Transport Protocols

Internet Transport Protocols

- 2 Internet transport layer protocols:
  - **TCP** (Transmission Control Protocol)
  - **UDP** (User Datagram Protocol)

- Comparison TCP ↔ UDP:

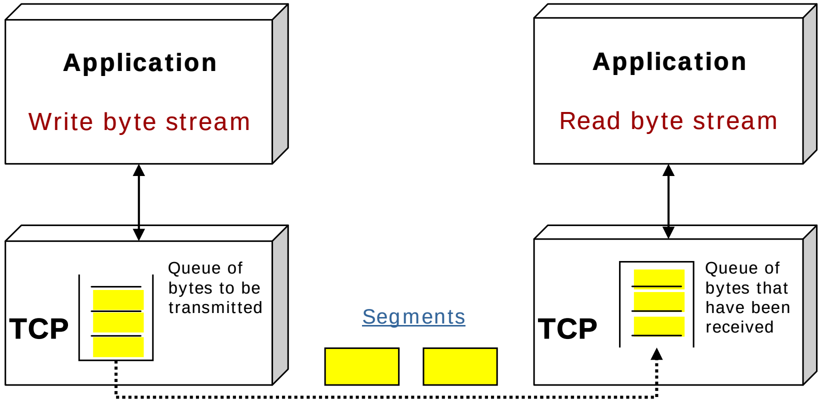
Criteria	TCP	UDP
connection oriented	Yes	No
reliable transport, error protection	Yes	No
overhead	Large	Small

TCP Protocol - Characteristics of TCP

- Connection oriented end-to-end transport protocol over IP
  - runs between end systems (3-way handshake for connection setup)
  - reliable data transport:
    - protection against lost, duplicated and in the wrong order received packets
    - error detection and correction
  - integrated mechanism for flow and congestion control
- Standardisation
  - original standard IETF RFC793 (September 1981)
  - revised standard RFC1122 (October 1989)
  - extensions:
    - extensions for long delay: RFC1072
    - big windows: RFC1106, RFC1110
    - selective acknowledgements (instead of Go back n): RFC 2018
    - increasing TCP's initial window: RFC2414
    - New Reno fast recovery: RFC2582

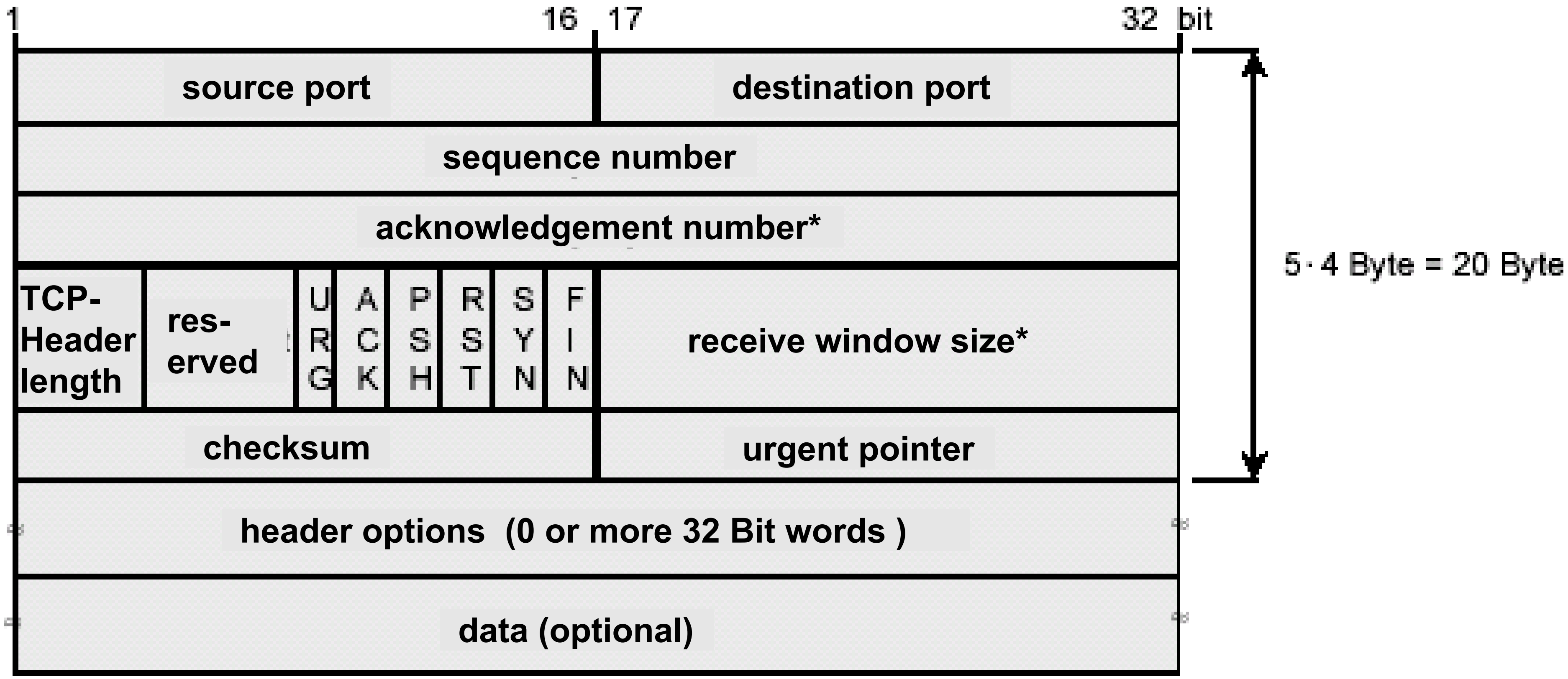
TCP Protocol - Characteristics of TCP

- (Byte-)oriented, buffered data transport
  - the application process provides a continuous byte stream to TCP for the duration of the connection
  - TCP collects enough bytes of the byte stream (in a buffer) and delivers these as a segment (with variable length) to the IP layer for transmission


Application
Write byte stream
Application
Read byte stream
TCP
Queue of bytes to be transmitted
Segments
TCP
Queue of bytes that have been received

- Full-Duplex operation
  - no separate establishment of the backward connection necessary
  - uses the piggyback principle for transmission of control information
- Negotiation of the max. segment size (MSS) between the sender and the receiver during connection setup (segment = TCP header + data)
  - MSS could be different for the forward and backward connection (because of possibly different forward/backward routes)
  - default value: 576 Bytes
  - determination of MSS e.g. via Path MTU Discovery (RFC1191)

TCP Protocol - TCP Segment Format



1

16

17

32 bit

source port

destination port

sequence number

acknowledgement number*

TCP-Header length

res-erved

URG

ACK

PSH

RST

SYN

FIN

receive window size*

checksum

urgent pointer

header options (0 or more 32 Bit words)

data (optional)

5 · 4 Byte = 20 Byte

*) field has an impact on the transmission behaviour of the receiver (of the TCP segment)

TCP Protocol - TCP Segment Header Fields

- Source/Destination Port:
  - (local) addresses of the end points of the TCP connection
- Sequence Number:
  - number of the first byte of the data of the TCP segment
  - assigned by the sender
  - initialized during connection setup
- Acknowledgement Number:
  - number of the byte which is expected to be send next by the sender (ack. of the correct reception of bytes with smaller sequence number)
- TCP Header Length:
  - length of the TCP header in 32 bit words (at least 5)
  - with this offset the start of the data field could be determined
- Reserved:
  - reserved for future use
  - all bits are set to "0"

TCP Protocol - TCP Segment Header Fields

- Flags:
  - used for controlling the connection setup and teardown as well as for acknowledgements (Piggybacking)
  - meaning of the flags (if set to "1"):
    - URG: the urgent pointer is used (and points to the last byte of valid data within the byte stream)
    - ACK: the acknowledgement number (in the acknowledgement number field) is valid
    - PSH: the data of this segment (and all previously sent data) should be forwarded to the application process as soon as possible (without waiting for any further data)
    - RST: request of a connection reset (e.g. in case of an error)
    - SYN: request for synchronization of the sequence numbers during connection setup
    - FIN: the sender indicates that all data of this connection is transmitted and no further data will be transmitted (one-way tear down of the connection)

TCP Protocol - TCP Segment Header Fields

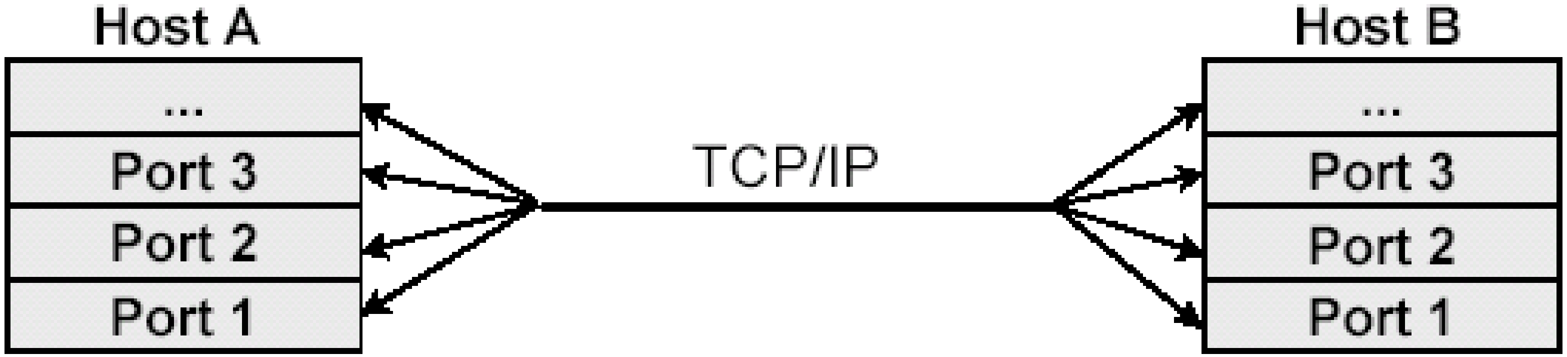
- Receiver Window Size:
  - denotes how many bytes (beginning with the current Ack number) a sender is accepting from its communication partner in backward direction
  - used for flow control
- Checksum:
  - calculated over the TCP header field, TCP data field and TCP pseudo header
  - **TCP pseudo header**: consists of parts of the IP header (IP Source/ Destination Address, Protocol, TCP Length) → violation of the OSI principle of layer separation!
- Urgent Pointer:
  - is only meaningful if the URG-flag is set
  - for “urgent” data transmission beyond the order of the current byte stream
  - the pointer is used to mark the “urgent” data in the data field of the TCP segment
  - usage example: software interrupt in a telnet session

