6. Transport Layer

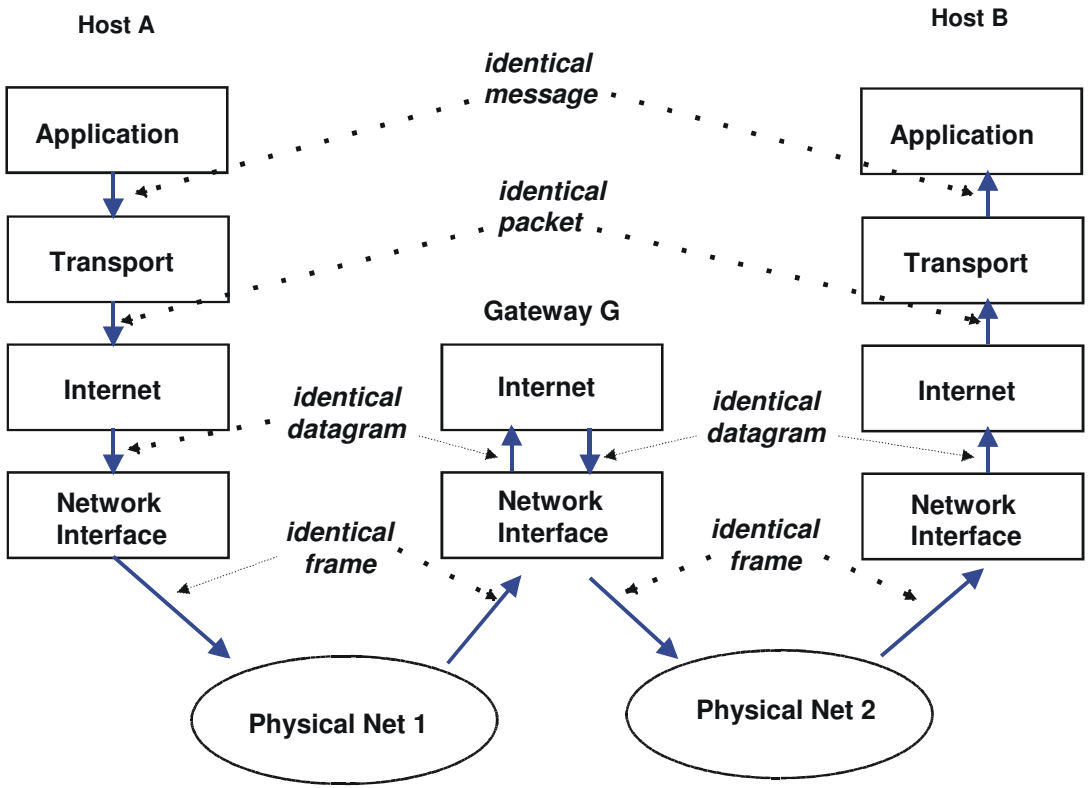
6.1 Internet Transport Layer Architecture

6.2 UDP (User Datagram Protocol)

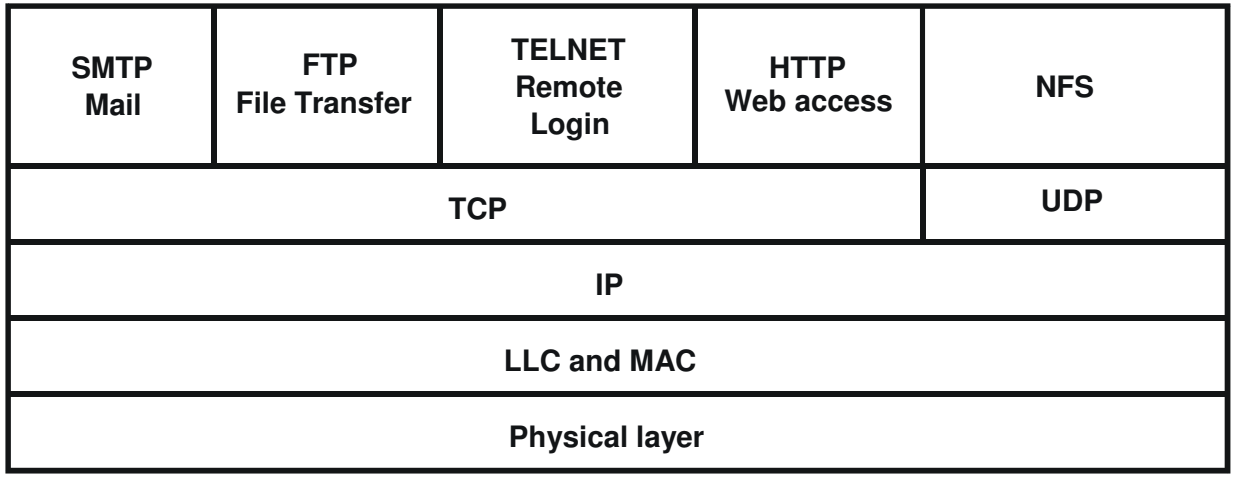
6.3 TCP (Transmission Control Protocol)

6.1 Internet Transport Layer Architecture

The Internet transport protocols are **end-to-end** protocols!



Important INTERNET Protocols



SMTP	=	Simple Mail Transfer Protocol
FTP	=	File Transfer Protocol
TELNET	=	Remote Login Protocol
UDP	=	User Datagram Protocol
NFS	=	Network File System
TCP	=	Transmission Protocol
IP	=	Internet Protocol
LLC	=	Logical Link Control
MAC	=	Media Access Control

Addressing of Service Processes: Ports

- The layer 4 address serves to identify a certain type of service (application type) which is assigned to an application process on the host computer.
- Addressing by process number (process ID) would be unsuitable because processes are generated and terminated dynamically by the operating system; therefore process numbers are unknown outside the local system.
- The mapping between a service and a process is not necessarily 1:1:
  - One process may provide several services.
  - Several processes may provide the same service.
- Thus we introduce the concept of an abstract communication end point: a **port**

Port Characteristics

- One service is assigned to exactly one port.
- Several connections can run over the same port at the same time.
- Asynchronous and synchronous port access is possible.
- Each port is associated with a buffer.
- The port provides an application programming interface (API).

