Class Notes

Application Layer

(1) Application layer protocols have been using the services of the transport layer (TCP or UDP) and we mostly considered them as **black boxes**. Now in chapter three, we will open those black boxes to see how transport layer does its work (process-to-process delivery).

Transport Layer

(1) Application components (for example, client and server) had the perception that, when they use TCP, they have a bi-directional pipe between them. What any of the two parties inserts on one end, emerges on the other end, in the exact same order, and exact same bytes. If application used UDP, if client didn't receive the reply in a reasonable time, client either re-sends the request or might choose a different server (as we saw in the case of DNS)
(2) Trasport layer protocols use the services of the layer below (the network layer). Network layer gives a **best-effort service** to the transport layer. Network layer provides host-to-host delivery where we assume that each host has a unique identifier (called an IP in the case of the Internet).
(3) Transport layer extends the host-to-host model to process-to-process communication where source process is on the source host, and the destination process is at the destination host. Transport layer does process-to-process communication using multiplexing and demultiplexing.
(4) Quirks of the packet-switched network and best-effort service provided by the IP
    a. Packets can get lost
    b. Packet data might corrupt (due to noise or due to malicious activity)
    c. Packets might reach the destination out of the sender's order
    d. Packets might be delayed (hard to give any tight guarantees on end-to-end latency and throughput)
    e. Duplicate packets might arrive at the destination (especially when sender retires because it didn't receive the response to its first request)
(5) What issues a transport layer can solve:
    a. Lost, corrupted, out-of-order packets
    b. Good network citizen (congestion control)
    c. Fair to all flows (fairness)
    d. Not overwhelming the end host (flow control)
(6) What issues a transport layer cannot solve:
    a. It is hard to provide any delay, or throughput guarantees if the underlying network is not providing it.
(7) Terminology:
    a. Segments (transport layer messages)
    b. At time UDP-based messages are called UDP-datagrams. Your textbook uses the word segment for both TCP and UDP based messages.

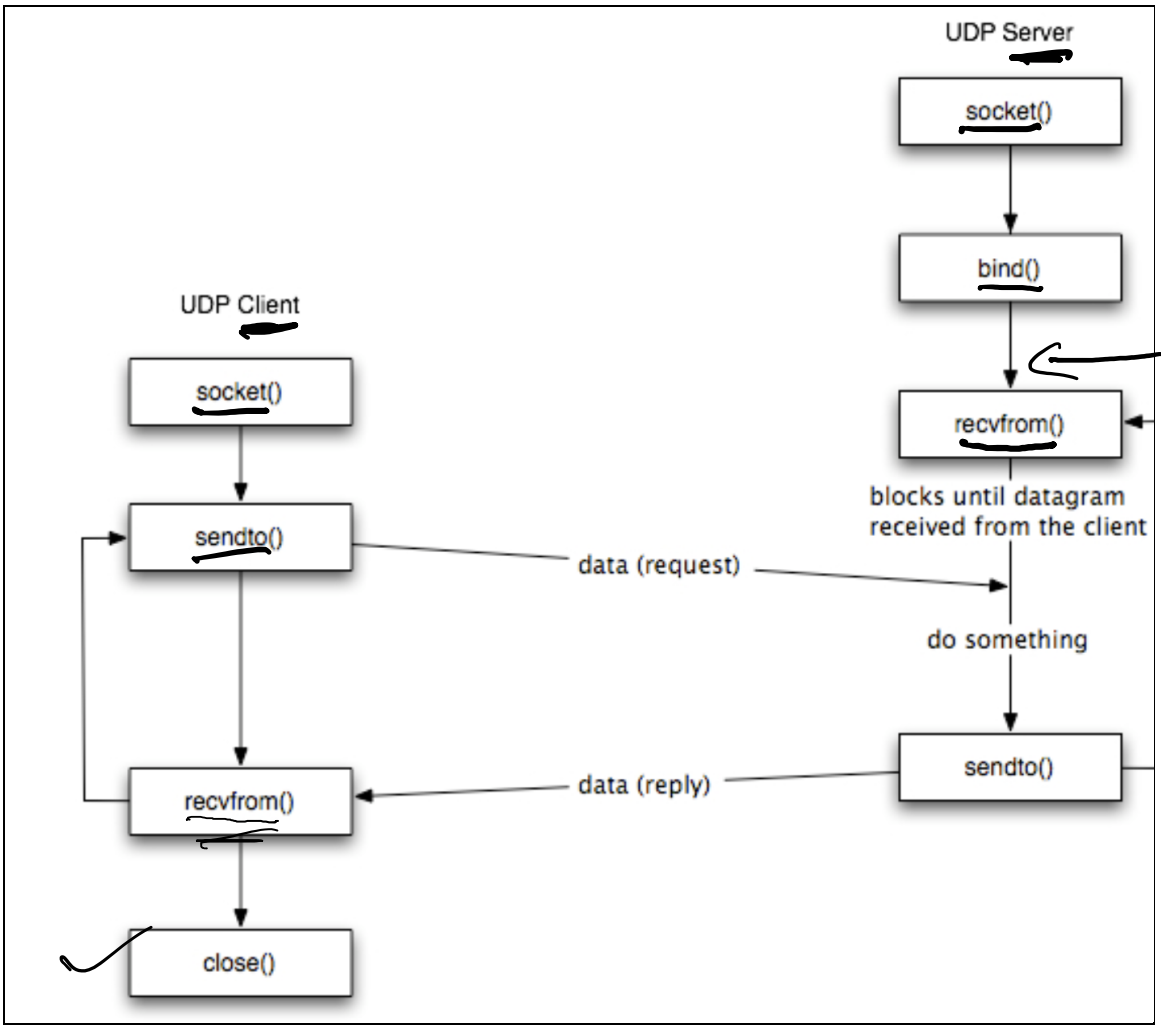
(8) UDP is connectionless while TCP is connection-oriented protocols. There are differences how they do multiplex and demultiplexing.

