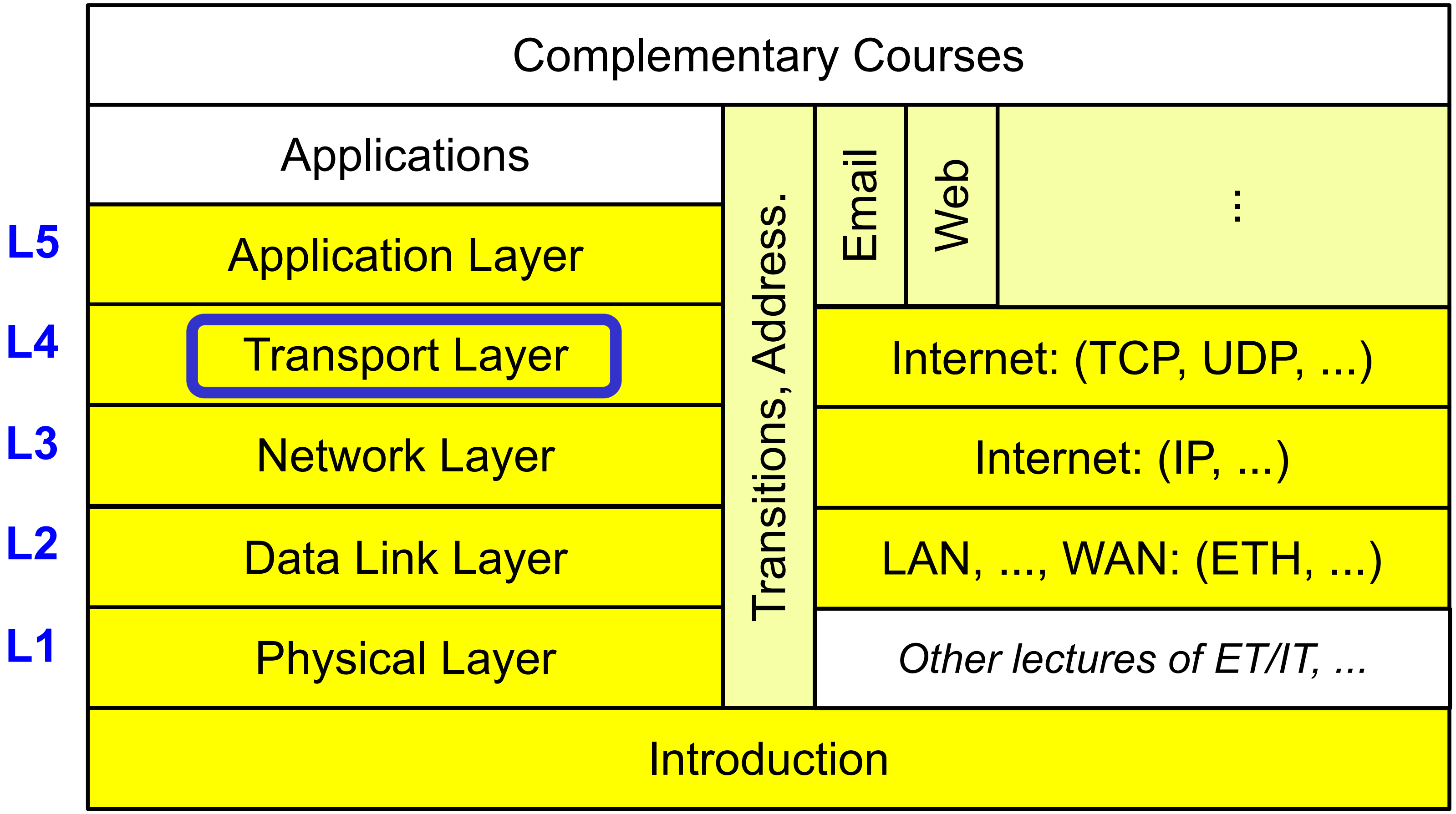
Computer Networks I

Transport Layer

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

IBR, TU Braunschweig
Mühlenpfordtstr. 23, D-38106 Braunschweig, Germany,
Email: wolf@ibr.cs.tu-bs.de

Scope


Complementary Courses
Applications
L5
Application Layer
L4
Transport Layer
L3
Network Layer
L2
Data Link Layer
L1
Physical Layer
Transitions, Address.
Email
Web
:
Internet: (TCP, UDP, ...)
Internet: (IP, ...)
LAN, ..., WAN: (ETH, ...)
Other lectures of ET/IT, ...
Introduction

Overview

1 Transport Layer Function

Provide data transport

- reliably
- efficiently
- at low-cost

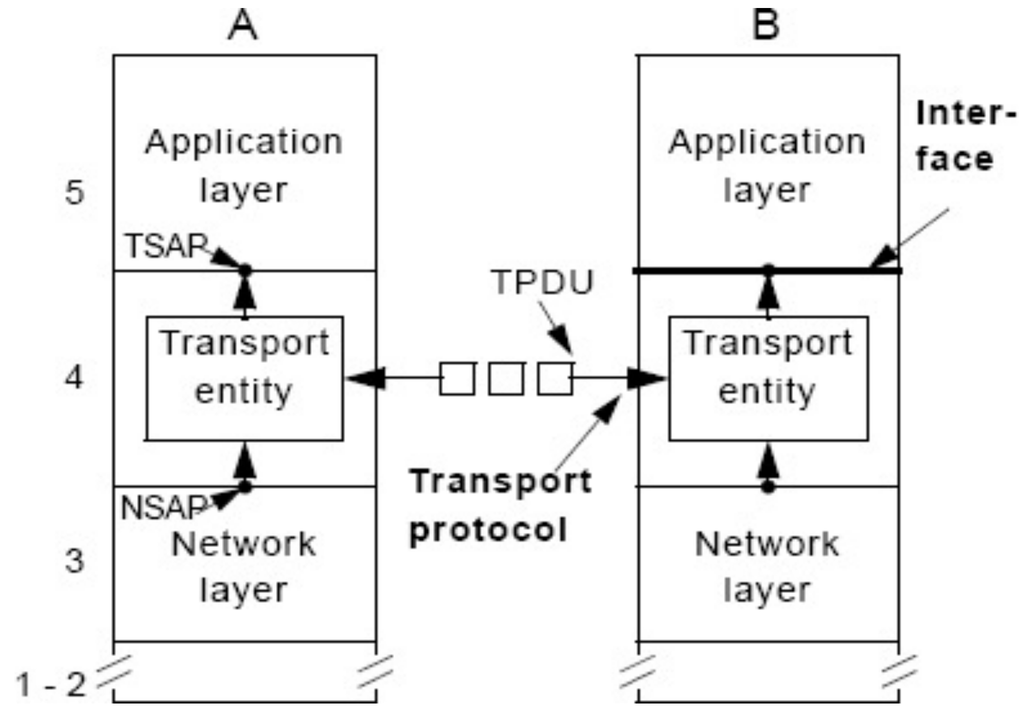
for

- process-to-process (applications)
- i.e. at endsystem-to-endsystem

(if possible) independent from

- particularities of the networks (lower layers) used

Transport Service



Connection oriented service
- 3 phases: connection set-up, data transfer, disconnect

