A) Encapsulation/decapsulation and mux-demux
B) Example from application perspective
C) Measuring performance of networked applications
D) Socket layer

A. Encapsulation/decapsulation and mux-demux
- Protocol at a particular layer needs to add information that helps the peer protocol handle messages to the right higher layer entity
- Encapsulation - what happens at the sender in this context
- Decapsulation - what happens at the receiver
- Encapsulation entails a protocol adding its “header” and handing message down to the lower layer, where the same thing happens. Adding headers for multiple higher layer protocols — multiplexing. e.g Transport layer adds header for each application
- Decapsulation similarly entails the protocol at the receiving end peeling off the corresponding header and handing message to the immediate higher layer. Removing headers and handing off to suitable higher layer protocols — demultiplexing

B. Example

Suppose a client running a web-browser wants to access a server running nytimes.com

- User enters http://www.nytimes.com
  - Invokes the HTTP application layer protocol on the client
  - Sends a GET message to the server
  - Message has “GET” in the header and some other information

- This message is handed to the TCP layer as HTTP only runs on TCP (since it needs a reliable connection abstraction)
  - TCP waits for a segment sized amount of data. Fills up the segment. (Figures out size ahead of time using MTU discovery)
  - TCP adds its header. Client “port” C.port and server “port” S.port added to the header. Port indicates where the client and server application are attached to the network subsystem on the corresponding host. Server port is usually 80 (implied)
  - TCP hands this packet to IP
  - Goal of TCP is to deliver application bytes reliably and in-order to the destination
IP adds a header that includes the IP address of the sending host C.IP and that of the remote host S.IP and an identifier for the transport protocol, the proto
  - Hands this off to the link layer
  - Goal of IP is to try and get the packet to the destination. Best effort

Link layer looks up the link layer address of the router R at the other end of the local internetwork. Call this R.LL
  - Puts the local C.LL and R.LL address in a header and sends the packet along

Each switch on the path in the local internetwork forwards on the basis of L and sends it to the router R
  1. R peels off the LL header and parses the IP header
  2. Looks up S.IP in a table and figure out which next router to forward to. Say R
  3. Figures outs the LL address of the next router R1.LL
  4. Hands off to Link layer
  5. This goes back to step 4
  6. Steps 4 and 5 repeat, until packet reaches the internetwork connected to S

At this point the packet with the LL header RS.LL, S.LL, C.IP and S.IP reaches the host S
  - LL layer peels off RS.LL and S.LL
  - IP layer peels off C.IP and S.IP and hands off to the protocol indicated by “proto”

The transport protocol parses the transport header and hands the packet off to S.port

The application attached to S.port parses the GET request, constructs a response and sends the message back down toward C

The above steps repeat but in a reverse order

Some things that come into play above:

1. How does C know S.IP (it only knows the name for S)?
   ANSWER: DNS, the domain name system

2. How does C know R in step 4?
   ANSWER: Hosts and routers have a routing table. At hosts, this is configured manually or when the host boots up. At routers this is computed by a routing protocol

3. How does C know R.LL in step 4?
   ANSWER: Address resolution protocol

C. Performance

- What impacts the performance of the application in the example above?

- Consider an even simpler case where two apps A and B are connected directly to each other.
  Say A wants to send a message of N bytes to B. Say bandwidth is C. Say the latency of link is D
• Time for first byte to get to B - D — “propagation delay” - depends on speed of light and distance
• Time for all bytes in the message to leave A: N/C — “transmission delay” - depends on link capacity and number of bytes
• It is possible the messages are queued in some buffer perhaps at A and/or B. Suppose all N bytes saw a queuing delay consistently of q.
• Total time for message to get from A to B = D + N/C + q = propagation + transmission + queueing
• Some observations:
  ◦ If N is small or C is very large for a modest N, D matters a lot. — remote login apps impacted by D
  ◦ If N is very large then D matters less and C matters a lot — file transfer apps impacted by C.
  ◦ When there are multiple links along a path, C = min(capacities of all on-path links) and D = sum(latencies of all on-path links).
  ◦ Variation in latency can also affect apps — this is called jitter. Voice/video.

• Another useful concept: bandwidth-delay product.
• This in the above example is C * D (the definitions of C and D change similarly for multi-link paths as above).
• When an application has C * D bits outstanding — that is, these bytes have left the sender and are yet to reach the receiver, then it is “keeping pipe full”, or fully utilizing the available capacity.
• If the application keeps any fewer bytes outstanding, then it isn’t fully utilizing the network.

D. Socket API
• This is the interface that the OS provides to its networking subsystem
• Socket Interface was originally provided by the Berkeley distribution of Unix — now universally supported
• Socket - the point where a local application process attaches to the network
  ◦ An interface between an application and the network
  ◦ An application creates the socket
The interface defines operations for

- Creating a socket
- Attaching a socket to the network
- Sending and receiving messages through the socket
- Closing the socket

In the above example, both server and a client create a socket. Arguments - protocol family, socket type and protocol.

Server - passive open, prepares to accept a connection.
- bind - creates a socket and attaches it to the supplied server address and port (e.g., 80, for HTTP)
- listen - defines maximum pending connections
- accept - passive open. blocking. does not return until connection set up. returns a new socket

Client - active open
- runs – connect

Then send and recv are used on the respective sockets.