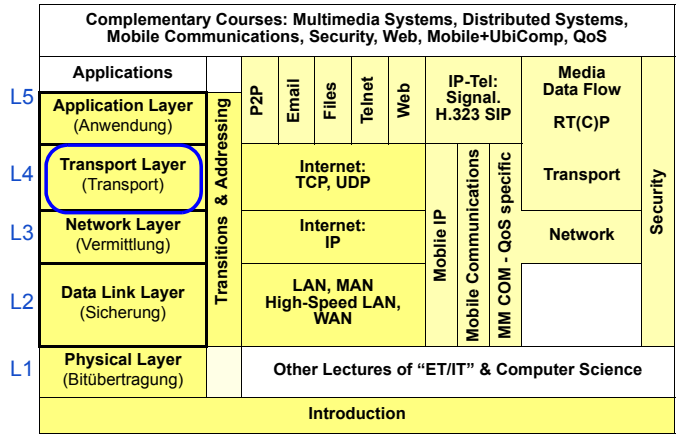
Communication Systems
Transport Layer

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

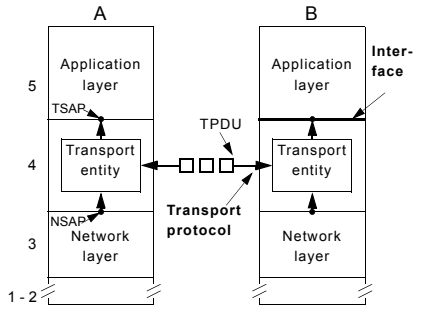
1. Transport Layer Function

To 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 used**

1.1 Transport Service



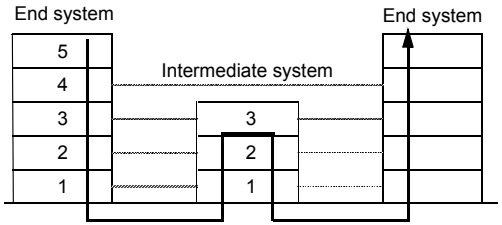
Connection-oriented service
- **3 phases: connection set-up, data transfer, disconnect**

Connectionless service
- **transfer of isolated units**

Realization: transport entity
- **software and/or hardware**
- **software part usually located in the kernel (process, library)**

Transport Service (2)

Similar services of
- **network layer and transport layer:**
  WHY 2 LAYERS?



Network service
- not to be self-governed or influenced by the user
- independent from application & user
  - enables compatibility between applications
- provides, for example
  - "only" connection-oriented communications
  - or "only" unreliable data transfer

Transport service: TO IMPROVE THE NETWORK SERVICE QUALITY
- users and layers want to get from the network layer
  - e.g. reliable service
  - necessary time guarantees

Transport Service (3)

Transport layer:
- **isolates upper layers from technology, design and imperfections of subnet**

Traditonally distinction made between
- **layers 1 - 4**
  - transport service provider
- **layers above 4**
  - transport service user

Transport layer has key role:
- **major boundary between provider and user of reliable data transmission service**

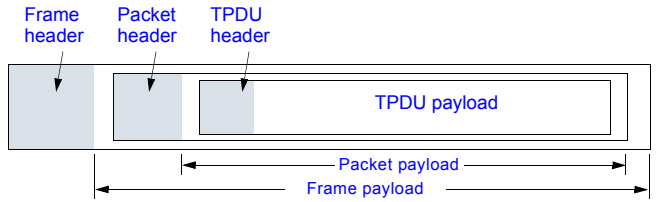
Transport Service: Terminology

Entities exchanged:

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

`TPDU: Transport Protocol Data Unit`

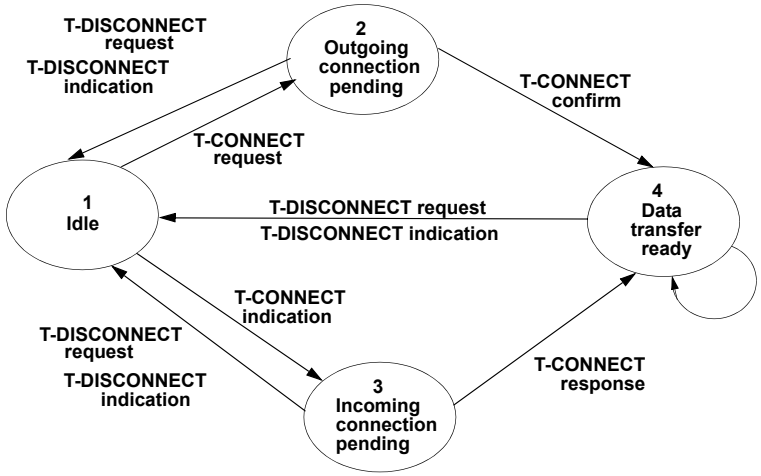
Nesting of TPDUs, packets, and frames:



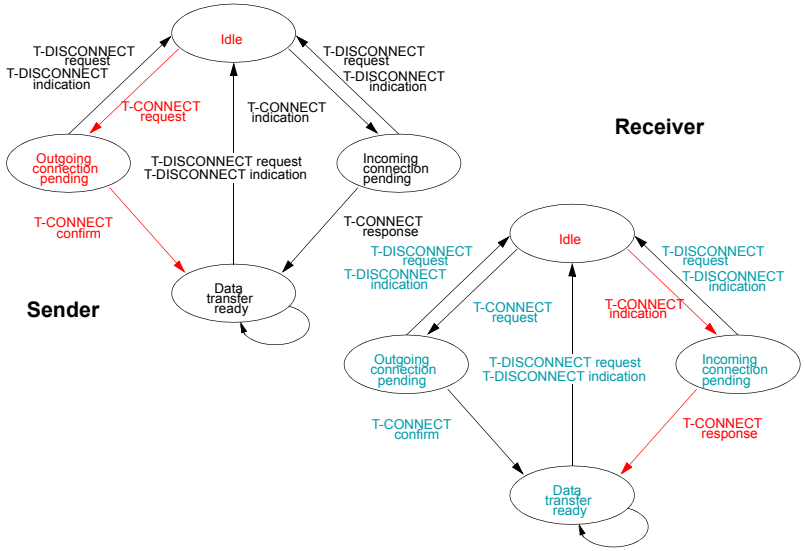
1.2 Connection-oriented Service: State Transition Diagram

Example: ISO-OSI terminology

- **state transition diagram**



Connection-oriented Service: State Transition Diagram (2)



Example: Parameters for Disconnect

Reason for any "T-Disconnect"

Reason	Notes
Normal disconnect initiated by session entity	1, 4
Remote congestion at transport entity during CC	1, 4
Failed connection negotiation	1, 3
Duplicated source reference for same NSAP pairs	1, 4
References are mismatched	1, 4
Protocol error	1, 4
Reference overflow	1, 4
Connection request refused	1, 4
Header or parameter length invalid	1, 4
No reason specified	2, 4
Congestion at TSAP	2, 4
TSAP and session entity not attached	2, 3
Unknown address	2, 3
(Note 1) Used for classes 1 to 4	
(Note 2) Used for all classes	
(Note 3) Reported to TS-user as persistent	
(Note 4) Reported to TS-user as transient	

1.3 Transport Service Primitives: More Practical

Primitives for a simple transport service:

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ	Actively attempt to establish a connection
SEND	DATA	Send Information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ	Request to release the connection

Transport Service Primitives: More Practical (2)

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; give queue size
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

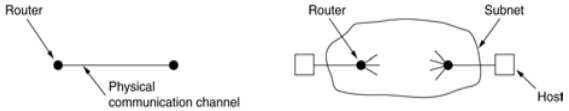
Transport service is implemented
- **by TRANSPORT PROTOCOL used between the two TRANSPORT ENTITIES**

Some issues:
- **Addressing**
- **Connection Establishment**
- **Connection Release**
- **Flow Control and Buffering**
- **Multiplexing**
- **Crash Recovery**

Elements of Transport Protocols (2)

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: more different situations

Storage capacity in the subnet:
- DL: neglectable
- 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 #connections requires flexible schemes

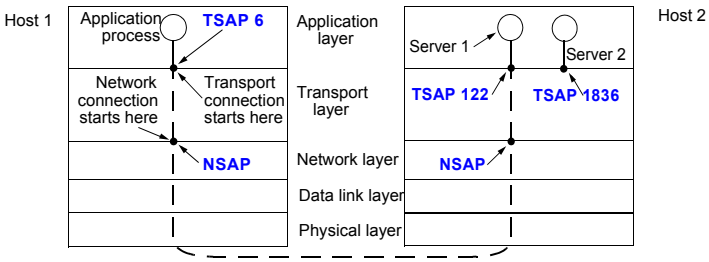
3. Addressing

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**
- **or e.g. ATM:** **AAL-SAP**