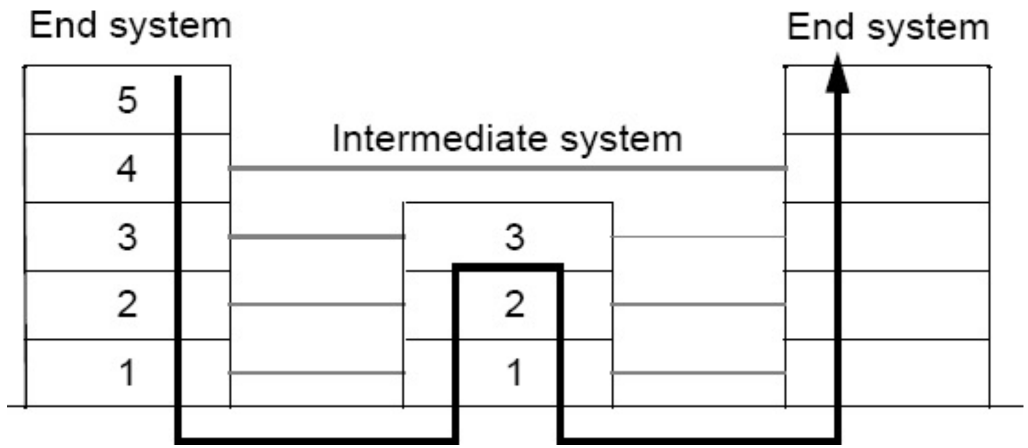
Connectionless service
- transfer of isolated units

Realization: transport entity
- software and/or hardware?
- software part usually contained within kernel (process, library)

Transport Service

Similar services of
- network layer
  and transport layer:

Why 2 Layers?



Network service
- not influenced by the user
- independent from application & user
- provides, for example,
  - "only" connection oriented communications
  - or "only" unreliable data transfer

Transport service: improve the quality of Network Service
- provide better service for users (and higher layers) than given by the network layer, e.g.
  - reliable service

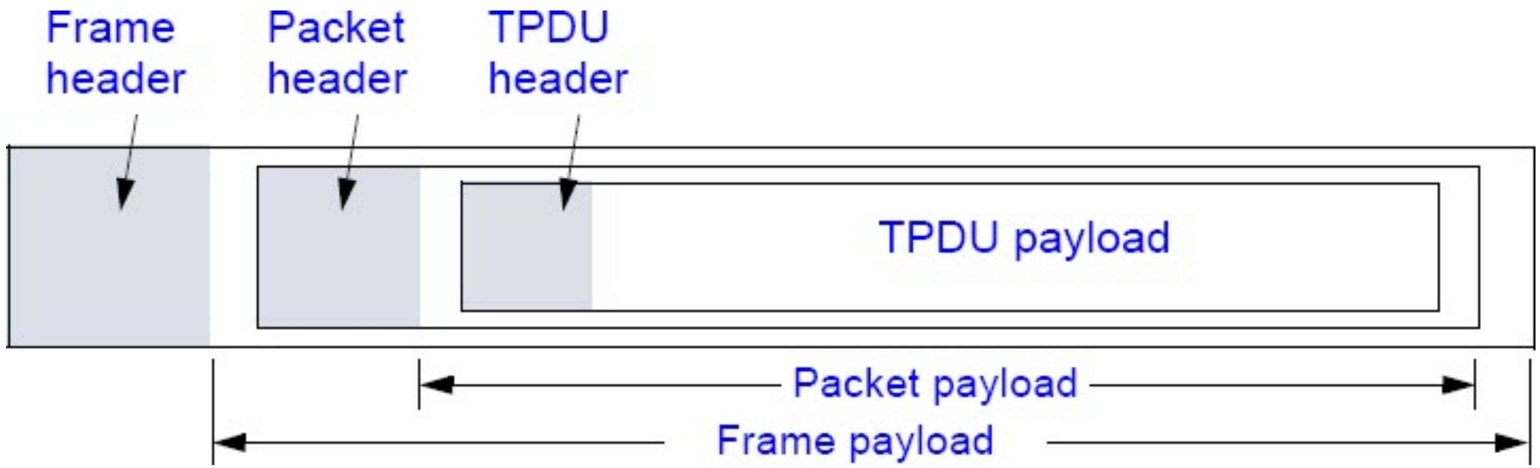
Transport Service: Terminology

Entities exchanged:

Layer	Data Unit
Transport	TPDU / Message
Network	Packet
Data Link	Frame
Physical	Bit/Byte (bitstream)

`TPDU: Transport Protocol Data Unit`

Nesting of TDPUs, packets, and frames:



Transport Service Primitives

Berkeley Socket Primitives:

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

2 Elements of Transport Protocols

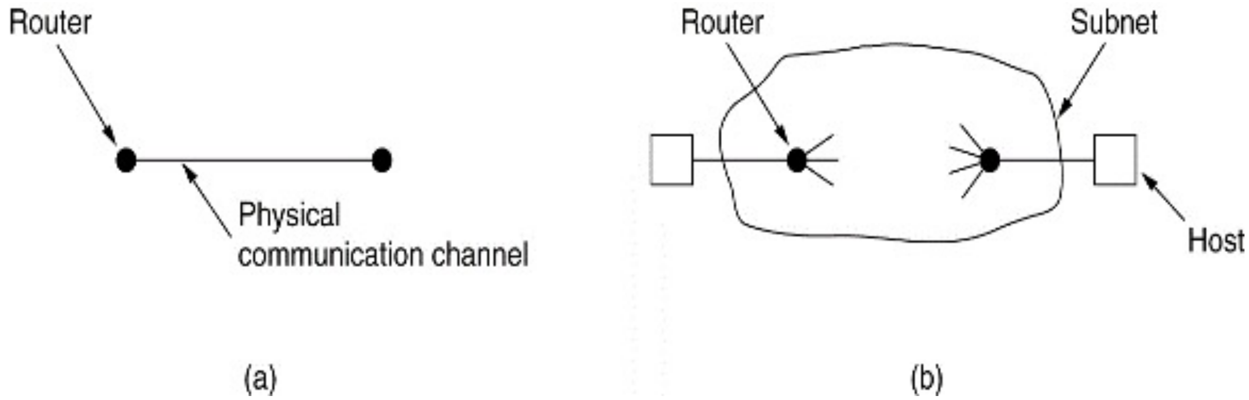
Transport service is implemented
- by transport protocol used
  between the two transport entities

Several issues have to be addressed such as:
- Addressing
- Connection Establishment
- Connection Release
- Flow Control and Buffering
- Multiplexing
- Crash Recovery

Elements of Transport Protocols

Transport protocols resemble somehow data link protocols:
- but significant differences exist

Transport and data link protocols operate in different environments!



Addressing:
- DL: outgoing line of a router specifies particular router
- TL: explicit addressing of destinations is required

Connection establishment:
- DL: peer is always there
- TL: reachability might be unclear

Storage capacity in the subnet:
- DL: negligible
- TL: serious problem, packet may be stored somewhere and delivered later

Buffering and flow control:
- DL: few outgoing lines allow simple buffer allocation schemes
- TL: potentially large, dynamically #conn. requires flexible schemes

3 Addressing (at Transport Layer)

Why identification?