a. "The job of delivering the data in a transport-layer segment to the correct socket is called **demultiplexing**"

b. "The job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information (that will later be used in demultiplexing) to create segments, and passing the segments to the network layer is called **multiplexing.**"

Aside: Every protocol layer that is used by multiple entities at the upper layer, needs some form of multiplexing and demultiplexing. Food for thought: We said HTTP protocol is used by many applications. What is the multiplexing and demultiplexing keys at HTTP level?

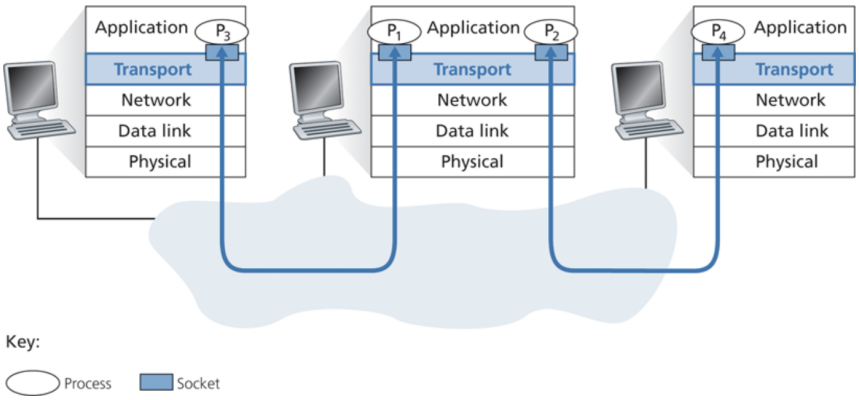
(9) UDP socket workflow:



Source:
https://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html#:~:text=The%20steps%20of%20establishing%20a,recvfrom()%20and%20sendto().

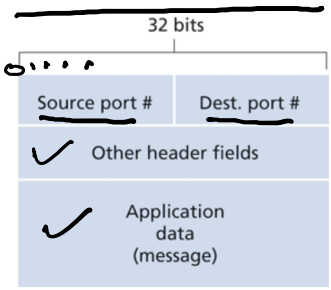
(10)Transport layer multiplexing and demultiplexing