TCP Protocol - TCP Segment Header Fields

- Options:
  - for additional functions (e.g. selective repeat instead of Go back n)
  - no or multiple options possible
  - option types:
    - one-octet-options
    - multiple-octet-options:
      - TLV-Format (type (1), length (1), value) where:
        - length: number of octets of type+length+value
        - type: type of the option
  - option examples:
    - "end of option list": type = 0, one-octet-option
    - "no operation": type = 1, one-octet-option
    - "maximum segment size": type = 2, multiple-octet-option (length = 4); used in SYN segments to transmit the MSS

TCP Protocol - Port Concept

- A port defines the access to TCP (and UDP) from the next higher layer (e.g. the application)
- For each application one or more ports are uniquely allocated - thus a port uniquely identifies an application



- Port numbers:

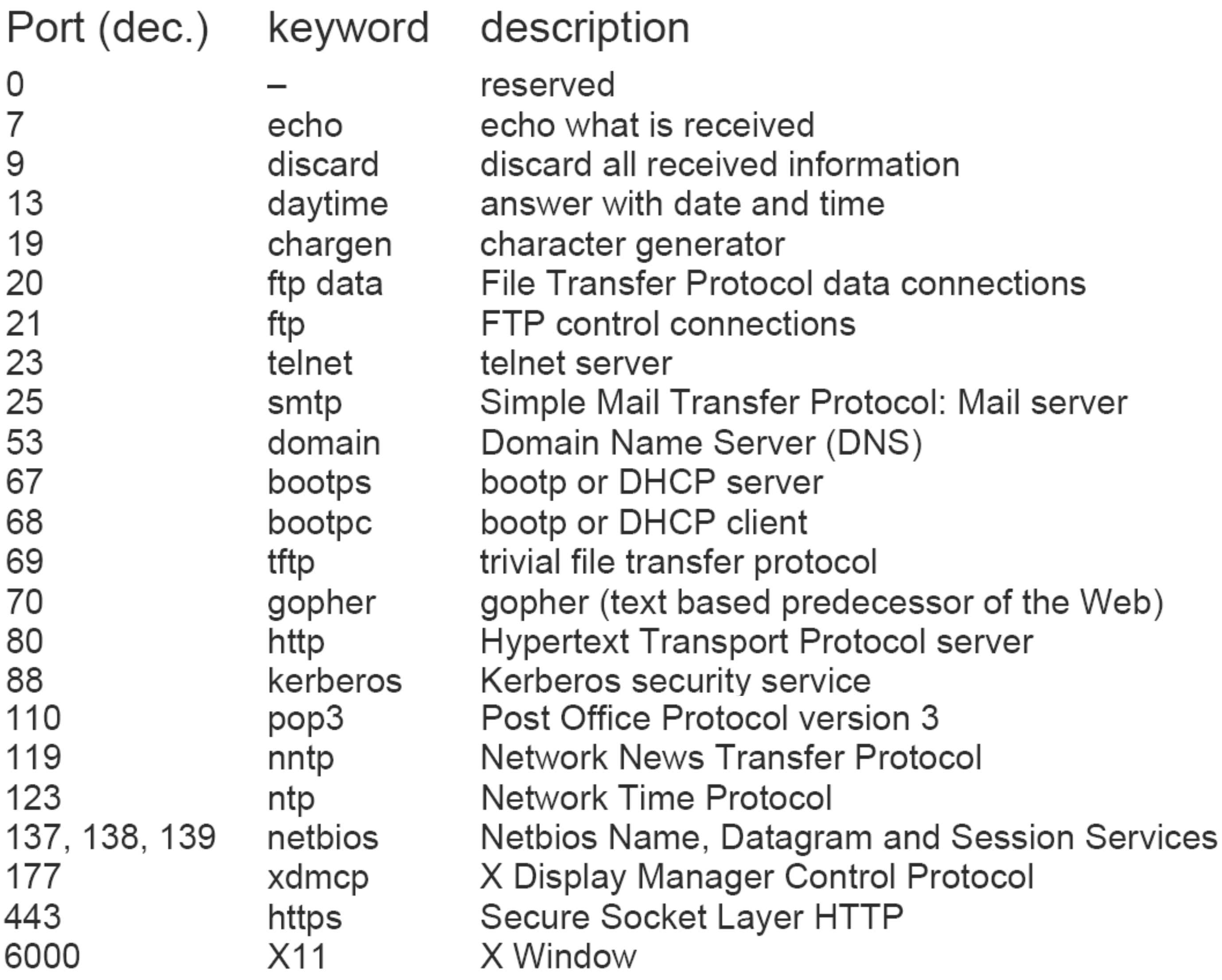
0 - 255: reserved for public (standardized) applications

256 - 1023: assigned to commercial applications

> 1023: not regulated, can be used freely

TCP Protocol - Port Concept

- **Well-known port numbers** for public applications (RFC1700)

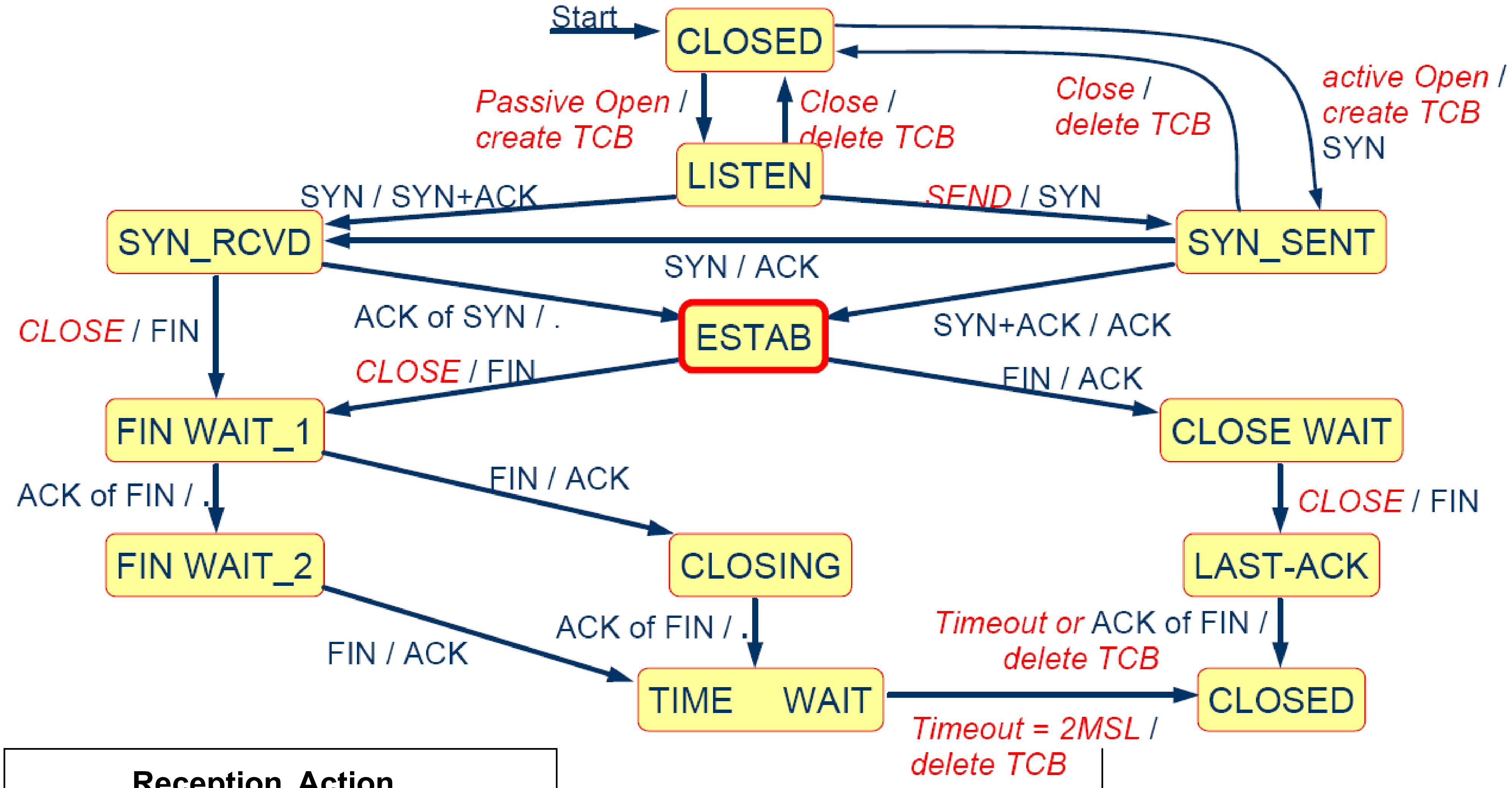


Port (dec.)	keyword	description
0	–	reserved
7	echo	echo what is received
9	discard	discard all received information
13	daytime	answer with date and time
19	chargen	character generator
20	ftp data	File Transfer Protocol data connections
21	ftp	FTP control connections
23	telnet	telnet server
25	smtp	Simple Mail Transfer Protocol: Mail server
53	domain	Domain Name Server (DNS)
67	bootps	bootp or DHCP server
68	bootpc	bootp or DHCP client
69	tftp	trivial file transfer protocol
70	gopher	gopher (text based predecessor of the Web)
80	http	Hypertext Transport Protocol server
88	kerberos	Kerberos security service
110	pop3	Post Office Protocol version 3
119	nntp	Network News Transfer Protocol
123	ntp	Network Time Protocol
137, 138, 139	netbios	Netbios Name, Datagram and Session Services
177	xdmcp	X Display Manager Control Protocol
443	https	Secure Socket Layer HTTP
6000	X11	X Window

TCP Protocol - Connection Concept

- TCP connections exist at layer 4 between end systems
  - the IP network layer works connectionless and is not aware of the TCP connection
- No explicit TCP connection identifier, instead implicit connection identification via the 5-tuple:
  - IP protocol number (6 = TCP)
  - source IP address
  - source port
  - destination IP address
  - destination port
- **TCP Control Block (TCB)** contains
  - 5-tuple
  - additional information (sequence number, timer values, …)

TCP Protocol - State Machine for TCP Connections



Start

CLOSED

Passive Open / *create TCB*

Close / *delete TCB*

Close / *delete TCB*

active Open / *create TCB* SYN

LISTEN

SYN / SYN+ACK

SEND / SYN

SYN_RCVD

SYN_SENT

SYN / ACK

CLOSE / FIN

ACK of SYN / .

ESTAB

SYN+ACK / ACK

CLOSE / FIN

FIN / ACK

FIN WAIT_1

CLOSE WAIT

ACK of FIN / .

FIN / ACK

CLOSE / FIN

FIN WAIT_2

CLOSING

LAST-ACK

ACK of FIN / .