Examples for Reserved Port Numbers

Some port allocations ("well-known addresses")

Decimal	Key word	Unix Key word	Description
0			Reserved
.	.	.	.
.	.	.	.
20	FTP-DATA	ftp-data	File Transfer Protocol (data)
21	FTP	ftp	File Transfer Protocol
23	TELNET	telnet	Terminal Connection
25	SMTP	smtp	Simple Mail Transfer Protocl
42	NAME-SERVER	name	Host Name Server
43	NICNAME	whois	Who Is

6.2 UDP (User Datagram Protocol)

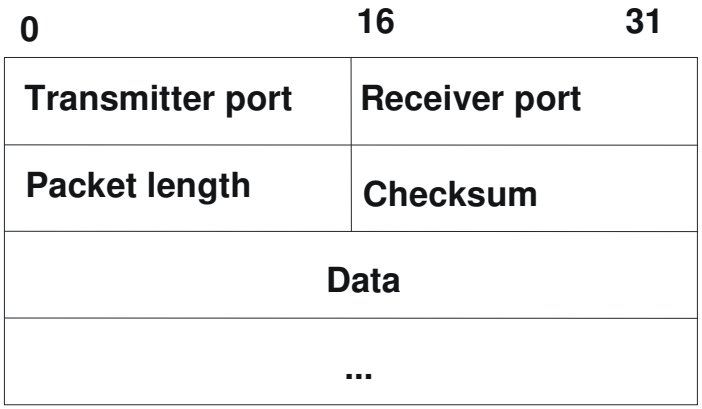
UDP is the **unreliable**, **connectionless** datagram transport protocol of the Internet. It basically serves to offer a programming interface for direct IP access to applications, supplemented by port addressing.

Characteristics

- Datagram transport
  - No guaranteed delivery of the packets to the receiver
  - No automatic retransmission in case of bit errors
  - No flow control
  - No congestion control
  - No guarantee of correct packet order at the receiver
- Multicast is possible

UDP Packet Format

UDP-Header



- The packet length is counted in bytes (including UDP header)
- The checksum is built over header fields and the user data field for error detection. A bit-column-wise EXOR is computed over the 16-bit words of the packet.
- The use of the checksum is optional (if UDP is carried over IPv4). It is mandatory if IPv6 is used.

Characteristics of UDP

- Minimal consumption of resources (storage, CPU time), very efficient
- No explicit connection establishment, minimal protocol message overhead
- Easy implementation

UDP is particularly suitable for simple one-way or client-server interactions. Examples are status reports or request/response protocols:

- a request packet from the client to the server
- a response packet from the server to the client

Examples for the Use of UDP

- Domain Name Service (DNS)
- SNMP: Simple Network Management Protocol
- NFS: Network File System
- many multimedia protocols that do not want error protection in layer 4
- all multicast protocols, including RTP for real time applications. Often used for real-time audio and video streams.

6.3 TCP (Transmission Control Protocol)

TCP is the first protocol in the Internet protocol hierarchy that achieves a **secured data communication between end systems**.

Characteristics

- **Stream-oriented**: TCP transports a serial bit stream of the application in the form of 8-bit bytes.
- **Connection-oriented**: Before data transmission begins a *connection* is established between both communication partners. Error control and flow control operate on each connection separately.
- **Segmentation**: The sequential data stream (byte stream) is cut up into *segments* (packets) for transmission.
- **Duplex communication**: A TCP connection is full-duplex: data can be transported in both directions at the same time.

What is contained in a TCP standard?

The TCP standard (RFC 793) specifies

- the formats of data packets and control information
- the procedures for
  - connection establishment and takedown
  - error detection and error correction
  - flow control
  - congestion control (by „abusing“ flow control)

RFC 793 does **not** specify the interface to the application program (*socket*).
This is local matter.

