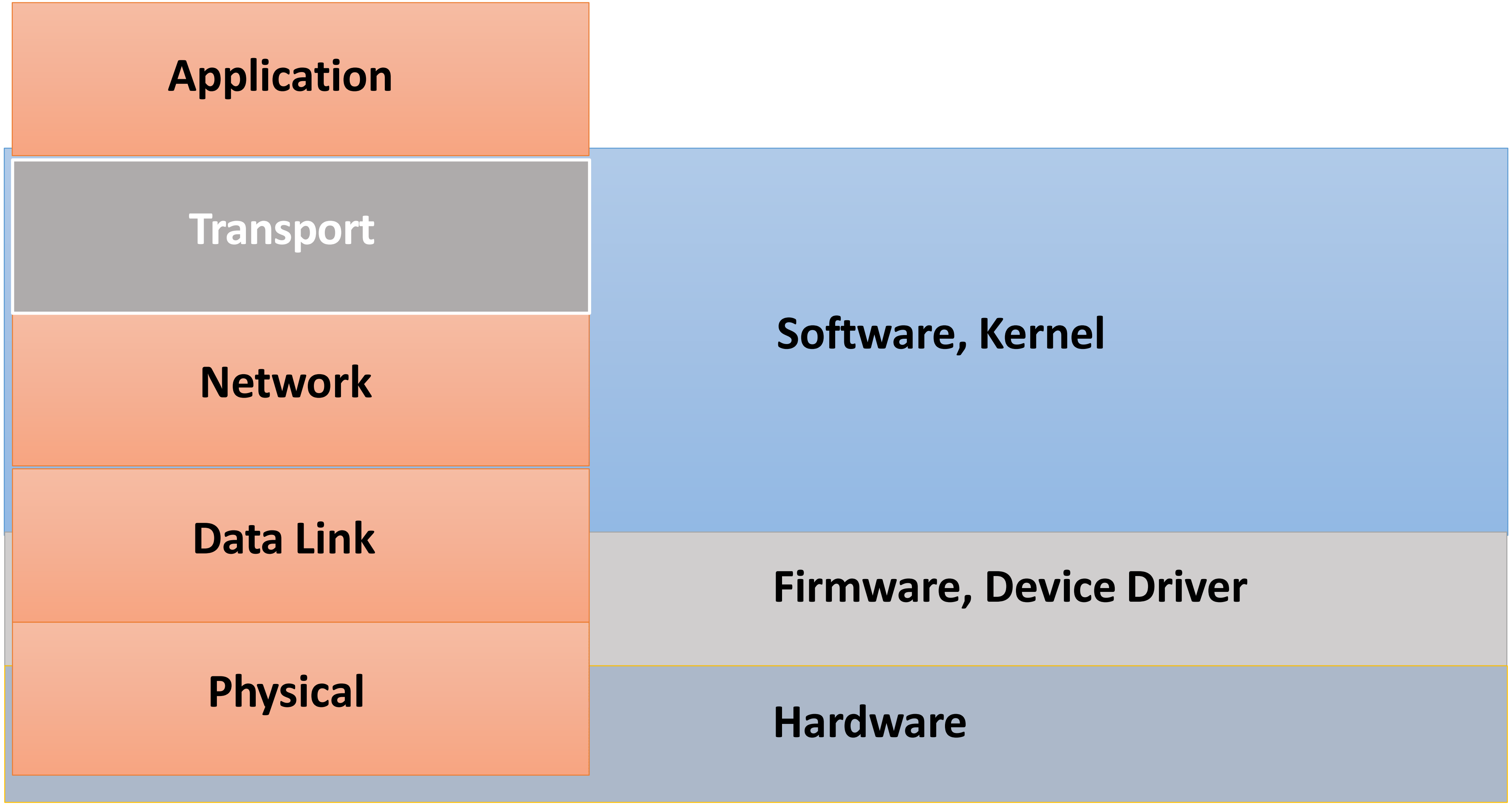
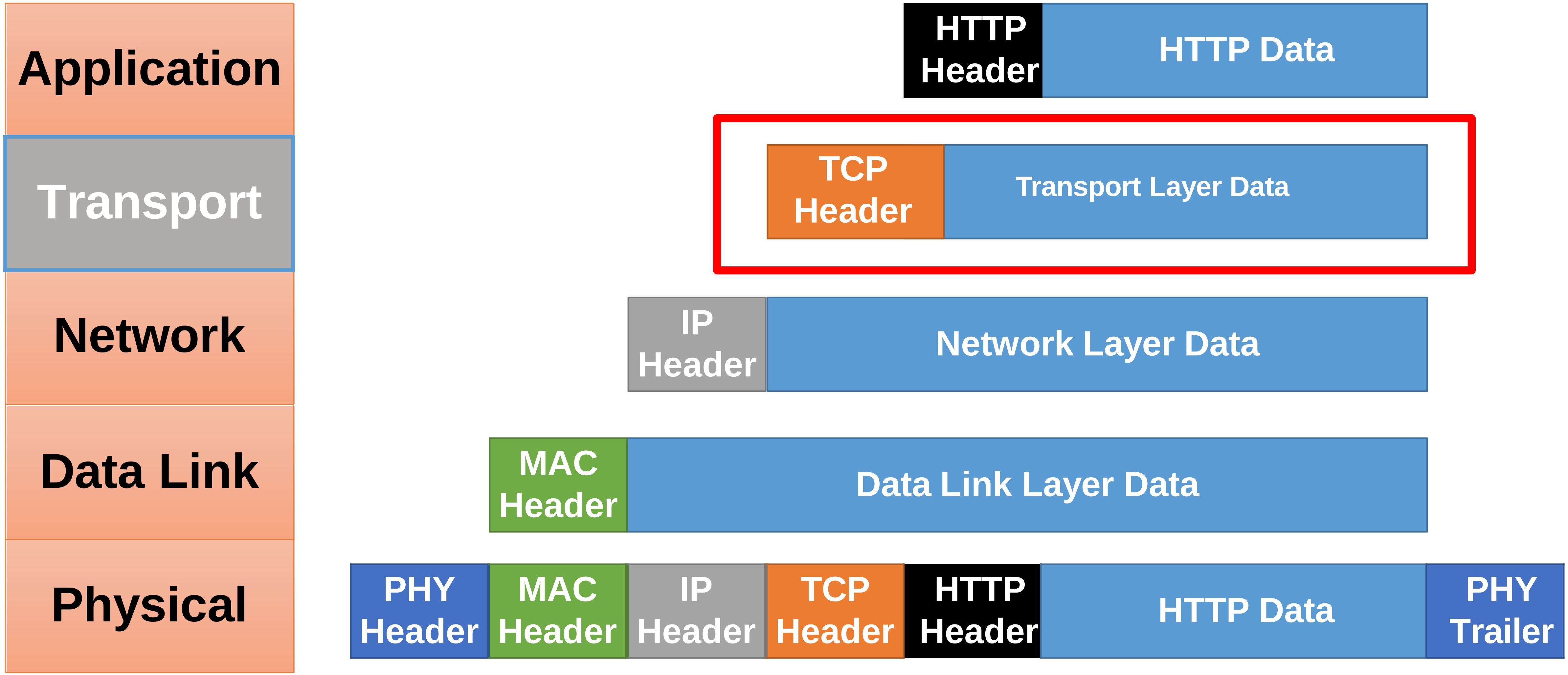
Transport Layer - I

[The Transport Layer Services]

