TRANSPORT PROTOCOLS

INTRODUCTION TO PRINCIPLES OF TRANSPORT PROTOCOLS FOR TCP/IP NETWORKS

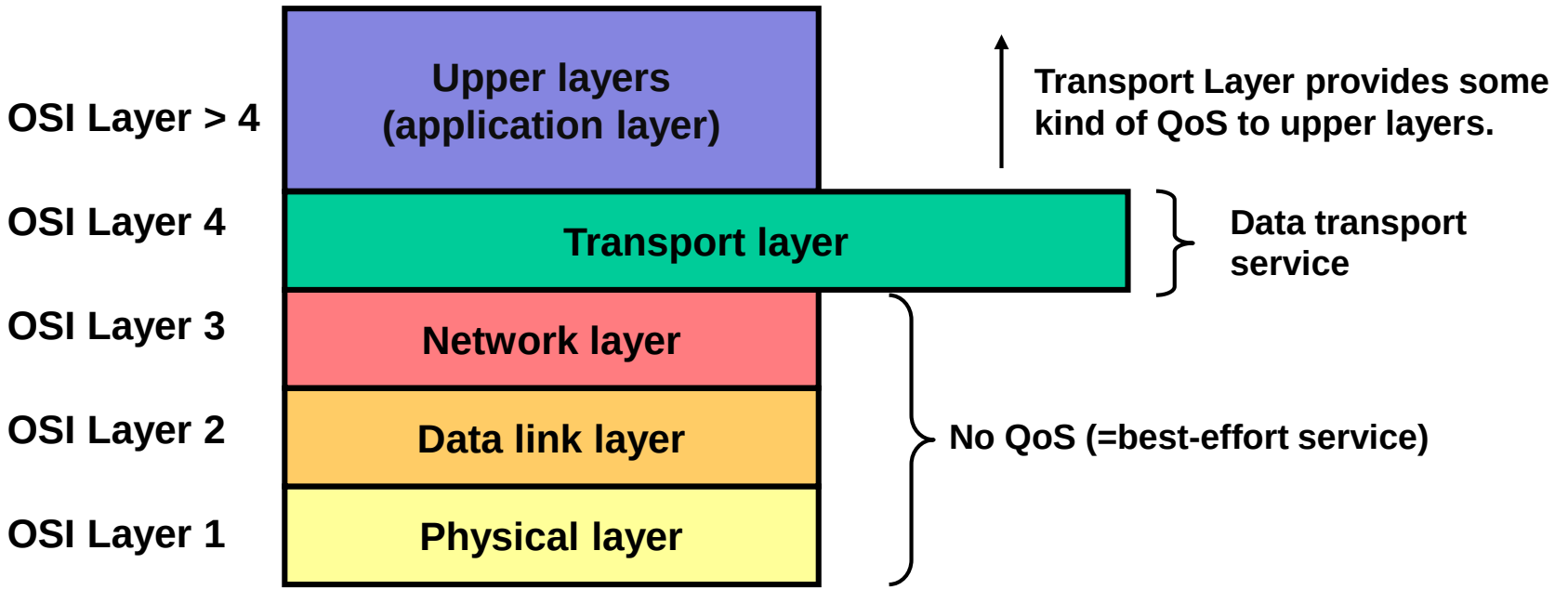
Peter R. Egli
peteregli.net

Contents

1. Transport layer functions (1/3)

The transport layer is the interface between the network and application („network API“).
The transport layer provides 2 main functions to the application:

1. **Data transport service (transport data to another remote or local application)**
2. **Some level of QoS (Quality of Service)**



QoS: Quality of Service
OSI: Open SystemS Interconnection

1. Transport layer functions (2/3)

QoS:

IP and lower layers do not provide any QoS. The service is best-effort (send packets, but do not provide any delivery guarantee). It is up to the upper layers to deal with network problems.

The transport layer may hide network imperfections (problems on the network) from the application.

Possible problems on the network are (incomplete list):

Packet loss:

Packet loss typically occurs in congested IP routers (too many packets have to be forwarded to the same outbound interface at the same time).

Packet duplicates:

Packet duplicates may occur due to routing loops or retransmissions due to a slow network.

Out of order packets:

Due to different transmission paths one IP packet may get ahead of another IP packet.

Bit errors:

Bit errors may occur due to various electromagnetic interferences. Bit errors are typically relatively high on wireless links.

Delay of packets:

Packets get delayed in the network due to buffering, transmission delay etc.

Different transport protocols provide different levels of QoS (from none to full QoS).

1. Transport layer functions (3/3)

Possible QoS (Quality of Service) characteristics or functions of the transport protocol:

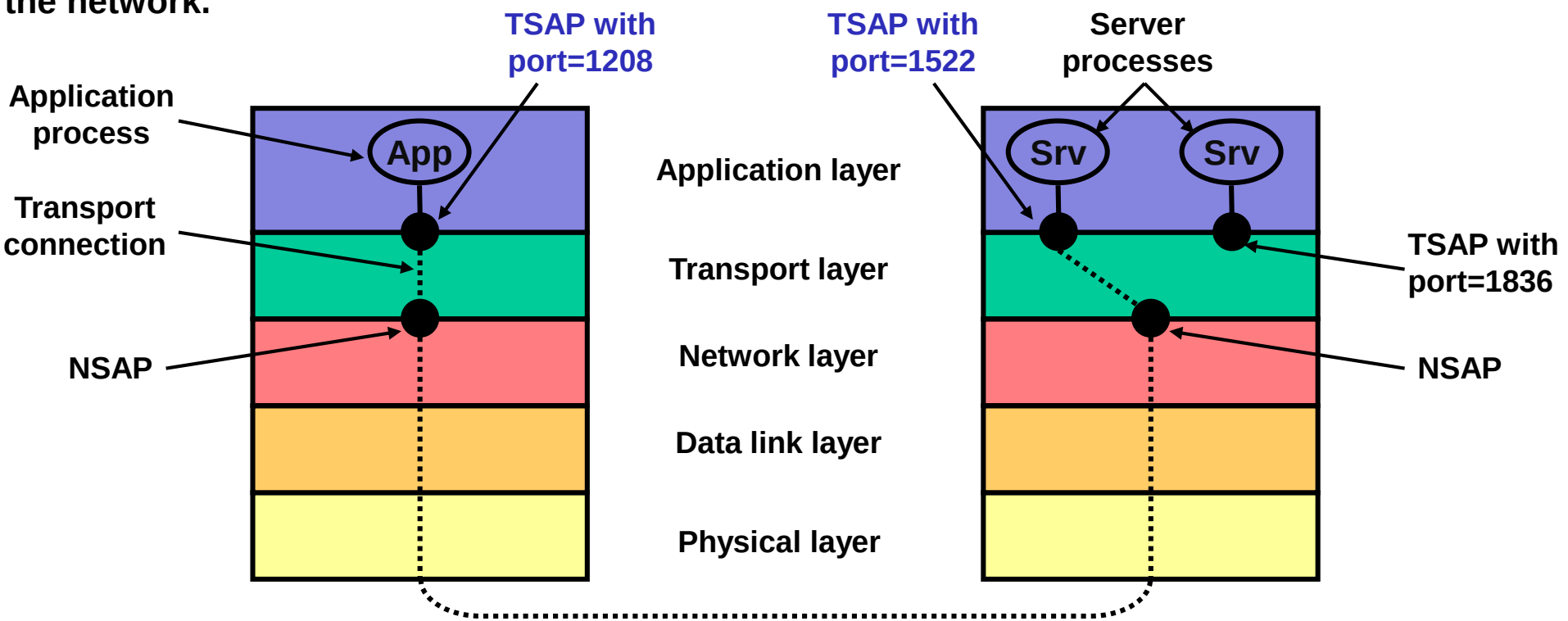
Connection establishment delay:
How long does it take to establish a transport connection?
Connection establishment failure probability:
How often does a connection establishment fail?
In-order delivery:
Does the transport protocol take care of packet ordering?
Throughput:
Does the transport protocol optimize throughput?
Transit delay:
Does the transport protocol minimize delay, i.e. send packets as quickly as possible?
Error ratio:
Does the transport protocol detect errors or even correct errors?
Priority:
Does the transport layer provide a priority mechanism (send high priority packets first)?
Resilience:
Do transport connections survive a system crash (persistence of connection)?
Protection:
Does the transport protocol provide protection against eavesdropping, wiretapping etc.?

