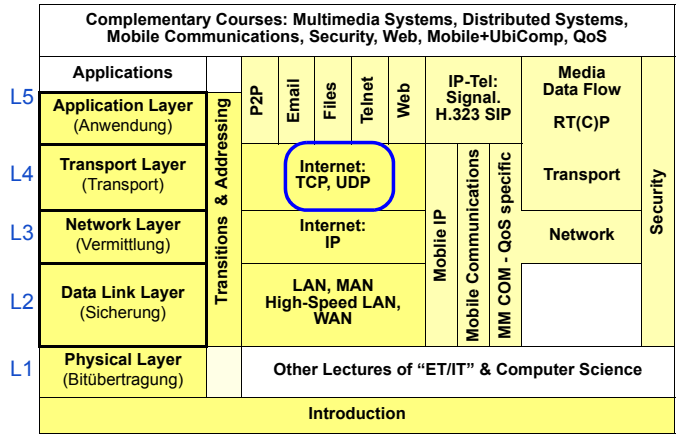
Communication Systems
Transport Layer - Protocols

Prof. Dr.-Ing. **Lars Wolf**

TU Braunschweig
Institut für Betriebssysteme und Rechnerverbund

Mühlenpfordtstraße 23, 38106 Braunschweig, Germany
Email: wolf@ibr.cs.tu-bs.de

Scope

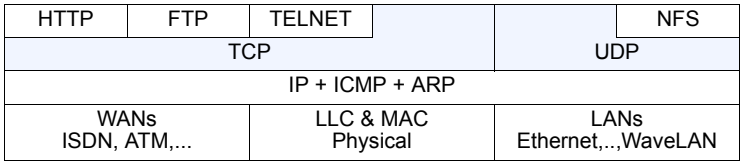


Overview

1. **Transport Protocols**
2. **ISO-OSI Transport Protocols**
3. **Internet Transport Layer (in General & Addressing)**
4. **UDP - User Datagram Protocol**
5. **TCP - Transmission Control Protocol**
6. **Stream Control Transmission Protocol (SCTP)**

1. Transport Protocols

Internet



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

ISO-OSI
- **practically irrelevant**
- **but show overall design space**

Other

2. ISO-OSI Transport Protocols

taking a practical view:
- **irrelevant today**
- **never got sufficient support with respect to implementations etc.**

general view:
- **provide for an overall classification of schemes**
- **show overall design space**

⇒ **we take a short look at them in a general manner**

2.1 ISO-OSI Transport Protocols & Network Services

L4 protocol depends on the quality of the L3 service (NS network service)
- **Network type A:**
  typically CONS (connection oriented network service) on LANs
  - network is **RELIABLE**
  - network recognizes data loss as an error
  - errors are displayed to the user(N-RESET), i.e., an acceptable rate of the errors
  - minor (for the user acceptable) error rate
  - network **NEVER** duplicates or manipulates packets
  - order of sent packets is **ALWAYS** maintained
- **Network type B:**
  typically CONS on (old) WANs
  - like type A, except
  - **REMAINING ERROR RATE** (for data loss) **IS NOT ACCEPTABLE**
- **Network type C:**
  typically CLNS (connectionless network service) on WANs
  - network is **UNRELIABLE**
  - errors due to losses, duplication and manipulation of packets, as well as faulty packet sequence errors possible
  - errors might remain undetected
  - transport protocol has to / should compensate for this

2.2 ISO-OSI Transport Protocols

Transport protocol classes
- **5 classes: ISO OSI TP0..TP4**

Protocol Class	Net-work Type	Network Properties	Name
TP 0	A	Acceptable error rate Acceptable rate of displayed errors	Simple class
TP 2			Multiplexing class
TP 1	B	Inacceptable error rate Acceptable rate of displayed errors	Basic error recovery
TP 3			Error recovery and multiplexing class
TP 4	C	Inacceptable error rate Inacceptable rate of displayed errors	Error recovery and multiplexing class

ISO-OSI Transport Protocols (2)

Class TP 0: simple class (A)
- **mechanisms for connect and disconnect**
- **segmentation / reassembly**
- **no error, sequence or flow control**
- **no expedited data**

Class TP 1: basic error recovery (B)
- **class 0 including additional error recovery**
- **error recovery masks N-RESETs**
  - TPDU numbering
  - TPDU storage until ACK
  - after N-RESET: resynchronization
- **expedited data optional**
  - important data, for example, have a higher priority
  - i.e., preferred processing before current data is processed

ISO-OSI Transport Protocols (3)

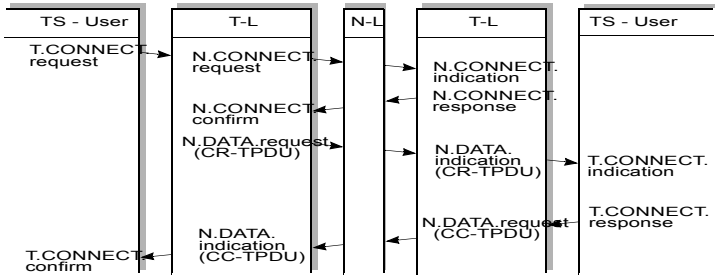
Class TP 2: multiplexing class (A)
- class 0 including additional multiplexing capability
- MULTIPLEXING: several L4 connections on one L3 connection
- flow control optional
- expedited data optional

Class TP 3: including multiplexing and error recovery (B)
- class 1 and 2 functions combined
- i.e. error recovery, expedited data, multiplexing

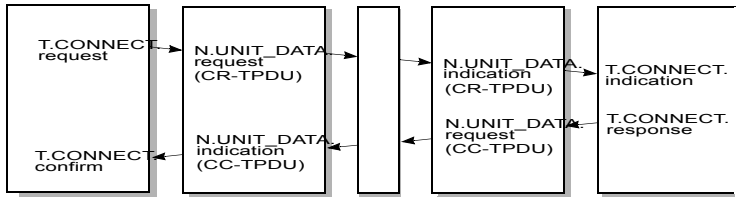
Class TP 4: error monitoring and recovery(C)
- detects and recovers
  - TPDU losses and TPDU duplication
  - sequence errors
- flow control
- multiplexing
- splitting (one T connection uses several N connections)
- expedited data

