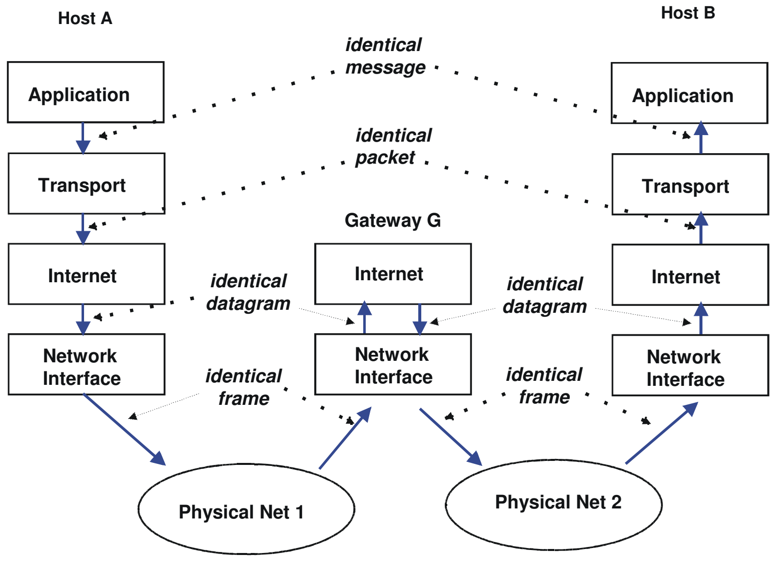
6. Transport Layer

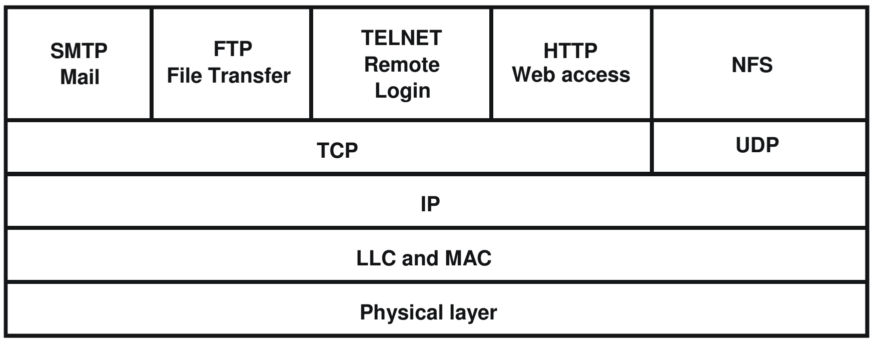
6.1 Internet Transport Layer Architecture
6.2 UDP (User Datagram Protocol)
6.3 TCP (Transmission Control Protocol)

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 Con-nection
25	SMTP	smtp	Simple Mail Transfer Protocol
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

0	16 31
Transmitter port	**Receiver port**
Packet length	**Checksum**
Data	
...	

- 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 (e.g., a *socket* interface). This is local matter.

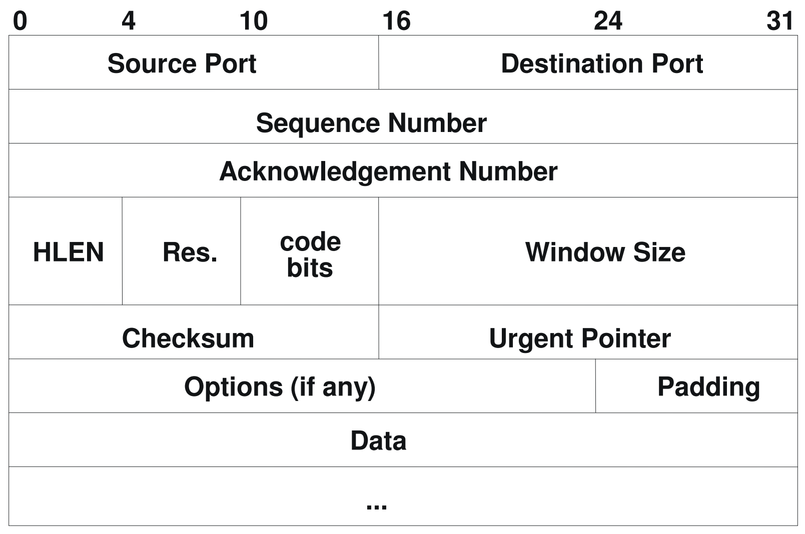
Addressing

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
code bits	6 bits from left to right as follows:
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 position of the first byte in the user data field in the logical byte stream.
- **Acknowledgement number:** Identifies the next byte in the data stream that the receiver expects from the sender. Note that not individual packets but *byte positions* in the logical byte stream are confirmed. Thus, an accumulation of confirmations over several TCP packets is possible. Acknowledgements control the re-transmission in the event of an error. They also influence the sliding window for flow control. Since data can flow in both directions at the same time (duplex operation), there is a separate numbering sequence for each direction.
- **Header length (HLEN):** Size of the TCP header in 32 bit words.