- sender (process) wants to address receiver (process)
  - for connection setup or individual message
- receiver (process) can be approached by the sender (process)

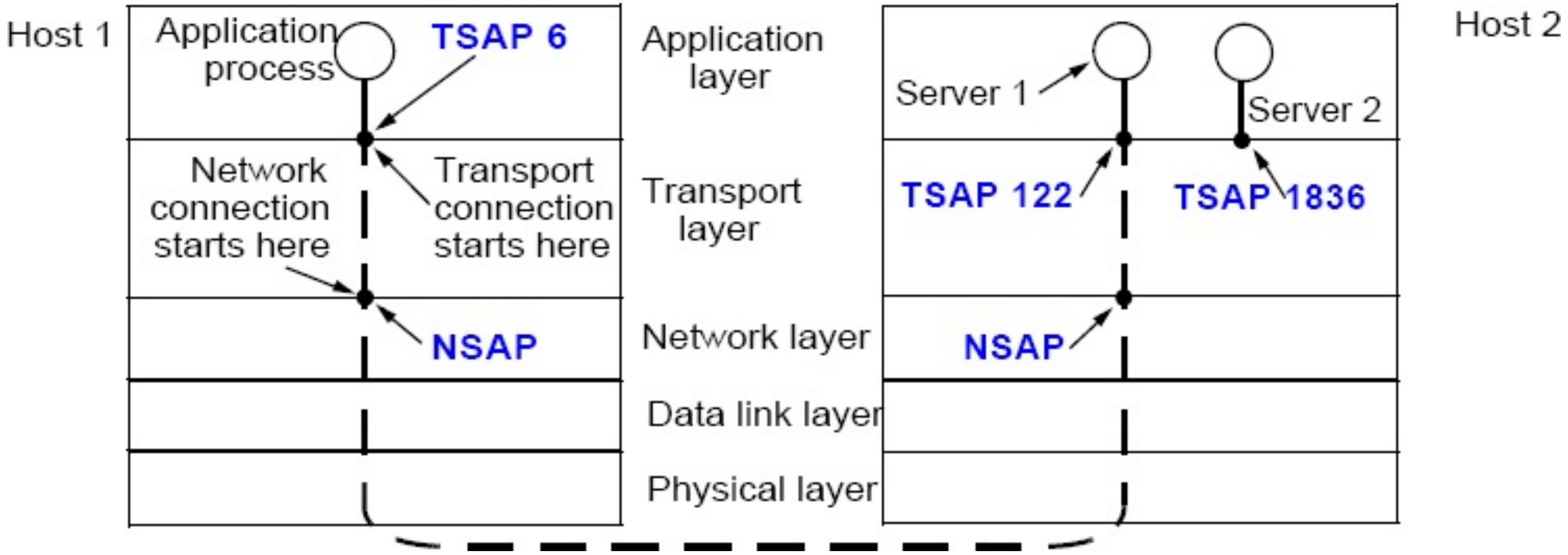
Define transport addresses:

- generic term: (Transport) Service Access Point TSAP
- Internet: port

Reminder: analogous end points in network layer: NSAP

- e.g., IP addresses

Model:

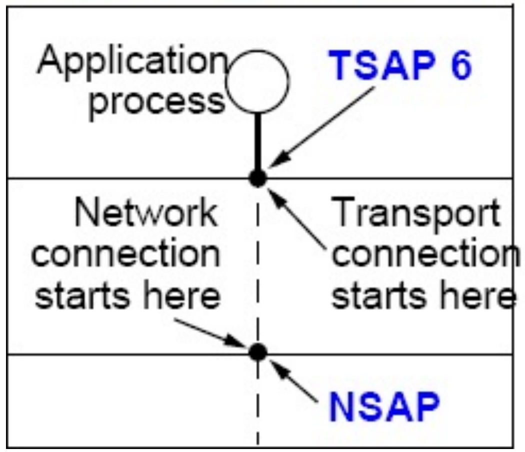


Steps

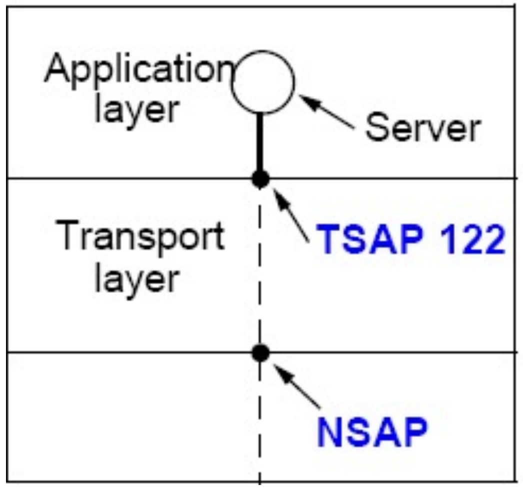
In general

1. Server (service provider)
   - connects itself to TSAP 122
   - waits for service request (polling, signalling, ..)
2. Client (application)
   - initiates connection via TSAP 6 as source and TSAP 122 as destination
     - i.e. connect.req
3. Transport system on host 1
   - identifies dedicated NSAP
   - initiates communication at network layer
   - communicates with transport entity on host2
   - informs TSAP 122 about desired connection
4. Transport entity on host 2
   - addresses the server
   - requests acceptance for the desired connection
     - i.e. connect.ind
5. etc.

Host 1: Sender - Client

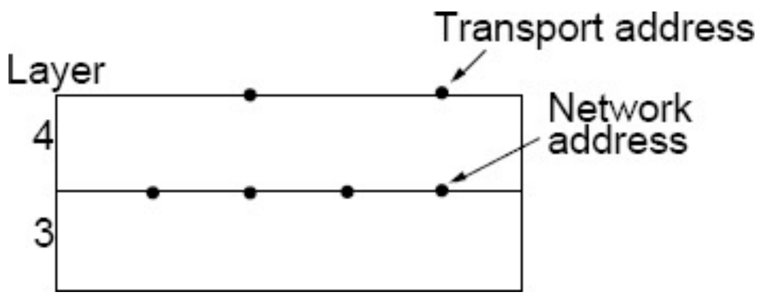


Host 2: Receiver - Server



Determination of Service Provider TSAP

How does the specific address of a service becomes known?



1. Approach: TSAP known implicitly
    - services that are well known and often used have pre-defined TSAPs
        - as "well known ports" of a transport protocol
    - e.g., stored in /etc/services file at UNIX systems

