

Introducing TCP & UDP

Internet Transport Layers

(C) Herbert Haas 2005/03/11

TCP Facts (1)



- Connection-oriented layer 4 protocol
- Carried within IP payload
- Provides a **reliable end-to-end transport** of data between computer processes of different end systems
 - ◆ Error detection and recovery
 - ◆ Sequencing and duplication detection
 - ◆ Flow control
- RFC 793

In this Chapter we talk about **TCP**. TCP is a connection-oriented layer 4 protocol and only works between the hosts. It synchronizes (connects) the hosts with each other with the “3-Way-Handshake” before the real transmission begins. After this a secure end-to-end transmission is established. TCP was standardized in September 1981 in RFC 793. (Remember: IP was standardized in September 1981 too, RFC 791). TCP always used with IP and it protects the IP-Packet. Such as the most transport protocol TCP makes error recovery, flow control and sequencing. The most important thing with TCP is the **Port-Number**, we will discuss later.

TCP Facts (2)



- **Application's data is regarded as continuous byte stream**
- **TCP ensures a reliable transmission of segments of this byte stream**
- **Handover to Layer 7 at "Ports"**
 - ◆ **OSI-Speak: Service Access Point**

Every IP-Packet which is send with TCP will be acknowledgment (error recovery). With error recovery, sequencing and the synchronization of the hosts before the transmission begins, TCP ensures a reliable transmission of segments of this byte stream. TCP hides the details of the network layer from the higher layers and frees them from the tasks of transmitting data through a specific network. TCP provides its service to higher layer through ports (Service Access Points).

Port Numbers



- Using port numbers TCP (and UDP) can **multiplex** different layer-7 byte streams
- Server processes are identified by **Well known** port numbers : 0..1023
 - ◆ Controlled by IANA
- Client processes use arbitrary port numbers >1023
 - ◆ Better >8000 because of registered ports

Each communicating computer process is assigned a port, identify by a special portal number. So the TCP software can serve multiple process such as browser or e-Mail simultaneously. The TCP software functions like a multiplexer and demultiplexer for TCP connections: Process 1, System A \leftrightarrow Process 3, System B

Registered Ports

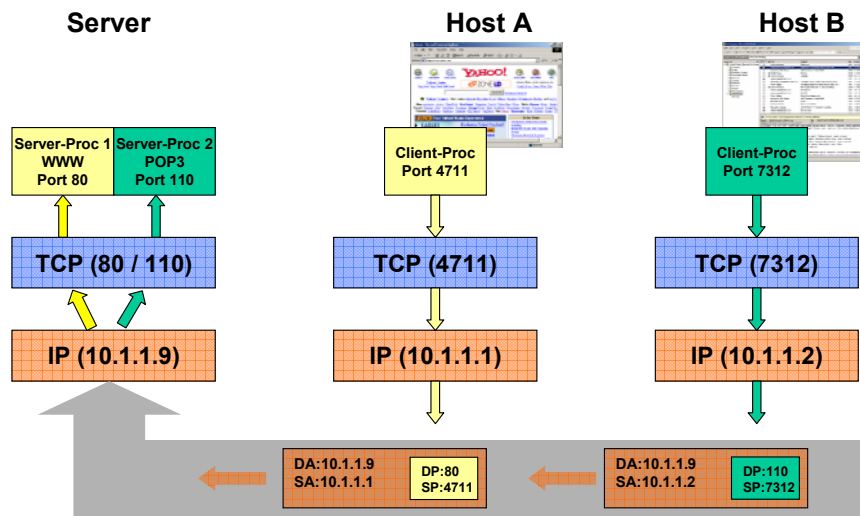


- For proprietary server applications
- Not controlled by IANA only listed in RFC 1700
- Examples
 - ◆ 1433 Microsoft-SQL-Server
 - ◆ 1439 Eicon X25/SNA Gateway
 - ◆ 1527 Oracle
 - ◆ 1986 Cisco License Manager
 - ◆ 1998 Cisco X.25 service (XOT)
 - ◆ 6000-6063 X Window System

There are some **registered ports** which start at 1024 (e.g. Lotus Notes, Cisco XOT, Oracle, license managers etc.). They are not controlled by the IANA, only listed in RFC1700.

