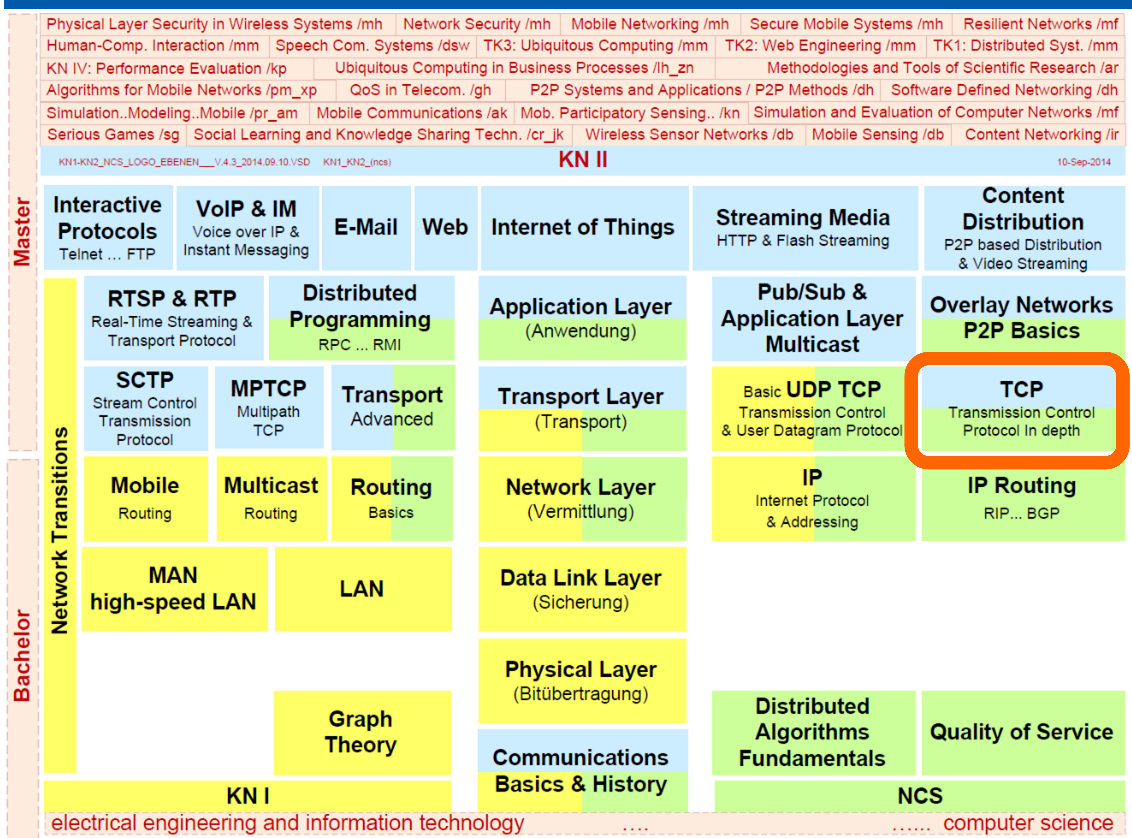
Communication Networks II

Transmission Control Protocol - TCP

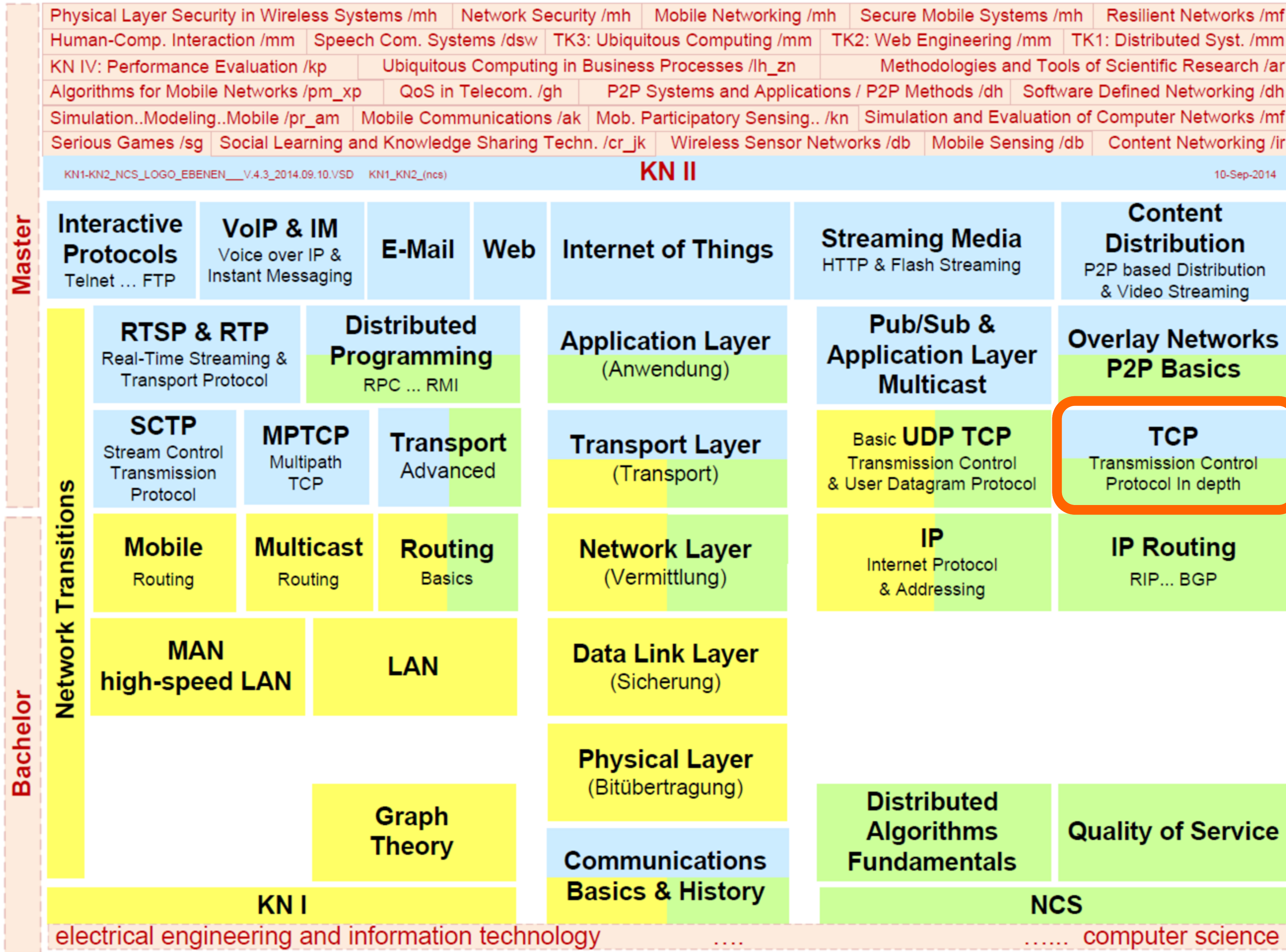
TECHNISCHE UNIVERSITÄT DARMSTADT



10. September 2014

Prof. Dr.-Ing. **Ralf Steinmetz**
KOM - Multimedia Communications Lab

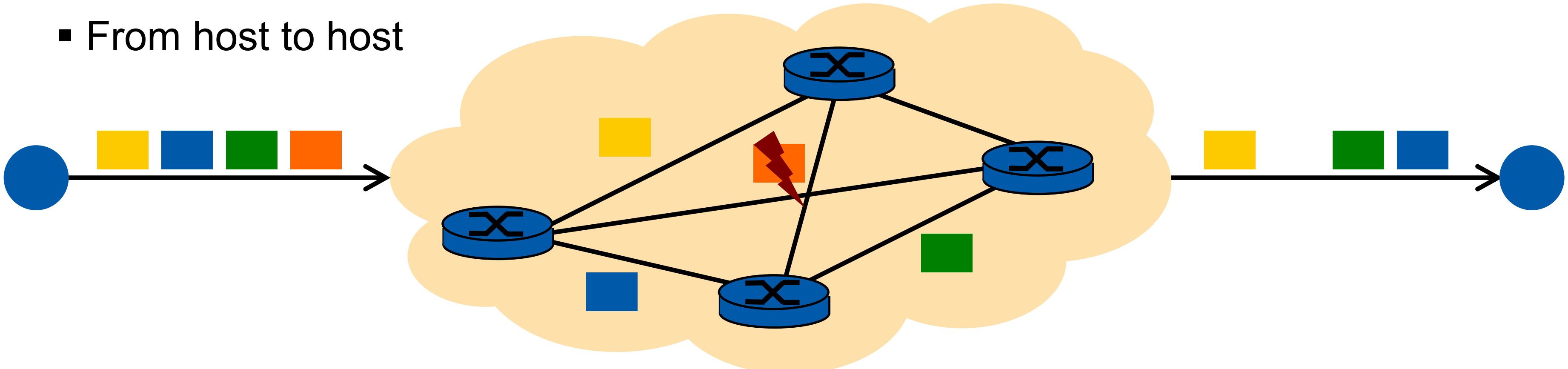
Overview



10. September 2014

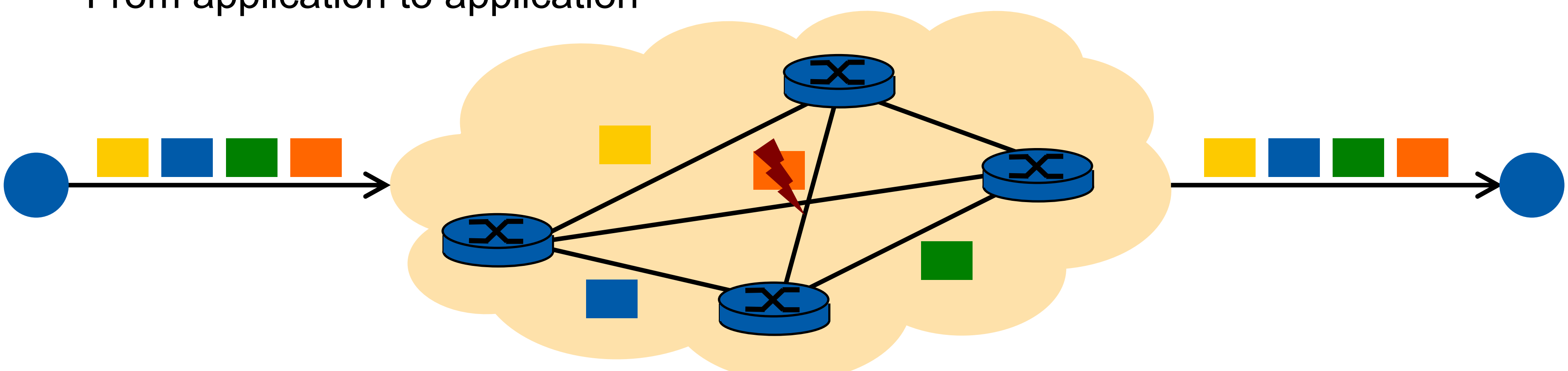
Internet layer offers best effort packet delivery

- From host to host



TCP offers reliable byte stream

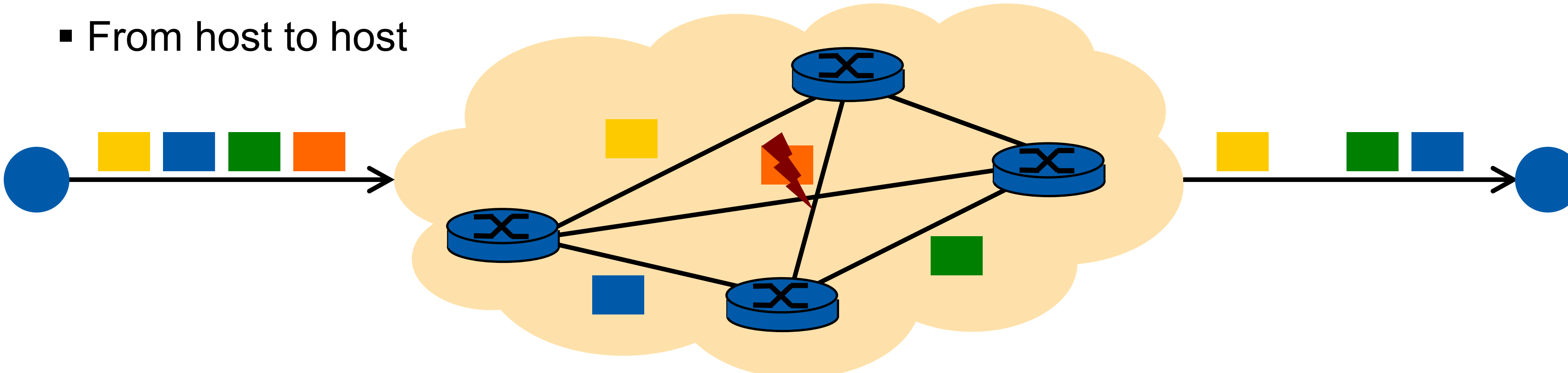
- From application to application



UDP – User Datagram Protocol (vs. TCP)

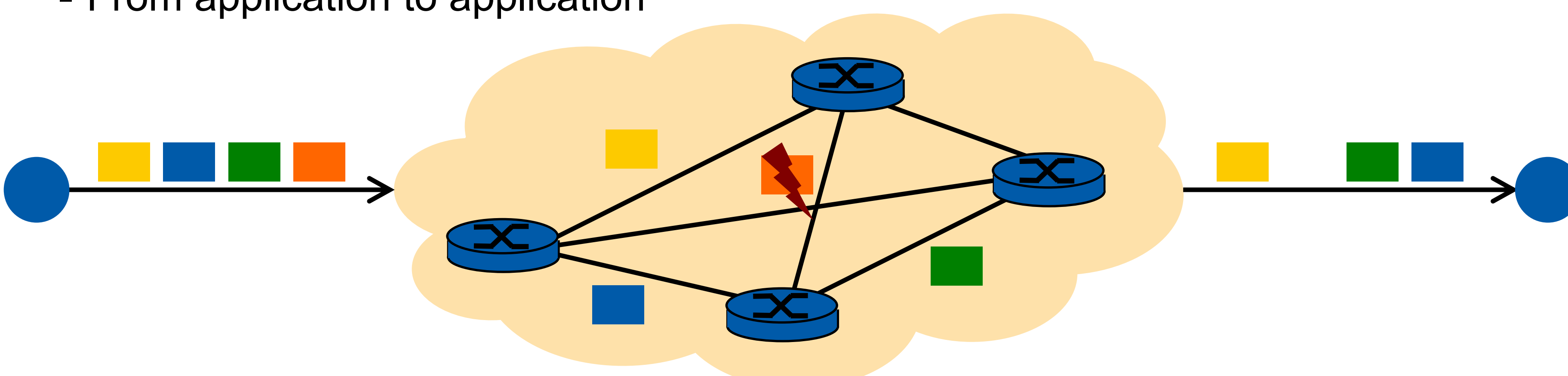
Internet layer offers best effort packet delivery

- From host to host



UDP offers best effort message delivery

- From application to application



TCP – Transmission Control Protocol - Basics

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
- ➔ do not 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 - Transmission Control Protocol: originally
- RFC 1122 and RFC 1323: errors corrected, enhancements implemented

TCP in Use & Application Areas

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Each machine supporting TCP has a TCP transport entity composed of

- library procedure
- user process
- part of kernel

TCP transport entity manages

- TCP streams
- interfaces to IP layer

TCP transport entity at sending side

- accepts user data streams from local processes
- splits them into pieces <= 64 KB
  - typically 1460 bytes (to fit into single Ethernet frame with IP and TCP headers)
- sends each piece as separate IP datagram

TCP transport entity at receiving side

- gets TCP data from datagram received at host
- reconstructs original byte streams

TCP in Use & Application Areas

Two-way communications (fully duplex)

- data may be transmitted simultaneously in both directions over a TCP connection

Point-to-point

- each connection has exactly two endpoints

TCP must ensure reliability

- IP layer doesn't guarantee that datagram will be delivered properly / in order
  - TCP must handle this, e.g. timeout and retransmit / reorder

  → i.e. reliable

- fully ordered, fully reliable
  - sequence maintained
  - no data loss, no duplicates, no modified data

TCP in Use & Application Areas

Benefits of TCP

- reliable data transmission
- efficient data transmission despite complexity
  - (up to 8 Mbps on 10 Mbps Ethernet)
- can be used with LAN and WAN for
  - low data rates
    (e.g. interactive terminal)
  - high data rates
    (e.g. file transfer)

Disadvantages when compared with UDP

- higher resource requirements
  - buffering, status information, timer usage
  - connection set-up and disconnect necessary
    - even in case of short data transmissions

Applications

- file transfer (FTP)
- interactive terminal (Telnet)
- e-mail (SMTP)
- X-Windows

Some Missing Characteristics

no broadcast

- no possibility to address all applications at the same time with a single message

no multicasting

- group addressing not possible

no QoS parameters

- not suited for different media characteristics

no real-time support

- no correct treatment/communications of audio or video possible
- e.g. no Forward Error Correction (FEC)

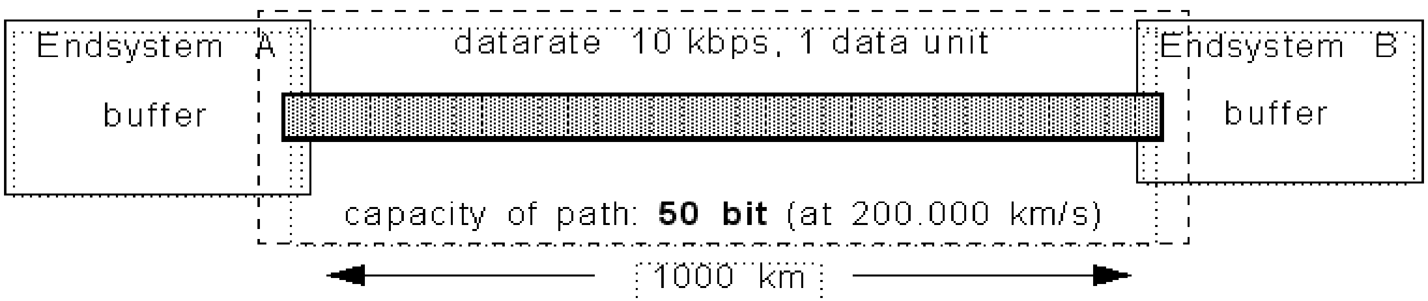
Motivation

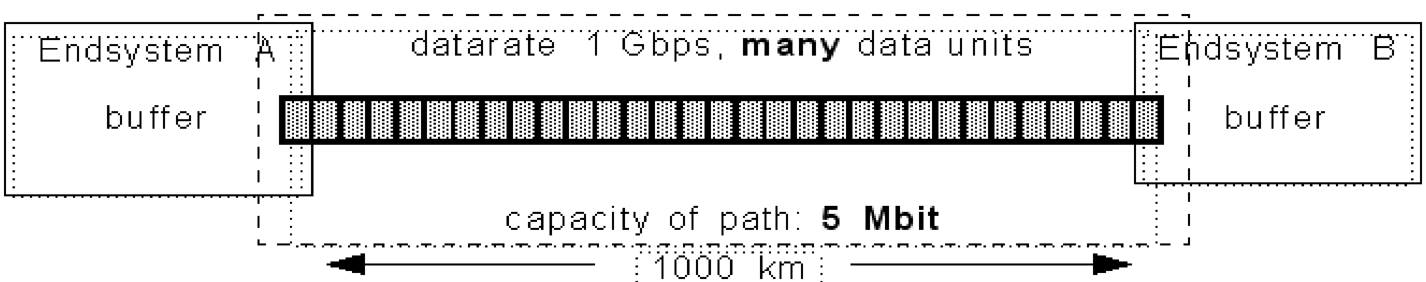
- networks and applications have changed

Networks

- higher data rates
- also farther distances (e.g. also via satellite)
- networks

$$\text{Data amount} = \frac{\text{Data rate} \times \text{Distance}}{\text{Velocity of Propagation}}$$





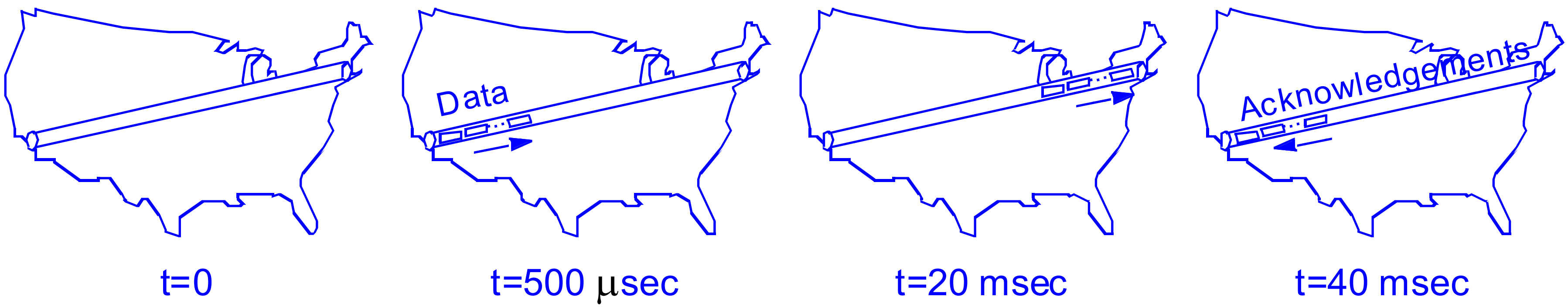
Further Development of Transport Protocols

Bandwidth-Delay Product increases

- bandwidth [bits/sec] * round-trip delay [sec]

Useful parameter for network performance analysis

- Capacity of pipe from sender to receiver and back (in bits)



Example:

- Transmission from San Diego to Boston
  - sending 64 KB burst (receiver buffer 64 KB), link: 1 Gbps
  - one-way propagation delay (speed-of-light in fiber): 20 msec
- Bandwidth-delay product: 40 million bit
- i.e.: sender would have to transmit burst of 40 million bits to keep pipe busy till ACK

Receiver window must be >= bandwidth-delay product

- for good performance

2 TCP Message Format

Reliable bidirectional in-order byte stream

- Socket: SOCK_STREAM

Connections established & torn down

Multiplexing/ demultiplexing

- ports at both ends

Error control

- users see correct, ordered byte sequences

End-to-end flow control

- avoid overwhelming the machines at either end

Congestion avoidance

- avoid creating traffic jams within network

TCP Header:

0	16 … 31
Source Port	Dest. Port
Sequence Number	
Acknowledgment Number (Ack. No.)	
HL/RESV/Flags	Advertised Win.
Checksum	Urgent Pointer
Options..	

