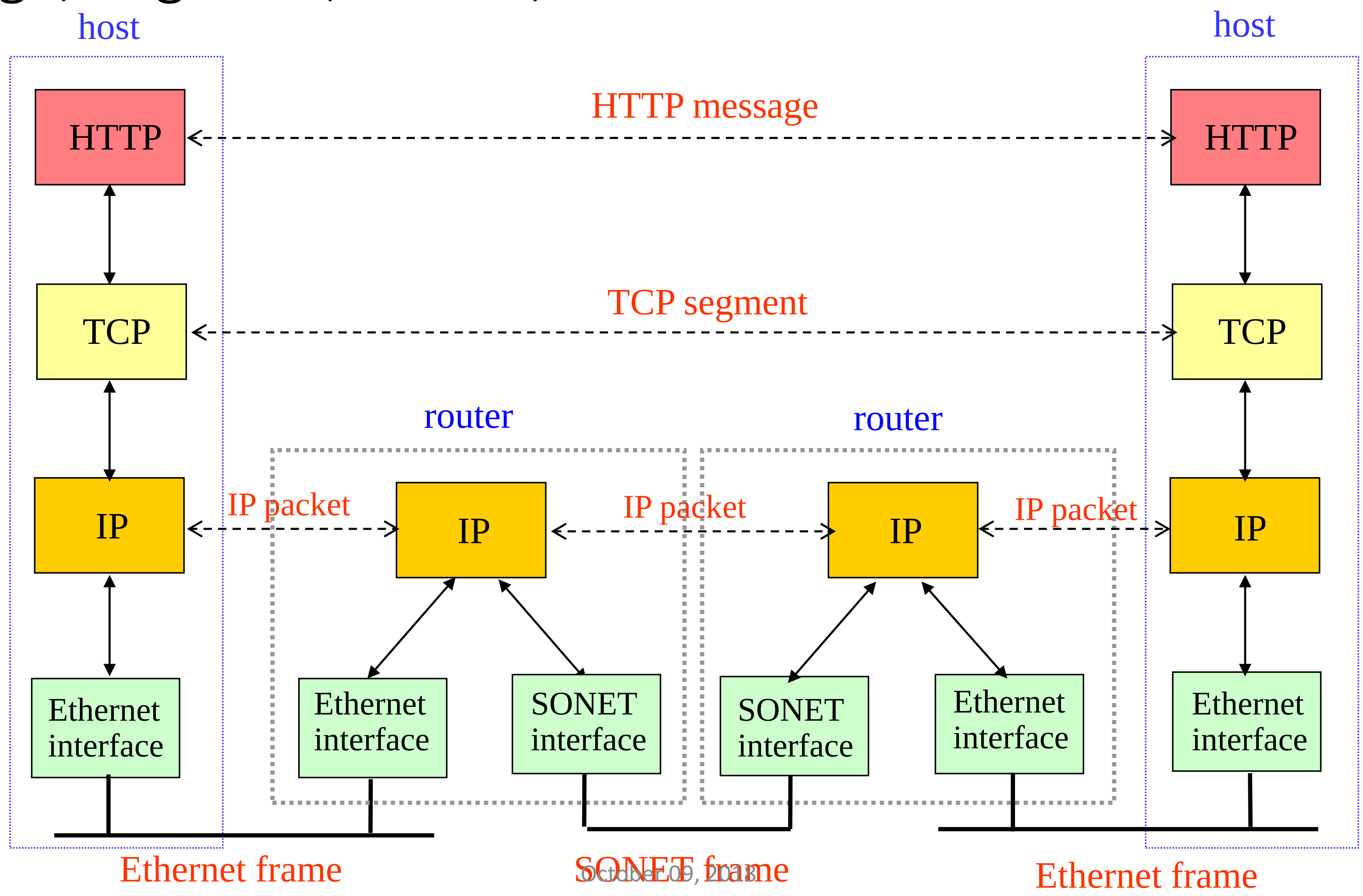
CMSC 417

Computer Networks

Prof. Ashok K Agrawala

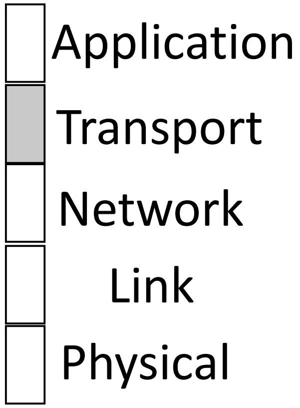
The Transport Layer

Message, Segment, Packet, and Frame



The Transport Layer

Responsible for delivering data across networks with the desired reliability or quality