Reminder: analogous end points in network layer: NSAP
- **e.g., IP addresses**

Model

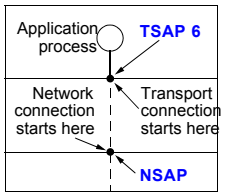


Steps

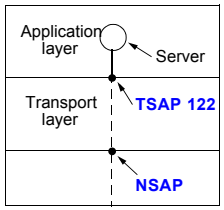
In general

1. **Server (service provider)**
   - connects itself to TSAP 122
   - waits for service request (polling, signalling, ..)
2. **Client (application)**
   - initiates connection with 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

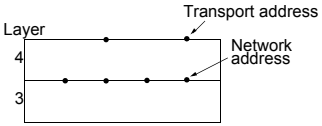


Host 2: Receiver - Server



3.1 Determination of Appropriate Service Provider TSAP

How does the specific address of a service becomes known?



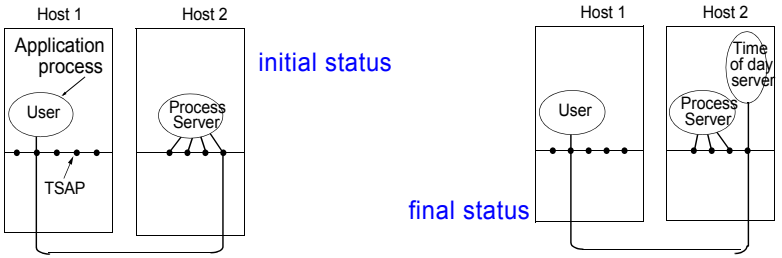
1. **Approach: TSAP known implicitly**
   - services that are well known and often used have pre-defined TSAPs (as "well known ports" of a transport protocol)
   - e.g., stored in /etc/services file on UNIX systems

example: service 'time of day'

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

Determination of Appropriate Service Provider TSAP (2)

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 (e.g., file server)**

Determination of Appropriate Service Provider TSAP (3)

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
  5. client establishes new connection with desired service (retrieved TSAP)

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

3.2 Determination of Appropriate NSAP

How to localize the respective end system NSAP?

- **TSAP known**
- **i.e., how to determine the appropriate NSAP**

1. approach: hierarchic addressing

- **TSAP contains this information**

```
example: <country>.<network>.<port>
```

2. approach: "flat" addressing

- **dedicated "name server"**
  - entry

```
TSAP address: address of the endsystem
```

- **request via broadcast**
  - e.g., as correlation of
    - ethernet address and internet address
  - i.e., possible in geographically and topologically limited spaces

4. Duplicates

Initial Situation: Problem

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

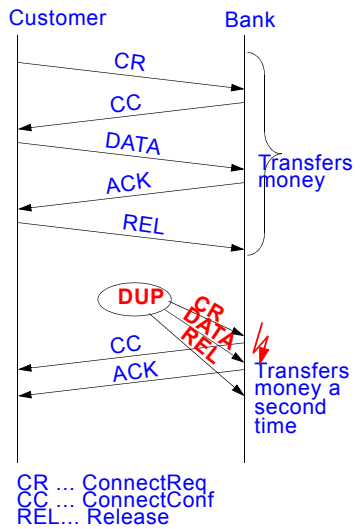
In the following, uniform term: "DUPLICATE"

- **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 scenario to describe possible error reasons and their potential consequences

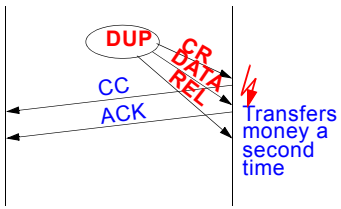
- **transaction realized using 5 packets**
- **due to network characteristics**
  - duplication of sender's packets
  - subsequent to the first 5 packets, duplicates are transferred in correct order to the receiver
  - also conceivable is that an old delayed DATA packet (with faulty contents) from a 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
  - would re-execute the transaction



Duplicates: Description of Problematic Issues

3 somehow disjoint problems

- **how to handle duplicates WITHIN A CONNECTION?**
- **what characteristics have to be taken into account regarding**
  - CONSECUTIVE CONNECTIONS or
  - CONNECTIONS which are being re-established after a crash?
- **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 the lower part of the previous illustration