Only the **well known ports** are reserved for common applications and services, such as Telnet, WWW, FTP etc. They are in the range from 0 to 1023. These are controlled by the Internet Assigned Numbers Authority (IANA).

TCP Communications



The client applications chose a free port number (which is not already used by another connection) as the source port. The destination port is the well-known port of the server application. For example: Host B runs a Mail-Program (POP3) and the client application uses the source port 7312. The packet is sent to the Server with a destination-port of 110. Now the Server knows Host B makes a Mail-Check over POP3.

Sockets

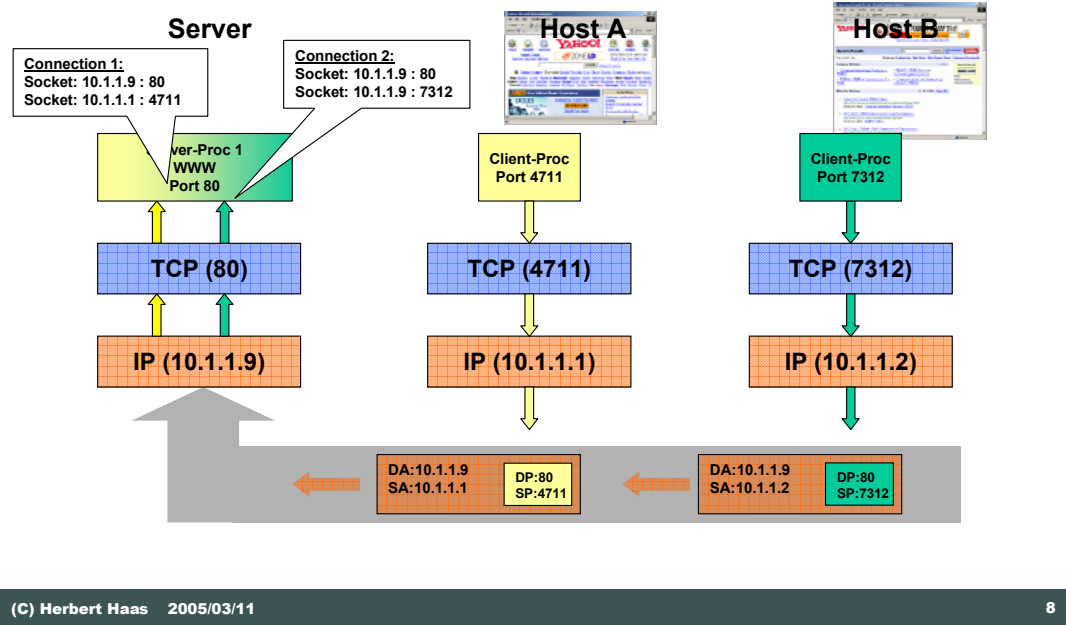


- **Server process multiplexes streams with same source port numbers according source IP address**
- **(PortNr, SA) = Socket**
- **Each stream ("flow") is uniquely identified by a socket pair**

In a client-server environment a communicating server-process has to maintain several sessions (and also connections) to different targets at the same time. Therefore, a single port has to multiplex several virtual connections. These connections are distinguished through sockets. The combination IP address and port number is called a "socket".