2. Elements of transport protocols (1/14)

Addressing:

A TSAP (Transport Service Access Point) is the access point to the transport service for an application. A TSAP contains a port number as transport address.
TSAPs provide multiplexing / demultiplexing between different applications.

Likewise the NSAP (Network Service Access Point) is the access point to the network service for the transport layer. An NSAP contains an IP address, i.e. the address of a hop / node in the network.



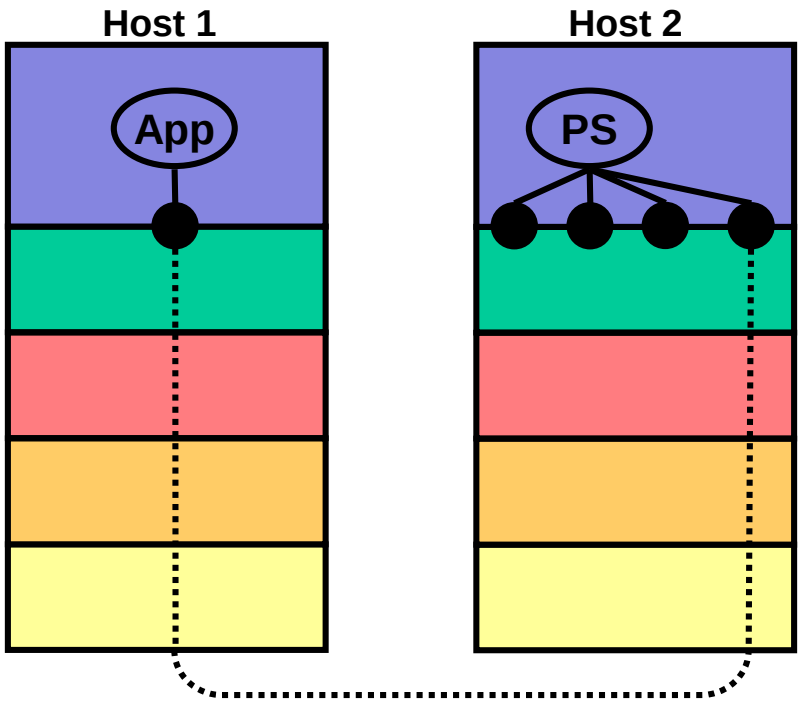
3. Elements of transport protocols (2/14)

Connection establishment (1/5):

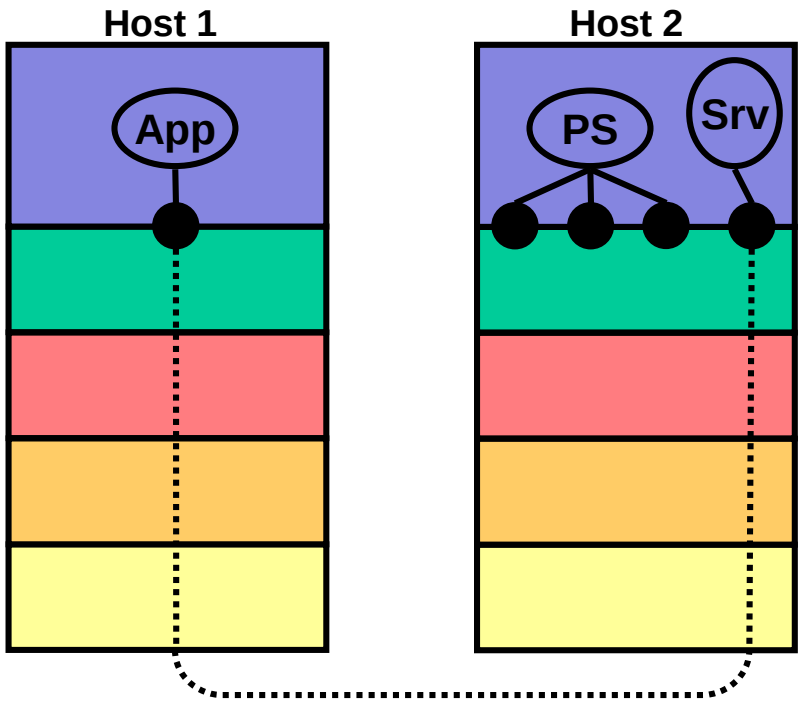
Prior to exchanging data a client and server must establish a connection (like a telephone connection).

On some architectures (Unix) a single server acts as a proxy and spawns the actual server process that provides the service (“xinetd” daemon).

1. The application connects to the process server’s (PS) TSAP.



2. The process server launches the respective application service (Srv) and passes it the connection (TSAP).



TSAPs

3. Elements of transport protocols (3/14)

Connection establishment (2/5):

Duplicate packet problem:

A network can duplicate packets. E.g. on a very slow network this happens when every packet is retransmitted once.