ISO OSI Transport Protocols: Model

Based on connection-oriented network service

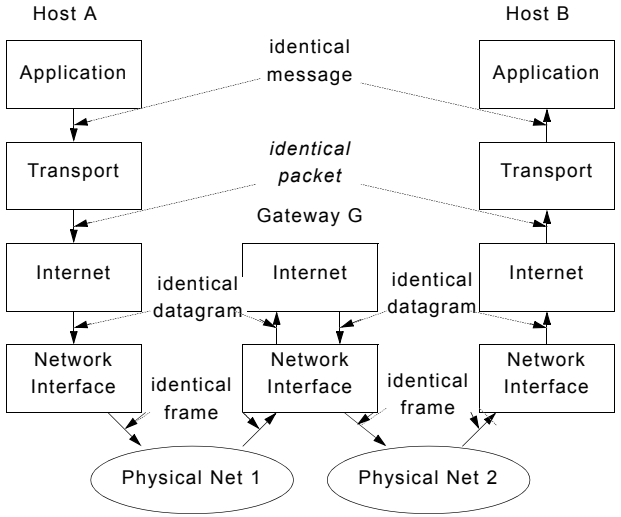


Based on connectionless network service

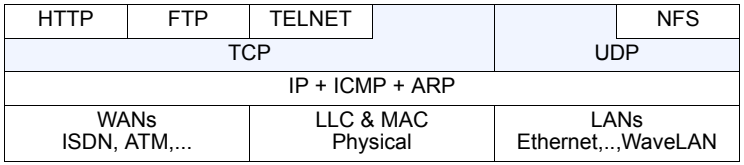


3. Internet Transport Layer (in General & Addressing)

Internet architecture



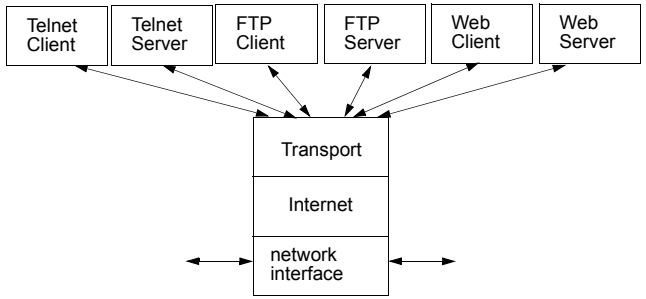
Well-Known Internet Protocols



ARP	=	**Address Resolution Protocol**
FTP	=	**File Transfer Protocol**
HTTP	=	**Hypertext Transfer Protocol**
IP	=	**Internet Protocol**
ICMP	=	**Internet Control Message Protocol**
IGMP	=	**Internet Group Management Protocol**
IMAP	=	**Interactive Mail Access Protocol**
LLC	=	**Logical Link Control**
MAC	=	**Media Access Control**
NFS	=	**Network File System**
NNTP	=	**Network News Transfer Protocol**
POP3	=	**Post Office Protocol**
SMTP	=	**Simple Mail Transfer Protocol**
TELNET	=	**Remote Login Protocol**
TCP	=	**TRANSMISSION CONTROL PROTOCOL**
UDP	=	**USER DATAGRAM PROTOCOL**

Transport Layer

Application

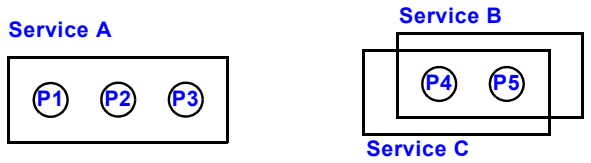


- **communication between applications required**
- **applications are realized by processes on the computer system**
  - i.e. interprocess communication

Transport layer
- **interprocess communication via communication networks**

Internet Protocol IP
- **enables endsystem-to-endsystem communication**

Port - Addressing Concept



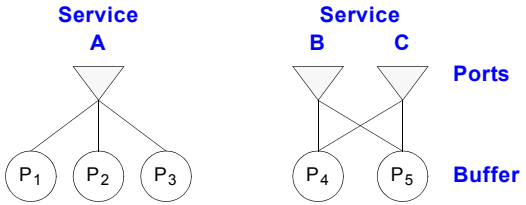
3 types of identifiers: names, addresses and routes

[Shoch 78]
"The NAME of a resource indicates WHAT we seek, an ADDRESS indicates WHERE it is, and a ROUTE tells HOW TO GET THERE

- **address identifies**
  - type of service or application
- **addressing by process number is unsuitable**
  - processes are generated/terminated dynamically, i.e., the process number is rarely known
  - relationship "service - process" not fixed
  - single process can supply multiple services
  - various processes can provide same service

⇒ **Concept of an abstract communication endpoint: Port**

Communication Ports



Service
- **related to exactly one single port**

Port access
- **asynchronous or**
- **synchronous**

Port
- **associated with buffer**

Reserved Port Numbers

Decimal	Keyword	UNIX Keyword	Description
0			Reserved
1	TCPMUX		TCP Multiplex
5	RJE		Remote Job Entry
7	ECHO	echo	Echo
9	DISCARD	discard	Discard
11	USERS	systat	Active Users
13	DAYTIME	daytime	Daytime
15		netstat	Network status program
17	QUOTE	qotd	Quote of the Day
19	CHARGEN	chragen	Character Generator
20	FTP-DATA	FTP-DATA	FILE TRANSFER PROTOCOL (DATA)
21	FTP	FTP	FILE TRANSFER PROTOCOL
23	TELNET	TELNET	TERMINAL CONNECTIONS
25	SMTP	SMTP	SIMPLE MAIL TRANSFER PROTOCOL
37	TIME	time	Time
42	NAMESERVER	name	Host Name Server

- **TCP and UDP have their own assignments**
  - this table shows some examples for TCP

Reserved Port Numbers (2)

Decimal	Keyword	UNIX Keyword	Description
43	NICNAME	whois	Who is
53	DOMAIN	nameserver	Domain Name Server
77		rje	any private rje service
79	FINGER	finger	Finger
80	**HTTP**	**HTTP**	**WORLD WIDE WEB**
101	HOSTNAME	hostname	NIC Host Name Server
102	ISO-TSAP	iso-tsap	ISO TSAP
103	X400	x400	X.400 Mail Service
104	X400-SND	x400-snd	X.400 Mail Sending
110	**POP3**	**POP3**	**REMOTE EMAIL ACCESS**
111	SUN RPC	sunrpc	SUN Remote Procedure Call
113	AUTH	auth	Authentication Service
117	UUCP-PATH	uucp-path	UUCP Path Services
119	NNTP	nntp	USENET News Transfer Protocol
129	PWDGEN		Password Generator Protocol
139	NETBIOS-SSN		NETBIOS Session Protocol
160-1023	Reserved		

4. UDP - User Datagram Protocol

Specification:
- **RFC 768**

UDP is a simple transport protocol
- **unreliable**
- **connectionless**
- **message-oriented**

⇒ **UDP is mostly IP with short transport header**
- source and destination port
- ports allow for dispatching of messages to receiver process

Characteristics
- **no flow control**
  - application may transmit as fast as it can / want and the network permits
- **no error control or retransmission**
  - no guarantee about packet sequencing
  - packet delivery to receiver not ensured
  - possibility of duplicated packets
- **may be used with broadcast / multicast**

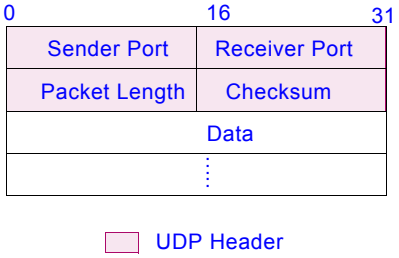
UDP: Message Format

Sender port
- **16 bit sender identification**
- **optional**
- **response may be sent to this port**

Receiver port
- **receiver identification**

Packet length
- **in byte**
- **including UDP header**
- **minimum: 8 (byte)**
  i.e. header without data

Checksum
- **of header and data for error detection**

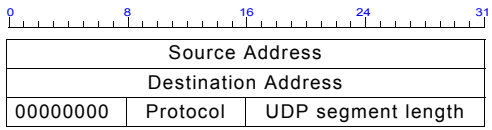


UDP: Checksum

Purpose:
- **error detection (header and data)**

Same algorithm as IP
- **one's complement of sum of 16-bit halfwords in one's complement arithmetic**

UDP checksum includes
- **UDP header (checksum field initially set to 0)**
- **data**
- **pseudoheader**
  - part of IP header
    - source IP address
    - destination IP address
    - protocol
    - length of (UDP) data
  - allows to detect misdelivered UDP messages



Use of checksum optional
- **i.e., if checksum contains only "0"s, it is not used**
- **transmit 0xFFFF if calculated checksum is 0**