Protocol Stack Implementation in a Host

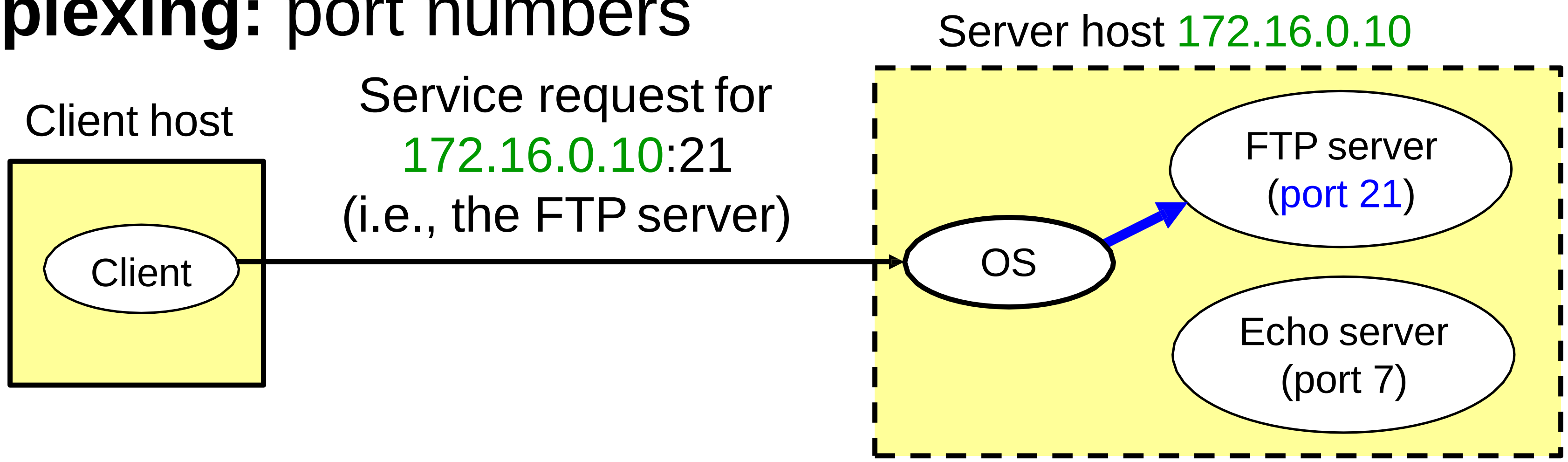


The Layered Abstraction

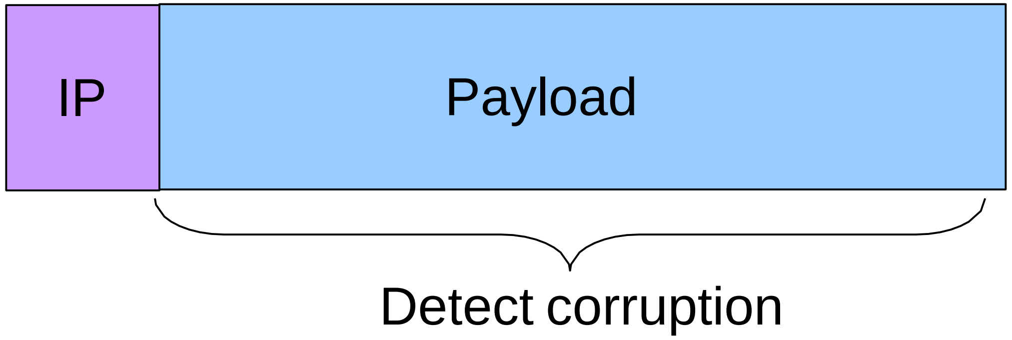


Two Basic Transport Layer Services

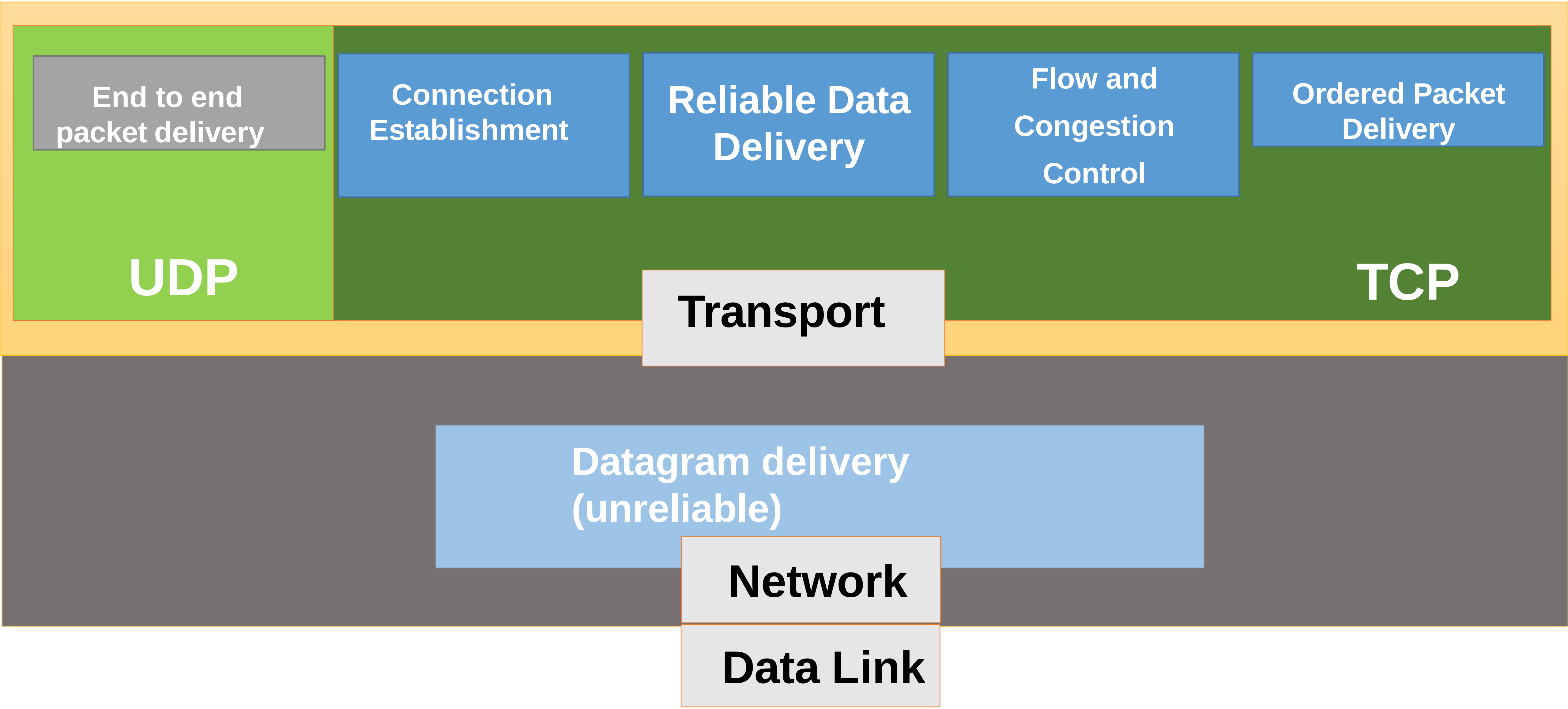