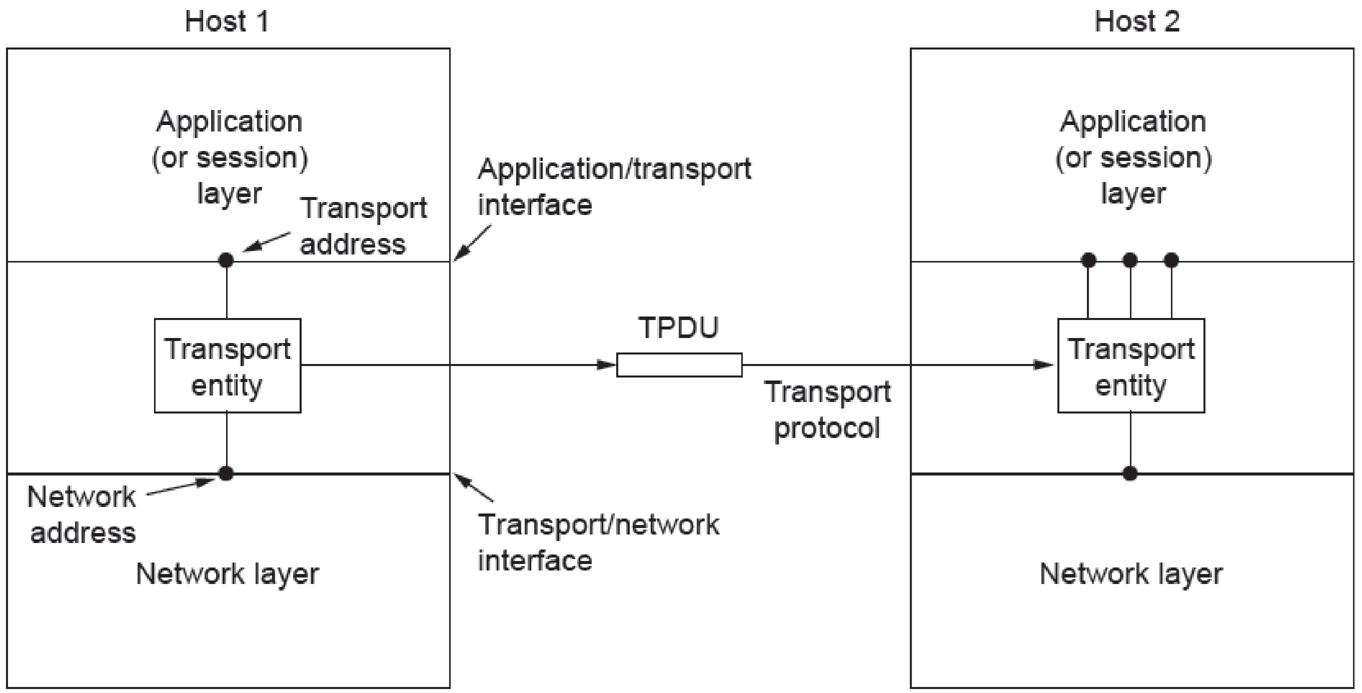
The Transport Service

- Services Provided to the Upper Layers
- Transport Service Primitives
- Berkeley Sockets
- An Example of Socket Programming:
    - An Internet File Server

Services Provided to the Upper Layers (1)

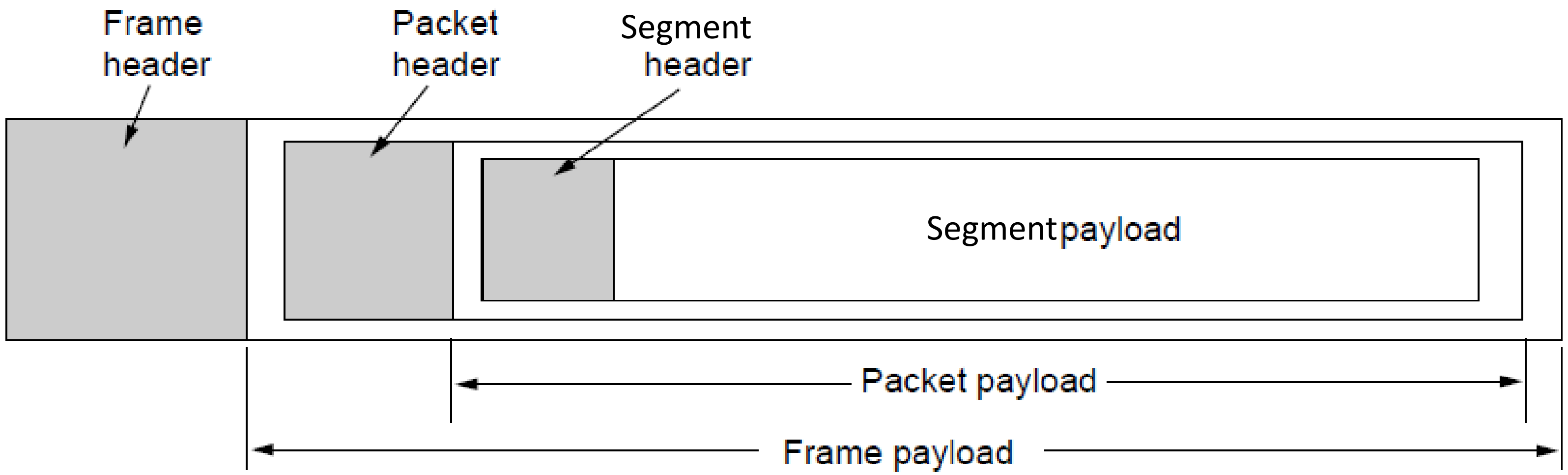
Transport layer adds reliability to the network layer

- Offers connectionless (e.g., UDP) and connection-oriented (e.g. TCP) service to applications

