged,
  this direction is shut down for new data

Directions are released independently:

- other direction may still be open
- full release of connection if both directions have been shut down

TCP Message Exchange



3.6 TCP Timer Management

TCP uses several timers for different purposes

Retransmission timer as most important one

- timer is set when segment is sent
- if ACK arrives before timer expires: timer is stopped
- if timer expires before ACK arrives: segment is retransmitted
    - and new timer is set

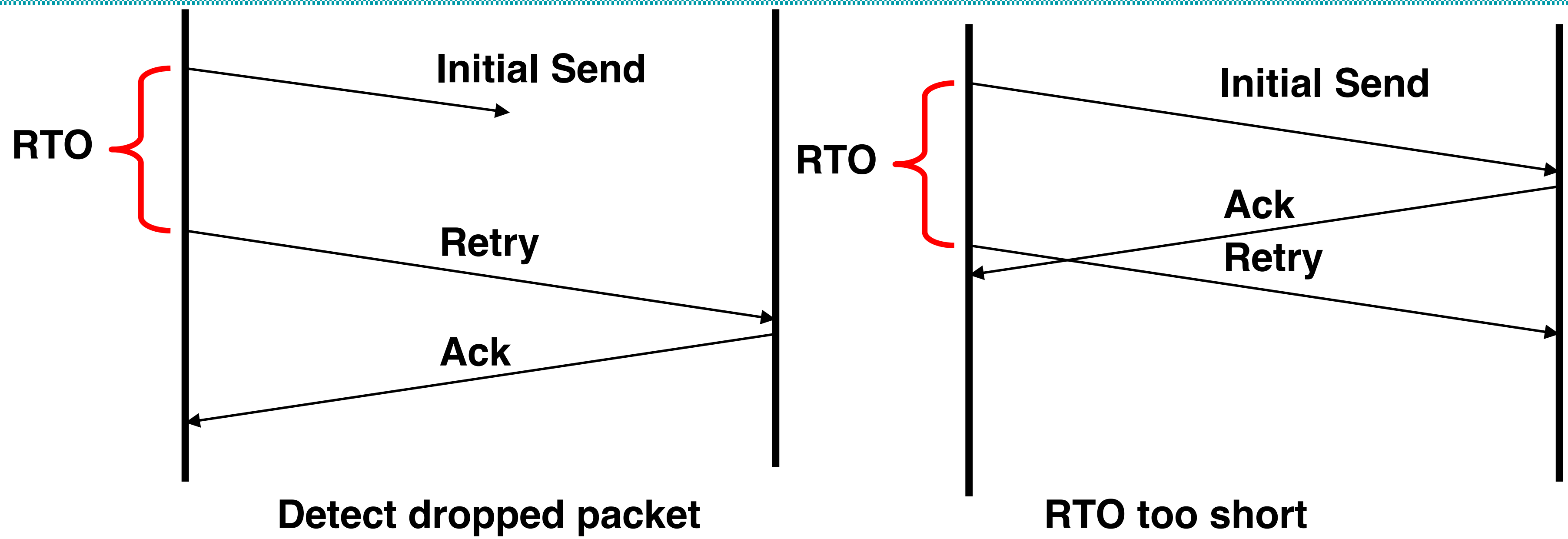
Question: How long should the timeout interval be?

TCP Must Operate Over Any Internet Path

- Retransmission time-out should be set based on round-trip delay
- But round-trip delay different for each path!

➔ Must estimate RTT dynamically

Setting Retransmission Timeout (RTO)

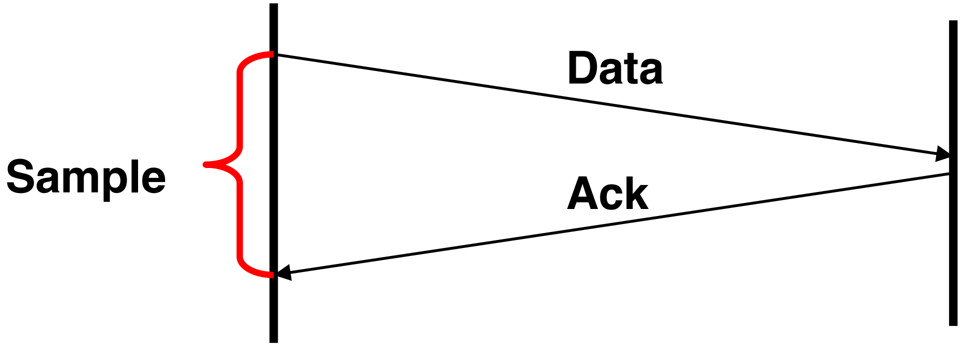


Retransmission Timeout (RTO)
- time between sending & resending segment

Challenge
- too long:
  - Add latency to communication when packets dropped
- too short:
  - Send too many duplicate packets
- general principle:
  - Must be > 1 Round Trip Time (RTT)