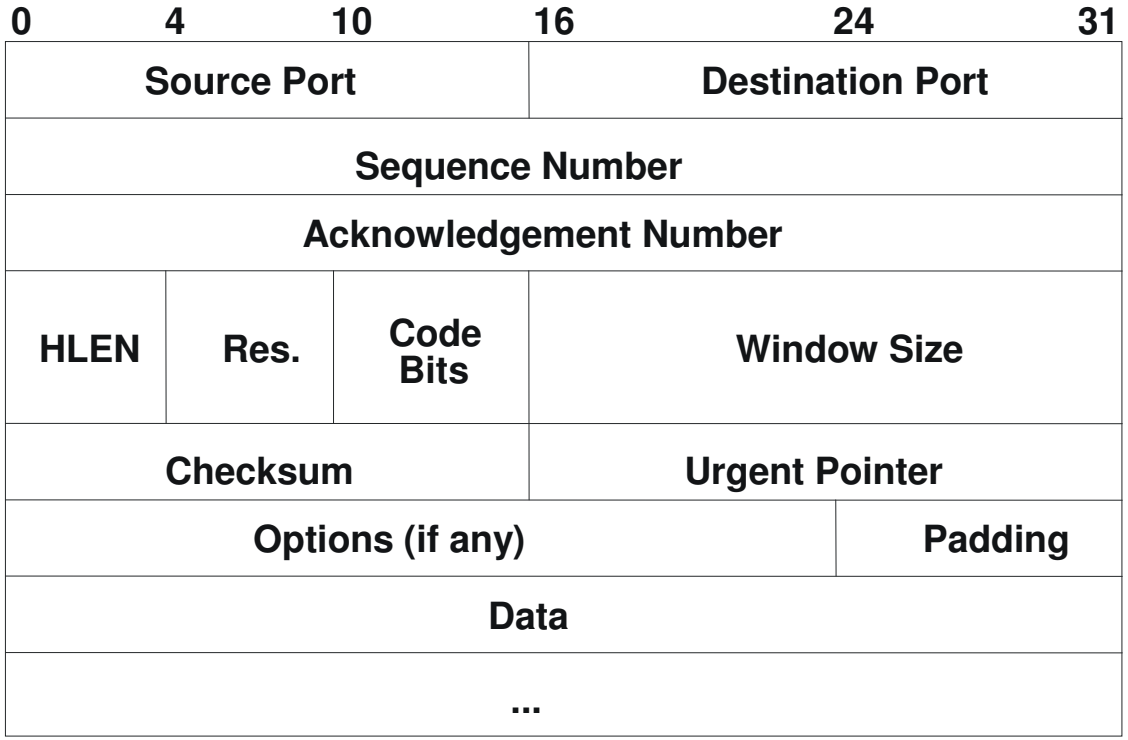
Addressing

A TCP connection is uniquely determined by a quintuple of

- IP addresses of the transmitter and the receiver
- Port addresses of the sender and the receiver
- TCP protocol identifier

Packet Format

Format of the TCP header



Data Fields in the TCP Header (1)

SEQUENCE NUMBER **ACKNOWLEDGMENT**	Byte numbers
HLEN	Header length = Offset of the data field
CODEBITS	(6 bits from left to rigth)
URG	Urgent Pointer is used
ACK	Ack Number field is valid
PSH	Push
RST	Reset of the connection
SYN	Synchronize sequence numbers
FIN	End of the data stream
WINDOW SIZE	Window size in bytes
URGENT POINTER	Byte Offset to the current sequence number at witch important data begins

Data Fields in the TCP Header (2)

- **Port number:** like for UDP.
- **Sequence number:** the logical position of the first byte in the user data field within the overall byte stream.
- **Acknowledgement number:** Identifes the next byte in the data stream that the receiver expects from the the sender. Note that not individual packets but byte positions in the logical data stream are confirmed. Thus cumulation of confirmations over several TCP packets is easily possible. Acknowledgements control the transmission repetition in the event of an error. They also define the sliding window for flow control. Since data can flow into both directions between the communication partners (duplex operation), there is a sequence number sequence for each direction.
- **Header length (HLEN):** Size of the TCP header in 32 bit words.

Data Fields in the TCP Header (3)

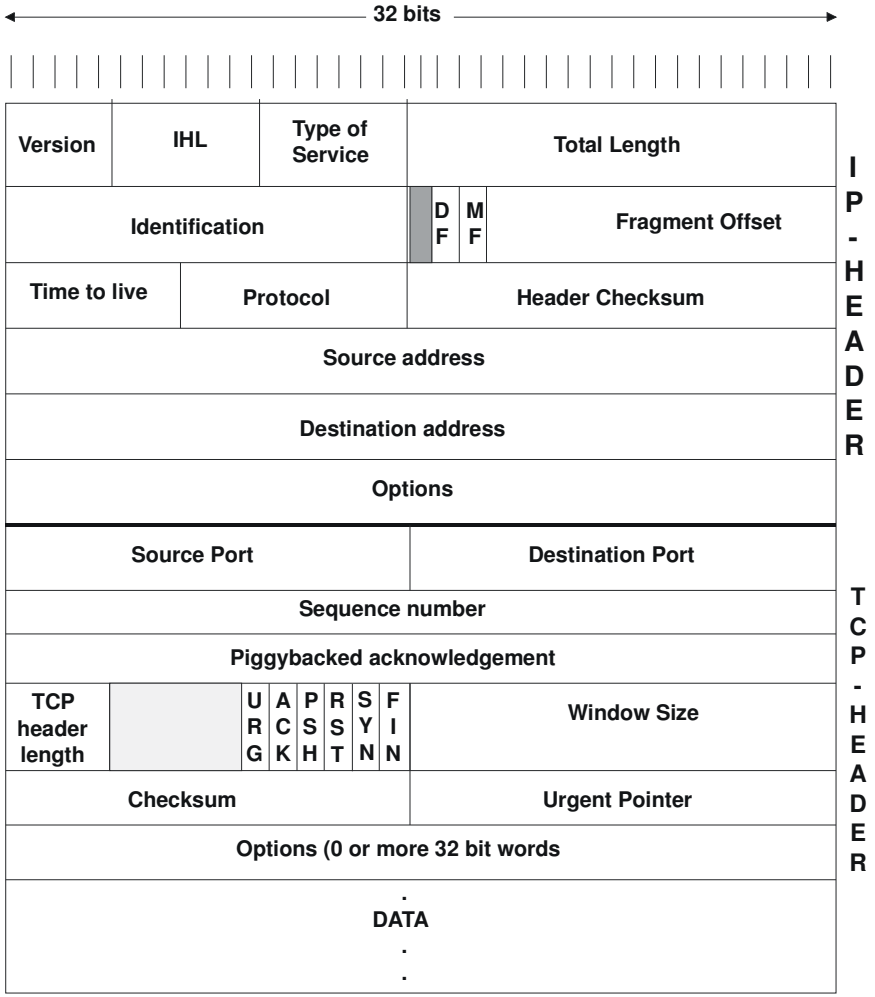
- **Codebits** (Flags)
  - URG: urgent pointer field is valid ("in use")
  - ACK: acknowledgement field is valid ("in use")
  - PSH: (push) The receiver is to provide data to the application as fast as possible.
  - RST: Reset the connection
  - SYN: Synchronisation of the initials sequence numbers with connection establishment
  - FIN: The transmitter has ended the transfer of its data.
- **Window size:** Number of bytes, that the transmitter of this packet can take, until its buffer is full (flow control).