Data Fields in the TCP Header (3)

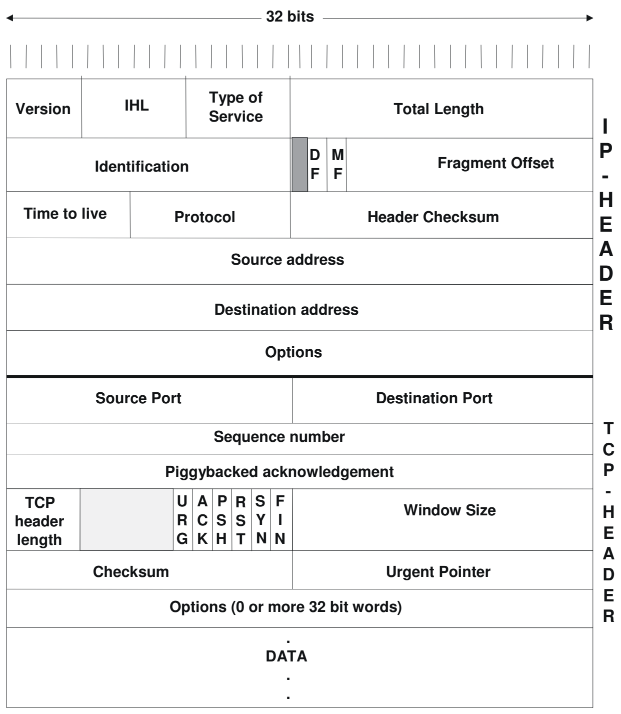
- **Code bits** (Flags)
  - URG: urgent pointer field is valid ("in use")
  - ACK: acknowledgement field is valid ("in use")
  - PSH: (push) The receiver should forward the data to the application as quickly as possible.
  - RST: reset the connection
  - SYN: synchronization of the initial sequence numbers at connection setup time
  - FIN: The transmitter has completed his data transfer.
- **Window size:** Number of bytes that the sender of this packet can accept in the opposite direction until his buffer is full (flow control).

Data Fields in the TCP Header (4)

- **Checksum:** The checksum is computed over the entire segment, including the header. A bit-column-wise EXOR is computed over the 16-bit words of the packet (as with UDP).
- **Urgent pointer:** Identifies the last byte in the data segment that should be processed with high priority. The data following this byte has normal priority.
- **Options:** optional data fields for special purposes.

The data segment of a TCP packet is optional. For example, an empty TCP packet can occur as a confirmation of received data if no data is ready to send in the opposite direction.

TCP/IP: Format of the Entire Header

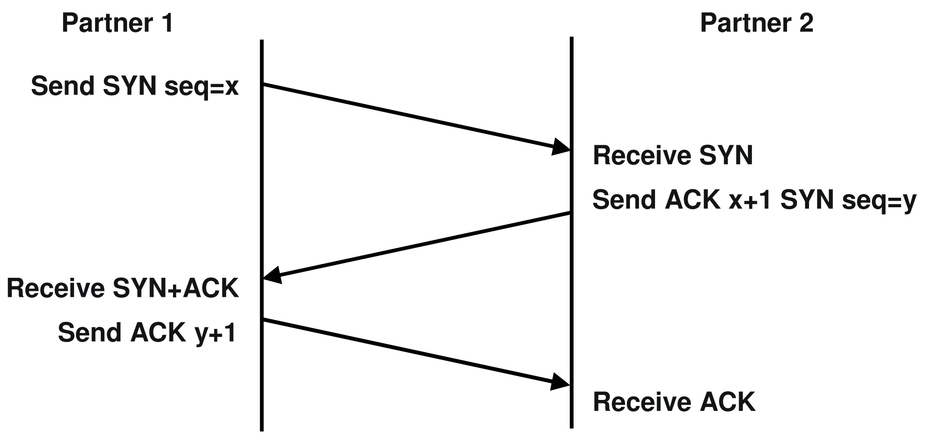


Main Functions in TCP

- Positive acknowledgement or retransmission (**PAR**)
- Retransmission after timeout by the sender. No NACKs in TCP!
- "sliding window" mechanism for flow control. *Variable window size*: with each acknowledgement from the receiver, a new window size is transmitted.
- Error detection by checksum.
- Piggybacking: Data and control information of the opposite direction can be transmitted in the same TCP packet.