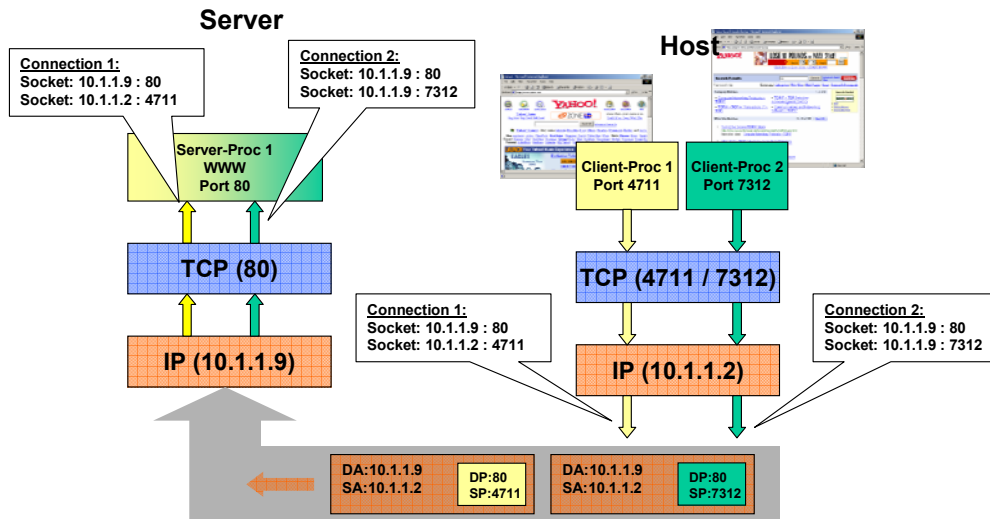
For example: 10.1.1.2:80 [IP-Address : Port-Number]

TCP Communications



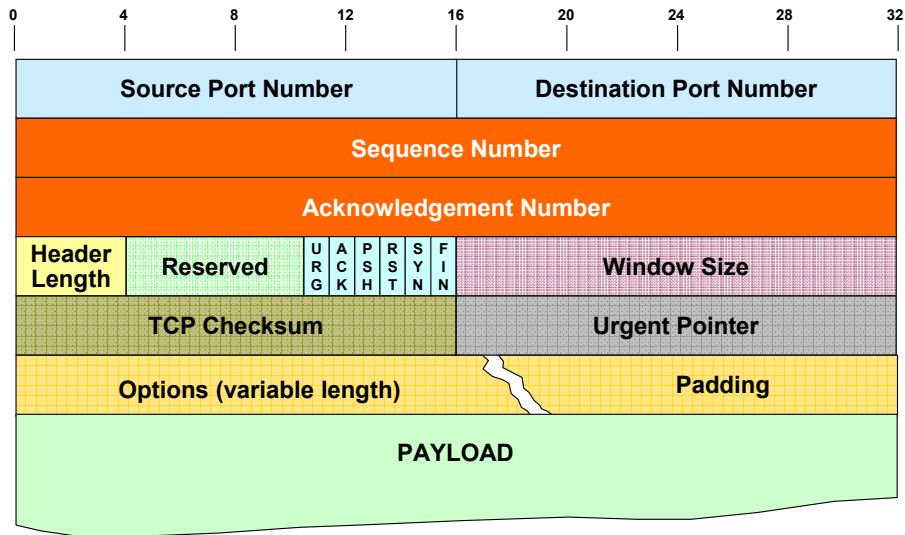
With the help of the Socket-Principle a server can easily remember between many connections and can send back his data without problems.

TCP Communications



Well-known ports together with the socket concept allow several simultaneous connections (even from a single machine) to a specific server application. Server applications listen on their well-known ports for incoming connections.

TCP Header



The picture above shows the 20 byte TCP header plus optional options. Remember that the IP header are also 20 bytes, so the total sum of overhead per TCP/IP packet is 40 bytes.

TCP Header (1)



- **Source and Destination Port**
 - ◆ 16 bit port number for source and destination process
- **Header Length**
 - ◆ Multiple of 4 bytes
 - ◆ Variable header length because of options (optionally)

The **Source** and **Destination Port** fields are 16 bits and used by the application. The **Header Length** indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.

TCP Header (2)



- **Sequence Number (32 Bit)**
 - ◆ Number of **first byte** of this segment
 - ◆ Wraps around to 0 when reaching $2^{32} - 1$
- **Acknowledge Number (32 Bit)**
 - ◆ **Number of next byte expected by receiver**
 - ◆ **Confirms correct reception of all bytes including byte with number AckNr-1**

Sequence Number: 32 bit. Number of the first byte of this segment. If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledge Number: 32 bit. If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

TCP Header (3)



- **URG-Flag**
 - ◆ Indicates urgent data
 - ◆ If set, the 16-bit "Urgent Pointer" field is valid and points to the last octet of urgent data
 - ◆ **There is no way to indicate the beginning of urgent data (!)**
 - ◆ Applications switch into the "**urgent mode**"
 - ◆ Used for quasi-outband **signaling**

URG-Flag: 1 Bit. Control Bit.