- **Demultiplexing:** port numbers



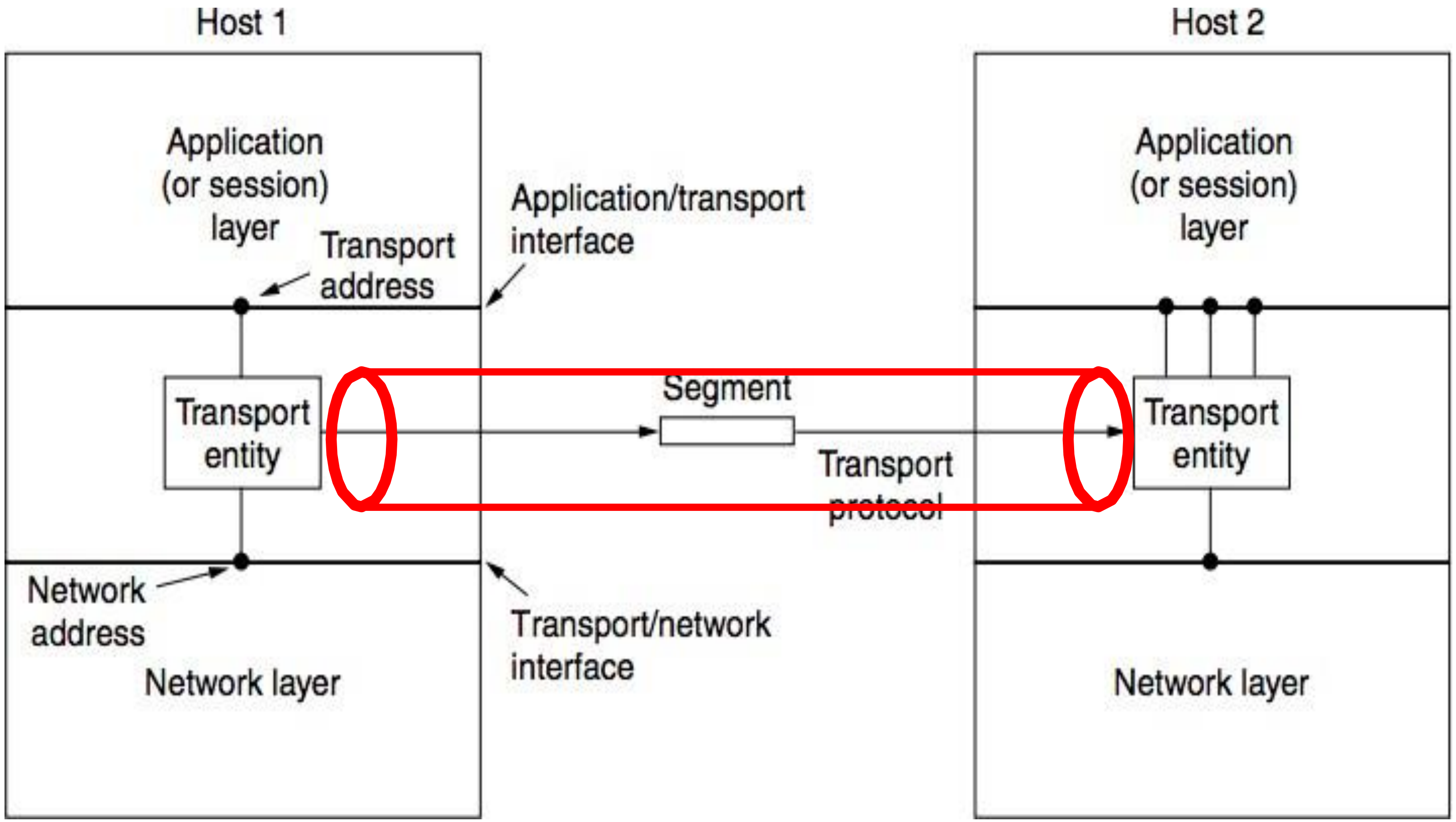
- **Error Detection:** Checksum



Transport Layer Services



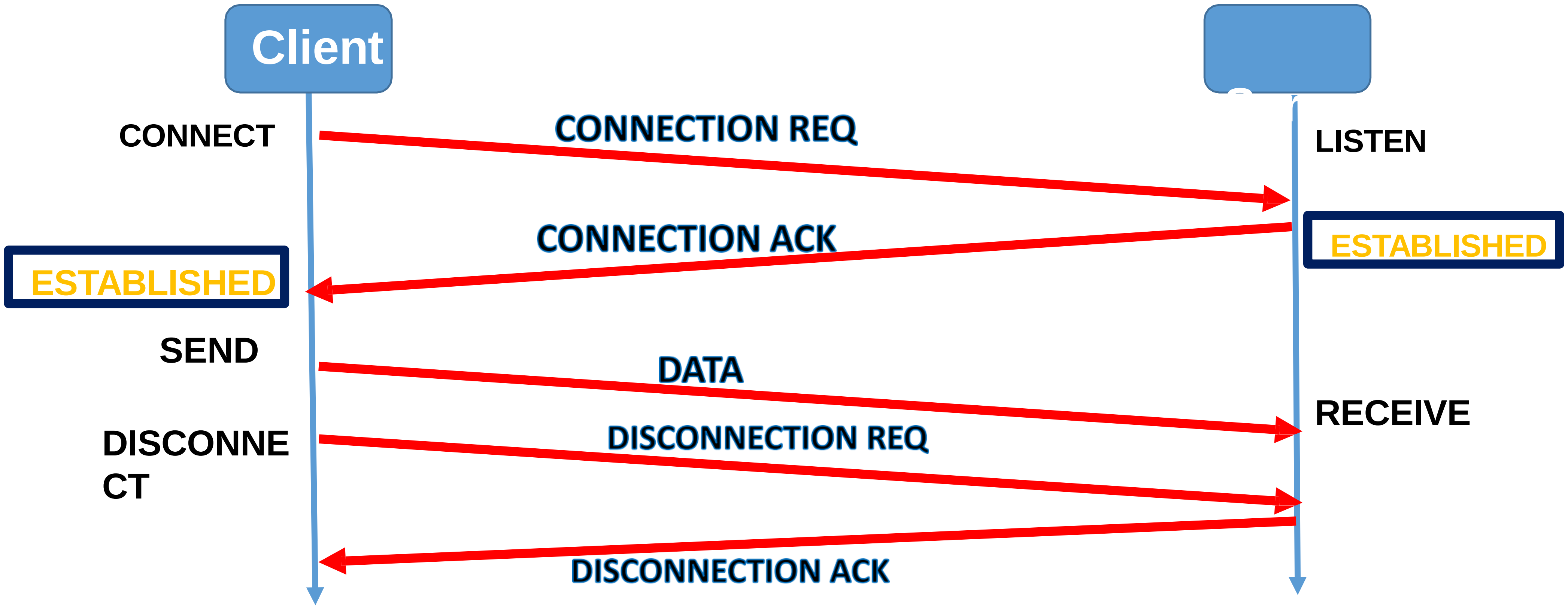
Transport Layer – Interfacing with Application and Network