Duplicates: Methods of Resolution

1. Using temporarily valid TSAPs
- **method:**
  - TSAP valid for one connection only
  - generate always new TSAP
- **evaluation**
  - process server addressing method not possible, because
    - server is reached via a designated TSAP
    - some TSAPs always exist as "well known"

2. Identify connections individually
- **method**
  - each individual connection is assigned a new SeqNo and
  - end systems remember already assigned SeqNo
- **evaluation**
  - end systems must be capable to store this information
  - prerequisite: connection-oriented system (what if connection-less?)
  - end systems, however, will be switched off and it is necessary that the information is reliably available whenever needed

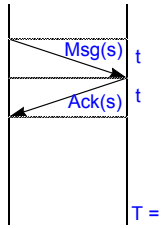
Duplicates: Methods of Resolution (2)

3. Individual sequence number for each PDU
- **method**
  - SeqNo basically never gets reset
  - e.g., 48 bit at 1000 msg/sec: reiteration after 8000 years
- **evaluation**
  - higher usage of bandwidth and memory
  - sensible choice of the sequence number range depends on
    - the packet rate
    - a packet's probable "lifetime" within the network
- **discussed in more detail in the following**

4.1 Duplicates: Approach to Limit Packet Lifetime

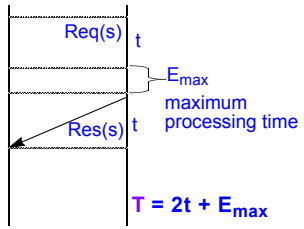
Enabling the above Method '3. Individual sequence number for each PDU'
- **SeqNo only reissued if**
  - all PDUs with this SeqNo or references to this SeqNo are extinct
- **i.e., ACK (N-ACK) have to be included**
  - otherwise new PDU may be wrongfully confirmed or non-confirmed by delayed ACK (N-ACK).

example 1 (in principle)



Mandatory PREREQUISITE for this solution
- **limited packet lifetime**
- **i.e. introduction of a respective parameter T**

example 2:Request/Response
taking processing time into account)



Limiting Packet Lifetime: Techniques

1. Limitation by appropriate network design
- inhibit loops
- limitation of delays in subsystems & adjacent systems

2. Hop-counter / time-to-live in each packet
- counts traversed systems
- if counter exceeds maximum value => packet is discarded

3. Timestamp each packet
- packet exceeds maximum configurable lifetime
  => packet is discarded
- **NOTICE:** requires "consistent" network time (synchronized clocks)

Limiting Packet Lifetime: Techniques (2)

Determining maximum time T which a packet may remain in the network

- T is a small multiple of the packet lifetime
- T time units after sending a packet
  - the packet itself is no longer valid
  - all of its (N)ACKs are no longer valid

TCP/IP term: Maximum Segment Lifetime (MSL)

- to be imposed by IP layer
- defined by and referenced by other protocol specifications
  - 2 minutes

4.2 Initial Sequence Number Allocation and Handling of Consecutive Connections

Problem (wrt. "3. Individual sequence number for each PDU")

- **to be considered packets from connections which can otherwise not be distinguished**
  - hence for TCP that is:
    - same source and destination address and same source and destination port
    - this is always unique at one point in time
- **using consecutive sequence numbers from sufficiently large sequence number range**
  RESOLVES PROBLEMS WITH DUPLICATES WITHIN A SINGLE CONNECTION
  - duplicates are all the other packets with the same sequence number
    - irrelevant is the origin of packets, sequence of creation

Problems:

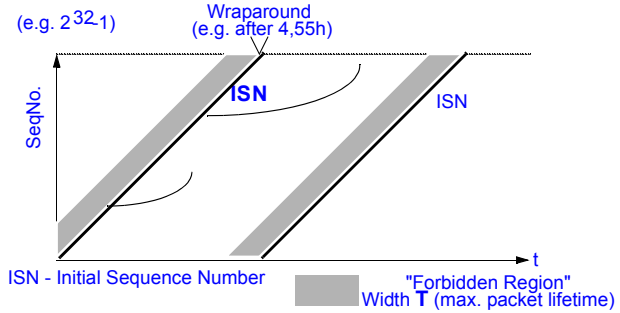
- restart after crash
- (very fast) reconnect between exactly the same communication entities
  - (addr./port see above, information about previous connections does not exist anymore after crash/ restart, generally all connections have to be reconsidered)
- complete usage of the range of available sequence numbers