Round-trip Time Estimation

Every Data/Ack pair gives new RTT estimate



Can get lots of short-term fluctuations

TCP Round-trip Estimator

maintain RTT variable per connection
- best current estimate of round-trip time to destination

start timer when sending a segment
- to determine duration until acknowledgement arrives
- and to trigger retransmission if needed

calculate new RTT estimate when acknowledgment is received before timer expires

Round trip times estimated as a moving average:
- new_RTT = α*(old_RTT) + (1 – α)*(new_sample)

Smoothing factor α (0 <= α <= 1 ; determines importance of history
- 0: only current/last value is relevant
- 7/8 = 0.875 for most TCP’s
- 1: only old values are relevant

TCP Round-trip Estimator (2)

Originally: Retransmit timer set to

- Timeout = β new_RTT,
  - where $\beta = 2$
  - want to be somewhat conservative about retransmitting

Problem

- static β not able to adapt to high variation in observed RTTs during high load conditions

Solution

- estimate both RTT and deviation in RTT
- maintain another smoothed variable D

  D = α * D + (1 – α) * |RTT - new_sample|

  α may be same or different value used to smooth RTT

- set timeout interval to Timeout = RTT + 4 * D

3.7 TCP Congestion Control

Objective:

- avoid that too much traffic exists which could collapse network

Basic idea:

- detect congestion, typically using packet loss as indicator
- on detection: adjust sending rate

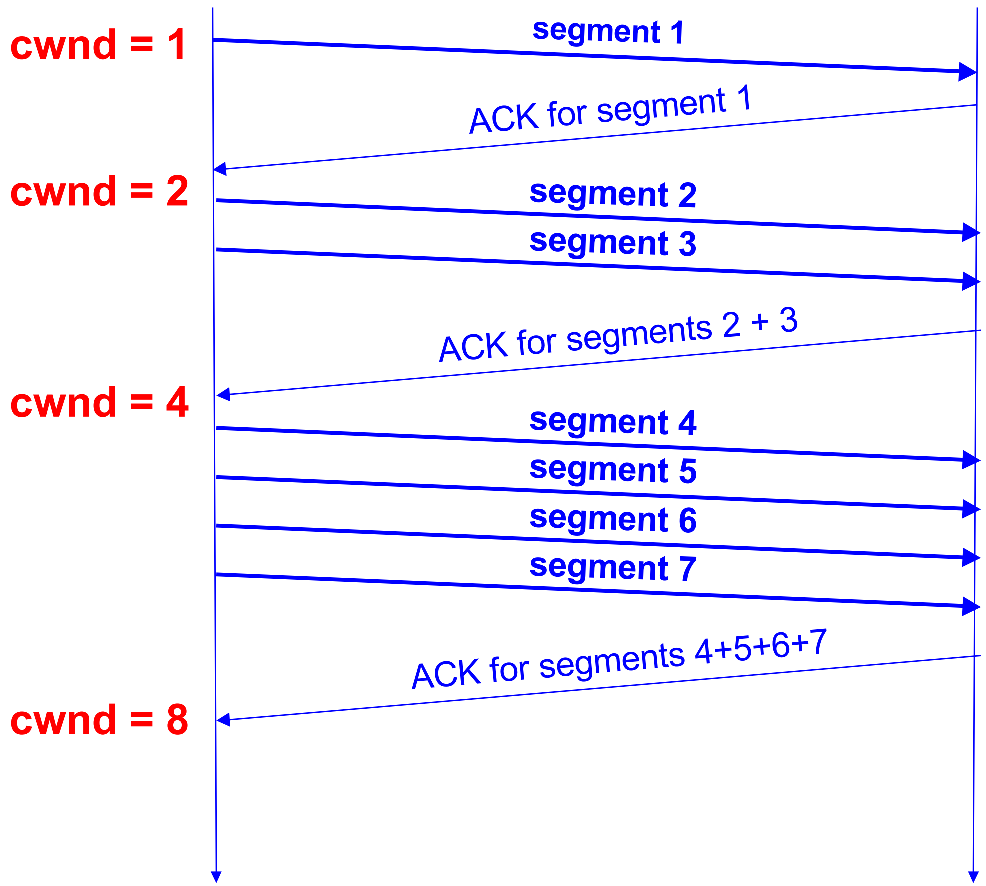
Adjustment of sending rate:

- use TCP window to control number of unacknowledged packets
- sending rate: ~Window/RTT → vary window size
- new parameter “congestion window” (cwnd) at sender
- upon receipt of ACK (of new data) → increase rate
  - data successfully delivered, perhaps can send faster
- upon detection of loss → decrease rate