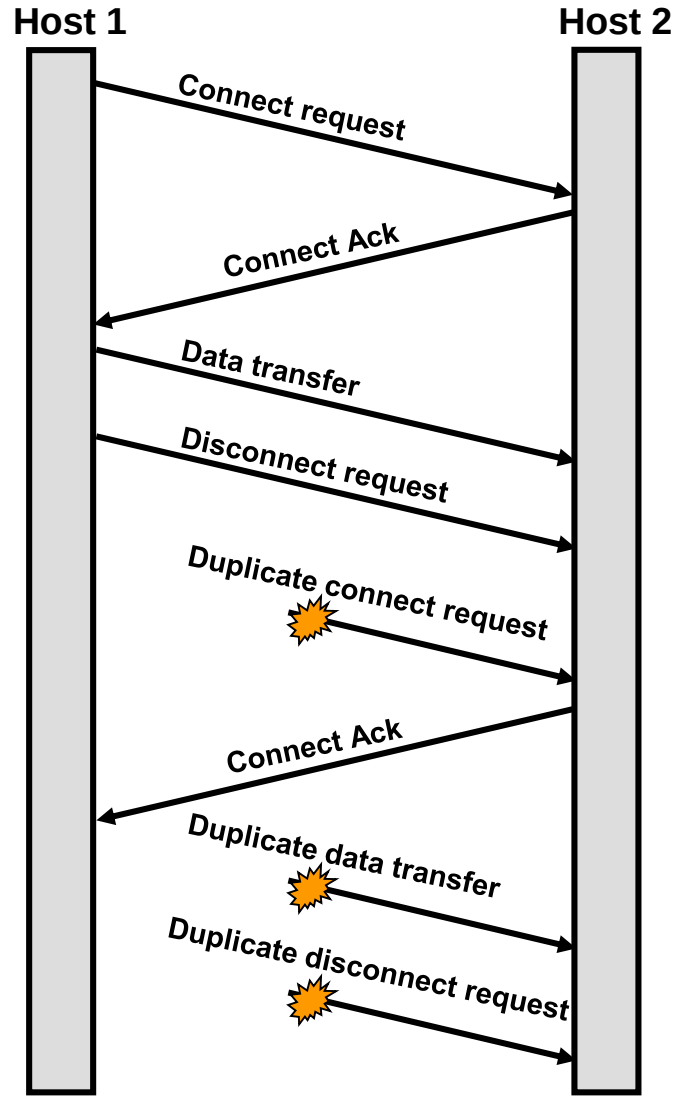
Proposed Solution 1:

The transport layers use a session / connection identifier. Each host maintains a table with used session identifiers.

- ☺ **Duplicate packets can be detected and discarded.**

But:

- ☹ **Possibly large tables.**
- ☹ **Difficult and slow to manage.**
- ☹ **Tables will not survive a host crash / reboot.**



3. Elements of transport protocols (4/14)

Connection establishment (3/5):

Proposed Solution 2 (duplicate packet problem):

A. Limit the lifetime (T) of packets in network through:

- *** Hop counter (TTL)**
- *** Restricted subnet design**
- *** Timestamp each packet (each host is required to have a clock that survives a crash).**

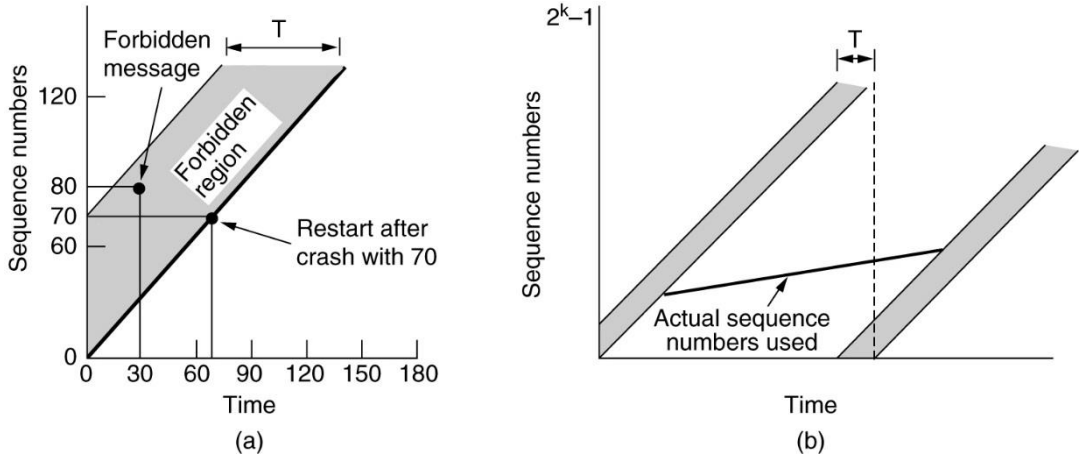
B. Make sure that time T passes after a packet with sequence number x is sent (forbidden regions).

☺ **Duplicate packets can be detected and discarded.**

But:

☹ **A real time clock (RTC) is required that survives system reboots (needs special hardware).**

☹ **Forbidden regions are difficult to avoid.**



Source: http://authors.phptr.com/tanenbaumcn4/

3. Elements of transport protocols (5/14)

Connection establishment (4/5):