UDP: Range of Application

Benefits of UDP
- needs very few resources (in endsystems, in each data unit)
- no connection establishment (hence, no overhead, faster transmission)
- simple implementation possible (e.g. suitable for boot PROM)
- applications can precisely control
  - packet flow
  - error handling / reliability
  - timing

Suitable for simple client-server interactions, i.e. typically
- 1 request packet from client to server
- 1 response packet from server to client

Used by
- DNS: Domain Name Service
- SNMP: Simple Network Management Protocol
- system status
- bootstrap protocol
- TFTP: Trivial File Transfer Protocol (but not ftp!)
- NFS: Network File System
- RTP: Real-time Transport Protocol

5. TCP - Transmission Control Protocol

TCP: the major Internet transport protocol

Motivation: network with connectionless service
- packets and messages may be
  - duplicated, in wrong order, faulty
  - i.e., with such service only, each application would have to provide recovery (error detection and correction)
- network or service can
  - impose packet length
  - define additional requirements to optimize data transmission
  - i.e., application would have to be adapted

⇒ **TCP is the Internet transport protocol providing**
- reliable end-to-end byte stream
- over an unreliable internetwork

History
- RFC 793: originally
- RFC 1122 and RFC 1323: errors corrected, enhancements implemented

What is TCP?

TCP is
- a communication protocol
  - not a piece of software
  - (compare: programming languages, compilers)

TCP specifies
- data and control information formats
- procedures for
  - flow control
  - error detection and correction
  - connect and disconnect
- as a primary abstraction
  - a connection
  - not just the relationships of ports (as a queue, like UDP)

TCP does not specify
- the interface to the application (sockets, streams)

TCP in Use

Each machine supporting TCP has a TCP transport entity
- library procedure
- user process
- part of kernel

TCP transport entity manages
- TCP streams
- interfaces to IP layer

TCP transport entity on sending side
- acceptes user data streams for local processes
- splits them into pieces <= 64 KB
  - typically 1460 bytes (to fit into single Ethernet frame with IP and TCP headers)
- sends each piece as separate IP datagram

TCP transport entity on receiving side
- gets TCP data from datagrams received at host
- reconstructs original byte streams

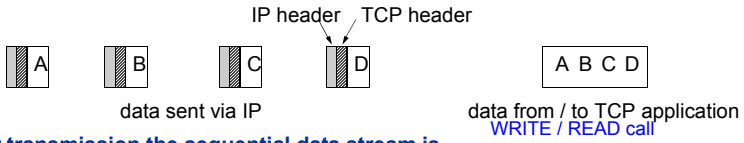
TCP must ensure reliability
- IP layer doesn't guarantee that datagrams will be delivered properly / in order
- TCP must handle this, e.g. timeout and retransmit / reorder

5.1 TCP Characteristics

Data stream oriented
- **TCP transfers serial byte stream**
- **maintains sequential order**

Unstructured byte stream
- **application often has to transmit more structured data**
- **TCP does not support such groupings into (higher) structures within byte stream**

Buffered data transmission
- **byte stream not message stream: message boundaries are not preserved**
  - no way for receiver to detect the unit(s) in which data were written



- **for transmission the sequential data stream is**
  - divided into segments
  - delayed if necessary (to collect data)
  - for more efficient transmission (e.g. network utilization)

TCP Characteristics (2)

Virtual connection
- **connection established between communication parties before data transmission**

Two-way communications (fully duplex)
- **data may be transmitted simultaneously in both directions over a TCP connection**

Point-to-point
- **each connection has exactly two endpoints**

Reliable
- **fully ordered, fully reliable**
  - sequence maintained
  - no data loss, no duplicates, no modified data

TCP Characteristics (missing)

(missing) Characteristics

- **no broadcast**
  - no possibility to address all applications
  - with connect, however, not necessarily sensible
- **no multicasting**
  - group addressing not possible
- **no QoS parameters**
  - not suited for different media characteristics
- **no real-time support**
  - no correct treatment/communications of audio or video possible
  - e.g. no forward error correction

Protocol Elements & Features

Error detection
- **through checksum**

Piggybacking:
- **control information and data can be transmitted within the same segment**

Out-of-band data: expedited data
- **important information sent to receiver**
- **i.e. should get to receiver's application before data that was sent earlier**

PUSH operation
- **data is not stored in a buffer**
- **sent immediately and immediately made available to application on receiver's end**

```
example <CR> end of line at terminal emulation
```

Urgent flag
- **send and transfer data to application immediately**

```
example <Crtl C>
arrival interrupts receiver's application
```

5.2 Connection - Addressing

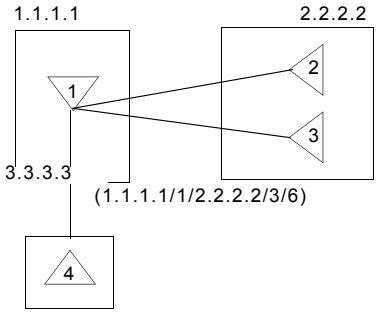
TCP service obtained via service endpoints on sender and receiver
- **typically socket**
- **socket number consists of:**
  - IP address of host and
  - 16-bit local number (port)

Port
- **TCP's name for TSAP**

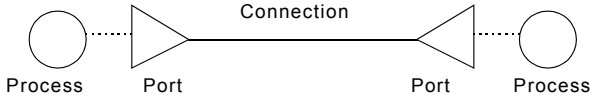
TCP connection is clearly defined by a quintuple consisting of
- **IP address of SENDER and RECEIVER**
- **port address of SENDER and RECEIVER**
- **TCP protocol identifier**

⇒ **Applications can use the same local ports for several connections**

IP addr.sender/port sender/ ..
(1.1.1.1/1/2.2.2.2/2/6)
../IP addr.rec/port rec/TCPid



Addressing



Passive open:
- **process indicates that it accepts connect request**

Active open:
- **process requests a connection**

Addressing:
- **port number + protocol identification**
  - clearly identifies entity in the endsystem
- **IP address**
  - clearly identifies endsystem

5.3 TCP Protocol

A key feature:
- **every byte on TCP connection has its own 32-bit sequence number**

Separate sequence numbers used for
- **data**
- **acknowledgements**
- **window mechanism**

Remember:
- **32-bit sequence number space was big in early days of the Internet**
- **nowadays, it can be consumed very fast**

TCP Protocol (2)

TCP entitites exchange data in form of SEGMENTS

TCP segment consists of
- **fixed 20 byte header (plus optional part)**
- **zero or more data bytes**

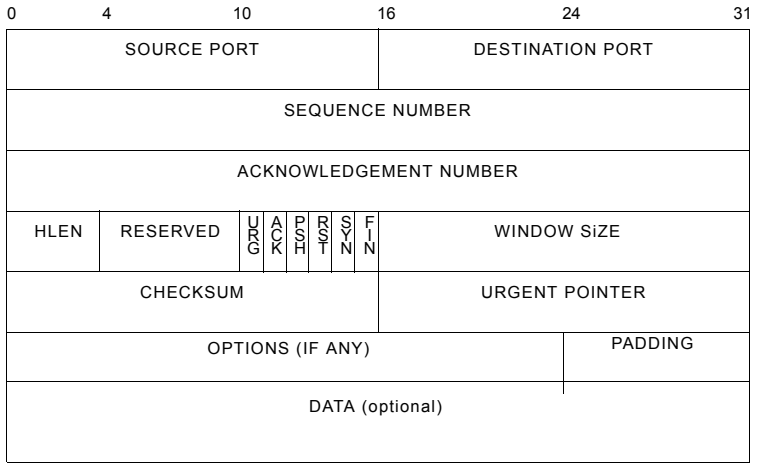
TCP software (entity) decides
- **segment size to be used**
  - data from several writes can be accumulated into one segment
  - or data from one write can be split into several segments
- **limits**
  - each segment (including TCP header) must fit into 65515 byte IP payload
  - segment must fit into maximum transfer unit (MTU) of visited networks
    - each network may have MTU, depending on L2 technology used
    - often 1500 byte (Ethernet payload size), typical upper bound on segm. size