Sequence number of last urgent octet = actual segment sequence number + urgent pointer

RFC 793 and several implementations assume the urgent pointer to point to the first octet *after* urgent data. However, the "Host Requirements" RFC 1122 states this as a mistake! When a TCP receives a segment with the URG flag set, it notifies the application which switch into the "urgent mode" until the last octet of urgent data is received. Examples for use: Interrupt key in Telnet, Rlogin, or FTP.

TCP Header (4)



- **PSH-Flag**
 - ◆ **TCP should push the segment immediately to the application without buffering**
 - ◆ **To provide low-latency connections**
 - ◆ **Often ignored**

PSH-Flag: 1 Bit. Control Bit.

A TCP instance can decide on its own, when to send data to the next instance. One strategy could be, to collect data in a buffer and forward the data when the buffer exceeds a certain size. To provide a low-latency connection sometimes the PSH Flag is set to 1. Then TCP should push the segment immediately to the application without buffering. But most of the time the PSH-Flag will be ignored.

TCP Header (5)



- **SYN-Flag**
 - ◆ Indicates a connection request
 - ◆ Sequence number synchronization
- **ACK-Flag**
 - ◆ Acknowledge number is valid
 - ◆ Always set, except in very first segment

SYN-Flag: 1 Bit. Control Bit.

If the SYN bit is set to 1, the application knows that the host want to established a connection with him. Also used to synchronization the sequence numbers. Most Firewalls through away packets with SYN=1 if the host want to established a connection to a application which the is server not allowed (security reasons).

ACK-Flag: 1 bit. Control Bit.

Acknowledgment Bit.

TCP Header (6)



- **FIN-Flag**
 - ◆ Indicates that this segment is the last
 - ◆ Other side must also finish the conversation
- **RST-Flag**
 - ◆ Immediately kill the conversation
 - ◆ Used to refuse a connection-attempt

FIN-Flag: 1 bit. Control Bit.

The FIN-Flag is used in the “disconnect process”. It indicates that this segment is the last. After the other side also send a segment with FIN=1, the connection is closed.

RST-Flag: 1 bit. Control Bit.

Reset the connection.

TCP Header (7)



- **Window (16 Bit)**
 - ◆ Adjusts the send-window size of the other side
 - ◆ Used with every segment
 - ◆ **Receiver-based flow control**
 - ◆ **SeqNr of last octet = AckNr + window**

Window Size: 16 bit. The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept. See Slide 27.

TCP Header (8)



- **Checksum**
 - ◆ Calculated over TCP header, payload and 12 byte **pseudo IP header**
 - ◆ Pseudo IP header consists of source and destination IP address, IP protocol type, and IP total length;
 - ◆ Complete socket information is protected
 - ◆ Thus TCP can also detect IP errors

TCP Checksum: 16 bit. The checksum includes the TCP header and data area plus a 12 byte pseudo IP header (one's complement of the sum of all one's complements of all 16 bit words). The pseudo IP header contains the source and destination IP address, the IP protocol type and IP segment length (total length). This guarantees, that not only the port but the complete socket is included in the checksum.