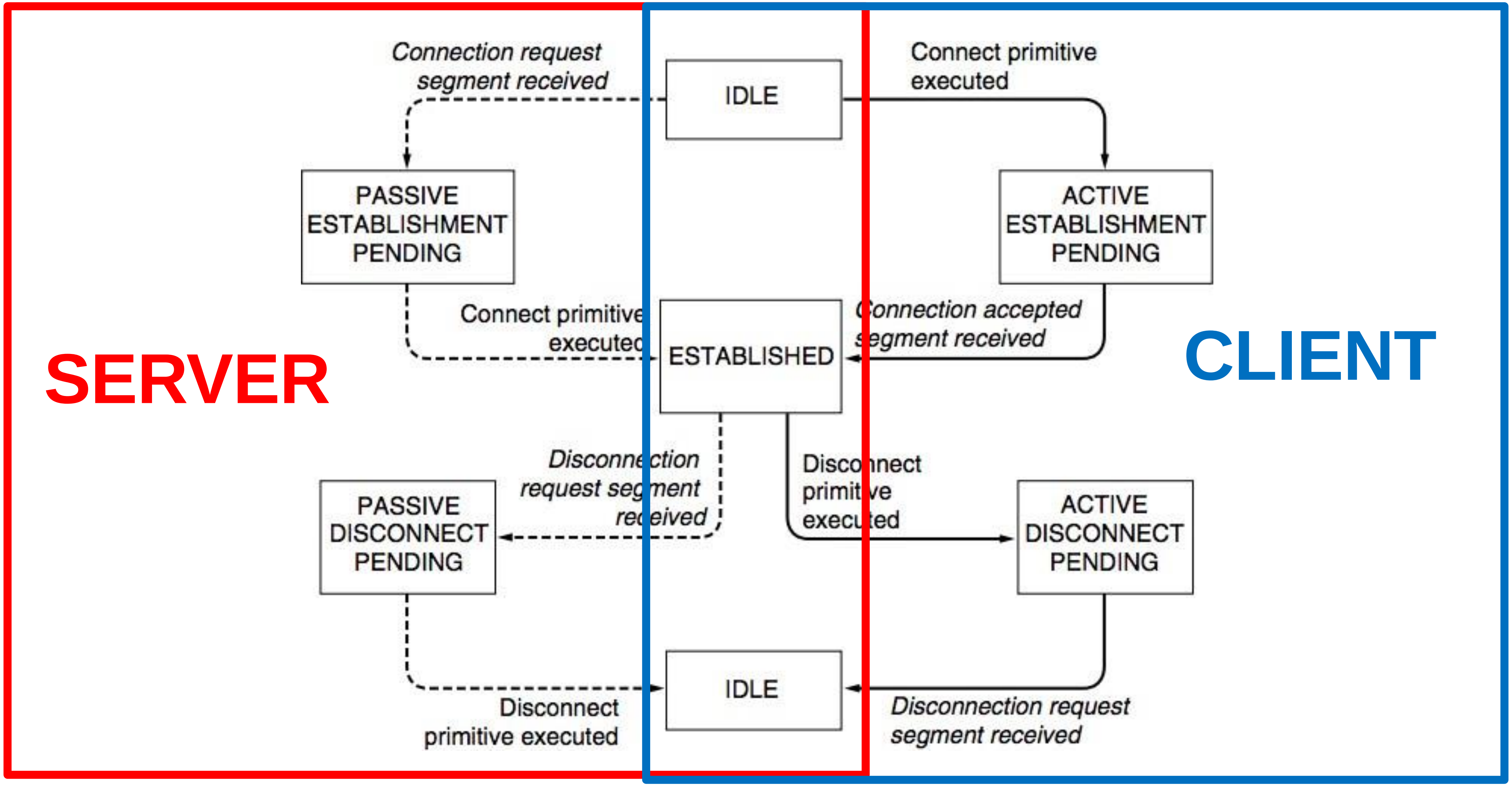
Create a logical pipe between the sender and the receiver and monitor the data transmission through this pipe

Transport Service Primitives

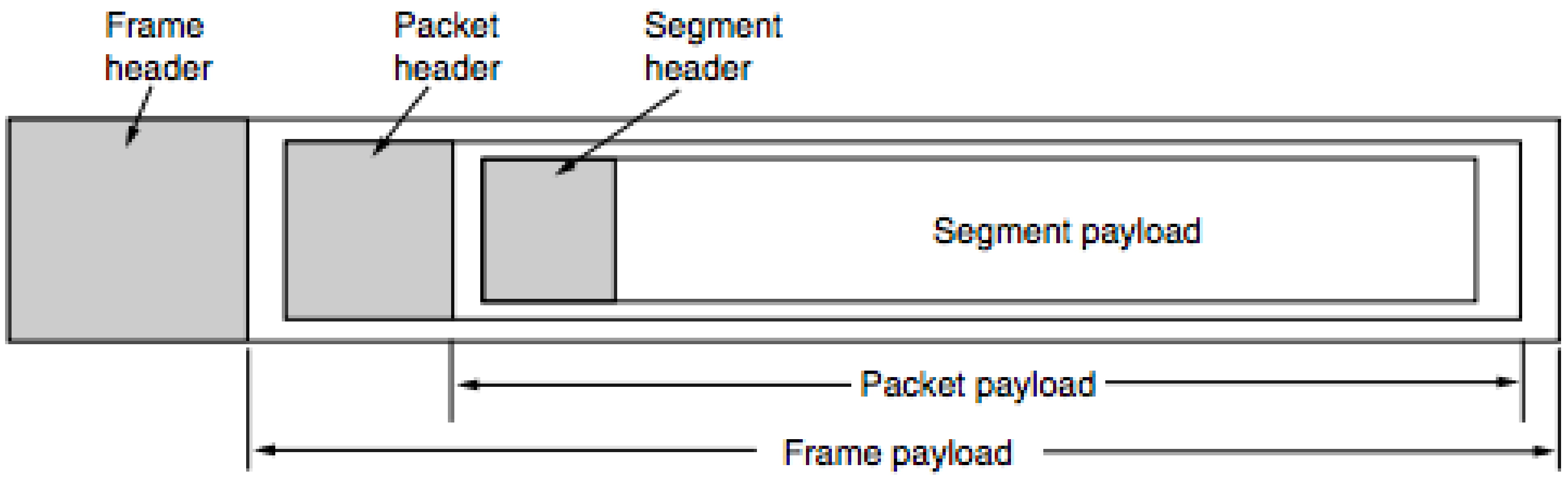
- The transport layer needs to remember the state of the pipe, so that appropriate actions can be taken. We need a stateful protocol for transport layer.