Initial Sequence Number Allocation and Handling of Consecutive Connections (2)

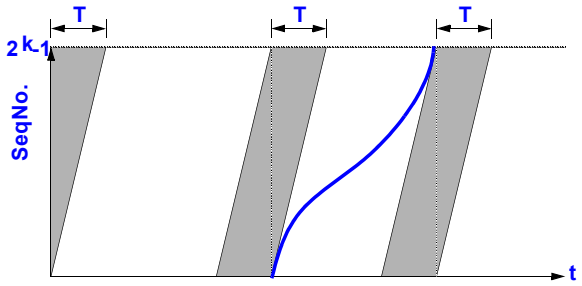
Method

- **end systems maintain timer continuously running at switch-off / system crash**
- **allocation of initial SeqNo (ISN) depends on timestamp**
  - linear curve in theory, staircase curve in reality because of discrete clock ticks
- **SeqNos can be allocated consecutively within a connection**
  - curve consisting of discrete points may have any gradient form depending on the method used for sending the data

Initial Sequence Number Allocation and Handling of Consecutive Connections (3)



Initial Sequence Number Allocation and Handling of Consecutive Connections (4)



No problem, if

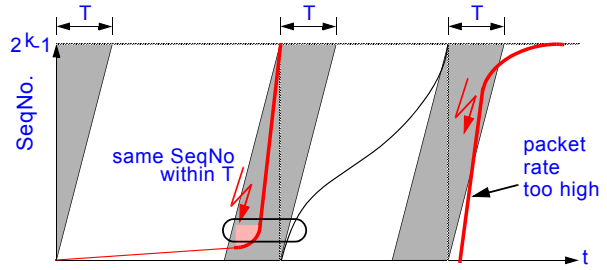
- **"short lived" session (shorter than wrap-around time) with data rate smaller than ISN rate (ascending curve less steep)**

Then, after crash reliable continuation of work always ensured

Initial Sequence Number Allocation and Handling of Consecutive Connections (5)

Problems possible, if

1. **SeqNo is used within time period T before it is being used as initial SeqNo**
   - =>"Forbidden Region" - begins T **BEFORE** Initial SeqNo (ISN) is generated
   - i.e. endsystem has to check if the PDU is in the forbidden region before it is sent (during the actual data phase)



2. **"long lived" session (longer than wrap-around time)**
3. **high data rate**
   - curve of the consecutively allocated sequence numbers steeper than ISN curve

Initial Sequence Number Allocation and Handling of Consecutive Connections (6)

Note:

- **32 bit sequence numbers considered as sufficient when designing TCP/IP (consider technology available at that time)**
- **sequence number range exploitation**
  - today at 1 Gbps
  - sequnce number space exhausted in 17 sec

⇒ **Using timestamps in**

- **"TCP extensions for highspeed paths"**
  - PAWS "Protect Against Wrapped Sequence Numbers"

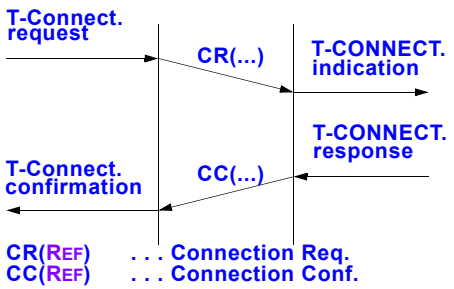
Further literature in addition to Tanenbaum

- **RFC 793 (TCP) / Sequence Numbers; "When to keep quiet"**
- **RFC 1185 / Appendix - Protection against Old Duplicates**
- **RFC 1323 / PAWS**
  - Protect Against Wrapped Sequence Numbers
  - Appendix B - Duplicates from Earlier Connection Incarnations

5. Reliable Connection Establishment

Connect (see also Connection-oriented Service: State Transition Diagram)

- **simple protocol**

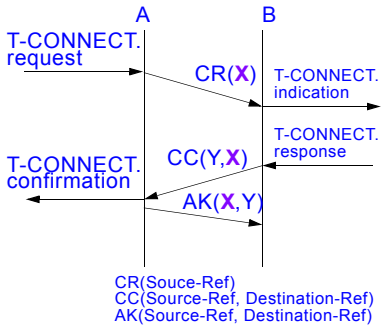


- **with approach using only 2 messages (phases) problems may occur due to delayed duplicates**
- **compare with previous example (bank transaction)**