TCP Header (9)



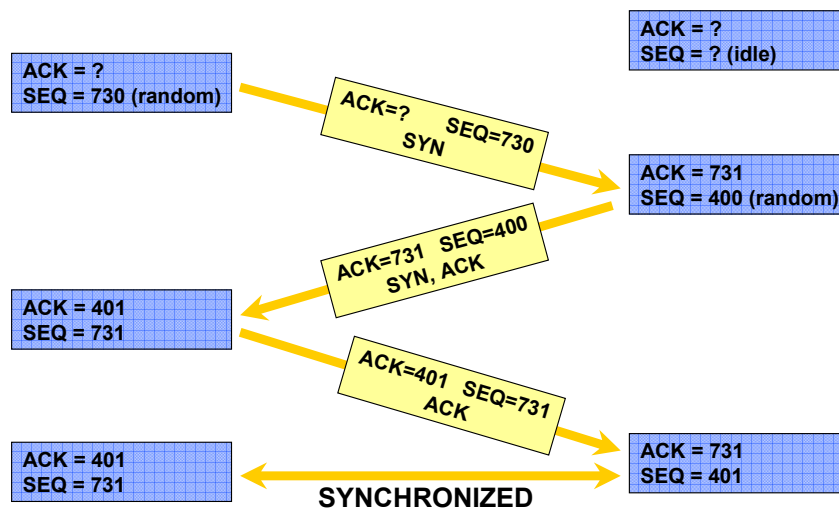
- **Urgent Pointer**
 - ◆ Points to the last octet of urgent data
- **Options**
 - ◆ Only MSS (Maximum Message Size) is used
 - ◆ Other options are defined in RFC1146, RFC1323 and RFC1693
- **Pad**
 - ◆ Ensures 32 bit alignment

Urgent Pointer: 16 bits. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

Options: Variable length. Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. Only the Maximum Message Size (MSS) is used. All options are included in the checksum.

Padding: Variable length. The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

TCP 3-Way-Handshake



(C) Herbert Haas 2005/03/11

20

The diagram above shows the famous TCP 3-way handshake. The TCP 3-Way-Handshake is used to connect and synchronize two host with each other, that is, after the handshake procedure, both stations know the sequence numbers of each other.

The connection procedure (3-Way-Handshake) works with a simple principle. The host sends out a segment with `SYN=1` (remember: if `SYN=1` the application knows that the host want to established a connection) and the host also choose a random sequence number (`SEQ`). After the Server receives the segment correct, he acknowledgment (`host-SEQ+1`), also choose a random `SEQ`, and send back the segment with `SYN=1`. Remember the `ACK`-flag is always set, except in very first segment. Because the server sends back a segment with `SYN=1` the host knows the connection is accepted. After the host sends a acknowledgement to the server the connection is established.

Sequence Number

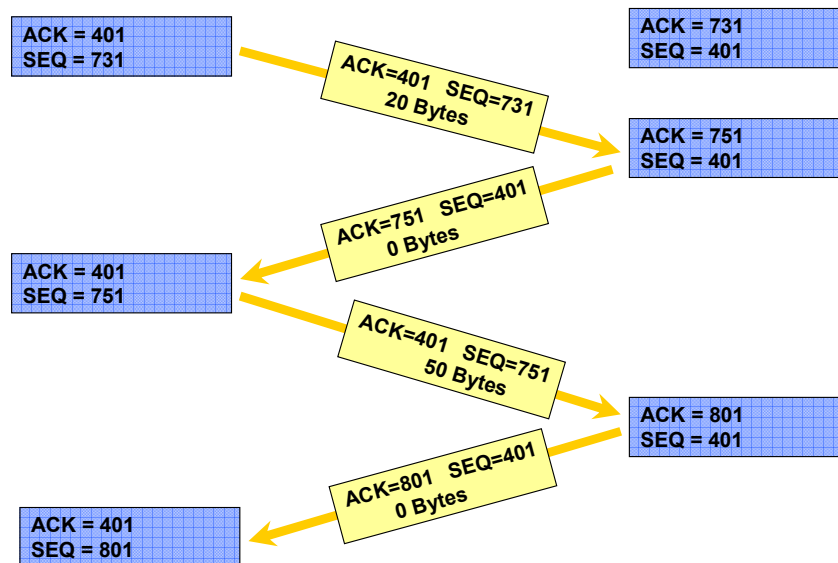


- **RFC793 suggests to pick a random number at boot time (e.g. derived from system start up time) and increment every 4 μ s**
- **Every new connection will increments SeqNr by 1**
- **To avoid interference of spurious packets**
- **Old "half-open" connections are deleted with the RST flag**

RFC 793 suggests to pick a random starting sequence numbers and an explicit negotiation of starting sequence numbers to make a TCP connect immune against spurious packets.