Timeout or ACK of FIN / *delete TCB*

FIN / ACK

TIME WAIT

CLOSED

Timeout = 2MSL / *delete TCB*

Maximum Segment Lifetime
(RFC793: 2 minutes)

Reception Action

↓ ↓

abc receive / execute local action

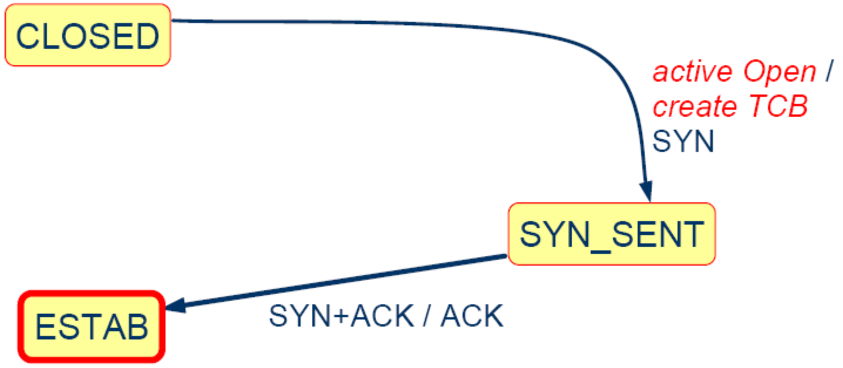
FIN receive / send packet

TCP Protocol - State Machine for TCP Connections

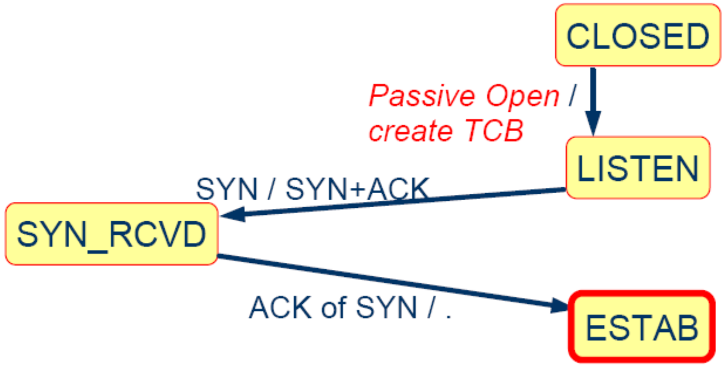
- Remarks:
  - all transmissions are a subject to an implicit time control → if an ACK is not received until the timer expires the TCP segment is retransmitted
  - the reception of a TCP segment with a RST flag set to “1” results in an immediate abortion of the transmission → immediate transition to CLOSED state
  - it is impossible to leave the CLOSED state through the reception of any kind of TCP segments
  - in the ESTAB state all TCP segments have to contain an acknowledgement number
  - the timer at connection tear down is used to distinguish old connections (with lost ACKs) from new connections; timer value = 2 MSL (maximum segment lifetime) ≈ 4 minutes

TCP Protocol - TCP Connection Setup

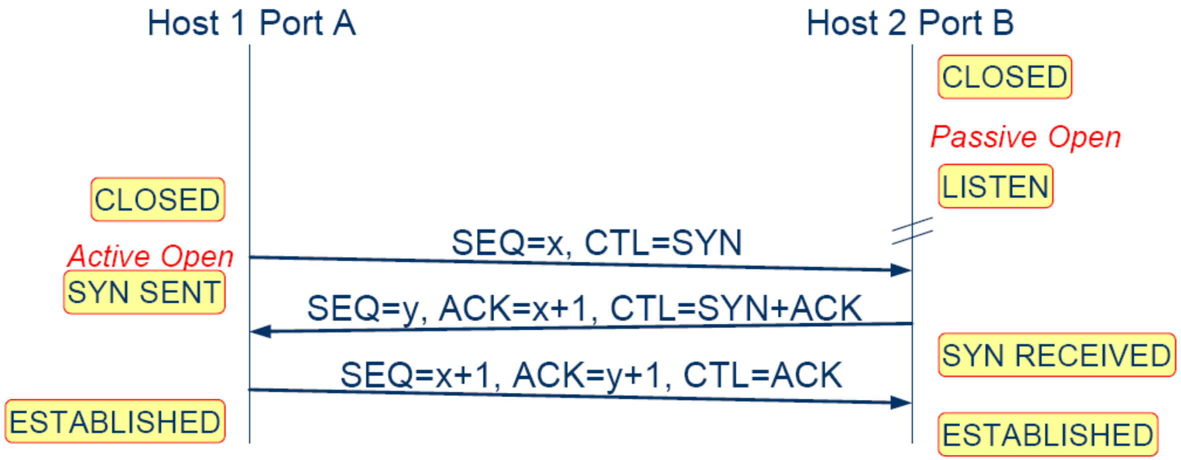
- Active Open:



- Passive Open:

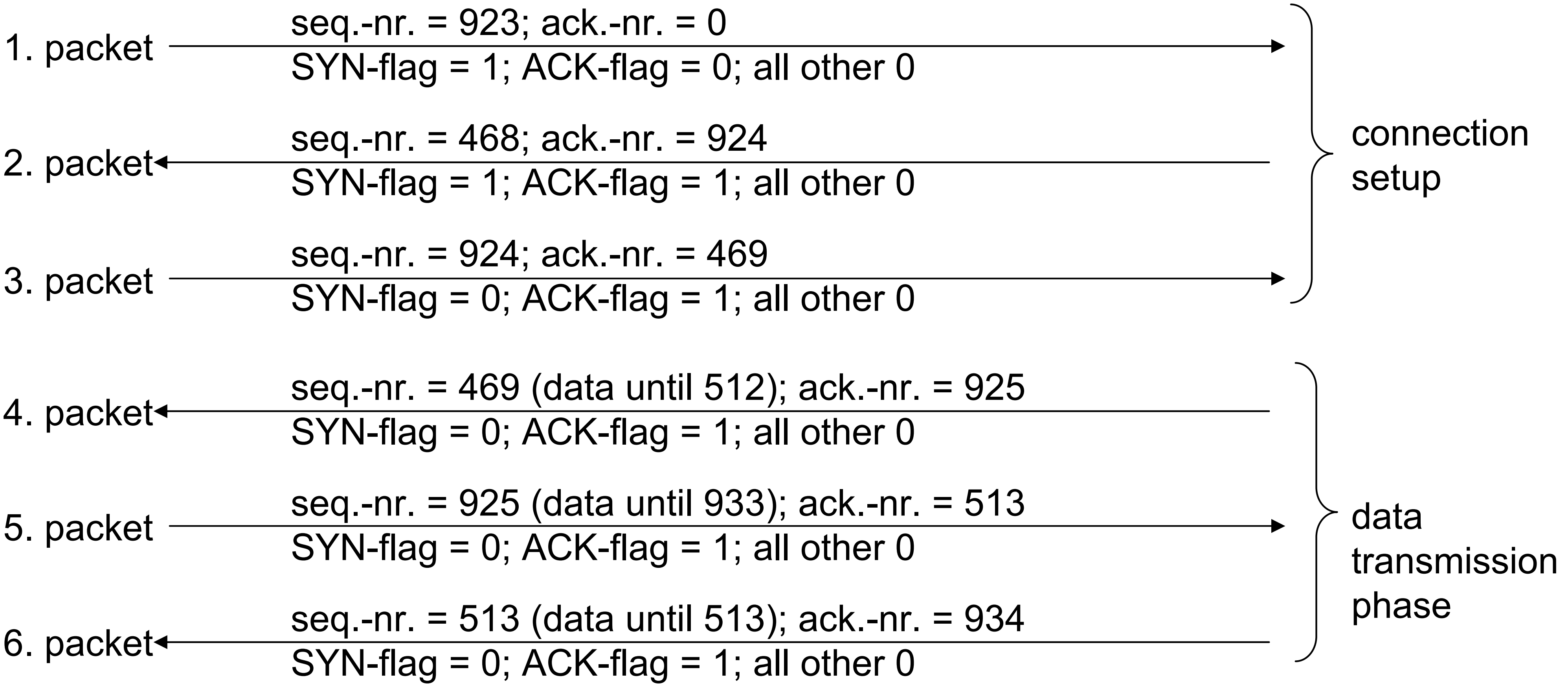


- Message flow during TCP connection set up (**"3-Way Handshake"**): example: Active Open of host 1 after Passive Open of host 2



TCP Protocol - TCP Connection Setup

- TCP connection setup + subsequent transmission phase (example):



1. packet → seq.-nr. = 923; ack.-nr. = 0 / SYN-flag = 1; ACK-flag = 0; all other 0

2. packet ← seq.-nr. = 468; ack.-nr. = 924 / SYN-flag = 1; ACK-flag = 1; all other 0

3. packet → seq.-nr. = 924; ack.-nr. = 469 / SYN-flag = 0; ACK-flag = 1; all other 0

connection setup

4. packet ← seq.-nr. = 469 (data until 512); ack.-nr. = 925 / SYN-flag = 0; ACK-flag = 1; all other 0

5. packet → seq.-nr. = 925 (data until 933); ack.-nr. = 513 / SYN-flag = 0; ACK-flag = 1; all other 0

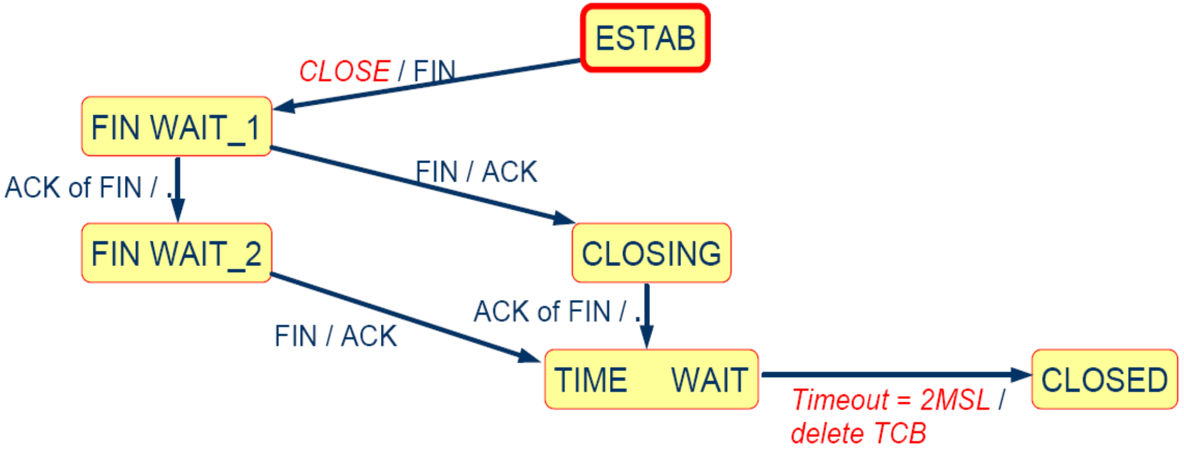
6. packet ← seq.-nr. = 513 (data until 513); ack.-nr. = 934 / SYN-flag = 0; ACK-flag = 1; all other 0

data transmission phase

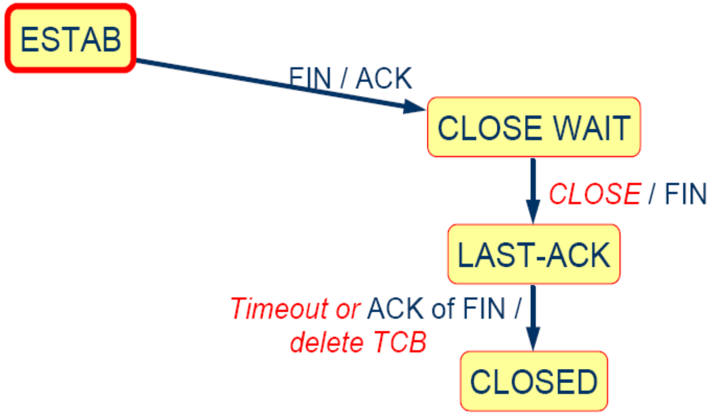
- during the connection set up and data transmission the ack.-nr. is always the number of the last transmitted byte incremented by one
- if no data is transmitted the ack.-nr. stays constant
- the seq.-nr. is the number of the first byte of the transmitted data