Transport layer protocols: state diagram



Segment, Packet (or Datagram) and Frame

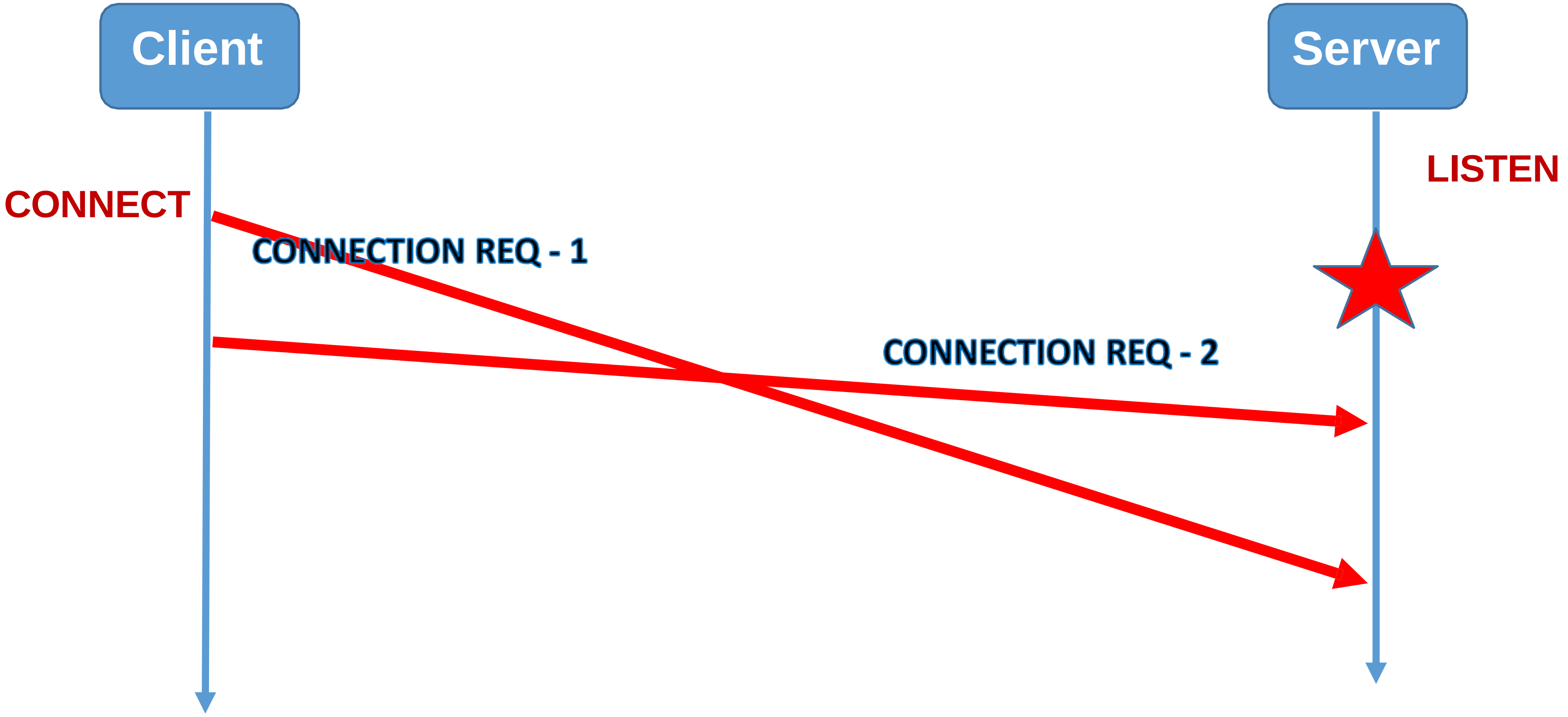


When the basic primitives for connection establishment fails ?

- Underlying network layer is unreliable! A packet can be lost, delayed, corrupted or delivered in duplicate
- How to ensure reliable delivery?
  - Through retransmission of packets of course!
  - Consider the overhead of packet-switched network here.
- The sender may think the packet is lost when actually it is not! It only got delayed due to network congestion. And the sender retransmits the packet again.

When the basic primitives for connection establishment fails

What if the server has crashed meanwhile and reinitiated the connection?



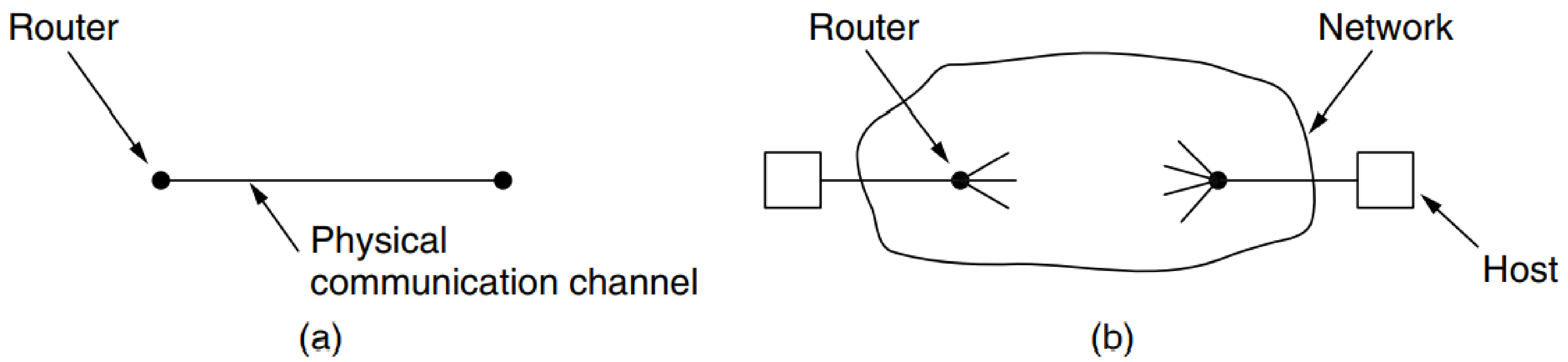
Delayed duplicates create a huge confusion in the packet switching network.

How the server is going to understand whether CONNECTION REQ-1 is a new connection request or a duplicate of the CONNECTION REQ-2?

Differences with Link Layer

- Difference 1: At the data link layer, two routers communicate directly via a physical channel, whether wired or wireless, whereas at the transport layer, this physical channel is replaced by the entire network.

- This difference has many important implications for the protocols.



(a) Environment of the data link layer. (b) Environment of the transport layer.

Differences with Link Layer

- Difference 2: Potential existence of storage capacity in the network.
- When a router sends a packet over a link, it may arrive or be lost, but it cannot be delayed.
- Difference 3: Buffering and flow control are needed in both layers, but the presence in the transport layer of a large and varying number of connections with bandwidth that fluctuates as the connections compete with each other may require a different approach than that used in the data link layer.

Handling delayed duplicates

- **Solution 1: Use Throwaway Transport Address (Port Numbers)**
  - Do not use a port number if it has been used once already – Delayed duplicate packets will never find their way to a transport process
  - Is this solution feasible?
- **Solution 2: Give each connection a unique identifier chosen by the initiating party and place it in each segment**
  - After each connection is released, each transport entity can update a table listing obsolete connections as (peer transport entity, connection identifier) pairs.
  - Whenever a connection request comes in, it can be checked against the table to see if it belongs to a previously released connection.
  - Can you see any problem in this approach?
  - History must persist, width of sequence numbers: (wrap around), crash history lost

Handling delayed duplicates

- **Solution 3: Rather than allowing packets to live forever within the network, devise a mechanism to kill off aged packets that are still hobbling about (Restrict the packet lifetime) –** Makes it possible to design a feasible solution.

- Three ways to restrict packet lifetime to a known maximum:
  - Restricted Network Design – Prevents packets from looping (bound the maximum delay including congestion). Difficult !!
  - **Putting a hop count in each packet – initialize to a maximum value and decrement each time the packet traverses a single hop (most feasible implementation).**
  - Timestamping each packet – define the lifetime of a packet in the network, need time synchronization across each router.