TCP Congestion Control: Phases

Phase 1: Slow start

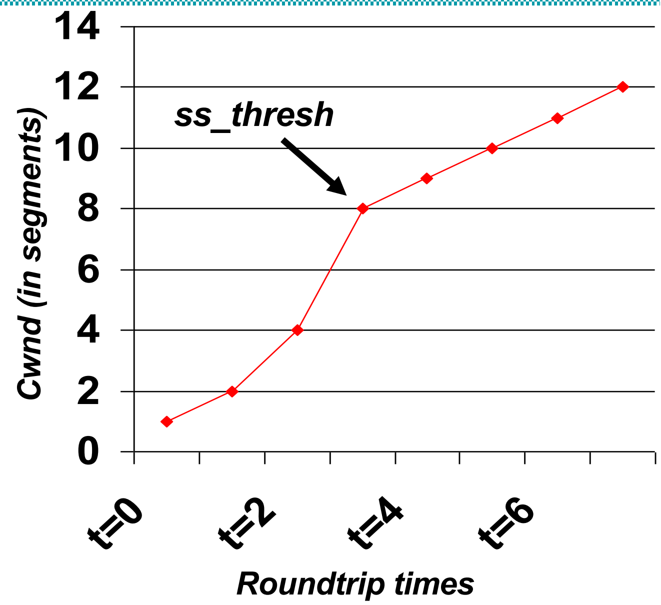
- discover roughly the proper sending rate quickly
- initialize cwnd =1
- each time a segment is ACKed
  → increment cwnd by 1 (cwnd++)
- until ss_thresh is reached
  or packet loss occurs



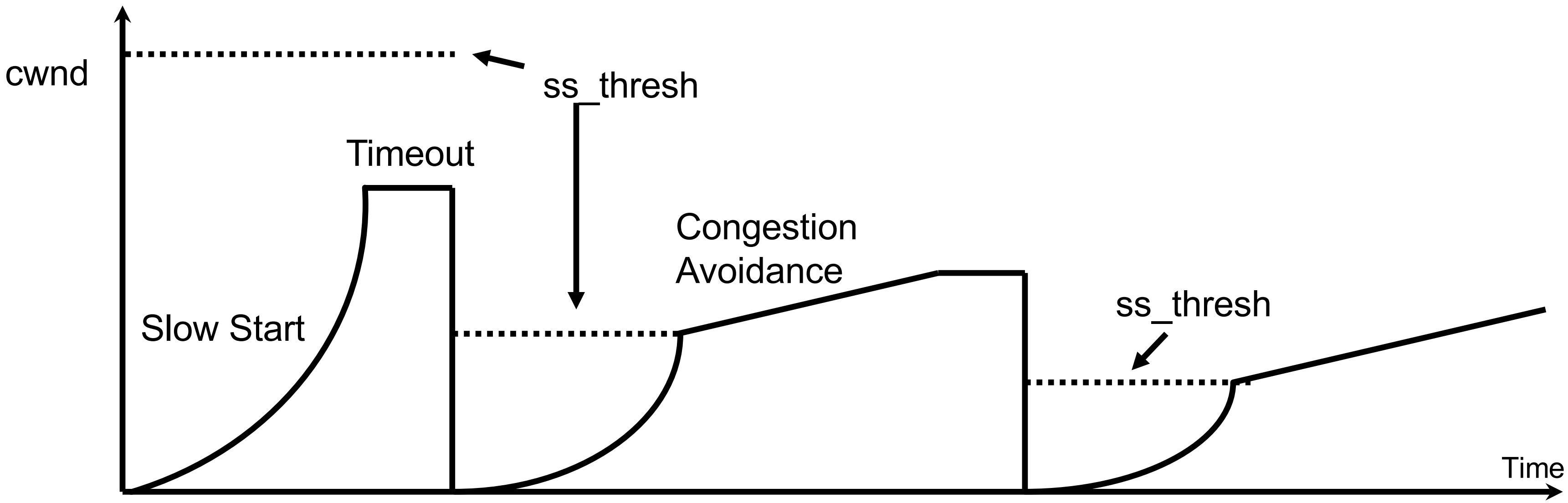
TCP Congestion Control: Phases

Phase 2: Congestion avoidance

- additive increase
  - gradually probe for additional bandwidth
  - if cwnd > ss_thresh then
    - each time a segment is acknowledged
    - increment cwnd by 1/cwnd



- multiplicative decrease
  - decreasing cwnd upon loss/timeout
  - ss_thresh = cwnd / 2 ; cwnd = 1 ; go into slow-start



TCP Variants - Responses to Congestion

Many alg. have been developed to respond to congestion:

- TCP Tahoe
  - the basic algorithm (discussed previously) + fast retransmit
- TCP Reno = Tahoe + fast retransmit & fast recovery
- TCP New Reno
  - Has been used widely for for many years

- TCP BIC
- TCP CUBIC

- TCP Vegas
  - window size adjusted according to timing of ACKs (difference between expected and actual RTT)
- TCP SACK
  - selective ACK
  - both sides, i.e., sender and receiver must support TCP SACK
  - state machine more complex