The Transport Layer

Service Primitives

a) Service generally includes set of various primitives.

b) A primitive simply means Operations.

c) Service is specified by set of primitives that are available and given to user or other various entities to access the service.

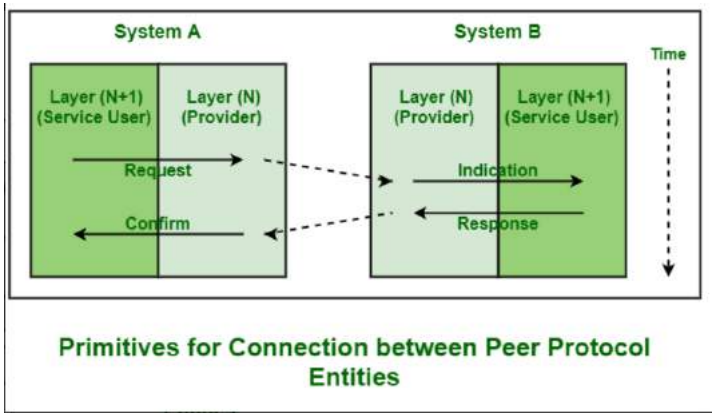
d) Primitives are called calling functions between the layers that are used to manage communication among the adjacent protocol layers i.e., among the same communication node.

e) All these primitives simply tell the service to perform some action or to report on action that is taken by peer entity.

Classification of Service Primitives

Primitive	Meaning
Request	**It represent entity that wants or request service to perform some action or do some work (requesting for connection to remote computer).**
Indication	**It represent entity that is to be informed about event (receiver just have received request of connection).**
Response	**It represents entity that is responding to event (receiver is simply sending the permission or allowing to connect).**
Confirm	**It represent entity that acknowledges the response to earlier request that has come back (sender just acknowledge the permission to get connected to the remote host).**

Four primitives



- **Request –** This primitive is transferred or sent to Layer N by Layer (N+1) to just request for service.
- **Indication –** This primitive is returned by Layer N to Layer (N+1) to just advise of activation of service that is being requested or of action that is initiated by the service of Layer N.
- **Response –** This primitive is simply provided by Layer (N+1) in reply to indication primitive. It might acknowledge or complete action that is previously invoked by indication primitive.
- **Confirm –** This primitive is returned by the N^{th} layer to the requesting $(N+1)^{st}$ layer to simply acknowledge or complete action that is previously invoked by request primitive.

Parameters of Service Primitives

- **Connect. Request –** The initiating entity does this Connect.Request. It just specifies and determines machine that we want to get connected to, type of service that is being desired, and maximum size of packet or message that is used on connection.
- **Connect. Indication –** The receiver gets this Connect.Indication. It just specifies caller's identity service that we want to use like FTP and Telnet, etc., and maximum size of packets that are exchanged.
- **Connect. Response –** It just specifies whether or not it wants to accept or simply reject connection that is being requested.
- **Connect. Confirm –** It just finds out or determines what happened using the entity that is issuing the initial Connect. Request.

Primitives of Connection-Oriented Service

Primitive	Meaning
Listen	**When server is ready to accept request of incoming connection, it simply put this primitive into action. Listen primitive simply waiting for incoming connection request.**
Connect	**This primitive is used to connect the server simply by creating or establishing connection with waiting peer.**
Accept	**This primitive simply accepts incoming connection from the peer.**
Receive	**These primitive afterwards block the server. Receive primitive simply waits for incoming message.**
Send	**This primitive is put into action by the client to transmit its request that is followed by putting receive primitive into action to get the reply. Send primitive simply sends or transfer the message to the peer.**
Disconnect	**This primitive is simply used to terminate or end the connection after which no one will be able to send any of the message.**