TCP Protocol (3)

TCP uses: SLIDING WINDOW protocol

- **sender starts timer when it transmits segment**
- **receiving TCP entity sends back a segment**
  - with data if any, otherwise without data
  - acknowledgment number equal to next sequence number it **EXPECTS** to receive
- **if sender's timer goes off before acknowledgement arrives, segment is retransmitted**

5.4 TCP Segment (PDU) Format



TCP Segment (PDU) Format (2)

SOURCE PORT **DESTINATION PORT**	**local endpoints of connection**
SEQUENCE NUMBER	**Number of transmitted bytes (each byte of the "message" is numbered)**
ACKNOWLEDGEMENT	**Byte number referring to the acknowledgement, specifies next byte expected**
HLEN	**Length of the header in 32 bit words needed since Options field is of variable length indicates start of data within segment (in 32-bit words)**
RESERVED	**not used**

TCP Segment (PDU) Format (3)

FLAGS — **Bits from left to right**

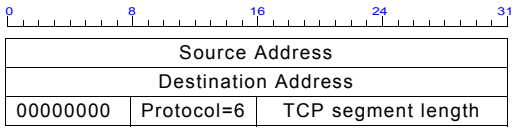
1. **URG** — **Urgent Pointer is being used**
2. **ACK** — **AckNo is valid (if ACK=0 then no acknowledgment in segment)**
3. **PSH** — **data transferred with PUSH**
   receiver should give received data to application immediately, not waiting & buffering until more / full buffer has been got
4. **RST** — **Reset: connection is being reset**
5. **SYN** — **used to establish connections (synchronize seq. numbers)**
   SYN=1 & ACK=0: connection request
   SYN=1 & ACK=1: connection accept
6. **FIN** — **release connection**

WINDOW SIZE — **Buffer size in bytes**
used for variable-sized sliding window
window size field indicates #bytes sender may transmit beginning with byte acknowledged
window size = 0 is valid: bytes up to ACK-1 received but receiver does not want more data at the moment later permission for more data by sending segment with same ACK and non-zero window size

TCP Segment (PDU) Format (4)

CHECKSUM **Checksum header, data, and pseudoheader**
for calculation: TCP checksum field is set to 0 and data field padded with additional zero byte if length is odd
one's complement of sum of 16-bit halfwords in one's complement arithmetic
receiver's calculation on entire segment including checksum field should result in 0

- **pseudoheader**
  - part of IP header
    - source IP address
    - destination IP address
    - protocol
    - length of TCP segment (including header)
  - allows to detect misdelivered packets
  - but violates protocol hierarchy

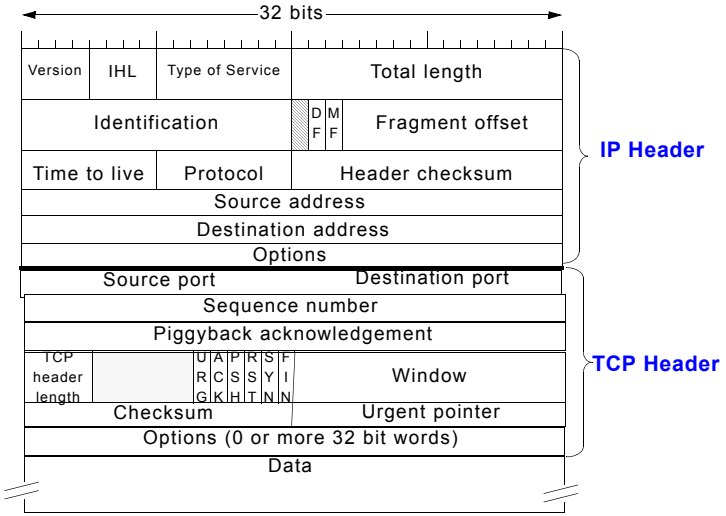


TCP Segment (PDU) Format (5)

URGENT POINTER **Byte offset to the current sequence number at which important data starts**

OPTIONS **for optional facilities, not covered by regular header, e.g.,**
allows to specify max. TCP payload host may accept without option, it defaults to 536 byte payload
window scale option (to deal with 64 KB window limitation)
shift window size field up to 14 bits to left
maximum window then 2^{30} byte
selective repeat instead of go-back-n
NAKs to allow receiver to request a specific segment

TCP/IP Header Format

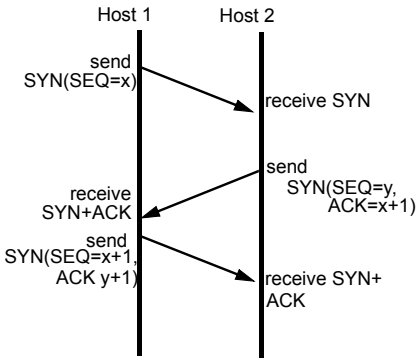


5.5 Establishing a Connection

One passive & one active side
- **server: wait for incoming connection using LISTEN and ACCEPT**
- **client: CONNECT (specifying IP addr. and port, max. TCP segment size)**

Three-Way-Handshake
- **CONNECTING through 3 packets**

Comments
- **when establishing a connection**
  - initial sequence numbers of both partners are also exchanged and acknowledged
  - initial seq.# is not 0
- **SYN segment consumes one byte of sequence space**
  - in order to be acknowledged unambiguously



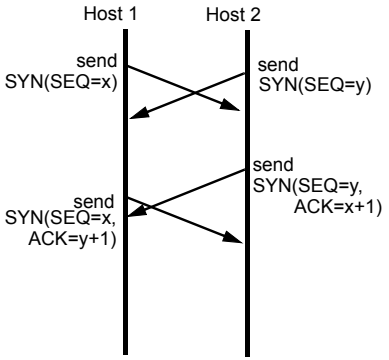
Establishing a Connection (2)

Comments

- **if on server side no process is waiting on port (no process did LISTEN)**
  - reply segment with RST bit set is send to reject connection attempt
- **process listening on port may accept or reject**

Call collision

- **still only one single connection will be established even when**
  - both partners actively try to establish a connection simultaneously



5.6 Connection Release

Connection release for pairs of simplex connections

- **each direction is released independently of other**

Connection release by either side sending a segment with FIN bit set

- **no more data to be transmitted**
- **when FIN is acknowledged, this direction is shut down for new data**
-

Directions are released independently:

- **other direction may still be open**
- **full release of connection if both directions have been shut down**

Connection Release (2)

Systematic disconnect by 4 segments

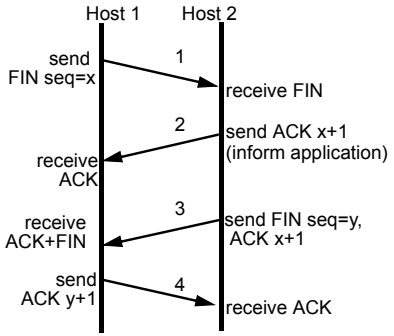
- **between 2nd and 3rd**
  - partner 2 can still send data to partner 1

