Fachhochschule
Dortmund
University of Applied Sciences and Arts

Communications and Computer Networks

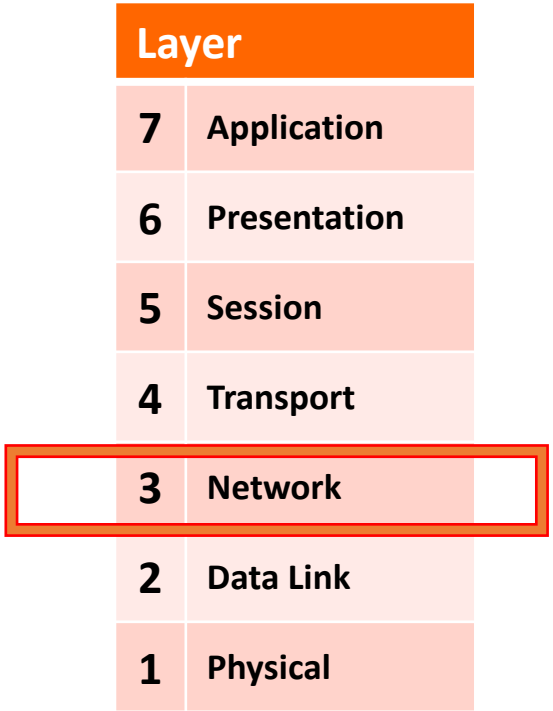
Summer Term 2023

Prof. Dr. Daniel Spiekermann
Faculty of Computer Science

Fachhochschule Dortmund
University of Applied Sciences and Arts

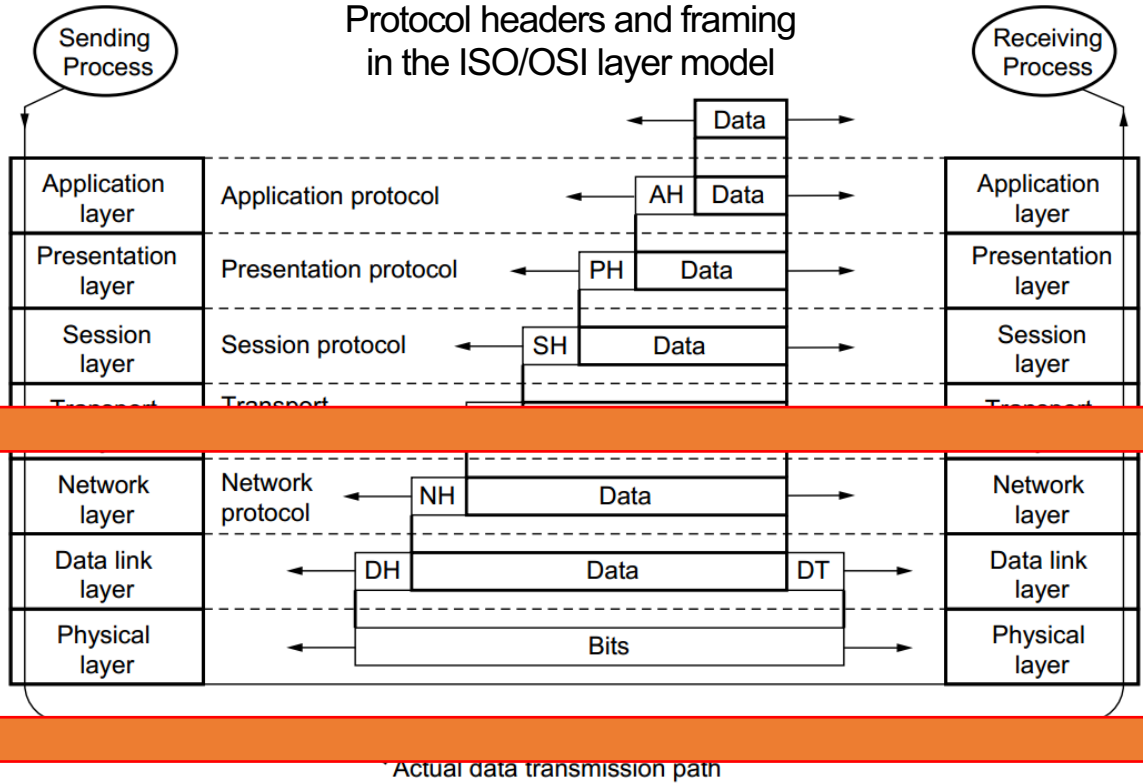
Recap of last lecture (1/7)

- Network layer and related topics
  - Fundamental tasks of the network layer
- Internet Protocol
  - You know the structure of an IPv4 and IPv6 header
  - You know subnetting, especially the use of VLSM
  - You can explain the task and procedure of the NDP and DAD
  - You know the process of fragmentation
- Routing and NAT



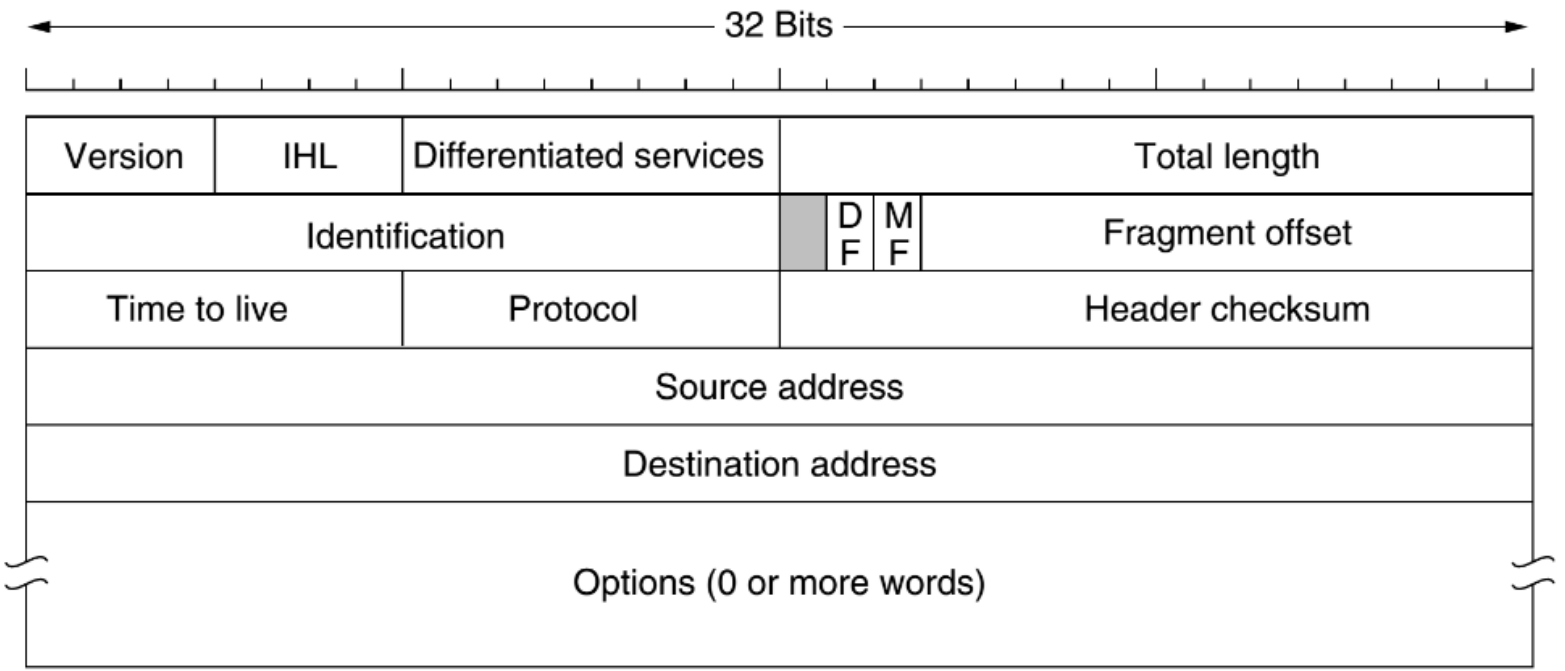
Fachhochschule Dortmund
University of Applied Sciences and Arts

Recap of last lecture (2/7)



Discussion till now:

Recap of last lecture (3/7)



Fachhochschule Dortmund
University of Applied Sciences and Arts

Recap of last lecture (4/7)

- IP-address + subnet mask define the allocation:
- 192.168.10.3 / 26

	IP	Subnet
Dot-decimal	192.168.10.3	255.255.255.192
binary	11000000.10101000.00001010.00000011	11111111.11111111.11111111.11000000

IP:	11000000	.10101000	.00001010	.00000011
Mask:	11111111	.11111111	.11111111	.11000000
Net:	192	.168	.10	.0
BC:	192	.168	.10	.63

6 Bit available = 2^6 = 64 addresses
All bits ”1” (= 111111) is BC-address = 63

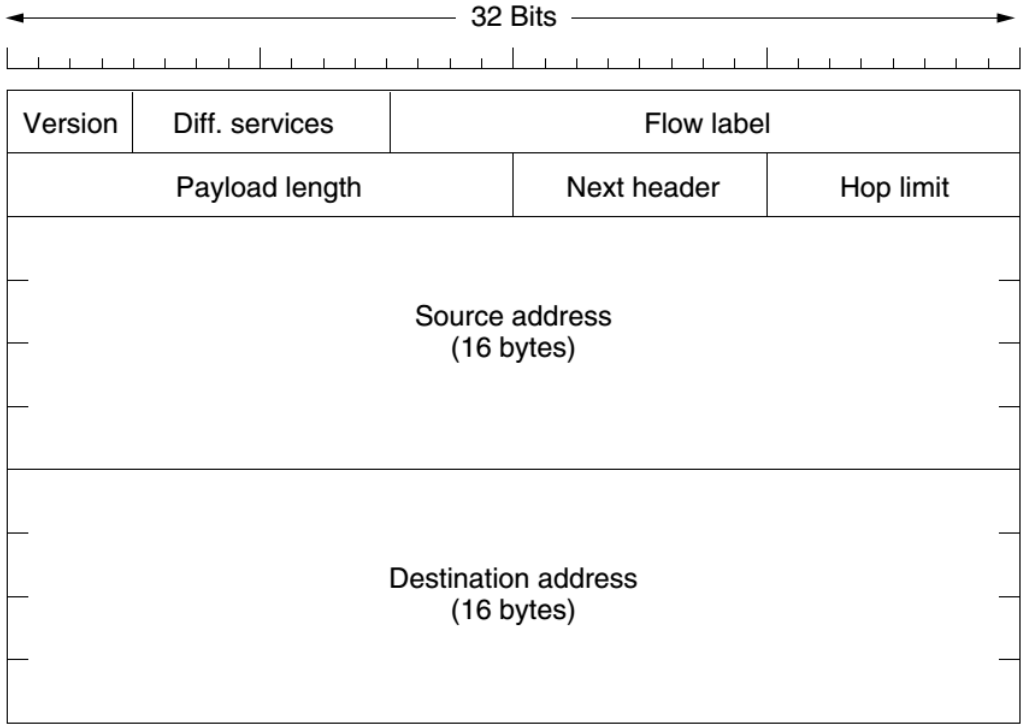
Recap of last lecture (5/7)

- Total Length
- Identification
- Flags

```
Total Length: 1500
Identification: 0xa964 (43364)
001. .... = Flags: 0x1, More fragments
   0... .... = Reserved bit: Not set
   .0.. .... = Don't fragment: Not set
   ..1. .... = More fragments: Set
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: ICMP (1)
```

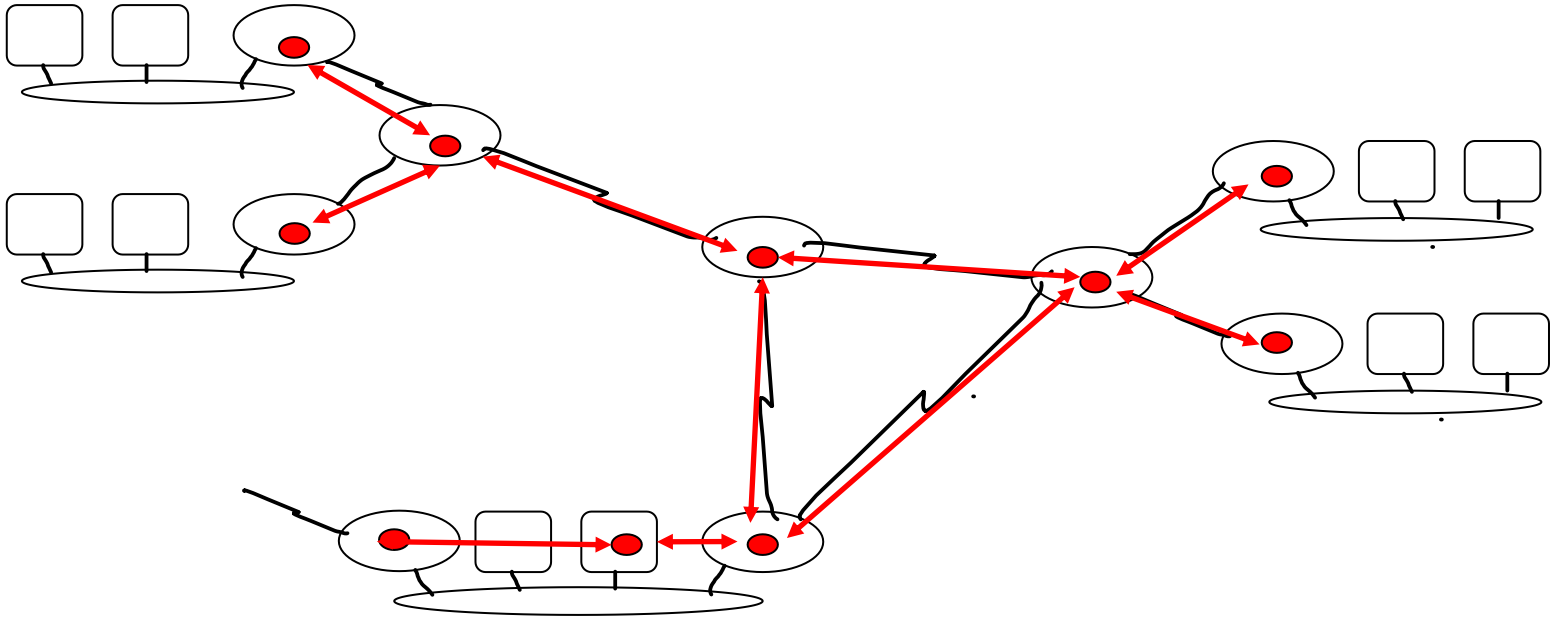
```
Total Length: 548
Identification: 0xa964 (43364)
000. .... = Flags: 0x0
   0... .... = Reserved bit: Not set
   .0.. .... = Don't fragment: Not set
   ..0. .... = More fragments: Not set
...0 0000 1011 1001 = Fragment Offset: 1480
Time to Live: 64
Protocol: ICMP (1)
```

Recap of last lecture (6/7)



Recap of last lecture (7/7)

Routing processes dialog between routers to exchange routing information.



Fachhochschule
Dortmund
University of Applied Sciences and Arts

Transport Layer - Fundamentals

Prof. Dr. Daniel Spiekermann
Faculty of Computer Science

Fachhochschule Dortmund
University of Applied Sciences and Arts

Content Overview

Fundamentals of the transport layer

UDP

TCP

- Reliable communication
- 3-Way-Handshake
- Sequence numbers
- Flow Control

QUIC

Learning Objectives

- You know the User Datagram Protocol (UDP) and can name its header structure
- You know the Transmission Control Protocol (TCP) and can name its header structure
- You know the principle of and can name selected ports
- You know the tasks of TCP and can explain the communication process, including setting up and disconnecting the connection.
- You know the structure of the TCP header and can explain the fields.
- You can explain the principle of TCP window management.
- You are familiar with the Bandwidth-Delay product and can explain its importance for overload control using window control.
- You know the methods fast-retransmit and fast-recovery and can explain them.
- You know the QUIC protocol

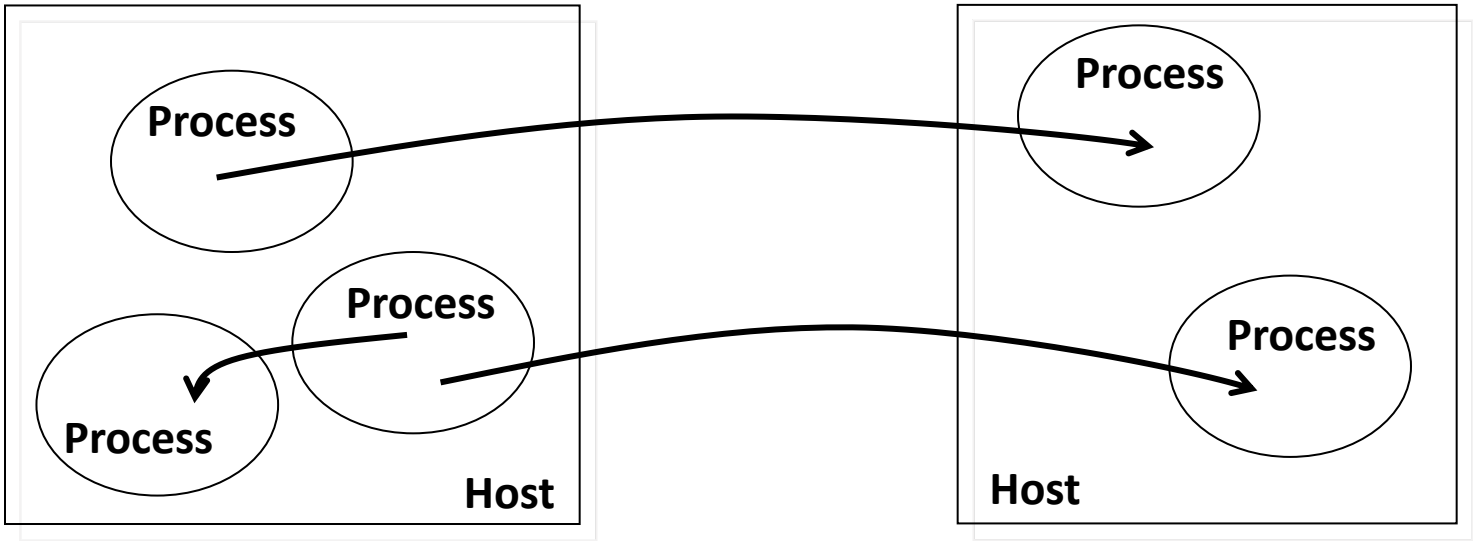
Fachhochschule Dortmund
University of Applied Sciences and Arts

Layer 4 – Transport layer

- Transport layer implements communciation between two transport entities
- It only knows the address of the source and destination station, formally defined as Transport Service Address Point (TSAP), typically **called ports**
- **Connection establishment as well as connection management is part of L4**
- In addition to the connection build-up and connection dismantling, the transport layer also **carries out flow control and monitoring for transmission errors**.
- Relevant protocols:
  - TCP
  - UDP
  - SCTP
  - QUIC

IPC Inter-Process Communication

The ultimate goal of the communication mechanism between computers is that a **process** can communicate with another process that runs on another machine (or on the same one) in a transparent way.