Primitives of Connectionless Service

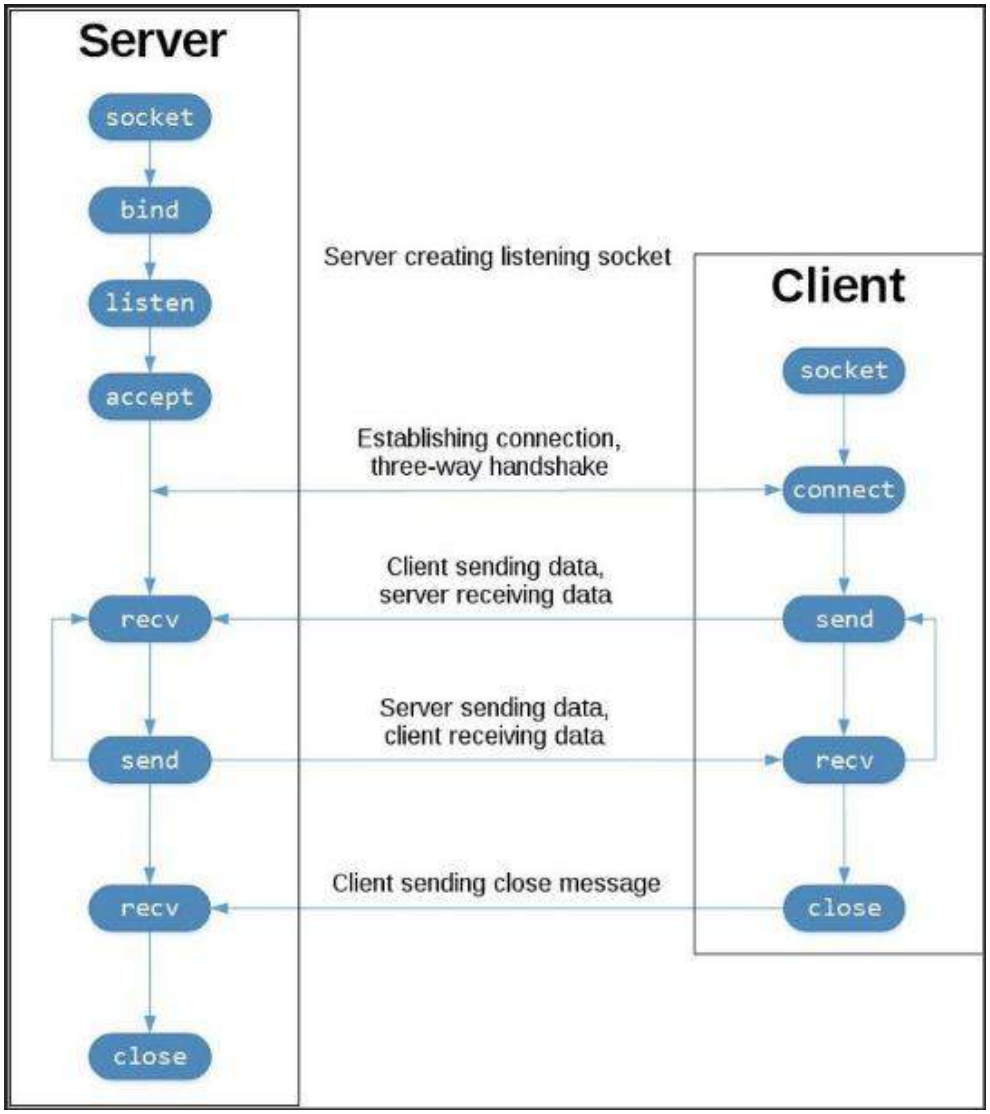
Primitive	Meaning
Unitdata	**Unitdata primitive is simply required to send packet of data or information.**
Facility, Report	**This primitive is required for getting details about the performance and working of the network such as delivery statistics or report.**

Sockets

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

The socket primitives for TCP

Flow diagram of Client – Server transaction



Similarity between data link and transport layer

- Connection establishment
- Connection release
- Error control and flow control

Difference between data link and transport layer

- At datalink layer, physical channel is used by two routers.
- At transport layer, physical channel is substituted by the entire network.

Addressing(1)

a) Addressing to which processes will able to listen for connection requests.