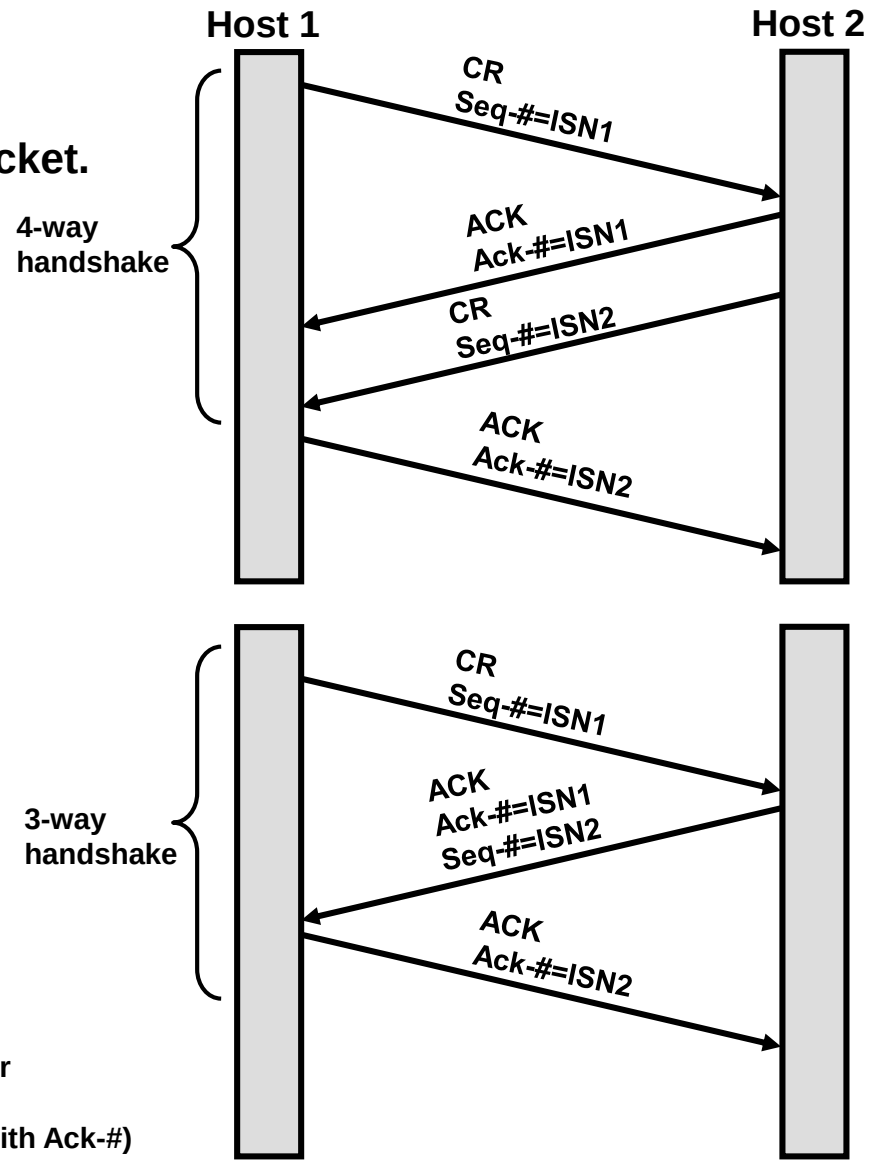
Proposed Solution 3 (duplicate packet problem):

A. Put a sequence number (~timestamp) into each packet.

B. Client and server acknowledge each others sequence numbers (synchronize each others sequence numbers).

- ☺ Fool-proof.
- ☺ Simple.
- ☺ No special requirements for sequence number (may be derived from system tick).

The 4-way handshake may be collapsed into a 3-way handshake.



Seq-#:	Sequence number
Ack-#:	Acknowledge number
ISN:	Initial Sequence Number
CR:	Connect Request
ACK:	Acknowledge (packet with Ack-#)

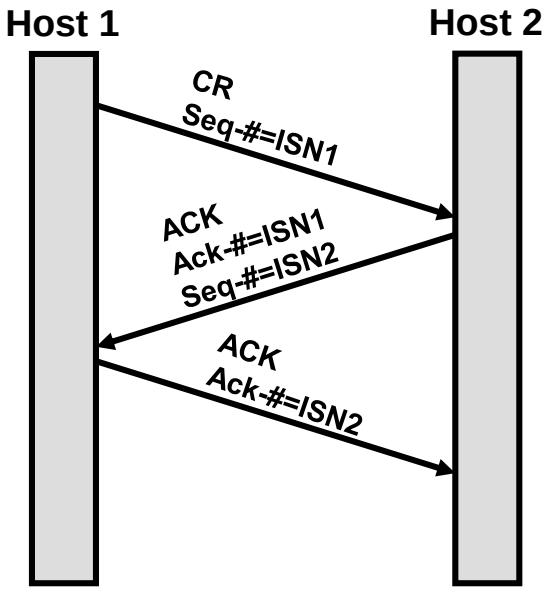
3. Elements of transport protocols (6/14)

Connection establishment (5/5):

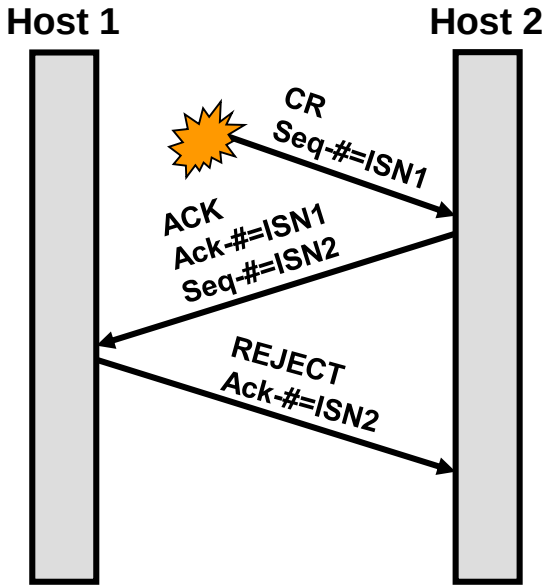
Proposed Solution 3 (duplicate packet problem):

The solution is foolproof. Duplicate packets can always be detected.

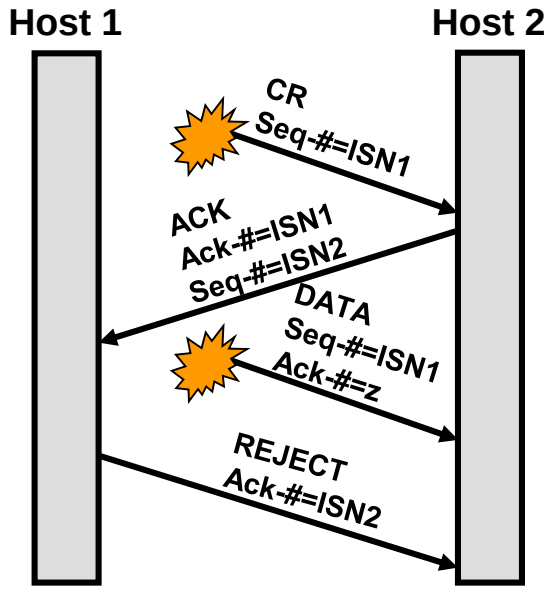
Normal operation:



Old connection request appearing out of the blue:



Duplicate connection request and duplicate Ack:



REJECT: Connection reject
DATA: Packet with user data

Old duplicate packets

4. Elements of transport protocols (7/14)

Connection release (1/3):

Problem:

Asymmetric release (only 1 peer closes the connection) is abrupt and may cause data loss.