Also disturbing segments (e.g. delayed TCP segments from old sessions etc.) and old "half-open" connections are deleted with the RST flag.

TCP Data Transfer



After the 3-way-handshake is finished the real data transfer is started. A 20 Byte segment is sent to the server (ACK 401, SEQ 731). After the server receives the segment, he sets the ACK-flag to 751 (SEQ+20 Byte) and the SEQ to 401. Then he sends the segment back (ACK 751, SEQ 401) to the host. After the host receives this segment he knows that his 20 byte of data delivered correctly (because he gets the ACK 751). The host continues sending his data to the server.

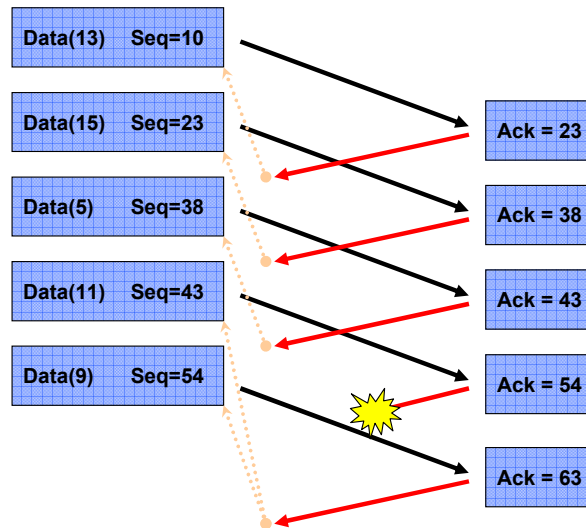
TCP Data Transfer



- **Acknowledgements are generated for all octets which arrived in sequence without errors (positive acknowledgement)**
- **Duplicates are also acknowledged (!)**
 - ◆ Receiver cannot know why duplicate has been sent; maybe because of a lost acknowledgement
- **The acknowledge number indicates the sequence number of the next byte to be received**
- **Acknowledgements are cumulative: Ack(N) confirms all bytes with sequence numbers up to N-1**
 - ◆ Therefore lost acknowledgements are no problem

The acknowledge number is equal to the sequence number of the next octet to be received.

TCP Multiple Acknowledgement



(C) Herbert Haas 2005/03/11

24

Its not a problem for TCP when a acknowledgment get lost, because TCP is acknowledge all receiving data with every acknowledgement.

TCP Duplicates:

There are some reasons for retransmission if a TCP duplicate occur:

- Because original segment was lost: no problem, retransmitted segment fills gap, no duplicate
- Because ACK was lost or retransmit timeout expired: no problem, segment is recognized as duplicate through the sequence number
- Because original was delayed and timeout expired: no problem, segment is recognized as duplicate through the sequence number

The 32 bit sequence numbers provide enough "space" to tell duplicates from originals: 232 Octets with 2 Mbit/s means 9h for wrap around (compare to usual TTL = 64 seconds)