Data Fields in the TCP Header (4)

- **Checksum:** The checksum is computed over the entire fragment, including UDP header. A column-wise EXOR is computed over the "vertically over each other written" 16-Bit words of the packet (like with UDP).
- the "urgent pointer" identifies the last byte in the data segment, which should be processed with special priority. The data following thereafter has normal priority.
- **Urgent pointer:** The "urgent pointer" identifies the last byte in the data division, which should be processed with special priority. The data following thereafter have normal priority.
- **Options:** (we will talk about this later on)

The data segment of a TCP packet is optional, an empty TCP packet can for example be sent as a pure confirmation of received data if no data has to be sent in the back direction at this point.

TCP/IP: Format of the entire Header

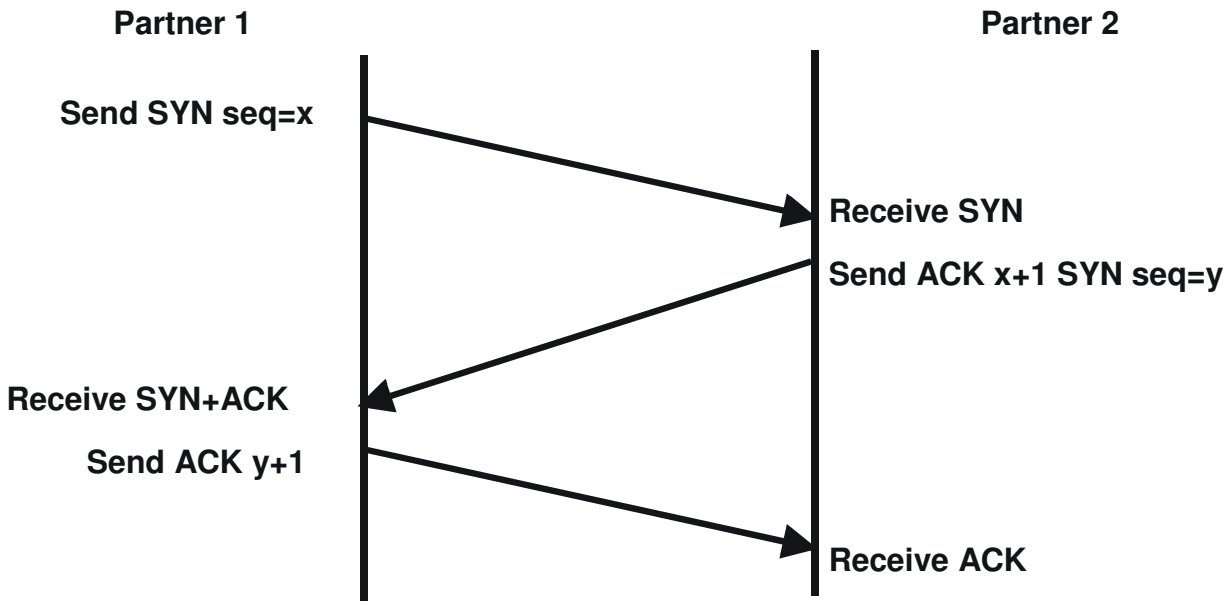


Important Functions in TCP

- Positive acknowledgement or retransmission (**PAR**)
- Transmission repetition after timeout at the transmitter. No NACKs!
- "Sliding Window" mechanis for flow control. Variable window size: with each acknowledgement of the receiver, the window size which is to be used from now on, is also transmitted.
- Error recognition by check sum.
- Piggybacking: Control information on the way back and data can be transmitted in the same TCP packet.
- Out-of-Band Data: important information is to be delivered to the receiver, previous to processing of data sent before . Example: Alarm notifications.

TCP Connection Establishment

Three-Way-Handshake: Connection Establishment by three packets:



With the connection establishment also the initial sequence numbers of both sides (both directions) are exchanged and confirmed.

Three-Way-Handshake

- The SYN flag indicates that the sequence numbers are to be synchronized.
- SYN "costs" one byte with regard to the allocation of sequence numbers.
- Pure acknowledgements do not "cost" bytes.

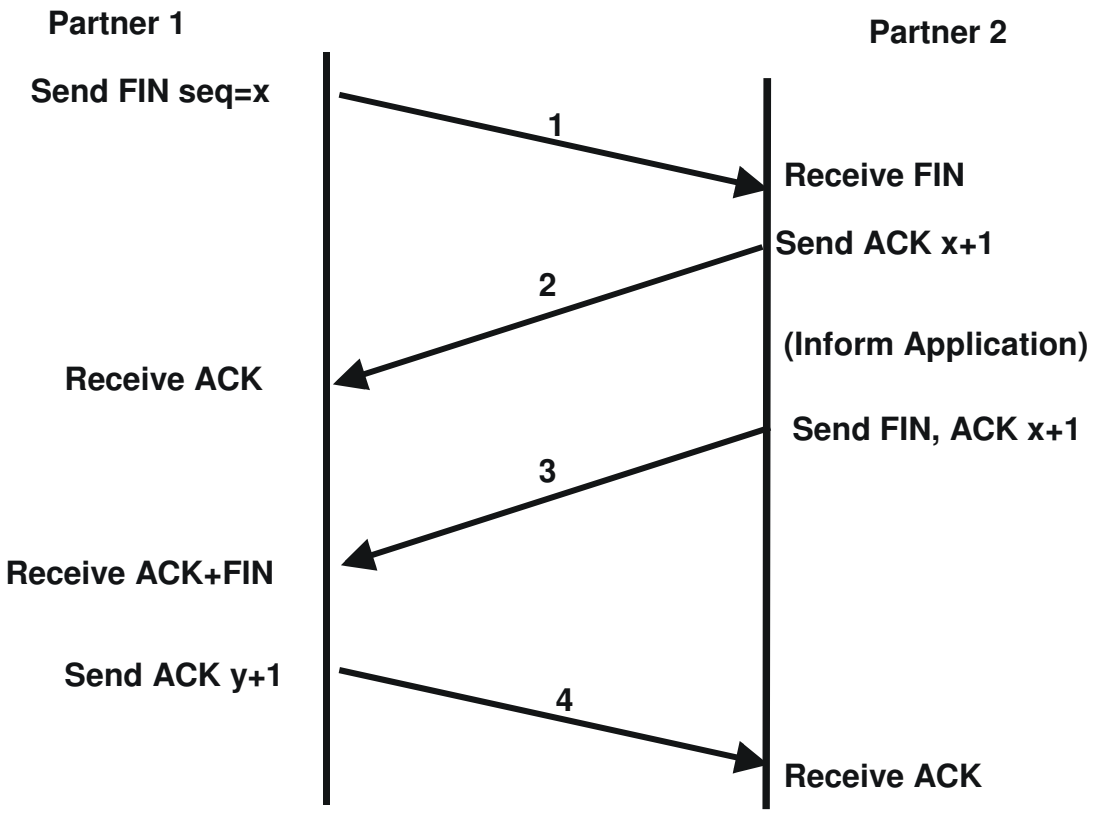
Timeout with the Connection Establishment