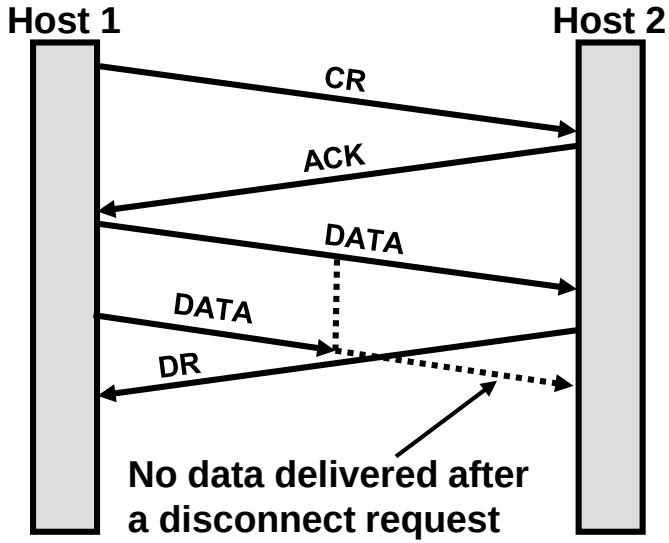
Proposed solution:

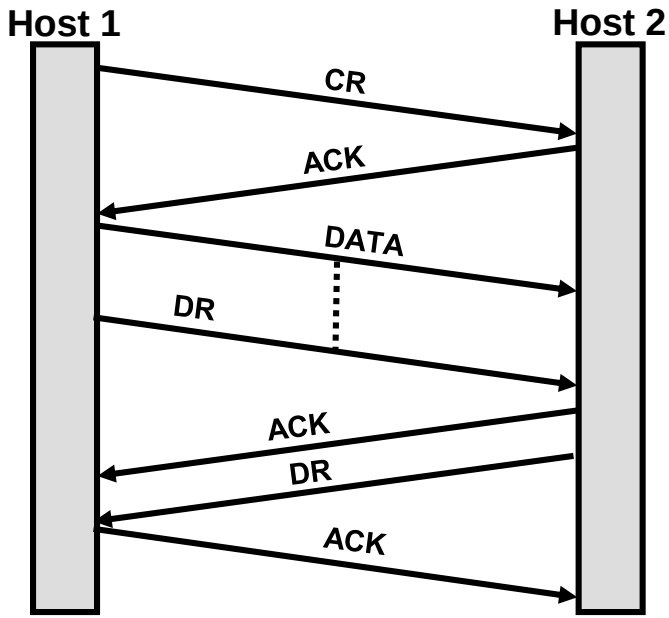
Either side will disconnect its (outgoing) direction of the duplex connection (possibly collapsed into 3-way handshake).

- 🙂 **No data loss.**

But:

- ☹ **Not fool-proof (DR packet may get lost and thus connection not closed, see next slide).**





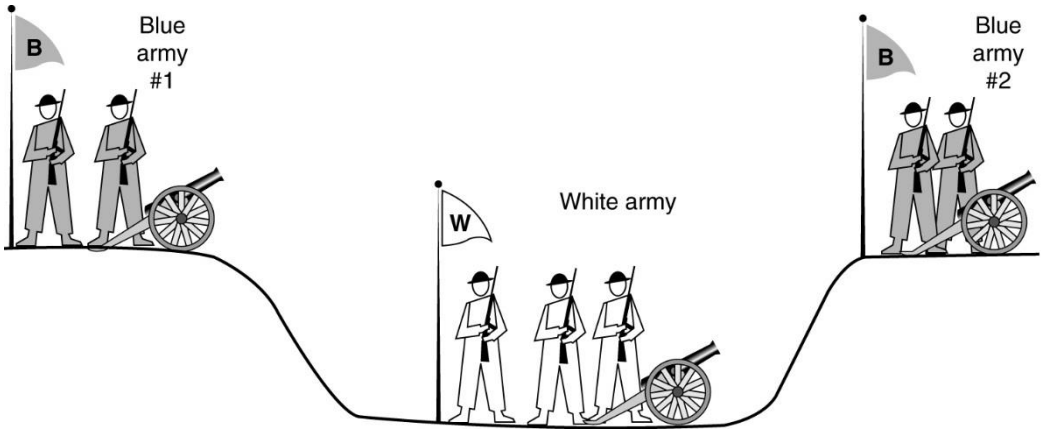
DR: Disconnect Request

4. Elements of transport protocols (8/14)

Connection release (2/2):

Problem:

No protocol/procedure exists that can guarantee the proper connection termination in case of packet loss ("Two-army problem", "Army-in-the-middle-problem").



Source: http://authors.phptr.com/tanenbaumcn4/

Army-in-the-middle situation:

The blue army has 4 troops (2 on either side of valley) while the white army has 3 troops. If both blue armies charge at the same time they can vanquish the white army. If only one of the blue armies charges it will succumb (3 white troops against 2 blue troops). This means: the blue armies have to synchronize their attack.
But in order to synchronize they need to send a messenger through the valley; of course the messenger can get caught by the white army ('lost packet').

Approach #1:

The blue army #1 sends a messenger to tell blue army #2 to attack @ 1400.

Problem:

The blue army #1 does not know if the messenger managed to convey message or if he was caught.
Thus blue army #1 will not attack.

Approach #2:

The blue army #2 sends back a messenger to acknowledge to blue army #1 that it got the message.

Problem:

The blue army #2 does not know if acknowledge-messenger reached blue army #1. Thus blue army #2 will not attack.

➔ This play can be continued ad infinitum. No algorithm exists to make the acknowledgment procedure fool-proof.

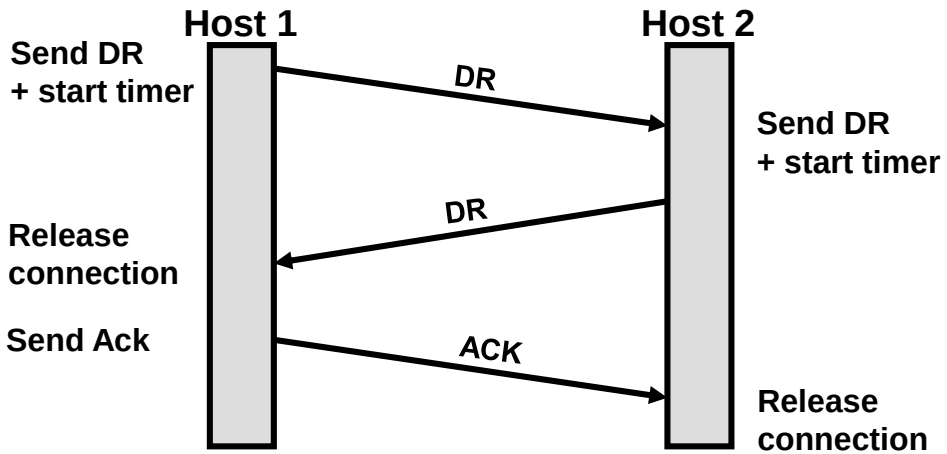
4. Elements of transport protocols (9/14)