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TCP Header Format

TCP Message Format

Variable length header

- Min. 20 byte
- Variable length options
- Multiple of 4 byte

Source port

- 16 bit identifier of sending application

Destination port

- 16 bit identifier of receiving application

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                      TCP Header Format
```

Image source: http://www.rfc-editor.org/rfc/rfc793.txt

Message Format - Ports

Port numbers identify sending/receiving application/process

- In analogy to UDP ports
  - Ports 0 – 40151 assigned by Internet Assigned Numbers Authority (IANA)

- System ports (or: well known ports), 0 – 1023
  - E.g. ports 20 and 21: File Transfer Protocol (FTP) data and control
  - E.g. port 22: Secure Shell (SSH)
  - E.g. port 25: Simple Mail Transfer Protocol (SMTP)
  - E.g. port 80: Hypertext Transfer Protocol (HTTP)

- User ports (or: registered ports), 1024 – 40151
  - E.g. port 1194: OpenVPN
  - E.g. port 3689: Digital Audio Access Protocol (DAAP)
  - E.g. port 17500: Dropbox LANsync data
    - Compare UDP port 17500 Dropbox LANsync discovery → conflict?

- Dynamic ports (or: private/ephemeral ports)
  - 40152 – 65535 for dynamic use

Message Format

Sequence number

- Connection setup:
  Initial sequence number negotiation
- Data transfer:
  Number of first byte in data field

Acknowledgment number

- Connection setup:
  Initial sequence number negotiation
- Data transfer:
  Next expected byte
  - Cumulative acknowledgment

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                        TCP Header Format
```

Image source: http://www.rfc-editor.org/rfc/rfc793.txt

Message Format

Data offset

- Header length in words (4 byte / 32 bit)
- Indicates beginning of data field
- Remember: variable length options
- Also (later RFCs) called header length

Reserved

- Originally 6 bits
  - RFC 793
- Now 3 bits
  - After RFCs 3168 and 3540
  - .. see next slide
- Must be zero

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TCP Header Format

Image source: http://www.rfc-editor.org/rfc/rfc793.txt

TECHNISCHE UNIVERSITÄT DARMSTADT

Further Flags

- NS: nonce sum
  - Used for Explicit Congestion Notification (ECN) congestion control
  - Added June 2003 in RFC 3540
- CWR: congestion window reduced
  - Used for ECN
  - Added September 2001 in RFC 3168
- ECE: ECN Echo
  - Used for ECN
  - Added September 2001 in RFC 3168

Header Length	Reserved	NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN

Image source: http://www.rfc-editor.org/rfc/rfc793.txt / rfc3540.txt

Message Format

Original flags

- URG: data field contains urgent data
  - Application layer to be notified immediately
  - Not used in practice
- ACK: message contains acknowledgment
  - Acknowledgment number significant
- PSH: push function
  - Data to be delivered to application layer immediately

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                        TCP Header Format
```

Image source: http://www.rfc-editor.org/rfc/rfc793.txt

Message Format

Original flags

Used for connection management

- RST: reset connection
- SYN: synchronize sequence numbers
- FIN: data transfer finished

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                       TCP Header Format
```

Image source: http://www.rfc-editor.org/rfc/rfc793.txt

Message Format

Window

- Number of bytes sender of message can accept
- Indicates available buffer space of sender
- Used for flow control

Checksum

- Used for error detection
- Same recipe as UDP

Urgent pointer

- Number of urgent bytes in data field
- Not used in practice

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                    TCP Header Format
```

Image source: http://www.rfc-editor.org/rfc/rfc793.txt

Message Format

Options

- Definition of protocol options
- Valid for connection

- E.g.
  - Definition of max. segment size
    - TCP message is called segment
  - Selective acknowledgment
    - Useful in certain cases of packet loss
  - Window scaling
    - Original window size limited to 2^{16}= 2 EE16 = 65535 bytes
    - Scaling factor used to optimize TCP for high bandwidth connections

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                      TCP Header Format
```