What happens, if the communication partner does not answer?

- The transmission of the packet is repeated, TCP regards this as usual packet loss.
- After a fixed time period (timeout) the connection attempt is aborted and the application is informed.

TCP Connection Establishment (1)

Arranged connection clearing by four packets:



TCP Connection Establishment (2)

Takedown of a connection by two times "half-close":

- Since the TCP connection is bi-directional, both directions should always be terminated separately from each other.
- The one who would like to terminate its transmission role, sets the FIN flag.
- FIN "costs" one byte and is therefore confirmed by an acknowledgement.
- The other participant could send further - however in practice one almost always sees the behavior that the other participant as a reaction also terminates its transmission role.

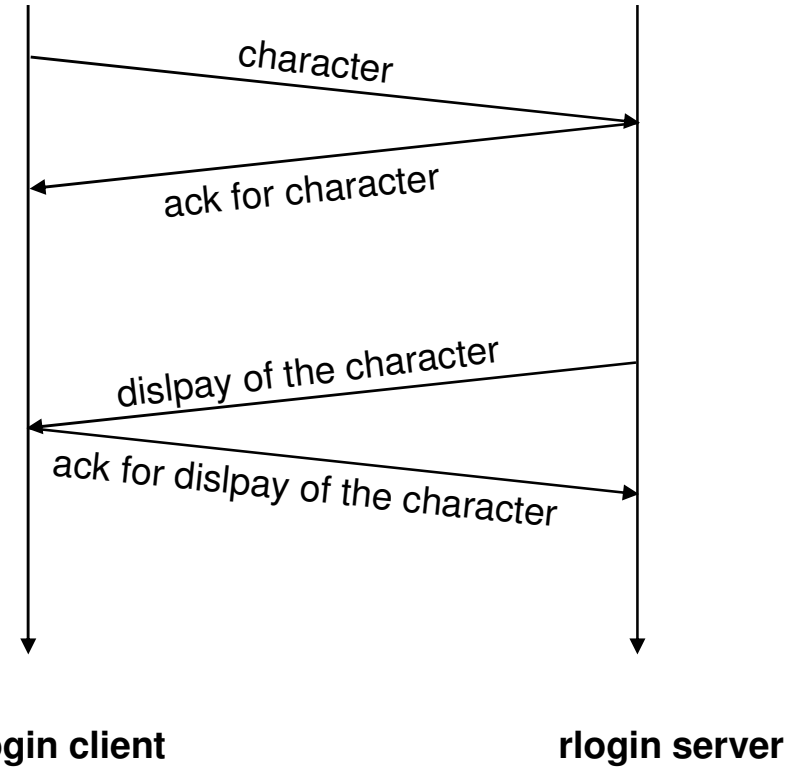
TCP Reset

A packet, in which the RST bit is set in the TCP header, terminates the connection in form of an "abortive release" (in contrast to an "orderly release" with FIN):

- All data, which are buffered at the transmitter, are dropped, and the reset packet is immediately sent. The connection is thereby instantly closed from the view of the RST transmitter.
- With reset data can be lost (that does not happen with the connection clearing with FIN).
- The receiver of a RST packet notifies the application and immediately terminates the connection.