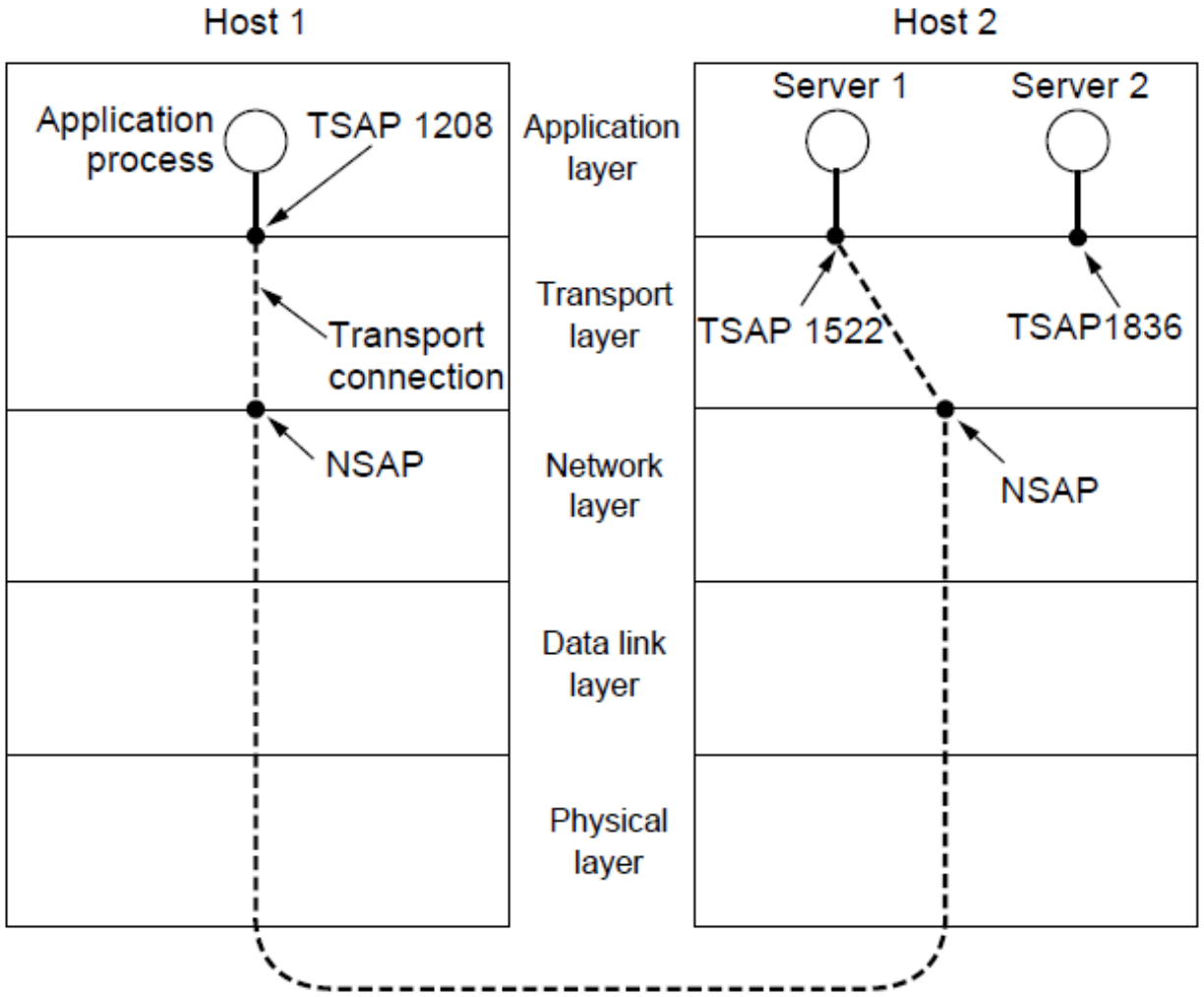
b) Those endpoints are known as ports. Particular endpoint in transport layer is referred as **Transport Service Access Point (TSAP)**.

c) The analogous endpoints(i.e., network layer address) present in the network layer are known as **Network Service Access Points (NSAP)**. E.g.: IP address.

d) **Application process**: both client and server have ability to attach themselves to a local TSAP for the purpose of setting connection to a remote TSAP.

e) These connections run through NSAPs on each host.

Addressing (2)



TSAPs, NSAPs, and transport connections

Connection Establishment (1)

Techniques to enable receiver to distinguish between retransmitted packets and packets delivered late:

A. Throwaway transport addresses.

B. Assign each connection a different identifier:
<Peer transport entity, connection entity>

Limitation: Nodes have to maintain history information indefinitely.

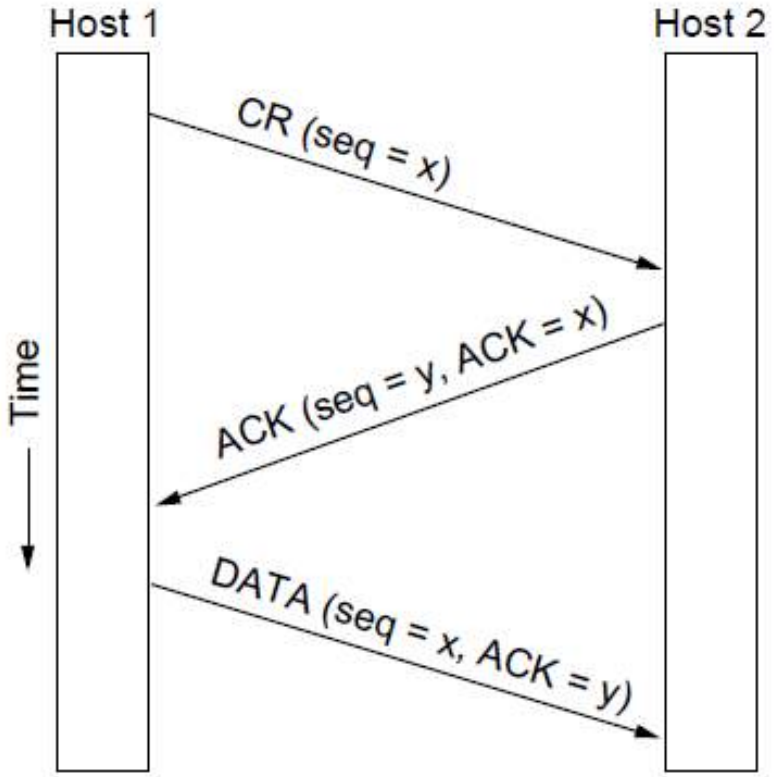
Connection Establishment (2)

Techniques for restricting packet lifetime:

A. Restricted network design-any method which avoids packets from looping
B. Putting a hop counter in each packet – decremented every time whenever the packet is forwarded.
C. Timestamping each packet.