Image source: http://www.rfc-editor.org/rfc/rfc793.txt

Message Format

Padding

- Zeros to align options to 4 byte boundary

Data

- Application data
  - Application header
  - User data

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                    TCP Header Format
```

Image source: http://www.rfc-editor.org/rfc/rfc793.txt

Segments

Challenge:

- Buffered data transmission
  - byte stream – is not a message stream

➔Segments are introduced

Transport layer

- reassembles segments

Fragments

Challenge:

- size of packets of underlying network are smaller than size of IP packet

➔Fragments are introduced

Network layer

- reassembles fragments at final destination

Challenge:
size of packets of underlying network are smaller than size of IP packet

- ➔Fragments are introduced

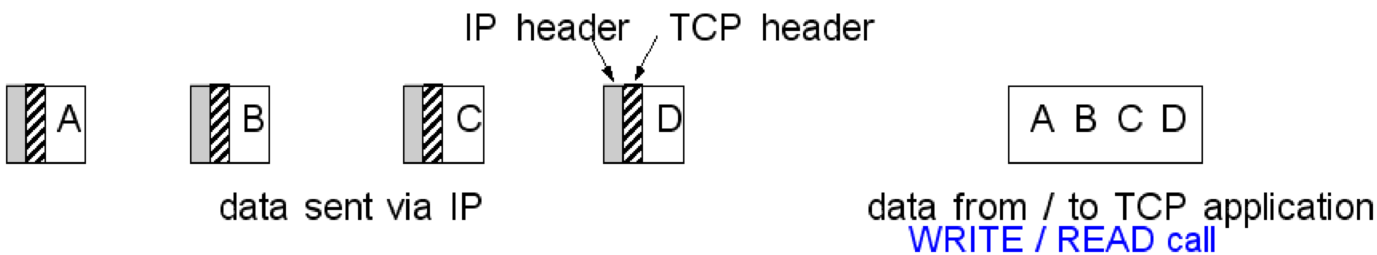
Fragments

- IP packets are split (if necessary) into FRAGMENTS
  - in order to adapt them to underlying networks
- max. IP packet size is limited by MTU (maximum transfer unit) of underlying network,
  - e.g. for Ethernet MTU=1500 byte

IP layer

- reassembles fragments at final destination

Segments



Challenge: Buffered data transmission

- byte stream – is not a message stream:
  - message boundaries are not preserved
  - no way for receiver to detect the unit(s) in which data were written

➔Segments are introduced

Segments

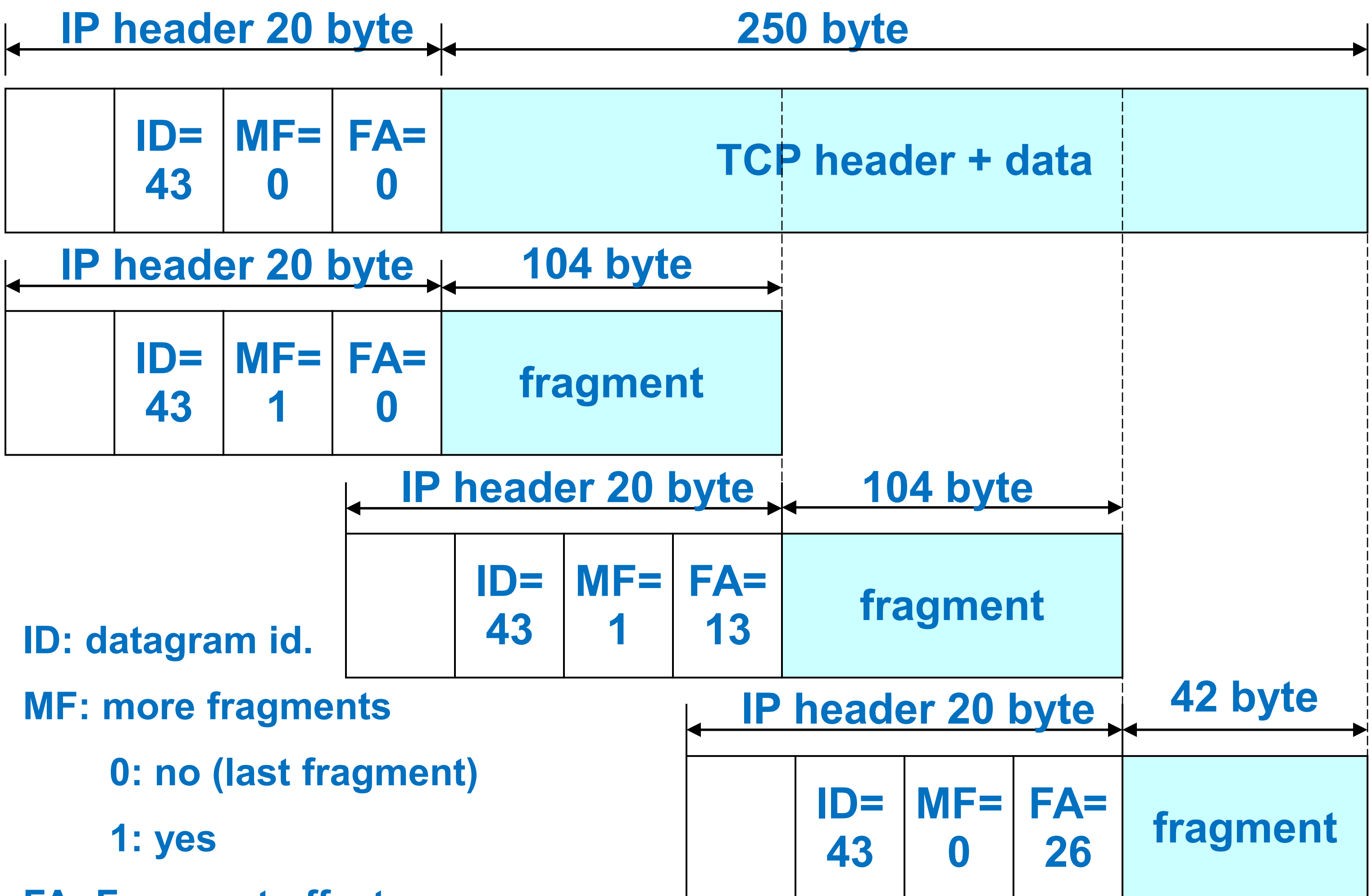
- TCP DATA STREAM is split into segments
  - SEGMENTS sent as payload of IP packets
  - max. segment size (mss) limits the size of a segment
  - mss is negotiated at connection setup
    - using TCP options (as discussed previously)

Transport layer

- reassembles segments

TECHNISCHE UNIVERSITÄT DARMSTADT

Fragmentation of segments e.g. case MTU = 128 Bytes



ID: datagram id.

MF: more fragments

- 0: no (last fragment)
- 1: yes

FA: Fragment offset

- n → n*8 byte

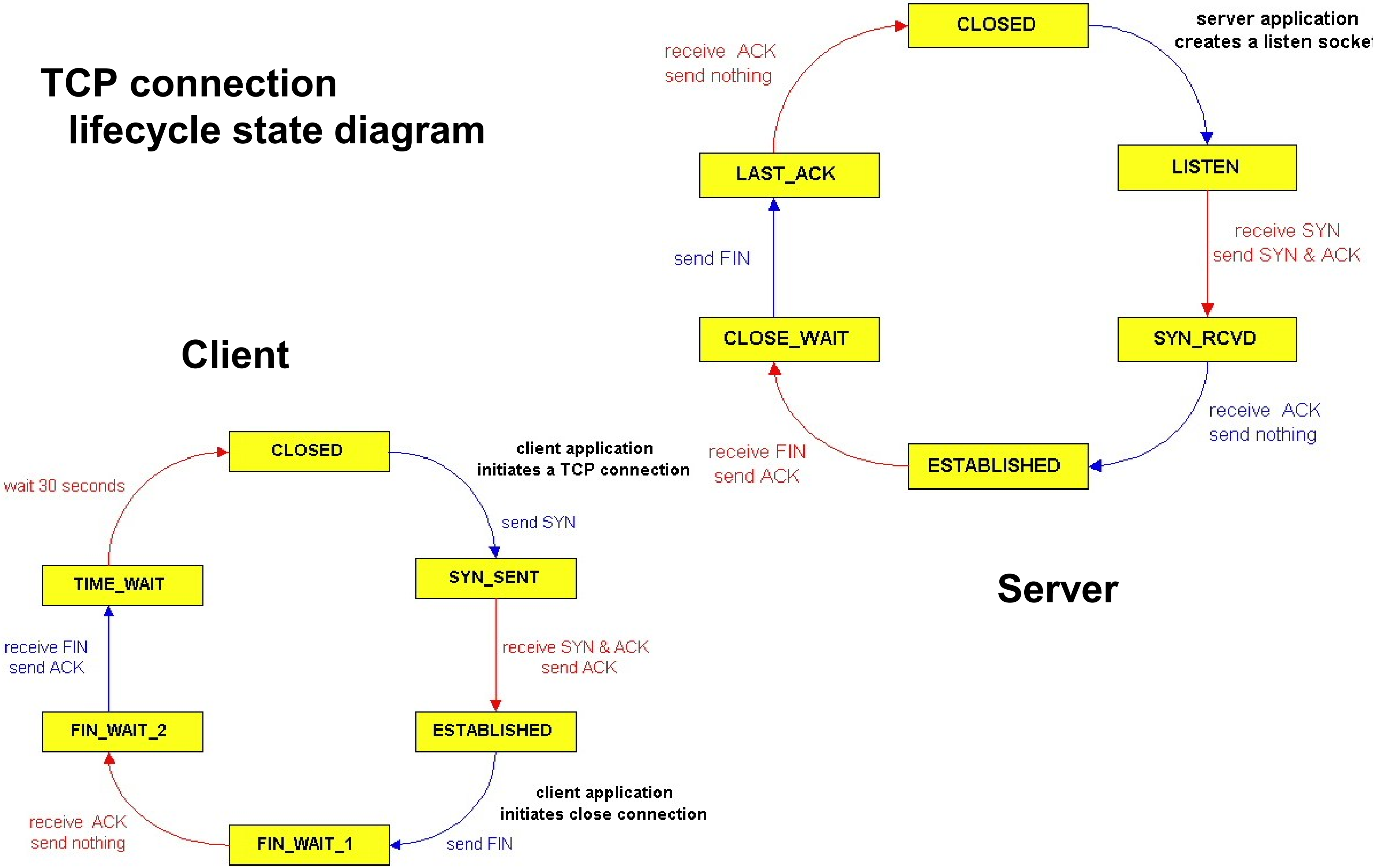
Why is there a need for connection setup?

Mainly to agree on starting sequence numbers

- starting sequence number is randomly chosen
- reason:
  - to reduce the chance that sequence numbers of old and new connections overlap

4.1 Connection Management

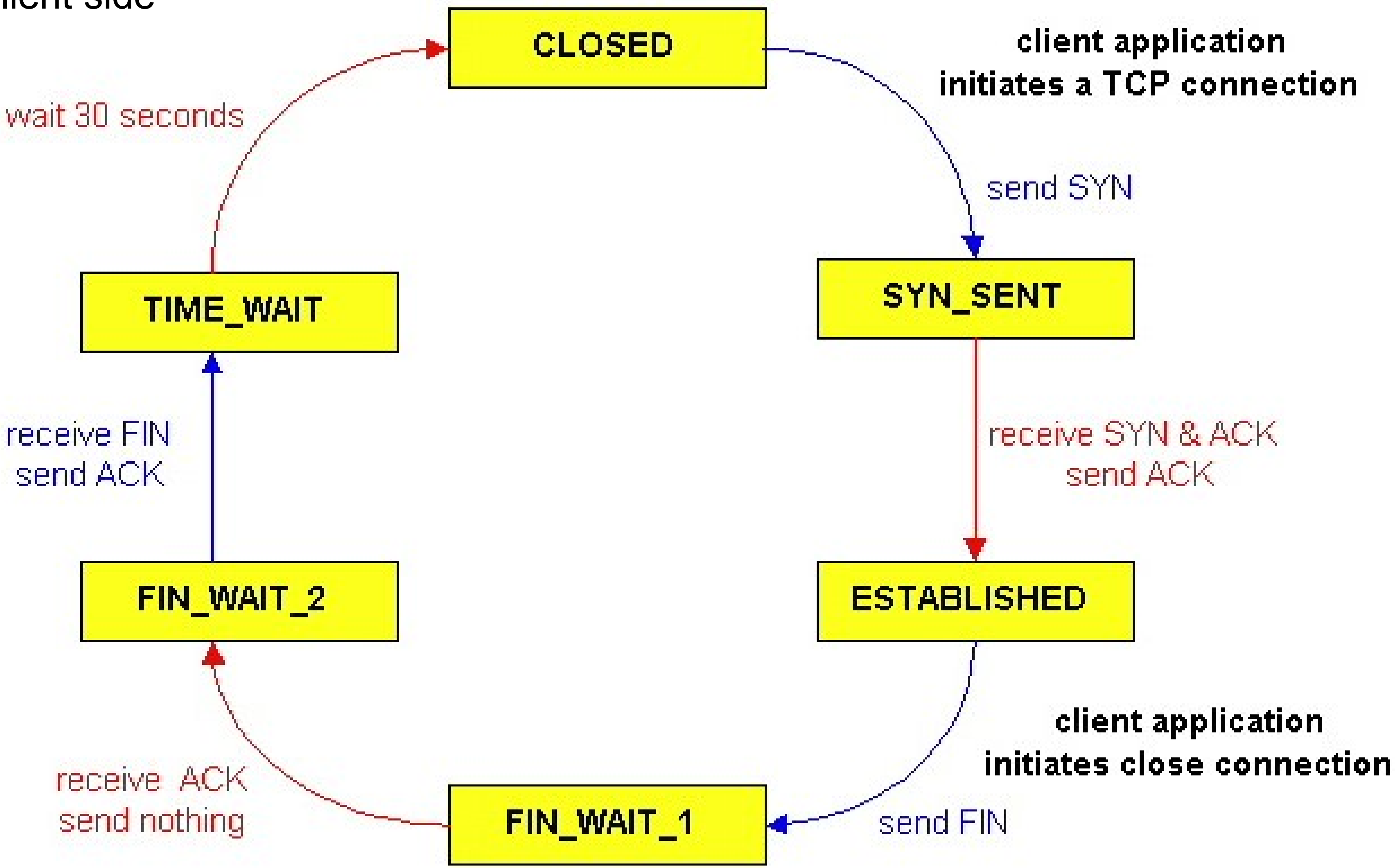
TCP connection lifecycle state diagram



TECHNISCHE UNIVERSITÄT DARMSTADT

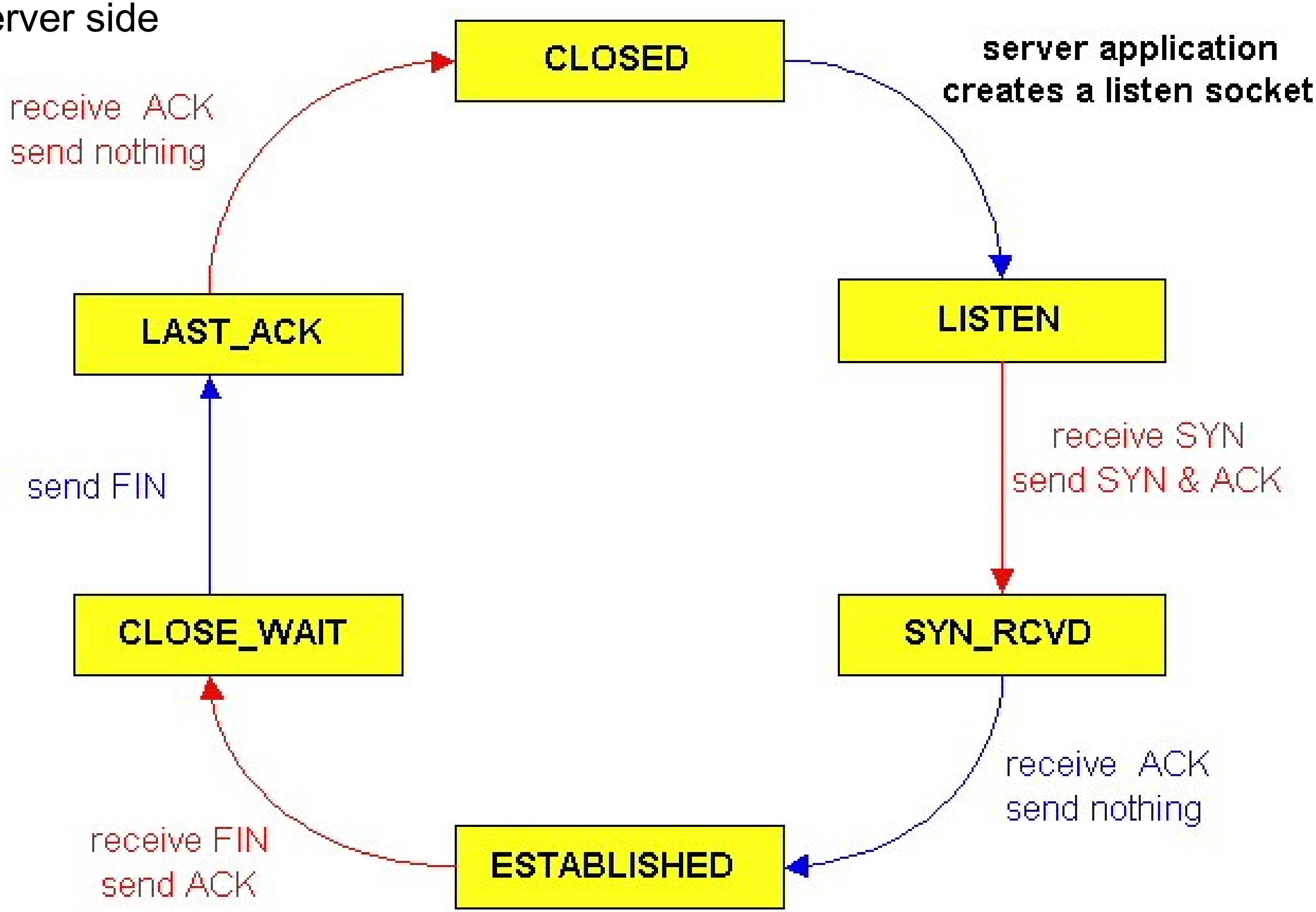
TCP connection lifecycle state diagram

- Client side



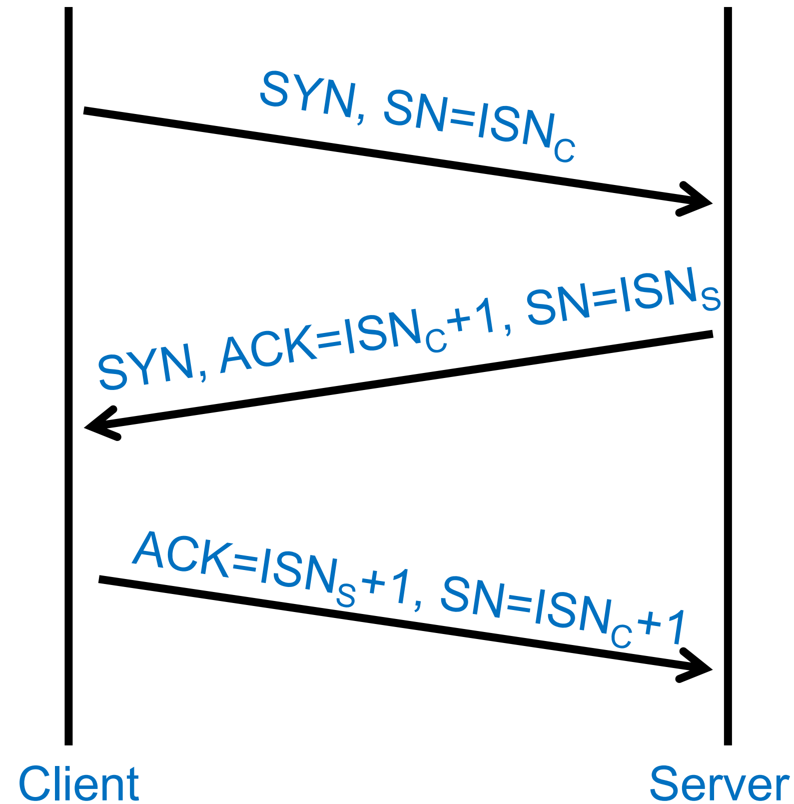
TCP connection lifecycle state diagram

- Server side



TCP connection setup:
3-way handshake

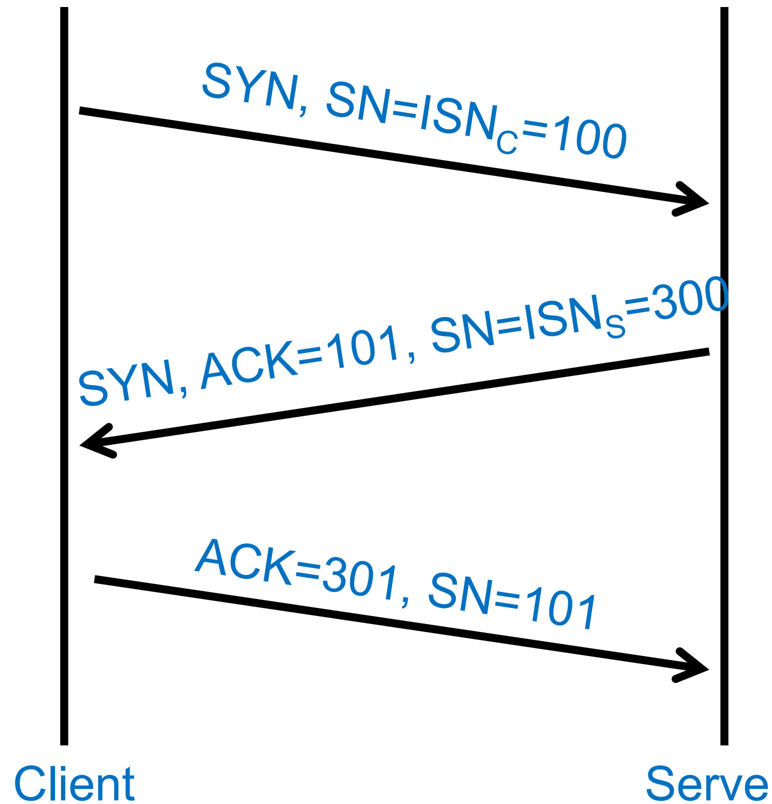
- Step 1:
  client sends message with
  - SYN flag set
  - Sequence number (SN) field containing Client Initial Sequence Number (ISN_C)
- Step 2:
  server sends message with
  - SYN and ACK flags set
  - Acknowledgment field containing ISN_C+1
  - Sequence number field containing Server Initial Sequence Number (ISN_S)
- Step 3:
  client send message with
  - ACK flag set
  - Acknowledgment field containing ISN_S+1
  - Sequence number field containing ISN_C+1



Connection Setup

TCP connection setup – numerical example

- Step 1:
  client sends message with
  - SYN flag set
  - Client Initial Sequence Number=100
- Step 2:
  server sends message with
  - SYN and ACK flags set
  - Client's Initial Sequence Number acknowledged
    - By setting acknowledgment field to 101
      → Next expected sequence number
  - Server Initial Sequence Number=300
- Step 3:
  client send message with
  - ACK flag set
  - Server's Initial Sequence Number acknowledged
    - By setting acknowledgment field to 301
      → Next expected sequence number
  - Sequence number field set to 101



4.3 Connection – Tear Down