Connect: Three Way Handshake

Principle

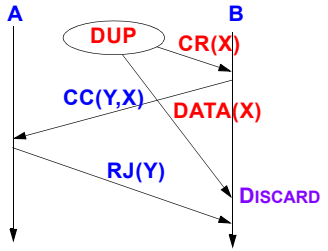
1. **CR: Connect Request**
- **initiator sends request with**
  - **SEQUENCENO (X)** selected by sender
2. **CC: Connect Confirmation**
- **participant responds with**
  - sequence number transmitted by the initiator (X) and
  - randomly selected sequence number by receiver (Y)
    - while observing the previously discussed criteria for selection, in order to avoid a collision with delayed duplicates



3. **Acknowledgment**
- **initiator acknowledges sequence number Y transmitted by the other participant**
- **only after receiving a valid ACK, participant accepts data**

Three Way Handshake Protocol: Results

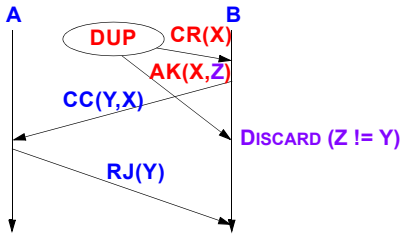
CR and data duplicate



- **duplicated data is discarded**

Three Way Handshake Protocol: Results (2)

Connect Request (CR) and Acknowledgment (AK) Duplicates



- **AK (X,Z) discarded because**
  - AK (X, Y) expected
  - **Y != Z**: will be ensured by B under the premise of a maximum packet lifetime by selecting the initial sequence number according to the described algorithms

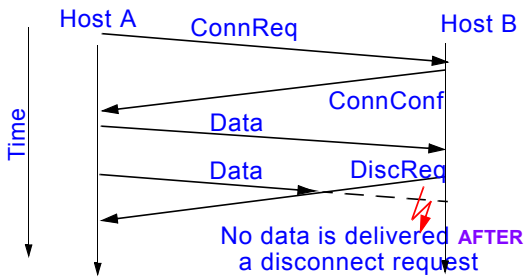
6. Disconnect

Two variants:
- **symmetric disconnect**
- **asymmetric disconnect**

Variant: symmetric disconnect
- **disconnect takes place separately for each direction**

Variant: asymmetric disconnect
- **disconnection in one direction implies disconnect for both directions**
  - analog to telephone
- **may result in data loss**

Example



Symmetric Disconnect

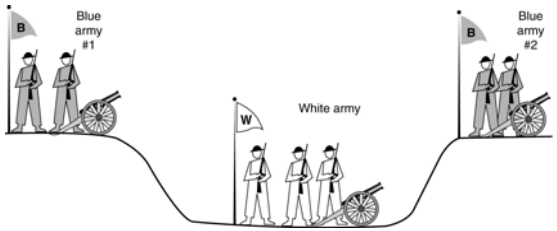
Idea: Avoid data loss incurred by asymmetric disconnect by using symmetric disconnect

- **host can continue to receive data even after it has sent DISCONNECT TPDU**

Properties

- **works when each process has fixed amount of data to send and knows when it has sent it**
- **not as obvious as considered if something can go wrong**

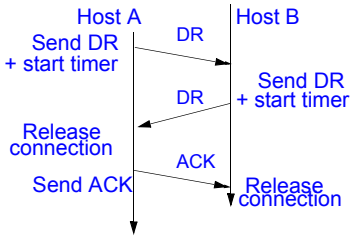
'Two-army problem':



- all messages need acknowledgements to be sure that other commander agrees
- but 'final' ACK can always be lost
- no perfect protocol exists

Disconnect with Three-Way Handshake

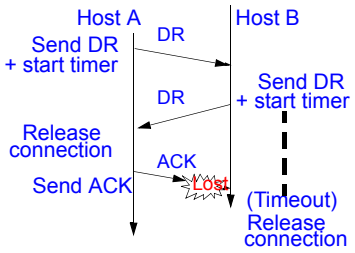
Regular disconnect with three-way handshake



Disconnect with Three-Way Handshake (2)