TCP Connection Establishment

Three-Way Handshake: connection establishment with three packets:



At connection establishment, the initial sequence numbers for both sides (both directions) are exchanged and confirmed, they are “synchronized“, thus the name SYN for the packet.

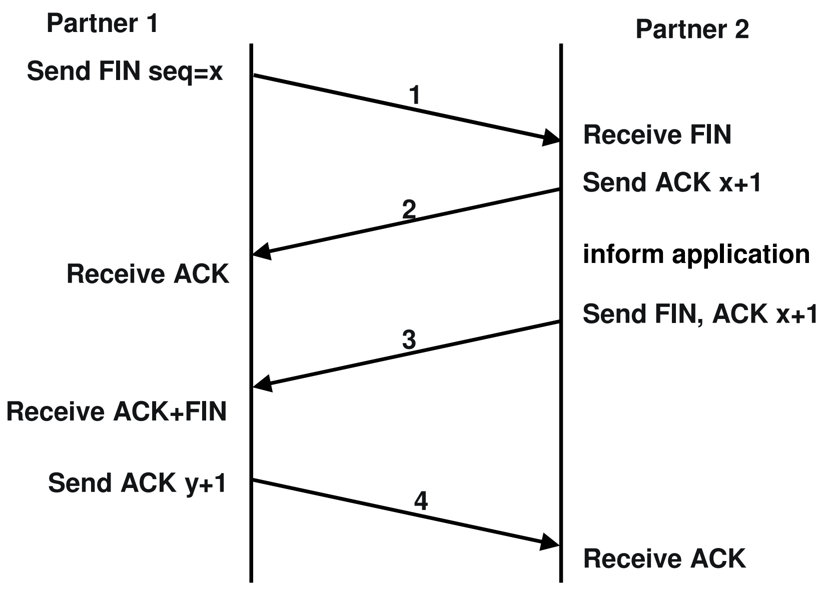
Timeout During Connection Establishment

What happens if the communication partner does not answer?

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

TCP Connection Termination (1)

Normal termination of a connection with four TCP messages:



TCP Connection Termination (2)

Takedown of a connection by two "half-close„ operations:

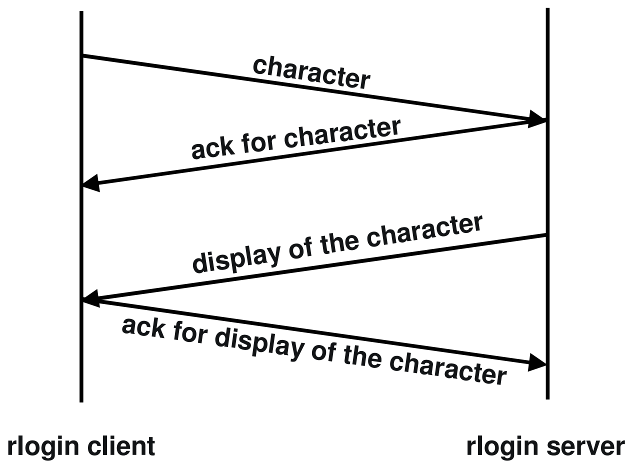
- Since the TCP connection is bi-directional, both directions must be terminated separately.
- The communication partner who wants to stop sending sets the FIN flag.
- FIN "costs" one byte and is therefore confirmed by an acknowledge-ment.
- The other participant can keep sending. However, in practice, one al-most always sees the behavior that the other participant terminates the connection.

TCP Reset

A packet in which the RST bit is set terminates the connection immediately, it signals an ABORT (in contrast to an orderly release with FIN).