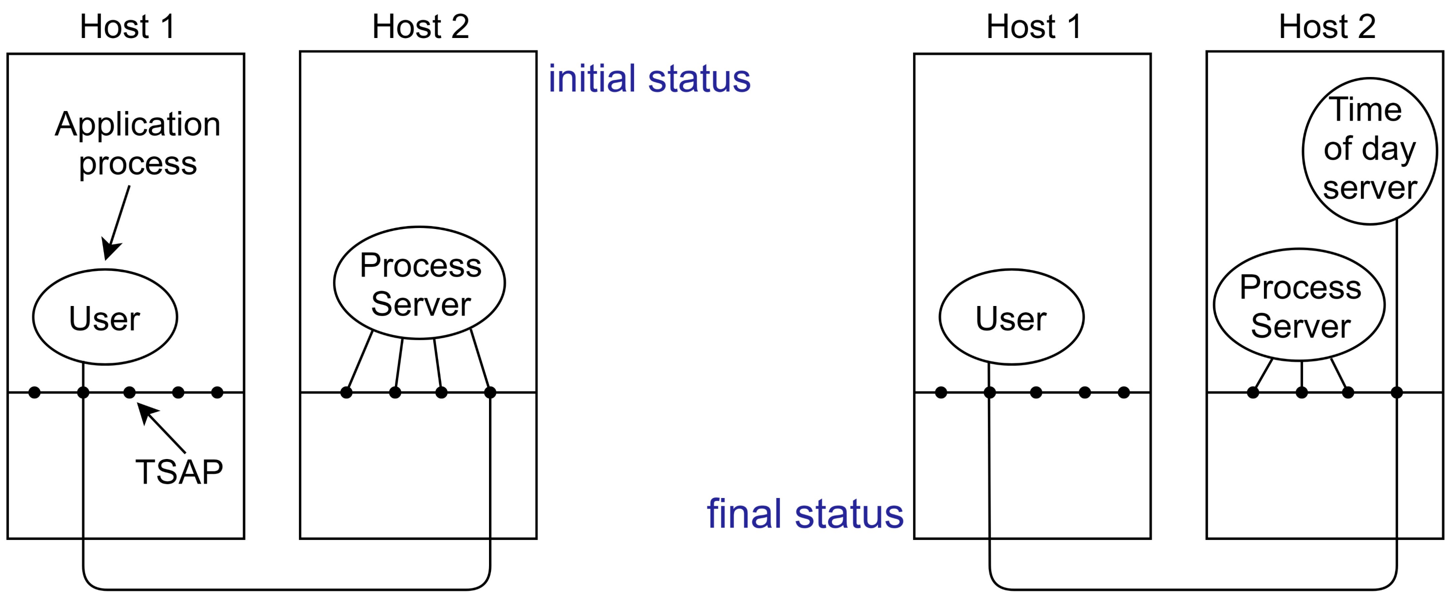
example: service ’time of day’ (port 13)

Characteristics:
- works well for small number of stable services
- not suitable for user specific processes
    - existing for short time, no known TSAP addr
- waste of resources to have seldomly used servers active and listening

Determination of Service Provider TSAP

2. Approach: “initial connection protocol”
    1. process server acting as proxy for less often used servers
    2. process server listens to set of ports at same time, waiting for connection requests
    3. creates the appropriate service provider process
    4. transfers connection and desired service
    5. waits for further requests

Characteristics:
- works well for servers which can be created on demand
- not suitable if service exists independently of process server at another machine (e.g., file server)



Determination of Service Provider TSAP

3. Approach: Name Server (directory server)

- context
  - server process already exists
- procedure
  1. client addresses Name server (establishing connection)
  2. client specifies the service as an ASCII data set
     - example “name of day”
  3. name server supplies TSAP
  4. client disconnects from name server
  5. client addresses TSAP provided by name server

     .....

- comments
  - new services
    - have to register at the name server
  - name server
    - adds corresponding information at the database

4 Duplicates

Initial (problematic) situation:

- network has
  - varying transit times for packets
  - certain loss rate
  - storage capabilities
- packets can be
  - manipulated
  - duplicated
  - resent by the original system after timeout

In the following, uniform term: "Duplicate"

- a duplicate originates due to one of the above mentioned reasons and
- is at a later (undesired) point in time passed to the receiver

Example: Duplicates

Example of problems caused by duplicates:

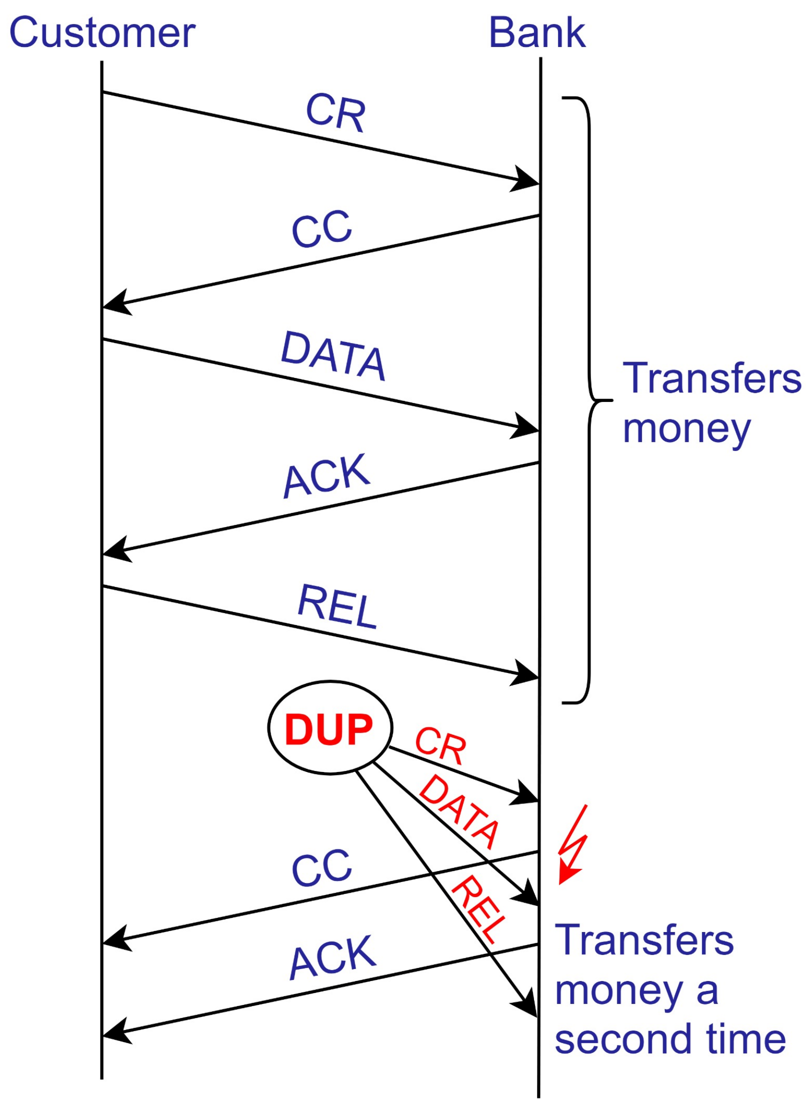
- Let's assume duplicates occur in the network
  - duplication of all sender's packets
  - subsequent to the first 5 packets, duplicates are transferred in correct order to the receiver

  - also conceivable is that an old delayed DATA packet (with faulty contents) from a previous session may appear; this packet might be processed instead of or even in addition to the correct packet

Result:

- without additional means the receiver cannot differentiate between correct data and duplicated data

→ Receiver would re-execute the transaction !



CR: ConnectReq
CC: ConnectConf
REL: Release

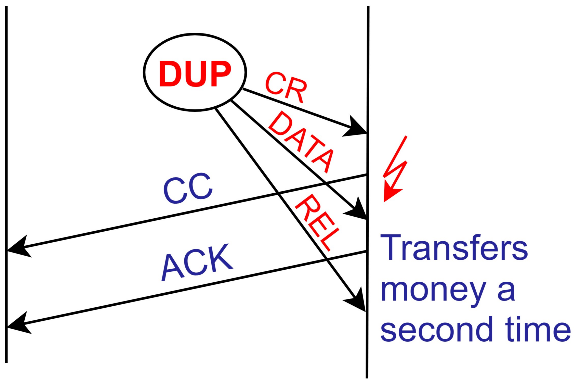
Duplicates – Problematic Issues

3 somehow disjoint problems

1. How to handle duplicates **within a connection**?

2. Which characteristics have to be taken into account regarding
   - **consecutive connections** or
   - connections which are being **re-established** after a crash?