Connection Establishment (3)

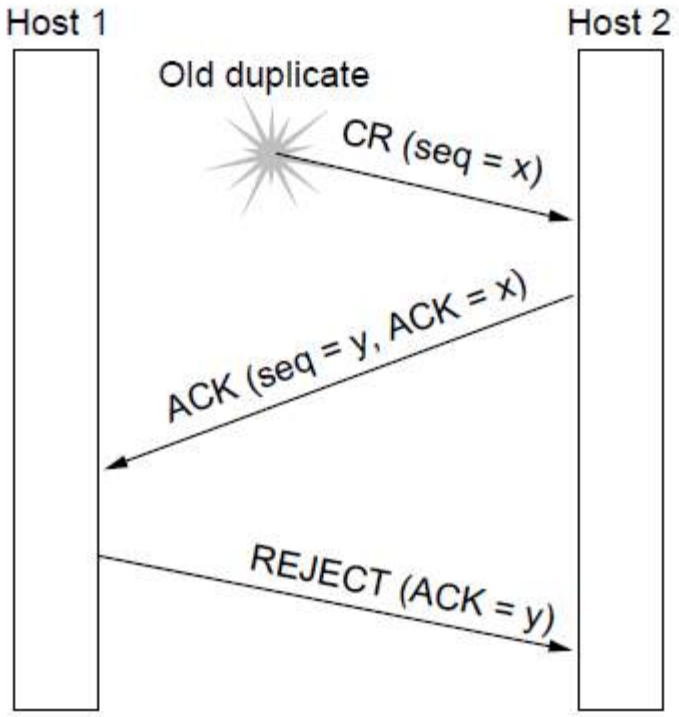
3 protocol scenarios for establishing a connection using a 3-way handshake. CR denotes CONNECTION REQUEST.



(1) Normal operation.

Connection Establishment (4)

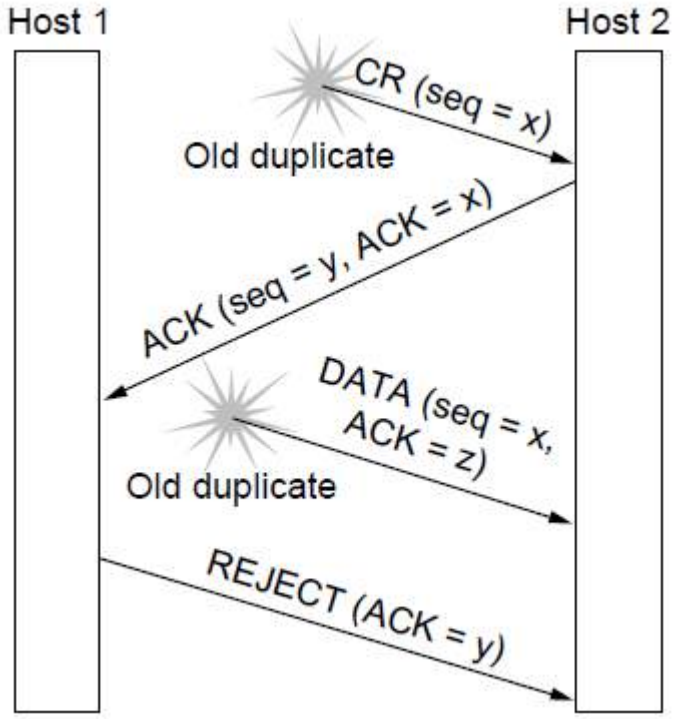
3 protocol scenarios for establishing a connection using a 3-way handshake. CR denotes CONNECTION REQUEST.



(2) Old duplicate CONNECTION REQUEST appearing out of nowhere.