TCP Timeout



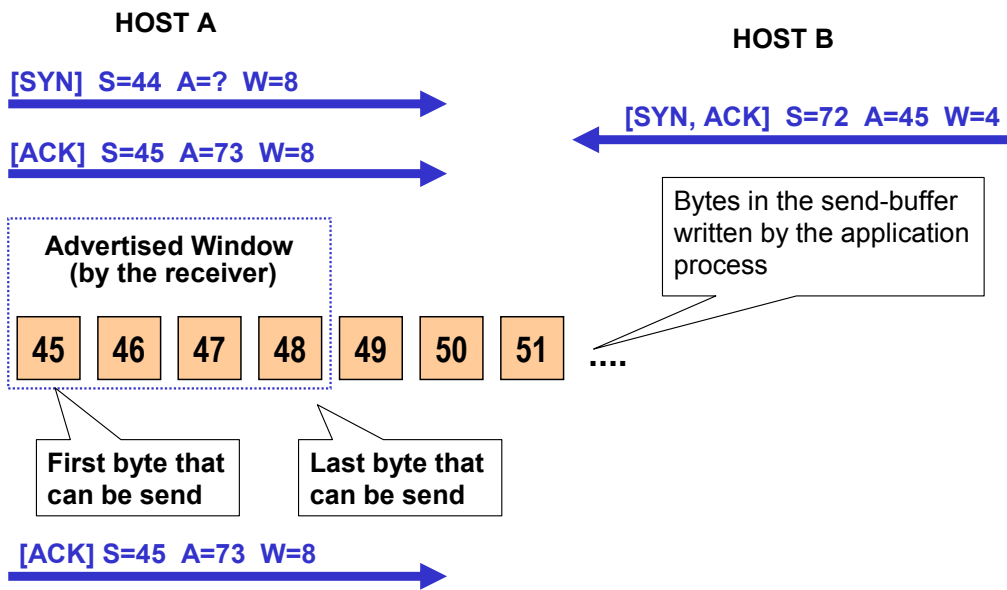
- **Timeout will initiate a retransmission of unacknowledged data**
 - ♦ Value of retransmission timeout influences performance (timeout should be in relation to round trip delay)
 - ♦ High timeout results in long idle times if an error occurs
 - ♦ Low timeout results in unnecessary retransmissions
- **Adaptive timeout**
 - ♦ KARN algorithm uses a backoff method to adapt to the actual round trip delay

TCP Sliding Window



- **TCP flow control is done with dynamic windowing using the sliding window protocol**
- **The receiver advertises the current amount of octets it is able to receive**
 - ♦ Using the window field of the TCP header
 - ♦ Values 0 through 65535
- **Sequence number of the last octet a sender may send = received ack-number -1 + window size**
 - ♦ The starting size of the window is negotiated during the connect phase
 - ♦ The receiving process can influence the advertised window, hereby affecting the TCP performance

TCP Sliding Window



TCP Sliding Window



- During the transmission the sliding window moves from left to right, as the receiver acknowledges data
- The relative motion of the two ends of the window open or closes the window
 - ♦ The window closes when data is sent and acknowledged (the left edge advances to the right)
 - ♦ The window opens when the receiving process on the other end reads acknowledges data and frees up TCP buffer space (the right edge moves to the right)
- If the left edge reaches the right edge, the sender stops transmitting data - zero window

TCP Enhancements

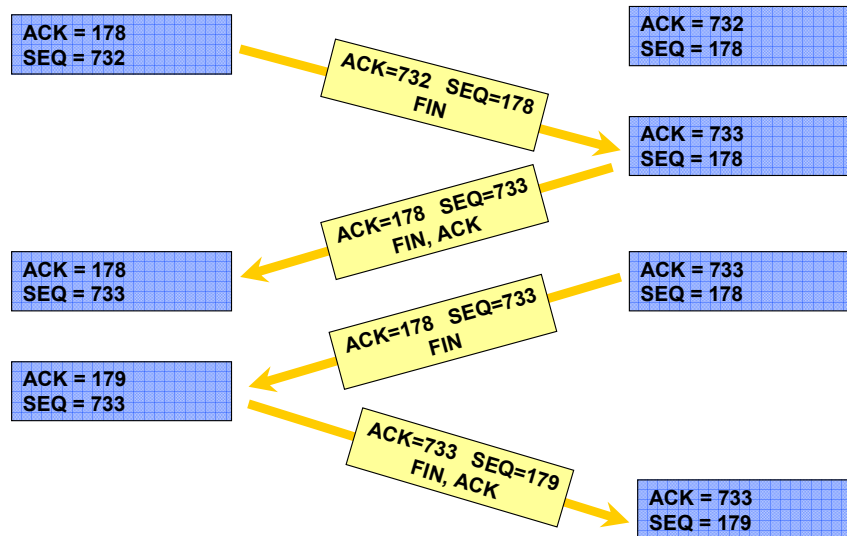


- **So far, only basic TCP procedures have been mentioned**
- **TCP's development still continues; it has been already enhanced with additional functions which are essential for operation of TCP sessions in today's IP networks:**
 - ♦ "Slow Start" and congestion avoidance mechanism
 - ♦ "Fast Retransmit" and "Fast Recovery" mechanism
 - ♦ "Delayed Acknowledgements"
 - ♦ "The Nagle Algorithm"
 - ♦

The "Slow Start" and congestion avoidance is a mechanism which controls the rate of packets which are put into a network.