TCP Protocol - TCP Connection Teardown

- Active Close:

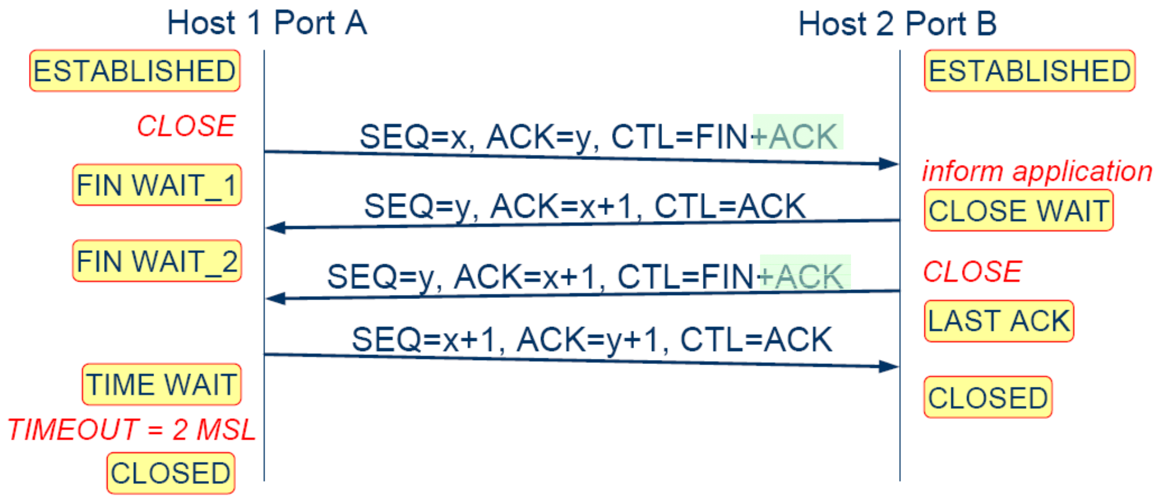


- Close through remote station:



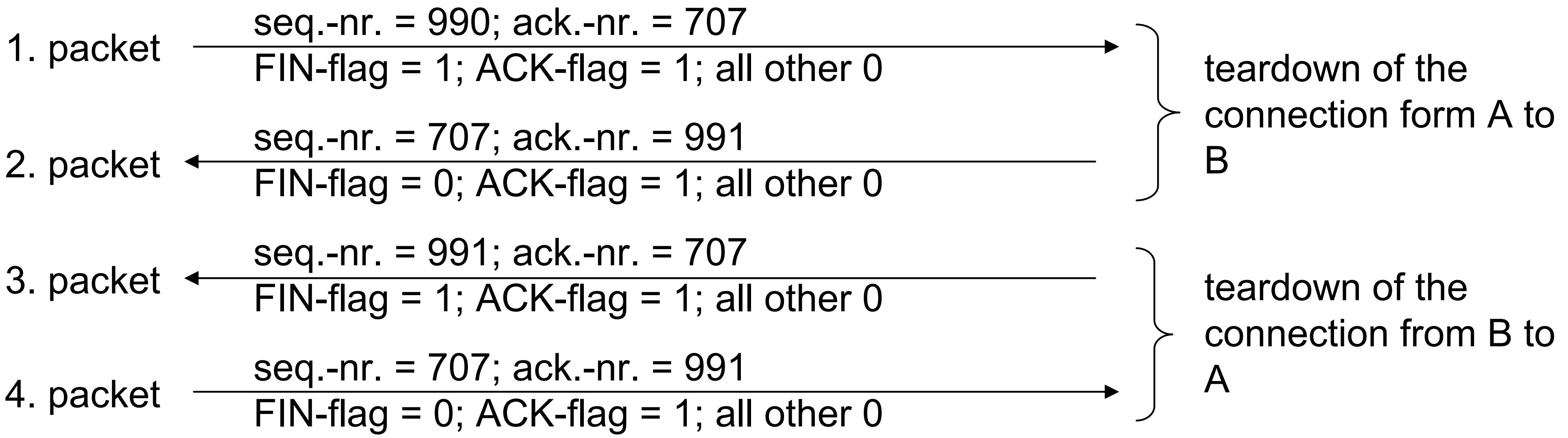
- Message flow during TCP connection teardown:

example: Active Close of host 1 - graceful close



TCP Protocol - TCP Connection Teardown

- TCP connection teardown (example):
  - variant "**Graceful Close**": normal connection teardown of A→B and B→A (example: termination of the TCP data transmission)



1. packet → seq.-nr. = 990; ack.-nr. = 707 FIN-flag = 1; ACK-flag = 1; all other 0

2. packet ← seq.-nr. = 707; ack.-nr. = 991 FIN-flag = 0; ACK-flag = 1; all other 0

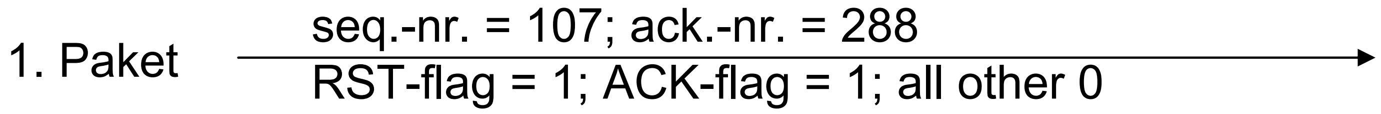
teardown of the connection form A to B

3. packet ← seq.-nr. = 991; ack.-nr. = 707 FIN-flag = 1; ACK-flag = 1; all other 0

4. packet → seq.-nr. = 707; ack.-nr. = 991 FIN-flag = 0; ACK-flag = 1; all other 0

teardown of the connection from B to A

  - variant "**Abort**": immediate abortion of the connection via reset (RST-flag = 1) without acknowledgement of the remote station (example: FTP abortion)



1. Paket → seq.-nr. = 107; ack.-nr. = 288 RST-flag = 1; ACK-flag = 1; all other 0

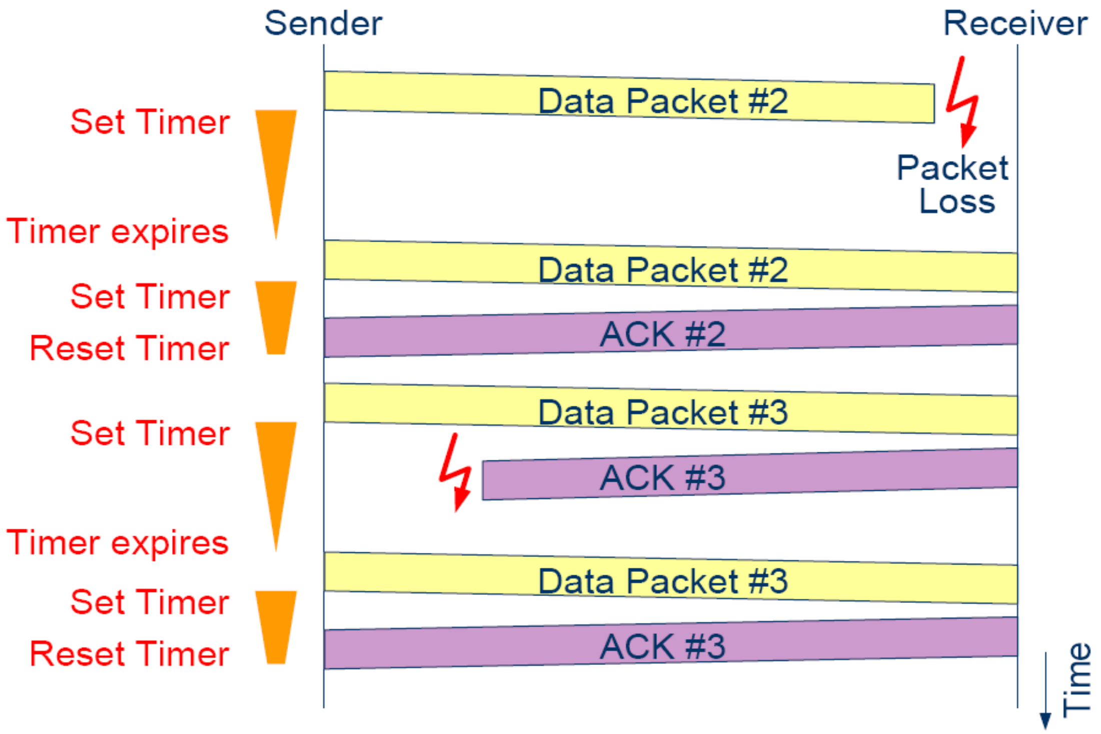
no acknowledgement is send from the remote station - it is simply assumed that the remote station has received the connection abortion request

TCP Protocol - Reliable Data Transmission

- Mechanisms
  - **positive acknowledgements:**
    - protection against packet loss (of user data)
    - receiver transmits positive acks in backward direction if the data was received correctly (requires duplex connection)
  - **time monitoring (retransmission timer):**
    - protection against packet loss (of user data)
    - sender starts timer after transmission of a TCP segment
    - retransmission of the segment if no positive ack is received until timeout
  - **numbering of user data (seq.-nr.) and acknowledgements (ack.-nr.)**:
    - protection against packet loss (user data/acknowledgements), duplicate segments and wrong packet order (Caution: the numbering refers to bytes and not to TCP segments!)