- **Design Challenge: We need to guarantee not only that a packet is dead, but also that all acknowledgements to it are also dead**

Handling delayed duplicates

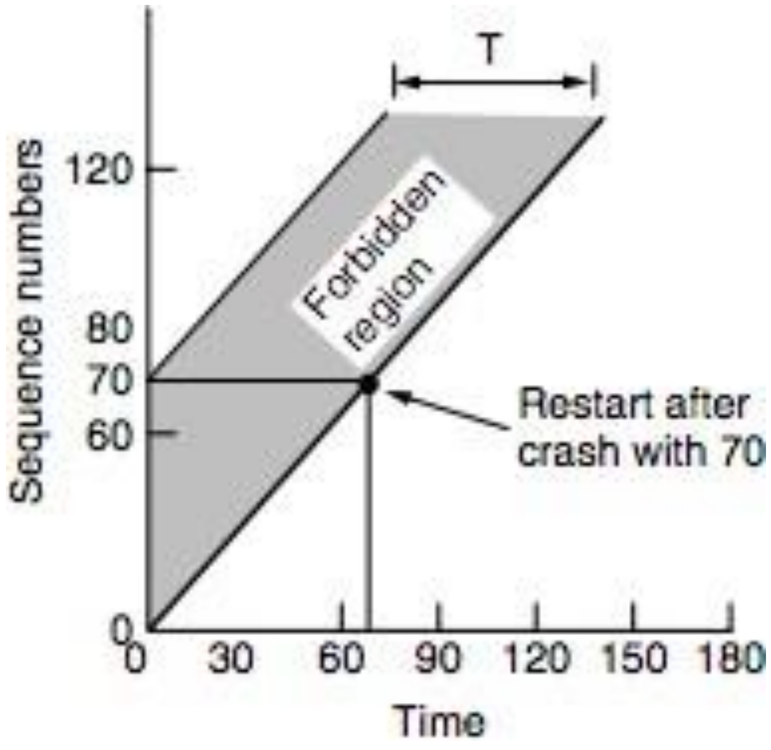
- Let us define a maximum packet lifetime T – If we wait a time T secs after a packet has been sent, we can be sure that all traces of it (packet and its acknowledgement) are now gone.
- The heart of the method is for the source to label segments with sequence numbers that will not be reused within T secs.
- The period, T, and the rate of packets per second determine the size of the sequence numbers.
- In this way, only one packet with a given sequence number may be outstanding at any given time.
- To handle the problem of a machine losing all memory of where it was after a crash, one possibility is to require transport entities to be idle for T secs after a recovery.
- The idle period will let all old segments die off, so the sender can start again with any sequence number.
- In a complex internetwork, T may be large, so this strategy is unattractive.

Handling delayed duplicates

- Instead, Tomlinson proposed equipping each host with a time-of-day clock. The clocks at different hosts need not be synchronized.
- Each clock is assumed to take the form of a binary counter that increments itself at uniform intervals.
- Furthermore, the number of bits in the counter must equal or exceed the number of bits in the sequence numbers.
- Last, the clock is assumed to continue running even if the host goes down.

Sequence Number

- When a connection is set up, the low-order *k* bits of the clock are used as the *k*-bit initial sequence number
- The sequence space should be so large that by the time sequence numbers wrap around, old segments with the same sequence number are long gone.
- Once both transport entities have agreed on the initial sequence number, any sliding window protocol can be used for data flow control.
- This window protocol will correctly find and discard duplicates of packets after they have already been accepted.

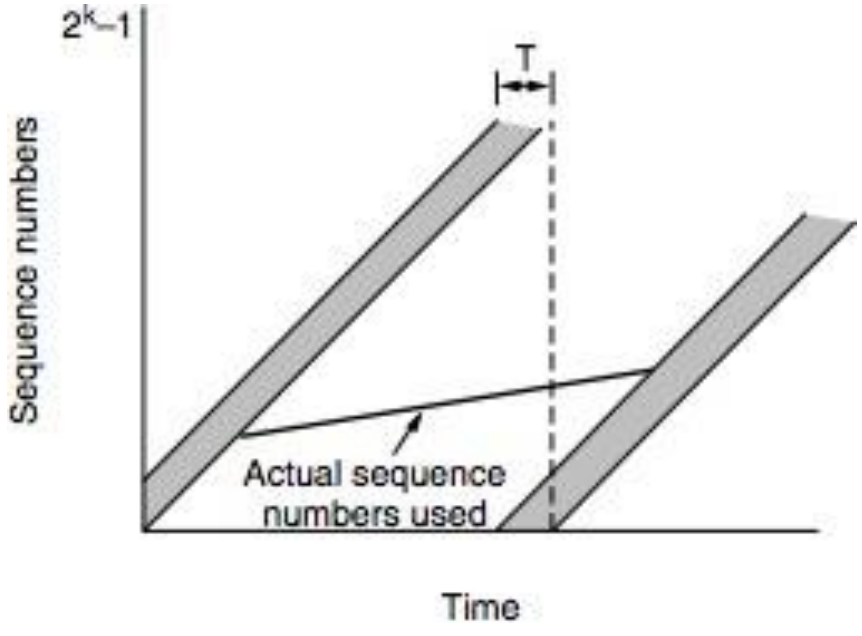


linear relation between time and initial sequence numbers

How to ensure that packet sequence numbers are out of the Forbidden Region?

Two possible source of problems:

- A host sends too much data too fast on a newly opened connection
- The data rate is too slow that the sequence number for a previous connection enters the forbidden region for the next connection

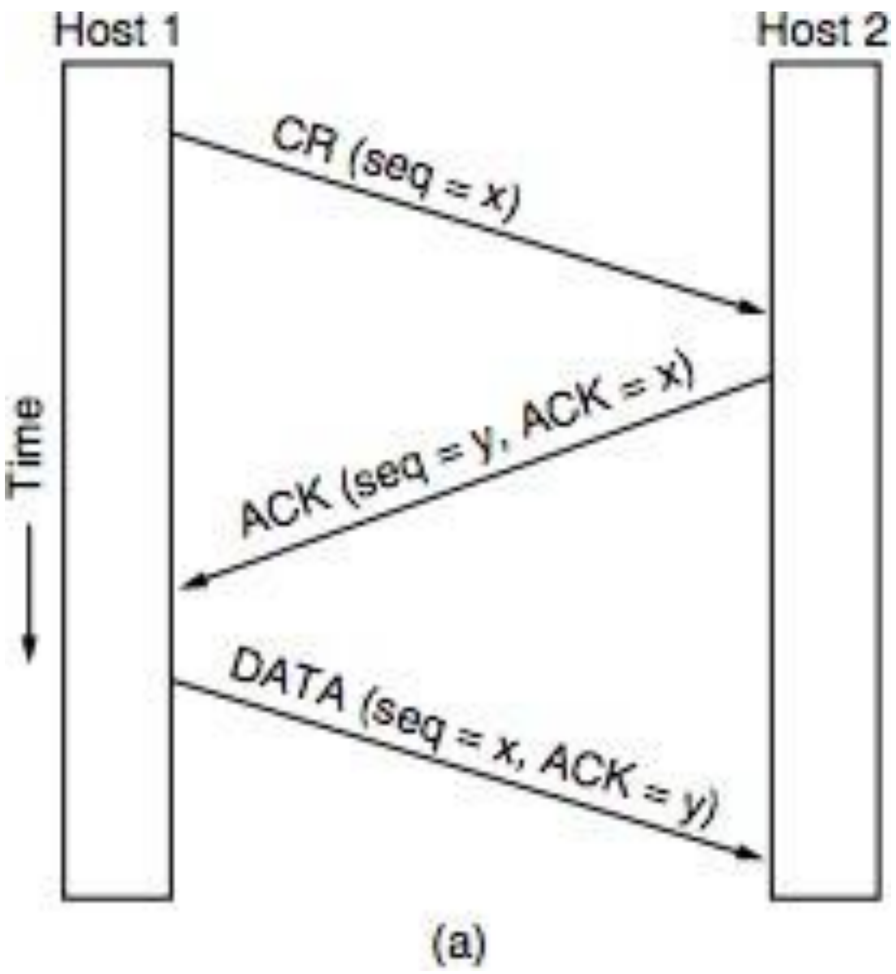


- The maximum data rate on any connection is one segment per clock tick
  - Clock ticks (inter-packet transmission duration) is adjusted based on the sequences acknowledged – **ensure that no two packets are there in the network with same sequence number**
  - **We call this mechanism as self-clocking (used in TCP)**
  - Ensures that the sequence numbers do not warp around too quickly (RFC 1323)