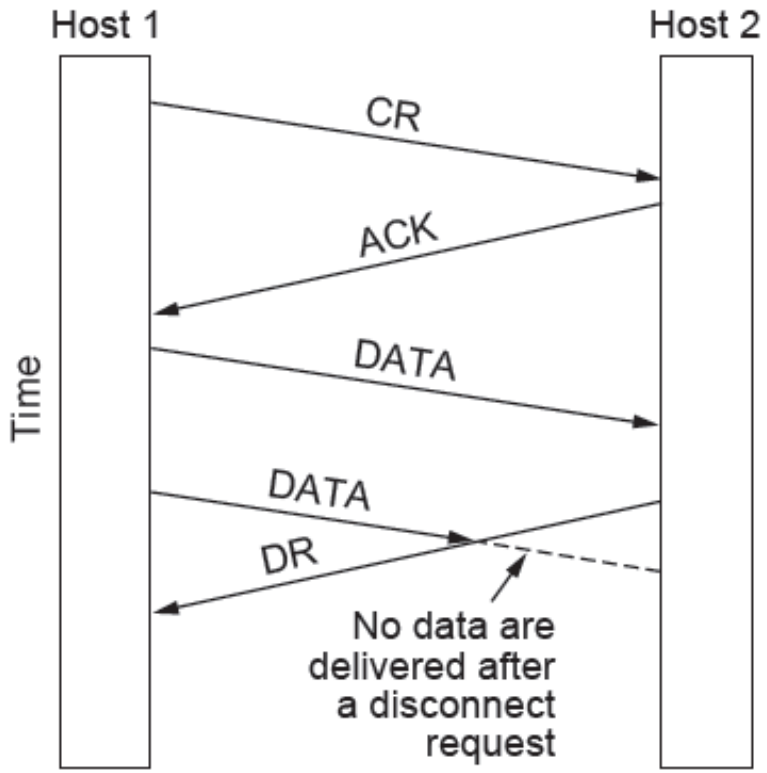
Connection Establishment (5)

3 protocol scenarios for establishing a connection using a 3-way handshake. CR denotes CONNECTION REQUEST.



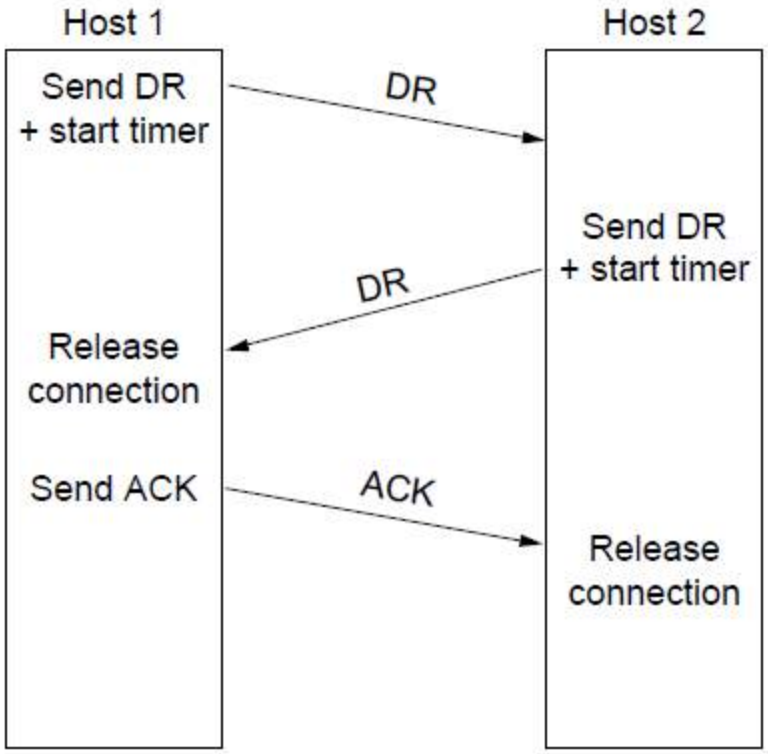
(3) Duplicate CONNECTION REQUEST and duplicate ACK.

Connection Release (1)



Abrupt disconnection with loss of data – one way release or two way release

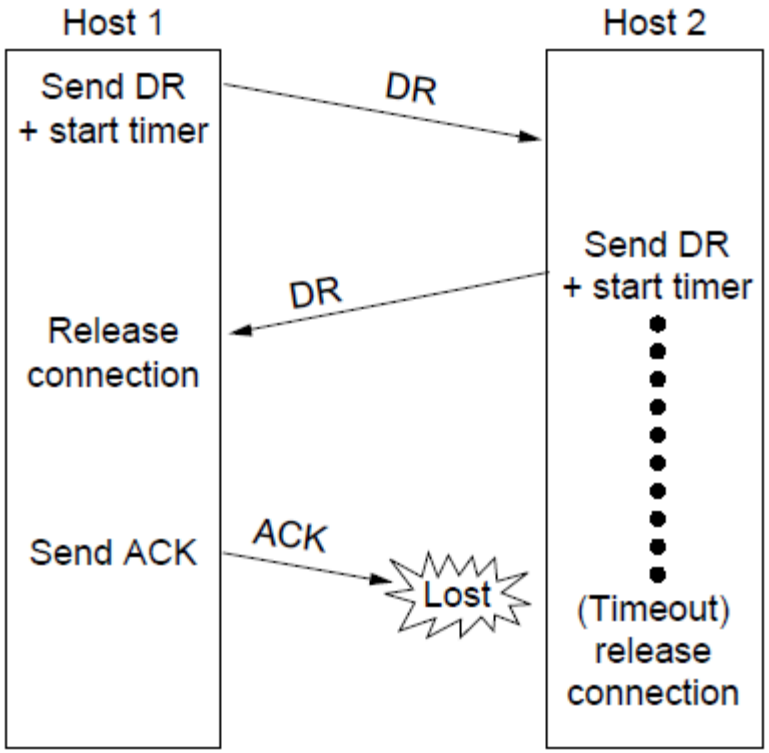
Connection Release (2)



Four protocol scenarios for releasing a connection.