TCP Protocol - Reliable Data Transmission

- Principle: positive acknowledgements and time monitoring



Timer setting:
- sufficiently long to account for delays in the network and in the remote host (avoid unnecessary retransmissions)
- sufficiently short, for fast reaction to packet loss
- → adaptive timer required, since the packet delay is variable

TCP Protocol - Reliable Data Transmission

- Adaptive retransmission timer
  - adaptive adjustment to varying packet delay in the network
  - basic principle: continuous estimation of the round trip time (RTT) (= time between transmission of a TCP segment and reception of the acknowledgement) via RTT probes
  - calculation of the retransmission timeout value (old method):
    - determination of the mean RTT value (EWMA): $RTT_{avg} = \alpha \cdot RTT_{old} + (1- \alpha) \cdot RTT_{newSample}$

      the smoothing factor $\alpha \in (0,1)$ determines the reaction rate:

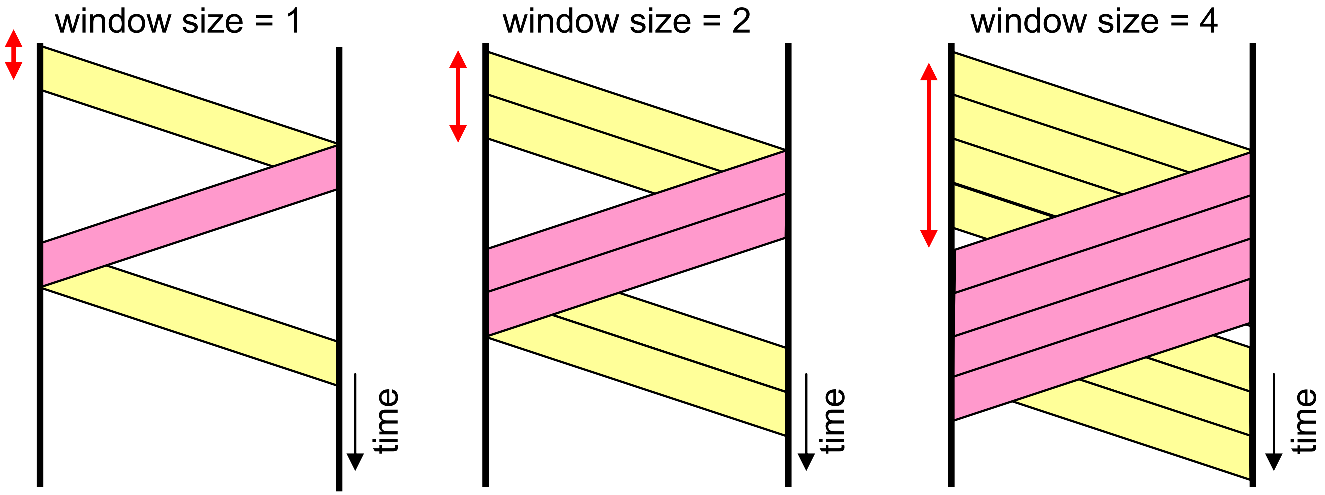
      $\alpha \rightarrow 0$: fast reaction, $\alpha \rightarrow 1$: slow reaction (RTT_{avg} stable)
    - retransmission timeout value = $\beta \cdot RTT_{avg}$, where:

      for $\beta=1$: no delay tolerance, fast detection of packet losses

      for $\beta>1$: less retransmissions, slow detection of packet losses
  - problem of this old method: a high variance in the delays (high load) will result in many unnecessary retransmissions → new method which includes the observed delay variance into the calculation formula of the retransmission timeout value
  - calculation of retransmission timeout value (new method RFC1122, 1989):
    - collection of RTT probes x_i
    - determination of the RTT mean value: $x = x_{i-1} + \delta \cdot \Delta x$ whereat $\Delta x = x_i - x_{i-1}$
    - determination of the RTT variance: $\sigma_i = \sigma_{i-1} + \rho \cdot (|\Delta x| - \sigma_{i-1})$
    - retransmission timeout value = $x + \eta \cdot \sigma_i$

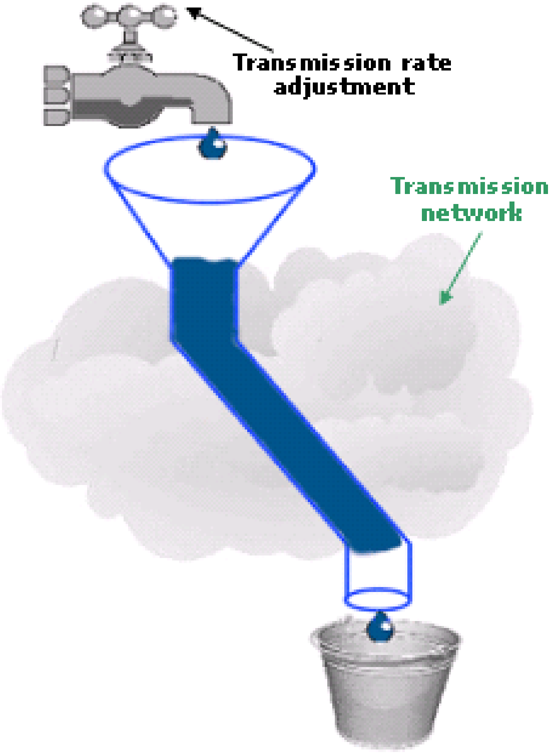
TCP Protocol - Efficient Data Transmission

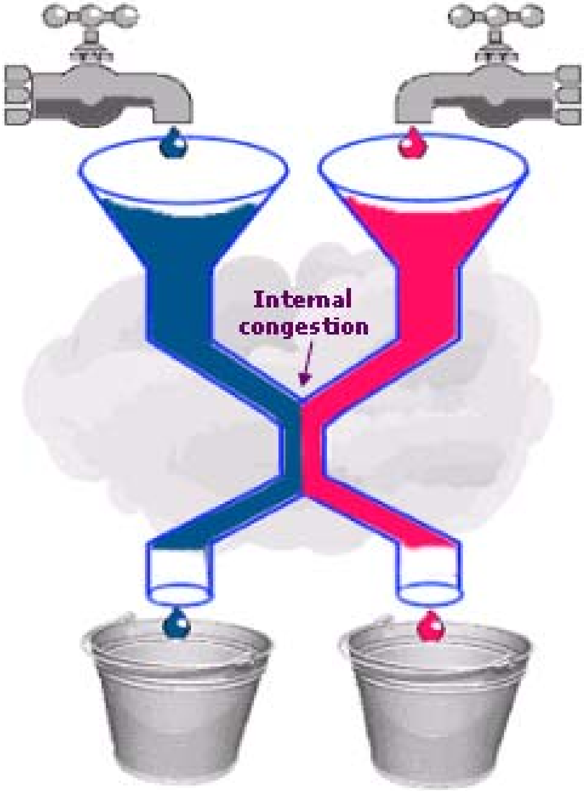
- Mechanisms
  - **sliding window mechanism:**
    - also used for flow and congestion control
    - window size = number of bytes a sender is allowed to transmit without receiving an acknowledgement
    - principle:



  - **cumulative acknowledgements:**
    - instead of acknowledging every single byte, all error free (and contiguously) received bytes are acknowledged at once
  - **piggyback method:**
    - the acks are sent within user data segments and not separately

Overload Control - Flow Control vs. Congestion Control





flow control (end system related)

- adaption of the transmission rate of the sender to the rate of the receiver
- avoids packet loss (caused by buffer overflow) at the receiver side

congestion control (network related)

- fair share of the transmission rates of all TCP connections at a (network internal) bottleneck
- avoids packet loss (caused by buffer overflow) inside the network

TCP Protocol - Flow Control

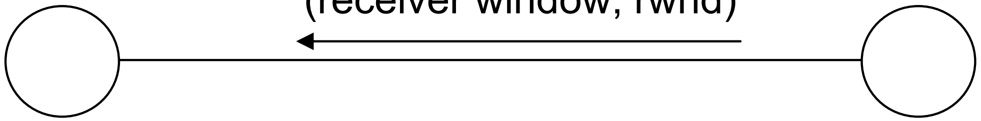
- Basic principle of flow control:
  - the dynamically adjustable receive window size is used by TCP for flow control between sender and receiver; the receiver tells the sender the current windows size (if needed) and by that controls the transmission rate of the sender
  - examples:
    - setting the windows size to 0 means that the receiver currently doesn’t want to receive any data
    - to continue the transmission the receiver sends a TCP segment (in backward direction to the sender) with the same ack.-nr and a window size ≠ 0

TCP Protocol - Flow Control

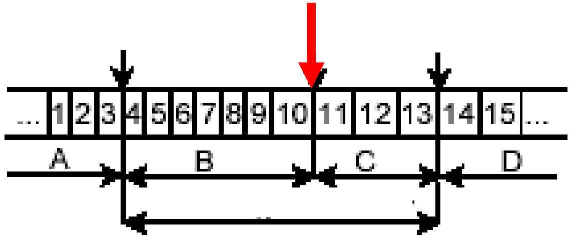
- Operation of flow control:

current receive window size
(receiver window, rwnd)

Sender

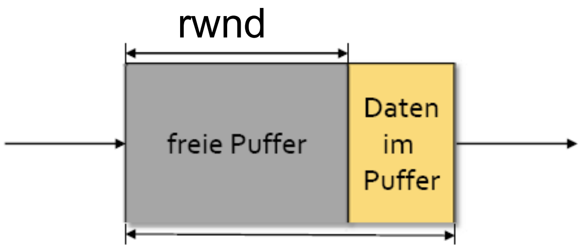


Receiver



A: transmitted and acknowledged bytes
B: transmitted but not yet acknowledged bytes
C: bytes that can be transmitted without receiving an prior acknowledgement
D: bytes that can not be transmitted yet