3. What can be done to ensure that a connection that has been established:
   - has actually been initiated by and **with the knowledge of both communicating parties?**
     - see also lower part of previous illustration



Duplicates – Problematic Issues

3 somehow disjoint problems

1. How to handle duplicates **within a connection**?

   → use consecutive sequential numbers from sufficiently large sequential number range
   - TCP uses 32 bit sequence numbers
   - 32 bit was large in 1980ies (link rates were low), but is small today
   - sequence number range exploitation
     - 10 Mbit/sec ~ 3400 sec (~57 Min)
     - 10 Gbit/sec ~ 3,4 sec

2. Which characteristics have to be taken into account regarding
   - **consecutive connections** or
   - connections which are being **re-established** after a crash?

   → Choose initial sequential number wisely, consider max. message lifetime (MSL)

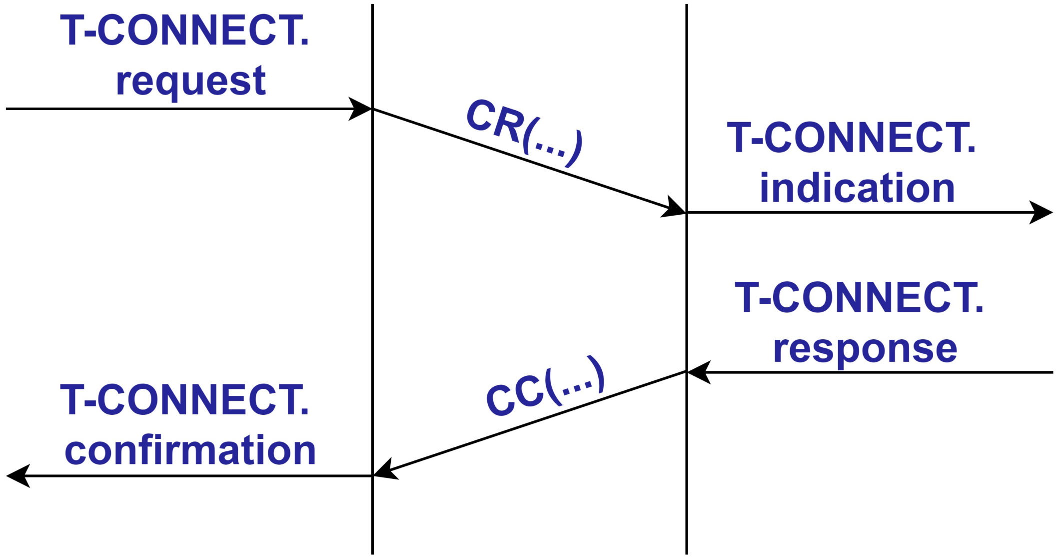
3. What can be done to ensure that a connection that has been established:
   - has actually been initiated by and **with the knowledge of both communicating parties?**

   → Well prepared connection setup: three-way handshake

5 Reliable Connection Establishment

Simple idea for connection establishment

- Use two messages



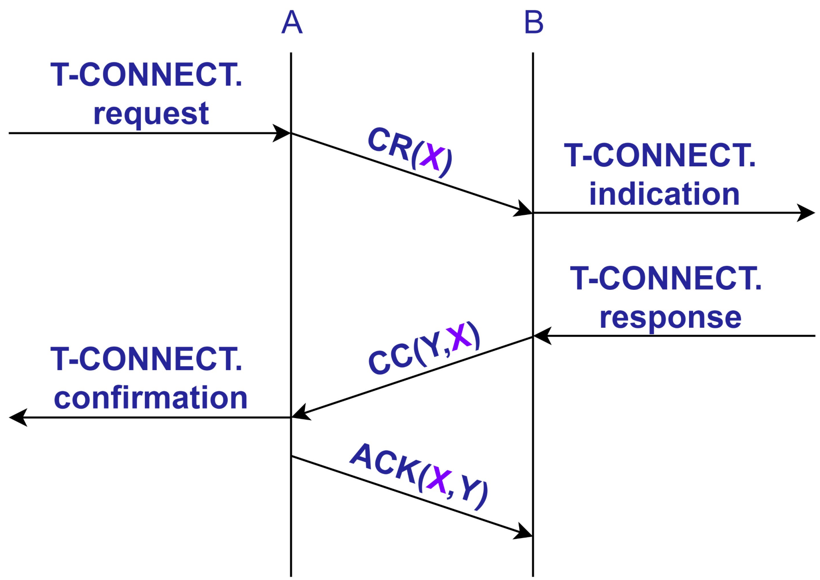
CR(REF): Connection Req.
CC(REF): Connection Conf.

- approach using 2 messages (2 phases) is **too simplistic**
  - problems may occur due to delayed duplicates
  - compare with previous example (bank transaction)

Connect: Three-way Handshake Protocol

Principle

1. CR: Connect Request
   - initiator (A) sends request with
     - SequenceNo (**X**) selected by sender
2. CC: Connect Confirmation
   - receiver (B) responds with
     - sequence number transmitted by the initiator (**X**) and
     - (randomly) selected sequence number (**Y**) by receiver



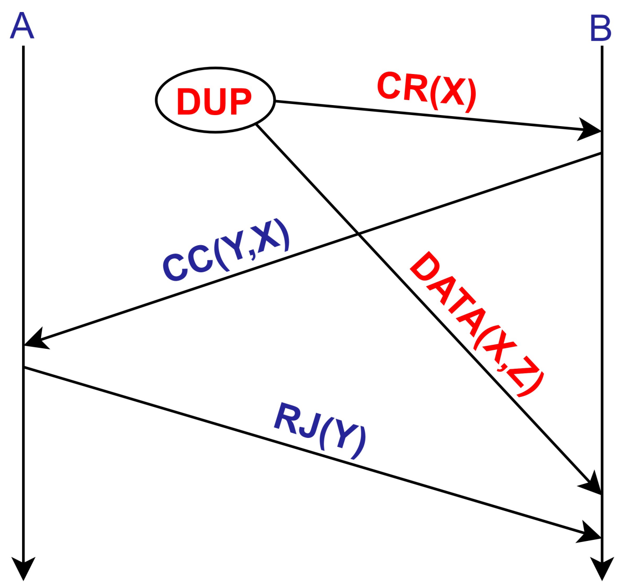
CR(Source-Ref)
CC(Source-Ref, Destination-Ref)
ACK(Source-Ref, Destination-Ref)

3. Acknowledgment
   - initiator (A) acknowledges
     - sequence numbers **X**, **Y** (as received before)
   - only after receiving a valid ACK, receiver (B) accepts data

Note:

- some protocols (including TCP) acknowledge the next byte expected
  - (ACK X+1,Y+1), not the last byte received