Reduction of packet number possible:

- **first ACK and second FIN may be contained in same segment (3 segments instead of 4)**

CONNECTION INTERRUPT: **Opposite side cannot transmit data anymore**

- **immediate acknowledgement, release of all resources**
- **data in transit may be lost**

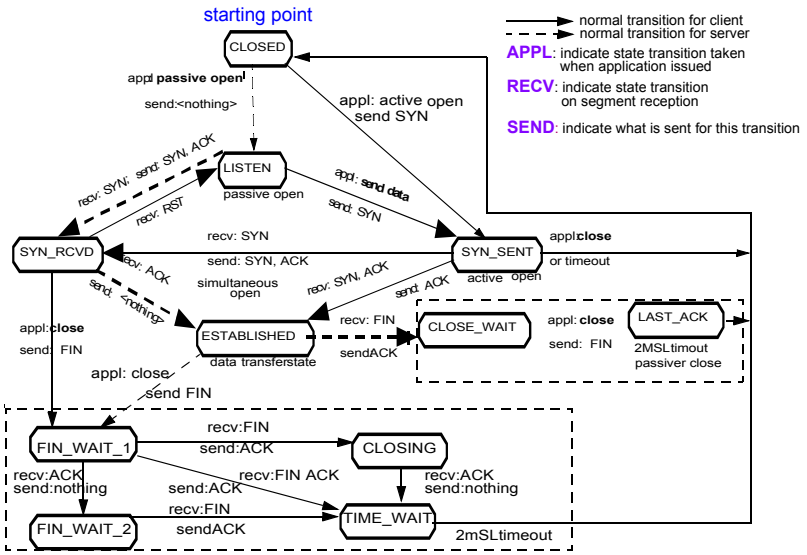


5.7 Connection Management Modelling

States

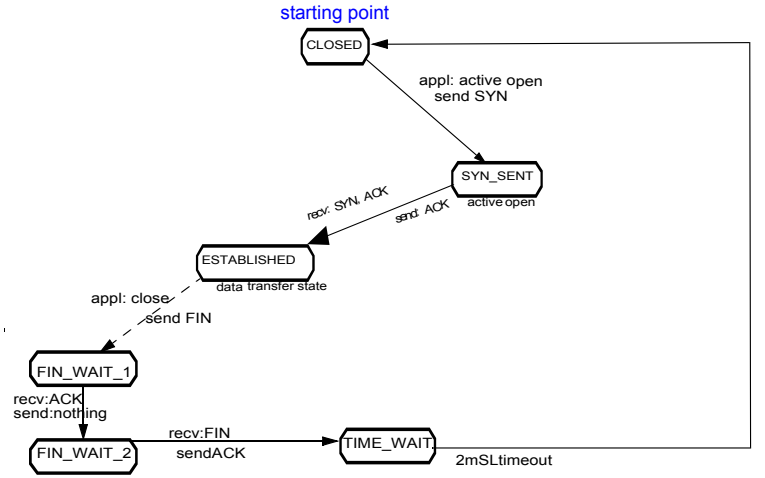
State	Description
CLOSED	No connection is active or pending
LISTEN	Server is waiting for an incoming call
SYN RCVD	Connection request has arrived; wait for ACK
SYN SENT	Application has started to open a connection
ESTABLISHED	Normal data transfer state
FIN WAIT 1	Application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

Connection Management Modelling: Finite State Machine



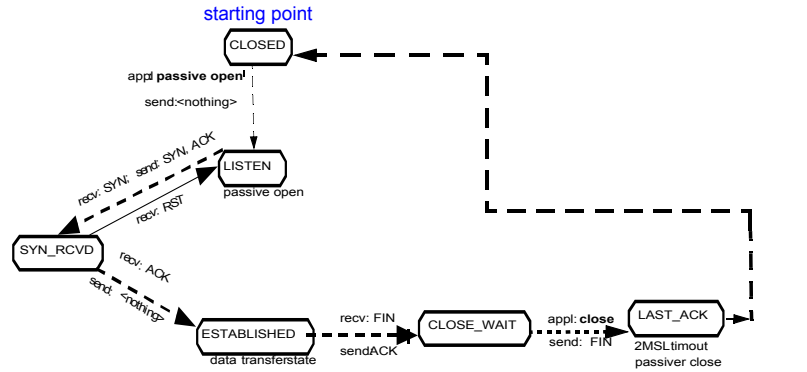
Connection Management Modelling: Finite State Machine

Typical sequence of TCP states visited by client TCP



Connection Management Modelling: Finite State Machine

Typical sequence of TCP states visited by server-side TCP



5.8 Flow Control

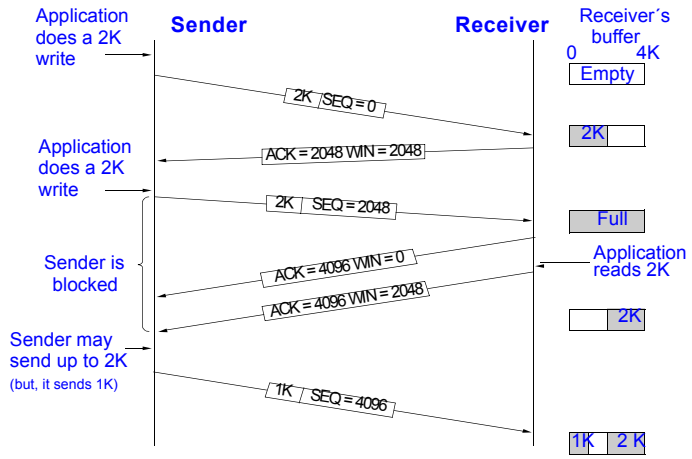
Flow control: "sliding window" mechanism

- **acknowledgement and sequence number**
  - acknowledgments refer to byte positions
    - not to whole segment
  - sequence numbers refer to the byte position of a TCP connection
- **positive acknowledgement**
- **cumulative acknowledgements**
  - byte position in the data stream up to which all data has been received correctly
  - reduction of overhead

Variable window sizes (credit mechanism)

- **implementation**
  - the actual window size is also transmitted with each acknowledgement
  - permits dynamic adjustment to existing buffer

TCP Flow Control: Example



TCP Flow Control: Special Cases

Optimization for low throughput rate

- **problem: Telnet connection to interactive editor reacting on every keystroke**
  - 1 character typed requires up to 162 byte
    - data: 20 bytes TCP header, 20 bytes IP header, 1 byte payload
    - ACK: 20 bytes TCP header, 20 bytes IP header
    - editor echoes character: 20 bytes TCP header, 20 bytes IP header, 1 byte payload
    - ACK: 20 bytes TCP header, 20 bytes IP header
- **approach often used**
  - delay acknowledgment and window update by 500 ms (hoping for more data)

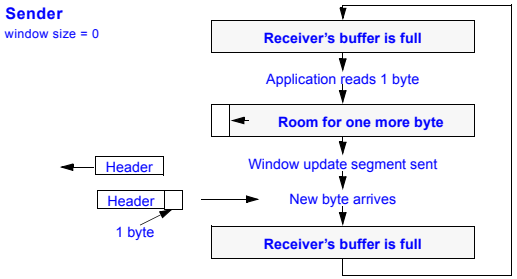
Nagle's algorithm, 1984

- **algorithm**
  - send first byte immediately
  - keep on buffering bytes until first byte has been acknowledged
  - then send all buffered characters in one TCP segment and start buffering again
- **comment**
  - effect at e.g. X-windows: jerky pointer movements