Transport Protocols

The transmission/transport layer protocols proposed by DARPA are:

- **TCP** - Transmission Control Protocol
- **UDP** - User Datagram Protocol

- TCP and UDP are **end-to-end** protocols.

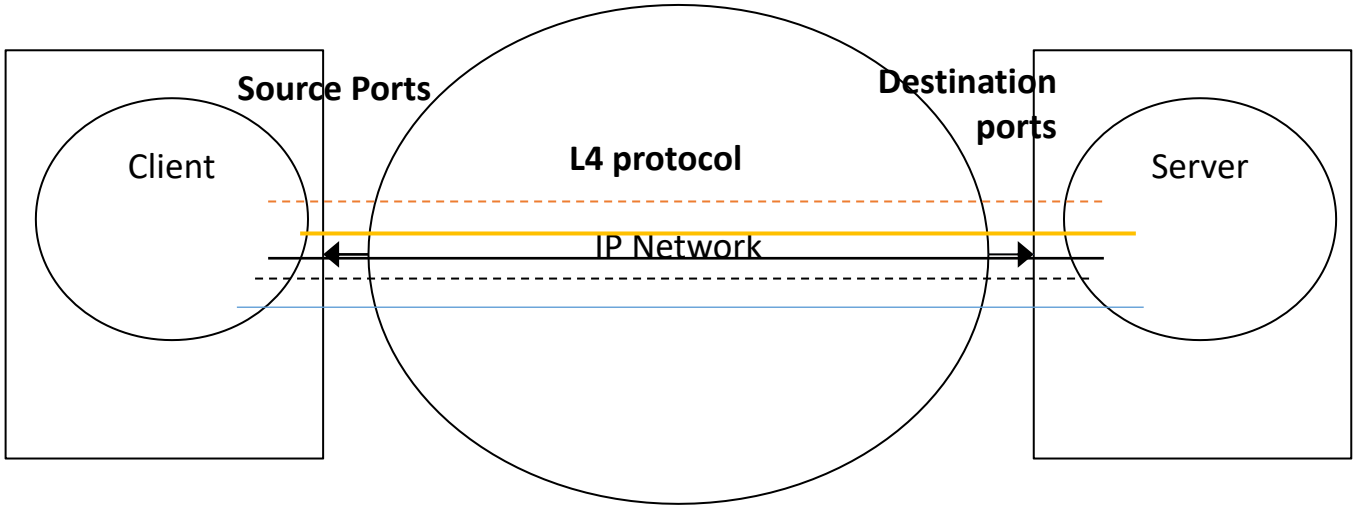
- TCP and UDP PDUs are encapsulated in IP Datagrams
  - TCP PDU: **TCP Segment**
  - UDP PDU: **UDP Datagram**

Multiplexing with Port Numbers

To implement Inter Processes Communication, L4 uses port numbers as TSAP

Sender and receiver use ports numbers, so we have source port and destination port

Multiple connections with different ports are possible

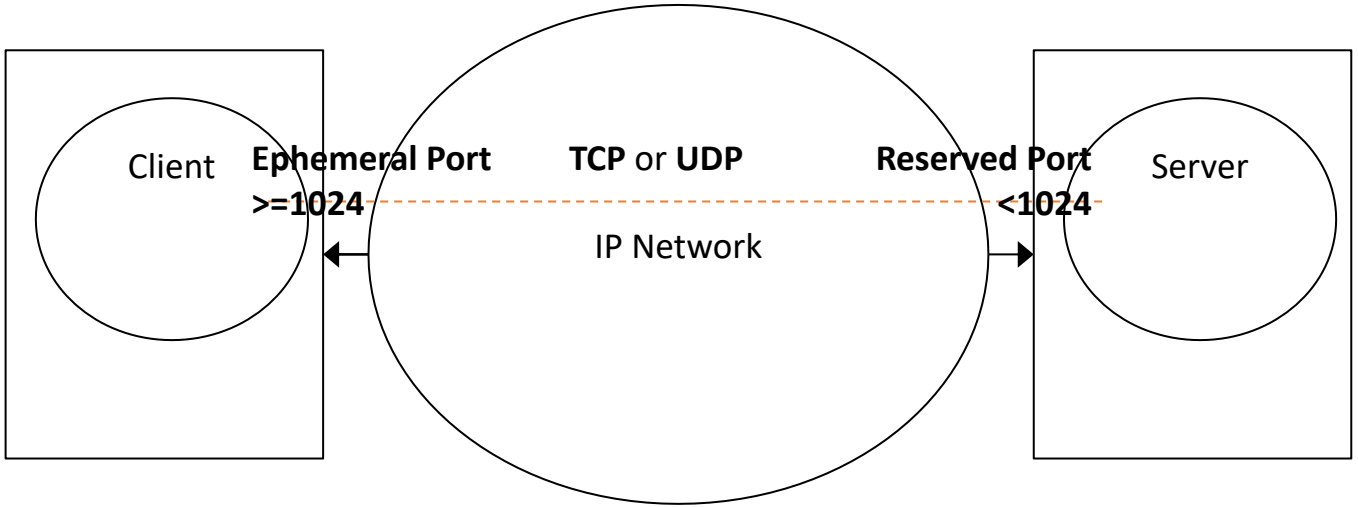


Multiplexing with Port Numbers

- The pair of parameters (IP Addr:Port Number) is called socket
  - 192.168.10.8:53
- The traffic is multiplexed between the different application processes using 5 parameters:
  - Source IP Address
  - Source Transmission Protocol Port Number
  - Transmission Protocol Number (TCP or UDP)
  - Destination IP Address
  - Destination Transmission Protocol Port Number
- This allows the communication by different applications or hosts simultaneously
  - 10.0.0.3:22488:10.0.0.7:80
  - 10.0.0.3:33597:10.0.0.7:80
  - 10.0.11.2:44100:10.0.0.7:80
- Of course with IPv6
  - [fe80::14ed:7b38:f8df:1170]:44322:[fe80::1c75:464a:3a57:a0d2]:443

Transport Protocols
Well Known Services / Reserved Port Numbers

Known applications use "**Well Known port**" (<1024) by the server and **ephemeral port** numbers (>= 1024) by the client



Fachhochschule Dortmund
University of Applied Sciences and Arts

Port numbers

Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry

Abstract

This document defines the procedures that the Internet Assigned Numbers Authority (IANA) uses when handling assignment and other requests related to the Service Name and Transport Protocol Port Number registry. It also discusses the rationale and principles behind these procedures and how they facilitate the long-term sustainability of the registry.

- Defined by the IANA, Details in RFC 6335
- Port number is between 0-65535 (2^{16})
- Dividing in three ranges:
  - System (well-known): 0 – 1023
  - Registered: 1024 – 49151
  - Dynamic (private): 49152 – 65535
- *nix-systems provide list in /etc/services
- Provide mapping between L7 application and L4 TSAP
  - Systems “know” the mapping, so http://www.fh-dortmund.de is requested on port 80 without a specific calling of the port separated with colon: www.fh-Dortmund.de**:80**

Well Known Services / Reserved Port Numbers

Well known service	Reserved Port Number
TELNET	23
FTP	20, 21
DNS	53
HTTP	80
HTTPS	443
SMTP	25
POP3	110
IMAP4	143
IMAP4 over TLS/SSL	993
SSH	22
SNMP	161, 162

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Transport Layer - UDP

Prof. Dr. Daniel Spiekermann
Faculty of Computer Science

User Datagram Protocol (RFC 768)

User Datagram Protocol (UDP) provides

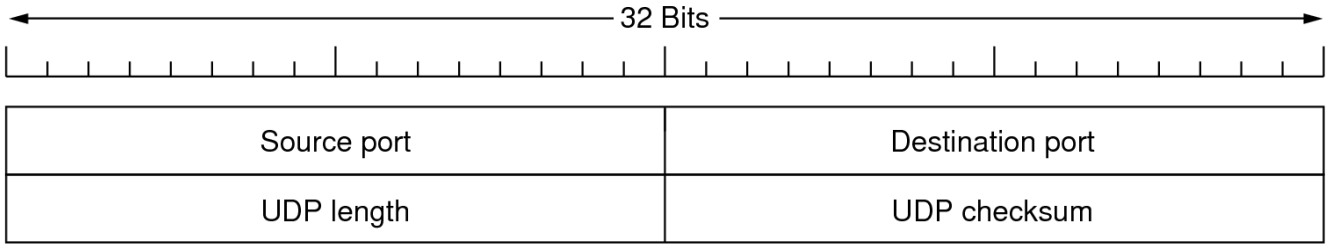
- an **end-to-end** transport mechanism
- with **minimal overhead**.

It is

- unreliable
- connectionless

The application program over UDP will be responsible for ensuring reliability and flow control.

UDP header



- **Source Port**: Port number of the sender
- **Destination Port**: Port number of the recipient
- **UDP Length**: Length of the datagram including header
- **UDP Checksum**: Checksum including pseudo-header

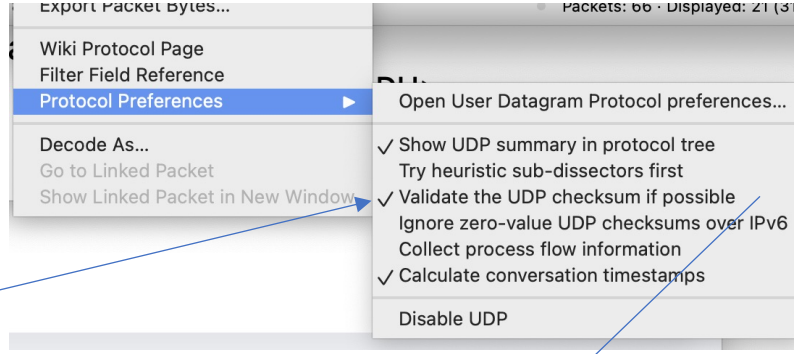
The use of Checksum is optional and is used to check the integrity of the header and the packet data. The checksum is calculated by adding a pseudo-header that contains the source and destination address, as well as the protocol number and the UDP packet length, in order to be able to verify without a doubt that the packet reached its correct recipient.

UDP Datagrams

- The UDP datagram is encapsulated over IP.
- The maximum size of a UDP Datagram is given by the maximum payload of an IP Datagram approximately 64 KB (2^16 minus the IP header = 65.515 Bytes).
- In the event that it is necessary to do fragmentation at the IP level, the UDP Datagram can be transported by several IP Datagrams. Only the 1st IP fragment carries the UDP header.
- UDP is used when reliability is not particularly important for the transferred information, or where retransmitted packets are annoying

Fachhochschule Dortmund
University of Applied Sciences and Arts

UDP in Wireshark




```
> Frame 5: 132 bytes on wire (1056 bits), 132 bytes captured (1056 b
> Ethernet II, Src: Tp-LinkT_af:48:d4 (54:e6:fc:af:48:d4), Dst: EIZO
> Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.183
v User Datagram Protocol, Src Port: 53, Dst Port: 49815
    Source Port: 53
    Destination Port: 49815
    Length: 98
    Checksum: 0xa361 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 1]
  > [Timestamps]
    UDP payload (90 bytes)
> Domain Name System (response)
```

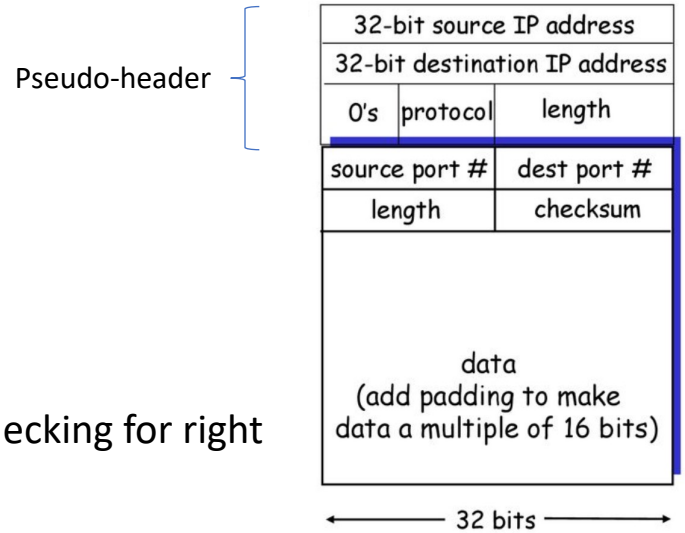
```
> Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.183
v User Datagram Protocol, Src Port: 53, Dst Port: 49815
    Source Port: 53
    Destination Port: 49815
    Length: 98
  v Checksum: 0xa361 [correct]
      [Calculated Checksum: 0xa361]
    [Checksum Status: Good]
    [Stream index: 1]
  > [Timestamps]
    UDP payload (90 bytes)
> Domain Name System (response)
```

Checksum calculation

- Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets

1. Add pseudo header
2. Fill checksum with 0
3. Divide in 16bit words (padding if needed)
4. Add words using 1s complement arithmetic
5. Complement result and put in checksum fileds
6. Drop pseudo-header
7. Deliver

Use of ip-addresses breaks separation of layers, idea was checking for right receiver, nowadays more or less useless



Protocols on top of UDP

- Common UDP applications do not employ reliability mechanisms and may even be hindered by them
- RTP
  Real Time Protocol
  used for delivering of audio and video over IP
  Unprivileged port (1024 – 65535), typically an even port number
- DNS
  Domain Name Service
  “Translation” of names to ip-addresses
  Port 53
- SNMP
  Simple Network Management Protocol
  Protocol for collecting and organizing information of IP-based network devices
  Port 161, 162

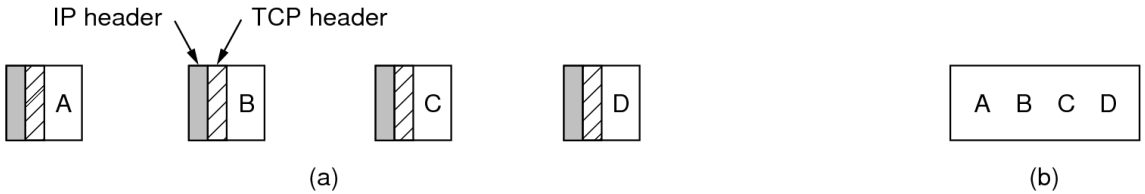
Fachhochschule
Dortmund
University of Applied Sciences and Arts

Transport Layer - TCP

Prof. Dr. Daniel Spiekermann
Faculty of Computer Science

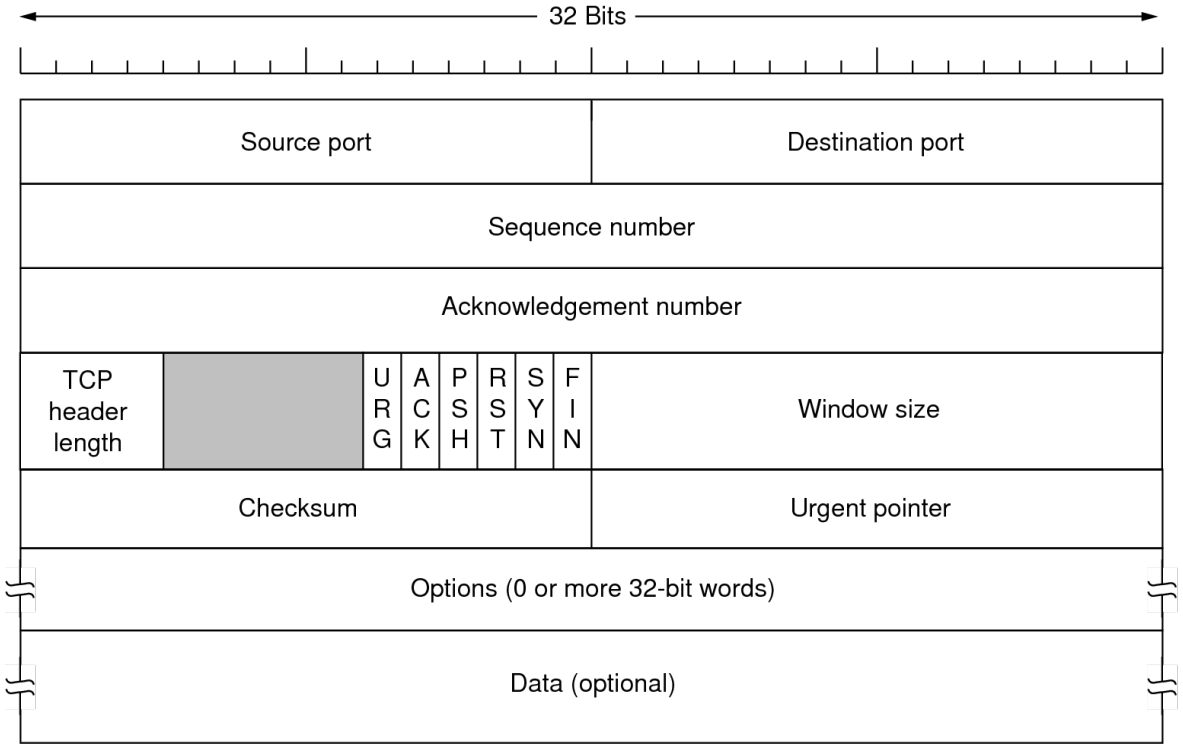
Transmission Control Protocol TCP (RFC 793, RFC 1323)

- The Transmission Control Protocol enables reliable communication between processes. It is guaranteed that the data is transported error-free, without loss or duplication in the original order.
- The protocols of higher layers pass data to the TCP for transmission to another process.
- TCP divides this data into segments and passes them to the IP, which divides these segments into datagrams and transports them to the target system.