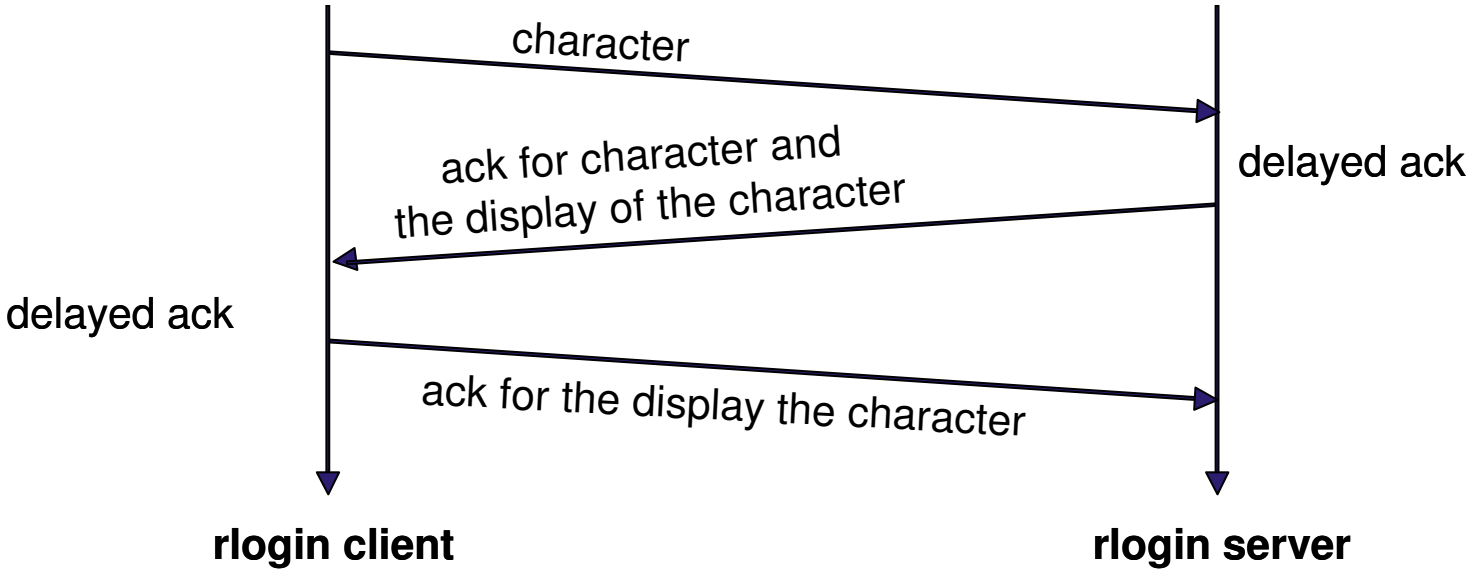
Typical Data Flow with TCP

Data flow example of an interactive application



Delayed Acknowledgements (1)

In order to prevent that redundant TCP packets, which only contain one ACK, are sent, sending of ACKs is frequently delayed:



Delayed Acknowledgements (2)

- An ACK is usually delayed around a maximum of 200 ms with "delayed acknowledgements"
- During this time data, which is sent in the opposite direction, can carry the ACK "piggybacked", this saves the transmission of a separate ACK packet.
- If there is no data to be sent within this time, a pure ACK packet without data is transferred.
- The 200 ms timer is not drawn up for each packet, but runs globally, i.e. every 200 ms all ACKs, which are still open, are sent. The timer however is of course connectio-oriented, i.e., for each TCP connection and each transmission direction there is a private timer.

Mass Data Transfer

What happens, if large amount of data is transfered by TCP (bulk data flow)? When is an ACK sent?

- So far: delayed ACK after 200 ms
- That would cause too many unconfirmed packets in the transit, if the data rate is high (bulk data flow).
- Therefore with the mass data transfer every second packet is confirmed immediately, even if the 200-ms-Timer did not run off yet.

Error Protection in TCP

- TCP divides the byte stream into units, which will be transfered each in one IP packet. These units are called **segments**.
- After TCP sent a segment by IP, a timer for this segment is started.
- If no confirmation of the successful receipt of this segment arrives within the timers running time, the transmission is repeated.
- The timer dynamically adapts to the "normal" round trip time of the connection.
- If a TCP receiver receives an error free segment from the transmitter, it sends a confirmation of the successful receipt to the transmitter.
- The receiver buffers further segments, which were received correctly after an error, until the error is corrected by transmission repetition ("go-back-n" with buffering).
- No negative ACKs (NACKs) are sent!

Error Recognition and Correction

- TCP computes a checksum over the entire segment. The computation takes place with the same algorithm as with UDP. If the check sum signals an error at the receiver, the segment is not confirmed. That leads to expiration of the time limit at the transmitter and as consequence of this to transmission repetition. The transmitter repeats the sending in the "go-bake-n"-procedure.
- If segments are delivered out of sequence of IP, TCP restores the correct order.
- If IP datagrams are doubled in the network, TCP filters the dupes.

Flow Conrol and Congestion Control in TCP

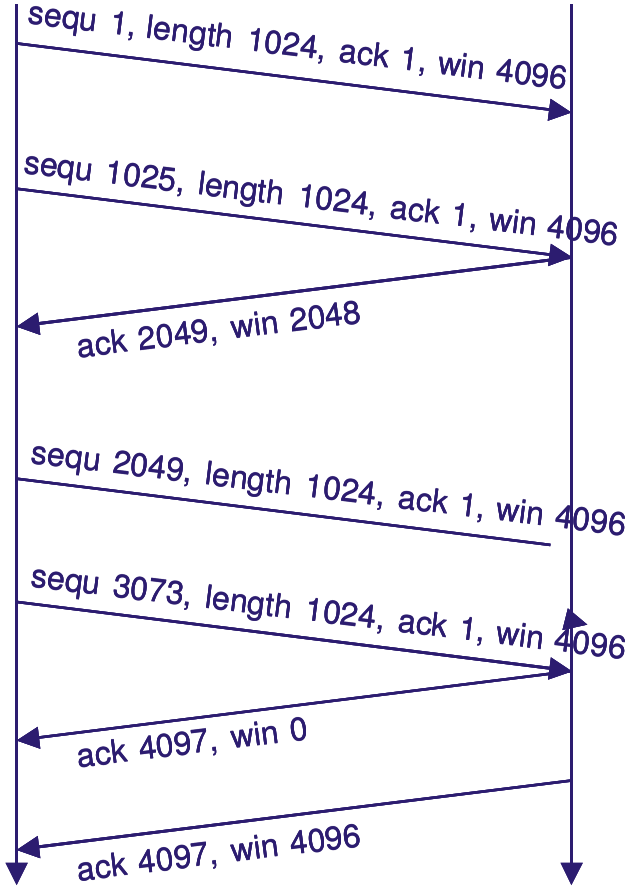
Flow Control

TCP uses a sliding window mechanism for flow control. Like always in TCP, the window size is expressed in bytes (not in packets!).

The size of the sliding windows is sent as the flow control parameter "window size" by the receiver to the transmitter.

The size of the window can be changed during the connection. If for example the receiver has only little buffer space, it is reduced.