Connection release (3/3):

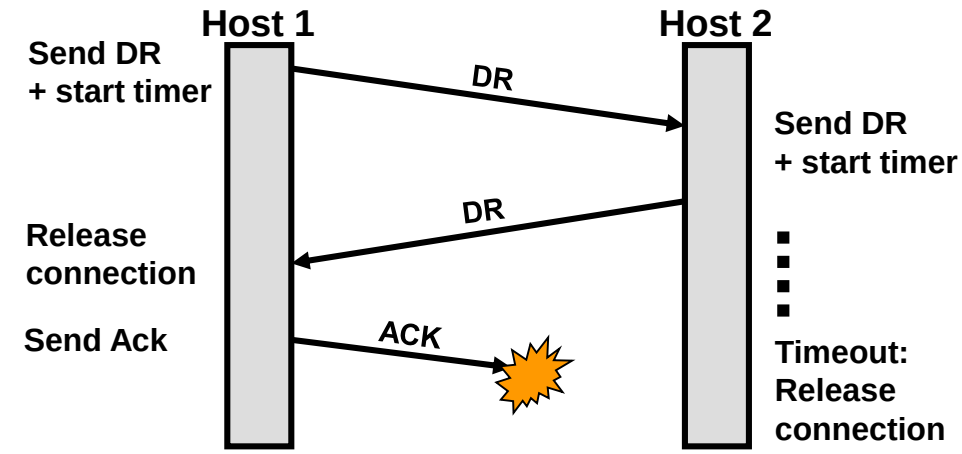
Proposed solution:

Start a timer when sending the DR. When it times out release the connection anyway.

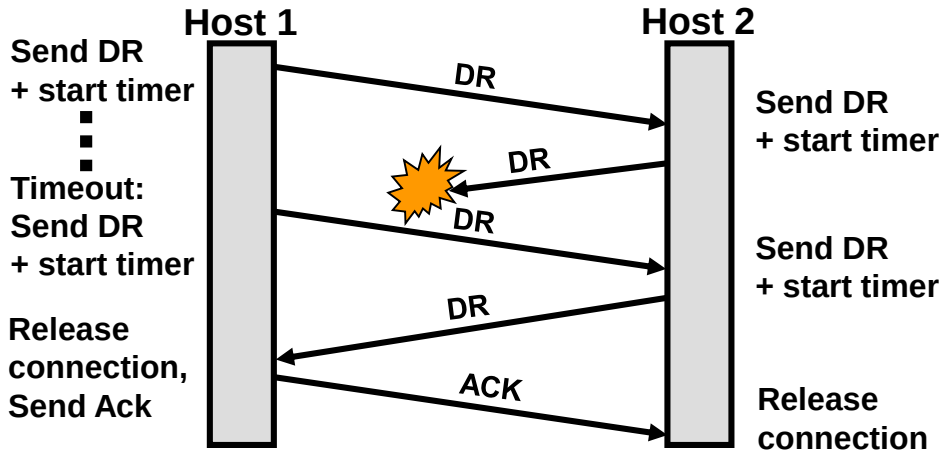
Normal case of 3-way handshake:



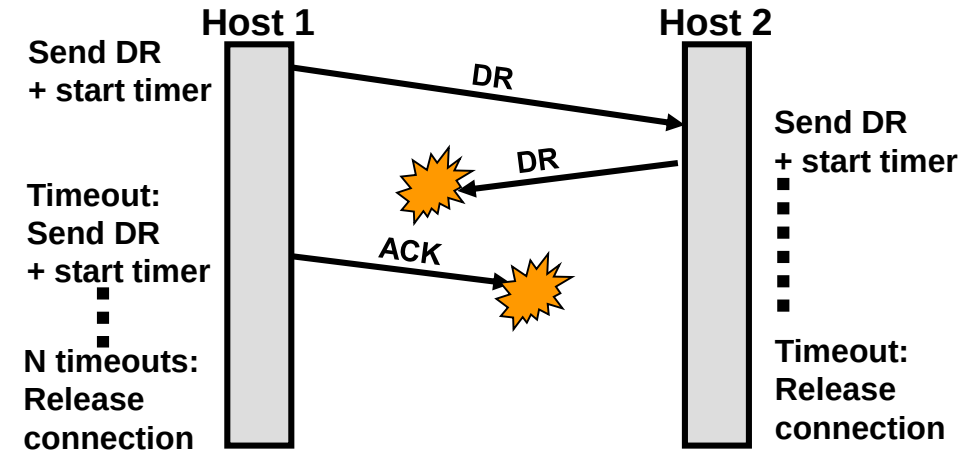
Final Ack lost:



Response lost:



Response and subsequent DR lost:



5. Elements of transport protocols (10/14)

Flow control and buffering:

Problem:

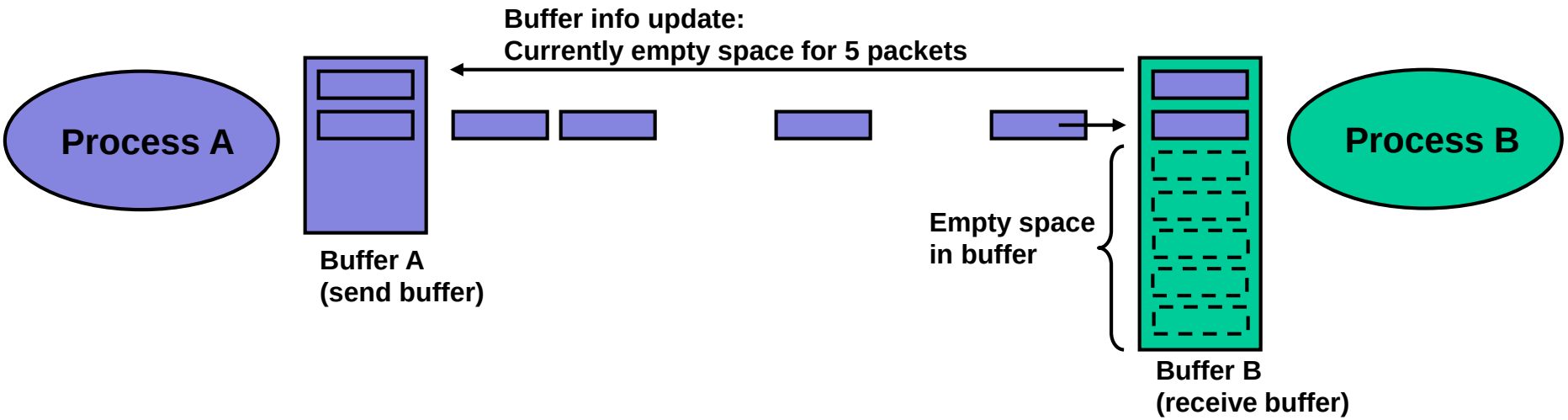
The sender process may send at much higher speed than the receiver process can handle the data thus causing overflow (= packet loss).