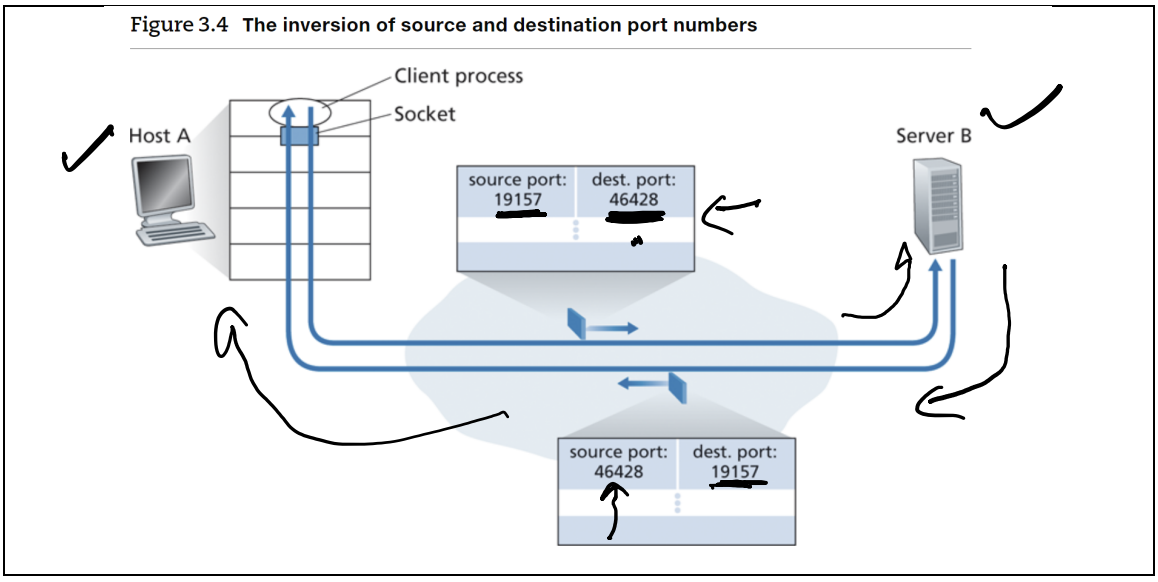
Figure 3.2 **Transport-layer multiplexing and demultiplexing**