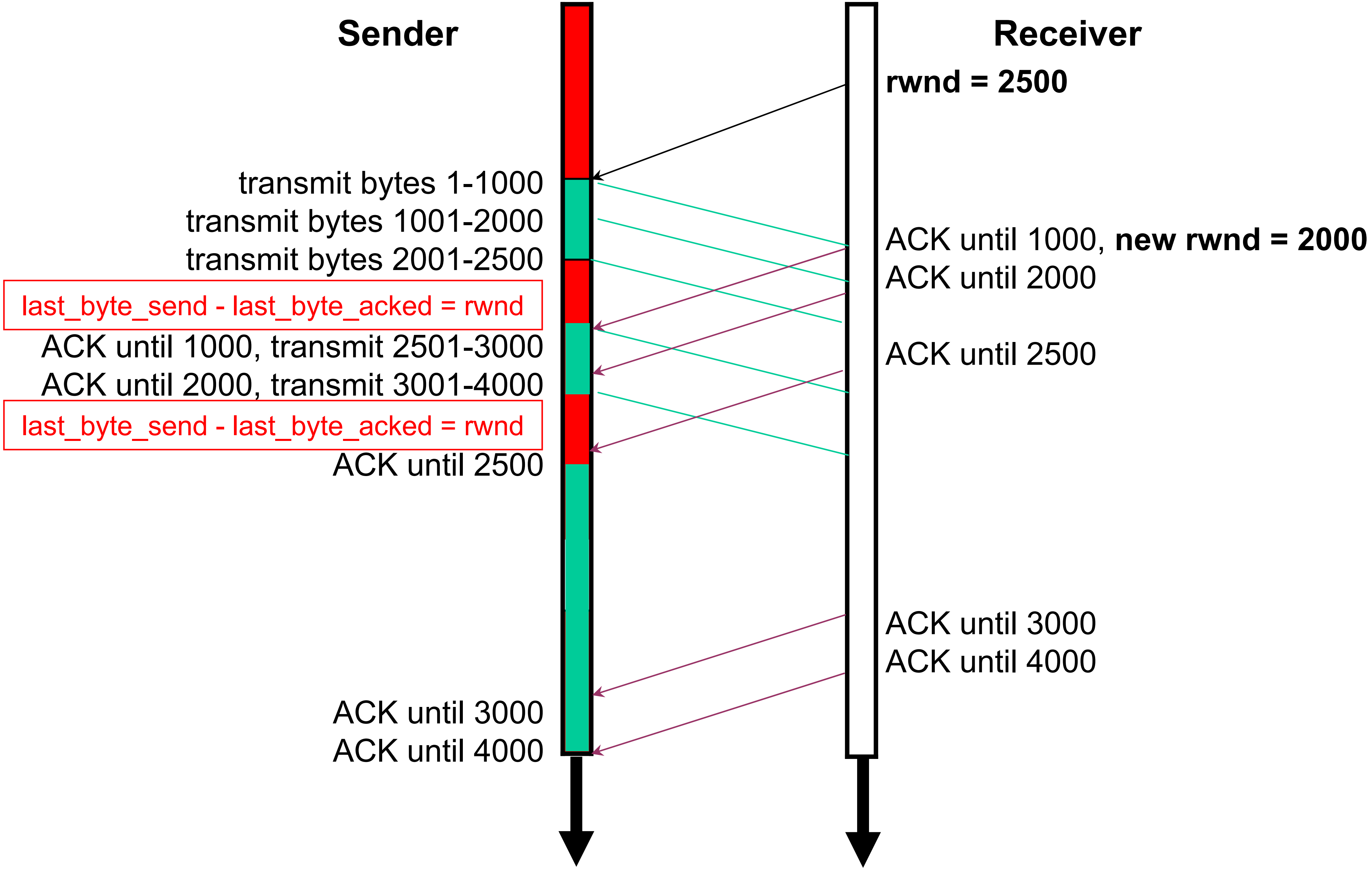
(transmit) window size = B+C = rwnd



receiver buffer (rbuf)

- at the receiver side:
  - the receive window size (rwnd) indicates how much buffer space is currently available for data received via this TCP connection
  - it holds: **rwnd = rbuf - (last_byte_received - last_byte_read)**
- at the sender side:
  - the sender may not transmit more than **rwnd unacknowledged data bytes**
  - it holds: **last_byte_sent - last_byte_acknowledged ≤ rwnd**

TCP Protocol - Flow Control (Example)



Sender

Receiver

rwnd = 2500

transmit bytes 1-1000

transmit bytes 1001-2000

transmit bytes 2001-2500

last_byte_send - last_byte_acked = rwnd

ACK until 1000, **new rwnd = 2000**

ACK until 2000

ACK until 1000, transmit 2501-3000

ACK until 2000, transmit 3001-4000

ACK until 2500

last_byte_send - last_byte_acked = rwnd

ACK until 2500

ACK until 3000

ACK until 4000

ACK until 3000

ACK until 4000

TCP Protocol - Congestion Control and dynamic Behavior

- Basic principle of congestion control:
  - increase of the transmission rate until a congestion occurs or until the maximum transmission rate is reached
  - reduction of the transmission rate in case of packet loss
- TCP operates with 2 window sizes:
  - **receiver window (*rwnd*)**
    - controlled by the receiver - for flow control
  - **congestion window (*cwnd*)**
    - depends on the congestion situation in the network - for congestion contr.
- **actual allowed window size = min(*rwnd*, *cwnd*)**
- the adjustment of the congestion window is performed in 2 phases:
  - **slow start phase**
  - **congestion avoidance phase**
- the boundary between slow start and congestion avoidance phase is dynamic; it is denoted as **slow start threshold (*sst*)**

TCP Protocol - Congestion Control and dynamic Behavior

- Operation of congestion control:
  - **start:** congestion window (*cwnd*) = 1 MSS (max. segment size)
  - as long as *cwnd* ≤ *sst* (and acks are received before timout):

    slow start phase:

    - for each acknowledged TCP segment: set ***cwnd* = *cwnd* + 1**
      → doubling of the *cwnd* per round trip (exponential increase)
  - if *cwnd* > *sst* and acks are received before timeout (i.e. no congestion):

    congestion avoidance phase:

    - for each acknowledged TCP segment: set ***cwnd* = *cwnd* + 1/*cwnd***
      → linear increase of *cwnd* by 1 per round trip of *cwnd* bytes

    in case of a congestion during the congestion avoidance phase:

    - in case of light congestion (3 duplicate ACKs):

      sst = *cwnd/2* and *cwnd* = *cwnd*/2; continuation with congestion avoidance
    - in case of heavy congestion (timeout):

      sst = *cwnd/2* and *cwnd* = 1 MSS; continuation with slow start

    if *cwnd* ≥ *rwnd*: set *cwnd* = *rwnd*

TCP Protocol - Congestion Control and dynamic Behavior

Slow Start and Congestion Avoidance – behavior without congestion (packet loss)

Starting point:
CWND= 32
Timeout

SST = 32 / 2 = 16
CWND = 1
Time 0: 1 segment is sent

Time 1: ACK is returned
and *CWND*= 2

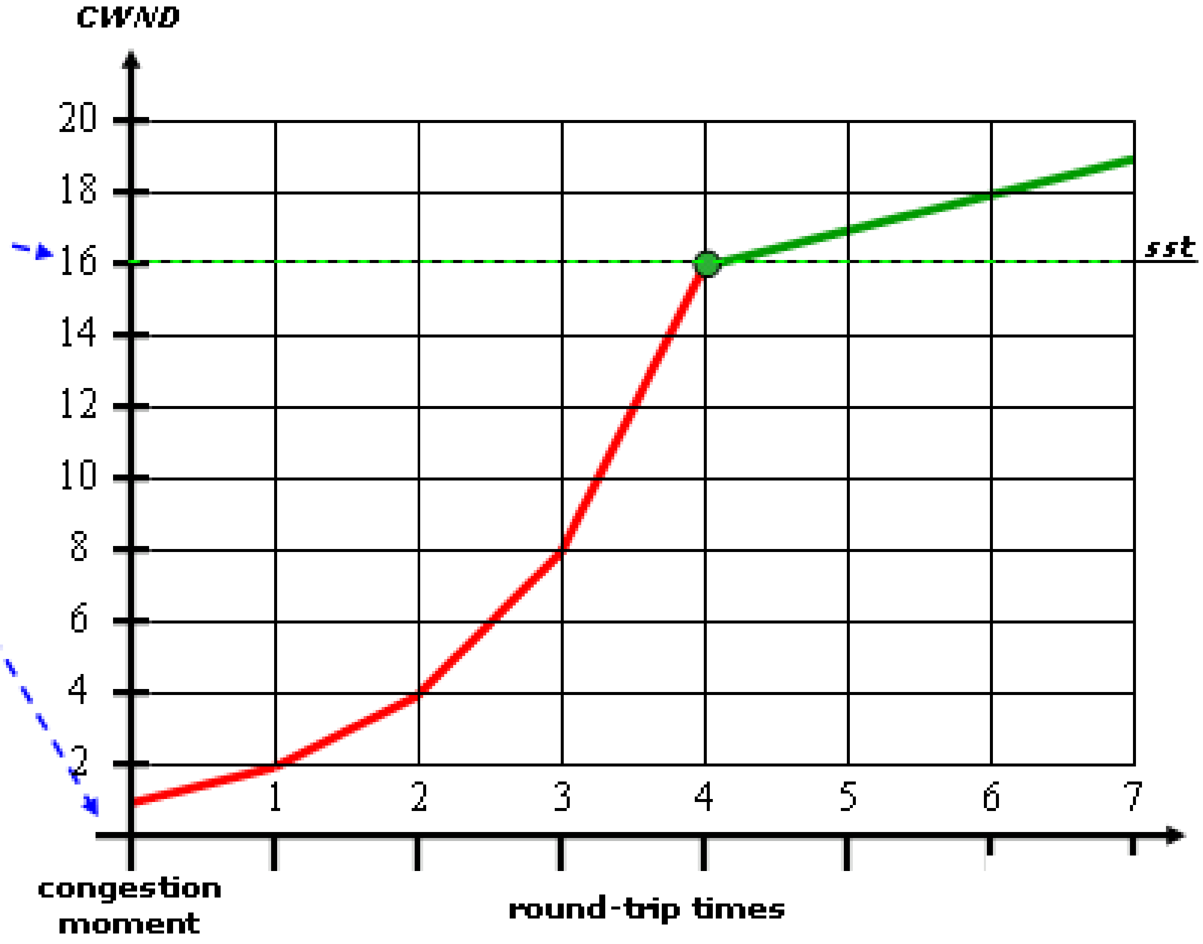
Time 2: two ACKs are returned
and *CWND* =2+2= 4

And so on...