TCP Flow Control: Special Cases (2)

Silly window syndrome (Clark, 1982)

- **Problem:**
  - data on sending side arrives in large blocks
  - but receiving side reads data at one byte at a time only

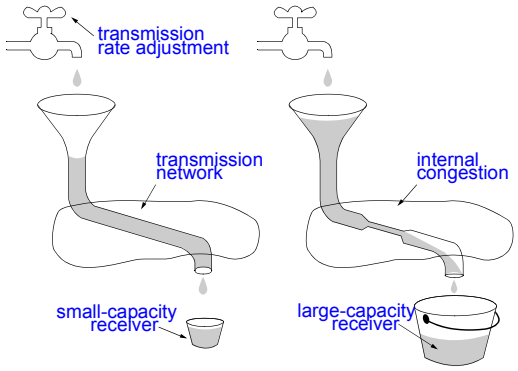


- **Clark's solution:**
  - prevent receiver from sending window update for 1 byte
  - certain amount of space must be available in order to send window update
    - min(X,Y)
      X = maximum segment size announced during connection establishment
      Y = buffer / 2

5.9 Congestion Control

(understandably) diverse objectives

- **endsystem**
  - optimize its own throughput
  - possibly at the expense of other endsystems
- **network**
  - optimizes overall throughput

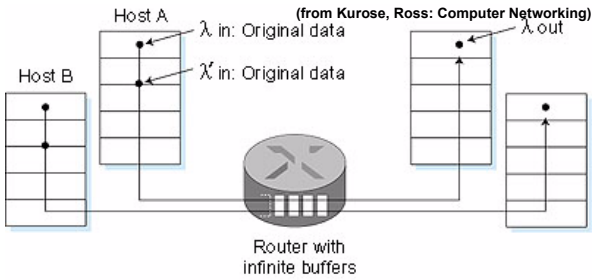


in example:

- **two different problems**
  - receiver capacity
  - network capacity
- **cannot be distinguished easily at all places**
- **should be differentiated**

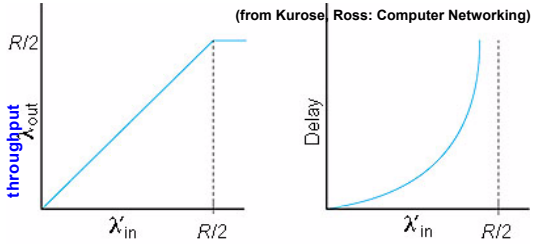
Congestion Considerations

Simple example
- **2 senders**
  - transmitting at avg. rate λ_{in} bytes/sec
- **router with infinite buffer**
- **outgoing link capacity R**
- **no error recovery**
- **no flow control**



throughput per connection
- **scales linearly up to R/2**

delay
- **scales exponentially**
- **large queueing delays as consequence of packet-arrival rate near link capacity!**



Congestion Considerations (2)

General problem
- congestion => more delays at endsystem
- higher delays => retransmissions (because of timeouts)
- i.e. => additional load, increased congestion

I.e. 2 problem areas: (to be considered separately)
- **receiver capacity ("receiver granted window")**
- **network capacity ("congestion window")**

i.e. to be applied to the send window

(effective send window) = Min (receiver granted window,congestion window)

TCP strategy
- **TCP reduces transfer rate at high network load**
- **methods**
  - "slow start" generalization: additive increase
  - "congestion avoidance" generalization: multiplicative decrease

Congestion Control - Additive Increase: Slow Start

Objective
- **to avoid immediate network overload**
  - after just recent overload has finished
  - after timeout
  - at the start of a data transmission

Method: congestion window
- defined/initiated at connection set-up
  - initial max. window size = 1 segment
  - threshold (based on previous max. window size)
- max. size of segment to be transferred
  - to be doubled as long as
    1. segment size below a certain threshold and
    2. all ACKs arrive in time (i.e. correct segment transfer)

    (each burst acknowledged, i.e. successfully transmitted, doubles congestion window)
  - to be linearly increased
    1. segment size above a certain threshold and
    2. all ACKs arrive in time (i.e. correct segment transfer)

Cong. Ctrl - Multiplicative Decrease: Congestion Avoidance

Threshold
- adaptive
- parameter in addition to the actual and the congestion window

Assumption
- threshold, i.e. adaptation to the network: "sensible window size"

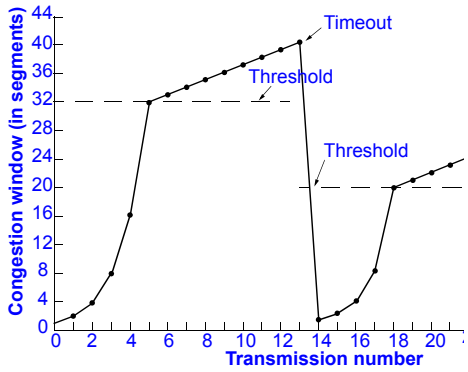
Use: On timeout
- threshold is set to half of current congestion window
- congestion window is reset to one maximum segment
- use slow start to determine what the network can handle
  - exponential growth stops when threshold is hit
  - from there congestion window grows linearly (1 segment) on successful transmission

Congestion window and threshold
- congestion window < threshold
  - exponential growth: congestion window doubled
- congestion window >= threshold
  - linear growth: increase congestion window by 1 segment each

Congestion Control: Example

Parameters:
- **maximum segment size = 1024 bytes**
- **initial congestion window = 64 KB**

Areas
- **previously (transmission# 0)**
  - congestion window = 64 KB
  - timeout: i.e. threshold 32 KB
  - congestion window = 1 KB
- **exponential area:**
  - "slow start"
- **linear area**
  - "congestion avoidance"



Note: but always
- **observe receiver's window size, i.e. use minimum of**
  - congestion window
  - actual send window (receiver status)

5.10 Timer Management

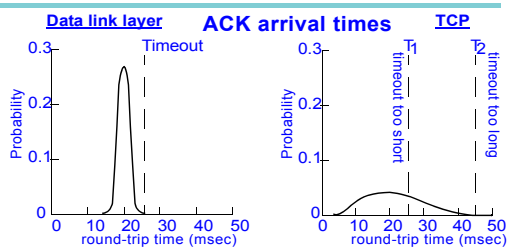
TCP uses several timers for different purposes

Retransmission timer as most important one
- **timer is set when segment is sent**
- **if ACK arrives before timer expires: timer is stopped**
- **if timer expires before ACK arrives: segment is retransmitted**
  - and new timer is set

Question: How long should the timeout interval be?