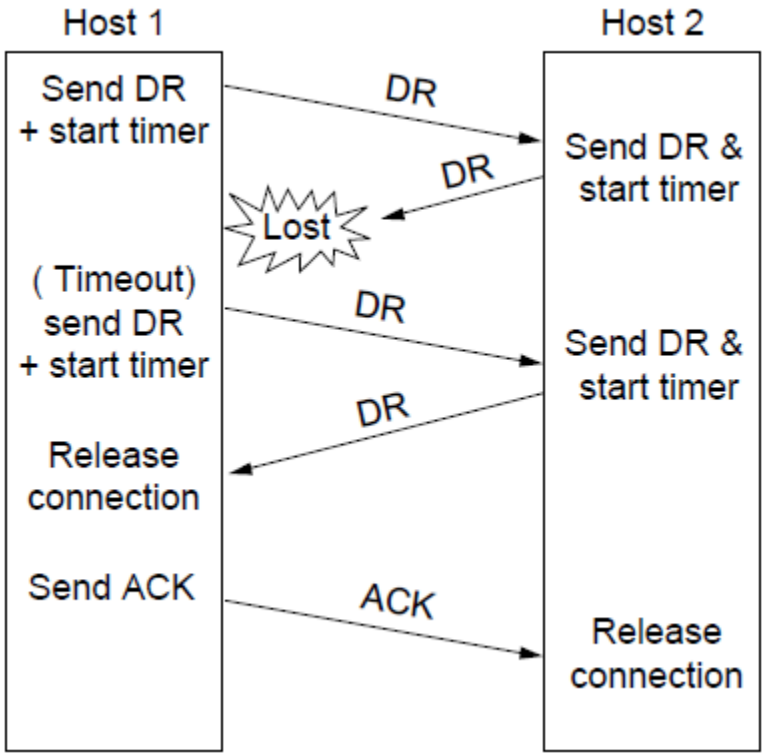
(1) Normal case of three-way handshake

Connection Release (3)



Four protocol scenarios for releasing a connection.
(2) Final ACK lost.

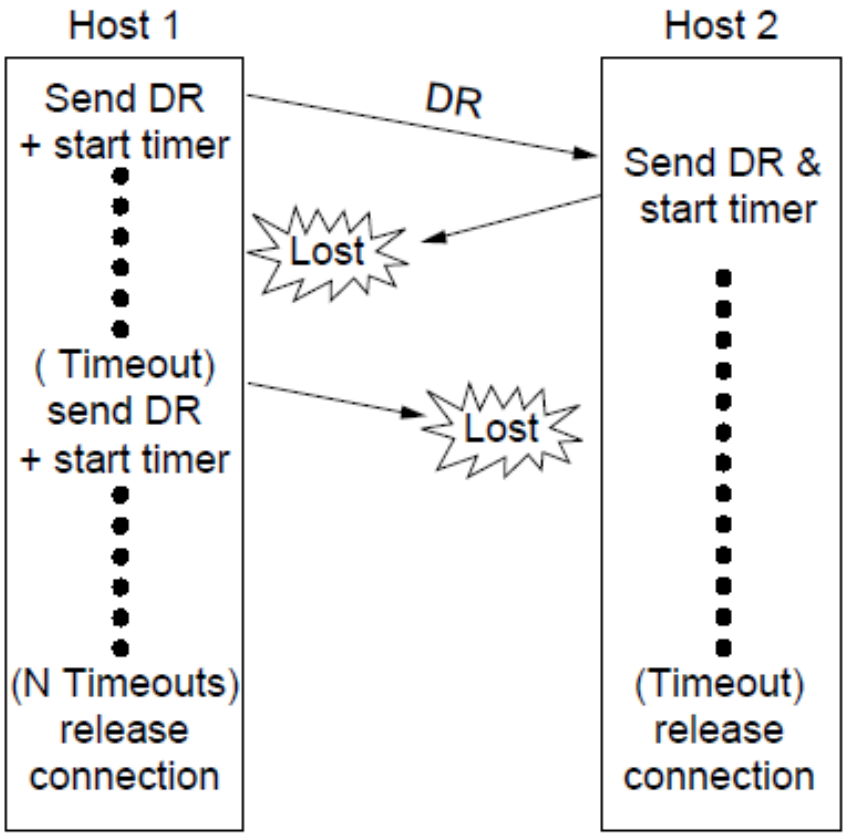
Connection Release (4)



Four protocol scenarios for releasing a connection.