Further challenge:

- The clock-based method solves the problem of not being able to distinguish delayed duplicate segments from new segments.
- **But, we do not remember sequence number at the receiver:** Use a **three way handshake** to ensure that the connection request is not a repetition of an old connection request
    - The individual peers validate their own sequence number by looking at the acknowledgement (ACK)
    - **Positive synchronization among the sender and the receiver**

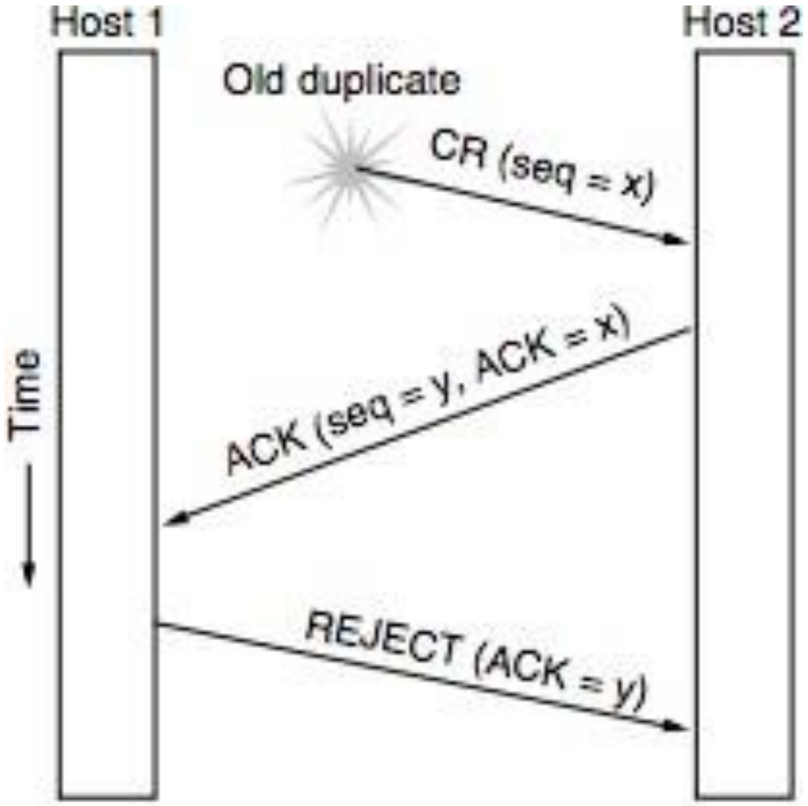
Three Way Handshake



- By looking at the ACK, Host 1 ensures that Sequence number x does not belong to the forbidden region of any previously established connection

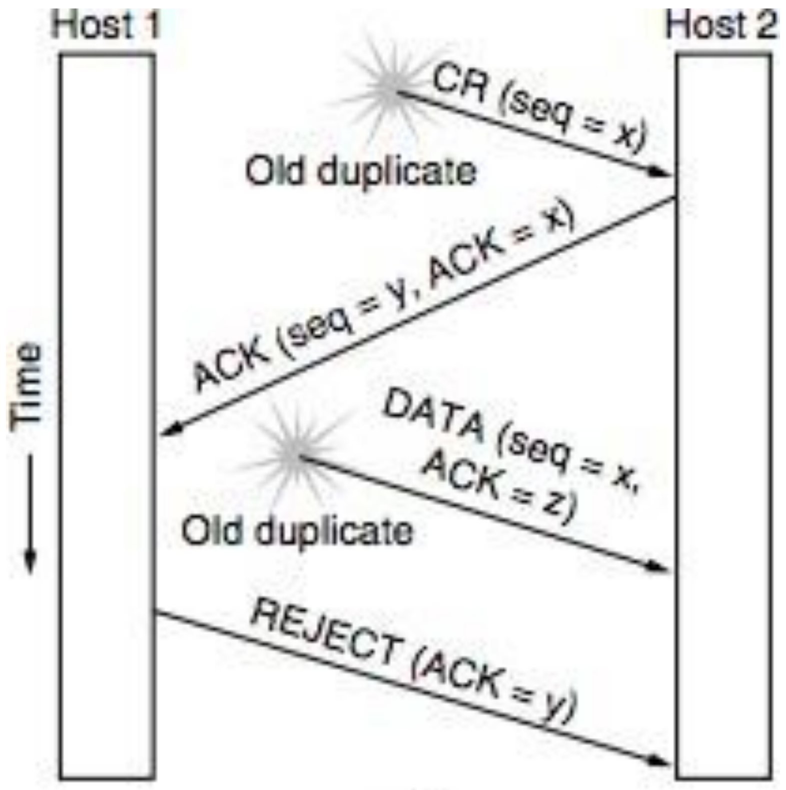
- By looking at the ACK in DATA, Host 2 ensures that sequence number y does not belong to the forbidden region of any previously established connection

Three Way Handshake – Handling Delayed Duplicate



(b)

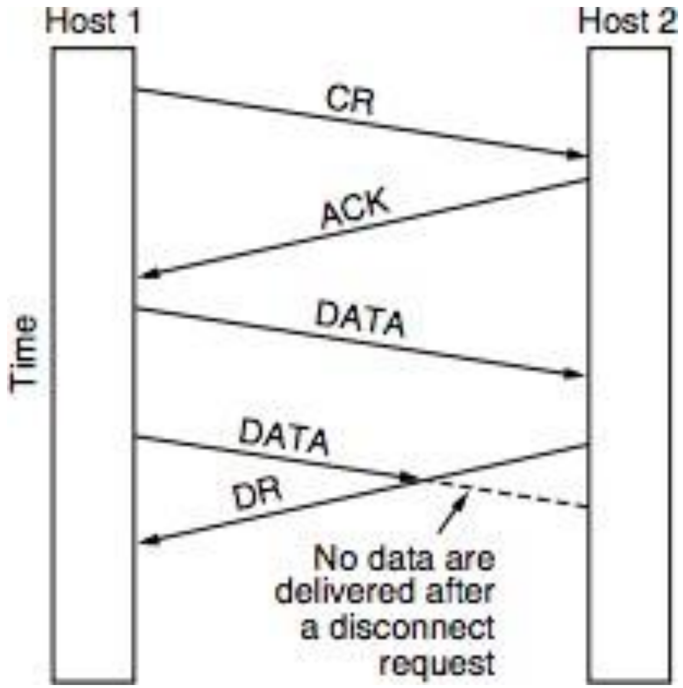
CONNECTION REQUEST is a Delayed Duplicate



(c)