Fast Transmitter and Slow Receiver (1)

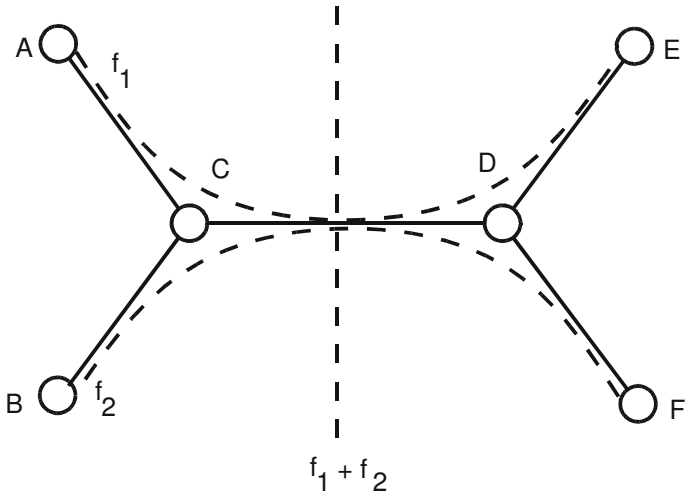


Fast Transmitter and Slow Receiver (2)

- The transmitter sends data faster than the receiver can read from its buffer and hand on to the higher layers.
- The buffer of the receiver gets filled up, it signalizes this by a window size of 0. This is an extension of the original sliding window mechanism, that allows to acknowledge packets concerning the error protection, without giving the transmitter the right to send further packets.
- Only if the buffer of the receiver again has free space, this free space is communicated to the transmitter in a further ACK in form of window size.

Congestion Control

Problem: If all transmitters in the network always send as many packets, as fit into the buffers of their receivers, an overload (blockage, congestion) may evolve inside the network.



If all connections have the same bandwidth and both A and B send with the full bandwidth of the connection, an overloading emerges inside the network (**congestion**). TCP connections recognize this and **voluntarily** regulate down the bandwidth!

Congestion Window

- In order to prevent congestion, an additional parameter **Congestion Window** (cwnd) is carried with the transmitter.
- cwnd is carried in bytes like the flow control window communicated by the receiver.
- A transmitter may always send only the MINIMUM of (cwnd, Flusskontroll Window) of data.
- There is no necessity to transfer cwnd between the partners!

Congestion Control in Steady State

Problem

How does the transmitter determine the correct size for the Congestion Window?

Solution

It gradually tries to increase the window until congestion occurs. Then it decreases its size again.

As long as no packets are lost, with each ACK cwnd is increased by 1/cwnd segments. Per round trip time the congestion window thus becomes larger approx. one segment ("*additives increase*").

Since there are no specific reporting packets from the inside of the network for congestion, TCP interprets each packet loss of the connection as sign for congestion!

There are two different reactions to segment losses, which will be described in the following.

Signs of Congestion

- **Triple Duplicate Acknowledgement** (TDACK): If a segment is lost, but the following segments arrive, the transmitter receives several ACKs with the same sequence number. After the third "Duplicate ACK" (thus four ACKs altogether for the same segment) the missing packet is transferred again, without waiting for the packet‘s timeout ("fast retransmit"). The transmitter interprets a TDACK as "light" overload and reduces cwnd **to half** of the original size ("*multiplicative decrease*").
- **Timeout**: If no more subsequent packets arrive after a lost packet, it does not result in a TDACK, but in a timeout for the lost packet. The transmitter interprets this as heavy overload and reduces cwnd **to one segment**.

Choise of the Timeout Value

Question: How large should the timeout value be selected?

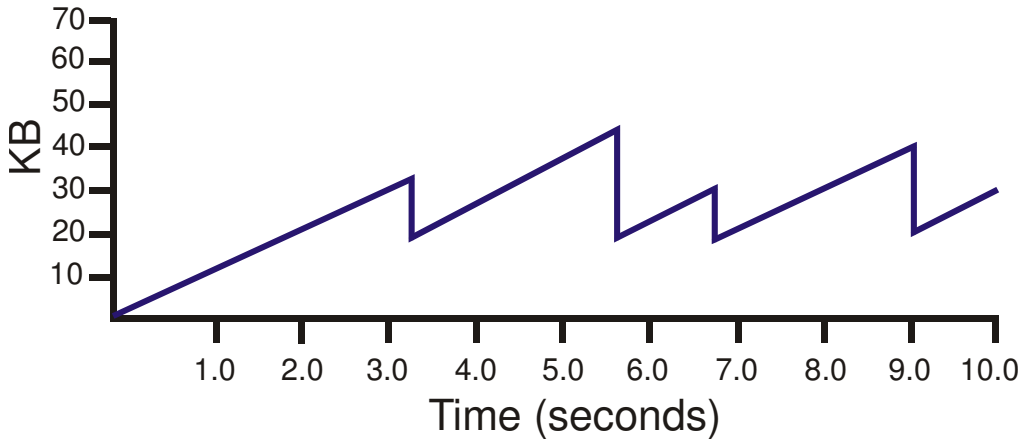
- Always larger than the Round Trip Time (RTT)
- RTT may vary à Safety addition depending on the RTT variance
  - EstimatedRTT = (1-x) * EstimatedRTT + x * SampleRTT
  - Deviation = (1-x) * Deviation + x * abs(SampleRTT - EstimatedRTT)
  - Timeout = EstimatedRTT + 4 * Deviation
- A well selected timeout value is important for high data throughput in networks, in which packet losses arise frequently.

Sawtooth curve of the TCP Throughput

The *Additive Increase*, *Multiplicative Decrease* algorithm results in a saw tooth shaped curve of the actual throughput of a TCP connection in the steady state. From regulation-technical view the procedure is stable, i.e. it does not swing up.

Observation: The area under the curve corresponds to the TCP data rate!

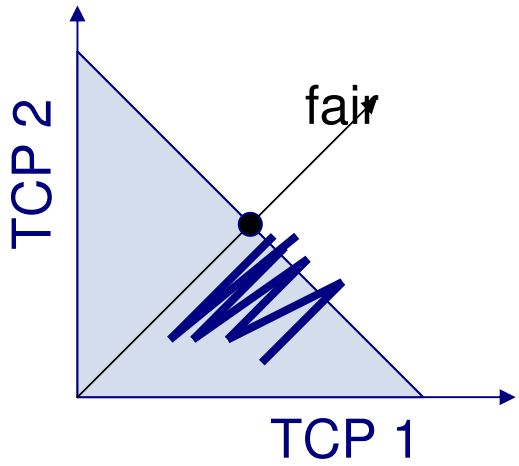
An example is shown in the illustration below.



