* Stacks of service abstraction
  + Short gets (DNS, DHCP) -> Use datagrams -> Internet Datagrams
  + Byte Stream –(TCP)--> User datagrams -> Internet Datagrams
    - Web requests/responses (HTTP) -> Byte Stream
      * Youtube/Wikipedia -> Web requests/responses
    - Email sending (SMTP) -> Byte Stream
    - Email receiving (IMAP) -> Byte Stream
* Multiplexing ByteStream
  + “u8 u8” (Which byte stream; what is the byte)
    - Any reading and writing of one byte would be actually two bytes, the first byte for which byte stream and the second for the actual byte
  + “u8 u8 {u8 u8 … u8}” (which byte stream; size of payload; sequence of characters of the string chunk)
    - Any reading and writing of n bytes would be actually n + 2 bytes
    - Tagged byte stream: HTTP/2 | SPDY
* How to make ByteStream push idempotent?
  + TCP Sender Message  
     first\_index  
     data   
     FIN  
    ****
  + This works for out-of-order or multiple deliveries. Since UDP has a checksum field, altered TCP Message would be ignored on the UDP layer.
  + What if datagrams are missing?
    - How does the sender know that a datagram needs to be sent multiple times?
    - DNS/DHCP: if we don’t receive an answer, then we retransmit. But such response/answer does not exist in `pushing` (void push())
  + Acknowledgement
    - TCP Receiver Message
      * “A B C D E F G” each byte sent as a separate TCP sender message, and “D” is not received.
      * “I got the sender message with first-index = 2, length = 1.”
        + Valid but more work. There will be one receiver message for each sender message.
      * “Got anything? Y/N. Next needed: #3.”
        + Acknowledgements are accumulative, and that make life simpler.
      * Give FIN flag a number: “A B C D E F G FIN”.  
        TCP Sender Message: {sequence number, data}  
        TCP Receiver Message: {Next needed: optional<int>}  
        and TCP Receiver Message {Next needed: optional(8)} would mean FIN is received.
      * TCP Receiver Message: {Next needed: optional<int>; available capacity:int}
        + {Next needed: optional(3); available capacity: 2} === Receiver wants to here about “DE”.
        + “DE” is the **window**. (Red area in that picture of lab 1)
    - **TCP Receiver Message: {Ack no: optional<int>; window size: int}**
  + TCP Segment  
    
  + This is the service abstraction that TCP is providing:
  + And this is what happens under the hood (and also what you will be implementing in the labs)



* + Rules of TCP
    - Reply to any nonempty TCP Server Message