Timer Management (2)

In principle similar to issue at data link layer, BUT:
- **at transport layer much more difficult**
  - L2: delay is highly predictable (low variance)
  - L4: delay can differ significantly



Situation in the Internet: round-trip time may vary immensely
- **timeout too short**
  - fast reaction due to any errors
  - but correct packets retransmitted
- **timeout too long**
  - too slow reaction due to errors
  - less additional traffic

Better: adaptive timeout intervals
- **timeout period (until retransmit)**
  - continuous adaptation to pre-calculated, expected waiting period
- **waiting period increases for each retransmission of the same segment**

Timer Management (3)

Algorithm (Jacobson, 1988)
- **maintain RTT variable per connection**
  - best current estimate of round-trip time to destination
- **start timer when sending a segment**
  - to determine duration until acknowledgement arrives
  - and to trigger retransmission if needed
- **when acknowledgment is received before timer expires**

$$RTT = (\alpha \times RTT) + ((1-\alpha) \times New\_Round\_Trip\_Sample)$$
$$Timeout = \beta \times RTT$$

α **Smoothing factor**
- 0..1 determines importance of the history
- 0: only current/last value is relevant
- **7/8 TYPICAL VALUE**
- 1: only old values are relevant

β **(later/today proportional to the standard deviation of ack arrival time)**
- 1: faster error correction
- 2: initial value in implementations
- **COMMENT:** usually hard to define (as fixed value)

RTT Calculation in Case of Error Recovery

How to handle dynamic RTT estimation when segment must be resent?

Situation: ambiguous measurement
- **t_1: retransmit initiated because of timeout**
- **t_2: acknowledgement arrives**

Question:
- **does confirmation refer to**
  - 1. segment which has previously been thought as being lost? or
  - 2. segment which has been sent last?
- **problem: no differentiation possible**

This influences the definition/parameter setting of the timeout
- **by the RTT measured**

RTT Calculation in Case of Error Recovery (2)

Situation: ambiguous measurement

1st assumption: reference to the data sent first (originally)
- **but situation is following**
  - short-term increase in the delay
    - for any unknown reason
  - but not really any loss of data
    - acknowledgement arrives shortly after timeout
- **result**
  - New-Round_Trip_Sample: has very high value
  - RTT increases
- **sender**
  - now detects (at later time) segments that have been lost
  - retransmits at a later point in time
  - consequently acknowledgements are also returned later
- **global effect:**
  - RTT may increase even more and more

RTT Calculation in Case of Error Recovery (3)

Situation: ambiguous measuring

2nd assumption: reference to last retransmitted data
- **but situation occurred due to**
  - Internet in fact lost data
- **result**
  - New_Round_Trip_Sample: has smaller value
  - RTT is reduced
- **sender**
  - may set time interval which may be too short
  - the acknowledgement of the first packages will again further reduce RTT
- **global effect:**
  - RTT gets too small
  - usually at 1/2 of the actual value
    - i.e., on the average each segment will be sent twice

RTT Calculation in Case of Error Recovery (4)

Fix: (Phil) Karn's algorithm
- **do not update RTT on any segments that have been retransmitted**
- **ignore the measurements of lost packets**
  - but this would not reflect actual system changes in a correct manner
- **previous algorithm**

$$RTT = (\alpha \times Old\_RTT) + ((1-\alpha) \times New\_Round\_Trip\_Sample)$$

$$Timeout = \beta \times RTT$$

- **timer backoff strategy**

$$New\_Timeout = \Upsilon \times TimeOut$$

  - increase timeout for each lost segment until segment has arrived
  - γ-value: typically 2
- **comment:**
  - originated from Phil Karn
  - Karn used TCP based on (very unreliable) radio transmission

RTT Calculation in Case of Error Recovery (5)

Further improvements

- **Karn's algorithm**
  - bad adjustment whenever
    - variation of consecutive delays is large
    - e.g. short-long-short-long ...
- **high load means**
  - often large fluctuations within the delays
- **therefore deviation to be used (instead of fixed β)**
  - make β roughly proportional to standard deviation of acknowledgment arrival time probability density function
  - large variance means large β and vice versa
  - use mean deviation as cheap estimator of standard deviation
  - maintain another smoothed variable D

$$D = \alpha \times D + (1 - \alpha) \times |RTT - New\_Round\_Trip\_Sample|$$

  - α may be same or different value used to smooth RTT
  - most TCP implemetations set timeout interval to $Timeout = \mathbf{RTT} + 4 \times \mathbf{D}$

5.11 Segments, Fragmenting (and Reassembling)

Segments

- **TCP DATA STREAM split into segments**
  - **SEGMENTS** sent as IP packets

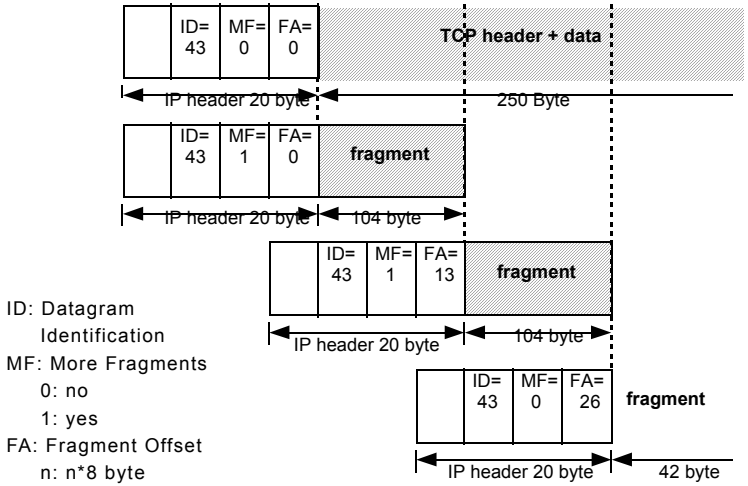
Fragments

- **IP packets are split (if necessary) into FRAGMENTS in order to adapt them to underlying networks**

Transport layer

- **reassembles segments and fragments**

Fragmenting (and Reassembling) (2)



5.12 TCP - Application Areas

Benefits of TCP

- **reliable data transmission**
- **efficient data transmission despite complexity**
  - (up to 8Mbps on 10Mbps ethernet)
- **can be used with LAN and WAN for**
  - low data rates (e.g. interactive terminal) and
  - high data rates (e.g. file transfer)

Disadvantages when compared with UDP

- **higher resource requirements**
  - buffering, status information, timer usage
- **connection set-up and disconnect necessary**
  - even with short data transmissions

Applications

- **file transfer (FTP)**
- **interactive terminal (Telnet)**
- **e-mail (SMTP)**
- **X-Windows**

TCP in Radio Networks

TCP characteristics
- **if timeout occurs, TCP assumes that network is congested**
- **therefore "slow start" algorithm**

Mobile communications channel
- **is unreliable and may contain losses**

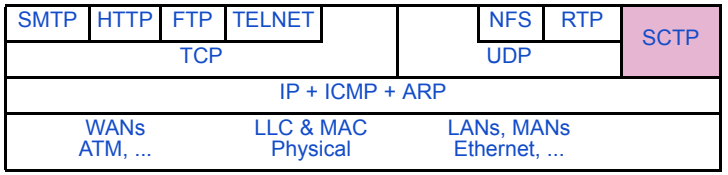
TCP's behavior on radio channels
- **TCP slows down transmission, inefficient**
- **here it would be better to immediately retransmit the segment**

TCP variants for wireless network scenarios
- **indirect TCP**
- **snooping TCP**
- **etc.**

⇒ **more in MOBILE COMMUNICATIONS lecture**

6. Stream Control Transmission Protocol (SCTP)

- **Additional transport protocol from IETF (add. to TCP+UDP)**



Specification in
- **RFC 2960**
  - Stream Control Transmission Protocol
- **RFC 2719**
  - Architectual Framework for Signaling Transport
- **RFC 3057**
  - ISDN Q.921-User Adaptation Layer
- **see also**