Either side can initiate tear down

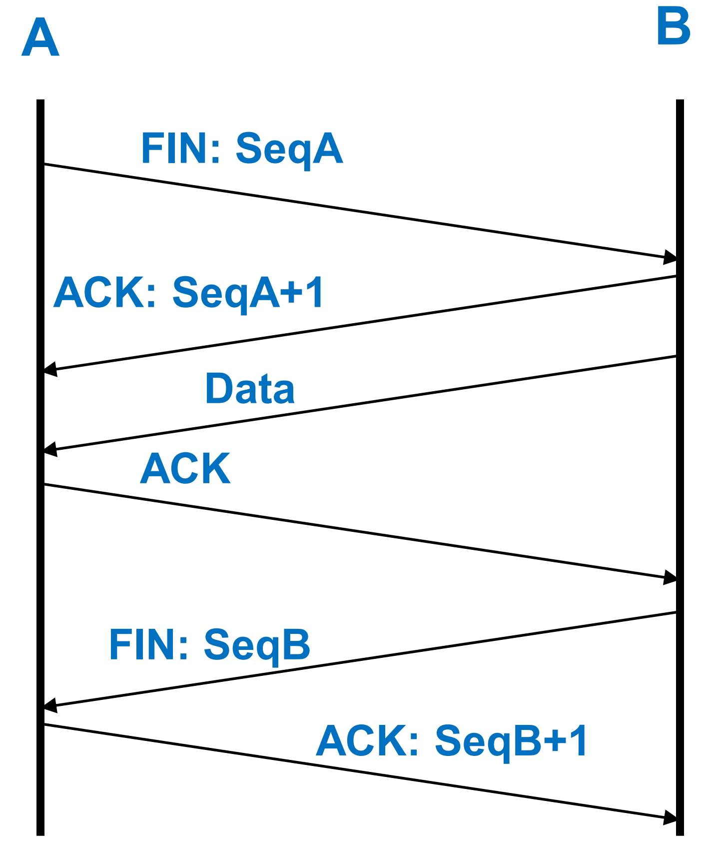
- send FIN signal
- i.e. “I’m not going to send any more data”

“other side” can continue sending Data

- half open connection
- Has to continue to acknowledge

Acknowledge with FIN

- acknowledge last sequence number + 1



Connection – Tear Down

TCP connection teardown:

4-way handshake

- Can be initiated by both sides
- E.g. client

Step 1: Client sends FIN message

Step 2: Server sends ACK message

- Half-duplex data transfer may continue
  - From Server to Client
  - Client still acknowledges data

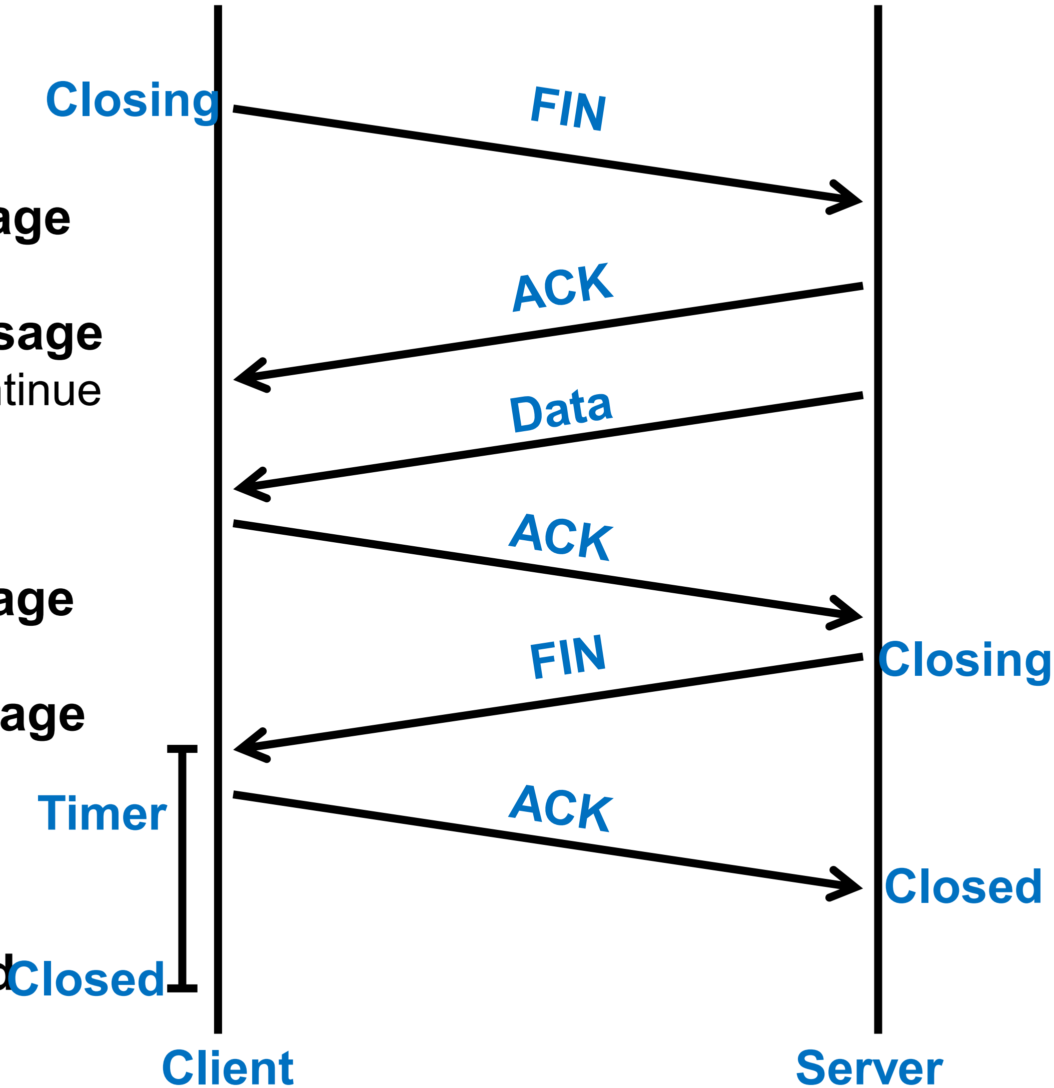
Step 3: Server sends FIN message

Step 4: Client sends ACK message

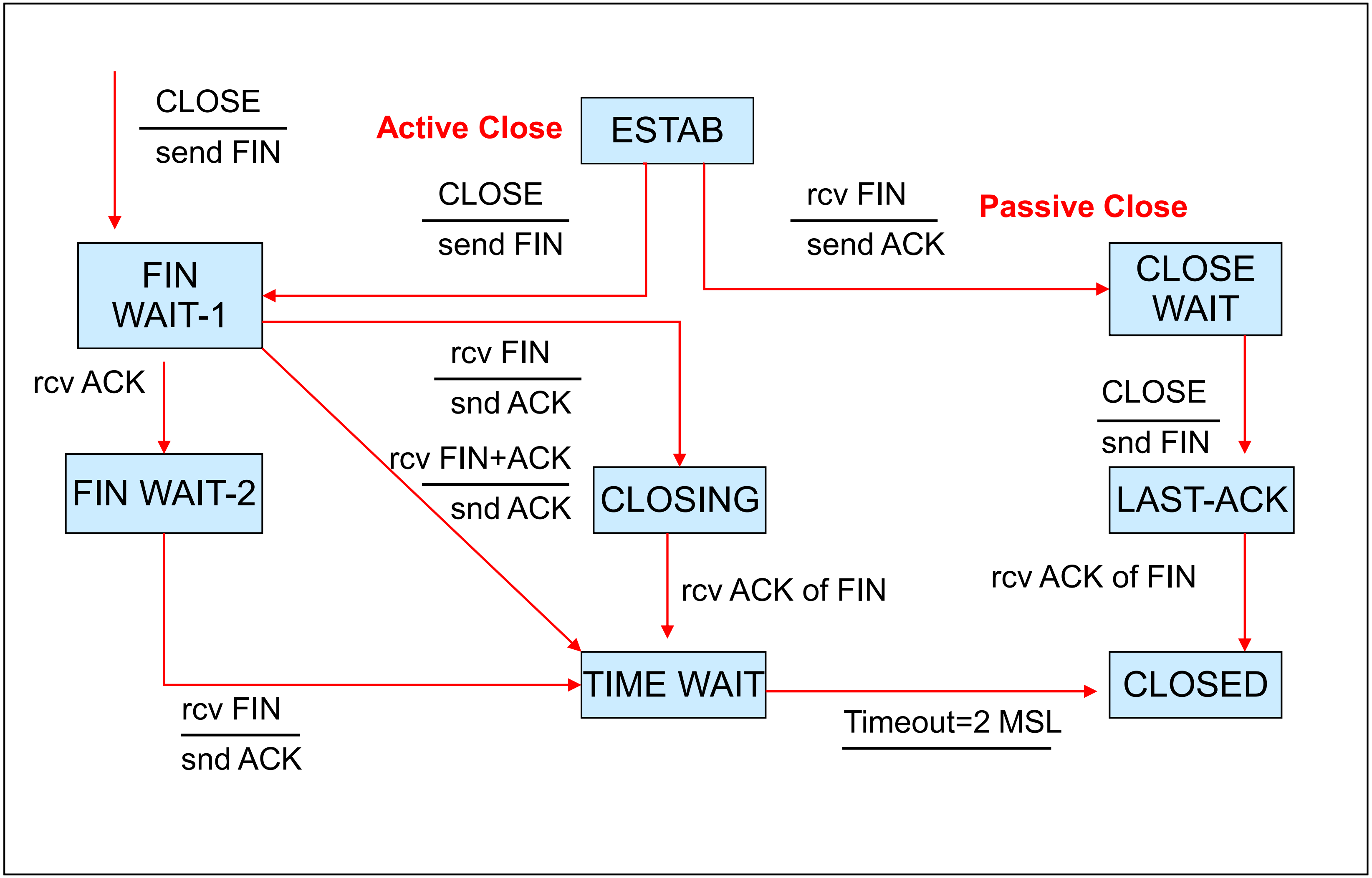
- Starts closing timer
- Closes connection at timeout
- May resend ACK message if lost

Steps 2 and 3 may be combined

- If server has no data to send



State Diagram: Connection Tear Down



4.4 Sequence Number Space

Each byte in the byte stream is numbered

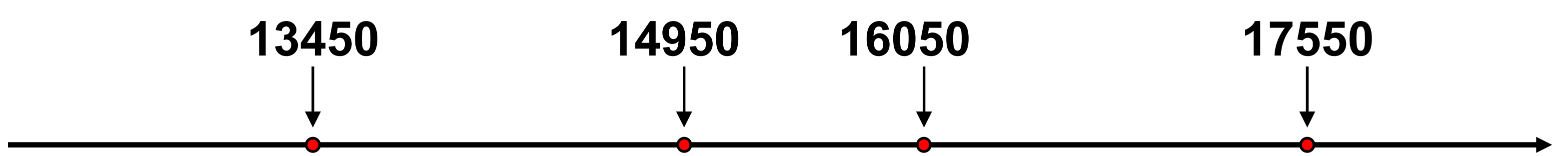
- 32 bit value
- wraps around
- initial values selected at start up time

TCP breaks up the byte stream in packets (“segments”)

- packet size is limited to the Maximum Segment Size
- set to prevent packet fragmentation

Each segment has a sequence number

- indicates where it fits in the byte stream



Sequence Numbers

32 Bits, Unsigned

Why so big?

- for sliding window, must have
  - |Sequence Space| > |Sending Window| + |Receiving Window|
  - 2^32 > 2 * 2^16.
  - No problem
- also, want to guard against stray packets
  (stray packets – “vagabundierende Pakete”)
  - with IP, assume packets have maximum segment lifetime (MSL) of 120 sec
    - i.e. can linger in network for up to 120sec
  - sequence number would wrap around in this time at 286Mbps

Additional reading:

- RFC 1323 / PAWS
  - protect against wrapped sequence numbers
  - TCP extension for high-speed paths

Checksum (mostly) guarantees end-to-end data integrity

Sequence numbers detect packet sequencing problems:

- duplicate: → ignore
- reordered: → reorder or drop
- lost: → retransmit

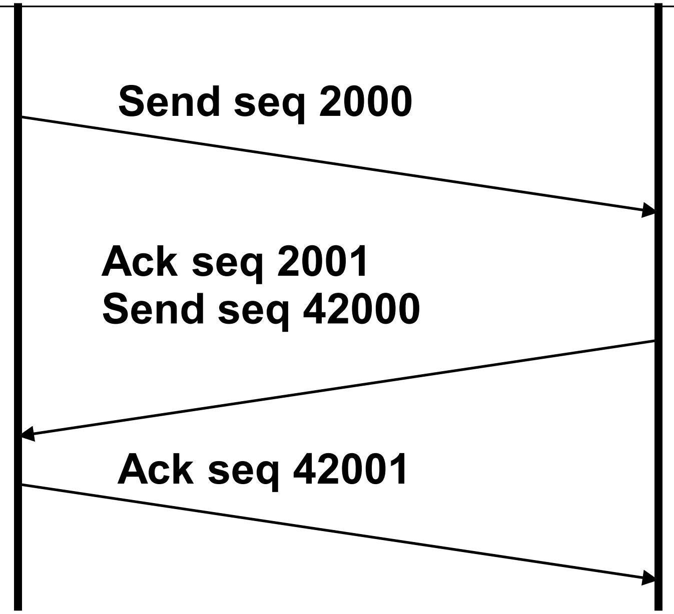
Lost segments detected by sender

- use time out to detect lack of acknowledgment
- need estimate of the roundtrip time to set timeout

Retransmission requires that sender keeps a copy of the data

- copy is discarded when ACK is received

Bidirectional Communication (Duplex)

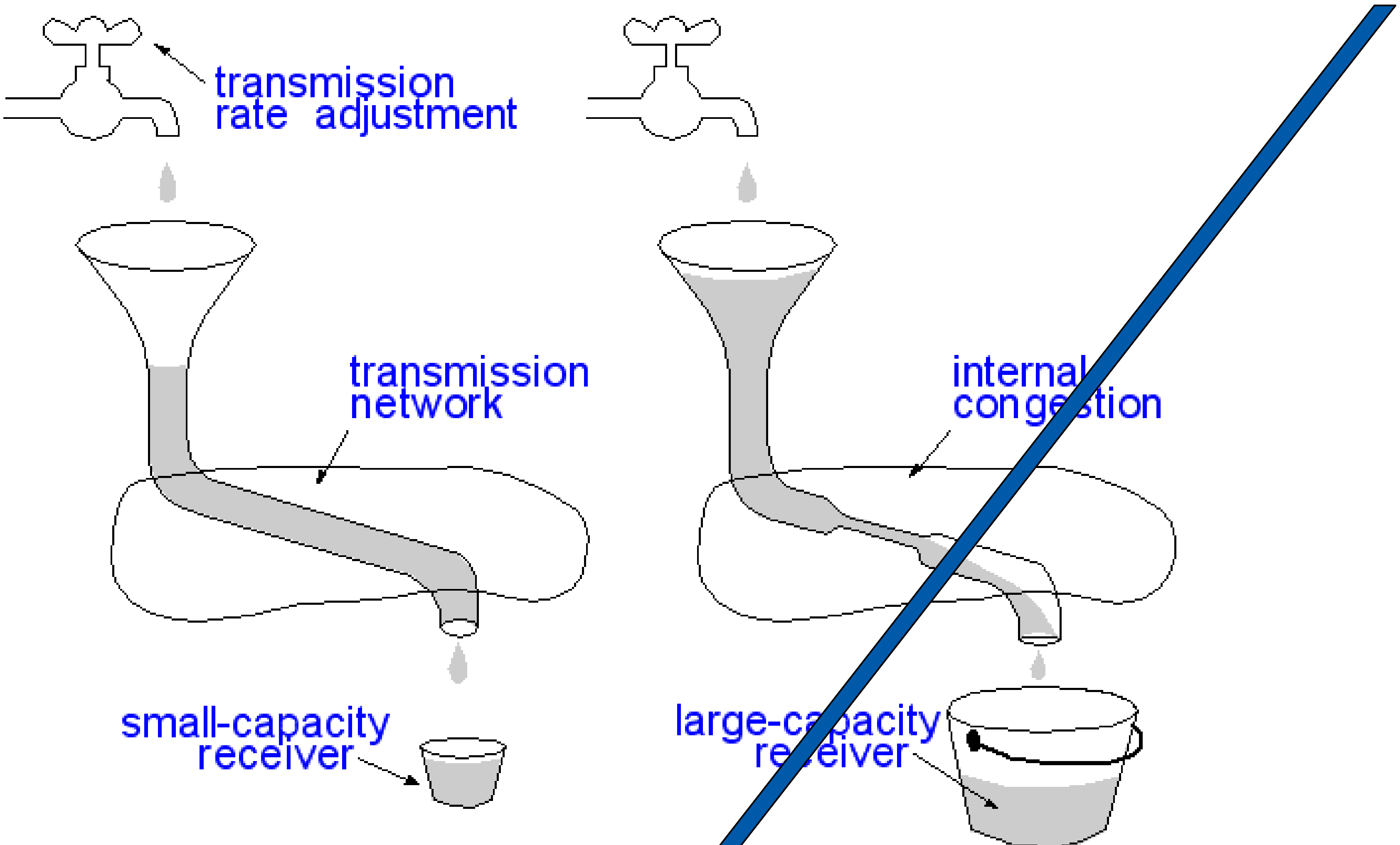


Each side of a connection can send and receive (duplex)

i.e.

- to maintain different sequence numbers for each direction
- single segment can contain new data for one direction, plus acknowledgement for the other
  - but some contain only data & others only acknowledgement

5 Flow Control – in General



Controlled by
Window: advertised window awnd

Controlled by
Window: congestion window cwnd

5.1 TCP Flow Control – in General

Sliding window protocol

- for window size n
  → sender can send up to n bytes without receiving an acknowledgement
- when the data is acknowledged
  then the window slides forward

Window size determines

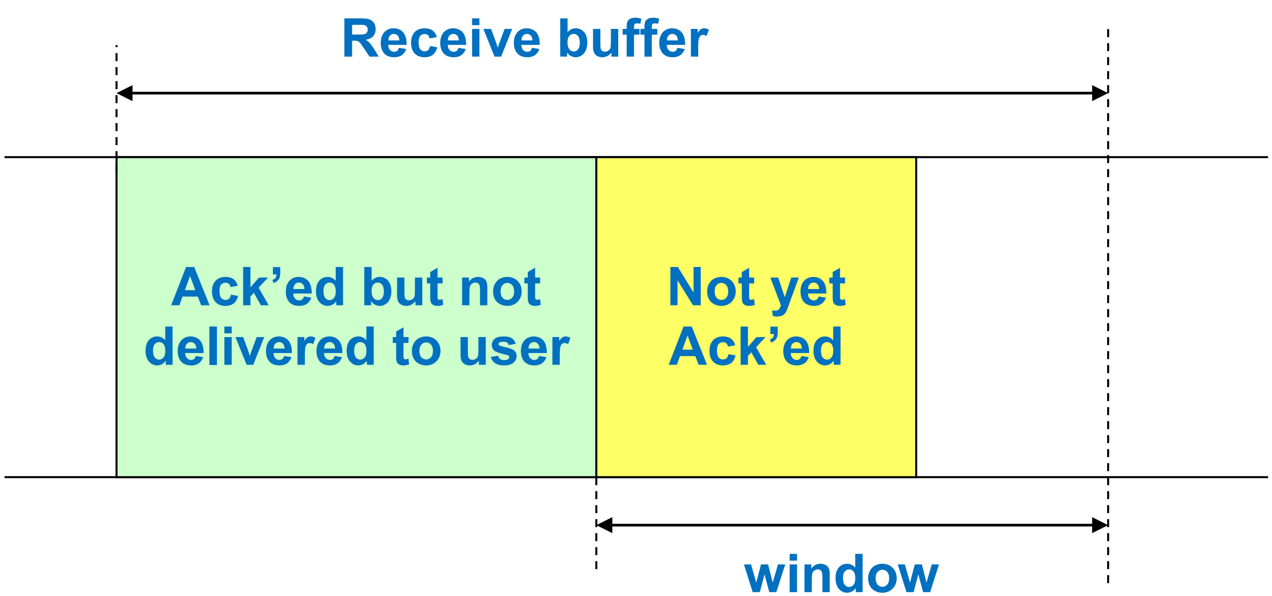
- how much unacknowledged data the sender can send

But there is one more detail …

TECHNISCHE UNIVERSITÄT DARMSTADT

Complication

- TCP receiver can delete acknowledged data
  - only after the data has been delivered to the application
- I.e. depending on how fast the application is reading the data,
  - the receiver’s window size may change!



Solution

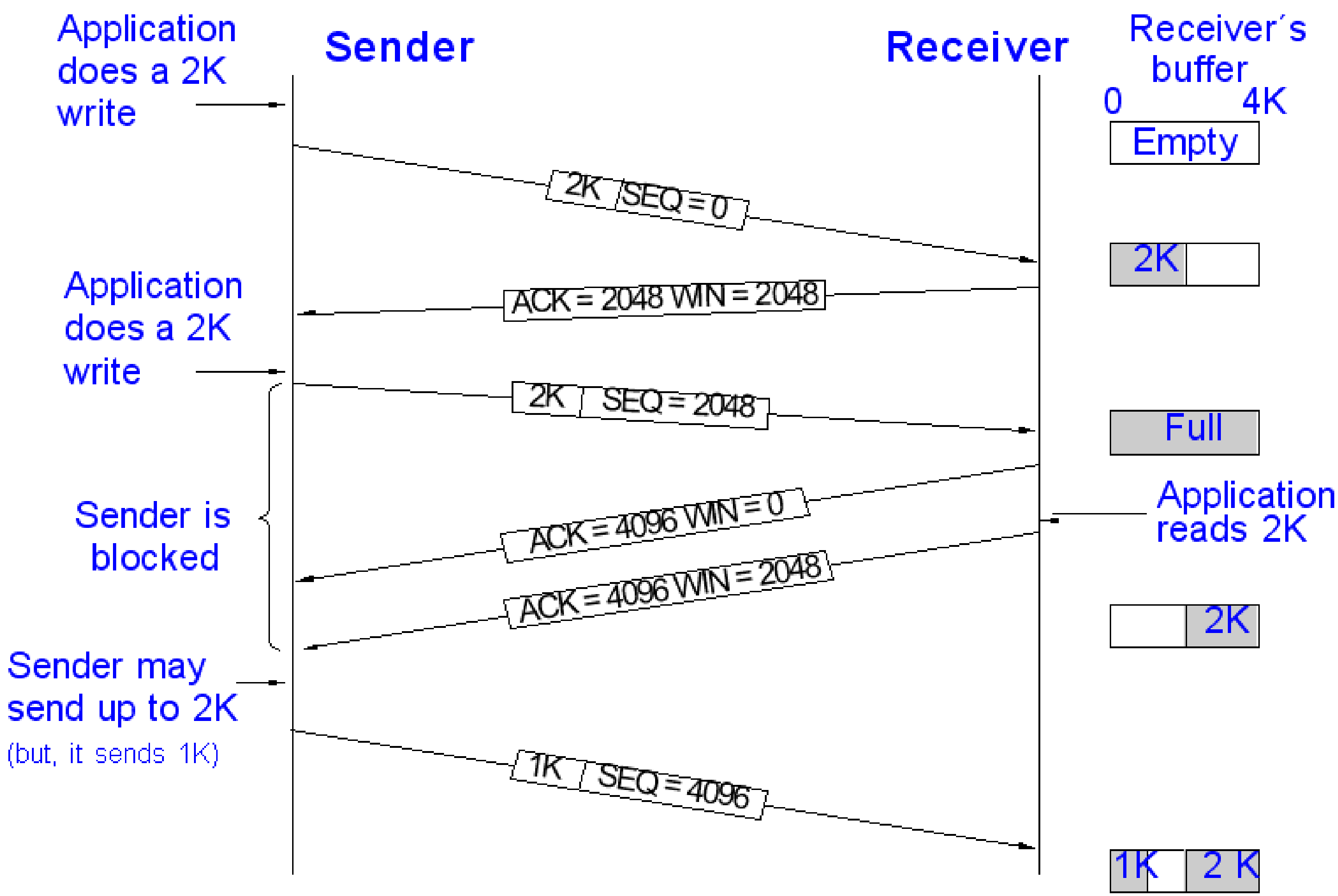
Receiver informs the sender
- about the current window size
  - in every packet it transmits to the sender

Sender uses
- this current window size