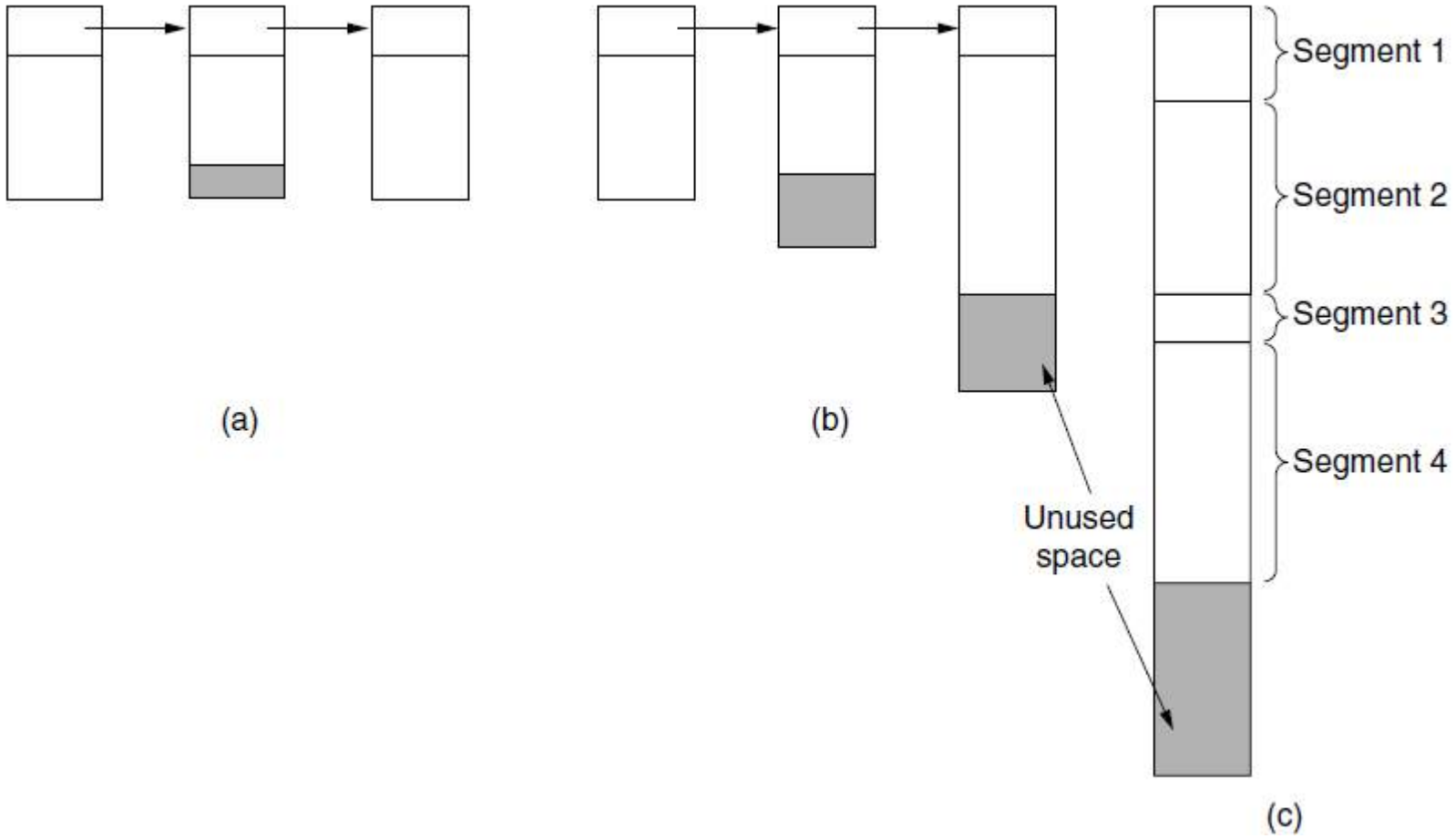
(3) Response lost

Connection Release (5)



Four protocol scenarios for releasing a connection.
(4) Response lost and subsequent DRs lost.

Error Control and Flow Control (1)



(a) Chained fixed-size buffers. (b) Chained variable-sized buffers. (c) One large circular buffer per connection.

Error Control and Flow Control (1)

a) However, if there is wide variation in TPDU size, from a few characters typed at a terminal to thousands of characters from file transfers, a pool of fixed-sized buffers presents problems. If the buffer size is chosen equal to the largest possible TPDU, space will be wasted whenever a short TPDU arrives. If the buffer size is chosen less than the maximum TPDU size, multiple buffers will be needed for long TPDUs, with the attendant complexity.

b) The advantage here is better memory utilization, at the price of more complicated buffer management.

c) This system also makes good use of memory, provided that all connections are heavily loaded, but is poor if some connections are lightly loaded.

Error Control and Flow Control (2)

a) For low-bandwidth bursty traffic, it is better not to allot buffer on connection establishment.

b) Dynamic allotment of buffer a better strategy.

c) Decouple sliding window protocol.