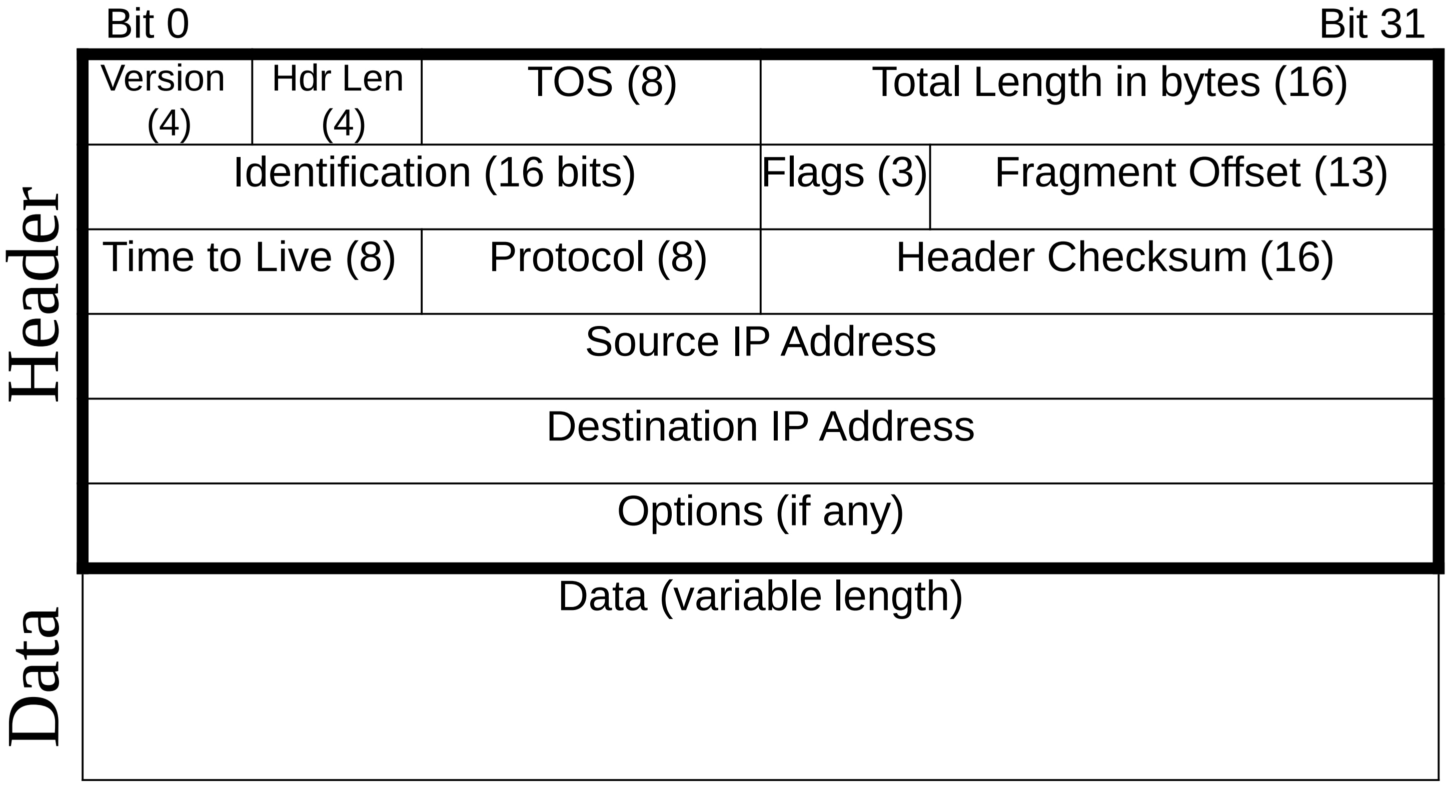


Services Provided to the Upper Layers (2)

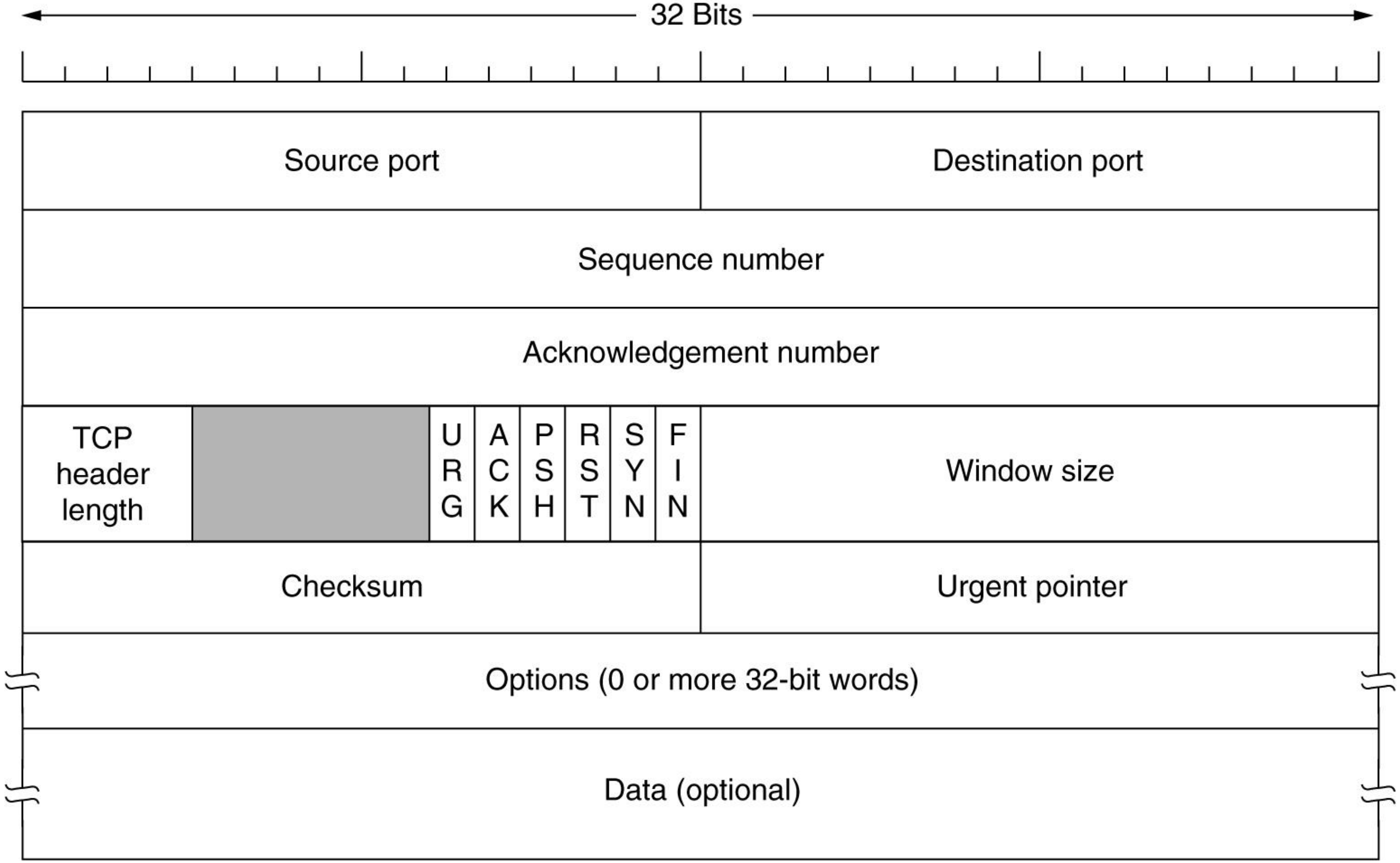
Transport layer sends <u>segments</u> in packets (in frames)



The IP Datagram



The TCP Segment Header



TCP Header.