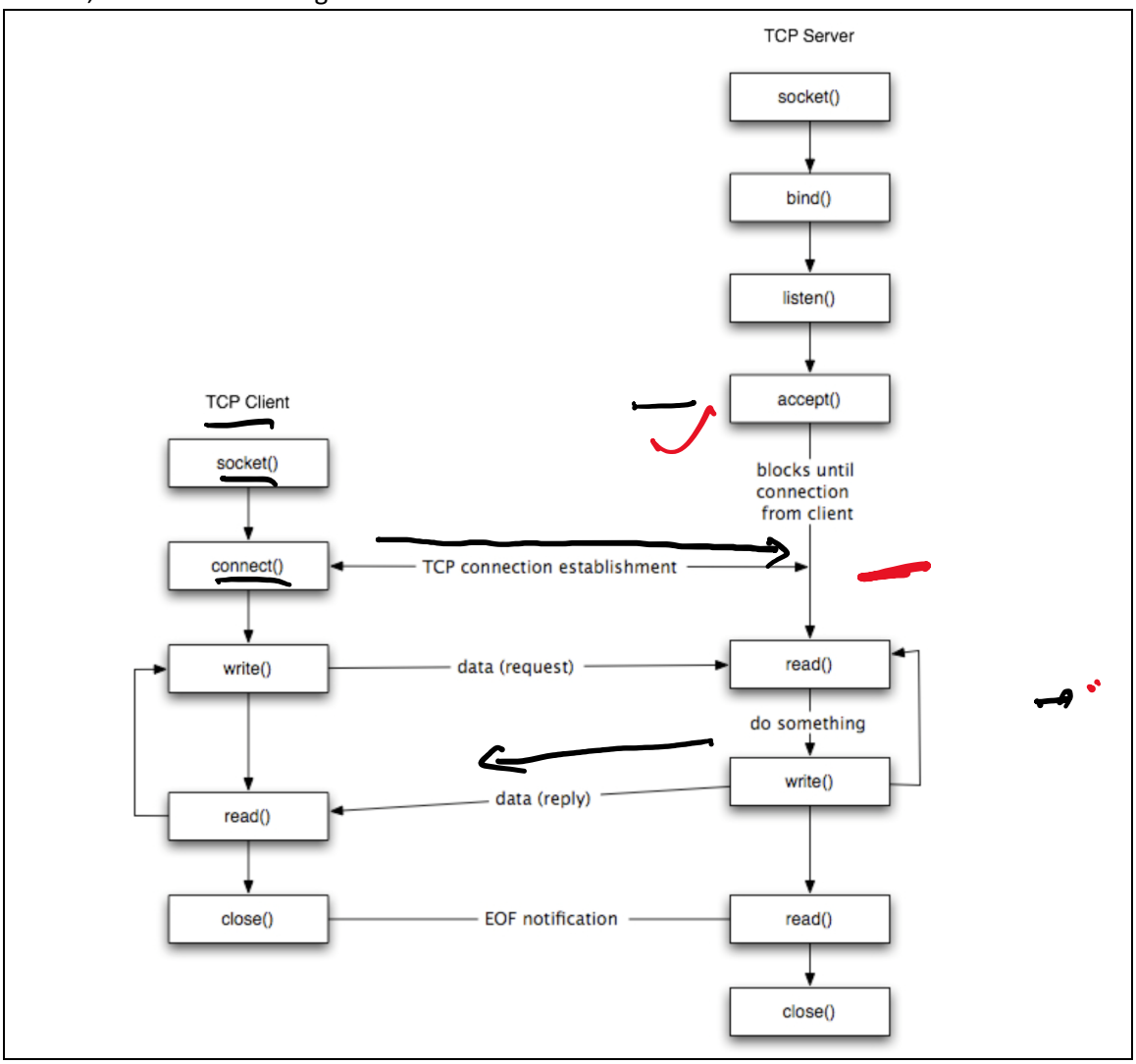
(11)For UDP

Figure 3.3 **Source and destination port-number fields in a transport-layer segment**





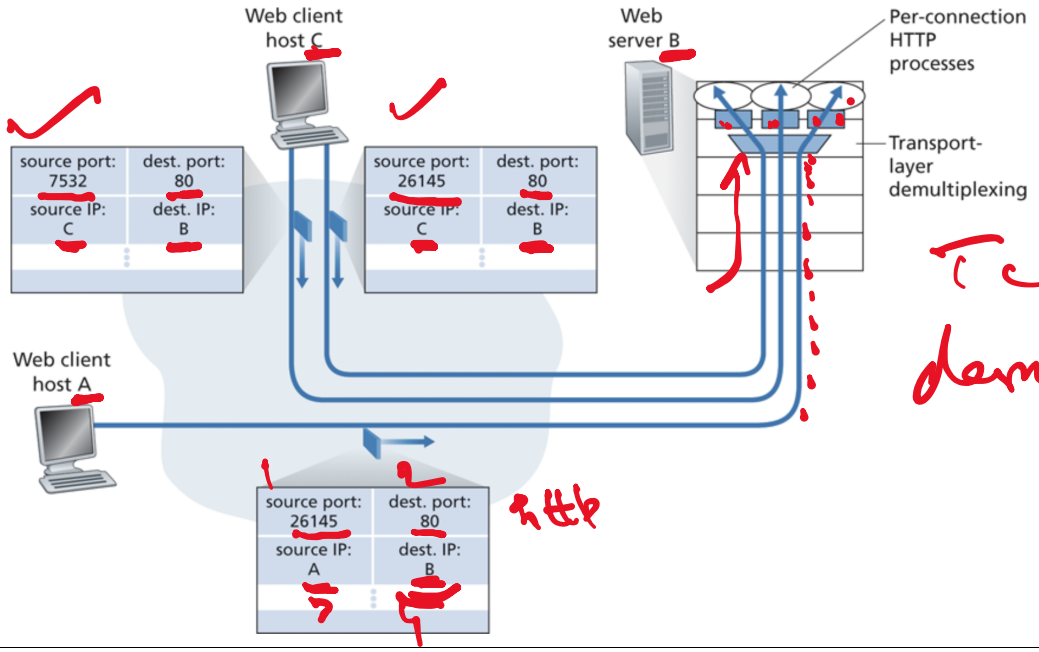
(12)For TCP, we use “welcoming socket”:



Source:
https://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html#:~:text=The%20steps%20of%20establishing%20a,recvfrom()%20and%20sendto().

(13)TCP example:

Figure 3.5 **Two clients, using the same destination port number (80) to communicate with the same Web server application**



(14) There is not always a one-to-one correspondence between connection sockets and processes.