Error Control and Flow Control (3)

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

Dynamic buffer allocation. The arrows show the direction of transmission. An ellipsis (...) indicates a lost segment

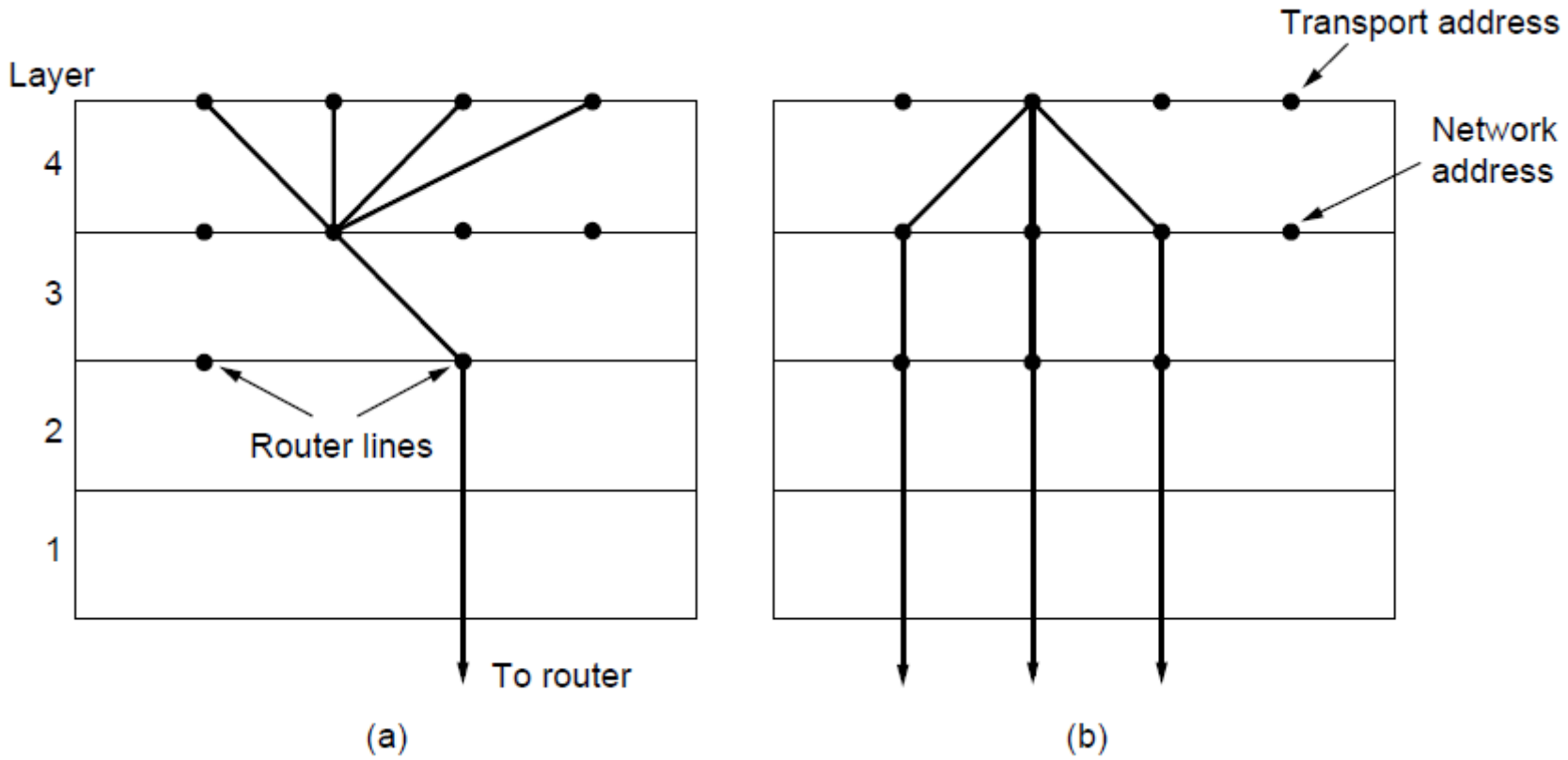
Error Control and Flow Control (4)

a) Assuming buffer size is infinite, carrying capacity becomes a bottleneck.

b) If adjacent routers can exchange at most x packets/sec, and there are k disjoint paths between a pair of hosts,

max. rate of segment-exchange between hosts = kx segments/sec

Multiplexing



(a) Multiplexing. (b) Inverse multiplexing.

Crash Recovery

Strategy used by sending host	Strategy used by receiving host: First ACK, then write			Strategy used by receiving host: First write, then ACK		
	AC(W)	AWC	C(AW)	C(WA)	W AC	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

OK = Protocol functions correctly
DUP = Protocol generates a duplicate message
LOST = Protocol loses a message

Different combinations of client and server strategies