  `http://www.sctp.de/`

SCTP: Motivation

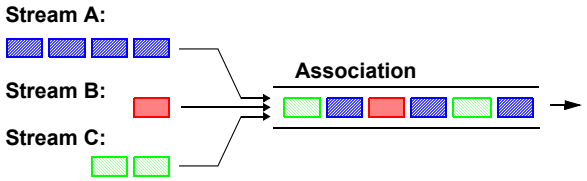
TCP too limited for some applications:
- **e.g., transport signaling from PSTN networks (SS7) over IP-based networks**

Goals:
- **initial goal: replace SS7 signaling in PSTN with SCTP**
- **now: SCTP as a universal transport protocol (e.g., for HTTP)**
- **future: replacing TCP??**

Examples:
- **Strict order-of-transmission delivery of data with multiple streams**
  - partial order within a stream of multiplexed streams sufficient
- **Stream-orientation of TCP inconvenient (application must set record markings)**
  - better: message-orientation
- **TCP cannot deal with multi-homing**
  - i.e., one server with several IP addresses
- **TCP is vulnerable to DoS attacks**
  - e.g., SYN flooding

SCTP Concepts: Association and Streams

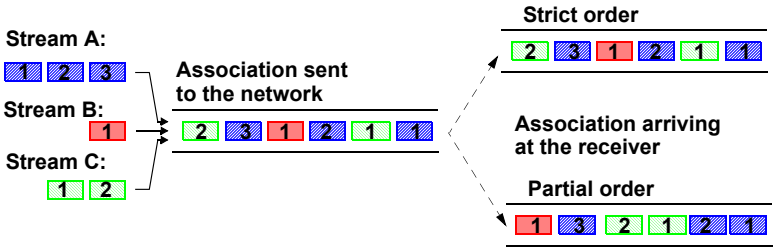
- **Connection-oriented: concept of 'association'**
  - bi-directional
  - generalization of TCP-connections:
    - each association endpoint can have several IP addresses (multi-homing)
    - each association can contain several streams (multi-streaming)



  - Stream: sequence of user messages to be delivered in order
    ⇒ in contrast to the notion of 'stream' of TCP!!
- **Reliable data transfer: confirmed, no duplicates, error-free**

SCTP Concepts: Strict vs. Partial Order

- **Strictly ordered delivery optional**
  - packets of a stream within an association are delivered in order (partial order)
  - optional: retain order between packets of all streams (strict order)
    - strict order: data transmission stalled if one stream is stalled
    - partial order: transmission for non-stalled streams can continue

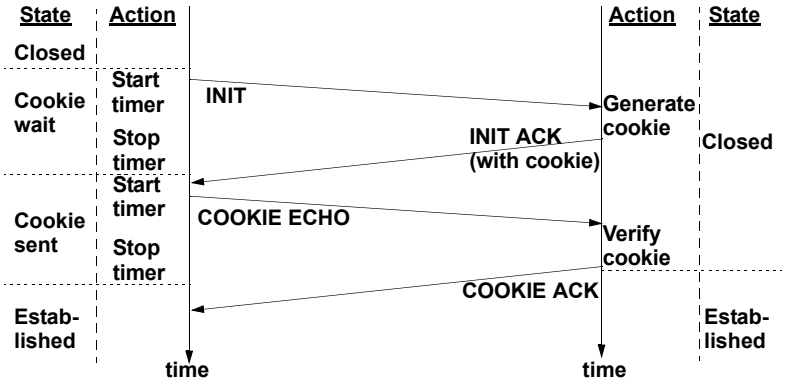


- Example: HTTP with multiple embedded files (images)
  - Order of arrival of image data not relevant
  - Retrieving the text can continue even if loading the image is blocked (e.g., if the image is located on a different server which is highly loaded)

SCTP: Further Concepts

- **Message segmentation according to path-MTU**
  - Path-MTU: maximum transfer unit supported on the path between the endpoints
  - Path-MTU discovery mechanism as specified in RFC 1191
- **Test whether the communication partner is alive: Heartbeats**
- **Flow control and congestion control similar to TCP (Selective Ack,...)**
  - Coexistance with TCP
- **Security means:**
  1. 32-bit checksum (Adler-32, CRC-32 under discussion)
  2. 4-way handshake using cookies against DoS attacks

SCTP: 4-way Handshake



- **No half-open states as in TCP**
- **No state information kept at the station receiving the 'INIT' message**
  - no vulnerability for SYN flooding
  - state information established only after the third step, the 'COOKIE' message
- **To increase efficiency**
  - user data can be sent already with the 'COOKIE' and 'COOKIE ACK' messages

SCTP: Message format

Multiplexing of several user messages: Chunk Bundling

- Chunk: part of an SCTP packet belonging to a single stream

Common Header
Chunk 1
Chunk 2
...
Chunk n

Common Header:

Source Port	Destination Port
Verification Tag	
Checksum	

- **source port / destination port (2 Byte each): As in TCP or UDP**
- **verification tag: for validation of the sender of the SCTP message**
  - protection against blind attacks (unauthorized shutdown of association)
- **checksum:**
  - Adler-32: currently proposed in the RFC
  - CRC-32: proposed recently, better error detection properties for small packets

SCTP: Chunk Format

General format of a chunk:

Chunk Type	Chunk Flags	Chunk Length
Chunk Value		

- **Chunk Type: type of information in the Chunk Value field**
  - e.g., type=0: payload data; type=1: INIT message,...
- **Chunk Flags: depend on the chunk type**
- **Chunk Length: length of the chunk in bytes (including type, flags, length)**

SCTP: INIT Chunk Format

Type=1	Flags: None	Chunk Length
Initiate Tag		
Advertised Receiver Credit Window		
Number of outbound streams	Number of inbound streams	
Initial Transmission Sequence Number		
Optional/Variable-Length Parameters		

- **Initiate Tag: random number used in all subsequent messages**
  - protect against blind attacks
- **Advertised Receiver Credit Window:**
  - dedicated buffer space reserved for the association
- **Number of outbound streams:**
  - the sender of this INIT chunk wants to open
- **Number of inbound streams:**
  - maximum the sender of this INIT message can support
- **Variable-Length parameter**
  - a.o.: list of IP addresses (multi-homing!) being part of the association

SCTP: DATA Chunk Format

Example: Format of a DATA chunk (i.e., non SCTP control message)

Type=0	Flags: U | B | E	Chunk Length
Transmission Sequence Number (TSN)		
Stream Identifier S	Stream Sequence Number n	
Payload Protocol Identifier		
User Data (seq n of Stream S)		

- **Flags (one bit each):**
  - **U**nordered bit: '1' indicates unordered DATA chunk, '0' an ordered DATA chunk
  - **B**eginning bit: '1' first fragment of a user message
  - **E**nd bit: '1' last fragment of a user message
- **Transmission Sequence Number (32 bits) => 4,300 Mio numbers**
  - unique per stream
  - used for acknowledgments and duplicate recognition
    - acknowledgements are given per message (selective ACK)
    -
- **...**

SCTP: DATA Chunk Format (2)

Type=0	Flags: U | B | E	Chunk Length
Transmission Sequence Number (TSN)		
Stream Identifier S	Stream Sequence Number n	
Payload Protocol Identifier		
User Data (seq n of Stream S)		

- **...**
- **Stream Identifier (16 bits)**
- **Stream Sequence Number (16 bits)**
  - unique per stream
  - used to assure sequenced delivery within a stream
  - separate acknowledgement mechanism from sequenced delivery
    - If fragmentation is necessary, stream sequence number is the same for all fragments
- **Payload Protocol Identifier (e.g., RTP)**
- **User Data: the actual user data to send via SCTP**