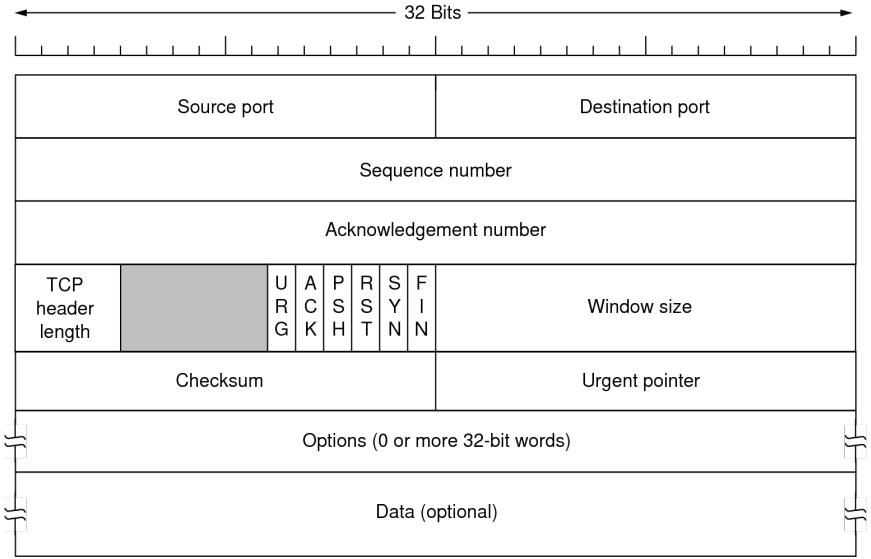
(a) Division of data into segments and transmission via IP
(b) Passing the composite data to the application (using a read statement)

TCP header


32 Bits
Source port
Destination port
Sequence number
Acknowledgement number
TCP header length
URG
ACK
PSH
RST
SYN
FIN
Window size
Checksum
Urgent pointer
Options (0 or more 32-bit words)
Data (optional)

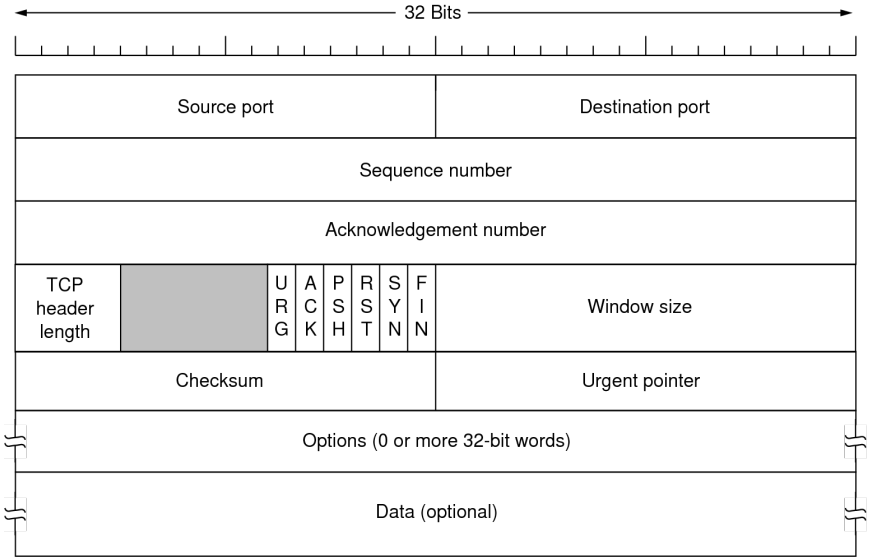
TCP header II

- 16 Bit Source Port: Port of the sender
- 16 Bit Destination Port: Port of the receiver
- 32 Bit Sequence Number:
  Sequence number, numbering of each byte
- 32 Bit Acknowledgement Number:
  Acknowledgement (the next expected byte)
- TCP Header Length:
  Length of the header in 32 bit words
- URG Flag: Urgent (urgent message)
- ACK Flag: Acknowledgement Number valid
- PSH Flag: Push data (forward data without buffering)

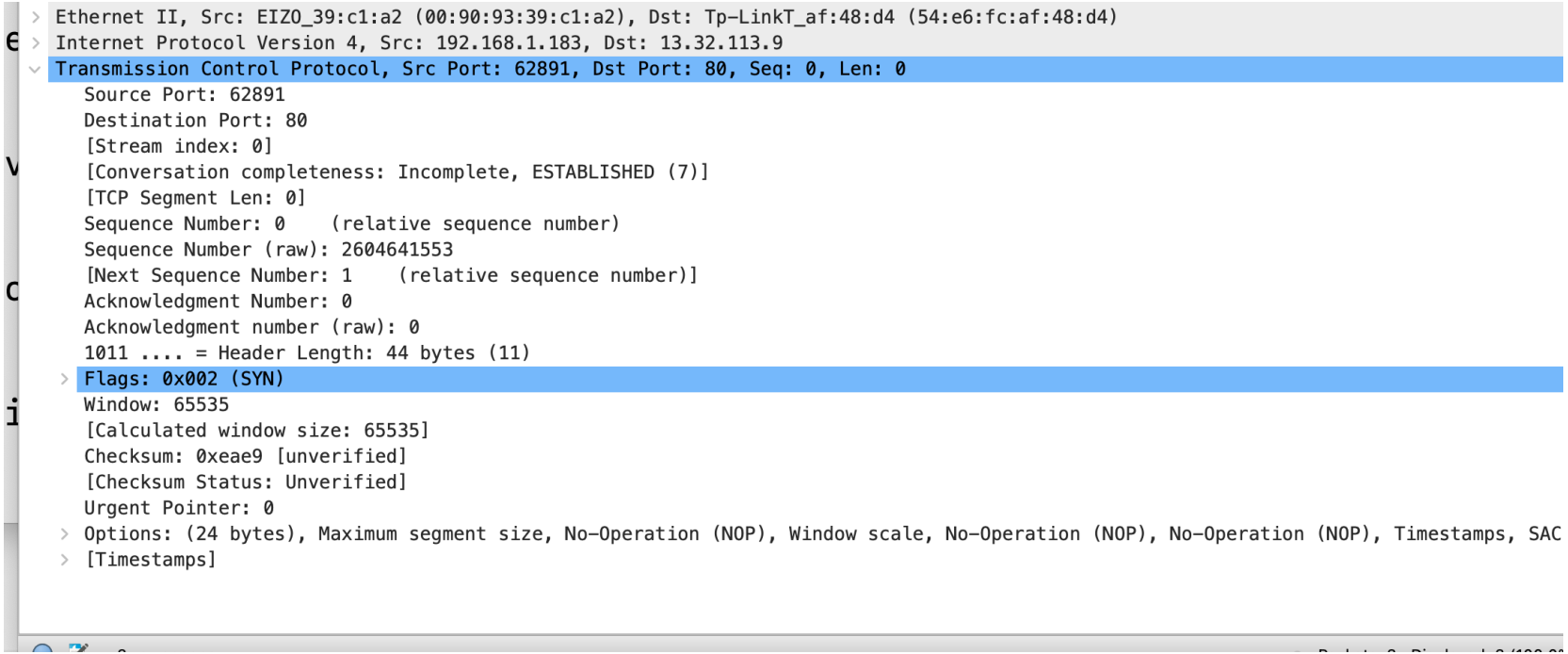


TCP header III

- RST Flag: Resetting the Connection
- SYN Flag: Establishing a connection
- FIN Flag: Disconnection
- Window Size:
  Slide window for flow control
  (number of bytes that can be sent before an acknowledgement)
- Checksum: Checksum including pseudo-headers (IP addresses, etc.)
- Urgent Pointer:
  points to urgent data (byte offset)
- Options: additional options, e.B. maximum size payload



TCP and Wireshark

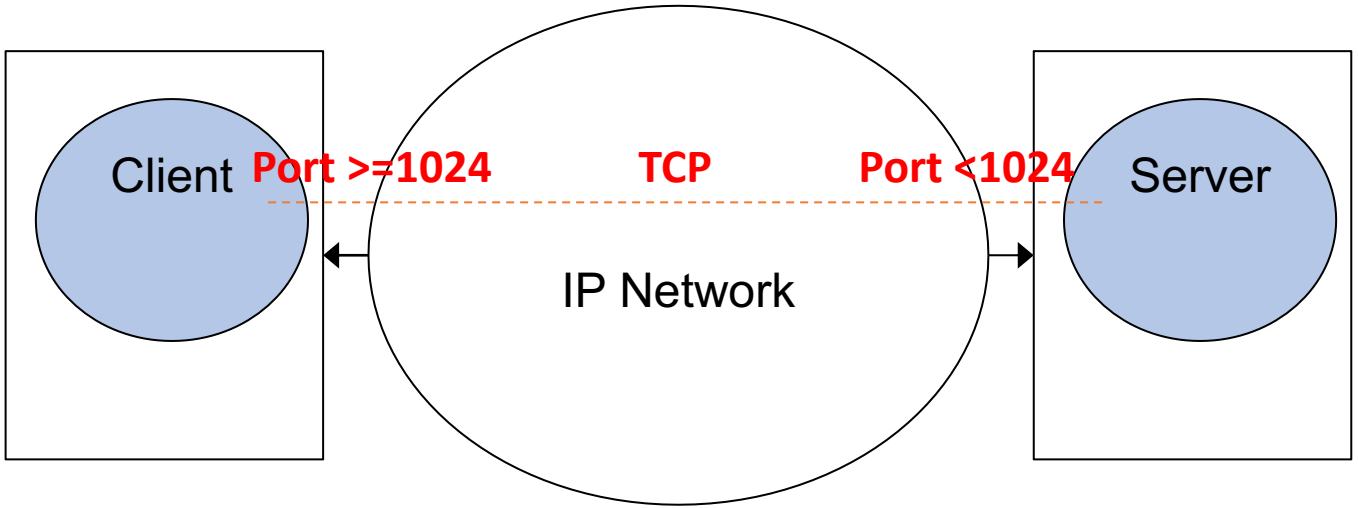


TCP - Functions

- Multiplexing of IPC traffic with Port Numbers
- Byte Oriented Full duplex transmission
- CO communication
  - Segmentation
  - Sequence control
  - Sliding Window
- Error control
- Explicit Flow Control
- Implicit Congestion Control

TCP Multiplexing with Ports

Multiplexing of IPC traffic with Port Numbers



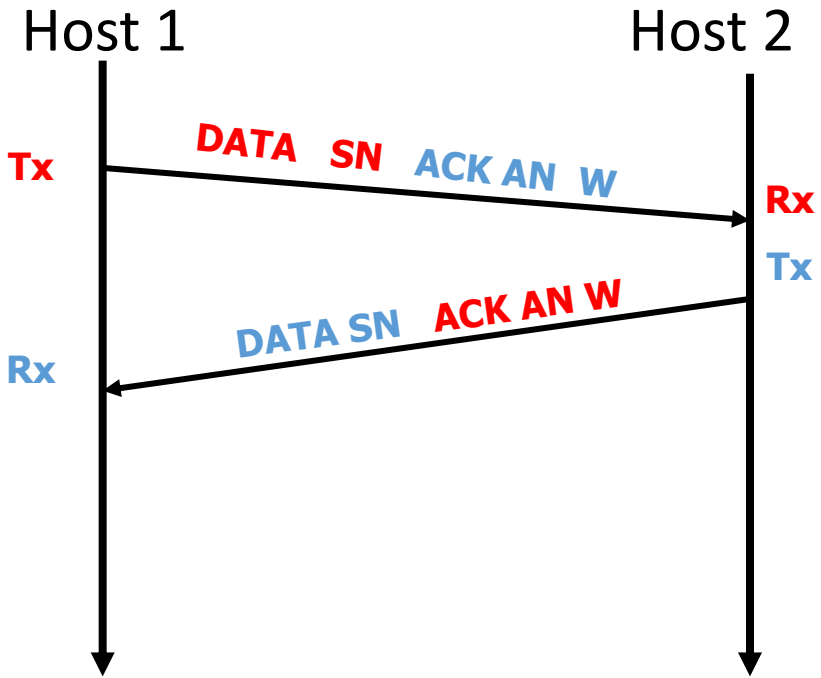
TCP – Byte stream full duplex

TCP is a **full duplex** data transmission service.

There are 2 data streams, 1 in each direction. The "control" of each stream is independent of the other.

Each TCP Entity has a **Transmitter** Tx and a **Receiver** Rx. Each data stream goes from Tx to Rx.

Along with the Data in one direction, **piggybacked** parameters from the opposite flow are sent.



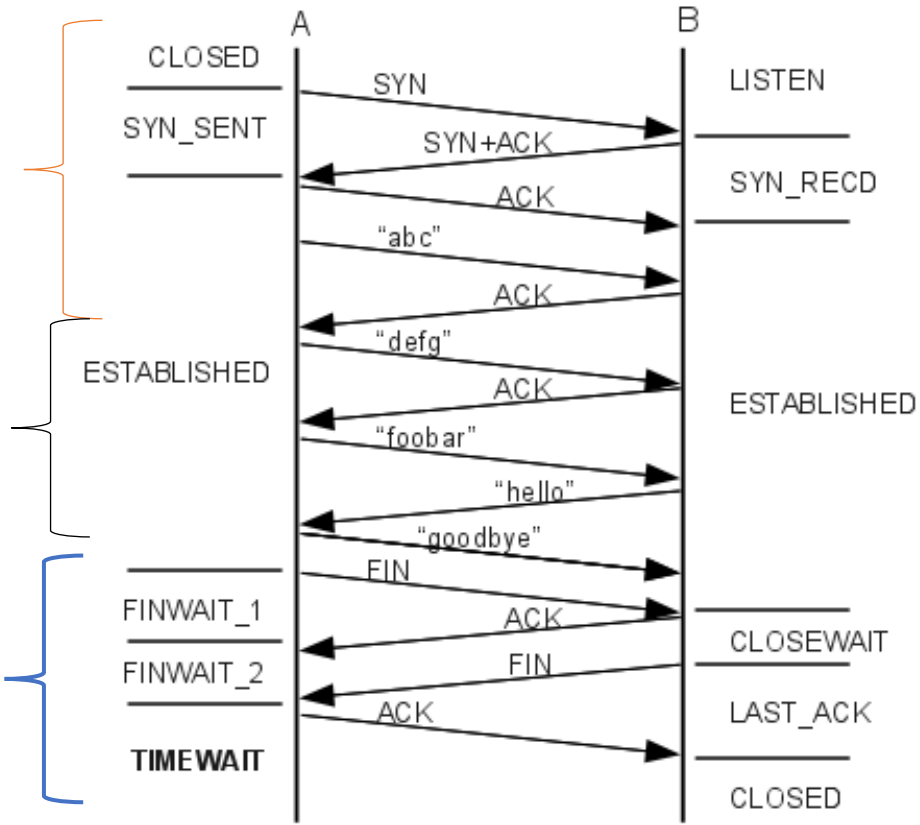
Fachhochschule
Dortmund
University of Applied Sciences and Arts

TCP Connection Control

Prof. Dr. Daniel Spiekermann
Faculty of Computer Science

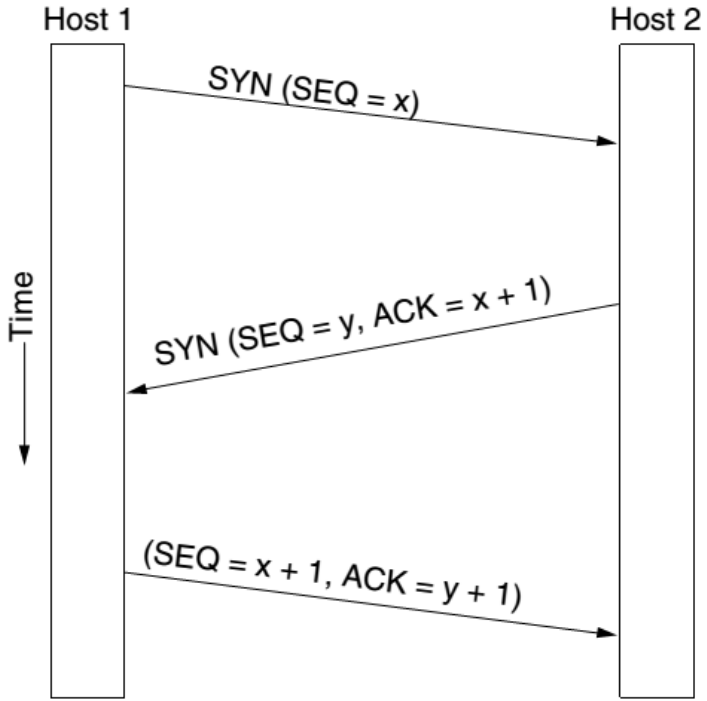
Connection control

- Initial connection establishment
- Correct data transfer
- Connection termination