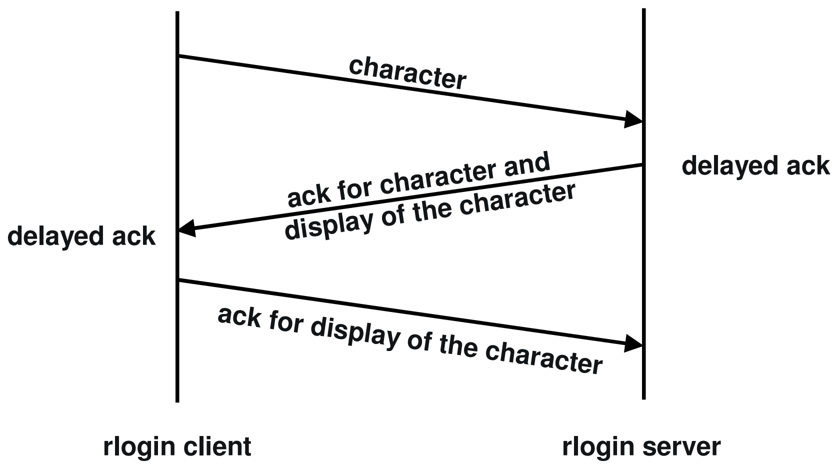
- All normal data buffered at the sender is dropped, and the RST packet is sent immediately. The connection is closed.
- As a consequence, with RST, data can be lost.
- The receiver of the RST packet notifies the application and immediately terminates the connection.

Data Flow of an Interactive Application over TCP



Delayed Acknowledgements (1)

In order to reduce the number of TCP packets that contain ACKs only, the sending of ACKs is often delayed:


character
delayed ack
ack for character and display of the character
delayed ack
ack for display of the character
rlogin client
rlogin server

Delayed Acknowledgements (2)

- An ACK is usually delayed for a maximum of 200 ms
- During this time, data 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 before timeout, a pure ACK packet without data is transmitted.
- The 200 ms timer is not set for every single packet but runs globally, i. e., after 200 ms all ACKs that are still open are sent. However, the timer is of course connection-oriented, i.e., for each TCP connection and each transmission direction there is a separate timer.

Mass Data Transfer

What happens, if large amount of data is transferred 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 transit if the data rate is high (bulk data flow).
- Therefore, every *second* packet is confirmed immediately even if the 200-ms timer has not yet expired.

Error Control in TCP

- TCP divides the byte stream into units, which will be transferred 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 Detection 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 by IP, TCP restores the correct order.
- If IP datagrams are doubled in the network TCP filters out the duplicates.

Flow Control and Congestion Control in TCP

Flow Control

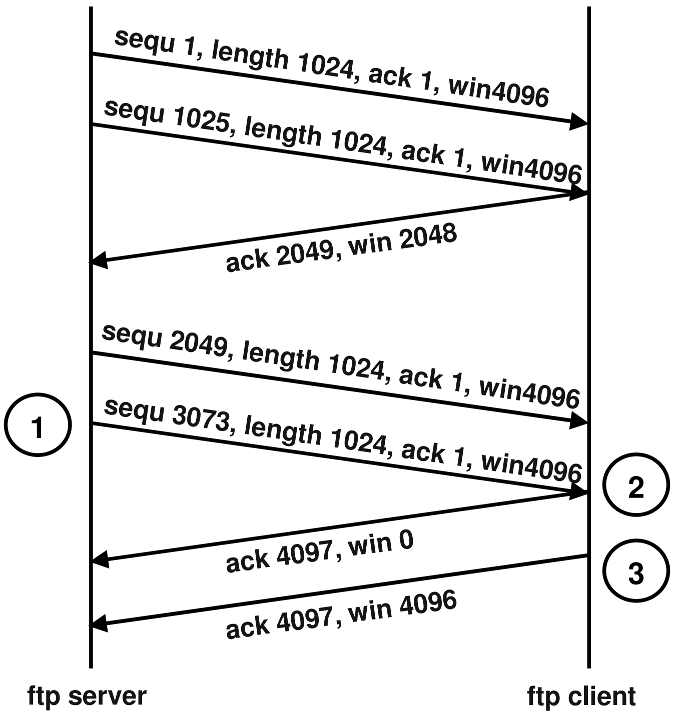
TCP uses a *sliding window* mechanism for flow control. As always in TCP, the window size is expressed in bytes (not in packets!).

The size of the open window 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. For example, if the receiver is running out of buffer space, he reduces the window size.