CWND = *SST*. Slow start is stopped
and congestion avoidance is started

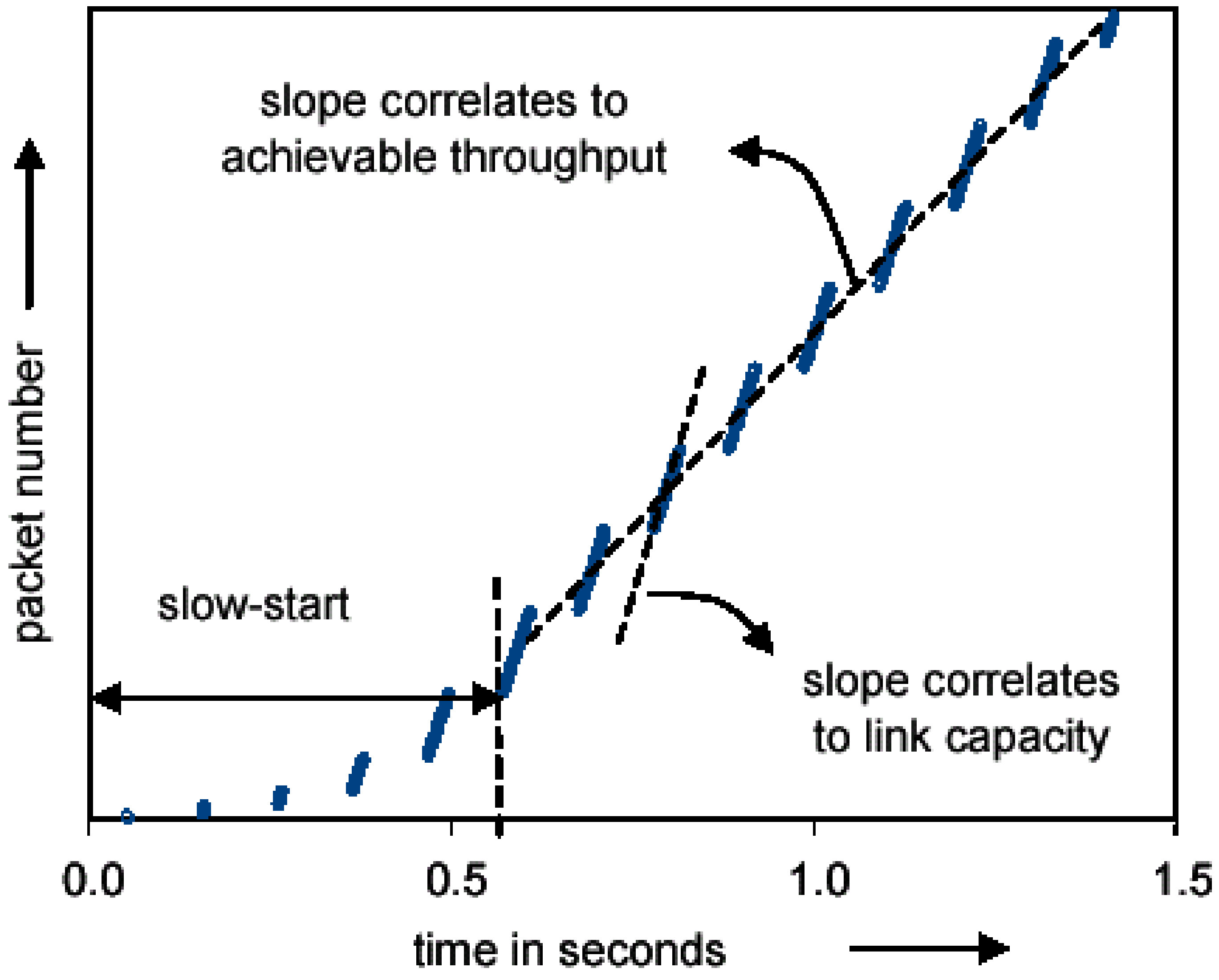
Increasing of *CWND* is linear:

one segment per round-trip time



TCP Protocol - Congestion Control and dynamic Behavior

Slow Start und Congestion Avoidance – behavior with congestion (packet loss)



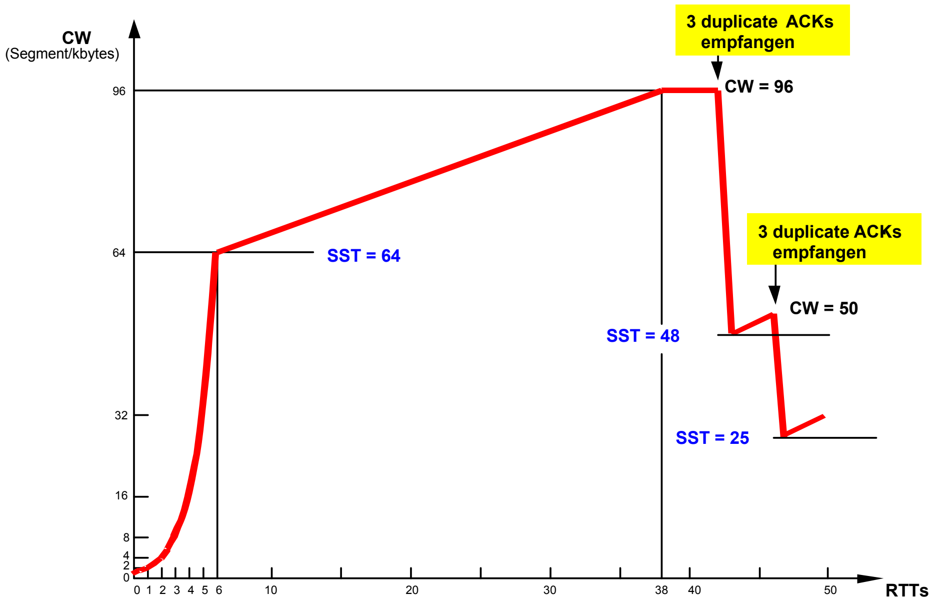
TCP Protocol - Congestion Control and dynamic Behavior

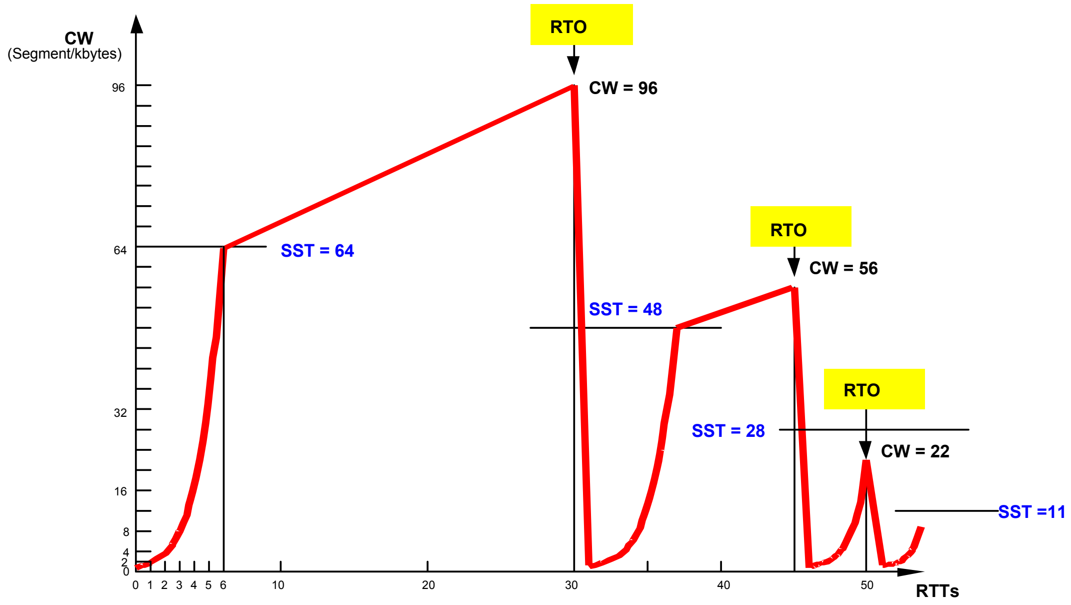
- **Congestion Avoidance without congestion (without packet loss):**
  - starting value for slow start threshold (*sst*): 64 Kbyte
  - if *cwnd* > *sst* and acks are received before timeout (i.e. no congestion):
    - increase *cwnd* linearly by 1 MSS per round trip (i.e. for all acknowledged TCP segments transmitted within the window):

      $cwnd_{new} = cwnd_{old}$ + 1 MSS **(additive increase)**
- **Congestion Avoidance with congestion (with packet loss):**
  - if (3) duplicate ACKs occur (i.e. moderate short term congestion):
    - set *sst* = *cwnd/2*
    - halve *cwnd*: $cwnd_{new} = cwnd_{old}/2$ (+ 3 MSS) **(multiplicative decrease)**
    - continue with **cong. avoidance** according to the rule above (no cong.)
  - if a timeout occurs (i.e. long term congestion):
    - set *sst* = *cwnd/2*
    - set *cwnd* = 1 MSS **(re-initialization)**
    - start again with **slow start**

TCP Protocol - Congestion Control and dynamic Behavior

Slow Start and Congestion Avoidance – behaviour with congestion (packet loss)



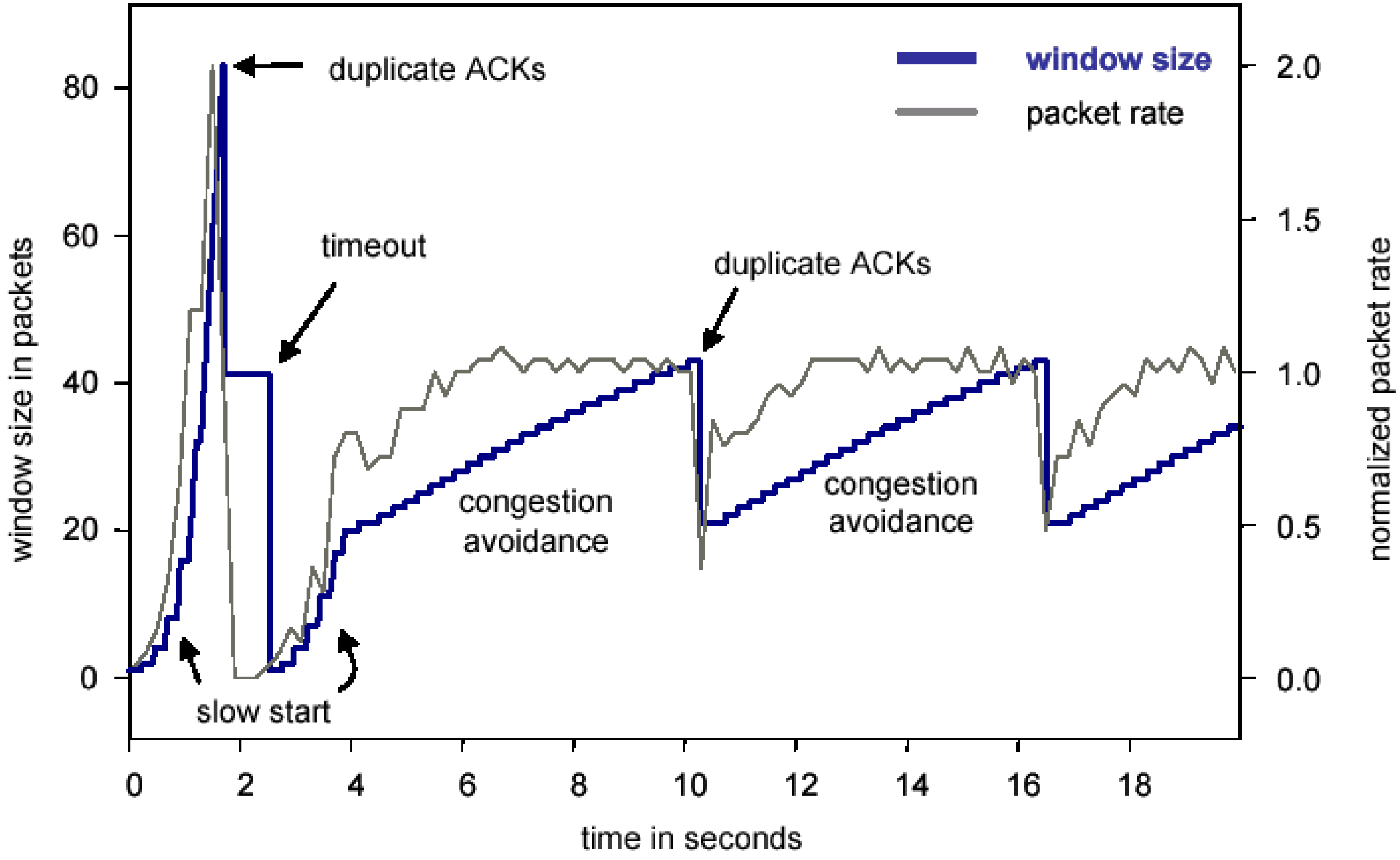


moderate congestion (→ duplicate ACKs)

heavy congestion (→ Timeout)