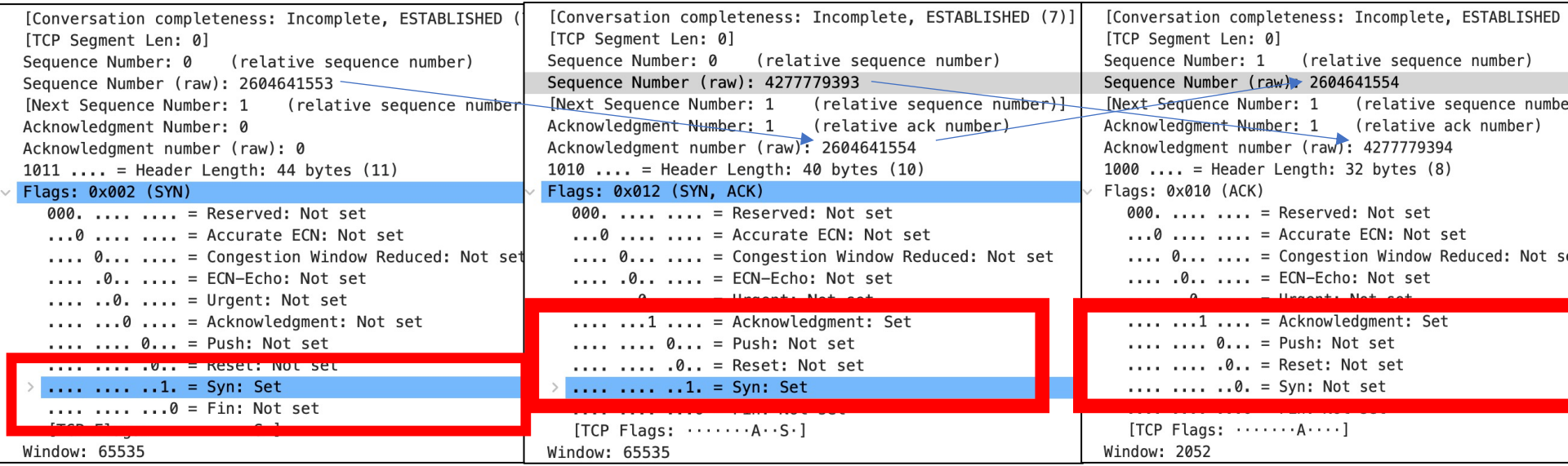
Connection establishment

- Before the actual data transfer is initiated, TCP establishes a logical computer-to-computer connection between sender and receiver with the so-called 3-way handshake.
- Step 1:
  Sender sends TCP packet with SYN = 1 and random SEQ (x)
- Step 2:
  Receiver answers with SYN = 1 and ACK = 1, ACK-no is SEQ+1 (x+1) and own SEQ-no (y)
- Step 3:
  Initial sender answers with SYN = 0, ACK = 1, ACK-no = y+1 and SEQ = x+1
- Additionally, MSS (Maximum Segment Size is transferred)



Fachhochschule Dortmund
University of Applied Sciences and Arts

3Way Handshake



Step 1 Step 2 Step 3

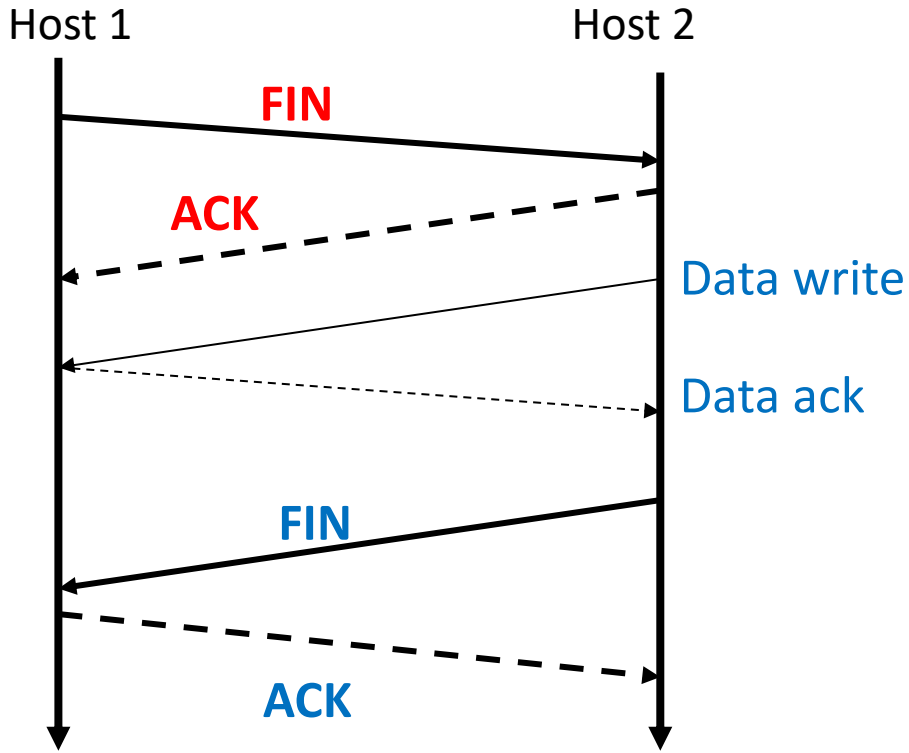
Connection termination

Both ends define completion

1. Initiator sends tcp.fin = 1
2. Receiver answers with tcp.ack = 1

   But: Receiver must terminate from its side (after all data is sent)
3. Receiver sends tcp.fin = 1
4. Initiator answers with tcp.ack = 1

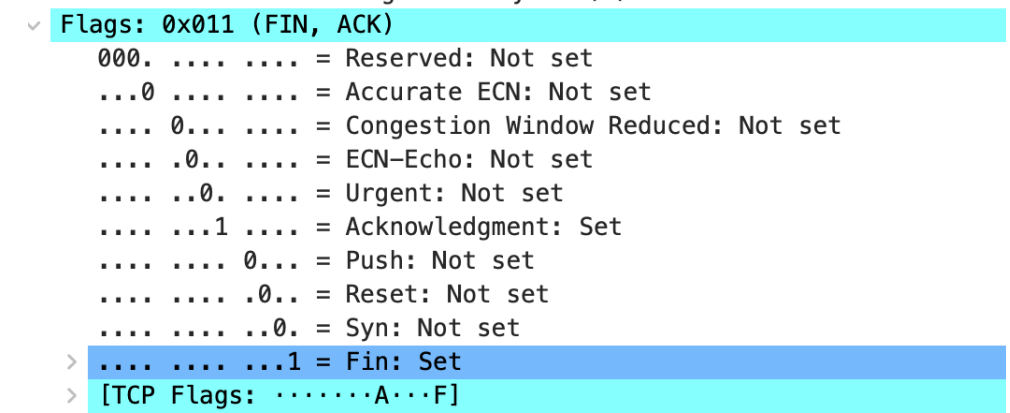
Typically, FIN+ACK are sent together



Fachhochschule Dortmund
University of Applied Sciences and Arts

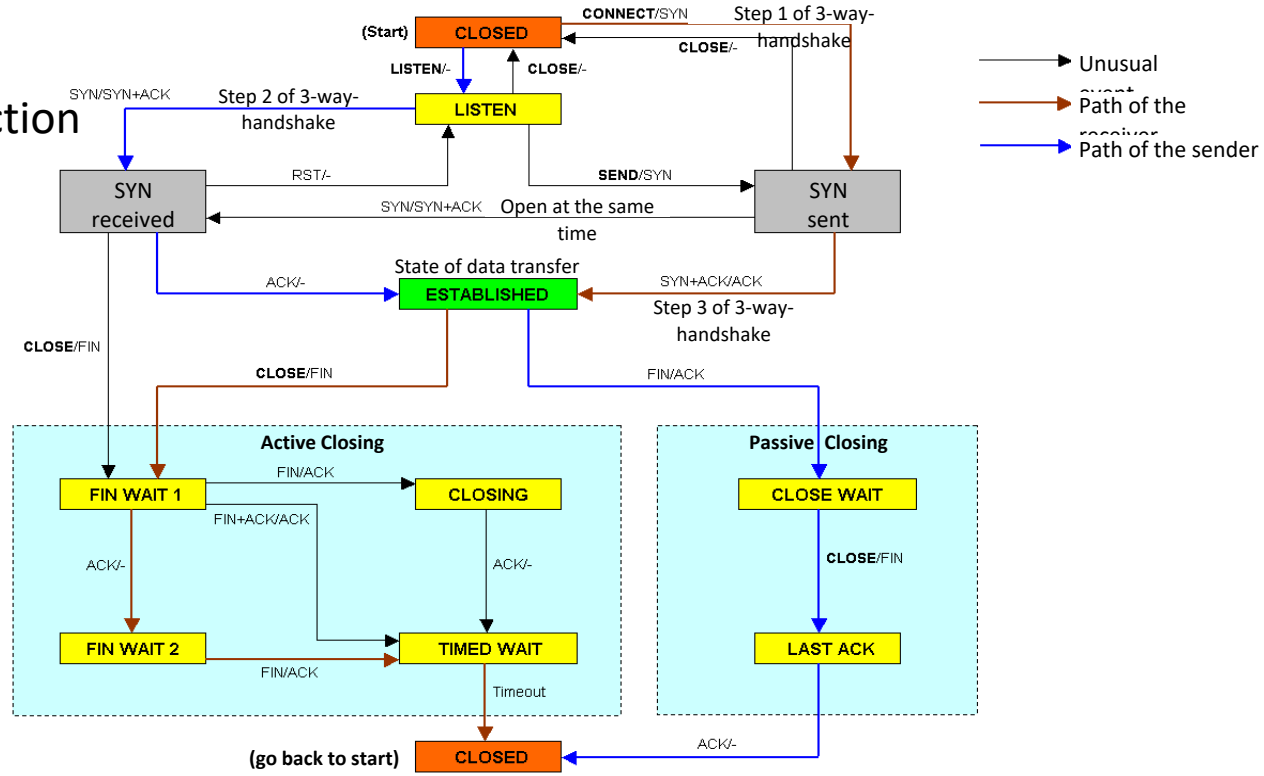
Connection termination

0.000000	192.168.1.183	52.222.226.205	TCP	66 63281 → 80 [FIN, ACK] Seq=1
0.005869	52.222.226.205	192.168.1.183	TCP	66 80 → 63281 [FIN, ACK] Seq=1
0.005912	192.168.1.183	52.222.226.205	TCP	66 63281 → 80 [ACK] Seq=2 Ack=



TCP Finite State Machine (FSM)

- "Lifetime" of a connection



Meanings of the states

State	Meaning
CLOSED	no connection active
LISTEN	Server waits for a connection
SYN RCVD	Arrival of a connection request, waiting for ACK
SYN SENT	Client has started to open connection
ESTABLISHED	Connection open
FIN WAIT 1	Client wants to close connection
FIN WAIT 2	Server agrees
TIMED WAIT	Wait until no more packages arrive
CLOSING	Both sides tried to use the conn. at the same time. to close
CLOSE WAIT	The other side has initiated disconnection
LAST ACK	Wait until no more packages arrive

Fachhochschule
Dortmund
University of Applied Sciences and Arts

TCP – Flow Control

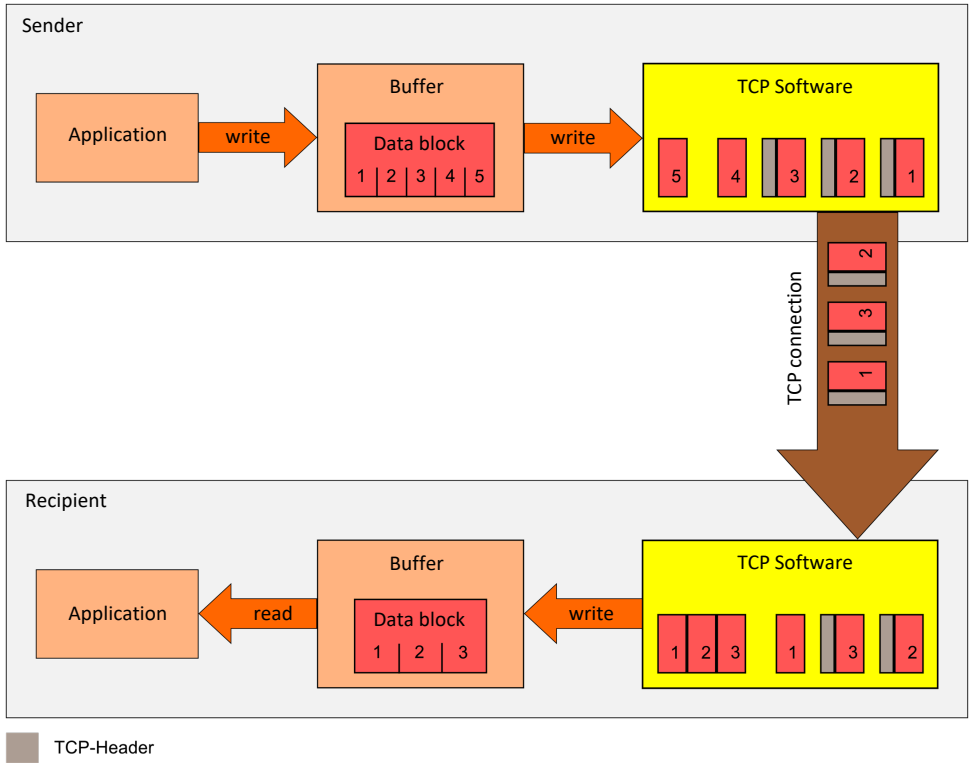
Prof. Dr. Daniel Spiekermann
Faculty of Computer Science

Flow Control

- On the target system, TCP gets its segments back from L3 and checks the error-free transmission.
- Correct segments are then assembled in the original order, passed to the higher protocol and the reception of the sender side confirmed.
- If a segment is damaged or lost, it is not confirmed, after which the sender side repeats its transmission.
- The TCP implements flow control between the communicating processes.
- During communication, the receiving party informs the sender in the confirmation messages how much data can be processed in the next transmission.

Segmentation of user data

- Sender:
  - Partition the data in segments
  - Segments might have different sizes
  - Give each segment an consecutive ID
  - Give data to L3
- Receiver:
  - Take data of L3
  - Check ID
  - Reconstruct the segments to the original data (might need buffering)



Sequence control and buffering

The Sequence number (SN) of a TCP segment indicates the sequence number of the **first byte** contained.

Based on the **SN** of each received TCP Segment, **the receiver saves the data in a position of the reception buffer**, thus reconstructing the **correct sequence** of the byte stream and can also detect and eliminate data **duplication**.