- instead of a fixed value

Window size (also called ADVERTISED WINDOW)
- is continuously changing

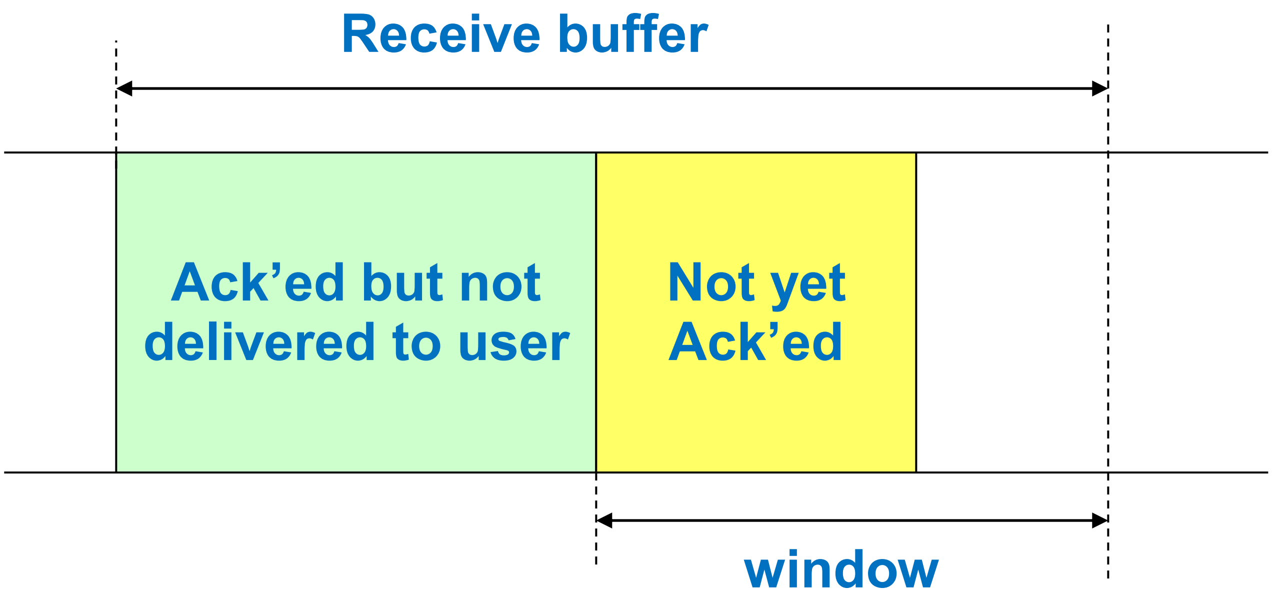
May go to zero!
- sender not allowed to send anything!

Solution, e.g.

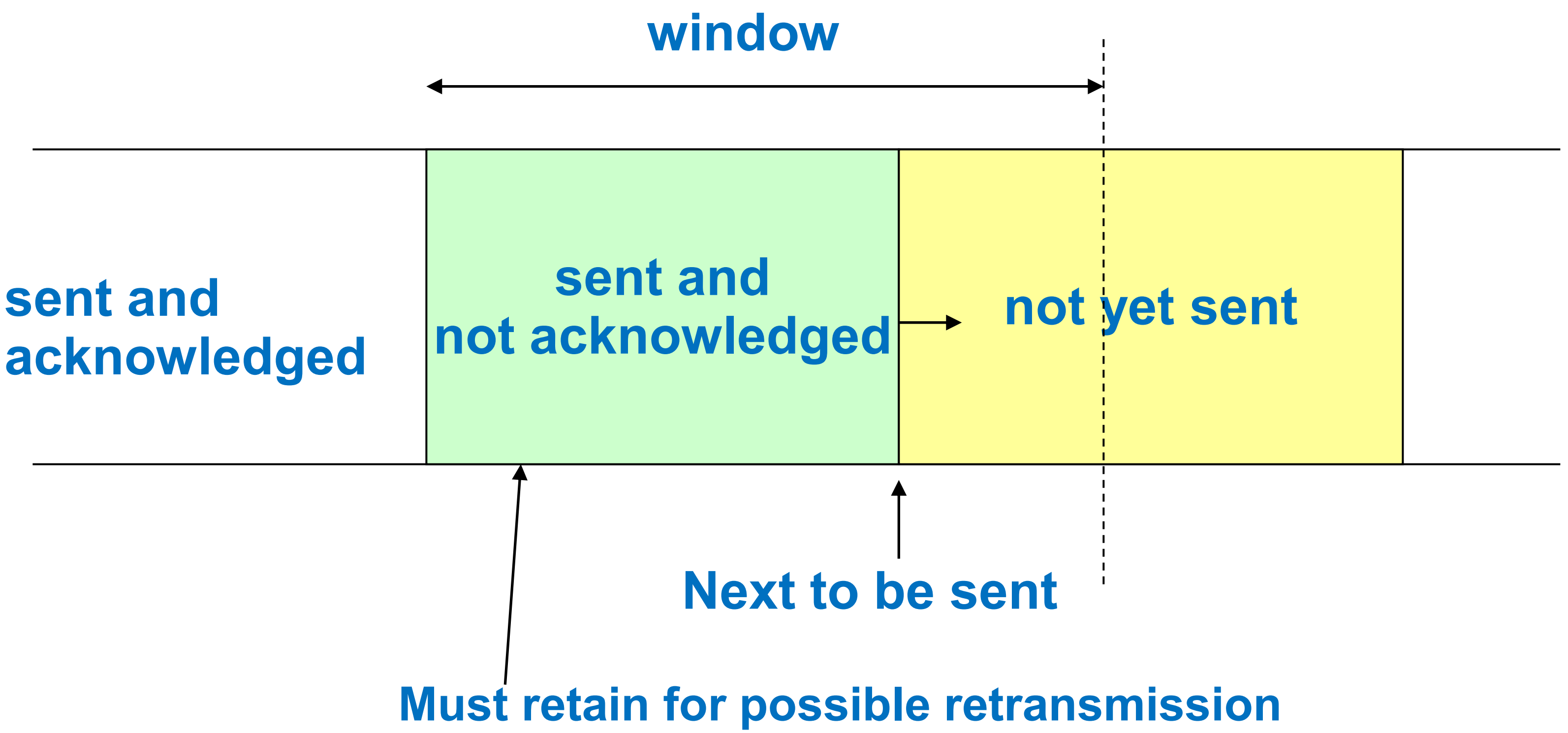


Application does a 2K write
Sender
Receiver
Receiver´s buffer
0
4K
Empty
2K SEQ = 0
2K
ACK = 2048 WIN = 2048
Application does a 2K write
2K SEQ = 2048
Full
Sender is blocked
ACK = 4096 WIN = 0
Application reads 2K
ACK = 4096 WIN = 2048
2K
Sender may send up to 2K
(but, it sends 1K)
1K SEQ = 4096
1K
2 K

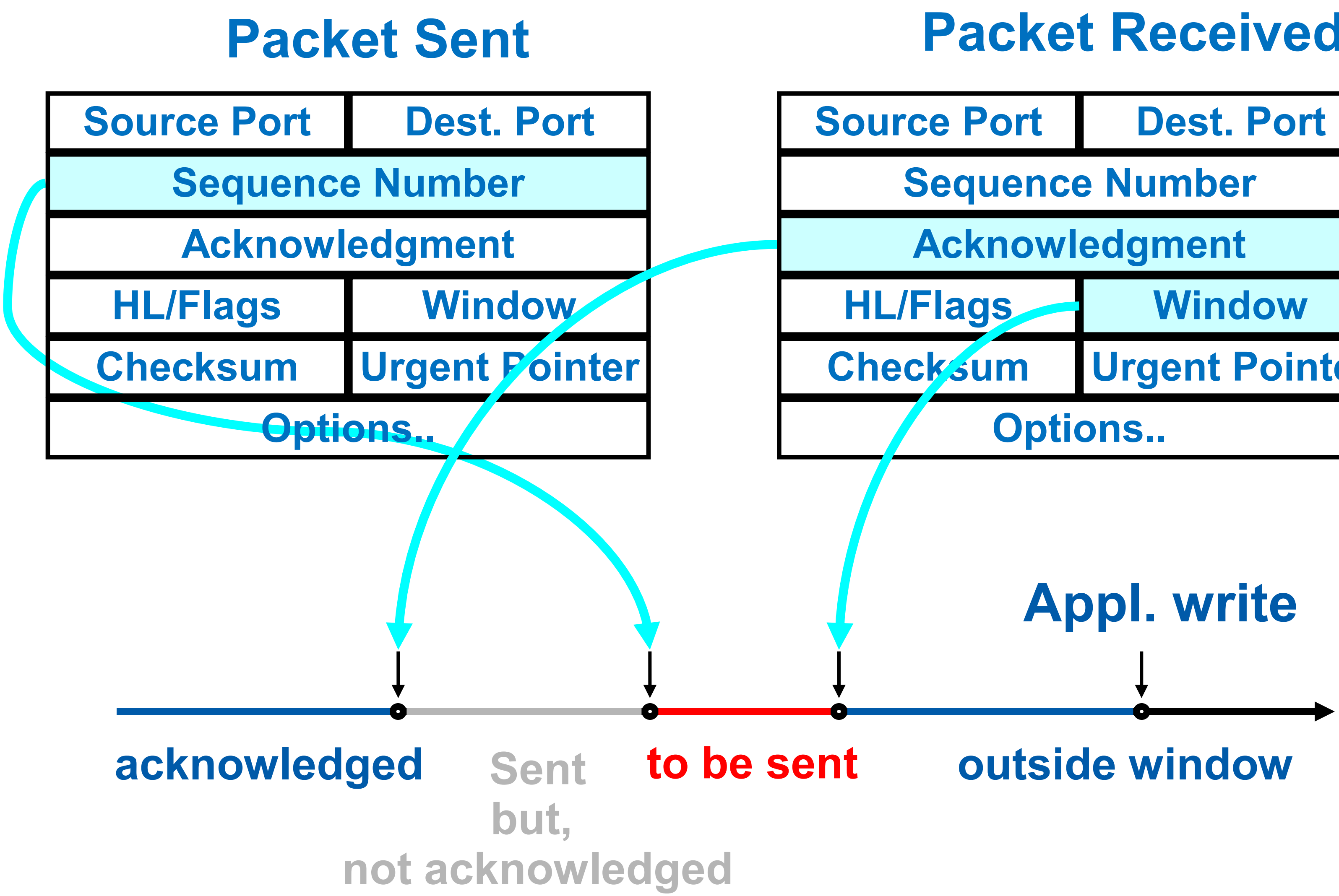
Window Flow Control: Receivers Side



Window Flow Control: Senders Side



Window Flow Control: Send Side



5.2 TCP Window Flow Control - Special Cases

Optimization for low throughput rate

Problem:

Telnet (and ssh) connection to interactive editor reacting on every keystroke

- 1 character typed requires up to 162 bytes
  - data:
    - 20 bytes TCP header,
    - 20 bytes IP header,
    - 1 byte payload
  - ACK:
    - 20 bytes TCP header, 20 bytes IP header
  - editor echoes character:
    - 20 bytes TCP header, 20 bytes IP header, 1 byte payload
    - ACK:
      - 20 bytes TCP header, 20 bytes IP header

commonly used solution

- delay acknowledgements and window update by 500 ms
  - hoping for more data to come

Nagle's algorithm, 1984

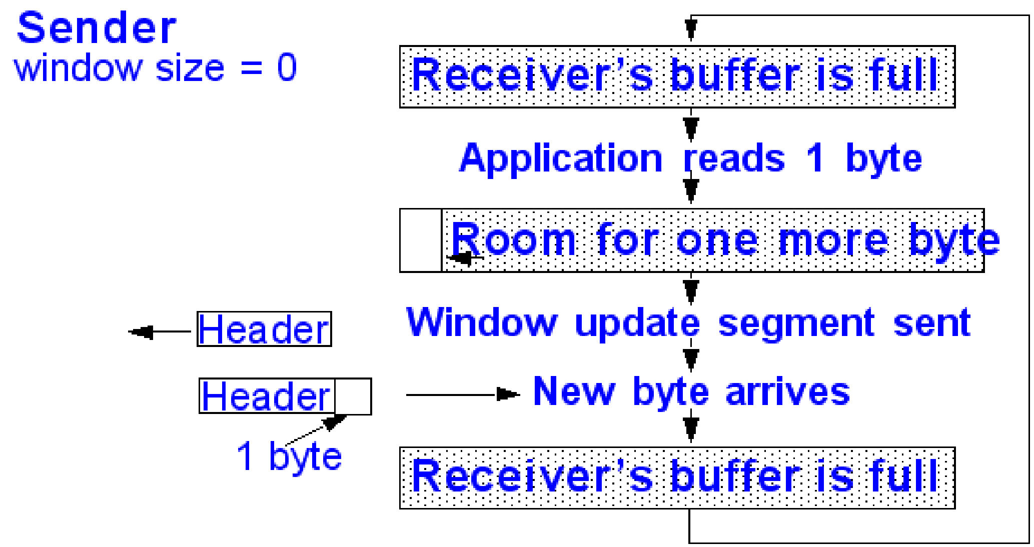
- send first byte immediately
- keep on buffering bytes until first byte has been acknowledged
- then send all buffered characters in one TCP segment and start buffering again

comment

- effect e.g. X-Windows: jerky pointer movements

TCP Window Flow Control: Special Cases

Silly window syndrome (Clark, 1982)



Problem

- data on sending side arrives in large blocks
- but receiving side reads data only one byte at a time

Clark's solution:

- prevent receiver from sending window update for 1 byte
- certain amount of space must be available in order to send window update
- min(X,Y):
  - X=max. segment size (MSS),
  - Y=buffer/2

Bidirectional Communication

- each side acts as sender & receiver

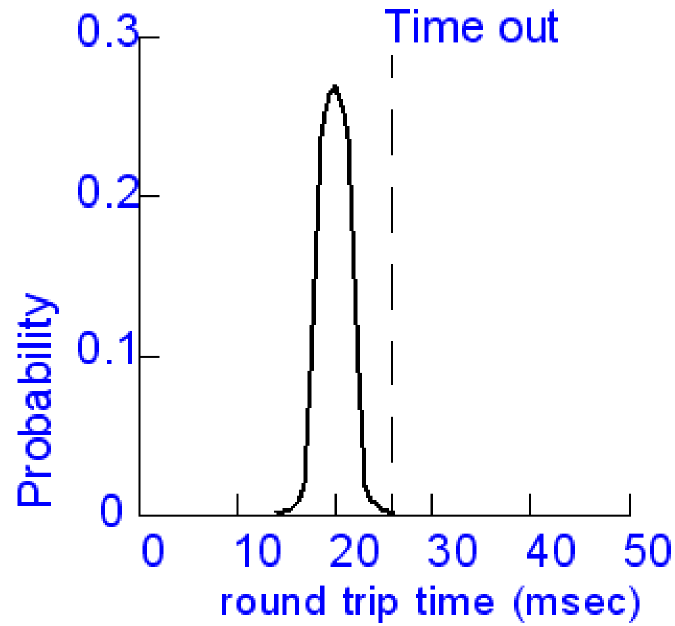
- every message
  - contains acknowledgement of received sequence
    - even if no new data has been received
  - advertises window size
    - size of its receiving window
  - contains sent sequence number
    - even if no new data is being sent

Ongoing Communication

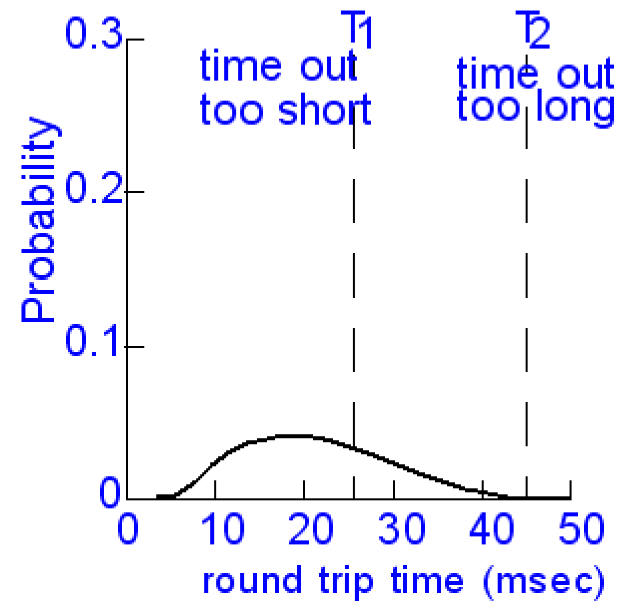
When does a sender actually sends a message?

- when sending buffer contains at least max. segment size (- header sizes) bytes
- when application tells it to
  - set PUSH flag for last segment sent
- when timer expires

Ideal situation:



Example of real situation

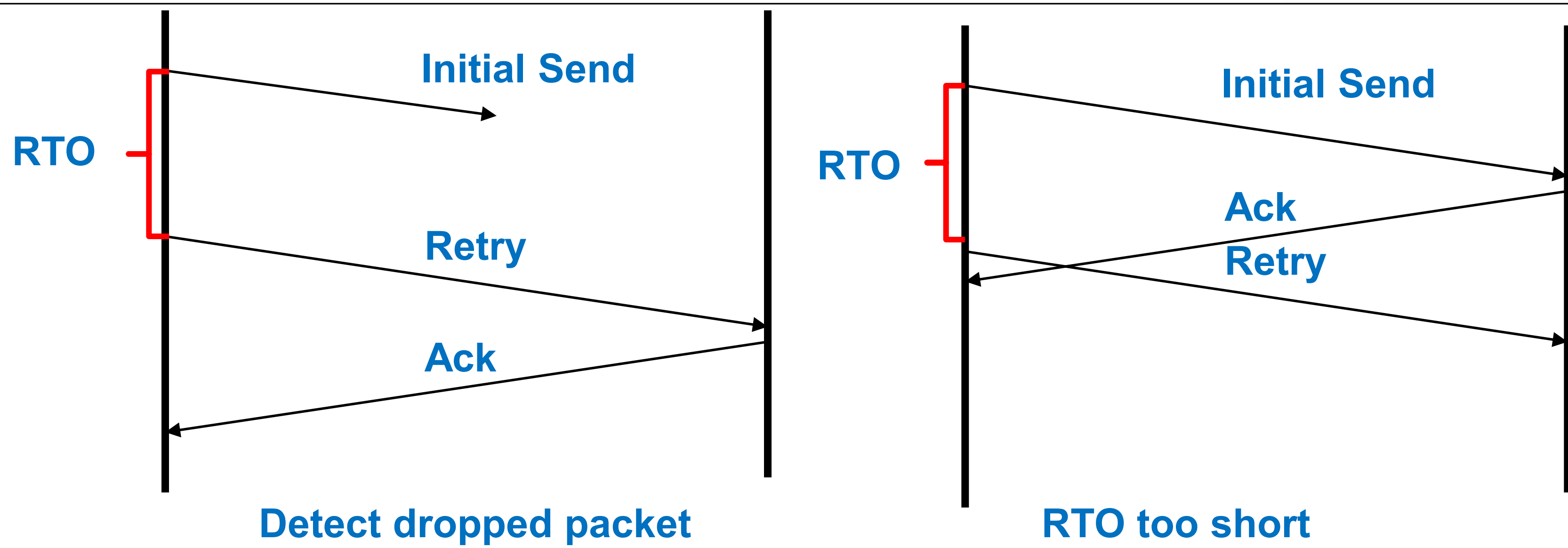


TCP Must Operate Over Any Internet Path

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

➔ Must estimate RTT dynamically

6.1 Setting Retransmission Timeout (RTO)



Retransmission Timeout (RTO)

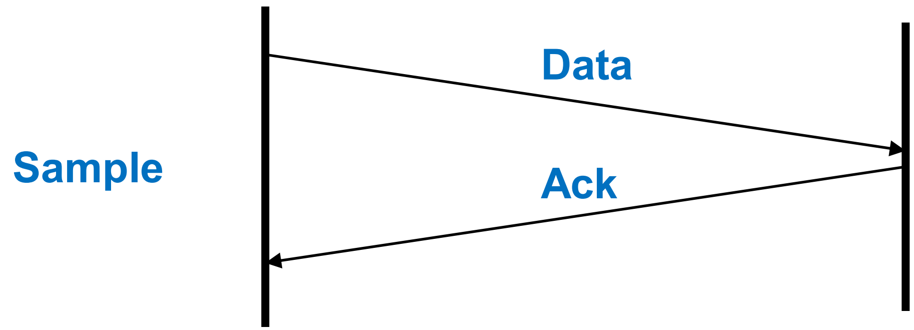
- time between sending & resending segment

Challenge

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

Round-trip Time Estimation

Every Data/Ack pair gives new RTT estimate



Can get lots of short-term fluctuations

6.2 Original TCP Round-trip Estimator (Jacobson, 1988)

Round trip times estimated as a moving average:

- new_RTT = α * (old_RTT) + (1 – α) * (new_sample)

Smoothing factor

- recommended value for α:
  - 7/8 (0.8 - 0.9)
  - 0.875 for most TCP's

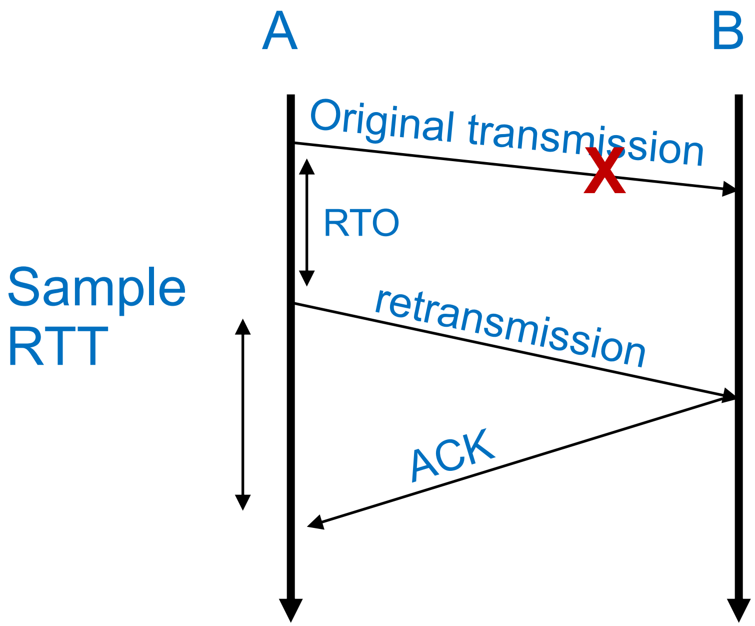
Retransmit timer set to

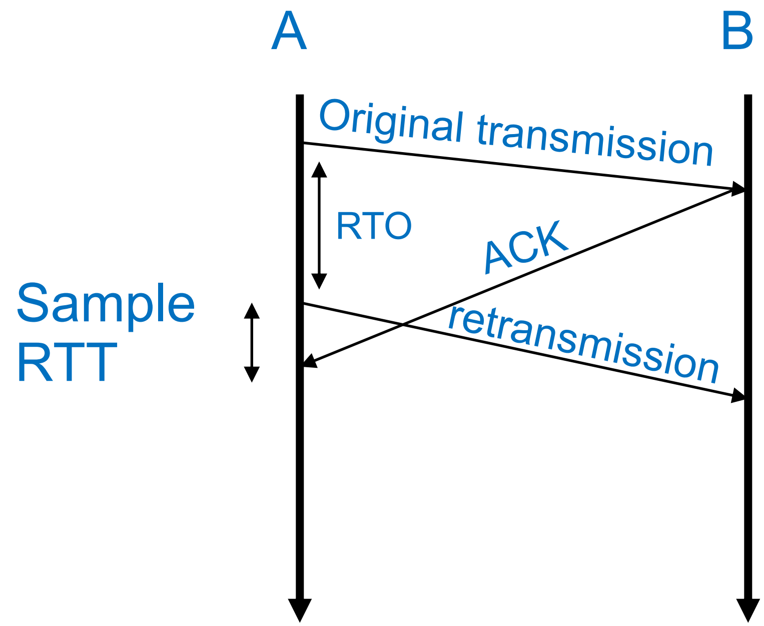
- RTO = β new_RTT,
  - where $\beta = 2$
  - want to be somewhat conservative about retransmitting