Fast Transmitter and Slow Receiver (1)

Flow control example



sequ 1, length 1024, ack 1, win4096
sequ 1025, length 1024, ack 1, win4096
ack 2049, win 2048
sequ 2049, length 1024, ack 1, win4096
1
sequ 3073, length 1024, ack 1, win4096
2
ack 4097, win 0
3
ack 4097, win 4096
ftp server
ftp client

Fast Transmitter and Slow Receiver (2)

Explanation of the flow control example

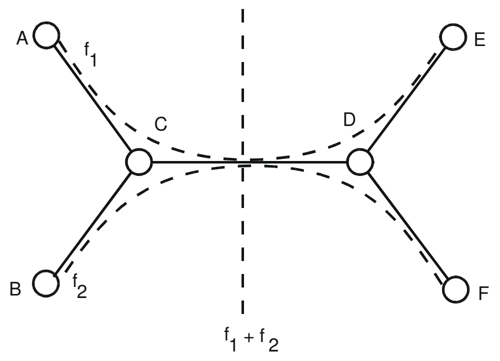
1.) The sender sends faster than the receiver can process the incoming packets.

2.) The buffer of the receiver fills up, he signals this fact to the sender by a window size of 0. Note that this is an extension of the original sliding window flow control protocol where the window size was fixed for the entire connection. This extension allows to acknowledge the correct receipt of packets from an error control point of view, without giving the sender the right to send more packets. In other words, it separates error control from flow control.

3.) When the receiver has enough free buffer space again, a new window size is communicated to the sender with the next ACK.

Congestion Control

Problem: Even if all senders in the network send only as many packets as their receivers can accept, congestion can still evolve inside the network.



If connections f_1 and f_2 have the same bandwidth, and both A and B send at full bandwidth, congestion might occur between nodes C and D. TCP connections try to detect this and slow down **voluntarily!**

Congestion Window

- In order to prevent congestion, an additional parameter **congestion window** (*cwnd*) is maintained locally at each sender. *cwnd* is counted in bytes (like the flow control window).
- A sender may always send only the MINIMUM of (*cwnd*, flow control window) of data.
- Note that *cwnd* it is NOT communicated to the partner!

Congestion Control in Steady State

Problem

How does the sender determine the appropriate size for the congestion window *cwnd*?

Solution

It gradually tries to increase *cwnd* until congestion occurs. Then it decreases it again.

As long as no packets are lost, with each ACK received, *cwnd* is increased by 1/*cwnd* segments. Thus, per round trip time, the congestion window increases by approximately one segment size ("***additive increase***").

Since there are no specific congestion report packets from inside the network, TCP interprets *every* packet loss as a sign for congestion!

Signs of Congestion

There are two different reactions to packet loss in TCP:

- **Triple Duplicate Acknowledgement** (TDACK): If a packet is lost but subsequent packets arrive correctly, the sender receives several ACKs with the same sequence number. After the third "Duplicate ACK" (i. e., 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***"). *ssthresh* is also reduced to half the value of *cwnd* (see below).
- **Timeout**: If no packets arrive anymore after a lost packet, this does not result in a TDACK but in a timeout for the lost packet at the sender. The sender interprets this as heavy overload and reduces *cwnd* **to one segment**. In other words, it starts sending again as if this were a new connection.

Setting the Timeout Value

Question: How large should the timeout value be selected? A well-chosen timeout value is important in order to achieve high data throughput in networks with frequent packet loss.

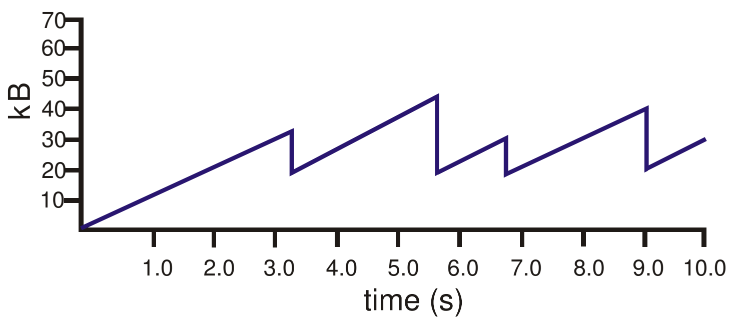
Solution

- Timeout must always be larger than the Round Trip Time (RTT) of a packet
- RTT may vary à add a safety margin depending on the observed RTT variance
- formulae:
  - estimatedRTT = (1-x) * estimatedRTT + x * sampleRTT
  - deviation = (1-x) * deviation + x * abs(sampleRTT - estimatedRTT)
  - timeout = estimatedRTT + 4 * deviation

Sawtooth Curve of TCP Throughput

The *Additive Increase*, *Multiplicative Decrease* (AIMD) algorithm results in a sawtooth-shaped curve for the actual throughput of a TCP connection in steady state.