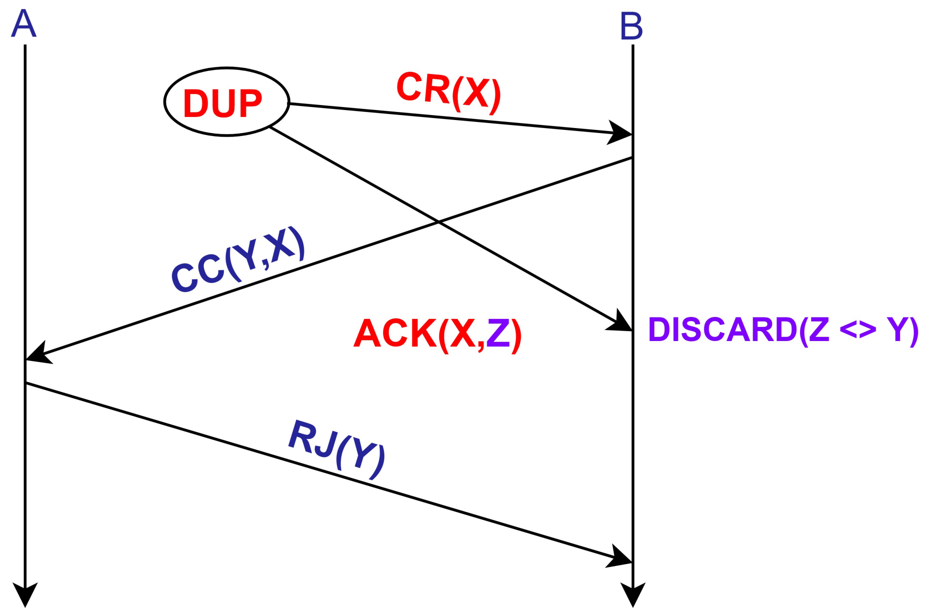
Three Way Handshake Protocol: Results



CR and data duplicate

- duplicate data is discarded

Three Way Handshake Protocol: Results



Connect Request (CR) Duplicate
and Acknowledgment (ACK) Duplicate

- ACK (X,Z) discarded because
  - ACK (X, Y) expected
  - ACK (X, Z) received,
    - Y != Z must be ensured by B under the premise of a maximum packet lifetime by selecting the initial sequence number accordingly

6 Disconnect

Two variants:

- asymmetric disconnect
- symmetric disconnect

Variant: symmetric disconnect

- disconnect takes place separately for each direction

Variant: asymmetric disconnect

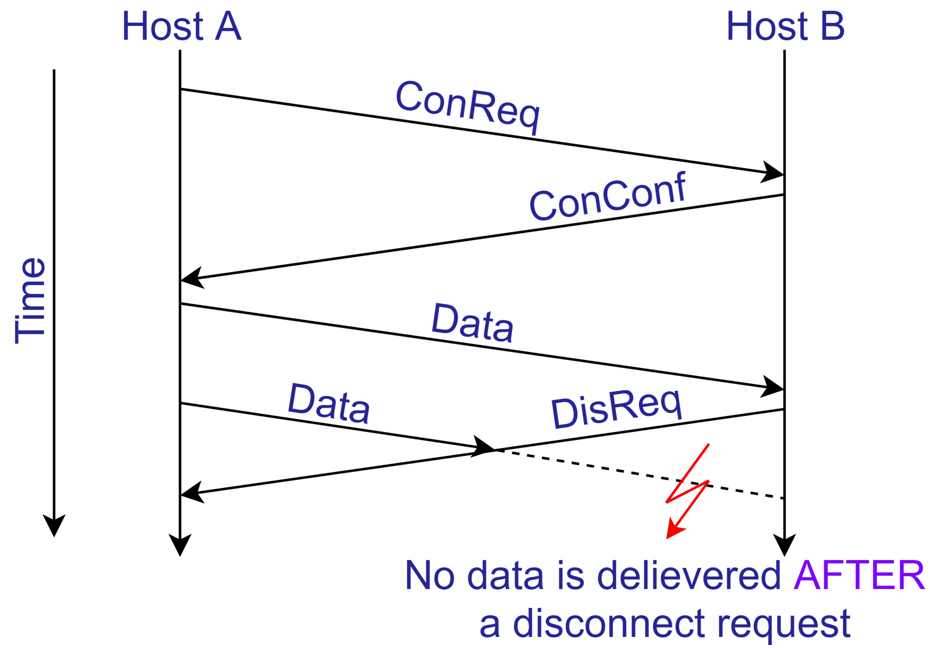
- disconnect in one direction implies disconnect for both "directions"

6.1 Asymmetric Disconnect

Approach

- disconnect in one direction implies disconnect for both "directions"
  - analog to telephone
- may result in data losses

Example



➔ approach for a solution:

- 3 phase-handshake-protocol
  - to implement a disconnect like a connect

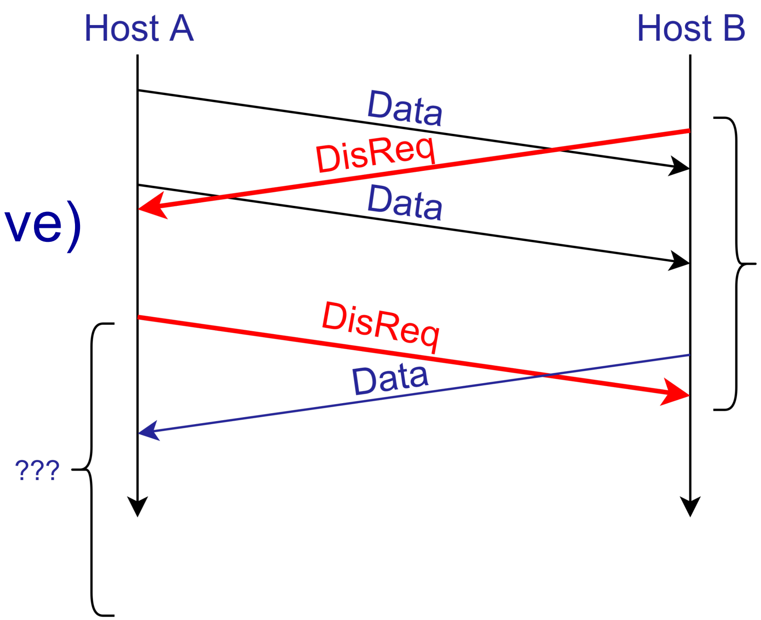
6.2 Symmetric Disconnect

Idea:
avoid data loss incurred by asymmetric disconnect
by using symmetric disconnect

- host can continue to receive data even after it has sent DISCONNECT TPDU
  - both sides have to issue a disconnect
  - host received DISC. → stops to send data
  - host sent DISC. → may continue to receive data