Berkeley Sockets

Very widely used primitives started with TCP on UNIX

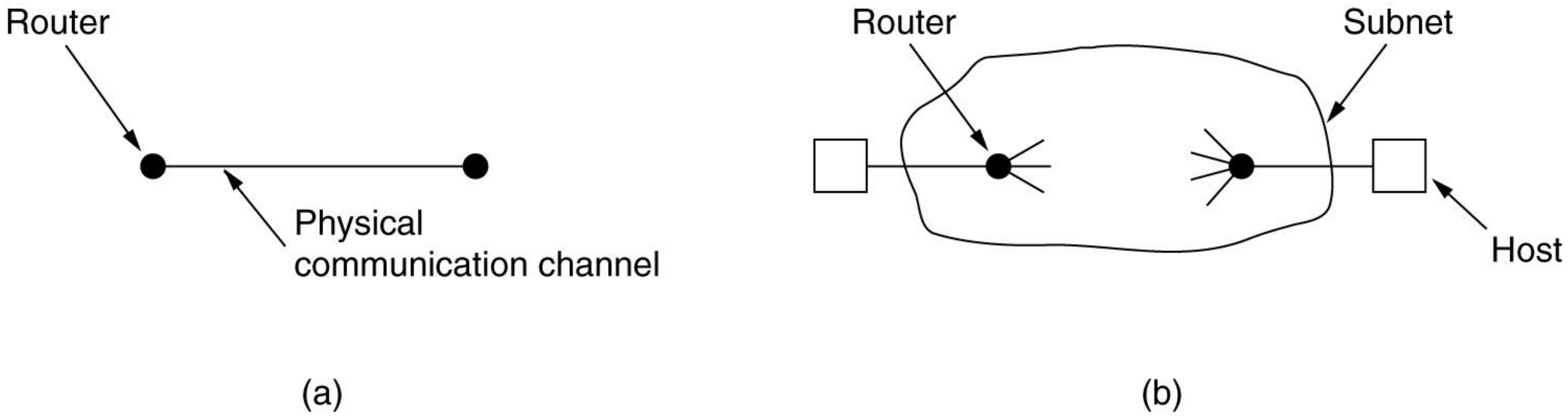
- Notion of "sockets" as transport endpoints
- Like simple set plus SOCKET, BIND, and ACCEPT

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
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

Elements of Transport Protocols

- Addressing »
- Connection establishment »
- Connection release »
- Error control and flow control »
- Multiplexing »
- Crash recovery »

Transport Protocol



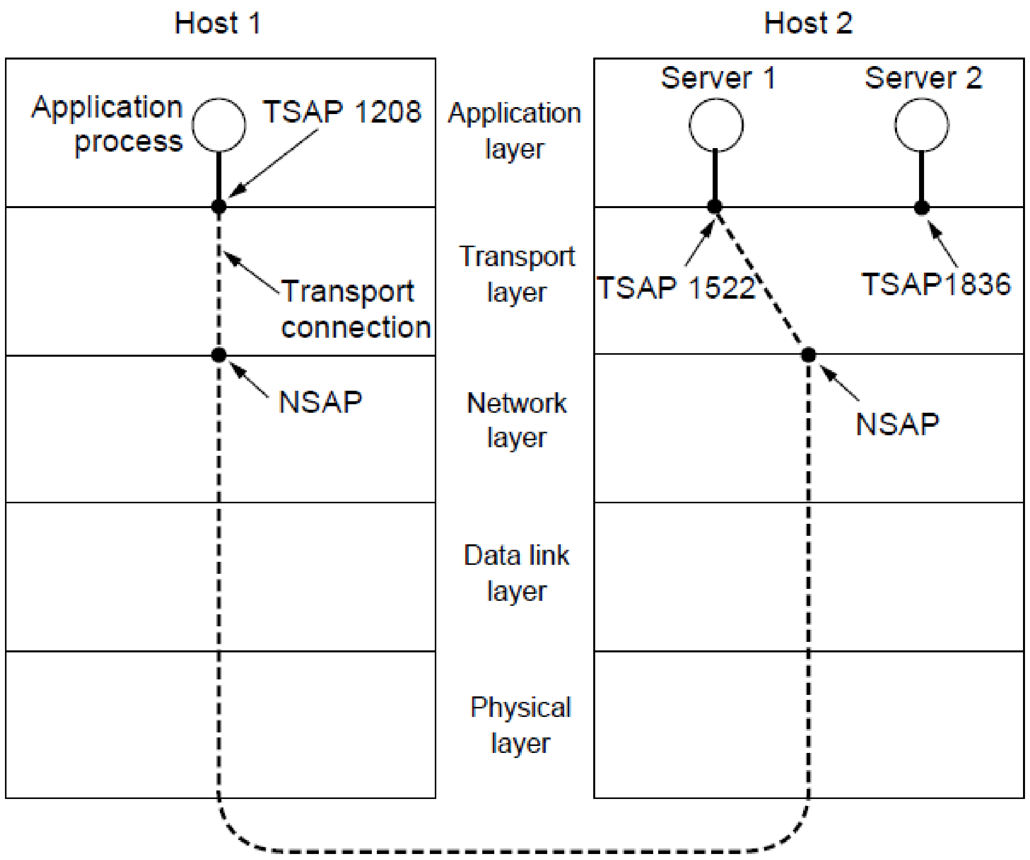
(a) Environment of the data link layer.

(b) Environment of the transport layer.