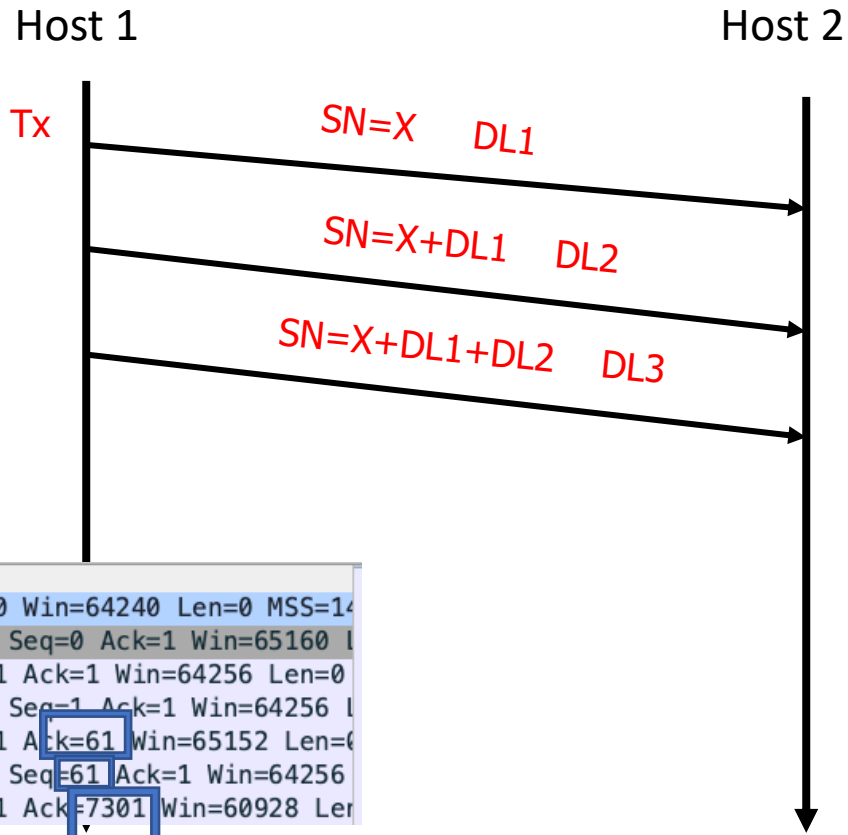
TCP needs buffer to store the segments

Fachhochschule Dortmund
University of Applied Sciences and Arts

Sequence control

Transmitter side:

- **SN** identify the sequence number of the **1st data byte** of each segment.
- The SNs are **not consecutive**.
  The SN of the segment following X is equal to X plus the Data Length

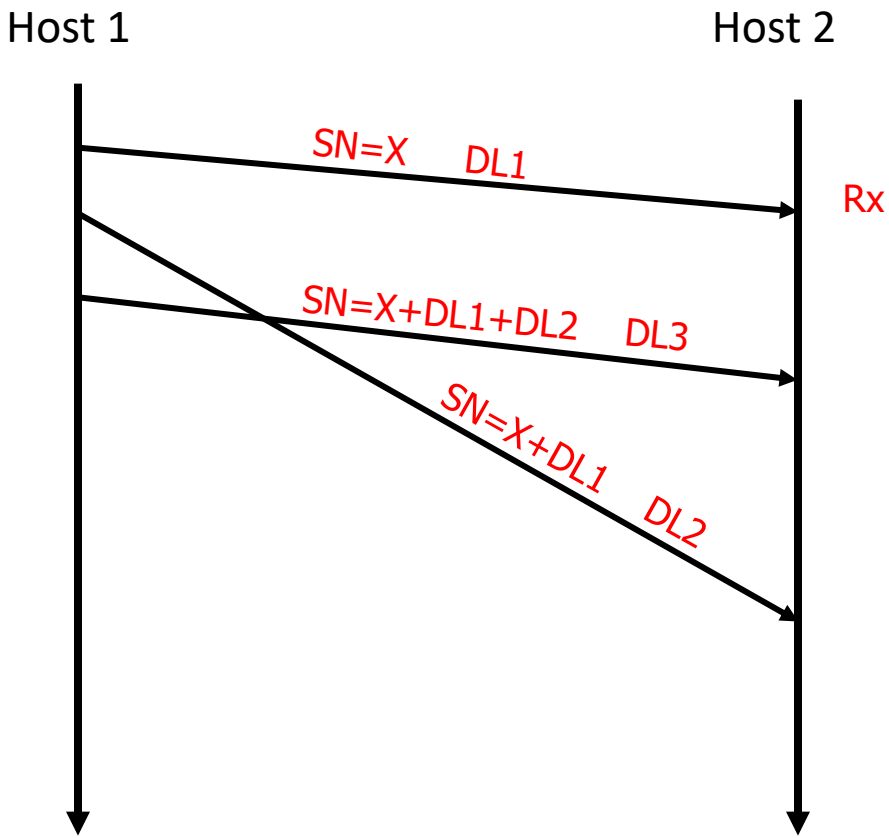


Protocol	Length	Bytes in flight	Calculated window :	Info
TCP	74		64240	47878 → 5001 [SYN] Seq=0 Win=64240 Len=0 MSS=14
TCP	74		65160	5001 → 47878 [SYN, ACK] Seq=0 Ack=1 Win=65160 L
TCP	66		64256	47878 → 5001 [ACK] Seq=1 Ack=1 Win=64256 Len=0
TCP	126	60	64256	47878 → 5001 [PSH, ACK] Seq=1 Ack=1 Win=64256 L
TCP	66		65152	5001 → 47878 [ACK] Seq=1 Ack=61 Win=65152 Len=0
TCP	7306	7240	64256	47878 → 5001 [PSH, ACK] Seq=61 Ack=1 Win=64256
TCP	66		60928	5001 → 47878 [ACK] Seq=1 Ack=7301 Win=60928 Len

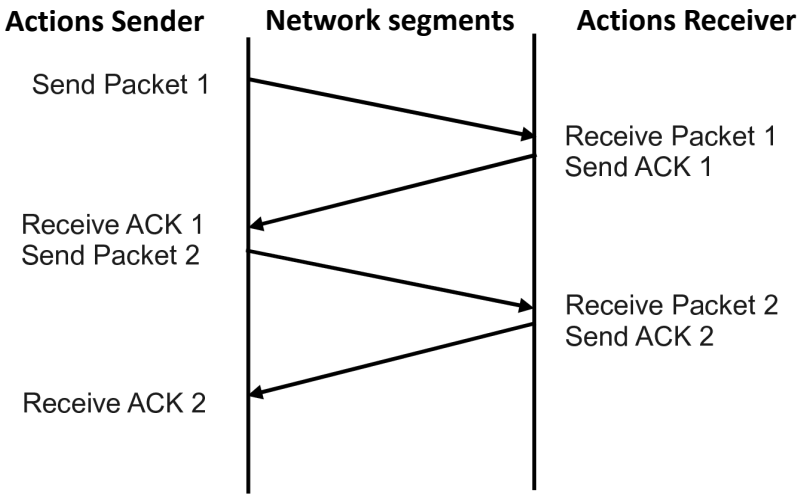
Sequence control

Receiver side

- Receives each segment and if the checksum is correct, **saves** the **DL bytes** of data in the receiver buffer **starting at position SN**.
- Even if the segments arrive out of sequence (e.g., going by different paths), the data is **stored in order**.



Positive Acknowledgement Protocol

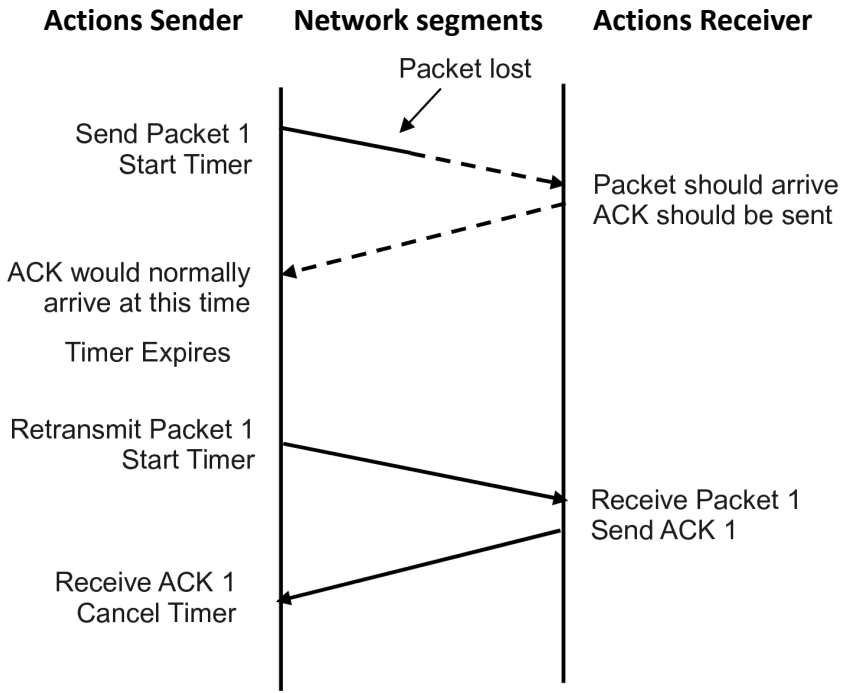


During the actual data transfer (state "Established"), each packet is acknowledged with an *acknowledgement*.

Receiver sends set ACK flag and acknowledgement number of the received segment

Also called stop-and-wait-protocol

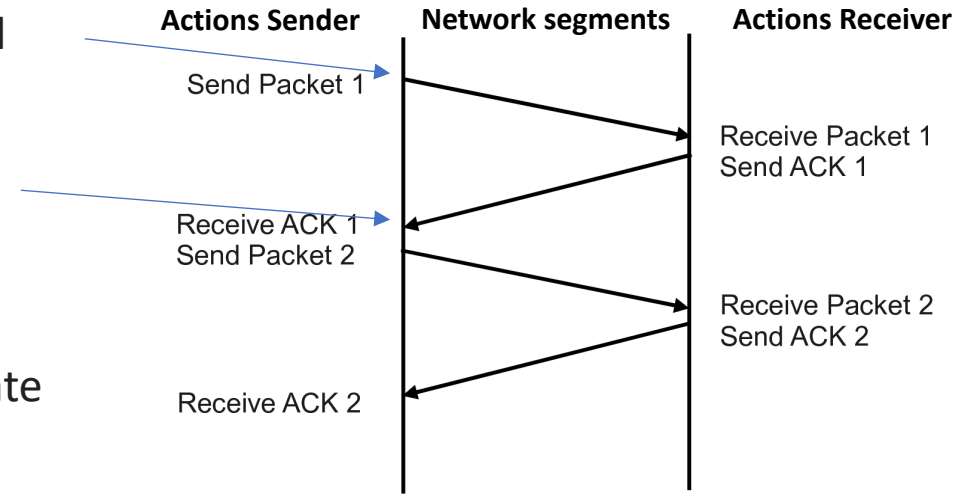
Timeout und Retransmission



In the event of an error (lost packet), a new transmission takes place after a timeout.

Round Trip Time

- Amount of time it takes for a signal to be sent *plus* the amount of time it takes for acknowledgement of that signal having been received.
- Includes propagation time
- High RTT with per-packet ACK reduces the overall transmission rate



RTT

- Sometimes called ping-time
- Local Ping:

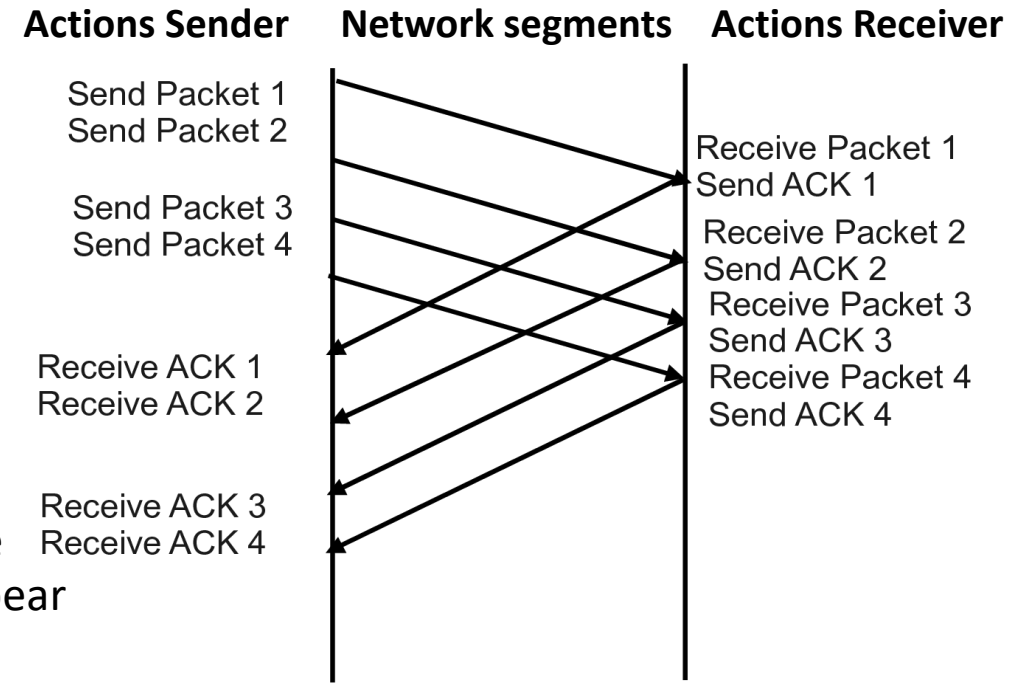
```
└─$ ping 192.168.10.1 -c1
PING 192.168.10.1 (192.168.10.1): 56 data bytes
64 bytes from 192.168.10.1: icmp_seq=0 ttl=64 time=17.344 ms
```

- Ping to Australia:

```
└─$ ping sydney.edu.au -c1
PING sydney.edu.au (20.248.131.216): 56 data bytes
64 bytes from 20.248.131.216: icmp_seq=0 ttl=108 time=304.788 ms
```

Sliding Window

- Improvement of simple stop-and-wait
- Multiple packets are send sequentially
- Problem:
  Buffer on the receiving side needs enough space, otherwise congestion problems might appear

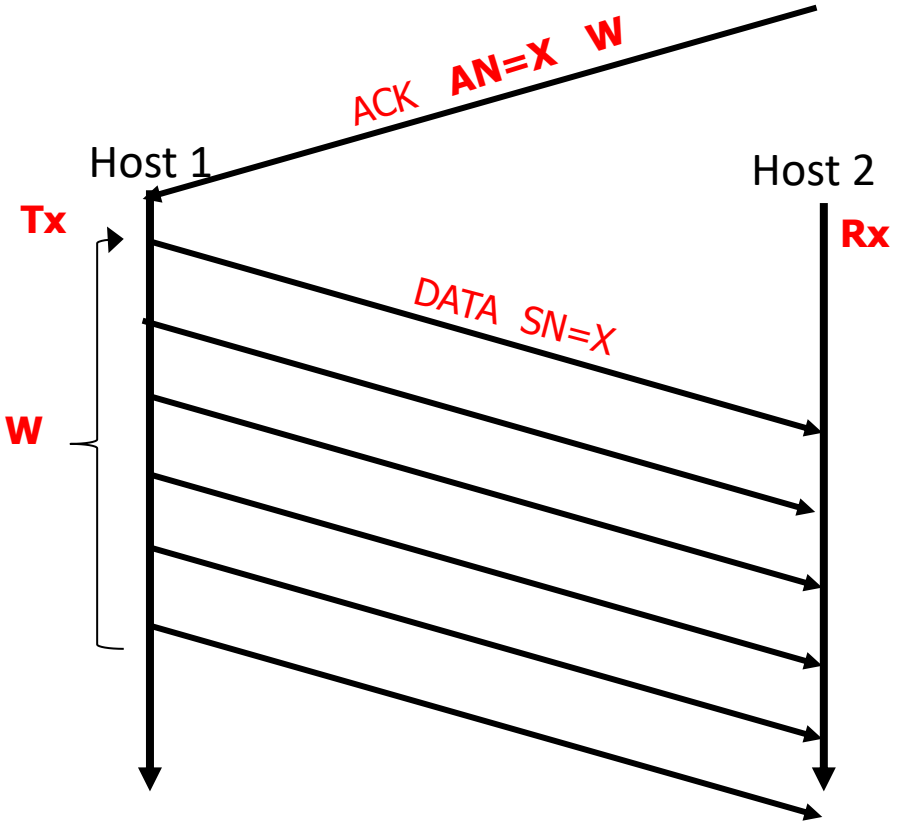


Receive window

Problem: The receiver buffer can overflow.

Solution: The receiver (Rx) sends the rwnd **W**, indicating the number of bytes that the sender (Tx) can send from the **AN Acknowledgment Number**.

When **W=0** it means that the Rx buffer is full and the Tx cannot continue transmitting.

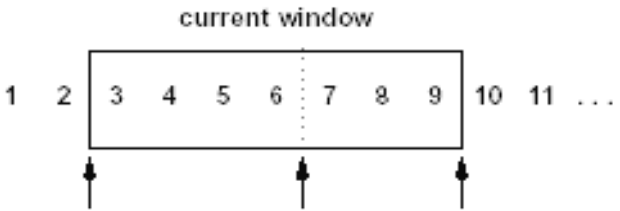


Sliding Window Protocol

The so-called "Sliding Window Protocol" is used for flow control:

Example of the "TCP sliding window":

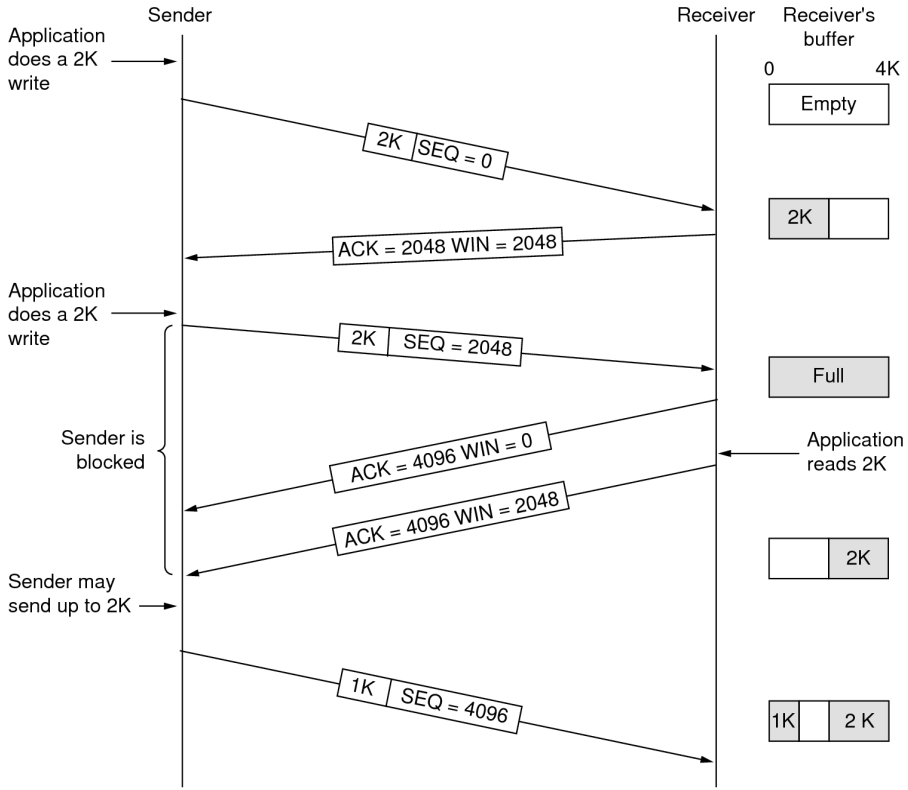
- Octets 1 and 2 were sent and confirmed,
- Octets 3 - 6 have been sent
  but not yet confirmed,
- Octets 7 - 9 are sent next and octets 10, 11,
  can only be sent if the window moves further
  after further confirmations of octet 3.