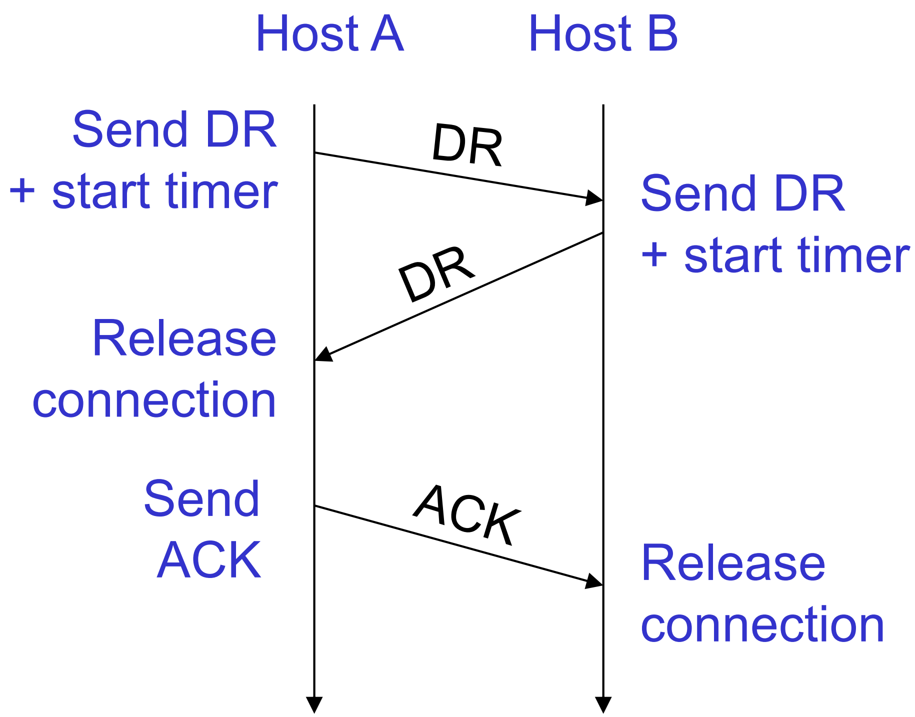
Properties

- works when each process has fixed amount of data to send and knows when it has sent it (and how much data will arrive)
- not as obvious as considered if something can go wrong
  - if host does not know about data to be received after having sent a disconnect



Disconnect with Three-Way Handshake

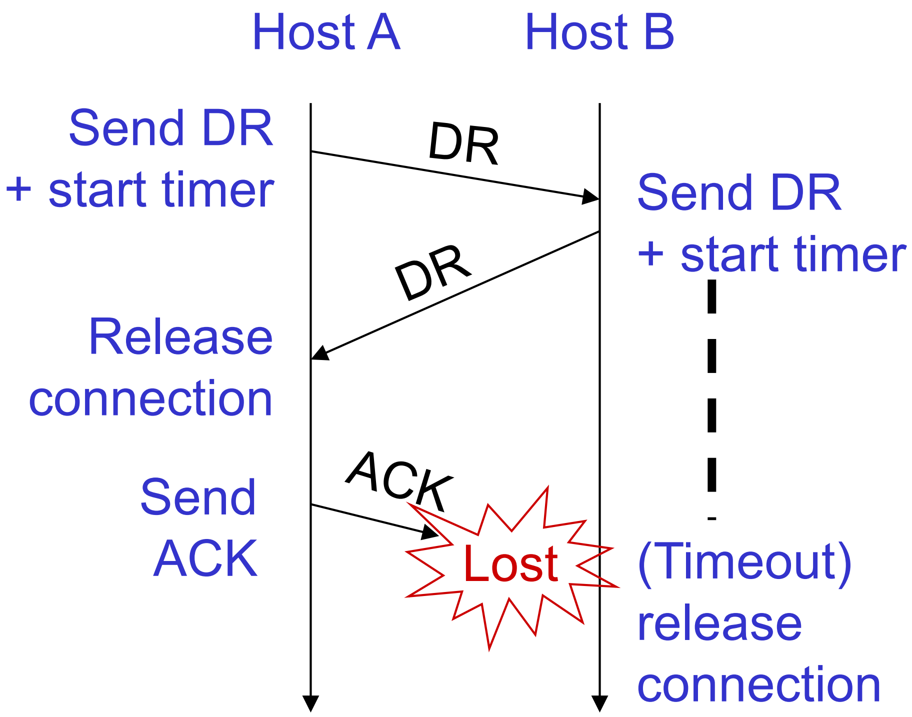
Regular disconnect with Three-Way handshake



Disconnect with Three-Way Handshake

Last acknowledgment lost

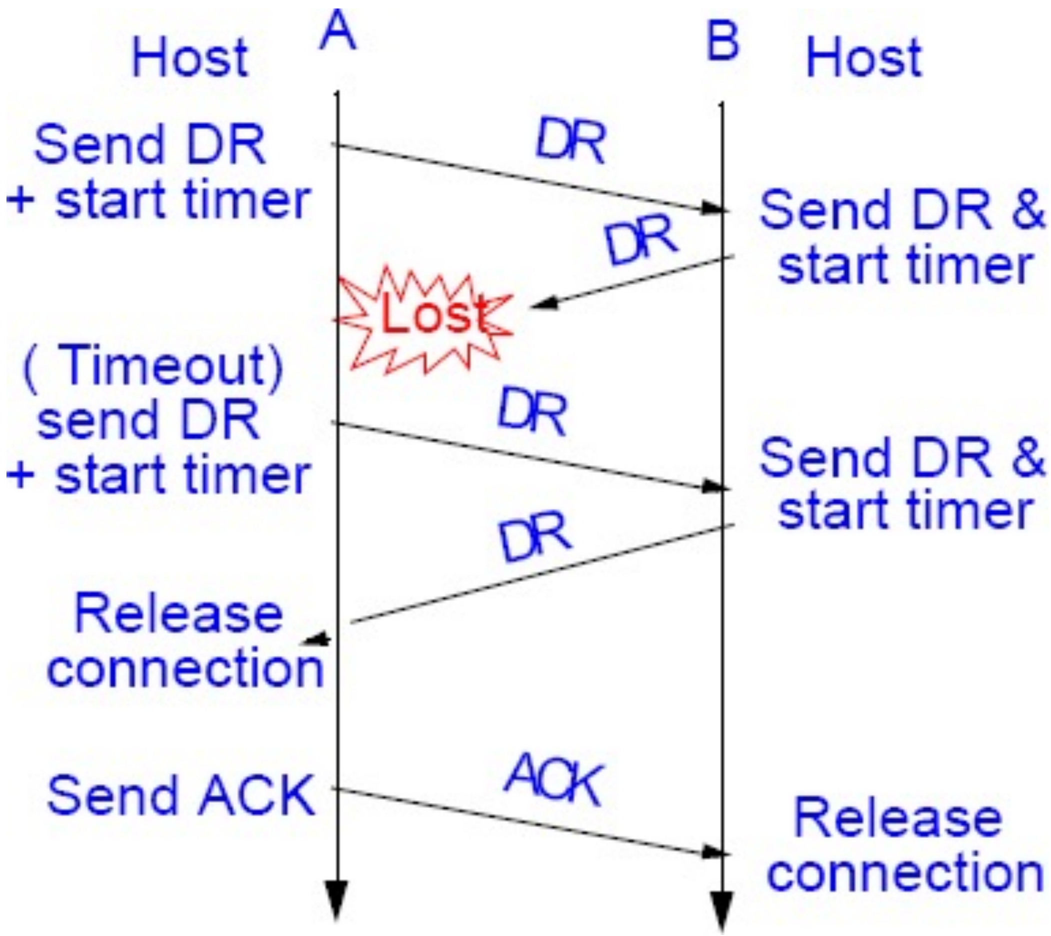
- timer disconnects from host 2
- therefore no further problems