CONNECTION REQUEST and ACKNOWLEDGEMENT both are Delayed Duplicates

Connection Release: Asymmetric Release

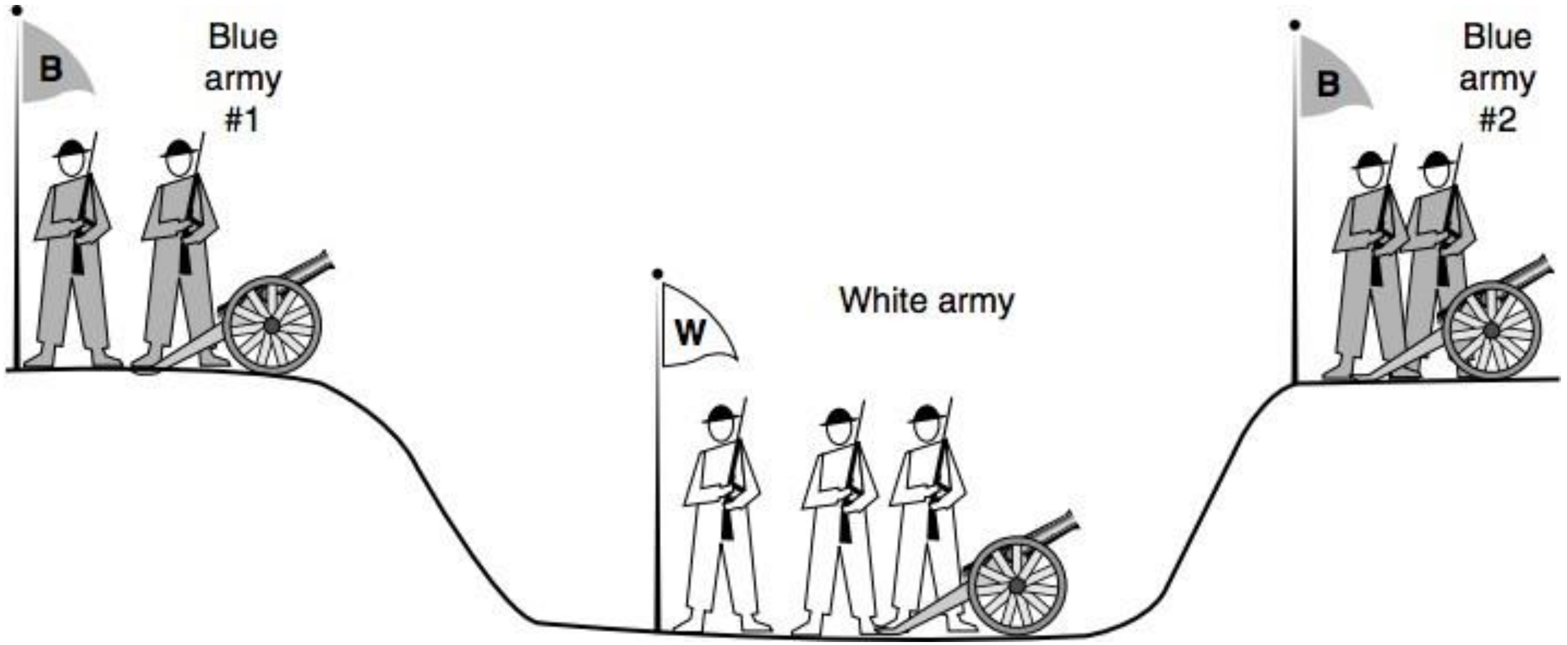


- when one side breaks connection abruptly, the connection is broken
- This may result in data loss

Connection Release: Symmetric Release

- Treats the connection as two separate unidirectional connections and requires each one to be released separately
- Does the job when each process has a fixed amount of data to send and clearly knows when it has sent it.
- What can be a protocol for this?
  - **Host 1: “I am done”**
  - **Host 2: ”I am done too”**
- **Does this protocol work good always?**

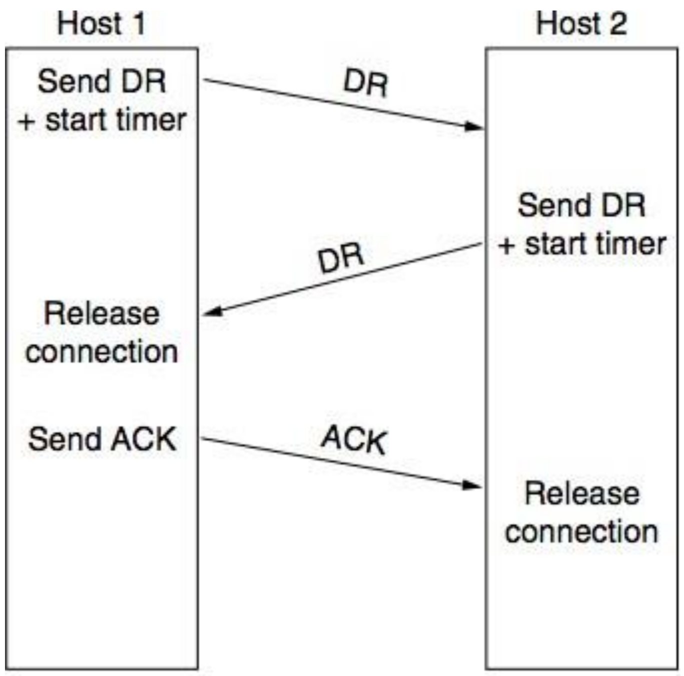
The Two Army Problem



No protocol exists to solve this

Let every party take individual decision

Connection Release

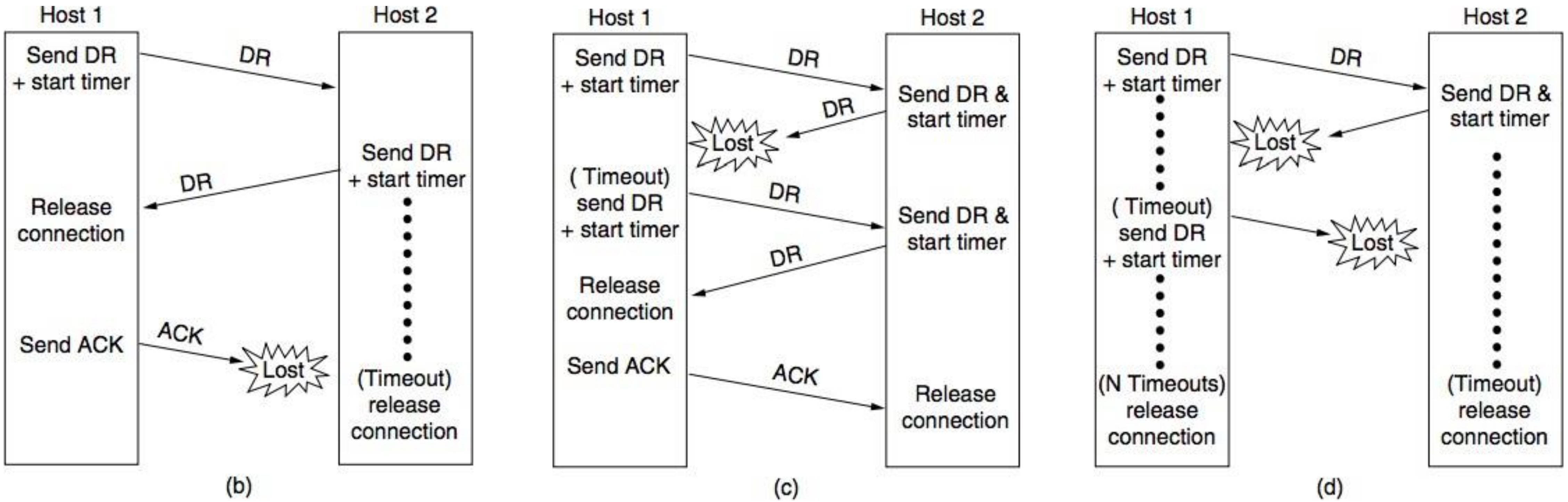


(a)

- Normal release sequence, initiated by transport user on Host 1
- DR=Disconnect Request
- Both DRs are ACKed by the other side

Connection Release

Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell



Final ACK lost, Host 2 times out

Lost DR causes retransmissions

Extreme: Many lost DRs cause both hosts to timeout

Next we will look into Transport Layer services for Flow Control and Congestion Control