Sliding Window

- When using the "Sliding Window" mechanism, the receiving transport instance does not need to confirm each segment. The confirmation mechanism works cumulatively.
- The ACK segment indicates by the value of the confirmation number that all previous octets of the data segments have been received correctly.
- The size of the window is determined by the receiving transport instance and essentially by the parameters:
  - Size of the TCP receive buffer,
  - Processing speed of the TCP and application process (depending on the operating system),
  - Bandwidth of the transmission channel

Window management in TCP



Bandwidth-delay-product

- The BDP is an amount of data measured in bits that is equivalent to the maximum amount of data on the network circuit at any given time, i.e., data that has been transmitted but not yet acknowledged
- The **capacity of the transmission channel** C_{BD} (corresponding to **the optimal window size** w_{opt}) can be determined:

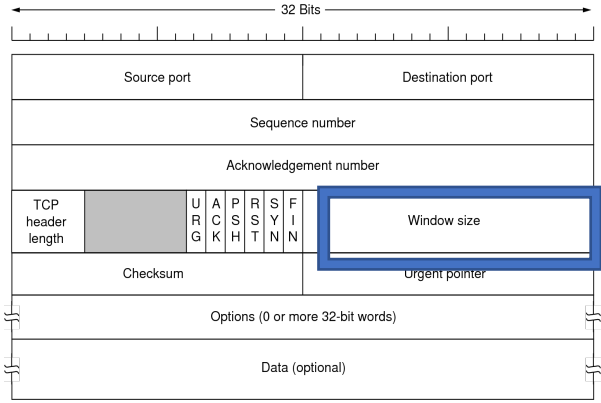
$$\boldsymbol{BDP = w_{opt} = D \cdot T_{RTT}}$$

- The size depends on the RTT value T_{RTT} and the **data rate D** (transmission rate) of the transmission channel.
- At some point we reach the state in which the transmission system is "full" of data segments and ACK segments: We get the so-called **steady state** of a TCP connection.
- If you know the RTT and transmission rate of the channel, you can determine the window size w to achieve the maximum possible throughput, i.e. **full utilization of the bandwidth** of the transmission channel.

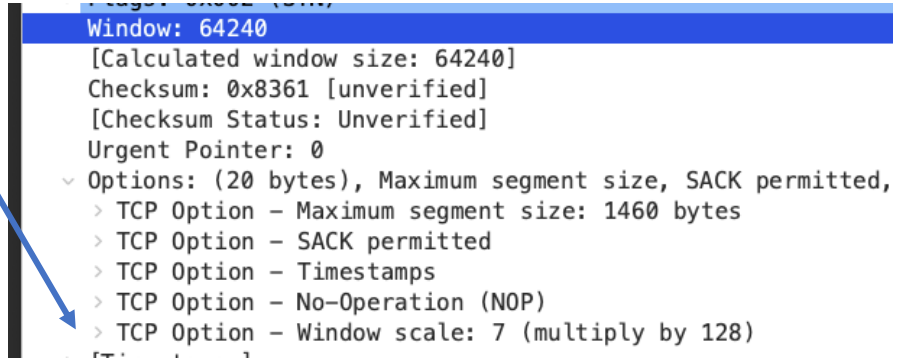
BPD example

Throughput in KB	Windows Size in Kbit
RTT in sec	64
0,5	1024
1	512
2	256
5	102
10	51
20	26
50	10
100	5
200	3

Sliding Window Improvement



- TCP field **Window Size** stores the valid value
- 16bit field provides win. size up to 64k
- This might limit nowadays networks
- RFC 7323 describes TCP Options and RFC 1323 describe TCP high performance
- Used window size is negotiated during the connection establishment
  - Windows scale factor is in the options
    - $2^7 = 128$
  - Scale value from 0 (no shift) to 14.
  - Up to 1 GB possible



BPD example with window scaling

Throughput in KB	Windows Size in Kbit					
RTT in sec	64	128	1024	16384	131072	1048576
0,5	1024	2048	16384	262144	2097152	16777216
1	512	1024	8192	131072	1048576	8388608
2	256	512	4096	65536	524288	4194304
5	102	205	1638	26214	209715	1677722
10	51	102	819	13107	104858	838861
20	26	51	410	6554	52429	419430
50	10	20	164	2621	20972	167772
100	5	10	82	1311	10486	83886
200	3	5	41	655	5243	41943

Congestion Control

- Even with the optimal window size, packets might get lost
- This can be detected if the sender does not receive a confirmation (ACK) within a certain timeout.
- It must be assumed that the packet was discarded by a router in the network due to too high network load; this is, so to speak, a traffic jam on the net. In order to resolve the congestion, all transmitters involved must reduce their network load.
- The sender calculates a **cwnd** "**congestion window**", based on the implicit detection of the state of network congestion.
  - When the Tx detects the **loss of a segment**, it assumes that it was discarded due to severe congestion, then drastically reduces the throughput by **reducing the cwnd**.
  - As it **receives ACKs**, it **increments cwnd** again.

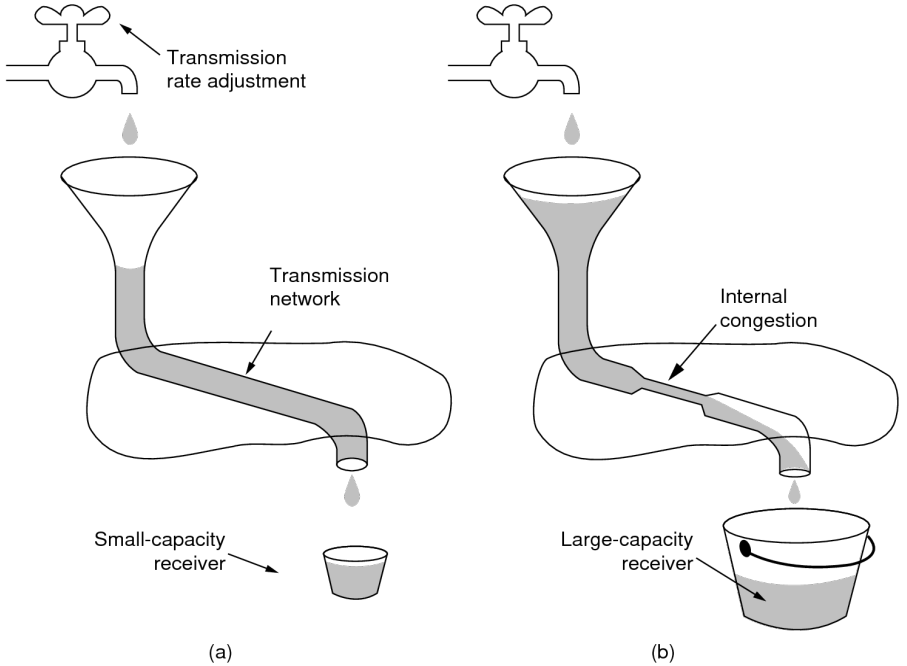
The sender implements the ARQ Sliding Window by choosing the **lowest window** between **rwnd** (**receive window**) and **cwnd**

Congestion monitoring

Two independent problems with overload monitoring in TCP need to be considered. For this purpose, two window sizes are managed in the transmitter

- Sliding Window: Receiver window size (rwin)
- Congestion Window: Network overload window (cwin)

The minimum of both window sizes is always selected.



(a) Fast network, low capacity receiver
(b) Slow network, high capacity receiver

Congestion Control - Algorithms

- For the implementation of CC, four algorithms are defined in *RFC 2581*:
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit
  - Fast Recovery

  which are used together in pairs.

Further improvements are:

- RENO (Jacobson 1990) Fast Recovery
- New RENO (Jacobson 1990) Fast Recovery improvements
- VEGAS (Brakmo & Peterson 1994) Congestión avoidance based on RTT
- SACK, D-SACK (M. Mathis 1996) Innecesary retransmissions detection
- BIC-TCP (L. Xu 2004) Binary Increase Congestion Control
- CUBIC (I. Rhee 2007) Cubic function instead of a linear window increase

Slow Start and Congestion Avoidance

- At the beginning of a data transfer, the slow-start algorithm is used to determine the Congestion Window in order to prevent possible overload management.
- One wants to avoid traffic jams, and since the current utilization of the network is not known, initially small amounts of data are started.
- The algorithm starts with a small window of a Maximum Segment Size (MSS), in which data packets are transmitted from the sender to the receiver.
- The receiver now sends a confirmation (ACK) back to the sender. For each ACK received, the size of the congestion window is increased by one MSS.
- Since an ACK is sent for each package sent upon successful transfer, this leads to a doubling of the *Congestion Window* within a round-trip time.

Slow start II

- At the beginning of the transmission, $w = 1 \cdot p, with$
  - $\boldsymbol{with\ p}$: **size** of a **segment** in bytes, where a constant segment size is assumed for simplicity.

  p <= MSS Maximum Segment Size
- This means that only one data segment is sent and waited for the ACK segment (confirmation), time until ACK is received = RTT.
- After receiving the ACK segment, the window size is set to $w = 2 \cdot p \Rightarrow$ Two data segments are now sent.
- Only after their confirmation the window size is set to $w = 4 \cdot p$ set, etc.
- You can see that more and more data segments and ACK segments are in flight between receiver and transmitter.
- Increased until rwin is reached or packet loss happens

Slow start III

Slow Start

each time Tx receives an ACK

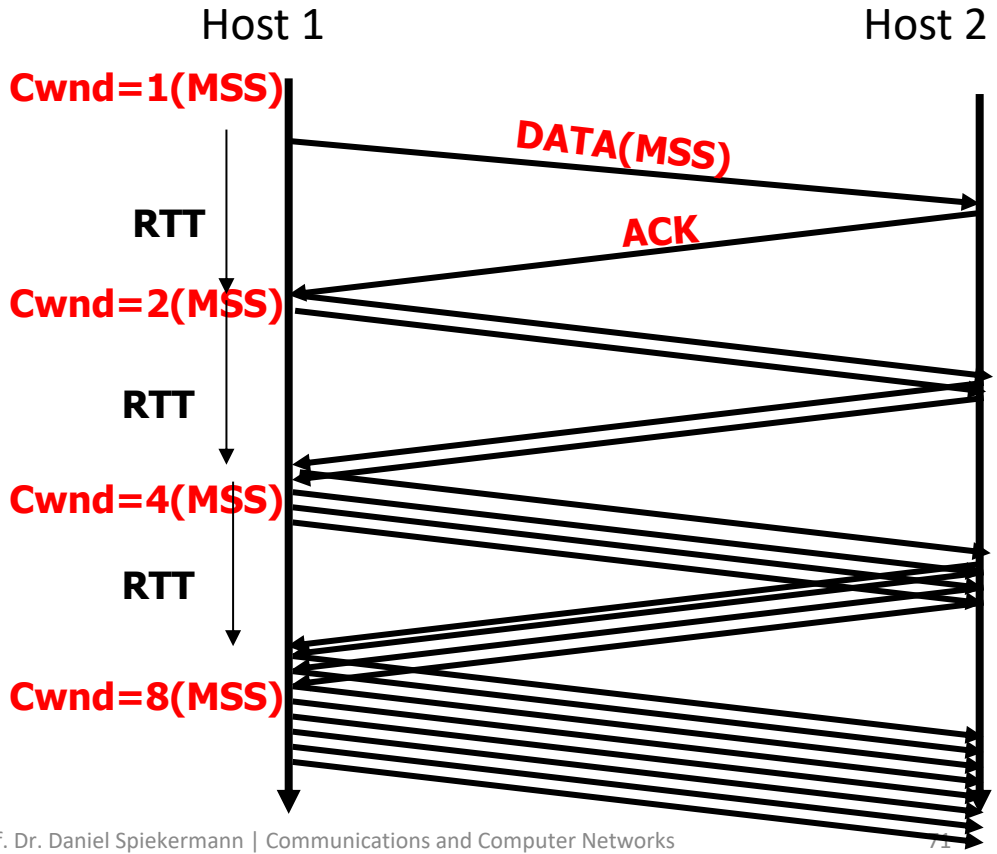
cwnd = cwnd + 1 MSS

At each RTT,

the number of ACKs is doubled,

then

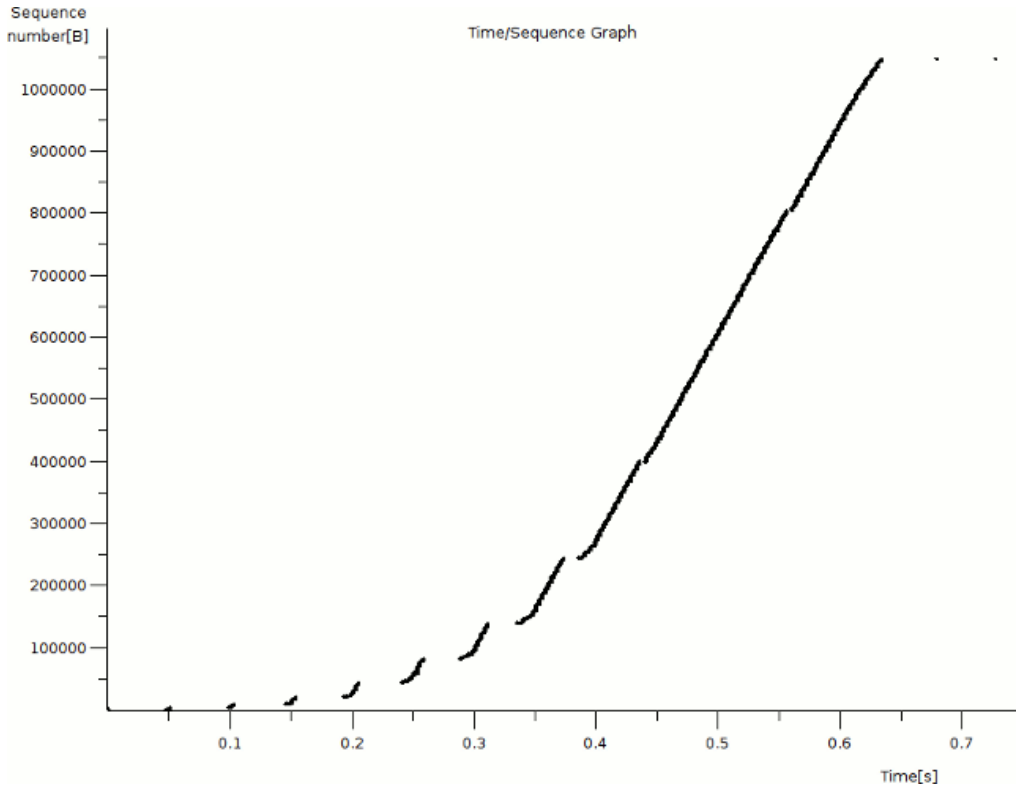
the **cwnd** increases **exponentially**



Slow start IV

- For example, if the window allows two packets to be sent, the sender also receives two ACKs and therefore increases the window by 2 to 4.
  This phase of exponential growth (also called the slow start phase) continues until the so-called Slow Start Threshold is reached.
- After that, the *Congestion Window* is only increased by one MSS if all packets from the window have been successfully transferred. So it only grows by one MSS per round trip time, i.e. only linearly.
- This phase is called the *Congestion Avoidance Phase*. Growth stops when the window size specified by the receiver has been reached.
- If there is a timeout, the *Congestion Window* is reset to 1 and the Slow Start Threshold is lowered to half of the old *Congestion Window*. The phase of exponential growth is thus shortened, so that the window grows only slowly with frequent packet losses.

Slow Start in Wireshark



Congestion Avoidance

Problem:

How to achieve the highest throughput while avoiding congestion?

If the sender continues to increase the throughput (=increasing the cwnd) with each ACK, at some point a segment loss may occur due to severe congestion.

Additive increase/multiplicative decrease

- AIMD is a feedback control algorithm
- Idea: increase the transmission rate (window size), probing for usable bandwidth, until loss occurs

- Linear growth of cwnd when no congestion
- Exponential reduction when congestion detected
  - $w(t + a) = w(t) + a$ (no congestion detected)
  - $w(t + a) = w(t)\ x\ b$ (congestion detected)
    - With w = cwnd, t = time, a = additive increase (MSS), b = multiplicative decrease (0.5)

Congestion Avoidance II

Congestion Control with Congestion Avoidance

When the sender detects a segment loss (= missing ACK), it assumes that it was due to severe congestion

Next steps:

- setting cwnd = 1 MSS to reduce the throughput
- restarts the Slow Start process
- sets ssthresh (slow start threshold) to half the value that the cwnd had when the loss of the segment was detected.