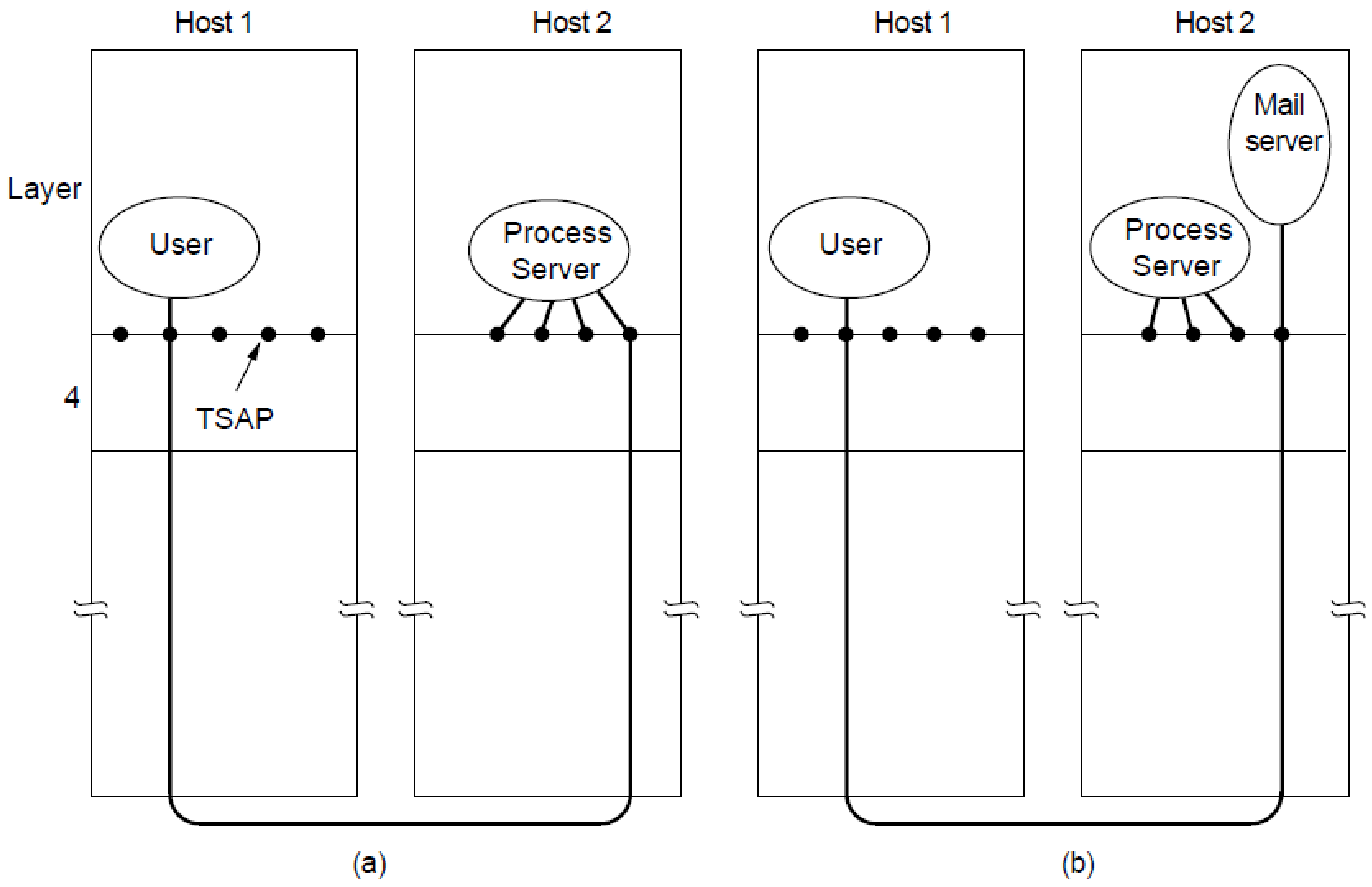
Packets may be lost, delayed, duplicated, out of order, etc.

Addressing

- Transport layer adds TSAPs
- Multiple clients and servers can run on a host with a single network (IP) address
- TSAPs are ports for TCP/UDP



Addressing



How a user process in host 1 establishes a connection with a mail server in host 2 via a process server.

Connection Establishment

- Connect
  - Expect confirmation that one and only one connection has been established
  - In a reasonable time
- Send Connection Request
- Expect Connection Accepted
- Segments (Packets) may be lost, corrupted, delayed, out of order, duplicated.

Connection Establishment

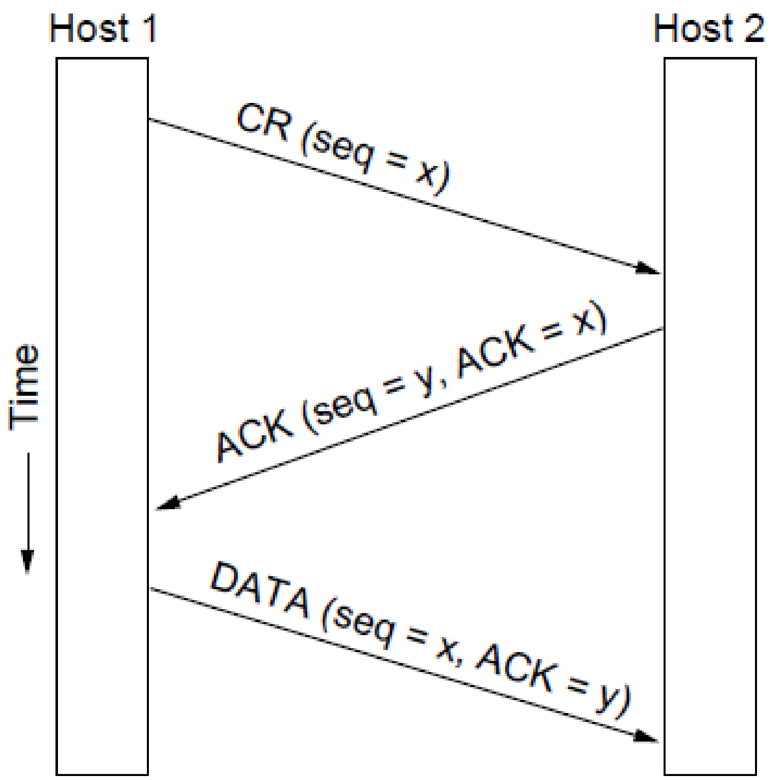
Key problem is to ensure reliability even though packets may be lost, corrupted, delayed, and duplicated

- Detect corrupted packets (error detection – e.g. checksum)
- Detect lost packets (Time Out)
- Identify retransmitted packets (sequence numbers)

Connection Establishment (3)

Three-way handshake used for initial packet

- Since no state from previous connection
- Both hosts contribute fresh seq. numbers
- CR = Connect Request



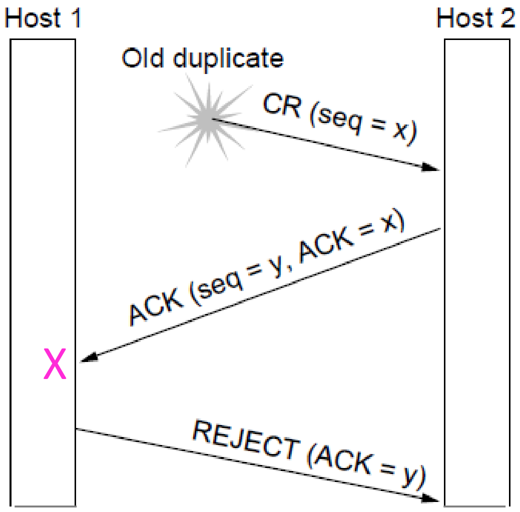
Connection Establishment (4)

Three-way handshake protects against odd cases:

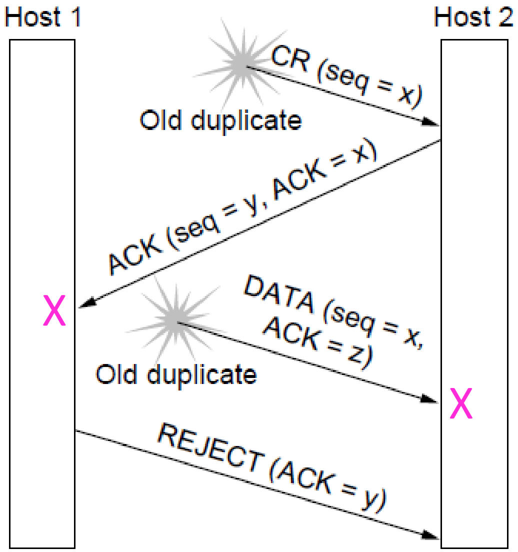
a) Duplicate CR. Spurious ACK does not connect

b) Duplicate CR and DATA. Same plus DATA will be rejected (wrong ACK).

a)



b)



Sequence Numbers

- An integer number with finite number of bits
  - 8 bits => 256
  - 16 bits => 64K
- Roll Over
  - How long does it take?
  - Can there be old duplicates in the system?