Problem

- static β not able to adapt to high variation in observed RTTs during high load conditions
- solution used today
  - estimate both RTT and variance in RTT
  - use the estimated variance in place of constant β

6.3 RTT Sample Ambiguity

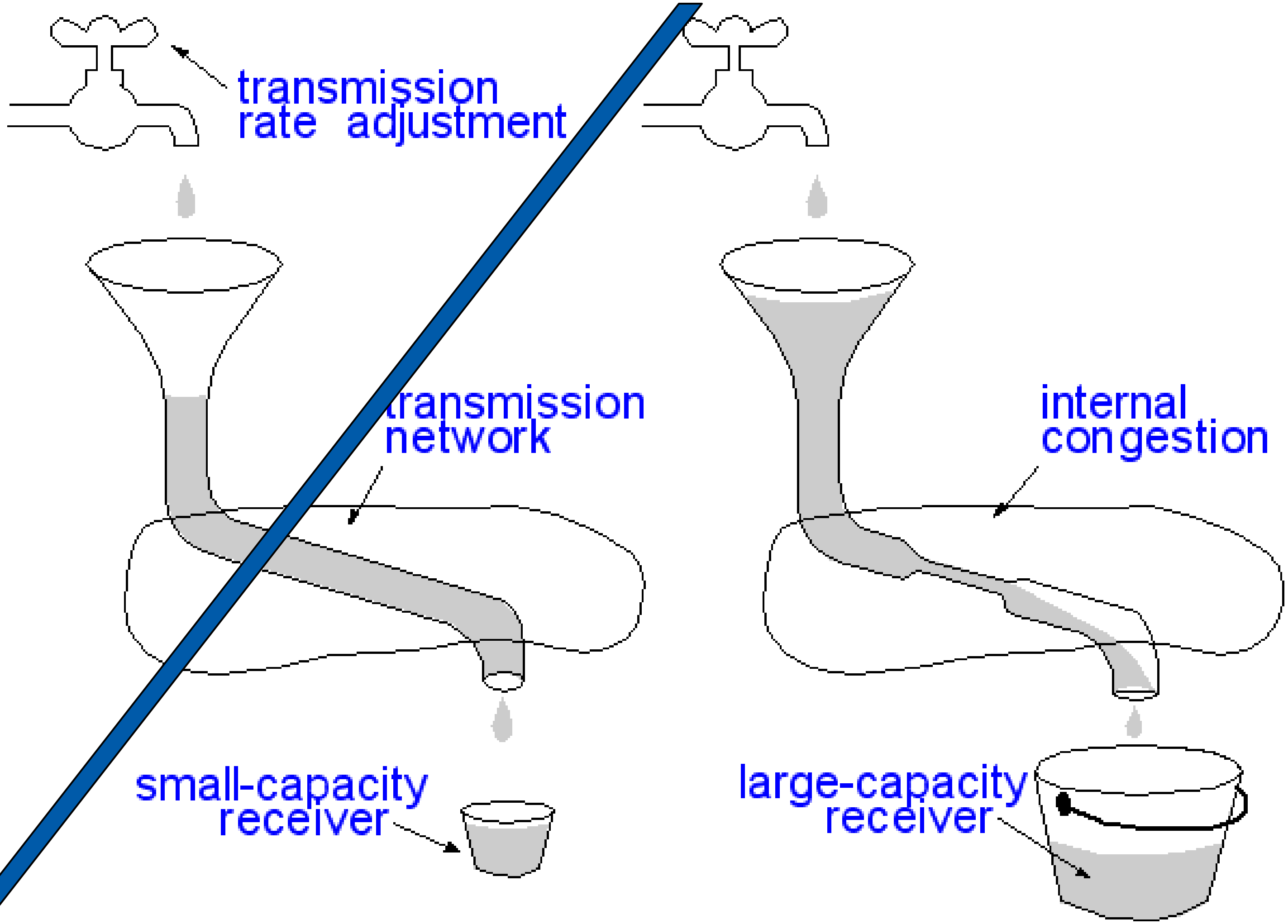




Solution

- Karn’s algorithm
  - ignore sample for segment that has been retransmitted
  - timer backoff strategy
    - retransmission timer (new_RTO) = γ · old_RTO
    - typical γ = 2
    - resume normal computation when ACK received for non-retransmitted segment

7 Congestion Control



Controlled by Window: advertised window awnd

Controlled by Window: congestion window cwnd

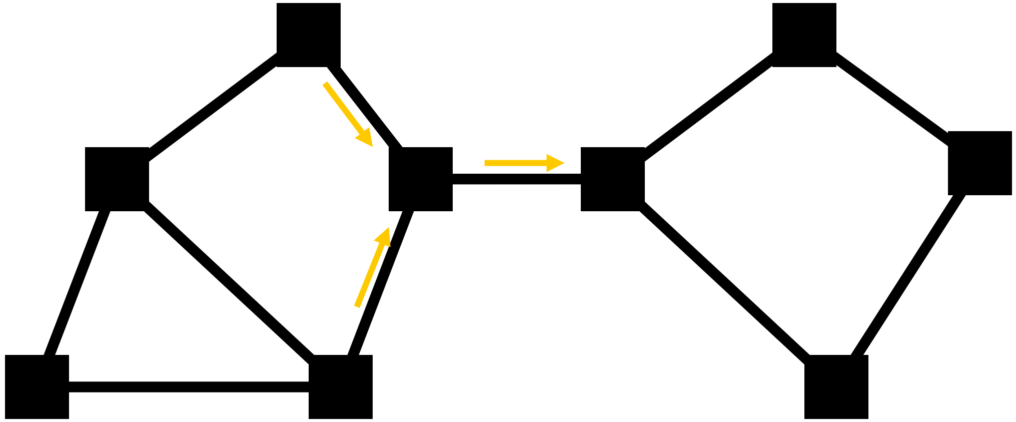
7.1 Congestion at Transport Layer - in General

Load placed on the network
is higher than
the capacity of the network

- not surprising: independent senders place load on network

Results in packet loss: routers have no choice

- can only buffer finite amount of data
- end-to-end protocol will typically react, e.g. TCP



Congestion – the Problem

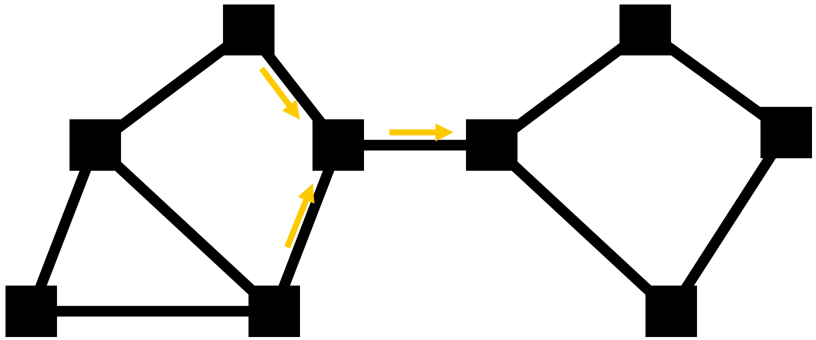
TECHNISCHE UNIVERSITÄT DARMSTADT

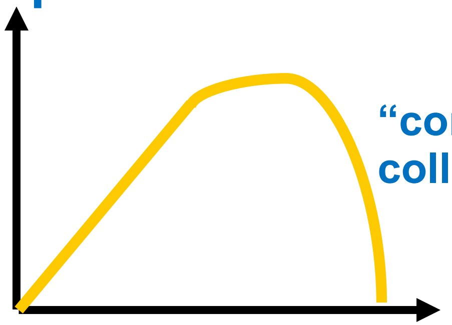
Wasted bandwidth: retransmission of dropped packets

Poor user service: unpredictable delay, low user goodput

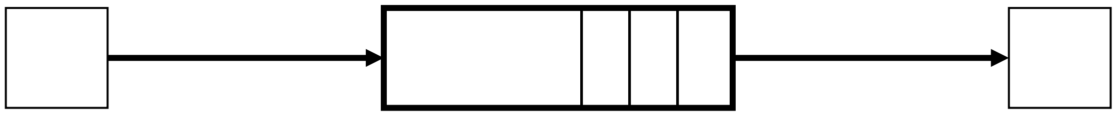
Increased load can even result in lower network goodput

- switched nets:
  - packet losses create lots of retransmissions
- broadcast Ethernet:
  - high demand → many collisions





Sending Rate of Sliding Window Protocol



Suppose

- Sender A uses a sliding window protocol to transmit a large data file to B
- Window size = 64KB (i.e. 2^16 Bytes)
- Network round-trip delay is 1 second

Which is the expected sending rate?

- → 64KB in 1 second (2^16 Bytes / 1 second = 65536Byte/second
  = 524288bits/second ≈ 524.3kbps)

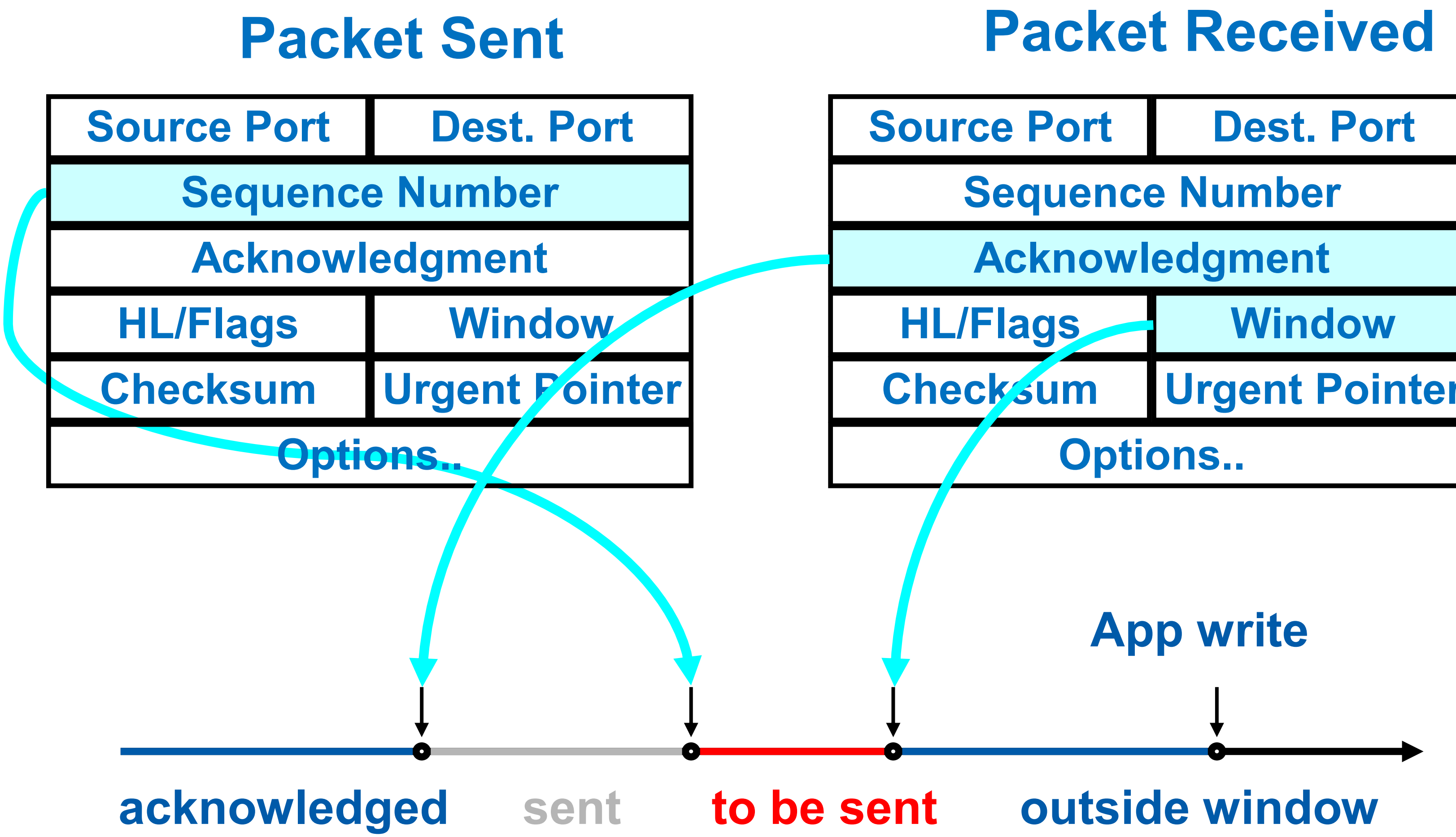
What if a

- network link is (only) 524.3kbps
- but there are 1000 people who are transferring files
  over that link using the sliding window protocol?

→ Packet losses, timeouts, retransmissions, more packet losses…

- nothing useful gets through, collapse due to congestion!

7.2 Limits of TCP Window Flow Control



TCP Flow Control alone is not enough

We have talked about how TCP's advertised window which is used for flow control

- to prevent the sender sending faster than the receiver can handle

If the receiver is sufficiently fast,
→ the advertised window will be maximized at all time

→ leads to collapse due to congestion

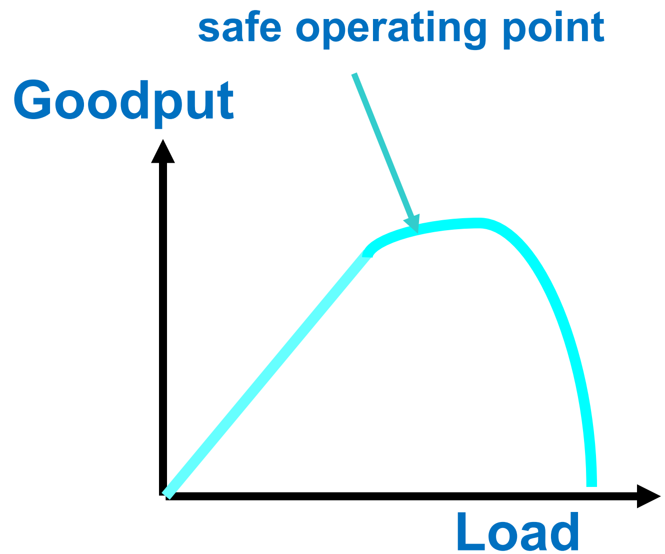
- as in the previous example if
  - there are too many senders or
  - the network is too slow

Key 1: Window size determines sending rate

Key 2: Window size must be dynamically adjusted to prevent collapse due to congestion

Send too slow:
link sits idle
- wastes time

Send too fast:
link is kept busy but....
- queue builds up in router buffer (delay)
- overflow buffers in routers (loss)
- many retransmissions, many losses
- network goodput goes down



Abstract View

We ignore

- internal structure of the network and
- model network as having a single bottleneck link

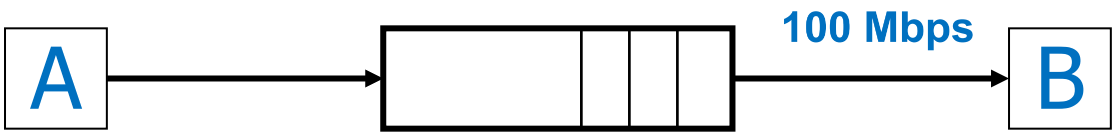
7.3 Three Congestion Control Problems

Adjusting to bottleneck bandwidth

Adjusting to variations in bandwidth

Sharing bandwidth between flows

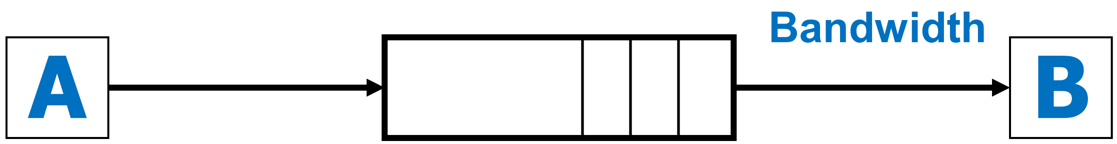
Single Flow, Fixed Bandwidth



Adjust rate to match bottleneck bandwidth

- without any a priori knowledge
- could be gigabit link, could be a modem

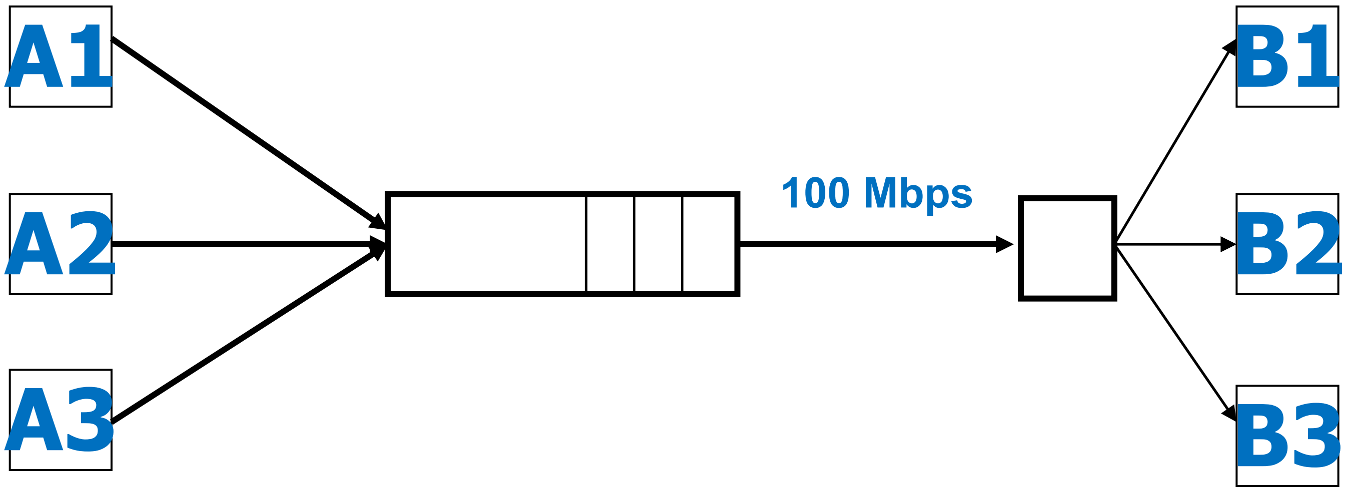
Single Flow, Varying Bandwidth



Adjust rate to match instantaneous bandwidth

Bottleneck can change because of a routing change