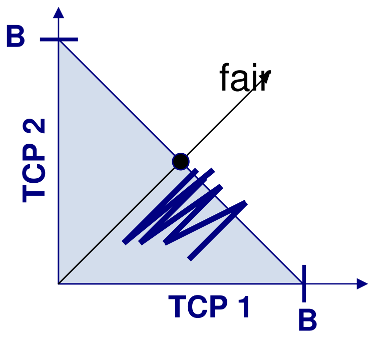
Observation: The area under the curve corresponds to the TCP data rate.



TCP Fairness

If *n* TCP streams share the same link each should use approx. 1/n of the available bandwidth B.

Example: Two TCP streams running over the same link:



The data rates of TCP1 and TCP2 converge to the point of a fair capacity distribution with full utilization of the available bandwidth B.

Slow-Start (1)

Problem:
When a TCP connection starts it takes a long time until the sender achieves the optimal bit rate since with each round trip time *cwnd* is increased only by one segment 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 (64 k bytes)
- Per Round-Trip-Time do the following:
  - If no loss and *cwnd* < *ssthresh*:
    Slow-Start-Phase: increase *cwnd* by MSS for every received ACK (exponential increase!)
  - If packet loss occurs or *cwnd* >= *ssthresh:*
    End of Slow-Start-Phase: the normal AIMD algorithm takes over.

Slow-Start (2)

A TCP slow start likewise takes place after a timeout (indication of heavy congestion). The *ssthresh* value is set to half of the current *cwnd* value, 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 determined by the **flow control** window of the receiver. This often led to immediate congestion and packet loss, not only for the new connection but also for other connections sharing links on the same path.

Slow Start Example

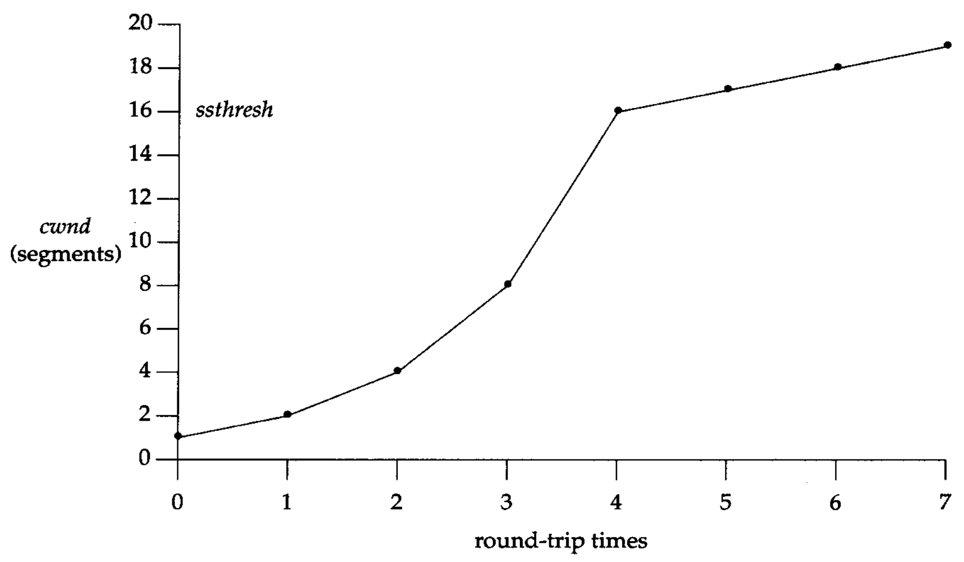


Figure 21.8 Visualization of slow start and congestion avoidance.

TCP Programming Interface (API)

Programming of TCP is done with **sockets**. A socket is the API for a port.

- The server waits for connection requests of clients on a well defined port.
- A client connects to the port.
- After the connection is established the server creates a thread (light-weight process) for this connection which then handles all the data traffic.

TCP Summary

Advantages of TCP

- Secure data communication
- Efficient data transfer in spite of complexity of the protocol (proven experimentally up to 100 MBit/s on standard computers)
- Applicable in LANs and WANs
- Runs well for low data rates (e.g., an interactive terminal) as well as high data rates (e.g., a file transfer)

Disadvantages in comparison with UDP

- Higher resource requirements (memory for status information and buffers, CPU, many timers)
- Connection establishment and termination cause considerable overhead for short data transfers
- Multicast is not possible