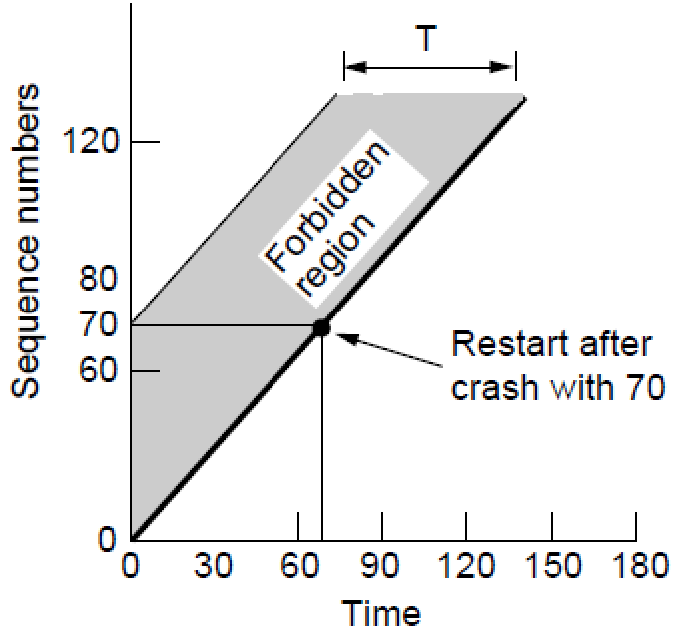
Approach:

- Don’t reuse sequence numbers within twice the MSL (Maximum Segment Lifetime) of 2T=240 secs
- Three-way handshake for establishing connection

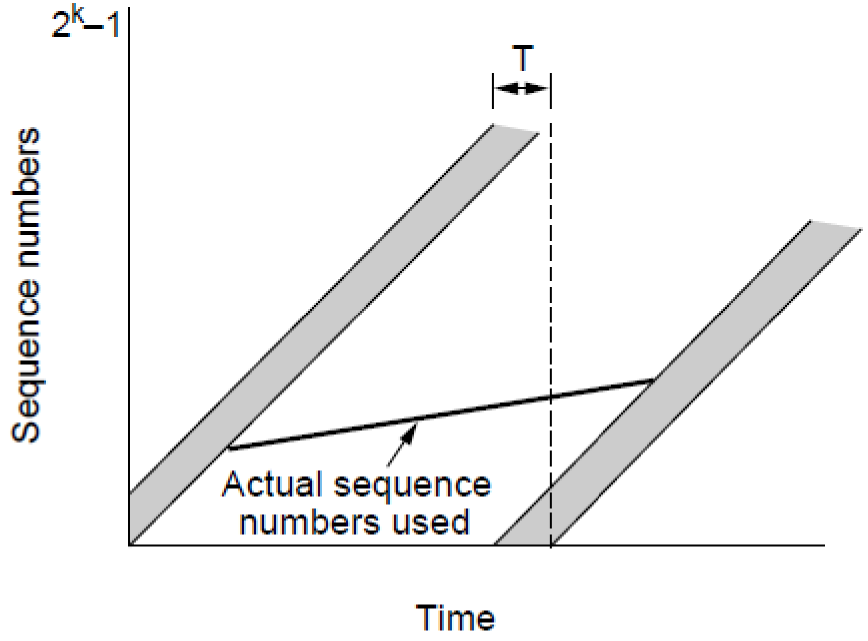
Connection Establishment (2)

Use a sequence number space large enough that it will not wrap, even when sending at full rate

- Clock (high bits) advances & keeps state over crash



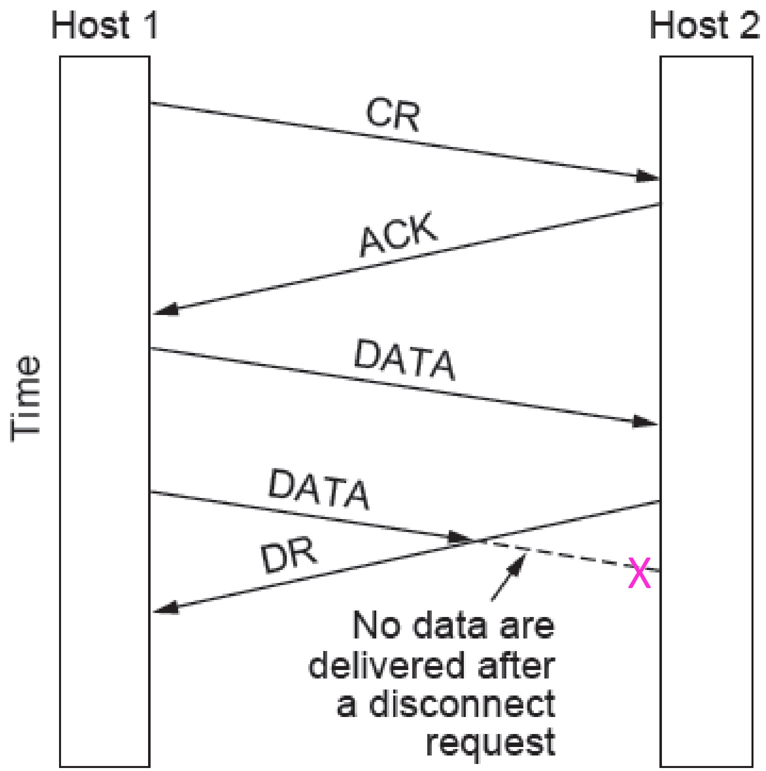
Need seq. number not to wrap within T seconds