Last acknowledgment lost

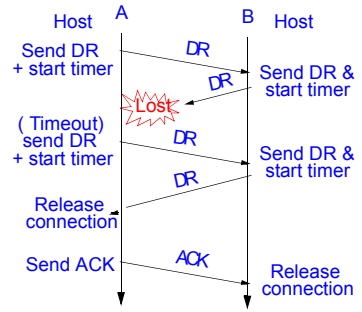
- **timer disconnects from host 2**
- **therefore no further problems**



Disconnect with Three-Way Handshake (3)

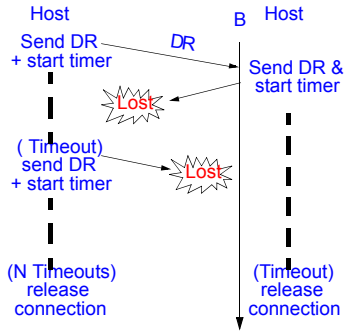
Second "disconnect request" lost

- **repeat to send "disconnect release"**
  - because response was an unexpected DR and ACK
- **loss is repaired**
  - otherwise procedure as described using timer



Disconnect with Three-Way Handshake (4)

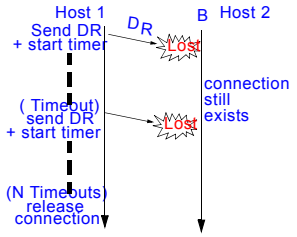
Second and all further "disconnect releases" lost



- **disconnects by timeout**

Disconnect with Three-Way Handshake (5)

All "disconnect releases" 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 in Transport Layer

Joint characteristics with flow control on data link
- **fast sender shall not overrun slow receiver**
- **sender has to store all un-acknowledged packets**

Differences (flow control on data link)
- **L2-DLL: router serves few "bitpipes"**
- **L4-TL: end system contains a multitude of**
  - connections
  - data transfer sequences
- **L4-TL: receiver may (but does not always have to) store packets**

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

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 if few connections exist

i.e.

⇒ **good for traffic with high throughput**
  - (e.g. data transfer)

⇒ **poor for intermittent traffic with low throughput**
  - (e.g. interactive applications)

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 storage utilization

- possibly high rate of discarded TPDUs during high traffic load

i.e.

⇒ good for intermittent traffic with low throughput

⇒ poor for traffic with high throughput

Credit Mechanism

Flow control
- **credit mechanism**

Buffer reservation
- **receiver associates buffers dynamically with 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

Example: with dynamic buffer allocation
- **4 bit SeqNo (0..15) and "..." corresponds to data loss**

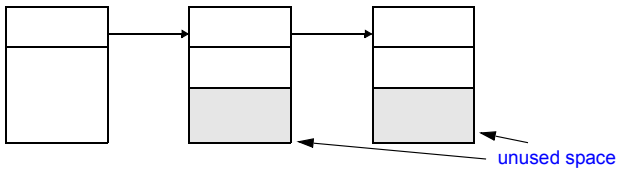
Comments		A	Message	B	Comments (buffers located at B)
A wants 8 buffers	1	→	< request 8 buffers>		
	2		<ack = 15, cred = 4>	←	B grants messages 0-3 only
A has 3 buffers left now	3	→	<seq = 0, data = m0>		
A has 2 buffers left now	4	→	<seq = 1, data = m1>		
Message lost but A thinks it has 1 left	5	→	<seq = 2, data = m2>	···	Message lost
	6		<ack = 1, cred = 3>	←	B acknowledges 0 and 1, permits 2-4
A has 1 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. 5	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 is now blocked again	14	→	<seq = 6, data = m6>		
A is still blocked	15		<ack = 6, cred = 0>	←	A is still blocked
	16	···	<ack = 6, cred = 4>	←	Potential deadlock

Dynamic adjustment to
- **buffer situation**
- **number of open connections**
- **type of connections**
  - high throughput: many buffers
  - low throughput: few buffers

8. Memory Management

Constant buffer size

Interlinked buffers per end system



+ simple because of consistent buffer size

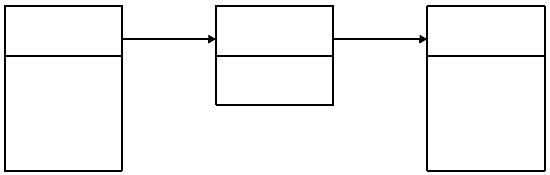
- buffer size difficult to determine
- too large: high fragmentation for small TPDUs
- too small: TPDU distributed over many buffers

i.e., increased complexity, buffer space wasted

Memory Management (2)