Multiple Flows

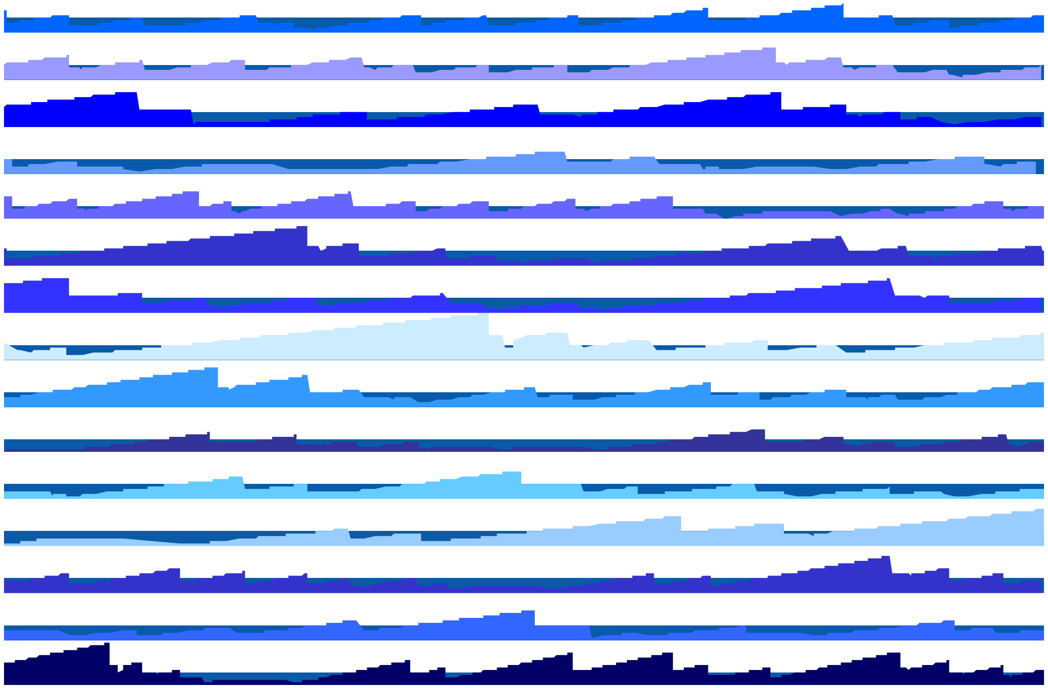


(two) issues:

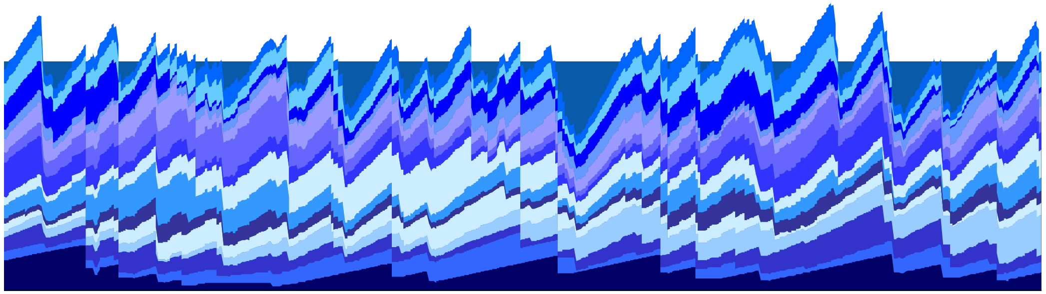
- Adjust total sending rate to match bottleneck bandwidth
- Allocation of bandwidth between flows

Multiple Flows and how TCP shares capacity

individual flow rates



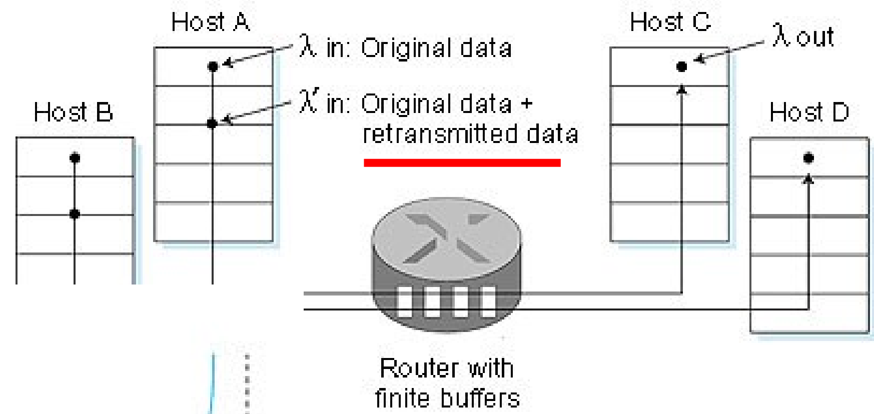
aggregate flow rate

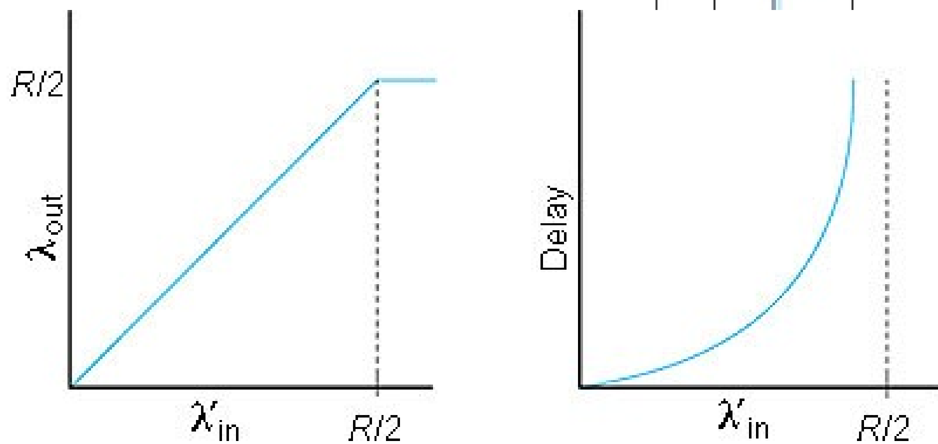


available capacity

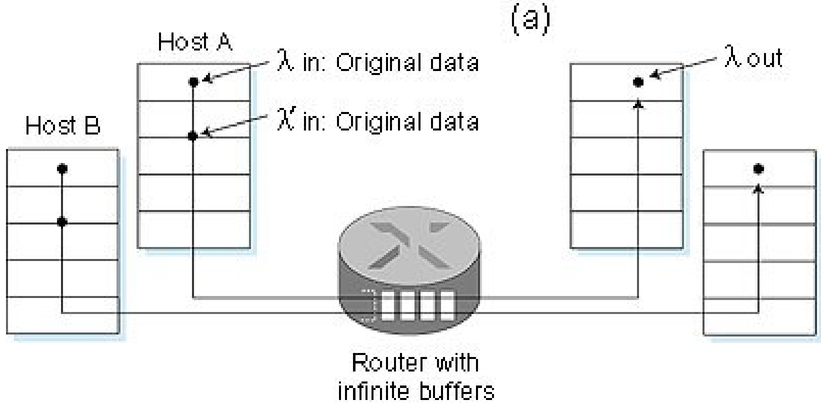
Source Damon Wischik, UCL

Why is Congestion Bad - Revised?





(a) (b)



From Kurose & Ross

Send without care

- many packet drops
- could cause collapse due to congestion

Reservations

- pre-arrange bandwidth allocations
- requires negotiation before sending packets

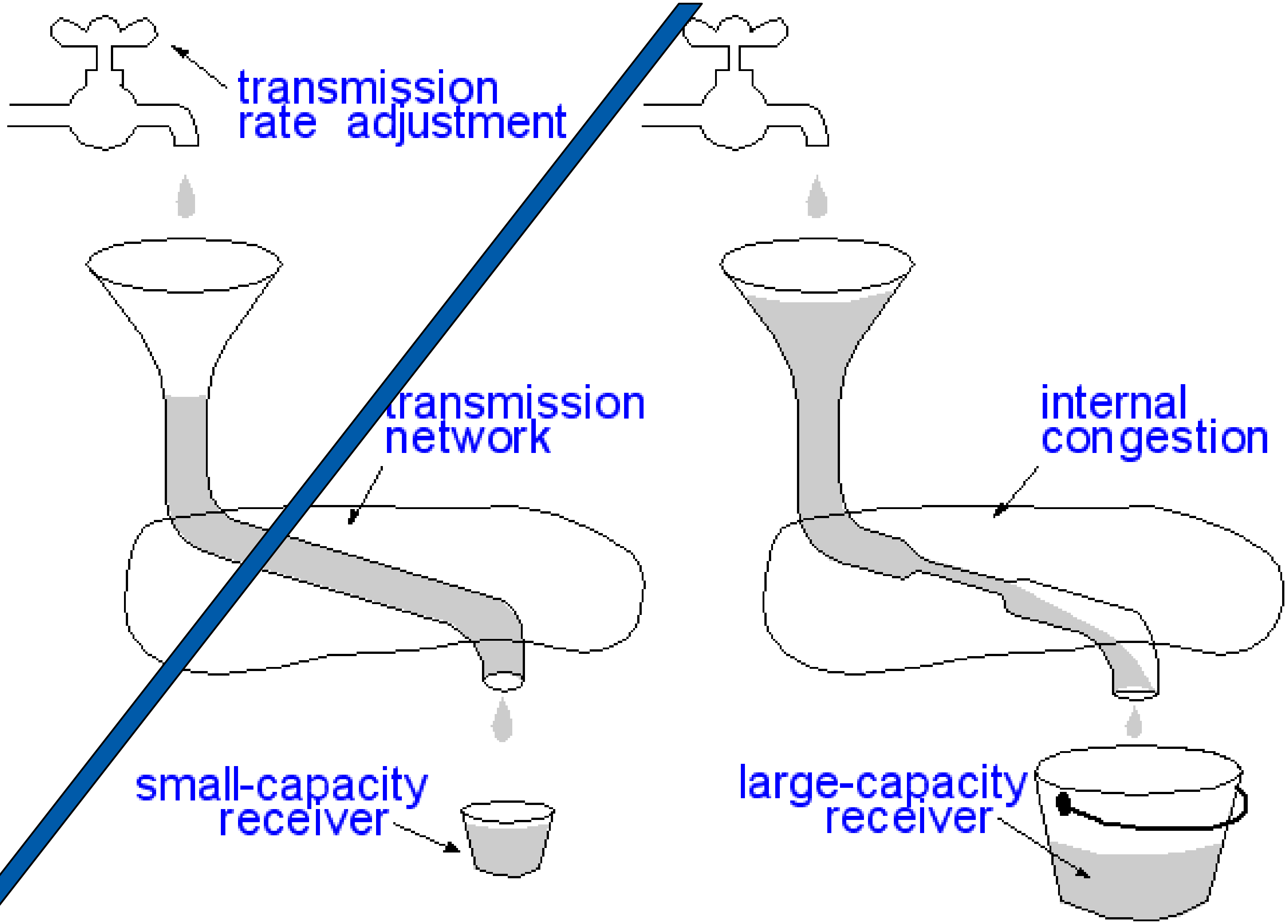
Pricing

- don’t drop packets for the high-bidders
- requires payment model

Dynamic Adjustment (TCP)

- technique
  - every sender probes network to test level of congestion
  - speed up when no congestion
  - slow down when congestion
- evaluation
  - suboptimal, messy dynamics, simple to implement
  - distributed coordination problem

7.5 TCP Congestion Control



Controlled by
Window: advertised window awnd

Controlled by
Window: congestion window cwnd

TCP Congestion Control

TCP connection has window

- controls number of unacknowledged packets
- Sending rate: ~Window/RTT
- Vary window size to control sending rate

Introduce a new parameter called congestion window (cwnd) at the sender

- congestion control is mainly a sender-side operation

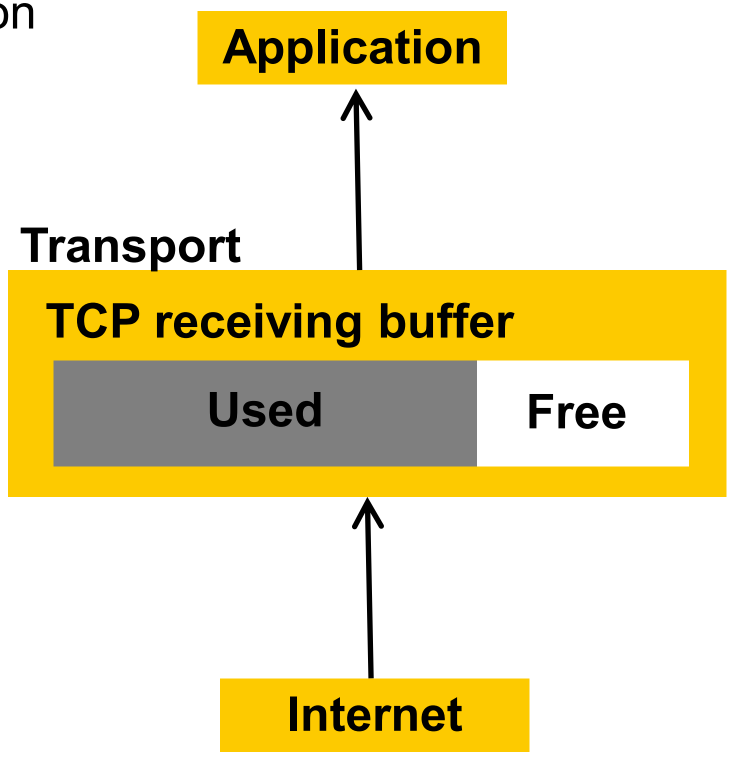
Flow control required to avoid flooding receiver

- Receiver allocates buffer space for receiving messages

- Buffer becomes available when
  - Data is acknowledged and
  - Data is read from buffer by receiving application

- But: application may be slower than network
  - E.g. high load situations

→ Receiving buffer may become full
  - How to react?



7.7 Congestion Window (cwnd)

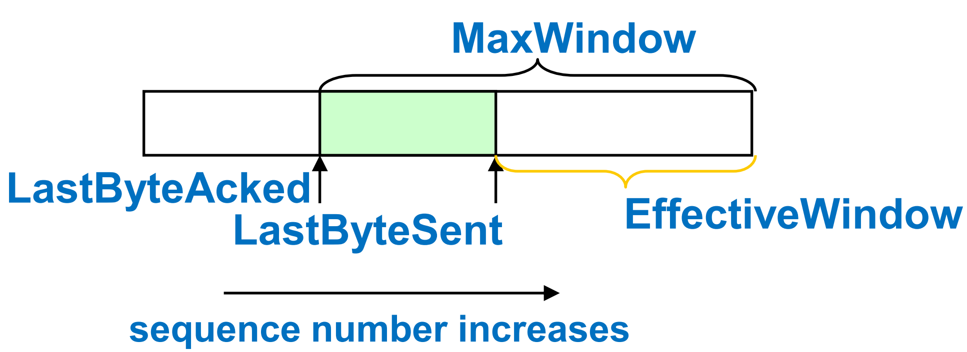
Limits how much data can be in transit

Implemented as # of bytes

Described as # packets in this lecture

MaxWindow = min(cwnd, awnd)

EffectiveWindow = MaxWindow – (LastByteSent – LastByteAcked)



Two Basic Components of Congestion Window (cwnd)

Detecting congestion

Rate adjustment algorithm (change cwnd size)

- depends on congestion or not

7.8 Basic Component 1: Detecting Congestion

Packet dropping is best indication for congestion

- delay-based methods are hard and risky

How do you detect packet drops? … ACKs

- TCP uses ACKs to signal receipt of data
- ACK denotes last contiguous byte received
  - actually, ACKs indicate next segment expected

Two signs of packet drops

- no ACK after certain time interval: time-out
- several duplicate ACKs (ignore for now)

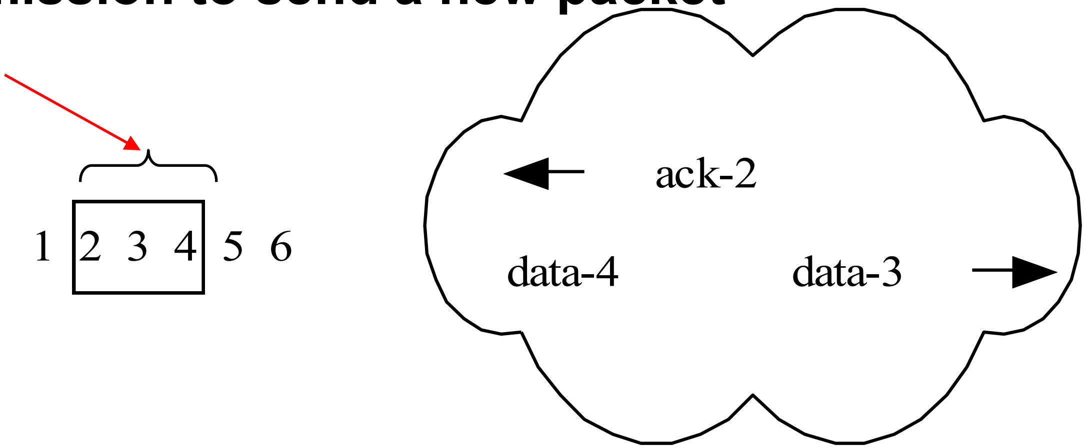
May not work well for wireless networks, why?

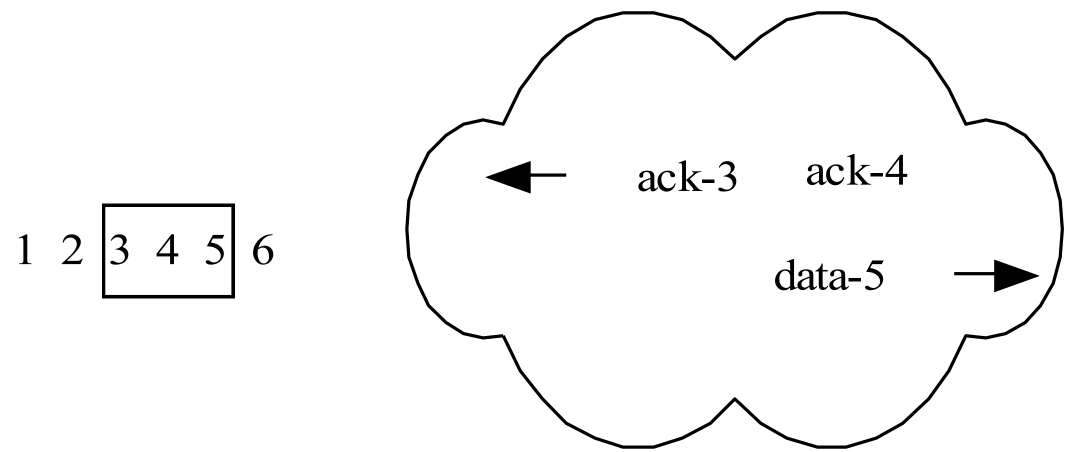
- Fading, echoes, .. → sporadic losses … → underutilized links with TCP

Sliding window:

each ACK = permission to send a new packet

- example cwnd = 3

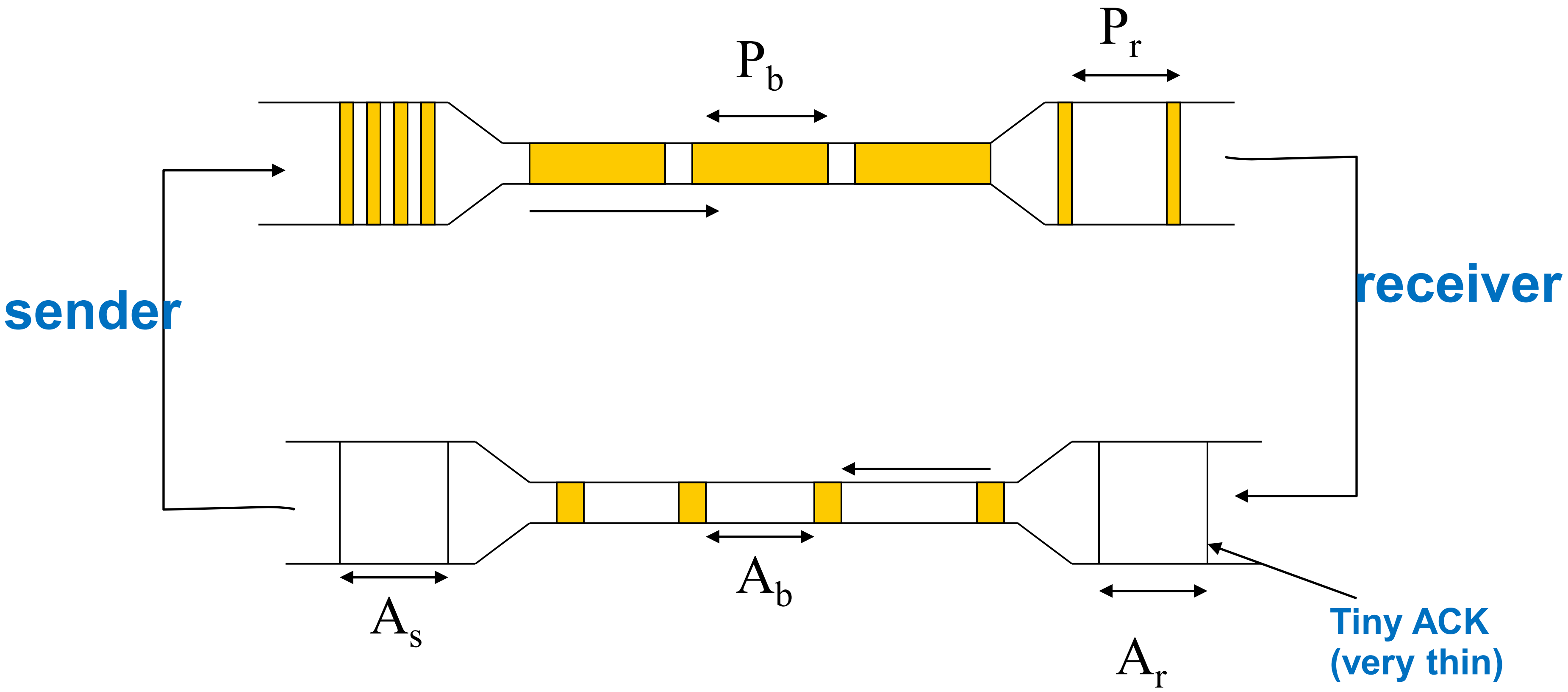




Self-clocking

If we have a large window,
ACKs "self-clock" the data to the rate of the bottleneck link

Observe: received ACK spacing ≅ bottleneck bandwidth



7.9 Basic Component 2: Rate Adjustment

Algorithm

- upon receipt of ACK (of new data):
  ➔ increase rate
  - data successfully delivered, perhaps can send faster

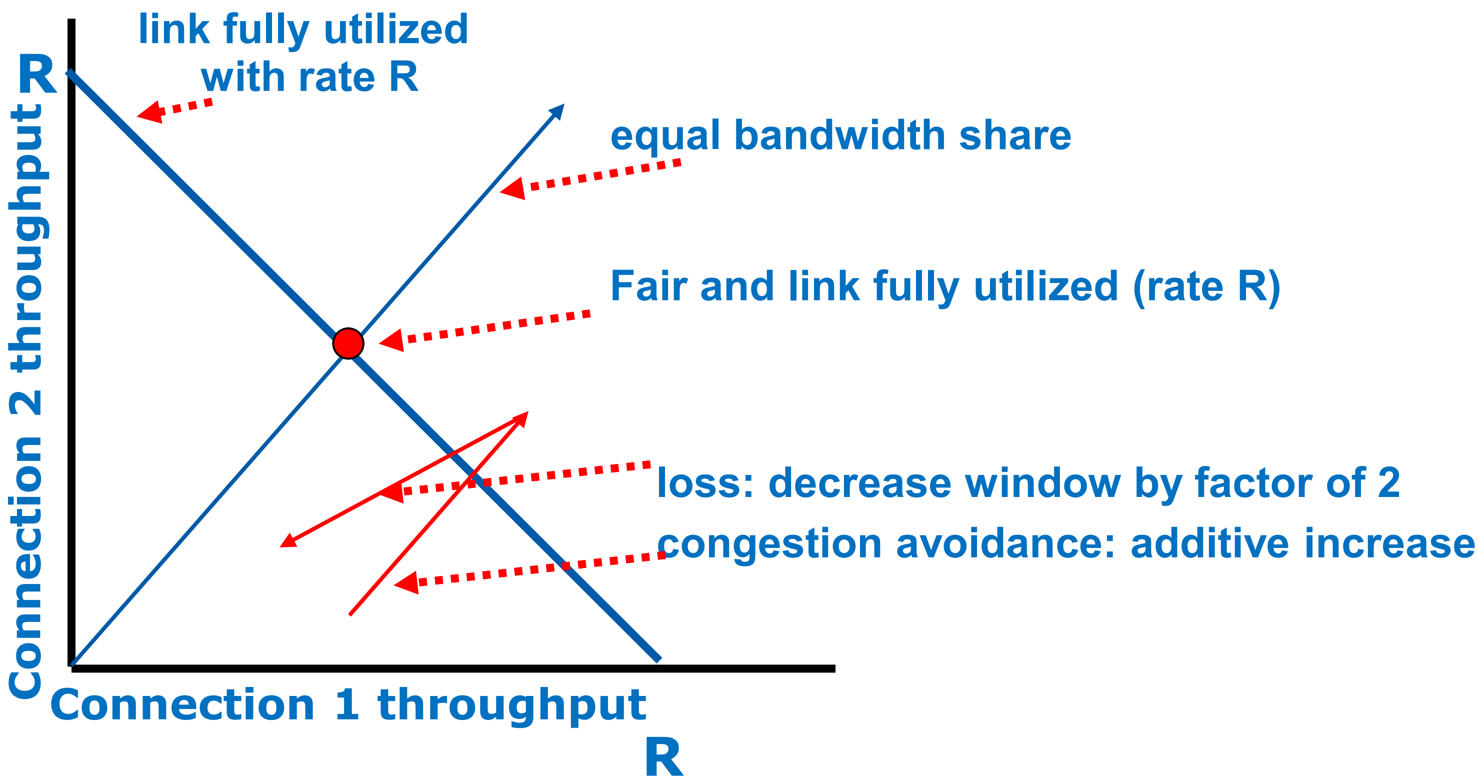
- upon detection of loss:
  ➔ decrease rate

But which increase/decrease functions should we use?

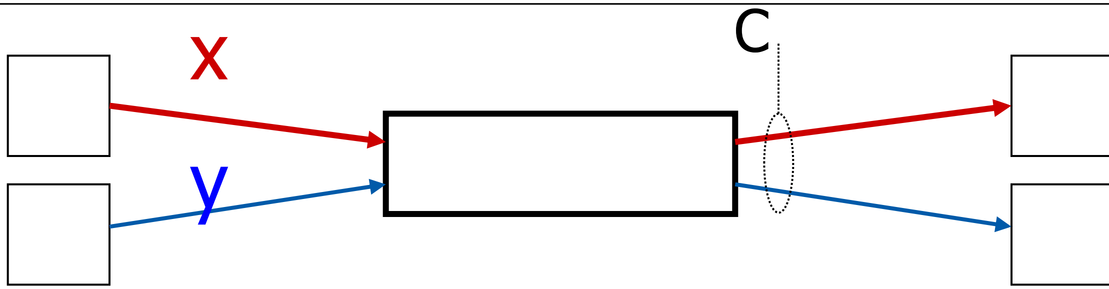
- …. depends on what problem we are solving

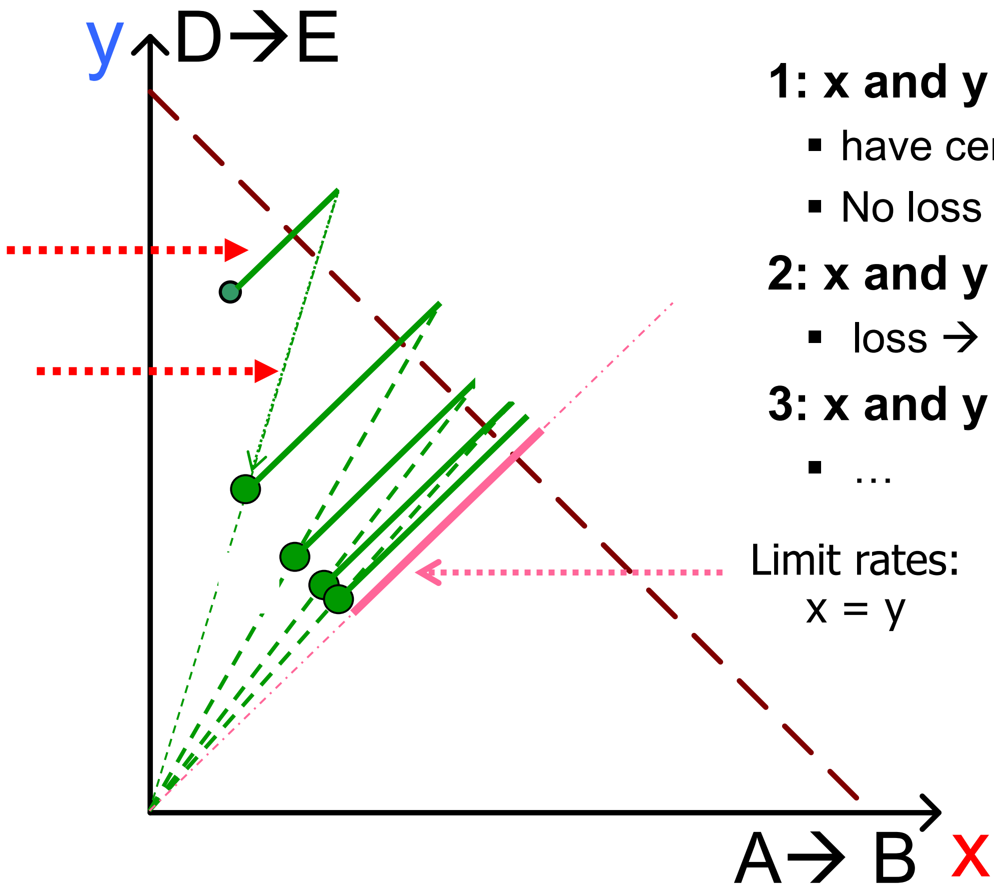