TCP Protocol - Congestion Control and dynamic Behavior

Slow Start and Congestion Avoidance – example trace



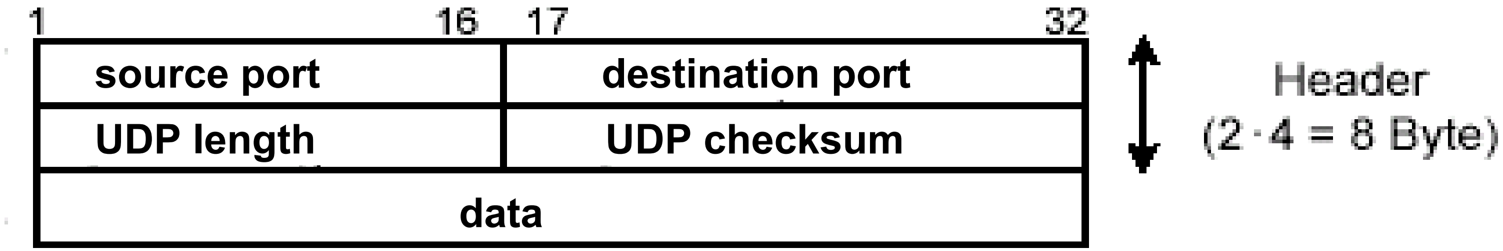
UDP Protocol

- **Connectionless** end to end transport protocol over IP
  - unreliable, connectionless data transport between a source and a destination
- Motivation:
  - many application don't need a reliable connection
  - avoiding the delays caused by TCP (important for real time services like Voice over IP)
  - avoiding the complexity and overhead of TCP (high efficiency)
  - TCP can not be used for multicast and broadcast connections
- Disadvantage:
  - applications need their own mechanisms to cope with erroneous, lost, duplicate and in the wrong order received packets
- Standardisation:
  - original standard IETF RFC768

UDP Protocol - Characteristics of UDP

- Unreliable, connectionless data transport service
  - no mechanisms to cope with erroneous, lost, duplicate and in the wrong order received packets
  - no error detection and correction
- Stateless
- Minimal additional protocol mechanisms (→ low complexity and small overhead)
  - source and destination applications (processes) are addressed via port numbers (source port optional)
  - (optional) error detection at the receiver side via a checksum over the UDP header, UDP data field and pseudo header (remark: the UDP pseudo header is defined similarly to the TCP pseudo header)
- Usage for multicast and broadcast connections possible

UDP Protocol - UDP Segment Format and Header Fields



1 … 16	17 … 32
source port	destination port
UDP length	UDP checksum
data	

Header (2 · 4 = 8 Byte)

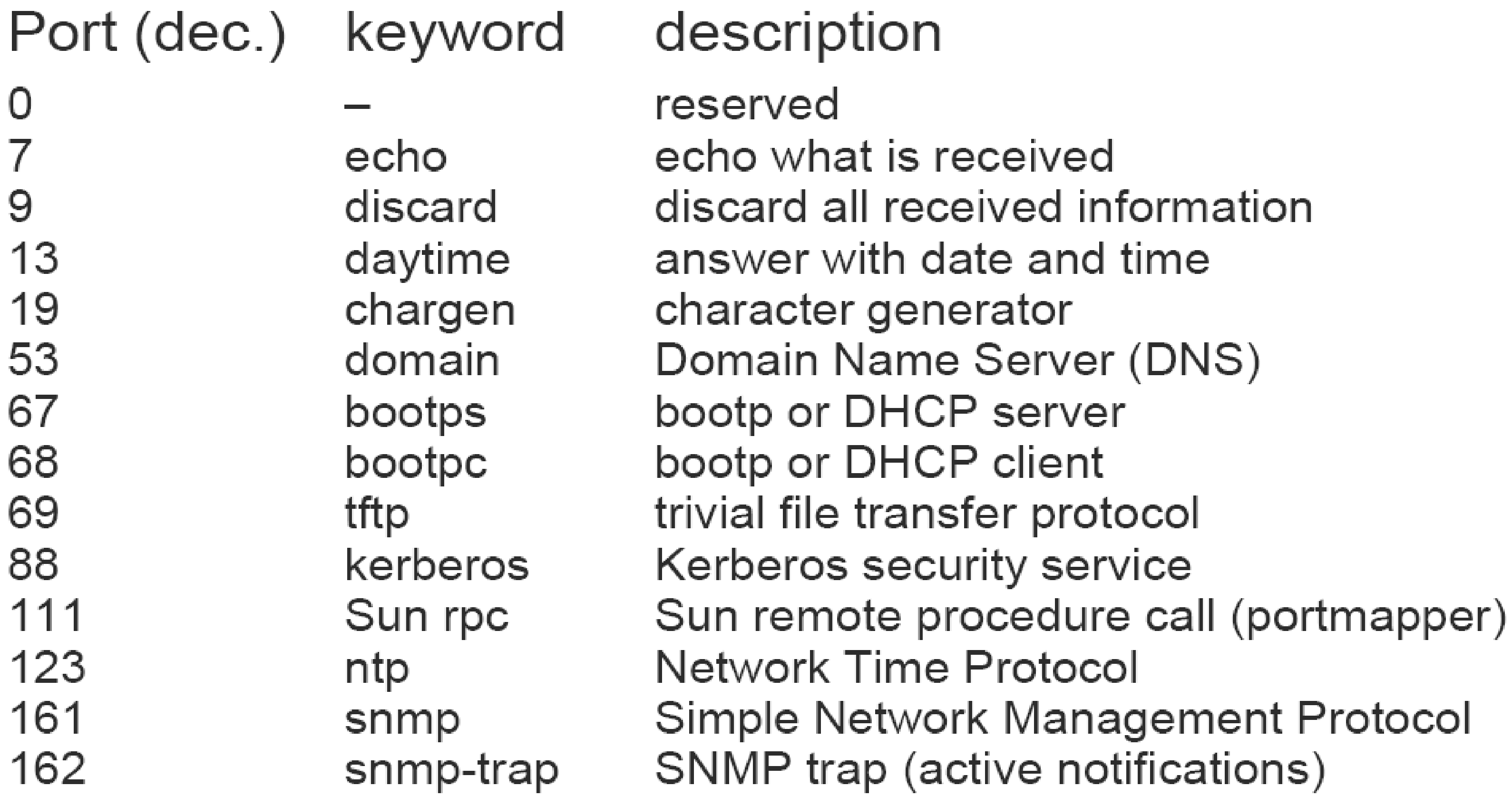
source port
destination port } end points of the source and destination applications

UDP length: 8-byte header and data

checksum: checksum over pseudo header, UDP header and UDP data (optional)

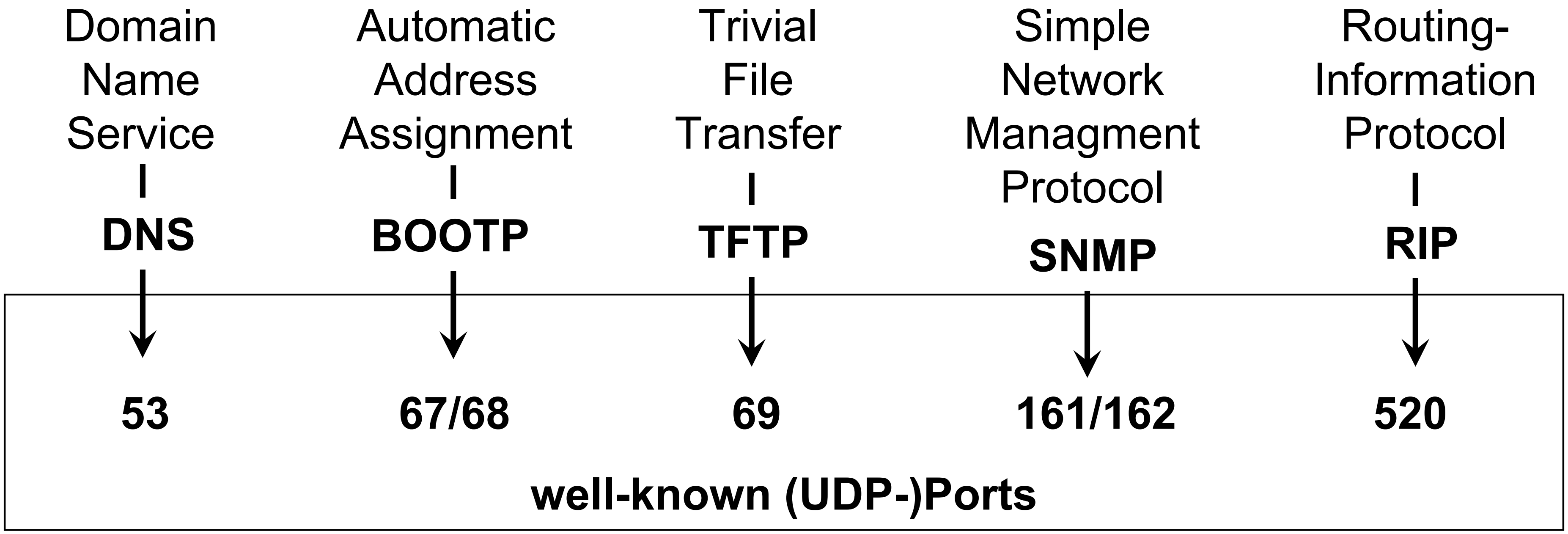
UDP Protocol - Port Numbers

- **Well-known port numbers** (for the complete list see RFC1700)



Port (dec.)	keyword	description
0	–	reserved
7	echo	echo what is received
9	discard	discard all received information
13	daytime	answer with date and time
19	chargen	character generator
53	domain	Domain Name Server (DNS)
67	bootps	bootp or DHCP server
68	bootpc	bootp or DHCP client
69	tftp	trivial file transfer protocol
88	kerberos	Kerberos security service
111	Sun rpc	Sun remote procedure call (portmapper)
123	ntp	Network Time Protocol
161	snmp	Simple Network Management Protocol
162	snmp-trap	SNMP trap (active notifications)

UDP Protocol - Application Examples



Domain Name Service – DNS → 53

Automatic Address Assignment – BOOTP → 67/68

Trivial File Transfer – TFTP → 69

Simple Network Managment Protocol – SNMP → 161/162

Routing-Information Protocol – RIP → 520

well-known (UDP-)Ports