Need seq. number not to climb too slowly for too long

Connection Release (1)

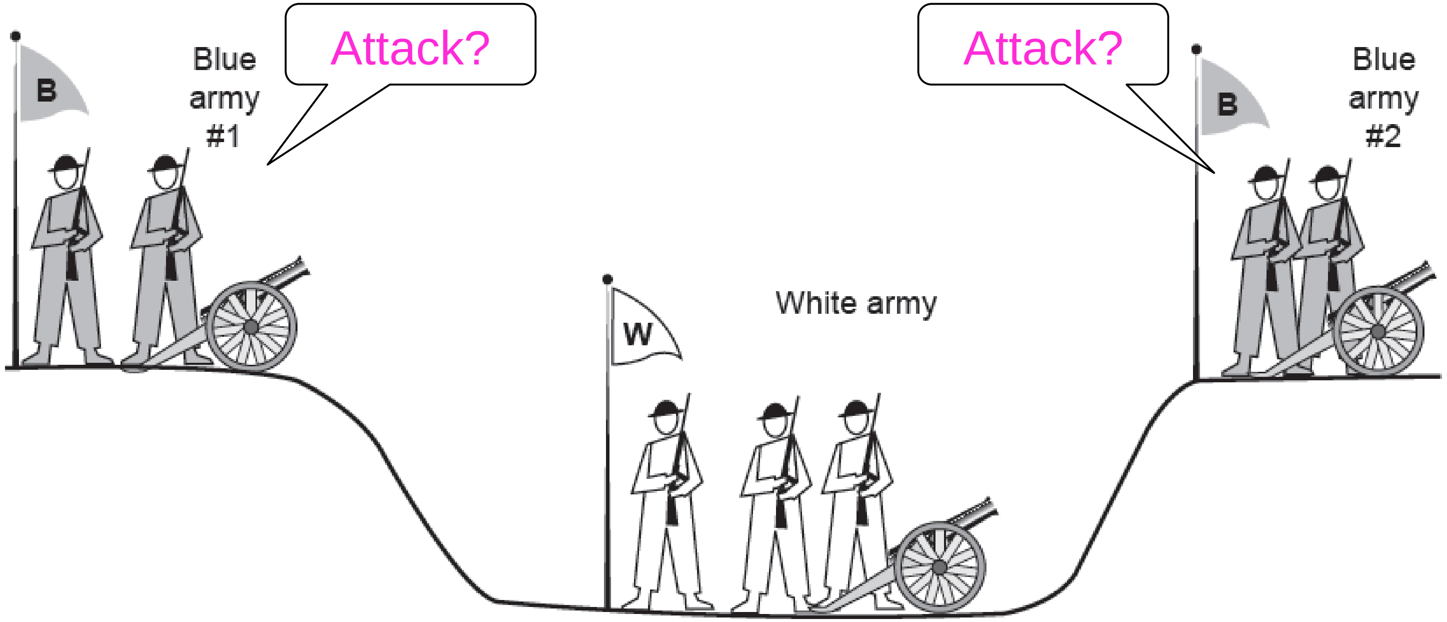
- Key problem is to ensure reliability while releasing
- Asymmetric release (when one side breaks connection) is abrupt and may lose data



Connection Release (2)

Symmetric release (both sides agree to release) can't be handled solely by the transport layer

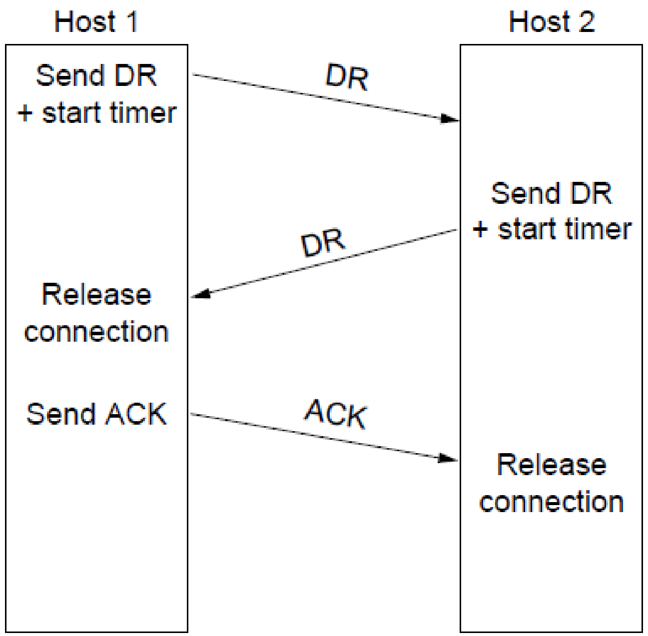
- Two-army problem shows pitfall of agreement



Connection Release (3)

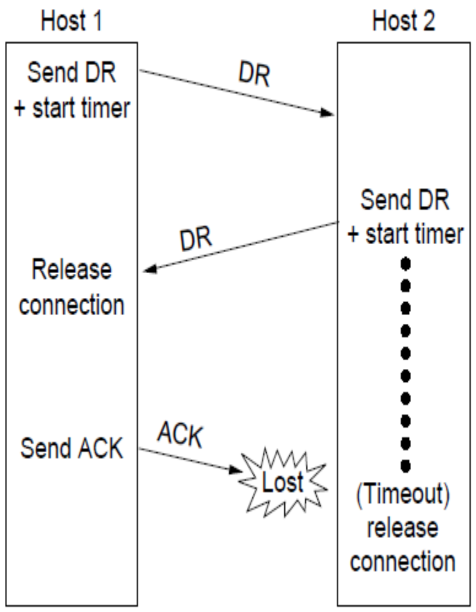
Normal release sequence, initiated by transport user on Host 1