Disconnect with Three-Way Handshake

Second “disconnect request” lost

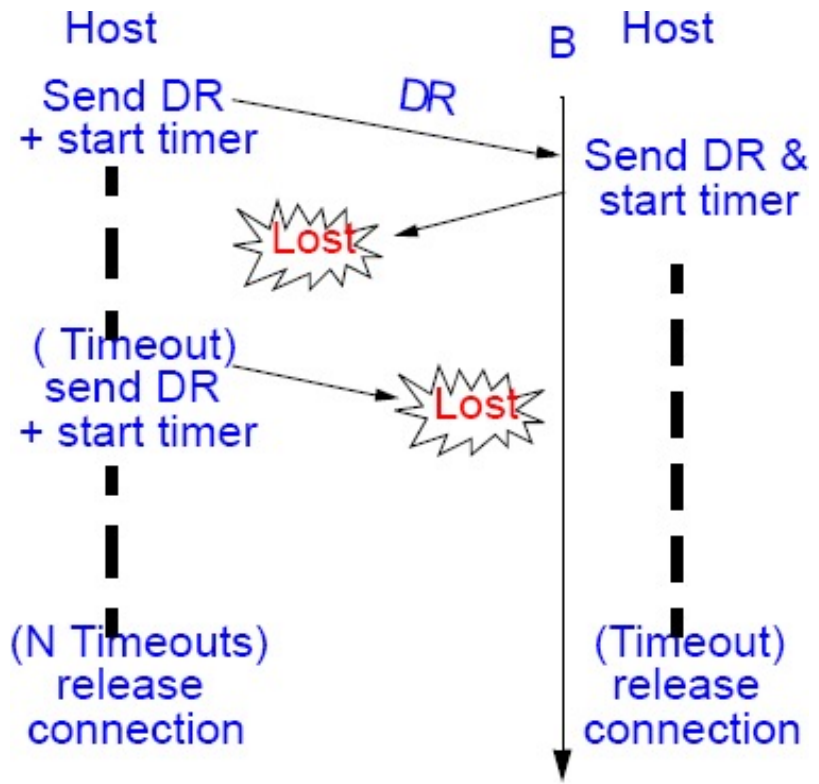
- repeat to send “disconnect request”
  - because response was an unexpected DR and ACK
- loss is repaired
  - otherwise procedure as described using timer



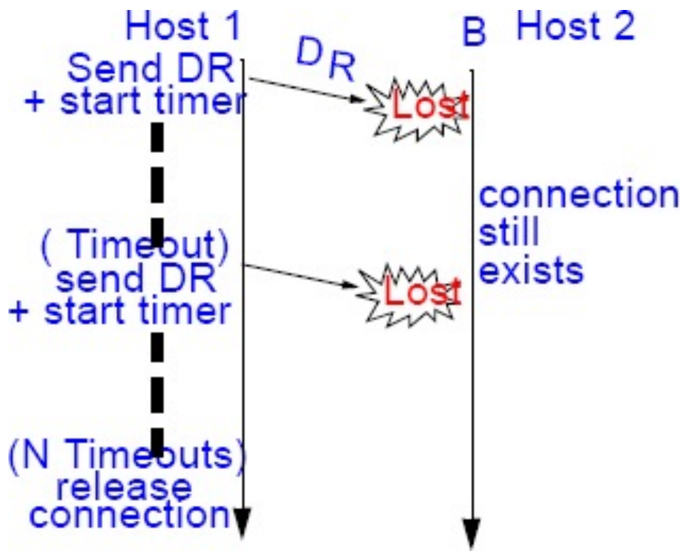
Disconnect with Three-Way Handshake

Second and all further "disconnect requests" lost

- disconnects by timeout



Disconnect with Three-Way Handshake



All "disconnect requests" lost

- resulting problem:
  - Host1 disconnects,
    but Host2 retains inconsistent information: "half-open" connections
- prevented by activity strategy
  - TPDUs have to arrive within a certain time
    - otherwise automatic disconnect
  - implementation
    - after TPDU has been sent: re-initiate timer
    - when timeout before data has been sent: send "Dummy-TPDU" to retain connection
      ("keep-alive" packets without actual data)
      (but these might be lost as well ...)

7 Flow Control on Transport Layer

Joint characteristics (with flow control on data link layer)

- fast sender shall not flood slow receiver
- sender has to store all unacknowledged packets

Differences (to flow control on data link layer)

- L2-DLL: router serves few “bitpipes”
- L4-TL: endsystem contains a multitude of
  - connections
  - data transfer sequences
- L4-TL: receiver may store packets (but does not always have to)

Strategies

1. sliding window / static buffer allocation
2. sliding window / no buffer allocation
3. credit mechanism / dynamic buffer allocation

7.1 Sliding Window / Static Buffer Allocation

Flow control

- sliding window - mechanism with window size w

Buffer reservation

- receiver reserves 2*w buffers per duplex connection

Characteristics

+ receiver can accept all PDUs
- buffer requirement may be very high
  - proportional to #transport-connections
- poor buffer utilization for low throughput connections

i.e.

➔ good for traffic with high throughput

- (e.g. data transfer)

➔ poor for intermittent (and potentially bursty) traffic with low throughput

- (e.g. interactive applications)

7.2 Sliding Window / No Buffer Allocation

Flow control
- sliding window (or no flow control)

Buffer reservation
- receivers do not reserve buffers
- buffer allocation upon arrival of TPDU
- TPDU will be discarded if there are no buffers available
- sender maintains TPDU buffer until ACK arrives from receiver

Characteristics

\+ optimized memory utilization

\- possibly high rate of discarded TPDUs during high traffic load

i.e.

➔ good for intermittent (and bursty) traffic with low throughput

➔ poor for traffic with high throughput

7.3 Credit Mechanism

Flow control

- credit mechanism

Buffer reservation

- receiver allocates buffers dynamically for the connections
- allocation depends on the actual situation

Principle

- sender requests required buffer amount
- receiver reserves as many buffers as the actual situation permits
- receiver returns ACKs and buffer-credits separately
  - ACK: confirmation only (does not imply buffer release)
  - CREDIT: buffer allocation
- sender will be blocked, when all credits have been used up

Credit Mechanism

Comments		A	Message	B	Comments (buffers located at B)
A wants 8 buffers	1	→	< request 8 buffers>		
	2		<ack = 15, cred = 4>	←	B grants messages 0-3 only
A has 3 buffers left now	3	→	<seq = 0, data = m0>		
A has 2 buffers left now	4	→	<seq = 1, data = m1>		
Message lost but A thinks it has 1 left	5	→	<seq = 2, data = m2>	···	Message lost
	6		<ack = 1, cred = 3>	←	B acknowledges 0 and 1, permits 2-4
A has buffer left	7	→	<seq = 3, data = m3>		
A has 0 buffers left, and must stop	8	→	<seq = 4, data = m4>		
A times out and retransmits	9	→	<seq = 2, data = m2>		
but A still blocked	10		<ack = 4, cred = 0>	←	Everything acknowledged, A still blocked
A may now send next msg.	11		<ack = 4, cred = 1>	←	
	12		<ack = 4, cred = 2>	←	B found a new buffer somewhere
A has 1 buffer left	13	→	<seq = 5, data = m5>		A has 1 buffer left
A is now blocked again	14	→	<seq = 6, data = m6>		A is now blocked again
A is still blocked	15		<ack = 6, cred = 0>	←	A is still blocked
	16	···	<ack = 6, cred = 4>	←	

Example: with dynamic buffer allocation

- 4 bit SeqNo (0..15) and "..." corresponds to data loss

Dynamic adjustment to

- buffer situation
- number of open connections
- type of connections
  - high throughput: many buffers
  - low throughput: few buffers