When cwnd >= ssthresh, the sender will go into “congestion avoidance”

- every time it receives an ACK it increases cwnd = MSS + MSS/cwnd

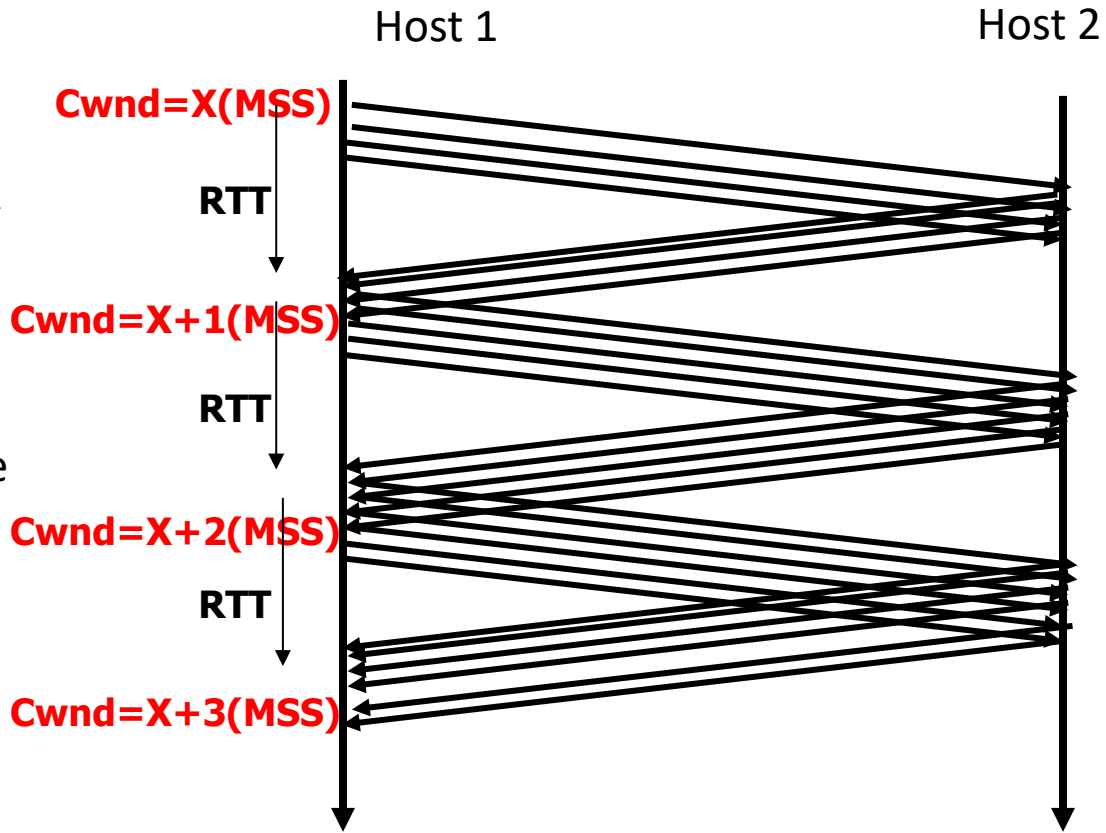
Congestion Avoidance III

Congestion avoidance

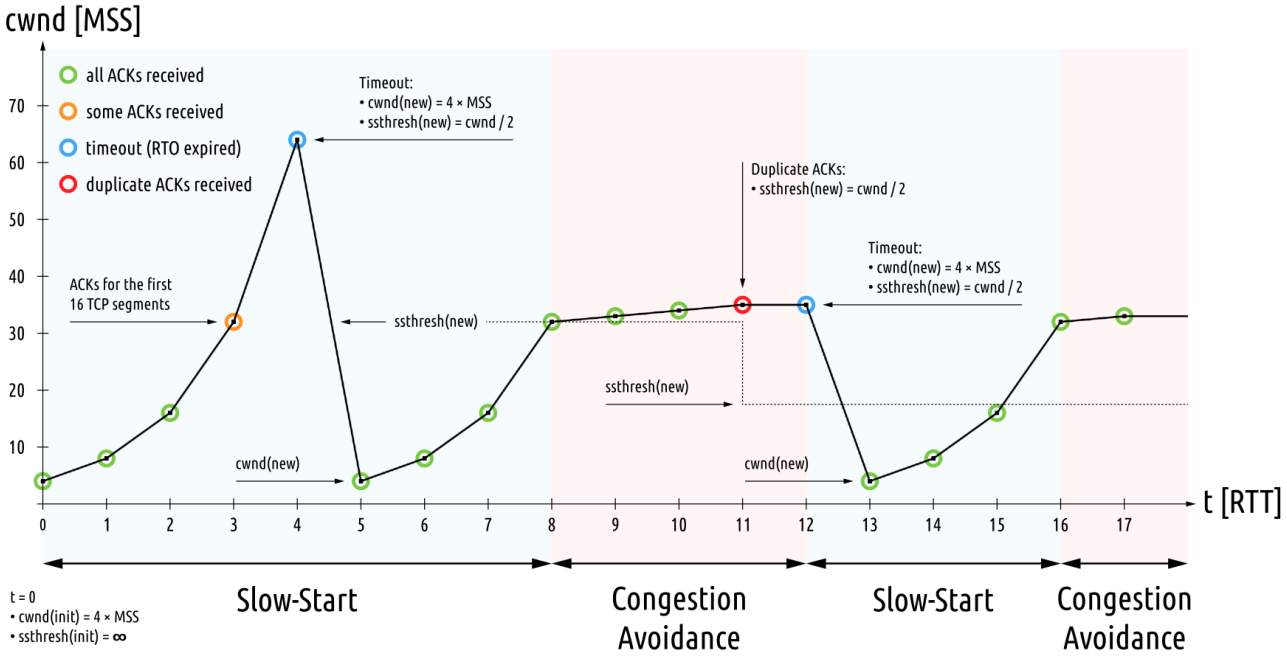
each time Tx receives an ACK

cwnd = cwnd + 1/cwnd

At each RTT, the number of ACKs increases by 1, then the cwnd increases **linearly**



Interaction between slow start and CA



Fast-Retransmit

- Fast-Retransmit and Fast-Recovery are used to react more quickly to the jam situation after a package loss.
- For this purpose, a receiver informs the sender if packets arrive out of line and thus there is a packet loss in between.
- For this purpose, the recipient reconfirms the last correct packet for each additional incoming packet out of line. This is called dup-acks (*duplicate acknowledgments*).
- The sender notices the duplicated confirmations, and after the third duplicate, he immediately resends the lost packet before the timer expires.
- Because there is no need to wait for the timer to expire, the principle is called Fast Retransmit.
- Algorithm implementing Fast Retransmit: TCP Tahoe

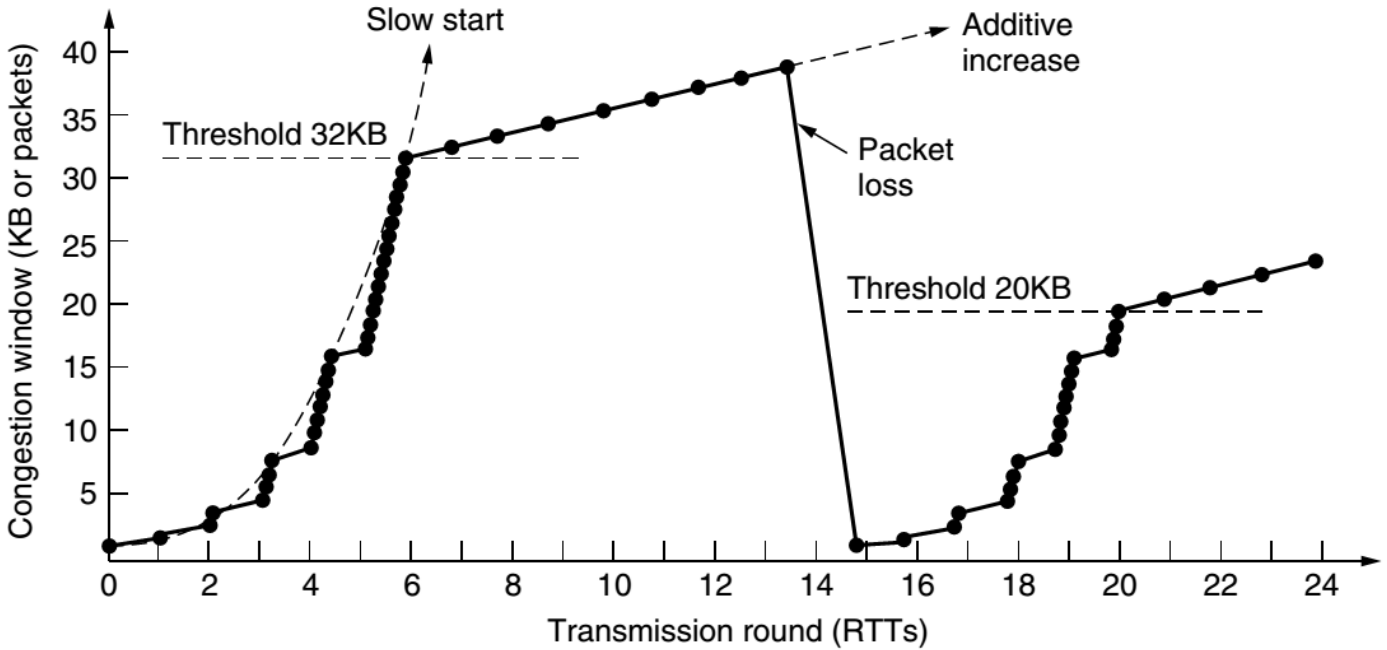
Fast-Recovery

- The *dup-acks* are also indications that although a packet loss took place, the following packages have arrived. Therefore, the transmission time is only halved after the error and not started again with *slow start* as with the timeout.
- In addition, the sending window can be increased by the number of *dup-acks*, because each stands for another packet that has reached the receiver, albeit out of line.
- Since this means that the full transmission power is reached more quickly after the error, the principle is called Fast-Recovery.
- Different algorithms implement Fast Recovery: TCP Reno, TCP New Reno

TCP congestion control

- Slow Start phase
  - doubles the congestion window size(cwnd) in one RTT
  - Initially ssthresh set to ∞
  - Problems during transmission, cwnd = 1 and restart
- If cwnd = ssthresh, congestion avoidence starts
- Fast Retransmit with TCP Tahoe
  - 3 dup-ACKs
    - cwnd = initial cwnd (typically 1 MSS)
    - ssthresh = cwnd/2

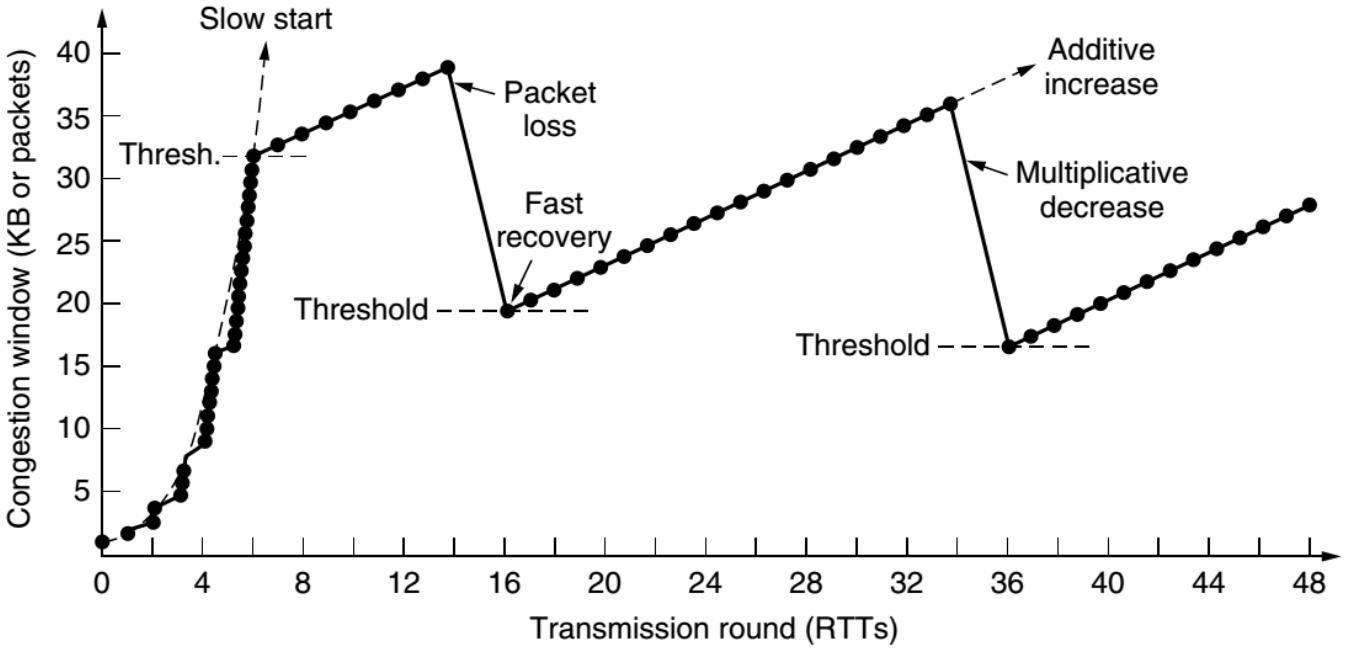
TCP Tahoe


Slow start
Additive increase
Threshold 32KB
Packet loss
Threshold 20KB
Congestion window (KB or packets)
Transmission round (RTTs)
0
2
4
6
8
10
12
14
16
18
20
22
24
5
10
15
20
25
30
35
40

TCP congestion control II

- Slow Start phase
  - doubles the congestion window size(cwnd) in one RTT
  - Initially ssthresh set to ∞
  - Problems during transmission, cwnd = 1 and restart
- If cwnd = ssthresh, congestion avoidence starts
- Fast Recovery with TCP Reno
  - 3 dup-ACKs
    - ssthresh = cwnd/2
    - cwnd = ssthresh

TCP Reno


Slow start
Packet loss
Additive increase
Thresh.
Multiplicative decrease
Fast recovery
Threshold
Threshold
Congestion window (KB or packets)
Transmission round (RTTs)
0
4
8
12
16
20
24
28
32
36
40
44
48
5
10
15
20
25
30
35
40

Fachhochschule
Dortmund
University of Applied Sciences and Arts

TCP Error Control

Prof. Dr. Daniel Spiekermann
Faculty of Computer Science

Problems during transmission

TCP segments, in transit through the network, can be

- **discarded** and do not arrive at destination
- arrive at destination but **altered**
- arrive at destination but **duplicated** (multiple paths)
- arrive at destination but **out of sequence** (different paths)
- Not enough free buffer

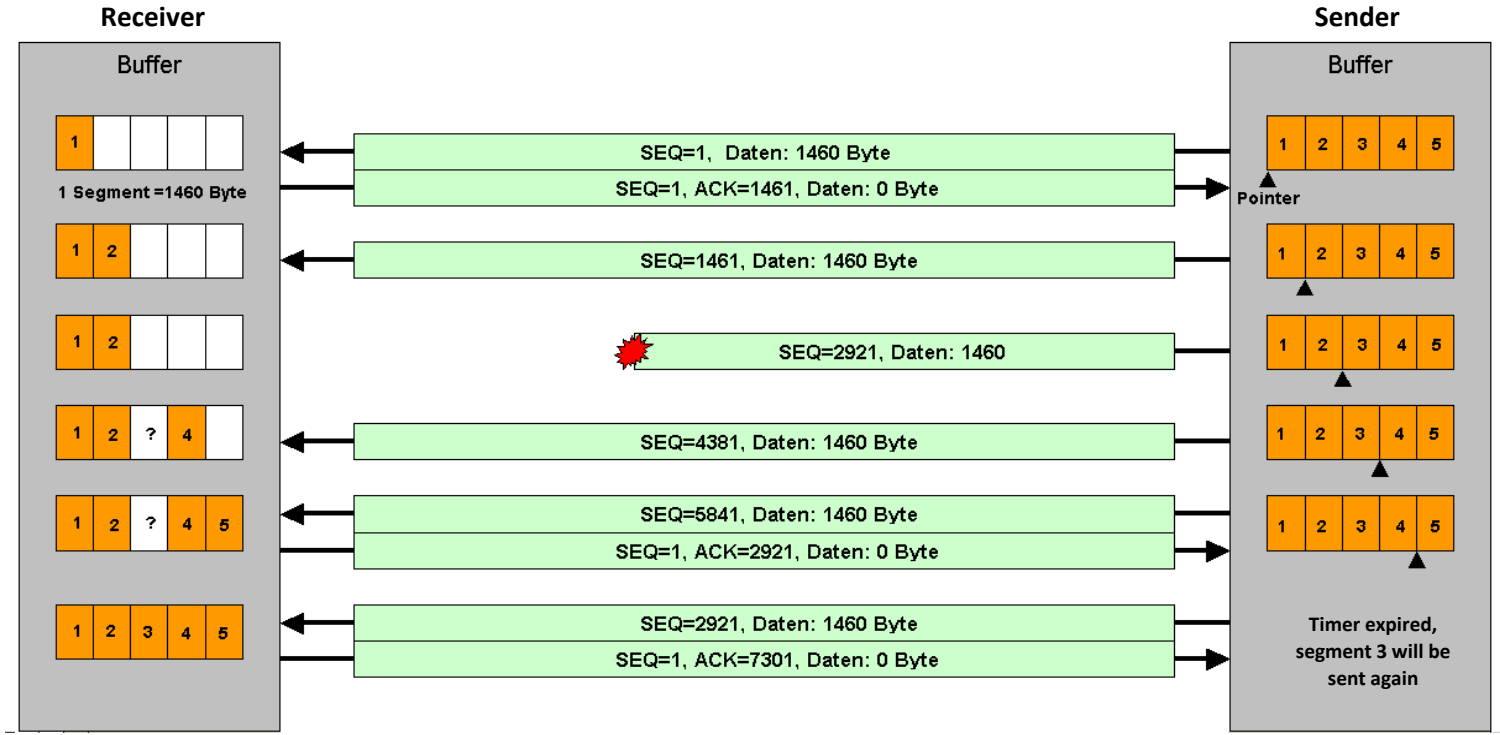
Checksum is used to check the integrity of the header and the data

Calculation similar to the UDP process

Error Management on TCP

- TCP provides FEC with the help of ARQ (Automatic Repeat Request)
- Each segment is ACK'ed with the Flags and AN
- Reason:
  - Missing ACK (lost or packet not arrived)