Proposed solution:

The receiver buffers incoming packets.

A sliding window mechanism provides a "backpressure" to the sender process when the buffer is imminent to overflow (or better prevents the receive buffer from becoming full in the first place). The receiver process continuously tells the sending process how much empty space is left in its receive buffer. The sender process never sends more data than can be accommodated in the receive buffer.

More details see TCP flow control.



6. Elements of transport protocols (11/14)

Multiplexing:

Multiplexing in the transport layer can be used for optimization.

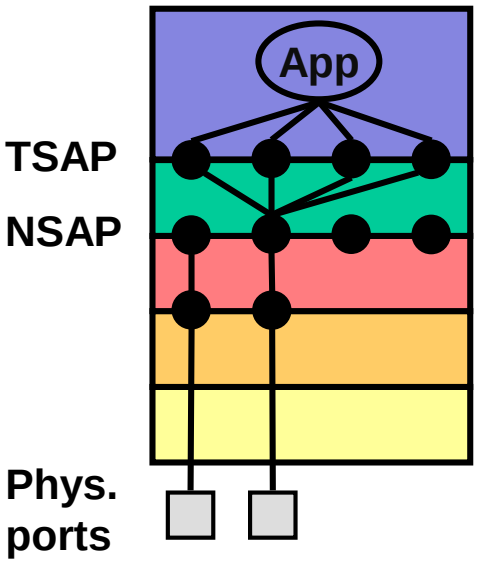
a. Upward multiplexing:

Traffic from a "data stream" is distributed over several transport connections (TSAPs). An application may use multiple TCP connections to improve throughput (TCP's throughput depends on delay, so overall throughput may be improved over physical lines with high delay).

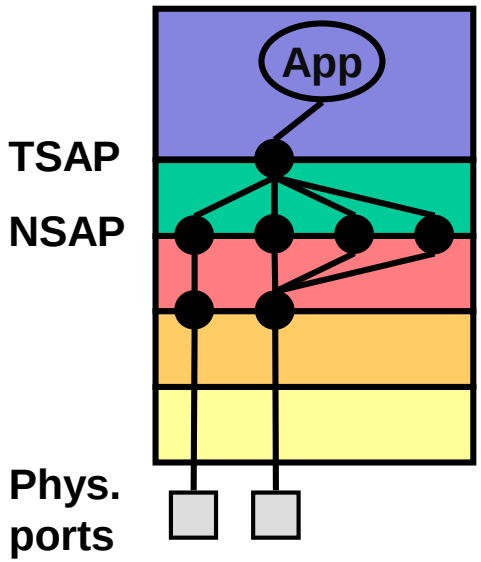
b. Downward multiplexing:

Many "data streams" share the same transport connection using multiple NSAPs, possibly over multiple network interfaces (load balancing). Stream Control Transmission Protocol (SCTP) is a transport protocol that may use downward multiplexing (multi-homing).

a. Upward multiplexing:



b. Downward multiplexing:



7. Elements of transport protocols (12/14)

Crash recovery (1/2):

Problem:

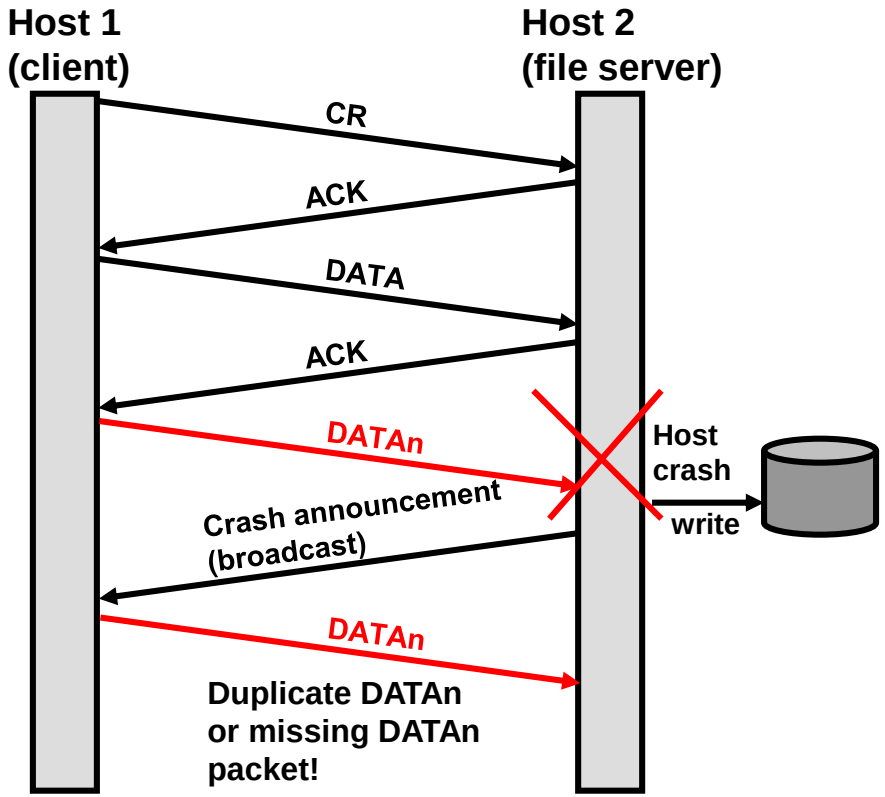
A crash of one host (server) during the transmission leads to a connection loss which results in data loss.

Proposed Solution:

The client retransmits only unacknowledged packets.

But:

- **Does not work in all cases because the server sends the ACK and writes the data to the application sequentially (see next slide).**



7. Elements of transport protocols (13/14)

Crash recovery (2/2):

No matter what the strategy of the hosts is, it is impossible to recover 100%-ly and transparently to the application from transport layer crashes. More generally: A crash at layer N can only be handled at layer N+1 (a system crash is a crash at every layer).

Thus:

It is left to the application layer to handle crashes of the remote host (client or server). Generally applications detect that the remote host has died and then simply restart the connection and retransmit everything.