Variable buffer size

Interlinked buffers per end system



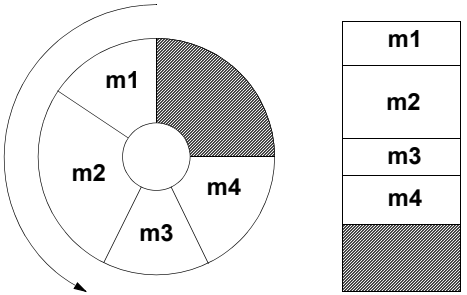
+ good storage use

- buffer management with increased complexity

⇒ more time-consuming

Memory Management (3)

Allocation of one circular buffer to each T-connection



+ good storage usage during high utilization of a connection

- poor storage usage during low utilization

Memory Management: Strategies

Strategies are traffic dependent

- **low data rate, bursty**
  - example: interactive terminal
    - => list of variable-sized buffers, dynamic allocation both on sender's and receiver's side
    - => sender should buffer
- **high constant data rate**
  - example: file transfer
  - provide sufficient buffer space
    - => list of fixed-sized buffers static allocation on the receiver's side
    - => receiver should provide more buffer space

Memory Management: Strategies (2)

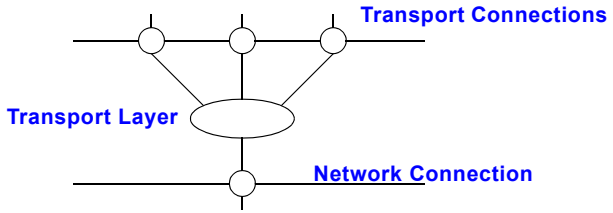
Interaction sender - receiver

- **sender**
  - knows traffic characteristics
  - should also be able to "control" buffer reservation on receiver's end
- **receiver**
  - can inform sender about reserved buffer space
  - sender can modify/control traffic

Allocation

- **per connection**
- **for all connections between two end systems**

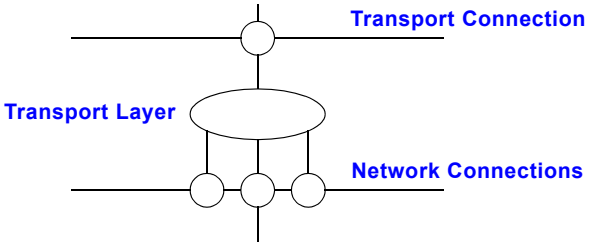
9. Multiplexing / Demultiplexing



Application
- **minimizing costs when # connections / connection time represent the main cost factors**

Multiplexing function
- **grouping of T connections by destination address**
- **each group is mapped to the minimum number of network connections**
  - too many L4- connections per L3- connection
    - ⇒ possibly poor throughput
  - too few L4 connections per L3 connection
    - ⇒ possibly transfer costs too high

Splitting / Recombining



Application:
- **implementation of T connections with high bandwidth**
  - (e.g., if subnet imposes max. data rate per L3 Virtual circuit)

Splitting function
- **distributing the TPDUs onto the various network connections**
- **usual algorithm: Round Robin**
- **e.g., for ISDN users two combine several lines**

Comment
- **also known as "downward multiplexing“**

10. Crash Recovery

Situation: transport layer looses all data (crash)
- **whereby “send confirmation” and “notify L5” are**
  - not simultaneous / **NOT ATOMIC**
- **faulty status possibilities**
  - sequences with: ACK, L5 (data transfer), crash

Sender-Strategy: to resend....	Situation at Recipient's Side: First ACK (confirm) then pass data to L5			First pass data to L5 then ACK (confirm)		
	ACK ↯ L5	ACK L5 ↯	↯ ACK L5	↯ L5 ACK	L5 ACK ↯	L5 ↯ ACK
last TPU(s)	OK	duplicate	OK	OK	duplicate	duplicate
No TPU(s)	loss	OK	loss	loss	OK	OK
Each acknowl-edged message	OK	duplicate	loss	loss	duplicate	OK
Each NON-ACKNOWLEDGED message	loss	OK	OK	OK	OK	duplicate

Crash Recovery

Solution principle:
- **application layer can compensate for transport layer error**
  - (complete loss of all states)
- **in general: L_{n+1} compensates for error of layer L_n**

Comment:
L4 acknowledgment on application level means that
- **message arrived at the application**
- **but that the application did not (yet?) acknowledged it**