TCP Fairness

If *n* TCP streams share one link, each should occupy a throughput of approx. 1/*n* of the available bandwidth.

Example: Two TCP stream run over the same link:



The data rates tend towards the point of fair capacity distribution under full utilization of the existing bandwidth.

Slow-Start (1)

Problem:

If a TCP connection is newly established, it takes a very long time until the transmitter achieves the optimal bit rate, since with each round trip time *cwnd* is increased only by one packet size (MSS = max. segment size).

Solution:

The **Slow-Start-Algorithm** by Van Jacobsen

- *cwnd* is initialized with the MSS of the receiver
- slow start threshold (*ssthresh*) is initialized with 65353
- Per Round-Trip-Time do the follwoing:
  - If no losses and *cwnd* < *ssthresh*:
    Slow-Start-Phase: increase *cwnd* by MSS for every received ACK (this is exponentially!)
  - If loss or *cwnd* >= *ssthresh:*
    End of the Slow-Start-Phase: normal AIMD-algorithm starts.

Slow-Start (2)

A TCP slow start likewise takes place after a timeout (signs of heavy congestion). The *ssthresh* value is halved, then a new slow start phase begins.

Note

This algorithm is called "slow start" for historical reasons, although it actually is a "Quick start algorithm" from today's point of view. Before it was built into TCP, every new connection began with a window size, which corresponded to the **flow control** window of the receiver. This frequently led to immediate congestion and packet losses.

Slow-Start: Example of a Progression

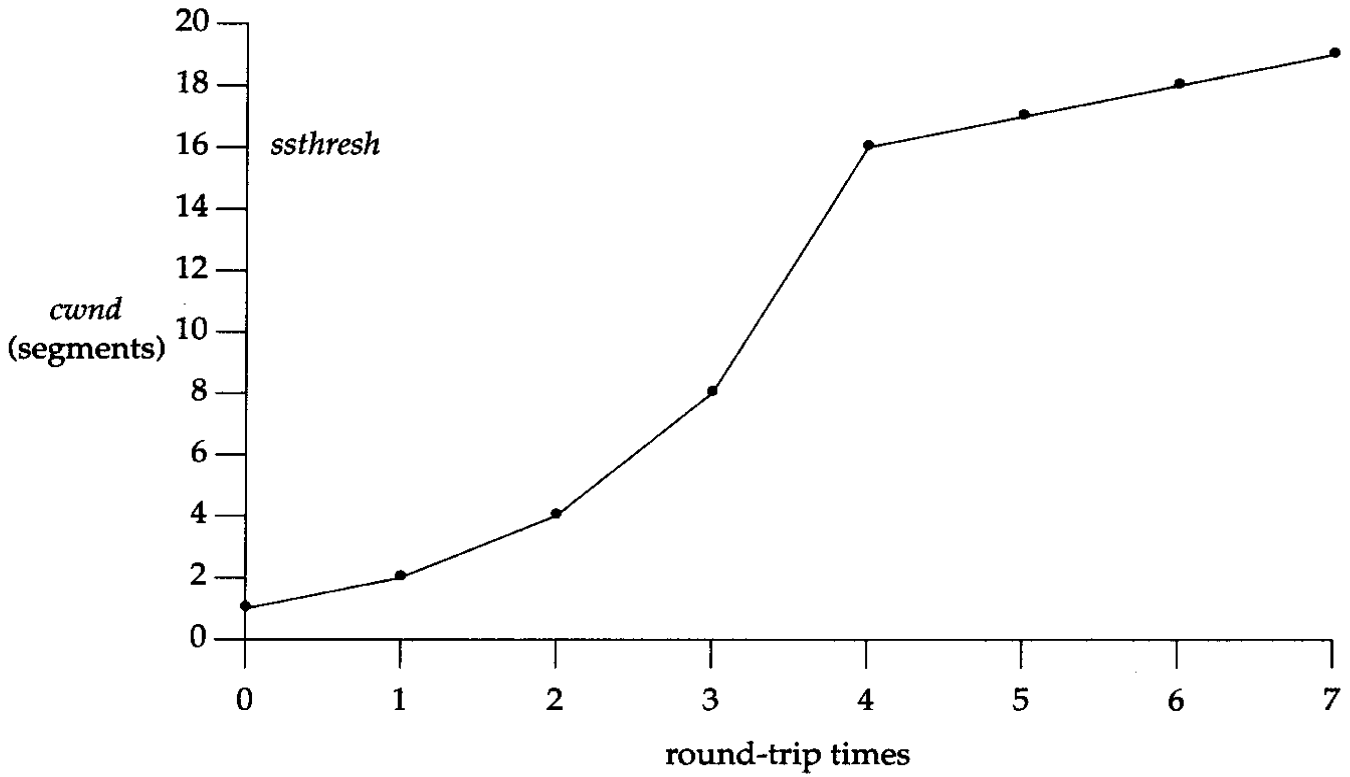


Figure 21.8 Visualization of slow start and congestion avoidance.

TCP Programming interface (API)

The programming of TCP takes place with the concept of **sockets**. A socket is the program-technical realization of a port.

- The server waits for connecting requests of clients on a well defined port.
- A client connects to the server
- After the connection is established, the server can create a thread (light-weight process) for this connection, which then handles further received packets of the connection.
- If it does not create a new thread, the connecting requests are sequentially processed (rare).

Summary TCP

Advantages of TCP

- Secured data communication
- Efficient data transfer despite complexity of the protocol (proven experimentally up to 100 MBit/s on standard machines)
- Applicable in LAN and WAN domains
- Well usable for low data rates (e.g. interactive terminal) and high data rates (e.g. file transfer)

Disadvantages towards UDP

- Higher resource requirements (intermediate memory, status informations of transmitter and receiver, many timers)
- Connection establishment and takedown also necessary with short data transfers
- Multicast impossible