Strategy used by the sending host	Strategy used by the receiving host					
	First ACK, then write			First write, then ACK		
	AC(W)	AWC	C(AW)	C(WA)	WAC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

Result:
OK = Protocol works correctly
DUP = Protocol creates a duplicate message
LOST = Protocol loses a message

State:
S0 = No unacknowledged packet outstanding
S1 = 1 unacknowledged packet outstanding

Action:
A = Server sending acknowledgment
W = Server writing to output process
C = Crashing

Source: http://authors.phptr.com/tanenbaumcn4/

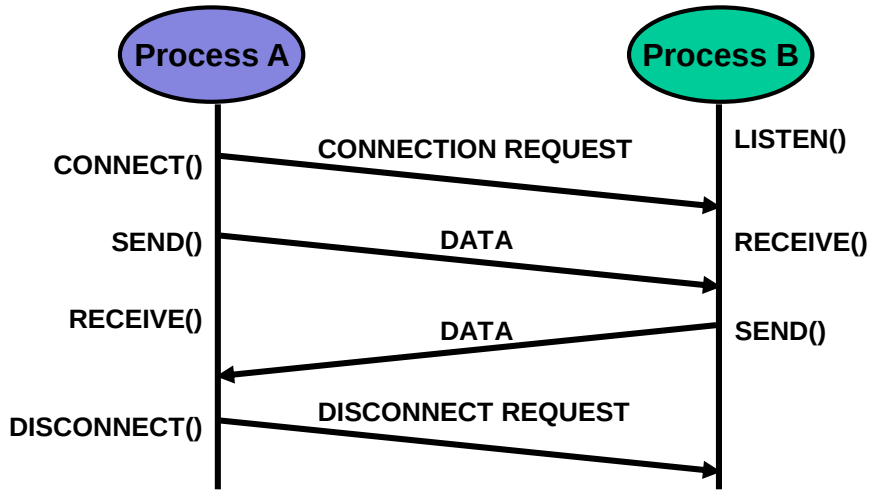
8. Elements of transport protocols (14/14)

Programming API:

The programming API of the transport layer depends on the platform / language / framework that is used. However, the transport APIs of different platforms are usually very similar.

A generic API of a connection-oriented transport protocol could look as follows:

Primitive (function)	Description
LISTEN	This function blocks until another process tries to connect (calls CONNECT), thus does a *passive open*.
CONNECT	Sends a connection request packet, thus does an *active open*. This function is the counterpart to the LISTEN function.
SEND	Send user data.
RECEIVE	Blocks until a user data packet arrives.
DISCONNECT	Sends a disconnect request to close the connection.

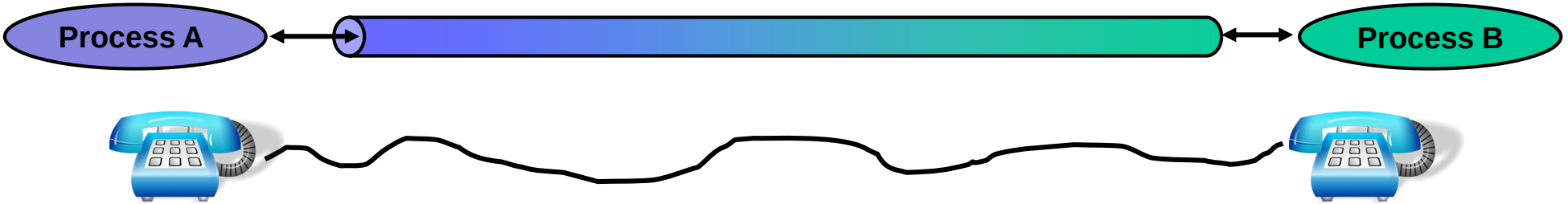


9. Transport layer characteristics (1/2)

A transport layer protocol is either *connection-oriented* or *connection-less*.

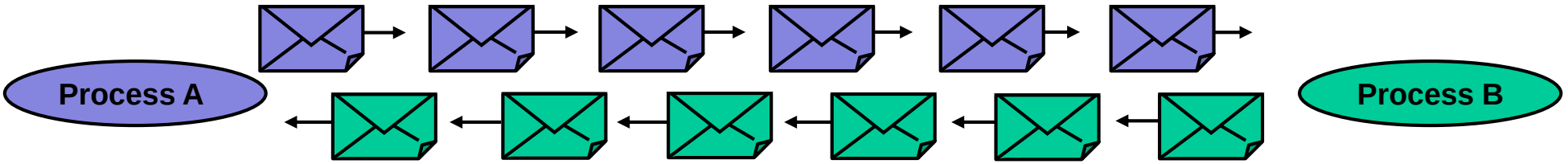
Connection-oriented transport protocols:

The peers establish a connection prior to a data exchange.
This is similar to a telephone line that needs setting up a connection prior to a conversation.



Connection-less transport protocols:

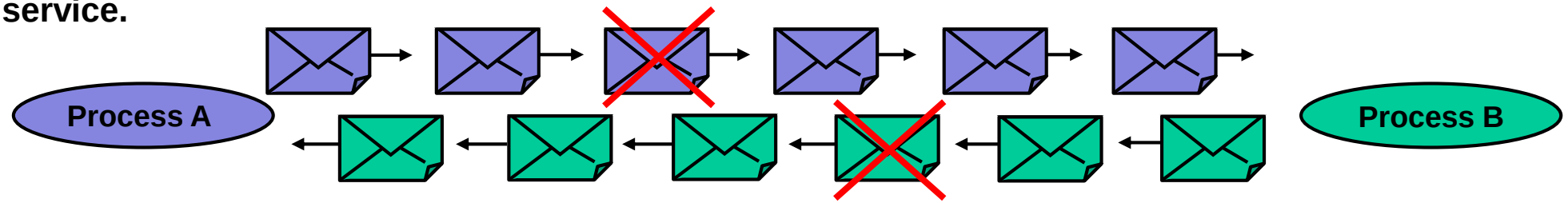
The peers send packets without a prior connection establishment.
This is similar to the traditional postal service.



9. Transport layer characteristics (2/2)

Reliable versus unreliable service:

Transport protocols provide reither eliable (guaranteed delivery) or unreliable (best-effort) service.



Combinations:

The characteristics connection-oriented / connection-less and reliable / unreliable can be combined. Usually connection-oriented protocols provide reliable transport service.

	Reliable	Unreliable
Connection-oriented	TCP, SCTP	-
Connection-less	RUDP	UDP

UDP: Unreliable, connection-less message (datagram) delivery protocol.
TCP: Reliable, connection-oriented stream transfer protocol.
SCTP: Reliable, connection-oriented message transfer protocol.
RUDP: Reliable UDP (mixture between TCP and UDP)