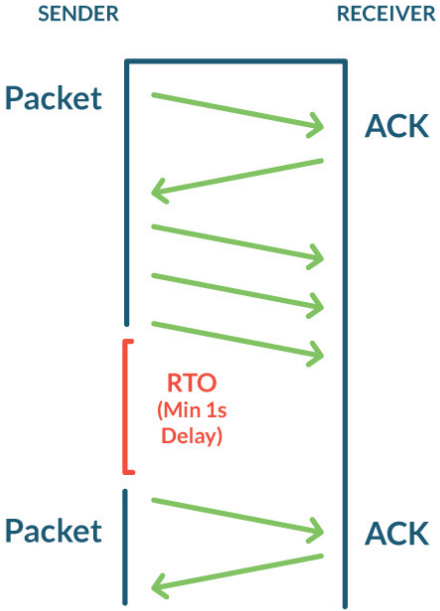
Missing segment



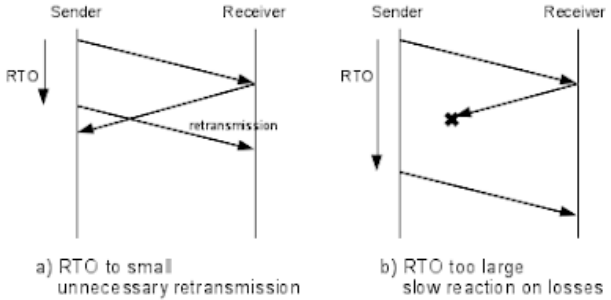
Error Management on TCP

- TCP provides FEC with the help of ARQ (Automatic Repeat Request)
- Each segment is ACK'ed with the Flags and AN
- Reason:
  - Missing ACK (lost or packet not arrived)
  - Timeout

RTO



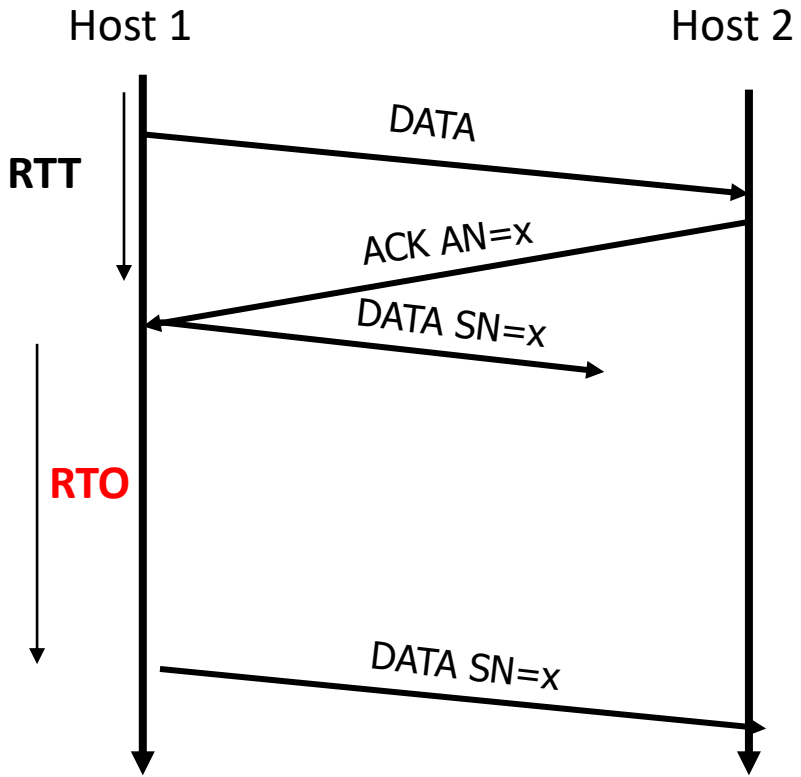
- Retransmission TimeOut
- Time to wait for an ACK before the packet is retransmitted
- Is related to **RTT**
  - If the RTO is too long => the transmission channel is underused
  - If the RTO is too short => unnecessary retransmissions



- **RTO** estimated dynamically, **based on the RTT**.

RTO calculation

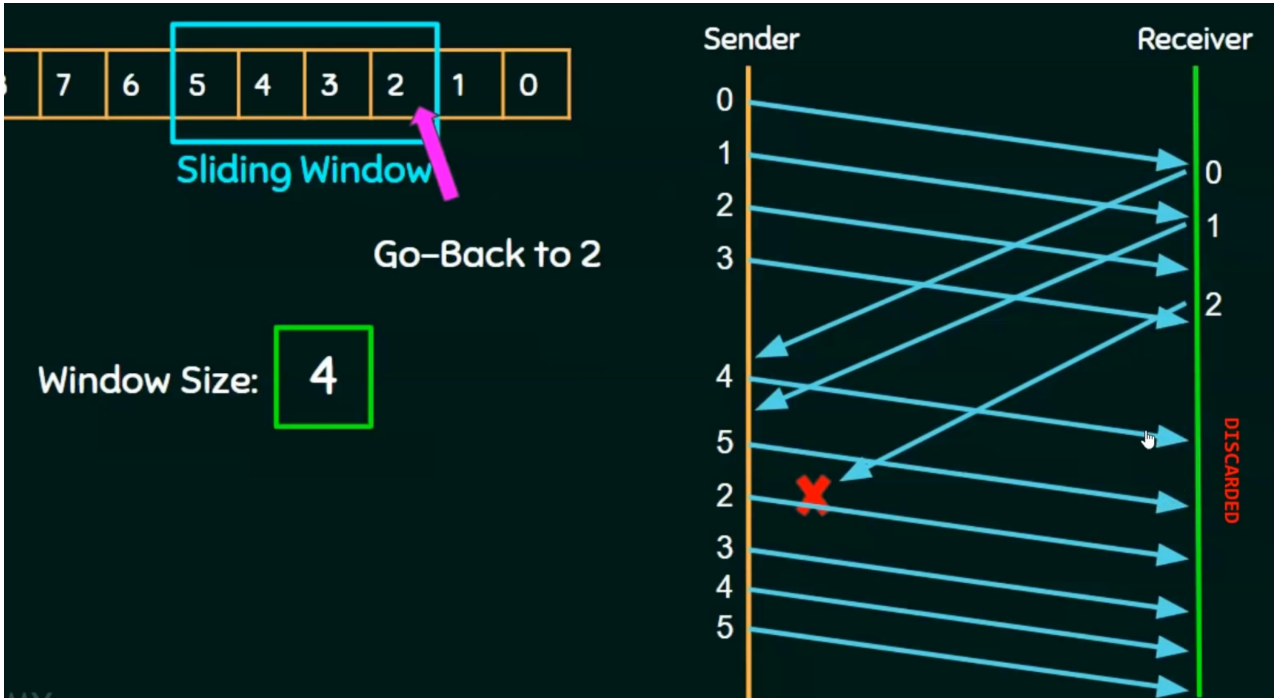
- RTO estimation based on RTT
- Two new variables are introduced
  - smoothed RTT (SRTT)
  - RTT variance (RTTVAR)
- Those two variables are updated, whenever we have a new RTT measurement
- RFC6298:
  RTTVAR = (1 - α) * RTTVAR + β * |SRTT - R‘|
  SRTT = (1 - α) * SRTT + α * R‘
- α and β are parameters that determine how fast we forget the past, R’ is the previous RTT



Error Management on TCP

- TCP provides FEC with the help of ARQ (Automatic Repeat Request)
- Each segment is ACK'ed with the Flags and AN
- Reason:
  - Missing ACK (lost or packet not arrived)
  - Timeout
- This leads the sender to retransmit the missing segment
- TCP uses Go-Back-N ARQ as an improvement of Stop-and-wait ARQ
  - N is Windows size
  - While the missing part is not ACK'ed, further packets are send
  - After the timeout, the first and all following frames of the Window size is sent

Go-Back-N-ARQ



Window Size is 4
4 packets sent without ACK

ACK for pkt 2 is missing
- missing pkt
- missing ACK
- RTO

Transmission of next pkt “stopped”
All pkts starting from missing one are sent again

RST flag

- RST flag
  - signifies that the connection should immediately be terminated
  - client doesn't expect an ACK
  - Server does not need to send a FIN/ACK exchange to terminate the connection
- Reasons:
  - The packet is an initial SYN packet trying to establish a connection to a server port on which no process is listening.
  - The packet arrives on a TCP connection that was previously established, but the local application already closed its socket or exited and the OS closed the socket.

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Transport Layer - QUIC

Prof. Dr. Daniel Spiekermann
Faculty of Computer Science

QUIC - History

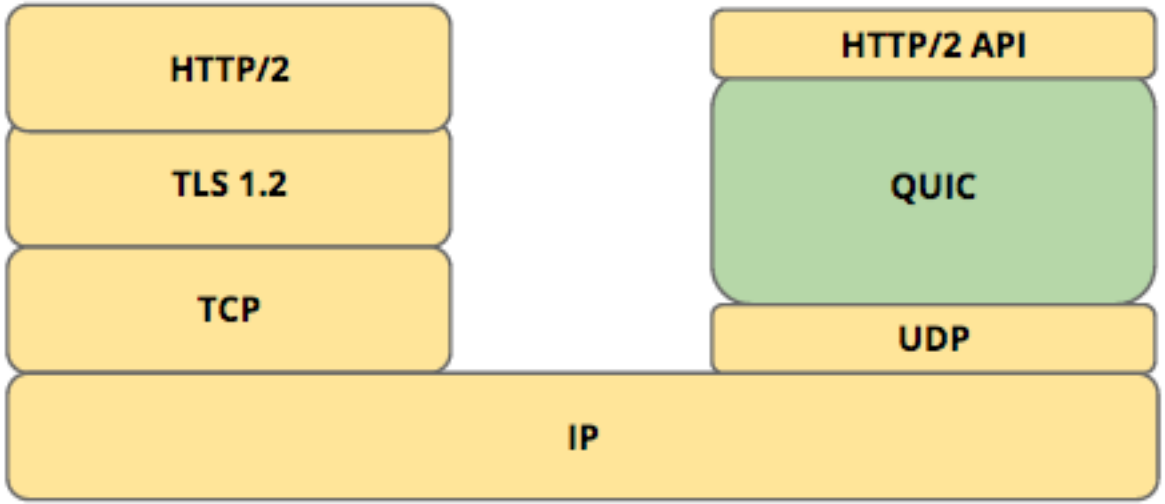
- TCP has different problems
  - RTT and ARQ
  - Overhead
  - Privacy
- Intention to create a new L4 protocol for an improved web access
  - Designed by Jim Roskind at Google in 2012
  - Deployed in 2013
  - Chrome 29 (2013) was the first browser implementing QUIC
  - Firefox since v72 (2019), Safari since v104 (2020)
- Formally known as Quick UDP Internet Connection, today it is only QUIC as the name
- In May 2021 standardized in RFC 9000

QUIC

- Creation of a new L4-protocol (equal to TCP or UDP) nearly impossible
  - Requires a complete new TCP/IP-protocol stack on every system
  - Therefore, QUIC needs an existing L4-protocol (TCP or UDP)
- UDP and its simplicity was chosen for the basement

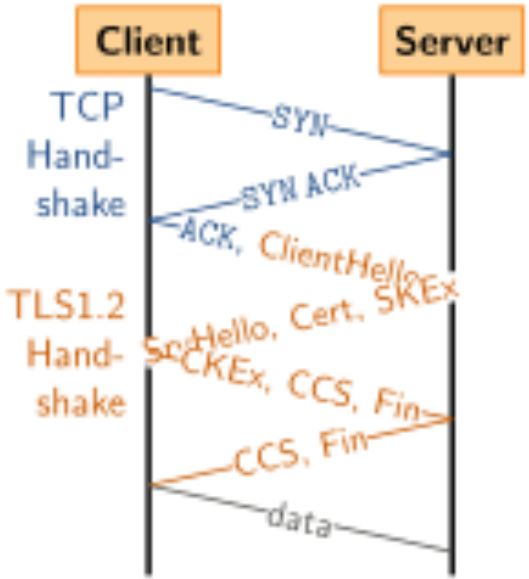
- Design goals:
  - Improving of performance of web application
  - Reduced connection and transport latency
  - Moving of congestion control into user space
  - Provides extension of FEC
  - Avoidance of protocol ossification

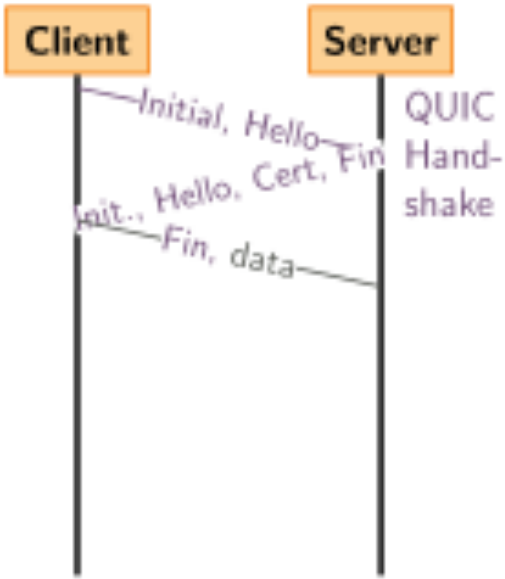
QUIC in OSI reference model



Comparison TCP / TLS and QUIC

- Faster transmission of data
- In-built use of encryption





QUIC internals

- Communication starts with connection establishment
- Key items: **Initial** and **Handshake** packet
- Two different header formats
  - Long header (used for initial comm.)
  - Short header (used for data)
- TLS records are used for encryption
- CRYPTO frames transfer this data

```
QUIC    1242 Initial, DCID=ac28c231a601f020, PKN: 0, CRYPTO, PADDING
QUIC    1242 Initial, SCID=ec28c231a601f020, PKN: 1, ACK, CRYPTO, PADDING
QUIC    1242 Initial, DCID=ec28c231a601f020, PKN: 1, ACK, PADDING
QUIC    1242 Handshake, SCID=ec28c231a601f020
QUIC    1242 Handshake, SCID=ec28c231a601f020
QUIC    1242 Handshake, SCID=ec28c231a601f020
QUIC    1242 Handshake, SCID=ec28c231a601f020
QUIC    1242 Handshake, SCID=ec28c231a601f020
QUIC      81 Handshake, DCID=ec28c231a601f020
QUIC    1105 Protected Payload (KP0)
QUIC      81 Handshake, DCID=ec28c231a601f020
QUIC     115 Handshake, DCID=ec28c231a601f020
QUIC      73 Protected Payload (KP0), DCID=ec28c231a601f020
QUIC      96 Protected Payload (KP0), DCID=ec28c231a601f020
```

Fachhochschule Dortmund
University of Applied Sciences and Arts

QUIC internals II - Initial

- Uses the long header
- Client sends Initial with set Desination Connection ID (DCID)
- Server answers with set SCID
- DCID and SCID can be set vice versa
- TLSv1.3 record layer used for exchanging encryption parameters

```
..00 .... = Packet Type: Initial (0)
.... 00.. = Reserved: 0
.... ..00 = Packet Number Length: 1 bytes (0)
Version: 1 (0x00000001)
Destination Connection ID Length: 0
Source Connection ID Length: 8
Source Connection ID: ec28c231a601f020
Token Length: 0
Length: 1182
Packet Number: 1
Payload: a3b482162d18b930bfaf5f09b3fb38016ea638aada572e62ab01b7f093e4919081
ACK
  Frame Type: ACK (0x0000000000000002)
  Largest Acknowledged: 0
  ACK Delay: 0
  ACK Range Count: 0
  First ACK Range: 0
CRYPTO
  Frame Type: CRYPTO (0x0000000000000006)
  Offset: 0
  Length: 90
  Crypto Data
  TLSv1.3 Record Layer: Handshake Protocol: Server Hello
    Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 86
      Version: TLS 1.2 (0x0303)
      Random: a76125ef205ca2b5605c1c346883d0b7d11676f582ea7d116d1090285acca
      Session ID Length: 0
      Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
      Compression Method: null (0)
      Extensions Length: 46
```

QUIC internals III - Handshake

- Uses the long header
- Transfer the rest of the TLS-session
- 0-RTT or 1-RTT
  - 0-RTT uses known enc. Param
    fewer packets for conn. establishment
  - 1-RTT
    Needed for first connection or conn.
    not used for a longer period
    more packets needed

```
QUIC IETF
  QUIC Connection information
    [Connection Number: 0]
  [Packet Length: 1200]
  1... .... = Header Form: Long Header (1)
  .1.. .... = Fixed Bit: True
  ..10 .... = Packet Type: Handshake (2)
  Version: 1 (0x00000001)
  Destination Connection ID Length: 0
  Source Connection ID Length: 8
  Source Connection ID: ec28c231a601f020
  Length: 1183
  [Expert Info (Warning/Decryption): Failed to creat
  Remaining Payload: 8ca50c5a760971efca9d829af46cc57
```

QUIC internals IV - Protected payload

- Used to transfer the data
- Uses the short header

```
> Frame 14281: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on i
> Ethernet II, Src: Cisco_ac:bc:00 (00:1c:b1:ac:bc:00), Dst: Apple_ae:7c:ef (
> Internet Protocol Version 4, Src: 172.217.16.206, Dst: 172.22.213.230
> User Datagram Protocol, Src Port: 443, Dst Port: 51560
v QUIC IETF
  v QUIC Connection information
      [Connection Number: 0]
    [Packet Length: 26]
  v QUIC Short Header
      0... .... = Header Form: Short Header (0)
      .1.. .... = Fixed Bit: True
      ..0. .... = Spin Bit: False
    Remaining Payload: 04eb83039fe7fe734dcec64676b5c0840101bfeedfd5c69338
```

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Additional slides

Prof. Dr. Daniel Spiekermann
Faculty of Computer Science