- DR=Disconnect Request
- Both DRs are ACKed by the other side

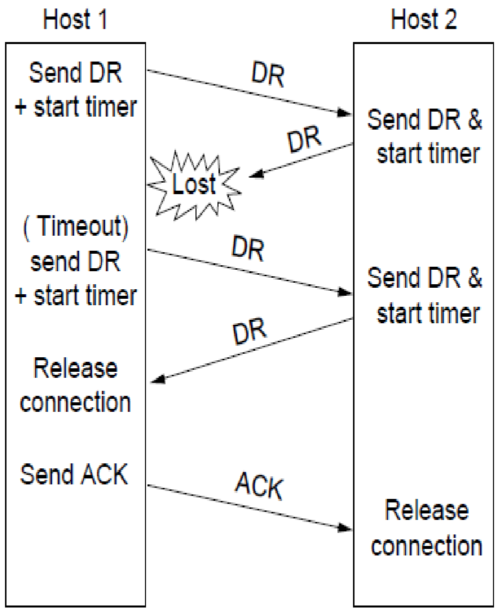


Connection Release (4)

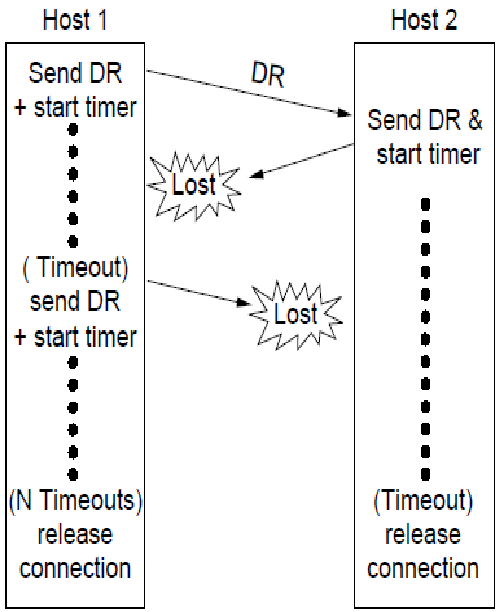
Error cases are handled with timer and retransmission



Final ACK lost,
Host 2 times out



Lost DR causes
retransmissions



Extreme: Many lost
DRs cause both hosts
to timeout

Flow Control

- Use Sliding Window
- Buffering
  - Sender buffers all TPDUs until acknowledged
  - TPDU lost by the network
    - Unreliable service
    - Receiver not having buffer
- How should buffers be managed
  - Dedicate
  - Acquire when needed
- Traffic
  - Low bandwidth, Bursty – buffer at sender – acquire at receiver
  - High bandwidth, smooth – Buffer at both ends
- Exchange Buffer information

Error Control and Flow Control (1)

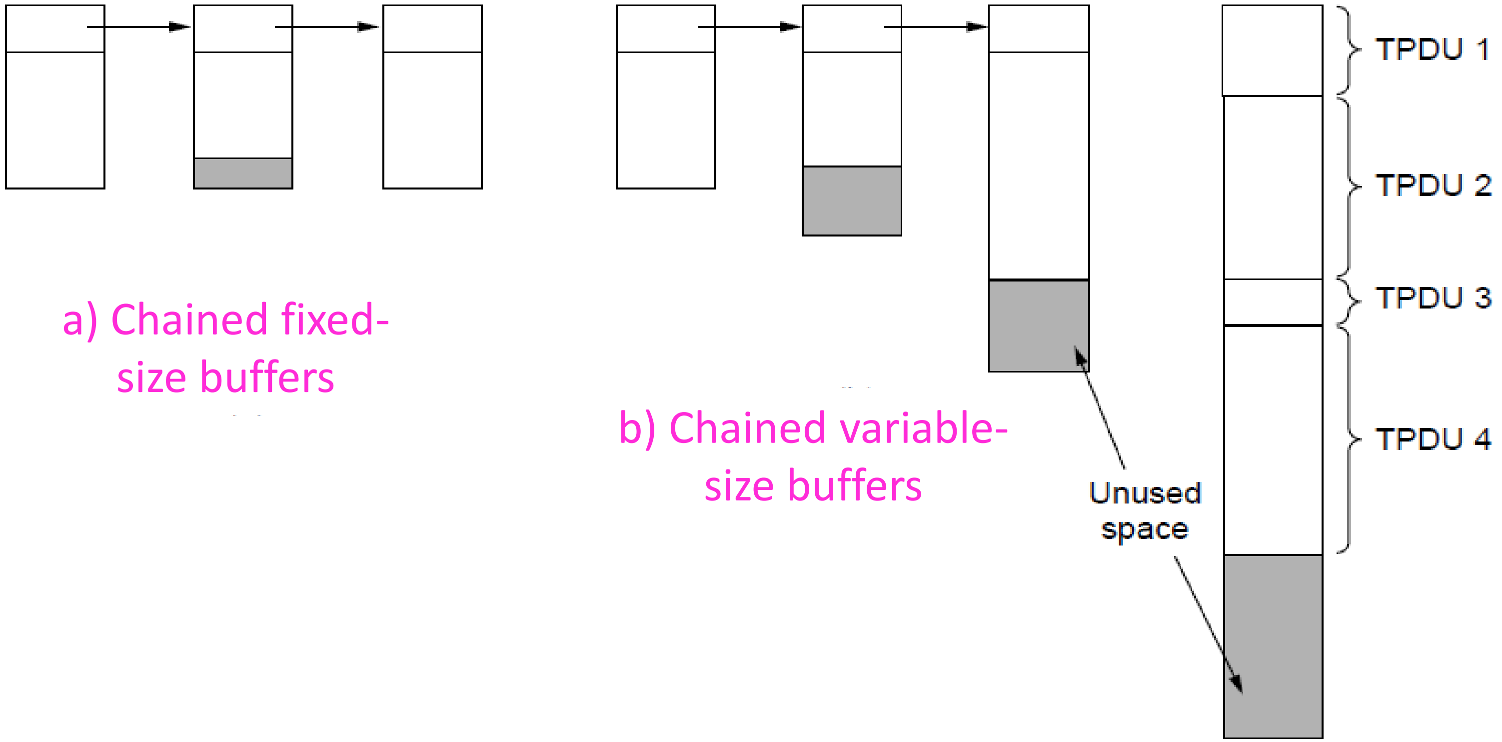
Foundation for error control is a sliding window (from Link layer) with checksums and retransmissions

Flow control manages buffering at sender/receiver

- Issue is that data goes to/from the network and applications at different times
- Window tells sender available buffering at receiver
- Makes a variable-size sliding window

Error Control and Flow Control (2)

Different buffer strategies trade efficiency / complexity



a) Chained fixed-size buffers

b) Chained variable-size buffers

c) One large circular buffer

Error Control and Flow Control (3)

Flow control example: A's data is limited by B's

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