The "Fast Retransmit" and "Fast Recovery" is a Mechanism to avoid waiting for the timeout in case of retransmission and to avoid slow start after a fast retransmission.

TCP Disconnect



The disconnect process is similar to the 3-Way-Handshake. The exchange of FIN and ACK flags ensures, that both parties have received all octets.

TCP Disconnect



- A TCP session is disconnected similar to the three way handshake
- The FIN flag marks the sequence number to be the last one; the other station acknowledges and terminates the connection in this direction
- The exchange of FIN and ACK flags ensures, that both parties have received all octets
- The RST flag can be used if an error occurs during the disconnect phase

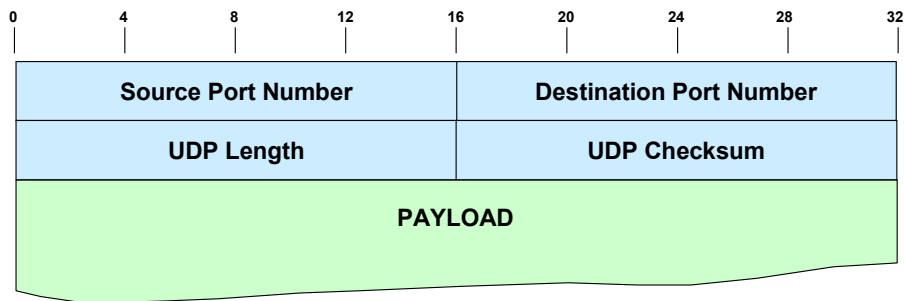
UDP



- **UDP is a connectionless layer 4 service (datagram service)**
- **Layer 3 Functions are extended by port addressing and a checksum to ensure integrity**
- **UDP uses the same port numbers as TCP (if applicable)**
- **UDP is used, where the overhead of a connection oriented service is undesirable or where the implementation has to be small**
 - ♦ **DNS request/reply, SNMP get/set, booting by TFTP**
- **Less complex than TCP, easier to implement**

UDP is connectionless and makes no error recovery or flow control. For example: UDP is used with VoIP.

UDP Header



The picture above shows the 8 byte UDP Header.

UDP



- **Source and Destination Port**
 - ♦ Port number for addressing the process (application)
 - ♦ Well known port numbers defined in RFC1700
- **UDP Length**
 - ♦ Length of the UDP datagram (Header plus Data)
- **UDP Checksum**
 - ♦ Checksum includes pseudo IP header (IP src/dst addr., protocol field), UDP header and user data;
one's complement of the sum of all one's complements

Compared to the TCP Header, the UDP is very small (8 byte to 20 byte) because UDP makes no error recovery or flow control.

Summary



- **TCP & UDP are Layer 4 (Transport) Protocols above IP**
- **TCP is "Connection Oriented"**
- **UDP is "Connection Less"**
- **TCP implements "Fault Tolerance" using "Positive Acknowledgement"**
- **TCP implements "Flow Control" using dynamic window-sizes**
- **The combination of IP-Address and TCP/UDP-Port is called a "Socket"**

Quiz



- **What are advantages of TCP over UDP?**
- **What are advantages of UDP over TCP?**
- **What are important values to define the optimal window-size?**
- **Would you use TCP or UDP for real-time traffic? (VoIP, Video...)**
- **When you download something from the Internet the download rate is first small and increases afterwards – WHY?**

Hints



- Q1:connection oriented, fault tolerant
- Q2:slim protocol, no overhead like connection establishment, Ack...
- Q3:round trip time (delay), bandwidth
- Q4:UDP (makes no sense to retransmit a lost voice sample)
- Q5:"slow start algorithm"