Two competing sessions:

- Additive increase (AI) gives slope of 1,
  - as throughput increases
- multiplicative decrease (MD) decreases throughput proportionally



8.1 Additive Increase Multiplicative Decrease (AIMD)





1: x and y

- have certain throughput
- No loss → x, y cwnd by +1 every RTT

2: x and y

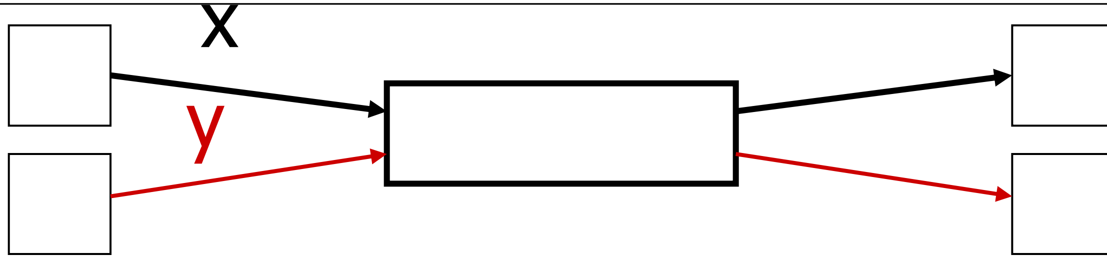
- loss → decrease by factor 2

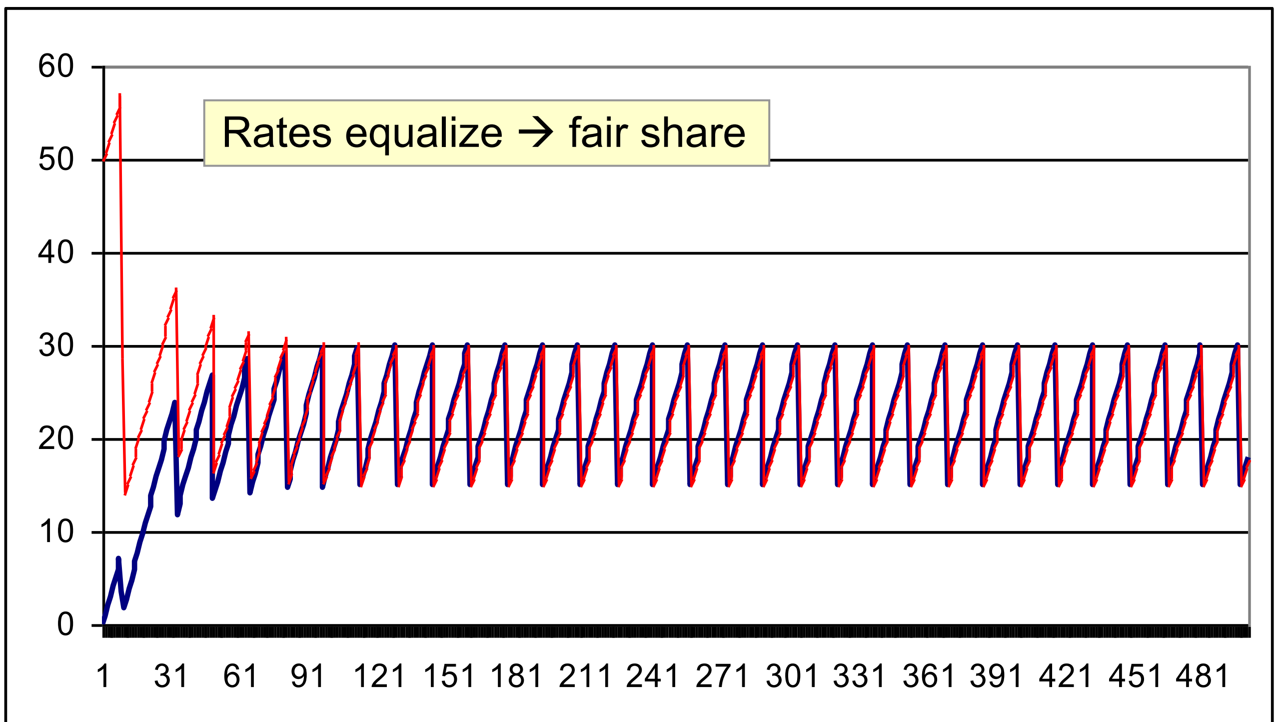
3: x and y

- …

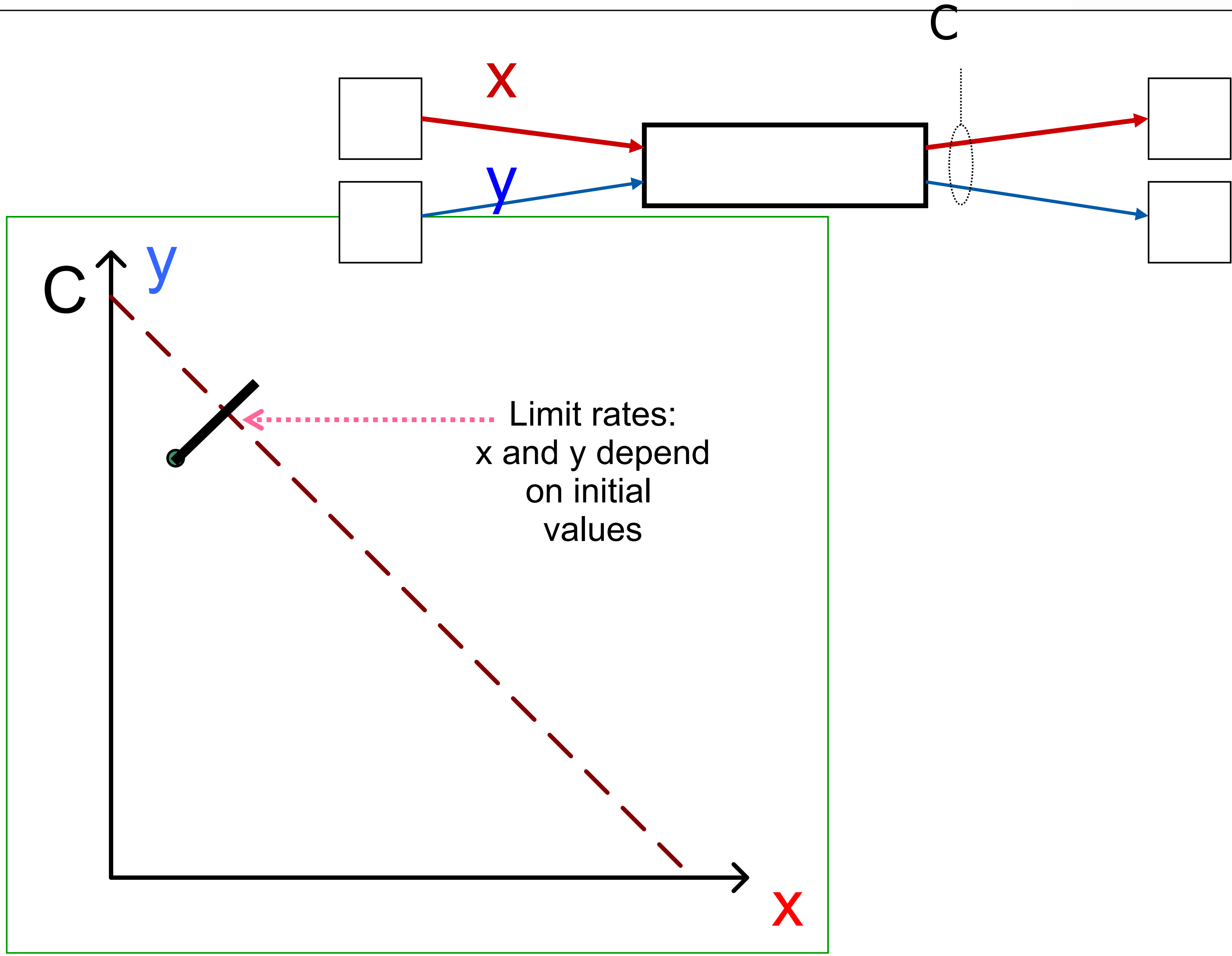
Limit rates:
x = y

Additive Increase Multiplicative Decrease: Sharing Dynamics

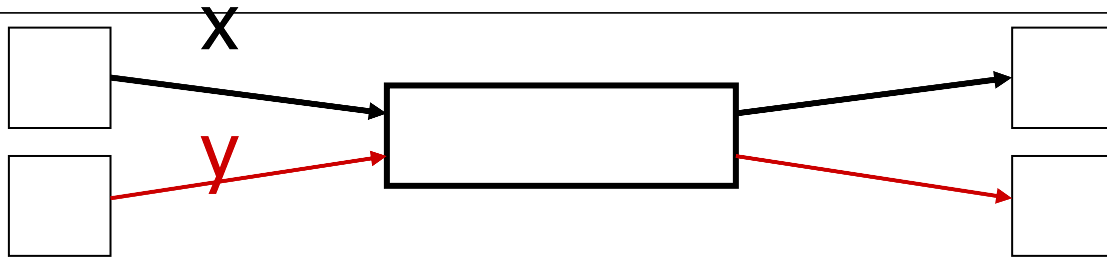


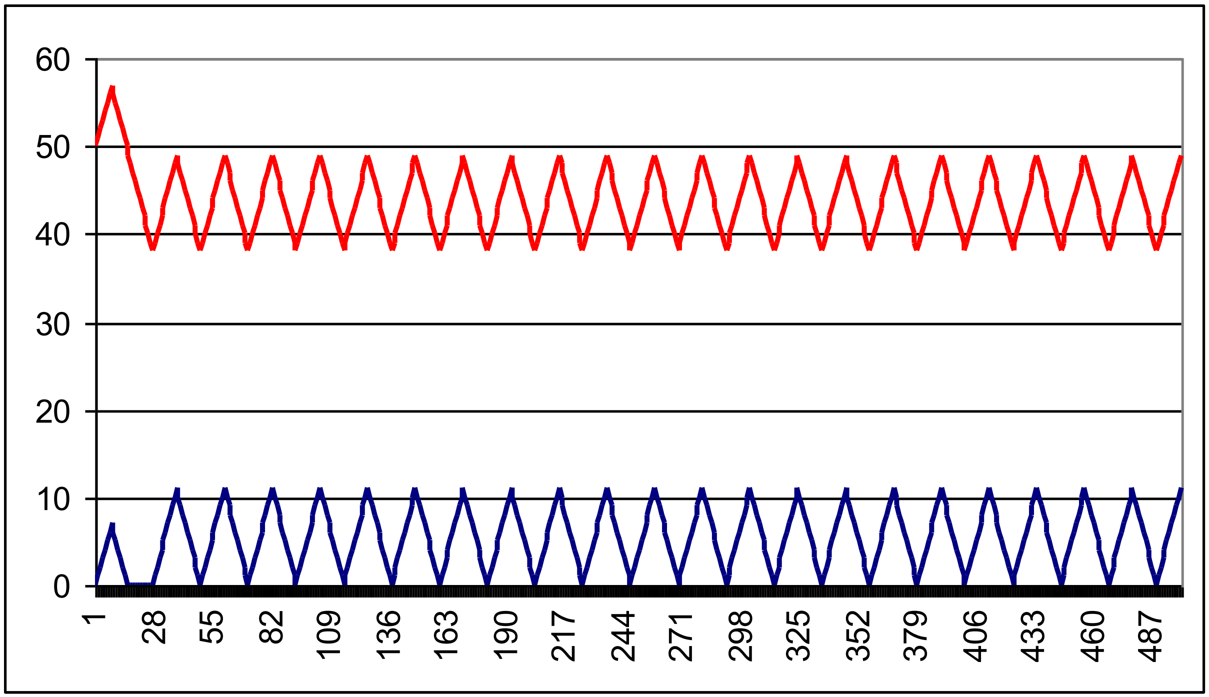


8.2 Additive Increase Additive Decrease (AIAD)



Additive Increase Additive Decrease: Sharing Dynamics





So far

- sliding window + self-clocking of ACKs

How to know

- the best congestion window cwnd ?
- and best transmission rate?

➔ Adapting Congestion Window cwnd

Phases of Congestion Control

Phase 1: Slow start (getting to equilibrium)

- want to find this extremely fast and wasting time

Phase 2: Congestion Avoidance

- additive increase
  - gradually probing for additional bandwidth
- multiplicative decrease
  - decreasing cwnd upon loss/timeout

TECHNISCHE UNIVERSITÄT DARMSTADT

Congestion Window (cwnd)

- Initial value is 1 MSS (=maximum segment size)
  - counted as bytes

Slow-start threshold Value (ss_thresh)

- Initial value is advertised window size

i.e. phase 1:

- slow start (cwnd < ss_thresh)

i.e. phase 2:

- congestion avoidance (cwnd >= ss_thresh)

9.2 Phase 1: TCP Slow Start

Goal:

- to discover roughly the proper sending rate quickly

Whenever

- starting traffic on a new connection, or whenever
- increasing traffic after congestion was experienced:

➔ initialize cwnd =1

each time a segment is acknowledged,

➔ increment cwnd by one (cwnd++)

Continue until

- reach ss_thresh
- packet loss

Slow Start Illustration

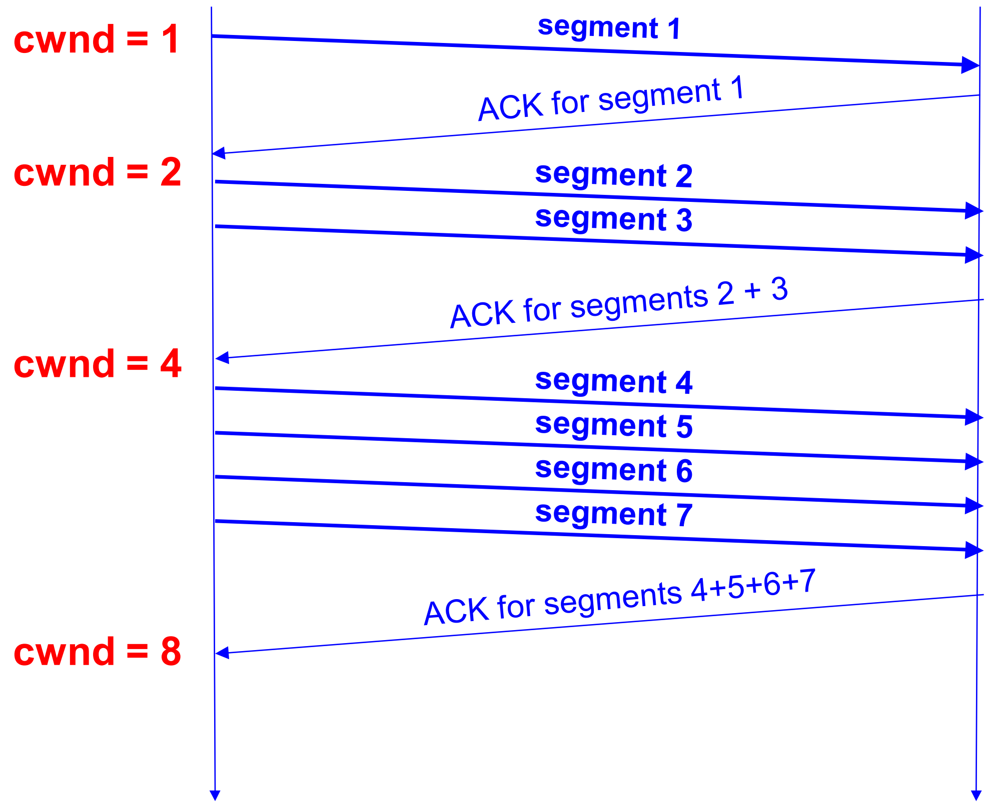
congestion window size grows very rapidly

TCP slows down the increase of cwnd

- when cwnd >= ss_thresh

Observe:

- each ACK generates 2 packets
- slow start increases rate exponentially
  - (doubled every RTT)



9.3 Phase 2: Congestion Avoidance (After Slow Start)

TECHNISCHE UNIVERSITÄT DARMSTADT

Slow Start

- roughly figures out rate at which the network starts to get congested

Congestion Avoidance

- continues to react to network condition
  - probes for more bandwidth
    - increase cwnd
      - if more bandwidth available
  - if congestion detected,
    - aggressive cut back cwnd

Phase 2: Congestion Avoidance (Additive Increase)

TECHNISCHE UNIVERSITÄT DARMSTADT

After exiting slow start

➔ slowly increase cwnd
to probe for additional available bandwidth

- competing flows may end transmission
- may have been “unlucky” with an early drop

If cwnd > ss_thresh then

- each time a segment is acknowledged
  - increment cwnd by 1/cwnd (cwnd += 1/cwnd).

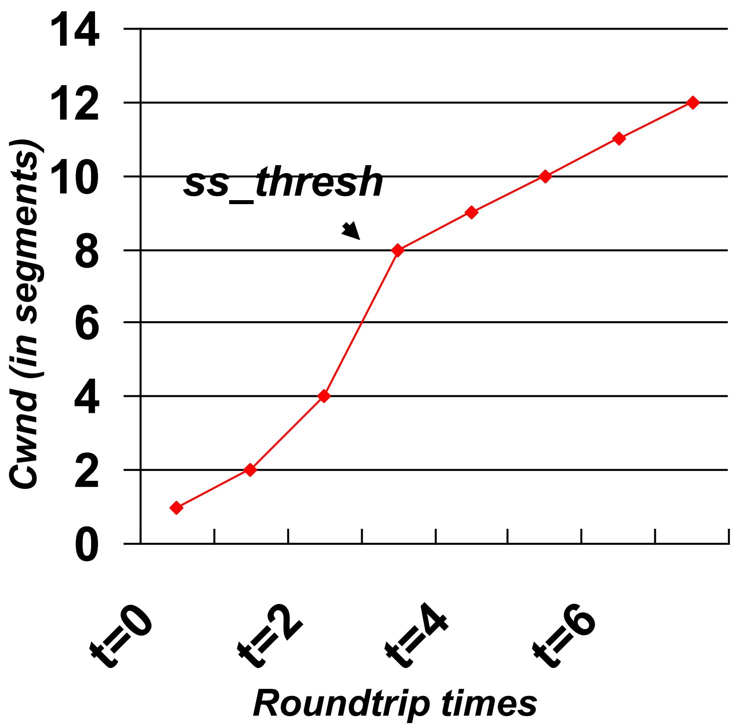
i.e. cwnd is increased by 1

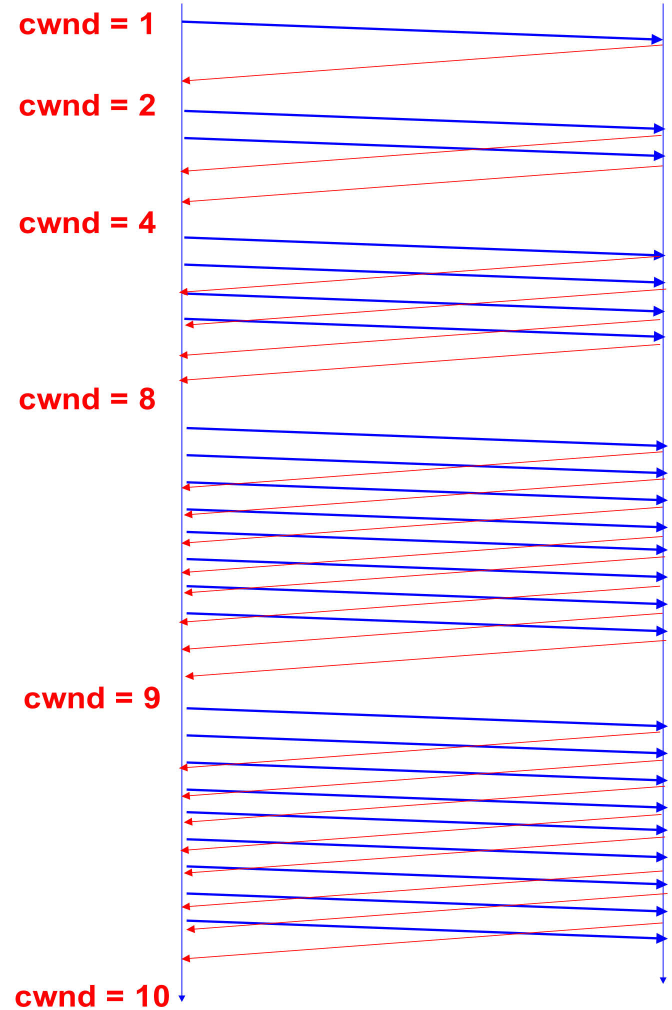
- only if all segments have been acknowledged

➔ increases by 1 per RTT, vs. doubling per RTT

9.4 Example of Slow Start + Congestion Avoidance

Assume that ss_thresh = 8





Detecting Congestion via Timeout

If there is a packet loss

- the ACK for that packet will not be received

packet will eventually time out

- no ACK is seen as a sign of congestion

Congestion Avoidance: Multiplicative Decrease

Timeout = congestion

Each time when congestion occurs,

- ss_thresh is set to 50%
  of the current size of the congestion window:
  - ss_thresh = cwnd / 2
- cwnd is reset to one:
  - cwnd = 1
- and
  slow-start is entered

9.5 TCP illustrated